Compte rendu général sur le projet ACL 2019-2020

Groupe EzMaze – François-Marshall Marius, Guehi Carl, LeBihan William, Poujon Adrien

BACKLOG GLOBAL

1. Le personnage

- Déplacer sur plateau
- Caractéristique du personnage
- Peut attaquer un monstre au contact
- Peut attaquer un monstre sur une case adjacente

2. Le labyrinthe

- On ne peut pas traverser les murs
- Case trésor pour gagner, finir le jeu
- Case de piège qui infligent des dégâts
- Cases magiques Cases de téléportation

3. Les monstres

- Instancier des monstres sur le plateau
- Infliger des dégâts au contact
- Se déplacer sur le plateau aléatoirement

4. Caractéristiques du jeu

- Sélection niveau de difficulté
- Sauvegarder partie
- Historique scores
- Chronomètre

BACKLOG : SPRINT 1

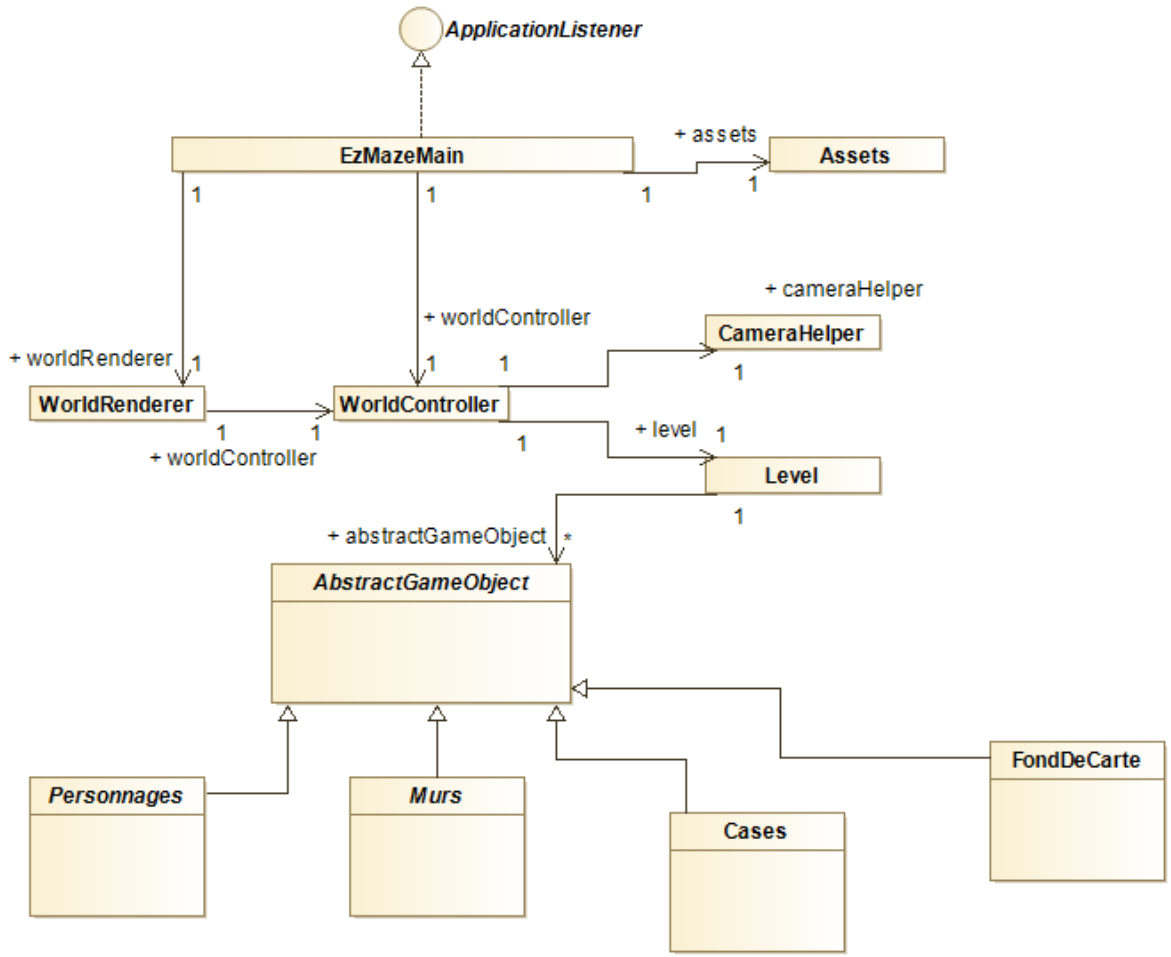
Pour ce premier sprint, on se propose d'abord de mettre en place une interface graphique. Celle-ci permettra d'afficher le labyrinthe en tant que tel avec ses murs. Le joueur pourra voir son personnage apparaître dans le labyrinthe, sur une case prédéfinie, et pourra alors se déplacer dans le labyrinthe avec une combinaison de touches. Il ne pourra pas traverser les murs. On ajoutera une case d'arrivée, appelée EzCase, qui permet au joueur de finir le jeu : il s'agit du but du jeu. En outre, on mettra en place un ou plusieurs monstres au sein du labyrinthe. Ces derniers provoqueront une remise à zéros du jeu s'ils sont rencontrés par le joueur.

SPRINT 1

On définit ci-après, par ordre de priorité les caractéristiques à mettre en place.

1. Déplacer sur plateau
2. On ne peut pas traverser les murs
3. Case trésor pour gagner, finir le jeu
4. Instancier des monstres et déplacement

Diagramme de classe pré-sprint 1



Rétrospective

A la fin de ce premier sprint, on peut voir que seules trois fonctionnalités sur quatre ont pu être effectivement implémentées. Cela traduit une ambition trop importante de notre part et nécessitera donc une correction de nos projections pour la suite du projet.

En outre, on a aussi pu voir que l'organisation pouvait poser problème. En particulier, nous n'avons pas su profiter pleinement de l'outil de travail de groupe que constitue GitHub. En effet, notre manque d'expérience avec la plateforme de travail nous a grandement ralenti dans l'avancement de notre projet. A plusieurs reprises, pour éviter les difficultés, nous sommes passés par l'utilisation d'un système D par clef USB. Dans la suite du projet, nous devrions être plus apte à utiliser GitHub et cela pourra faciliter notre travail.

D'autre part, une fonctionnalité additionnelle non demandée a été mise en place au cours de ce premier sprint. Il s'agit de la possibilité de zoomer et de déplacer la caméra sur le plateau de jeu. Pour cela, il suffit d'appuyer sur entrée et de déplacer la caméra avec les flèches directionnelles du clavier. Le zoom in se fait avec la touche 'I' et le zoom Out avec la touche 'O'. 'U' permet de réinitialiser la vue de la caméra. Au

regard de l'absence des autres fonctionnalités du programme, on peut se dire que l'on voit ici la marque d'un certain manque de respect de l'organisation prévue, qui devra aussi être surveillé pour le sprint 2.

Cependant, le temps passé à débloquer le build avec gradle et à travailler sur les classes et leur organisation nous permettra probablement de gagner du temps pour la suite du projet. De plus, puisque le travail sur Github était difficile pour nous, cela nous a poussé à travailler plus souvent les uns avec les autres, sur une même machine, ce qui permet à chacun d'avoir une vision d'ensemble du code, qu'il faudra préserver dans la suite du projet.

BACKLOG : Sprint 2

Pour ce deuxième sprint, on se propose d'abord d'améliorer le système de monstres. On voudrait que le personnage puisse attaquer les monstres présents sur le plateau, on voudrait aussi ajouter un système résurrection du personnage (on a la droit à trois essais pour rejoindre la case de fin avant de faire *game over*). On voudrait en outre ajouter des cases spécifiques, en particulier, on voudrait introduire une case de téléportation ainsi qu'une case de boue qui ralentirait le joueur.

En outre, on voudrait aussi mettre en place une interface graphique utilisateur (GUI) qui permette d'afficher le temps écoulé depuis le début du jeu, le nombre de monstres battus ainsi que le nombre de résurrections restantes pour le personnage.

SPRINT 1 - Héritage des objectifs non-atteints

1. Il ne se passe actuellement rien lorsque monstre et joueur se rencontrent ;
2. Les monstres ne sont pas actuellement capables de se déplacer ;

SPRINT 2

On définit ci-après, par ordre de priorité les caractéristiques à mettre en place.

1. Attaquer les monstres
2. Système de résurrection
3. Case de téléportation
4. Case de boue
5. GUI

Diagramme de classes pré-sprint 2

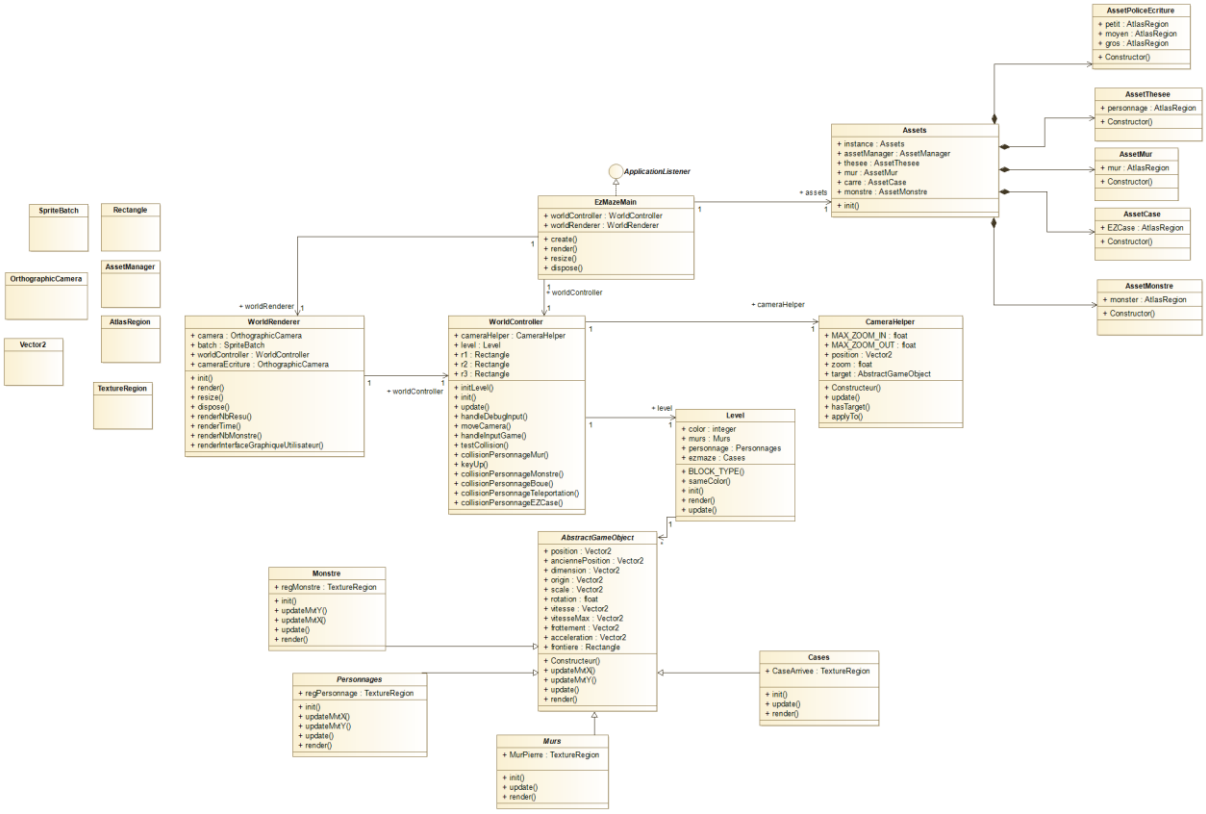
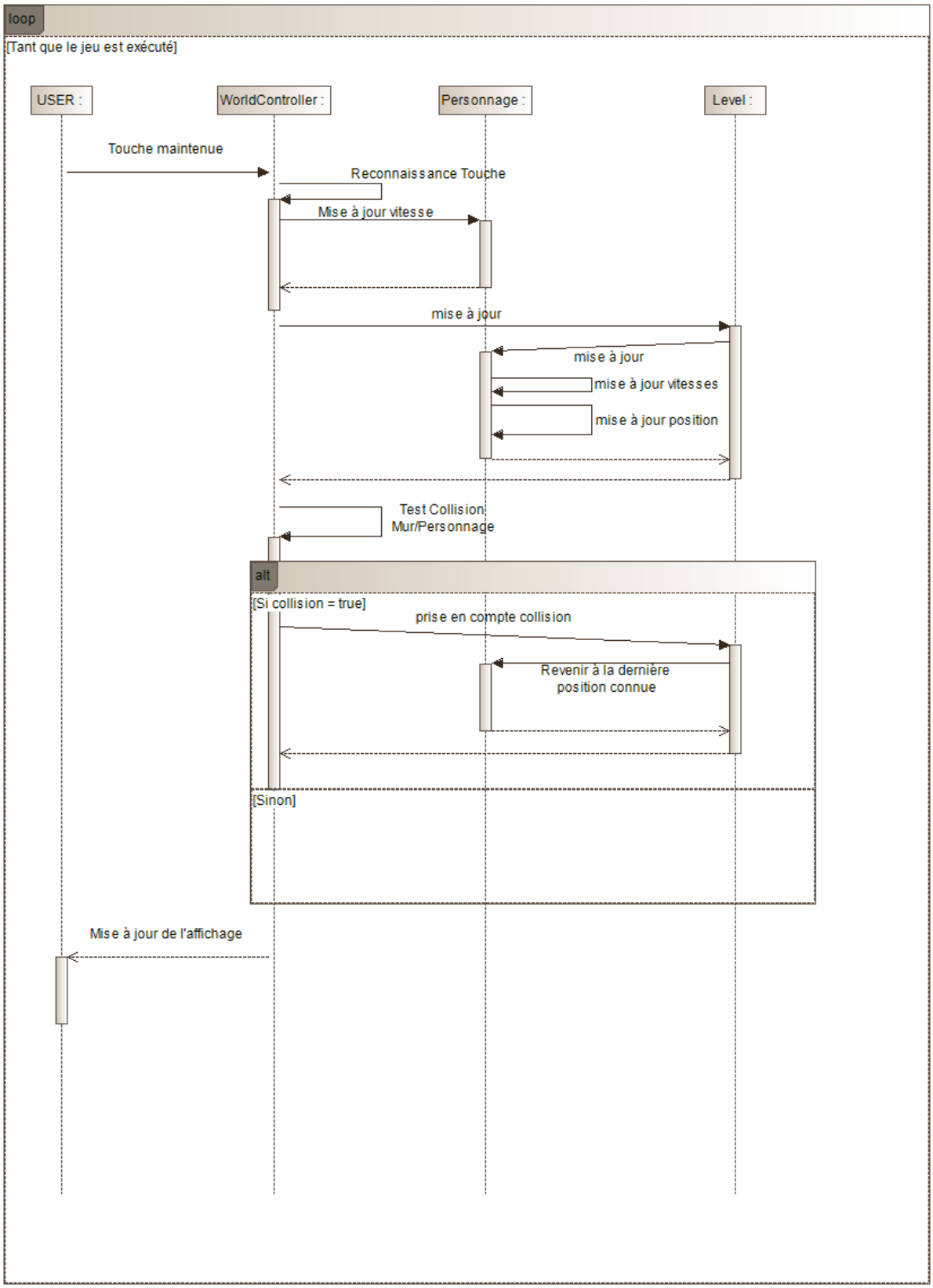


Diagramme de Séquence : Mouvement du personnage


loop
[Tant que le jeu est exécuté]
USER :
WorldController :
Personnage :
Level :
Touche maintenue
Reconnaissance Touche
Mise à jour vitesse
mise à jour
mise à jour
mise à jour vitesses
mise à jour position
Test Collision
Mur/Personnage
alt
[Si collision = true]
prise en compte collision
Revenir à la dernière
position connue
[Sinon]
Mise à jour de l'affichage

GUI

On ajoute : Une inner-class *policeEcriture* à la class Assets. Et on définit 3 typographies (petit, moyen, gros). Pour utiliser ce nouvel Asset, On ajoute un attribut *OrthographicCamera cameraEcriture* dans notre WorldRenderer. On y associe autant de méthodes que nécessaire : une méthode pour l'affichage de chaque élément.

1. *renderNbResu()*
2. *renderTime()*
3. *renderNbMonstresTues()*

On compilera toutes ces méthodes dans une méthode *renderInterfaceGraphiqueUtilisateur()*. Celle-ci sera appelée dans la méthode *render()* de la même classe *WorldRenderer*.

Attaque Joueur

On ajoute une énumération d'état comme *attaque* et *paisible* dans la classe *Personnage*. Ces énumérations nous permettrons de connaître l'état du joueur. L'action d'attaquer sera traitée pour elle-même, comme le déplacement, dans le *WorldController*.

Systeme de résurection

On dira que lorsque le joueur meurt (pour l'instant il s'agit d'une simple collision avec un monstre), il est renvoyé au départ du labyrinthe et perd une vie. Il ne s'agit pas d'un système de points de vie (puisque le joueur meurt) mais bien d'un système de résurrection : on repart du départ.

1. Le nombre de résurrection sera géré par une variable de résurection ajoutée à la classe des *Constantes*.
2. Le compteur est mis à jour pour le *WorldController* lorsque le joueur meurt.
3. Lorsque le compteur est à zéros on affiche un message de défaite.

Case de téléportation

On dira que lorsque le joueur marche sur la case de téléportation, il est envoyé sur une autre case visuellement identifiable. On implémentera un labyrinthe spécifique où l'utilisation d'une telle case est nécessaire si l'on souhaite atteindre l'arrivée.

1. Ajouter une nouvel asset dans l'inner-class *AssetCase* de la classe *Case*.
2. Gérer l'effet lorsque le personnage se trouve sur la case dans le *WorldController*.

Case de boue

On dira que lorsque le joueur marche sur la case de boue, il est grandement ralenti.

1. Ajouter une nouvel asset dans l'inner-class *AssetCase* de la classe *Case*.
2. Gérer l'effet lorsque le personnage se trouve sur la case dans le *WorldController*.

Rétrospective

Lors du sprint 2, nous avons disposé de plus de temps. Malgré les contrariétés rencontrées au cours du sprint 1 (objectifs pas tous atteints, difficultés à utiliser Git) nous avons tout de même décidé de prévoir l'implémentation de très nombreuses nouvelles fonctionnalités. Le temps disponible nous a permis d'effectivement mettre en place l'ensemble des objectifs que nous nous étions fixés. En cela, nous sommes particulièrement satisfaits de notre travail. En outre, nous avons aussi pu laisser vaquer notre imagination pour introduire des concepts imprévus : notamment les fantômes et les caisses que l'on peut pousser.

On regrette cependant que certaines de nos implémentations ne répondent pas exactement à ce que nous avions imaginé. En particulier, nous n'avons pas mis en place d'attaque au corps à corps (nous avons préféré nous tourner vers un système de projectiles) qui auraient pues apporter un plus au jeu (cependant la poussée de caisse n'est en fait pas très éloignée d'une forme d'attaque au corps à corps). D'autre part et quant à l'implémentation des cases téléportations, on est encore peu satisfait du mode de fonctionnement mis en place. En effet, il n'est pour le moment pas possible de choisir lors de la conception du niveau vers quelle case de téléportation de destination une case de téléportation de départ nous conduira (chaque case de téléportation conduit en effet toujours à la case de destination qui se trouve au même rang dans une liste donnée). Enfin, il est à noter que nous n'avons pas su implémenter une IA pour le monstre du labyrinthe et que celui-ci se contente pour l'instant de déambuler le long des murs.

Même si nous avons par ailleurs continuer à travailler en groupe, nous avons mieux su tirer profit de Git au cours de ce sprint. Nous avons développé une organisation combinant GitHub avec Facebook, de sorte que chaque push vers le dépôt à distance soit notifié à tous les membres du groupe afin que nul n'oublie de pull lorsque nécessaire.

Dans d'éventuels sprint suivants, il faudrait que les spécifications du backlog soient mieux détaillées. On gagnerait en effet en temps et en efficacité si la précision était accrue dès le début et nous fixait des objectifs clairs et détaillés. Cependant, nous avons aimés pouvoir bénéficier d'une certaine liberté d'imagination lors du développement des différents aspects du jeu et il nous semble que cela reste fécond.

Dans l'idéal il faudrait donc trouver un compromis entre clarté, précision et degré de liberté pour que le jeu puisse se développer dans une direction maîtrisée tout en laissant place à un certain degré d'indétermination qui peut se révéler intéressant.

BACKLOG : Sprint 3

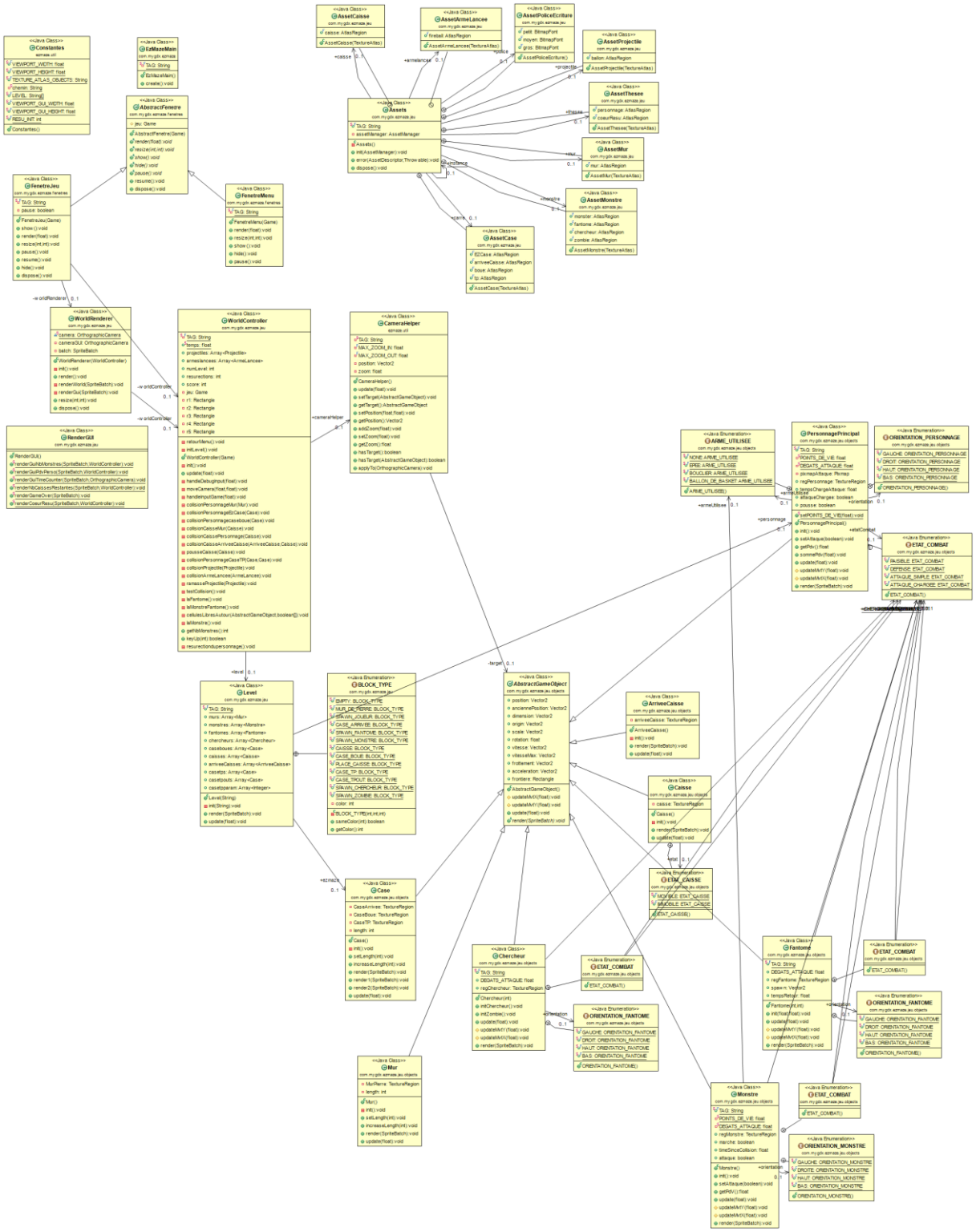
Pour ce troisième sprint, on se propose de mettre en place de nouvelles entités ennemis qui seraient dotés d'une meilleure intelligence artificielle. On voudrait aussi revenir sur l'implémentation des cases de téléportation de sorte que l'on puisse enfin paramétrer leur case de destination comme il nous plait. Pour finir, nous souhaiterions compléter l'interface utilisateur d'un menu qui pourrait permettre de sélectionner le niveau à jouer.

SPRINT 3

On définit ci-après, par ordre de priorité les caractéristiques à mettre en place.

1. Nouveaux monstres Intelligents
2. Paramétrage des cases TP
3. Implémentation d'un menu de sélection des niveaux

Diagramme de classes – post-sprint 3



Monstres Intelligents

Dans l'implémentation des monstres intelligents, on s'inspirera des intelligences artificielles utilisées dans PacMan. On implémentera deux monstres : une tête chercheur qui se contentera de poursuivre le joueur, infligera peu de dégâts, et ne pourra pas être attaquée ; un zombie qui se déplacera très rapidement et pourra tuer

le joueur quasi-instantanément. Le zombie sera tuable via l'orbe de feu et l'emplacement de sa mort sera alors hanté par un fantôme.

Paramétrage case TP

On ajoutera un système de fichiers textes lus aux chargement des niveaux. Le fichier texte comportera les informations nécessaires pour décrire les liens entre les cases de téléportation.

Menu

L'implémentation du menu sera la plus difficile. Il va falloir bouleverser l'organisation de nos classes d'affichage afin de pouvoir implémenter différents écrans de jeu. Cela passera par la mise en place d'une fenêtre principale composée d'un écran de jeu et d'un écran de menu. On pourra passer de l'un à l'autre via la pression de la touche *échap*. Il nous faudra aussi introduire un système de sélection sur le menu.

Rétrospective

Lors de ce sprint particulièrement court, nous avons pu implémenter pleinement deux objectifs sur les trois objectifs fixés. Nous ne sommes cependant pas trop critiques sur notre résultat car nous savions que le temps imparti ne nous permettrait probablement pas de venir à bout de tous les changements structurels engagés par l'implémentation d'un menu.

Si les monstres à intelligence artificielle et le paramétrage des cases de téléportation ont donc bien été mis en place, c'est du côté de l'implémentation du menu que nous n'avons pas pu atteindre nos objectifs. La mise en place de plusieurs écrans s'est révélée moins évidente que prévu et il a fallu se replonger dans la lecture de différents wiki et autres tutoriels sur libgdx pour avoir une nette idée de ce qu'il fallait faire. L'écran correspondant au menu est bien implémenté, et l'on peut effectivement passer du menu au jeu en pressant *échap*. Cependant, le menu est vide et n'a à ce jour pas de réelle utilité.

Addendum général sur la répartition des tâches

Concernant la répartition des tâches, il est à noter que celles-ci étaient réparties en amont de chaque sprint. Chaque personne se voyait alors attribuer l’implémentation d’un point donné du backlog. Cependant en pratique, le travail s’est souvent réalisé en groupe de deux. De plus, si nous n’avons pas mis en place de classes de tests via JUnit, nous avons tout de même pratiqué une revue de code régulière et portant sur des parties de code que chacun n’avait pas lui-même écrites. En cela la répartition des tâches perdait peu à peu de sa netteté à mesure de l’avancée du sprint puisque les

relecteurs pouvaient tout à fait venir modifier une partie de code ne correspondant pas à la tâche qui leur était attribuée.

Une grande part du travail a aussi porté sur des éléments annexes à l'écriture pure du code. Il nous a fallu aboutir à une architecture de classes à laquelle chacun d'entre-nous pouvait adhérer et qui nous apparaissait accessible dans sa mise en œuvre. Le choix de LibGdx a constitué un atout autant qu'une difficulté. Un atout d'abord car il est possible de trouver de nombreuses aides et ressources en lignes permettant de comprendre le fonctionnement de LibGdx et de prendre connaissance des classes qui sont implémentées dans la librairie.

Une difficulté d'autre part parce que la documentation nombreuse et foisonnante nous a demandé un investissement et un travail particulièrement long de compréhension et d'assimilation. De plus, il était impossible de discuter de nos éventuelles difficultés avec les autres groupes compte tenu que nous ne partagions pas les mêmes bases de code. Enfin, la contrainte de Gradle a aussi constitué un problème, mais dont la résolution nous a permis de nous plonger pleinement dans les fichiers de LibGdx et d'en comprendre -au moins en partie- le fonctionnement de manière plus claire et précise.

Annexe – Build via Gradle

Première approche de Gradle & configuration pour notre projet

François-Marchal Marius - Guehi Carl - LeBihan Willam - Poujon Adrien

2019 ACL-ENSEM-ISN2A

Intoduction

Dans ce document, on se propose une étude sommaire de l'outil de build *GRADLE*. On commencera par expliquer pourquoi nous sommes amenés à utiliser cet outil dans le cadre de l'utilisation de libGDX, ce qui nous permettra au passage de justifier de l'architecture -en termes d'arborescence des dossiers- de notre projet.

La suite du document consistera en une navigation explicative au sein des différents fichiers nécessaires au bon fonctionnement du build.

Au cours de cette navigation, on donnera un aperçu des difficultés et des bugs rencontrés tout au long de la mise en place de l'outil de build. Ce faisant, les diverses modifications apportées aux fichiers de build seront expliquées et justifiées.

I/ LibGDX : Une architecture imposée

LibGDX est un framework dont l'objectif est d'apporter une aide aux développeurs dans la création et le déploiement de jeux vidéo. Ce framework permet non-seulement de réaliser des jeux exécutables sur Desktop (Windows, Linux, ...), mais offre aussi la possibilité de développer des jeux s'exécutant sur Android, différents iOS, ou dans des navigateurs web, par le biais d'HTML.

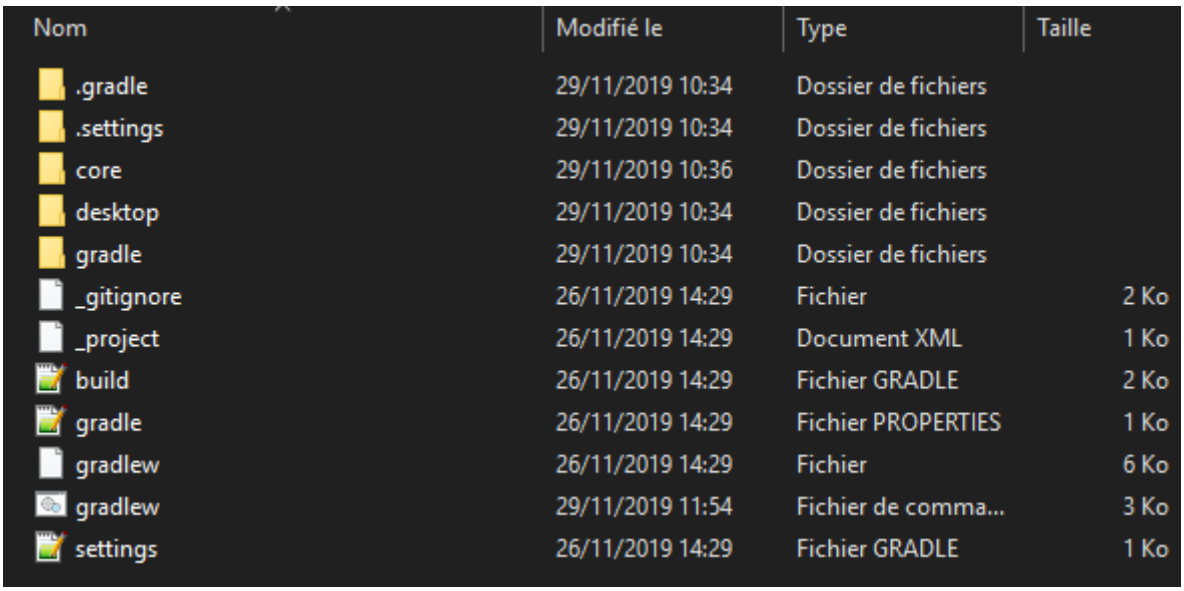
Pour garantir le bon développement de projets qui peuvent être multi-plateforme, libGDX s'appuie sur l'arborescence de fichiers suivante :

```
  1. Un dossier “core” : qui contient toutes les informations importantes du
logiciel, et en particulier la logique de jeu ;

  2. Un dossier par plateforme de développement, par exemple _“Desktop”_ : qui
contient une unique classe pour mettre en place le Launcher propre à la plateforme
qui nous intéresse ;

  3. Un dossier _“Gradle”_ : qui contient les informations importantes pour
Gradle, notamment la version qui est utilisée
```

Dans notre cas on se retrouve donc avec une racine de projet qui se présente comme sur l'image suivante :



Nom	Modifié le	Type	Taille
.gradle	29/11/2019 10:34	Dossier de fichiers	
.settings	29/11/2019 10:34	Dossier de fichiers	
core	29/11/2019 10:36	Dossier de fichiers	
desktop	29/11/2019 10:34	Dossier de fichiers	
gradle	29/11/2019 10:34	Dossier de fichiers	
_gitignore	26/11/2019 14:29	Fichier	2 Ko
_project	26/11/2019 14:29	Document XML	1 Ko
build	26/11/2019 14:29	Fichier GRADLE	2 Ko
gradle	26/11/2019 14:29	Fichier PROPERTIES	1 Ko
gradlew	26/11/2019 14:29	Fichier	6 Ko
gradlew	29/11/2019 11:54	Fichier de comma...	3 Ko
settings	26/11/2019 14:29	Fichier GRADLE	1 Ko

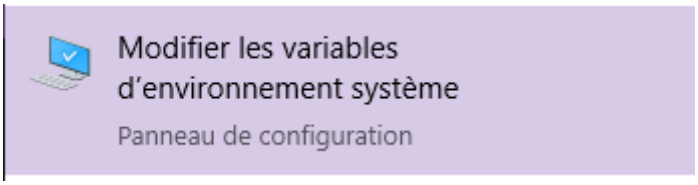
On voit que le *SETUP* par défaut de libGDX inclut en outre les fichiers nécessaires au déploiement du projet, basés sur l'outil de build Gradle. C'est parce que libGDX utilise par défaut l'outil de build Gradle que nous serons amenés au cours de ce projet à l'utiliser à notre tour. En effet, en ne nous éloignant pas excessivement de la structure normale d'un projet libGDX, nous pourrons plus facilement trouver de l'aide si rencontrons des difficultés dans la suite du projet. Au contraire, choisir une configuration exotique par rapport au *SETUP* traditionnel de libGDX pourrait nous faire perdre du temps en faisant apparaître des erreurs dont la ou les sources ne seraient pas facilement identifiables.

II/ Gradle & notre projet : parcours des différents fichiers

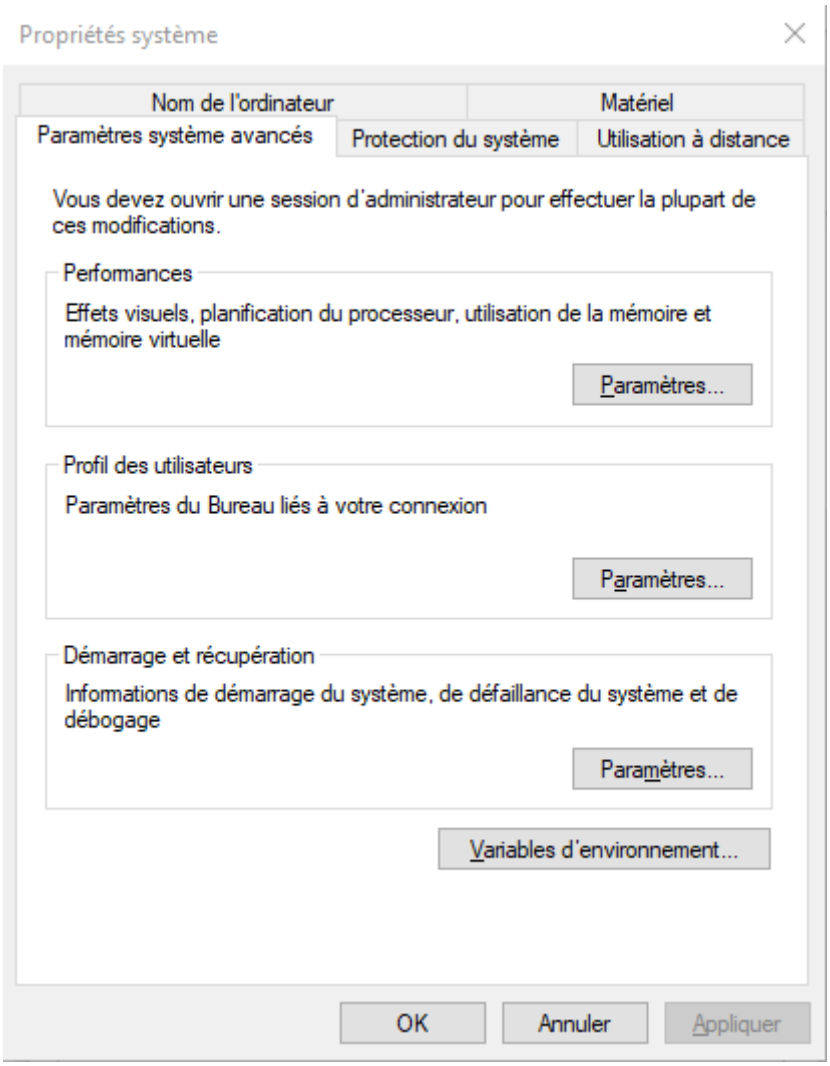
1- Les Gradlew

La racine du projet présente deux fichiers nommés *gradlew* et *gradlew.bat*. Le premier fichier permet de démarrer l'outil de build Gradle dans le cas où l'on se trouve sur une machine Unix, le second (le .bat) permet de faire de même mais dans un environnement Windows.

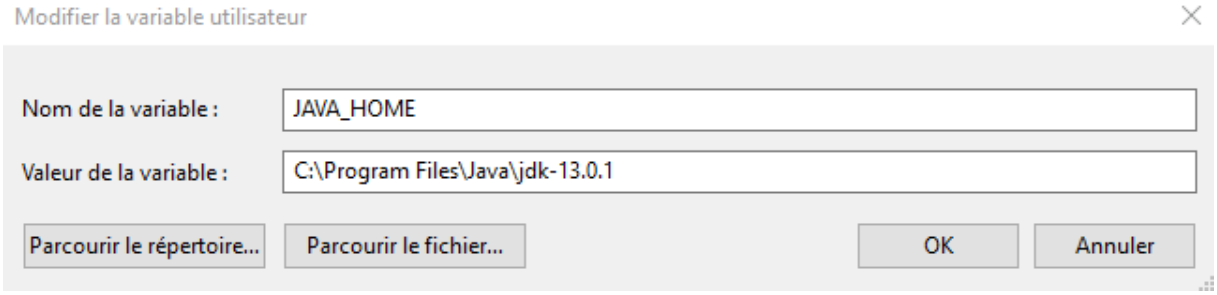
Dans les deux cas, le rôle du fichier est donc identique. On commence par aller chercher le jdk de java, on fait pour cela appel à la variable d'environnement système *JAVA_HOME*. Il faut par conséquent avoir correctement configuré cette variable. Sur Windows, on recherchera dans le menu démarrer "variables environnement" et on se rendra sur :



La fenêtre est alors la suivante :



On clique ensuite sur le bouton *Variables d'environnement…* et dans la fenêtre qui s'ouvre on choisit le bouton *Nouvelle…* . Une nouvelle fenêtre s'ouvre alors, où l'on peut entrer le nom et le chemin de notre variable d'environnement *JAVA_HOME* :



Dans le fichier *gradlew.bat*, on a :

```
@rem Set local scope for the variables with windows NT shell
if "%OS%"=="Windows_NT" setlocal

set DIRNAME=%~dp0
if "%DIRNAME%" == "" set DIRNAME=.
set APP_BASE_NAME=%~n0
set APP_HOME=%DIRNAME%

@rem Add default JVM options here. You can also use JAVA_OPTS and GRADLE_OPTS to
pass JVM options to this script.
set DEFAULT_JVM_OPTS="-Xmx64m" "-Xms64m"

@rem Find java.exe
if defined JAVA_HOME goto findJavaFromJavaHome

set JAVA_EXE=java.exe
%JAVA_EXE% -version >NUL 2>&1
if "%ERRORLEVEL%" == "0" goto init

echo.
echo ERROR: JAVA_HOME is not set and no 'java' command could be found in your
PATH.
echo.
echo Please set the JAVA_HOME variable in your environment to match the
echo location of your Java installation.

goto fail
:findJavaFromJavaHome
set JAVA_HOME=%JAVA_HOME:"=%
set JAVA_EXE=%JAVA_HOME%/bin/java.exe

if exist "%JAVA_EXE%" goto init

echo.
echo ERROR: JAVA_HOME is set to an invalid directory: %JAVA_HOME%
echo.
echo Please set the JAVA_HOME variable in your environment to match the
echo location of your Java installation.

goto fail
```

On voit que l'on va à l'adresse de *JAVA_HOME* pour récupérer *java.exe* dans le dossier /bin. Dans le cas où *JAVA_HOME* n'est pas trouvé, l'invite de commande renverra une erreur indiquant que le chemin est mal défini. Une erreur est aussi renvoyée dans le cas où *JAVA_HOME* est trouvée mais pointe sur une adresse ne correspondant pas au JDK attendu.

Si tout se déroule correctement, le code passe à la phase *init* :

```
:init
@rem Get command-line arguments, handling Windows variants

if not "%OS%" == "Windows_NT" goto win9xME_args
```

Dans cette phase, on commence par vérifier la version de l'OS sur laquelle on est en train d'exécuter nos commandes. Cela permet de mettre à jour certaines variables si

l'on ne se trouve pas sur l'OS attendu (Windows_NT, puisque l'on est dans le .bat qui ne concerne que Windows). Dans tous les cas, on arrive à l'étape *execute* :

```
:execute
@rem Setup the command line

set CLASSPATH=%APP_HOME%\gradle\wrapper\gradle-wrapper.jar

@rem Execute Gradle
"%JAVA_EXE%" %DEFAULT_JVM_OPTS% %JAVA_OPTS% %GRADLE_OPTS% "-
Dorg.gradle.appname=%APP_BASE_NAME%" -classpath "%CLASSPATH%"
org.gradle.wrapper.GradleWrapperMain %CMD_LINE_ARGS%

:end
```

On crée la variable *CLASSPATH* qui pointe sur l'adresse du *graddle-wrapper.jar*, puis on se rend à cette adresse et on exécute le .jar pour démarrer la construction du build.

2- Le gradle-wrapper.properties

Pour exécuter le *graddle-wrapper.jar*, il va falloir consulter les propriétés avec lesquelles on veut lancer notre build sous Gradle. C'est pourquoi on trouve, dans le même dossier, un second fichier nommé *gradle-wrapper.properties*. Ce fichier contient les quelques lignes suivantes :

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-5.4.1-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

La ligne qui nous intéresse particulièrement ici est la ligne définissant la variable *distributionUrl*. On peut en effet remarquer qu'elle va aller chercher la distribution 5.4.1 de gradle. Cependant, nous souhaitons utiliser la version 6.0.1, qui est plus récente et se comporte mieux avec libGDX. On remplace donc dans ce fichier la ligne concernée par une nouvelle ligne :

```
distributionUrl=https\://services.gradle.org/distributions/gradle-6.0.1-bin.zip
```

Gradle pourra enfin se lancer correctement et commencer la véritable tâche que l'on s'était proposée : déployer notre projet.

3- Le build.gradle de la racine du projet

Retournons dans la racine du projet. Cette fois, nous nous intéressons au fichier *build.gradle*. On peut réduire gradle à deux types de possibilités : soit il forme des projets (*project*), soit il exécute des tâches (*task*). En général, former un projet nécessite d'exécuter une à plusieurs tâches. Le fichier qui nous intéresse fait notamment référence à deux *project* :

```
project(":desktop") {
```

```
    apply plugin: "java-library"


    dependencies {
        implementation project(":core")
        api "com.badlogicgames.gdx:gdx-backend-lwjgl:$gdxVersion"
        api "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-desktop"
        api "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-desktop"

    }
}

project(":core") {
    apply plugin: "java-library"


    dependencies {
        api "com.badlogicgames.gdx:gdx:$gdxVersion"
        api "com.badlogicgames.gdx:gdx-box2d:$gdxVersion"

    }
}
```

Les deux *project* sont *:core* et *:desktop*. Il n'y a rien de particulier indiqué entre les accolades qui suivent ces projets mis à part quelques dépendances. Pour aller chercher les *task* supplémentaires à exécuter sur ces *project*, le fichier *setting.gradle* comporte l'unique ligne suivante :

```
include 'desktop', 'core'
```

Et se charge donc de faire référence à deux fichiers de build que nous parcourrons dans les parties 3 et 4.

Dans le cadre de notre projet, nous faisons appel à une libraire externe à libGDX, l'outil Texture Packer, uniquement sous forme de code et non sous sa forme user-friendly. Dans notre code, on fait donc plusieurs fois appel à cette librairie et la compilation du programme ne peut se dérouler correctement qu'uniquement si la librairie, nommée *runnable-texturepacker*, est ajoutée au build. Pour ce faire on modifie la partie concernant le *:desktop* du code du *build.gradle* :

```
project(":desktop") {
    apply plugin: "java-library"

    dependencies {
        //On ajoute spécifiquement la ligne suivante :
        implementation files("libs/runnable-texturepacker.jar")

        implementation project(":core")
        api "com.badlogicgames.gdx:gdx-backend-lwjgl:$gdxVersion"
        api "com.badlogicgames.gdx:gdx-platform:$gdxVersion:natives-desktop"
        api "com.badlogicgames.gdx:gdx-box2d-platform:$gdxVersion:natives-desktop"

    }
}
```

La librairie *runnable-texturepacker.jar* devra donc être placée dans un dossier *libs* lui-même situé dans le dossier *desktop* de la racine du projet (le processus réel a été de

créer le dossier *desktop/libs* et d'y placer la librairie *runnable-texturepacker.jar* lors du développement sous *eclipse*, puis d'ajouter en conséquence cette ligne d'implémentation, de sorte que la librairie soit trouvée au bon chemin).

Pour l'instant, si l'on lance le build, on se retrouve avec un fichier de la forme *build-1.0.jar*. Ce nommage du fichier est doublement problématique. D'abord, si l'on réalise une modification dans le code et que l'on souhaite relancer le build, alors le fichier *build-0.1.jar* est écrasé. En d'autres termes, toute version nouvelle écrase la précédente sans possibilité de conserver l'évolution des builds (sauf si l'on décide de les renommer à la main une fois créés...). D'autre part, il est clair que le nom donné au build ne convient absolument pas à notre projet et qu'il n'a pas de sens précis.

Pour répondre à ces deux problèmes, on commence par définir une nouvelle méthode dans le fichier *build.gradle* de la racine du projet :

```
 def getDate(){
        def date = new Date()
        def formattedDate = date.format('HHmmss_ddMMyyyy')
        return formattedDate
}
```

La méthode *getDate()* ainsi définie récupère la date courante sur la machine, et la formate de sorte qu'elle soit écrite sous la forme *HeureHeureMinuteMinuteSecondeSeconde_JourJourMoisMoisAnneeAnnee*. Ce faisant on peut ajouter entre les accolades de *allprojects* la spécification de la *version* créée par le build en cours d'exécution :

```
allprojects {

    //On se contente de remplacer 'version = "1.0"' par la ligne suivante :
    version = "sprint.1-build-"+getDate()

    ext {

        appName = "ez-maze"
        gdxVersion = '1.9.10'
        roboVMVersion = '2.3.8'
        box2DLightsVersion = '1.4'
        ashleyVersion = '1.7.0'
        aiVersion = '1.8.0'
    }

    repositories {
        mavenLocal()
        mavenCentral()
        jcenter()
        google()
        maven { url "https://oss.sonatype.org/content/repositories/snapshots/" }
        maven { url "https://oss.sonatype.org/content/repositories/releases/" }
    }
}
```

De cette façon le build créé sera maintenant de la forme *build-sprint.1-build-HHmmss_ddMMyyyy.jar*. Reste encore à modifier la première occurrence de build afin

d'afficher en lieu et place le véritable nom de notre projet : "*EzMaze*". Ce dernier ajustement va nécessiter de s'aventurer encore un peu plus loin dans les fichiers de build de Gralde.

4- Le build.gradle du dossier *desktop*

Le premier *project* a être appelé dans le *build.gradle* de la racine du projet est *project(":desktop")*. On a vu que les tâches qui s'exécutent dans ce *project* sont en réalité définies dans un autre fichier de build, lequel est rattaché au build de la racine du projet par le fichier *settings.gradles* et la commande

```
include 'desktop', 'core'
```

C'est par conséquent en se rendant dans le dossier *desktop* que nous pouvons continuer notre expédition en terres gradleliennes. Ici, on trouve de nouveau un fichier *build.gradle*. Cette fois, il a la forme suivante :

```
apply plugin: "java"

sourceCompatibility = 1.7
sourceSets.main.java.srcDirs = [ "src/" ]
sourceSets.main.resources.srcDirs = ["../core/assets"]

project.ext.mainClassName = "com.mygdx.ezmaze.desktop.DesktopLauncher"
project.ext.assetsDir = new File("../core/assets")

task run(dependsOn: classes, type: JavaExec) {
    main = project.mainClassName
    classpath = sourceSets.main.runtimeClasspath
    standardInput = System.in
    workingDir = project.assetsDir
    ignoreExitValue = true
}

task debug(dependsOn: classes, type: JavaExec) {
    main = project.mainClassName
    classpath = sourceSets.main.runtimeClasspath
    standardInput = System.in
    workingDir = project.assetsDir
    ignoreExitValue = true
    debug = true
}

task dist(type: Jar) {
    manifest {
        attributes 'Main-Class': project.mainClassName
    }
    from {
        configurations.compileClasspath.collect { it.isDirectory() ? it :
zipTree(it) }
    }
    with jar
}
```

```
dist.dependsOn classes
```
Dans ce fichier, on commence par charger certaines adresses de dossiers qui seront utiles par la suite. On repère immédiatement la présence de trois *task* : une tâche *run*, une tâche *debug* ainsi qu'une tâche *dist*. On pourra appeler l'une ou l'autre de ces *task* depuis la commande lorsque nécessaire. La *task run* permettra de lancer directement le programme, la *task debug* permettra de lancer le programme en mode *debug = true* et la *task dist* permettra de créer le fichier .jar. C'est donc cette dernière *task dist* qui nous intéresse dans le cadre du déploiement de notre projet. Et c'est aussi dans celle-ci que l'on va pouvoir remplacer la première itération du mot "build" dans le nom du .jar exécutable actuellement créé par l'outil de build :

```
task dist(type: Jar) {
    //On ajoute cette ligne pour spécifier le nom du build à créer !
    baseName = "EzMaze"

    manifest {
        attributes 'Main-Class': project.mainClassName
    }
    from {
        configurations.compileClasspath.collect { it.isDirectory() ? it :
zipTree(it) }
    }
    with jar
}
```
Dorénavant, le fichier .jar exécutable créé par lancement du build est de la forme *EzMaze-sprint.1-build-HHmmss_ddMMyyyy*.

5- Le build.gradle du dossier *core*

Si on lance à présent le build avec gradle en utilisant l'option *--debug* ou l'option *--info*, on se rend compte qu'il demeure quelques erreurs lors de la lecture des classes. Une attention particulière aux messages d'erreurs affichés permet de se rendre compte que ces erreurs proviennent en fait uniquement des commentaires faits tout au long du code. Plus particulièrement, ce sont certains caractères spécifique propre au français (comme les accents par exemple) qui semble provoquer l'apparition d'erreurs (négligeables mais tout de même présentes) dans le déploiement du projet.

Le *build.gradle* du dossier *core* présente le code suivant :

```
apply plugin: "java"

sourceCompatibility = 1.7
[compileJava, compileTestJava]*.options*.encoding = 'UTF-8'

sourceSets.main.java.srcDirs = [ "src/" ]
```
Puisque l'ensemble des classes composant le projet se trouvent dans les sous-dossiers du dossier *core* (par construction sur *eclipse*), on est bien obligé d'utiliser le plugin *"java"* et de lancer les compilations avec le compilateur java. Cependant, l'encoding ici proposé, *'UTF-8'*, ne correspond en fait pas à celui utilisé au sein

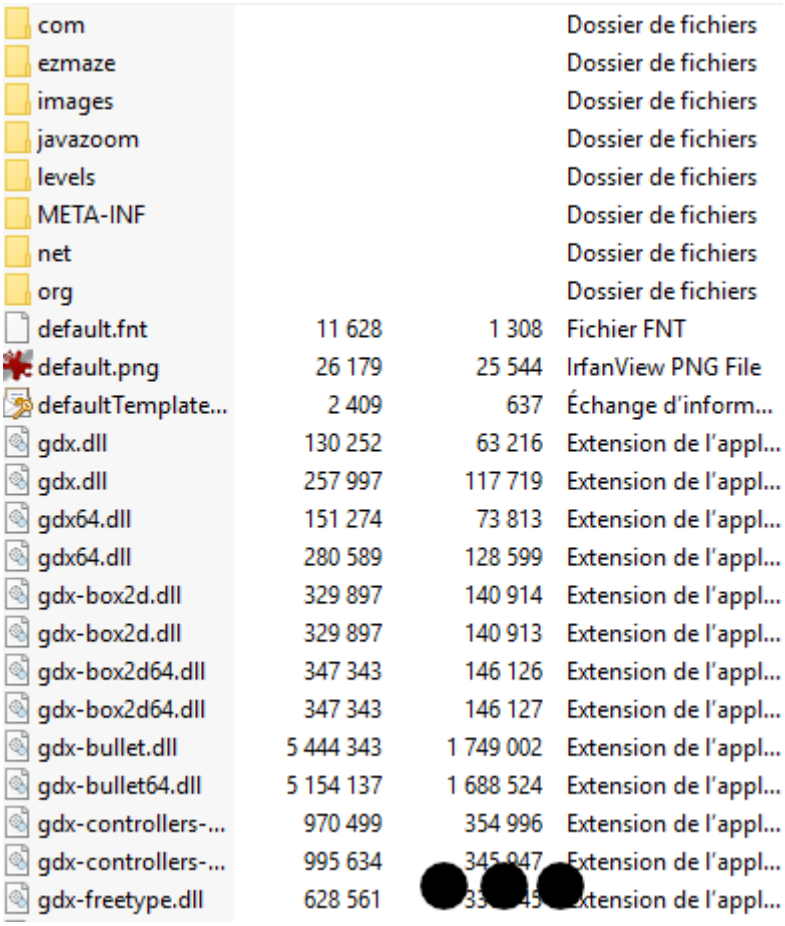
d'*eclipse*. On choisit donc l'encoding adapté '*Cp1252*', et les erreurs de lecture sont dès lors absentes.

A ce stade, après avoir réalisé toutes les opérations indiquées au-dessus dans cette page, on est finalement en mesure de lancer l'outil de build avec gradle sans obtenir d'erreur. OUF !

6- « Et c'est pas fini ! »

Mais cependant, si l'on a l'intention d'utiliser l'exécutable .jar que l'on vient de créer (et on a effectivement l'intention de le faire, sinon à quoi bon avoir fait tout ce qui précède...) on risque d'avoir une mauvaise surprise. En effet, au double-clic la fenêtre de jeu se lance effectivement, mais seulement pour quelques secondes avant de se refermer ! Que s'est-il passé ?

Pour le savoir, on réessaie l'exécution du build .jar à partir du shell. Cette fois, on a accès à l'erreur qui provoque la fermeture de la fenêtre. Il s'agit d'un problème dans la résolution de certains chemins d'adresse relatifs qui ne fonctionnent plus dans le .jar. En particulier, c'est le chargement des images qui pose problème. Pour mieux comprendre, on décide de jeter un coup d'oeil à l'intérieur du .jar, en utilisant 7zip ou WinRAR par exemple.



com			Dossier de fichiers
ezmaze			Dossier de fichiers
images			Dossier de fichiers
javazoom			Dossier de fichiers
levels			Dossier de fichiers
META-INF			Dossier de fichiers
net			Dossier de fichiers
org			Dossier de fichiers
default.fnt	11 628	1 308	Fichier FNT
default.png	26 179	25 544	IrfanView PNG File
defaultTemplate...	2 409	637	Échange d'inform...
gdx.dll	130 252	63 216	Extension de l'appl...
gdx.dll	257 997	117 719	Extension de l'appl...
gdx64.dll	151 274	73 813	Extension de l'appl...
gdx64.dll	280 589	128 599	Extension de l'appl...
gdx-box2d.dll	329 897	140 914	Extension de l'appl...
gdx-box2d.dll	329 897	140 913	Extension de l'appl...
gdx-box2d64.dll	347 343	146 126	Extension de l'appl...
gdx-box2d64.dll	347 343	146 127	Extension de l'appl...
gdx-bullet.dll	5 444 343	1 749 002	Extension de l'appl...
gdx-bullet64.dll	5 154 137	1 688 524	Extension de l'appl...
gdx-controllers-...	970 499	354 996	Extension de l'appl...
gdx-controllers-...	995 634	345 047	Extension de l'appl...
gdx-freetype.dll	628 561	[illegible]	Extension de l'appl...

On remarque que les images sont pourtant bien là !

Pour remédier à ce problème, c'est donc le code java qu'il faut modifier. Pour ce faire, on indique à *eclipse* que le dossier *assets* du dossier *core* constitue son dossier de travail (*Working Directory*). On remarquera que c'est aussi ce qui est fait dans le fichier *desktop/build.gradle* où l'on a :

```
project.ext.mainClassName = "com.mygdx.ezmaze.desktop.DesktopLauncher"
project.ext.assetsDir = new File("../core/assets");

//Puis plus loin dans le code :
workingDir = assetsDir
```

On modifie ensuite les chemins utilisés en java en considérant que l'on se trouve déjà dans le dossier *core/assets*. Cette fois, on peut lancer le .jar exécutable du projet correctement : la fenêtre de jeu reste ouverte et toutes les commandes répondent comme espéré.