**Projet Java**

# Jeu d’aventure

# Gestionnaire de version

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Auteur | Modification(s) | Version | Statut |
| 14/03/2020 | Xavier | Création du document, mise en page et plan | V1.0 | FIN |
| 14/03/2020 | Xavier | Différence scénario | V1.1 | FIN |
| 15/03/2020 | Abd-Rahim | Diagramme UML | V1.2 | FIN |
| 15/03/2020 | Abd-Rahim | Description des tests automatisés mis en place | V1.3 | FIN |
| 15/03/2020 | Abd-Rahim | Gestion de projet | V1.4 | FIN |

Sommaire

[Jeu d’aventure 1](#_Toc35252158)

[Gestionnaire de version 3](#_Toc35252159)

[Scénario 5](#_Toc35252160)

[Zones 5](#_Toc35252161)

[Eléments 5](#_Toc35252162)

[Personnages 6](#_Toc35252163)

[Enigmes 7](#_Toc35252164)

[Pièges 7](#_Toc35252165)

[Différence entre le scénario envisagé et le scénario programmé 8](#_Toc35252166)

[Solution du point de vue de la conception et de la programmation 10](#_Toc35252167)

[Diagramme UML 11](#_Toc35252168)

[Description des tests automatisés mis en place 12](#_Toc35252169)

[Gestion de projet 13](#_Toc35252170)

[Planification 13](#_Toc35252171)

[Planning initial et répartition des tâches 14](#_Toc35252172)

[Planning final 15](#_Toc35252173)

[ANNEXES 16](#_Toc35252174)

[WBS (Work Breakdown Structure) 16](#_Toc35252175)

# Scénario

On arrive dans la zone principale du jeu et une bombe vient d’être amorcé. Le compte à rebours est lancé. Le but du jeu est de désamorcer la bombe en trouvant un code à trois chiffres, cependant le joueur dispose de seulement cinq minutes avant que cette dernière explose. Mais comment trouver ce fameux code à trois chiffres ?

Le poseur de bombe étant joueur, il a laissé une machine permettant de gonfler trois ballons. En effet, en réussissant à l’activer, chaque ballon gonflé laissera apparaitre un chiffre, et la combinaison de ces derniers donnera le code pour désactiver la bombe.

Toutefois, pour mettre en marche cette machine, il faudra payer trois pièces. Le joueur principal gagnera une pièce à chaque petite énigme réussie.

## Zones

On compte 4 zones.

La zone principale où le joueur apparaît quand il lance le jeu, et trois autres zones, contenant chacune une petite énigme à résoudre.

## Eléments

Dans la zone principale, on trouve la bombe avec le compte à rebours, et la machine avec les trois ballons dégonflés dessus. Pour fonctionner, la machine nécessite trois pièces.

Dans les trois autres zones, on pourra gagner une pièce par zone.

## Personnages

Le joueur principal qui doit parvenir à désamorcer la bombe.



Figure 1- Sprite du joueur principal

Trois complices du poseur de bombe qui nous attendent pour résoudre leurs mini énigmes et nous donner une pièce si l’on réussit à trouver la solution.

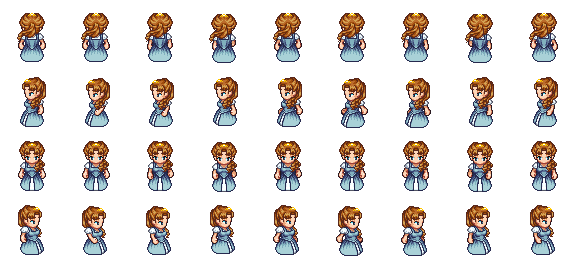


Figure 2 - Sprite général des complices du poseur de bombe

## Enigmes

Trois mini énigmes à résoudre :

- « Quand j’avais 6 ans, ma sœur avait la moitié de mon âge. Aujourd’hui j’ai 50 ans, quel âge a ma sœur ? »

-Verticalement : les deuxièmes lettres de chaque mot se suivent avec 2 rangs d’écart. Horizontalement : les premières lettres de chaque mot se suivent avec 3 rangs d’écart. Donner la suite des 3 lettres.

CFJ

YHM

LJY

TLP

SWO ? MXI JNQ GVZ

-" Quel est le plus grand chiffre du monde ? "

## Pièges

Si le joueur répond par une mauvaise réponse aux complices ces derniers peuvent téléporter le joueur à la salle de départ pour lui faire perdre du temps.

# Différence entre le scénario envisagé et le scénario programmé

Le scénario programmé correspond exactement au scénario envisagé. Afin de le rendre plus compétitif et attrayant, il nous est venu à l’idée de mettre en place un système de leaderboard (classement).

Pour cela, nous avons rajouté au début du jeu l’obligation de fournir un nom, qui correspond au pseudo du joueur.

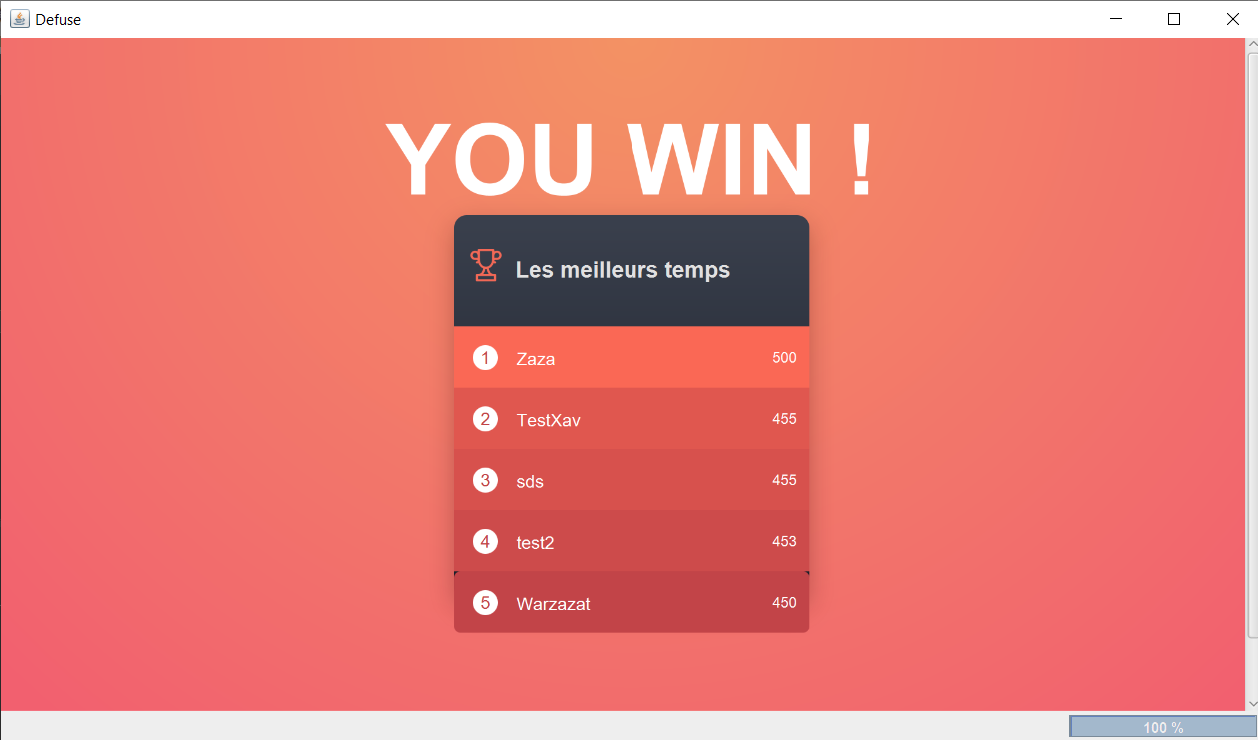
Le but n’est donc plus seulement de désamorcer la bombe en moins de cinq minutes mais d’effectuer le meilleur temps possible.

Lorsque le joueur parvient à désamorcer la bombe, alors son temps ainsi que son pseudo sont envoyés sur notre base de données et un écran affichant les cinq meilleurs temps permet de nous situer.

Pour communiquer avec la base de données, il nous fallait un SGBD (Système de Gestion de Base de Données). Nous nous sommes orientés vers l’API JDBC qui est fournit avec Java car celle-ci répondait parfaitement à ce que l’on recherchait.

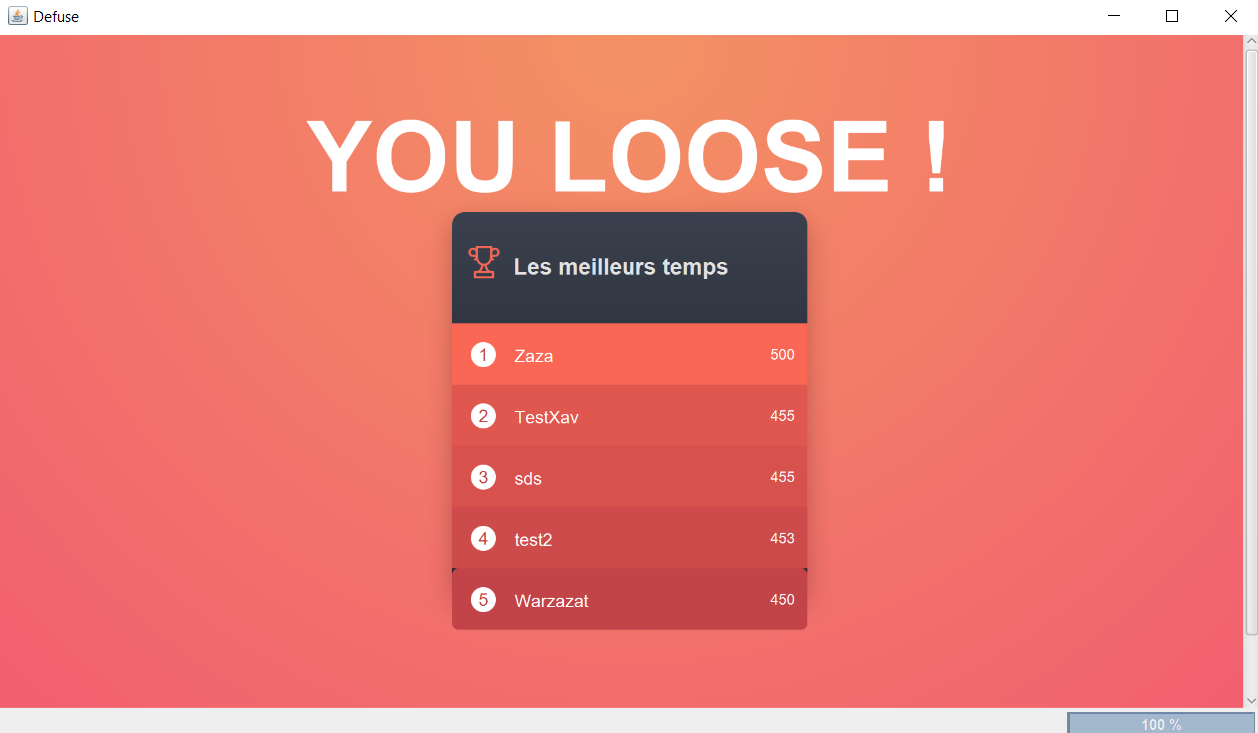
De plus, JDBC bénéficie des avantages de Java, dont la portabilité du code, ce qui lui vaut en plus d'être indépendant de la base de données d'être indépendant de la plate-forme sur laquelle elle s'exécute.

Par exemple, voici la page qui s’affichera si vous réussissez à finir le jeu avant les cinq minutes :



Evidemment, une connexion internet est nécessaire pour permettre l’envoi et la récupération des données.

Voici la page qui s’affichera si vous ne réussissez pas à finir le jeu avant les cinq minutes où si vous vous trompez dans le code de la bombe :



Comment fonctionnent les scores ?

Imaginons que vous désamorcer la bombe à 3min 31. Votre score sera alors de 331.

La minute est multipliée par 100 et les secondes y sont ajoutées.

# Solution du point de vue de la conception et de la programmation

Le Projet a été construit à partir de la source de jeu sommaire et modifiant en fonction des besoins de notre jeu.

Pour la classe gui par exemple, elle reste presque la même mais nous avons rajouté une fonction « creerGuiGame » qui affiche la fenêtre finale avec le leaderboard, et un ajout au classement en fonction de la victoire.

Concernant la classe Commande, nous avons gardé le même modèle puis rajouter les commandes suivantes :

* Demander
* Credit
* Reponse
* Defuse
* Gonfler
* Retour
* Solution

Pour comprendre la suite il faut se visualiser le jeu, tout d’abord on incarne un joueur qui a un nom qui sera ensuite utilisé dans le leaderboard et un nombre de pièces.

La classe Personne dont vont hériter les classes Joueur et malfaiteur, possède un constructeur avec un nom et un getter.

La classe Joueur qui hérite de Personne, nous appelons donc le constructeur de la super classe en rajoutant un nombre de pièces égale à 0, elle aura des fonctions d’ajout des pièces et permettra de connaitre le nombre de pièces disponible.

La gestion des énigmes se fait dans sa classe Enigme, qui retourne les questions, valide ou pas les réponses, et donne un statut si les énigmes ont déjà été résolu ou pas.

Les énigmes sont posées par des malfaiteurs qui sont présent dans 3 zones de la carte. Les malfaiteurs héritent de la classe Personne avec un appel du constructeur de la super classe et l’ajout en paramètre d’une énigme. Elle possède aussi un getter afin de renvoyer l’énigme du malfaiteur concerné.

La classe Zone elle permet de gérer l’image de la zone, le malfaiteur affiché, une description brève de la zone ainsi que les sorties les plus proche à l’aide de hashMap qui contient toutes les sorties pour une Zone.

La classe bombe est la classe qui donne du dynamisme à notre jeu, car elle donne une limite de temps et possède le code de désamorçage de la bombe. De plus c’est elle qui va déclencher la perte de la partie lorsque le temps sera écoulé.

Jeu est une des classes les plus importante car elle fait la jointure entre les toutes les autres classes , elle va servir à affecter chaque zone avec son malfaiteur ainsi que le nombre de zone, et toutes les sorties possibles pour chaque zone. Toutes les fonctions qui servent au traitement des commandes tapés par l’utilisateur sont disponible.

Gonfler va gonfler les ballons afin de récupérer le code tout en vérifiant toutes les conditions nécessaires.

La méthode « Defuse » va permettre au joueur de terminer le jeu, en cas de mauvaise réponse il perdra. En cas de victoire on sauvegardera le nom du joueur afin de l’enregistrer dans notre base de données pour notre leaderboard.

La méthode « Reponse » permettra à l’utilisateur de répondre aux différentes énigmes, en cas de bonne réponse on ajoutera une pièce au joueur, si l’énigme a déjà été résolu le gui affiche un message et si la réponse est mauvaise on va téléporter le joueur à la zone de départ afin de lui faire perdre du temps.

La méthode « allerEn » a été modifié par rapport à la fonction de base, s’il n’y a pas de solution affiche une erreur. Et pour chaque appel de fonction garde la zone précédente dans une ArrayList de zone afin de l’utiliser ultérieurement.

On va donc utiliser cette Arraylist dans la fonction retour, si cette ArrayList est vide on affiche un message d’erreur sinon on fait des retours en arrière et on enlève cette zone de la liste.

La fonction « Solution » qui est demandé dans la consigne du projet a été possible à l’aide de la bibliothèque java.io. La fonction va donc lire le fichier « Solution.txt » présent à la base du projet, ligne par ligne. Tant qu’il y a des lignes on appelle la fonction « traiterCommande » avec les bonnes commandes.

On possède aussi deux autres packages, images qui possède les images du jeu et test, qui contient tous les test unitaires généré grâce à l’outil Junit 5 pour chaque classe du jeu.

# Diagramme UML

# Description des tests automatisés mis en place

Tout programme doit être testé pour vérifier la fiabilité des fonctions et assurer un bon fonctionnement.

Les tests ont été réalisé sur 10 classes :

* Bombe
* Commande
* Enigme
* GUI
* ImagePanel
* Jeu
* Joueur
* Malfaiteur
* Personne
* Zone

La création des fichiers test ont été simple, à l’aide de l’IDE Eclipse. Eclipse propose sur un clic droit sur la classe que l’on veut tester, de créer un fichier ***« JUnit Test Case »***. Ensuite je peux décider de créer le fichier avec des commentaires ainsi qu’une fonction ***BeforsetUpBeforeClass()*** qui s’exécutera avant toutes les fonctions test, ce qui permet d’initialiser une variable qui sera ensuite utilisé dans les fonctions test, cela évite de créer plusieurs fois la même variables. JUnit Test Case nous créer aussi toutes les fonctions à tester, il ne reste plus qu’à remplir le contenu pour effectuer le test.

Pour tester les fonctions, il suffit de vérifier le contenu qu’elle retourne. Exemple, pour la classe test JoueurTest, nous voulons savoir si la fonction qui met à jour le nombre de pièces fonctionne. Il a donc fallu appeler la méthode qui modifie le nombre de pièce en lui passant en paramètre « 19 ». Ensuite à l’aide de ***« assertEquals(var1, var2) »***, il faut transmettre deux paramètres qui seront comparés pour vérifier qu’ils sont identiques. Ici, la première variable correspond aux nombres de pièces qui sera récupéré à l’aide d’une méthode et la seconde variable sera l’entier 19. Si les deux variables sont identiques alors la fonction test affichera un message dans la console « Succès : Modification du nombre de pièce » mais si elles ne sont pas identiques rien ne s’affichera, de plus JUnit indiquera une erreur.

Ce procédé s’appliquera à tous les autres tests, la chose qui variera sera seulement est le type (String, Int, Boolean …).

# Gestion de projet

## Planification

Concernant l’organisation et le planning, une première réunion a été faites pour définir l’organisation.

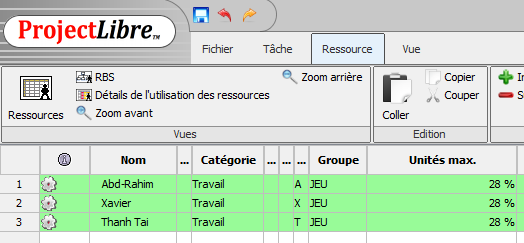
Dans un premier temps, la gestion de projet a été confié à Abd-Rahim Boumkassar.

Il avait donc pour objectif d’identifier les tâches à réaliser à l’aide d’une modélisation WBS (fournis en annexe), puis de réunir le reste du groupe pour que l’on se répartisses les tâches.

Une fois tout cela fait, il a donc dû établir un planning qui devrait prendre en compte les ressources ainsi que les disponibilités mais aussi l’effort nécessaire pour une tâche. Il a donc utilisé ***ProjectLibre*** pour établir un Gantt qui prenait en compte les facteurs cités précédemment.

Il a été difficile de trouver la disponibilité des ressources étant donné qu’il fallait prendre en compte le travail à faire en plus du projet ainsi que les cours, et aussi les choses personnelles. Après réflexion et consultation avec l’équipe, nous en sommes venus à dire que la disponibilité lors des jours de cours serait de 20% et que le weekend le taux de disponibilité serait de 50%. Ce qui nous donne une moyenne de 28% de disponibilité.

Avec cela il a donc pu réaliser le planning en prenant en compte les ressources.



## Planning initial et répartition des tâches

Interprétation du planning :

* Nous voyons sur le planning initial que le projet est constitué d’un premier jalon « ***lancement »***, qui comme dit précédemment a eu pour but de se réunir pour identifier et répartir les tâches.
* La tâche « ***Scénario*** » avait pour objectif de trouver un scénario pour le jeu, tout le monde à participer, Xavier Maisse à trouver le scénario puis Thanh Tai Tran et moi-même avons trouvé des énigmes pour le jeu.
* Le jalon « ***Rendre Scénario*** », comme son nom l’indique correspond à rendre tout simplement le scénario.
* Les deux tâches « ***Modélisation UML*** » et « ***Création des Images*** » suivent ce jalon, elle commence simultanément. La tâche de l’UML m’a été confié et pour la création des images du jeu, Xavier Maisse s’en est occupé.
* La tâche « ***développement*** » pourra débuter une fois que les deux tâches précédentes seront terminées. Cette tâche qui est la plus conséquente a été confié à Xavier Maisse ainsi que Thanh Tai Tran. Xavier c’est occupé du jeu en général et Thanh Tai c’est lui occupé d’ajouter certaines options.
* Le développement est ensuite suivi de la tâche « ***Test*** » qui pourra débuter une fois le développement du jeu terminer. Les tests ont été réalisé par Abd-Rahim Boumkassar.
* Ensuite, tout le monde participe à la tâche « ***documentation*** » suivi de la réalisation de la tâche « ***rapport*** ».

Comme nous pouvons le voir sur ***le planning initial***, nous avons une très large marge avant le jalon final qui correspond à la fin du projet.

Mais le planning a dû évoluer car sur certaines tâches, il y a eu du retard.

## Planning final

Comme nous pouvons le voir, il y a un décalage entre le planning initial, ici représenté avec une barre grise et le planning final.

Cela est dû à un retard dans le développement et les tests, mais cela n’a pas impacté la date pour rendre le projet. Nous conservons tout de même une marge qui va nous permettre de faire des vérifications en plus.

# ANNEXES

## WBS (Work Breakdown Structure)

* C’est un organigramme des tâches du projet, cela aide pour identifier les tâches à réaliser pour un projet et facilite la planification.