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 3 (Robin)

1.1.1 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[1]. 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.

1.1.2 Static Analysis

1.1.2.1 Code metrics (CodeMR)

The dashboard illustrated by Figure 1 informs this software is doing really good.

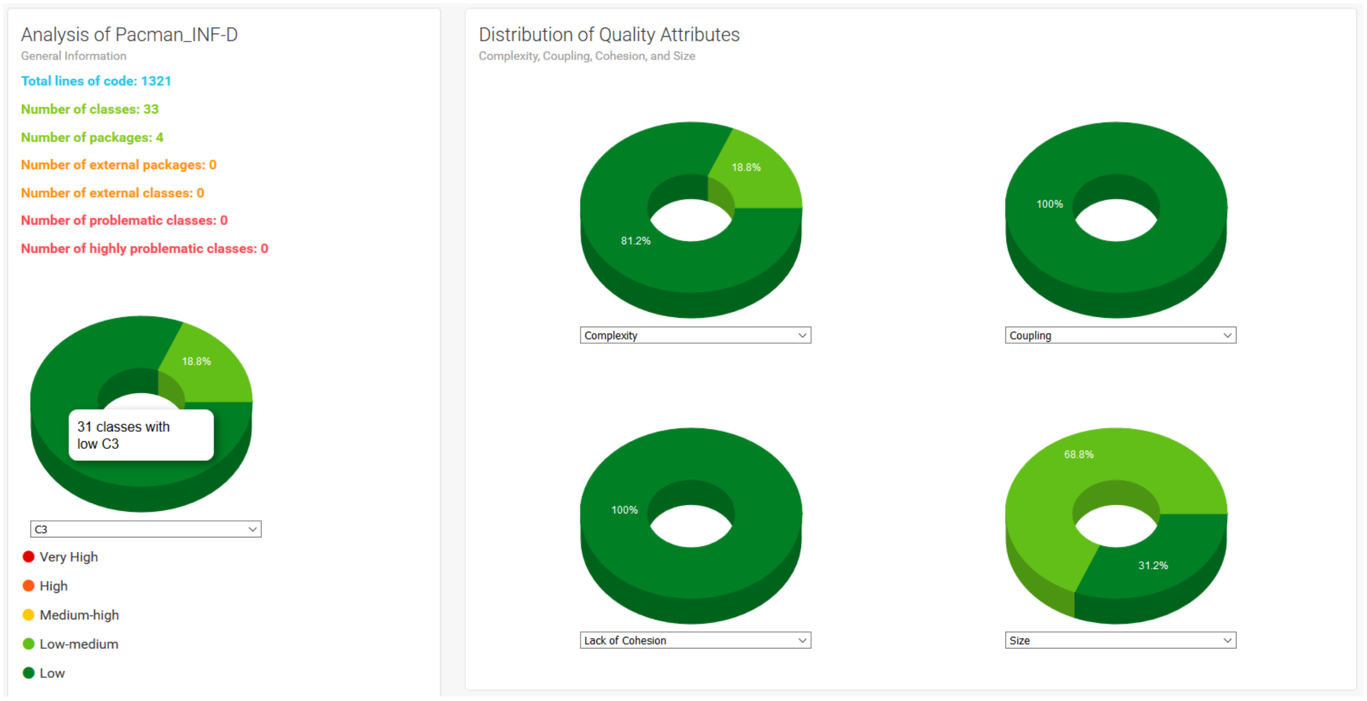

Analysis of Pacman_INF-D
General Information
Total lines of code: 1321
Number of classes: 33
Number of packages: 4
Number of external packages: 0
Number of external classes: 0
Number of problematic classes: 0
Number of highly problematic classes: 0

Figure 1: CodeMR dashboard summarizing health of system 3

[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 can see that every classes is doing good.

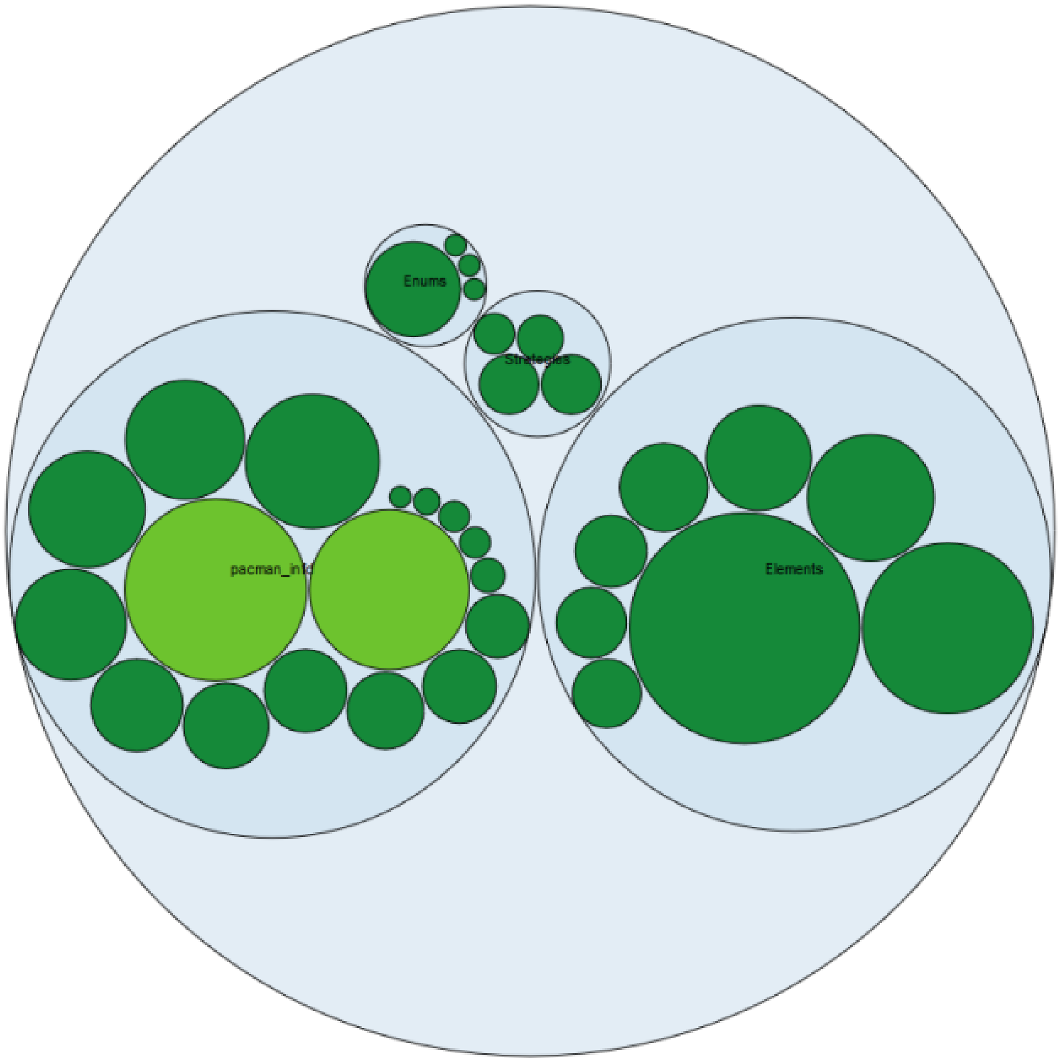
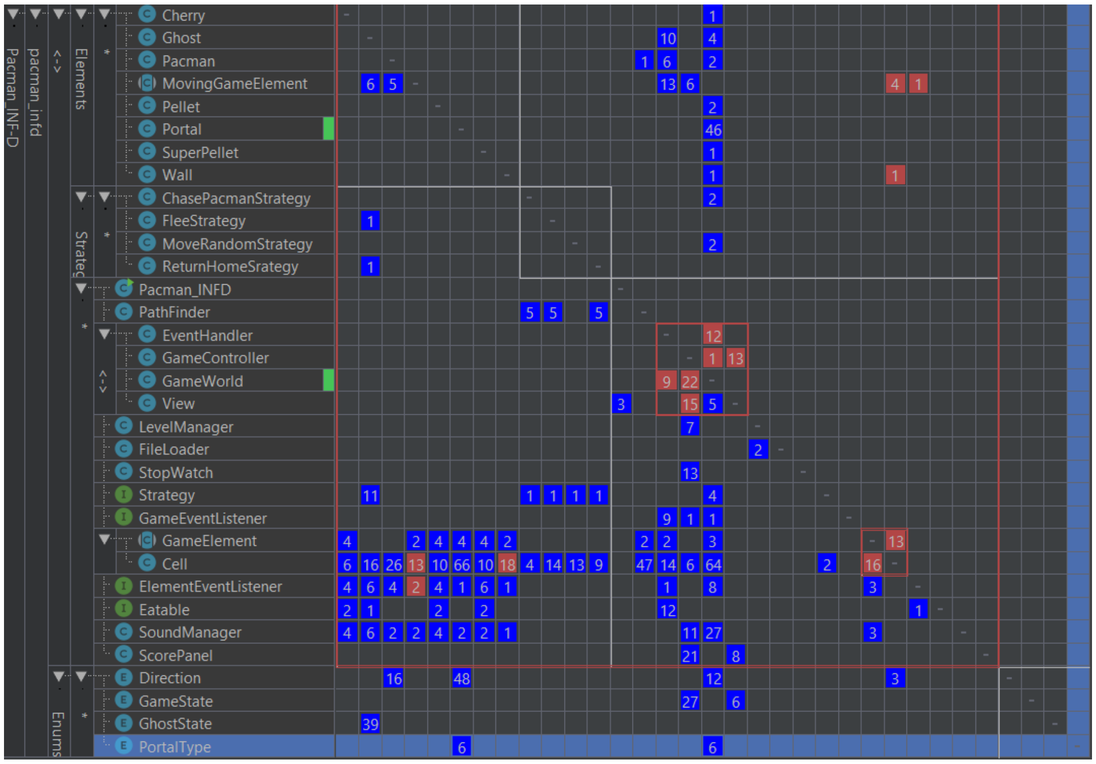


Figure 2: C3 Metric by package for system 3

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.

1.1.2.2 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 3.



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																-

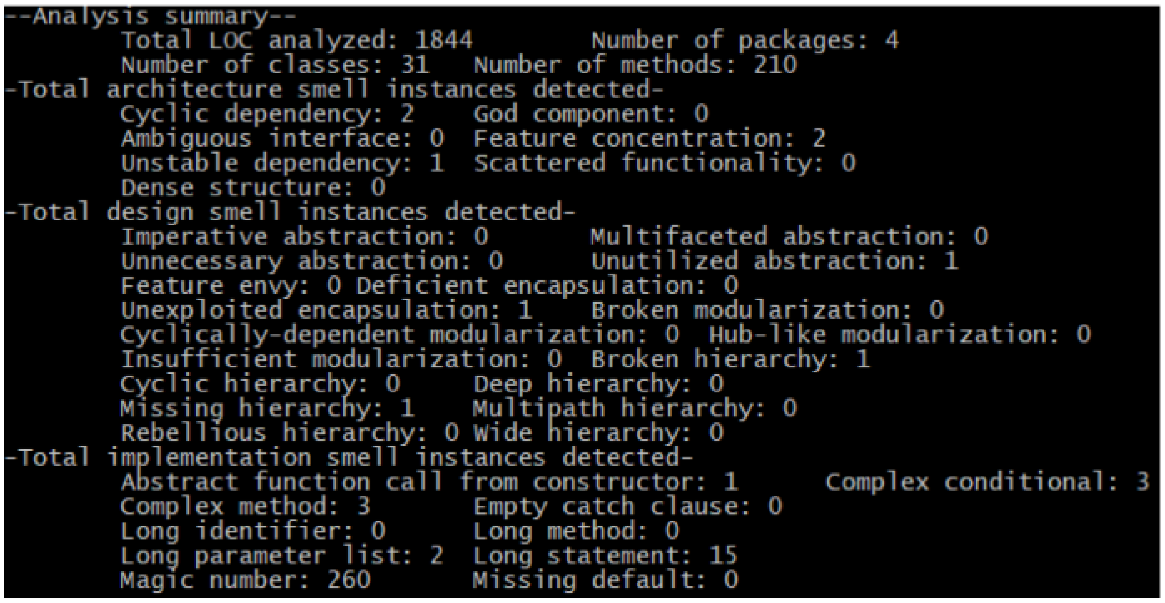
Figure 3: Dependency matrix for system 3

1.1.2.3 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 4.



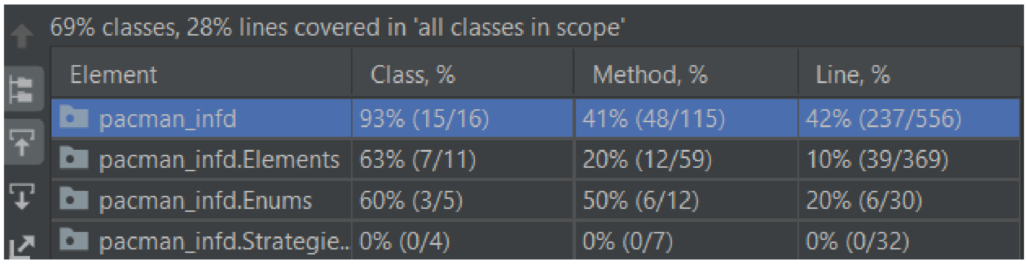
```
--Analysis summary--
        Total LOC analyzed: 1844        Number of packages: 4
        Number of classes: 31   Number of methods: 210
-Total architecture smell instances detected-
        Cyclic dependency: 2    God component: 0
        Ambiguous interface: 0  Feature concentration: 2
        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 4: Designite in-line use results for system 3

1.1.3 Dynamic Analysis

1.1.3.1 Test coverage (Intellij built-in tool)

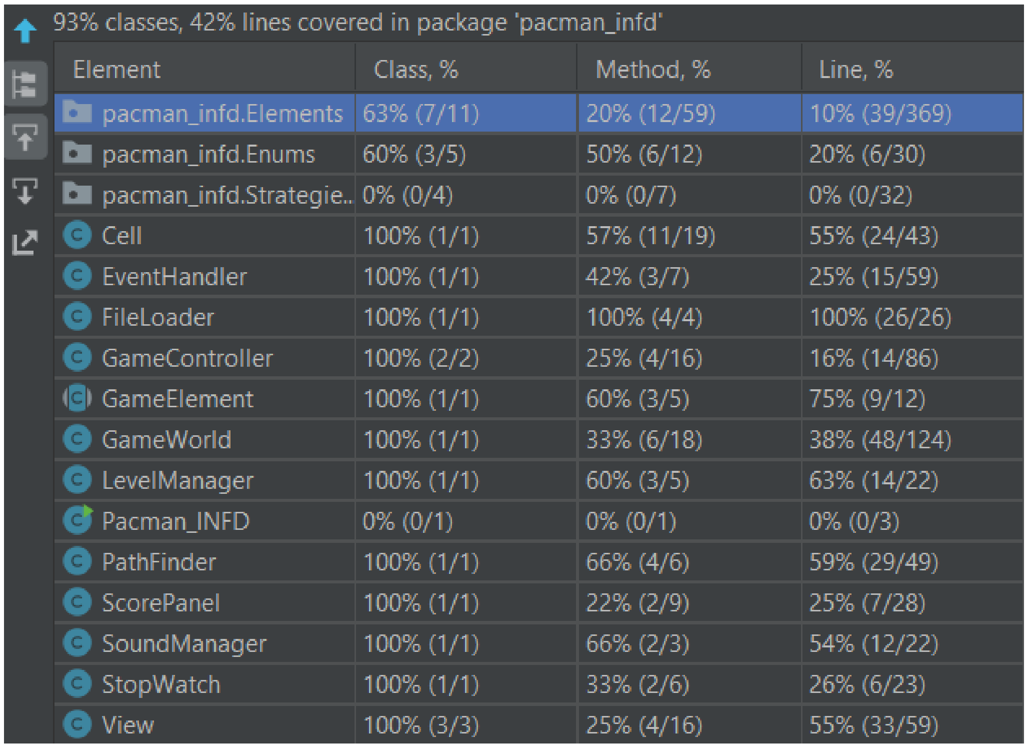
The test coverage of the given tests was analyzed, and the three following figures 5, 6 and 7 give the results. We can see that only 28% of the code is tested and 69% of the classes. We can see that neither of the strategies in the Strategies packages is tested, and in the Elements package only 12 methods 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)

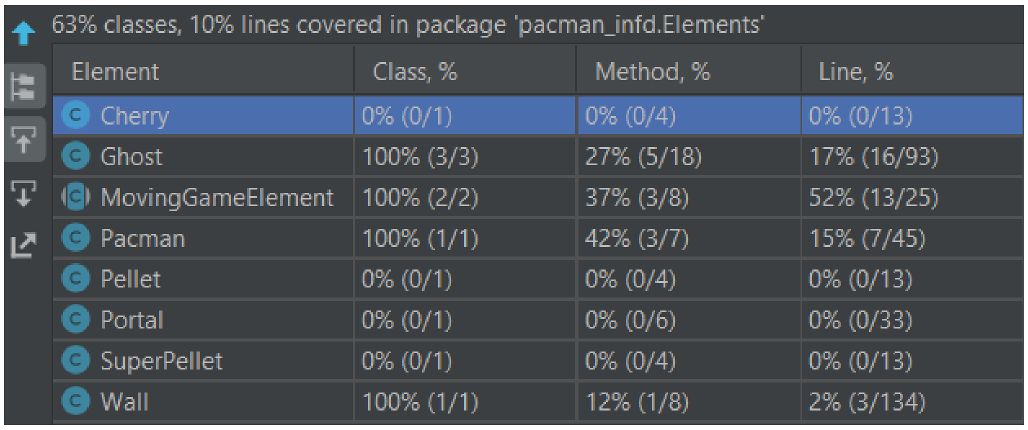
Figure 5: Test coverage for system 3



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 6: 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 7: 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

4.1 System 2 (Robin Schérer)

Working on this system to implemeent new feature wasn't easy because almost all the logic of the collisions was done in the WorkerProcess class, in this class there is a lot of Feature Envy, this class also have a lot of condition that test the Ghost or Pacman state to see what can happen.

This system also doesn't provide a way to change the speed of the DynamicTargets so I had to implement that. The problem is that the run method in WorkerProcess is calling the move function of Pacman on each run, so doing a proper way to handle Target's speed would have needed a lot of refactoring. So I choose to change the REFRESH_RATE of the run method to 0.1s and the Target has a speed between 0 and 10. At a speed of 0 the target never moves, at 10 the target moves on each run call, at 5 the target moves every 5 call, etc.

In this system the map is really small and adding new element on the map isn't an easy task. I choose that every type of fruits is available at the beginning of the game, just so you can try them, also some more fruit will randomly pop in the board if Pacman has enough score.

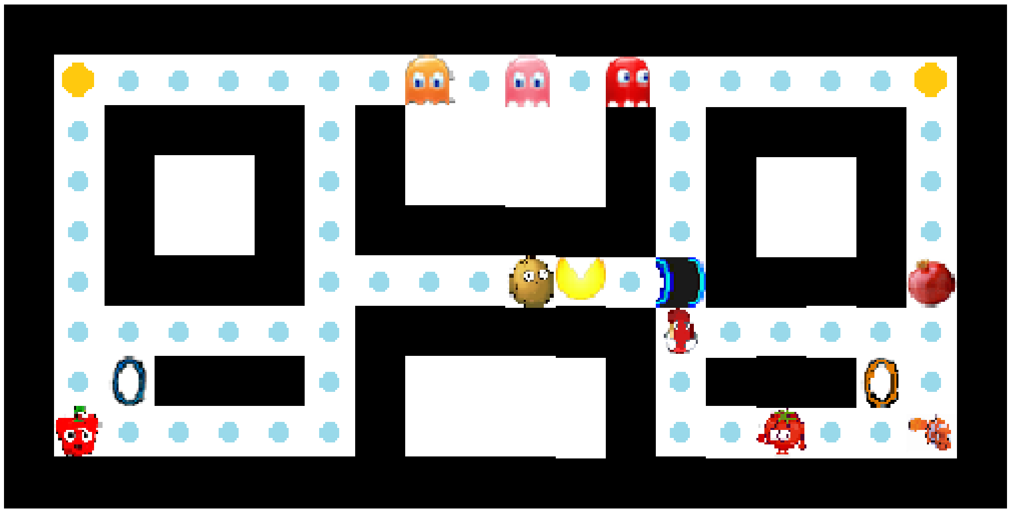


Figure 8: The system 2 map with the new blocs/fruits

The fruits were created using the StaticTarget abstract class as eating a fruit will behave like eating a coin, exept that the eated fruit will trigger a new action. The action will, depending on the fruit, change the way Pacman behave for a certain period of time. Some of the fruits will change Pacman state, like the Bean or the Tomato.

Fish This 'fruit' will make Pacman stops for 3 seconds. It will just change the speed of pacman to 0 for these 3 seconds.

Grenade I chose for this fruit to kill every ghosts in a range of 4 blocs and it doesn't care of walls.

Pepper The pepper will increase Pacman's speed to the maximum when eaten, so it will be set to 10 for 10 seconds.

Potato When eaten, the potato will increase ghost's speed to 7 for 5 seconds.

Tomato This fruit will change Pacman's state to INVINSIBLE for 4 seconds, in this mode Pacman cannot be eaten by the ghosts. So as this systems handle collisions in the WorkerProcess I added a condition that the ghost won't eat pacman if he is in this mode. The figure 9 shows pacman when he eats this fruit.

RedBean This very hot fruit will transform Pacman into Fire mode for 5 seconds, while in this state, he will

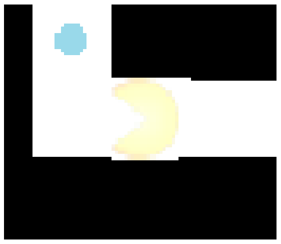


Figure 9: Pacman when in invinsible mode

throw Fireball each time he moves. The fireball kills every Ghost it met and it can cross multiple ghosts. Fireball is implemented with the DynamicTarget abstract class. The figure 10 shows pacman when in fire mode.

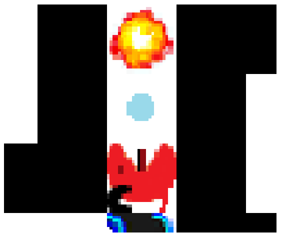


Figure 10: Pacman when in fire mode

The new map objects were implemented using a new abstract class that extends the MapObject abstract class. These objects will trigger some action when a Target is on it.

Trap This box will make the ghost or pacman's speed to 0 for 3 seconds, it can only handle 1 target at a time, so if a ghost is stuck on the trap another ghost can cross it.

Teleporter There is 2 types of teleporter, an entry and a exit. When created a teleporter will be linked to another teleporter or liked to nothing, if the teleporter has a link it will move Pacman to the exit teleporter. On Figure 8 we can see the two types of teleporter, the blue on is the entry teleporter and the orange one is the exit.

Bridge The bridge can normally be used to make pacman cross crossroads, but in this systems there is none on the map, so in this case it will just block Pacman. For this block I created a new kind of state for DynamicTarget, the bridgeState, this state will have 3 main states: NOT_ON, UNDER and ON. So it will be used to knwo if a target is not on a bridge, or under/on a bridge, collision between a ghost and pacman that are not on the same state won't happen. The Figure 11 shows Pacman on a bridge and figure Figure 12 'shows' pacman under a bridge.

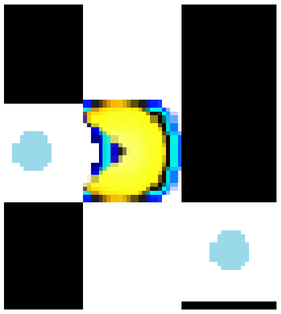


Figure 11: Pacman when on a bridge

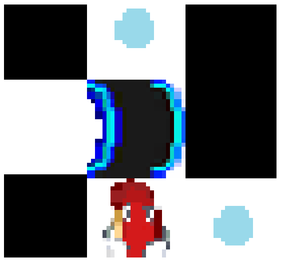


Figure 12: Pacman when under a bridge

5 Quality evolution analysis

5.1 System 2 (Robin Schérer)

5.1.1 Generalities

This system is a mess, Rémy improved it but the system was too bad to begin with, everything could have been changed but it would have taken too much time. The way the collisions are handled is, I think, bad, it's not really clear how we have to create new collision type. Also creating or modify the map is dreadful, it was a pain to change the map, but of course changing that would have taken a lot of time. Creating new type of object is also not that easy because we have to create new Containers, ... And WorkerProcess is, in my mind, a mess in general, it doesn't really respect object oriented programming, everything is compared with else-if with a lot of Feature Envy, everything could have been reworked to make it clearer and simpler to update. But still tanks to Rémy this system is clearer than before.

5.1.2 Static Analysis

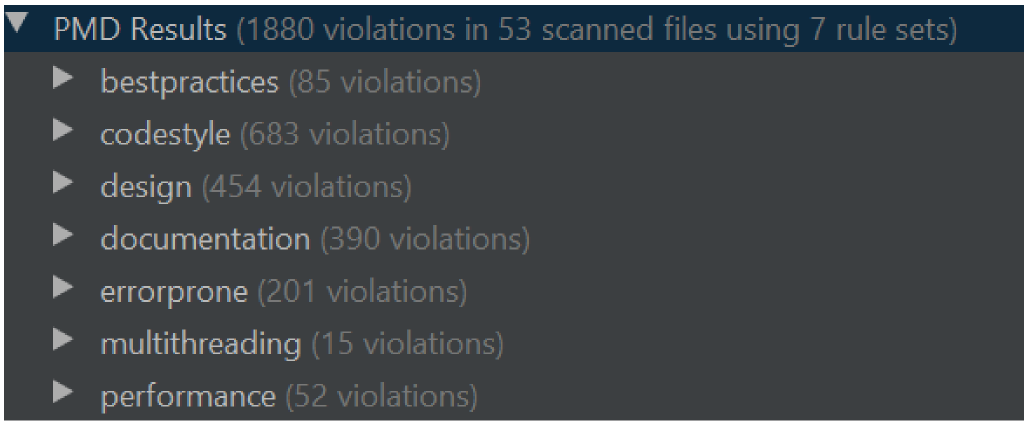
5.1.3 Code Metrics (CodeMR)

CodeMR was used before to anylise the system but now we cannot do it because of the free version, the system is now too big for the free trial. We couldn't afford to buy the full version.

5.1.4 Compliance bad smells (PMD, Designite)

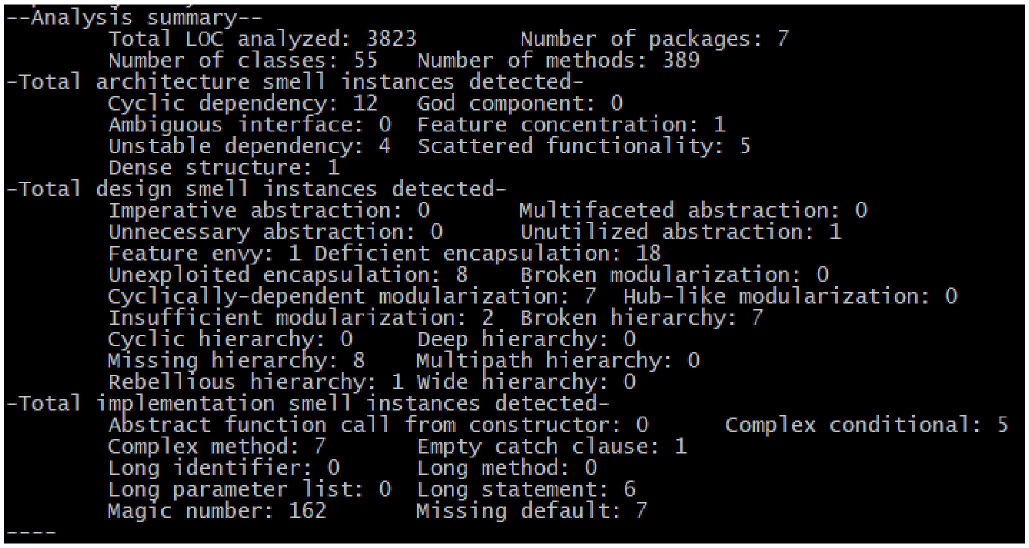
PMD The Figure 13 shows the results of the PMD analysis, we can see that there is 1880 violations.

Designite The Figure 14 shows the results of the Designite analysis, there is still a lot of code smels, like 12 cyclic dependencies, 7 complex method, 1 Feature Envy and 162 magic numbers.



```
▼ PMD Results (1880 violations in 53 scanned files using 7 rule sets)
    ▶ bestpractices (85 violations)
    ▶ codestyle (683 violations)
    ▶ design (454 violations)
    ▶ documentation (390 violations)
    ▶ errorprone (201 violations)
    ▶ multithreading (15 violations)
    ▶ performance (52 violations)
```

Figure 13: PMD results for system 2

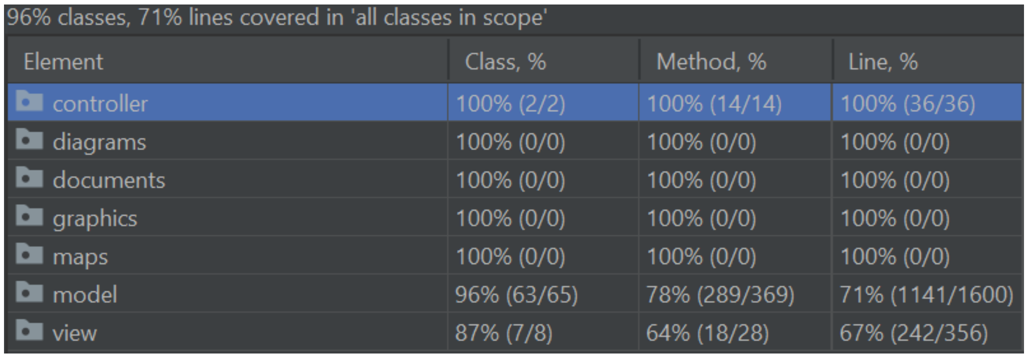


```
--Analysis summary--
        Total LOC analyzed: 3823        Number of packages: 7
        Number of classes: 55   Number of methods: 389
-Total architecture smell instances detected-
        Cyclic dependency: 12   God component: 0
        Ambiguous interface: 0  Feature concentration: 1
        Unstable dependency: 4  Scattered functionality: 5
        Dense structure: 1
-Total design smell instances detected-
        Imperative abstraction: 0       Multifaceted abstraction: 0
        Unnecessary abstraction: 0      Unutilized abstraction: 1
        Feature envy: 1 Deficient encapsulation: 18
        Unexploited encapsulation: 8    Broken modularization: 0
        Cyclically-dependent modularization: 7  Hub-like modularization: 0
        Insufficient modularization: 2  Broken hierarchy: 7
        Cyclic hierarchy: 0     Deep hierarchy: 0
        Missing hierarchy: 8    Multipath hierarchy: 0
        Rebellious hierarchy: 1 Wide hierarchy: 0
-Total implementation smell instances detected-
        Abstract function call from constructor: 0      Complex conditional: 5
        Complex method: 7       Empty catch clause: 1
        Long identifier: 0      Long method: 0
        Long parameter list: 0  Long statement: 6
        Magic number: 162       Missing default: 7
----
```

Figure 14: Designite result for system 2

5.1.5 Test coverage (Intellij built-in tool

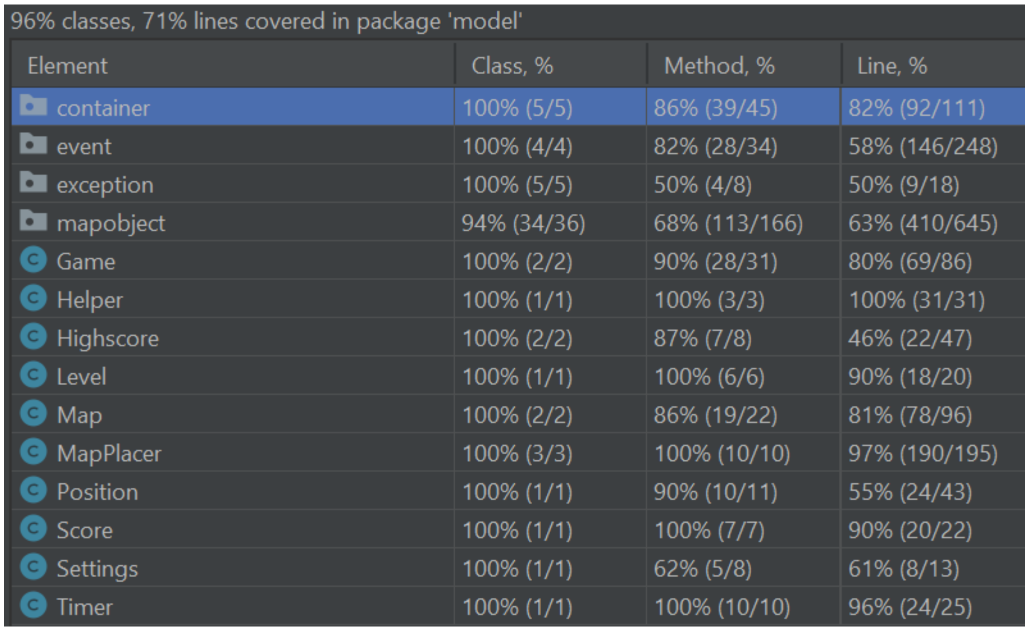
The following figures 15, and 16 shows the test coverage. All the tests passed and almost all the code is covered. We can see that 96% of the classes were tested and 71% of the code line.



96% classes, 71% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
controller	100% (2/2)	100% (14/14)	100% (36/36)
diagrams	100% (0/0)	100% (0/0)	100% (0/0)
documents	100% (0/0)	100% (0/0)	100% (0/0)
graphics	100% (0/0)	100% (0/0)	100% (0/0)
maps	100% (0/0)	100% (0/0)	100% (0/0)
model	96% (63/65)	78% (289/369)	71% (1141/1600)
view	87% (7/8)	64% (18/28)	67% (242/356)

Figure 15: System 2 test coverage



96% classes, 71% lines covered in package 'model'

Element	Class, %	Method, %	Line, %
container	100% (5/5)	86% (39/45)	82% (92/111)
event	100% (4/4)	82% (28/34)	58% (146/248)
exception	100% (5/5)	50% (4/8)	50% (9/18)
mapobject	94% (34/36)	68% (113/166)	63% (410/645)
Game	100% (2/2)	90% (28/31)	80% (69/86)
Helper	100% (1/1)	100% (3/3)	100% (31/31)
Highscore	100% (2/2)	87% (7/8)	46% (22/47)
Level	100% (1/1)	100% (6/6)	90% (18/20)
Map	100% (2/2)	86% (19/22)	81% (78/96)
MapPlacer	100% (3/3)	100% (10/10)	97% (190/195)
Position	100% (1/1)	90% (10/11)	55% (24/43)
Score	100% (1/1)	100% (7/7)	90% (20/22)
Settings	100% (1/1)	62% (5/8)	61% (8/13)
Timer	100% (1/1)	100% (10/10)	96% (24/25)

Figure 16: System 2 test coverage