Project Assignment 1

Eskil Opdahl Nordland and Julian Jark.

Project repository: https://github.com/Eskalol/INF3121

Program Description

We have decided to work on 5.Labyrinth-Java, which is a labyrinth game written in java, where the objective is to escape the labyrinth. The program also has functionality to store and show high scores (in memory).

Analysis for testable parts

Requirements:

- Program have to be able to generate a random solvable maze.
- Program have to be able to print the current state of the maze.
- Program have to be able to evaluate input left, right, up, down, exit and restart.
- When the input is none of the above, an error message will be displayed and the user will be prompted for new input.
- When the user inputs directional input, his/hers position in maze is updated accordingly.
- If a user tries to move into a wall in the maze, it will not be accepted and reported to the user as an invalid move.
- when the user have reached the edge of the mace, then the user have escaped and won.
- for each game a counter with the number of valid moves made is accumulated and kept.
- when the user has won the user get an option to write his/her name which then will be stored in the programs highscore board with the number of moves made to escape the maze if its number of moves made is less or equal to the #5 on the highscore board.
- The programs highscore board will always be sorted by the number of moves made in ascending order.
- The highscore board will be printed after each game.
- A new game will be started after the completion a game

Design of manual tests

We have used black box test design technique.

Table of manual test cases

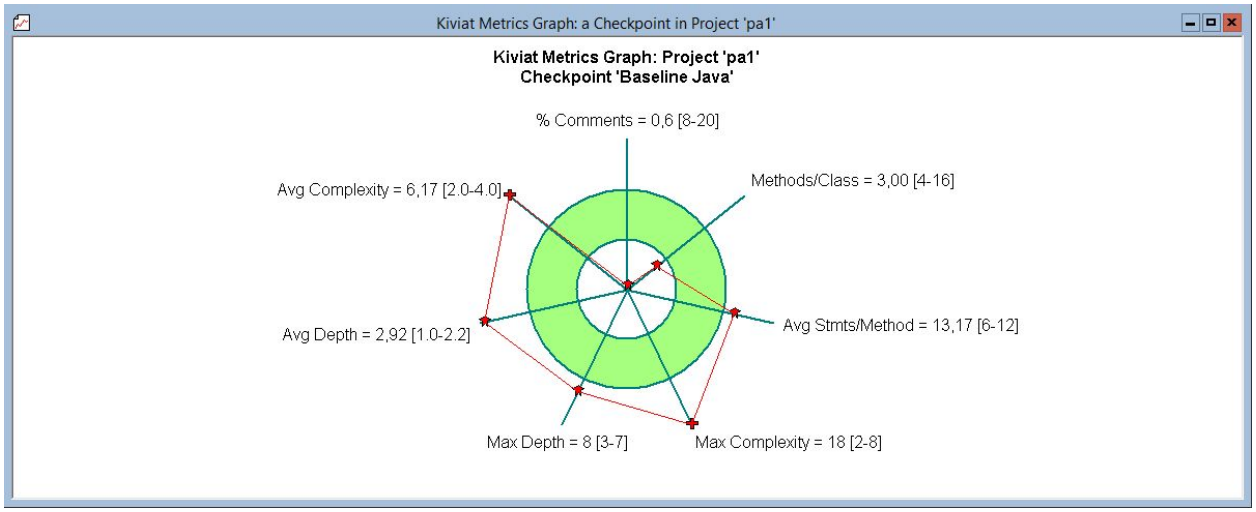
Name	**Pre-condition**	**Post-condition**	**Steps**	**Expected results**	**Actual result**
Maze given is solvable	Start of the game	Start of the game	Trace with eyes and/or fingers and try to exit the maze	maze is solvable	
Wrong input is handled	Start of the game	Error message is displayed	Write jiberish	It's handled	
Directional input works	Start of the game	Player is moved in the chosen direction	Write L, D, R, U	Player position is updated.	
Move into a wall is denied.	Right next to a wall.	Right next to a wall.	Move into a wall.	Error message is displayed.	
Player wins by reaching the edge.	Player is next to the edge.	Player has won	Move out of the edge.	Player wins	
User is able to write his/her name, and user is displayed on the highscore board.	Has just won	Player is displayed on the highscore board	User write his/her name	Player is displayed on the highscore board with amount of moves made.	
Highscore board is updated correctly and only top 5 will be added.	Start of game #1.	End of game #6.	Play 6 games	The top fives games is displayed in ascending order.	

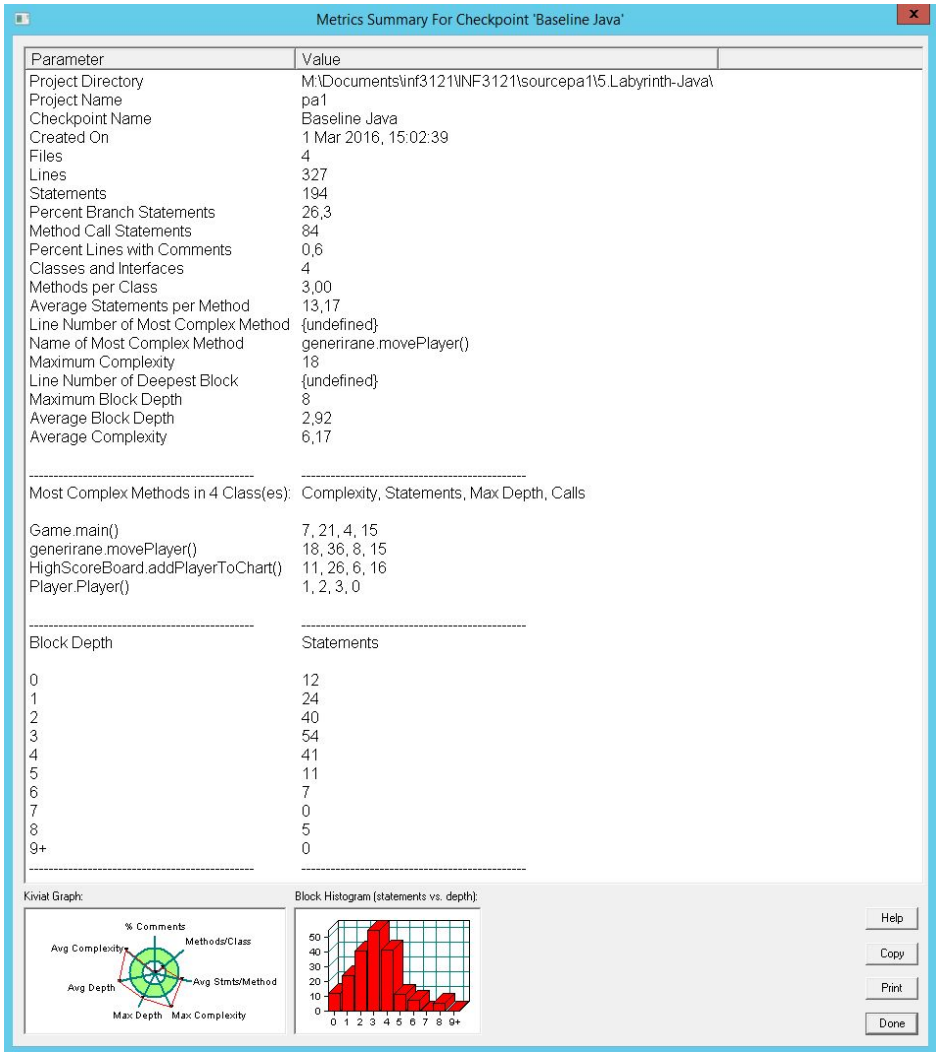
Cannot tell if our experience did help us writing the tests, it probably did.

Functional vs non-functional tests

No. Since this program is a small simple game, we only care that it works as according the requirements we made, which is that it’s playable. There functional testing is sufficient, and factors like reliability is indirectly tested when we test for functionality.

Metrics at project level





Parameter	Value
Project Directory	M:\Documents\inf3121\INF3121\sourcepa1\5.Labyrinth-Java\
Project Name	pa1
Checkpoint Name	Baseline Java
Created On	1 Mar 2016, 15:02:39
Files	4
Lines	327
Statements	194
Percent Branch Statements	26,3
Method Call Statements	84
Percent Lines with Comments	0,6
Classes and Interfaces	4
Methods per Class	3,00
Average Statements per Method	13,17
Line Number of Most Complex Method	{undefined}
Name of Most Complex Method	generirane.movePlayer()
Maximum Complexity	18
Line Number of Deepest Block	{undefined}
Maximum Block Depth	8
Average Block Depth	2,92
Average Complexity	6,17
----------	----------
Most Complex Methods in 4 Class(es):	Complexity, Statements, Max Depth, Calls
Game.main()	7, 21, 4, 15
generirane.movePlayer()	18, 36, 8, 15
HighScoreBoard.addPlayerToChart()	11, 26, 6, 16
Player.Player()	1, 2, 3, 0
----------	----------
Block Depth	Statements
0	12
1	24
2	40
3	54
4	41
5	11
6	7
7	0
8	5
9+	0
----------	----------

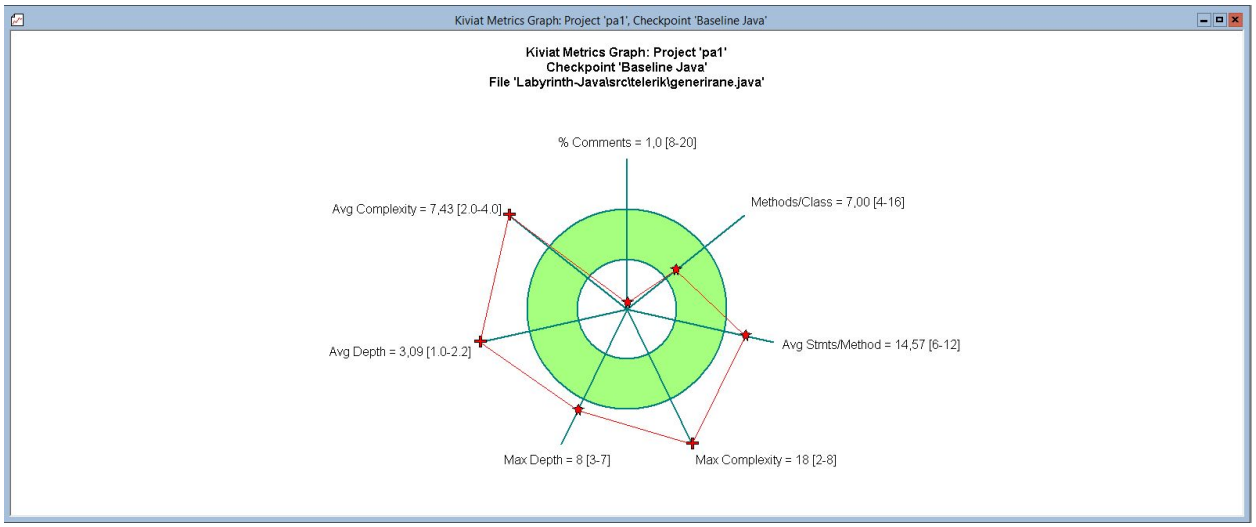
- What metrics do you spot for the whole project in the window Metrics Summary for Checkpoint? Write a brief description of the metrics. Try to explain their values (below what is expected, as expected or above the expected level). What metrics do you think need to change?
    - Files - *count of files in project.*
        - value: 4
    - Lines - *total line count in project.*
        - value: 327
    - Statements - *total statement count in project.*
        - value: 194
    - Percent Branch Statements - *percent of statements which also are branch statements in project.*
        - value: 26.3
    - Method Call Statements - *number of statements that calls another method*
        - value: 84
    - Classes and Interfaces - *number of classes and interfaces*
        - value: 4
    - Methods per Class - *average methods per class*
        - value: 3.00
        - **because of the high complexity and block depth, this will change as a result of refactorization of the code**
    - Average Statements per Method - *…*
        - value: 13.17
    - Maximum Complexity - *the max complexity found in a method*
        - value: 18
        - **Needs to change, no method should have so high complexity.**
    - Max Block Depth - *the max block depth*
        - value: 8
        - **Needs to go down**
    - Average Block Depth - *average block depth*
        - value: 2.92
    - Average Complexity - *average complexity*
        - value: 6.17
        - **Needs to go down**
    - For each most complex method in each class (Complexity, statements, max depth, calls):
        - Game.main() - 7, 21, 4, 15
        - generiane.movePlayer() - 18, 36, 8, 15
            - 18 complexity and 8 max depth is really high, this method should refactored
        - HighScoreBoard.addPlayerToChart() - 12, 26, 6, 16
            - not as high as movePlayer, but still needs to be refactored
        - Player.Player() - 1, 2, 3, 0

    - Block Depth with the number of statements for each depth

- Which is the biggest file you have in your project by the number of lines of code?
    - generirane.java
- Which is the file with most branches in your project?
    - generirane.java
- Which is the file with most complex code? What metric did you choose to answer to this question?
    - generirane.java. We looked at Average and Max complexity, both those numbers where highest for generirane.java, in addition generiane has the highest depth and methods so the choice is pretty obvious. If it wasn't that obvious we could mulitiply average complexity with the number of methods and compare those (since the average complexity is based on the number of methods)

Metrics at file level

We have chosen to view generirane.java



- How do you interpret the metrics applied on your file?
  We see that everything except comments and method/class is above the optimal level. There are absolutely none comments, which is bad.
- How are they different the metrics you obtained on the whole project, compared with the metrics on this file?
    - Average complexity: this higher than the whole project.
    - Average depth: this is a bit higher than the project average depth
    - Max depth: this file contains the highest depth in the whole project, if we change this the whole project will benefit from it.
    - Max complexity: this file also contains the highest complexity in the whole project, this needs to go down.
    - Average statements per method: this is a little bit higher than the project

- Would you refactor (re-write) any of the methods you have in this file?
    - Yes, movePlayer(), isSolvable(), inputCommand() has all complexity over 8.

After code changes

Identification

We need to improve average statements/method, Depth, Complexity and write more comments.

Example of improved and refactored code.

Before:

```
46          public boolean isSolvable(int row, int col){
47                  if((row==6)||(col==6)||(row==0)||(col==0)){
48                          isExit = true;
49                          return isExit;
50                  }
51                  if((maze[row-1][col]=='-')){
52                          if((isVisited[row-1][col]==false)) {
53                                  isVisited[row][col] = true;
54                                  isSolvable(row - 1, col);
55                          }
56                  }
57                  if((maze[row+1][col]=='-')){
58                          if((isVisited[row+1][col]==false)){
59                          isVisited[row][col]=true;
60                          isSolvable(row+1, col);
61                          }
62                  }
63                  if((maze[row][col-1]=='-')){
64                          if((isVisited[row][col-1]==false)) {
65                                  isVisited[row][col] = true;
66                                  isSolvable(row, col - 1);
67                          }
68                  }
69                  if((maze[row][col+1]=='-')){
70                          if((isVisited[row][col+1]==false)) {
71                                  isVisited[row][col] = true;
72                                  isSolvable(row, col + 1);
73                          }
74                  }
75                  return isExit;
76          }
```

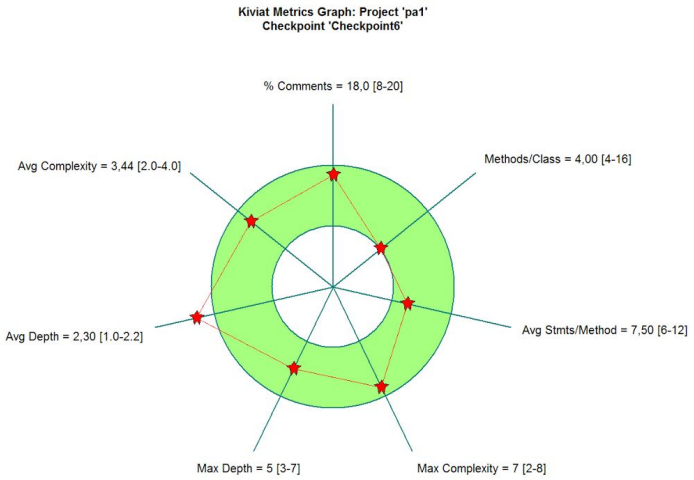
:

After:

```
59          /**
60           * Try to solve the maze by recursively going into all the adjacent directions
61           * until we have expunged all valid paths, if one of the paths reaches the edge, true is returned.
62           *
63           */
64          public boolean isSolvable(int row, int col){
65                  // Check if this is a valid position
66                  if(maze[row][col] != '-' || isVisited[row][col]) {
67                          return false;
68                  }
69
70                  // Check if we have escaped the maze
71                  if(isAtEdge(row, col)){
72                          return true;
73                  }
74
75                  // Mark this position as visited
76                  isVisited[row][col] = true;
77
78                  // go into adjacent directions
79                  return isSolvable(row-1, col) || isSolvable(row+1, col) || isSolvable(row, col-1) || isSolvable(row, col+1);
80          }
```

We can see that the code is now much smaller, easier to read, and better designed.

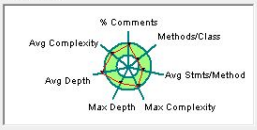
Metrics at project level



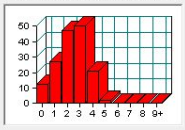
Metrics Summary For Checkpoint 'Checkpoint6'

Parameter	Value
Project Directory	M:\INF3121\sourcepa1\5.Labyrinth-Java\
Project Name	pa1
Checkpoint Name	Checkpoint6
Created On	10 Mar 2016, 16:25:17
Files	4
Lines	289
Statements	159
Percent Branch Statements	23,9
Method Call Statements	77
Percent Lines with Comments	18,0
Classes and Interfaces	4
Methods per Class	4,00
Average Statements per Method	7,50
Line Number of Most Complex Method	{undefined}
Name of Most Complex Method	generirane.isSolvable()
Maximum Complexity	7
Line Number of Deepest Block	{undefined}
Maximum Block Depth	5
Average Block Depth	2,30
Average Complexity	3,44
---	---
Most Complex Methods in 4 Class(es):	Complexity, Statements, Max Depth, Calls
Game.main()	4, 20, 4, 16
generirane.isSolvable()	7, 6, 3, 5
HighScoreBoard.addPlayerToChart()	5, 13, 4, 10
Player.Player()	1, 2, 2, 0
---	---
Block Depth	Statements
0	12
1	27
2	47
3	50
4	21
5	2
6	0
7	0
8	0
9+	0
---	---

Kiviat Graph:



Block Histogram (statements vs. depth):

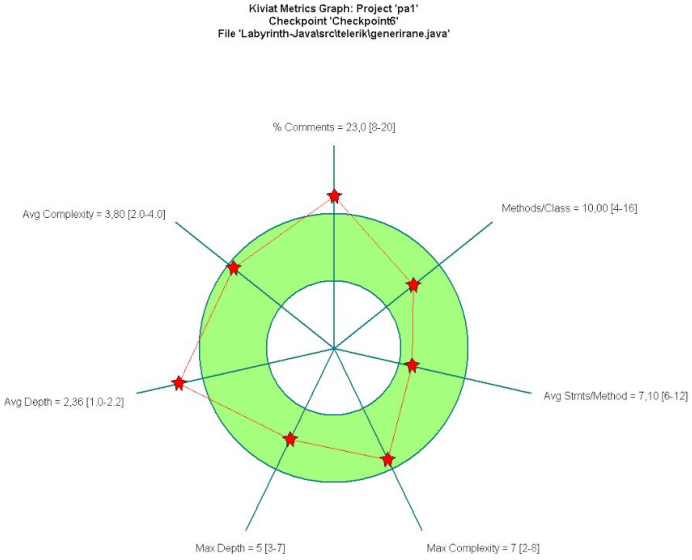


Help
Copy
Print
Done

We can see that all points except Avg. Depth is now inside the optimal kiviat range, which is good. We have now reduced average complexity and average statements/method with almost 45%. Max depth is reduced by 3. Average depth is still a bit high, but we can't do much more about it.

Metrics at file level

We look at generaine.java



Here we can also see that almost all points are inside the optimal levels, which they were not in the original code.

Apparently it says that number of comments is higher than optimal, but that should be taken with a grain of salt, say for example if you describe a complex algorithm, it might take a whole lot more space than the actual algorithm, but the description is very much needed for a programmer to understand it.

Remarks

The methods should been more functional, there a lot of methods with non-obvious side effects, like printing to the terminal.
The file and class name generaine makes no sense, it should rather be named Labyrinth