Report
Assignment 1

by

Group 33

Teaching Assistant
B. Reijm

Group members
Wytze Elhorst
Thomas Kolenbrander
Steven Meijer
Bart van Oort
Steven Meijer

Contents

Exercise 1 - The Core

Exercise 1.1

First is the Game class, this class needs to be able to set up the game for the player to be able to play it. To do this the game collaborates with Board to create a board and Player to collect input from.

Game	
Superclass: none	
Subclass: none	
Responsibilities: Creates an instance of Board with cells on start-up Allow the player to make moves on the board Able to stop the game when done	Collaborators: Board Player

The second class is Board. This class creates a board with cells and is responsible for keeping the board updated. It collaborates with Cells and Gems, since the board needs Gems to recognise chains and the board is made up of Cells.

Board	
Superclass: none	
Subclass: none	
Responsibilities: Fill the cells on board with gems upon creation Recognise chains and remove them Refill empty cells Make the cells fall down properly	Collaborators: Gem Cell Direction Position

The class Cell only needs to be able to contain a Gem and to be able to remove that gem. Therefore Cell only collaborates with Gem.

Cell	
Superclass: none	
Subclass: none	
Responsibilities: Contain a gem Have the gem be removed	Collaborators: Gem

Gem only needs to have gems of different colours.

Gem	
Superclass: none	
Subclass: none	
Responsibilities: Have a certain colour	Collaborators:

The player class needs to be able to interact with the board and move things on the board. It also needs to keep a score for all the chains created. It collaborated with game to perform moves on the board and uses board to get information about chains to calculate the score with.

Player	
Superclass: none	
Subclass: none	
Responsibilities: Move cells on the board Gain and keep scores through creating chains	Collaborators: Game Board

There are a few differences with this design and our actual design. First and foremost we used the classes Direction and Position to facilitate changes on the board and recognition of chains. These two classes could have been included in the board class, but we think these separate classes make the code easier to understand.

Another thing we have is the enumeration GemType that creates a set of constants for the Gems. This makes the gems able to have a certain amount of types and it allows new types to be added easily.

A last small thing is that the Board class also calculates the score and the Player class only adds this score, this means the Player class doesn't have to keep asking the board for chains and just gets a score from it instead.
Another thing to note is that we have not used inheritance in either the design above or our actual design. It could possibly have been used for some different gemtypes, but it seemed to us that our method allows for easier additions.

Exercise 1.2

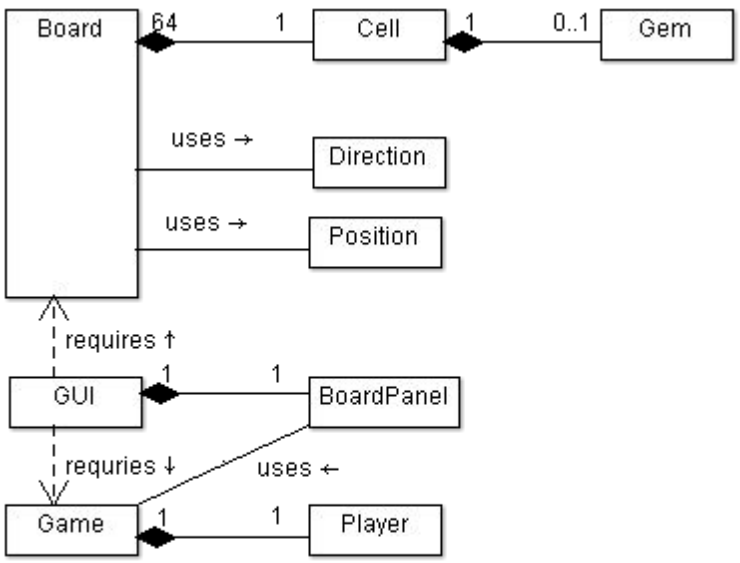
Our main classes (Game and Board) are very similar to the classes described above in terms of responsibilities and collaborations.
The Game creates a new instance of Board on start-up and allow the player to interact with it and thus collaborates with the board and player classes. Responsibilities that are added though are making a move and handling the consequences of the move. This means Game has to call the right methods in Board to make sure the board is in the right state.
The Board is responsible for having all the knowledge of the board and how to influence it and recognise chains, it is however not responsible for making sure its methods are called at the right time and relies on Game for that. It collaborates with Cell and Gem since the board is made up of cells and needs cells to recognise chains. It also collaborates with Direction and Position to get more information about the cells.

Exercise 1.3

We consider the other classes less important for the following reasons:
Direction and position could both be added to the board class, but are separate to provide more clarity.
Cell and Gem could possibly be merged together to form another main class, with Gemtype either staying as an enum or being replaced by subclasses. The reason this is not done is because it is easier to denote an empty cell this way. Also, it is possible to add extra features to gems in the future this way.
Finally, the only function of the Player class is to keep the score. This could easily be done by the Game class for now, but we keep this class in to keep a good structure for future development, for example, when multiple players, each with their respective own scores are introduced.

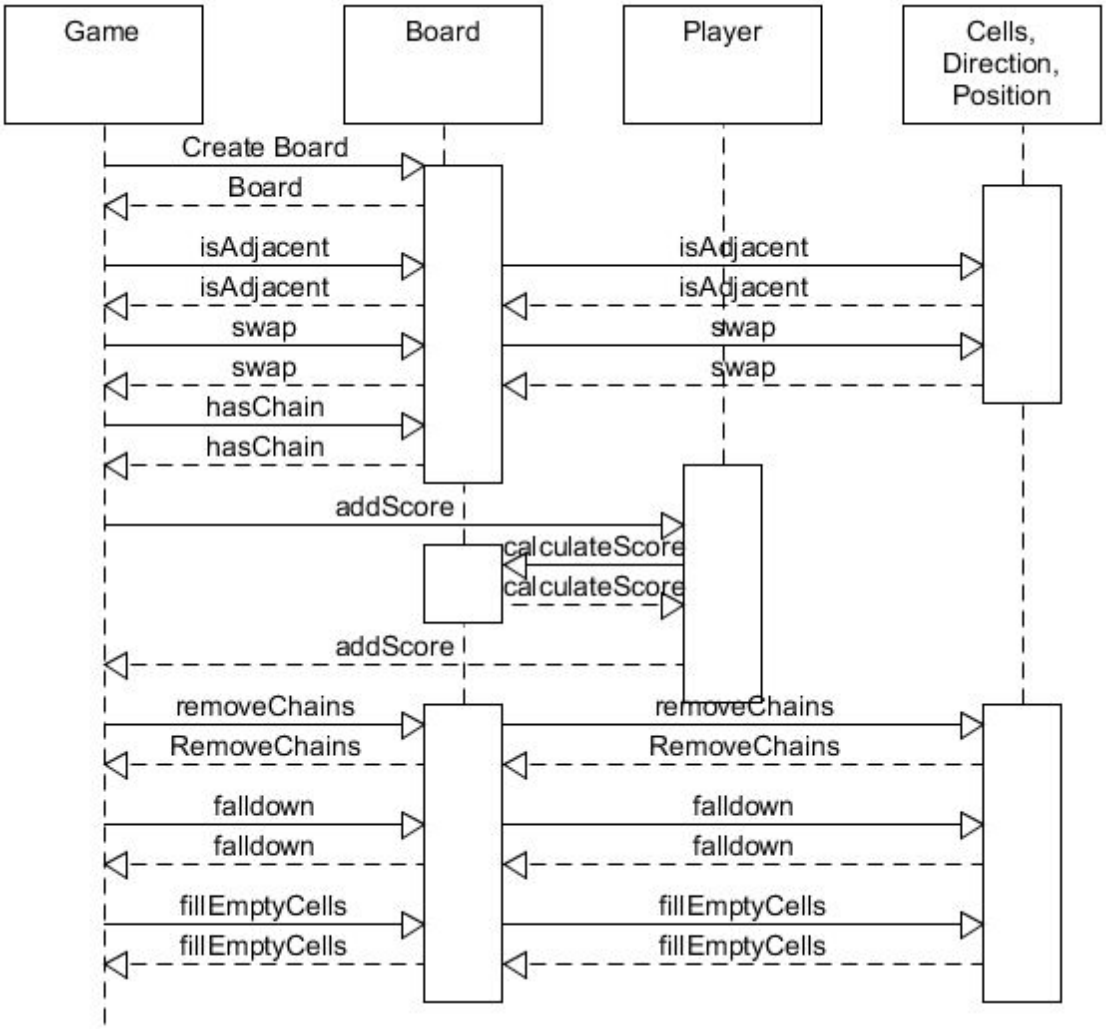
Exercise 1.4

On the right is the class diagram for our main classes and the classes they collaborate with.



Exercise 1.5

The sequence Diagram:



Exercise 2 - UML in practice

Exercise 2.1

Aggregation and composition are very much alike. They both imply that a class is part of another class. However, there is a difference between the two. Composition is a very strict constraint because one part only belongs to one whole, while aggregation implies that parts may be shared between classes. With composition the child can only exist as a part of the parent and not on its own. With aggregation the child can exist without the parent existing.

In our project we only use composition, because most of our classes are only used by one class. They are useless for other classes. In total we have five composition relations in our project:

1. Board - Cell, because cells are only used in the board class.
2. Cell - Gem, because you can only reach a gem by its cell, it is contained by the cell class.
3. GUI - ScorePanel, ScorePanel is just a part of the GUI, nothing difficult.
4. GUI - BoardPanel, the same as number 3, BoardPanel is part of the GUI
5. Game - Player, the player class is mostly used to keep score of the player. This is done through the game class, thus the relationship should be a composition.

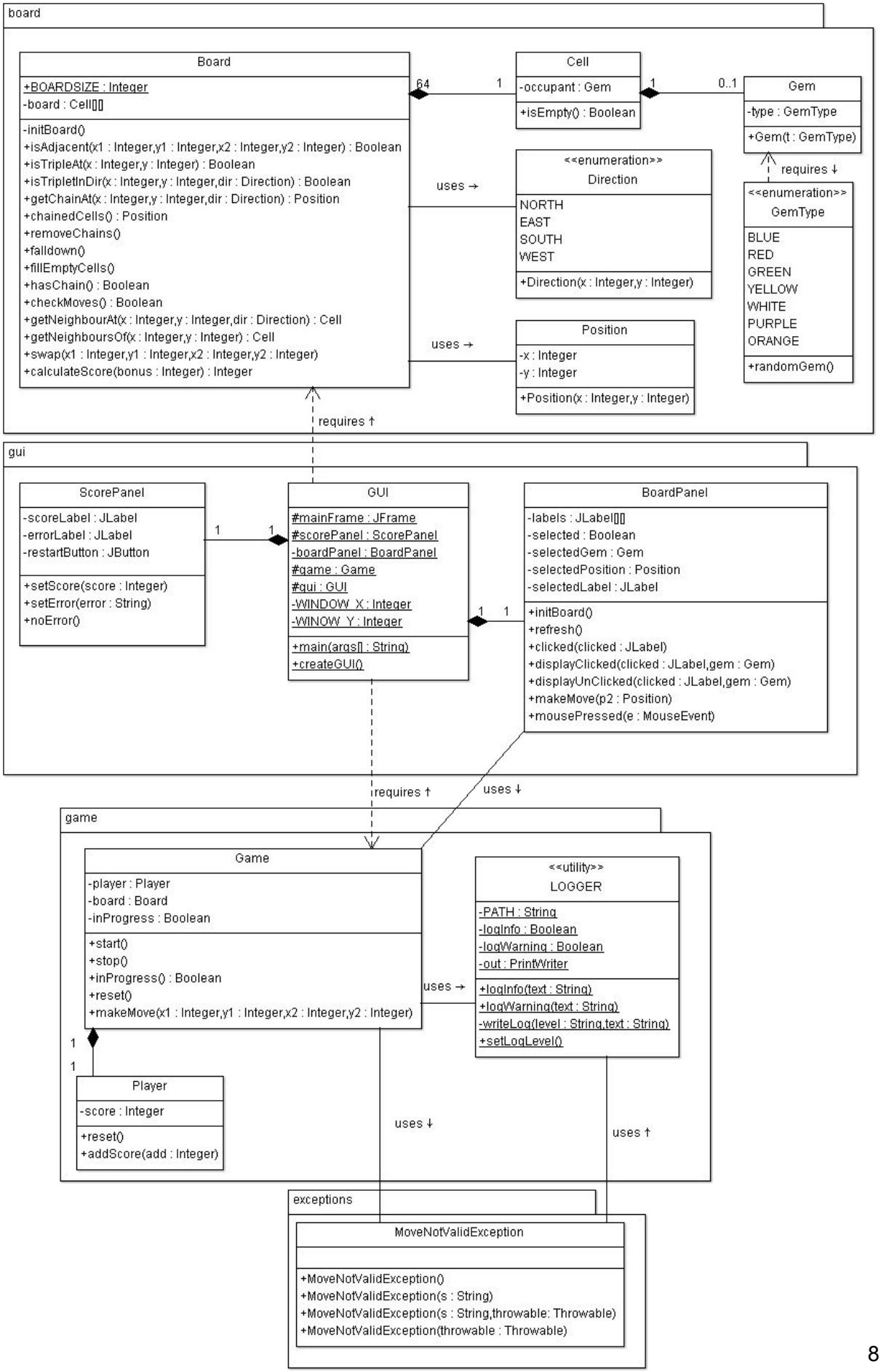
Exercise 2.2

We have not used any parametrized classes in our code. Parameterized classes are used when an object can be used with different types of objects. The parameter shows which type of object is used.

Exercise 2.3

The class diagram of our code can be found on the next page. A more detailed version of the diagram can be found on our GitHub.

We do not use any inheritance in our code so we do not have the standard hierarchy types like “Is-a” and “Polymorphism”. But when we take a look at our diagram the ‘Board’ class really stands out. There are a lot of methods contained in this class, it might be a good idea to break this class down in several smaller classes in order to avoid a ‘god’ class.


board
Board
+BOARDSIZE : Integer
-board : Cell[][]
-initBoard()
+isAdjacent(x1 : Integer,y1 : Integer,x2 : Integer,y2 : Integer) : Boolean
+isTripleAt(x : Integer,y : Integer) : Boolean
+isTripletInDir(x : Integer,y : Integer,dir : Direction) : Boolean
+getChainAt(x : Integer,y : Integer,dir : Direction) : Position
+chainedCells() : Position
+removeChains()
+falldown()
+fillEmptyCells()
+hasChain() : Boolean
+checkMoves() : Boolean
+getNeighbourAt(x : Integer,y : Integer,dir : Direction) : Cell
+getNeighboursOf(x : Integer,y : Integer) : Cell
+swap(x1 : Integer,y1 : Integer,x2 : Integer,y2 : Integer)
+calculateScore(bonus : Integer) : Integer
64
1
Cell
-occupant : Gem
+isEmpty() : Boolean
1
0..1
Gem
-type : GemType
+Gem(t : GemType)
requires
uses
<<enumeration>>
Direction
NORTH
EAST
SOUTH
WEST
+Direction(x : Integer,y : Integer)
<<enumeration>>
GemType
BLUE
RED
GREEN
YELLOW
WHITE
PURPLE
ORANGE
+randomGem()
uses
Position
-x : Integer
-y : Integer
+Position(x : Integer,y : Integer)
requires
gui
ScorePanel
-scoreLabel : JLabel
-errorLabel : JLabel
-restartButton : JButton
+setScore(score : Integer)
+setError(error : String)
+noError()
1
1
GUI
#mainFrame : JFrame
#scorePanel : ScorePanel
-boardPanel : BoardPanel
#game : Game
#gui : GUI
-WINDOW_X : Integer
-WINOW_Y : Integer
+main(args[] : String)
+createGUI()
1
1
BoardPanel
-labels : JLabel[][]
-selected : Boolean
-selectedGem : Gem
-selectedPosition : Position
-selectedLabel : JLabel
+initBoard()
+refresh()
+clicked(clicked : JLabel)
+displayClicked(clicked : JLabel,gem : Gem)
+displayUnClicked(clicked : JLabel,gem : Gem)
+makeMove(p2 : Position)
+mousePressed(e : MouseEvent)
requires
uses
game
Game
-player : Player
-board : Board
-inProgress : Boolean
+start()
+stop()
+inProgress() : Boolean
+reset()
+makeMove(x1 : Integer,y1 : Integer,x2 : Integer,y2 : Integer)
uses
<<utility>>
LOGGER
-PATH : String
-logInfo : Boolean
-logWarning : Boolean
-out : PrintWriter
+logInfo(text : String)
+logWarning(text : String)
-writeLog(level : String,text : String)
+setLogLevel()
1
1
Player
-score : Integer
+reset()
+addScore(add : Integer)
uses
uses
exceptions
MoveNotValidException
+MoveNotValidException()
+MoveNotValidException(s : String)
+MoveNotValidException(s : String,throwable: Throwable)
+MoveNotValidException(throwable : Throwable)

Exercise 3 - Logging

Below is a class responsibility card for the Logger class that we have created. The reason the Logger class has no collaborators, is because this class does not call upon any of our classes. Instead, classes that want something to be logged, call upon the Logger to get stuff logged.

Logger	
Superclass: none	
Subclass: none	
Responsibilities: Log messages to a log file	Collaborators: None.