

Gdx Game

Quality Management Document

Giamakidis – Kiosses Christos ics21018

Thomasiadis Konstantinos ics21058

Kanidou Elisavet – Persephone ics21095

Kontaxis Ioannis ics21105

Kotoula Aristeas ics21167

Tsavalias Vasilios – Ephraim ics21083

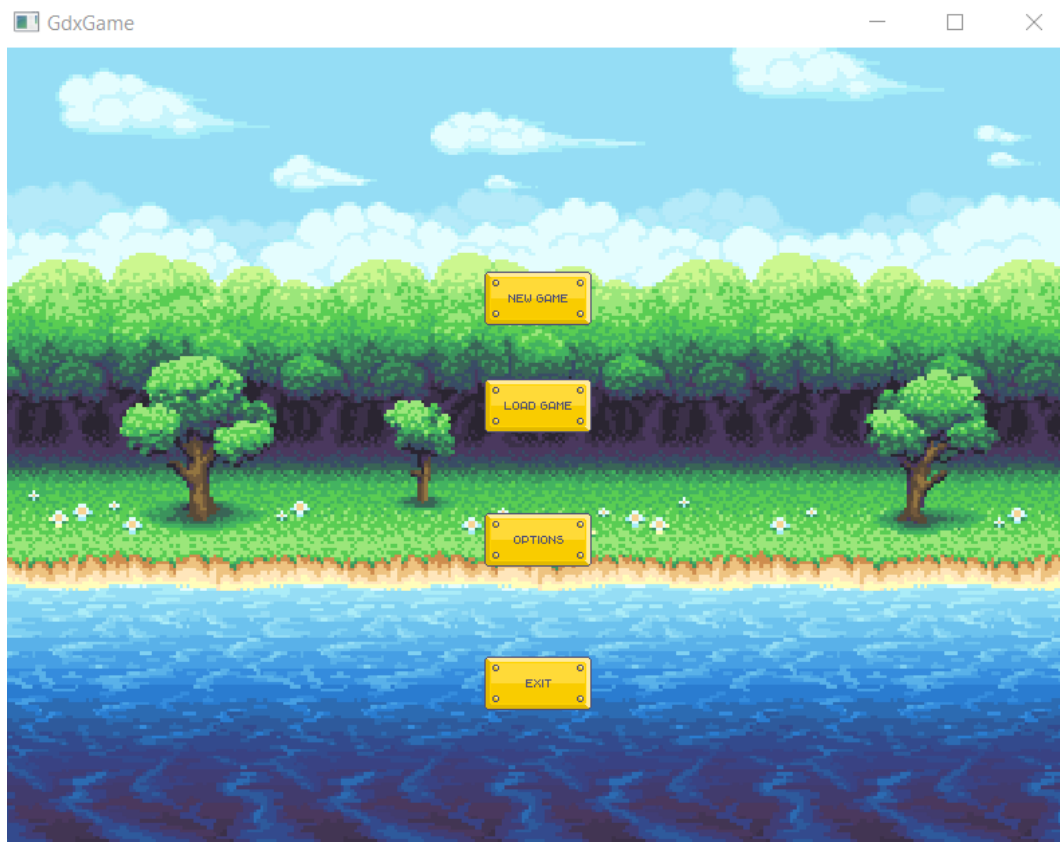
[YouTube link](#)

Contents

Contents.....	2
1 Introduction.....	3
2. Measurement of Technical Debt.....	5
2.1 Measurement of Capital (Principal).....	6
2.1.1 Principal by Code Smell Category.....	6
2.1.2 Chapter by TypeCode Smell.....	7
2.2 Interest Measurement.....	7
2.2.1 Finding Distance from Optimum.....	9
2.2.2 Calculation of Interest per Line of Code.....	11
2.3 Calculation of Interest over time.....	12
2.4 Interest on Working Time.....	13
2.5 Breaking Point.....	13
2.6 Interest Probability Measurement.....	14
3. Software Quality Monitoring.....	15
3.1 Evolution of Technical Debt.....	15
3.1.1.1 Charts by Metric.....	16
3.1.1.2 Lines of Code (LOC).....	16
3.1.1.3 Number of Children (NC).....	17
3.1.1.4 WMC (Weighted Method per Class).....	17
3.1.1.5 Lack of Cohesion (LCOM).....	18
3.1.1.6 Depth of Inheritance (DIT).....	18
3.1.1.7 NOM (Number of Methods).....	19
4. Identifying and Resolving Quality Issues.....	23
4.1 Code Analysis and UML Diagrams:.....	24
(BATTLEHUD) – Vasilis Tsavalias.....	24
PlayerHUD (Christos Giamakidis-Kiosses):.....	36
PlayerPhysicsComponent (THOMASSIADIS KONSTANTINOS):.....	40
StatusUI (Aristea Kotoula) :.....	48
QuestGraph (Kontaxis Ioannis) :.....	58
Inventory UI (Elisabet Persephone Kanidou ics21095) :.....	62

1 Introduction

"GdxGame" is an RPG (Role Playing Game) game, in which the user is asked to protect the inhabitants of a virtual village from various monstrous creatures, while at the same time he can complete a series of side missions by talking to the aforementioned characters. More specifically, as can be seen from the images below, the user initially selects the game settings, i.e. the keys he wishes to use and the volume of the music. He then chooses between uploading an existing game and creating a new one with the same name, as well as the hero he wants to play in the game. Then the map with all the entities appears, in the form of two-dimensional graphics (2D Graphics) and the user starts browsing the environment.



Picture 1

In addition, there are functions such as the collection of the items that the hero has in his possession and a list of the current missions that he has to perform. In fact, every time the user completes an event in the game, his character receives in return experience points (Experience Points - XP) which raise the level of the character (Level) resulting in him gaining additional reinforcements for the battlefield (Defense Points / Defense/Attack Points).

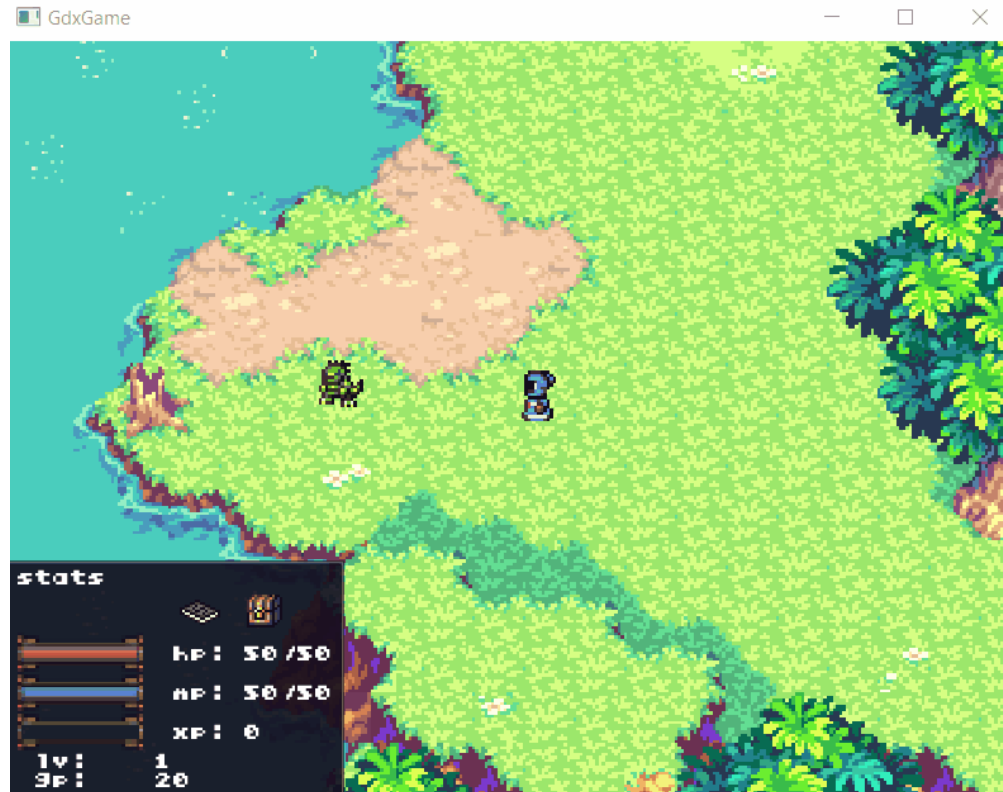


Figure 2

2. Measurement of Technical Debt

This chapter measures the technical debt of GdxGame. A detailed description of the methodology used to calculate the technical debt, the metrics, as well as the individual measurements of principal, interest and breaking point will follow.

Assessment Tool

To calculate the metrics and technical debt, the tool used is DesigniteJava, which is a software code quality evaluation tool, specifically for Java code. With this program, the code is analyzed and smells related to the architecture, design and implementation are detected. In addition, metrics are calculated, with the result that this tool contributes to the reduction of technical debt, but also to the improvement of software maintenance.

Through the DesigniteJava program, in the latest version of the GdxGame 2.5.3 software, a variety of code smells (Implementation Code Smells) were detected and they concern the following categories:

- 1.**Complex Condition:**Complex selection structure conditions.
- 2.**Complex Method:**Complex methods (usually violate the uniqueness principle jurisdiction).
- 3.**Long Method:**A method that involves many lines of code.
- 4.**Long Parameter Lists:** Large number of parameters to a function. A large list of parameters can be generated after merging several types of algorithms into a single method.
- 5.**Long statement:**Very long statements (eg prints)
- 6.**Magic Number:**Using numbers directly in code.
- 7.**Missing Default:**Switch statement that does not contain the default state
- 8.**Long Identifier:**Variable or method name too long.
- 9.**Empty Catch:**Empty code block in try/catch

2.1 Measurement of Capital (Principal)

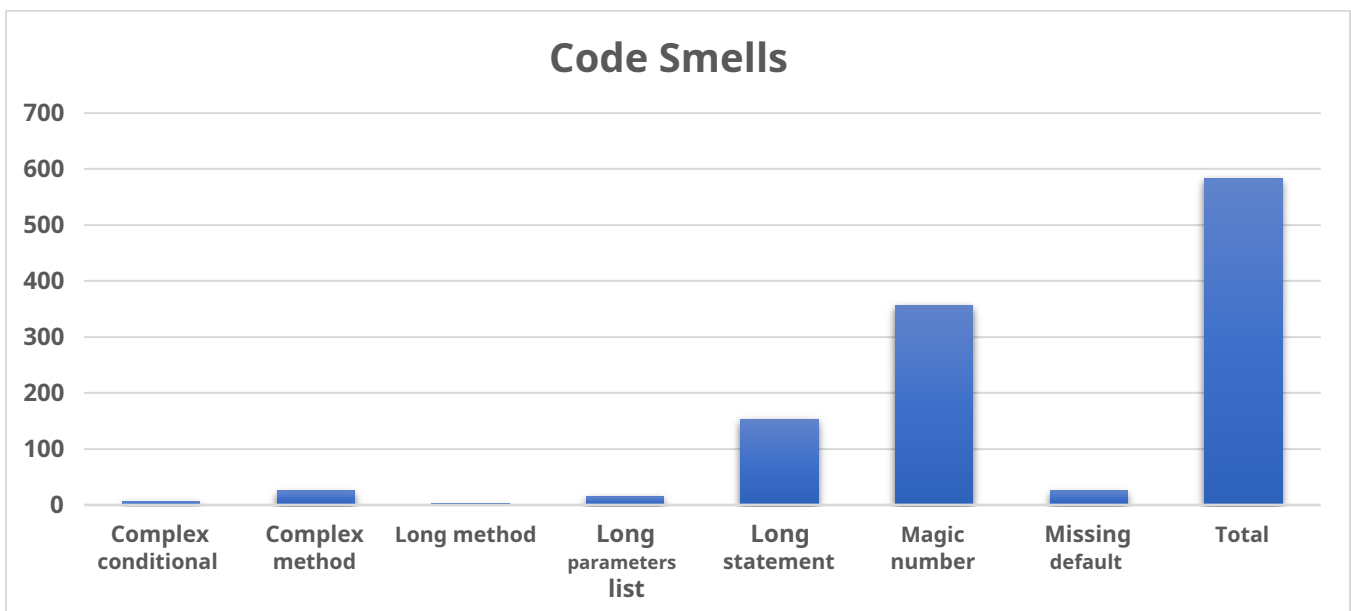
The capital is measured using the following formula:

$$TDPrincipal = \sum \#instances * minutesforSolution$$

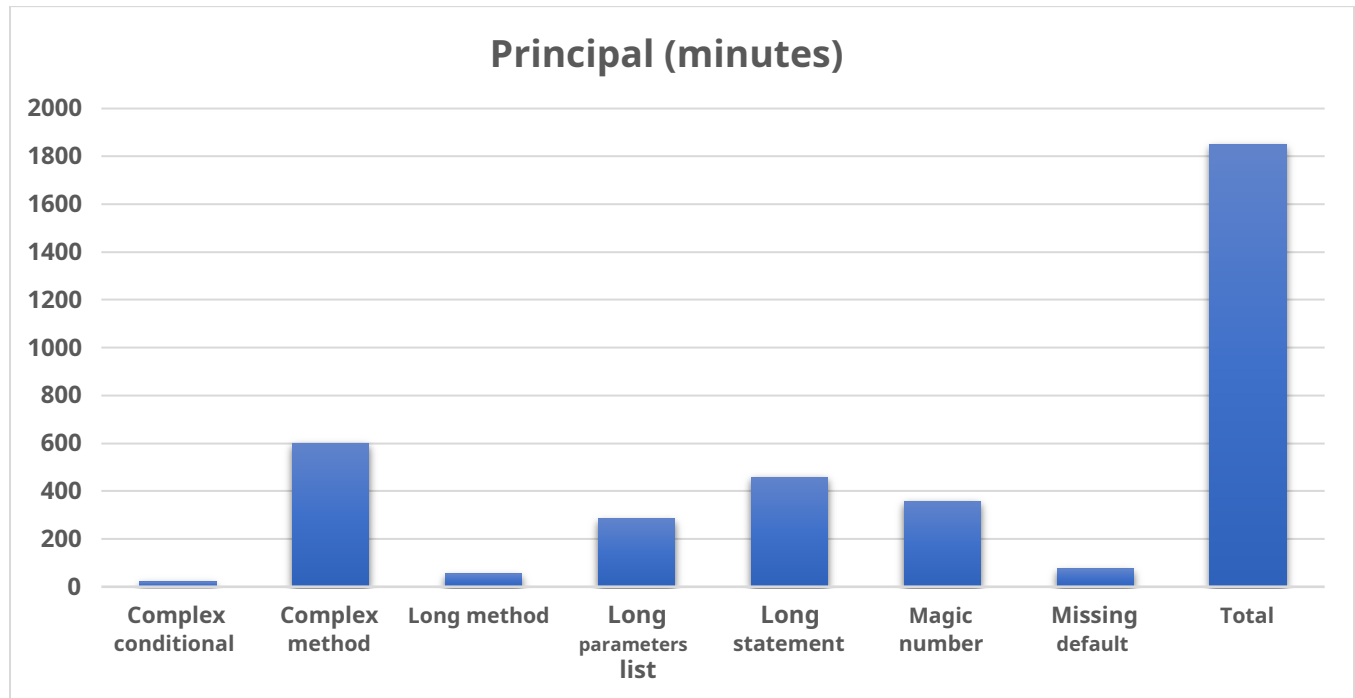
Therefore, the following table is obtained:

Types of Smells	Number of Smells	Average Remediation Time (minutes)	Principal (minutes)
Complex conditional	7	3	21
Complex method	25	24	600
Long method	2	27	54
Long parameter list	15	19	285
Long statement	153	3	459
Magic number	357	1	357
Missing default	25	3	75
Total	584	80	1851

2.1.1 Principal by Code Smell Category



2.1.2 Chapter by TypeCode Smell



2.2 Calculation of Interest (interest)

In general, the calculation of interest for a specific period of time (e.g. one month, one year, etc.) is done by the following formula:

$$\text{Interest} = \text{Distance from optimum maintainability} * \text{Period}(x) * \text{Load of MaintenanceHistory}$$

In the following table we see the size of some indicative metrics:

Name	DIT	NOCC	RFC	LCOM	WMC*	NOM	MPC	DAC
PlayerHUD	0	0	199	325	3.19231	26	173	10
InventoryUI	0	0	81	276	2.58333	24	57	1
BattleHUD	0	0	109	210	2.57143	21	88	12
StoreInventoryUI	0	0	25	105	1.6	15	10	1
QuestGraph	0	0	80	351	2.53571	27	53	0
GameScreen	0	0	68	66	2.5	12	56	4

In the following, the classes with the greatest similarity to the user, which in our case is PlayerHUD, are presented, where their similarity to it ranges from 61% to 85%.

Name	SIZE1	SIZE2	SIMILARITY_1	SIMILARITY_2	SIMILARITY
PlayerHUD	581	44	100%	100%	100%
InventoryUI	417	43	72%	98%	85%
BattleHUD	341	50	59%	86%	73%
StoreInventoryUI	221	41	38%	93%	66%
QuestGraph	267	35	46%	80%	63%
GameScreen	210	38	36%	86%	61%

	InventoryUI	StoreInventoryUI	QuestGraph	GameScreen	PlayerHUD
DIT	0	0	0	0	0
NOCC	0	0	0	0	0
RFC	81	25	80	68	199
LCOM	276	105	351	66	325
WMC	2.583333254	1.600000024	2.535714388	2.5	3.19230771
NOM	24	15	27	12	26
MPC	57	10	53	56	173
DAC	1	1	0	4	10
SIZE1	417	221	267	210	581
SIZE2	43	41	35	38	44

optimal class	diff	interest per LOC
0.000	0.000	
0.000	0.000	
25,000	174,000	696.00%
66,000	259,000	392.42%
1,600	1,592	99.52%
12,000	14,000	116.67%
10,000	163,000	1630.00%
0.000	10,000	
210,000	371,000	176.67%
35,000	9,000	25.71%

 Input
 Output

448%

k = 26.91666667 lines added per version (on average)

1.076666667	hours (assuming that the developer writes 25 LoC per hour)
\$49.32	dollars (supposing that the developer is paid ~45 dollars per hour)
\$221.03	interest in dollars (by considering that the developer needs ~24% extra time due to quality distance)

SIZE1 VERSION N	65	10
SIZE1 VERSION N-1	55	7
SIZE1 VERSION N-2	48	3
SIZE1 VERSION N-3	45	7
SIZE1 VERSION N-4	38	

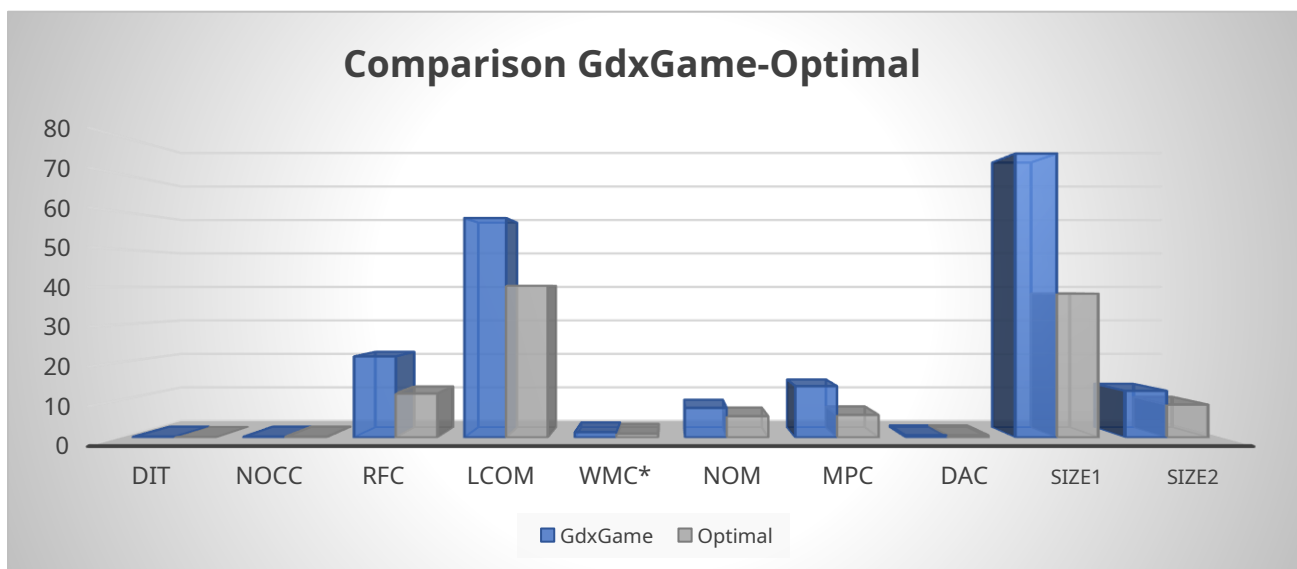
2.2.1 Finding Distance from Optimum

First, 3 other open source games (Rpg-Game type) were found which have a similar number of classes as GdxGame. GdxGame and the 3 additional games, which were used to find the values per metric, are listed by name in the table below along with the number of classes and their line difference:

The results of the measurements are presented in the table below, with the minimum (optimal) values:

Game	No. of Classes	Difference
GdxGame	126	0
minecraft-plus-revived	184	58
OasisGame	61	- 65
Unlucky	72	- 54

Metrics	GdxGame	minecraft	OasisGame	Unlucky
DIT	0.24	1.36	0.09	0.11
NOCC	0.22	0.72	0.25	0.38
RFC	21.54	32.20	11.69	17.07
LCOM	58,29	43.62	40,31	57.47
WMC*	1.43	2.22	1.02	2.35
NOM	7.88	6.47	5.67	6.65
MPC	13.66	25.74	6.01	10.41
DAC	0.592	0.95	0.67	1.8
SIZE1	75.40	93.60	38.2	120.84
SIZE2	12.42	10.82	8.75	18.65



2.2.2 Calculation of Interest per Line of Code

Interest per line of code equals:

Average((GdxGameMetric – OptimalMetric) / OptimalMetric) and is calculated in the table below

Metrics	GdxGame	Optimal	Difference from optimal	Interest per LoC
DIT	0.24	0.09	0.15	1.66
NOCC	0.22	0.22	0	0
RFC	21.54	11.69	9.85	0.84
LCOM	58,29	40,31	17.98	0.45
WMC*	1.43	1.02	0.41	0.40
NOM	7.88	5.67	2.21	0.39
MPC	13.66	6.01	7.65	1.27
DAC	0.592	0.592	0	0
SIZE1	75.4	38.2	37.2	0.97
SIZE2	12.42	8.75	3.67	0.42

M.O. of Interest per Loc is : 0.64. So interest per line is 0.64 lines.

2.3 Calculation of Interest over time

For the calculation of the interest in terms of time, a necessary feature is the code lines of the software, therefore the table below contains the code lines and also the date for each version of the software. In the last line the averages are calculated, specifically the second column contains the average of the difference of code lines from version to version, while the third column contains the average of the difference of days from version to version.

Version	Lines of Code	Date
1	2230	6/1/2021
2	7242	21/7/2021
2.1	7787	8/16/2021
2.1.1	7816	8/19/2021
2.1.2	7823	21/8/2021
2.2	7951	27/9/2021
2.2.1	7970	10/10/2021
2.2.2	8404	20/1/2022
2.3	8404	20/1/2022
2.4	8741	25/1/2022
2.5	8973	20/2/2022
2.5.1	9072	28/2/2022
2.5.2	9120	16/3/2022
2.5.3	9426	22/8/2022
MO	553.6923077	36.23076923

Dividing the above averages (553.69/36.23) we arrive at 15.28 lines of code per day.

Therefore $\text{interestperDay} = \text{InterestperLoC} * \text{LoCperDay} = 4.48 * 15.28 = 68.45$ (2)

2.4 Interest on Working Time

In his book "Software Estimation: Demystifying the Black Art (Developer Best Practices)", according to Steve McConnell the average developer writes 20-125 lines of code per day on small projects (a total of 10,000 project lines), while on large projects 1.5-25 lines (total 10,000,000 project lines), so 72.5 and 13.25 lines of code on average respectively.

So it is easy to see that the larger the program the fewer lines of code the programmer writes per day on average.

Based on the above examples, when the number of lines of code is multiplied by 1,000 the speed of the programmer drops to 13.25/ 72.5 , or about 81.72%.

The GdxGame software consists of approximately 10,000 lines of code.

Therefore, assuming that the reduction is linear, it follows that the efficiency is reduced by:

$$\frac{81.72\%}{1000} \text{ to } \text{LoCSpeedLoss} \Rightarrow \text{LoCSpeedLoss} = 0.08\%$$

A programmer at 99.92% (100-0.08) would therefore be expected to have the ability to write 72.44(72.5*99.92/100) (3) lines of code per day in that program. Accordingly, under (2) and (3) a developer may repay the interest on the code of one

day in :InterestinDays = InterestinAverageLOC = 68.45 = 0.9 days
 $\frac{\text{InterestinAverageLOC}}{\text{LOCWrittenPerDay}} = \frac{68.45}{72.44}$

Payback will cost: InterestinMinutes = 0.9*480 = 432 minutes of work, assuming an average developer works 8 hours = 480 minutes per day.

2.5 Breaking Point

The calculation of days after which the GdxGame software will reach the point

breaking, is easily done with the following formula: DaysUntilBreakingPoint = $\frac{\text{Principal(min)}}{\text{InterestinMinutes(min)}}$

Therefore we have:

DaysUntilBreakingPoint = $\frac{1851}{432} = 4.28$

If it is considered that an average programmer works 8 hours a day or 480 minutes.

2.6 Interest Probability Measurement

$$\text{REM} = \frac{+ +}{+} = \frac{127}{270} = 0.47$$

NDMC: Number of different method calls from the dependent class to the source class

(plus parent class method calls for the generalization case). Number of polymorphic methods in the source class

NOP (valid for generalization only): Number of protected attributes in the source class

NPrA (only valid for generalization or friend classes): Number of methods in the source class

NOM: Number of methods in the source class

NA: Number of attributes in the source class

Classes	NOP	NPrA	NDMC	NOM	NA
BattleHUD	0	0	13	20	29
InventoryUI	0	0	25	27	8
PlayerHUD	0	0	24	43	18
QuestGraph	0	0	22	27	16
StatusUI	0	0	38	40	26
PlayerPhysicsComponent	0	1	5	7	6
Total	0	1	127	164	106

3. Software Quality Monitoring

In this chapter, considering the metric values for each version of the software under consideration (**GdxGame**), we will examine the evolution of technical debt. Finding the matches metrics, grouping in a table the averages of their values (Table 1), we make an explanatory presentation using diagrams.

3.1 Evolution of Technical Debt

In order to analyze the technical debt of the software, the values of the 10 metrics should be extracted, giving us an image of the maintainability of the code per version. The table below details the metrics for each version of the software:

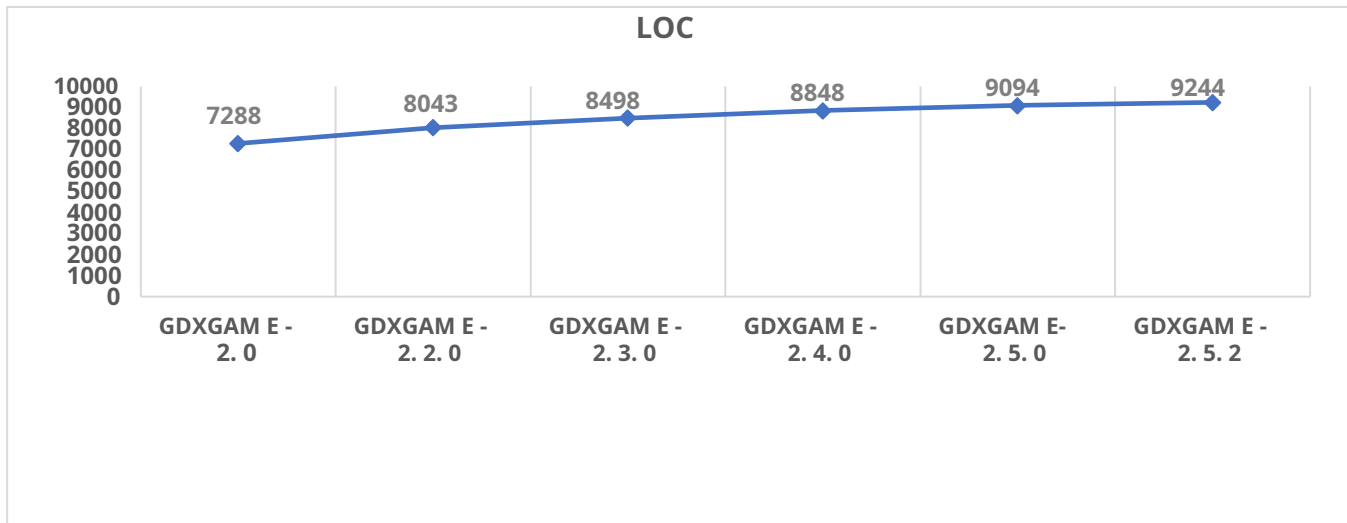
Version	LOC	NC	WMC	LCOM	DIT	NOM	NOM+NOF	CBO
GdxGame-2.0	7288	0.22	14.29	54.08	0.19	8.44	12.40	2.68
GdxGame-2.2.0	8043	0.23	14,18	51.83	0.20	8.35	12.65	2.71
GdxGame-2.3.0	8498	0.23	14.46	55.97	0.20	8.61	13,16	2.76
GdxGame-2.4.0	8848	0.23	15.05	58,03	0.20	8.78	13.38	2.87
GdxGame-2.5.0	9094	0.225	15,24	58,53	0.20	8.92	9.30	2.90
GdxGame-2.5.3	9244	0.22	15,098	58.85	0.19	8.85	13.47	2.87

Table 1: Metric Values by System Version

3.1.1.1 Charts by Metric

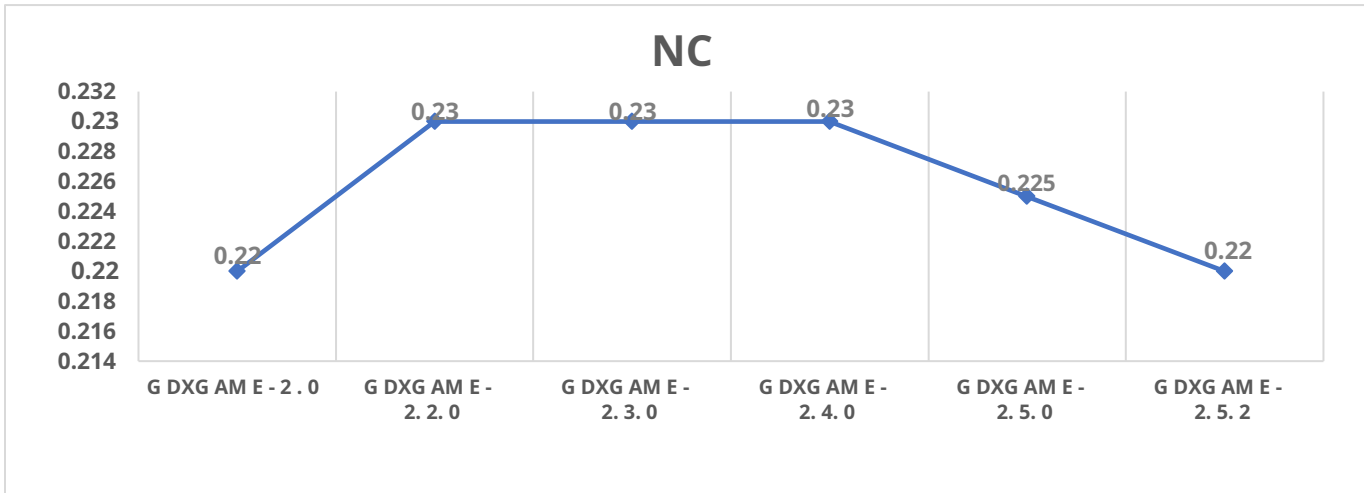
In the following subsection, we will perform a presentation of the graphs of the metrics, followed by an explanation of their value changes. The diagrams and their annotation were drawn up based on the data in Table 1.

3.1.1.2 Lines of Code (LOC)



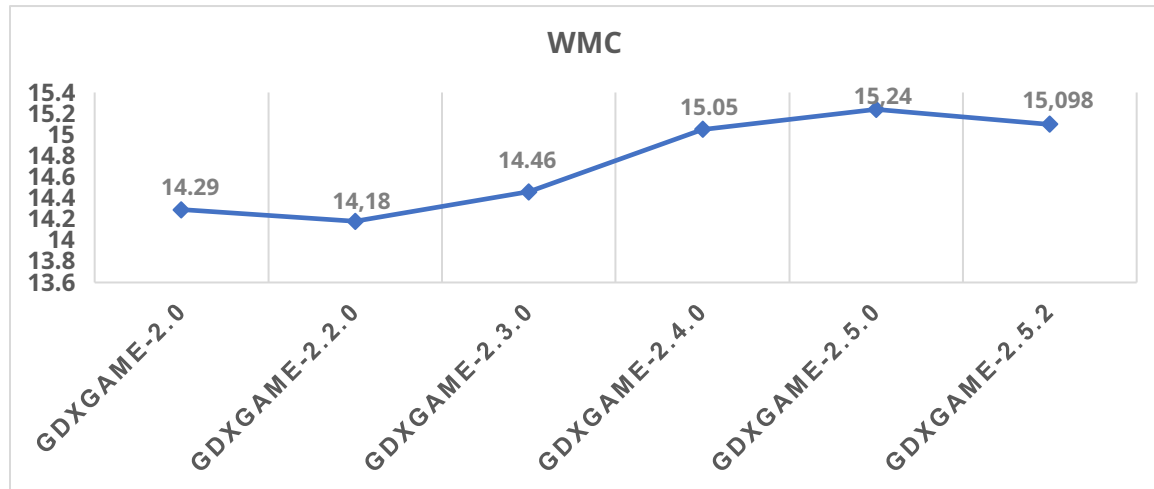
As we know from theory the LOC (Lines of Code) metric counts the lines of code for each class. In this case, the sum of the lines of code of each class per software version was found. One notices in the chart above that there is a general increase (sometimes more sometimes less) in lines of code, with the largest (755 lines of code) occurring from GdxGame-2.0 (7288 lines) to GdxGame-2.2.0 (8043).

3.1.1.3 Number of Children (NC)



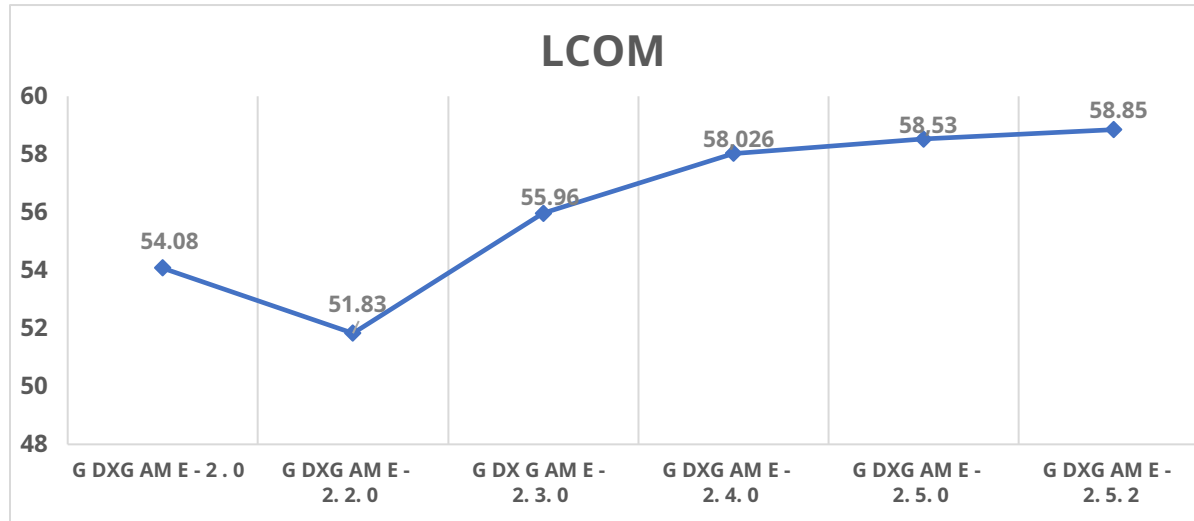
The diagram above shows the evolution of the NC (Number of Children) metric with the end of the software versions. There is a sharp increase from GdxGame-2.0 to GdxGame-2.2.0 while a steep decrease from GdxGame-2.0 to GdxGame-2.5.2. However, a point worth mentioning is the stability observed at the value of 0.23 at the end of GdxGame-2.2.0, GdxGame-2.3.0 and GdxGame-2.5.0.

3.1.1.4 WMC (Weighted Method per Class)



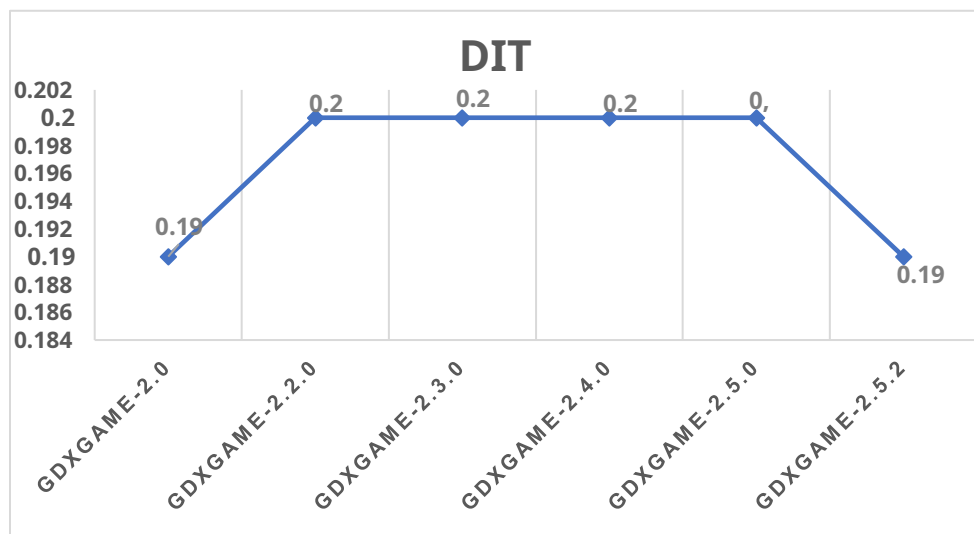
As you can see in the graph above, the WMC (Weighted Method per Class) metric seems to be mainly following a continuous upward trend until the version of GdxGame-2.5.0 with the only decreases being observed in the version of GdxGame-2.2.0 and GdxGame-2.5.2 of the software.

3.1.1.5 Lack of Cohesion (LCOM)



The graph above shows the LCOM (Lack of Cohesion) metric, which is characterized by two nodal fluctuations. The first significant change is seen from GdxGame-2.0 to GdxGame-2.2.0 with the price drop (54.08 -> 51.83). While the second change is seen with the price increase from GdxGame-2.2.0 to GdxGame-2.3.0.

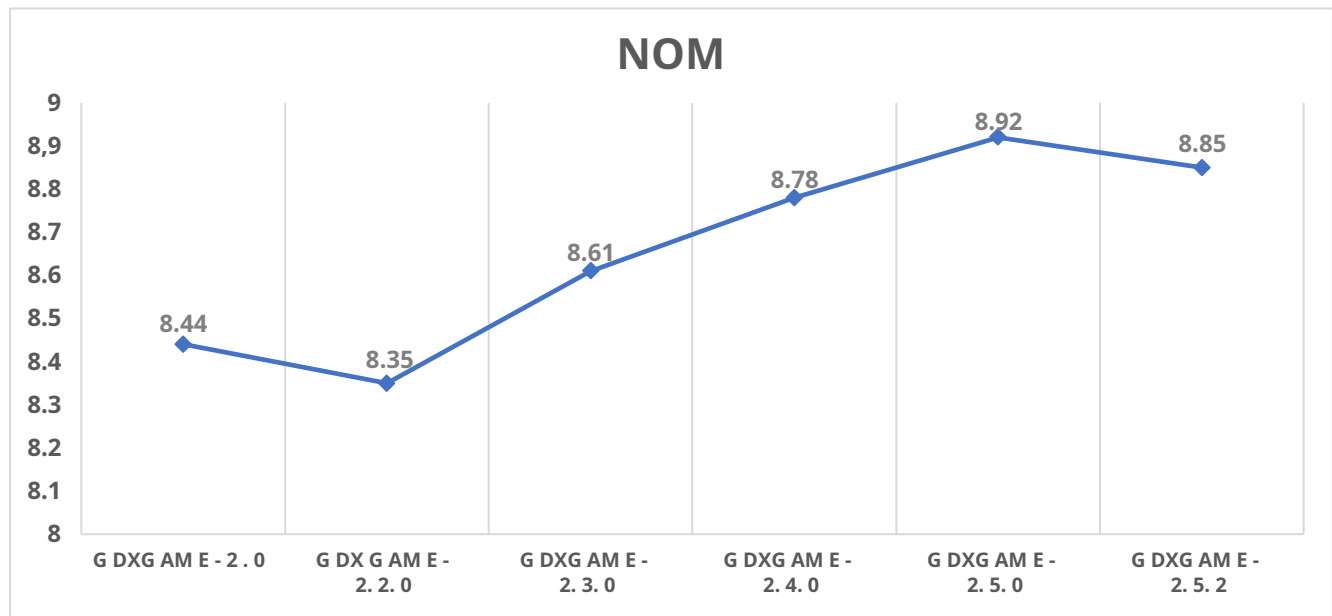
3.1.1.6 Depth of Inheritance (DIT)



Above the DIT (Depth of Inheritance) metric is mentioned, we observe three kinds of changes. At first there is an upward trend since the release GdxGame-2.0 to GdxGame-2.2.0 (0.19 -> 0.2) while a steep drop from GdxGame-2.5.0 to GdxGame-2.5.2 (0.2 -> 0.19). However, the most important change

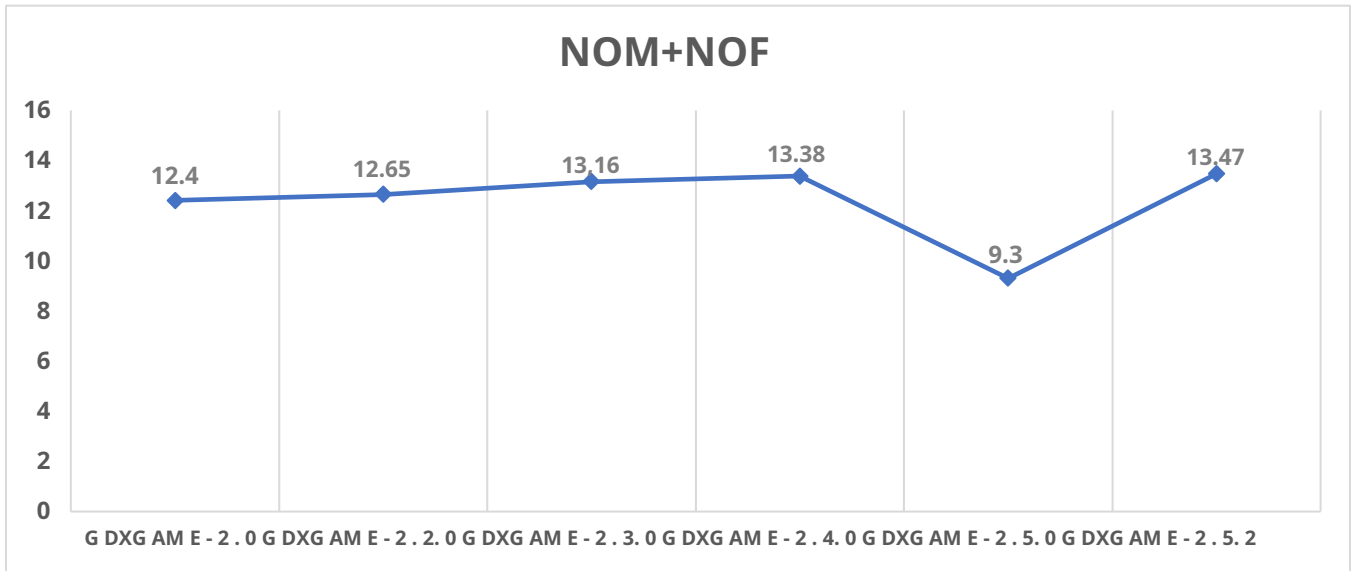
observed with the end of versions GdxGame-2.2.0 , GdxGame-2.3.0 , GdxGame-2.4.0 , GdxGame-2.5.0 where we distinguish a stability of the aforementioned versions at the value 0.2.

3.1.1.7 NOM (Number of Methods)



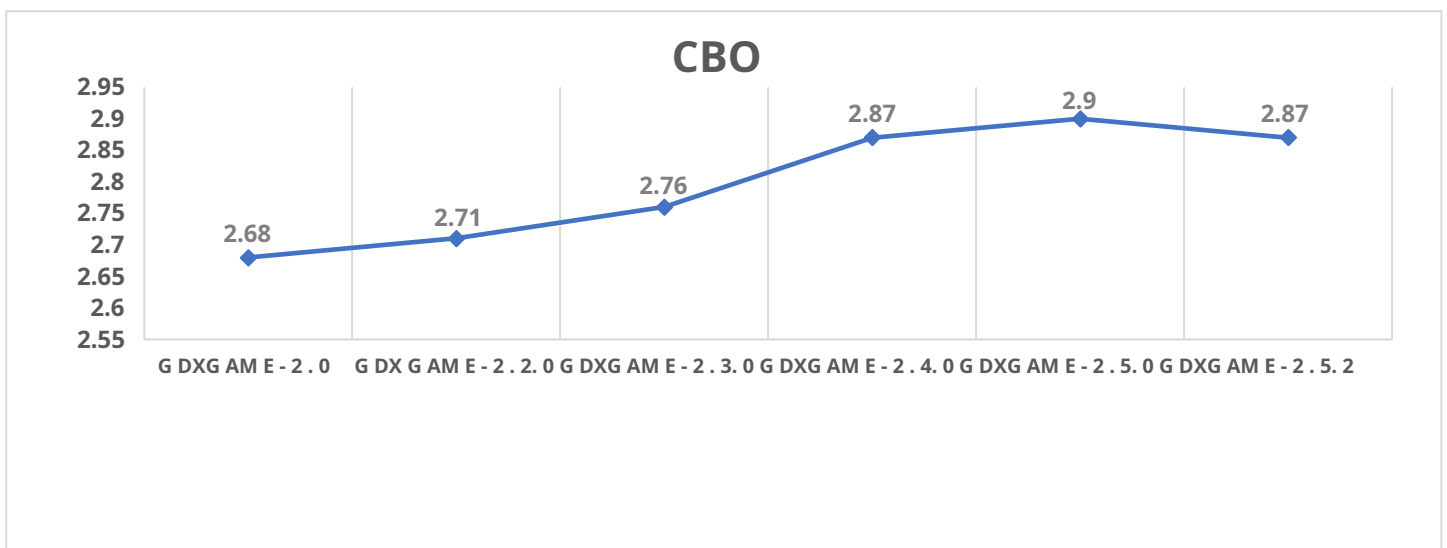
In the above diagram we observe the NOM(Number of Methods) metric, which mainly follows an upward trend except for small falls in its value. The first decrease is observed from GdxGame-2.0 to GdxGame-2.2.0 (8.44 -> 8.35) while the second from GdxGame-2.5.0 to GdxGame-2.5.2 (8.92 -> 8, 85) respectively.

3.1.1.8 NOM (Number of Methods) + NOF (Number of Fields)



Above we observe the diagram of the sum of the metrics NOM (Number of Methods) + NOF (Number of Fields) which mainly follows an increasing course. The only time we see a different change is from GdxGame-2.4.0 to GdxGame-2.5.0(13.38->9.3), where there is a steep drop of 4.08 points.

3.1.1.9 Coupling Between Objects (CBO)



In the above diagram we observe the metric CBO (Coupling Between Objects) which throughout the duration follows an upward trend with a single point of decrease observed from GdxGame-2.5.0 version to GdxGame-2.5.2 (2,9 -> 2, 87).

Metric		1.0	2.0	2.2.0	2.3.0	2.4.0	2.5.0	2.5.2	2.5.3
NOC		39	102	112	115	116	117	120	125
DIT	SUM	7	28	29	30	30	30	30	30
	AVG	0.18	0.27	0.26	0.26	0.26	0.26	0.25	0.24
	MAX	1	3	3	3	3	3	3	3
NOCC	SUM	11	23	26	27	27	27	27	27
	AVG	0.28	0.23	0.23	0.23	0.23	0.23	0.22	0.22
	MAX	5	7	9	9	9	9	9	9
CBO	SUM	86	273	303	318	333	340	334	359
	AVG	2.21	2.68	2.71	2.77	2.87	2.91	2.87	2.87
	MAX	10	14	14	15	15	15	15	18
LCOM	SUM	2,431.00	5,516.00	5,805.00	6,436.00	6,731.00	6,848.00	7,062.00	7,286.00
	AVG	62,33	54.08	51.83	55.97	58.03	58,53	58.85	58,29
	MAX	528	703	703	703	703	703	780	780
CC/ WMC*	SUM	48,38	144.31	160.51	164.46	175.87	178.64	172.64	179.04
	AVG	1.24	1.41	1.43	1.43	1.52	1.53	1.43	1.43
	MAX	5	6.8	6.8	6.8	8.75	11.25	6.8	6.8
LoC	SUM	2,230.00	7,242.00	7,951.00	8,404.00	8,741.00	8,973.00	9,120.00	9,426.00
	AVG	57.18	71	70.99	73.08	75.35	76.69	76	75.41
	MAX	199	475	510	511	513	515	539	581

In the table above, the software versions are represented horizontally and some of the metrics in the entire code are shown vertically. For each metric there are the following functions:

- cumulative (SUM)
- Average (AVG)
- maximum (MAX).

From the table and the metrics we conclude that, from the second edition onwards, most of the metrics are at their lowest level. That is, there is an increase from the first version, but we notice that the second version is the one with the highest quality, as all the metrics are at their minimum. In fact, the second version is about 20 classes away from the final version, therefore a large part of the project has already been implemented, while in the first version there are only 39 classes. So the second version is a good benchmark we can rely on. Then we see that from version to version there are changes that lead mainly to a deterioration in the quality of the software but there are also cases in which they are either better as in AVG, LCOM in version 2, or equal as in DIT in MAX,

MAX, LCOM to MAX, CC to MAX etc. Throughout the duration of the software as the versions progress we notice that the metrics either increase or remain constant. As we reach the last two versions, we see that, while our software takes its final form, some metrics instead of growing are decreasing, as in AVG for DIT, NOCC, WMC* and in the last 2 versions. In conclusion, our software in versions 2.2 up to 2.5 created technical debt, as all the metrics had an upward trend, while as we approach the final version we notice that the metrics become softer and smoothed out.

	1.0-2.0	2.0-2.2.0	2.2.0-2.3.0	2.3.0-2.4.0	2.4.0-2.5.0	2.5.0-2.5.2	2.5.2-2.5.3	MO
Class Growth:	161.54%	9.80%	2.68%	0.87%	0.86%	2.56%	4.17%	26.07%
CBO Growth:	217.44%	10.90%	4.90%	4.71%	2.10%	- 1.78%	7.48%	35.11%
CC Growth:	198.28%	11.23%	2.46%	6.94%	1.58%	- 3.36%	3.71%	31.55%
Size Growth:	224.75%	9.79%	5.70%	4.01%	2.65%	1.64%	3.36%	35.99%

The table above shows us the percentage change from version to version of some key metrics. As we observe for Class Growth in the passage of versions, the highest value is marked by 1st in 2nd version, in which, as we saw in the previous results, most of the code was implemented. Then we see a gradual decrease from 2.4 to 2.5 while in 2.5, 2.5.3 there is a very small increase which (from version to version about 2%), which shows us that in some classes code was added and on average we have an increase in class size of 26%. Correspondingly to the coupling between objects, but also to all the others, we can notice that the biggest increases are from the first version to the second as most of the code was implemented there and then there is a gradual small decrease in each version up to 2.5 .2. Finally from 2.5.3 there is a very small increase in the sizes of all metrics. The average general increase is 31%.

4. Identifying and Resolving Quality Issues

In this specific chapter, an attempt is made to identify the technical debt of the software that is to be studied and to solve certain quality problems. For each subsection (issue) of software that is to be studied and resolved certain quality issues. In this specific chapter, an attempt is made to identify the technical debt of the software that is to be studied and to solve certain quality problems. For each subsection (problem) first a description of the function of the code is made, then a description is made in relation to the first a description of the function of the code is made, then a description is made in relation to maintainability. It also mentions a possible need to add new features and how this need would lead to maintainability issues. Yet,

Name	CBO	LCOM	WMC*	SIZE1	AP
main/java/com/gdx/game/entities/player/PlayerPhysicsComponent.java	6	21	4.857143	203	TOP
main/java/com/gdx/game/entities/player/PlayerHUD.java	18	325	3.192308	581	TOP
main/java/com/gdx/game/battle/BattleHUD.java	15	210	2.571429	341	TOP
main/java/com/gdx/game/inventory/InventoryUI.java	9	276	2.583333	417	TOP
main/java/com/gdx/game/quest/QuestGraph.java	7	351	2.535714	267	TOP
main/java/com/gdx/game/entities/npc/NPCGraphicsComponent.java	6	6	4.75	103	MEDIUM
main/java/com/gdx/game/entities/player/PlayerGraphicsComponent.java	6	3	5	76	MEDIUM

To solve the problem, an extensive analysis was performed, both on the results of metrics and code-smells, as well as on the current code of the class. For this purpose the MetricsCalculator.jar tool and the Designite software were used respectively.

MetricsCalculator.jar Results:

As we can see from the table above, the value of the WMC* metric is high, which implies an increase in the complexity of the code's methods, which makes it difficult to maintain and reuse.

Designite Results:

2 kinds of code-smells were detected, Complex Method, which verifies for us the result of the metric for the complexity of the code's methods, and Feature Envy, which indicates that a method calls methods in another class more times than in the source class.

At the same time, after a general study of the code, unnecessary repetition of commands was identified, which makes future modifications difficult since it increases the possibility of error.

4.1 Code Analysis and UML Diagrams:

The UML diagrams are saved in .svg format (large in size) and for this reason they are in the link below. They are distinguished between those built before and after the refactoring of the code:

UML DIAGRAMS

BATTLEHUD – (Vassilis Tsavalias)

Below, basic metrics are presented, which are related to the class we are interested in as well as its corresponding odors before optimization.

MetricsCalculator.jar Results: (Before)

NAME	CBO	LCOM	WMC*	SIZE
BATTLE HUD	15	210	2.57	341

MetricsCalculator.jar Results: (After)

NAME	CBO	LCOM	WMC*	SIZE
BATTLE HUD	14	190	1.29	320

Designite Results:

NAME	CODE SMELLS	
BATTLE HUD	COMPLEX METHOD	FEATURE ENVY

As we can see from the table above, the value of the WMC* metric is high, which implies an increase in the complexity of the code's methods, which makes it difficult to maintain and reuse. We notice that the "BattleHUD" class presents a high CBO. We know that excessive coupling with other classes makes the class more difficult to maintain and reuse. As a logical consequence, we want to reduce it as much as possible without simultaneously increasing (to a prohibitive degree) the others.

Additionally, after using the Designite tool, we noticed that this particular class suffers from 2 code smells. Specifically:

2 kinds of code-smells were detected, Complex Method, which verifies for us the result of the metric for the complexity of the code's methods, and Feature Envy, which indicates that a method calls methods in another class more times than in the source class. Therefore, we notice that the BattleHUD.java class performs functions, for which it is not responsible.

More specifically, the repetitions were observed in the player's operations to implement a battle. At the same time, after a general study of the code, unnecessary repetition of commands was identified, which makes future modifications difficult since it increases the possibility of error.

To solve the problem, the principle of single competence was implemented, with the aim that each method performs a single function and the repeated structures mentioned above are merged under a common abstraction.

These problems need to be addressed and in the following ways we managed to reduce all the metrics we are interested in (CBO, LCOM, WMC*, SIZE). Specifically, we define 3 abstraction levels:

Level 1: The event itself (event, eg PlayerTurnDoneEvent)

Level 2: The event family (eventManager, eg BattleEvent)

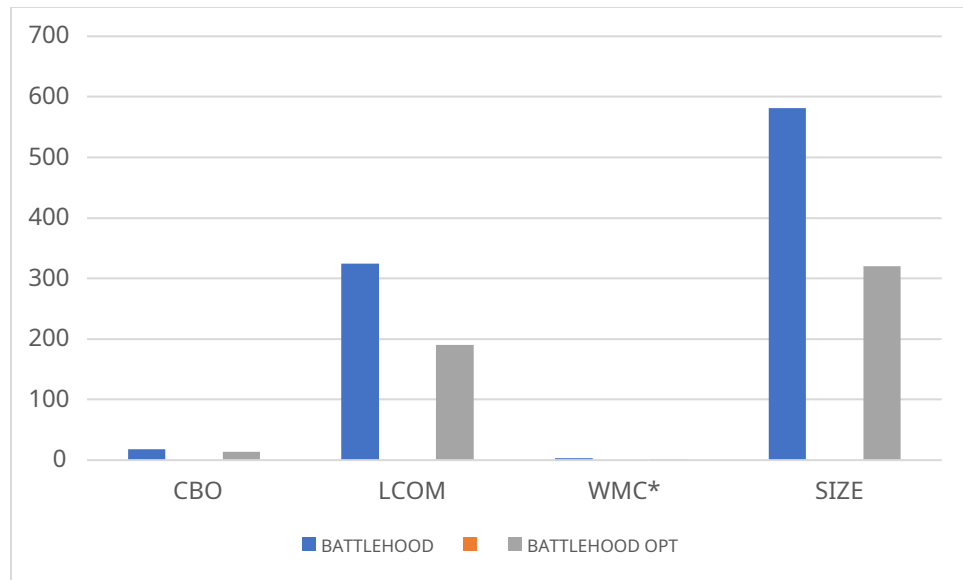
Level 3: The entity that helps implement (eventEntity, e.g. BattleEntity)

* Level 4: The family that entities belong to (generalEntity) *

* Level 4 is an abstraction stage and exists to capture the logic we followed and for future purposes.

In a similar way respectively, all functions performed by the player or by the npc with the item warehouse (inventory), the state of the battle (status) as well as the systems needed (components) were isolated into separate "events" (events) and in turn into separate entities. More specifically, to realize the above, separate classes were created for each function, an abstract class that concerns the family of functions that inherit and finally an entity interface. Through the ActivateEvent method, the appropriate function is performed depending on how it is called in each separate event, and onNotify informs the action to perform through the respective entity.

In conclusion, the above improvements can be seen in the diagram below:



BattleHUD before:

```

@Override
public void onNotify(Entity entity, BattleEvent event) {
    switch(event) {
        case PLAYER_TURN_START:
            LOGGER.debug("Player turn start");
            battleUI.setVisible(false);
            battleUI.setTouchable(Touchable.disabled);
            break;
        case PLAYER_ADDED:
            playerImage.setEntity(entity);
            playerImage.setCurrentAnimation(Entity.AnimationType.WALK_RIGHT);
            playerImage.setSize(playerWidth, playerHeight);
            playerImage.setPosition(x: 0, y: 200);
            playerImage.addAction(Actions.moveTo(x: 200, y: 200, duration: 2));

            currentPlayerImagePosition.set(((MoveToAction) playerImage.getActions().get(0)).getX(), playerImage.getY());
            LOGGER.debug("Player added on battle map");
            break;
        case OPPONENT_ADDED:
            opponentImage.setEntity(entity);
            opponentImage.setCurrentAnimation(Entity.AnimationType.IMMOBILE);
            opponentImage.setSize(enemyWidth, enemyHeight);
            opponentImage.setPosition(x: 600, y: 200);

            currentOpponentImagePosition.set(opponentImage.getX(), opponentImage.getY());
            LOGGER.debug("Opponent added on battle map");

            break;
        case PLAYER_HIT_DAMAGE:
            int damagePlayer = Integer.parseInt(player.getEntityConfig().getPropertyValue(EntityConfig.EntityProperties.ENTITY_HIT_DAMAGE_TOTAL.toString()));
            dmgPlayerValLabel.setText(String.valueOf(damagePlayer));
            dmgPlayerValLabel.setY(origDmgPlayerValLabelY);
            //battleShakeCam.startShaking();
            dmgPlayerValLabel.setVisible(true);

            int hpVal = ProfileManager.getInstance().getProperty( key: "currentPlayerHP", Integer.class);
            battleStatusUI.setHPValue(hpVal);

            if(hpVal <= 0){
                GameScreen.setGameState(GameScreen.GameState.GAME_OVER);
            }
            break;
    }
}

```

```

case OPPONENT_HIT_DAMAGE:
    int damageEnemy = Integer.parseInt(entity.getEntityConfig().getPropertyValue(EntityConfig.EntityProperties.ENTITY_HIT_DAMAGE_TOTAL.toString()));
    dmgOpponentValLabel.setText(String.valueOf(damageEnemy));
    dmgOpponentValLabel.setY(origDmgOpponentValLabelY);
    //battleShakeCam.startShaking();
    dmgOpponentValLabel.setVisible(true);
    LOGGER.debug("Opponent deals " + damageEnemy + " damages");
    break;

case OPPONENT_DEFEATED:
    dmgOpponentValLabel.setVisible(false);
    dmgOpponentValLabel.setY(origDmgOpponentValLabelY);
    setOpponentDefeated();
    LOGGER.debug("Opponent is defeated");

    int xpReward = Integer.parseInt(entity.getEntityConfig().getPropertyValue(EntityConfig.EntityProperties.ENTITY_XP_REWARD.toString()));
    battleStatusUI.addXPValue(xpReward);

    battleUI.setVisible(false);
    battleUI.setTouchable(Touchable.disabled);
    battleStatusUI.setVisible(false);
    battleConversation.notifyBattleResume(enemy);
    break;

case OPPONENT_TURN_DONE:
    if(GameScreen.getGameState() != GameScreen.GameState.GAME_OVER) {
        battleUI.setVisible(true);
        battleUI.setTouchable(Touchable.enabled);
    }
    LOGGER.debug("Opponent turn done");
    break;

case PLAYER_TURN_DONE:
    battleState.opponentAttacks();
    LOGGER.debug("Player turn done");
    break;

default:
    break;
}
}

```

```

@Override
public void onNotify(String value, ComponentEvent event) {
    switch(event) {
        case LOAD_RESUME:
            EntityConfig config = json.fromJson(EntityConfig.class, value);
            notificationUI.loadResume(config);
            break;
        case SHOW_RESUME:
            EntityConfig configShow = json.fromJson(EntityConfig.class, value);

            if (configShow.getEntityID().equalsIgnoreCase(notificationUI.getCurrentEntityID())) {
                notificationUI.setVisible(true);
            }
            break;
        default:
            break;
    }
}

```

```

@Override
public void onNotify(String value, InventoryEvent event) {
    switch(event) {
        case ITEM_CONSUMED:
            String[] strings = value.split(Component.MESSAGE_TOKEN);
            if(strings.length != 2) {
                return;
            }

            int type = Integer.parseInt(strings[0]);
            int typeValue = Integer.parseInt(strings[1]);

            if(InventoryItem.doesRestoreHP(type)) {
                //notify(AudioObserver.AudioCommand.SOUND_PLAY_ONCE, AudioObserver.AudioTypeEvent.SOUND_EATING);
                battleStatusUI.addHPValue(typeValue);
            } else if(InventoryItem.doesRestoreMP(type)) {
                //notify(AudioObserver.AudioCommand.SOUND_PLAY_ONCE, AudioObserver.AudioTypeEvent.SOUND_DRINKING);
                battleStatusUI.addMPValue(typeValue);
            }
            break;
        default:
            break;
    }
}

```

```

@Override
public void onNotify(String value, InventoryEvent event) {
    switch(event) {
        case ITEM_CONSUMED:
            String[] strings = value.split(Component.MESSAGE_TOKEN);
            if(strings.length != 2) {
                return;
            }

            int type = Integer.parseInt(strings[0]);
            int typeValue = Integer.parseInt(strings[1]);

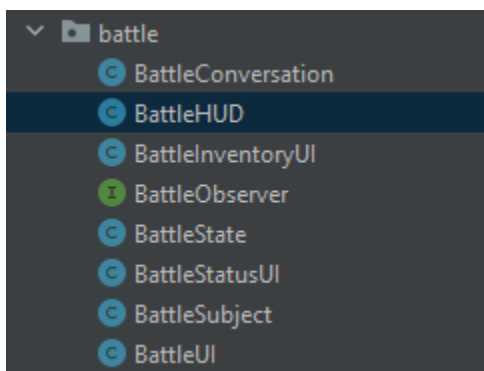
            if(InventoryItem.doesRestoreHP(type)) {
                battleStatusUI.addHPValue(typeValue);
            } else if(InventoryItem.doesRestoreMP(type)) {
                battleStatusUI.addMPValue(typeValue);
            }
            break;
        default:
            break;
    }
}

```

```

@Override
public void onNotify(int value, StatusObserver.StatusEvent event) {
    switch(event) {
        case UPDATED_HP:
            ProfileManager.getInstance().setProperty("currentPlayerHP", battleStatusUI.getHPValue());
            break;
        case UPDATED_LEVEL:
            ProfileManager.getInstance().setProperty("currentPlayerLevel", battleStatusUI.getLevelValue());
            createStatsUpUI(battleStatusUI.getNbrLevelUp());
            break;
        case UPDATED_MP:
            ProfileManager.getInstance().setProperty("currentPlayerMP", battleStatusUI.getMPValue());
            break;
        case UPDATED_XP:
            ProfileManager.getInstance().setProperty("currentPlayerXP", battleStatusUI.getXPValue());
            break;
        case LEVELED_UP:
            //notify(AudioObserver.AudioCommand.MUSIC_PLAY_ONCE, AudioObserver.AudioTypeEvent.MUSIC_LEVEL_UP_FANFARE);
            break;
        default:
            break;
    }
}
}

```



BattleHUD after (the OpponentAddedEvent "event" is used as an example with BattlEvent removed - the same logic is followed for the rest of the functions):

```
@Override
public void onNotify(com.gdx.game.entities.Entity entity, BattleEvent event) {

    event.activateEvent(entity, battleHUD: this);
}
```

```
@Override
public void onNotify(String value, ComponentEvent event) {

    event.activateEvent(value, componentEvent: this);
}
```

```
@Override
public void onNotify(String value, InventoryEvent event) {

    event.activateEvent(value, inventoryEvent: this);
}
```

```
public class BattleHUD implements Screen, BattleEntity, ClassObserver
```

```
public void onNotify(int value, StatusEvent event) {

    event.activateEvent(value, StatusEvent: this);
}
```




```
package com.gdx.game.battle.generalEvents.specificEvents.BattleEvents;
```

```
import ...
```


12 inheritors  Vasilis Tsavalias

```
public abstract class BattleEvent implements BattleEntity {
```

12 implementations  Vasilis Tsavalias

```
    public abstract void activateEvent(Entity entity, BattleHUD battleHUD);  
}
```

```
public class OpponentAddedEvent extends BattleEvent {
```

 Vasilis Tsavalias *

```
@Override
```

```
public void activateEvent(Entity entity, BattleHUD battleHUD) {
```

```
    battleHUD.opponentImage.setEntity(entity);
```

```
    battleHUD.opponentImage.setCurrentAnimation(com.gdx.game.entities.Entity.AnimationType.IMMOBILE);
```


```
    battleHUD.opponentImage.setSize( width: 50, height: 50);
```

```
    battleHUD.opponentImage.setPosition( x: 600, y: 200);
```

```
    battleHUD.currentOpponentImagePosition.set(battleHUD.opponentImage.getX(),battleHUD.opponentImage.getY());
```

```
    LOGGER.debug("Opponent added on battle map");
```

```
}
```

14 implementations  Vasilis Tsavalias

```
public interface BattleEntity extends GeneralEntity {  
    // enum BattleEvent {  
    //     OPPONENT_ADDED,  
    //     OPPONENT_HIT_DAMAGE,  
    //     OPPONENT_DEFEATED,  
    //     OPPONENT_TURN_DONE,  
    //     PLAYER_ADDED,  
    //     PLAYER_HIT_DAMAGE,  
    //     PLAYER_RUNNING,  
    //     PLAYER_TURN_DONE,  
    //     PLAYER_TURN_START,  
    //     PLAYER_USED_MAGIC,  
    //     RESUME_OVER,  
    //     NONE  
    // }  
}
```

1 usage

```
BattleEvent opponentAdded = new OpponentAddedEvent();
```

1 usage

```
BattleEvent opponentHitDamage = new OpponentHitDamageEvent();
```

1 usage

```
BattleEvent opponentDefeated = new OpponentDefeatedEvent();
```

1 usage

```
BattleEvent opponentTurnDone = new OpponentTurnDoneEvent();
```

1 usage

```
BattleEvent playerAdded = new PlayerAddedEvent();
```

1 usage

```
BattleEvent playerHitDamaged = new PlayerHitDamageEvent();
```

1 usage

```
BattleEvent playerRunning = new PlayerRunningEvent();
```

2 usages

```
BattleEvent playerTurnDone = new PlayerTurnDoneEvent();
```

1 usage

```
BattleEvent playerTurnStart = new PlayerTurnStartEvent();
```

1 usage


```
BattleEvent playerUsedMagic = new PlayerUsedMagicEvent();
```

1 usage

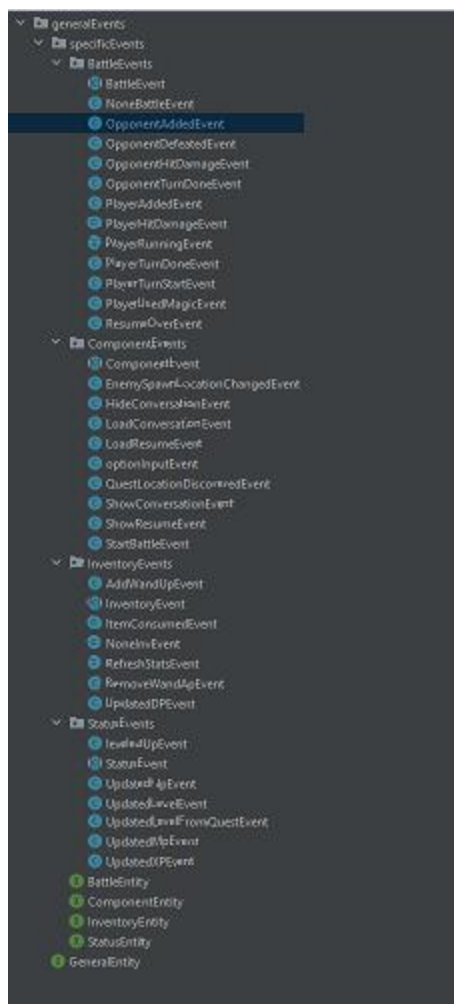
```
BattleEvent ResumeOver = new ResumeOverEvent();
```

no usages

```
BattleEvent None = new NoneBattleEvent();
```

13 implementations  Vasilis Tsavalias

```
void onNotify(final com.gdx.game.entities.Entity enemyEntity, BattleEvent event);
```



PlayerHUD (Christos Giamakidis-Kiosses):

MetricsCalculator.jar Results: (Before)

NAME	CBO	LCOM	WMC*	SIZE
PLAYERHUD	18	325	3.19	581

MetricsCalculator.jar Results: (After)

NAME	CBO	LCOM	WMC*	SIZE
PLAYERHUD	10	351	2	408
NAME	CBO	LCOM	WMC*	SIZE
PLAYERHUD	10	351	2	408

Designite Results:

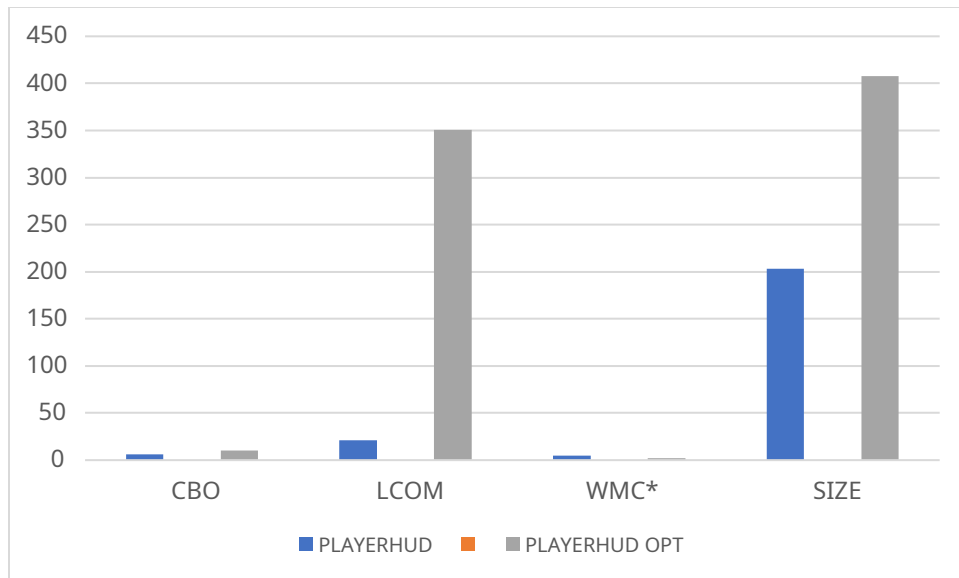
NAME	CODE SMELLS	
PLAYERHUD	INSUFFICIENT MODULARIZATION	FEATURE ENVY

Initially, it was noticed that in this particular class the profile event enumeration class was used for the events taking place in the game. More specifically, there were some constants, such as PROFILE_LOADED, which represented integers 0, 1, 2, etc. In addition, it was observed that the value of the WMC* metric was particularly high, since the implementation was carried out according to the enumerations, which was caused by the existence several switch statements, with just as many cases.

In order to eliminate this problem the existing enumeration classes were replaced, with a class for each of them. Practically a hierarchy is constructed, which starts with a large abstraction of event, which is an Interface that inherits ProfileEvent. More specifically, ProfileEvent is an abstract class with abstract methods, which are implemented in each of the events separately. Therefore, with polymorphism we managed to create the corresponding classes for each of the events of the profile. In fact, instead of the clear current event we create the ProfileClearCurrentEvent, for PROFILE_LOADED the profile loaded event, while for saving the saving event.

Therefore, the code that previously existed in the switch statements was transferred to the body of the abstract methods that inherited the respective events from the corresponding superclass. So, in this way by calling a single method on the event object the code of the switch statements became polymorphic.

[Below we see them](#) various[of the 2 versions:](#)



Below are indicative screenshots of the code before and after the refactors:

PlayerHUD before:

[illegible]

PlayerHUD after:

```

@Override
public void onNotify(ProfileManager profileManager, ProfileEvent event) {
    event.ProfileManagerActivation(profileManager, playerHUD: this);
}

```

```

public abstract class ProfileEvent implements Event {
    1 usage 3 implementations Kostas Thomson
    public abstract void ProfileManagerActivation(ProfileManager profileManager, PlayerHUD playerHUD);
    1 usage 3 implementations Kostas Thomson
    public abstract void ProfileManagerActivation(ProfileManager profileManager, MapManager mapManager);
}

```

```

public class ProfileSavingEvent extends ProfileEvent {
    1 usage Kostas Thomson +1
    @Override
    public void ProfileManagerActivation(ProfileManager profileManager, PlayerHUD playerHUD) {
        profileManager.setJsonPropertiesToSaveProfile(playerHUD);
    }

    1 usage Kostas Thomson +1
    @Override
    public void ProfileManagerActivation(ProfileManager profileManager, MapManager mapManager) {
        profileManager.setJsonPropertiesToSaveMap(mapManager);
    }
}

```

```

public class ProfileLoadedEvent extends ProfileEvent{
    1 usage Kostas Thomson +1
    @Override
    public void ProfileManagerActivation(ProfileManager profileManager, PlayerHUD playerHUD) {
        profileManager.setLoadedProfile(playerHUD);
    }

    1 usage Kostas Thomson +1
    @Override
    public void ProfileManagerActivation(ProfileManager profileManager, MapManager mapManager) {
        mapManager.loadMapFromManager(profileManager);
        mapManager.setPlayerStartPosition(profileManager);
    }
}

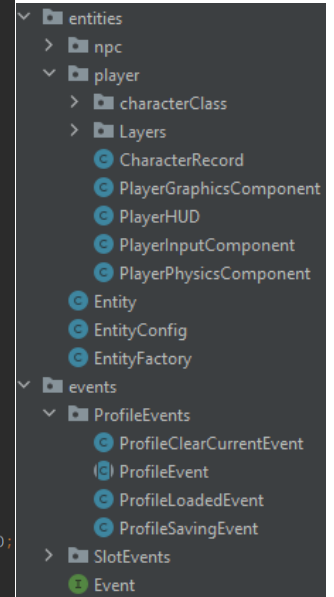
```

```

public class ProfileClearCurrentEvent extends ProfileEvent {
    1 usage  ▲ Kostas Thomson
    @Override
    public void ProfileManagerActivation(ProfileManager profileManager, PlayerHUD playerHUD) {
        profileManager.setProperty("playerInventory", new Array<InventoryItemLocation>());
        profileManager.setProperty("playerEquipInventory", new Array<InventoryItemLocation>());
        profileManager.setProperty("playerCharacter", null);
        profileManager.setProperty("currentPlayerGP", 0);
        profileManager.setProperty("currentPlayerLevel", 0);
        profileManager.setProperty("currentPlayerXP", 0);
        profileManager.setProperty("currentPlayerXPMax", 0);
        profileManager.setProperty("currentPlayerHP", 0);
        profileManager.setProperty("currentPlayerHPMax", 0);
        profileManager.setProperty("currentPlayerMP", 0);
        profileManager.setProperty("currentPlayerMPMax", 0);
        profileManager.setProperty("currentPlayerAP", 0);
        profileManager.setProperty("currentPlayerDP", 0);
        profileManager.setProperty("currentTime", 0);
    }

    1 usage  ▲ Kostas Thomson
    @Override
    public void ProfileManagerActivation(ProfileManager profileManager, MapManager mapManager) {
        mapManager.setCurrentMap(null);
        profileManager.setProperty("currentPlayerPosition", null);
        profileManager.setProperty("currentMapType", MapFactory.MapType.TOPPLE.toString());
        MapFactory.clearCache();
        profileManager.setProperty("toppleMapStartPosition", MapFactory.getMap(MapFactory.MapType.TOPPLE).getPlayerStart());
        profileManager.setProperty("toppleRoad1MapStartPosition", MapFactory.getMap(MapFactory.MapType.TOPPLE_ROAD_1).getPlayerStart());
    }
}

```



PlayerPhysicsComponent (THOMASSIADIS KONSTANTINOS):

Initially, from the analysis we did, we found that this particular class received the designation "TOP" as the metrics CBO (6), WMC* (4.86) and SIZE1 (203) had much higher values than the rest of the classes. Therefore, to reduce the specific metrics:

1. An interface was used as a handler, which handled the following classes.

2. Classes were used to implement the logic for each level.

After these changes, we achieved a reduction in WMC* (1.68) and SIZE1 (112), while necessary compromises were an increase in CBO by one unit (7) and an increase in LCOM to 120.

Analytically:

In the original code it was observed that there were three large methods, which implemented the same logic in a similar way, namely the `updatePortalLayerActivation`, `updateEnemySpawnLayerActivation` and `updateDiscoverLayerActivation` methods. More specifically, each of the above respectively, was responsible for renewing the activation of the Portal level, the EnemySpawn level and the Discover level.

Due to the above observation, i.e. the great similarity of the code implemented by the specific methods, it was deemed necessary to modify them. To solve the problem, a new interface and three classes were defined. More specifically, the `LayerActivation` interface was set to handle the operation of each layer. Then, the three abstract classes simulated the levels and provided a static method that would execute the required logic, at each level. As a result, the three methods from the `PlayerPhysicsComponent` class are moved to the class they represent.

Therefore, thanks to the elimination of repetitive code, the use of a controller and the distribution of logic in discrete entities, a large reduction was observed in the WMC* and SIZE metrics. On the other hand there was a slight increase in the CBO metric, but still necessary, and finally there was the inevitable increase in the LCOM metric, as the software needs many more adjustments to achieve the best possible values.

MetricsCalculator.jar Results: (Before)

NAME	CBO	LCOM	WMC*	SIZE
PLAYERPHYSICSCOMPONENT	6	21	4.85	203

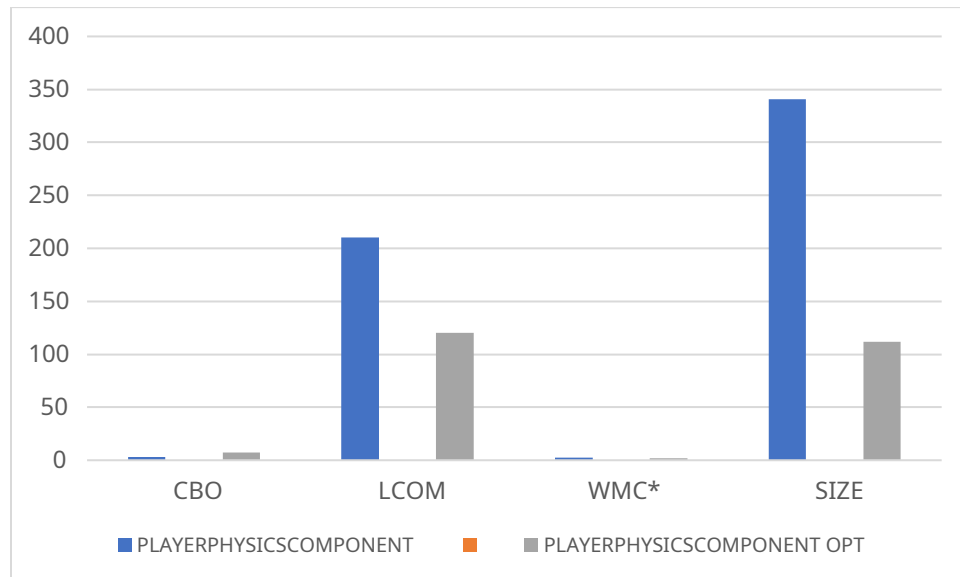
MetricsCalculator.jar Results: (After)

NAME	CBO	LCOM	WMC*	SIZE
PLAYERPHYSICSCOMPONENT	7	120	1.68	112

Code smells:

NAME	CODE SMELLS	
PLAYER PHYSICSCOMPONENT	COMPLEX METHOD	FEATURE ENVY

The chart below shows the differences in our improvements:



PlayerPhysicsComponent before:

```

private boolean updateDiscoverLayerActivation(MapManager mapMgr) {
    MapLayer mapDiscoverLayer = mapMgr.getQuestDiscoverLayer();

    if(mapDiscoverLayer == null) {
        return false;
    }

    Rectangle rectangle;

    for(MapObject object: mapDiscoverLayer.getObjects()) {
        if(object instanceof RectangleMapObject) {
            rectangle = ((RectangleMapObject)object).getRectangle();

            if(boundingBox.overlaps(rectangle)) {
                String questID = object.getName();
                String questTaskID = (String)object.getProperties().get("taskID");
                String val = questID + MESSAGE_TOKEN + questTaskID;

                if(questID == null) {
                    return false;
                }

                if(previousDiscovery.equalsIgnoreCase(val)) {
                    return true;
                } else {
                    previousDiscovery = val;
                }

                notify(json.toJson(val), ComponentObserver.ComponentEvent.QUEST_LOCATION_DISCOVERED);
                LOGGER.debug("Discover Area Activated");
                return true;
            }
        }
    }

    return false;
}

```

```

private boolean updateEnemySpawnLayerActivation(MapManager mapMgr) {
    MapLayer mapEnemySpawnLayer = mapMgr.getEnemySpawnLayer();

    if(mapEnemySpawnLayer == null) {
        return false;
    }

    Rectangle rectangle;

    for(MapObject object: mapEnemySpawnLayer.getObjects()) {
        if(object instanceof RectangleMapObject) {
            rectangle = ((RectangleMapObject)object).getRectangle();

            if(boundingBox.overlaps(rectangle)) {
                String enemySpawnID = object.getName();

                if(enemySpawnID == null) {
                    return false;
                }

                if(previousEnemySpawn.equalsIgnoreCase(enemySpawnID)) {
                    return true;
                } else {
                    LOGGER.debug("Enemy Spawn Area " + enemySpawnID + " Activated with previous Spawn value: " + previousEnemySpawn);
                    previousEnemySpawn = enemySpawnID;
                }

                notify(enemySpawnID, ComponentObserver.ComponentEvent.ENEMY_SPAWN_LOCATION_CHANGED);
                return true;
            }
        }
    }

    //If no collision, reset the value
    if(!previousEnemySpawn.equalsIgnoreCase(String.valueOf(0))) {
        LOGGER.debug("Enemy Spawn Area RESET with previous value " + previousEnemySpawn);
        previousEnemySpawn = String.valueOf(0);
        notify(previousEnemySpawn, ComponentObserver.ComponentEvent.ENEMY_SPAWN_LOCATION_CHANGED);
    }

    return false;
}

```

```

private boolean updatePortalLayerActivation(MapManager mapMgr) {
    MapLayer mapPortalLayer = mapMgr.getPortalLayer();

    if(mapPortalLayer == null) {
        return false;
    }

    Rectangle rectangle;

    for(MapObject object: mapPortalLayer.getObjects()) {
        if(object instanceof RectangleMapObject) {
            rectangle = ((RectangleMapObject)object).getRectangle();

            if(boundingBox.overlaps(rectangle)) {
                String mapName = object.getName();
                if(mapName == null) {
                    return false;
                }

                mapMgr.setClosestStartPositionFromScaledUnits(currentEntityPosition);
                mapMgr.loadMap(MapFactory.MapType.valueOf(mapName));

                currentEntityPosition.x = mapMgr.getPlayerStartUnitScaled().x;
                currentEntityPosition.y = mapMgr.getPlayerStartUnitScaled().y;
                nextEntityPosition.x = mapMgr.getPlayerStartUnitScaled().x;
                nextEntityPosition.y = mapMgr.getPlayerStartUnitScaled().y;

                LOGGER.debug("Portal Activated");
                return true;
            }
        }
    }

    return false;
}

```

PlayerPhysicsComponent after:

```

public interface LayerActivation {
    1 usage  👤 kostasthompson
    static boolean[] updateLayerActivation(PlayerPhysicsComponent ppComponent, MapManager mapMgr) {
        return new boolean[]
        {
            PortalLayer.Activate(ppComponent, mapMgr),
            DiscoverLayer.Activate(ppComponent, mapMgr),
            EnemySpawnLayer.Activate(ppComponent, mapMgr)
        };
    }
}

@Override
public void update(Entity entity, MapManager mapMgr, float delta) {
    //We want the hit-box to be at the feet for a better feel
    updateBoundingBoxPosition(nextEntityPosition);
    LayerActivation.updateLayerActivation(ppComponent: this, mapMgr);
    if(isMouseSelectEnabled) {
        selectMapEntityCandidate(mapMgr);
        isMouseSelectEnabled = false;
    }

    if(!isCollisionWithMapLayer(entity, mapMgr) && !isCollisionWithMapEntities(entity, mapMgr) && state == Entity.State.WALKING) {
        setNextPositionToCurrent(entity);

        Camera camera = mapMgr.getCamera();
        camera.position.set(currentEntityPosition.x, currentEntityPosition.y, 0f);
        camera.update();
    } else {
        updateBoundingBoxPosition(currentEntityPosition);
    }

    calculateNextPosition(delta);
}

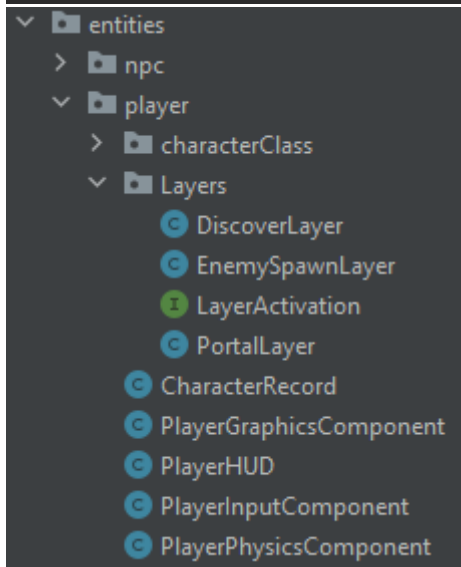
```

```

public class PortalLayer {
    1 usage  ▲ kostasthompson
    public static boolean Activate(PlayerPhysicsComponent ppComponent, MapManager mapMgr) {
        MapLayer mapPortalLayer = mapMgr.getPortalLayer();

        if (mapPortalLayer != null) {
            Rectangle rectangle;
            for(MapObject object: mapPortalLayer.getObjects()) {
                if(object instanceof RectangleMapObject) {
                    rectangle = ((RectangleMapObject)object).getRectangle();
                    if(ppComponent.isBoundingBoxOverlaps(rectangle)) {
                        String mapName = object.getName();
                        if(mapName == null) {
                            return false;
                        }
                        mapMgr.setClosestStartPositionFromScaledUnits(ppComponent.getCurrentEntityPosition());
                        mapMgr.loadMap(MapFactory.MapType.valueOf(mapName));
                        ppComponent.setCurrentEntityPosition(new Vector2(mapMgr.getPlayerStartUnitScaled().x, mapMgr.getPlayerStartUnitScaled().y));
                        ppComponent.setNextEntityPosition(new Vector2(mapMgr.getPlayerStartUnitScaled().x, mapMgr.getPlayerStartUnitScaled().y));
                        ppComponent.printLOGGER("Portal Activated");
                        return true;
                    }
                }
            }
        }
        return false;
    }
}

```



```

public class DiscoverLayer {
    1 usage  ▲ kostasthompson
    public static boolean Activate(PlayerPhysicsComponent ppComponent, MapManager mapMgr) {
        MapLayer mapDiscoverLayer = mapMgr.getQuestDiscoverLayer();
        if (mapDiscoverLayer != null) {
            Rectangle rectangle;
            for(MapObject object: mapDiscoverLayer.getObjects()) {
                if(object instanceof RectangleMapObject) {
                    rectangle = ((RectangleMapObject)object).getRectangle();

                    if(ppComponent.isBoundingBoxOverlaps(rectangle)) {
                        String questID = object.getName();
                        String questTaskID = (String)object.getProperties().get("taskID");
                        String val = questID + ppComponent.getMessageToken() + questTaskID;
                        if(questID == null) {
                            break;
                        }
                        ppComponent.setPreviousDiscovery(val);
                        ppComponent.notify(ppComponent.toJson(val), ComponentObserver.ComponentEvent.QUEST_LOCATION_DISCOVERED);
                        ppComponent.printLOGGER("Discover Area Activated");
                        return true;
                    }
                }
            }
        }
        return false;
    }
}

```

```

public class EnemySpawnLayer {
    1 usage  ▲ kostasthompson
    public static boolean Activate(PlayerPhysicsComponent ppComponent, MapManager mapMgr) {
        MapLayer mapEnemySpawnLayer = mapMgr.getEnemySpawnLayer();
        if (mapEnemySpawnLayer != null) {
            Rectangle rectangle;
            for(MapObject object: mapEnemySpawnLayer.getObjects()) {
                if(object instanceof RectangleMapObject) {
                    rectangle = ((RectangleMapObject)object).getRectangle();
                    if(ppComponent.isBoundingBoxOverlaps(rectangle)) {
                        String enemySpawnID = object.getName();
                        if (enemySpawnID != null && ppComponent.isPreviousEnemySpawnEquals(enemySpawnID)) {
                            ppComponent.notify(enemySpawnID, ComponentObserver.ComponentEvent.ENEMY_SPAWN_LOCATION_CHANGED);
                            return true;
                        }
                        ppComponent.callLOGGER(enemySpawnID);
                        return false;
                    }
                }
            }
            //If no collision, reset the value
            if(!ppComponent.isPreviousEnemySpawnEquals(String.valueOf(0))) {
                ppComponent.resetLOGGER(String.valueOf(0));
                ppComponent.notify(String.valueOf(0), ComponentObserver.ComponentEvent.ENEMY_SPAWN_LOCATION_CHANGED);
            }
        }
        return false;
    }
}

```

StatusUI (Aristea Kotoula) :

Initially, from the analysis we did, we found that this particular class received the designation "MEDIUM" as the metrics LCOM (780) and SIZE1 (288) had much higher values than the rest of the classes. So to reduce the metrics:

We followed the strategy of isolation. That is, for each function we divided it into separate methods where each method performs the specific operation (eg, handle XP). As a level of abstraction we defined an abstract class "Handle" which contains the methods of managing the player's operations.

In the original code there were 3 big methods that similarly implemented the logic of layer activation (Layer Activation). Each one for it

MetricsCalculator.jar Results: (Before)

NAME	CBO	LCOM	WMC*	SIZE
STATUSUI	3	780	1.14	288

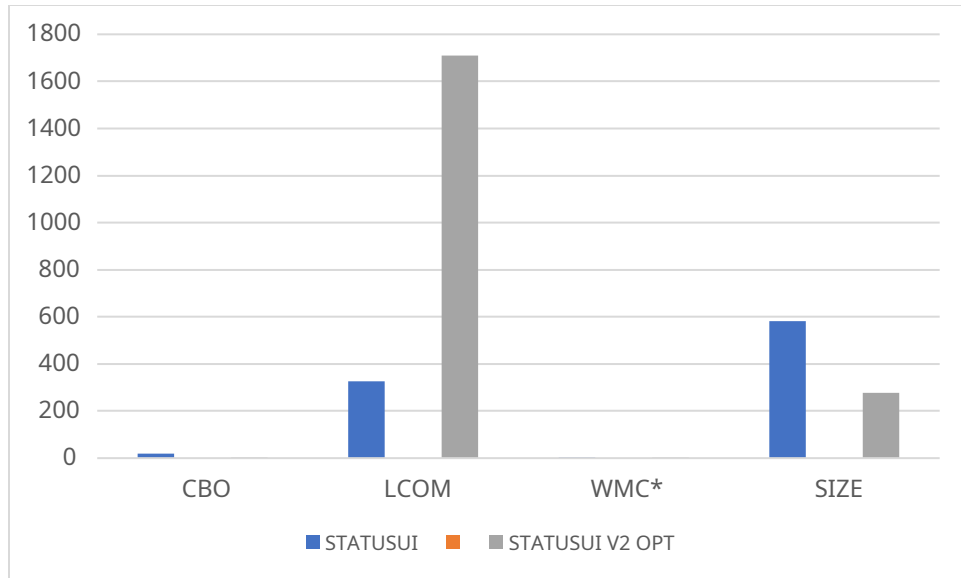
MetricsCalculator.jar Results: (After)

NAME	CBO	LCOM	WMC*	SIZE
STATUSUI	3	1711	1.1	278

Code smells:

NAME	CODE SMELLS
STATUSUI	INSUFFICIENT MODULARIZATION

In addition, the improvements we made in this class are also visible:



StatusUI before:

```

private void handleXpBar() {
    WidgetGroup group3 = new WidgetGroup();

    xpBar = new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "XP_Bar"));
    Image bar3 = new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "Bar"));

    Label xpLabel = new Label( text: " xp: ", STATUS_UI_SKIN);
    xpValLabel = new Label(String.valueOf(xpVal), STATUS_UI_SKIN);

    xpBar.setPosition( x: 3, y: 6);

    group3.addActor(bar3);
    group3.addActor(xpBar);

    this.add(group3).size(bar3.getWidth(), bar3.getHeight()).padRight(10);
    this.add(xpLabel);
    this.add(xpValLabel).align(Align.left).padRight(20);
    this.row();
}

```

```

private void handleInventoryButton() {
    inventoryButton = new ImageButton(STATUS_UI_SKIN, styleName: "inventory-button");
    inventoryButton.getImageCell().size( width: 32, height: 32);

    this.add(inventoryButton).align(Align.right);
}

```

1 usage

```

private void handleQuestButton() {
    questButton = new ImageButton(STATUS_UI_SKIN, styleName: "quest-button");
    questButton.getImageCell().size( width: 32, height: 32);

    this.add(questButton).align(Align.center);
}

```

```

private void handleHpBar() {
    WidgetGroup group = new WidgetGroup();

    hpBar = new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "HP_Bar"));
    Image bar = new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "Bar"));

    Label hpLabel = new Label( text: " hp: ", STATUS_UI_SKIN);
    hpValLabel = new Label(String.valueOf(hpVal), STATUS_UI_SKIN);
    Label separator = new Label( text: "/", STATUS_UI_SKIN);
    hpValLabelMax = new Label(String.valueOf(hpCurrentMax), STATUS_UI_SKIN);

    hpBar.setPosition( x: 3, y: 6);

    group.addActor(bar);
    group.addActor(hpBar);

    this.add(group).size(bar.getWidth(), bar.getHeight()).padRight(10);
    this.add(hpLabel);
    this.add(hpValLabel);
    this.add(separator);
    this.add(hpValLabelMax);
    this.row();
}

```

```

private void handleLevelLabel() {
    Label levelLabel = new Label( text: " lv: ", STATUS_UI_SKIN);
    levelValLabel = new Label(String.valueOf(levelVal), STATUS_UI_SKIN);

    this.add(levelLabel).align(Align.left);
    this.add(levelValLabel).align(Align.left);
}

```

1 usage

```

private void handleGoldLabel() {
    Label goldLabel = new Label( text: " gp: ", STATUS_UI_SKIN);
    goldValLabel = new Label(String.valueOf(goldVal), STATUS_UI_SKIN);

    this.add(goldLabel);
    this.add(goldValLabel).align(Align.left);
}

```

```

private void handleMpBar() {
    WidgetGroup group2 = new WidgetGroup();

    mpBar = new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "MP_Bar"));
    Image bar2 = new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "Bar"));

    Label mpLabel = new Label( text: " mp: ", STATUS_UI_SKIN);
    mpValLabel = new Label(String.valueOf(mpVal), STATUS_UI_SKIN);
    Label separator = new Label( text: "/", STATUS_UI_SKIN);
    mpValLabelMax = new Label(String.valueOf(mpCurrentMax), STATUS_UI_SKIN);

    mpBar.setPosition( x: 3, y: 6);

    group2.addActor(bar2);
    group2.addActor(mpBar);

    this.add(group2).size(bar2.getWidth(), bar2.getHeight()).padRight(10);
    this.add(mpLabel);
    this.add(mpValLabel);
    this.add(separator);
    this.add(mpValLabelMax);
    this.row();
}

```

```

public StatusUI(){
    super( title: "stats", STATUS_UI_SKIN);

    levelTables = LevelTable.getLevelTables(LEVEL_TABLE_CONFIG);

    observers = new Array<>();

    //Add to layout
    defaults().expand().fill();

    //account for the title padding
    this.pad( top: this.getPadTop() + 10, left: 10, bottom: 10, right: 10);

    this.add();
    handleQuestButton();
    handleInventoryButton();
    this.row();

    handleHpBar();
    handleMpBar();
    handleXpBar();

    handleLevelLabel();
    this.row();
    handleGoldLabel();

    //this.debug();
    this.pack();

    barWidth = hpBar.getWidth();
    barHeight = hpBar.getHeight();
}

```

StatusUI after:

```

public StatusUI(){
    super( title: "stats", STATUS_UI_SKIN);

    levelTables = LevelTable.getLevelTables(LEVEL_TABLE_CONFIG);

    observers = new Array<>();

    //Add to layout
    defaults().expand().fill();

    //account for the title padding
    this.pad( top: this.getPadTop() + 10, left: 10, bottom: 10, right: 10);

    this.add();
    ButtonHandler.handleQuest( statusUI: this);
    ButtonHandler.handleInventory( statusUI: this);
    this.row();
    BarHandler.handleHP( statusUI: this);
    BarHandler.handleMP( statusUI: this);
    BarHandler.handleXP( statusUI: this);
    LabelHandler.handleLevel( statusUI: this);
    this.row();
    LabelHandler.handleGold( statusUI: this);
    //this.debug();
    this.pack();

    barWidth = hpBar.getWidth();
    barHeight = hpBar.getHeight();
}

```

```

public abstract class BarHandler implements Handler {
    1 usage  👤 kostasthomson
    public static void handleHP(StatusUI statusUI) {
        statusUI.setHPBar(new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "HP_Bar")));
        statusUI.setHPBarPosition( x: 3, y: 6);
        statusUI.updateHPValLabel();
        statusUI.updateHPValMaxLabel();

        Image bar = new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "Bar"));
        WidgetGroup group = new WidgetGroup(bar, statusUI.getHPBar());
        statusUI.add(group).size(bar.getWidth(), bar.getHeight()).padRight(10);

        statusUI.addHPToGroup();
    }

    1 usage  👤 kostasthomson
    public static void handleMP(StatusUI statusUI) {
        statusUI.setMPBar(new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "MP_Bar")));
        statusUI.setMPBarPosition( x: 3, y: 6);
        statusUI.updateMPValLabel();
        statusUI.updateMPValMaxLabel();

        Image bar = new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "Bar"));
        WidgetGroup group = new WidgetGroup(bar, statusUI.getMPBar());
        statusUI.add(group).size(bar.getWidth(), bar.getHeight()).padRight(10);

        statusUI.addMPToGroup();
    }

    1 usage  👤 kostasthomson
    public static void handleXP(StatusUI statusUI) {
        statusUI.setXBar(new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "XP_Bar")));
        statusUI.setXBarPosition( x: 3, y: 6);
        statusUI.updateXPValLabel();

        Image bar = new Image(STATUS_UI_TEXTURE_ATLAS.findRegion( name: "Bar"));
        WidgetGroup group = new WidgetGroup(bar, statusUI.getXBar());
        statusUI.add(group).size(bar.getWidth(), bar.getHeight()).padRight(10);

        statusUI.addXPToGroup();
    }
}

```

```

public abstract class ButtonHandler implements Handler {

    1 usage  👤 kostasthompson
    public static void handleInventory(StatusUI statusUI) {
        statusUI.setInventoryButton(new ImageButton(STATUS_UI_SKIN, styleName: "inventory-button"));
        statusUI.setInventoryButtonSize(x: 32, y: 32);
        statusUI.addAndAlign(statusUI.getInventoryButton(), Align.right);
    }

    1 usage  👤 kostasthompson
    public static void handleQuest(StatusUI statusUI) {
        statusUI.setQuestButton(new ImageButton(STATUS_UI_SKIN, styleName: "quest-button"));
        statusUI.setQuestButtonSize(x: 32, y: 32);
        statusUI.addAndAlign(statusUI.getQuestButton(), Align.center);
    }

}

```

- ▼ status
 - ▼ Handlers
 - 🔍 BarHandler
 - 🔍 ButtonHandler
 - 🔍 Handler
 - 🔍 LabelHandler
 - 🔍 LevelTable
 - 🔍 StatsUpUI
 - 🔍 StatusObserver
 - 🔍 StatusSubject
 - 🔍 StatusUI


```

public abstract class LabelHandler implements Handler {

    1 usage  👤 kostasthomson
    public static void handleLevel(StatusUI statusUI) {
        Label levelLabel = new Label( text: " lv: ", STATUS_UI_SKIN);
        statusUI.updateLevelValLabel();
        statusUI.addAndAlign(levelLabel, Align.left);
        statusUI.addAndAlign(statusUI.getLevelValLabel(), Align.left);
    }

    1 usage  👤 kostasthomson
    public static void handleGold(StatusUI statusUI) {
        Label goldLabel = new Label( text: " gp: ", STATUS_UI_SKIN);
        statusUI.updateGoldValLabel();
        statusUI.add(goldLabel);
        statusUI.addAndAlign(statusUI.getGoldValLabel(), Align.left);
    }

}

```

QuestGraph (Kontaxis Ioannis) :

Initially from the analysis we did we found that this particular class received the designation "TOP" as the metrics CBO (6), LCOM (351) and "SIZE1 (267) had much higher values than the rest of the classes. Therefore, to reduce the specific metrics:

1. An external interface was used which handled the complex calculations (Handler).
2. The principle of single responsibility (Single Responsibility) was used to reduce the complexity of large methods.
3. The Information Expert principle was used to maximize consistency between methods and also to apply logic to methods that have direct access to information.

In these ways we managed to reduce CBO to 5, WMC* to 1.62 (from 2.53) and SIZE1 to 158 lines. However, there was a small increase in LCOM which was necessary, as more software changes are needed to solve this problem.

MetricsCalculator.jar Results: (Before)

NAME	CBO	LCOM	WMC*	SIZE
QUESTGRAPH	7	351	2.53	267

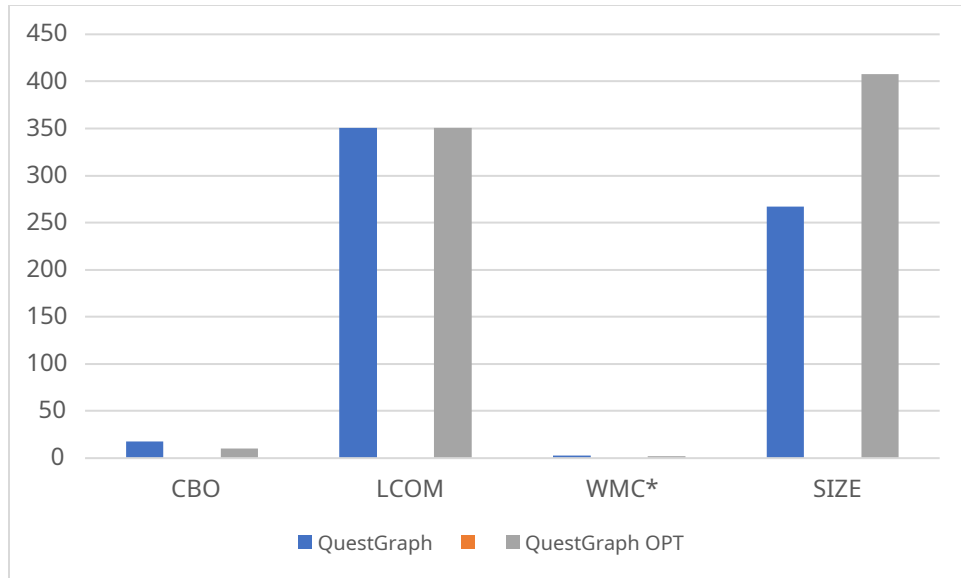
MetricsCalculator.jar Results: (After)

NAME	CBO	LCOM	WMC*	SIZE
QUESTGRAPH	5	378	1.62	158

Code smells:

NAME	CODE SMELLS	
INSUFFICIENT MODULARIZATION	INSUFFICIENT MODULARIZATION	CYCLICALLY DEPENDENT MODULARIZATION

Finally, in the diagram below we see the differences in metrics between the optimized class we implemented and the original one:



QuestGraph before:

```

public void update(MapManager mapMgr) {
    ArrayList<QuestTask> allQuestTasks = getAllQuestTasks();
    for(QuestTask questTask: allQuestTasks) {

        if(questTask.isTaskComplete()) {...}

        //We first want to make sure the task is available and is relevant to current location
        if(!isQuestTaskAvailable(questTask.getId())) {...}

        String taskLocation = questTask.getPropertyValue(QuestTask.QuestTaskPropertyType.TARGET_LOCATION.toString());
        if(taskLocation == null || taskLocation.isEmpty() || !taskLocation.equalsIgnoreCase(mapMgr.getCurrentMapType().toString())) {...}

        switch(questTask.getQuestType()) {
            case FETCH:
                String taskConfig = questTask.getPropertyValue(QuestTask.QuestTaskPropertyType.TARGET_TYPE.toString());
                if(taskConfig == null || taskConfig.isEmpty()) {...}
                EntityConfig config = Entity.getEntityConfig(taskConfig);

                Array<Vector2> questItemPositions = ProfileManager.getInstance().getProperty(config.getEntityID(), Array.class);
                if(questItemPositions == null) {...}

                //Case where all the items have been picked up
                if(questItemPositions.size == 0) {
                    questTask.setTaskComplete();
                    LOGGER.debug("TASK : " + questTask.getId() + " is complete of Quest: " + questID);
                    LOGGER.debug("INFO : " + QuestTask.QuestTaskPropertyType.TARGET_TYPE.toString());
                }
                break;
            case KILL:
                break;
            case DELIVERY:
                break;
            case GUARD:
                break;
            case ESCORT:
                break;
            case RETURN:
                break;
            case DISCOVER:
                break;
        }
    }
}

```

QuestGraph after:

```

public void update(MapManager mapMgr) {
    Handler.RunRutime(mapMgr, graph: this, Handler.Action.UPDATE);
}

```

👤 Kostas Thomson +1

```

public void init(MapManager mapMgr) {
    Handler.RunRutime(mapMgr, graph: this, Handler.Action.INIT);
}

```

- ▼ quest
 - Handler
 - QuestGraph
 - QuestTask
 - QuestTaskDependency
 - QuestUI

```

public interface Handler {
    3 usages 👤 kostasthompson
    enum Action {
        2 usages
        UPDATE,
        2 usages
        INIT
    }
    2 usages 👤 kostasthompson
    static void RunRutime(MapManager mapMgr, QuestGraph graph, Action action) {
        ArrayList<QuestTask> allQuestTasks = graph.getAllQuestTasks();
        for(QuestTask questTask: allQuestTasks) {
            if (questTask.isTaskComplete()) {
                continue;
            }
            //We first want to make sure the task is available and is relevant to current location
            if (!graph.isQuestTaskAvailable(questTask.getId())) {
                continue;
            }
            String taskLocation = questTask.getPropertyValue(QuestTask.QuestTaskPropertyType.TARGET_LOCATION.toString());
            if (taskLocation == null || taskLocation.isEmpty() || !taskLocation.equalsIgnoreCase(mapMgr.getCurrentMapType().toString())) {
                continue;
            }
            switch (action) {
                case UPDATE:
                    if (questTask.getQuestType() == FETCH) {...}
                    break;
                case INIT:
                    if (questTask.getQuestType() == FETCH) {...}
                    break;
                default:
                    break;
            }
        }
    }
}

```

Inventory UI (Elisabet Persephone Kanidou ics21095) :

Regarding the InventoryUI class, it was initially noticed that in this particular class in the original code an enumeration class was used for the events that occurred during the execution of the game. More specifically the slot event enumeration class.

At the same time, due to the above implementation, according to the enumerations, a high value of the WMC* metric was observed, an inevitable event since there were several switch statements which also had many cases, with the result that the complexity of the class methods increases.

To address the above problem, polymorphism was implemented and a separate class was created for each of the enumerations. More specifically, a class hierarchy was created with the initial Event abstraction. In more detail, this abstraction is an Interface that inherits SlotEvent which is an abstract class that contains abstract methods with which each subclass polymorphically implements the code. In particular, the previous code of the switch statements was replaced with a polymorphic call and the implementation was taken over by the subclasses of SlotEvent in the body of the methods that inherit from the superclass.

MetricsCalculator.jar Results: (Before)

NAME	CBO	LCOM	WMC*	SIZE
INVENTORYUI	9	276	2.58	417

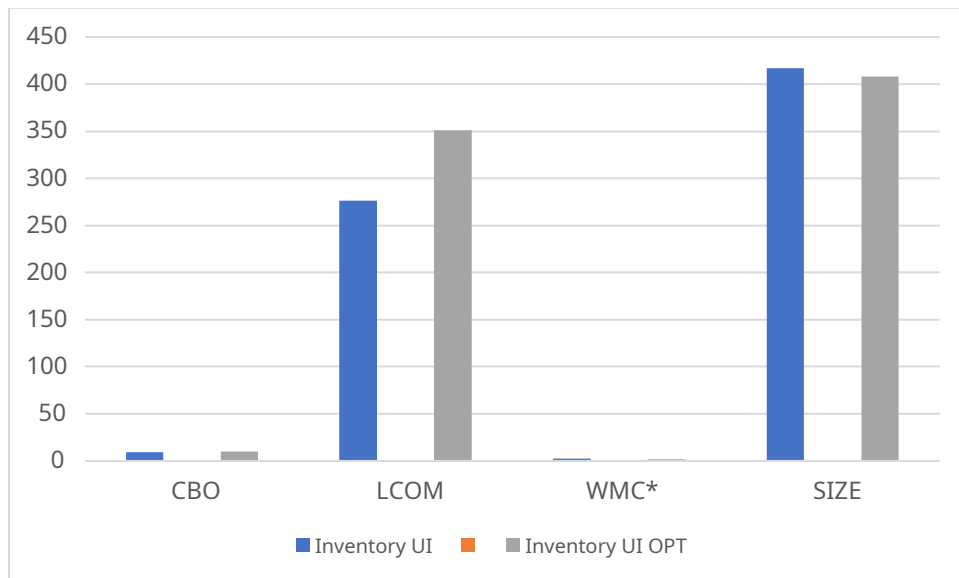
MetricsCalculator.jar Results: (After)

NAME	CBO	LCOM	WMC*	SIZE
INVENTORYUI	10	351	2	408

Code smells:

NAME	CODE SMELLS	
INVENTORYUI	DEFICIENT ENCAPSULATION	INSUFFICIENT MODULARIZATION

The metric changes are shown below in diagrammatic form:



InventoryUI before:

```

@Override
public void onNotify(InventorySlot slot, SlotEvent event) {
    switch(event) {
        case ADDED_ITEM:
            InventoryItem addItem = slot.getTopInventoryItem();
            if(addItem == null) {...}

            if(addItem.isInventoryItemOffensive()) {
                APVal += addItem.getItemUseTypeValue();
                APValLabel.setText(String.valueOf(APVal));
                notify(String.valueOf(APVal), InventoryObserver.InventoryEvent.UPDATED_AP);

                ProfileManager.getInstance().setProperty("currentPlayerAP", APVal);

                if(addItem.isInventoryItemOffensiveWand()) {
                    notify(String.valueOf(addItem.getItemUseTypeValue()), InventoryObserver.InventoryEvent.ADD_WAND_AP);
                }
            } else if(addItem.isInventoryItemDefensive()) {...}
            break;
        case REMOVED_ITEM:
            InventoryItem removeItem = slot.getTopInventoryItem();
            if(removeItem == null) {...}

            if(removeItem.isInventoryItemOffensive()) {
                APVal -= removeItem.getItemUseTypeValue();
                APValLabel.setText(String.valueOf(APVal));
                notify(String.valueOf(APVal), InventoryObserver.InventoryEvent.UPDATED_AP);

                ProfileManager.getInstance().setProperty("currentPlayerAP", APVal);

                if(removeItem.isInventoryItemOffensiveWand()) {
                    notify(String.valueOf(removeItem.getItemUseTypeValue()), InventoryObserver.InventoryEvent.REMOVE_WAND_AP);
                }
            } else if(removeItem.isInventoryItemDefensive()) {...}
            break;
        default:
            break;
    }
}

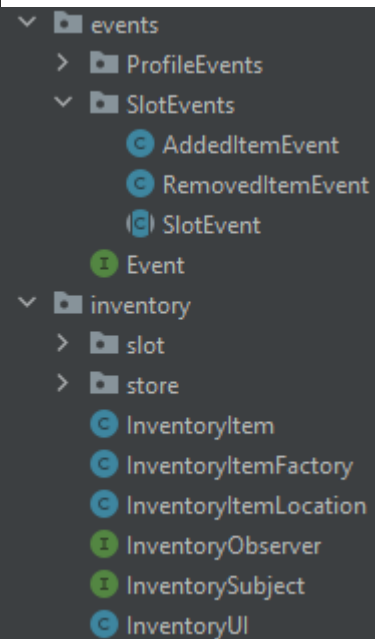
```

InventoryUI after:

```
public class AddedItemEvent extends SlotEvent{
    1 usage  ▲ kostashomson
    @Override
    public void PerformInventoryUIAction(InventorySlot slot, InventoryUI inventoryUI) {
        int APVal = inventoryUI.getAPVal();
        Label APValLabel = inventoryUI.getAPValLabel();
        int DPVal = inventoryUI.getDPVal();
        Label DPValLabel = inventoryUI.getDPValLabel();
        InventoryItem addItem = slot.getTopInventoryItem();
        if(addItem != null) {
            return;
        }
        if(addItem.isInventoryItemOffensive()) {
            APVal += addItem.getItemUseTypeValue();
            APValLabel.setText(String.valueOf(APVal));
            inventoryUI.notify(String.valueOf(APVal), InventoryObserver.InventoryEvent.UPDATED_AP);
            ProfileManager.getInstance().setProperty("currentPlayerAP", APVal);
            if(addItem.isInventoryItemOffensiveWand()) {
                inventoryUI.notify(String.valueOf(addItem.getItemUseTypeValue()), InventoryObserver.InventoryEvent.ADD_WAND_AP);
            }
        } else if(addItem.isInventoryItemDefensive()) {
            DPVal += addItem.getItemUseTypeValue();
            DPValLabel.setText(String.valueOf(DPVal));
            inventoryUI.notify(String.valueOf(DPVal), InventoryObserver.InventoryEvent.UPDATED_DP);
            ProfileManager.getInstance().setProperty("currentPlayerDP", DPVal);
        }
    }
}

1 usage  ▲ kostashomson
@Override
public void PerformStoreInventoryUIAction(InventorySlot slot, StoreInventoryUI storeInventoryUI) {
    int tradeInVal = storeInventoryUI.getTradeInVal();
    int fullVal = storeInventoryUI.getFullVal();
    Label sellTotalLabel = storeInventoryUI.getSellTotalLabel();
    Label buyTotalLabel = storeInventoryUI.getBuyTotalLabel();

    //moving from player inventory to store inventory to sell
    if(slot.getTopInventoryItem().getName().equalsIgnoreCase(InventoryUI.PLAYER_INVENTORY) &&
        slot.getName().equalsIgnoreCase(InventoryUI.STORE_INVENTORY)) {
        tradeInVal += slot.getTopInventoryItem().getTradeValue();
        sellTotalLabel.setText("SELL : " + tradeInVal + "GP");
    }
    //moving from store inventory to player inventory to buy
    if(slot.getTopInventoryItem().getName().equalsIgnoreCase(InventoryUI.STORE_INVENTORY) &&
        slot.getName().equalsIgnoreCase(InventoryUI.PLAYER_INVENTORY)) {
        fullVal += slot.getTopInventoryItem().getItemValue();
        buyTotalLabel.setText("BUY : " + fullVal + "GP");
    }
}
}
```



```
@Override
public void onNotify(InventorySlot slot, SlotEvent event) {
    event.PerformInventoryUIAction(slot, inventoryUI: this);
}
```



```

public class RemovedItemEvent extends SlotEvent{
    1 usage  ▲ kostasthompson
    @Override
    public void PerformInventoryUIAction(InventorySlot slot, InventoryUI inventoryUI) {
        int APVal = inventoryUI.getAPVal();
        Label APValLabel = inventoryUI.getAPValLabel();
        int DPVal = inventoryUI.getDPVal();
        Label DPValLabel = inventoryUI.getDPValLabel();
        InventoryItem removeItem = slot.getTopInventoryItem();
        if(removeItem == null) {
            return;
        }
        if(removeItem.isInventoryItemOffensive()) {
            APVal -= removeItem.getItemUseTypeValue();
            APValLabel.setText(String.valueOf(APVal));
            inventoryUI.notify(String.valueOf(APVal), InventoryObserver.InventoryEvent.UPDATED_AP);
            ProfileManager.getInstance().setProperty("currentPlayerAP", APVal);
            if(removeItem.isInventoryItemOffensiveWand()) {
                inventoryUI.notify(String.valueOf(removeItem.getItemUseTypeValue()), InventoryObserver.InventoryEvent.REMOVE_WAND_AP);
            }
        } else if(removeItem.isInventoryItemDefensive()) {
            DPVal -= removeItem.getItemUseTypeValue();
            DPValLabel.setText(String.valueOf(DPVal));
            inventoryUI.notify(String.valueOf(DPVal), InventoryObserver.InventoryEvent.UPDATED_DP);
            ProfileManager.getInstance().setProperty("currentPlayerDP", DPVal);
        }
    }
}

1 usage  ▲ kostasthompson
@Override
public void PerformStoreInventoryUIAction(InventorySlot slot, StoreInventoryUI storeInventoryUI) {
    int tradeInVal = storeInventoryUI.getTradeInVal();
    int fullValue = storeInventoryUI.getFullValue();
    Label sellTotalLabel = storeInventoryUI.getSellTotalLabel();
    Label buyTotalLabel = storeInventoryUI.getBuyTotalLabel();
    if(slot.getTopInventoryItem().getName().equalsIgnoreCase(InventoryUI.PLAYER_INVENTORY) && slot.getName().equalsIgnoreCase(InventoryUI.STORE_INVENTORY)) {
        tradeInVal -= slot.getTopInventoryItem().getTradeValue();
        sellTotalLabel.setText("SELL : " + tradeInVal + "GP");
    }
    if(slot.getTopInventoryItem().getName().equalsIgnoreCase(InventoryUI.STORE_INVENTORY) && slot.getName().equalsIgnoreCase(InventoryUI.PLAYER_INVENTORY)) {
        fullValue -= slot.getTopInventoryItem().getItemValue();
        buyTotalLabel.setText("BUY : " + fullValue + "GP");
    }
}
}

public abstract class SlotEvent implements Event {
    1 usage  2 implementations  ▲ kostasthompson
    public abstract void PerformInventoryUIAction(InventorySlot slot, InventoryUI inventoryUI);
    1 usage  2 implementations  ▲ kostasthompson
    public abstract void PerformStoreInventoryUIAction(InventorySlot slot, StoreInventoryUI storeInventoryUI);
}

```