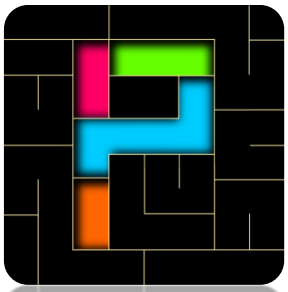
# PIXELLY
# Application Android




**Présenté par le groupe 5.1 :**

Manon BRUN, Loïc EZRATI, Jérémy LERICHE, Mélodie MEISSEL, Aurélia RAHARISON

**Commanditaire :**

Richard OSTROWSKI

**Tuteur :**

Jacques VINCENSINI


**Année :** 2017/2018

# SOMMAIRE

# I. Présentation du projet

## 1) Le commanditaire et l'équipe de BrainItAll

Le commanditaire de ce projet est Mr Richard Ostrowski, enseignant-chercheur à l'Université d'Aix-Marseille. Pour ce projet, ce dernier a demandé à l'équipe de BrainItAll d'appréhender les bases de la programmation sous Android à travers le développement d'une application de type puzzle/logique ou d'un jeu de plateau. Le choix du type d'application que l'on souhaitait développer était totalement libre mais devait tout de même être soumis à son approbation. Une fois que cela a été fait, il nous a donné pour mission d'élaborer nous même notre cahier des charges et de s'y tenir.

Qu'est-ce que BrainItAll ?

Petite par la taille mais grande par ses idées, BrainItAll est une entreprise française qui a vu le jour au sein de l'Université d'Aix-Marseille. Composée de cinq membres de la licence professionnelle SIL-NTI : Manon Brun, Loïc Ezrati, Jérémy Leriche, Mélodie Meissel et Aurélia Raharison, elle a pour but de se spécialiser dans le développement de jeux sous *Android*.

## 2) Le projet

Afin d'appréhender les bases de la programmation sous *Android*, nous avons décidé de s'inspirer du jeu Pixel Pop, disponible sur le Play Store, afin de créer notre propre application de type puzzle/logique, nommée Pixelly.

Qu'est-ce que Pixelly ?

Pixelly permet de résoudre de nombreux puzzles à l'aide de chiffres et de couleurs, le but étant de découvrir les images en pixel art qui se cachent derrière chaque casse-tête. Ce jeu s'adresse à un public âgé d'au moins dix ans et appréciant à la fois les jeux de puzzle et le pixel art.

Quelles sont les règles du jeu ?

Il s'agit de remplir l'intégralité d'une grille avec des chemins. Pour ce faire, il faut relier deux terminaisons ayant le même nombre et la même couleur. La longueur des chemins correspond au nombre indiqué dans les terminaisons par lesquelles ces mêmes chemins doivent commencer ou finir. Cependant, un chemin ne peut pas croiser d'autres chemins ou passer au travers de terminaisons.

Le reste des spécificités fonctionnelles et non fonctionnelles sont quant à elles décrites avec précision dans le cahier des charges (Annexe 1).

# II. Organisation du travail

## 1) Les diagrammes de Gantt et la répartition des tâches

Pour organiser notre travail, nous avons dû utiliser un diagramme pour modéliser la planification des tâches nécessaires à la réalisation du projet. Pour cela nous avions la possibilité d'utiliser deux diagrammes différents. Le premier, le diagramme de Gantt qui est souvent utilisé pour des projets simples dont les tâches s'enchaînent

facilement. Le second, le diagramme de PERT qui lui, est utilisé dans des projets plus complexes sur une plus longue période.

Aux vues de la durée du projet et des connaissances du groupe, nous avons choisi d'utiliser le diagramme de GANTT et pour se faire nous nous sommes servis du logiciel libre GanttProject.

### a) Les diagrammes

Durant ce projet nous avons produit trois diagrammes de GANTT correspondant à trois phases différentes du projet :

- Le premier correspond à son commencement et nous a permis de planifier le projet dans son ensemble et de structurer au mieux nos pensées (figure 1).

- Le second correspond à la fin de la phase d'analyse et nous a permis, en fonction de notre avancement, de planifier avec plus de détails la suite du projet et plus particulièrement la partie 'Développement' (Annexe 2).

- Et pour finir, un dernier correspondant à la fin du projet et nous permettant de pouvoir comparer ce qui était initialement prévu et ce qui s'est finalement passé (figure 2). Il nous permet également d'apprécier les différentes erreurs de planification que nous avons pu faire au cours de ce projet pour ne plus les reproduire par la suite.

Diagramme de GANTT initial

Au début du projet, nous avons donc créé un premier diagramme de GANTT : le diagramme initial. Il se comporte d'une première partie 'Analyse', d'une partie 'Développement ' et ainsi que de trois autres parties correspondant aux tests, déploiement et la préparation à la présentation du projet.

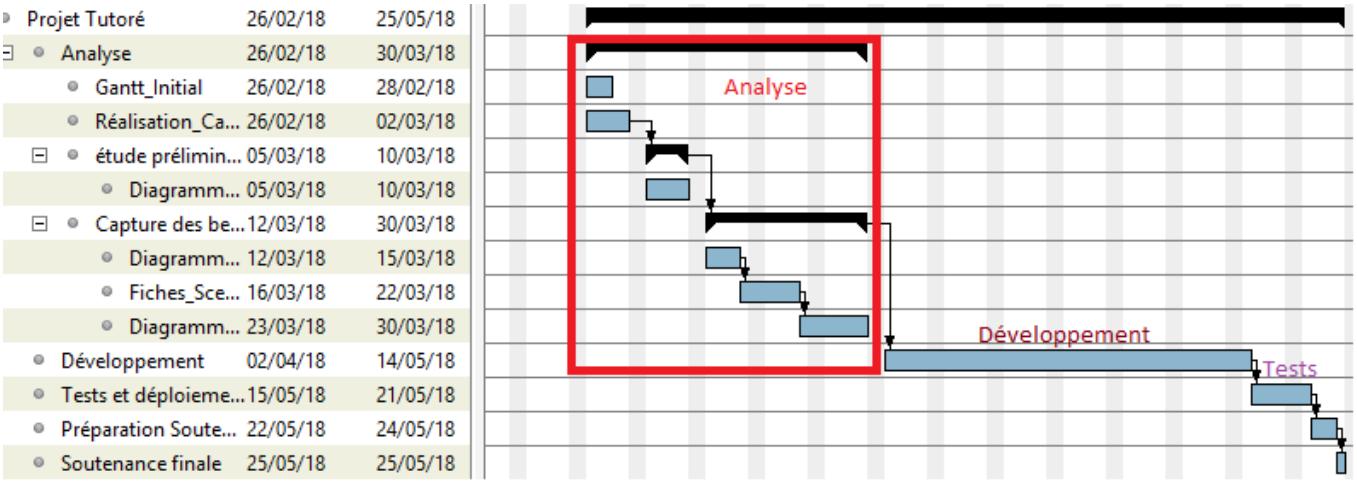



*Figure 1 : Diagramme de Gantt initial*

Dans ce diagramme, on peut également constater que la partie 'Analyse' a été réalisée par l'intégralité de l'équipe qui a travaillé ensemble à l'élaboration du cahier des charges et au reste de l'analyse.

On peut noter que ce diagramme a été le premier diagramme de GANTT du projet. Il comporte donc des erreurs comme par exemple le fait qu'une tâche principale ne contienne qu'une seule sous-tâche ou le fait qu'il manque certaines informations.

Diagramme de GANTT final

A la fin du projet, nous avons réalisé un diagramme de GANTT final nous permettant d'analyser le projet dans son ensemble et de voir s'il s'était correctement déroulé comme initialement prévu ou s'il y avait eu des erreurs d'appréciation, ou des tâches qui auraient été oubliées.

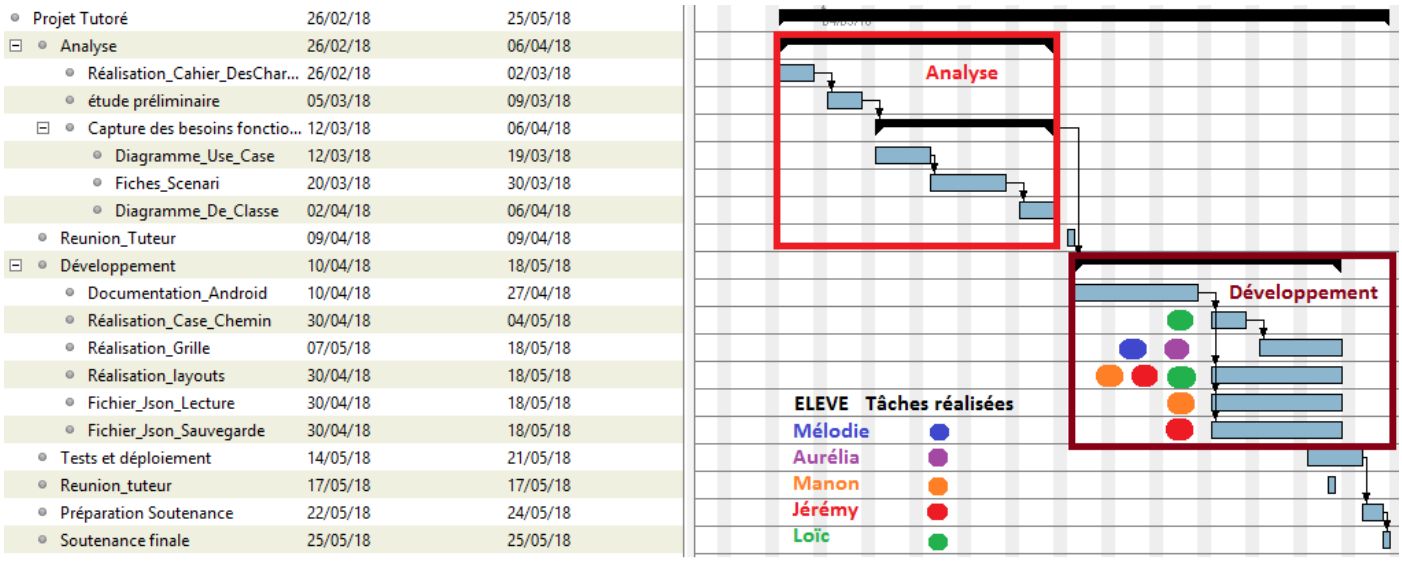



*Figure 2 : Diagramme de Gantt final*

Dans ce dernier diagramme, on peut donc voir que la partie 'Développement' a été approfondie avec différentes sous tâches, telles que la documentation au sujet d'*Android*, et la réalisation des divers éléments constituant notre projet.

Contrairement à la partie 'Analyse' où toute l'équipe avait travaillé ensemble, la partie 'Développement' a été divisée en sous-tâches pour lesquelles nous avons dû nous répartir le travail pour avancer le plus efficacement possible et respecter au mieux le cahier des charges, pour obtenir en fin de compte une application qui soit la plus fonctionnelle possible.

Si l'on compare le diagramme de GANTT initial avec le diagramme de GANTT final, on peut voir que les parties 'Analyse' et 'Développement ' ont eu une durée plus longue que ce que nous avions déterminé au début du projet. Cela peut s'expliquer par le fait que nous avions principalement sous-estimé la période d'apprentissage de la programmation sous *Android*.

En effet, afin de mieux la comprendre et d'en acquérir de solides bases, nous avons suivi différents cours et tutoriels en ligne. Nous avons également testé de notre côté, diverses fonctionnalités qu'offre la programmation sous *Android* et qui nous étaient inconnues jusqu'ici. De plus, nous avons eu à installer divers logiciels nécessaires à

la bonne réalisation de notre projet. Ces différents retards pris sur les parties 'Analyse', nous ont donc imposés une période de développement de l'application plus courte qu'initialement prévue.

Nous pouvons également ajouter qu'à la fin du projet et donc après avoir réalisé différents diagrammes de GANTT, nous avons appris à mieux utiliser cet outil de gestion de projet, en évitant les erreurs que nous avions pu faire dans les premiers diagrammes.

### b) La répartition des tâches

Chaque élève a eu différentes tâches à réaliser dans la conception de ce projet. Comme énoncé précédemment, l'analyse à été réalisée par l'ensemble de l'équipe alors que le développement a été divisé en plusieurs parties afin d'être au mieux réparties entre les différents membres du groupe en fonction de leur affinité (figure 3).

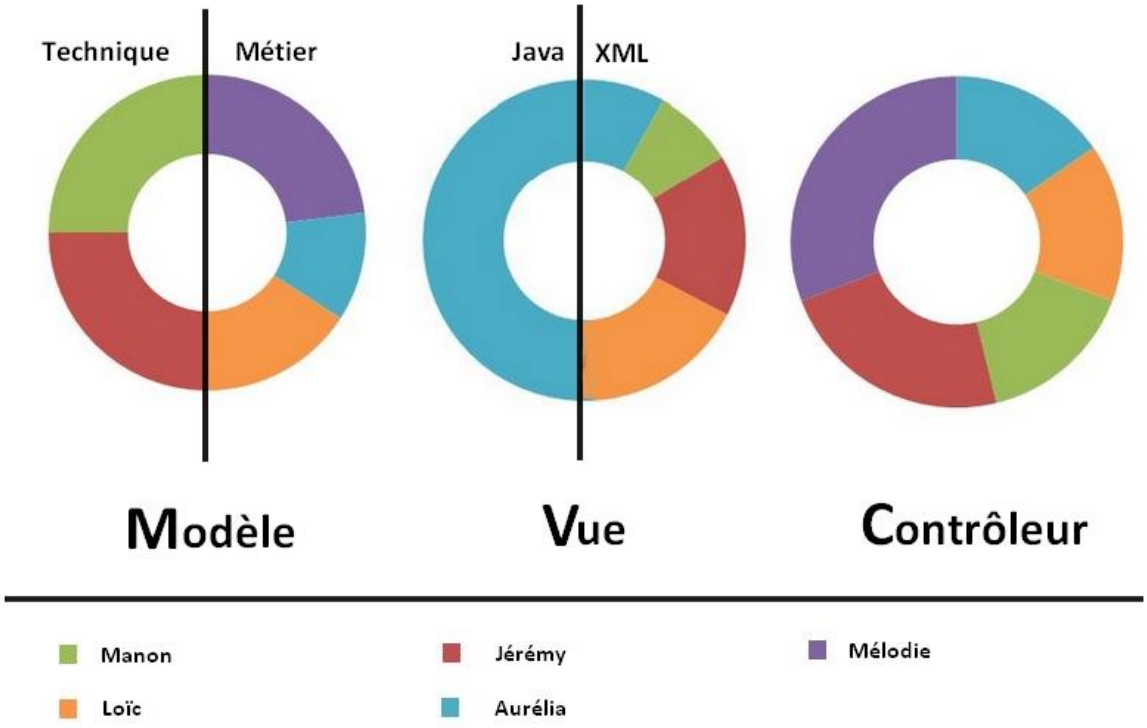



*Figure 7 : Répartition des tâches en fonction de l'architecture MVC*

Concernant la partie Modèle de l'application, nous nous sommes répartis le travail en deux. Manon et Jérémy se sont occupés de la partie Modèle Technique avec l'écriture et la lecture de fichier au format JSON correspondant aux sauvegardes et aux niveaux, tandis qu'Aurélia, Mélodie et Loïc se sont occupés de concevoir le Modèle Métier du jeu. En effet, Mélodie s'est occupée de gérer les divers mécanismes de jeu au travers de la grille et de certains contrôleurs ainsi que de l'implémentation du Design Pattern MVC. Aurélia quant à elle, s'est chargée d'implémenter la classe de vue personnalisée, appelée widget sous Android, représentant un niveau. Elle a également géré toutes les interactions possibles entre l'utilisateur et le niveau jouable. Cela comporte :

- le traçage de la grille, des chemins, des terminaisons.
- la gestion de la fin du niveau à la fois dans le modèle et la vue

Loïc quant à lui s'est occupé de créer les autres classes du modèle métier à savoir : les cases (cases simples et terminaisons), ainsi que les chemins.

Pour les parties Vue et Contrôleur de notre application, nous nous sommes organisés de la manière suivante : chacun d'entre nous s'est occupé d'un contrôleur, appelé 'Activité' sous Android, et de la vue qui lui était associée en XML, excepté Mélodie dont le travail était plus de se concentrer sur les mécanismes et la logique de jeu (comme le déblocage d'un niveau une fois le dernier de la liste réussi). De ce fait, elle n'a pas touché au layout XML est s'est en revanche focalisée sur la communication entre deux contrôleurs.

Les tests unitaires prévus ont été réalisés par Loïc sur certaines méthodes des classes Java qu'il a réalisé. Il s'est également occupé de gérer la mise à jour des diagrammes de Gantt. Par ailleurs, nous pouvons préciser que les différents niveaux du jeu ont été quant à eux pensé et créé par Manon.

## *2) Le travail collaboratif*

Différents outils ont été utilisés pendant toutes la duré du projet. La communication étant essentiel pour un bon avancement, nous avons utilisé principalement deux logiciels que sont Discord et Google Drive. Néanmoins, nous pouvons noter qu'il était très important pour nous d'effectuer des réunions physiques afin de discuter et débattre de certains points du projet ensemble.

### a) Les réunions physiques

Avec le tuteur :

Pendant toute la durée du projet nous avons eu seulement deux réunions avec le tuteur. Elles avaient pour but de montrer notre avancement et d'aborder des points précis à éclaircir. Notons néanmoins qu'il aurait été préférable de notre part de programmer plus de réunion ou d'échanger davantage avec notre tuteur.

Avec le groupe :

De nombreuses réunions ont été programmées et plus particulièrement pour faire l'analyse du projet. Durant ces rassemblements c'était l'occasion pour nous de poser des questions si l'on bloquait sur un point en particulier, de voir l'avancement de chacun et de définir le travail qu'il nous restait à faire ou à modifier.

### b) Les réunions virtuelles

Suite à divers imprévus, n'étant pas en mesure de se voir a chaque fois qu'on le souhaitait, nous nous somme servi du logiciel Discord. Il s'agit d'un logiciel gratuit basé sur un principe de serveurs. De ce fait on peut créer un serveur et inviter des gens pour le rejoindre. Les serveurs sont composés d'autant de salons vocaux ou textuels que l'on souhaite. L'administrateur peut donner ses droits à tous les membres s'il le souhaite. Ainsi il est possible de rendre des salons visibles ou non selon le rôle de chaque membre.

Nous y avons donc créé un serveur pour le projet en donnant le rôle d'administrateur à chacun. De ce fait nous étions en mesure de créer un salon textuel sur des thèmes bien précis pour pouvoir y partager nos questions ou encore nos réflexions et y débattre. Le salon vocal nous a été utile pour communiquer, se répartir le travail et organiser les réunions pour les jours à venir.

Notons également que les partages d'écrans sont aussi possibles sur cet outil. Cela nous a était très utile lors de réunions comme par exemple pour installer Android Studio. De cette façon tout le monde avait la même version et le même paramétrage.

#### c) Le partage des documents produits

En ce qui concerne le partage des différents documents produits, nous avons opté pour l'utilisation d'un service de stockage et de partage de fichier en ligne. Ayant tous un compte Google, notre choix s'est rapidement porté sur Google Drive. L'intérêt de ce service pour nous vient du fait que les documents mis en ligne sont modifiables par toutes les personnes avec qui ils sont partagés et ce en même temps. Un historique de modification est disponible et permet de revenir à des versions antérieures. Nous avons principalement partagé des documents texte (cahiers des charges, documents d'analyse, ce même rapport), des images et notre présentation. Il était vraiment indispensable pour nous que l'intégralité de ces documents soient disponibles et modifiables par toute l'équipe et ce tout au long du projet.

### *3) Le versionning*

Il était important de se poser les bonnes questions avant de se lancer dans le développement de notre application. Des questions telles que:

- qui a modifié le fichier X, il marchait bien avant et maintenant il provoque des bugs ?
- qui a ajouté cette ligne de code dans ce fichier ?
- à quoi servent ces nouveaux fichiers et qui les a ajoutés au code du projet ?
- quelles modifications avions-nous faites pour résoudre le bug de la page qui se ferme toute seule ?

C'est là qu'est intervenue la notion de logiciel de gestion de versions. De tels logiciels sont utilisés principalement par les développeurs et sont quasi exclusivement utilisés pour gérer des codes sources et garder les anciennes versions de chacun d'eux.

De plus, ils proposent des fonctionnalités utiles tout au long de l'évolution d'un projet informatique et qui les rendent d'autant plus intéressants :

- ils sont capables de dire qui a écrit chaque ligne de code de chaque fichier, qui les a modifiés et dans quel but.

- si deux personnes travaillent simultanément sur un même fichier, ils sont capables de fusionner leurs modifications et d'éviter que le travail d'une de ces personnes ne soit écrasé et donc perdu.

Ces logiciels dit logiciels de gestion de version de type centralisés avaient donc un grand intérêt pour nous, afin de pouvoir suivre l'évolution de notre code source tout en ayant la possibilité de revenir en arrière en cas de problème et de pouvoir surtout travailler dessus à plusieurs sans risquer de perdre des informations.

Il en existe plusieurs sur le marché tels que Mercurial, Bazaar, Git, etc… N'ayant pas eu le temps de nous pencher sur une comparaison poussée de chacun d'eux, Jérémy nous a conseillé d'utiliser Git, car il avait déjà travaillé dessus par le passé et qu'il pouvait facilement nous y initier.

Git

Git présente de grands avantages et quelques inconvénients (figure 4) :

| Avantages | Inconvénients |
|---|---|
| Rapide | |
| Travail par branches | |
| | Complexe d'utilisation |
| Interface graphique | |

Figure 4 : Avantages et inconvénients de Git

Comme indiqué ci-dessus, c'est un logiciel assez complexe. Il faut donc un certain temps d'adaptation pour bien le comprendre et le manipuler, mais c'est également valable pour les autres outils.

Concernant l'existence d'interface graphique que ce soit sur Linux, Windows et MacOS, elles permettent de simplifier le travail des développeurs en évitant d'avoir à passer par des lignes de commandes pour envoyer et recevoir les différentes modifications apportées.

Une des particularités de Git est l'existence de sites web collaboratifs basés sur ce logiciel. GitHub, par exemple, qui est très connu et utilisé par de nombreux projets (jQuery, Symfony, Ruby on Rails…). C'est une sorte de réseau social pour développeurs : il est possible de regarder tous les projets évoluer et décider de participer à l'un d'entre eux. Il est également possible d'y créer son propre projet : c'est gratuit pour les projets open source et il existe une version payante pour ceux qui l'utilisent pour des projets propriétaires.

Au final, c'est donc sur GitHub que notre choix s'est porté afin d'héberger notre projet Pixelly (figure 5).

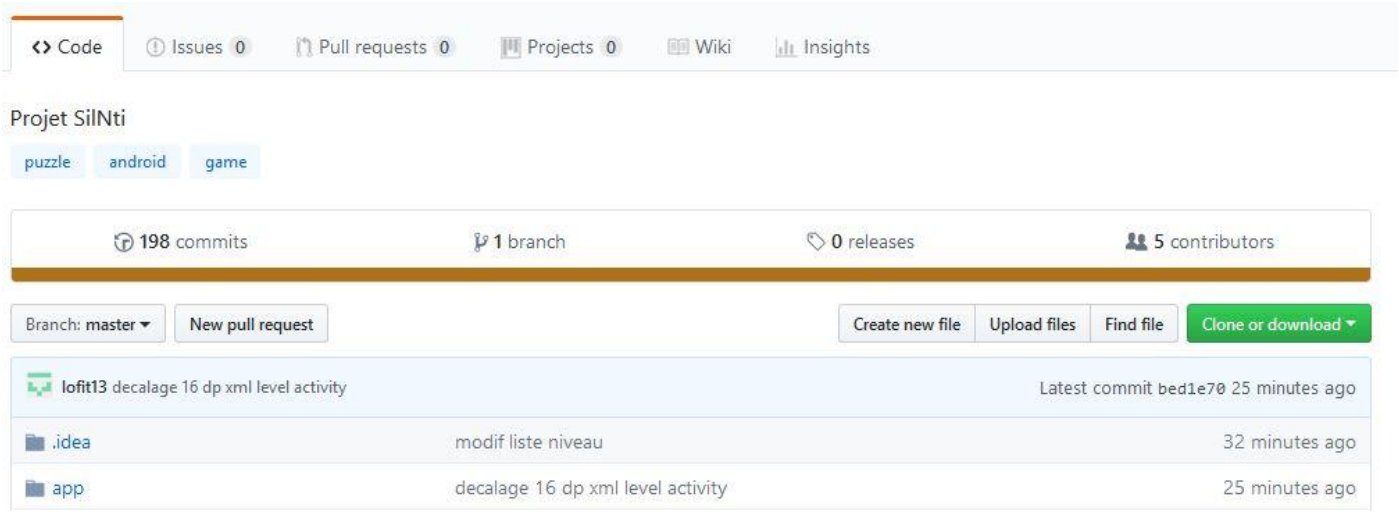



*Figure 5 : extrait de notre GitHub*

Les problèmes rencontrés

Au dépend du temps qu'il nous était accordé pour développer cette application, nous avons opté pour la version "desktop" de GitHub. Ainsi, l'interface est bien plus intuitive que par ligne de code pour mettre à jour le

projet. Cependant, l'interface ne gère pas encore assez bien le fait que plusieurs personnes modifient un même fichier en même temps. Il n'est pas possible de décider pour chaque ligne laquelle garder ou non.

Nous avons donc solutionné ce problème en nous tenant systématiquement informé lorsque l'on commençait à éditer un fichier et lorsque l'on envoyait les modifications apportées sur ce dernier afin d'éviter que quelqu'un d'autre n'écrase notre travail. Ceci aurait bien entendu, très bien pu se gérer avec le terminal mais appréhender cette manière de faire aurait été trop longue et l'on aurait débordé sur le temps disponible pour projet.

# III. Analyse

Etant donné que notre projet se focalise sur l'appréhension des bases de développement sous Android, il est important d'expliquer rapidement ce qu'est Android.

Il s'agit d'un système d'exploitation mobile, respectant les principes de l'Open Source. Il utilise des bibliothèques open source puissantes comme par exemple SQLite pour les bases de données et OpenGL pour la gestion d'images 2D et 3D. La réalisation d'application sous Android est totalement gratuite sauf s'il s'agit de les commercialiser. Quant au développement Android, il repose sur le langage Java et est couplé avec le langage XML pour la mise en place des divers éléments graphiques des applications.

## *1) Le choix de l'environnement de développement (IDE)*

Pour développer en Java, il existe plusieurs IDE, cependant 2 sortent du lot. A savoir : Eclipse et Android Studio. D'après les recherches que nous avons effectué, Eclipse aurait sûrement été l'IDE idéal si notre projet aurait été développé avant 2015, et pour cause, en juin 2015 la firme de Mountain View a cessé le support et le développement d'Android Développer Tools (ADT) pour Eclipse, dont le plugin ADT et Android Ant.

Tout ceci dans le but de concentrer ses efforts sur Android Studio. Quant à ce dernier, il s'agit de l'environnement de développement officiel pour tout ce qui touche à Android. Il a été publié en 2014 par Google pour remplacer les outils de développement Android d'Eclipse (ADT).

Nous avons effectué de nombreuses recherches afin d'évaluer les avantages et les inconvénients de chacun afin de pouvoir choisir au mieux, l'IDE qui nous correspondrait (figure 6).

| | ADT (Eclipse) | Android Studio |
|---|---|---|
| **Facilité d'installation** | ~ | + |
| **Langues** | + | ~ |
| **Performance** | ~ | + |
| **Système de build** | ~ | + |
| **Complétion et Refractoring** | ~ | + |
| **Editeur d'interface graphique** | + | + |

*Figure 6 : Tableau représentant les avantages et inconvénients entre les deux IDE : Eclipse et Android Studio*

Ce sont les performances, l'aspect complétion et refractoring et le système de build Gradle qui a principalement guidé notre choix vers Android Studio. En effet, étant donné que nous n'avions jamais développé sur Android jusqu'à présent, nous voulions d'un IDE qui soit simple d'utilisation et on peut dire que la complétion de code et le refractoring avancé sous Android Studio était très appréciable. De plus, on peut également évoquer notre volonté de découvrir un IDE différent d'Eclipse sur lequel nous avons eu bien le loisir de développer jusqu'à présent. C'était peut-être un risque à prendre de notre part car nous n'avions jamais élaboré de projet sous Android Studio mais c'était réellement un choix qui nous tenait à cœur.

## 2) L'Analyse du projet

En ce qui concerne l'Analyse de notre projet, nous avons opté pour une analyse s'inspirant fortement de la méthodologie 2TUP (*Two Tracks Unified Process*) afin de modéliser au mieux l'aspect fonctionnel de notre application. Pour y procéder, nous avons réalisé deux types de documents : une étude préliminaire (Annexe 3) et une capture des besoins fonctionnels (Annexe 4).

### a) L'étude préliminaire

L'objectif principal de l'étude préliminaire est de modéliser au mieux le contexte de l'application. Dans celle que nous avons réalisée, nous avons défini un seul et unique acteur principal qui interagit avec l'application. De ce fait, il peut lancer, paramétrer l'application, choisir un niveau et y jouer, et peut quitter l'application (figure 7).

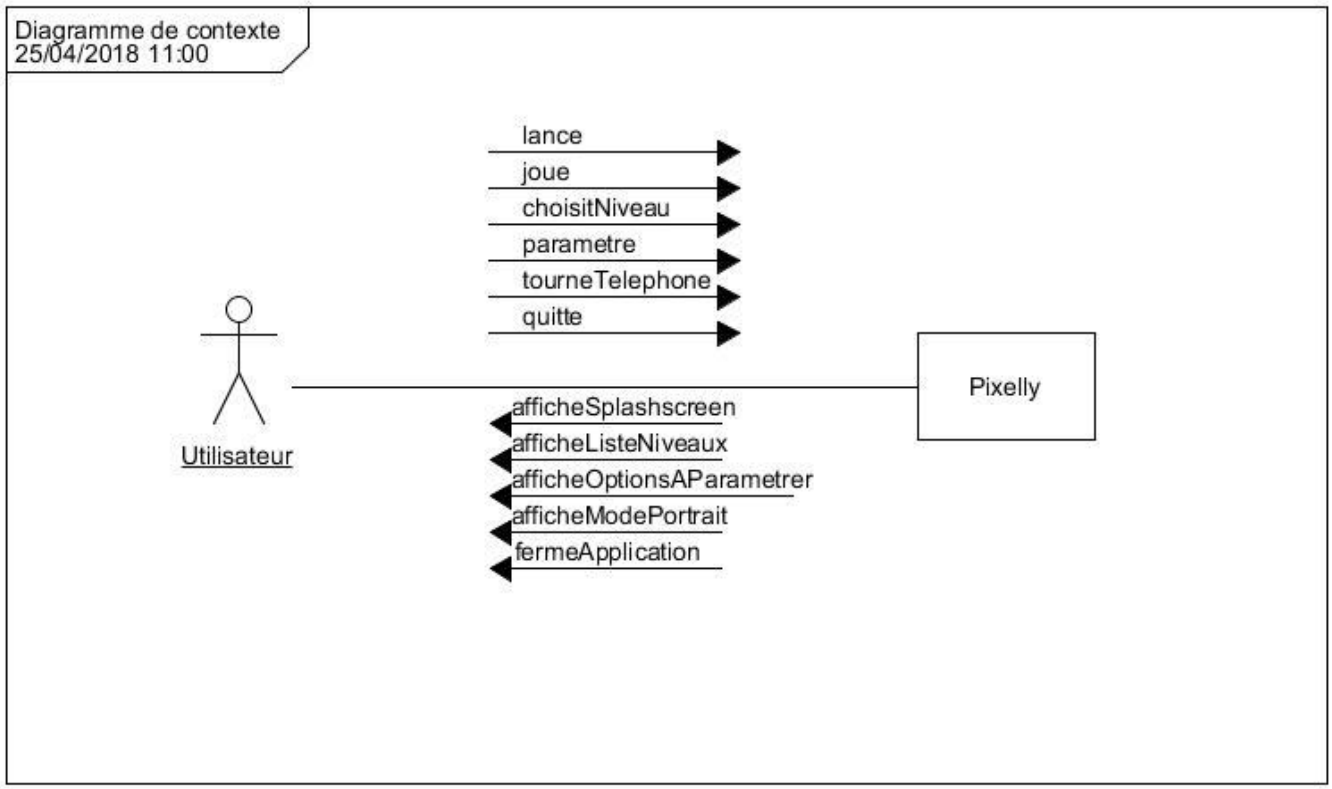



*Figure 7 : Diagramme de contexte*

### b) La capture des besoins fonctionnels

La capture des besoins fonctionnels nous a permis d'approfondir la modélisation du contexte grâce aux diagrammes de use case, de définir les fonctionnalités attendues de la part de l'application via des fiches scénarii, de pouvoir définir le Métier (figure 8) et enfin de pouvoir décrire la dynamique de l'application.

Définition du Métier de l'application

Le modèle métier s'inspire d'un modèle de jeu de plateau composé de cases dont certaines possèdent des caractéristiques particulières. Les cases ont une position x et y dans la grille et une couleur. On distingue ensuite les cases simples des cases de type "terminaison" qui contiennent quant à elle une valeur correspondante à la taille maximale d'un chemin commençant ou finissant par elle.

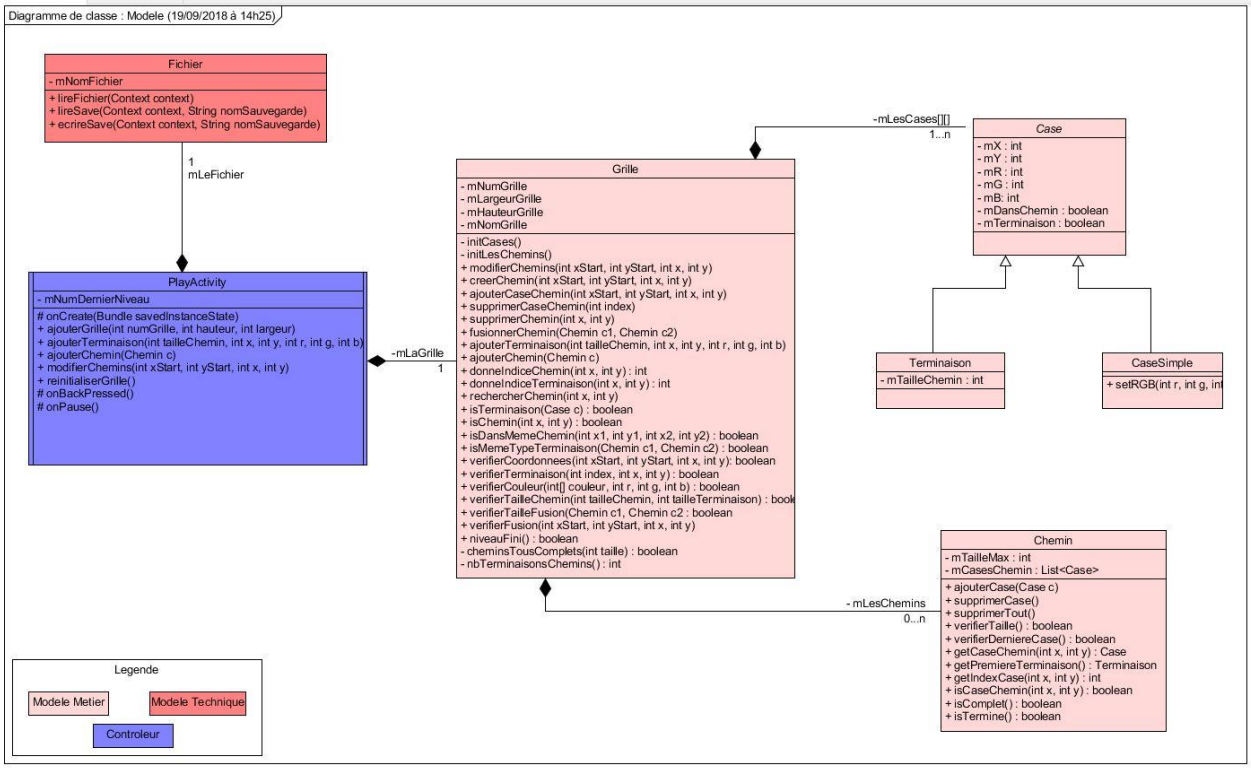



*Figure 8 : Diagramme de classes Métier final*

La grille est la composante principale du Métier qui gère tous les mécanismes de jeu. Elle se compose de la matrice de cases correspondant au plateau de jeu, ainsi que d'une liste de chemins tracés au fur et à mesure du jeu par l'utilisateur. C'est elle qui gère les ajouts, suppressions de cases ainsi que la fusion de deux chemins si cela est possible. Elle a donc pour responsabilité majeure de gérer l'ensemble des règles du jeu.

Dynamique de l'application

Nous avons ensuite défini la manière dont les interactions entre l'utilisateur et l'application devaient se dérouler dans le temps à travers deux types de diagrammes (Annexe 4) :

- le diagramme de séquence, adapté pour la description de la chronologie des échanges entre utilisateur et application.

- le diagramme d'activité, adapté pour représenter le fonctionnement interne de l'application.

# IV. Réalisation

## *1) Bilan*

De manière générale, nous avons réussi à réaliser une grande majorité de fonctionnalités voulues et imposées par le cahier des charges. On peut décrire les fonctionnalités qui étaient à réaliser en trois parties :

- Les fonctionnalités globales et propres à l'application (tableau 1)
- Les fonctionnalités techniques de l'application (tableau 2)
- Les fonctionnalités du jeu lui-même (tableau 3)

Les fonctionnalités de l'application :

| Fonctionnalités | Réalisées | A Améliorer | Non Réalisées | Commentaires |
|---|---|---|---|---|
| **Charger un niveau** | X | | | |
| **Jouer à un niveau** | X | | | |
| **Réinitialiser la grille au sein même d'un niveau** | X | | | |
| **Zoomer sur la grille** | | | X | |
| **Sauvegarder un niveau de manière automatique (quitter l'application ou le niveau)** | X | | | |
| **Débloquer un nouveau niveau une fois le dernier de la liste terminé** | X | | | |
| **Gérer le son** | | X | | *Quand le jeu est en en arrière plan (fond de tâche) la musique continue de tourner* |
| **Réinitialiser le jeu** | X | | | |
| **Accéder à l'aide du jeu** | X | | | |
| **Accéder aux crédits du jeu** | X | | | |
| **Quitter le jeu** | X | | | |

*Tableau 1 : tableau présentant un bilan des fonctionnalités attendues de la part de l'application*

L'application en elle-même est parfaitement fonctionnelle et tout à bien été implémenté. Les seuls points que l'on aurait souhaité améliorer mise à part une meilleure gestion du son, aurait été le design qui, bien que travaillé par nos soins (figure 9), ne sont pas tout à fait à notre goût et s'éloigne un peu de ce qui avait été défini dans le cahier des charges (Annexe 1).


*Figure 9: illustration présentant l'écran d'accueil de notre application Pixelly.*

De plus, nous pouvons préciser que nous avons pris la liberté d'ajouter deux types de menus, absents du cahier des charges mais pourtant essentielle selon nous. Il s'agit de l'aide du jeu et des crédits des développeurs du jeu.

Les fonctionnalités techniques :

| Fonctionnalités | Réalisées | Non réalisées | A améliorer | Commentaires |
|---|---|---|---|---|
| **Implémentation du Design Pattern MVC** | X | | | |
| **Implémentation d'une vue personnalisée (widget)** | X | | | |
| **Lecture de fichier depuis le dossier Assets** | X | | | |
| **Stockage interne** | X | | | |
| **Gestion des crash de l'application (sauvegarde)** | | X | | |
| **Tests Unitaires** | | | X | Les tests unitaires ont été réalisés que sur certaines méthodes de certaines classes et donc pas sur leur totalité, par manque de temps |

*Tableau 2 : tableau présentant un bilan des fonctionnalités techniques attendues de la part de l'application*

Principalement par manque de temps, nous n'avons pas eu l'opportunité de pouvoir gérer les crashs potentiels de l'application et ce que cela impliquait d'un point de vue de la sauvegarde de l'état du jeu. Pour le reste, mise à part les tests unitaires qui auraient dû être plus approfondis, tout ce que l'on souhaitait implémenter d'un point de vue technique a été réalisé.

Les fonctionnalités du jeu :

| Fonctionnalités | Réalisées | Commentaires |
|---|---|---|
| **Ajout d'une case** | X | |
| **Ajout de plusieurs cases à la fois (glissement de doigt sur l'écran)** | X | |
| **Suppression d'une case** | X | |
| **Suppression de plusieurs cases à la fois (glissement de doigt sur l'écran)** | X | |
| **Suppression d'un chemin fini (clique sur n'importe quelle case d'un chemin)** | X | |
| **Suppression d'un chemin non fini (clique sur la terminaison initiant le chemin)** | X | |
| **Fusion de deux chemins s'ils ont la même terminaison initiale et que leurs deux tailles ne dépassent pas la taille maximale autorisée par la terminaison** | X | |

*Tableau 3 : tableau présentant un bilan des fonctionnalités du jeu attendues de la part de l'application*

En ce qui concerne les diverses fonctionnalités du jeu souhaitées lorsque l'utilisateur essaie de résoudre un niveau ont été implémentées et fonctionnent correctement (figure 10).

Nous pouvons également rajouter que l'intégralité des classes Java sont documentées avec de la JavaDoc et que notre application est livrée avec un manuel d'utilisation.

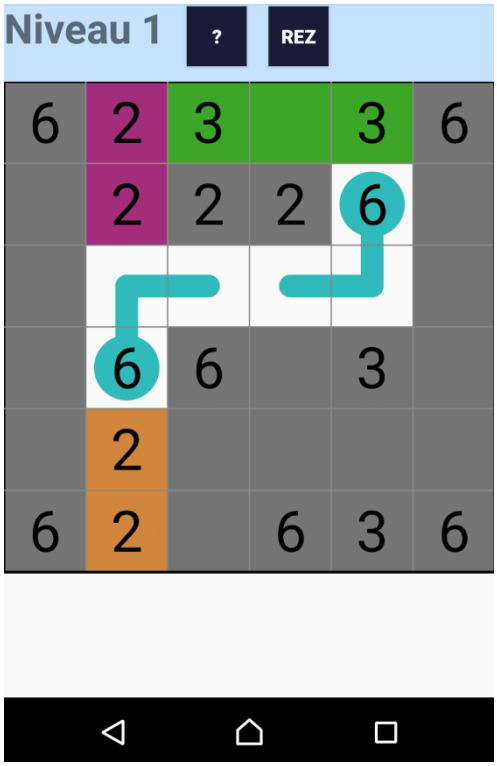



*Figure 10 : illustration présentant un niveau en cours de résolution dans notre application Pixelly*

## 2) Les difficultés rencontrées

Lors de la réalisation du projet, les difficultés rencontrées se sont portées essentiellement sur trois points bien distincts. À savoir, l'implémentation du Design Pattern MVC, l'implémentation de la vue pour représenter au mieux la grille de jeu, ainsi que la sauvegarde de l'état du jeu et de l'avancement du joueur dans chaque niveau.

L'implémentation en MVC

Sous Android, chaque écran que voit et manipule l'utilisateur est géré par une activité, jouant le rôle de contrôleur dans l'architecture MVC. Dans notre application, nous avons ainsi six activités différentes, correspondant à un type de vue différent. L'enjeu de cette implémentation en MVC était donc de comprendre comment faire le lien entre le modèle de l'application et la vue via l'activité et de comprendre comment faire transiter des données entre ces plusieurs activités.

Dans un premier temps, nous avions opté pour l'implémentation d'un singleton couplé à une façade, court-circuitant ainsi le rôle principal de l'activité en tant que contrôleur. Mais suite à une discussion avec notre tuteur, nous avons poursuivi nos recherches dans le but de réussir à implémenter le MVC en se servant cette fois directement l'activité et nous avons fini par y parvenir après plusieurs essais.

L'implémentation de la vue représentant la grille

A partir de l'instant où nous sommes entrés en développement et que les premières classes Java Métier ont été implémentées, la question de la représentation de la grille de jeu s'est très vite imposée à nous. Il nous fallait une grille qui réagisse aux différents glissements de doigts de l'utilisateur sur l'écran, mettant ainsi à jour le Modèle Métier, ainsi qu'une grille qui dessine ces changements en temps réel afin qu'ils soient visibles pour l'utilisateur. Il nous ait donc apparu qu'il nous serait nécessaire d'utiliser une grille qui puisse gérer deux processus en même temps et donc que l'on devrait utiliser un Thread.

La principale difficulté ici a donc été de trouver quelles classes Android utiliser pour dessiner notre grille, de comprendre ensuite comment les implémenter et de gérer les différents évènements liés aux glissements de doigts de l'utilisateur.

Pour le choix des classes, Android permet au développeur de créer leur propre classe de vue, en la faisant hériter de la classe View. Or, cette seule classe ne nous permettait pas de la faire fonctionner avec un Thread. Pour se faire, il nous a fallu utiliser la classe SurfaceView qui héritait elle-même de View. C'est donc de cette dernière que nous avons fini par faire hériter notre classe de vue (figure 11).

En ce qui concerne la classe représentant le thread, nous avons choisi de l'implémenter comme une classe interne à notre classe de vue personnalisée. L'utilisation d'une telle classe se justifie par le fait qu'à terme, l'application est destinée à être exportée sur mobile et qu'elle doit prendre le moins de place possible, ce que permet justement les classes internes. L'utilisation de telles classes est d'ailleurs très courante dans les classes Android elles-mêmes telle que la classe SurfaceView par exemple.

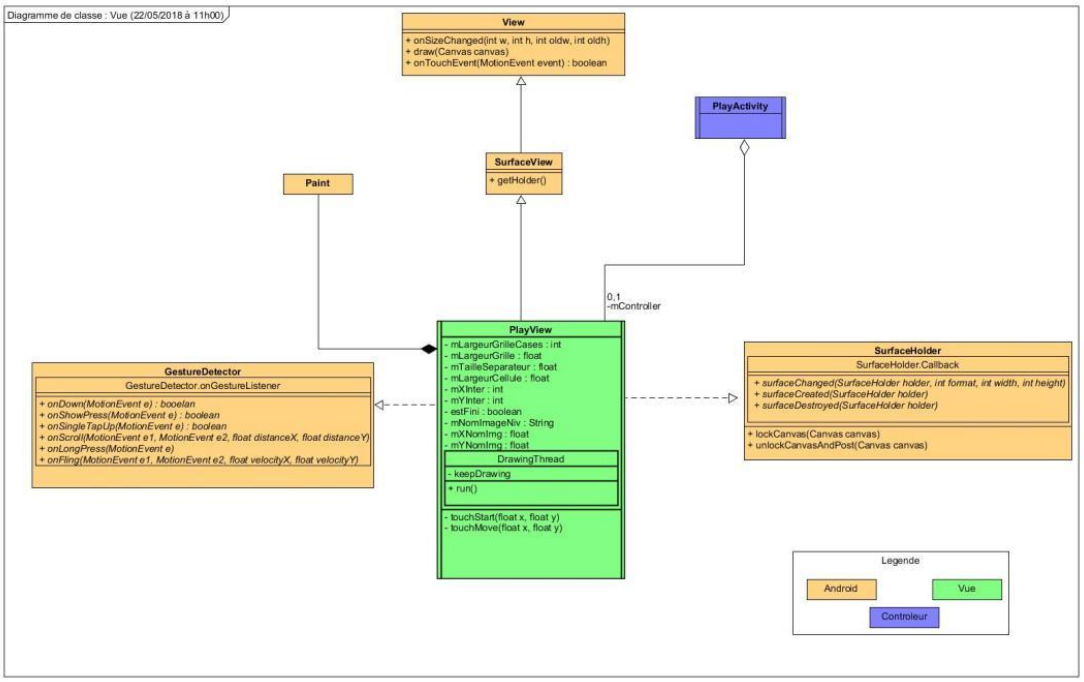



*Figure 11 : Diagramme de classes représentant le widget.*

La sauvegarde de l'état du jeu et des niveaux

La lecture d'un fichier existant est facile à implémenter sous Android, grâce à l'utilisation du dossier 'Assets' spécialement conçu pour stocker des fichiers. Cependant, lorsqu'est venu le moment d'implémenter la sauvegarde

et donc l'écriture, nous nous sommes heurtés à un problème que nous n'avions pas prévu. En effet, le dossier 'Assets' permet de stocker des fichiers qui pourront seulement être lu. On ne peut donc ni en créer et ni modifier le contenu d'un fichier existant. Nous avons donc été dans l'obligation de chercher un autre moyen de procéder, pour stocker les données de sauvegarde de l'utilisateur.

La seule solution que nous avons trouvée pour palier à ce problème a été de stocker ces données de sauvegarde et également d'état du jeu (le numéro du dernier niveau débloqué par l'utilisateur par exemple) dans la mémoire interne attribuée à l'application.

L'avantage est que ces données sont uniques pour chaque utilisateur du jeu. L'un des inconvénients possibles pour l'utilisateur, est que ce système fait en sorte que toute progression sera perdue s'il décidait de désinstaller son application de son téléphone.

Mise à part ces trois points bien distincts, on peut également noter que la documentation d'Android a également été un problème en soit pour nous, surtout lors de l'implémentation de la vue personnalisée pour la grille. En effet, la documentation est moins, voire pas du tout, fournie en exemples, en comparaison avec la documentation de Java fournie par Oracle. De ce fait, il nous est arrivé plusieurs fois de devoir trouver des solutions en ligne sur des forums, alors qu'un exemple sur la documentation aurait suffit à expliquer comment implémenter certaines classes spéciales.

### *3) Les prolongements possibles*

Si nous avions eu plus de temps devant nous, nous aurions souhaité pouvoir générer plusieurs autres niveaux et améliorer le design de l'application pour qu'il soit plus en accord avec ce qui avait été décidé dans le cahier des charges. De plus, nous aurions bien voulu pouvoir réaliser toutes nos fonctionnalités non implémentées décrites précédemment.

### *4) Adéquation avec la licence professionnelle*

La réalisation de ce projet tuteuré, nous a permis d'utiliser des notions et langages vus tout du long de cette année de licence professionnelle.

En effet, la partie Analyse du projet à été réalisée à l'aide de la méthodologie 2TUP enseignée cette année, ce qui a eu pour cause de nous faire gagner du temps sur l'avancement du projet.

De plus, ce projet à été réalisé avec une architecture MVC ( Model - View - Controler ) et l'étude de ce Design Pattern au cours de l'année nous à été d'une grande aide dans la réalisation du projet malgré le fait que son implémentation a légèrement différé de celle que nous avions pu voir jusqu'à présent. Nous pouvons également ajouter que la réalisation de différents projets en langage Java au cours de l'année, nous a été d'une grande aide pour la conception de notre modèle par exemple.

## V. Conclusion

Pour conclure, nous pouvons affirmer que ce projet tuteuré en groupe nous a apporté beaucoup, et ce, autant sur le plan personnel que sur le plan professionnel. En effet, il nous aura appris que l'une des choses les plus importantes pour mener un projet à bien est de savoir communiquer entre les différents membres de l'équipe. Cependant, bien que la structuration et la planification du projet ne semblent pas êtres des plus lucratives et

importantes de prime abord, nous avons appris qu'elles étaient néanmoins nécessaires et primordiales afin d'obtenir un résultat étant le plus proche possible de ce qui était dicté dans le cahier des charges et ce, surtout, dans les meilleures conditions qui soient.

De plus, même si chacun possède des niveaux différents en langage Java, cela ne nous a pas empêché de nous entraider et de nous faire progresser dans la compréhension de certaines notions. Cependant, il ne faut pas oublier que l'apprentissage des langages Java et XML au cours de l'année, nous a aidé à acquérir plus rapidement les bases de programmation sous Android.

## VI. Annexes

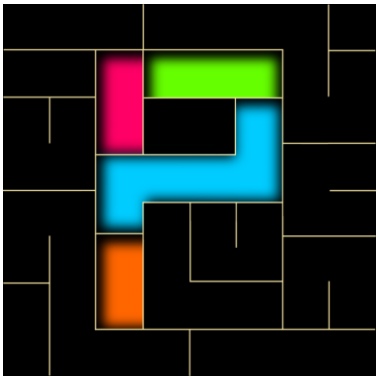
### *Annexe 1 : Cahier des charges*

BrainItAll

BrainItAll

---

# Cahier des charges :
# Pixelly



**Commanditaire : Richard Ostrowski**
Rédigé à Marseille, le 01/03/2018

# Introduction

## À propos de *BrainItAll*

***BrainItAll*** est une mini-entreprise française qui a vu le jour au sein de l'Université d'Aix-Marseille. Elle est spécialisée dans le développement de jeu de type puzzle/logique sous *Android*. Notre équipe se compose de cinq membres de la Licence Professionnelle SIL-NTI : Manon BRUN, Loïc EZRATI, Jérémy LERICHE, Mélodie MEISSEL et Aurélia RAHARISON.

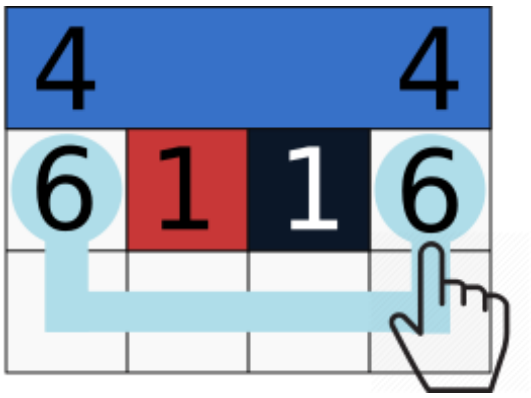
## Objectifs du projet

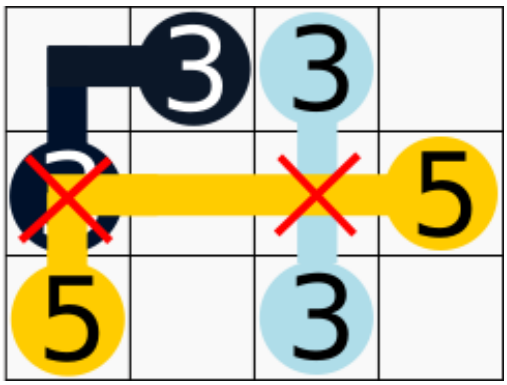
L'objectif principal de ce projet est d'appréhender les bases de la programmation sous *Android* via le développement d'une application type puzzle/logique ou jeu de plateau. Dans notre cas, il s'agira de développer un jeu de puzzle/logique inspiré du jeu *Pixel Pop* se nommant ***Pixelly***.

Le Principe du jeu

Le principe du jeu est de pouvoir remplir l'intégralité d'une grille avec des chemins sachant que :

- Les chemins ne peuvent relier que deux terminaisons, les deux avec le même nombre et la même couleur.
- La longueur des chemins doit correspondre au nombre de points indiqué dans les terminaisons par lesquelles ils doivent commencer ou finir.
- Un chemin ne peut pas croiser d'autres chemins ou passer au travers de terminaisons.




Une fois le puzzle résolu, le joueur découvre une image pixélisée.

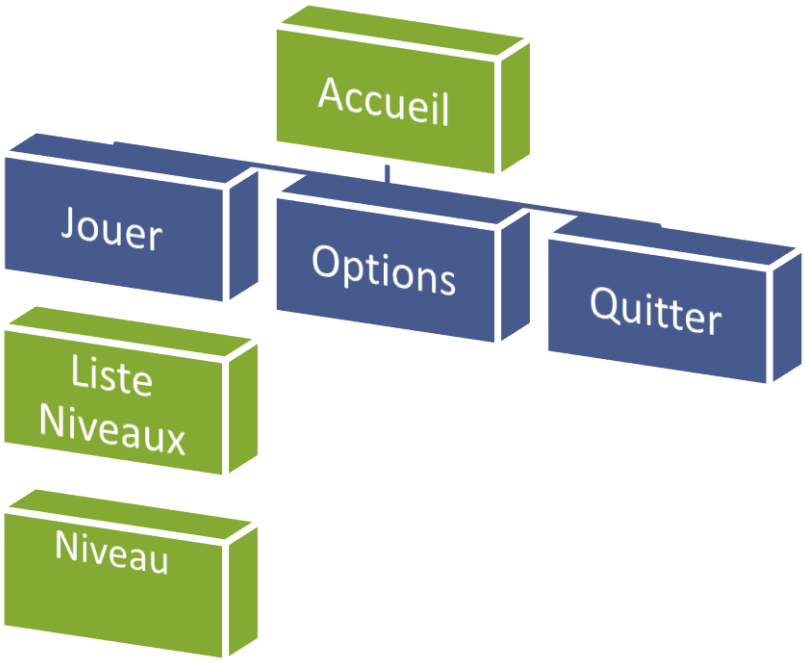
## Audience cible

***Pixelly*** est un jeu s'adressant à un joueur âgé d'au moins 10 ans et appréciant à la fois les jeux de puzzle combiné au pixel art.

# Spécifications fonctionnelles

## Les interfaces de l'application

### Structure de l'application




### Description des fonctionnalités par écran

- **Accueil**

L'utilisateur peut jouer.
L'utilisateur peut paramétrer les options de jeu.
L'utilisateur peut quitter l'application.
L'utilisateur peut avoir accès à une aide concernant le mode d'emploi de l'application.

- **Jouer**

L'utilisateur peut avoir accès à la liste des différents niveaux.

- **Liste Niveaux**

L'utilisateur peut choisir un niveau débloqué et y jouer.

- **Options**

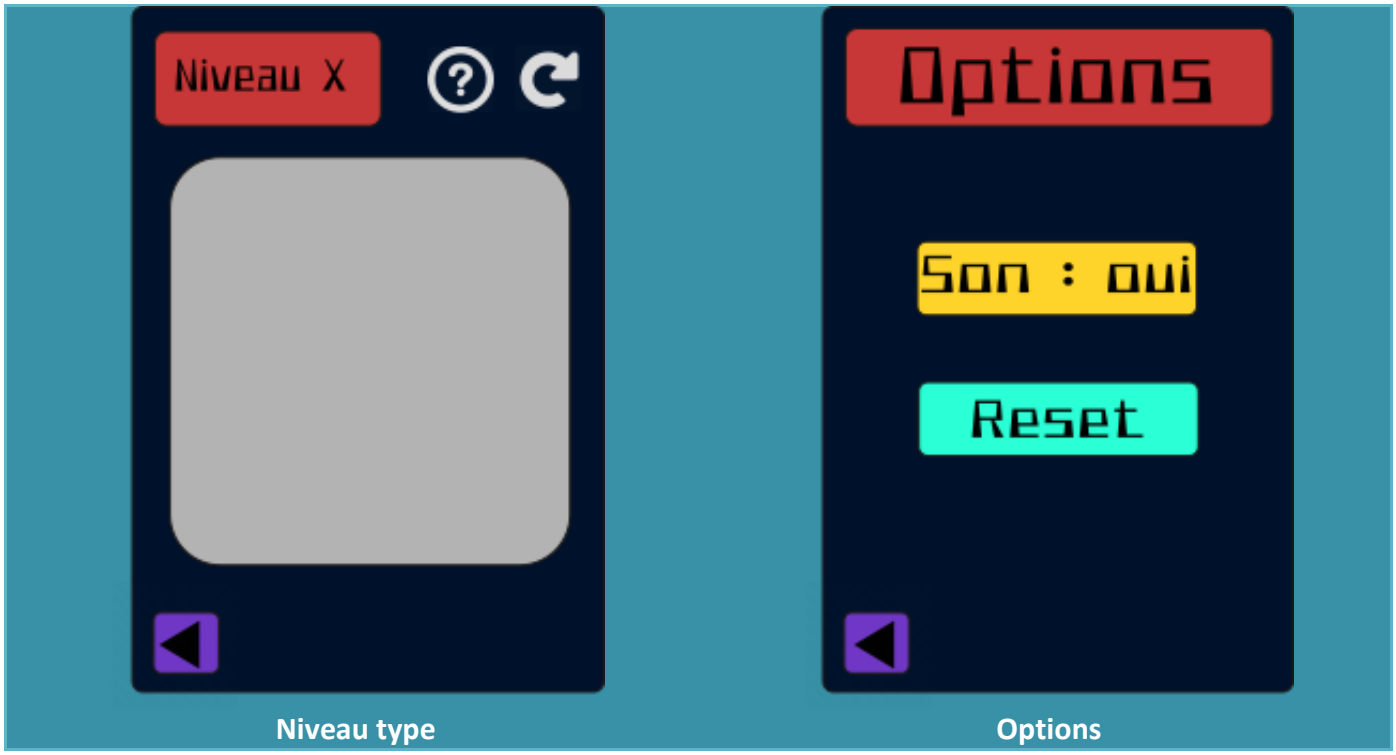
L'utilisateur peut gérer le son de l'application.
L'utilisateur peut remettre les paramètres à zéro.

- **Quitter**

L'utilisateur peut quitter l'application

### Maquettes


**Accueil** **Liste des niveaux**




**Niveau type** **Options**

### Les logiciels utilisés

Le développement de l'application s'effectuera sous *Android Studio*.

La réalisation des images pixélisées sera réalisée en amont grâce au logiciel de dessin *Graphic Gale* avant d'être convertie sous forme de grille.

### Le stockage des données

Les données concernant les différents niveaux (les grilles, et les positions des différentes terminaisons) du jeu seront stockées dans des fichiers au format *JSON*.

## Spécifications non-fonctionnelles

### Utilisation

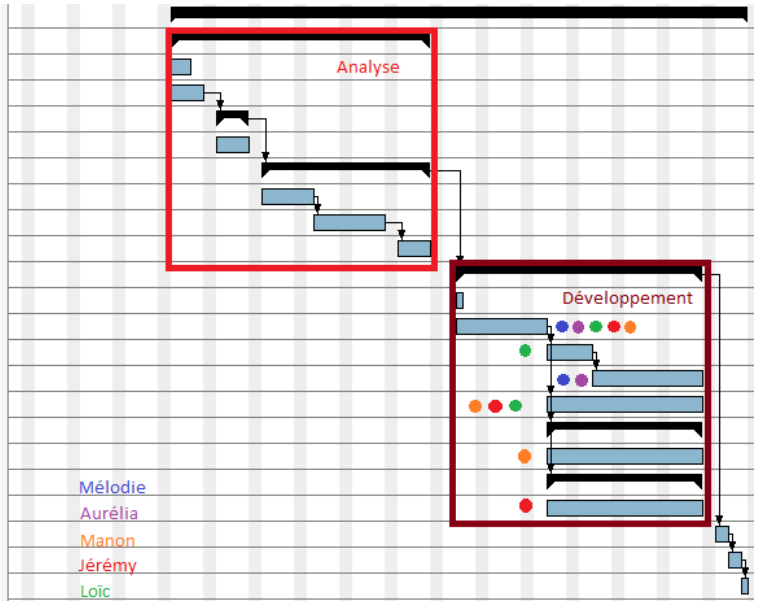
Les utilisateurs ont accès à l'application via leur smartphone strictement *Android*. L'application devra gérer autant le format portrait que le format paysage, les deux.

### Compatibilité

Etant dans l'impossibilité de couvrir la totalité des versions d'Android, nous nous baserons sur les versions les plus utilisées sur le marché actuel ici. De cette table et diagramme, l'on peut en déduire qu'il est intéressant de laisser comme version minimum à notre jeu la 4.4 aussi connu sous le nom de « KitKat ». Le tout se fera depuis le fichier build.gradle préalablement configuré sur AndroidStudio.

# Annexe 2 : Diagramme de GANTT Intermédiaire

| | | |
|---|---|---|
| Projet Tutoré | 26/02/18 | 25/05/18 |
| Analyse | 26/02/18 | 06/04/18 |
| Gantt_Initial | 26/02/18 | 28/02/18 |
| Réalisation_Cahier_DesChar... | 26/02/18 | 02/03/18 |
| étude préliminaire | 05/03/18 | 09/03/18 |
| Diagramme_de_contexte | 05/03/18 | 09/03/18 |
| Capture des besoins fonctio... | 12/03/18 | 06/04/18 |
| Diagramme_Use_Case | 12/03/18 | 19/03/18 |
| Fiches_Scenari | 20/03/18 | 30/03/18 |
| Diagramme_De_Classe | 02/04/18 | 06/04/18 |
| Développement | 11/04/18 | 18/05/18 |
| Gantt_Intermédiare | 11/04/18 | 11/04/18 |
| Documentation_Android | 11/04/18 | 24/04/18 |
| Réalisation_Case_Chemin | 25/04/18 | 01/05/18 |
| Réalisation_Grille | 02/05/18 | 18/05/18 |
| Réalisation_layouts | 25/04/18 | 18/05/18 |
| Fichier_Json | 25/04/18 | 18/05/18 |
| Lecture_ficher | 25/04/18 | 18/05/18 |
| Fichier_Json | 25/04/18 | 18/05/18 |
| Sauvegarde_CheminTracé | 25/04/18 | 18/05/18 |
| Tests et déploiement | 21/05/18 | 22/05/18 |
| Préparation Soutenance | 23/05/18 | 24/05/18 |
| Soutenance finale | 25/05/18 | 25/05/18 |



## *Annexe 3 : Etude Préliminaire*


| *Type de document* | *Document d'analyse* |
|---|---|
| *Version de type* | *0.2* |
| *Date de version* | *23.02.2018* |
| Nom | Etude Préliminaire |
| Version | 0.2 |
| Date | 25.04.2018 |
| Auteurs | Manon BRUN, Loïc EZRATI, Jérémy LERICHE, Mélodie MEISSEL, Aurélia RAHARISON |

**BrainItAll**

*Mini-entreprise spécialisée dans le développement d'applications sous Android*

---

# ETUDE PRELIMINAIRE : PIXELLY

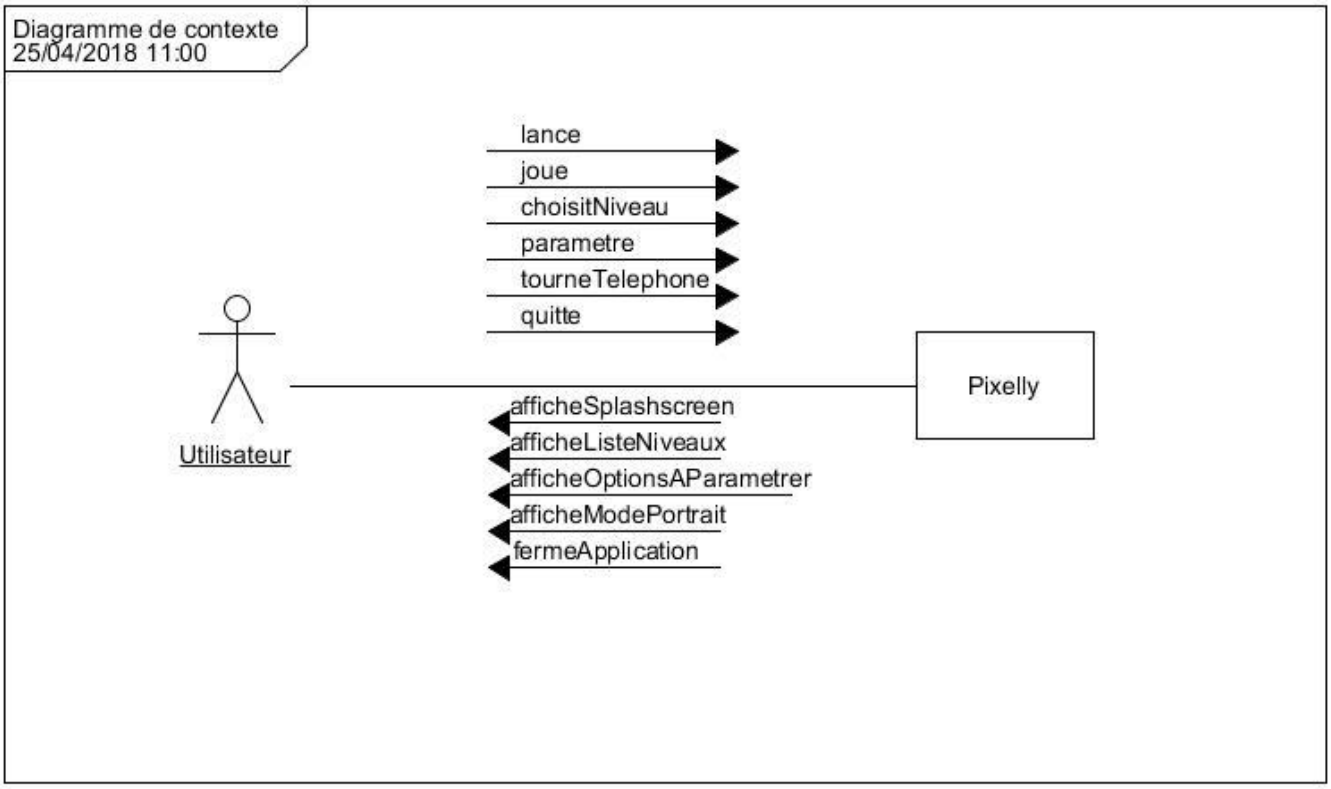
## 1) Les Acteurs

- Utilisateur : un utilisateur peut lancer l'application, paramétrer les options du jeu, choisir un niveau, jouer à un niveau, tourner son téléphone pour changer d'orientation et quitter l'application.

## 2) Les Messages

| Acteurs | Emission (Acteur → Système) | Réception (Système → Acteur) |
|---|---|---|
| Utilisateur | Lance l'application | - Affiche du SplashScreen<br>- Affiche le menu d'accueil |
| | Paramètre l'application | - Affiche les options à paramétrer |
| | Choisit un niveau | - Affiche de la liste des niveaux |
| | Joue à un niveau | - Affiche le niveau sélectionné |
| | Quitte l'application | - Ferme l'appli |

## 3) Diagramme de Contexte



## *Annexe 4 : Capture des besoins fonctionnels*


| *Type de document* | *Document d'analyse* |
|---|---|
| *Version de type* | *0.3* |
| *Date de version* | *23.02.2018* |
| Nom | Capture des besoins fonctionnels |
| Version | 0.3 |
| Date | 22.05.2018 |
| Auteurs | Manon BRUN, Loïc EZRATI, Jérémy LERICHE, Mélodie MEISSEL, Aurélia RAHARISON |

**BrainItAll**

*Mini-entreprise spécialisée dans le développement d'applications sous Android*

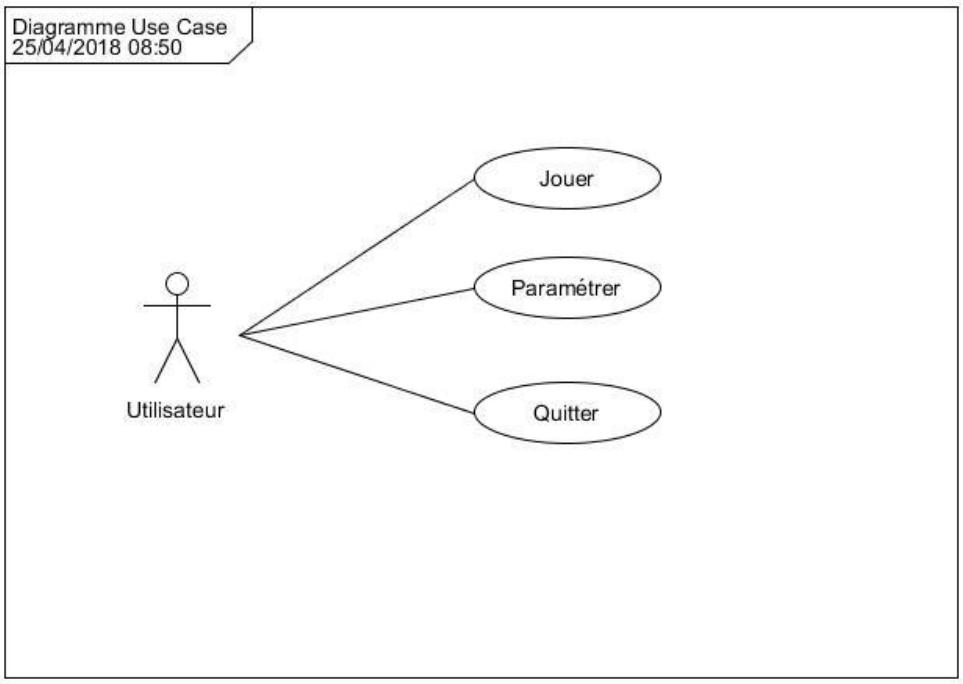
---

# CAPTURE DES BESOINS FONCTIONNELS : PIXELLY

## 1) Liste des cas d'utilisation

| **Nom UC** | **Acteurs** | **Message** |
|---|---|---|
| Jouer | Utilisateur | -lancer jeu<br>-jouer<br>-choisir niveau |
| Paramétrer | Utilisateur | -paramétrer jeu |

## 2) Diagramme des cas d’utilisations



# 3) Fiches scénarii

## FICHE SCENARIO « JOUER »

Date de création : 22/03/2018
Date de dernière modification : 22/05/2018
Version : 0.3
Auteurs : Manon BRUN, Loïc EZRATI, Jérémy LERICHE, Mélodie MEISSEL, Aurélia RAHARISON

Pré-conditions :

Cas nominal :

1) Pixelly propose toutes les possibilités de jeu

2) L'utilisateur joue

3) Pixelly donne l'accord au joueur

4) L'utilisateur quitte le jeu.

Variantes :

**2.a)**
1. Pixelly propose une liste de niveaux.
2. L'utilisateur choisit un niveau parmi ceux débloqués

2.a.2)
1.1. L'utilisateur choisit un niveau non débloqué
1.2. Pixelly empêche l'accès au niveau
2.1. L'utilisateur redemande le tutoriel
2.2 Pixelly propose un tutoriel

**2.b)**
1. L'utilisateur termine le niveau
2. Pixelly affiche le nom de l'image du niveau et débloque le niveau supérieur

2.b.2)
1. Pixelly n'a plus de niveaux à débloquer : fin du jeu.

**2.c)** L'utilisateur trace un chemin complet (terminaisons)

2.c.1)
1. L'utilisateur trace une partie du chemin

2.c.2)
1. L'utilisateur efface un chemin

2.c.3)
1. L'utilisateur trace un chemin sur un autre
3.1. Pixelly ne donne pas l'accord au joueur

2.c.4)
1. L'utilisateur trace un chemin sur une mauvaise terminaison
3.1. Pixelly ne donne pas l'accord au joueur

2.c.5)
1. L'utilisateur trace un chemin dont la taille est plus grande que celle indiquée
3.1. Pixelly ne donne pas l'accord au joueur

2.c.6)
1. L'utilisateur commence un trait sur une case vide
3.1. Pixelly ne donne pas l'accord au joueur

2.d)
1. L'utilisateur redemande le tutoriel
2. Pixelly propose un tutoriel

2.e)
1. L'utilisateur réinitialise le niveau
2. Pixelly affiche la grille initiale

2.f)
1. L'utilisateur retourne à la liste des niveaux
2. Pixelly propose une liste de niveaux tout en sauvegardant la progression du niveau que l'utilisateur vient de quitter

Exceptions :

Post-conditions :

# FICHE SCENARIO « PARAMETRER »

Date de création : 22/03/2018
Date de dernière modification : 25/04/2018
Version : 0.2
Auteurs : Manon BRUN, Loïc EZRATI, Jérémy LERICHE, Mélodie MEISSEL, Aurélia RAHARISON

Pré-conditions :

Cas nominal :
1) Pixelly propose toutes les possibilités de jeu

2) L'utilisateur choisit de paramétrer

3) Pixelly donne l'accord au joueur

4) L'utilisateur retourne à l'écran d'accueil.

Variantes :

2.a)
1. L'utilisateur coupe/active le son

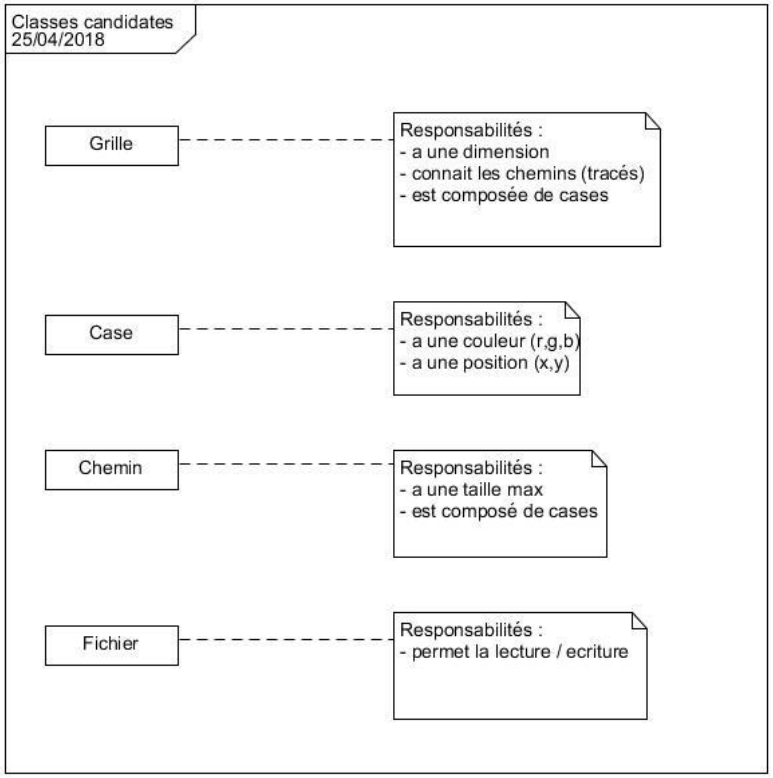
2.b)
1. L'utilisateur reset le jeu
2. Pixelly supprime toute la progression de l'utilisateur

Exceptions :

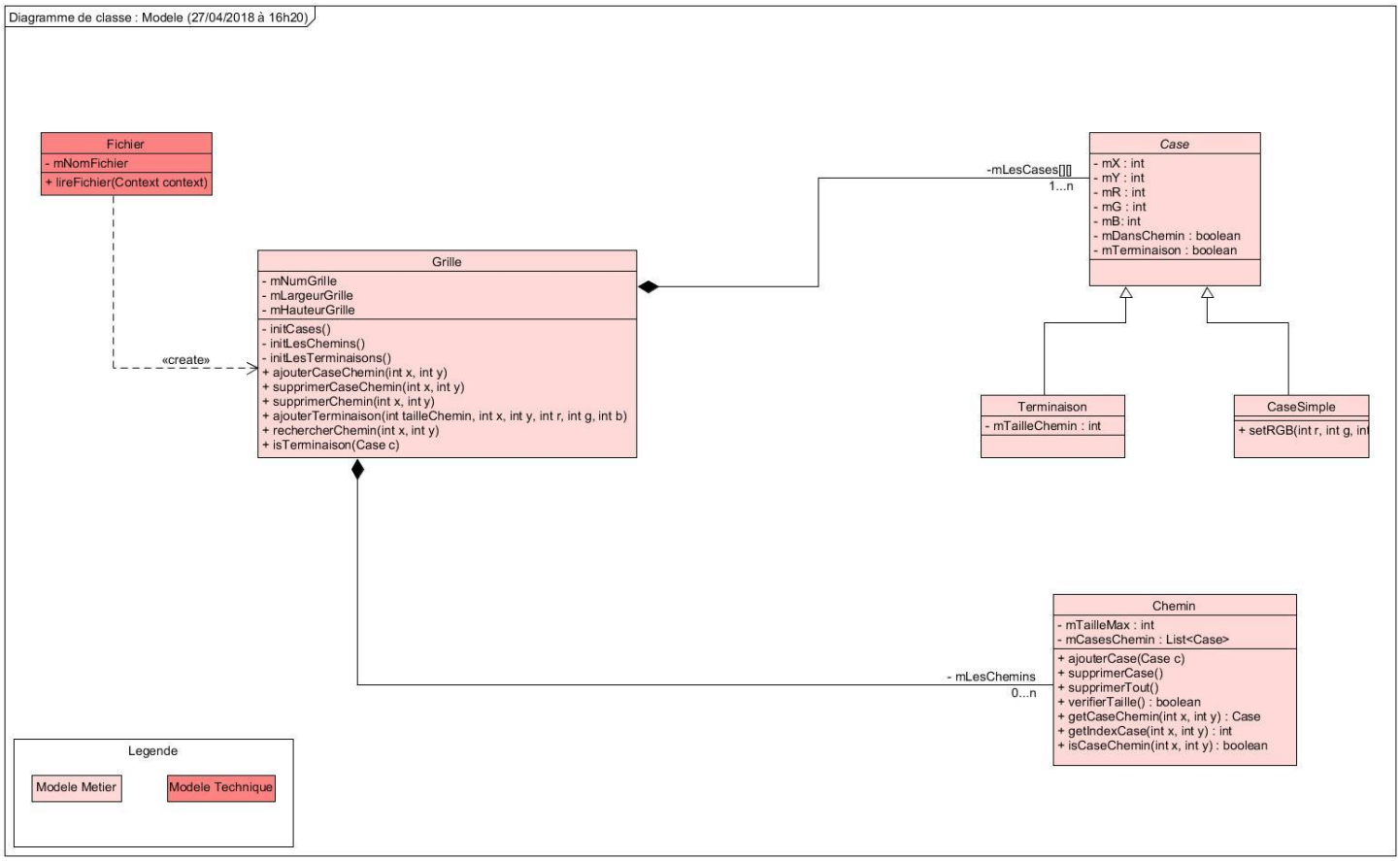
Post-conditions :

# 4) Diagramme de classes métiers

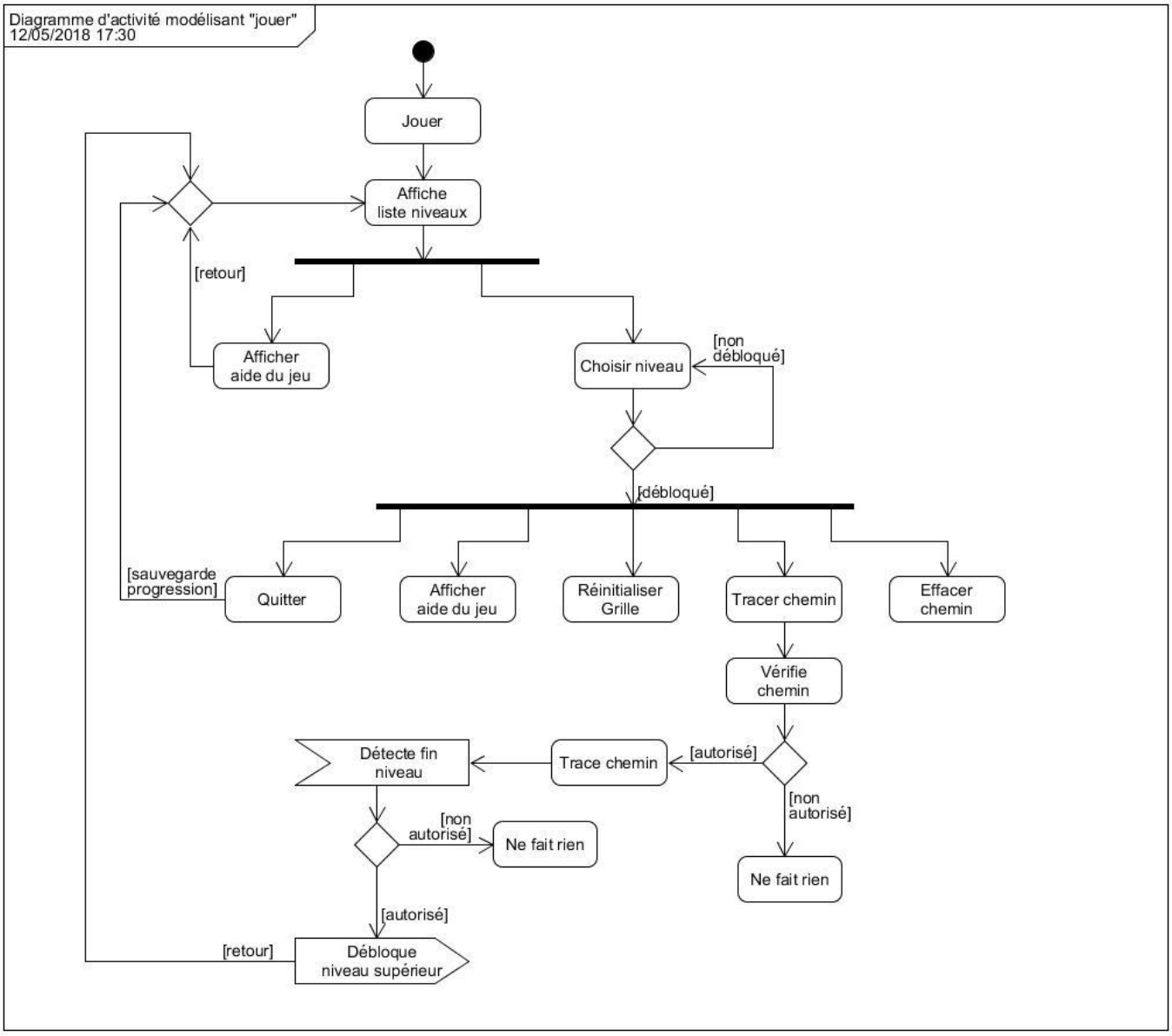
## Classes candidates :




## Diagramme de classe :

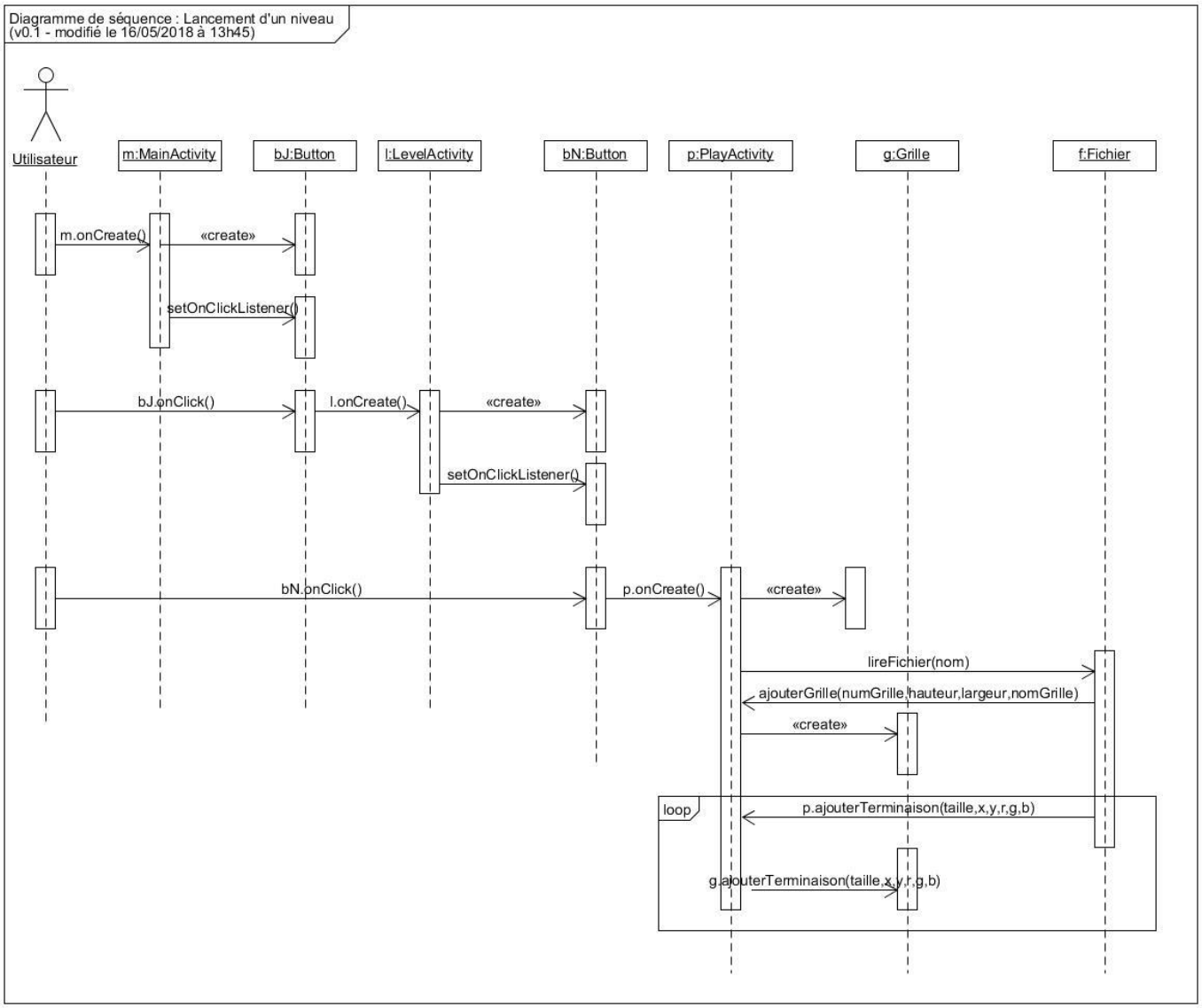


# 5) Diagramme d'activité de « jouer »



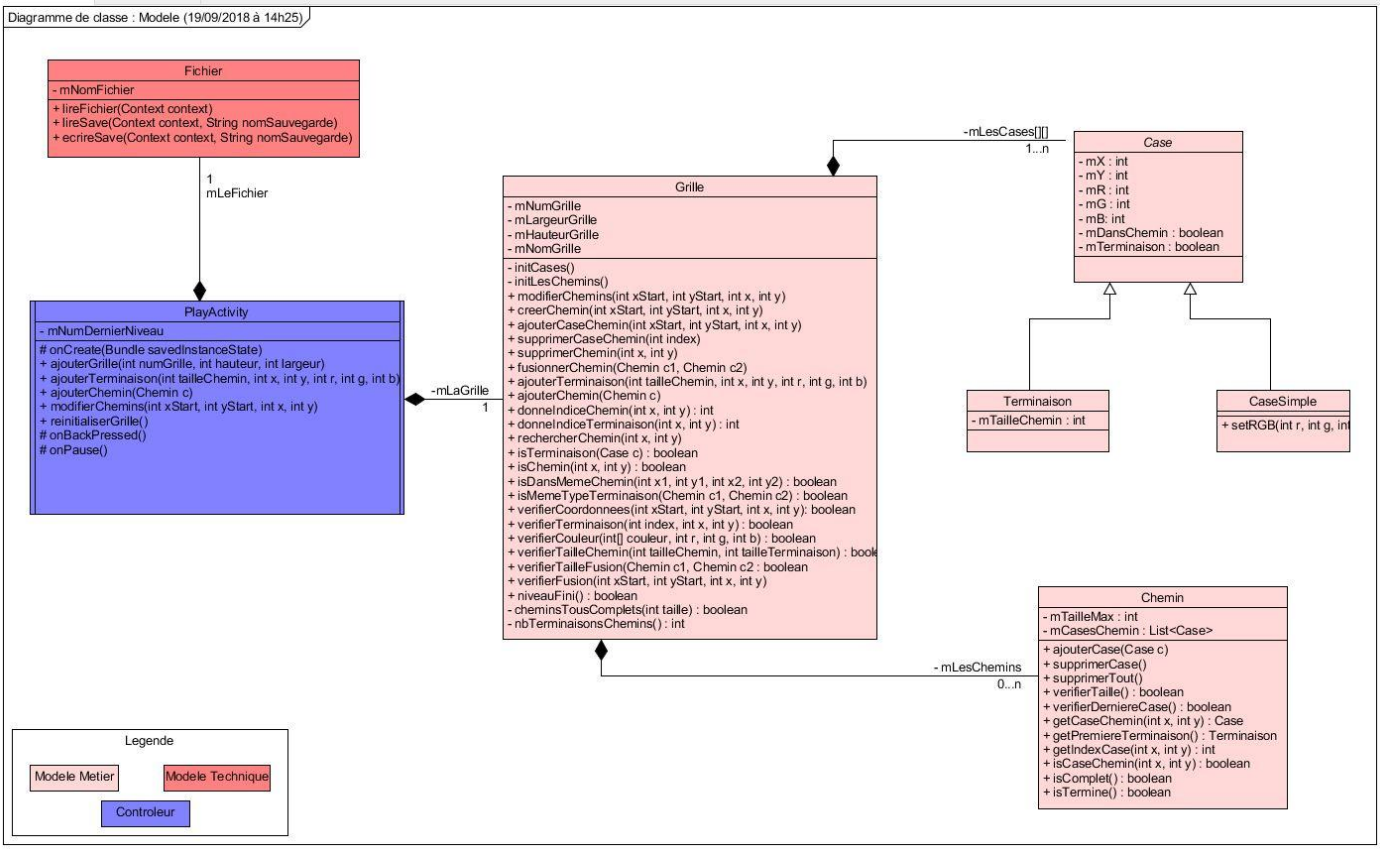
# 6) Diagramme de Séquence

- **Chargement du niveau**



# Annexe 5 : Diagrammes de classes (Modele + Vue) finaux

## Diagramme de classe Modèle



## Diagramme de classe Vue