DAI Anthony
LEFEVRE Arnaud
MAES Quentin
VANDERSTICHEL Guillaume

21/05/17

Encadrant : Paul Gibson

Projet-Z : Rapport.

TELECOM SudParis

Table des matières

I. Analyse des besoins

A) Contexte du projet

Ce projet informatique se déroule dans le cadre d'un cours de première année (PRO3600) de l'école d'ingénieur de Télécom SudParis, à Evry. Dans le cadre de ce projet, nous avons décidé de créer un jeu de rôle informatique en deux dimensions du type de *Zelda, oracle of seasons* (https://www.puissance-zelda.com/8-Oracle_of_Seasons).

B) Description de la demande

Les objectifs

Ce jeu aura pour but de divertir l'utilisateur. Les fonctionnalités qui devront être offertes par l'application sont les suivantes :

Personnage principal

Le personnage principal, incarné par le joueur, est caractérisé par :

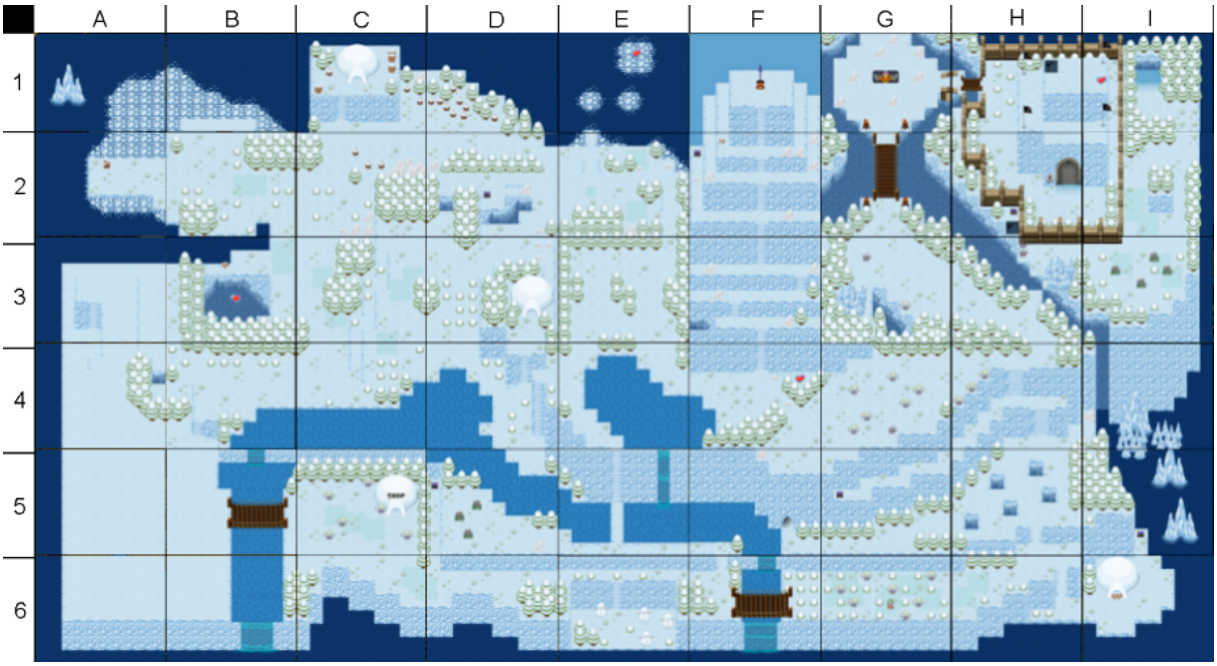
- sa vie maximale , vie qui ne peut augmenter que sous certaines conditions ;
- sa vie symbolisée par des coeurs, elle diminue si le joueur subit des dégâts, augmente s'il récupère un coeur de vie (défini plus bas). Si sa vie est nulle, le joueur est mort et a un game over ;
- sa force. Elle définira les dégâts qu'il inflige ;

Histoire

- Une histoire est racontée au joueur en début de partie pour lui donner un semblant d'objectif ;

Carte

- Il y a une unique zone comprenant 54 sous-cartes ;
- Chaque sous-carte affichée sera en 600px par 480px ;
- Le joueur pourra avoir accès à une cartographie globale du jeu. Les sous-cartes seront obstruées tant que le joueur n'aura pas été dessus ;
- Lorsque le joueur aura été sur toutes les sous-cartes, la carte totale affichée est celle-ci :



***

Note : l'igloo en I6 avait pour but d'accueillir un certain personnage qui amenait à l'île suivante.

Note 2 : le bas gauche de la carte est tel quel dû au fait que l'ordinateur de Guillaume ne marchait plus avant que l'on ne puisse récupérer les décors qu'il avait placé.

***

Le joueur aura accès à différents endroits autres que la carte principale:

- le donjon (décris plus bas) ;
- des igloos ;
- des grottes ;

Igloos

Il existe trois igloos sur la carte dans lesquels le joueur peut rentrer.

Le premier situé sur la sous-carte C1 correspond à la 'maison' du joueur, dedans s'y trouvons l'item bouclier.

Le deuxième situé sur la sous-carte D3 renferme un personnage non-joueur qui propose une énigme permettant d'acquérir l'item plume en cas de bonne réponse à cette énigme.

Enfin le troisième situé en C5 correspond au magasin du jeu où le joueur pourra échanger des essences (définis plus bas) contre des produits.

Grottes

Il y a trois grottes sur la carte dans lesquelles le joueur peut rentrer.

La première se situe en F5. Le joueur y trouve un réceptacle de coeur dans la salle 3, une partie de la clé du donjon dans la salle 4 et un tigre (voir plus loin) dans la salle 5. Les salles sont à découvrir en plaçant des bombes près des murs. Mis à part pour la salle 3, les emplacements des bombes sont indiqués.

La deuxième se situe en I1. Le joueur y trouve l'item arc dans la dernière salle. Pour y accéder, il doit passer par la salle 3 dans laquelle il doit suivre un chemin précis puisque dans le cas contraire, des flèches le blesseront. Pour connaître ce chemin, il doit aller voir une des tombes en D5.

La troisième se situe en A4. Cette grotte est cachée puisque nous y accédons secrètement par un trou. Si le joueur parvient à entrer dedans, il trouvera un marchand lui proposant les gants de force pour 130 essences.

Donjon

- Il y a un donjon sur l'ensemble de la zone, accessible en H2 si le joueur a rassemblé les trois parties de la clé dispersée sur la carte principale ;
- Le donjon a 10 salles ;
- Dans la salle 2, le joueur trouvera une carte du donjon. Pour cela il faut qu'il tue les monstres présents ;
- Dans la salle 5, un monstre appelé grand singe occupe la pièce (voir plus bas pour plus détails) ;
- Dans la salle 6, il trouvera la torche et s'il détruit le tonneau le plus en haut à gauche il trouvera un escalier l'amenant sur le dessus du donjon (sous-carte H1) ;
- Il doit traverser la salle 7 qui est rempli d'araignées afin d'atteindre le mini Boss de la salle 8. Ces araignées s'approcheront du joueur tant que celui-ci n'aura pas eu recours à la torche obtenue à la salle 6 pour les éloigner ;
- Dans la salle 8, il doit vaincre un mini-boss (voir plus bas) ;
- La salle 9 correspond à un labyrinthe. Le joueur doit trouver quelles portes prendre pour arriver à la salle 10 du boss. Pour cela des tonneaux sont arrangés différemment sur chaque niveau de la salle 9, il y a 6 niveaux ;
- Le boss est dans la salle 10 (voir plus bas) ;

Personnages non-joueurs (pnj)

- Il peut communiquer avec des personnages non-joueurs “amicaux” dont le but sera d’aider le personnage à avancer dans l’aventure :

    - le **fantôme**, présent dans l’igloo en D3. Il propose une énigme au joueur et si celui-ci répond correctement, il obtient en récompense l’item plume ;

    - le **bonhomme de neige** qui, en échange de carottes, donne une partie de la clé du donjon ;

    - le **tigre** qui, après avoir été libéré de sa prison de glace, suivra le joueur ;

    - la **dompteuse** qui donne en échange du tigre de la grotte F5 donne l’item potion ;

    - le **vieux marchand** tient une boutique d’objets dans l’igloo situé en C5 ;

    - le **marchand** , dans la grotte A4, qui échange l’item gant de force contre 130 essences ;

- Il y a des pnj “ennemis” sur certaines sous-cartes qui sont réinitialisés à chaque fois que nous quittons la sous-carte ;
- En général, ces personnages hostiles se déplacent de façon aléatoire, et ils auront seulement tendance à se diriger vers le joueur que s’ils subissent des dégâts ;
- Ils peuvent alors infliger au joueur des dégâts de deux manières différentes :
    - soit ils sont “suffisamment” proches du joueur et ce dernier recevra automatiquement des dégât proportionnels à la puissance des pnj considérés ;
    - soit ils ont en leur possession un objet permettant d’infliger des dégâts au joueur ;
- Lorsqu’un pnj ennemi est tué, ce dernier lâche des éléments pouvant être récupérés par le personnage principal, tels que des coeurs de vie ou des essences (voir ci-dessous les items) ;

Les différents pnj ennemis sont :

- la **chauve-souris** : elle se déplace aléatoirement jusqu’à ce qu’elle soit attaquée et ne peut induire des dégâts qu’à proximité du joueur. Elle a 8 points de vie et 1 point d’attaque ;

- le **monstre aquatique** : il se comporte comme la chauve-souris. Elle possède 10 points de vie et 2 points d’attaque ;

- le **slim** : il réagit de la manière que les deux monstres précédents mais de plus, il crée deux petits slims en mourant. Il a 8 points de vie et 2 points d'attaque ;

- le **petit slim** : c'est un monstre commun comme la chauve-souris. Il a 12 points de vie et 3 points d'attaque ;

- le **squelette** : si le joueur est en face de lui, il tirera une flèche mais ne le poursuivra pas. Si, par contre, il se fait blesser, il attaquera le joueur. Il possède 8 points de vie et 3 points d'attaque. Les flèches enlèvent 4 points de vie au joueur ;

- l'**araignée** : elle se dirige directement vers le joueur si celui-ci n'utilise pas l'item torche. Elle s'écarte de lui s'il l'utilise. Elle a 20 de vie et 1 d'attaque ;

- le **grand singe** : présent dans le donjon dans la salle 5, il permet d'accéder à la salle 6 s'il est tué. Il se trouve de l'autre côté d'un fossé et ne peut donc être touché que par les flèches. Le but étant d'associer les flèches aux bombes pour faire plus de dégâts. Il va "chasser" le joueur et cherchera à le blesser en lançant des rayons laser par la bouche. Il a 30 en vie et fait 3 d'attaque ;

- le **mini boss** du donjon : il se positionne aléatoirement autour du joueur toutes les 6 secondes et une seconde après être placé, il se déplace vers la direction où était placé le joueur : par exemple, s'il s'est positionné en dessous du joueur, il ira vers le haut. Il ne peut être affecté que par les bombes. Il a 20 en vie et 4 en attaque ;

- le **boss** du donjon : celui-ci aura deux facettes, il agira de deux façons différentes lors du combat. Au début, il possède une forme humaine et il se déplace en se téléportant. Le joueur ne peut pas l'atteindre à moins de placer une bombe dans une sorte de faille spatio-temporelle que le boss aura laissé derrière lui à chaque téléportation. Arrivé à un certain niveau de vie, il se transformera en un dragon dont les mouvements seront chaotiques. Le combat deviendra alors plus 'classique'. Il a 60 de vie et 4 d'attaque. Il se transforme à 40 de vie ;

Déplacement et interaction

- Le joueur peut se déplacer selon sa volonté avec certaines touches du clavier ;
- Le joueur peut interagir avec l'environnement :
  - en se déplaçant vers un objet ou un personnage non-joueur ;
  - en utilisant un de ses items ;
- Le joueur peut 'tomber' dans un trou ou dans de l'eau : une animation graphique est associée à ce genre d'évènements. De plus, dans le cas échéant, il subit des

dégâts et retourne à la position initiale d'arrivée de la sous-carte ;
- Le joueur subit des dégâts lorsqu'un personnage non-joueur est à une certaine distance de lui ou s'il utilise un item infligeant des dégâts ;
- Lorsqu'il subit des dégâts ou qu'un monstre en subit, le personnage est éjecté dans la direction inverse d'où il a subi le dégât ;
- Il peut récupérer des items ou des objets lorsque ces derniers sont au sol ;

Items et objets

Le joueur a la possibilité d'utiliser des items :

- en les plaçant du sac à la barre d'items où il y a un onglet item de gauche et un onglet item de droite (voir impression écran dans le menu utilisateur) ;

- en appuyant sur une touche du clavier correspondant à l'onglet item de gauche ou à l'onglet item de droite pour activer l'effet correspondant ;

Les différents items et objets:

Les essences :

Les essences correspondent à la monnaie du jeu. Avec un maximum de 999, le joueur peut acheter des objets dans le magasin en échange de ses essences. Il peut en trouver dans des coffres, lorsqu'un monstre est tué ou lorsqu'un buisson est coupé.

Les coeurs de vie :

Les coeurs de vie sont de petits coeurs que les monstres ou les buissons délaissent généralement après s'être faits détruire et que le joueur peut alors récupérer pour se soigner. Un coeur de vie permet de récupérer un point de vie qui correspond à un quart de coeur affiché à l'écran.

Les réceptacles de coeur :

Les réceptacles de coeur sont de gros coeurs dispersés sur la carte qui permettent au joueur d'augmenter son nombre maximal de points de vie : ramasser quatre de ces réceptacles permet au joueur d'augmenter son nombre total de coeurs d'un (c'est-à-dire de gagner quatre points de vie).

Il y en a huit en tout ce qui permet au joueur d'augmenter sa vie de deux coeurs avant d'aller affronter le boss.

L'épée :

L'épée permet d'interagir avec les monstres et certains décors. En effet, lorsque le joueur utilise l'épée, suivant sa direction ou son type d'attaque, il pourra enlever de la vie aux monstres, couper les buissons et donc récupérer les éléments récupérables tels les essences ou les coeurs de vie.

L'épée bénéficie de deux types d'attaques : une simple, uni-directrice et une complexe consistant à faire tourner le joueur sur lui-même pour effectuer deux fois plus de dégâts.

Le bouclier :

Le bouclier est accessible dans l'igloo sur la sous-carte C1.
Il permet d'annuler les dégâts si le joueur se situe face à l'attaque.
Lorsque le joueur utilise le bouclier, sa direction reste la même qu'importe ses déplacements.

La plume :

La plume est accessible dans l'igloo en D3 après avoir répondu à l'énigme posée. Elle permet de 'sauter' par dessus les obstacles non physiques (c-à-d où il n'y a pas de corps : trous et eau).

Le gant de force :

Le gant de force permet de soulever certains types de rochers présents sur la carte. Il est accessible dans la grotte A4 contre 130 essences.

La bombe :

Les bombes sont accessibles dans le magasin. Le joueur peut les utiliser seules ou avec l'arc. La bombe inflige des dégâts de zone (pouvant atteindre les monstres mais également le joueur). Elle permet aussi d'interagir avec certains éléments du décors afin de débloquer des zones ou des objets.

L'arc :

L'arc est accessible dans la grotte de la sous carte I1. Il permet d'envoyer des flèches en fonction de la direction du joueur. Ces flèches permettent d'infliger des dégâts aux monstres (ou au joueur). Si l'item qui se trouve dans la barre d'items au côté de l'arc est une bombe, alors la flèche est associée à cette bombe et elle explosera quand elles atteindra un obstacle, réalisant le même effet qu'une bombe utilisée seule.
Pour récupérer des flèches, le joueur devra tuer des monstres, ces derniers pouvant en lâcher à partir du moment où le joueur aura récupéré l'arc.

La potion :

La potion est accessible auprès de la dompteuse de la sous-carte G6. Elle permet de redonner au héros jusqu'à la moitié de ses points de vie maximals. Cependant, elle n'est utilisable qu'une fois toutes les trente minutes et affiche ainsi un temps d'attente en minutes avant la prochaine utilisation.

La torche:

La torche est accessible dans la salle 6 du donjon. Elle permet d'augmenter la surface visible quand le joueur se trouve dans une pièce sombre (telle que la salle 7 du donjon par exemple). Elle permet aussi d'éloigner les monstres 'araignées'.

Les coffres :

Les coffres sont dispersés sur le terrain et fournissent divers objets ou items.

Les menus

Le menu démarrer :

C'est le premier menu que le joueur voit quand il ouvre le jeu. Il a la possibilité de charger sa partie, de la recommencer ou de s'informer sur les différentes commandes .

Le menu pause :

Il s'agit du menu que le joueur voit lorsqu'il met la partie en pause. Il peut reprendre la partie là où il l'a laissée, retourner au menu principal ou sauvegarder sa partie.

Le menu sac :

Il s'agit du menu où le joueur pourra prendre l'item dont il a besoin. Il pourra aussi voir le nombre de réceptacles qui lui manque avant de pouvoir augmenter sa vie maximale, les partis de la clé qu'il possède et d'autres objets qu'il aura récupérés.

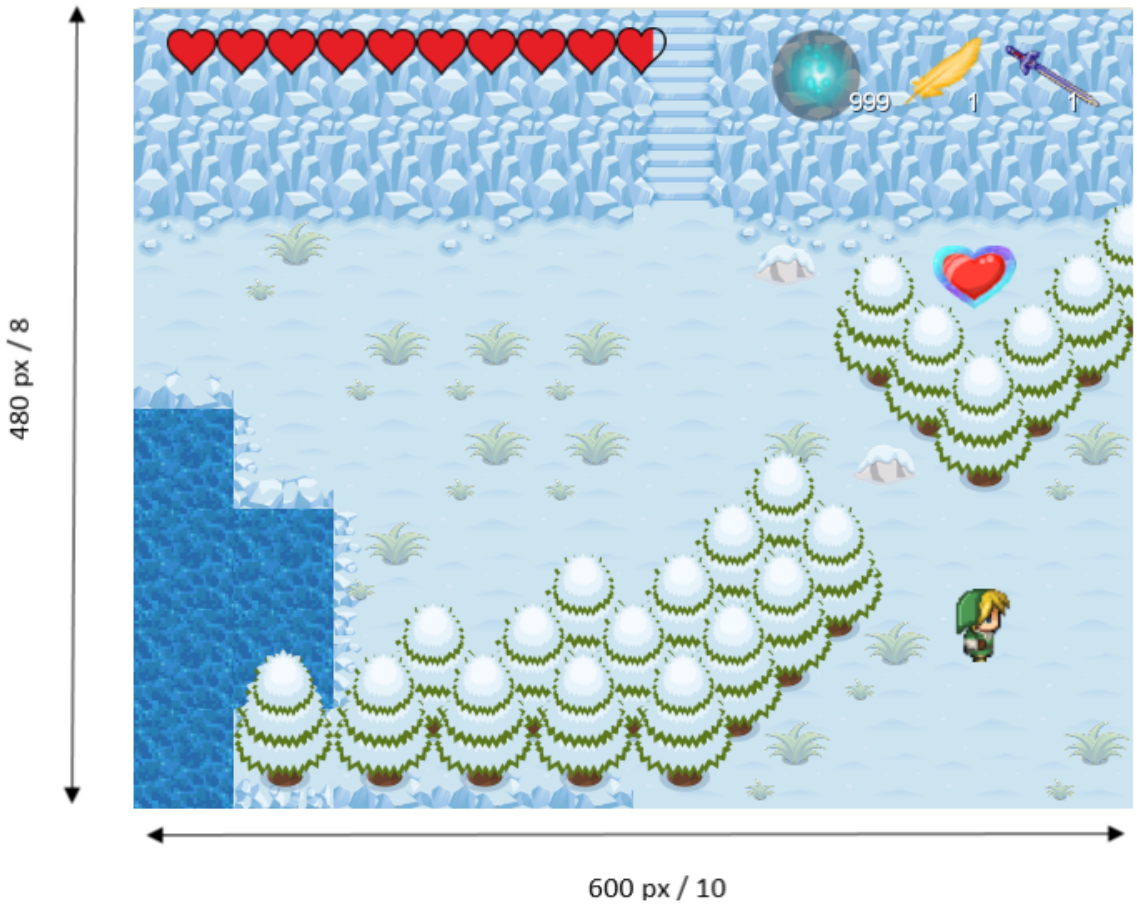
Le menu game over :

Il s'agit à peu près du même menu que le menu pause néanmoins le joueur ne peut pas sauvegarder sa partie sur celui-ci puisque de toute façon lorsqu'il meurt il ne peut redémarrer le jeu qu'à partir de sa dernière sauvegarde) .

Graphismes

Les fonctionnalités graphiques seront affichées à l'aide de l'interface graphique libGDX. Pour cela, nous associons à chacun des personnages, joueur ou non-joueurs, à chaque item et à chaque élément du décor une image en 60px par 60px. Ensuite, nous les placerons en fonction de leurs coordonnées à l'aide du logiciel.
Exemple d'affichage :

Sous carte F4



C) Contraintes

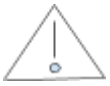
1) Contrainte de délais

Temps : le produit final devra être livré pour le 22 mai 2017 au plus tard.

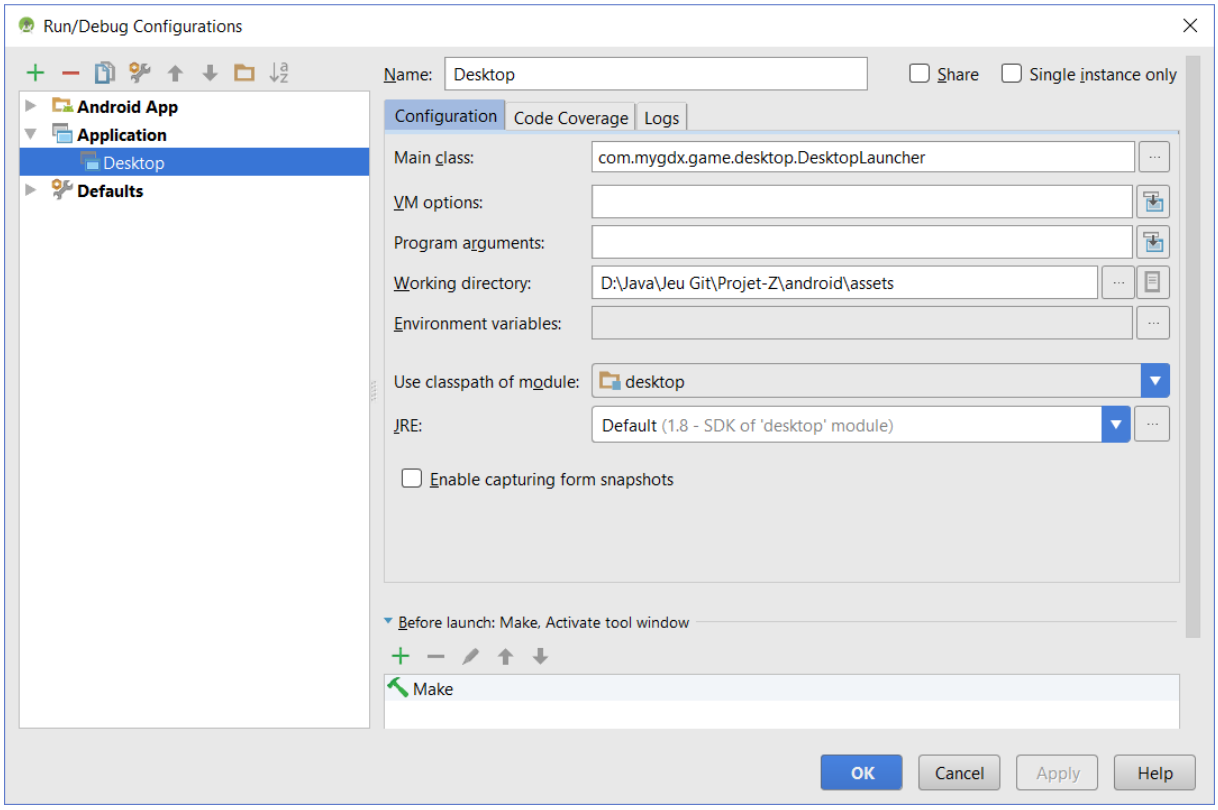
2) Contraintes techniques

Pour ce projet, nous utilisons :

- Le langage Java ;
- Le logiciel Android Studio pour compiler le programme, et il faut avoir associé android sdk à Android Studio ;
- La librairie libGDX pour le graphisme et le son ;
- GitHub pour travailler en équipe ; (https://github.com/Thorond/Projet-Z)



Pour lancer le jeu, il faut se rendre dans les configurations et créer l'application suivante :



II. Spécification fonctionnelle générale et Regroupement modulaire des fonctionnalités

Fonctions principales

1) Sauvegarde

Pour sauvegarder l'avancée du joueur, nous utilisons une classe Sauvegarde dans laquelle nous plaçons toutes les données à enregistrer.

Puis, nous utiliserons la fonction **sendClass(Sauvegarde nom de la sauvegarde)** qui sera appelée à chaque fois que nous irons dans le menu pour effectuer une sauvegarde.

Enfin, la fonction **acceptClass(nom de la sauvegarde)** permettra de chercher les données de la partie courante et la fonction **chargerSauvegarde()** de restaurer la partie à l'endroit où nous l'avions laissée.

La fonction **créerSauvegarde()** permet quand à elle de remettre les paramètres initiaux pour créer une nouvelle sauvegarde.

Pour tout cela, nous utilisons l'interface 'Serialization' de Java.

2) Carte

Afin de créer la carte et de la diviser en sous-cartes, nous allons avoir recours aux fonctions suivantes :

- La fonction **sousMap()** de chaque classe SousMapXX : elle disposera les différents éléments du décor à l'aide de coordonnées et sera appelée en fonction de la position du joueur ;

- la variable **positionSousMap** permet de savoir sur quelle sous-carte le joueur est ;

- la fonction **posiSousMap(MainCharacter)** : permet le changement de la sous-carte du joueur en fonction de ses coordonnées et de la sous-carte précédente. Cette fonction permet ainsi de réinitialiser les objets, les corps et les pnjs ;

- la fonction **affichageDeSousCarte(GameMain)** permet l'affichage de la sous-carte sur laquelle on se trouve ;

- la fonction **destructionDesCorps()** permet la destruction des corps de toutes les sous-cartes. En effet, même si chaque sous-carte détruit ses corps lorsque nous la quittons, lorsque le joueur meurt par exemple, il faut détruire les corps (voir page 30 pour définir ce qu'est un corps) ;

Pour le changement de carte, nous avons mis au point un défilement permettant de changer élégamment de sous-cartes : la fonction **défilementDeMap(GameMain)** permet un défilement adapté en fonction de la sous-carte où nous nous trouvons et de la direction de changement de sous-carte.

Ces fonctions et variables sont dans les classes **PlacementMainZoneGlace** et **GestionDesMapsZoneGlace**.

Pour les éléments de décors avec lesquels nous pouvons avoir une interaction, tels que les buissons, les réceptacles, les trous, et autres, une classe **CadrillageMap** permet de contenir des fonctions définissant une matrice qui représente la zone affichée. Ainsi, pour chaque élément de cette matrice nous lui associons un type et nous pouvons ainsi interagir avec eux si la position du joueur correspond à une position où il y a un type particulier.

3) Menus

Les fonctions **updateDémarrage(float)** et **affichageMenuDémarrer(GameMain)** qui apparaîtront au lancement du jeu permettent d'afficher et de choisir entre lancer une partie, en créer une nouvelle (en écrasant celle déjà existante) et de consulter les options.

De même, les fonctions **updatePause(delta)** et **affichageMenuPause(GameMain)** permettent d'afficher et d'accéder à un menu où le joueur pourra mettre en pause sa partie, la sauvegarder, quitter le jeu, ou éventuellement retourner au menu principal.

Il existe les mêmes fonctions pour les autres menus.

4) Personnages

Chaque personnage, qu'il soit non-joueur ou principal, aura recours au constructeur **Characters(World (intrinsèque à libgdx), Texture (de même), health, strength, (position) x, (position) y, direction)**. Cela définira les points de vie, la force, des dégâts qu'il peut infliger en fonction de cela, une position (x,y) et une direction, "bas", "gauche", "droite" ou "haut". La texture correspondra au graphisme associé (voir plus bas).

Certains personnages non-joueurs tels que la dompteuse ou le marchand ont des fonctions **update***(float)** permettant l'acquisition des touches du joueur et de dérouler un scénario.

5) Représentation

La fonction **render(float)** (fonction libGDX) est la fonction principale du code. Elle est appelée avec une certaine fréquence. Elle vérifie les différents états des menus et du joueur et appelle en fonction de cela les fonctions **update()** (voir plus bas sa conception).

Les fonctions **update***()** permettent de récupérer les choix de l'utilisateur et ainsi d'appliquer les modifications que cela entraîne.

6) Items

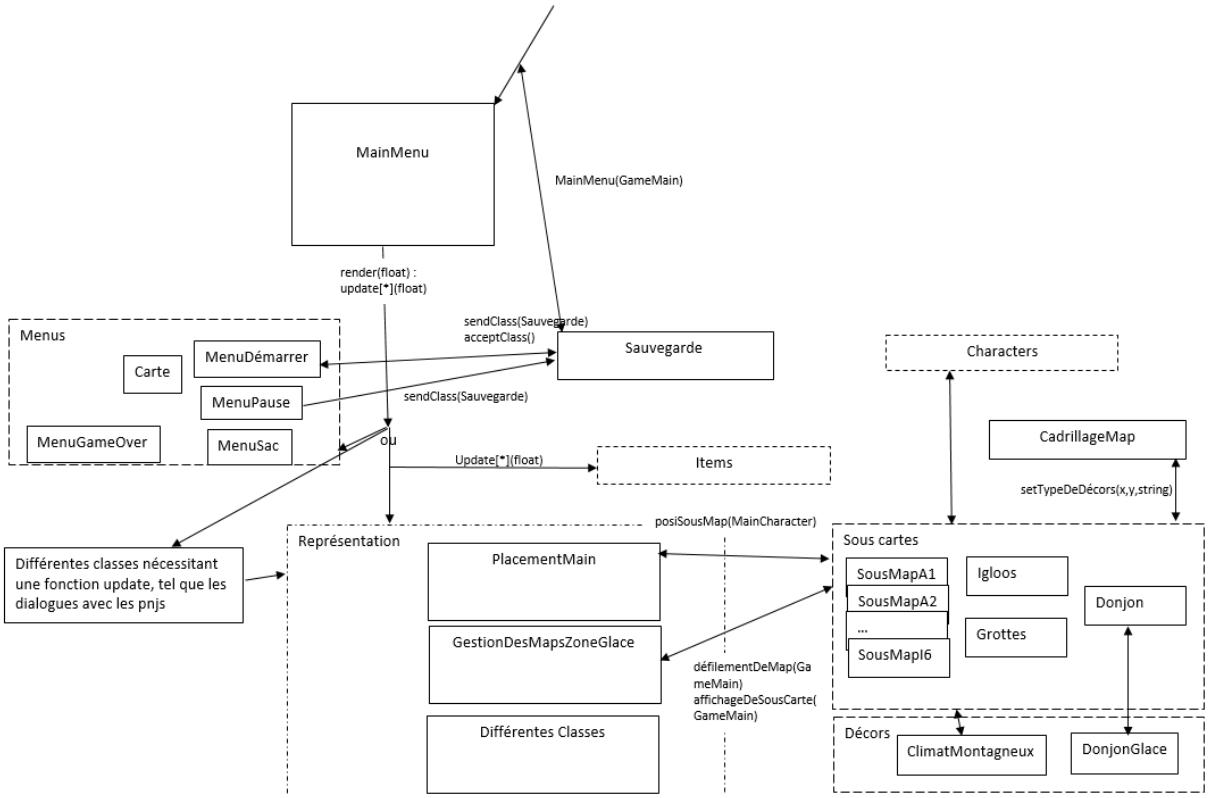
La fonction **utilisationItem(MainCharacter)**, commune à chaque item, permettra d'activer sa spécificité. Par exemple, si l'item est l'épée alors son utilisation permettra d'infliger des dégâts aux ennemies positionnés en face du personnage principal, ou de couper les herbes hautes, etc.

7) Main

Le constructeur **MainMenu(GameMain)** permet de définir les variables et les classes du jeu. Définition des items, du personnage principale, acquisition de la sauvegarde, etc.

III. Description du flux des données entre les modules

Ici, nous allons présenter les échanges entre les différents modules afin de mieux comprendre comment ils s'utilisent les uns par rapport aux autres.



Beaucoup de classes renferment des valeurs à sauvegarder. En effet, les données du personnage telles que sa position, sa vie, ce qu'il y a dans le sac, etc. ; et certains éléments du décor avec lesquels nous avons interagi et qui ne devront plus apparaître constituent des données que nous devons conserver.

C'est pour cela que nous avons placé la classe Sauvegarde au centre mais que nous ne l'avons pas reliée à toutes les classes avec lesquelles elle est en contact par souci de visibilité.

Les sous-cartes ont besoin d'images et de fonctions représentant l'animations des images, d'où son lien avec les décors. Elles ont besoin aussi de la classe *CadrillageMap* permettant de définir les cases spéciales (par exemple, l'eau). Enfin, elles ont aussi besoin d'appeler les classes et fonctions des pnjs ennemies.

De même, le programme a besoin d'accéder aux propriétés des items par l'intermédiaire de la fonction updateInGame(float).

Pour la représentation, nous récupérons les sous-cartes et les données des personnages pour afficher à l'écran et pour gérer la physique, les collisions, les interactions, etc.

IV. Conception détaillée

Les projets et classes de base libgdx :

Lors de la génération du projet par le logiciel de libGDX, nous avons obtenu trois projets nommés ‘android’, ‘desktop’ et ‘core’.

Dans le projet ‘android’, le seul endroit où nous apportons des modifications est le sous-dossier ‘assets’ où nous plaçons les images qui doivent être utilisés pour le jeu et éventuellement les musiques.

De même, dans le projet ‘desktop’, la seule classe que nous modifions pour configurer le jeu est ‘DesktopLauncher’ et à l’intérieur nous avons mis les dimensions de l’écran.

Tout le reste se situe dans le projet ‘core’.

Les variables de base de libgdx :

Afin de pouvoir utiliser certaines propriétés de libGDX, nous avons utilisé à travers le code certaines de ses variables essentielles : World world, GameMain game, Body body ;

Les fonctions de base de libgdx :

Afin de pouvoir récupérer les choix du joueur sur le clavier, on utilise la ligne de code suivante, intrinsèque à libGDX :

```
Gdx.input.isKeyJustPressed(Input.Keys.'ToucheClavier')
```

Ceci renverra ‘true’ si la touche a été actionnée, ‘false’ sinon. Une variante ‘`isKeyPressed`’ existe pour savoir si le joueur garde la touche appuyée.

La fonction *render(float)* :

Cette fonction, que nous pouvons considérer comme la fonction principale du jeu, est la fonction qui est appelée par le compilateur un certain nombre de fois par minute (variable float, défini par libGDX).

```
public void render(float dlt) {
      si menuDémarrer : afficher le menu et updateDémarrage(dlt)
      sinon
            si gameOver : afficher le gameover et updateGameOver(dlt)
            sinon si pause : afficher et update Pause
            sinon si carte : afficher et update Carte
            sinon
                  si sac : afficher et update Sac
                  sinon
                        si scenario personnage principal :
                        updateJoueur(dlt)
                        sinon si scénario fantôme : updateGhost(dlt)
                        sinon si [ etc ]
                        fin si
                        updatePlayer()
                        si zoneGlace :
                              si défilement : lancer le défilement
                              si ! défilement :
                                    acquisition de la position
                                    ajuster taille image joueur
                                    afficher les sous cartes
                              fin si
                        fin si
                        affichage des coeurs de vie, des essences, des
                        flèches, etc.
                        affichage du personnage principal
                  affichage des textes de dialogues
                  affichage des items, de la monnaie et de la barre de
                  vie
}
```

Les fonctions ‘update’ :

Ces fonctions déterminent l’état du jeu. En effet, elles permettent l’acquisition des touches et vont réagir en fonction de l’état dans laquelle nous nous trouvons. Il peut s’agir d’un update de menu, de l’update ‘en jeu’, ou de l’update suite à une situation avec un personnage et des dialogues, par exemple.

```
void updateInGame(float dt){
      si joueur en vie :
            s’il reçoit un item ou un réceptacle :
                  graphisme associé
                  vitesse nulle
                  si joueur appui sur entrée : fin
            sinon s’il est en train d’utiliser l’épée :
```

```
      animation suivant l'attaque
      si l'attaque est fini :
            épée.utilisationItem(Link);
      fin si
sinon :
      si le défilement de carte n'est pas enclenché :
            si le joueur tombe dans un trou :
                  animation et affichage
                  après une seconde, il est replacé à
                  l'entrée de la carte s'il est en vie
            sinon :
                  s'il est touché :
                        attente de 0.3 sec
                  sinon si utilisation de la plume :
                        attente de 0.7 sec ou fin si le
                        personnage ne bouge plus
                  sinon :
                        si touche appuyé est Q :
                              ajout vitesse à gauche
                              si bouclier pas utilisé,
                              direction = gauche
                              représentationLink(Link)
                        de même pour la droite, le haut et
                        le bas
                        sinon si aucune touche :
                              ralentissement et attribution
                              de l'image associé
                        fin si
                  fin si
                  si touche = O :
                        arrêt du joueur et des monstres
                        pause = true
                  fin si
                  si touche = P :
                        arrêt du joueur et des monstres
                        carte = true
                  fin si
                  si touche = K et s'il y a un item et si
                  le joueur n'est pas dans un igloo :
                        utilisation de l'item de gauche
                  fin si
                  de même pour L
                  sinon si touche = M :
                        arrêt du joueur et des monstres
                        sac = true
                  fin si
                  si le bouclier a été utilisé et que le
                  joueur garde le doigt sur la touche, le
                  bouclier reste activé
```

```
                              si le bouclier est utilisé et que le
                              joueur utilise l'épée : première attaque
                              sinon :
                                    si il reste appuyé plus de 0.7 sec :
                                          deuxième attaque
                                    sinon : première attaque
                              // après c'est une succession de fonction
                              // permettant de détecter ce qu'il y a en
                              // jeu, les items, les coeurs de vie, les
                              // personnages amicaux, etc.
                              PlacementMainZoneGlace.setDéplacement(Lin
                              k);
                              // détection des trous en fonction et si
                              // la plume n'est pas utilisée
                        fin si
                  fin si
            fin si
            si la vie < 0 :
                  gameover = true
            fin si
      sinon si :
            // si le joueur veut passer l'écran noir du gameover,
            // il appuie sur entrée
}
```

Note : ces deux fonctions ont été détaillées puisqu'elles constituent les deux fonctions principales du jeu. Pour les fonctions suivantes, il n'y aura qu'une description de ce qu'elles font.

```
void updateSc1Ghost(float dt){
      // cette fonction va dans un premier temps placé le joueur
      // devant le personnage fantôme puis va faire défiler le
      // texte lorsque loe joueur appuiera sur entrée
      // la particularité est que lorsque l'état du dialogue sera
      // arrivé au niveau 9, on lancera ici l'update associé à
      // l'alphabet
}

void updateAlEtAc(float dt){
  AlphabetEtAcquisition.acquisitionTouche();
  // sans rentrer dans les détails, cette fonction update pour
  // l'alphabet va permettre au joueur d'écrire sa réponse à l'aide
  // des touches du clavier en pouvant retourner en arrière dans son
  // mot
}
```

Toutes les fonctions updates des menus tel que `void updateSac(float dt){}` , `void updateGO(float dt){}` et autres fonctionnent de la même manière : un curseur est affiché et le déplacement par les touches 's','q','z' et 'd' existe. Pour faire son choix il suffit d'appuyer sur entrée et cele réagira en fonction de la position du curseur ;

Les graphismes :

Comme évoqué précédemment, les images utiles au jeu sont répertoriées dans le sous-dossier 'assets' du projet 'android'.

Pour représenter les graphismes, la bibliothèque de libgdx possède deux types de variables que nous pouvons représenter.

Il existe la basique, *Texture* , variable à laquelle nous affectons une image :

Texture texture = new Texture("chemin/jusqu'à/l'image.png") ;

L'autre est en fait issue d'une classe *Sprites* héritée. Nous l'utilisons pour les personnages afin de pouvoir contrôler les différents paramètres de l'image (taille, longueur, etc.). Elle placée dans le constructeur sous la forme *super(texture)* ;

Après, qu'il s'agisse d'une Texture ou d'un Sprite, nous utilisons la fonction libgdx pour placer l'image :

game.getBatch().draw(Texture/Sprites , int x , int y) ;

Pour avoir plus de manoeuvre sur le Sprite, nous pouvons changer ses caractéristiques avec les fonctions que libGDX nous propose et l'afficher en faisant : *sprite.draw(game.getBatch())*

Les images doivent nécessairement être sous le format png.

La musique :

Bien que la musique ne soit pas présente en jeu, pour en rajouter une il faut utiliser la classe Music de libGDX de cette façon :

```
public static Music music =
Gdx.audio.newMusic(Gdx.files.internal("musique/NomDeLaMusique.mp3"))
;
```

puis il faut la lire, voir l'éteindre :

```
music.play();
music.setLooping(true);
music.dispose();
```

La musique doit nécessairement être sous le format mp3.

Les corps :

Les corps sont les variables qui définissent les endroits où il y a quelque chose d'inaccessible pour d'autres corps. Ainsi, puisque le personnage principal est muni d'un corps à sa conception, il ne pourra se déplacer sur les autres corps.

Soit les corps sont 'dynamic' et sont soumis aux forces créé par libGDX, soit ils sont 'static' et ne sont pas déplaçables.

La fonction à utiliser est **createBody((position) x, (position) y , largeur, hauteur)**.

Créer un corps en (x,y) avec une certaine hauteur et une certaine largeur créera un corps solide rectangulaire autour de cette position (x,y). Son centre sera (x,y).

La gestion des cartes et le positionnement du joueur:

Plusieurs classes et fonctions sont nécessaires à cela, comme expliqué précédemment. Nous allons alors juste présenter les fonctions importantes ici :

Celle de PlacementMainZoneGlace :

```
public static void réinitialisation(){
	// permet de réinitialiser les éléments qui étaient au sol
	// tel que les coeurs de vie, les essences, les bombes, les
	// flèches etc. et de remettre à 0 les types de décors ( trou
	// et eau )
}
public static void posiSousMap(MainCharacter perso){
	// cette fonction permet de changer de map en fonction de la
	// direction, de la position du joueur et de la sous carte sur
	// laquelle on etait, elle va changer la variable
	// nouvelle sous carte  positionSousMap en la
	// elle fera appelle à réinitialisation() à chaque fois
	// elle lancera le défilement en mettant la variable défilement
	// en 'true'
	//
	// la logique du labyrinthe de la salle 9 du donjon apparaît
	// ici
}
public static void détectionEscalier(MainCharacter Link){
	// cette fonction va faire comme précédemment à la différence
	// qu'elle vérifiera que le joueur se trouve sur une dalle de
	// type 'Escalier' ( avec CadrillageMap )
}
```

Les deux fonctions suivantes `void setDéplacement(MainCharacter Link)` et `void détectionTrou(MainCharacter Link)` (et détectionEauP() bien que cette fonction soit en fait la même que détectionTrou()) sont des fonctions qui aident à faire en sorte que le joueur ne tombe pas sans comprendre dans un trou ou dans l'eau. En effet, puisque la position d'un corps se situe à l'extrémité bas gauche de l'image affichée, lorsque nous nous rapprochons du bord par la droite, nous pouvions tomber alors que nous sommes censés avoir encore de la marge.

Ainsi, la première fonction permet de savoir si le joueur se situe plus vers le haut, vers le bas, vers la gauche ou vers la droite d'une même dalle. Aussi, pour la détection du trou ou de l'eau, nous pouvons affiner la recherche en fonction de cela. Par exemple, si nous nous trouvons plus sur la gauche de la dalle, alors dans la vérification du type de dalle à l'aide de CadrillageMap se fera avec la position du personnage plus un petit quelque chose.

Ensuite, si le 'trou' est bien présent, nous déclenchons la chute du personnage.

Celle de GestionDesMapsZoneGlace :

```
public static void défilementDeMap(GameMain game){
	// cette fonction est appelé quand on change de carte
	// elle permet le défilement
	// elle regarde la position que le joueur avait au moment du
	// changement pour comprendre de quel côté il venait, et
	// en sachant la carte sur laquelle il est maintenant , elle va
	// afficher l'ancienne carte et la nouvelle on les décalant
	// petit à petit
	// à la fin du défilement, elle va enregistrer la position du
	// joueur, position qui permet de le replacer quand il tombe
}
public static void affichageDeSousCarte(GameMain game){
	// c'est la fonction qui va afficher le décors en fonction
	// de la position du joueur ( positionSousMap )
	// elle va aussi dévoiler la sous carte sur laquelle on est
	// dans la carte
}
public static void destructionDesCorps(){
	// bien que les corps soient détruits à chaque changement de
	// sous carte, cette fonction est utile pour détruire les corps
	// quand on meurt par exemple
}
```

Les sous-cartes :

Les graphismes :

Le designer doit placer les graphismes comme expliqué ci-dessus en faisant attention aux 'couches' (pour que le rendu global soit cohérent). Les dalles des images étant en 60px par 60px et les graphismes secondaires (du type arbre, rocher , etc.) sont du même ordre de grandeur.

Les corps :

Afind e placer les corps des objets, chaque sous-carte doit définir deux variables pour chaques corps : une fonction créant les corps et une fonction les détruisants.

Variables :

La première variable est celle correspondant au corps en lui même, *Body* ;

La deuxième variable correspond au fait que le corps est bien présent ou non, c'est un *boolean* ;

Création : Dans la fonction *createBodyAndType(World)*

Ici il faut créer le corps si celui ne l'est pas déjà avec le code :

```
if ( boolean == false ) {
   Body = ClimatMontagneux.createBody(*,*,*,*);
   boolean = true;
}
```

Destructions : Dans la fonction *destroyBody()*

Nous détruisons le corps s'il existe :

```
if ( boolean) MainMenu.world.destroyBody(Body);
boolean = false;
```

Les décors :

ClimatMontagneux

Dans la classe ClimatMontagneux, nous retrouverons 90% des textures que nous utilisons en jeu pour les décors. Il y a aussi plusieurs fonctions permettant l'animation des décors tels que l'eau, la cascade, les torches, etc.

Il y a aussi deux fonctions pour créer les corps, la fonction habituel `Body createBody(float x, float y, int largeur, int taille)` présentée précédemment et une fonction un peu personnalisée avec des tailles et largeurs fixes en fonction de l'élément que nous voulons placer : `Body createBodyPerso( String décor, String type, float x, float y.`

DonjonGlace

Dans la classe DonjonGlace, bien que nous y retrouvons des textures et des fonctions comme précédemment, il s'y trouve un peu plus de logique.

La détection des serrures : `détectionSerrure(MainCharacter Link)` et `détectionSerrureBoss(MainCharacter Link)`. Si le joueur a récupéré les clés correspondantes et qu'il se place en face de la serrure alors le mécanisme activera l'ouverture de la porte du donjon et la 'porte' dans la salle 3 du donjon.

Nous y trouverons ainsi les deux animations d'ouverture de porte de donjon et de porte sur la carte des totems.

Les personnages non joueurs :

Ennemis

Pour que le personnage puisse attaquer les monstres, une variable static existe dans la classe Pnj qui va être rempli à chaque fois que nous entrons sur une nouvelle sous-carte et qui va être vidé quand nous en sortons. Il y a le nombre de monstres présents sur la sous-carte ainsi qu'un tableau Pnj[] les regroupants.

Les personnages non-joueurs ennemis héritent tous de la classe Pnj et possèdent donc tous des variables et des fonctions de déplacement, d'attaque et autres. Les voici :

```
void déplacementAléa(){
      // un nombre aléatoire est généré et en fonction de s'il se
      // se situe entre 0.25,0.5,0.75 et 1 va donner toutes les
      // secondes une vitesse et une direction au monstre
      // mais on va vérifier que le monstre ne se situe par en
      // bordure de map et qu'il n'y a pas de 'trou' ou 'd'eau' dans
      // la direction dans laquelle il va
      // au bout de 0.7 seconde le monstre s'arrête momentanément
}
void déplacementVersJoueur(){
      // en fonction de sa position relative par rapport au
      // personnage principal, le monstre ajustera sa position
      // pour se rapprocher du joueur
}
void déplacement(){
      // si le monstre est touché, il va être rejeté donc pendant
      // 0.3 seconde il ne va pas avoir la maîtrise de son
      // déplacement
      // puis s'il est attaqué il va se déplacer vers le joueur
      // sinon il se déplacera aléatoirement
}
```

```
void drop(){
      // cette fonction permet de créer un coeur de vie ou des
      // essences ou encore des flèches à récupérer quand le monstre
      // meurt
}
void subirDégats( int cha, String direction){
      // si le pnj a une vie < 0 après l'attaque, son corps est
      // momentanément envoyé en -500 -500 avant d'être détruit en
      // changeant de map
      // la fonction drop() est appelé
      // sinon,
      // en fonction de sa position il sera rejeté et subira
      // des dégâts
}
void subirDégatsBombe( Bombe cha, int x, int y ){
      // cette fonction revient à peu près au même sauf qu'il faut
      // ajuster la position relative du monstre par rapport à la
      // bombe pour l'éjection et de plus le dégâts est différent
}

void infligéDégatLink(){
      // cette fonction permet d'infliger des dégâts au personnage
      // principale toutes les 0.7 secondes si celui ci se trouve
      // à une certaine distance du pnj et s'il n'utilise pas le
      // bouclier dans le bon sens
}
```

Tous les monstres possèdent aussi une fonction permettant leur animation en jeu ;

```
void représentation**(){
      // cette fonction va changer la texture associé au monstre
      // toutes les 0.2 sec en fonction de sa direction et pour
      // certain de s'il est arrêté ou pas
}
```

Ces fonctions ont néanmoins été réécrites pour certains monstres ayant des caractéristiques qui diffèrent. Voici les plus importantes réécritures :

pour l'araignée :

```
@Override

void déplacement(){
      // si l'item torche est activé, les araignés vont se déplacer
      // en s'éloignant du joueur, sinon elle se rapprocheront
      // elle s'immobilise si elle sont sur le point de sortir de la
      // carte
}
```

pour le grand singe (ape en jeu) :

```
void déplacement(){
      // ce monstre va suivre le joueur met uniquement vers le bas
      // ou vers le haut, il s'arrêtera quand il sera au même niveau
      // que le joueur pour lancer son attaque puis repartira au bout
      // d'une seconde
}
void infligéDégatLink(){
      // puisque le grand singe ne sera jamais en contact avec le
      // personnage principal, son attaque, qui est composé d'un
      // laser fera des dégâts au joueur
      // il fera des dégâts si le laser est présent, si le joueur se
      // situe au niveau du laser , si celui ci n'utilise pas le
      // bouclier et avec une limite de dégâts d'une fois par
      // quart de seconde
}
```

pour le mini boss :

```
void déplacementVersJoueur(){
      // le mini boss n'utilisera uniquement cette fonction
      // il va se placer au dessus ou en dessous ou à droite ou
      // à gauche du joueur avec une probabilité équiprobable et
      // toutes les 7 secondes
      // une seconde après s'être placé il se déplacera dans la
      // direction du joueur au moment où il s'est placé
}
void subirDégats( int cha, String direction){
      // ici il n'y a plus rien, il ne peut subir de dégâts autrement
      // qu'avec la bombe
}
void infligéDégatLink(){
      // cette fonction a été réadapté à la taille du monstre
}
```

pour le boss :

```
void déplacementAléa(){
      // cette fonction a été légèrement réécrite pour créer un
      // déplacement plus chaotique et avec le moins d'arrêt possible
}
void déplacement(){
      // ici, soit le boss a plus de 40 de vie, et il va appeler
      // une fonction déplacementTéléportation() présenter plus bas
      // soit il a moins de 40 et on appelle la fonction
      // déplacementAléa()
}
void subirDégats( int cha, String direction){
      // cette fonction a aussi été modifié pour différencier les
      // deux phases du combat avec le boss
```

```
      // lors de la première phase ( > 40 de vie ) le boss ne peut
      // subir des dégâts norma
}
void subirDégatsBombe(Bombe cha, int x, int y ){
      // de même ici,les bombes ne font pas des dégâts ordinaire
      // dans la fonction explosion() de la bombe des lignes de
      // code on été rajouté pour que le dégât soit effectué
      // uniquement si la bombe est placé sur la faille que le boss
      // laisse derrière lui lors de la téléportation
}
```

Enfin, certaines classes de monstres possèdent des fonctions particulières, les voici :

Les squelettes :

```
void attaque(MainCharacter Link){
      // les squelettes vont envoyer des flèches avec une limite
      // d'une toutes les 0.8 sec, vers le joueur si celui se trouve
      // 'en face' d'eux
}
```

Le grand singe :

```
void attaque(){
      // si la position du personnage principal coïncide avec celle
      // du singe, celui ci s'arrêtera et commencera à lancer son
      // laser au bout de 0.2 sec
      // si le joueur n'a pas bougé d'ici là, l'attaque est réitéré
      // au bout d'une seconde
}
```

Le boss :

```
void déplacementTéléportation(){
      // cette fonction va donc décrire le déplacement du boss lors
      // de sa première phase
      // s'il est touché par le joueur à l'épée, ou si 5 sec se sont
      // écoulés depuis sa dernière téléportation, il va l'activer
      // il va ainsi se repositionner aléatoirement sur la sous carte
      // et laisser derrière lui un 'résidu' de téléportation
}
```

Amicaux

Chaque personnage non-joueur amical, ou la plupart en tout cas, possède un état de scénario permettant de suivre l'avancée du joueur dans l'interaction qu'il a avec celui-ci. Cet état correspond à un nombre entier et est décrit dans chaque classe.

D'autre part, chacune de ses classes possède une fonction de représentation/animation, une fonction de détection et une fonction update permettant de contrôler l'échange avec le pnj.

Seul le ‘marchand’ possède une fonction déplacement tel que les pnj ennemis.

Le totem

Le but de ce défi est de replacer les deux parties du totem au centre dans le bon sens. Quand le joueur va rentrer sur le damier de la carte, la fonction `void déplacementTotems()` va être appelée.

```
void déplacementTotems(){
      // si le joueur se déplace vers la gauche, le totem 1 ( celui
      // en bas à droite au début ) monte et le totem 2 descend
      // s’il se déplace vers la droite, inversement
      // s’il se déplace vers le haut, le totem 1 va à droite
      // le totem 2 va à gauche, s’il se déplace vers le bas
      // inversement
}
```

UtilisationItem() :

Pour chacun des items, il y a une fonction récupération de l’item qui permet au joueur de récupérer cet item si le joueur est placé au-dessus de cet item. Cela affiche ensuite une animation indiquant au joueur qu’il vient de récupérer quelque chose.

Le bouclier

```
void utilisationItem( MainCharacter cha) {
      // très simplement, si le bouclier n’est pas encore utilisé,
      // on diminue la vitesse d'affichage de l’animation et on met
      // le booléen de l’utilisation du bouclier sur vrai.
}
```

La plume

```
void utilisationItem( MainCharacter cha) {
      // de même ici, le booléen de la plume est changé en vrai et
      // on lance le chrono pour la durée d’utilisation
}
void timerPlume(MainCharacter Link){
      // après 0.7 sec ou si le joueur rencontre un obstacle, le
      // booléen de la plus retourne à faux
}
```

La torche

```
void utilisationItem( MainCharacter cha) {
      // soit la torche est allumé et elle s'étend
      // soit l'inverse, à l'aide d'un booléen
}
```

Le gant de force

```
void utilisationItem( MainCharacter cha) {
      // le gant de force va permettre de modifier le terrain en
      // vérifiant que dans CadrillageMap le type de la dalle
      // correspond à petitePierre dans la direction dans laquelle le
      // joueur regarde
      // si c'est le cas, il va donc changer le type de décors et
      // détruire les corps de la carte afin que celle ci assimile
      // la nouvelle information
}
```

La potion

```
void utilisationItem( MainCharacter cha) {
      // la potion a un minuteur aussi, si la potion a été utilisé
      // il y a plus de trente minute ( y compris après avoir
      // éteint le jeu ) alors on peut activer son effet
      // son effet étant d'ajouter à la vie actuelle du joueur
      // la moitié de sa vie maximale
      // on réinitialise le chrono
}
void affichageTemps(){
      // cette fonction met remplace nombre d'item par temps qu'il
      // reste avant la prochaine utilisation
      // ( puisque le nombre d'item est ce qui est affiché en jeu
      // - voir plus bas dans le menu sac )
}
```

L'épée

```
void utilisationItem( MainCharacter cha) {
      // s'il s'agit de la première attaque, c'est à dire que le
      // joueur n'a pas gardé son doigt appuyé très longtemps
      // sur la touche du clavier, alors :
      // - on récupérera des objets s'ils sont présent
      // - on regardera si le joueur est en face d'un élément
      // destructible tel un arbuste ou une plante et si tel est
      // le cas on appellera les fonction permettant de faire
      // apparaître des objets
      // - on regardera la présence ou non de monstre dans la
```

```
        // direction du joueur et on effectuera des dégâts en fonction
        // s'il s'agit de la seconde attaque, il se passera la même
        // chose mais sur une zone tout autour du joueur ( attaque
        // circulaire )
        // c'est dans cette fonction qu'on lancera le chrono pour
        // savoir depuis quand le joueur appui sur la touche
        // et la vitesse du joueur est nulle quand il attaque
}
```

De plus, il y a les deux fonctions représentant l'animation de l'attaque du joueur en fonction du type d'attaque.

****

Pour la bombe, les flèches, et les objets qui sont présentés plus loin, chacune de ces classes possèdent un tableau. Ce tableau se remplit à chaque fois que l'un de ces items ou objets est en jeu et c'est ce qui permet l'interaction avec le joueur et l'affichage.

C'est pourquoi ces items/objets sont plus complexes que les items précédents.

Ils ont des positions, des états et des minuteurs.

La bombe

```
void utilisationItem(MainCharacter cha){
        // si le nombre de bombe est positif, remplirBombes(), et on
        // retranche une bombe
}
void remplirBombes(int X, int Y){
        // lorsque cette fonction est appelé, est va parcourir le
        // tableau des bombes et placé une nouvelle bombe à
        // l'emplacement où il n'y en a pas ou que la bombe précédente
        // à exploser
        // elle va initialisé son état
}
void représentationBombe(GameMain game){
        // cette fonction va gérer la représentation mais aussi va
        // changer l'état des bombes du tableau au bout de 0.5 sec
        // lorsque l'état aura atteint 4, la fonction explosion()
        // est appelée
}
void explosion( ){
        // s'il y a un monstre :
        // si on se situe dans la salle du boss et que la bombe se
        // se situe sur une 'faille' , dégât au boss
        // sinon, vérification qu'il y est des monstres au niveau
        // de la bombe et explosion
        // de même pour le joueur
        // vérification qu'il y est un élément 'destructible' au
        // niveau de la bombe puis si c'est le cas, changement du type
```

```
      // de décors en 'détruit'
}
```

L'arc

```
void utilisationItem( MainCharacter cha) {
      // lancement d'une flèche avec déplacementInitial()
}
void utilisationNonJoueur(String direction , float x ,float  y){
      // de même pour les monstres :
      déplacementInitialNonJoueur(direction , x ,y);
}
```

Les flèches

```
Flèches(String direction, int x, int y , Texture texture){
      // le constructeur sera appelé pour chaque nouvelle flèche
      // chaque flèche aura une position de départ et une direction
}
void déplacementInitial(String direction , float x ,float  y){
      // une flèche sera tiré si : il le joueur a plus d'une flèche,
      // et avec une limite d'une toutes les 0.4 sec
      // la nouvelle flèche remplira le tableau associé aux flèches
      // la direction de la flèche dépend de la direction du joueur
      // au moment où il utilise l'arc
      // ensuite on associe une vitesse à la flèche
      // si l'autre item est la bombe alors la flèche sera spéciale,
      // elle aura une bombe accroché au bout et fera des dégâts
      // comme une bombe
      // il y aura une flèche en moins dans l'inventaire du joueur
}
void déplacementInitialNonJoueur(String direction , float x ,float
y ){
      // de même mais la position sera celle de l'endroit où
      // la flèche est tiré ( monstre ou mur de la grotte en I1 )
}
void déplacement(MainCharacter Link ){
      // cette fonction permet de contrôler le déplacement de la
      // flèche ; si sa vitesse diminue ( elle a rencontré un
      // obstacle ) ou si elle sort de la carte, elle va déclencher
      // l'explosion si la bombe est associé ou alors va faire des
      // dégâts au monstre sur lequel elle a buté ( si c'est bien
      // un monstre )
      // le dégât peut aussi être sur le joueur
      //
      // petite particularité, si le joueur est dans la première
      // salle de la grotte I1, il a besoin des flèches pour
      // attraper le coeur de vie, donc ici le code vérifie que la
      // flèche est tiré au bon endroit
```

```
}
void explosionFlèche( ){
      // cette fonction va faire à peu près comme la fonction
      // explosion() de la bombe mais avec la position de la
      // flèche quand elle rencontre un obstacle
}
```

*** Nous allons présenter ici comment fonctionnent les objets qui sont lâchés au sol avec le cas des flèches, mais les essences et les coeurs de vie fonctionnent à peu près de la même manière. ***

```
void remplirDropFlèches(int xDrop, int yDrop){
      // cette fonction, appelé quand on tue un monstre et qu'on a
      // récupéré l'item arc , va avec une probabilité de 0.4 remplir
      // le tableau des 'flèches lâchées' et disant que cette flèche
      // est 'présente'
}
void déposerFlèches(int xDrop, int yDrop){
      // cette fonction va placer la flèche autour du monstre qui
      // l'a fait apparaître
}
void détectionFlèches(MainCharacter Link){
      // cette fonction est appelé à chaque instant et vérifie
      // qu'au niveau de la position du joueur ne se trouverait
      // pas une flèche, en parcourant le tableau
      // si tel est le cas, le joueur augmente son nombre de flèche
      // de 3
}
void détectionFlèchesEpée(MainCharacter Link){
      // même chose mais avec l'épée ( la direction du joueur quand
      // il utilise l'épée )
}
void représentationFlèchesDrop(GameMain game){
      // représentation des flèches au sol
      // au bout d'un certain temps, ces objets se mettent à
      // disparaître, à clignoter, si le joueur ne les a
      // pas récupérer, ils sont perdus
}
void réinitialisationDrop(){
      // réinitialise le tableau de flèches lâchées, quand on sort
      // de la carte par exemple
}
```

Les objets :

Essences

```
void détectionEssence(MainCharacter Link){
      // en fonction du type de l'essence, va rajouter 1,5,10 ou 25
      // essences au joueur
}
```

Coeurs de vie

```
void représentationNombreCoeur(GameMain game, MainCharacter Link) {
      // représentation de la vie en haut à gauche de l'écran
      // fonction développer en prenant en compte une vie maximale
      // largement plus grande que ce que le joueur peut avoir
      // dans le jeu
      // ce pourquoi il y a un code pour afficher deux lignes
      // de vie au bout d'un certain nombre de vie maximale
}
```

Réceptacle de coeur (dans la classe CoeurDeVie)

```
void receptacleDeCoeur(){
      // fonction appelé lorsqu'un réceptacle est détecté
      // d'une part, la carte où le réceptacle a été trouvé s'en
      // verra modifié pour noter que le réceptacle a été trouvé
      // et le nombre de réceptacle trouvé par le joueur sera
      // augmenter, s'il en a 4, il obtiendra 4 points de vie
      // supplémentaire sur sa vie maximale
}
```

Les menus :

Chaque menu a une fonction représentation et une fonction update pour savoir ce dont le joueur a envie. Ici, ne seront présentées que celles du sac puisqu'elles sont les plus complexes. Par ailleurs, le menu sac possède d'autres fonctions à expliciter.

```
void updateSac(float dt){
      // en fonction de la position du curseur et des touches sur
      // lesquelles appui le joueur, celui ci va se déplacer
      // ( dans cette fonction, ceci est représenté par l'entier
      // itemSelect mais dans les autres c'est tout simplement
      // 'choix' )
      // puis si le joueur appui sur 'K' ou 'L', ceci appelle
```

```
      // acquisitionItemsK()ou son homologue
      // si le joueur appui sur 'M', on sort du menu
}
void affichéSac(GameMain game){
      // affichage de tout l'interface du menu, et 'animation'
      // lorsque l'on se situe sur une case en particulier
      // affichage des items que l'on possède et aussi de certains
      // objets, comme la clé du donjon, la clé du boss,
      // les réceptacles que l'on possède, etc.
}
```

Autres fonctions de menu sac : le menu sac a deux sortes de tableau, un pour les items du sac et un pour les items affichés en jeu que le joueur peut utiliser directement

```
void acquisitionItemsK(){
      // en fonction de la position du curseur, échange de l'item K
      // avec l'item situé au niveau du curseur
} // de même pour son homologue
void affichageItemK(GameMain game){
      // affichage de l'item K en haut à droite de l'écran en jeu
      // ainsi que le nombre de ces items que le joueur possède
} // de même pour son homologue
void setItem(Item ite){
      // si le joueur détecte un nouvel item, celui ci est ajouté
      // en premier lieu dans le tableau des items du jeu, s'il est
      // rempli, dans le tableau du sac
}
```

La sauvegarde :

Les classes AcceptClass et SendClass ont des fonctions qui ont été recopiées sur internet pour la serialization.

Pour ajouter un élément dans la classe sauvegarde il faut :

- le redéfinir : son type et son nom ;
- le placer dans le constructeur et lui associer la variable a qui il fait référence ;
- l'ajouter dans la fonction *chargerSauvegarde()*, la variable du jeu prendra la valeur de la variable de la sauvegarde ;
- ajouter la variable du jeu dans *créerSauvegarde()* avec sa valeur initiale (au début du jeu) ;

La fonction *créerSauvegarde()* redéfinit chacune des variables sauvegarder en leur valeur initiale puis sauvegarde.

La fonction *chargerSauvegarde()* donne la valeur présente dans la sauvegarde aux variables du jeu.

Le constructeur est ce qui va être envoyé dans l'archive.

V. Bilan

Nous nous sommes rendu compte que la conception d'un tel jeu requiert une quantité considérable de temps et de travail. Ceci nous a conduit inexorablement à reconsidérer certaines idées de base que nous avions eu en les modifiant ou en les abandonnant.

Ainsi, nous avons dû en abandonner un certain nombre : par exemple, nous avions imaginé au début la création de trois zones au total mais par manque de temps, nous avons pu n'en faire qu'une seule. Nous avons aussi choisi de renoncer à inclure certaines animations supplémentaires telles que le changement de couleur des personnages pour signifier un contact avec d'autres personnages.

En outre, parce que nous voulions avoir un produit complet, nous avons des morceaux de codes redondants ou non optimisés. Aussi la fonction de détection de trou est-elle pratiquement la même que la fonction détection de l'eau profonde : nous aurions dû les confondre. De plus, les menus que nous avons créé auraient pu hériter de fonction de la classe Menu au lieu de les redéfinir à chaque fois.

Par ailleurs, nous pouvons également relater quelques erreurs de codage. Il subsiste notamment un léger défaut au niveau du codage des flèches avec les bombes : lorsqu'une flèche-bombe explose, elles explosent toutes.

Nous n'avons pas fini la carte en A3, A4, A5, A6, B5, B6, comme indiqué précédemment pour des raisons techniques et matérielles : l'ordinateur de Guillaume a cessé de fonctionner quelques jours avant le jour du livrable 3. Puisque cela nécessite un certain temps, nous n'avons pu le refaire.

Finalement, le temps constitue la contrainte majeure à notre projet dans la mesure où il nous a été nécessaire de revoir nos ambitions à la baisse : il nous restait un grand nombre d'idées de conception que nous n'avons pu exploiter davantage; mais nous sommes tout de même très fiers du rendu final et estimons le jeu complet compte tenu de la diversité de ce que l'on peut trouver dedans.

VI. Manuel Utilisateur

Au démarrage

Un menu démarrer s'affiche, appuyez sur entrée.

Ici vous aurez le choix entre charger une partie, créer une nouvelle partie ou accéder aux options qui ne sont ni plus ni moins qu'un condensé des commandes importantes du manuel d'utilisateur.

Se déplacer dans les menus en général :

Monter : touche 'Z' ;

Descendre : touche 'S' ;

Aller à gauche : touche 'Q' ;

Aller à droite : touche 'D' ;

Sélectionner : touche 'Entrée' ;

En jeu

Accéder au menu pause :

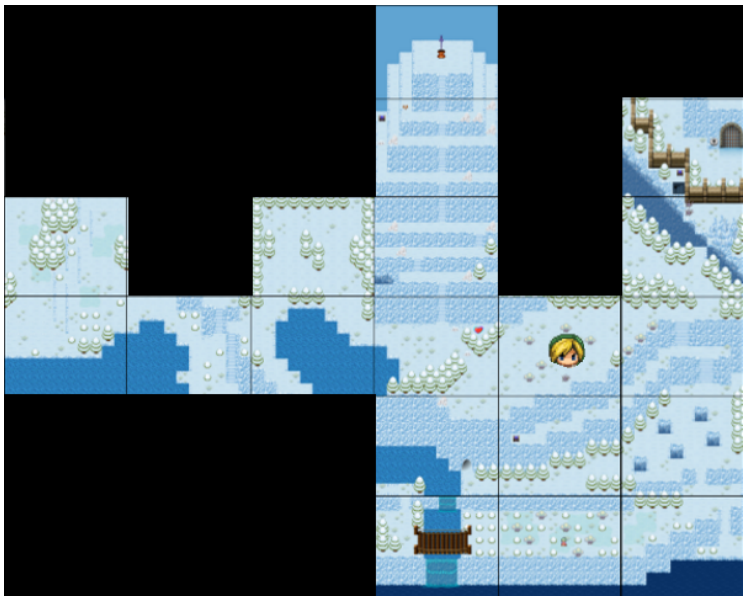
Touche 'O' ;

Sauvegarder :

Aller dans le menu pause ('O') et appuyez sur entrée en se plaçant sur sauvegarder.

Afficher la carte totale :

Touche 'P' ;



Accéder et sortir du sac :

Touche ‘M’ ;

Dans le menu sac :

Sélectionner un item : item de gauche : touche ‘K’, item de droite : touche ‘L’ ;

Se déplacer :

En avant : touche ‘Z’ ;

En arrière : touche ‘S’ ;

A gauche : touche ‘Q’ ;

A droite : touche ‘D’ ;

Utiliser un item :

Item de gauche : touche ‘K’ ;

Item de droite : touche ‘L’ ;

Si l’item sélectionné est le bouclier, il faut rester appuyé.

Si l’item sélectionné est l’épée, il y a une possibilité de faire une attaque circulaire qui produit plus de dégâts en restant appuyé suffisamment longtemps sur la touche.

Lorsqu’il y a du texte :

Si ce n’est pas spécifié, touche ‘entrée’ ;

Ecran de jeu :

1 : vie du joueur

2 : essences : monnaie du jeu

3 : item K

4 : item L

Fin de partie :

Si votre nombre de points de vie descend à 0, vous devrez recommencer la partie à partir de votre dernière sauvegarde.