Design Assignment

Marshal Powell, Ryan Wendling, Dalainee Viernes

February 4, 2016

1 Introduction

1.1 Purpose

The purpose of this design document to outline a possible implementation design for a programmed version of the board game Deadwood by Cheapass Games. The ”problem” is that Deadwood exists only in a physical format and has not yet been made into an electronic copy. Moreover, Deadwood is complex game that includes many components and thus this document is a brainstorm design for how to tackle the implementation and organization of it.

1.2 Scope

This document includes no coding implementation of Deadwood, but rather the overall design including classes and their attributes and methods, and the relationships between the classes (their interdependencies).

1.3 Organization

We have nine outlined classes in this design of Deadwood. These classes include:

- **Deadwood:** The main class. This class starts the game and ends the game by keeping count of the days and then scoring the players. It is composed (aggregated) of Actors and the Board.
- **Actor:** The class that implements the movements of the players themselves. The Actor class holds all of the current information about the player, including rank, credits, dollars, rehearsals, and their current Room. It also can be associated with a Role, when the Actor has taken a role. When an Actor wishes to upgrade it uses the Casting Office class, or when a roll must be made it uses the Dice class. When an Actor wishes to move from one room to another, it must use the Room class to see what Rooms are adjacent to the current room they are in.
- **Board:** The Board class is an aggregation of Rooms, containing the information about every Room object. When the game is started, Deadwood starts the game through the Board class, and at the end of every day, the Board class resets using the Scene Library class to assign Scenes to Rooms.
- **Room:** This class is composed of an associated Scene and an aggregation of Roles. Every Room object has an array to the its adjacent Rooms, and holds information about the current Actors that have taken roles in it. Every Room object knows how many shots in total it has, and how many shots have been taken. When the last shot has been taken, it uses the Dice class and the Actors associated with it to dole out the bonus money.
- **Scene Libary:** The class that is an aggregation of all of the Scenes.
- **Scene:** The Scene class contains information about a Scene’s budget and the Roles that compose it.
- **Role:** Every Role has information about its rank, whether or not it is a Scene Role or a Room Role (main/extra), and whether or not it has been taken already (when an Actor associates with it).
- **Casting Office:** This class contains all of the information regarding rank upgrading. When an Actor wishes to upgrade, it uses that Actor to check their eligibility and then based on user input, upgrades that Actor.
- **Dice** A class that contains a random number generator for [1...6].

2 Body

When a user starts Deadwood, the program begins with the Number of Players use case (see **section 3.7**). Deadwood asks for the number of players (1...8), the user inputs the number of players, either correctly or incorrectly. When input incorrectly, it redisplays the selections and asks again. (This occurs whenever an incorrect input is given, for all cases.) When correctly selected, Deadwood sets the appropriate amount of days for itself and creates the specified number of Actor objects with the correct rank, credits, and dollars based on the case. Deadwood creates the Board, which assigns random Scenes using the Scene Library to the Rooms and marks these Scenes as used (useScene).

The players can then begin their turns. For each player at every turn, Deadwood gives the user the option to upgrade in rank at both the beginning and the end of every turn. This is the Upgrade use case (see **section 3.1**) If a user chooses to upgrade, their Actor requests an upgrade from the Casting Office. The Casting Office then looks at that Actor's rank, credits, and dollars and determines what rank(s) the Actor could upgrade to: the rank must be greater than the Actor's current rank and the Actor must have enough credits and/or dollars for that rank. If a rank upgrade is viable for the Actor, the Casting Office then asks the user which upgrade they would like. Then, based off of user input, the Casting Office upgrades the Actor's rank and decreases their credits or dollars by the specified amount. (If an upgrade is not viable, the process is canceled.)

After this optional upgrade, if a player has not already taken a Role (and their Room has not ended, ending the Role), then they can choose to move from their current room to another room, which is the Move use case (see **section 3.2**) To do so, when Deadwood prompts the user to move and the user selects move, their Actor class looks at its associated Room (currentRoom) and finds its adjacent Rooms (adjRooms), then prompts the user to select one of these Rooms. When the user inputs the selection, Actor removes itself from its associated Room by using updateActors (which edits the currentActor array in a Room), and then changes it's currentRoom to the selection. HOWEVER, the Actor does not place itself in that Room's currentActor array UNTIL they take a Role- this is so that the Actor is not given bonus money when they have not taken a role. Once an Actor has moved into this Room, Deadwood will prompt the user to take a Role in the Room. In a text-based implementation, whether or not the Room has been previously visited is not as important as in a graphical interface implementation; Deadwood will only "reveal" the scene once a user has moved into the room by prompting them with a "Take a Role" option.

If the user decides to opt towards taking a Role, this implements the Take a Role use case (see **section 3.3**). The user's Actor class looks at its currentRoom, that Room's Roles, the Room's currentScene, and that Scene's Roles. The Actor decides what Roles it is eligible for (depending on their rank and the Role's rank, and whether the Role is taken or not), then prompts the User to select a Role or not (which cancels the process). The prompt will include what Roles it is eligible for, whether those Roles are on the Scene card or not (main or extra), and the Scene's budget. If the user selects a Role, their Actor associates itself with that Role (currentRole), updates that Role to let it know its taken (takeRole), and places itself in that Room (updateActors).

If the user has already taken a Role in a previous turn, then they have the choice to either rehearse or act. If they choose to rehearse, then their Actor increases their rehearsals by one. If they choose to act, then Act use case begins (see **section 3.4**). The Actor class uses the Dice class to roll, then looks at the budget of their currentRoom's currentScene, then adds the roll to their rehearsals and compares it to the budget. If the budget is greater, then the Act was unsuccessful and Actor looks at its associated Role to see if it was extra or not. If it was extra, the Actor increases its dollars by one. If not, their move ends. If the Act was successful (roll plus rehearsals was equal to or greater than the budget of the scene), then the Actor class "takes the shot" from its currentRoom (updateShots) and looks at its currentRole to see if it was extra or not. If it was extra, the Actor increases its dollars by one and its credits by one. If it was main, the Actor increases its credits by two.

When the Actor updates the currentShots of the Room (updateShots), the Room checks to make sure it was not the last shot. If it was the last shot, then first the Bonus use case is implemented (see **section 3.6**). The Room looks at all the Roles in the associated Scene to see if any are taken. If none are taken, then there were no main actors and the Bonus process ends. If there was at least one Role taken, then the Room looks the currentActors array to find all Actors in the Room with a Role (recall that only Actors with a Role in the

Room are stored in this array), then looks at their associated Role to determine which are extra (if there are any). Room increases the dollar amount of Actors with extra Roles by their rank. Actors that are main are temporarily stored by the rank of their associated Role. The Room then looks at the budget of the Scene and uses Dice to roll an amount of dice equaling the budget of that Scene and temporarily stores this information. Room then increases the Actor's dollar amount by the roll number from the rolls (highest rank gets highest roll, next highest rank gets next highest roll, and wrapping around until all dice rolls are accounted for).

After the Bonus use case, the End Room use case occurs (see **section 3.5**). The Room "ends" itself by resetting its currentShots to shots, then removing all Actors from their associated Role (removeRole in Actor), and finally removing all Actors from its currentActors array. The Room then updates the number of Rooms in Board (updateNumRooms) by subtracting one.

When updateNumRooms is called in Board, it subtracts from numRooms and checks that numRooms is greater than one. If it is not (if it equals one), then it is the End Day use case (see **section 3.8**). Board calls endDay, updates the day in Deadwood (updateDay). Deadwood then checks to make sure that that was not the last day. If it was not, Deadwood resets the Board, which calls Scene Library to associate random unused Scenes to the Rooms and removes any Role associated with an Actor (which sets that Role's taken status back to 0), resets all Room currentActor arrays, and changes all Actors' currentRoom to the Trailer Room object. The day then starts anew. However, if it was the last day, then Deadwood calls endGame, which looks at each Actor's rank, credits, and dollars and scores them, then announces the winner to the user. The game process ends.

3 Use Cases and Sequence Diagrams

3.1 Upgrade:

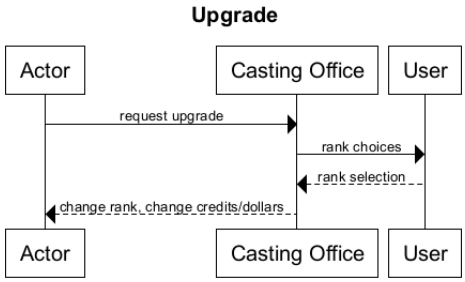
3.1.1 Main flow:

1. Asks if actor wants upgrade
2. Yes
3. Casting Office displays possible upgrade based on rank, credits, and dollar, or the option to cancel
4. User inputs upgrade selection from choices
5. User is upgraded; rank is increased, credits/dollars decreased based on choice
6. Upgrade process ends

3.1.2 Alternate Flow:

2. A. No
   a. Upgrade process ends
3. A. Casting office finds no upgrades possible based on credits/dollars and rank, displays message
   a. Upgrade process ends
4. A. User improperly inputs selection
   a. Displays error message, redisplays options, asks for input. Go back to Step 4
   B. User cancels upgrade
   a. Upgrade process ends

3.1.3 Sequence Diagram:



3.2 Move:

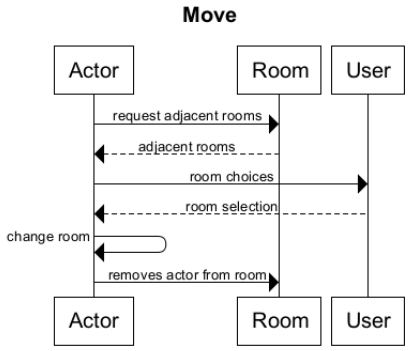
3.2.1 Main flow:

1. Displays adjacent rooms that are possible to move to
2. User inputs choice from selection
3. User is moved to that room
4. Room unvisited; flips scene card
5. Move process ends

3.2.2 Alternate flow:

2. A. User choose not to move
   a. Move process ends
   B. User improperly inputs selection
   a. Displays error message, redisplays options, asks for input. Go back to Step 2
4. A. Room is previously visited
   a. Move process ends

3.2.3 Sequence Diagram:



3.3 Take a Role:

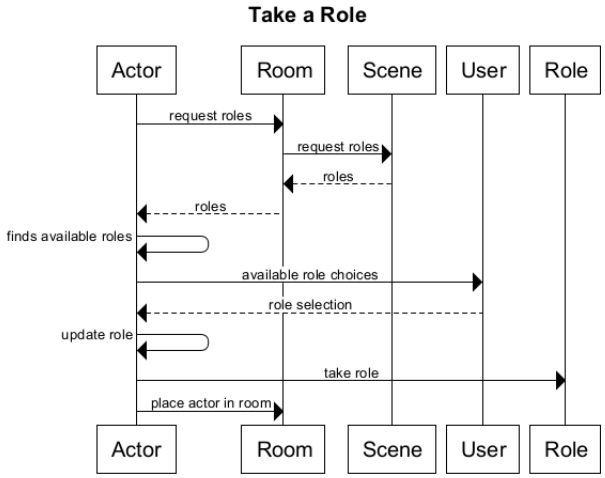
3.3.1 Main flow:

1. Displays possible roles based on rank, room, and scene in room, and cancel option
2. User inputs role selection
3. Role process ends

3.3.2 Alternate flow:

1. A. No roles possible for room or scene
   a. Role process ends
2. A. User improperly inputs selection
   a. Displays error message, redisplays options, asks for input. Go back to Step 2
   B. User cancels
   a. Role process ends

3.3.3 Sequence Diagram:



3.4 Act:

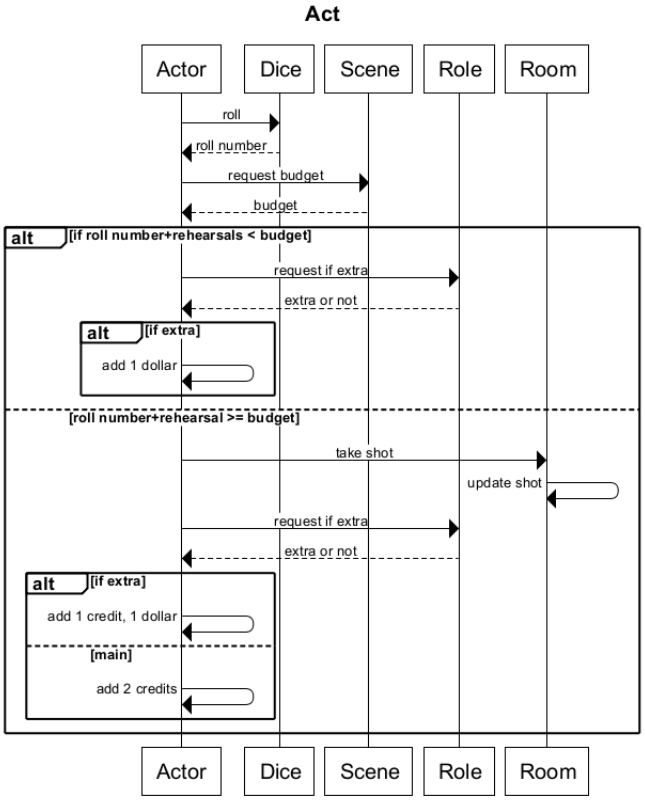
3.4.1 Main flow:

1. User rolls the dice
2. Roll is high enough; act is successful
3. User gets shot, it is not the last shot in the room
4. Act process ends

3.4.2 Alternate flow:

2. A. Roll is not high enough, act is unsuccessful
   a. Actor has main role
      I. Act process ends
   b. Actor has extra role
      I. Actor receives one dollar
      II. Act process ends
3. A. Shot is the last shot in the room
   a. Room ends. See Bonus, **section 3.6**
   b. See End Room, **section 3.5**
   c. Act process ends

3.4.3 Sequence Diagram:



3.5 End Room:

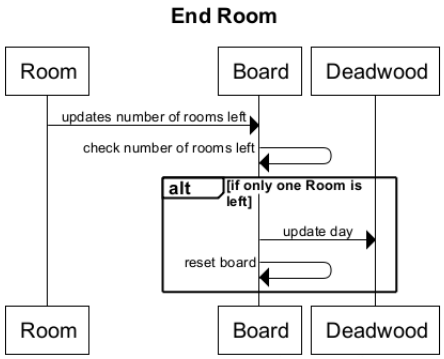
3.5.1 Main flow:

1. Room that ends is not the second-to-last room
2. Room end process ends

3.5.2 Alternate flow:

1. A. Room is second-to-last room
   a. See End Day, **section 3.8**

3.5.3 Sequence Diagram:



3.6 Bonus:

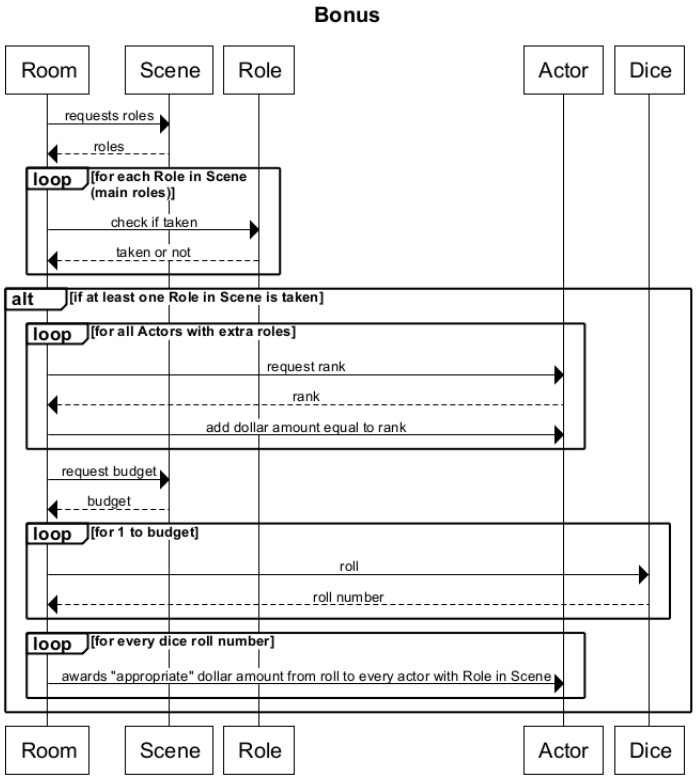
3.6.1 Main flow:

1. There is an actor with a main role (a role on the scene card)
2. A number of dice are rolled based on the budget of the scene
3. Dollar amounts are rewarded to the main actors based on the rank of their role and the numbers on the dice rolled
4. Extra actors get dollar amounts based on their rank
5. Bonus process ends

3.6.2 Alternate flow:

1. A. There are no main actors
    a. Bonus process ends
4. A. There are no extras
    a. Bonus process ends

3.6.3 Sequence Diagram:



3.7 Number of players:

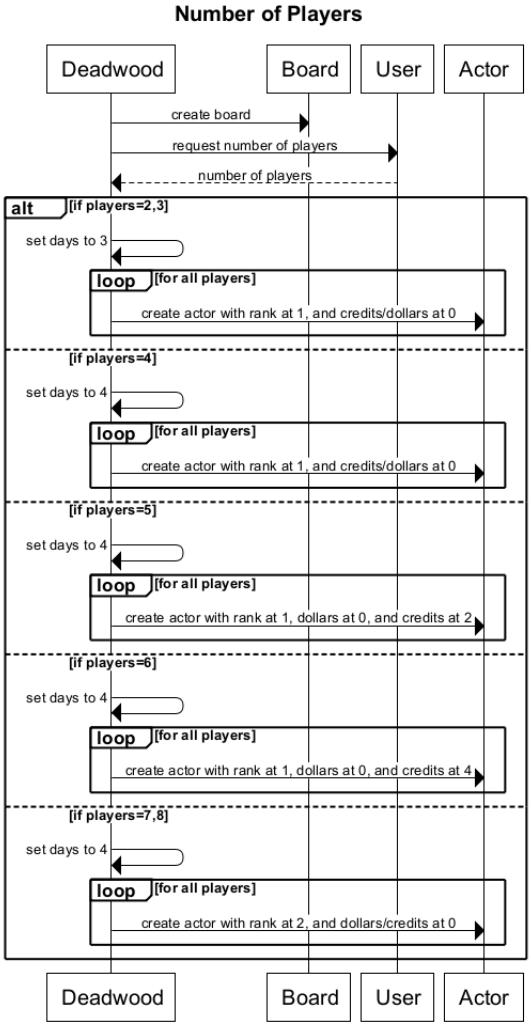
3.7.1 Main flow:

1. Asks for user input for number players
2. There are four players; four actors are created, number of days is set to 4, rank starts at 1, credits/dollars start at 0
3. Number of players process ends

3.7.2 Alternate flow:

2. A. There are two or three players
   a. Two or three actors are created, number of days is set to 3, rank starts at 1, credits/dollars start at 0
   B. There are five players
   a. Five actors are created, number of days is set to 4, rank starts at 1, dollars start at 0, credits start at 2
   C. There are six players
   a. Six actors are created, number of days is set to 4, ranks starts at 1, dollars start at 0, credits start at 4
   D. There are seven or eight players
   a. Seven or eight actors are created, number of days is set to 4, ranks starts at 2, dollars start at 0, credits start at 0

3.7.3 Sequence Diagram:



3.8 End Day:

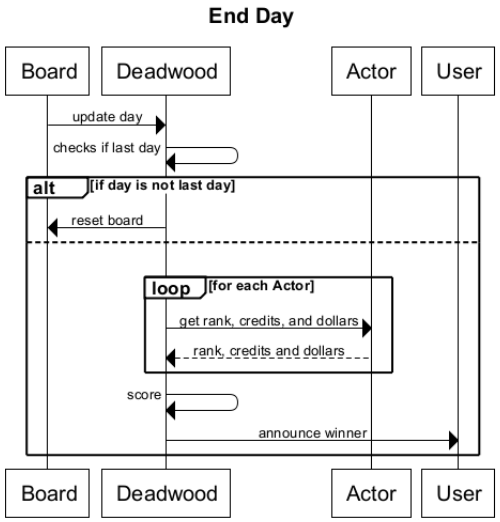
3.8.1 Main flow:

1. Day is not the last day
2. Board is reset, players move to trailer
3. End day process ends

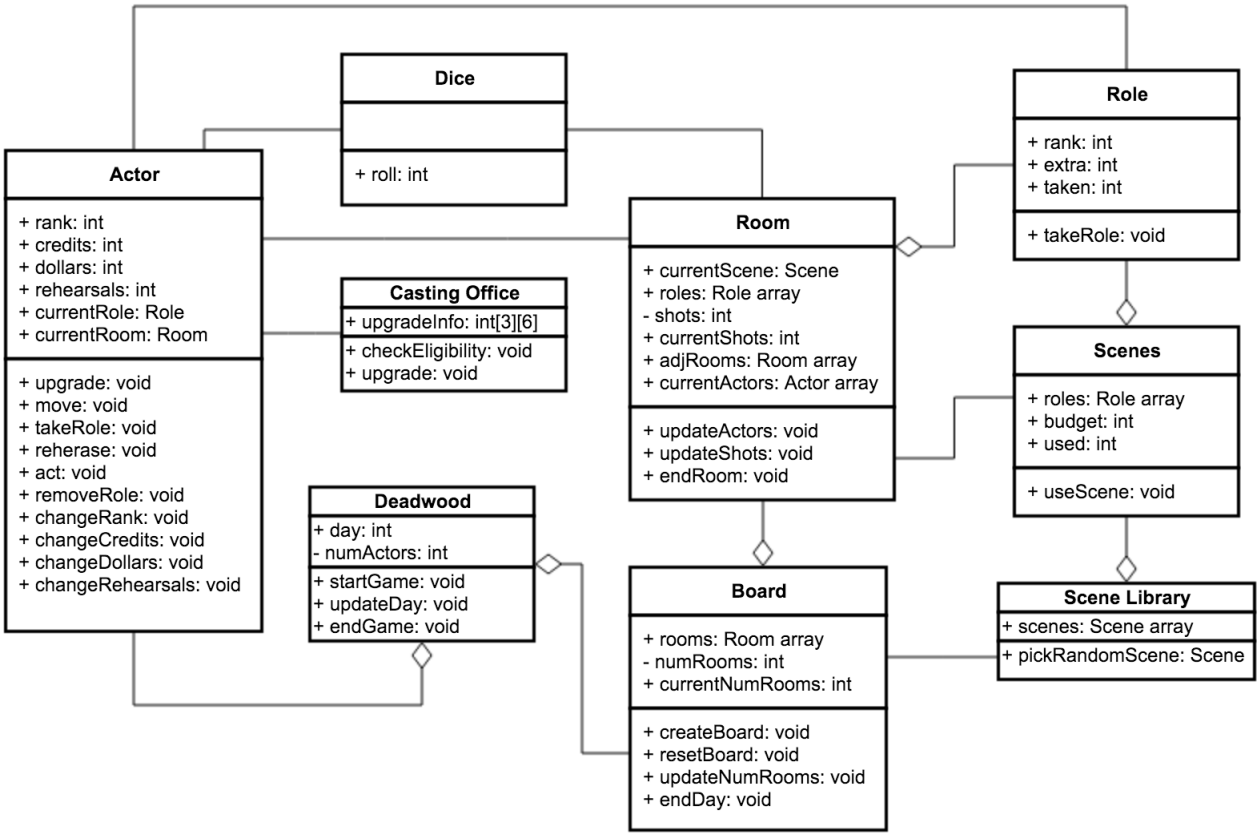
3.8.2 Alternate flow:

1. A. Day is the last day
    a. Game ends; score and announce winner
    b. Game process ends

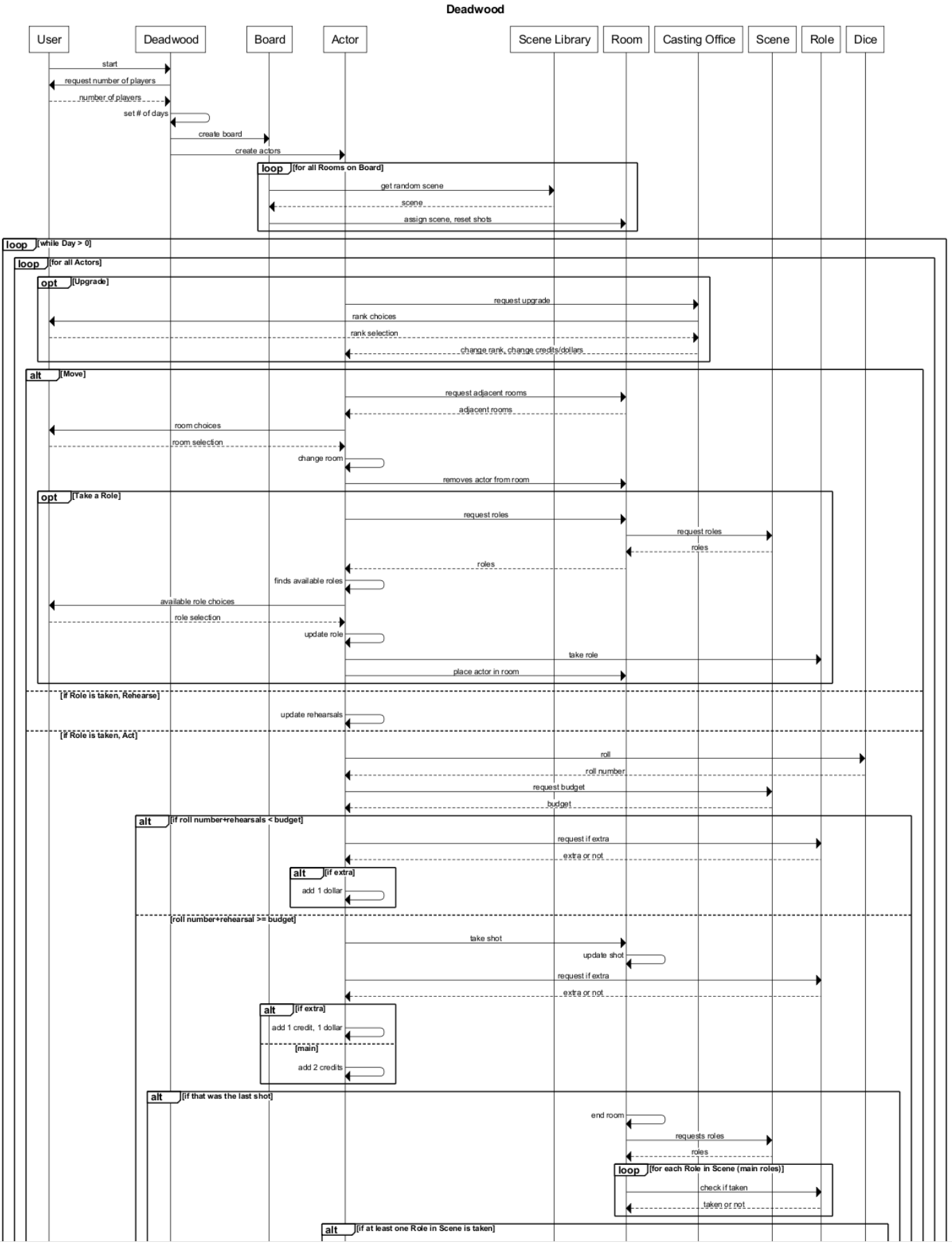
3.8.3 Sequence Diagram:

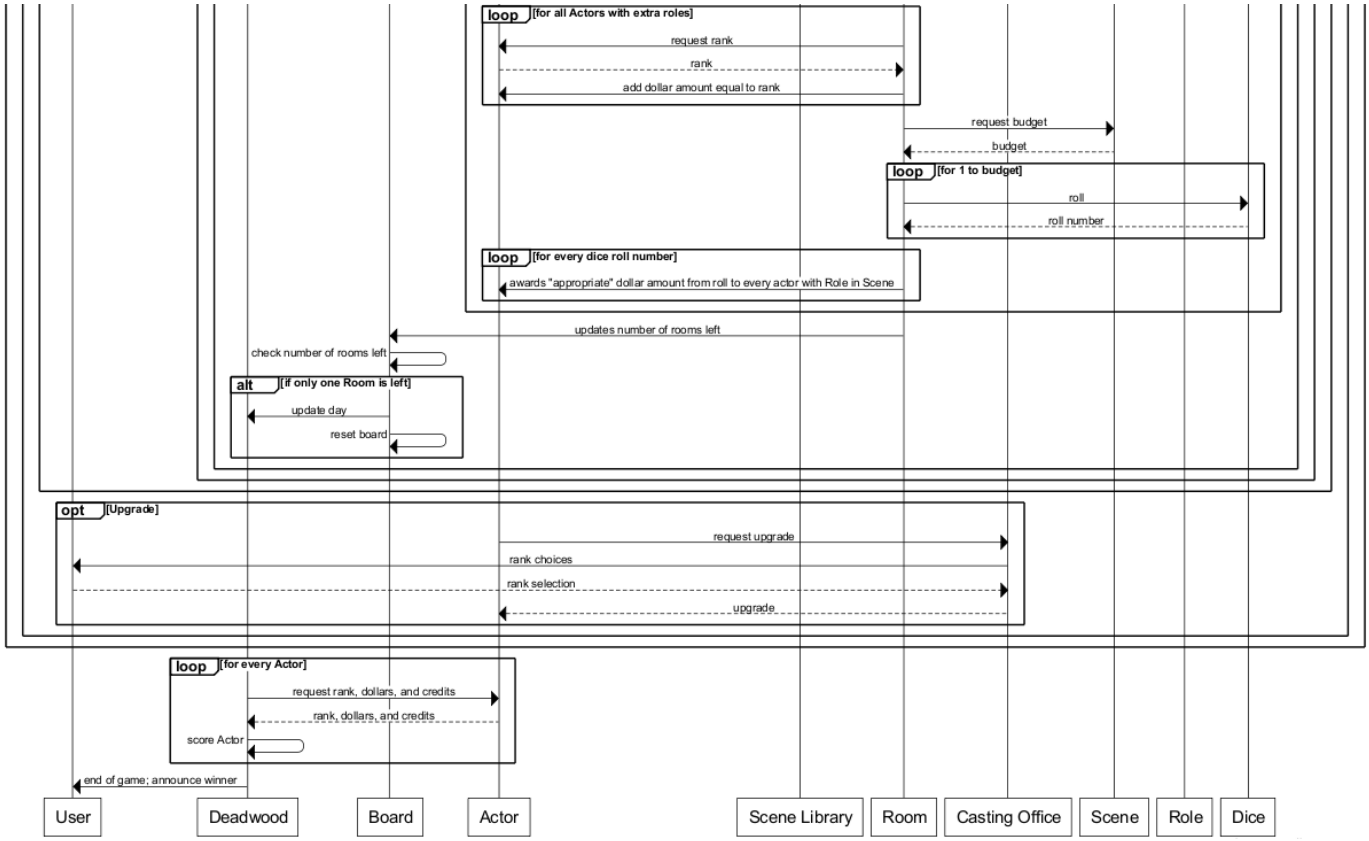


4 UML Diagram



5 Overall Sequence Diagram




loop [for all Actors with extra roles]
request rank
rank
add dollar amount equal to rank
request budget
budget
loop [for 1 to budget]
roll
roll number
loop [for every dice roll number]
awards "appropriate" dollar amount from roll to every actor with Role in Scene
updates number of rooms left
check number of rooms left
alt [if only one Room is left]
update day
reset board
opt [Upgrade]
request upgrade
rank choices
rank selection
upgrade
loop [for every Actor]
request rank, dollars, and credits
rank, dollars, and credits
score Actor
end of game; announce winner
User
Deadwood
Board
Actor
Scene Library
Room
Casting Office
Scene
Role
Dice