Labo 3

Génie Logiciel

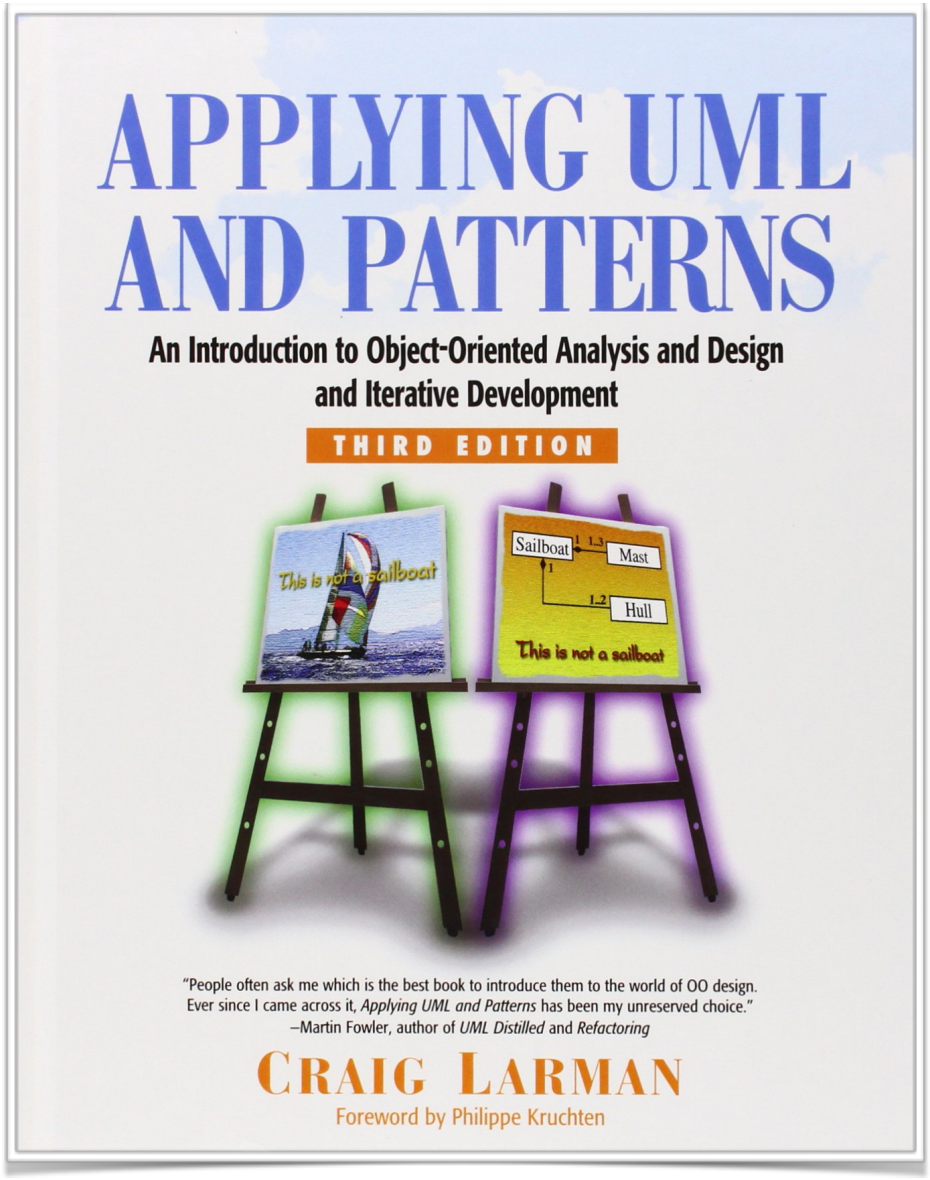
Buts du laboratoire

- Mettre en oeuvre les deux premières itérations du jeu de monopoly de « Applying UML and Patterns »
- Utiliser de nouvelles annotations et assertions dans JUnit
- Utiliser Git, Maven et Travis en les configurant vous-même pour que Travis fasse tourner vos tests unitaires à chaque pull request.

Itération 1

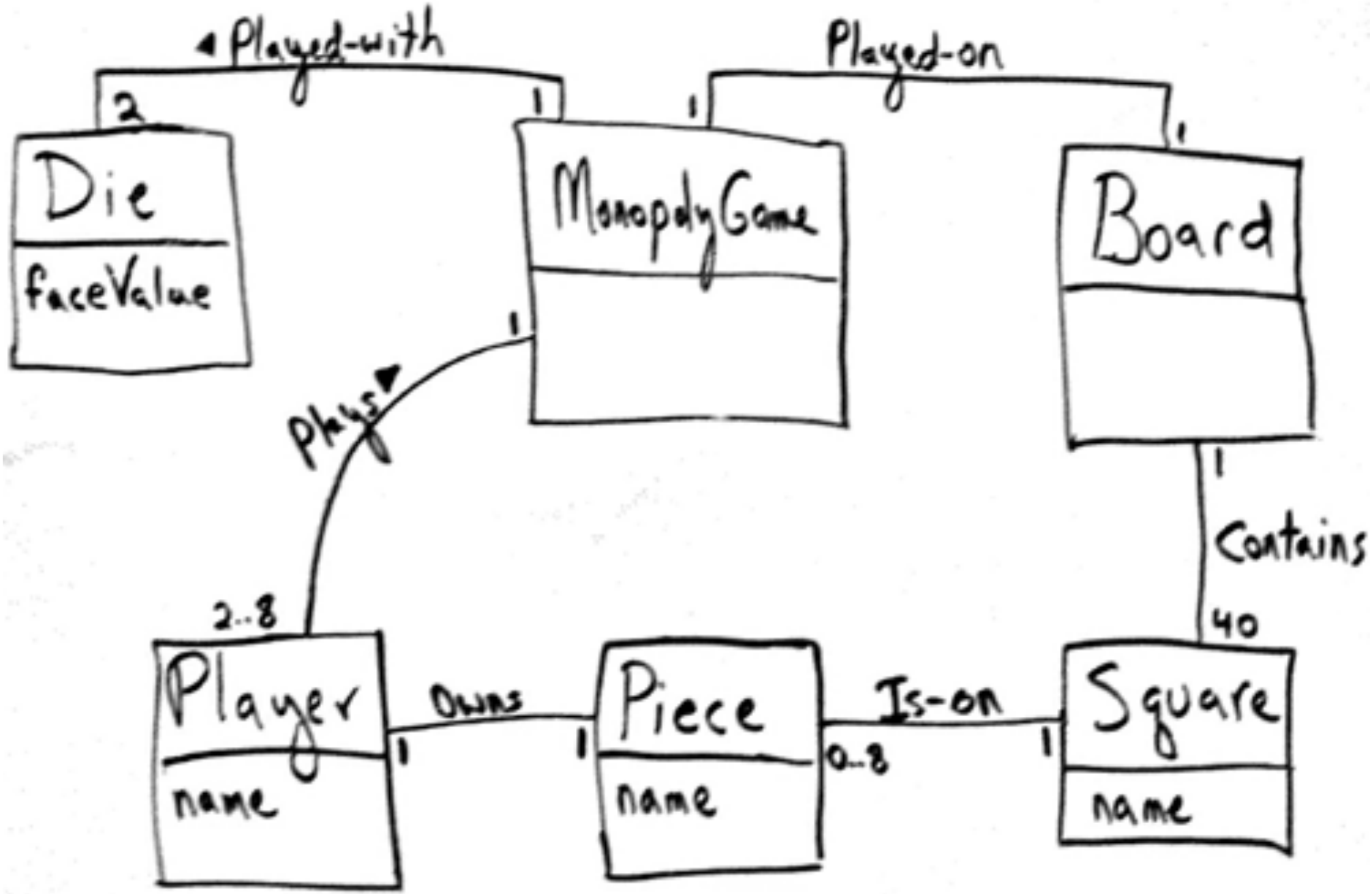

MONOPOLY
The Classic Edition
GO
BOARDWALK
PARK PLACE
CHANCE
MONOPOLY

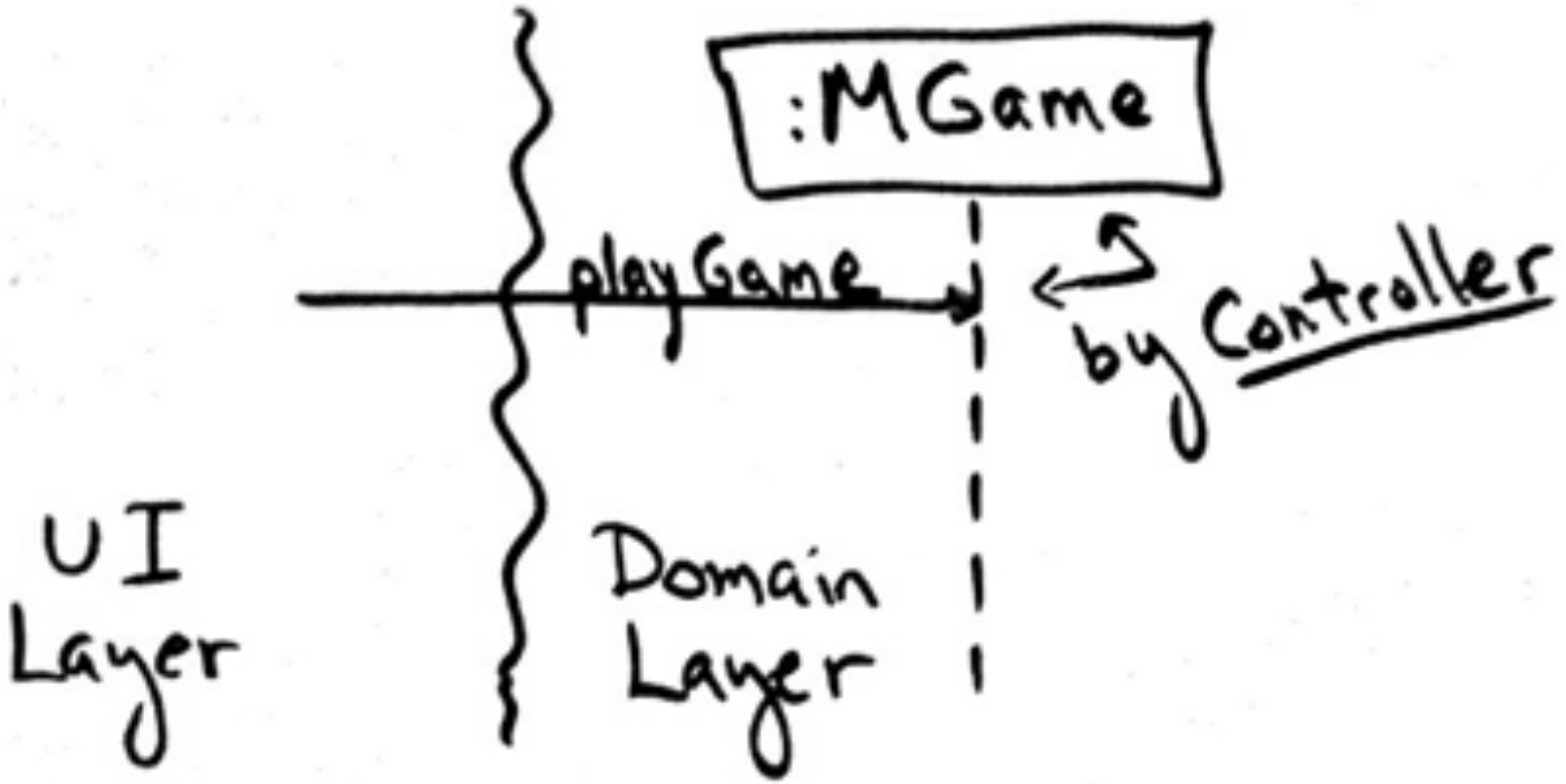
- Chapitre18.5 - Use case realization for the monopoly game

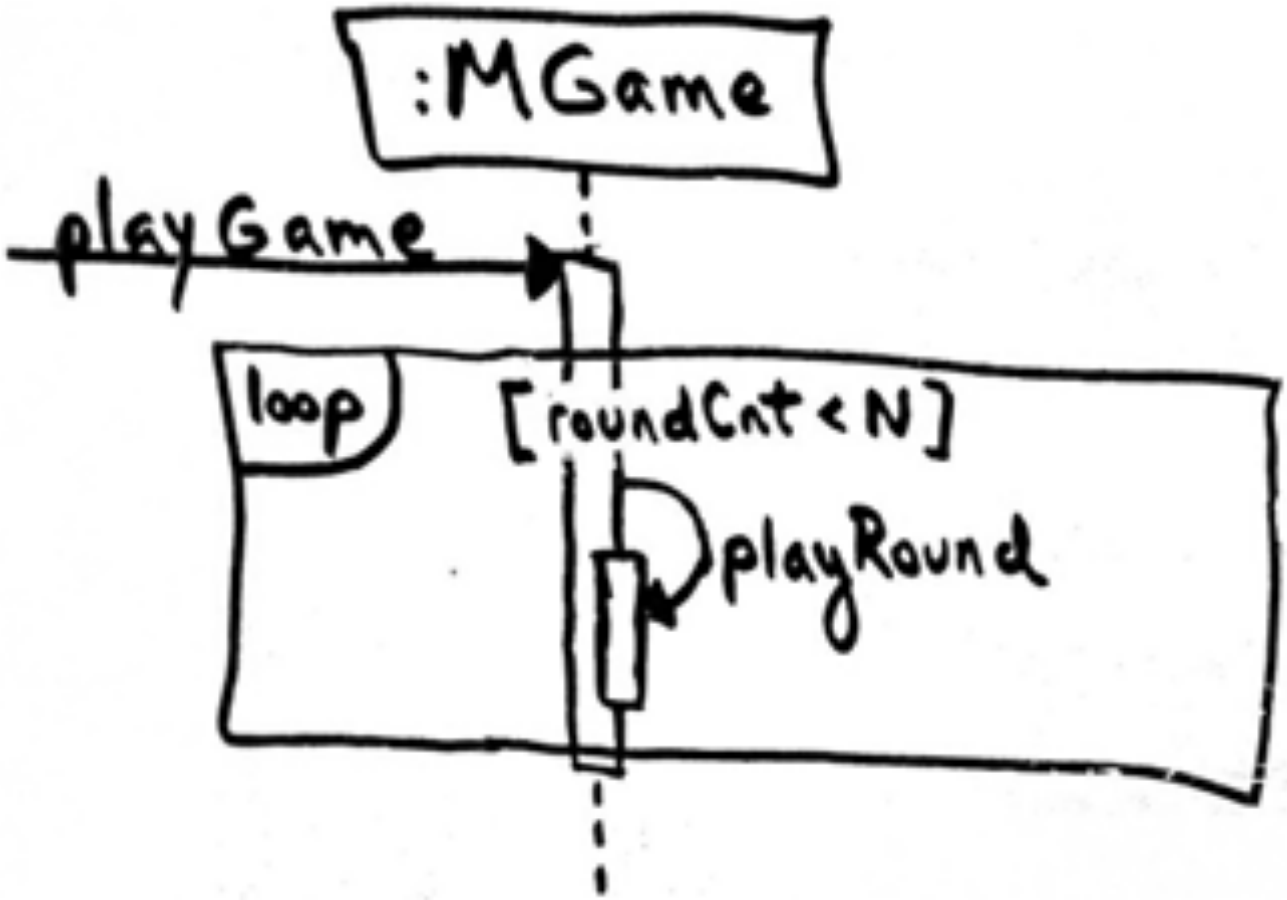


Scénario simplifié (approche itérative)

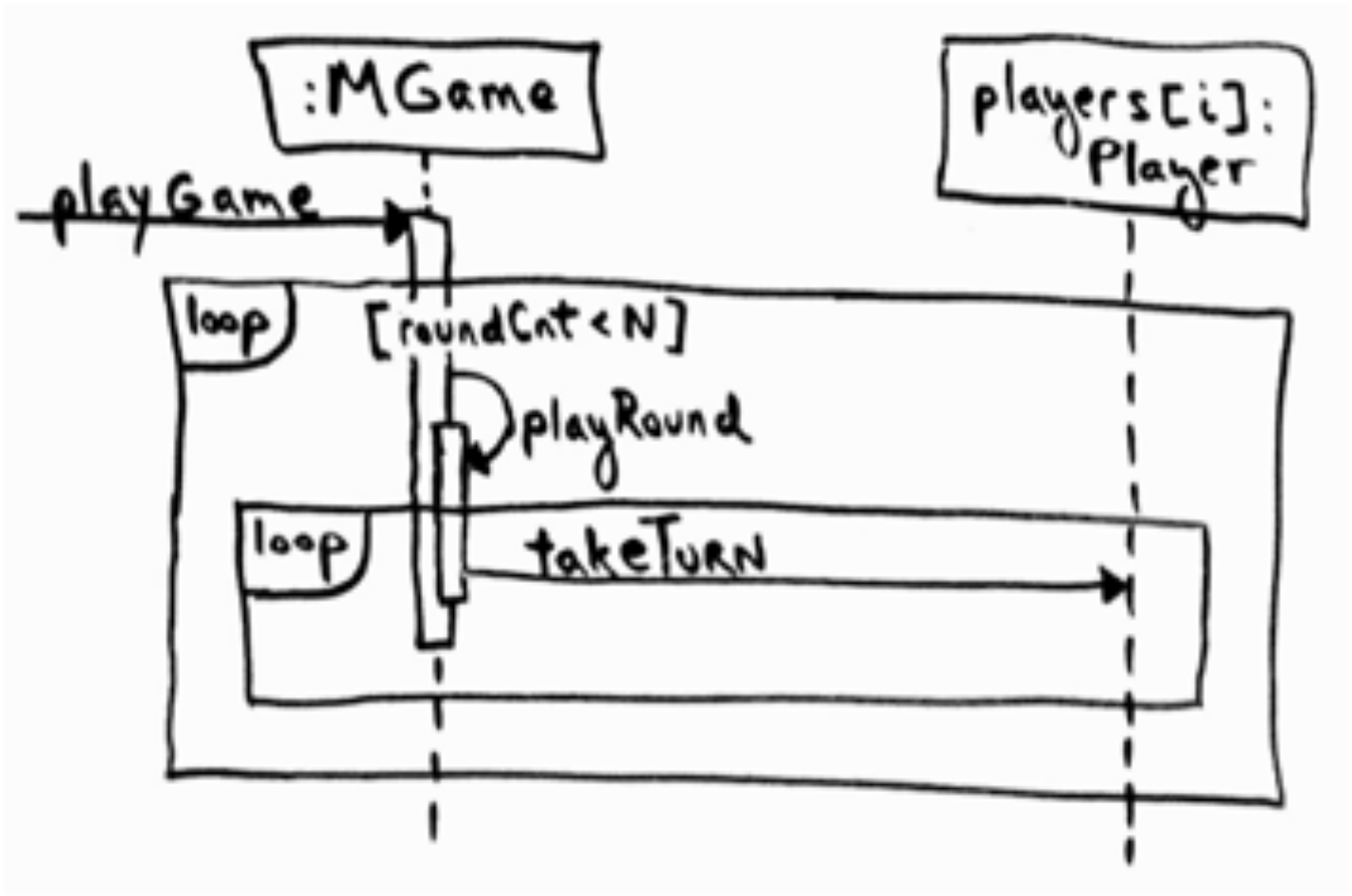
- Players move around the squares of the board.
- Two to eight players can play.
- A game is played as a series of 20 rounds. During a round, each player takes one turn. In each turn, a player advances his piece clockwise around the board a number of squares equal to the sum of the number rolled on two six-sided dice.
- After the dice are rolled, the name of the player and the roll are displayed. When the player moves and lands on a square, the name of the player and the name of the square that the player landed on are displayed.
- Each square has a name. Every player begins the game with their piece located on the square named “Go.” The square names will be Go, Square 1, Square 2, ... Square 39
- Run the game as a simulation requiring no user input, other than the number of players.


Played-with
Played-on
2
1
1
1
Die
faceValue
MonopolyGame
Board
1
Plays
1
Contains
2..8
40
Player
name
Owns
1
1
Piece
name
Is-on
0..8
1
Square
name

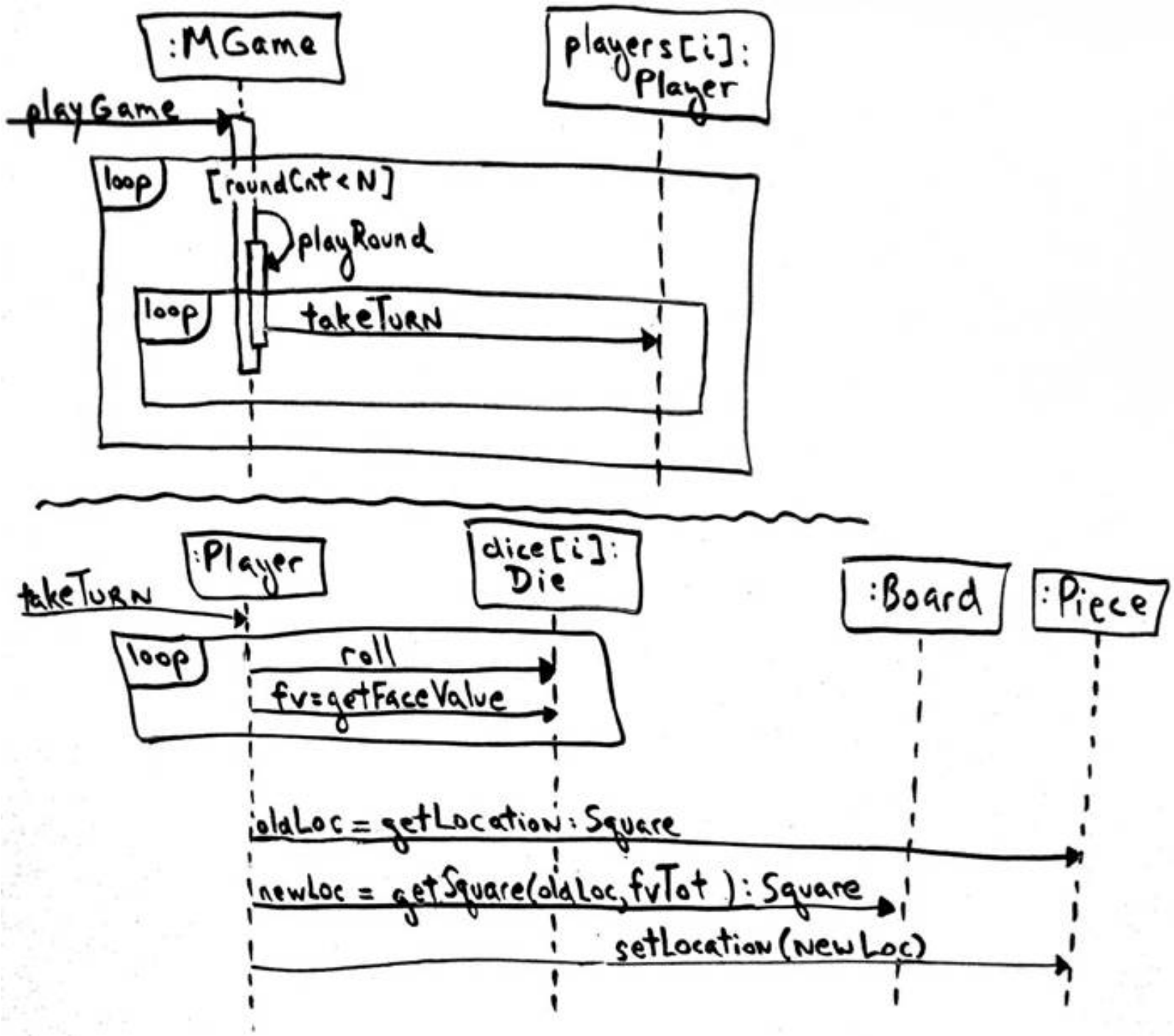

:MGame
playGame
by Controller
UI
Layer
Domain
Layer

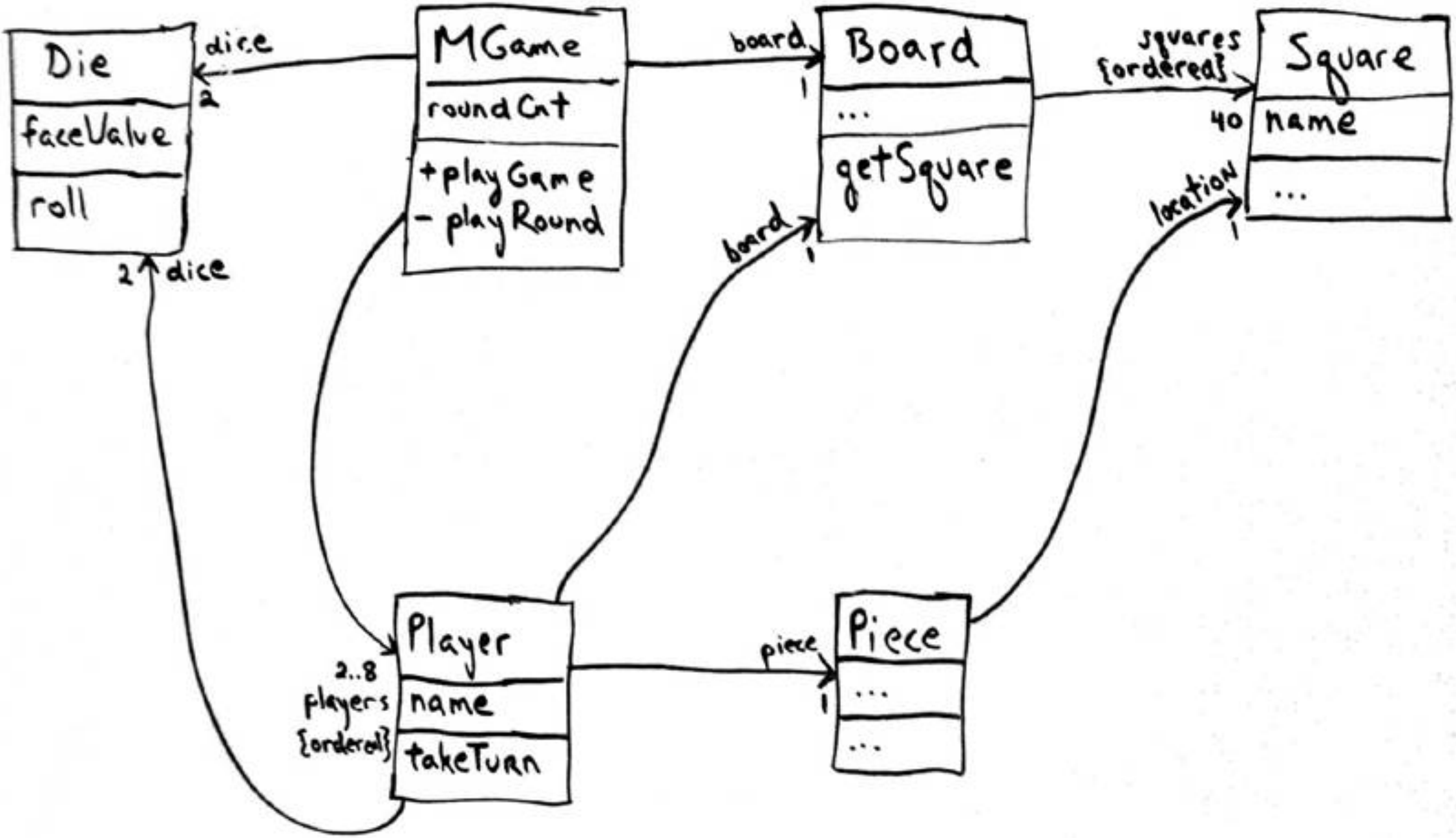


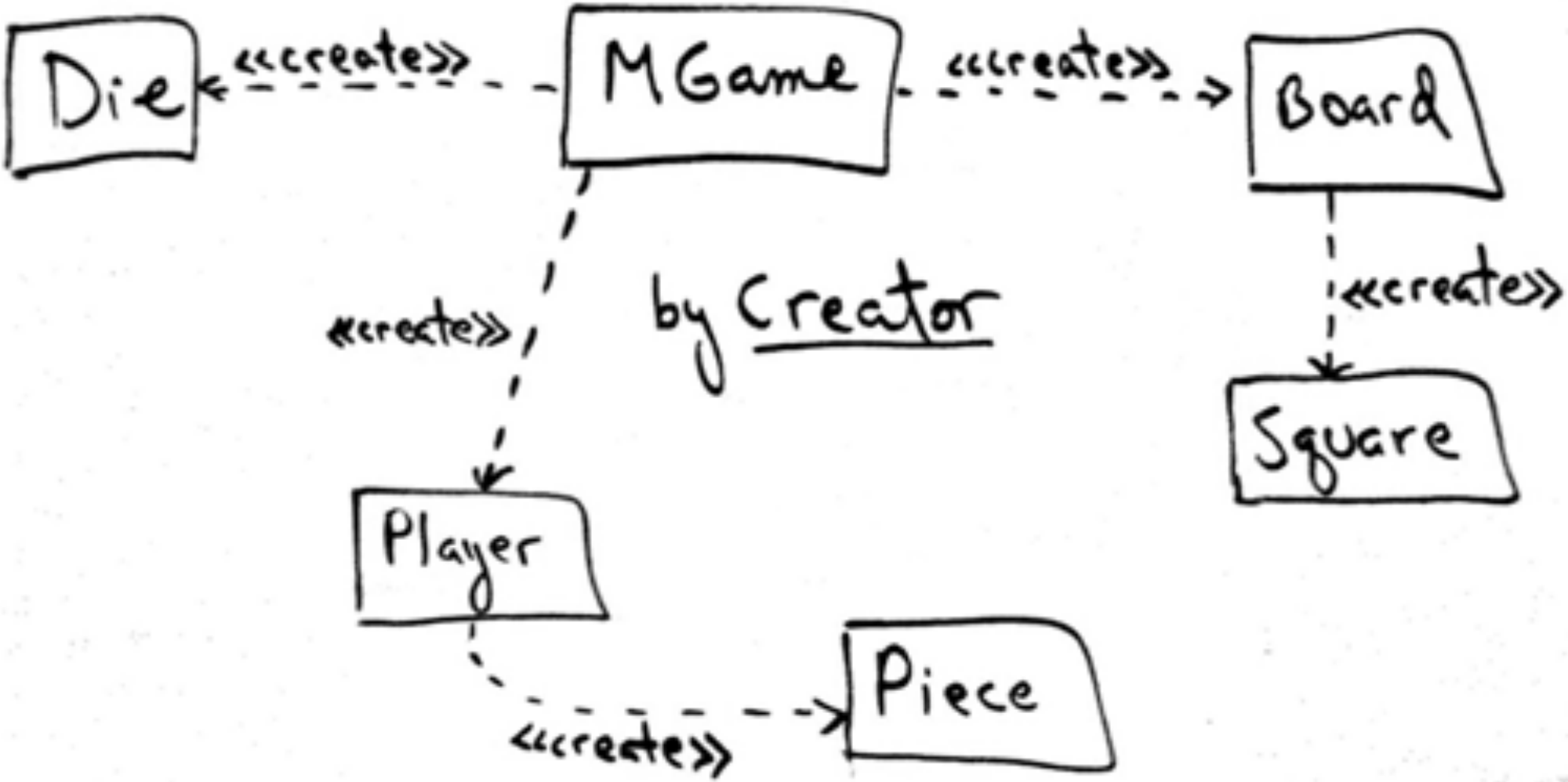
by Expert, the MGame controls the game loop


:MGame
players[i]:
Player
playGame
loop
[roundCnt < N]
playRound
loop
takeTurn


MGame
roundCnt
+playGame
-playRound
Player
2..8
players
{ordered}
name
takeTurn



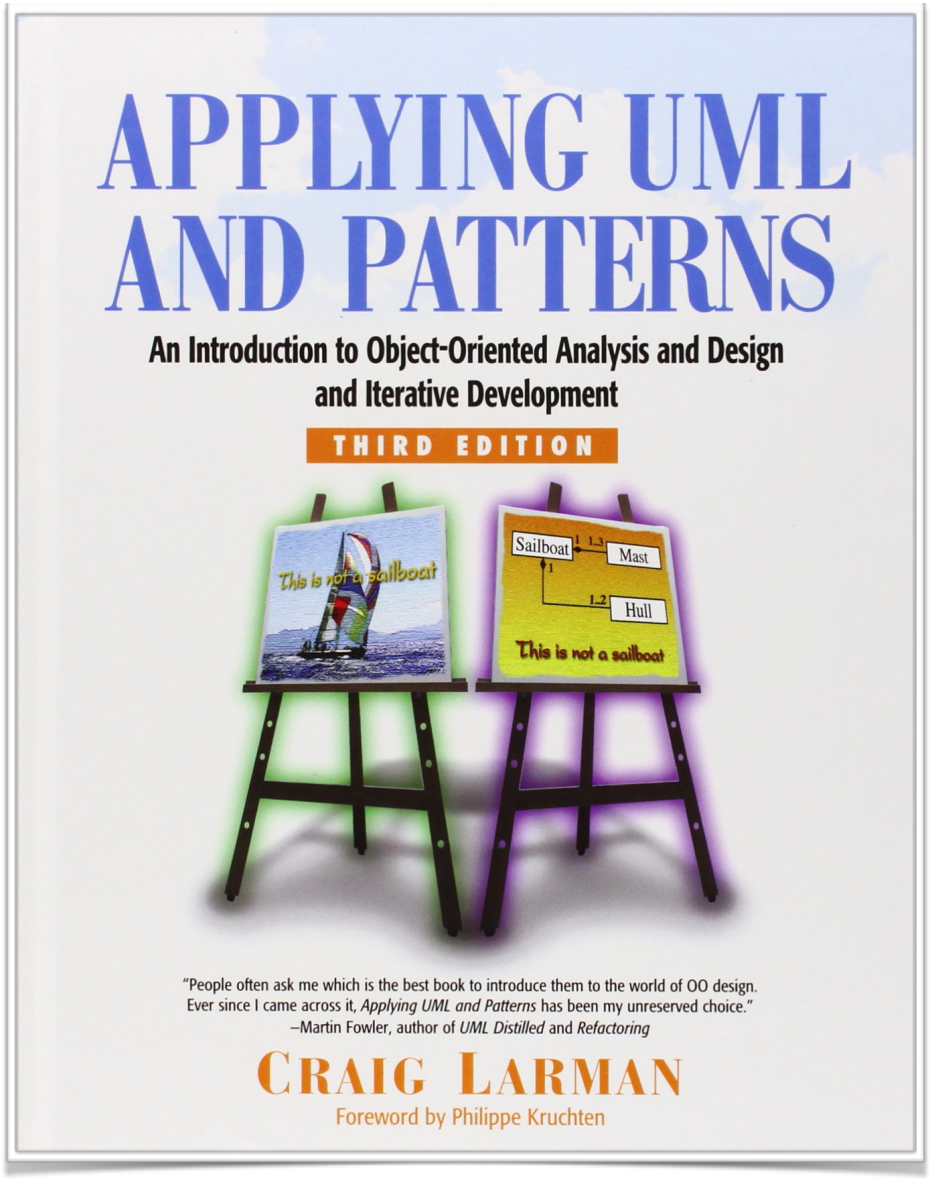

Die
faceValue
roll
dice
2
2
dice
MGame
roundCnt
+playGame
-playRound
board
1
Board
...
getSquare
squares
{ordered}
40
Square
name
...
location
1
board
1
Player
name
takeTurn
2..8
players
{ordered}
piece
1
Piece
...
...


Die
«create»
MGame
«create»
Board
«create»
Square
«create»
by Creator
Player
«create»
Piece

Itération 2


MONOPOLY
The Classic Edition
GO
BOARDWALK
PARK PLACE
CHANCE
PRICE $400
PAY $75.00
PRICE $350
MONOPOLY

- 23.2. Iteration-2 Requirements and Emphasis: Object Design and Patterns
- 24.2. Case Study: Monopoly

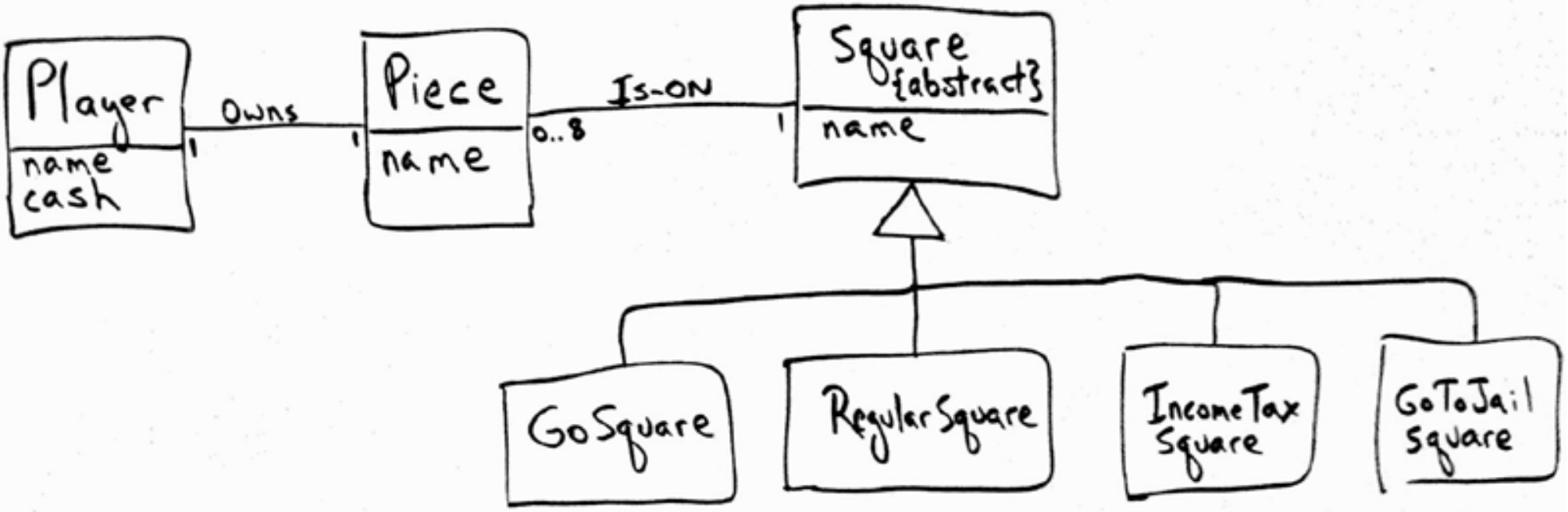


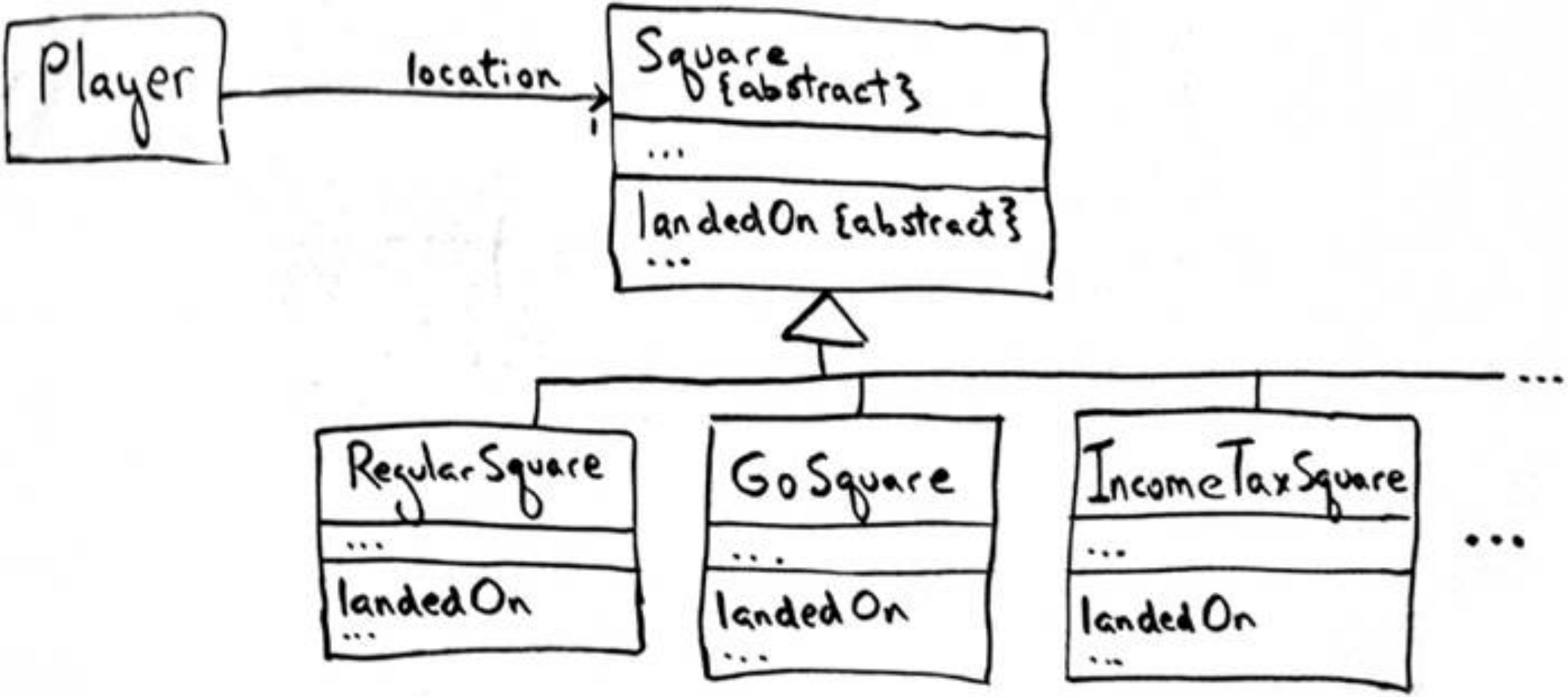
Itération 2

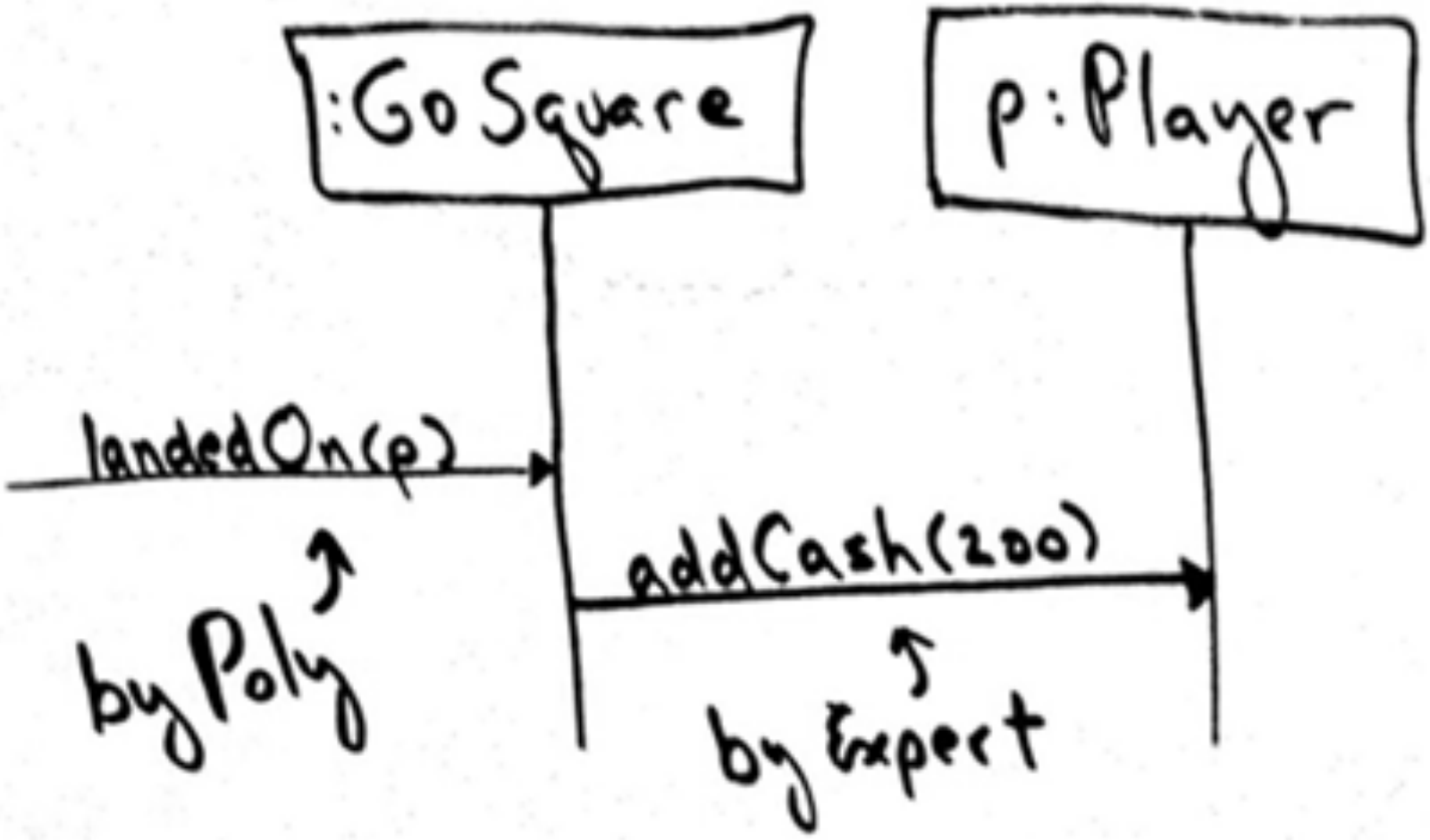
Players moving around the squares of the board. Run the game as a simulation requiring no user input other than the number of players. In iteration-2 some of the special square rules apply. These are described in the following points...

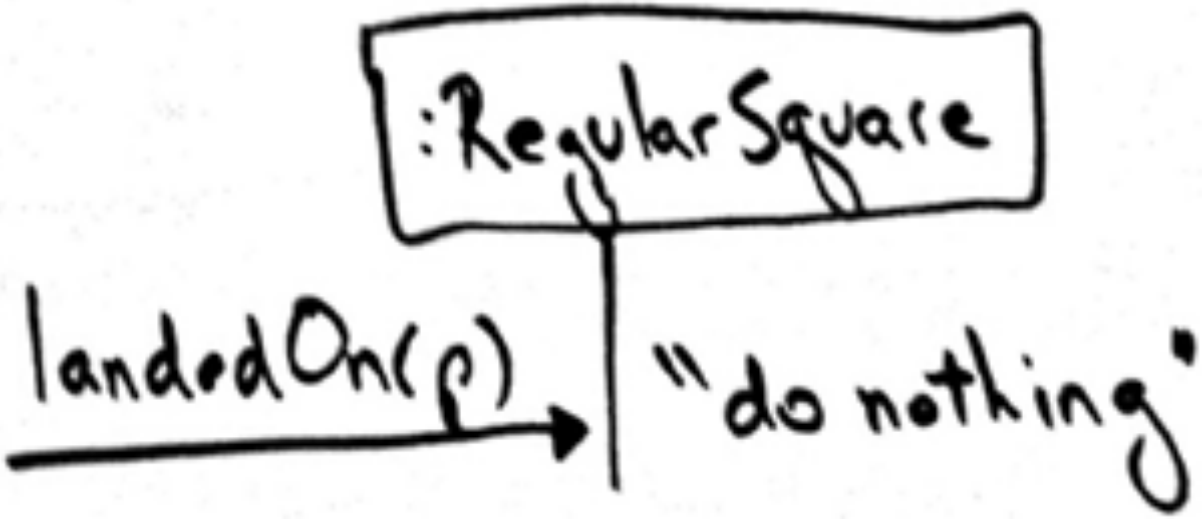
- Each player receives $1500 at the beginning of the game.
- When a player lands on the Go square, the player receives $200.
- When a player lands on the Go-To-Jail square, he moves to the Jail square.
- However, unlike the complete rules, they get out easily. On their next turn, they simply roll and move as indicated by the roll total.
- When a player lands on the Income-Tax square, the player pays the minimum of $200 or 10% of their worth.

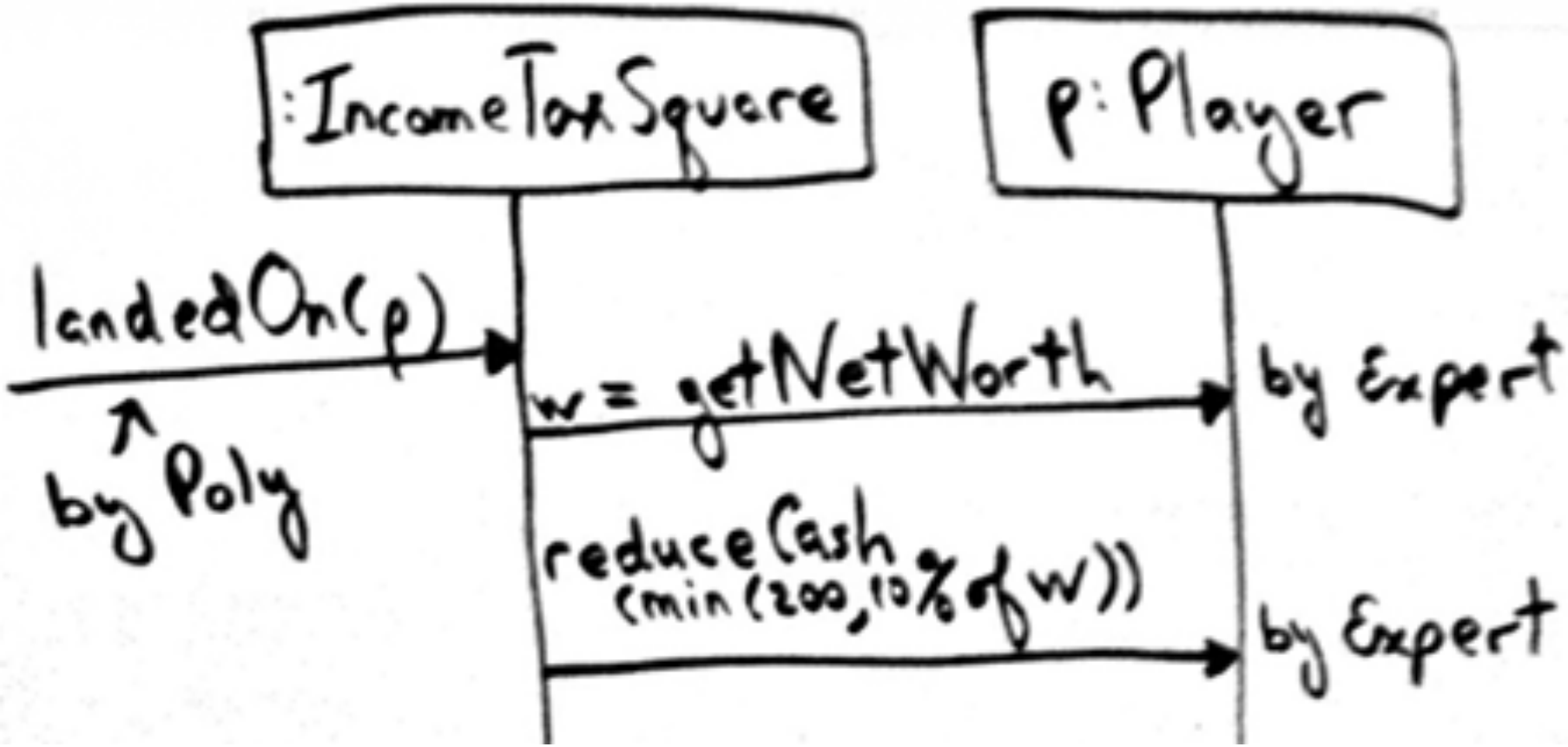
Diagramme de domaine modifié

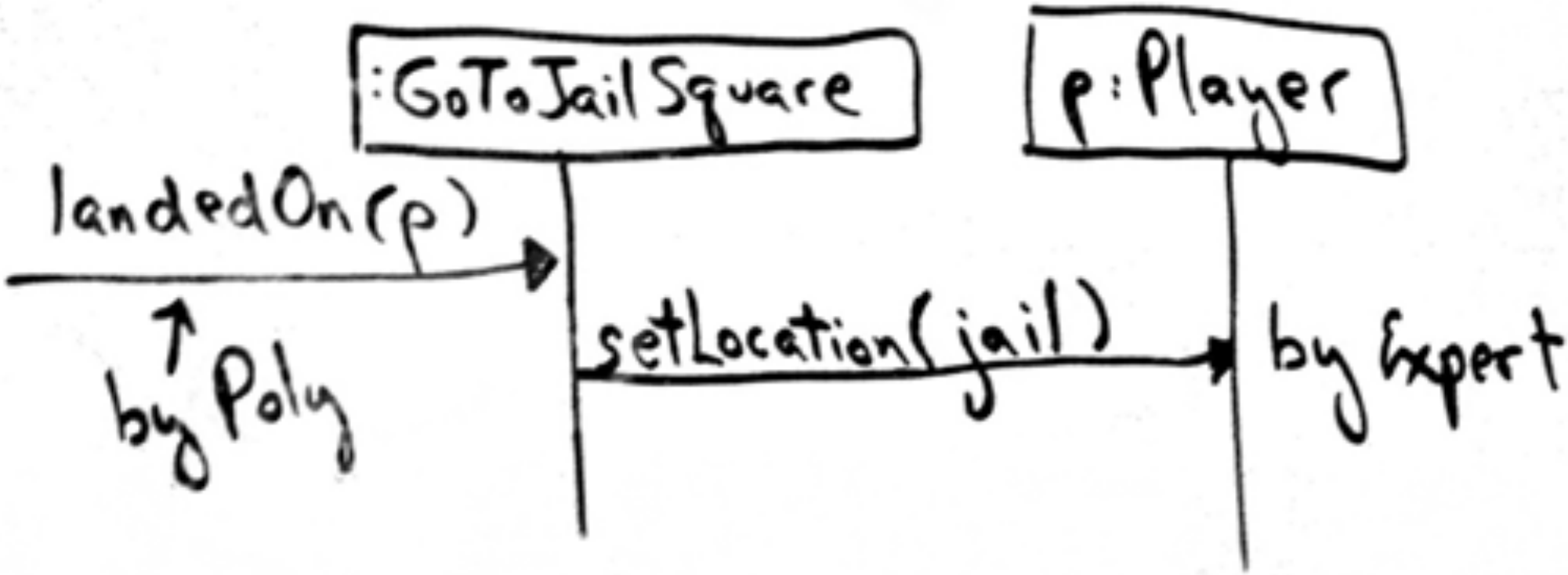


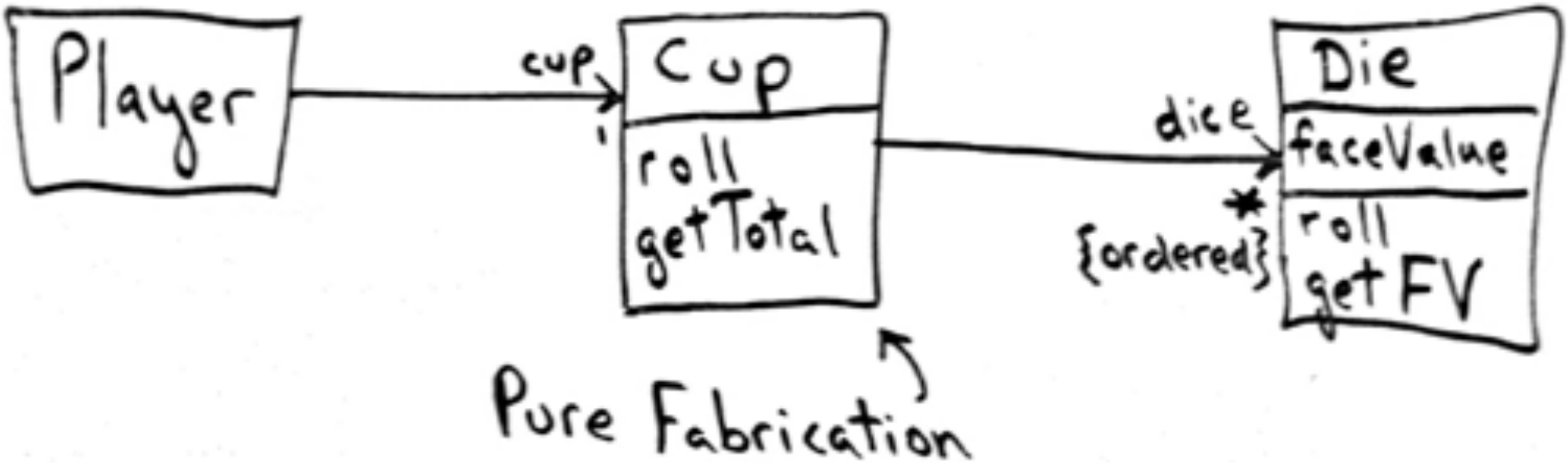

Player
location
1
Square
{abstract}
...
landedOn {abstract}
...
RegularSquare
...
landedOn
...
GoSquare
...
landedOn
...
IncomeTaxSquare
...
landedOn
...
...
...

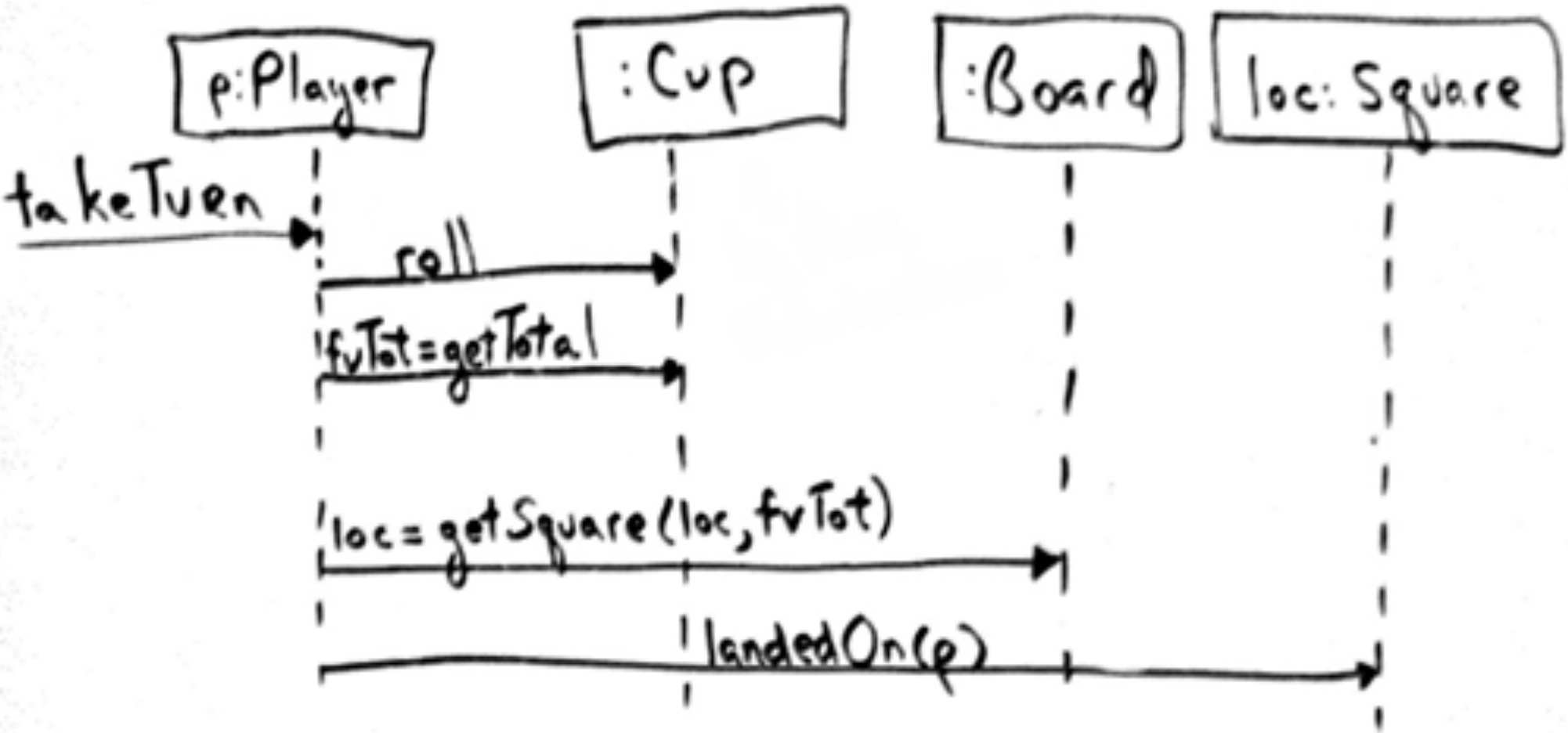

:GoSquare
p:Player
landedOn(p)
addCash(200)
byPoly
byExpert

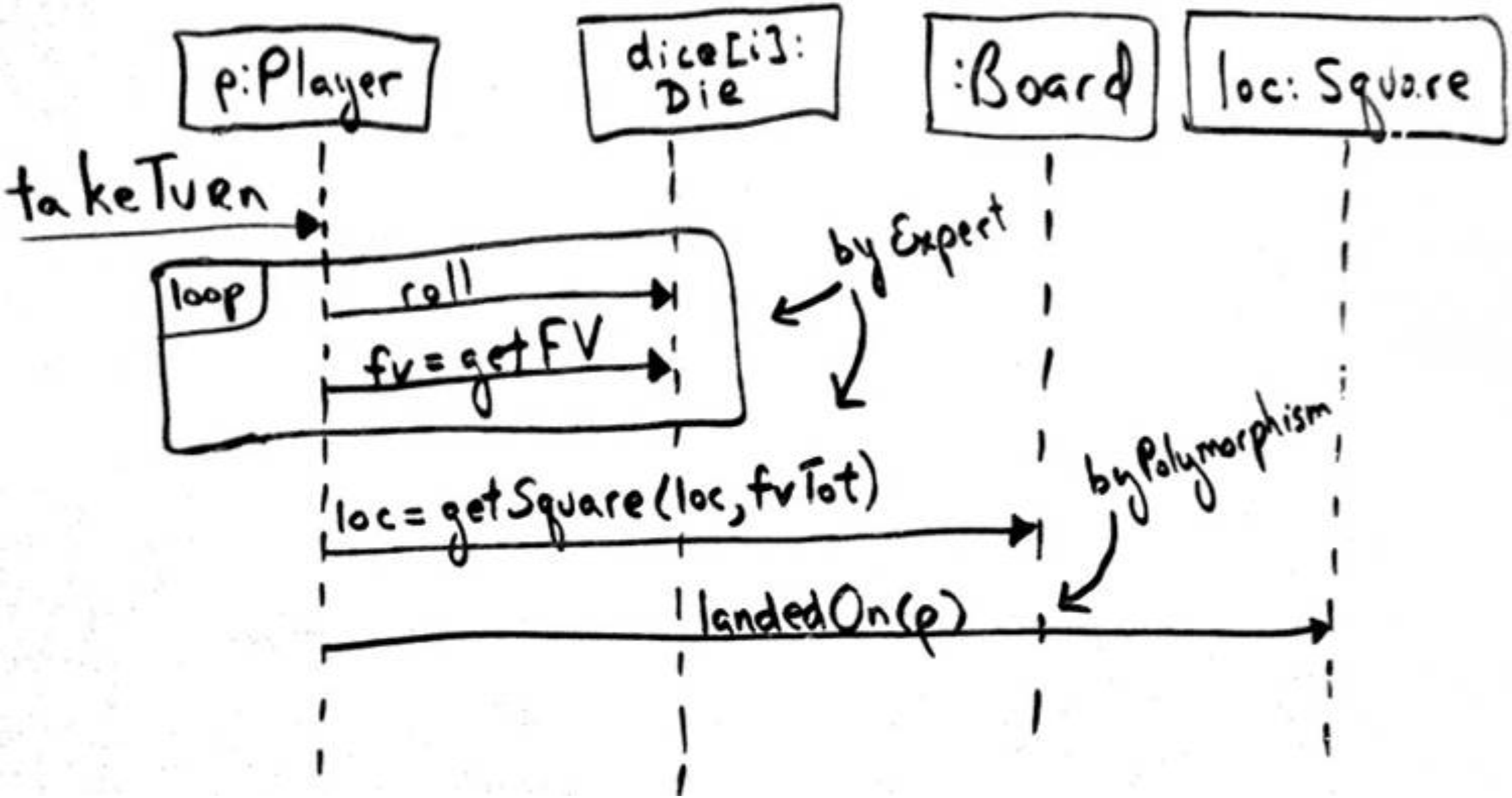

:RegularSquare
landedOn(p)
"do nothing"


:IncomeTaxSquare
p:Player
landedOn(p)
by Poly
w = getNetWorth
by Expert
reduceCash
(min(200, 10% of w))
by Expert


:GoToJailSquare
p:Player
landedOn(p)
by Poly
setLocation(jail)
by Expert


Player
cup
1
Cup
roll
getTotal
dice
*
{ordered}
Die
faceValue
roll
getFV
Pure Fabrication


p:Player
:Cup
:Board
loc:Square
takeTuen
roll
fvTot=getTotal
loc=getSquare(loc,fvTot)
landedOn(p)


p:Player
dice[i]: Die
:Board
loc: Square
takeTurn
loop
roll
fv = getFV
by Expert
loc = getSquare(loc, fvTot)
by Polymorphism
landedOn(p)

- https://junit.org/junit5/docs/current/user-guide/

JUnit 5 Annotations

- @Test
- @BeforeAll / @AfterAll
- @BeforeEach / @AfterEach
- @DisplayName / @Disabled

JUnit 5 annotations

- @ParameterizedTest / @…Source
- @RepeatedTest
- …

JUnit assertions

- assertArrayEquals
- assertEquals
- assertTrue / assertFalse
- assertNull / assertNotNull
- assertSame / assertNotSame
- fail

JUnit 5 assertions

- assertAll
- assertIterableEquals
- assertLinesMatch
- assertNotEquals
- assertThrows
- assertTimeout / assertTimeoutPreemptively

Pour ce laboratoire

- Utiliser au moins une fois @BeforeEach, @ParameterizedTest et @RepeatedTest
- Utiliser au moins 5 assertions différentes

Travis CI

https://travis-ci.org/getting_started

Workflow git

- Utiliser des repositories publics sur GitHub
- Créer une organisation pour votre groupe
- Créer un repository central pour ce labo dans cette organisation
- Forker ce repo dans chacun de vos comptes. Developper dans vos comptes et pousser vers le repo central uniquement via des pull requests

Work flow (suite)

- Créez le projet avec Maven sous IntelliJ. Inclure une classe de test pour chaque classe créée
- Activez Travis CI sur chaque repo forké et sur le repo central.
- Fusionnez régulièrement avec le repo central via des pull requests et attendez que Travis aie validé tous les tests unitaires avant d'accepter la requête.