Besson Pierre-Victor
Desmazières Julien
Surget Kevin
Thaveau Joris

RAPPORT TECHNIQUE DU PROJET DÉVELOPPEMENT V1

10 mai 2017

Projet 70 : Développement d'un prototype de jeu vidéo en Java

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Résumé du rapport

La programmation à travers l'informatique est une science importante de l'ére du numérique. Cependant avant d'obtenir une version finale d'un logiciel, il est nécessaire de passer par des phases où l'on crée des prototypes.

Ce rapport présente la programmation d'un prototype de jeu vidéo en Java*. Il s'agit d'un jeu de stratégie au tour par tour, dans lequel on offre au joueur la possibilité d'enrichir le jeu en ajoutant de nouveaux personnages jouables ou de nouvelles cartes de jeu.

Dans un premier temps nous nous sommes mis d'accord sur toutes les spécifications du jeu. Puis, en plus d'une interface graphique programmée avec Swing*, des outils ont été créés afin de faciliter la tâche du développeur lors de la création d'instances. Enfin un menu a été mis en place pour pouvoir sélectionner une partie, ou tout simplement quitter le logiciel. Tous ces éléments permettent ainsi d'avoir un prototype fonctionnel qui peut par la suite être enrichi afin d'assurer la longévité du jeu.

SOMMAIRE

Résumé du rapport....2
I. Introduction....4
II. Cahier des charges....4
III. Fonctionnement du jeu....6
III.1. Spécification de la carte....6
III.2. Spécification d'un personnage....7
III.3 Spécification d'une partie....8
IV. Système de jeu....8
IV.1 Gestion des Sauvegardes....8
IV.2. Gestion des déplacements....9
IV.3. Gestion de la ligne de vue....10
V. Développement du GUI....11
V.1. Fonctionnalités d'un GUI....11
V.2 Choix de la bibliothèque pour l'interface graphique....11
V.3. Utilisation de Swing....11
V.3.1. JFrame....11
V.3.2 JPanel....12
V.3.3 JButton....12
V.4. Structure du GUI....12
V.4.1 Case....13
V.4.2. MatriceBoard....13
V.4.3. Board....14
VI. Menu....15
VI.1. Principe du menu....15
VI.2 Les différentes classes du menu....15
VI.2.1. Background....15
VI.2.2. ComingSoon....16
VI.2.3. LevelButton....16
VI.2.4. MenuButton....16
VI.2.5 ScreenLevels....16
VI.2.6. ScreenMenu....17
VI.2.7. Window....18
VI.2.8. Menu....18
VII. Logger....18
VIII. Conclusion....18
IX. Bibliographie....19
X. Glossaire....19

Besson Pierre-Victor : Rédacteur partie 5, relecteur parties 6 et 7
Desmazières Julien : Rédacteur parties 1, 3, 4, 8 et résumé, relecteur partie 2
Surget Kevin : Relecteur parties 6 et 7
Thaveau Joris : Rédacteur parties 6 et 7, relecteur de toutes les parties.

I. Introduction

Le marché du jeu vidéo est en pleine expansion depuis quelques décennies. Cependant avant de mettre un jeu sur le marché, il doit d'abord entrer dans plusieurs phases de conception. En particulier, il est nécessaire de concevoir des prototypes.

C'est dans ce cadre que nous avons décidé de développer un prototype de jeu vidéo, afin de rendre compte des différentes étapes à réaliser pour parvenir à un prototype fonctionnel. Il s'agit d'un jeu de stratégie au tour par tour fonctionnant sur ordinateur sous les systèmes d'exploitation Windows ou Linux, avec la particularité de permettre à un joueur de créer ses propres personnages ainsi que ses propres carte de jeu.

C'est pourquoi ce travail a été effectué en Java*, car il s'agit d'un des langages de programmation les plus utilisés et aussi des plus adaptés à notre logiciel.

Dans un premier temps, les besoins de l'utilisateur ont été spécifiés. Puis, toutes les modalités et contraintes du jeu ont dû être décrites de manière exhaustive. Une fois ces étapes accomplies, la programmation s'est faite selon deux axes principaux. D'une part toutes les classes et leurs interactions décrites précédemment ont été réalisées. D'autre part une interface Homme-Machine a été développée afin de détecter les actions du joueur. Enfin la liaison de ces deux parties a été effectuée, donnant suite à la réalisation de différents tests vérifiant l'intégrité du code. Ces tests nous ont permis d'améliorer et de corriger certaines parties du code que nous n'avions pas remarqué en amont. Mais ils nous permettent aussi d'assurer la durabilité du logiciel.

Tout d'abord, nous décriront toutes les contraintes liées au cahier des charges de ce jeu. Puis nous détaillerons le fonctionnement du jeu, la manière dont les classes interagissent entre-elles, ainsi que toutes les modalités et spécifications que nous avons choisies d'implémenter. Ensuite nous expliquerons le fonctionnement de certains principes spécifiques à notre jeu, donnant alors une description exhaustive du logiciel. Dans un second temps, nous entrerons dans les choix de programmation de l'interface graphique. Pour cela, nous allons tout d'abord expliciter les fonctionnalités de cette interface. Puis nous expliquerons le choix de la bibliothèque Swing* pour son développement. Nous détaillerons ensuite la structure de cette interface. Nous finirons alors par la description du fonctionnement du Menu, ainsi que du logger utilisé. Ainsi, l'ensemble des éléments constituant notre prototype sera abordé.

II. Cahier des charges

Menu principal

Le logiciel devra comporter un menu principal, affiché à son lancement. Ce menu devra permettre de naviguer entre :
- Une sélection de niveaux parmi ceux disponibles.
- Un accès à une sauvegarde.
- Un menu d'options. (optionnel)

La navigation s'effectuera à l'aide de commandes à la souris (clavier optionnel).

Sauvegarde

Le logiciel devra comporter une fonction de sauvegarde, accessible à chaque tour du joueur. Elle permettra d'enregistrer l'état d'une partie en cours sous la forme de fichiers texte et sera conservée même après fermeture du logiciel.
Ces sauvegardes pourront également être chargées afin de reprendre la partie en cours.

Chargement

Le logiciel devra comporter différents niveaux de jeu, qui différeront de par les personnages et ennemis présents, ainsi que par la carte. L'utilisateur pourra choisir de jouer à l'un ou l'autre de ces niveaux via le menu. Ces cartes seront composées d'un quadrillage de cases carrées (cf Cases).
Le logiciel pourra aussi charger les précédentes sauvegardes.

PvP* et IA*

Les personnages non joueurs se devront d'être contrôlés par une intelligence artificielle basique, capable de choisir aléatoirement une action parmi une liste d'actions prédéfinies.

Il y a aussi possibilité de contrôler les autres personnages par un autre joueur.

Système de combat

Cette catégorie regroupe toutes les fonctions nécessaires au bon déroulement d'une partie du jeu. Elle comporte :

- Personnages.

Les personnages se diviseront en 2 catégories : des personnages contrôlés par le joueur, et des personnages ennemis. Les 2 catégories seront, en pratique, codées par la même classe. Les personnages disposent de statistiques et compétences propres, ainsi que d'une apparence unique. Ils seront définis au moyen d'un fichier texte.

- Compétences.

Elles seront appelées par les personnages lors de leurs tours de jeu. Chaque compétence sera définie par un fichier texte et disposera de ses caractéristiques propres, comme sa portée ou ses dégâts.

- Tours de jeu.

Les combats s'effectueront au tour par tour, chaque personnage pouvant effectuer des actions selon un ordre prédéfini. Durant un tour de jeu, un personnage pourra effectuer des compétences et/ou se déplacer.

Ces tours seront interrompus à la demande du joueur dans le cas de ses personnages sauf exception, et à la fin des actions de l'IA* dans le cas des personnages non joueur.

- Cases.

Les personnages évolueront sur des cases, définies par la carte. Ces cases pourront être occupées par un personnage, un obstacle, ou des pièges, qui auront des effets sur les personnages se situant dessus.

Ces cases seront également l'unité de base déterminant la portée des attaques et des déplacements des personnages.

- Ligne de vue.

Certaines compétences nécessiteront d'avoir une ligne de vue entre le personnage souhaitant effectuer cette compétence et sa cible. Cette ligne de vue sera déterminée par la présence ou non d'obstacles entre les 2.

- Déplacement

Les personnages seront capables de se déplacer d'une case à une autre, sur demande du joueur ou choix de l'IA*. Ces déplacements seront soit effectués case par case, soit calculés par algorithmes de plus court chemin dans le cas d'un trajet plus long.

GUI*

Le logiciel disposera d'une interface utilisateur graphique, permettant l'affichage d'une fenêtre où s'afficheront des informations concernant la partie. Cette interface sera également capable de recevoir des commandes du joueur via le clavier ou la souris et de les transmettre au logiciel.

III. Fonctionnement du jeu

La création d'une partie se fait à travers trois outils dédiés à la création d'une carte, la création d'un personnage et enfin la création d'une partie. Ces outils permettent de faciliter ces créations tout en respectant leurs spécifications.

III.1. Spécification de la carte

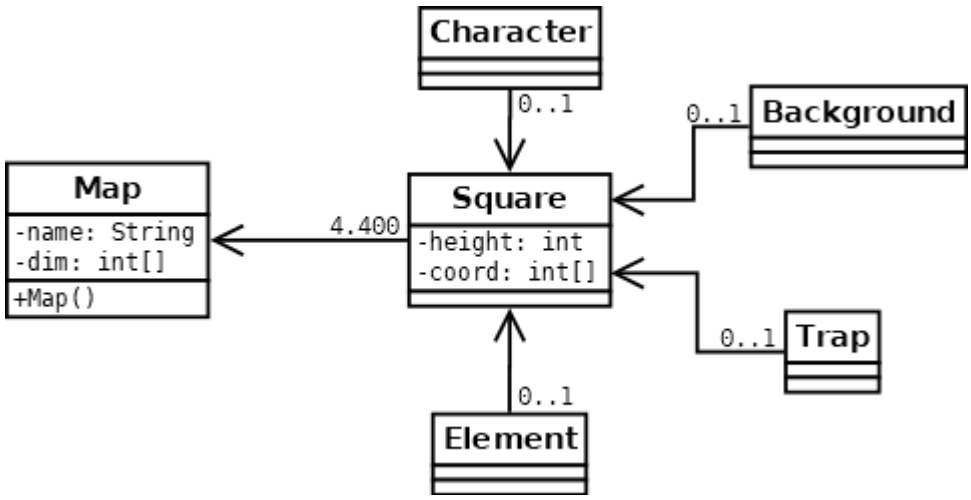

Illustration 1: Diagramme de classe simplifié des composants d'une carte (Map)

La carte est un quadrillage rectangulaire de largeur et de hauteur compris entre 2 et 20 cases. Chaque case peut être vide ou composée d'un arrière plan (Background), d'un élément de décor (Element), d'un piège (Trap), d'une hauteur (Height), et d'un personnage (Character).

L'arrière plan correspond au terrain de base, les différentes valeurs qu'il peut prendre sont "eau", "terre" ou "sable"'. Chacune de ces valeurs va affecter les déplacements des personnages, ainsi que leurs attaques.

Une case peut aussi avoir un (et un seul, dans ce cas) élément de décor comme un mur ou un arbre. Ces éléments vont aussi affecter les déplacements des personnages, ainsi que la ligne de vue.

Un piège est une attaque spéciale d'un personnage qui permet de poser sur une case un élément (un piège) qui sera déclenché au passage d'un autre personnage.

Il est possible d'attribuer une hauteur à une case, ce qui peut modifier les déplacements et la ligne de vue avec les autres cases.

Enfin sur chaque case, il peut y avoir un personnage. Ces personnages peuvent interagir avec la carte en se déplaçant dessus, en interagissant avec les éléments du décor, ou en posant ou déclenchant des pièges.

III.2. Spécification d'un personnage

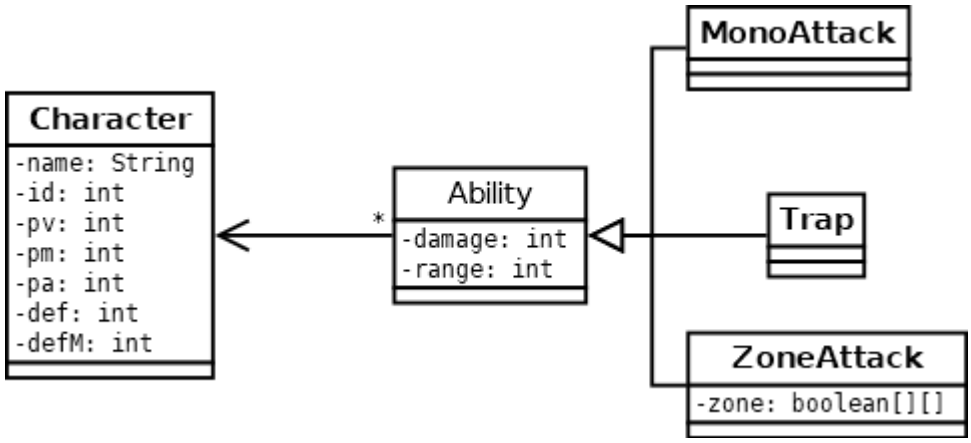

Illustration 2: Diagramme de classe simplifié des composants d'un Personnage (Character)

Un personnage est une unité contrôlable du jeu. Chaque personnage possède un nom, un identifiant utile lors de la sauvegarde ou du chargement d'une partie (cf 2.1), un état, des statistiques qui lui sont propres, ainsi que des attaques.

Les statistiques sont au nombre de 5, les points de vie (pv), les points de mouvement (pm) qui permettent au personnage de se déplacer, les points d'action (pa) qui permettent au personnage d'utiliser ces attaques, la défense (def) et la défense magique qui permettent de diminuer les dégâts infligés par une attaque subie.

Il existe trois types d'attaques : les attaques monocibles, les attaques de zone et les pièges. Les attaques monocibles permettent d'infliger des dégâts directement à une cible unique, alors que les attaques de zone infligent des dégâts à tous les personnages et éléments qui se situent dans le rayon de l'attaque.

Les pièges sont eux placés sur une case ne possédant pas déjà un autre piège ou un personnage. Le déclenchement du piège se fait à distance par la personne qui l'a posé ou par un personnage qui passe sur la case en question.

Enfin l'état du personnage modifie ses caractéristiques. Il peut être "mort" dans ce cas, il ne pourra effectuer aucune action, "inerte" il ne peut alors plus bouger ou "paralysé" dans ce cas il ne peut plus attaquer.

III.3 Spécification d'une partie

Une partie est composée d'une carte et d'un nombre de personnages compris entre 2 et 20 (si la taille de la carte le permet). De plus une partie possède une météo pouvant modifier la ligne de vue, les attaques ou les déplacements des personnages.

La partie se déroule au tour par tour, et consiste en la confrontation de deux équipes contrôlées par des joueurs humains ou par une Intelligence Artificielle (IA). L'IA* n'exécute que des actions aléatoires parmi toutes celles à sa disposition. Chaque joueur peut exécuter ou non toutes les actions que ses personnages ont à leur disposition avant de finir son tour.

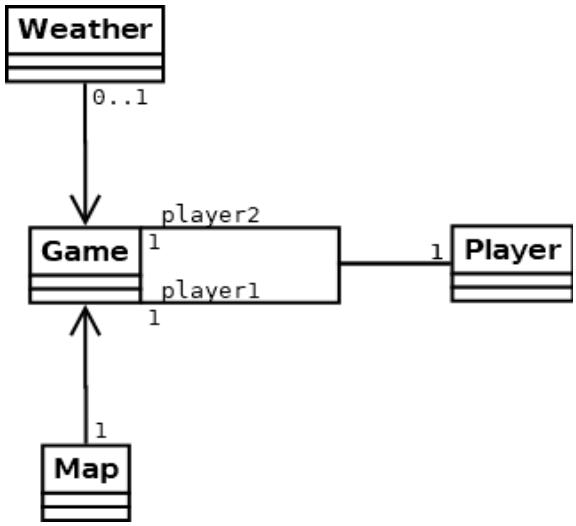

Illustration 3: Diagramme de classe simplifié des composants d'une partie (Game)

La fin d'une partie est déclarée à partir du moment où tous les personnages d'une des deux équipes sont morts.

Afin de faciliter la création de carte, de personnages et de parties, des outils de développement ont été mis en place. Un générateur de carte, un générateur de personnage et un générateur de partie permettent ainsi de créer de façon beaucoup plus rapide des instances des classes Map, Character et Game.

IV. Système de jeu

IV.1 Gestion des Sauvegardes

Lors d'une partie, un joueur peut sauvegarder de manière persistante l'état actuel de sa partie à n'importe quel moment. Pour cela, on utilise des fichiers texte dans lesquels toutes les informations nécessaires sont enregistrées sous forme de chaîne de caractères respectant des contraintes spécifiques.

En effet lorsqu'un joueur décide de sauvegarder, un dossier est créé portant le nom de la partie en cours, dans lequel il y a un fichier texte qui contient les spécifications de la carte, ainsi que des fichiers texte contenant chacun les spécifications d'un des personnages de la partie. Le fichier décrivant la carte est construit de la manière suivante :

- Nom de la carte sous la forme d'une chaîne de caractères

- Taille de la carte sous la forme largeur x longueur

La matrice représentant les cases de la carte dont chaque colonne est délimitée par le caractère tiret bas _ , et dont chaque case est définie par une chaîne de caractères qui est de la forme suivante :

- entier b entier h entier e entier p où l'entier devant b correspond à l'id* de l'objet Background associé à la case,

- l'entier devant h correspond à la hauteur de la case,

- l'entier devant e correspond à l'id* de l'objet Element associé à la case,

- l'entier devant p correspond à l'id* de l'objet Character associé à la case

Le fichier décrivant un personnage est construit de la manière suivante :

- Nom du personnage

- Identifiant

- Points de vie

- Points de Mouvement

- Points d'action

- Points de défense

- Points de défense Magique

- Liste des attaques du personnage

Le chargement d'une partie se fait grâce à la lecture de tous les fichiers du dossier correspondant. Cependant pour les attaques des personnages on utilise la réflexivité. En effet un personnage possède une liste d'objets héritant de la classe Ability. De ce fait, on utilise la reflexivité pour obtenir une instance d'une classe à partir d'une chaîne de caractères correspondant au nom de cette classe.

IV.2. Gestion des déplacements

Lorsqu'on sélectionne un personnage, celui-ci a la possibilité de se déplacer sur la carte. Cependant, ce déplacement est soumis à plusieurs contraintes. Le personnage ne se déplace qu'à travers des cases adjacentes horizontalement ou verticalement. Il ne peut se déplacer sur une case où se situe déjà un autre personnage, ou un élément de décor ne permettant pas qu'on se positionne dessus.

De plus le déplacement est limité par le nombre de points de mouvement que le joueur possède. Enfin chaque case ne peut, en fonction de l'élément de décor qu'elle possède, qu'être traversée que d'une certaine manière. Ainsi un tableau de booléens correspondant aux quatre points cardinaux permet de retranscrire cette propriété en autorisant ou non les déplacements provenant d'une direction donnée. La distance maximale de déplacement est affectée par l'état du personnage et par la case sur laquelle il se trouve.

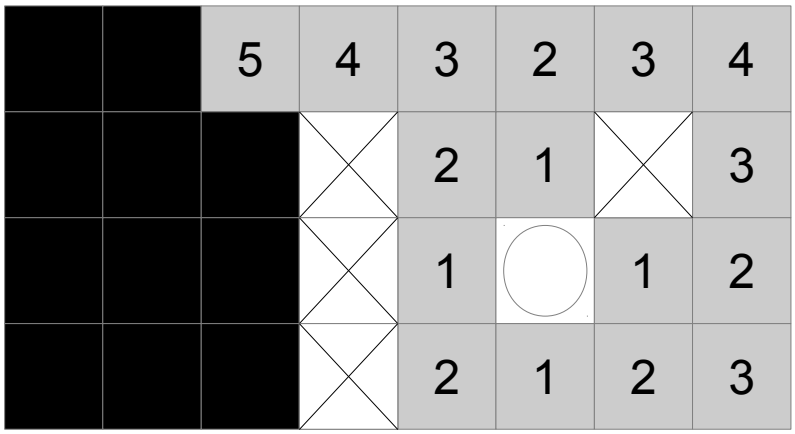

Illustration 4: Schéma illustrant les déplacements d'un personnage possédant 5 points de mouvement

IV.3. Gestion de la ligne de vue

Lorsqu'un personnage souhaite attaquer, il est contraint par la ligne de vue. Certains éléments de décor peuvent bloquer la ligne de vue du personnage, ce qui l'empêche d'atteindre une cible située derrière. La ligne de vue est construite de la manière suivante :

On trace la droite réelle passant par les coordonnées des deux cases. Si aucun carré, situé entre les deux précédentes cases, n'a plusieurs points qui vérifient l'équation de la droite, alors la ligne de vue est ouverte. Dans le cas contraire, la ligne de vue est considérée comme bloquée. Comme les déplacements, les éléments du décor peuvent gêner la ligne de vue uniquement suivant une certaine direction horizontale, verticale ou diagonale.

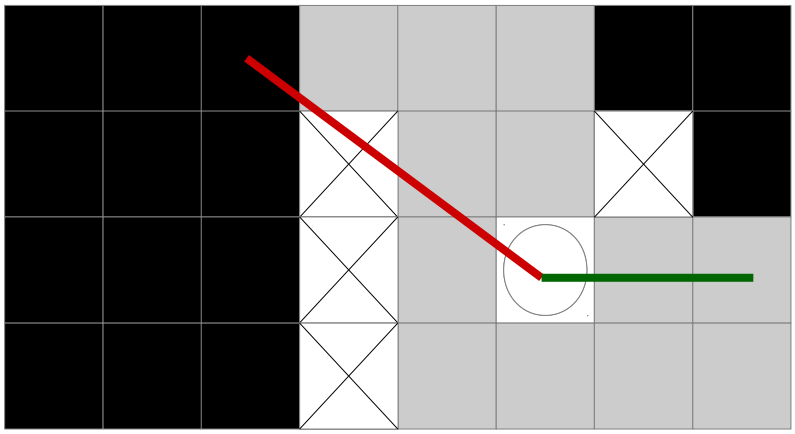

Illustration 5: Schéma illustrant les déplacements d'un personnage possédant 5 points de mouvement

V. Développement du GUI

V.1. Fonctionnalités d'un GUI

Le GUI* permet à l'utilisateur de l'application d'interagir avec le jeu via une interface graphique. Conformément au cahier des charges, le GUI* affiche un plateau de jeu composé de cases cliquables.

Ces cases ont des apparences modifiables selon les besoins du jeu.

Il assure également la cohérence entre cette interface graphique et un modèle du jeu non accessible à l'utilisateur .

V.2 Choix de la bibliothèque pour l'interface graphique

Deux bibliothèques sont proposés par Java* afin d'afficher une interface graphique :

- AWT*, la plus ancienne, mais aussi la plus lourde, celle-ci sera néanmoins utilisée pour certains éléments non présents dans l'autre bibliothèque.

- Swing*, cette bibliothèque est plus légère que son homologue, et permet d'effectuer la plupart des actions que nous souhaiterons faire.

Afin d'éviter tout conflit entre les deux bibliothèques, puisqu'elles agissent sur les mêmes composants, nous priviligérions l'utilisation de Swing pour l'interface graphique.

V.3. Utilisation de Swing

Elle est importée par la ligne de code `import javax.swing.*;`

Elle est choisie ici pour réaliser le GUI* de ce projet, et aura nécessité beaucoup de recherches et de pratique avant de la maîtriser suffisamment.

Les 3 classes de Swing* les plus utilisées dans ce projet sont JPanel*, JButton* et JFrame*. Elles sont détaillées dans leur partie correspondante. De plus, les Layout* seront aussi détaillés.

V.3.1. JFrame

JFrame* permet de créer une fenêtre graphique. Cette fenêtre pourra ensuite être remplie par d'autres composants Swing*.

Après la création d'un objet JFrame*, il est possible d'interagir avec lui de plusieurs manières. Il est par exemple possible de :

- Choisir le titre de la fenêtre.
- Choisir la taille de la fenêtre.

- Choisir l'opération à exécuter en cas de fermeture de la fenêtre.
- Afficher ou non la fenêtre.
- Interdire ou autoriser le redimensionnement de la fenêtre par l'utilisateur.
- Ajouter des éléments Swing* à la fenêtre.

V.3.2 JPanel

Un objet JPanel* est une sorte de toile de peinture qui peut être remplie puis ajoutée à une JFrame*.

Tout comme JFrame*, il est possible d'en modifier de nombreux paramètres (taille, visibilité, couleur de fond...). Il est également possible d'y ajouter d'autres éléments Swing*, et surtout, de les ajouter selon un motif via des Layout*, disponibles dans la bibliothèque java.awt.

Par exemple, la commande `setLayout(new GridLayout(length, height)) ;` permet de subdiviser le JPanel* en une grille de taille length x height.

V.3.3 JButton

Un objet JButton* est un bouton cliquable. Il est possible d'en définir la taille, l'icône, et surtout l'action à effectuer si il est cliqué.

Cependant, pour pouvoir détecter si un bouton est cliqué, ou de manière générale détecter une action de l'utilisateur sur un élément Swing*, il est nécessaire d'y ajouter un actionListener, à l'aide de l'interface ActionListener disponible dans la bibliothèque java.awt.event. Ceci permet de générer un objet ActionEvent à chaque action de l'utilisateur, qui pourra ensuite être utilisé pour réagir de manière appropriée à cette action.

Il est aussi possible d'associer directement une ActionCommand à un bouton via la commande `setActionCommand(String nomAction)`. Cela permet d'associer un nom d'action à effectuer à un bouton et de récupérer ce nom quand le bouton est appuyé pour choisir l'action à effectuer.

Ceci est utile dans le cas de nombreux boutons qui doivent effectuer des actions similaires. En effet au lieu d'écrire une action à effectuer pour chaque bouton, il suffira de définir pour chaque bouton une ActionCommand, que l'on pourra récupérer ensuite et donner en argument d'une fonction générale à appliquer lors de l'appui d'un bouton.

V.4. Structure du GUI

Le code du GUI* est décomposé en 3 classes, qui seront détaillées dans leur propre partie :

- La classe MatriceBoard, qui représente le modèle du jeu sous la forme d'une matrice modifiable via des fonctions de cette classe.
- La classe Case, qui représente une case de l'interface graphique. Elle est cliquable, et de ce fait hérite de la classe JButton de Swing.

– La classe Board, qui constitue l'interface graphique à proprement parler, et qui affiche le plateau de jeu. Elle assure aussi les interactions entre l'utilisateur et MatriceBoard .

Les spécifications de chaque classe détaillées ci-dessous sont susceptibles d'évoluer d'ici la fin du projet.

V.4.1 Case

Case est une classe héritant de JButton*. Elle dispose en attributs :

– De coordonnées (i,j), i et j étant des int.

– D'une icône ImageIcon.

– D'un Square

Son apparence sera constituée d'une couleur de fond et éventuellement d'une icône.

L'attribut Square, donné lors de la création de Case par le constructeur, permet de déterminer l'apparence de la case. En effet, la couleur de fond sera déterminée par l'attribut Background du Square, et si un personnage de la classe Character est présent dans Square, l'icône de Case sera l'icône de Character, récupérée dans les fichiers du jeu par getSpriteLoc.

À chaque Case est associée une ActionCommand, qui contient ses coordonnées.

V.4.2. MatriceBoard

MatriceBoard n'hérite d'aucune classe. Elle dispose en attributs :

– D'une hauteur height et d'une longueur length, toutes les deux des integer

– D'un Square [] [] matrice.

– D'une Map map.

– De paramètres qui aideront au déplacement de Character (int xToMove, int yToMove, boolean needToMove, Character charToMove).

MatriceBoard représente l'état du jeu à un instant donné via l'attribut matrice.

La Map map en attribut ne sert qu'à sauvegarder et charger une map, et est par exemple paramètre du constructeur MatriceBoard.

Outre les getters et les setters (de matrice, de Square, de Character...) , il est possible de modifier la matrice via la fonction moveCharacter, qui permet de déplacer l'attribut Character d'un Square à un autre Square si needToMove est vrai.

MatriceBoard n'interagit directement avec l'interface graphique, et de ce fait n'utilise pas la bibliothèque Swing*. En effet c'est plutôt l'interface graphique qui va utiliser MatriceBoard que l'inverse.

V.4.3. Board

Board représente l'interface graphique elle-même. Elle hérite de JPanel* et implémente ActionListener. Elle dispose en attributs :

- D'une hauteur height et d'une longueur length, toutes 2 int.
- D'une MatriceBoard matriceBoard.
- D'une Case[] listeCase.
- D'une instance JFrame frame.

La construction de la fenêtre se fait selon en plusieurs étapes.

L'objet Board est d'abord créé en prenant en argument un objet matriceBoard, qui lui même est chargé à partir d'un objet Map.

L'objet JFrame frame est créé, sa taille son nom et sa position sont choisis et il y est ajouté un ActionListener.

Enfin l'objet Board se voit attribuer un Layout* en forme de Grille, et est ensuite rempli d'objets Case. Les objet Case ajoutés ont été construits en donnant en argument du constructeur Case les Square de MatriceBoard.matrice, qui est une Square[][].

Les composants sont alors rendus visibles.

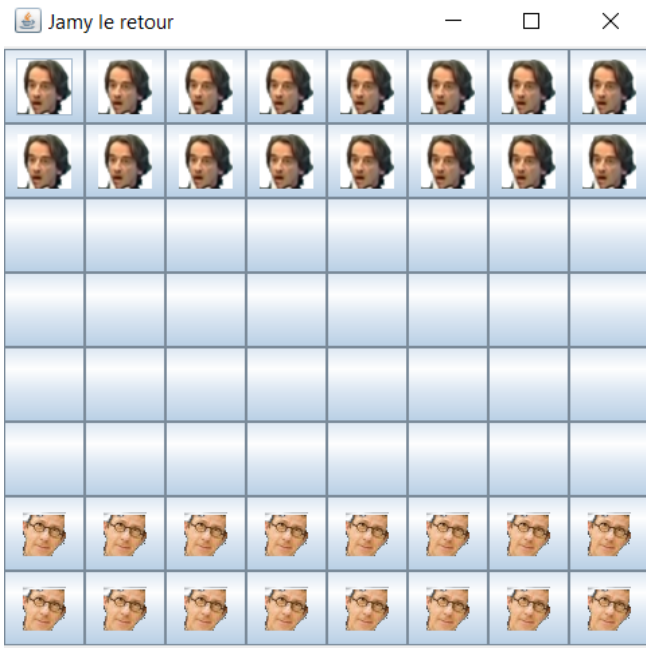

*Illustration 6: Vue d'un Board de taille 8*8 pour 2 types de personnages avec des cases sans Background*

Board gère également les actions de l'utilisateur. Au moment de l'écriture de ce rapport, il est possible de déplacer un personnage d'une case à une autre en cliquant sur le personnage puis sur une case vide.

Lors d'un clic de l'utilisateur sur une des cases, les coordonnées de la case sont récupérées grâce au ActionCommand de l'objet Case sur lequel il a cliqué.

L'attribut MatriceBoard matrice garde en mémoire si un personnage a été cliqué au clic précédent ou non. Si c'est le cas, cliquer sur une case vide bougera le personnage. Si ce n'est pas le cas, cliquer sur un personnage modifiera des attributs de MatriceBoard.

Dans la version actuelle, la fermeture de Board (via la croix) procède également à une sauvegarde de l'état du plateau de jeu

D'autres fonctionnalités seront implémentées à l'avenir en accord avec le cahier des charges (notamment les attaques de personnages et la gestion des portées de déplacement).

VI. Menu

VI.1. Principe du menu

Le menu est accessible au joueur dès le lancement du programme. En effet, celui-ci peut à partir du menu :

- Reprendre la partie en cours

- Choisir un niveau parmi les différents niveaux déjà créés

- Créer un niveau à partir de l'éditeur de niveau

- Changer les options (taille de la fenêtre, plein écran)

- Quitter

La reprise de la partie en cours charge les données liées à celle-ci, et renvoie donc à la partie sur le GUI.

Le choix du niveau affiche un autre menu, affichant les différents niveaux disponibles, dont le défilement se fait à l'aide de flèches sur le côté. La partie centrale montre un descriptif du niveau, qui est lu à partir du fichier de sauvegarde de celui-ci.

L'éditeur de niveau demande à l'utilisateur une taille de carte, puis lui affichera celle-ci, qu'il pourra modifier avec les différentes options disponibles.

Le choix des options comprend une case à cocher pour le mode plein écran et un choix de résolutions parmi différentes.

Le dernier bouton lui permet de fermer la fenêtre.

VI.2 Les différentes classes du menu

VI.2.1. Background

Cette classe hérite de JPanel, afin d'afficher le fond d'écran du menu, avec une image choisie lors de la création. Celle-ci affiche un cadre doré au centre, où s'y trouve les différents menus de sélection avec les différents boutons.

La classe possède une méthode qui permet de redimensionner l'image de fond d'écran pour qu'elle soit affichée de la manière la plus optimale, tout en conservant les dimensions initiales de l'image.

De plus, l'affichage du fond d'écran est actualisé à chaque modification de la fenêtre.

VI.2.2. ComingSoon

Cette classe hérite de JFrame et affiche une fenêtre annexe par dessus l'autre. Celle-ci correspond à un bouton affichant Coming Soon pour signaler à l'utilisateur que le contenu n'est pas encore disponible.

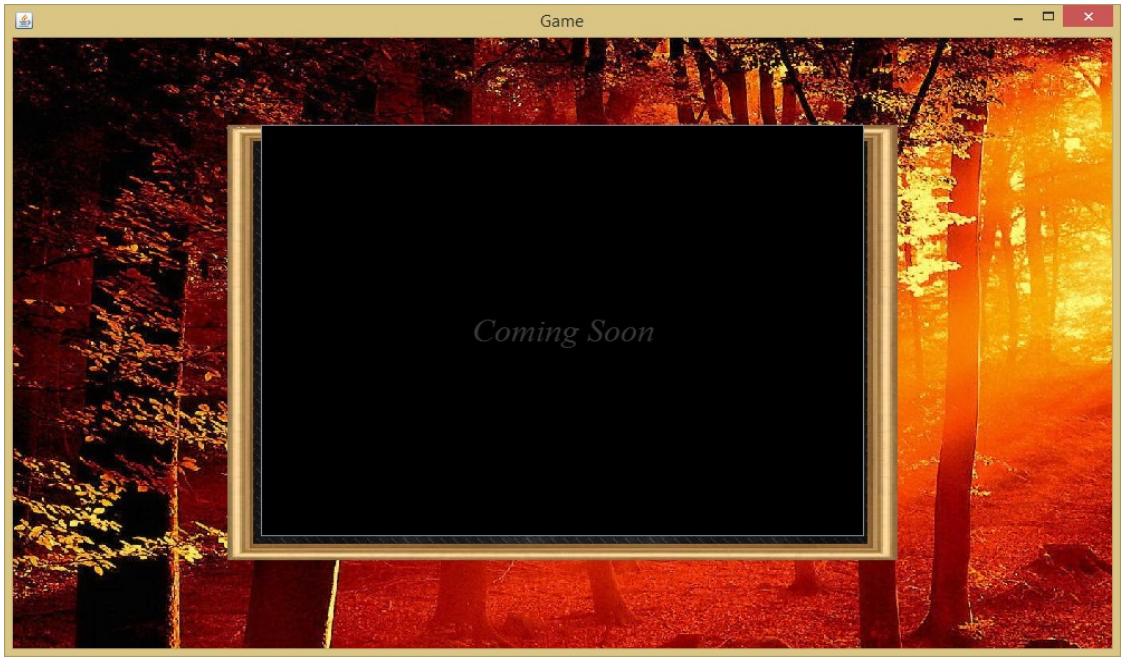

Illustration 7

VI.2.3. LevelButton

Cette classe hérite de JButton et affiche un bouton sur lequel est affichée une image représentant :

- Un aperçu de la carte sous forme de matrice

- Un petit descriptif du niveau afin de donner des informations au joueur

- Un nom centré à l'aide d'une méthode pour distinguer les différents niveaux

VI.2.4. MenuButton

Cette classe hérite aussi de JButton et affiche les boutons du menu principal. Ceux-ci possèdent une police qui leur est propre, importée à partir de fichier. Ils possèdent aussi un fond et une couleur qui changent lorsque l'utilisateur passe sa souris dessus. Ils permettent de sélectionner les autres menus.

VI.2.5 ScreenLevels

Cette classe est celle qui affiche l'écran de sélection de niveau. L'écran est alors composé :

- D'un bouton central qui est un LevelButton

- De boutons sur les côtés qui permettent de naviguer dans les différents niveaux

- D'un bouton retour permettant de revenir au menu principal

De plus, celle-ci est créée à partir d'un Background.

On obtient :

Illustration 8: Affichage actuel de l'aperçu d'un niveau (en cours)

VI.2.6. ScreenMenu

L'objet obtenu est créé à partir d'un objet Background, et possède 5 MenuButton qui permettent au joueur de naviguer parmi les différents menus, grâce aux ActionListener qui récupèrent les clics du joueur. Ce qui donne alors :

Illustration 9

VI.2.7. Window

Cette classe, qui hérite de JFrame, gère les différents écrans de menu à l'aide d'un Layout, et affiche le menu nécessaire lors du clic de souris.

VI.2.8. Menu

Cette classe correspond au main. Celle-ci appelle une instance Window et spécifie sa taille initiale et les différents paramètres de la fenêtre à partir des informations fournies par défaut, ou par l'utilisateur si celui-ci a modifié ses paramètres.

VII. Logger

Le logger permet de récupérer les données lors des tests d'exécution du programme. Celui-ci envoie alors des données relatives au test dans la console afin de pouvoir être analysées afin de régler les problèmes.

Le logger a été implementé à partir des méthodes déjà présentes dans Java avec le package *java.util.logging* afin de proposer un logger personnalisé qui répond aux besoins du jeu, ainsi que les différents tests à effectuer.

De plus, celui-ci peut envoyer les logs dans un fichier, qui peut alors être récupéré plus tard.

Il renvoie aussi un niveau d'erreur permettant de savoir le type : information, erreur.

VIII. Conclusion

Dans le cadre de notre projet, nous devions réaliser un prototype de jeu vidéo en Java. Pour cela, dans un premier temps, nous avons dû spécifier toutes les fonctionnalités et comportements désirés pour notre prototype. Puis dans un second temps, nous avons pu développer le logiciel en suivant deux axes principaux. D'une part nous avons réalisé toutes les classes et toutes les interactions internes nécessaires au fonctionnement du jeu. D'autre part, grâce à la bibliothèque Swing, nous avons mis en place une interface Homme-Machine, permettant à un utilisateur de pouvoir interagir avec le logiciel de la manière souhaitée.

Cela nous a permis de comprendre en profondeur les différentes phases de conception avant l'obtention d'un prototype fonctionnel. De plus nous avons dû apprendre à travailler efficacement et de manière organisée, notamment lors des phases d'intégration de code.

Aussi, ce logiciel peut être enrichi afin d'ajouter de nouvelles fonctionnalités qui n'étaient pas encore présentes, comme la gestion d'une intelligence artificielle. De ce fait, ce projet pourra être continué dans le futur afin de dépasser l'étape de prototype.

IX. Bibliographie

Un site expliquant tout sur le logging en Java, avec les différentes options disponibles : https://www.jmdoudoux.fr/java/dej/chap-logging.htm

Un tutoriel général sur Java, avec une grosse partie sur l'utilisation de Swing et sur son fonctionnement général : https://openclassrooms.com/courses/apprenez-a-programmer-en-java

Le site d'Oracle concernant la documentation Java : https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html

https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html

https://docs.oracle.com/javase/8/docs/api/java/io/BufferedWriter.html

Un tutoriel de java pour recréer quelques jeux connus (Tetris, Pacman…) : http://zetcode.com/tutorials/javagamestutorial/

Des tutoriels vidéo sur Swing : https://www.youtube.com/watch?v=IMjfmWVxd2E

Un site communautaire : https://stackoverflow.com/

X. Glossaire

AWT = Bibliothèque Java pour les interfaces graphiques

GUI = Graphical User Interface

IA = Intelligence Artificielle

id = identifiant

Java = Langage de programmation

JButton = Bouton sous Java programmable, et utilisable

JFrame = Fenêtre sous Java programmable et utilisable

JPanel = Panneau sous Java, pouvant contenir d'autres éléments

Layout = Organisation des éléments, comme une mise en page

Logger = Protocole définissant un service de journaux d'évènements d'un système informatique, c'est-à-dire un historique

PvP = Player versus Player, soit Joueur contre Joueur