Alex Petralia and Shannon Melious
Assignment 2 Class Diagram: DeadWood

Moderator

-numPlayers: int
-allPlayers: Player[ ]
-allScenes: ArrayList<Scene>
-daysLeft: int

-calcScores(Player[ ]): int [ ]
-calcWrapBonus(Player [ ]): int[ ]
-useScene( ): Scene
-Moderator

Player

-diceColor: String
-rank: int
-fame: int
-money: int

+setMoney(int): void
+act(): boolean
+setRank(int): void
+Player

0..1

2..8

Deadwood

(No Attributes)

+main: void

Role

+assignedPlayer: String
+reqRank: int
+line: String

+Role
+checkAvailability(): boolean
+getRoleName(): String

0..1

Room

-roles: ArrayList<Role>
-shots: ArrayList<Shot>
-currentScene: Scene
-adjacentRooms: ArrayList<Room>

+getRoles(): ArrayList<Role>
+getShots(): ArrayList<Shot>

name

Scene

-title: String
-roles: ArrayList<Role>
-curRoom: Room

+advanceScene(): void
+setRoles(): void
+Scene

1..3

0..1

0..1

1..3

Upgrade

+xCoord: int
+yCoord: int
+hCoord: int
+wCoord: int
+currency: String

+Upgrade
+setX: int
+setY: int
+setCurrency(String): void

Shot

+xCoord: int
+yCoord: int
+hCoord: int
+wCoord: int

+Shot
+setX: int
+setY: int

0..1

0..1

ParseBoardXML

(No Attributes)

+getDocFromFile(): Document
+createRooms(): ArrayList<Room>

ParseCardXML

(No Attributes)

+getDocFromFile(): Document

+createSceneDeck(): Document

Reasons behind our Class Diagram:

First of all we tried to design our system so a lot of the classes were containers for information regarding the class's purpose. For example the class "Player" holds all information about a specific player; including their fame number, amount of total money, the scene they are in, etc. Classes similar to these responsibilities would be the Room, Role, Scene and Shot classes. The Moderator class is used for operations on the data in contrast to these classes that act as containers for information. The Moderator class is responsible for calculations using one object or multiple objects as input for these functions such as calculating the success or failure of acting a role.

Some notable relationships between these classes include the Shot class being an aggregation to Room as Shots are contained in a Room. Deadwood is an association of Player as the Deadwood class depends on the Player objects to execute the cycle of the game. Also the upgrade class depends on the information of the player so an upgrade can go through for the ranking system. As for the parsers; Room inherits ParseBoardXML and Scene inherits parseCardXML as the non-parser classes inherit the needed data the parsers output.

To implement Cohesion we made a lot of our classes specifically for being used as containers of data like the Player class. In rare cases these classes will have calculations but calculations are mostly done in the Moderator and Deadwood class. As for coupling most of the classes have very low coupling since calculations of the data is distributed quite evenly. The Moderator class handles a lot of the reward calculations and the Deadwood class handles a lot of the calculations related to the day cycles of the game like making sure a Player has the required rank for a role. The only bad coupling example we have is the Deadwood class. The main method in this class can be distributed more in a later version so the Deadwood class isn't so dependent on other classes to compute calculations.