Homework 6 Design

Qiang Fang

Class Design

Difference compared to homework5 design are listed as follows:

I designed an interface Controller and implemented with 2 classes, SwingController and ConsoleController. SwingController is used for the game with GUI, ConsoleController is used for the texted based game.

SwingController controls the workflow, interacts with model and view, receives events coming from view, calls model to process, then responds, tells the view to refresh.

For GUI based game, I designed an interface View and a class SwingView to implement the interface.

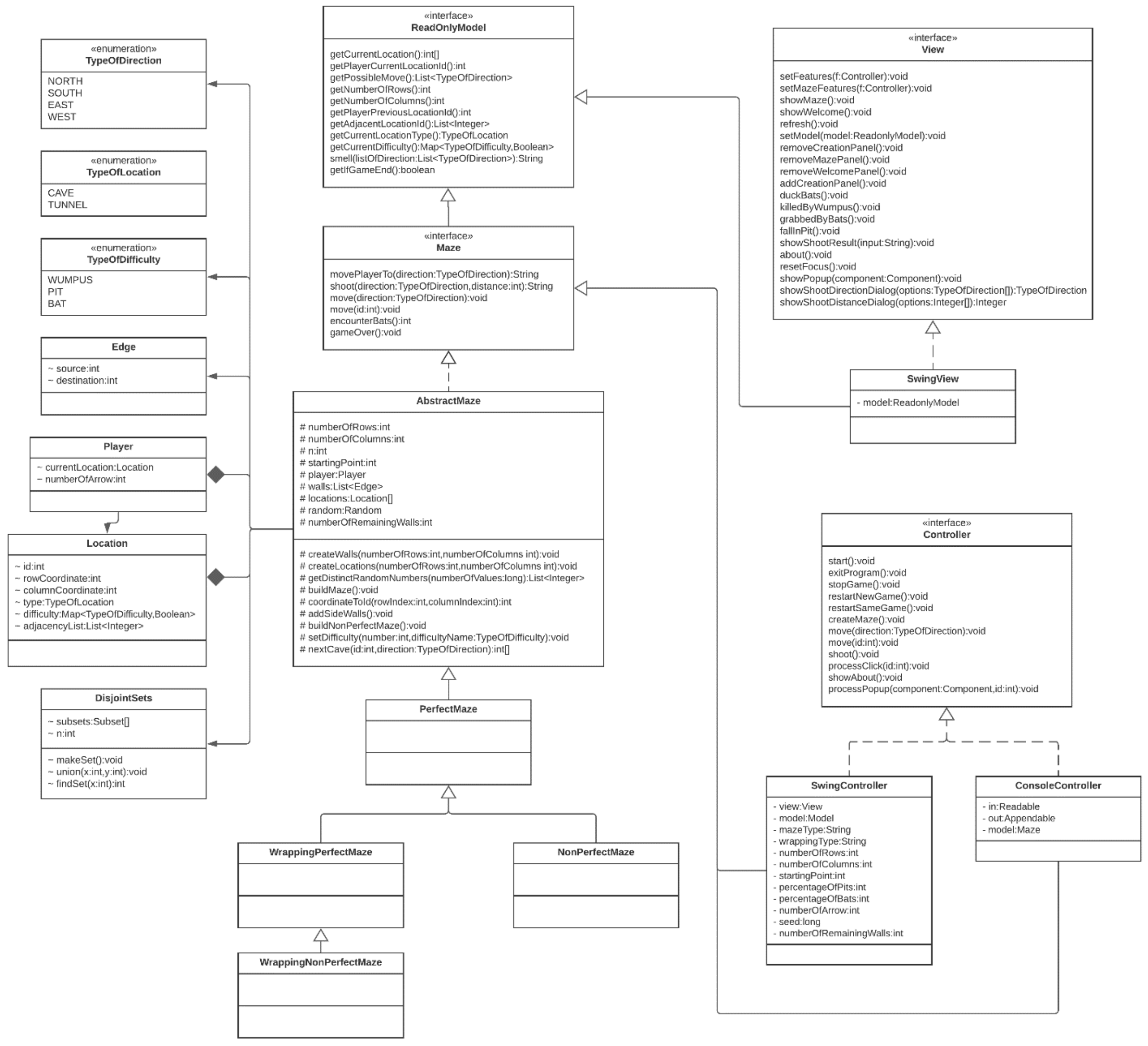
I used SpringUtilities class from Java website to handle SpringLayout.

For the existing model part, I split the original interface into two interfaces, Maze and ReadonlyModel. The ReadonlyModel only contains read only methods, Maze interface contains other methods that can change the model.

The ReadonlyModel is used for View class. The View needs access to the application state, so it can display it. In some cases, it makes it easier to give the View a read-only Model so that when the View needs to access the application state, it can get it directly from the Model.

I modified the model to support multi-player mode. Change player field to a list. And change corresponding methods and constructors. Add arguments to distinguish which player the methods need to handle.

UML:


«enumeration» TypeOfDirection
NORTH
SOUTH
EAST
WEST
«enumeration» TypeOfLocation
CAVE
TUNNEL
«enumeration» TypeOfDifficulty
WUMPUS
PIT
BAT
Edge
~ source:int
~ destination:int
Player
~ currentLocation:Location
~ numberOfArrow:int
Location
~ id:int
~ rowCoordinate:int
~ columnCoordinate:int
~ type:TypeOfLocation
~ difficulty:Map<TypeOfDifficulty,Boolean>
~ adjacencyList:List<Integer>
DisjointSets
~ subsets:Subset[]
~ n:int
~ makeSet():void
~ union(x:int,y:int):void
~ findSet(x:int):int
«interface» ReadOnlyModel
getCurrentLocation():int[]
getPlayerCurrentLocationId():int
getPossibleMove():List<TypeOfDirection>
getNumberOfRows():int
getNumberOfColumns():int
getPlayerPreviousLocationId():int
getAdjacentLocationId():List<Integer>
getCurrentLocationType():TypeOfLocation
getCurrentDifficulty():Map<TypeOfDifficulty,Boolean>
smell(listOfDirection:List<TypeOfDirection>):String
getIfGameEnd():boolean
«interface» Maze
movePlayerTo(direction:TypeOfDirection):String
shoot(direction:TypeOfDirection,distance:int):String
move(direction:TypeOfDirection):void
move(id:int):void
encounterBats():int
gameOver():void
AbstractMaze
# numberOfRows:int
# numberOfColumns:int
# n:int
# startingPoint:int
# player:Player
# walls:List<Edge>
# locations:Location[]
# random:Random
# numberOfRemainingWalls:int
# createWalls(numberOfRows:int,numberOfColumns int):void
# createLocations(numberOfRows:int,numberOfColumns int):void
# getDistinctRandomNumbers(numberOfValues:long):List<Integer>
# buildMaze():void
# coordinateToId(rowIndex:int,columnIndex:int):int
# addSideWalls():void
# buildNonPerfectMaze():void
# setDifficulty(number:int,difficultyName:TypeOfDifficulty):void
# nextCave(id:int,direction:TypeOfDirection):int[]
PerfectMaze
WrappingPerfectMaze
NonPerfectMaze
WrappingNonPerfectMaze
«interface» View
setFeatures(f:Controller):void
setMazeFeatures(f:Controller):void
showMaze():void
showWelcome():void
refresh():void
setModel(model:ReadonlyModel):void
removeCreationPanel():void
removeMazePanel():void
removeWelcomePanel():void
addCreationPanel():void
duckBats():void
killedByWumpus():void
grabbedByBats():void
fallInPit():void
showShootResult(input:String):void
about():void
resetFocus():void
showPopup(component:Component):void
showShootDirectionDialog(options:TypeOfDirection[]):TypeOfDirection
showShootDistanceDialog(options:Integer[]):Integer
SwingView
- model:ReadonlyModel
«interface» Controller
start():void
exitProgram():void
stopGame():void
restartNewGame():void
restartSameGame():void
createMaze():void
move(direction:TypeOfDirection):void
move(id:int):void
shoot():void
processClick(id:int):void
showAbout():void
processPopup(component:Component,id:int):void
SwingController
- view:View
- model:Model
- mazeType:String
- wrappingType:String
- numberOfRows:int
- numberOfColumns:int
- startingPoint:int
- percentageOfPits:int
- percentageOfBats:int
- numberOfArrow:int
- seed:long
- numberOfRemainingWalls:int
ConsoleController
- in:Readable
- out:Appendable
- model:Maze

Test design

Test controller, view and the modified parts of the model, each concrete class’s public methods.

Test the whole game manually, do every possible click and move, do every possible operation, test whether the game works correctly.

Test the controller in isolation by creating a mock model and mock view. The mock model covers all the branched in controller and return the input passed to it instead of executing the real model logic. The mock view just returns a log indicating the corresponding methods are called properly. Test player1 and player2 correctly take turns to play the game.

Test the model separately. Test cases include:

Player1 and player2 are died and game over, no one wins.

Player1 and player2 run out of arrows and game over, no one wins.

Player1 slay the Wumpus, player2 is alive, player1 wins and game ends.

Player2 slay the Wumpus, player2 is alive, player2 wins and game ends.

Player1 died, player2 kills the Wumpus, player2 wins and game ends.

Player2 died, player1 kills the Wumpus, player1 wins and game ends.

Challenges

How to choose proper components is chanllenging, implementing two-player mode is chanlenging.