SOFTWARE DESIGN DOCUMENT

Team Undecided

Johns Hopkins University
Foundations of Software Engineering
605.601.83
Fall 2019

Team Members:
Sean Walsh
Kira Ullman
Andrew Johnson
Trey Hoffman
Joel Huddleston

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this software design document is to layout the main functionality of the ClueLess Game and its major components.

1.2 Scope

The scope of this design document will focus on entity classes and essential use cases.

1.3 Overview

This document will provide a top-level Class Diagram of the major game components and their connections. Additionally, a number of dynamic models are provided to detail essential use cases.

2 System Overview

The design of this project focuses largely around a Model-View-Controller concept. A User will interact with their Client application and provide input to the game's Message Server. This server will relay input to the main GameController.

The GameController controls all aspects of the game. It will handle the processing of User input for their Player characters as well as informing the GameBoard on how to update its state.

All of this is then fed back to the User through the Message Server to the User's Client so the User can decide on their next course of action.

3 System Architecture

3.1 Architectural Design

3.1.1 Game Flow Subsystem

This subsystem is largely concern with handling the processing of main game components. This system sends and receives messages from the Messaging Subsystem in order to process user events and update users as to the game state. Additionally, this subsystem will utilize the other back end subsystems when required and provide them necessary data.

3.1.2 Game Board Subsystem

This subsystem is only concerned about the game's game board state. It controls the placement and status of all the game pieces, weapons, and rooms in relation to each other. This subsystem will inform the Game Flow Subsystem of its state when changes occur.

3.1.3 Deck Subsystem

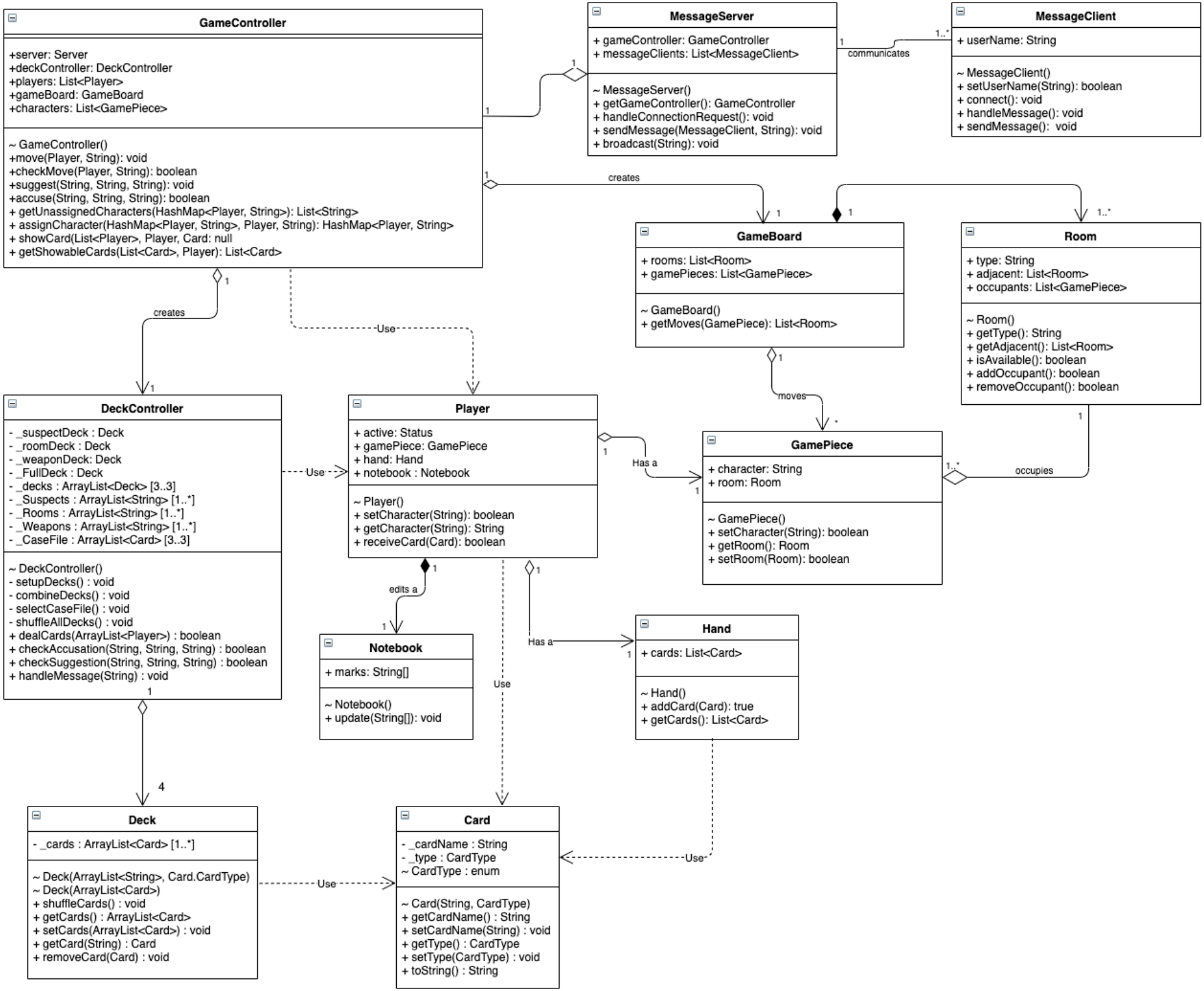
The Deck Subsystem handles the construction and management of the various decks used for game play. This subsystem will create, modify, shuffle, deal and retrieve cards when required. Additionally, this subsystem will handle the game's CaseFile cards which will be used during accusation events.

3.1.4 Messaging Subsystem

The Messaging Subsystem handles all message traffic between the User and the Game Flow Subsystem. Users will interact with Client applications and provide input that will be passed to the Message Server. The server will then inform the GameController of this input want provide the users with updates.

3.2 Class Diagram

This class diagram shows the high-level connections between the major components of the various subsystems.


GameController
+server: Server
+deckController: DeckController
+players: List<Player>
+gameBoard: GameBoard
+characters: List<GamePiece>
~ GameController()
+move(Player, String): void
+checkMove(Player, String): boolean
+suggest(String, String, String): void
+accuse(String, String, String): boolean
+ getUnassignedCharacters(HashMap<Player, String>): List<String>
+ assignCharacter(HashMap<Player, String>, Player, String): HashMap<Player, String>
+ showCard(List<Player>, Player, Card: null
+ getShowableCards(List<Card>, Player): List<Card>
MessageServer
+ gameController: GameController
+ messageClients: List<MessageClient>
~ MessageServer()
+ getGameController(): GameController
+ handleConnectionRequest(): void
+ sendMessage(MessageClient, String): void
+ broadcast(String): void
MessageClient
+ userName: String
~ MessageClient()
+ setUserName(String): boolean
+ connect(): void
+ handleMessage(): void
+ sendMessage(): void
communicates
creates
GameBoard
+ rooms: List<Room>
+ gamePieces: List<GamePiece>
~ GameBoard()
+ getMoves(GamePiece): List<Room>
Room
+ type: String
+ adjacent: List<Room>
+ occupants: List<GamePiece>
~ Room()
+ getType(): String
+ getAdjacent(): List<Room>
+ isAvailable(): boolean
+ addOccupant(): boolean
+ removeOccupant(): boolean
moves
occupies
DeckController
- _suspectDeck : Deck
- _roomDeck : Deck
- _weaponDeck: Deck
- _FullDeck : Deck
- _decks : ArrayList<Deck> [3..3]
- _Suspects : ArrayList<String> [1..*]
- _Rooms : ArrayList<String> [1..*]
- _Weapons : ArrayList<String> [1..*]
- _CaseFile : ArrayList<Card> [3..3]
~ DeckController()
- setupDecks() : void
- combineDecks() : void
- selectCaseFile() : void
- shuffleAllDecks() : void
+ dealCards(ArrayList<Player>) : boolean
+ checkAccusation(String, String, String) : boolean
+ checkSuggestion(String, String, String) : boolean
+ handleMessage(String) : void
Use
Player
+ active: Status
+ gamePiece: GamePiece
+ hand: Hand
+ notebook : Notebook
~ Player()
+ setCharacter(String): boolean
+ getCharacter(String): String
+ receiveCard(Card): boolean
Has a
GamePiece
+ character: String
+ room: Room
~ GamePiece()
+ setCharacter(String): boolean
+ getRoom(): Room
+ setRoom(Room): boolean
edits a
Notebook
+ marks: String[]
~ Notebook()
+ update(String[]): void
Has a
Hand
+ cards: List<Card>
~ Hand()
+ addCard(Card): true
+ getCards(): List<Card>
Use
Deck
- _cards : ArrayList<Card> [1..*]
~ Deck(ArrayList<String>, Card.CardType)
~ Deck(ArrayList<Card>)
+ shuffleCards() : void
+ getCards() : ArrayList<Card>
+ setCards(ArrayList<Card>) : void
+ getCard(String) : Card
+ removeCard(Card) : void
Use
Card
- _cardName : String
- _type : CardType
~ CardType : enum
~ Card(String, CardType)
+ getCardName() : String
+ setCardName(String) : void
+ getType() : CardType
+ setType(CardType) : void
+ toString() : String
Use

3.3 Class Descriptions

Class name: GameController	
Description: Handles the flow of the game. Utilizes the other subsystems when needed to process User input.	
Attributes	**Descriptions**
MessageServer server	The instance of the MessageServer
DeckController deckController	The deck controller object
List<Player> players	The current list of active players
GameBoard gameboard	The instance of the GameBoard
List<GamePiece> characters	The list of game pieces in the game
Methods	**Descriptions**
GameController()	Constructor. Setups initial game state.
move()	Moves a player to a new location on the game board
checkMove()	Checks for available moves allowed to the active Player
suggest()	Processes a suggestion from a User
accuse()	Processes an accusation from a User
getUnassignedCharacters()	Returns a list of unused character game pieces
assignCharacter()	Assigns a character to a Player
showCard()	Shows a User a Card of another User
getShowableCards()	Returns a list of cards that can be revealed

Class name: Player	
Description: Holds the state of a Player	
Attributes	**Descriptions**
Status active	Determines if a Player is active in the game
GamePiece gamePiece	The game piece assigned to this Player
Hand hand	This Player's hand of cards
Notebook notebook	This Player's notebook
Methods	**Descriptions**
Player()	Constructor
setCharacter()	Sets the Player's character for the game
getCharacter()	Gets the Player's character reference
receiveCard()	Places Card into this Player's hand

Class name: Hand	
Description: Holds the cards of a Player	
Attributes	**Descriptions**
List<Card> cards	The cards in this Player's hand
Methods	**Descriptions**
Hand()	Constructor
addCard()	Adds a Card to this hand
getCards()	Returns a list of cards in this hand

Class name: Notebook	
Description: Handles the operations of the Player's notebook.	
Attributes	**Descriptions**
String[] marks	An array of marks made by the Player
Methods	**Descriptions**
Notebook()	Constructor
update()	Updates the marks made by the Player

Class name: DeckController	
Description: Handles the creation and management of the decks and cards for the game.	
Attributes	**Descriptions**
Deck _suspectDeck	The deck of suspects
Deck _roomDeck	The deck of rooms
Deck _weaponDeck	The deck of weapons
Deck _FullDeck	The combine deck of cards after the CaseFIle is selected
ArrayList<Deck> _decks	The set of the suspect, room, and weapon decks
ArrayList<String> _Suspects	The list of suspects
ArrayList<String> _Rooms	The list of rooms
ArrayList<String> _Weapons	The list of weapons
ArrayList<Card> _CaseFile	The selection of Cards players need to guess in order to win the game
Methods	**Descriptions**
DeckController()	Constructor. Sets up all the decks, selects the Case File and shuffles the remaining cards into the Full Deck
setupDecks()	Creates the three main decks
combineDecks()	Combines the three main decks into the Full Deck
selectCaseFile()	Selects one card from each of the three main decks
shuffleAllDecks()	Shuffles all three of the main decks
dealCards()	Deals the Full Deck to the Players
checkAccusation()	Checks a Player accusation against the Case File
checkSuggestion()	Checks a Player suggestion against the other Player hands
handleMessage()	Creates a message to send to the GameController

Class name: Deck	
Description: Controls the aspects of a Deck of cards	
Attributes	**Descriptions**
ArrayList<Card> _cards	The list of cards in this deck

Methods	**Descriptions**
Deck()	Creates a deck from a list of card names and a card type
Deck()	Creates a deck from a list of Cards
shuffleCards()	Shuffles the cards in this deck
getCards()	Returns a list of this deck's cards
setCards()	Takes a list of Cards and set this deck's cards to that list
getCard()	Returns a single Card from this deck
removeCard()	Removes a Card from this deck

Class name: Card	
Description: Creates a Card object and sets its name and type	
Attributes	**Descriptions**
String _cardName	The name of this card
CardType _type	The type of this card
enum CardType	The list of available card types
Methods	**Descriptions**
Card()	Constructor
getCardName()	Returns this card's name
setCardName()	Sets the name of this card
getType()	Returns this card's type
setType()	Sets the type of this card
toString()	Constructs the output string for this card

Class name: MessageServer	
Description: Handles the message traffic between the GameController and the MessageClients.	
Attributes	**Descriptions**
GameController gameController	The instance of the game controller
List<MessageClient> messageClients	The list of active clients connected to this server
Methods	**Descriptions**
MesageServer()	Constructor. Sets up game server and creates the game controller.
getGameController()	Returns the instance of the game controller
handleConnectionRequest()	Processes client connection attempts
sendMessage()	Sends updates to a single MessageClient
broadcast()	Sends updates to all MessageClients

Class name: MessageClient	
Description: The entry point for the User into the game server. Handles User interaction and provides them with updates from the server.	
Attributes	**Descriptions**
String userName	The name of the User
Methods	**Descriptions**

MessageClient()	Constructor
setUserName()	Sets the name of the user for this client
connect()	Connects this client to the game MessageServer
handleMessage()	Process messages from the server
sendMessage()	Package and send messages from the User to the server

Class name: GameBoard	
Description: Controls the state of the game board and objects.	
Attributes	**Descriptions**
List<Room> rooms	The list of rooms on the game board
List<GamePiece> gamePieces	The list of game pieces used in this game
Methods	**Descriptions**
GameBoard()	Constructor. Sets up initial state of game board.
getMoves()	Provides a list of rooms a game piece can move to

Class name: Room	
Description: A room is where Players move to perform accusations or suggestions.	
Attributes	**Descriptions**
String type	The name of the room
List<Room> adjacent	The list of adjacent rooms to this one
List<GamePiece> occupants	The list of Players in the room
Methods	**Descriptions**
Room()	Constructor
getType()	Returns this room's name
getAdjacent()	Returns the list of adjacent rooms
isAvailable()	Returns whether this room is available or not
addOccupant()	Attempts to add a game piece to this room
removeOccupant()	Removes a game piece from this room

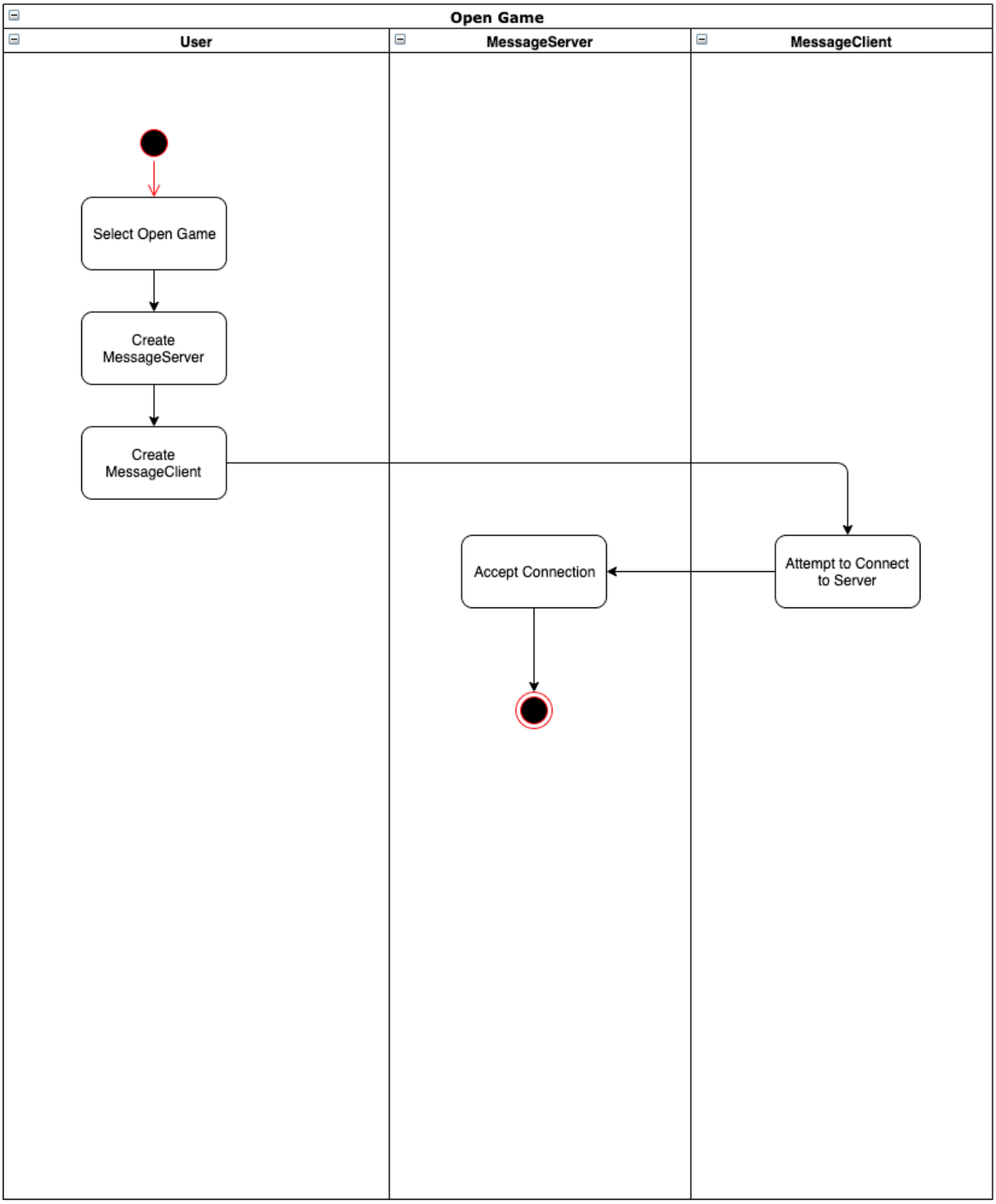
Class name: GamePiece	
Description: The game piece is the Player representation on the game board.	
Attributes	**Descriptions**
String character	The name of the game piece
Room room	The game pieces location
Methods	**Descriptions**
GamePiece()	Constructor
setCharacter()	Sets the name of this game piece
getRoom()	Gets the location of this game piece
setRoom()	Sets the location of this game piece

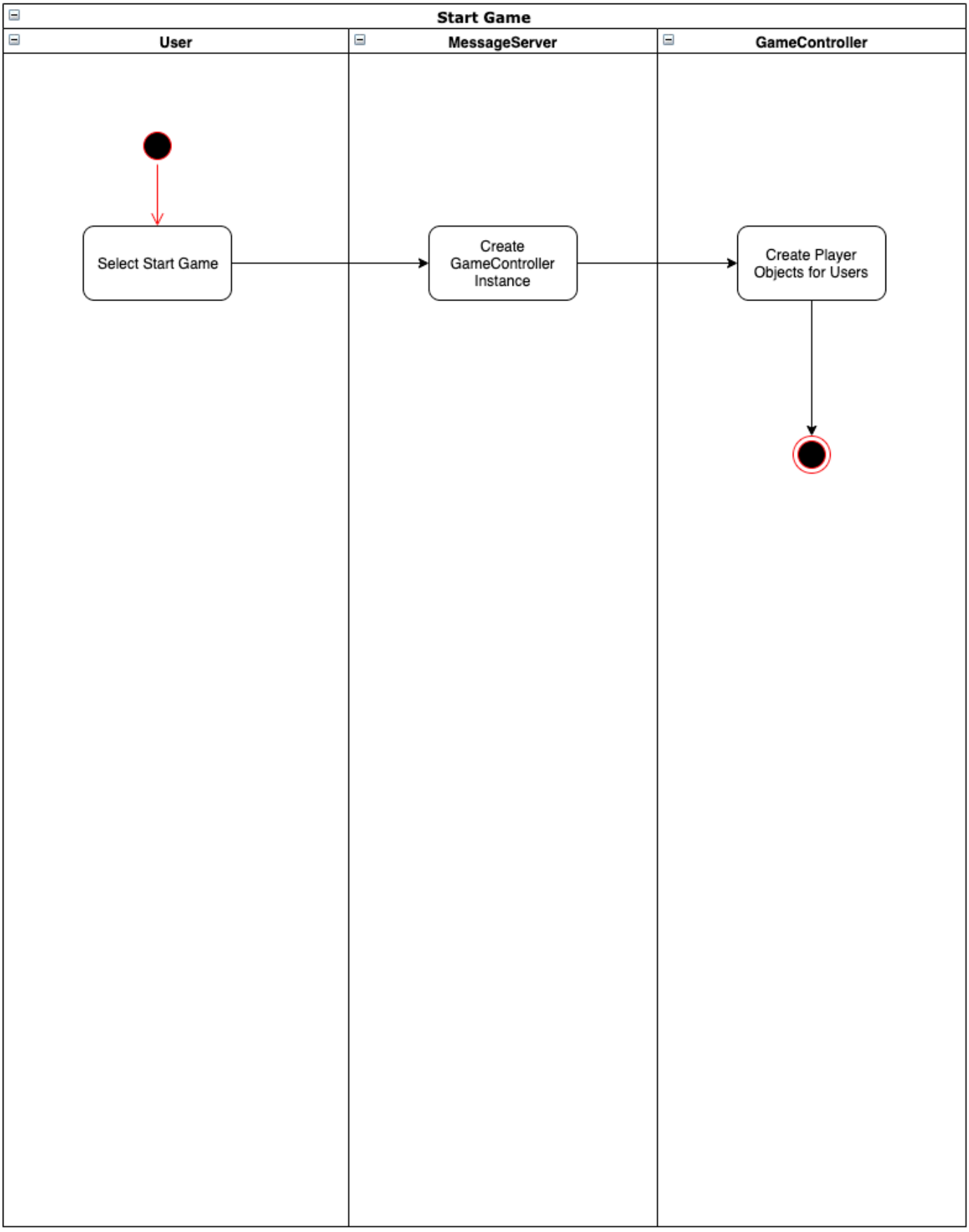
3.4 Design Rationale

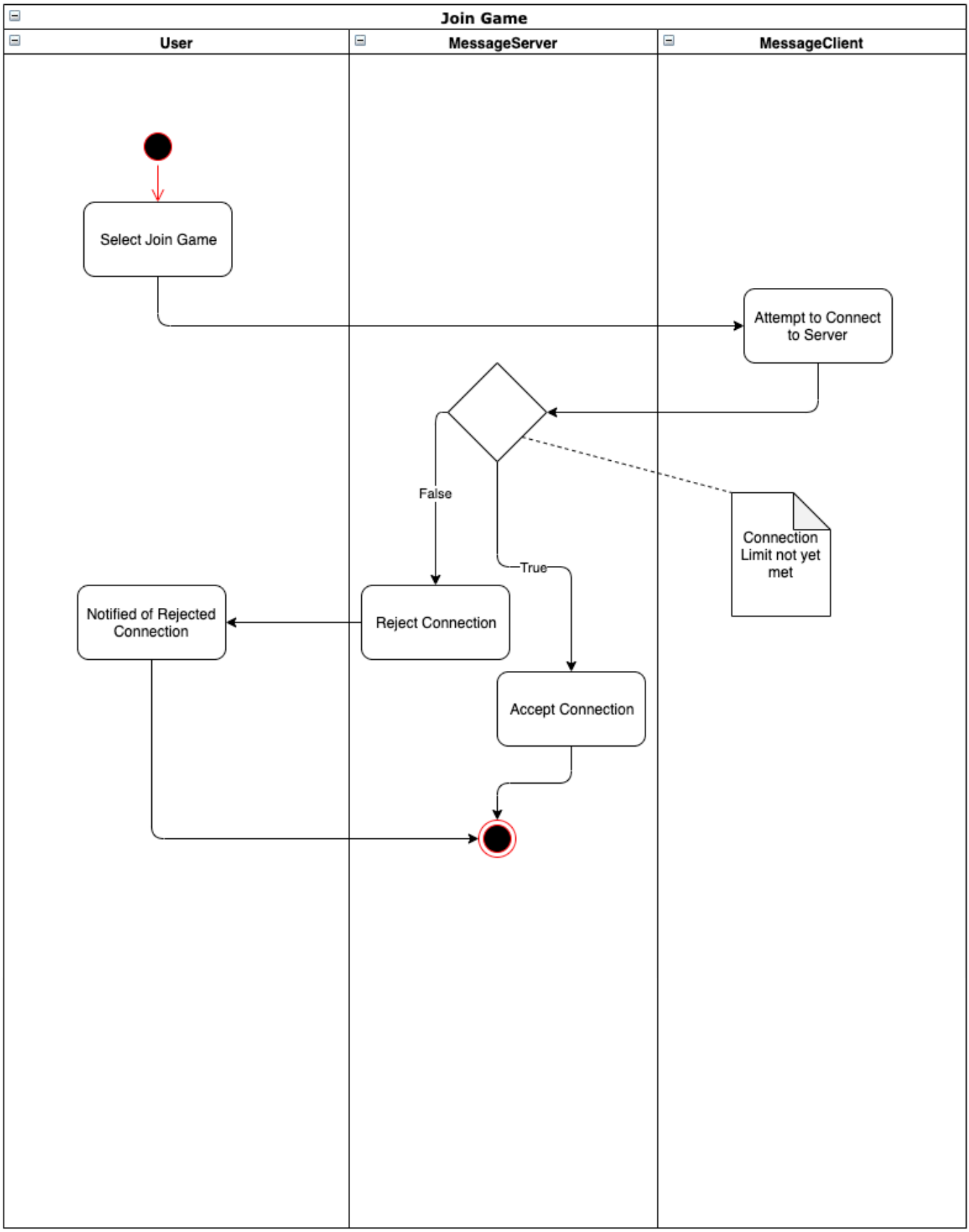
We chose our architectural design, as described in section 3.1, because we believe it to make the most sense in constructing the components of this application. By encapsulating the game state and processing within the controller concept allows for a simpler design for the message server and client. Additionally, this design allows for the best reuse of components between the different subsystems.

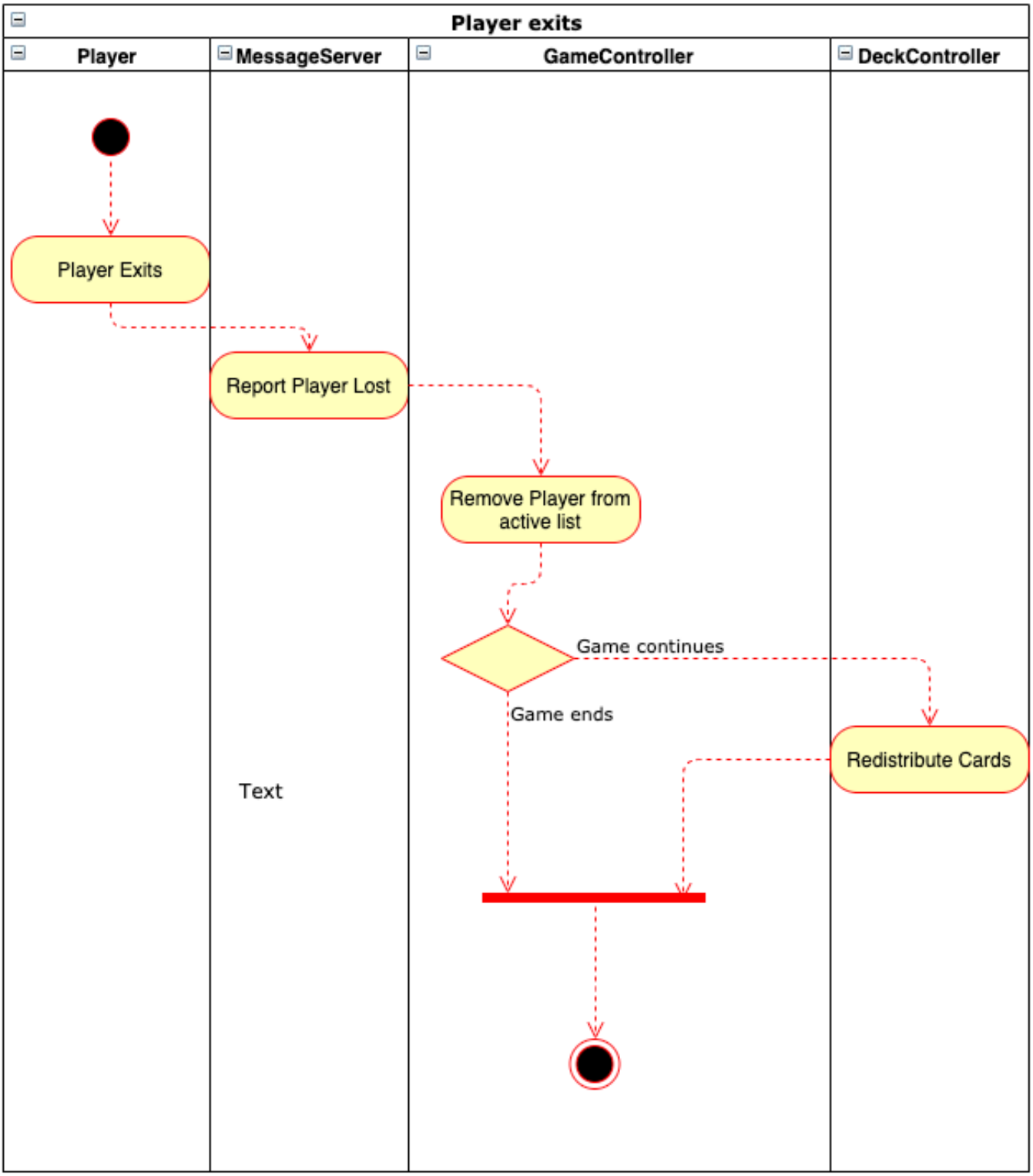
4 Dynamic Models

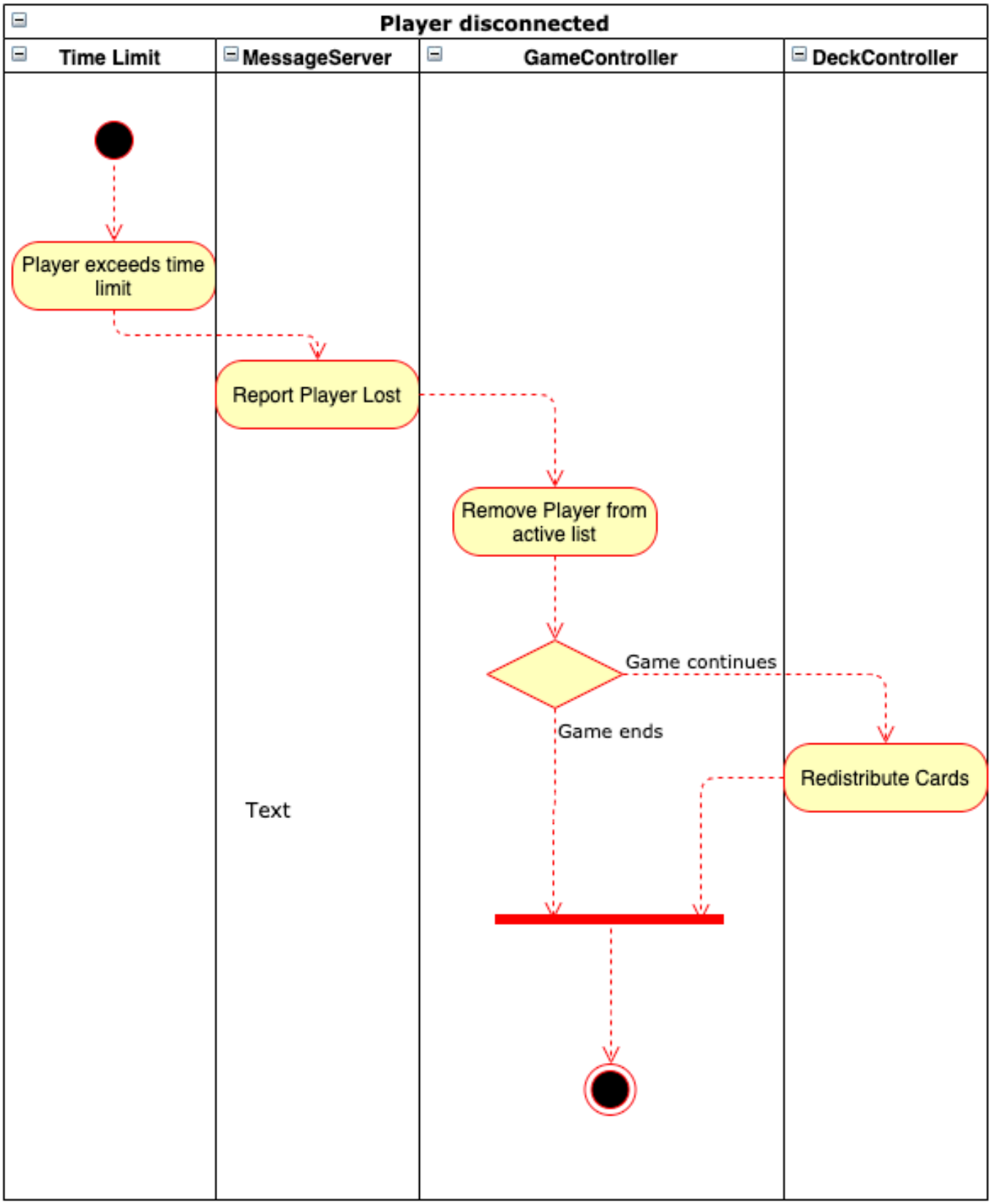
The following models show dynamic interaction between various objects and classes for some of the major scenarios of this application.


Open Game
User
MessageServer
MessageClient
Select Open Game
Create MessageServer
Create MessageClient
Accept Connection
Attempt to Connect to Server

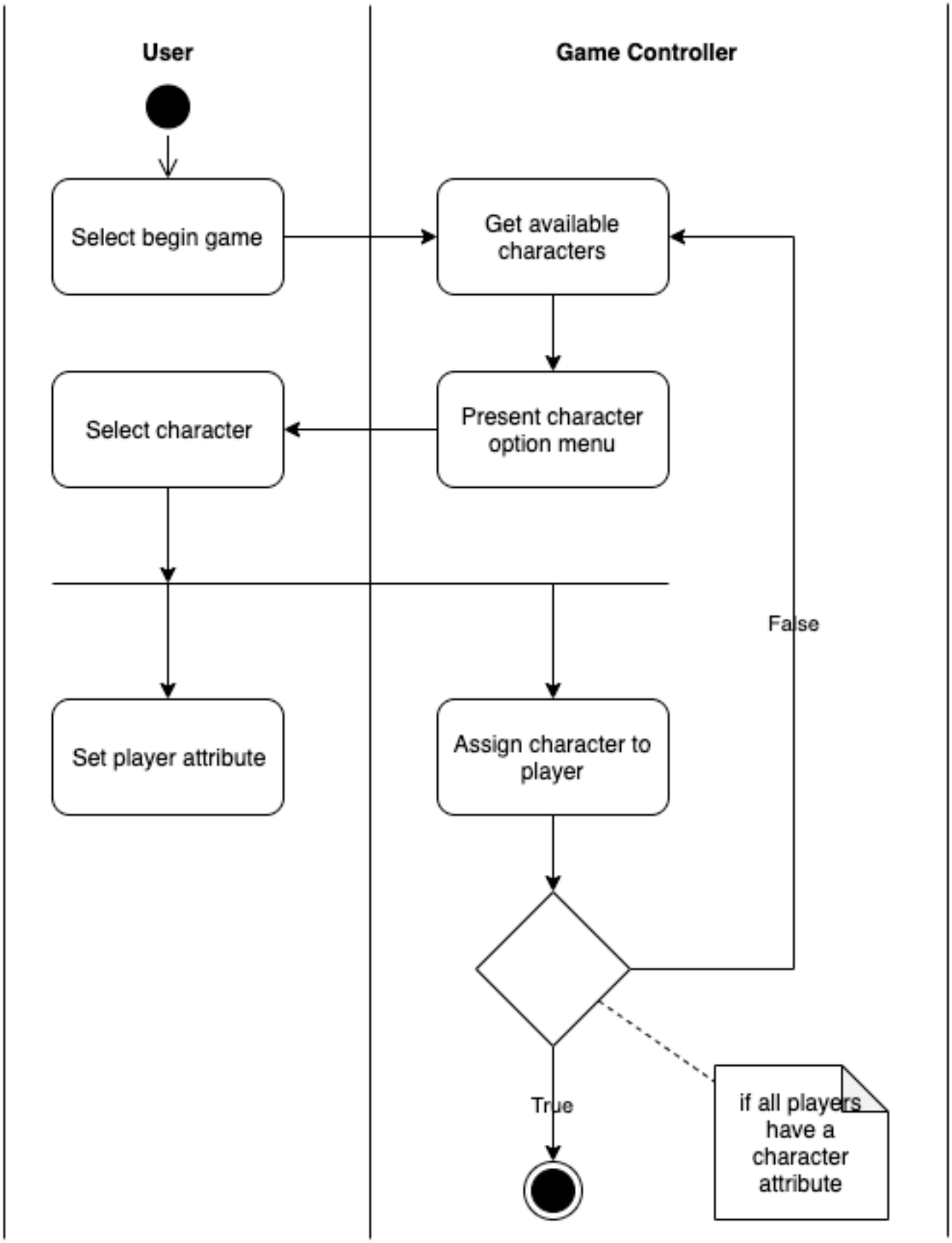

Start Game
User
MessageServer
GameController
Select Start Game
Create GameController Instance
Create Player Objects for Users

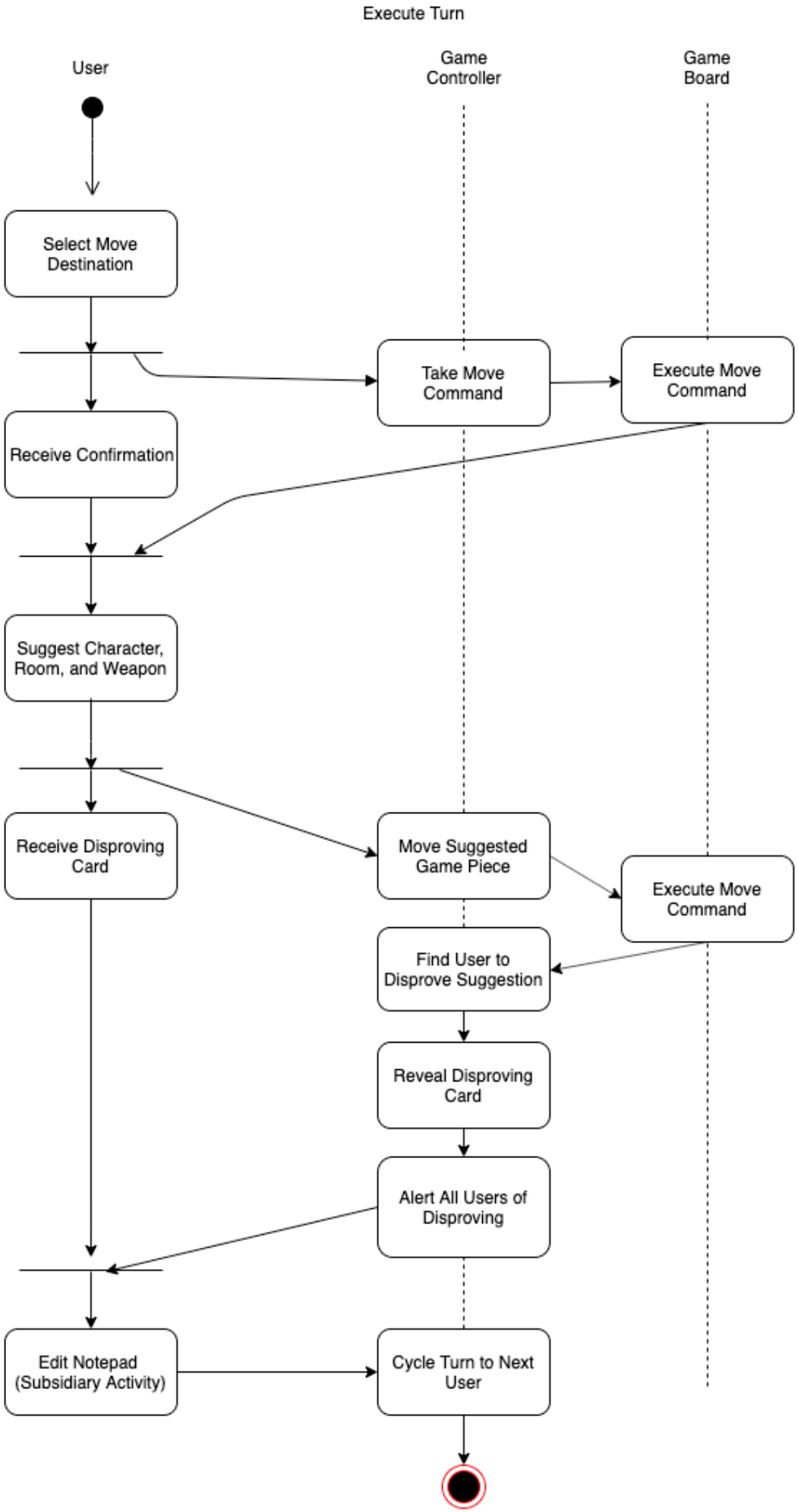

Join Game
User
MessageServer
MessageClient
Select Join Game
Attempt to Connect to Server
False
True
Connection Limit not yet met
Notified of Rejected Connection
Reject Connection
Accept Connection

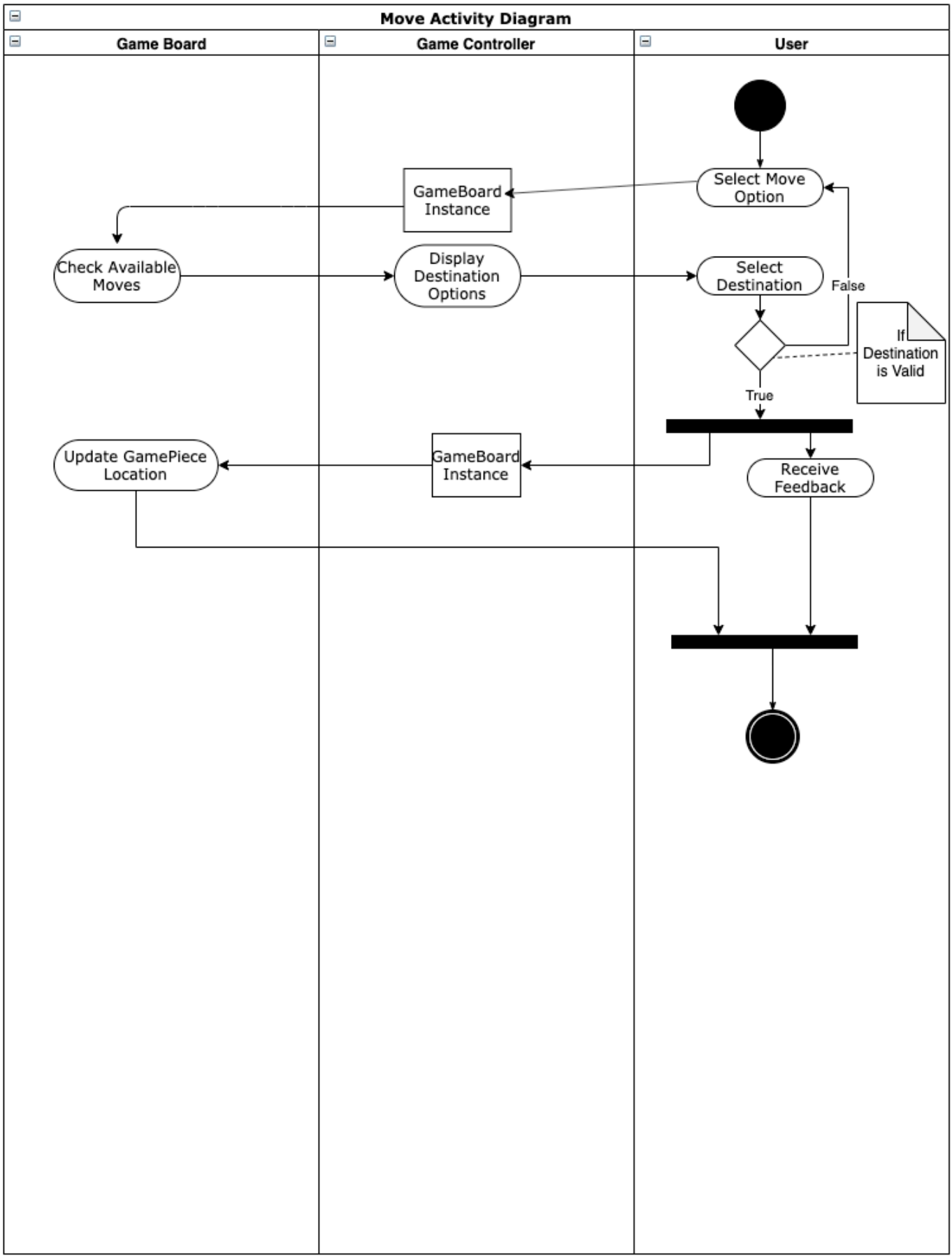

Player exits
Player
MessageServer
GameController
DeckController
Player Exits
Report Player Lost
Remove Player from active list
Game continues
Game ends
Redistribute Cards
Text

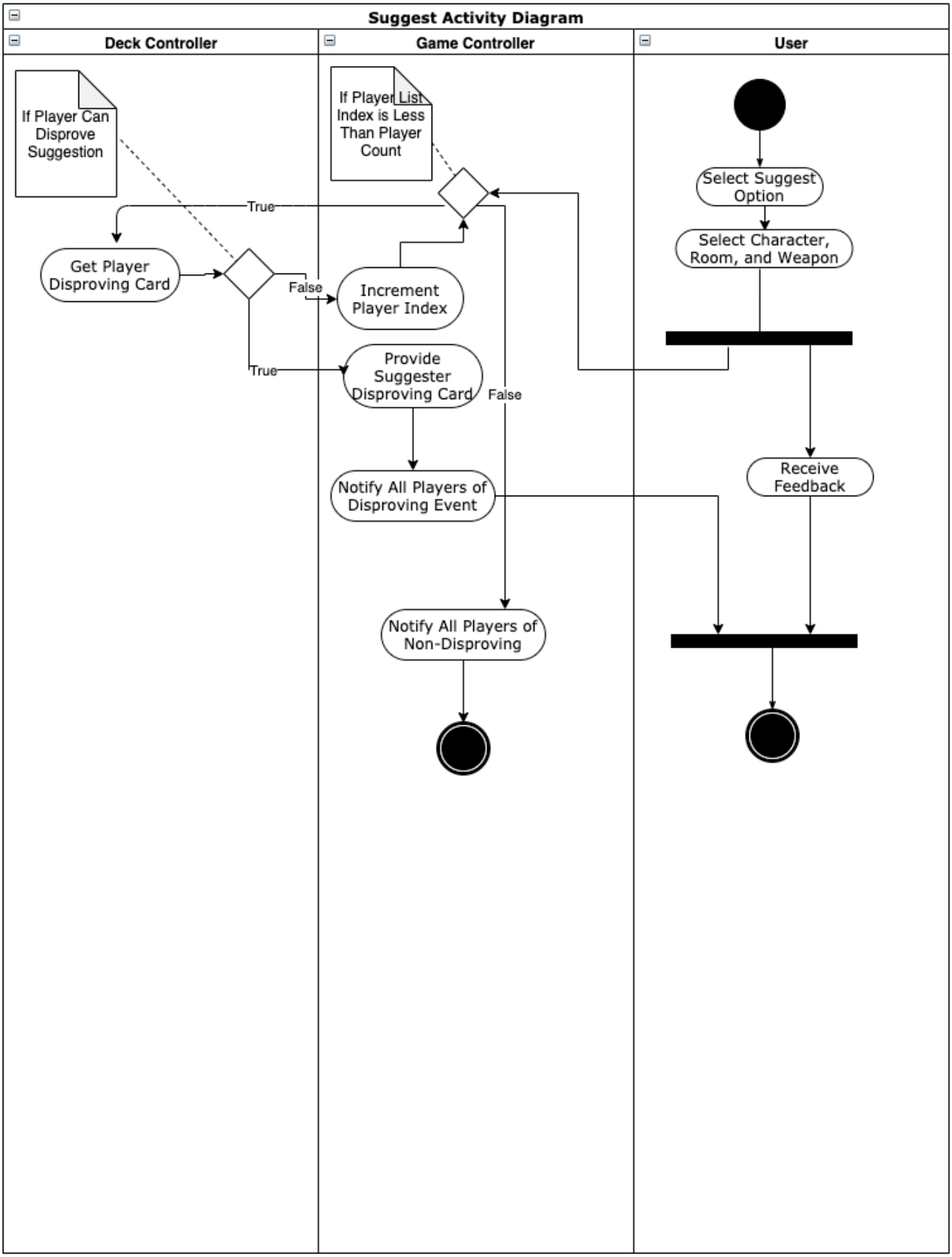

Player disconnected
Time Limit
MessageServer
GameController
DeckController
Player exceeds time limit
Report Player Lost
Remove Player from active list
Game continues
Game ends
Redistribute Cards
Text

Choose Character

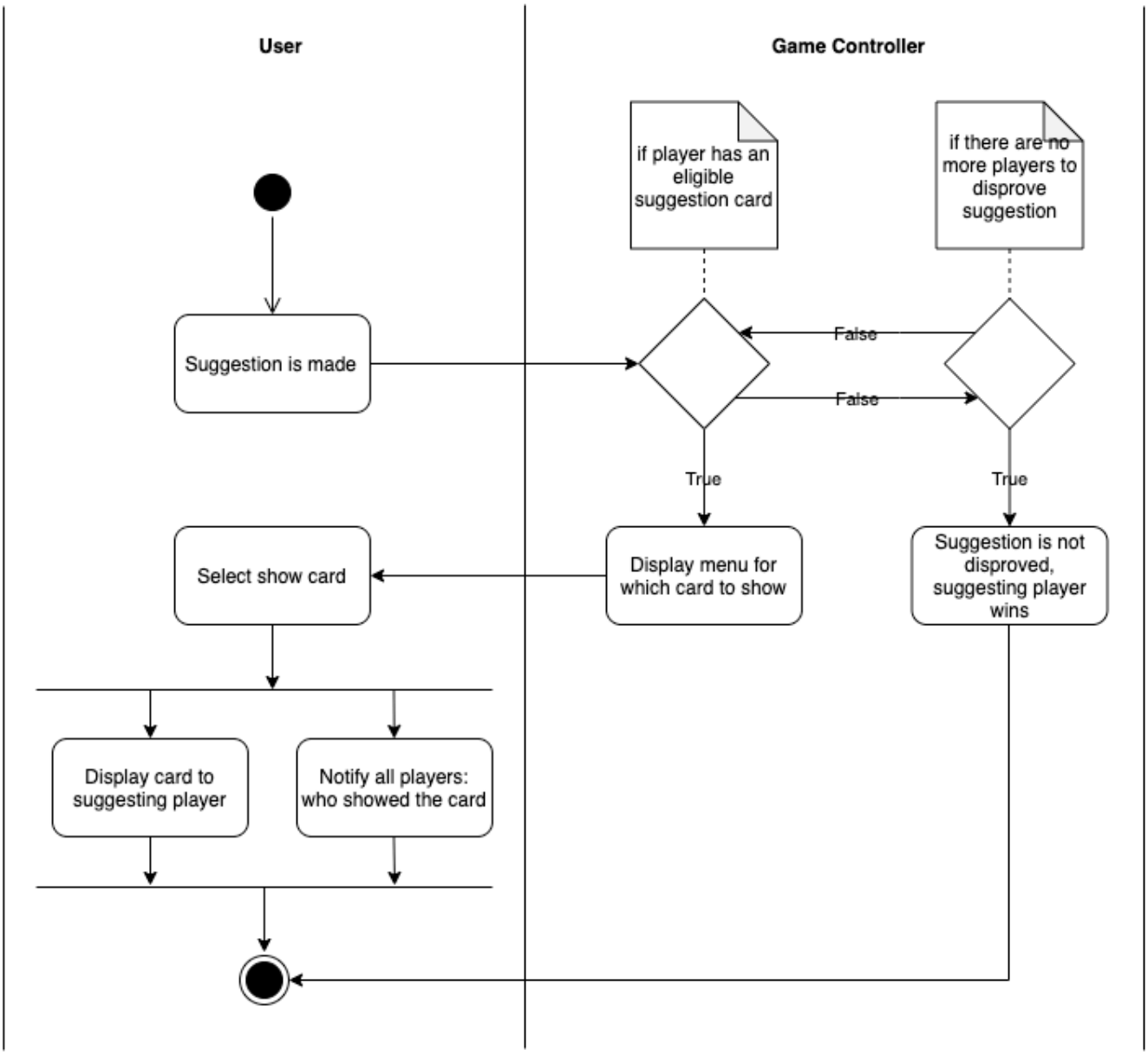


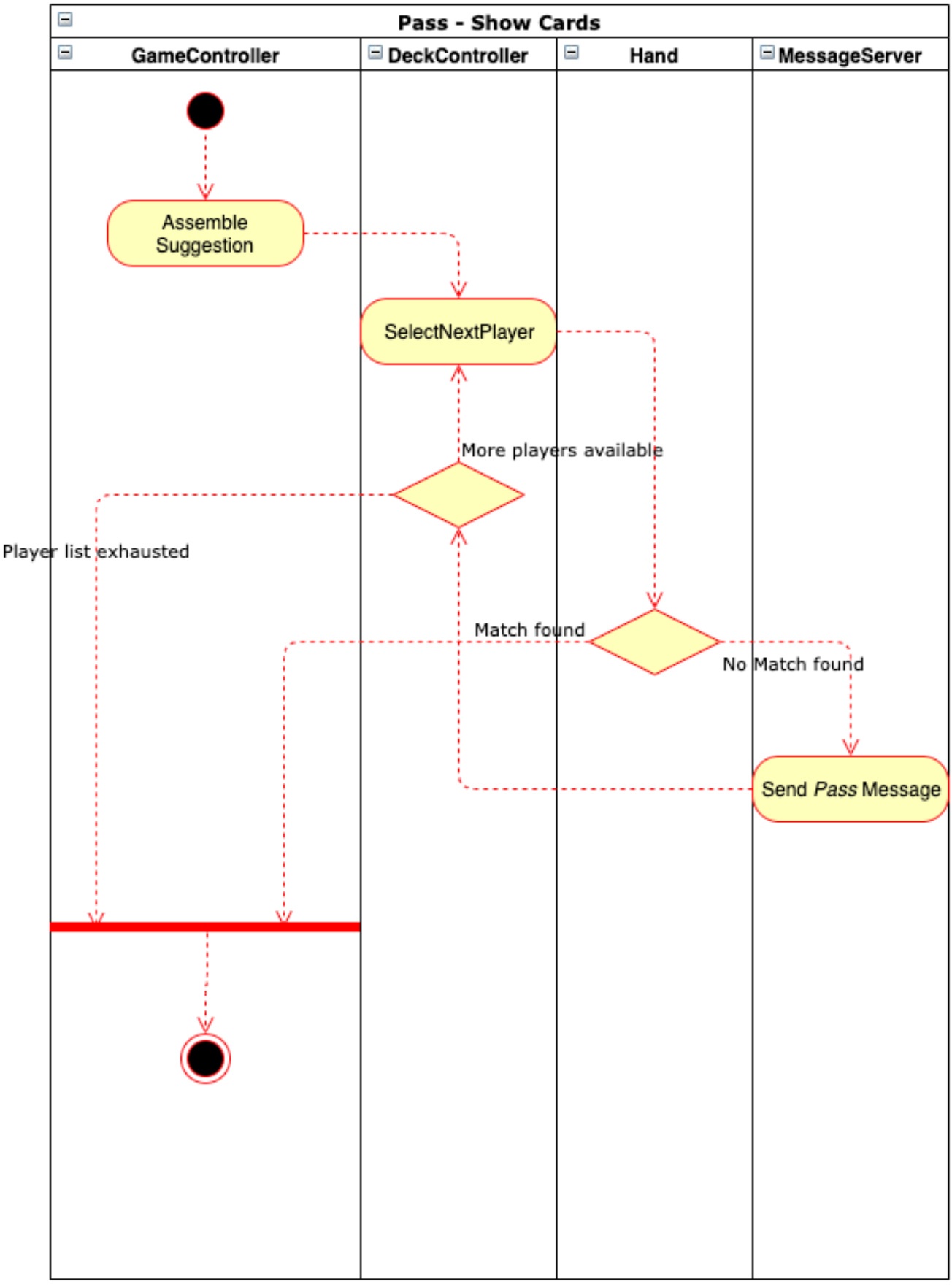

Execute Turn
User
Game Controller
Game Board
Select Move Destination
Take Move Command
Execute Move Command
Receive Confirmation
Suggest Character, Room, and Weapon
Receive Disproving Card
Move Suggested Game Piece
Execute Move Command
Find User to Disprove Suggestion
Reveal Disproving Card
Alert All Users of Disproving
Edit Notepad (Subsidiary Activity)
Cycle Turn to Next User


Move Activity Diagram
Game Board
Game Controller
User
Select Move Option
GameBoard Instance
Check Available Moves
Display Destination Options
Select Destination
False
If Destination is Valid
True
Update GamePiece Location
GameBoard Instance
Receive Feedback

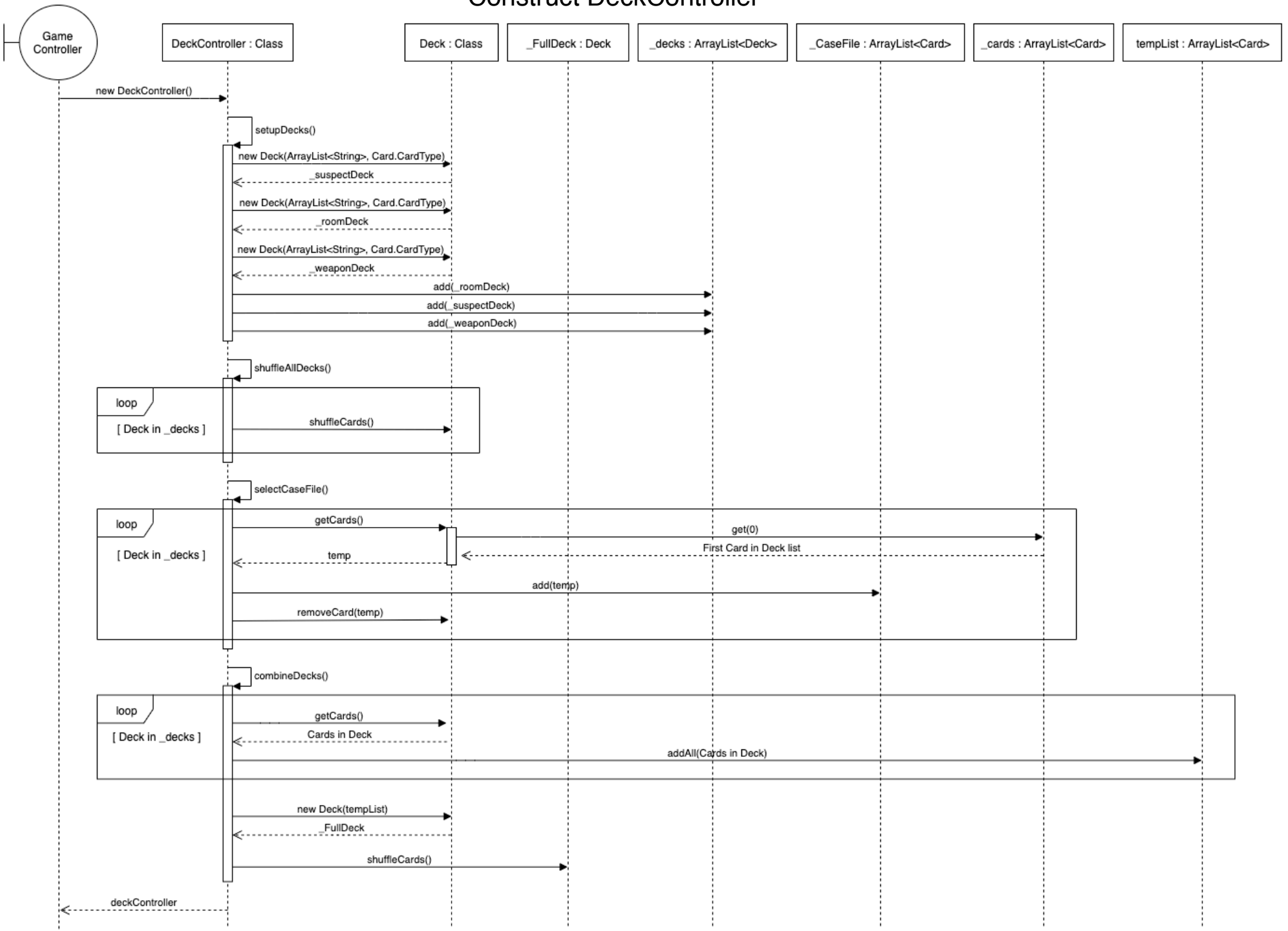

Suggest Activity Diagram
Deck Controller
Game Controller
User
If Player Can Disprove Suggestion
If Player List Index is Less Than Player Count
Select Suggest Option
Select Character, Room, and Weapon
True
Get Player Disproving Card
False
Increment Player Index
True
Provide Suggester Disproving Card
False
Receive Feedback
Notify All Players of Disproving Event
Notify All Players of Non-Disproving

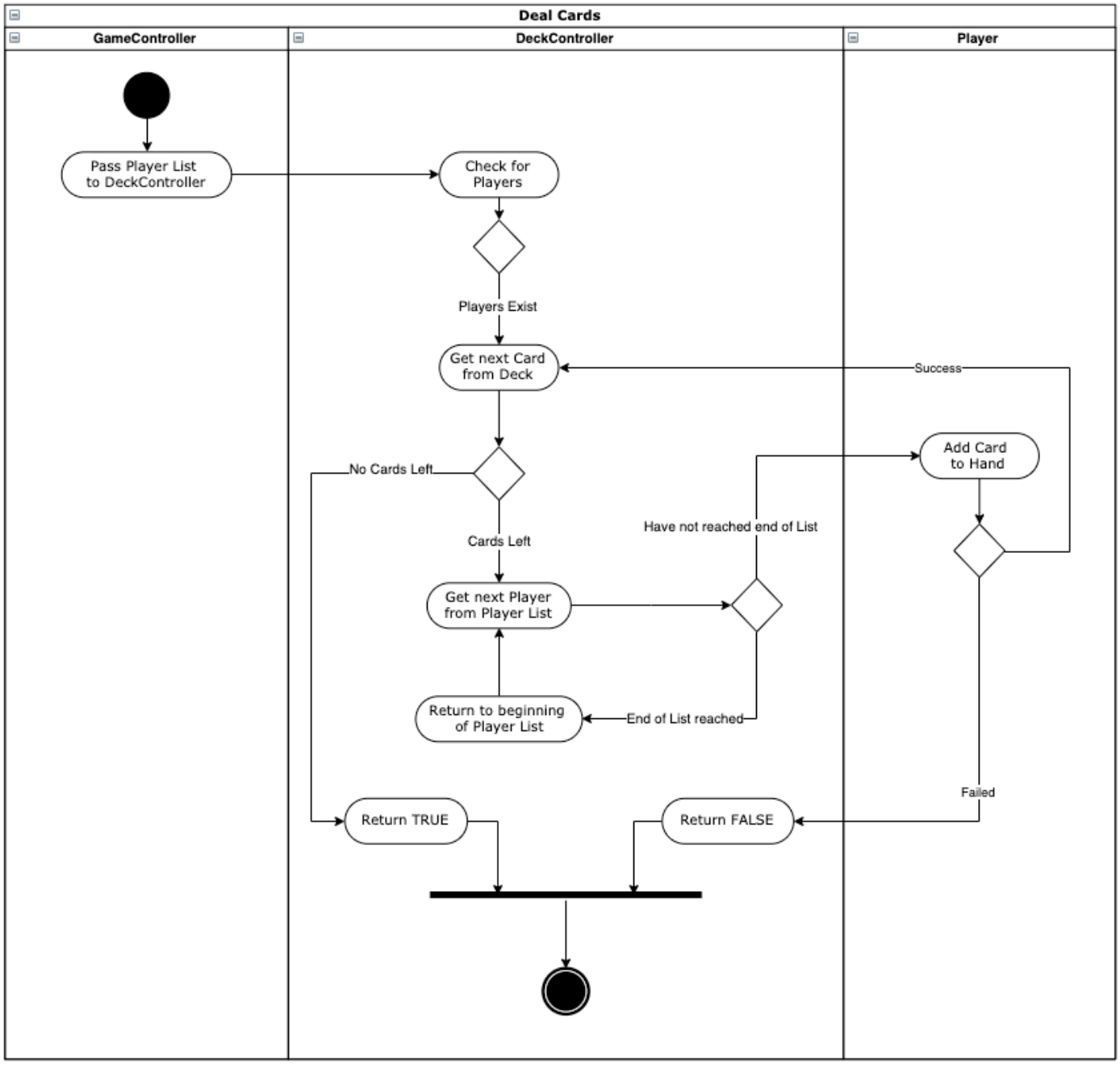
Show Card

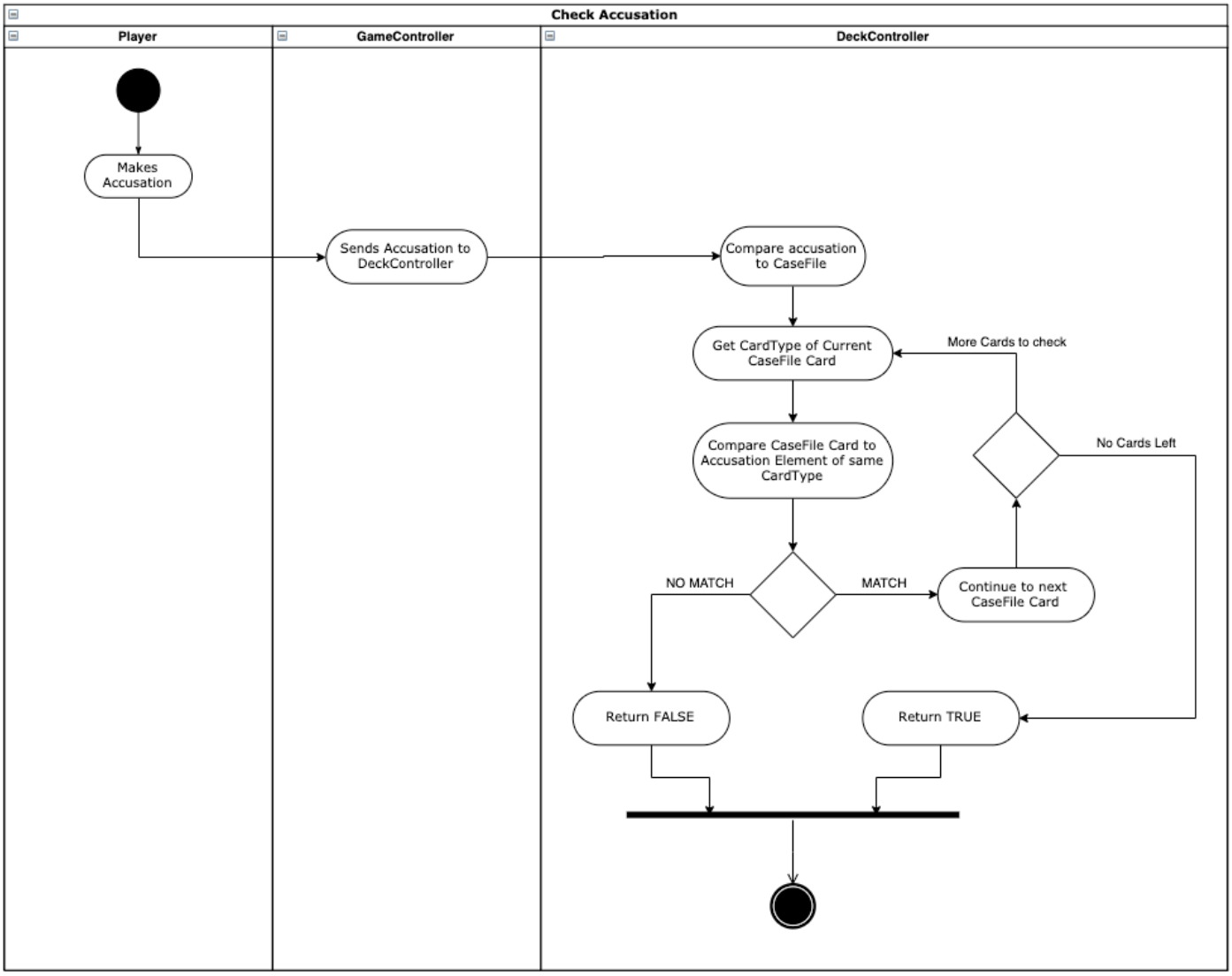

User
Game Controller
if player has an eligible suggestion card
if there are no more players to disprove suggestion
Suggestion is made
False
False
True
True
Select show card
Display menu for which card to show
Suggestion is not disproved, suggesting player wins
Display card to suggesting player
Notify all players: who showed the card


Pass - Show Cards
GameController
DeckController
Hand
MessageServer
Assemble Suggestion
SelectNextPlayer
More players available
Player list exhausted
Match found
No Match found
Send *Pass* Message

Construct DeckController




Deal Cards
GameController
DeckController
Player
Pass Player List to DeckController
Check for Players
Players Exist
Get next Card from Deck
Success
No Cards Left
Add Card to Hand
Have not reached end of List
Cards Left
Get next Player from Player List
Return to beginning of Player List
End of List reached
Failed
Return TRUE
Return FALSE


Check Accusation
Player
GameController
DeckController
Makes Accusation
Sends Accusation to DeckController
Compare accusation to CaseFile
Get CardType of Current CaseFile Card
More Cards to check
Compare CaseFile Card to Accusation Element of same CardType
No Cards Left
NO MATCH
MATCH
Continue to next CaseFile Card
Return FALSE
Return TRUE