THE MONOPOLY GAME SYSTEM

• The software version of the game will run as a simulation
• One person will start the game and indicate the number of simulated players
• Thereafter the person will watch while the game runs to completion
• The Monopoly Game System will present a trace of the activity during the simulated player turns
USE CASES

• There is only one significant use case
  • *Play Monopoly Game*

• The game is run as computer simulation watched by one player (more accurately, observer)

• The many possible (simulated) player actions are captured in the domain rules, rather than the extension section
  • Domain rules - requirements or policies that transcend one software project
UCI: PLAY MONOPOLY GAME

Scope: Monopoly application
Level: user goal
Primary Actor: Observer
Stakeholders and Interests:
  - Observer: Wants to easily observe the output of the game simulation

Main Success Scenario:
1. Observer requests new game initialization, enters number of players
2. Observer starts play
3. System displays game trace for next player move (see domain rules, and “game trace” in glossary for trace details
   Repeat step 3 until a winner is found or Observer cancels.

Extensions:
  a. At any time, System fails
     (To support recovery, System logs after each completed move)
     1. Observer restarts System
     2. System detects prior failure, reconstructs state, and prompts to continue
     3. Observer chooses to continue (from last completed player turn)

Special Requirements:
  - Provide both graphical and text trace mode
USE CASE DIAGRAM

Observer

Monopoly

Play Monopoly

Game
FIRST ITERATION PLAN

• Implement a basic, key scenario of the Play Monopoly Game use case: players moving around the squares of the board

• Implement a Start Up use case as necessary to support the initialization needs of the iteration

• Two to eight players can play

• A game is played as a series of 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

• Play the game for only 20 rounds
FIRST ITERATION PLAN ...  

- After the dice is 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 are displayed.

- In this iteration there is no money, no winner or looser, no properties to buy or rent to pay, and no special squares of any kind.

- Each square has a name. Players begin the game with their piece located on the square “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.
CANDIDATE CONCEPTUAL CLASSES

Monopoly Game
Player
Piece
Die
Board
Square
ADDING ASSOCIATIONS
ADDING ATTRIBUTES
MONOPOLY SSD

initialize(numOfPlayers) → System
playGame → Observer
loop [no winner]
  - dice total, player, square
FIRST DESIGN STEPS

- Observer
  - clicks button
  - press to Play
  - a Java Swing Window

- Monopoly
  - action Performed
  - JButton click message
  - play Game
  - system operation

- UI Layer

- Domain Layer

- Controller of this system operation

- Board? Square? Die?
ANOTHER STEP

Poor (Low) Cohesion in the Monopoly Game object

Better

Monopoly Game

playGame

doA

doB

doC
ITERATIVE AND EVOLUTIONARY OBJECT DESIGN

• Keep it light and short
• Move quickly to code and test
• Don’t try to detail everything in the UML models
  • Model the creative difficult parts of the design
• Case study - Monopoly Game
  • More detailed than necessary
FIRST ITERATION

• A simplified version for a scenario of the use case *PlayMonopolyGame*.

• Two system operations:
  
  • *initialize* (or *startUp*)
  
  • *playGame*

• We’ll ignore *initialize* until the last step and focus first on main system operations – *playGame*. 
DOMAIN MODEL FOR MONOPOLY
HOW TO DESIGN **PLAYGAME**

- *playGame* system operation occurs when the human game observer performs some UI gesture to request the game to play as a simulation (e.g. click on “play game” button).
CHOOSING THE CONTROLLER CLASS
THE GAME LOOP ALGORITHM

for N rounds

for each Player p

p takes a turn
WHO CONTROLS GAME LOOP?
CANDIDATES FOR TAKING A TURN

• Player
  • Knows the current location
    • We need to know the starting point of a move

• MGame
  • The two Die objects are part of the Game
    • We need to roll them and calculate their total

• Board
  • Knows all the squares (their organisation)
    • We need to move to the correct square
PLAYER TAKES A TURN
TAKING A TURN

• We need to decide about the following responsibilities
  • Who calculates the dice range?
  • Who calculates the new square location?
  • Who moves the player’s piece from an old location to a new square location?
DYNAMIC DESIGN FOR PLAYGAME
STATIC DESIGN FOR PLAYGAME
DESIGNING INITIALIZE

- Recall two system operations:
  - `initialize` (or `startUp`)
  - `playGame`

- `Initialize` occurs in a `Start Up` use case.

- We choose `MGame` as a suitable root object that will create other objects.

- By Creator `MGame` can justifiably create the `Board` and `Players`, which in turn create `Squares` and `Pieces`. 
CREATION DEPENDENCIES
SECOND ITERATION

- Basic key scenario of the *PlayMonopolyGame* use case
  - Run the game as simulation. Players moving around the board
- Each player receives $1500 at the beginning of the game
- When the player lands on the “Go” square, the player receives $200
- When the player lands on the “Go-To-Jail” square, they move to “Jail” square (next turn they get out)
- On the “Income-Tax” square, the player pays $200 or 10% of their worth
MODIFIED DOMAIN MODEL
DESIGNING FOR DIFFERENT SQUARE ACTIONS
APPLYING POLYMORPHISM
GOSQUARE CASE

```
GoSquare

landedOn(p) by Poly

Player

addCash(200) by Expert
```
INCOMETAXSQUARE CASE

\[
\text{IncometaxSquare}
\]

\[
\text{Player}
\]

\[
\text{landedOn}(p) \quad \text{by Poly}
\]

\[
w = \text{getNetWorth}
\]

\[
\text{reduceCash(min(200, 10\% of w)))} \quad \text{by Expert}
\]

32
GOTOJAILSQUARE CASE
DICE CUP
USING THE CUP
THIRD ITERATION

• Basic key scenario of the *PlayMonopolyGame* use case
  - Run the game as simulation. Players moving around the board
• There are “Lots”, “Railroads” and “Utility” squares
  - Players who land on these squares may buy them
  - If the square is owned by the player that landed on it, nothing happens
  - If the square is owned by a player other than the one that landed on it, the player that landed on the square must pay its owner rent
LANDING ON PROPERTY SQUARE

```
s: Property Square
```

```
\text{landed On}(p) \quad \text{attemptPurchase}(s) \quad \text{payRent}(p) \\
\text{not owned} \quad \text{by Expert} \quad \text{"almost" by Poly and by Expert}
```

```
\text{owner} \neq p
```

```
p: Player
```
PURCHASING PROPERTY
PAYING RENT

Diagram:

- PropertySquare
  - payRent(p)
  - by Poly
    - r = getRent
    - addCash(r)
    - reduceCash(r)

- Player
  - owner:
  - Player

- by Expert
POLYMORPHIC GETRENT

```
:LotSquare
r = getRent
  by Poly
  return index

:RailRoadSquare
r = getRent
  by Poly
  c = getRR Count
  return c * 25

:UtilitySquare
r = getRent
  tot = getTotal
  return tot * 4

owner:
Player

:Cup
```
PARTIAL DCD