Sciences de l'informatiques | Projet d'informatique : BABA IS YOU

1ère Bachelier

(Arnaud De Lena)
Romain Eloy
Julien Ladeuze

Enseignant : Hadrien Melot

2021-2022

UMONS
Université de Mons

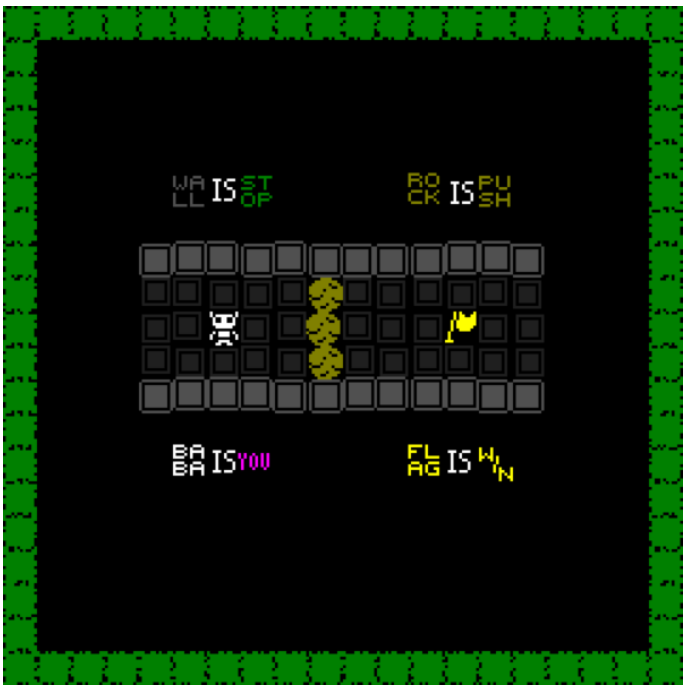
Table des matières

1 INTRODUCTION

Présentation du jeu

Dans le cadre de notre cours « projet d'informatique », il nous est demandé de reproduire un jeu vidéo. Ce jeu a été fait pour la première fois durant le Nordic Games Jam (Un endroit où les développeurs peuvent exprimer leurs talents sur des jeux vidéo) de 2017 où il a d'ailleurs remporté le prix. Ce jeu est « Baba Is You ». Un jeu vidéo de style « Sokoban*» puzzle/casse-tête. La particularité de ce jeu est que les règles se trouvent elles-mêmes dans le niveau et peuvent être déplacées afin de réussir la partie. Ceci est l'essence même du jeu.

A noter que des fonctionnalités de bases devaient être implémentées : chargement d'une carte à partir d'un fichier texte, système de sauvegarde, implémentation des 4 premiers niveaux de la version jam*



(Image issu de la version jam)

*https://fr.wikipedia.org/wiki/Sokoban (explication de Sokoban)

*https://hempuli.itch.io/baba-is-you (lien de la version jam)

Répartition des tâches

-Romain : partie interface graphique (View), score, texture, rapport

-Julien : partie logique (Model), sauvegarde, utilisateur, accès niveau, rapport

-Arnaud : absent (dernier commit sur github : 01/03/2022)

2 STRUCTURE DU PROJET

Lors du cadre de notre projet, nous avons eu un cours concernant l'architecture MVP (Model View Presenter). Nous avons décidé d'adopter cette architecture car elle convenait bien pour l'implémentation de l'application. Les parties du MVP seront présentées dans les points suivants.

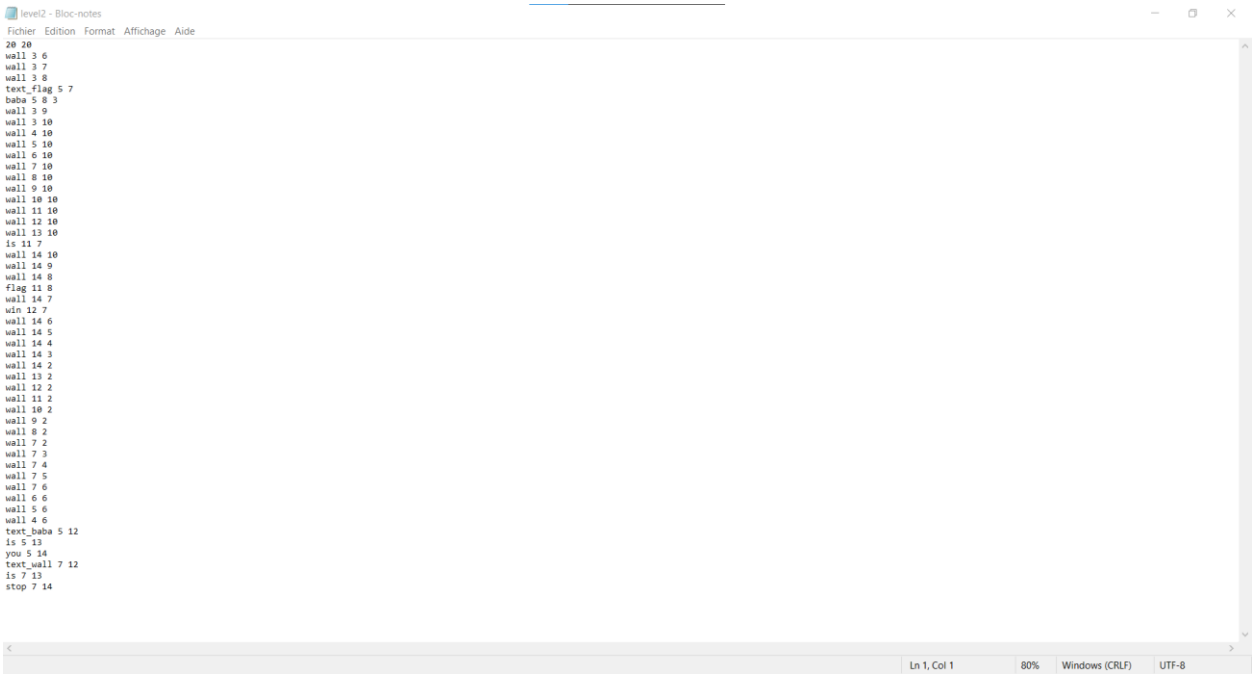
2.1 Model

La partie Model correspond à la « machinerie » de l'application. C'est ici que le travail de données va se jouer.

Fonctionnalités de bases

Niveau = fichier texte

Comme dit dans l'introduction, nous avons dû implémenter les niveaux comme des fichiers texte d'un format bien précis voir ci-dessous.



Ce fichier représente le 2eme niveau du jeu.

La première ligne représente la taille de la carte. Dans ce cas-ci, 20 en longueur et 20 en largeur. Pour le reste, on y trouve des éléments du jeu comme baba. A côté de leur nom, on y trouve les coordonnées associées en x et y. La quatrième valeur possible est un chiffre compris entre 0 et 3 représentant la direction initiale de l'objet (dans notre cas, on ne l'a pas utilisé car on y a pas trouvé d'utilité).

Pour implémenter tout ça dans notre jeu, nous avons eu recours à un tableau à double dimensions de type String contenant les données du fichier. Nous avons fait ce choix car le développement de cette partie s'est fait aux alentours du cours de programmations et algorithmiques 2 sur les tableaux.

La structure du tableau se fait de cette manière. Un sous-tableau représente une ligne du fichier. Celui-ci est de taille 4 pour accueillir le maximum d'information et toujours dans le même ordre (sauf la premier sous tableaux <-> 1 ère ligne du fichier) : indice 0 : élément du jeu, indice 1 : coordonné en x, indice 2 : coordonné en y (et indice 3 : direction initiale).

L'initialisation se fait dans la classe « Extract » qui est indépendante des autres classes. Après que le tableau aura été initialisé, on l'appliquera dans la classe « Map » (à voir par la suite).

L'emplacement de ces fichiers se trouvent dans le répertoire « default» et save (resources/level/default(save)).

Sauvegarde = l'inverse de la mise en place d'un niveau

Dans le cas d'une sauvegarde, on va devoir enregistrer la carte du jeu à partir du niveau en cours (modifié ou non). Étant donné que notre carte est faite à partir d'un tableau (voir plus bas), on va tout simplement prendre le nom de l'objet dans le tableau et ses coordonnées (x,y). On va placer tout ça dans un fichier texte de la même manière que le fichier représenté ci-dessus.

Toutes ces sauvegardes se situent dans le dossier « save » (resources/level/save) dans lequel on peut retrouver un fichier « history.txt ». Ce fichier contient le nom des sauvegardes et la date à laquelle elles ont été prises. Ceci permettra de récupérer la dernière sauvegarde en priorité car elle se trouve à la fin du fichier (pour reprendre une partie).

Les 4 niveaux obligatoires

Les 4 premiers niveaux se situant dans le dossier « default » sont ceux imposés par l'énoncé.

Principes OO utilisés

Voici une liste exhaustive des concepts orientés objets utilisés : Classe, classe interne, encapsulation, héritage, interface, polymorphisme et l'énumération.

Où ?

-Classe : partout

-Classe interne : dans la classe BlockRules

-encapsulation : partout où il y a des attributs

-héritage : de la classe Environment aux classes enfantes d'Item

-interface : les classes qui l'implémentent sont Item et BlockRules

-polymorphisme : Environment

- énumération : dans la classe Rules

Pourquoi ?

Le concept de classe est l'essence même du langage Java du POO. En plus de ça, l'orienté objet est un paradigme plus simple pour faire un jeu que le procédural.

Pour rappel, une classe interne est une classe se trouvant dans une autre classe. Ici, la notion de classe interne a été utilisé pour une meilleure gestion de classe et de lisibilité au niveau du code.

L'encapsulation des données est un moyen de protéger ses variables de classe d'un autre programmeur. Sans cette encapsulation, la personne pourrait modifier les attributs sans passer par une méthode au préalable. Cela pourrait être problématique dans le cas le programmeur mettrait qu'il est tout le temps gagnant dans le jeu…

L'héritage est un moyen de factoriser du code et de rendre de plus en plus spécifique les classes via l'ajout de méthode, d'attribut, etc.… Ceci nous a fait gagner un temps considérable dans la réalisation du projet.

Une interface est une sorte de caractéristique attribuée à différents objets. Plus précisément, une interface est un contrat régi par un ensemble de méthode à implémenter dans les classes qui utilisent cette interface. Ceci nous a permis de cataloguer nos classes et d'utiliser le concept de polymorphisme.

Le polymorphisme peut rendre plus accessible certaines classes. Dans notre cas, le polymorphisme a été utilisé pour représenter tous les objets dans une même catégorie (voir plus bas).

L'énumération est un moyen de représenter quelque chose sans lui attribuer aucune caractéristique et méthode. Cela nous a permis de manipuler nos règles de la manière la plus simple possible.

Comment ? (Interface, héritage, polymorphisme, énumération)

La structure du model se présente comme ceci :

On retrouve en amont une interface (Entity) représentant tous les objets disponibles dans le jeu. Cette interface comprend en majorité des méthodes booléennes qui font office de caractéristique pour les objets les implémentant.

A côté d'Entity, on retrouve Environment. Une classe qui crée la carte du jeu. Cette carte est un tableau de type Entity. On y retrouvera à l'intérieur des classes implémentant cette interface. C'est comme ça que le concept de polymorphisme est utilisé.

Ensuite, on retrouve deux grandes classes : Item et BlockRules. Comme son nom l'indique, BlockRules est une classe représentant tous les blocs de règles du jeu. Quant à Item, c'est tout le reste. Autrement dit, tous les objets représentés par les règles.

Dans BlockRules, on y retrouve des classes internes. Ces classes internes descendent toutes de la classe dans laquelle elles sont contenues. Alors pourquoi avoir fait des classes internes ? Dans ce cas-ci, comme les règles ont les mêmes caractéristiques c'est-à-dire qu'elles peuvent être poussées, elles ne permettent pas la victoire (directement), etc… Elles sont quasiment toutes les mêmes à une différence près : leur apparence. Cela permet donc d'avoir moins de fichier .java inutile.

Du côté d'Item, on va retrouver dans cette classe trois choses importantes : l'implémentation des méthodes issues de l'interface, la gestion d'une carte de jeu temporaire et la méthode « move (String input) ».

LES MÉTHODES D'ENTITY

Ces méthodes vont discuter avec la classe « BigAlgorithm ». Cette classe s'occupe de la gestion des règles du jeu. Ces méthodes ont pratiquement toutes le même moyen d'implémentation. Elles vont vérifier la condition de l'objet en fonction des permissions issues de « BigAlgorithm ».

LA CARTE TEMPORAIRE

Comme son nom l'indique, c'est une carte qui ne dure qu'un temps. On va y trouver des éléments qui ont toutes des règles permettant à certains objets de passer au-dessus d'elles. Cette carte permet donc de les stocker en attendant qu'il n'y ait plus rien à leur emplacement sur la carte principale.

LA MÉTHODE : MOVE (STRING INPUT)

Cette méthode est le point central du jeu. Elle va implémenter quasiment TOUTES les autres méthodes d'Item. À l'aide d'un input (entrée représentant un mouvement), la méthode va effectuer tous les changements nécessaires à la carte principale (en plus d'établir et d'actualiser la carte temporaire).

Au plus bas de la structure, on retrouve les classes enfants d'Item comme Baba, Wall, etc.… Ces classes ont toutes le même moyen d'implémentation à deux exceptions près : Leur apparence et la règle qui les représentent.

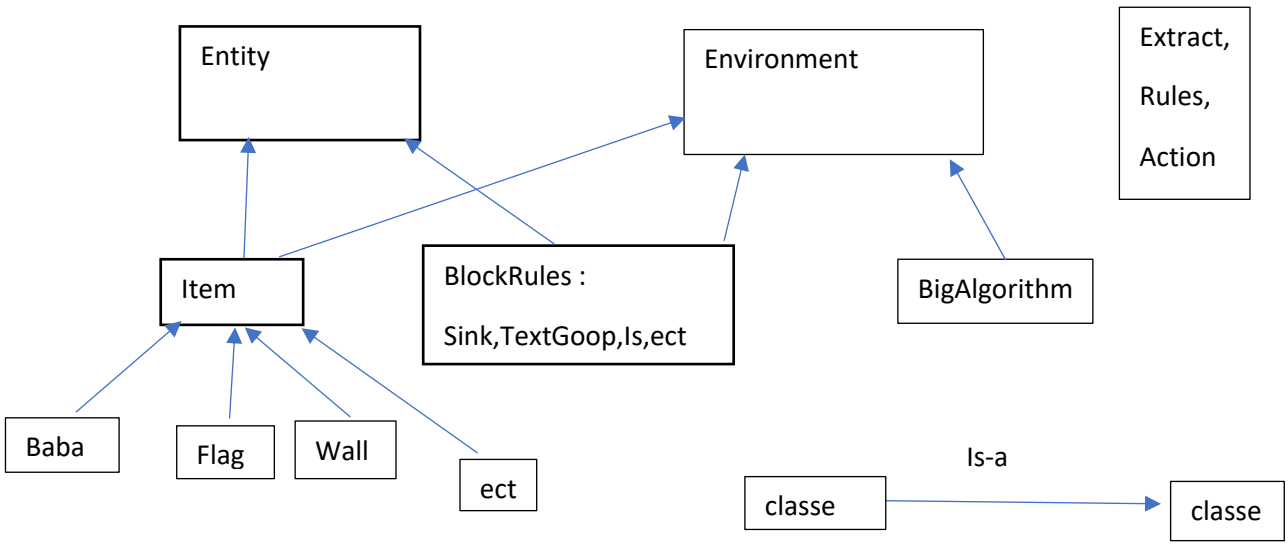
À l'extérieur de ces classes, on retrouve :

La classe Rules est de type « Enum ». Elle permet de représenter les règles du jeu.

La classe Extract qui manipule les fichiers représentant les niveaux.

La classe Action qui représente tous les mouvements possibles du jeu. Autrement dit, pousser et bouger.

Voici la structure du Model :



La classe BigAlgorithm sert à créer un tableau à deux dimensions de type Enum contentant les règles. Ce tableau est présenté de cette manière : {{Rules.BABA, Rules.YOU}}.

Structure de données utilisées

Les structures de donnés utilisées sont : les tableaux, les Arraylists (tableaux dynamiques) et les dictionnaires(HashMap).

Les tableaux sont utilisés dans de nombreux cas mais on peut en retenir un qui est très important : la carte du jeu. Alors pourquoi l'utiliser pour ça ? Tout simplement parce que la taille de notre carte est fixe et d'un type bien particulier. En plus ce ça, ce tableau est à double dimension. Les éléments qui composent le tableau de base sont des sous tableaux (représente la coordonné y). Tandis que les éléments des sous tableaux sont des objets (représente la coordonné en x).

L'arraylist est utilisé pour stocker les différentes coordonnés des objets qui sont « win ». Autrement dit, les objets qu'il faut atteindre pour gagner la partie. Cette structure de données nous permet d'ajouter et de supprimer des objets plus facilement qu'un tableau.

Le dictionnaire est utilisé dans le cadre de faire une correspondance entre deux objets. Dans notre cas, elle est faite pour faire le lien entre un objet de type entité et un objet de type Rules. Cela permet d'éviter des « switch () » un peu trop long (même s'il y en reste dans les autres classes…).

Complexité globale

Notre utiliserons la complexité dans le pire des cas pour cette partie. Comme vous vous en doutez, la méthode qui a la complexité la plus « lourde » est « move () ». Pour des raisons de simplification, les méthodes implémentées dans « move () » sont déjà évaluées et on évalue qu'un « case » du switch vu que c'est la même manière pour tous les autres. (seul l'input change)

Move :

```
switchObject() T(n*2)
setTempObjectMap() T(n*2)
searchWin()  T(n*2)
If(thingIsYou()) T(n)
    switch () T(1)
        for (int i = 0 ; i <= map.length – 1 ; i++) T(n)
            for (int j = 0 ; j <= mapO[i].length – 1 ; j++) T(n)
                If(this.getClass().isInstance(maO[i][j])) T(1)
                    Item item = (Item) mapO[i][j] T(1)
                    If (canMove()) T(1)
                        Action.up() T(1)
                    Else if(thingIsPushing()) T(1)
                        Action.pushY() T(n)
                    If(thingHasWin()) T(n)
                    winStatus = true T(1)
        break T(1)
BigAlgorithm.actualise() T(n*2)
actualiseObjectMap()  T(n*2)
```

Nous avons donc T(n*2 + n *2 + n* 2 + n + n x n x (1 + 1 + n + n + 1)).

Après simplification, nous avons un temps en T(6n*2 + 2n*3 + n).

Ce qui nous donne une complexité en O(n*3).

A noter que la complexité globale est en O(n*2). O(n*3) est la plus « lourde ».

2.2 View

La partie view correspond à la partie interface graphique, dans celle-ci nous allons parler de comment nous avons géré l'interface graphique du jeu. Tout d'abord nous avons utilisé « scene builder » pour avoir plus de facilitée lors du codage.
« Scene builder » nous a permis de pourvoir plus facilement créer les menus du jeu ou placer des images. Ces fichiers sont en « .fxml » et chaque fichier représente une scène du jeu. Les scènes sont contrôlées via Controller.

Controller

La classe Controller possède plusieurs méthodes qui auront un impact en fonction de ce que l'utilisateur fait d'où le nom « Controller ». Par exemple, si le joueur décide de lancer une partie, alors il va devoir cliquer sur un bouton qui va lancer la partie et ceci est de même pour d'autres fonctionnalités ; comme la musique dans le menu « settings » ou voir le score du jeu grâce à un bouton dans le menu. Controller est une classe qui répond aux actions du joueur dans la partie pour les déplacements ou ouvrir un menu pause par exemple.

Main

La classe Main est la classe de lancement du jeu, elle va lancer l'application sur une scène choisie, ici ça sera « connection ».

2.3 Presenter

Le Presenter est « l'échangeur d'information ». En somme, il va récupérer les données du Model et va les transmettre au View (en plus de la partie technique). Dans cette partie, nous allons montrer l'utilité de chaque classe et comment elles fonctionnent

User

La classe User est une classe représentant l'utilisateur quand il joue. Elle va vérifier via des méthodes si le joueur s'est identifié et s'il est nouveau. Dans le cas où c'est un joueur lambda (pas de nom), aucune recherche à faire. Sinon, on va chercher s'il est inscrit. Si l'utilisateur n'a encore jamais testé ce jeu, on l'inscrit.

A noter que le jeu a une capacité maximale d'inscription (10). Dans le cas où le joueur voudrait créer une 11 IIème fois un « compte », il devient un personnage lambda.

Info

Cette classe est ce qui permet de communiquer entre les différentes classes de Presenter et les fichiers texte. Ces méthodes peuvent être distinguées entre deux types : lire et écrire. En effet, certaines méthodes ne font qu'aller dans un fichier et cherche de l'information à propos d'un joueur. A contrario, les autres vont écrire de l'information.

Level

Comme son nom l'indique, cette classe va gérer tout ce qui est sur les niveaux. En d'autres termes, elle va gérer les accès aux niveaux et le changement des niveaux (passer d'un niveau à l'autre).

Score

Le score va permettre aux joueurs de savoir si ils finissent rapidement les niveaux ou non. Grace à un système de temps, les joueurs auront 2 minutes 30 pour chaque niveau afin d'obtenir des points sinon ils n'en gagnent pas. Ils perdent 10 points par seconde et un niveau donne maximum 1500 points. Ils pourront voir le score dans le menu « score ». Même si ils mettent pause cela n'arrête pas le chrono pour éviter la triche. Et si ils quittent une partie en cours alors ils ne pourront pas reprendre leurs scores.

Save

Cette classe va créer, charger, des sauvegardes (en plus d'écrire dans le registre des sauvegardes).

Le système de sauvegarde se présente comme ceci : la sauvegarde s'effectue si et seulement si le joueur a un pseudo. Chaque joueur a sa propre sauvegarde (une seule) qui lui est attitré quand il appuie la pour la première fois sur le bouton « quit and save ». Cette sauvegarde s'actualise à chaque fois qu'il réappuie sur le bouton.

Le lien entre le Model et View(Game)

La classe qui va s'occuper de ça est Game. View ne pourra communiquer avec le Model et les autres classes de Presenter que par cette classe.

2.4 Resources

La structure du dossier Resources se présente comme ceci :

|----->fxml

|----->level

|--------->default

|--------->save

|--------->test

|----->music

|----->sprite

|----->users

-Le dossier « fxml » contient tous les fichiers fxml nécessaires au jeu.

-le dossier « level » contient :

-le dossier « default » contient les niveaux fait par les développeurs.

-le dossier « save » contient toutes les sauvegardes faites par le joueur et le registre des sauvegardes.

-le dossier « test » contient tous les niveaux fait pour les tests unités

-le dossier « music » contient la musique du jeu

-le dossier « sprite » contient toutes les images du jeu

-le dossier « user » contient le registre d'inscriptions, tableaux des scores et accès aux niveaux

3 AJOUTS APPORTES

- dessin des textures

- implémentation de gif pour un jeu plus dynamique

-musique

- lave et la règle « Kill »

-la colle (glue) et la règle « sticky »

-des niveaux en plus (5 dont un labyrinthe)

-score

-sentiment de progression dans le jeu

-un système de navigation dans le jeu (menus, sous-menus, ect)

- menu pause

-choix des niveaux en corrélation avec la progression du joueur

4 DIFFICULTES RENCONTREES

En commun

Notre plus grosse difficulté a été évidemment de perdre un membre du groupe en début de projet. Nous avons eu du mal à l'accepter au début car nous n'étions pas sûr de cet abandon vu que cette personne ne donnait plus de nouvelle et ne se présentait plus à l'université.

Julien

Mes difficultés ont été le temps, les autres cours et les idées. Concernant le temps, j'avais un gros stress à l'idée de finir le projet à la dernière minute. Etant donné que dans la consigne, il mettait plus ou moins 60 heures par personne et que j'ai dû prendre la partie du projet de quelqu'un d'autre en plus…Je craignais de ne pas pouvoir finir. Pour les autres cours, c'était très compliqué de gérer les cours et le projet. Je consacrais quasiment tout mon temps au projet. Pour les idées, j'ai vraiment eu du mal à faire fonctionner le jeu au début. J'ai pratiquement passé deux jours entiers pendant les vacances à chercher un moyen de mettre en lien toutes les classes d'Item.

Romain

Mes difficultés ont été d'apprendre javafx, les sprites et mettre en commun l'interface graphique avec le code de Julien. J'avais déjà pratiqué du GUI en python, mais jamais en java. Donc, pour mes premières fois, j'ai du regarder beaucoup de tutoriels. Pour les sprites au début j'ai utilisé ceux du jeu Steam. Mais il fallait changer, j'ai donc moi-même dessiné les sprites et les dessins dans le menu sur krita (un logiciel gratuit) ce qui n'était pas simple. La mise en commun de la partie graphique avec la partie logique a été compliquée car au début les dessins n'étaient pas placés dans des panes dans des Hbox dans une Vbox. Ils étaient juste placés normalement dans le anchor Pane. Mais suite à une réunion, nous avons trouvé la solution.

5 POINTS FORTS ET POINTS FAIBLES

Points forts

-Tous les ajouts apportés.

-Le système de gestion du chargement de la map

-Le style graphique du jeu

-Le système de gestion des niveaux

-Dans les niveaux en plus : des niveaux originaux

-Personnalisation de l'expérience de jeu (avec User et Score et Level)

-Nouveau comportement de quelque chose qui est « WIN » et « YOU »

Points faibles

-La complexité du Model

-Les soucis avec la carte temporaire

-Taille de la carte non modulable

6 ERREURS CONNUES

Malheureusement, notre jeu n'est pas exempt de défaut. Les bugs répertoriés dans le Model sont :

Bugs liés à la carte :

Impossibilité de superposer plus de deux choses à un emplacement sur la carte. La Piste serait de contenir plusieurs cartes dans une liste.

Un objet stocké dans la carte temporaire et qui n'a plus besoin d'être stocké est bloqué (quand un objet est au-dessus de lui).

NOTE : À voir pour le reste…

7 MINI GUIDE

Quand on lance l'application, le jeu nous demande un nom pour se connecter avec un pseudo. Après avoir cliqué sur le bouton connexion le menu s'ouvre. On y retrouve un bouton « play », « settings », « exit » et « score ». Le bouton « play » amène vers le menu let's play qui permet de choisir une nouvelle partie ou de reprendre une partie ou de choisir un niveau directement, si il est débloqué. Pour le menu « settings » il y a un bouton « Music » et « No music » pour mettre de la musique ou non. Et il y a des explications pour les touches du jeu. Ensuite le bouton « Exit » permet de fermer l'application et le bouton « score » ouvre le menu score avec un bouton « load score » pour afficher les scores des joueurs. Quand une partie est lancée, il y a 9 niveaux à réussir pour finir le jeu. Il y a aussi un menu pause pour quitter, sauvegarder ou reprendre la partie en cours.

8 CONCLUSION

Ce PREMIER projet de notre bachelier nous a permis de faire connaissance avec l'amour de la programmation et le gout de l'autodidactisme. Le souci de perfection s'est également manifesté par l'efficacité de nos algorithmes. Ce fut une expérience incroyable et très enrichissante où la cohésion d'équipe était au premier plan.

En conclusion, c'est le premier d'une longue série de software à apparaître au sein de l'équipe Ladeuze-Eloy.

9 BIBLIOGRAPHIE

Partie introduction :

-https://fr.wikipedia.org/wiki/Baba_Is_You

Version jam :

-https://hempuli.itch.io/baba-is-you

Documentation javafx :

-https://openjfx.io/javadoc/11/

-https://www.youtube.com/watch?v=9XJicRt_FaI

Musique :

https://www.youtube.com/watch?v=mRN_T6JkHc&list=PLwJjxqYuirCLkq42mGw4XKGQlpZSfxsYd

10 REMERCIEMENTS

Nous remercions particulièrement notre beta tester Alexandre qui nous a permis de déceler quelque bug.