Exercise 4:
The Settlers of Catan

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Software Engineering Design & Construction
WS 2016/17 - Dr. Michael Eichberg, M.Sc. Matthias Eichholz

Although this exercise is not graded, it is highly recommended to also do them on your own. Just looking at a solution is much easier in comparison to actually coming up with it. Support can be found in the forum:
`https://www.fachschaft.informatik.tu-darmstadt.de/forum/viewforum.php?f=234`

Introduction

In this exercise, you will design a simplified version of the board game *The Settlers of Catan*[1]. Make yourself familiar with the game rules as specified in `http://www.catan.com/files/downloads/catan_5th_ed_rules_eng_150303.pdf`. We assume a reduced set of game components, i.e., the following components are **not** included:

- Sea frame pieces
- Harbor pieces
- Development cards
- *Longest Road* or *Largest Army* cards
- Robber

You can ignore all rules that refer to any of theses cards or pieces. For example, nothing happens if a player rolls a 7. Your task is to design The Settlers of Catan as a computer game. Adhere to the five SOLID principles (Single Responsibility, Open Closed, Liskov Substitution, Interface Segregation and Dependency Inversion).
First, think about a design for your class hierarchy. Decide for which game components you need classes and which can be omitted or represented differently for a computer simulation. There should be two kinds of players, human and computer players. Second, think about how a turn of a game of two proceeds. Imagine it is the turn of a computer player in which it should

- Attempt to trade a non-zero amount of resources with the other player (you choose the exact trade), and
- Build a road and a settlement if it has the required resources[2].

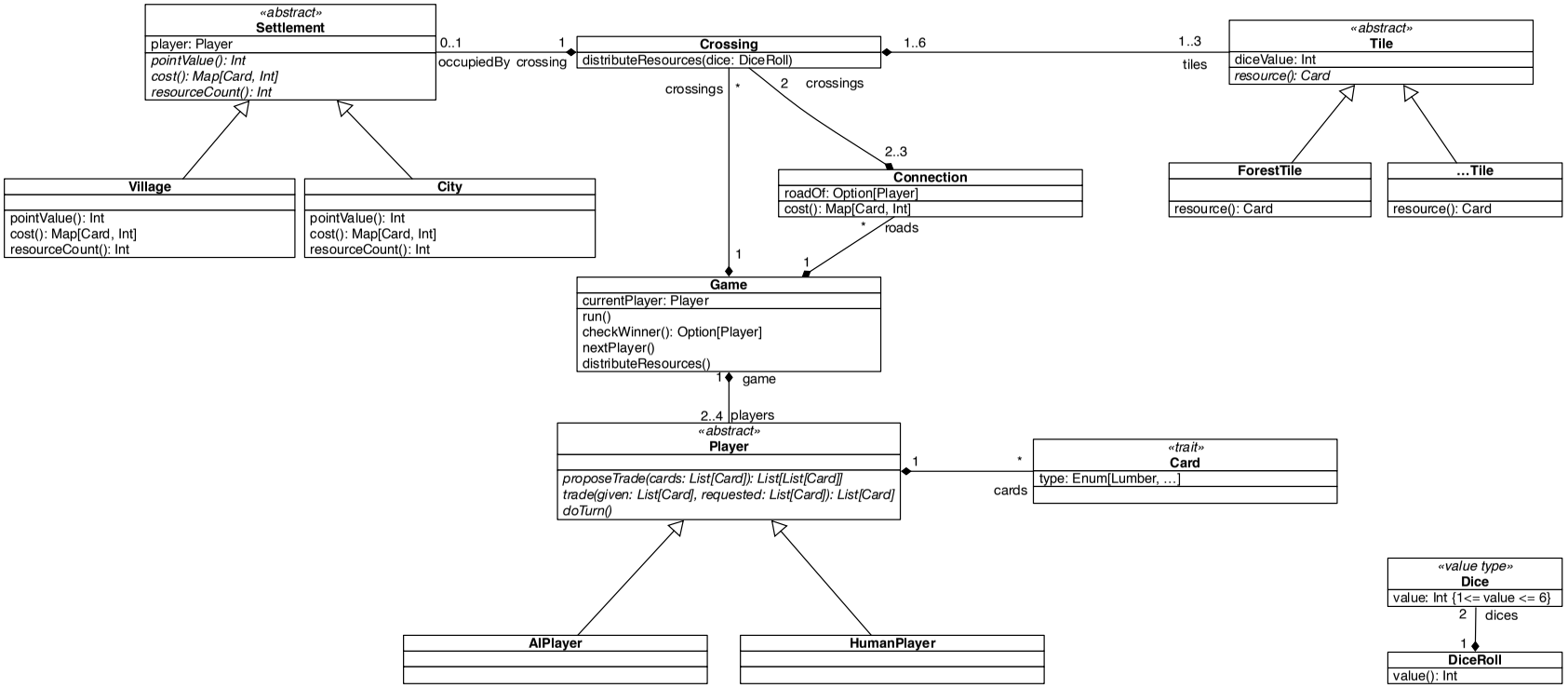
Think about the interface between a player and the game and how they interact with each other. Note that the game logic needs to prevent cheating (doing things in the wrong order, building things the player doesn't have the resources for etc). Reduce the number of ways a player can cheat by carefully designing the interface and interactions between the game logic and the player.
Since designing software is an incremental process, you might need to adapt your class design to address any deficiencies you discover while thinking about the turn design.

Task 1 Class design

Draw a UML 2.0 class diagram of your class hierarchy. Give inheritance relationships and associations (do not forget to provide correct multiplicities!). Also provide methods you need to simulate a turn.

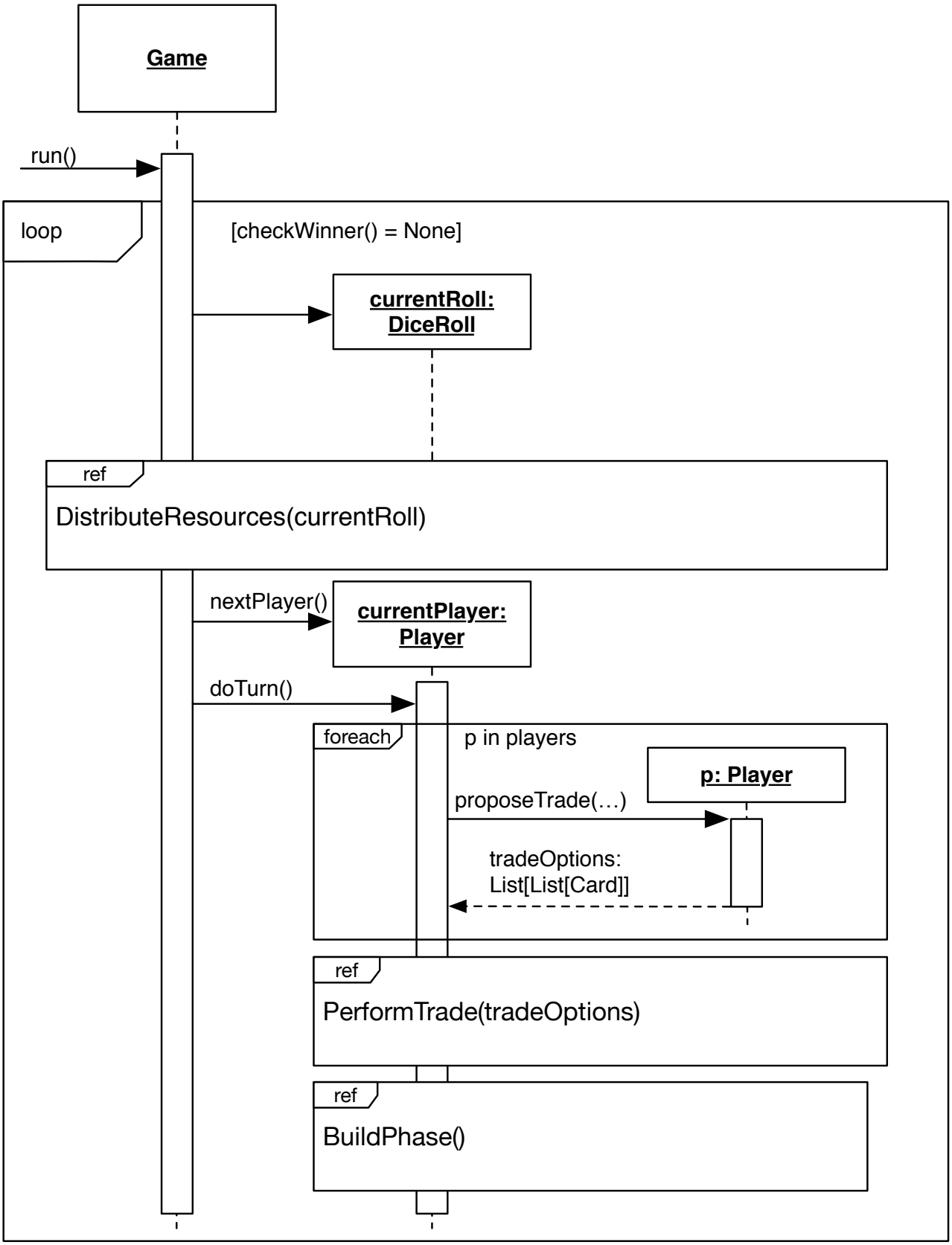
[1] `http://www.catan.com/`

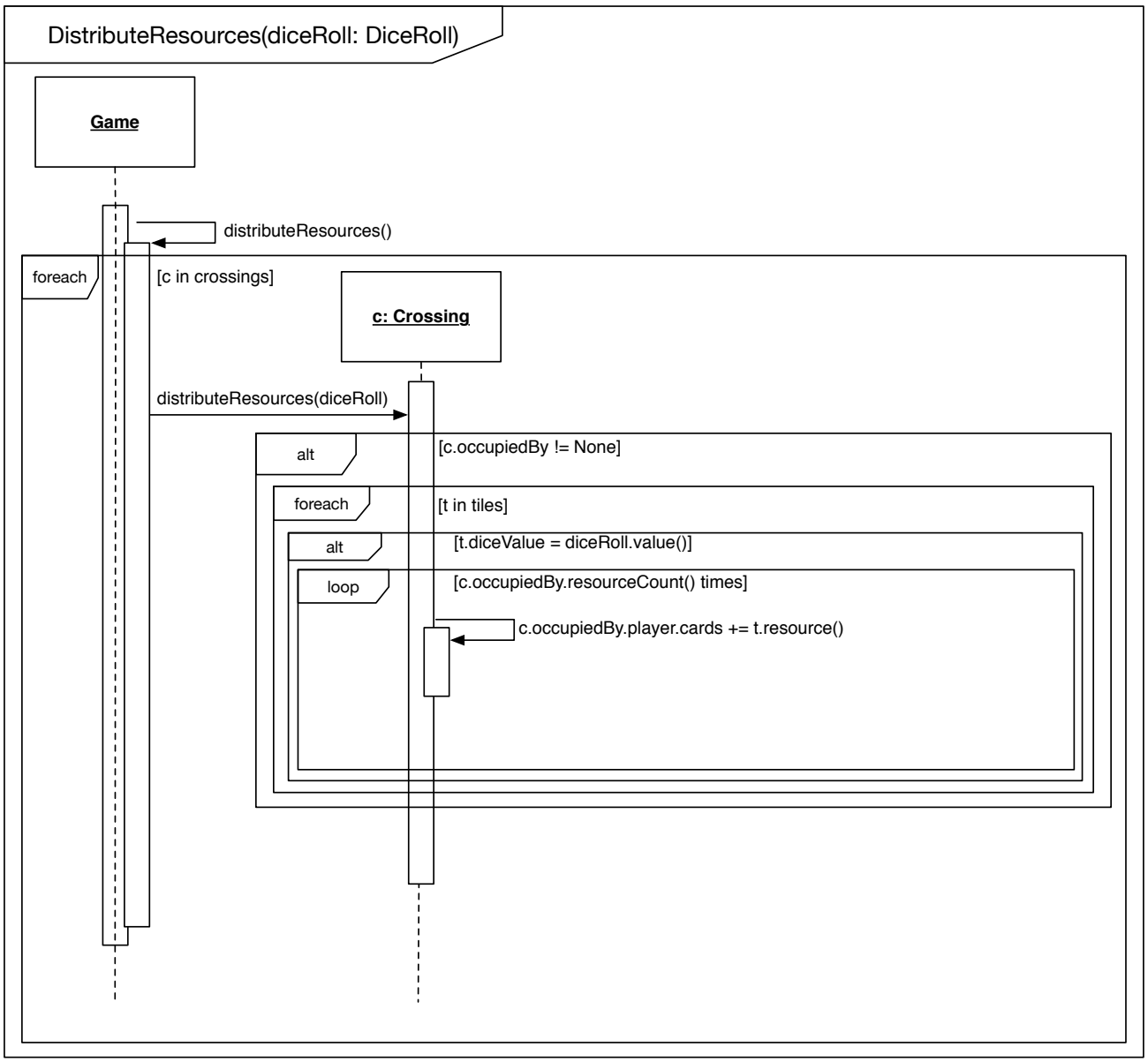
[2] You might want to read `http://www.ibm.com/developerworks/rational/library/3101.html` for how to model guards and conditions in a sequence diagram.


«abstract»
Settlement
player: Player
pointValue(): Int
cost(): Map[Card, Int]
resourceCount(): Int
0..1
occupiedBy
1
crossing
Crossing
distributeResources(dice: DiceRoll)
1..6
1..3
tiles
«abstract»
Tile
diceValue: Int
resource(): Card
crossings
*
2
crossings
2..3
Village
pointValue(): Int
cost(): Map[Card, Int]
resourceCount(): Int
City
pointValue(): Int
cost(): Map[Card, Int]
resourceCount(): Int
Connection
roadOf: Option[Player]
cost(): Map[Card, Int]
*
roads
ForestTile
resource(): Card
…Tile
resource(): Card
1
1
Game
currentPlayer: Player
run()
checkWinner(): Option[Player]
nextPlayer()
distributeResources()
1
game
2..4
players
«abstract»
Player
proposeTrade(cards: List[Card]): List[List[Card]]
trade(given: List[Card], requested: List[Card]): List[Card]
doTurn()
1
*
cards
«trait»
Card
type: Enum[Lumber, …]
AIPlayer
HumanPlayer
«value type»
Dice
value: Int {1<= value <= 6}
2
dices
1
DiceRoll
value(): Int

Task 2 Turn design

Draw a UML 2.0 sequence diagram for the sample turn above in order to demonstrate the interactions between objects of the different classes. For simplicity reasons, assume that all checks against cheating pass. Your diagram should focus on the interactions between the game and a player. For example, it is not important to go into detail *how* the player finds the position of where to build a road, but it needs to get a handle to that position from some method call (model the AI as a black box). There should be no free variable names in your diagram.


Game
run()
loop
[checkWinner() = None]
currentRoll:
DiceRoll
ref
DistributeResources(currentRoll)
nextPlayer()
currentPlayer:
Player
doTurn()
foreach
p in players
p: Player
proposeTrade(…)
tradeOptions:
List[List[Card]]
ref
PerformTrade(tradeOptions)
ref
BuildPhase()


DistributeResources(diceRoll: DiceRoll)
Game
distributeResources()
foreach
[c in crossings]
c: Crossing
distributeResources(diceRoll)
alt
[c.occupiedBy != None]
foreach
[t in tiles]
alt
[t.diceValue = diceRoll.value()]
loop
[c.occupiedBy.resourceCount() times]
c.occupiedBy.player.cards += t.resource()

Task 3 SOLID

Assess your design with respect to the SOLID principles. Shortly describe the responsibilities of each class. Write for each SOLID principle, how your design honors or violates it and why in the latter case this might not be a problem. For the Open Closed Principle, name two extensions your design makes possible. Think about whether your design shows signs of rigidity and fragility.

Solution

Task 3.1 Responsiblities

- **Dice:** Represents a single dice (a value between 1 and 6). Produces random values in that range.
- **DiceRoll:** Represents a dice roll in settlers. It consists of two dice and produces a fair two dice roll.
- **Game:** Contains the main game logic and the board and manages turns.
- **Tile:** A single board tile. Has a dice value (pays out on that value) and an associated resource.
- **Crossing:** Connects up to three tiles and roads. Can be occupied by a settlement.
- **Connection:** A connection between two crossings. Can be occupied by a player.
- **Settlement:** A settlement belongs to a player and is either a village or a city.
- **Village:** A settlement that produces one resource.
- **City:** A settlement that produces two resources.
- **Player:** Players can be either human players or AI players. Players are responsible for handling turns (trading, building...).
- **Card:** Awarded for villages and cities producing resources. Used for building.

Task 3.2 SOLID

- **SRP:** The SRP is violated in the Player class, since the player has multiple responsibilities - trading and building. However, this violation is not critical, since the game rules are represented more naturally in this design.
- **OCP:** The design mostly honors the OCP principle. It is open for extension on both `Tile` and `Settlement`, but not on `Card`.
- **LSP:** LSP holds for the inheritance relations in the class diagram.
- **ISP:** There are no clients of any interface / trait in our design that are forced to depend on methods they do not use, so we adhere to ISP.
- **DIP:** We adhere to DIP, as for example `Game` does not depend on `HumanPlayer`, but only on `Player` which is a abstract class which all kinds of players have to inherit from. The same holds for `Settlements` and `Tiles`.