Rapport général du projet Java : Mario 2020

Groupe E13 :
*OUHOU HOUDA (animatrice)
*FAICAL TOUBALI HADAOUI
*YOUNESS EL BOUZEKRAOUI
* ALEXIS CARN
*YASSIR BOUISSE

Département Sciences de numérique - 1 ère année

2019-2020

TABLE DES MATIÈRES

I. Introduction :

Super Mario est un jeu classique qui a connu une grande popularité pendant les années 90 et c'était une véritable mascotte pour Nintendo. Il fera son apparition dans plus de deux cents jeux vidéo. D'où l'intérêt de faire ce choix par les membres de groupe.

Le but principal du système est de fournir un jeu similaire, programmé en Java, avec des petits changements et améliorations apportées par notre groupe. Le jeu peut être classé comme un jeu d'aventure et basé sur le niveau.

Avons-nous donc de manipuler et programmer en langage Java ? , Est-ce qu'on va se limiter à la programmation de jeux en 2D ? Est-ce qu'on va utiliser Swing ? A-t-on besoin de détecter des collisions ?

Durant ces 3 itérations, nous avons essayé de comprendre le fonctionnement du jeu, améliorer nos compétences en programmation Java, et essayer de simuler l'objectif et les fonctionnalités recherchées à travers ce jeu.

Le résultat, donc, est un jeu simple, qui offrira au joueur un niveau de difficulté intentionnellement maintenu pour que le jeu puisse devenir une plateforme sur laquelle les joueurs peuvent rivaliser.

Essayons-donc de vous laisser s'aventurer dans ce monde avec un nouveau jeu Mario programmé par notre groupe E13.

II. Présentation du jeu :

L'objectif du jeu est d'effacer autant de niveaux que possible avec un nombre de vies limité. Chaque niveau sera composé d'obstacles, de monstres et de pièges. Il y aura également des bonus comme des pièces et des power-ups. Le jeu aura également un score net du joueur. Le score dépendra du nombre de pièces collectées, du nombre de monstres tués et du nombre de vies restantes.

Le jeu sera contrôlé uniquement par le clavier. De plus, jouer au jeu est très simple car l'utilisateur n'a qu'à utiliser les touches fléchées et la touche de saut (barre d'espace).

Le choix de ce type de contrôle vise à permettre aux utilisateurs quelques soit leur âge pour prendre plaisir à jouer.

III. Les exigences :

*L'utilisateur pourra contrôler Mario.

*L'utilisateur pourra casser des briques sur les cartes en faisant sauter Mario vers elles.

*L'utilisateur pourra aller à droite sur la carte en allant à droite.

*lorsque Mario est au tiers de l'écran de jeu.

*L'utilisateur pourra tuer des monstres.

*L'utilisateur pourra collecter des pièces.

*L'utilisateur aura trois vies au début.

*Le jeu sera terminé lorsque l'utilisateur passera toutes ses vies

*L'utilisateur commencera un nouveau niveau lorsqu'il atteindra la fin d'un niveau.

*L'utilisateur pourra suspendre / reprendre le jeu.

*L'icône Mario aura des simulations de marche, de saut et de défense.

*Les briques auront des simulations de rupture.

*Le mouvement de Mario sera soumis à certaines règles de physique qui augmenteront la difficulté du jeu.

IV. **Fonctionnalités**:

En revenant sur les fonctionnalités définies précédemment dans le fichier fonctionnalites_1.pdf, on trouve que la majorité de ces attendues ont été réalisés, et ajoutés au jeu durant ces 3 itérations, à savoir :

1.Définir différentes apparences du joueur Mario : SUPER, SMALL, FIRE.

2.Vu la difficulté rencontrée lors de la définition d'un niveau sans fin, on s'est limité dans un type simple : un niveau qui contient à la fin un drapeau, cependant, ce niveau sera limité par une contrainte du temps, c'est-à-dire, si le joueur dépasse le temps défini, il perdra la partie.

3.Définir 3 hitbox des ennemis : Goomba, Bowser, et KoopaTroopa.

4.Pour la partie : associer à l'ennemi un système d'attaque. On a essayé d'associer à Bowser un système de défense : FIRE, cependant, vu la contrainte du temps, c'était difficile.

5.Définir des objets collectés de résultat différents. Une fois ramassés, les bonus doivent modifier l'apparence et les caractéristiques du personnage jouable :

Flower -> transforme mario en MarioVSEnnemi, qui envoie de FIRE face à un ennemi.

Mushroom -> ajoute une vie au joueur Mario.

SuperMushroom -> agrandir mario.

6.Il manque donc la partie, qui permet à l'utilisateur de choisir Un sprite personnel, vu qu'on n'a pas défini assez de données faciles à manipuler.

V. Modèles de système :

Le joueur veut jouer le jeu alors il commence le jeu. Ce processus initialisé par la méthode

demarrer () de MoteurJeu. MoteurJeu dessine ensuite la carte, non visuellement, et appelle la

méthode creerMap de GestionMap pour créer les classes requises. CreerMap est l'information sur

les briques, les Players, les ennemis, etc. qui est collectée à partir de la méthode dessinerMap ().

Après cela, MoteurJeu crée les composants demandés, puis renvoie à nouveau creerMap à

MoteurJeu. A partir de maintenant, le joueur fournit une entrée à MoteurJeu, MoteurJeu met à jour

l'état avec la méthode updateState () puis appelle la méthode updateMap qui change de vue.

Ensuite, il choisit une carte parmi les options possibles (il ne peut pas choisir une carte sans passer

chaque carte avant), enfin la carte est chargée et le joueur joue le jeu.

Le joueur saute sous un BriqueSurprise qui contient un Mushroom, flower, ou SuperMushroom. MoteurJeu met à jour l'état après avoir reçu l'entrée, puis appelle la méthode jump () de Mario. Mario appelle la fonction reveal () de Brique, ce qui conduit à révéler un champignon. Ensuite, Mario appelle la fonction makeSuper () du champignon en le mangeant. Après tout, finalement, GestionMap est mis à jour et ainsi voir.

VI. Découpage en paquetages :

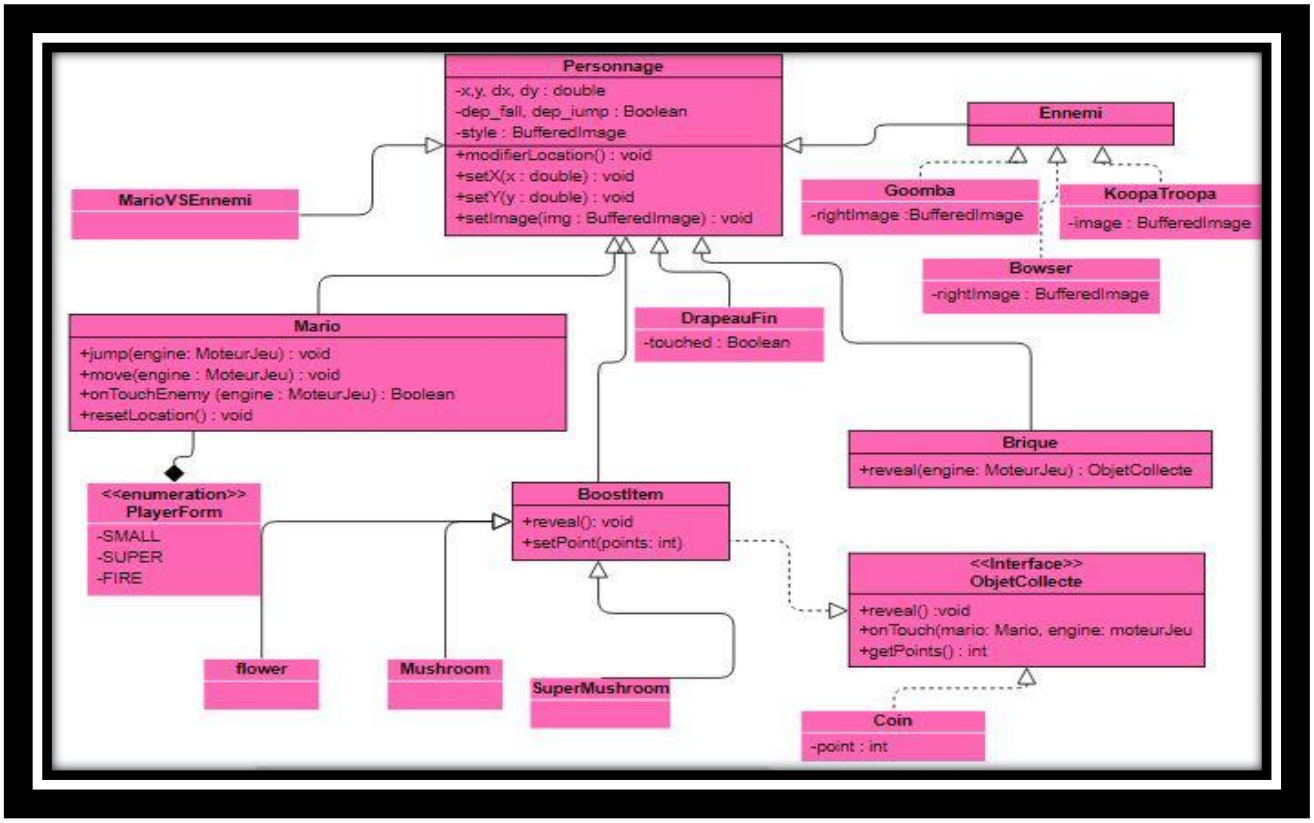
Le jeu est découpé en 4 paquetages : affichage, gestionJeu, media, et model, qui regroupent différentes fonctionnalités :

*Model :

-Ce paquetage gère deux principaux objets de jeu : Personnage, et MAP.

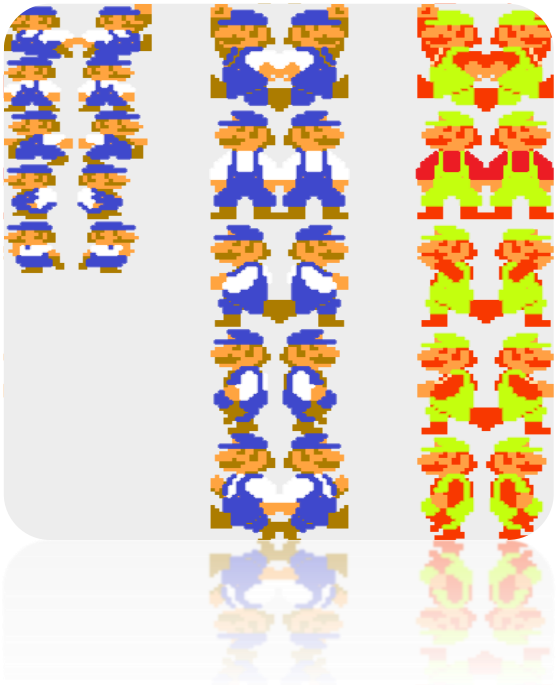
1. La classe abstraite Personnage est une classe qui factorise le code commun entre tous les classes des objets définissant le jeu.

Dans cette classe, on définit les objets fixes et dynamiques qui constitueront le Map du jeu à travers : leur position x, y, sprite, déplacement, etc., comme le montre la figure suivante:



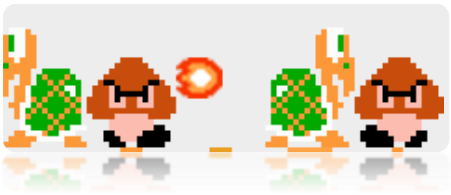
-Comme vous allez remarquer, dans cette conception, Mario, MarioVSEnnemi, Ennemi, Brique, et BoostItem héritent de cette classe abstraite qui offrent différentes méthodes et fonctionnalités.

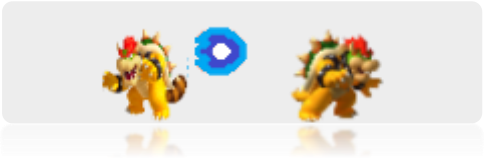
-De plus, on ajoute une énumération « Player Form » à la classe Mario, afin de lister tous les types de sprites possibles de ce joueur, et c'est ce qu'on trouve dans la figure mario-forms (SMALL, SUPER, et FIRE) :



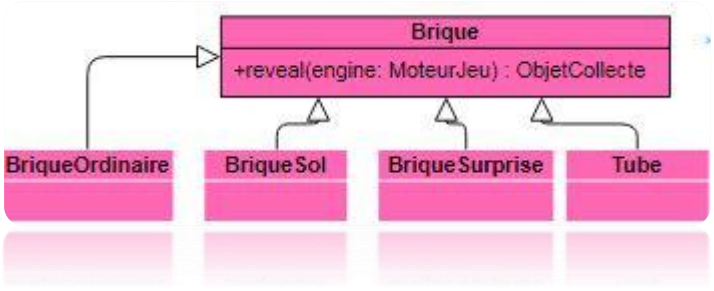
-MarioVSEnnemi : est une classe qui hérite de Personnage, et qui gère le cas du jeu , où mario se trouve face à un ennemi. Dans ce cas, il a un mouvement particulier : un mouvement de défense.

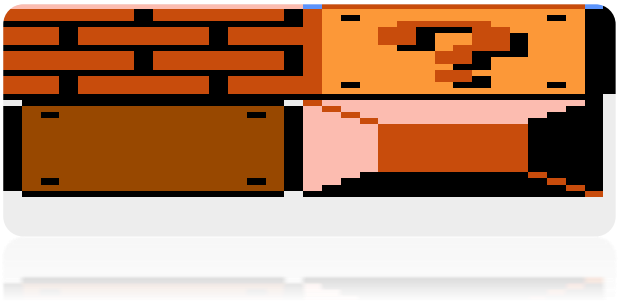
-Ennemi : est un personnage du jeu, dont les sprites sont différents. Dans ce cas, on définit trois classes Goomba, KoopaTroopa, Bowser.



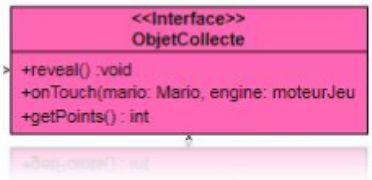


-Classe brique : hérite aussi de Personnage, mais avec différents sprites qui illustrent les objets qu'on ajoute au jeu, autre que les joueurs. De plus, on a différents types (classes) héritant de Brique :





- De plus de l'héritage de Personnage, la classe BoostItem réalise aussi l'interface : Objet Collecte où on regroupe les opérations manipulant la relation entre le joueur et l'objet collecté (Mushroom, SuperMushroom, flower) :

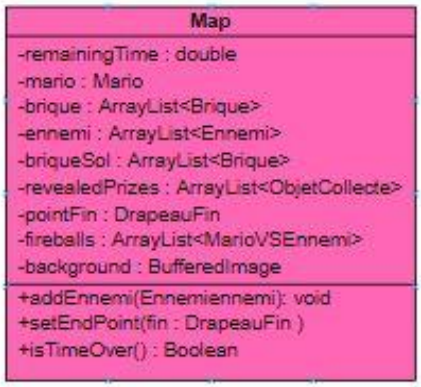


-Classe coin, réalise aussi cette interface ObjetCollecte, utilise un sprite particulier, et utilise des méthodes qui manipulent le gain du joueur au cours du jeu.

-La classe DrapeauFin : qui définit le point final du niveau de jeu. Dans notre jeu, on la définit souvent auprès d'une image Palais figurant dans le Background du Jeu.

De plus, le drapeau est toujours placé à la fin avec l'icône suivant :

2.Le deuxième objet principal est MAP :



*Où on regroupe ces différents objets héritant de la classe Personnage, et on les attribue des nouvelles méthodes et fonctions : comme ajouter des briques, ajouter des ennemies, …..

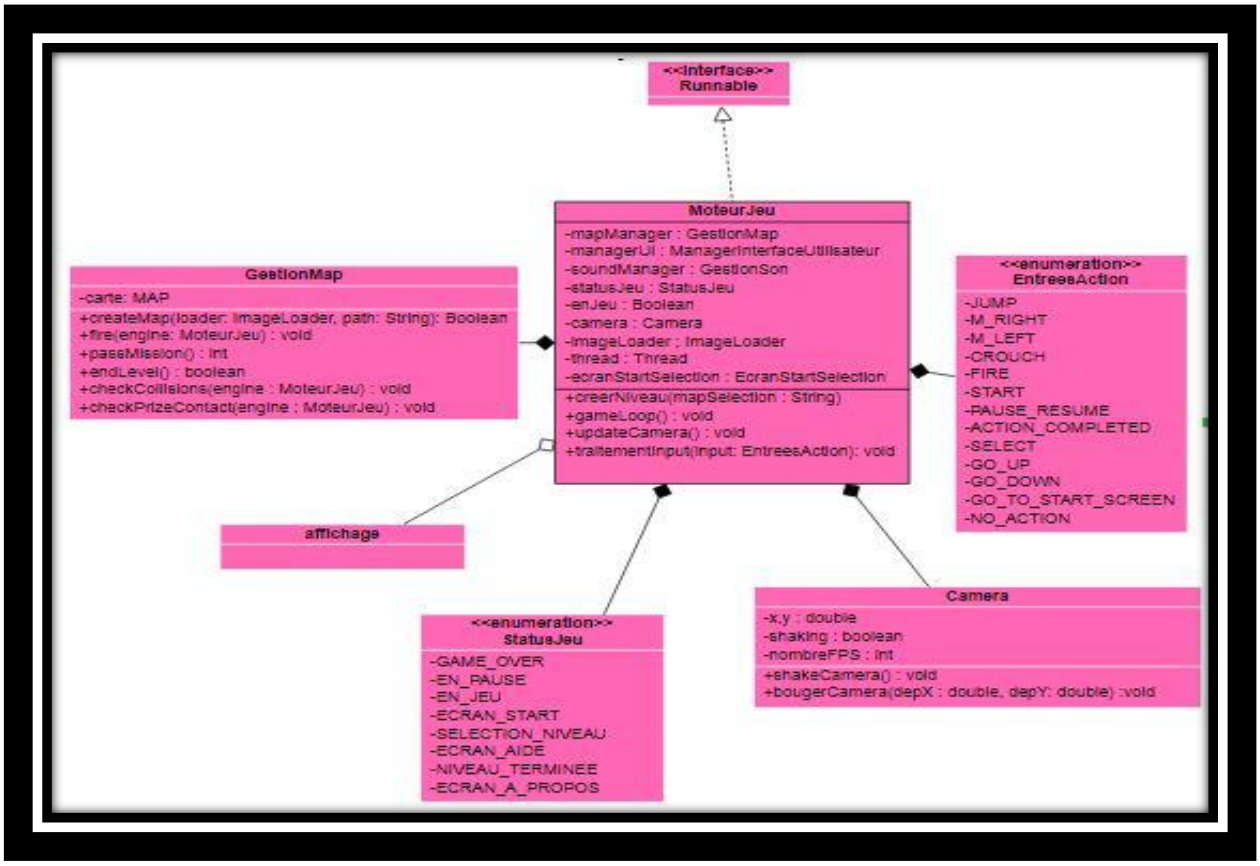
-Dans notre jeu, on va se baser sur deux types de map, qui offrent les mêmes fonctionnalités, mais avec une distribution des objets différents.

Map 1 :

Map 2 :

*gestionJeu :

-Dans ce paquetage, on regroupe les classes suivantes :

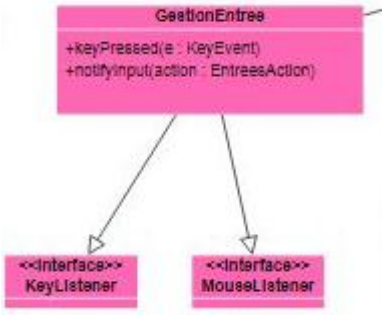


-Classe GestionMap : On manipule une variable de type MAP défini dans le paquetage model, grâce aux différentes méthodes illustrées ci-dessus qui permettent par exemple : la mise à jour du positionnement du map, ajouter des attributs (objets) : mario, les coins, renvoyer le score du jeu. Ceci montre donc, l'importance de cette classe dans la gestion du jeu et l'organisation du MoteurJeu.

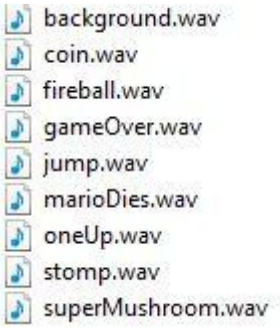
-StatusJeu, et EntreesAction sont des énumérations qui regroupent respectivement, les cas possibles du jeu (en pause, en jeu, …) et des entrées (sauter, marcher, …).

-Caméra : est une classe qui gère l'aspect affichage du jeu en tant que scénario continu de séquences préprogrammées du jeu, grâce à un attribut booléen (shaking) qui s'occupe du déplacement.

-On utilise aussi une classe GestionEntree qui récupère, et traite les entrées de l'utilisateur en héritant des deux interfaces KeyListener, et MouseListener.



-On ajoute aussi une classe GestionSon, qui sert à manipuler des clips mario, enregistrés dans le paquetage media. Et pour avoir une cohérence entre le statut du jeu (mario gagnant, mario perdant,..) et le clip audio, on ajoute des méthodes qui manipulent ce choix dans cette classe, par exemple : jouerJump (), jouerCoin (), jouerGameOver (), …



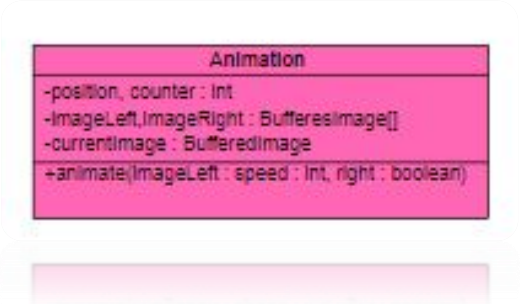
-Toutes ces fonctionnalités, plus d'autres regroupées dans le paquetage affichage, servent pour gérer la partie MoteurJeu, à partir de laquelle on va lancer le jeu. Cette classe fait appel à une méthode privée init () qui permet l'initialisation du jeu avec les données que nous avons préprogrammés, comme les backgrounds, les sprites des joueurs ainsi que les ennemies, et génère une fenêtre du jeu dont le nom choisi est « Super Mario Bros ». Aussi resetCamera (), demarrer () qui manipulent l'affichage du jeu.

-MoteurJeu hérite aussi de Runnable, car on utilise dans cette classe comme attribut : thread de type Thread afin de pouvoir répartir différents traitements d'un même programme en plusieurs unités distinctes pour permettre leurs exécutions « simultanées », et on trouve donc le résultat souhaité.

*affichage :

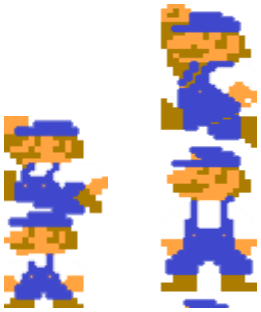
Qui regroupe des classes gérant les l'affichage des sprites, leurs animations avec d'autres qui seront listés dans la suite :

-Classe Animation :

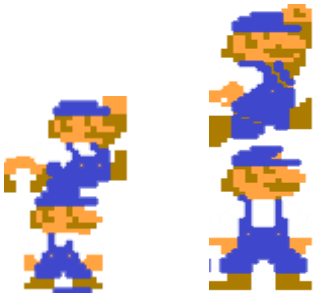


Qui sert pour animer le jeu, en agissant sur la liste des sprites manifestant les mouvements possibles de mario :

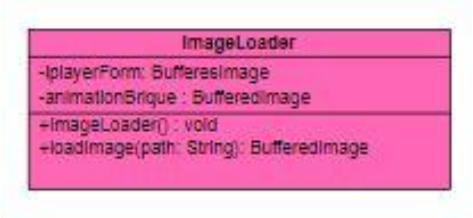
*à gauche :



*à droite :



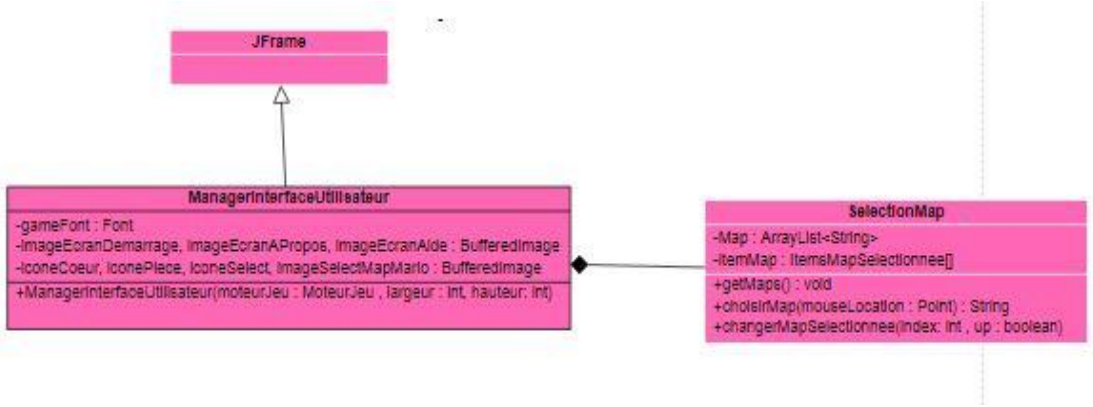
- Classe ImageLoader :



Qui gère les images de sprite (mario-forms) et les briques (brick-animation) du jeu trouvées dans le paquetage media, et les récupère à l'aide de la fonction ImageIO.read (getClass ().getResource ("/media" + path));

- Pour les sprites de l'ennemi, on définit une fonction : getSubImage (BufferedImage image, int col, int row, int w, int h) qui à partir de l'image regroupant des blocs d'images différents, on récupère le sprite souhaité, en appliquant sur l'image cible la fonction prédéfinie en Java.

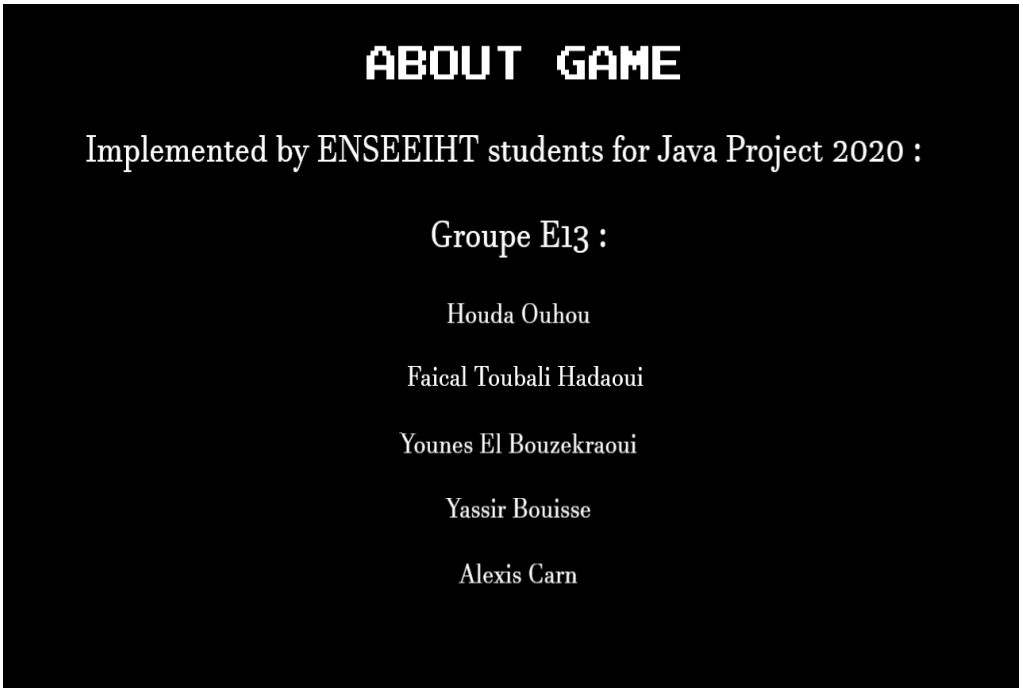
-La classe ManagerInterfaceUtilisateur :



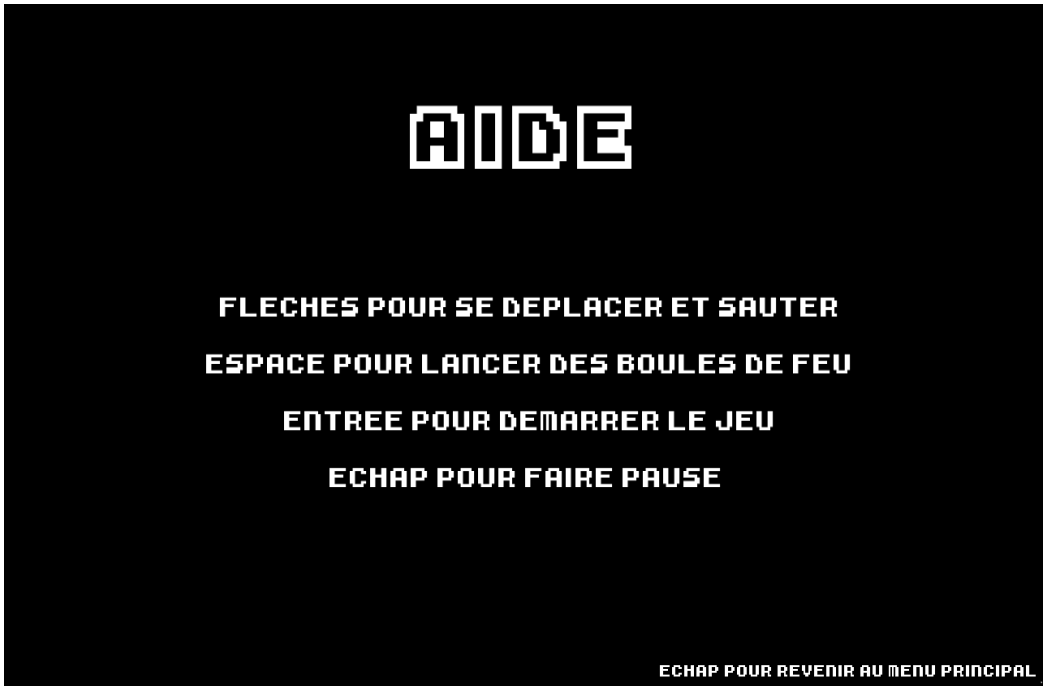
Cette classe hérite de JFrame, gère les dimensions, charge les images nécessaires pour l'affichage des éléments du MoteurJeu afin de les afficher (avec la méthode g2.drawImage()), par exemple :

1. loadImage ("/start-screen.png") :

2. loadImage("/about-screen.png") :



3. `loadImage("/help-screen.png")` :



4. `loadImage("/game-over.png")` :

-Classe ItemsMapSelectionnee : qui pour un path de type string, une position, et des dimensions choisies, renvoie une image contenue dans le dossier du paquetage media.

-La classe SelectionMap : qui gère une liste contenants les deux maps définies, afin d'offrir à l'utilisateur deux choix possibles de jeu, au début, on initialise la liste ArrayList<String> nosMaps, avec les path des 2 maps jouables définies, on trouve donc, le résultat suivant :

-La classe EcranStartSelection de type énumération: qui gère les 3 options proposées à l'utilisateur au début du jeu, comme le montre la figure :

-Dans cette partie, selon le choix de l'utilisateur, c'est-à-dire le numéro de ligne cliqué, la fonction getSelection (int choix) défini un type : ***DEMARRER_JEU*** dans le cas de la ligne 0, ***VOIR_AIDE*** dans le cas de 1, et ***A_PROPOS*** dans le cas de 2.

-Le résultat de cette manipulation de cette classe sera donc, utiliser pour initialiser l'attribut de classe MoteurJeu.

-Concernant la partie programmation et implantation, on a essayé de définir plus de niveaux mario,

avec différents sprites et différents temps, cependant vu la situation actuelle, ainsi que la difficulté

de s'organiser à distance, nous avons juste réussi à définir un niveau, avec les fonctionnalités

promises dans la première itération du projet.

VII. Principales conceptions/réalisations :

-Le choix de découpage du jeu était la partie la plus pertinente de notre conception, car, d'abord, il nous a facilité la tâche de l'organisation du travail, en effet, chaque membre du groupe a défini un paquetage, et après la vérification de la compilation des classes, on a essayé de regrouper tous les
modalités et ajouter l'appel des paquetages au début de chaque script afin de coordonner et vérifier
la cohérence du jeu.
-Définir ObjetCollecte de type Interface afin de faciliter l'ajout d'autres types d'objets collectés dans le jeu.

-Définir aussi la classe Personnage de type abstrait nous permet, de factoriser le code commun entre l'ensemble d'objets du jeu.

-De plus, on définit les Players, ennemis, briques de type Personnage, avec des sprites, et des méthodes personnalisées, ce qui facilite leur implémentation.

-La classe MoteurJeu hérite de Runnable, car on veut manipuler un attribut thread de type Thread.

VIII. Problèmes rencontrés :

-Vu la situation actuelle, le confinement, et les problèmes de Connexion, le contact entre les membres de notre groupe était un peu difficile, surtout pour discuter plusieurs fonctionnalités. Aussi, les contraintes du temps, n'ont pas permis de régler le jeu, Et le développer.

IX. Organisation de groupe

-Notre aventure dans la programmation de ce jeu ne date pas d'hier, au cours de ces 3 itérations, on a défini plusieurs types, plusieurs représentations, ainsi que plusieurs fonctionnalités. Cette diversité n'était pas un blocage devant l'avancement du groupe, mais plutôt un point positif qui nous a aidés à développer un sens de coopération, de cohérence entre les membres de groupe. De plus, on a utilisé un ensemble d'applications à savoir, discord, Messenger, ainsi que Trello pour s'organiser, communiquer et échanger des idées à propos du projet.