Software Analysis : pacman systems
Project report for Software Evolution course

Group 3

BOOSKO Sam
DECOCQ Rémy
SCHERER Robin

Academic Year 2019-2020
Master Computers Science, block 2
Faculté des Sciences, Université de Mons

Contents

Introduction

1 Quality analysis of the initial versions

1.1 System 2 (Rémy)

1.1.1 Generalities

First of all, this is noticeable that authors provide some documents coupled with the implementation, even if it is not mentioned in the README file. This additional material is available under the `out/` directory at project roots and comprises :

- A `.pdf` file describing shortly the game, the controls and the multiplayer (2 players) mode available
- A complete class diagram covering the whole implementation
- A sequence diagram stating the execution flow when Pacman arrives on a cell and so "eat" what is at this place
- A graph of the mathematical function used to correlate difficulty with player's progression

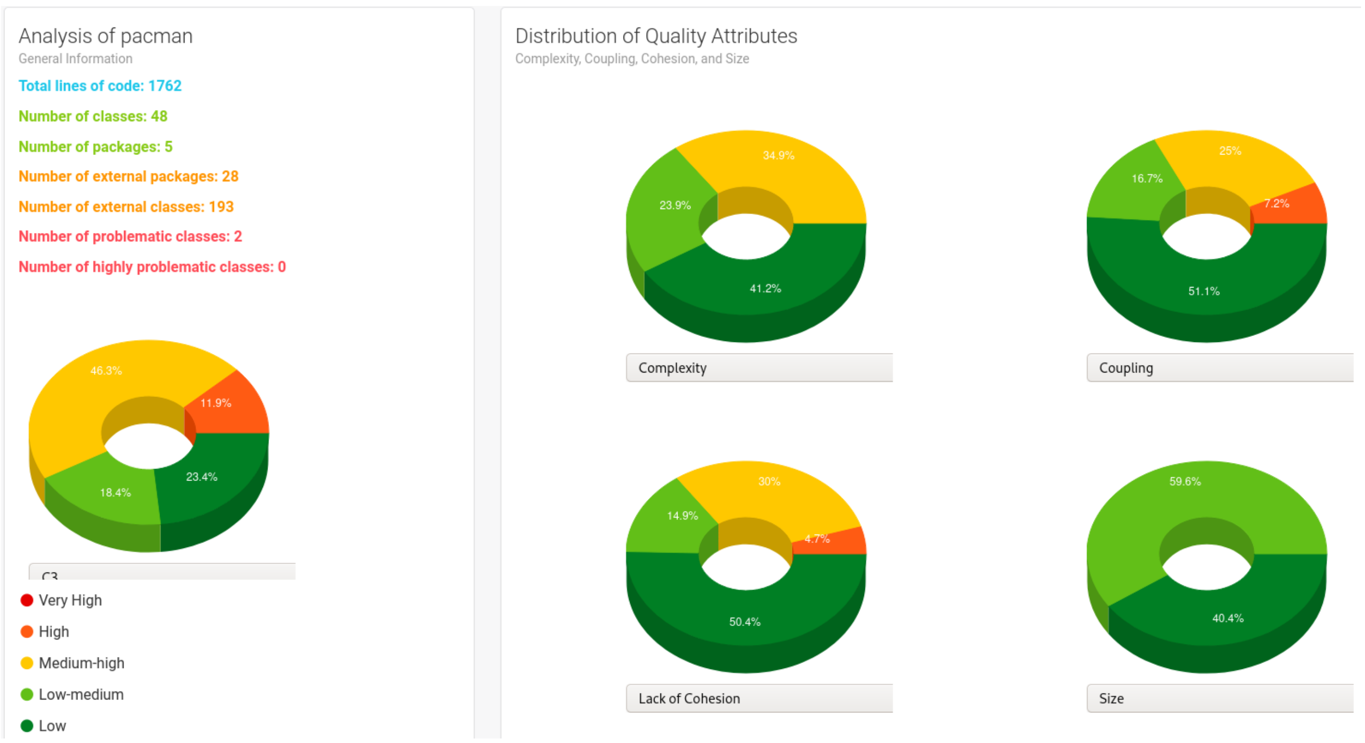
We also observe that in this Pacman implementation maps are modelized under `.tmx` format, that is a popular way to deal with board games[1]. Only one single basic map is provided.

The project structure is classic, we have `main` and `test` separation under the `src` directory, each containing packaged sources. The building system provided with the implementation is hold by Gradle. So a switch to Maven will be required to comply with directives.

1.1.2 Static Analysis

1.1.2.1 Code metrics (CodeMR)

CodeMR allows to get an overall idea of the actual health of the system considering several metrics. The dashboard illustrated by Figure 1 informs this software is doing quite good.



Analysis of pacman
General Information
Total lines of code: 1762
Number of classes: 48
Number of packages: 5
Number of external packages: 28
Number of external classes: 193
Number of problematic classes: 2
Number of highly problematic classes: 0

Distribution of Quality Attributes
Complexity, Coupling, Cohesion, and Size

Figure 1: CodeMR dashboard summarizing health of system 2

[1] https://doc.mapeditor.org/en/stable/reference/support-for-tmx-maps/

The Figure 2 illustrates also the C3 metric but coupled with detailed packages view. We notice authors apparently tried to follow some Model-View-Controller pattern to design their application. The C3 metric is defined as the maximum between 3 other well representative metrics : *Coupling*, *Cohesion* and *Complexity*. These are defined in the codeMR documentation[2]. We notice that, following the dashboard overview, two classes are impacting the software quality from the point of view of C3 metric.

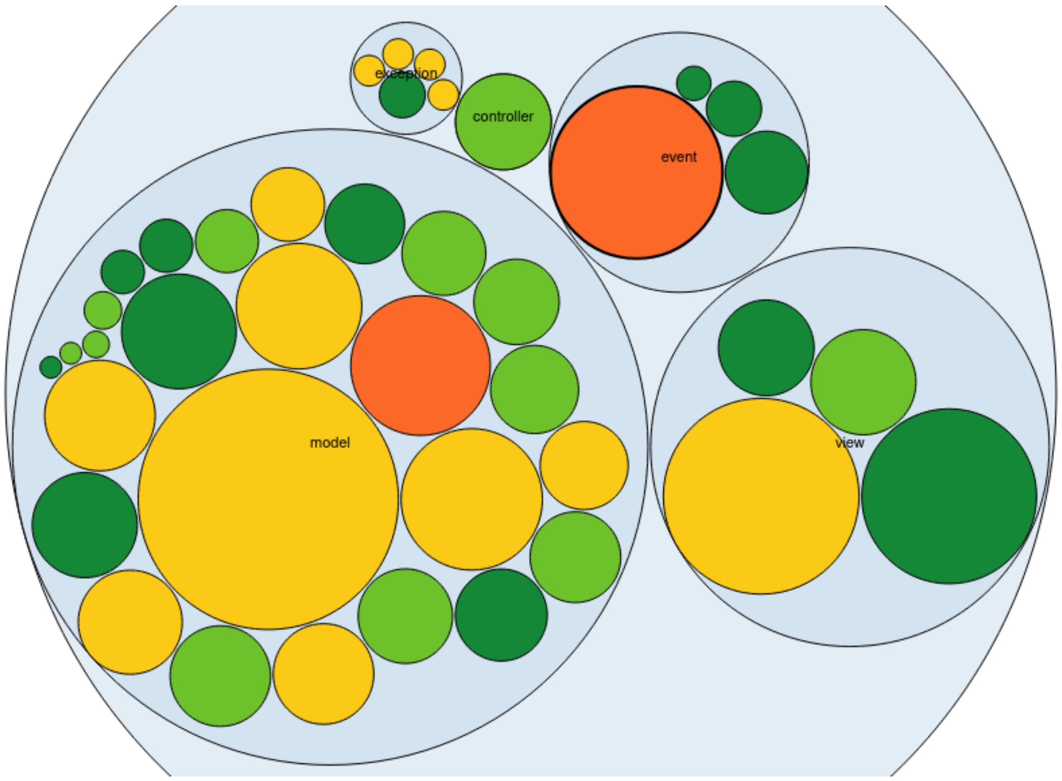


Figure 2: C3 Metric by package for system 2

The details of the measurements on the two more problematic classes are given by Figure 3 and Figure 4, for respectively *event.WorkerProcess* and *model.Ghost*. For both, two metrics are considered as high value, the meaning described by CodeMR is

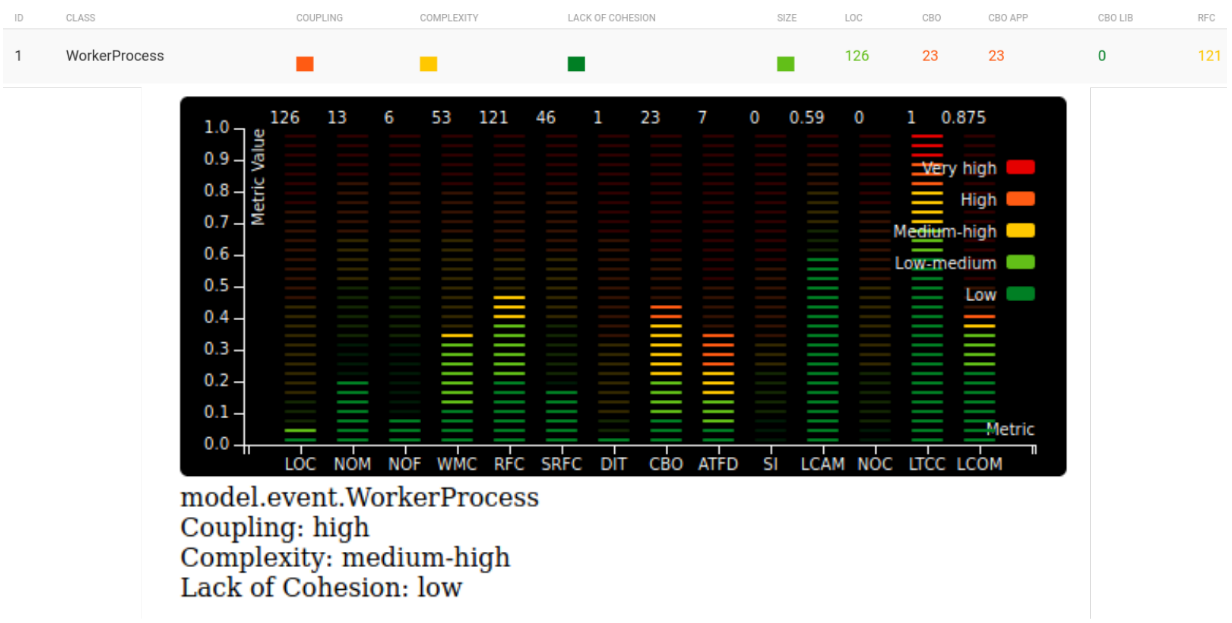
- LTCC : The Lack of Tight Class Cohesion metric measures the lack cohesion between the public methods of a class. That is the relative number of directly connected public methods in the class. Classes having a high lack of cohesion indicate errors in the design.
- LCOM : Measure how methods of a class are related to each other. Low cohesion means that the class implements more than one responsibility. A change request by either a bug or a new feature, on one of these responsibilities will result change of that class. Lack of cohesion also influences understandability and implies classes should probably be split into two or more subclasses.

In addition, for *event.WorkerProcess* we have :

- CBO : The number of classes that a class is coupled to. It is calculated by counting other classes whose attributes or methods are used by a class, plus those that use the attributes or methods of the given class.
- AFTD : Access to Foreign Data is the number of classes whose attributes are directly or indirectly reachable from the investiggated class. Classes with a high ATFD value rely strongly on data of other classes and that can be the sign of the God Class.

Other codeMR metrics did not revelate relevant problems in the implementation.

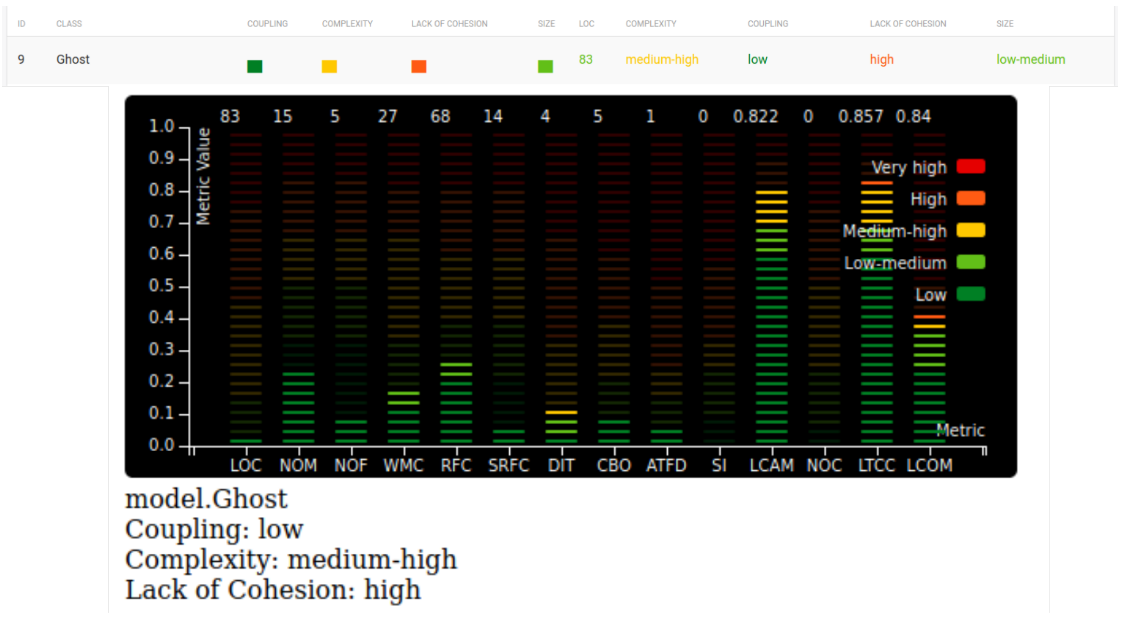
[2]https://www.codemr.co.uk/documents



ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	CBO	CBO APP	CBO LIB	RFC
1	WorkerProcess	🟧	🟨	🟩	🟩	126	23	23	0	121

model.event.WorkerProcess
Coupling: high
Complexity: medium-high
Lack of Cohesion: low

Figure 3: *event.WorkerProcess* class main metrics measurements



ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
9	Ghost	🟩	🟨	🟧	🟩	83	medium-high	low	high	low-medium

model.Ghost
Coupling: low
Complexity: medium-high
Lack of Cohesion: high

Figure 4: *model.Ghost* class main metrics measurements

1.1.2.2 Dependencies (CodeMR, Intellij analyzer)

CodeMR allows also to inspect dependency relations between classes coupled with the metrics measured for each. We observe in the Figure 5 the same structure that in the class diagram. Once again the class *event.WorkerProcess* is displayed as problematic, being too complex and coupled with other classes.

We use the standard built-in tool of IntellIJ IDEA to instantiate the dependency matrix, illustrated by Figure 7. We clearly see reading 8th column that *event.WorkerProcess* depends on a lot of other classes from package *model*. This is also the case for *model.Map* that presents a lot of cyclic dependencies (red marked).

1.1.2.3 Compliance & bad smells (PMD, Designite)

PMD is a statical analyzer that checks for problems of several natures in the code. It detected more than 1500 violations in the system 2, related to various topics (see Figure 6).

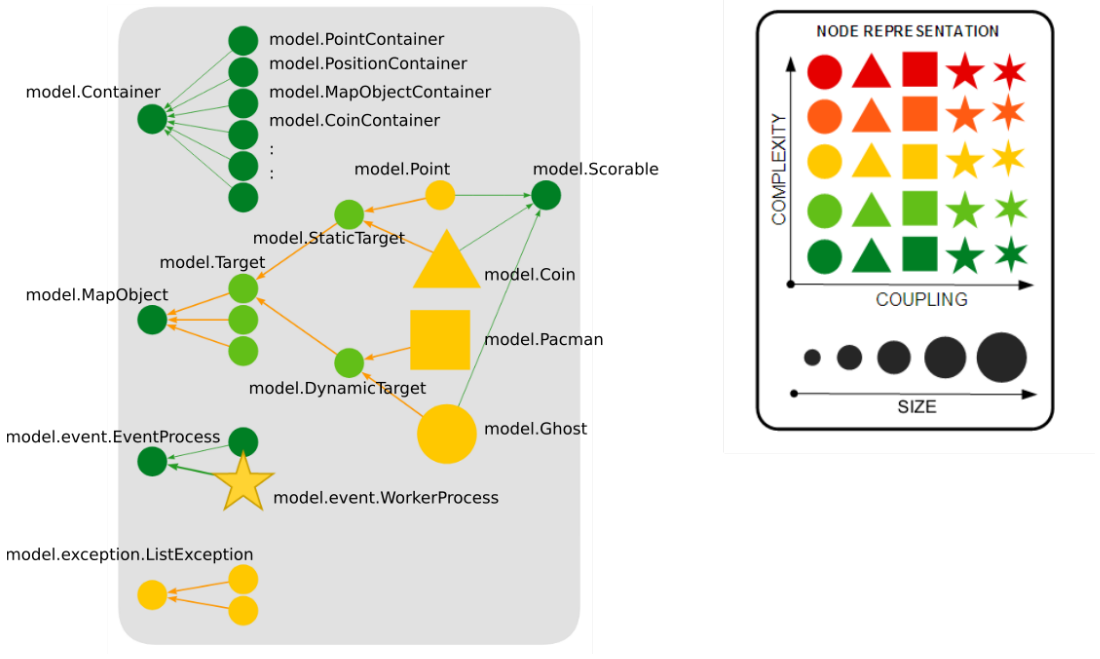
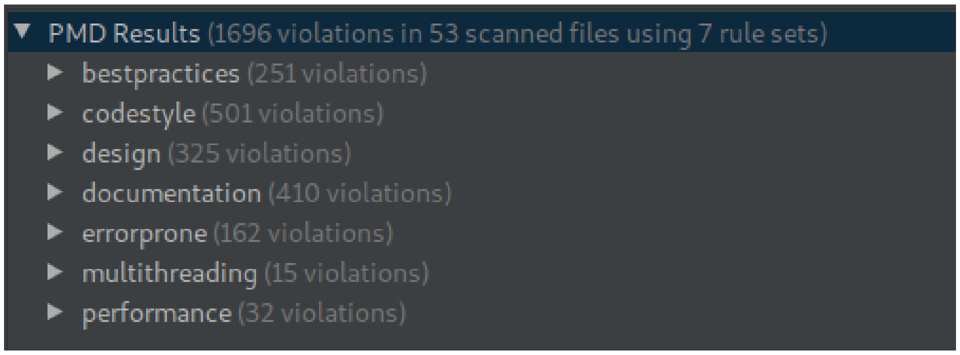
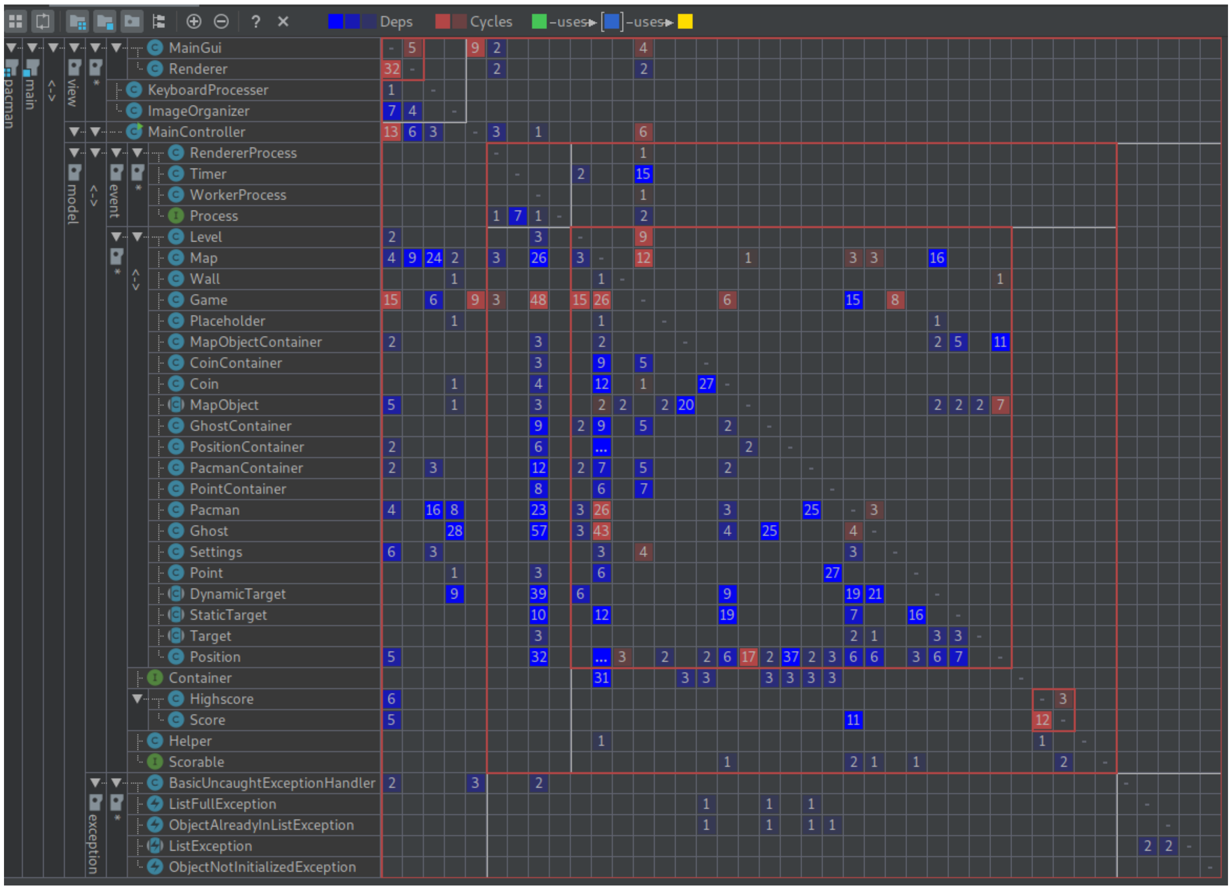


Figure 5: Inheritance relations between classes in system 2



```
▼ PMD Results (1696 violations in 53 scanned files using 7 rule sets)
   ▶ bestpractices (251 violations)
   ▶ codestyle (501 violations)
   ▶ design (325 violations)
   ▶ documentation (410 violations)
   ▶ errorprone (162 violations)
   ▶ multithreading (15 violations)
   ▶ performance (32 violations)
```

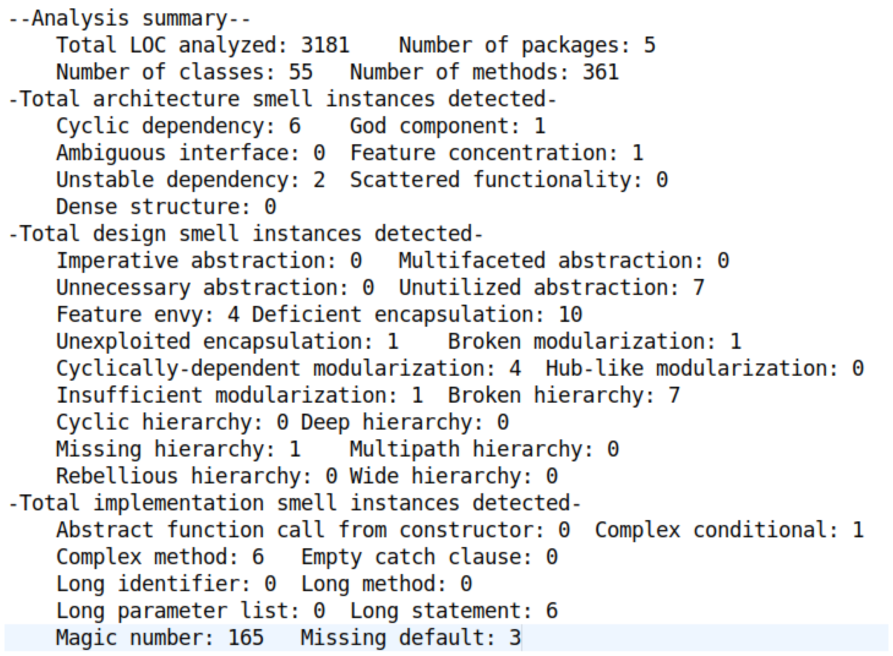
Figure 6: Violations found by PMD in system 2



MainGui	-	5			9	2							4																											
Renderer	32	-				2							2																											
KeyboardProcesser	1		-																																					
ImageOrganizer	7	4		-																																				
MainController	13	6	3		-	3		1					6																											
RendererProcess						-							1																											
Timer							-			2			15																											
WorkerProcess								-					1																											
Process						1	7	1	-				2																											
Level	2							3		-			9																											
Map	4	9	24	2		3		26		3	-		12					1					3	3			16													
Wall				1							1	-																		1										
Game	15		6		9	3		48		15	26		-				6						15		8															
Placeholder				1							1			-													1													
MapObjectContainer	2							3			2				-												2	5		11										
CoinContainer								3			9		5			-																								
Coin				1				4			12		1			27	-																							
MapObject	5			1				3			2	2		2	20			-									2	2	2	7										
GhostContainer								9		2	9		5				2		-																					
PositionContainer	2							6			...							2		-																				
PacmanContainer	2		3					12		2	7		5				2				-																			
PointContainer								8			6		7									-																		
Pacman	4		16	8				23		3	26						3				25		-	3																
Ghost				28				57		3	43						4		25				4	-																
Settings	6		3								3		4										3		-															
Point				1				3			6											27				-														
DynamicTarget				9				39		6							9						19	21			-													
StaticTarget								10			12						19						7			16		-												
Target								3															2	1			3	3	-											
Position	5							32			...	3		2		2	6	17	2	37	2	3	6	6		3	6	7		-										
Container											31				3	3			3	3	3	3									-									
Highscore	6																															-	3							
Score	5																						11									12	-							
Helper											1																					1		-						
Scorable																	1						2	1		1							2		-					
BasicUncaughtExceptionHandler	2				3			2																												-				
ListFullException																1			1		1																-			
ObjectAlreadyInListException																1			1		1	1																-		
ListException																																					2	2	-	
ObjectNotInitializedException																																								-

Figure 7: Dependency matrix for system 2

Designite is used to detect bad smells, a summary of the analyse is presented by Figure 8. The number of lines of code and classes is higher than the ones returned by CodeMR because the test sources were considered in the analysis.



```
--Analysis summary--
    Total LOC analyzed: 3181    Number of packages: 5
    Number of classes: 55   Number of methods: 361
-Total architecture smell instances detected-
    Cyclic dependency: 6    God component: 1
    Ambiguous interface: 0  Feature concentration: 1
    Unstable dependency: 2  Scattered functionality: 0
    Dense structure: 0
-Total design smell instances detected-
    Imperative abstraction: 0   Multifaceted abstraction: 0
    Unnecessary abstraction: 0  Unutilized abstraction: 7
    Feature envy: 4 Deficient encapsulation: 10
    Unexploited encapsulation: 1    Broken modularization: 1
    Cyclically-dependent modularization: 4  Hub-like modularization: 0
    Insufficient modularization: 1  Broken hierarchy: 7
    Cyclic hierarchy: 0 Deep hierarchy: 0
    Missing hierarchy: 1    Multipath hierarchy: 0
    Rebellious hierarchy: 0 Wide hierarchy: 0
-Total implementation smell instances detected-
    Abstract function call from constructor: 0  Complex conditional: 1
    Complex method: 6   Empty catch clause: 0
    Long identifier: 0  Long method: 0
    Long parameter list: 0  Long statement: 6
    Magic number: 165   Missing default: 3
```

Figure 8: Designite in-line use results for system 2

1.2 System 3 (Robin)

1.2.1 Generalities

For this project the author provides no documents.

The Pacman maps are modelized under a `.txt` format, where each type of case are attributed a certain letter.

We also observe that in this Pacman implementation maps are modelized under `.tmx` format, that is a popular way to deal with board games[3]. Only one single basic map is provided.

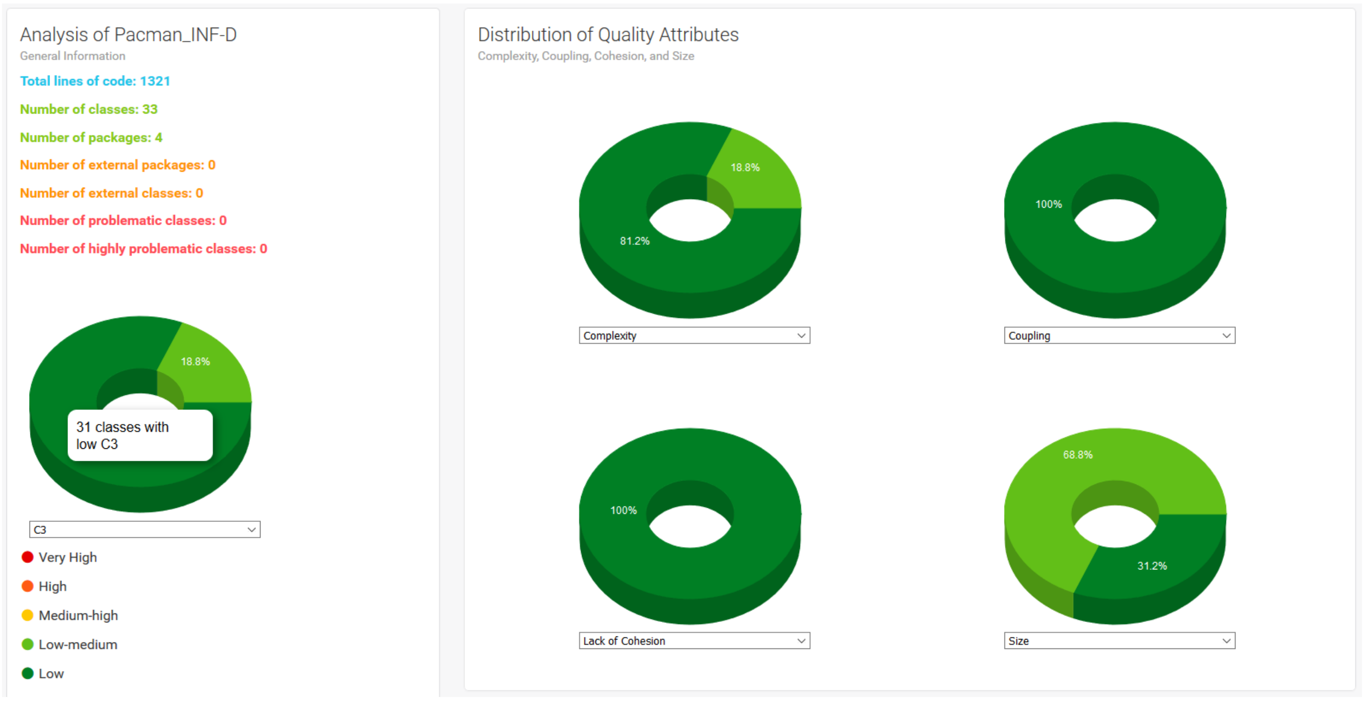
In the project structure, there is `test` and `src` directory. In the first one there are the test classes and in the latter there are two directories, `Ressources` and `pacman_infd`. In `Ressources` we have the Pacman maps that are modelled under a `.txt` format, where each type of case are attributed a certain letter, and also the sound files in `.wav` format. `pacman_infd` contains all Java classes.

The building system provided with the implementation is hold by Ant. So a switch to Maven will be required to comply with directives.

1.2.2 Static Analysis

1.2.2.1 Code metrics (CodeMR)

The dashboard illustrated by Figure 9 informs this software is doing really good.



Analysis of Pacman_INF-D
General Information
Total lines of code: 1321
Number of classes: 33
Number of packages: 4
Number of external packages: 0
Number of external classes: 0
Number of problematic classes: 0
Number of highly problematic classes: 0

Distribution of Quality Attributes
Complexity, Coupling, Cohesion, and Size

Figure 9: CodeMR dashboard summarizing health of system 3

[3] https://doc.mapeditor.org/en/stable/reference/support-for-tmx-maps/

The Figure 10 illustrates also the C3 metric but coupled with detailed packages view. We can see that every classes is doing good.

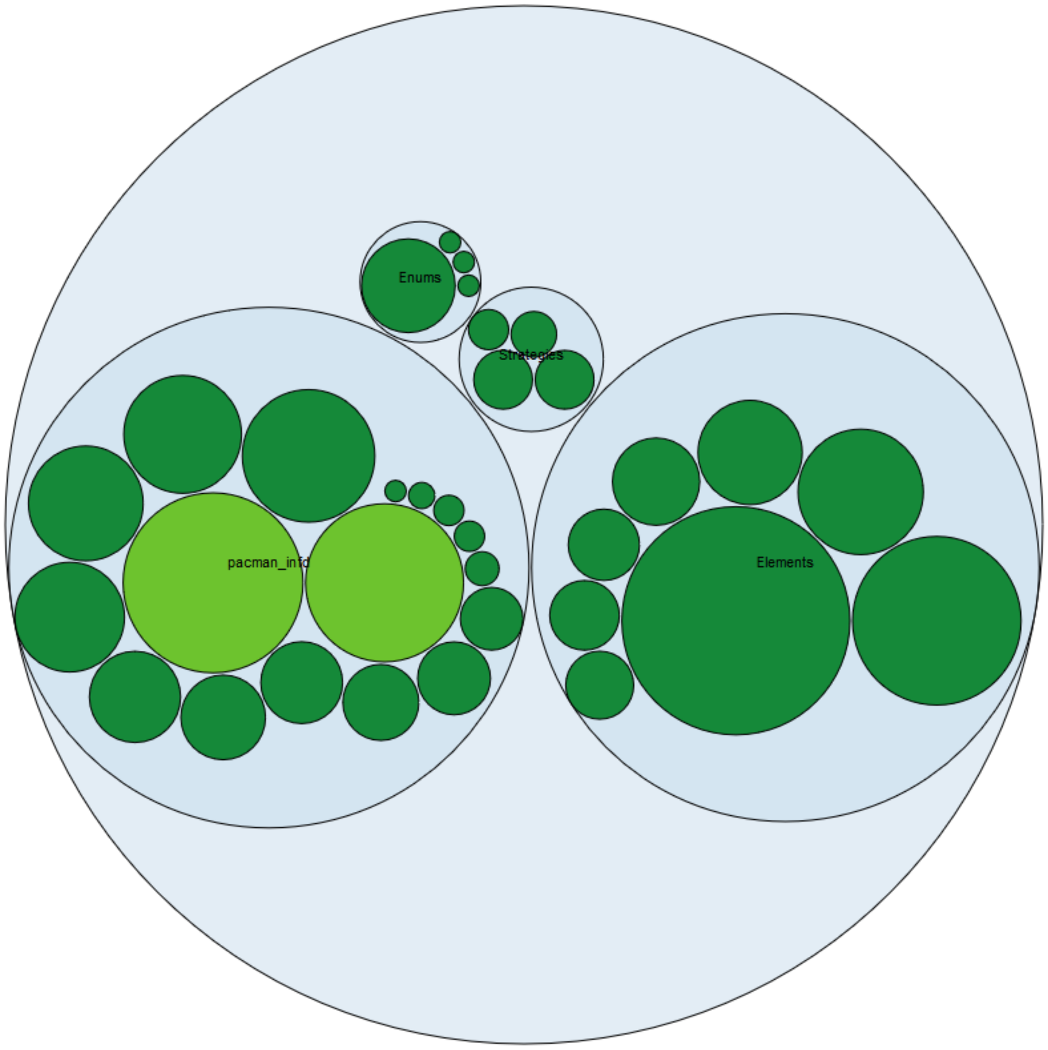
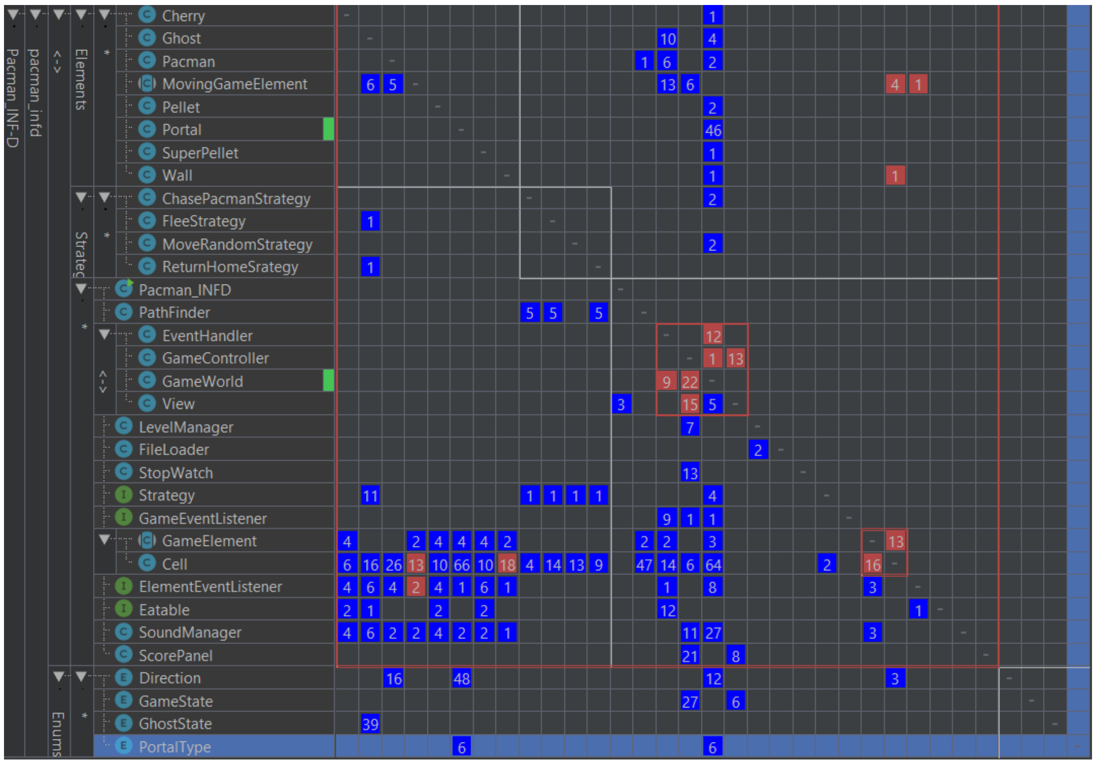


Figure 10: C3 Metric by package for system 3

There are no problems in the classes for almost all the metrics, only for two of them, we can see a problem: the Lack of Tight Class Cohesion and the lack of Cohesion of Methods. For the first one, nine classes have High or Very-High risks like the GameController or the View. And for the latter, three classes have High risks: GameController, ScorePanel and GameWorld.

1.2.2.2 Dependencies (CodeMR, Intellij analyzer)

We use the standard built-in tool of IntellIJ IDEA to analyze the dependency matrix, the result is shown on Figure 11.



Cherry	-																1																
Ghost		-													10		4																
Pacman			-											1	6		2																
MovingGameElement		6	5	-											13	6									4	1							
Pellet					-												2																
Portal						-											46																
SuperPellet							-										1																
Wall								-									1								1								
ChasePacmanStrategy									-								2																
FleeStrategy		1								-																							
MoveRandomStrategy											-						2																
ReturnHomeSrategy		1										-																					
Pacman_INFD													-																				
PathFinder									5	5		5		-																			
EventHandler															-		12																
GameController																-	1	13															
GameWorld															9	22	-																
View													3			15	5	-															
LevelManager																7			-														
FileLoader																			2	-													
StopWatch																13					-												
Strategy		11							1	1	1	1					4					-											
GameEventListener															9	1	1						-										
GameElement	4			2	4	4	4	2						2	2		3							-	13								
Cell	6	16	26	13	10	66	10	18	4	14	13	9		47	14	6	64					2		16	-								
ElementEventListener	4	6	4	2	4	1	6	1							1		8							3		-							
Eatable	2	1			2		2								12											1	-						
SoundManager	4	6	2	2	4	2	2	1								11	27							3				-					
ScorePanel																21		8											-				
Direction			16			48											12								3					-			
GameState																27		6													-		
GhostState		39																														-	
PortalType						6											6																-

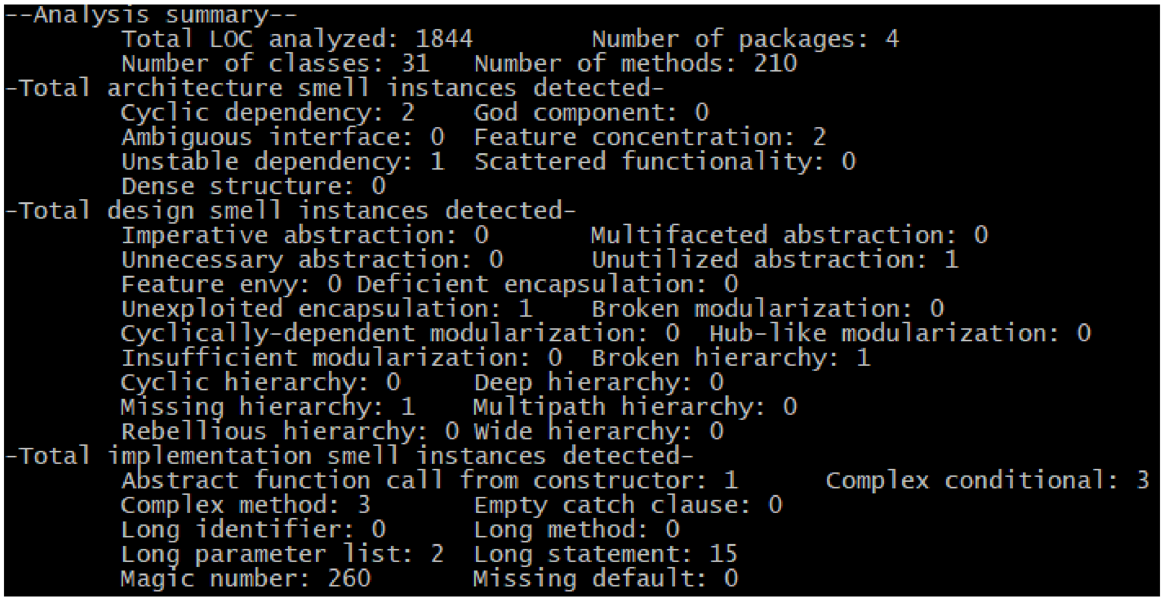
Figure 11: Dependency matrix for system 3

1.2.2.3 Compliance & bad smells (PMD, Designite)

We first used PMD, it detected more than 1343 violations in the system 3, here is the result:

- 1343 violations:
  - best practice: 54
  - code style: 414
    - 125: method arg could be final
    - 98: local var could be final
    - 60: short variable name
  - design: 503
    - 449: most is law of demeter 'only talk to friends'
    - 31: immutable field: private field values never change once object init could be final
  - documentation: 192
  - error prone: 167
    - 70: Bean member should serialize: make variable transient or static
    - 21: avoid literal in if condition
  - performance: 13

The result of Designite analysis is on Figure 12.



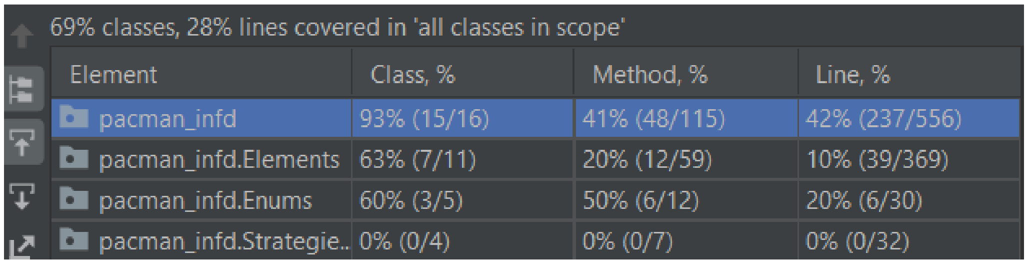
```
--Analysis summary--
        Total LOC analyzed: 1844        Number of packages: 4
        Number of classes: 31   Number of methods: 210
-Total architecture smell instances detected-
        Cyclic dependency: 2    God component: 0
        Ambiguous interface: 0  Feature concentration: 2
        Unstable dependency: 1  Scattered functionality: 0
        Dense structure: 0
-Total design smell instances detected-
        Imperative abstraction: 0       Multifaceted abstraction: 0
        Unnecessary abstraction: 0      Unutilized abstraction: 1
        Feature envy: 0 Deficient encapsulation: 0
        Unexploited encapsulation: 1    Broken modularization: 0
        Cyclically-dependent modularization: 0  Hub-like modularization: 0
        Insufficient modularization: 0  Broken hierarchy: 1
        Cyclic hierarchy: 0     Deep hierarchy: 0
        Missing hierarchy: 1    Multipath hierarchy: 0
        Rebellious hierarchy: 0 Wide hierarchy: 0
-Total implementation smell instances detected-
        Abstract function call from constructor: 1      Complex conditional: 3
        Complex method: 3       Empty catch clause: 0
        Long identifier: 0      Long method: 0
        Long parameter list: 2  Long statement: 15
        Magic number: 260       Missing default: 0
```

Figure 12: Designite in-line use results for system 3

1.2.3 Dynamic Analysis

1.2.3.1 Test coverage (Intellij built-in tool)

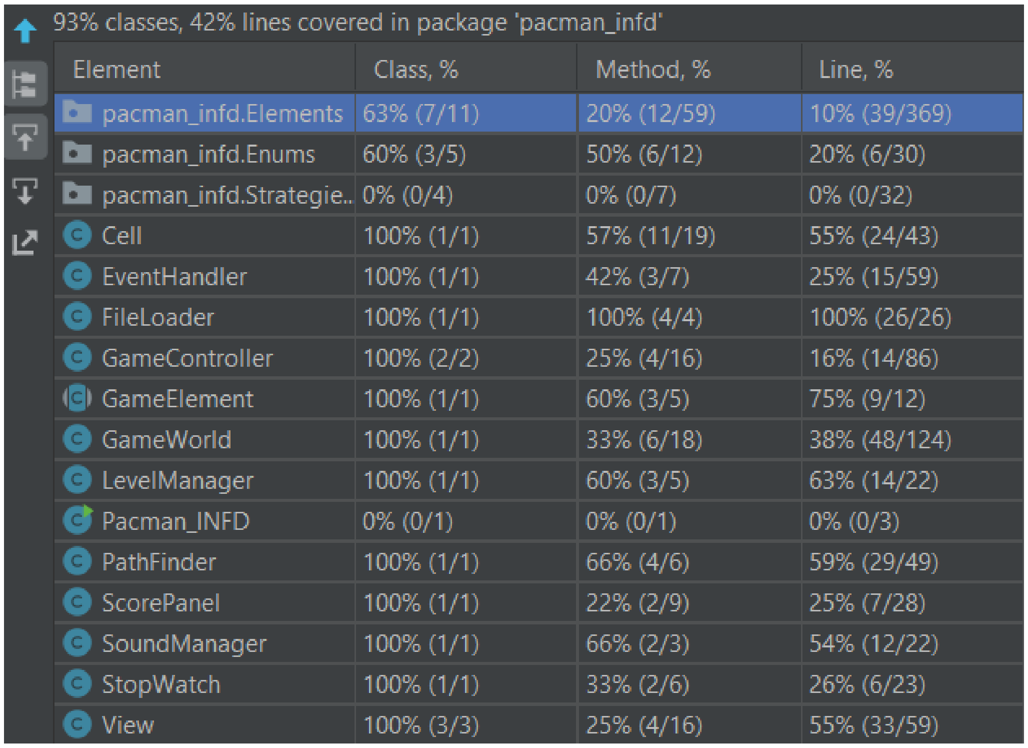
The test coverage of the given tests was analyzed, and the three following figures 13, 14 and 15 give the results. We can see that only 28% of the code is tested and 69% of the classes. We can see that neither of the strategies in the Strategies packages is tested, and in the Elements package only 12 methods is tested on the 59 methods that are available.



69% classes, 28% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
pacman_infd	93% (15/16)	41% (48/115)	42% (237/556)
pacman_infd.Elements	63% (7/11)	20% (12/59)	10% (39/369)
pacman_infd.Enums	60% (3/5)	50% (6/12)	20% (6/30)
pacman_infd.Strategie..	0% (0/4)	0% (0/7)	0% (0/32)

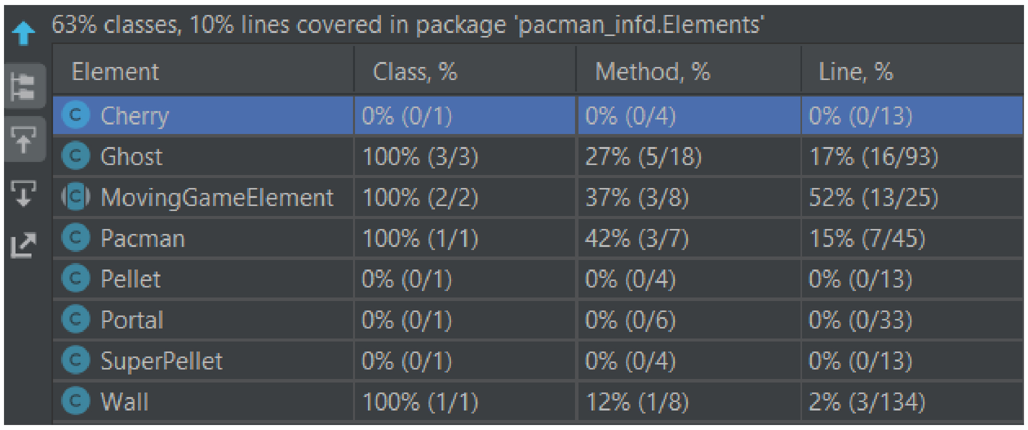
Figure 13: Test coverage for system 3



93% classes, 42% lines covered in package 'pacman_infd'

Element	Class, %	Method, %	Line, %
pacman_infd.Elements	63% (7/11)	20% (12/59)	10% (39/369)
pacman_infd.Enums	60% (3/5)	50% (6/12)	20% (6/30)
pacman_infd.Strategie..	0% (0/4)	0% (0/7)	0% (0/32)
Cell	100% (1/1)	57% (11/19)	55% (24/43)
EventHandler	100% (1/1)	42% (3/7)	25% (15/59)
FileLoader	100% (1/1)	100% (4/4)	100% (26/26)
GameController	100% (2/2)	25% (4/16)	16% (14/86)
GameElement	100% (1/1)	60% (3/5)	75% (9/12)
GameWorld	100% (1/1)	33% (6/18)	38% (48/124)
LevelManager	100% (1/1)	60% (3/5)	63% (14/22)
Pacman_INFD	0% (0/1)	0% (0/1)	0% (0/3)
PathFinder	100% (1/1)	66% (4/6)	59% (29/49)
ScorePanel	100% (1/1)	22% (2/9)	25% (7/28)
SoundManager	100% (1/1)	66% (2/3)	54% (12/22)
StopWatch	100% (1/1)	33% (2/6)	26% (6/23)
View	100% (3/3)	25% (4/16)	55% (33/59)

Figure 14: Test coverage of the pacman_infd package



63% classes, 10% lines covered in package 'pacman_infd.Elements'

Element	Class, %	Method, %	Line, %
Cherry	0% (0/1)	0% (0/4)	0% (0/13)
Ghost	100% (3/3)	27% (5/18)	17% (16/93)
MovingGameElement	100% (2/2)	37% (3/8)	52% (13/25)
Pacman	100% (1/1)	42% (3/7)	15% (7/45)
Pellet	0% (0/1)	0% (0/4)	0% (0/13)
Portal	0% (0/1)	0% (0/6)	0% (0/33)
SuperPellet	0% (0/1)	0% (0/4)	0% (0/13)
Wall	100% (1/1)	12% (1/8)	2% (3/134)

Figure 15: Test coverage for the Elements package

2 Quality improvement

2.1 System 3 (Robin)

The first change was changing the building system from Ant to Maven.

After that, we used PMD and Designite to find problems and refactor them. The access of classes, methods and variables was changed depending of the need, it was set to protected or private if it was possible. Some variables were set to final. We found magic number, and tried to remove them if it was posisble and usefull.

Then some long methods were divided into multiple methods for more readability. Some switch-case were also added instead of long if-else. Some new method were also added.

The Direction Enumerator class was changed to make it smaller and not just a succesion of switch-case.

The structure of the classes was also changed. The source code of the project is now divided in 5 packages: Elements, Enums, Fileloader, Game and Strategies. Elements contains all the moving elements of the game, like Pacman or the ghosts, and the other game elements like the cherry or the pellet. The Enums packages contains all the enumeration that are used in the project. The FileLoader package contains all the classes that are used to manage the loading of a level. Strategies contains all the strategies and pathfinders that are used by the ghosts. And the Game package contains everything else that is used for the game, the GameWorld, the Controller or the Listener for exemple.

Then the GUI was changed for Pacman, before changing the code Pacman wasn't changing his orientation on the application, now we can see Pacman change it's orientation. There is also a little animation for the mouth of Pacman, he opens and shut his mouth to simulate eating.

Some new test where also created on the test package to cover more code. The different strategies of the Strategy package was tested. New tests fot the eating of pellet, cherry and super pellet was also created.

3 Adding basic functionalities

3.1 System 3 (Robin)

In this implementation, we couldn't advance to the next level once we finished eating all the pellets. We first changed that. Now a level can be succeded and when it's done we advance to the next level, there is 3 level in total but the first 2 are the same.

The levels were also re-writen, in the levels there were some disconnected regions, so we removed that.

After that the behavior for the super pellet/power pill was also changed. The clock was not set in pause when the pill was activated. The scrone gained by eating the ghosts was also not the one set in the rules, so it was also changed. The time for the pills was also changed, the first two lasts 7 seconds and the last two lasts 5 seconds.

Eating a cherry on the initial implementation wasn't making pacman gain an additional live, so that was also added.

4 Adding new features

5 Quality evolution analysis

6 Conclusion