Learning from a Sokoban implementation

S. Ducasse, L. Fabresse, G. Polito, and P. Tesone
Goals

- Think about model
- Think about messages and conditions
Studying a Sokoban Implementation

Sokoban is a puzzle video game genre in which the player pushes crates or boxes around in a warehouse, trying to get them to storage locations.

https://en.wikipedia.org/wiki/Sokoban
Studying a Sokoban Implementation

- Developed by some students of I. Franko University (Lviv)
- Thank you so much!
Looking the implemented core model

- Block
  - EmptyBlock
  - Wall
- GameModel
- GameState
- Maze
- MazeTemplate
- MoveResult
  - Move
  - Push
  - NoMove
Let us "Speculate about Design"

- Apply **Speculate about Design** object-oriented reengineering pattern

- **Intent:** Progressively refine a design against source code by checking hypotheses about the design against the source code

- Use your development expertise to conceive a hypothetical class diagram representing the design
Take some minutes to sketch a list of classes

- ...
- ...
- ...
- ...
A possible model

- Wall
- Floor
- Box
- Robot
- Target
- Board

And

- Template/Level
- Moves
Let us go back to our case

- Block
  - EmptyBlock
  - Wall
- GameModel
- GameState
- Maze
- MazeTemplate
- MoveResult
  - Move
    - Push
  - NoMove
Gut feeling analysis

- The implemented Block model looks too 'shallow'
- Remember classes
  - are representing cases
  - are the basis for dispatch
- Not enough classes leads to **tricky conditionals** and **monolithic** systems
- Remember the lectures **Implementing not, or...**
Let us check the class API

Classes define:

- isEmptyBlock
- isWall
- hasPlayer
- hasTarget
- hasBox

Let us check the way this API is used
GameView >> drawBlock: aBlock on: aCanvas
  aBlock isWall
    ifTrue: [ self drawWall: aCanvas ]
    ifFalse: [ aBlock isEmptyBlock
      ifTrue: [ aBlock hasPlayer
        ifTrue: [ aBlock hasTarget
          ifTrue: [ self drawTargetAndPlayer: aCanvas ]
          ifFalse: [ self drawPlayer: aCanvas ]]
        ifFalse: [ aBlock hasBox
          ifTrue: [ aBlock hasTarget
            ifTrue: [ self drawTargetAndBox: aCanvas ]
            ifFalse: [ self drawBox: aCanvas ]]
          ifFalse: [ aBlock hasTarget
            ifTrue: [ self drawTarget: aCanvas ]
            ifFalse: [ self drawEmptyBlock: aCanvas ]]]]
Analysis

The model only defines EmptyBlock and Wall

- No Player, no Target, no Box.
- Too much logic is put in EmptyBlock
- Too many questions, not enough Tell (Do not Ask, Tell)
With a better model

- Tile
  - Box
  - BoxOnTarget
  - EmptyBlock
  - Player
  - Wall

- We can send **messages** to the 'correct' object
- We can tell and not ask!
A first nicer solution

GameView >> drawBlock: aBlock on: aCanvas
    aBlock isWall ifTrue: [ self drawWall: aCanvas ].
    aBlock isEmptyBlock ifTrue: [
        aBlock hasPlayer ifTrue: [ ...

Becomes

GameView >> drawBlock: aBlock on: aCanvas
    aBlock drawOn: aCanvas

Wall >> drawOn: aCanvas
    "Cairo code"

EmptyBlock >> drawOn: aCanvas
    "Cairo code"
A solution supporting multiple canvases

To supporting multiple rendering back-ends (morphic, Cairo...), drawing should not be in the Block classes
A solution supporting multiple canvases

GameView >> drawBlock: aBlock on: aCanvas
    aBlock isWall ifTrue: [ self drawWall: aCanvas ].
    aBlock isEmptyBlock ifTrue: [
        aBlock hasPlayer ifTrue: [
    ...

Becomes

GameView >> drawBlock: aBlock on: aCanvas
    aBlock drawOn: aCanvas for: aView

Wall >> drawOn: aCanvas for: aView
    aView drawWall: aCanvas

EmptyBlock >> drawOn: aCanvas for: aView
    aView drawEmptyBlock: aCanvas
Double dispatch

Each block tells the view how to draw it.

- **GameView** >> `drawBlock: aBlock on: aCanvas`
  - `aBlock drawOn: aCanvas view: self`

- **Wall** >> `drawOn: aCanvas view: aView`
  - `aView drawWall: aCanvas`

- **EmptyBlock** >> `drawOn: aCanvas view: aView`
  - `aView drawEmptyBlock: aCanvas`

- It is double dispatch with more objects
- Sending messages is powerful
- Modular
Intermezzo: Testing methods

Wall >> isWall
  ^ true

EmptyBlock >> isWall
  ^ false

- What do you think about it?
Disgueded kind testing method

Wall >> isWall
  ^ true

EmptyBlock >> isWall
  ^ false

and

GameView >> drawBlock: aBlock on: aCanvas
  aBlock isWall ifTrue: [ self drawWall: aCanvas ]

is nearly the same as

GameView >> drawBlock: aBlock on: aCanvas
  (aBlock isKindOf: Wall) ifTrue: [ self drawWall: aCanvas ].
Back to the model

What are:

- MoveResult
  - Move
  - Push
  - NoMove

- Reification of player actions
- Good to record and replay
Let us study the API

MoveResult >> isMove
  \(^\) false

MoveResult >> isPush
  \(^\) false

MoveResult >> isNoMove
...

- Again testing kind methods
- Testing kind methods are the same as \( x \text{ class } = \text{MoveResult} \)
Checking testing method use

```smalltalk
GameState >> moveBy: aDirection
| move |
move := maze moveBy: aDirection.
move isMove ifTrue: [ moves := moves + 1 ].
move isPush ifTrue: [
  pushes := pushes + 1.
  moves := moves + 1 ].
self addMove: move
```

What is the problem?

... move isMove ifTrue: [ moves := moves + 1 ].
move isPush ifTrue: [
  pushes := pushes + 1.
  moves := moves + 1 ].
...

• How can we do it better?
GameState >> moveBy: aDirection
| move |
move := maze moveBy: aDirection.
move updateGameState: self.
self addMove: move

Move >> updateGameState: aGameState
aGameState incrementMoves

Push >> updateGameState: aGameState
super updateGameState: aGameState.
aGameState increasePushes

NoMove >> updateGameState: aGameState
self
Conclusion

- **Challenge** classes
- **Little** class hierarchies are **good**
- Better **many small classes than a big one**
- **Challenge** kind testing methods
- **Check** their use
- Messages act as **dispatcher**
Advanced Object-Oriented Design and Development with Pharo

A course by
S. Ducasse, L. Fabresse, G. Polito, and P. Tesone