Dossier du Projet

GROUPE : Omar Foudane, William Lin, Dorian Cassagne

1:Documents pour CPOO

Architecture (5/60 points)

Nos choix principale d'architecture :
La gameloop doit être responsable de faire jouer un tour au personnage. Pendant un tour de jeu, la map collecte les nouveaux personnages ajoutés à la map. Pour éviter les conflits de lecture et écriture sur la HashMap, cette dernière contient les personnages sur la map . Être une HashMap nous permet vraiment d'avoir un identifiant par entité ajouté à la map, cet identifiant est un entier partagé entre le contrôleur et le model. Vu que la HashMap est consulté par la gameloop qui elle est un Thread, cela nous oblige à faire plusieurs listes, tel que la liste des nouveaux personnages à ajouter ou la liste de personnages à retirer. Quand la gameLoop va donc dérouler dans cet ordre : récupérer les nouveaux personnages à ajouter, puis les personnages à retirer et finalement collecter l'ensemble des actions produites. La gameLoop ne s'intéresse qu'aux mouvements. Pour synchroniser les mouvements entre le modèle et la vue, on a décidé de faire en sorte que la vue soit aussi appelée chaque tour pour qu'elle effectue un tour qui est un mouvement d'un nombre k de pixels

Détails : diagrammes de classe (5/60points)

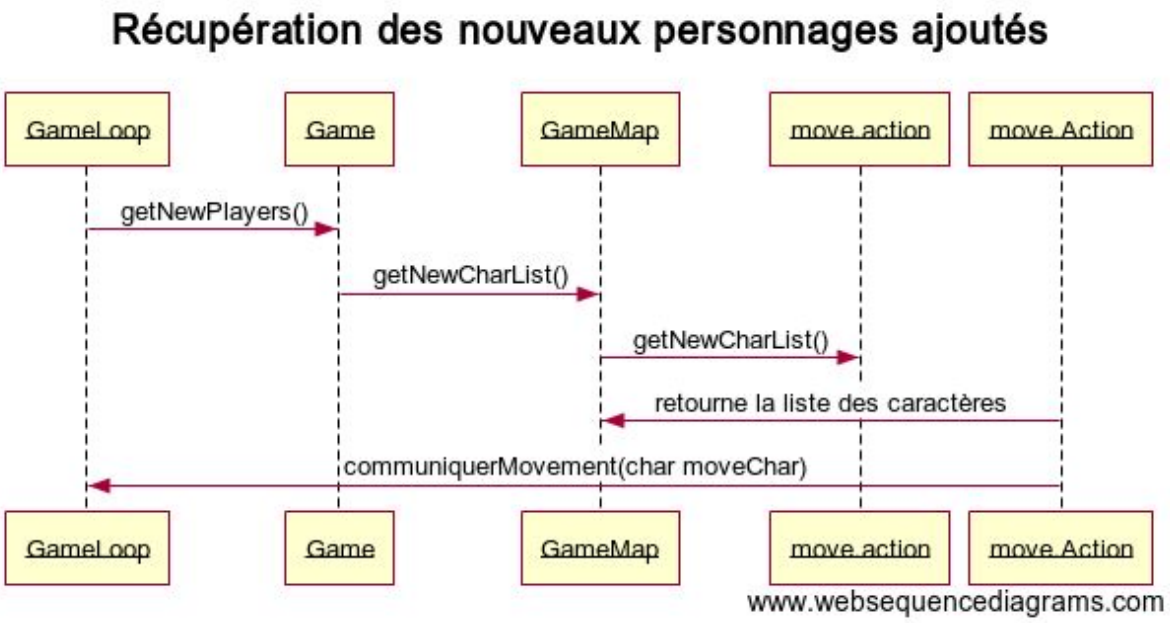
Diagramme complet : Vous le trouverez sur git

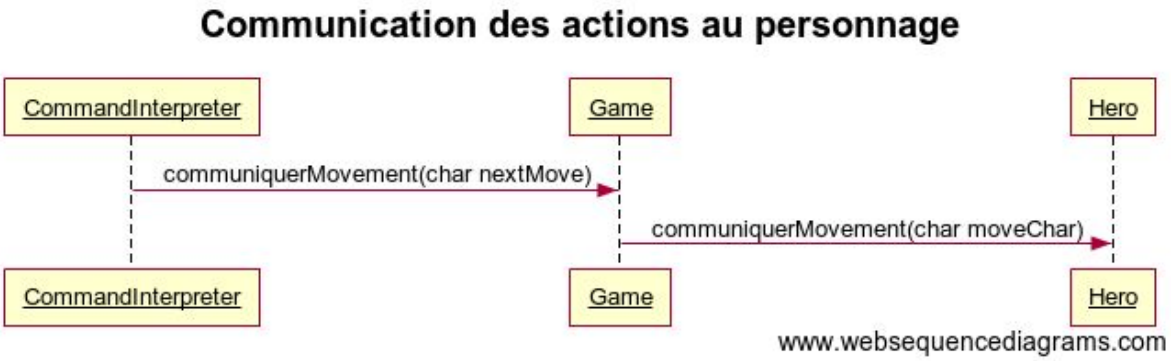
La classe principale de notre jeu, c'est movable et ses sous classes, en effet movable représente tout objet pouvant se déplacer (changer de map), ce qui implique qu'ils auront un cycle pendant lequel ils peuvent établir une action, il existe deux sous classes qui étendent cette classe et qui sont aussi abstracts: Attaque et gameCharacter, la différence entre les deux est que l'une occupe une case et interdit d'y accéder (GameCharacter) et qu'attaque ne bloque pas la case mais y est stocké pour être infligé à ceux qui vont y accéder.
Les attaques sont eux mêmes divisés en attaques statiques et attaques dynamiques, les attaques dynamiques sont ceux qui dépendra d'un item donnée, ils sont spécials à au héro, tant dis que normal attaque ne dépend pas d'items et peut avoir des comportements différents autre que d'infliger des dégâts au héro.

Concernant GameCharacter, il existent des ennemis autres que le personnage principale qui sont équipés du bfs. Les ennemis normaux sont différents du boss vu que le boss est composé de deux entités et donc nécessite une gestion différente des attributs. Un integerProperty partagé entre les deux entités. De plus, pour que le boss restera dans son

état lorsqu'on change de map, il sont mise en statique au lieu d'être des variables d'instances. Le boss n'est tué qu'une seule fois. Le héro quant à lieu à un autre système pour la gestion des propriétés (HP, Mp …) ce qui exige l'isoler des autres entités (ennemie). Dans l'exemple, Simple Arrow est un exemple d'attaque spéciale, mais elle est toujours une attaque.

Diagrammes de séquence (5/60 points)





2 :Documents pour IHM

Maquettes (5 points/30)

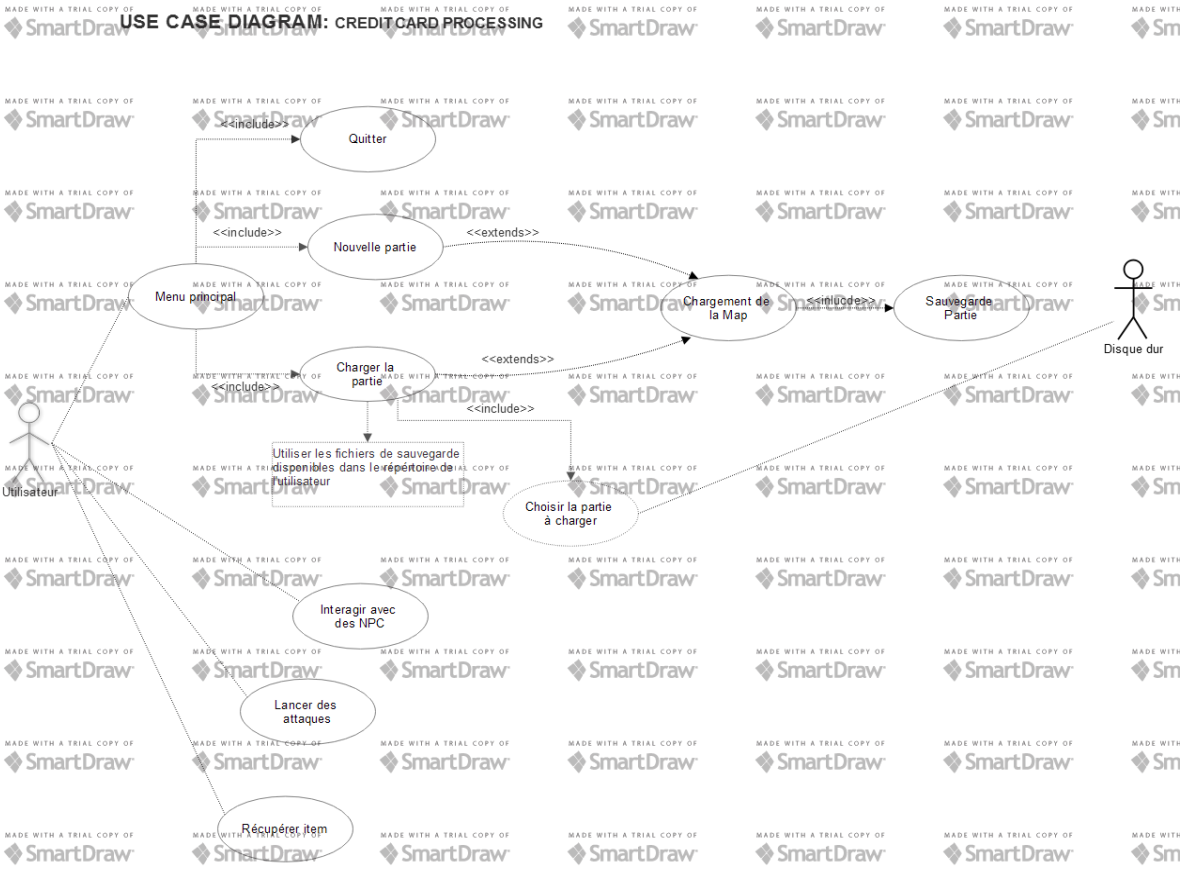
Disponibles sur Git : En raison de gros images

3:Documents pour gestion de projet

Cas d'utilisation (3 points/20)

Acteur	*Action*
1-Choix nouvelle partie	*2-Chargement de la Map et début de jeu*
3-Choix de sauvegarde	*4-Sauvegarde réussi*

Acteur	*Action*
1-Choix charger Partie	*2-Recherche disque dûr pour les parties sauvegardés* *3-Affichage des partie sauvegardé*
4-Choix de chargement de partie	*5-1->Si la partie est éxistante alors on la charge* *5-2->Chargement échouée*
6-Retour au menu principale	*6-1->Si le menu est lancé depuis une nouvelle partie alors retour à la partie* *6-2->Si le menu est lancé par la page d'accueil, retour au menu d'accueil*
7-Quitter	*8-Fermer le jeu*



Cahier d'initialisation (2 points/20)

Scénario

Nous sommes à la 42ème réunion des personnages fictifs, comme à son habitude notre Link d'Hyrale se pavane en montrant sa grosse mastersword a Sanus et Mariu tout en racontant comment il avait réalisé son dernier sauvetage de la princesse Zolda prise par Ganonderp.
Lorsque tout à coup, un étrange personnage apparaît et prend la parole :
"Mes frères et soeurs réjouissez vous!

vous qui en avait assez de vous faire taper dessus par des héros cherchant juste a loot, j'ai entendu votre appel, votre détresse , aujourd'hui je vous sauve"
L'homme mystérieux fait disparaître tous les personnages du congrès,
Les helpers confus ne comprennent pas ce qu'il se passe.
"Je suis Domino votre Sauveur! Joignez vous à moi et prenez la place de ceux que vous appeliez jadis *héros*"
(Les mobs hostiles et Certains helpers le rejoignent…)
"Je vois que tout le monde ne partage pas notre point de vue... Disparaissez!"
Il claque des doigts et tous les dissidents explosent/disparaissent…
"Mes compagnons! Nous partons pour Hyrale 1, mère de toutes créations, nous allons retournez à la source et restaurez votre place légitime dans votre univers"
Il claque à nouveaux les doigts et disparaît avec les traîtres.
Après la pagaille un étrange calme s'est installé, Novi se relève et se met à rechercher de potentiel survivant, elle est seule… Elle ne peut pas laisser Dormino finir son entreprise!
Une lueur est toujours présente là où Domino s'est évanouie, en voulant l'inspecter Novi est aspiré et transporté vers Hyrale 1 elle entends les voix des héros "Sauves nous!" "Je ne passerais plus tes dialogues".

Novi se retrouve en pleine forêt Emeraude, Dormimo ne doit pas etre très loin elle entend des bruits de lutte, en face d'elle se trouve deux monstres hostile mais Novi n'a aucun pouvoir….
Elle cherche autour d'elle quelque chose qui pourrait l'aider a passer les mobs, elle trouve une arme.
Ce légume lui confère le pouvoir de porter des armes. Novi détruit les monstres.

Elle aperçoit une autre brèche dimensionnelle qu'elle emprunte, elle se voit transporté dans un univers celui de Sanus, dans un vaisseau
Elle rencontre des monstres et des portes, une clé ouvre une porte et elle trouve aussi un objet dont elle ne connaît pas l'utilisation encore (permet de pousser)
elle se bat contre un mini boss .
Elle arrive dans une autre dimension il y a un labyrinthe à résoudre,il faut qu'elle pousse des blocs de rochers et elle affronte un autre mini boss

mécanismes

- Flêches directionnelles : Pour attaquer vers une direction
- Z : Déplacement vers le haut
- Q: Déplacement vers la droite
- S: Déplacement vers le bas

- D: Déplacement à droite
- E: Changer d'attaque (basculer vers une attaque spéciale …)
- ESCAPE: Afficher le menu de jeu
- CTRL+ESPACE : Cacher un message, le choix de le rendre plus compliqué est pour s'assurer que le joueur lis le message, vu qu'il contient des informations sur la mission
- X: Parler à un NPC

items

- bottes pour se deplacer plus vite (augmenter c/s)
- armes : épée attaque la case devant perso
- lance attaque jusqu'à 3 cases devant perso
- arc, tire devant le perso, flèche disparaît en selon la distance (upgrade augmente la portée)
- hache attaque 3 cases devant perso mais en ligne]
- coeur de régénération: récupération des PV
- bombe : détruit certains décors possibles
- bouclier : qui réduit les dégâts

Ennemis :

- Boss : NyaNyaNay : se déplace horizontalement, lance un arc en ciel si le héro est sur la ligne horizontale, sinon lance des attaques qui varient et qui changent de façon aléatoire.
- BadMonkey : Ennemie normale qui attaque le héro dès qu'il est à sa portée
- IntelligentTower/Tower : Deux tourelles, la première attaque de façon intelligente vers la direction du héro, tandis que l'autre attaque vers toutes les directions d'une façon aléatoire. Son attaque est TowerAttack et peux ralentir le héro ou toute autre monstre qu'elle touche.
- StaticBadMonkey : Un monstre stable qui ne bouge pas, n'attaque pas et qui est comme un NPC auquel on ne pourra pas parler mais on pourra tuer.

Cahier de spécifications fonctionnelles (8 points/20)

Cahier de spécification fonctionnelles

Le héro :

- Le héro doit avoir des propriétés : ces propriétés seront stockés dans une instance de la classe HeroStats . Cette classe nous permettra plus d'encapsulation. Elle ne sera capable de contrôler le mouvement du héro mais contrôle juste ses propriétés. Ses propriétés ne peuvent pas être communiqué avec l'extérieur, mais une copie en lecture seule pourra être. Cette copie sera une instance de CopyOfHeroStats. Elle sera utilisé pour lire les propriétés par la vue essentiellement (barre de vie et barre de mana, item …). Les propriétés ne peuvent être changer, par son instance en GameHero qui ne partagera que la copie de ses propriétés avec l'extérieur.
- Les propriétés du héro sont :
    1. HP : Points de vie du héro, consommée chaque fois une attaque inflige des dégâts au héro. C'est un attribut obligatoire, vu qu'ils détermine l'arrêt de la gameLoop, par une méthode : isAlive qui doit renvoyer vrai si le héro est vivant, cela implique que ses PV sont supérieur à 0.
       Ils sont visualisé avec une progressBar .
    2. MP : Points de magie du héro. Ils sont consommée par chaque attaque, Les attaques normaux, vu qu'ils ne consomment pas de MP auront une consommation de 0. (voir section attaque)
    3. maxHP: Points de vie maximale du héro, Il est détérminé par l'item utilisé en tant qu'armure de PV.
    4. maxMP : Points de magie maximale du héro, comme maxHP il est détérminé par l'item de MP.
    5. Le héro a aussi l'id (d'image) du dernier item colllecté, soit du defense ou d'attaque.
    6. Le héro a aussi la liste d'attaque qui peuvent être lancé. La première occurence dans cette liste doit être l'attaque par défaut qui consomme 0 MP, et qui dépends de l'item collecté. En étant première dans la liste, il est facilement accessible (vu qu'on est sûr qu'il y'a au moins l'attaque par défaut).

Les attaques :

Toute attaque est lancée sur une map, l'attaque va donc jouer son premier tour, au lancement sans attendre que ca soit son tour. Un tour d'attaque suit les étapes suivantes :

Méthode d’action

1. L’attaque se lance sur la case où elle est
2. L’attaque supprime toutes ses occurrences dans les cases déjà visités (Pendant le tour précédent).
3. L’attaque bouge d’un nombre donnée de case et s’enregistre à chaque case.

Ces étapes sont obligatoires pour synchroniser le mouvement au niveau model et mouvement au niveau vue. Un petit exemple est que : Si le héro joue son tour avant l’attaque et que l’étape 1 n’est pas vérifiée, alors au niveau model le personnage ne sera pas touché par l’attaque. Au niveau visuel l’attaque passera sur le personnage, ce qui est un bug. Le même problème se posera si l’attaque ne s’enregistre pas à la case .

Attributs

- cellPerTurn : Une attaque peut avancer par k cases, cet attribut définit le nombre de fois l’attaque établira son action .
- maxDistance : On est sûr que toutes les attaques ont une distance maximale. Il n’existera pas d’attaque sans arrêt.(maxDistance) .
- lastTurn : le dernier tour de l’attaque, pour éviter que l’attaque disparaissent au moment de déplacement, supposé instantané au ni

Une attaque peut avancer de plusieurs cases par tour (Attribut prévu dans la classe Attack)

Selon le type d’attaque, elle a un traitement différent en fonction du type de case rencontré, traversable, avec personnage (handleMove()) …

Chaque attaque a un effet relative à l’attaque (handlePlay()).
Il n’est pas nécessaire qu’une attaque soit enlevée de la map juste car elle a touché un personnage.

Une attaque ne différencie pas un ennemi et un héro, tous personnage touché par l’attaque recevra des dégâts

Les ennemis:

Les ennemis sont tout comme le héro capable de:

- D'effectuer un tour de jeu en revoyant ou non un Move

- De se deplacer sur la map
- De lancer des attaques
- De se faire attaquer
- De disparaitre

Personnages

Il existe plusieurs types de personnages :

1. Boss : NyaNyaNay : se déplace horizontalement, lance un arc en ciel si le héro est sur la ligne horizontale, sinon lance des attaques qui varient et qui changent de façon aléatoire.
2. BadMonkey : Ennemie normale qui attaque le héro dès qu'il est à sa portée
3. Bomber : La bombe, on pourra pas marcher sur la case où elle est. Elle est généralement posée par le héro
4. IntelligentTower/Tower : Deux tourelles, la première attaque de façon intelligente vers la direction du héro, tandis que l'autre attaque vers toutes les directions d'une façon aléatoire. Son attaque est TowerAttack et peux ralentir le héro ou toute autre monstre qu'elle touche.
5. StaticBadMonkey : Un monstre stable qui ne bouge pas, n'attaque pas et qui est comme un NPC auquel on ne pourra pas parler mais on pourra tuer.
6. Les NPC : Sont des personnages qui occupent une case et nous interdisent d'y accéder. Ses NPC doivent recevoir un message en entrée, vu que finalement y'aura pas le même message pour tous les NPC.

Chaque personnage dynamique : ennemie ou héro a un nombre de cycle qui devra attendre avant d'effectuer sa prochaine action, La superclasse movable sert à cela. Elle est responsable de déterminer le tour du personnage auquel il pourra établir son action, si le personnage n'a pas atteint son cycle, son action sera null. Le choix d'implémenter cette fonctionnalité dans la plus haute classe du jeu (Movable) est pour qu'on aie plus de contrôle sur cette classe. En effet les sous classe n'auront pas droit de contrôler leur tour de jeu, c'est à dire qu'ils ne peuvent pas jouer sans que ça soit leur tour, mais ils peuvent toujours réduire leurs tour avec un setWait(), mais cela ne prendra effet que pendant un et un seul tour de jeu réussi.
Au moment de la création du personnage, il s'ajoute automatiquement à la map où il démarre, vu que ce n'est pas logique de créer un personnage sans qu'il soit utilisé.

Pour rendre le mouvement d'un personnage plus au moins fluide, on a décidé d'avoir un coefficient pour chaque personnage. Cela nous permettra par exemple de montrer que notre jeu se fait vraiment case par case (le coefficient est mise à 0.3) quand on est ralentie et permet au personnage de glisser sur la map (niveau visuel).

Les items:

Les item sont posé sur une case mais n'en empêche pas l'accès. Ils sont capable de recevoir un héro et de le modifier en fonctions de leurs spécifications. (exemple augmenter la vie max, augmenter la vitesse du hero…). Les items sont divisés en plusieurs catégorie :

1. AttackItem : Des items qui changent l'attaque du héro ce qui implique une réduction ou augmentation de ces PV.
2. DefenseItem : Des items qui augmentent les points de défense du héro
3. HPPotion : Des items qui augmentent les points de vie du héro sans qu'ils augmentent la seuil maximale. Si la seuile maximale est atteinte, les points de vie seront inchangés
4. MapChange : Tous les items qui permettent de changer de Map
5. MPPotiion : Les items qui augmentent les points de magie du héro sans augmenter sa seuil maximale. Même cas que le HPPotion si on atteint la seuil maximale .
6. MPItem : Items qui augmentent la seuil maximale des points de magie sans qu'ils augmentent ses MagicPoint. Y'a que la seuil qui augmentent et non pas les points MP.
7. PVItem : Des items qui permettent d'augmenter la seuil maximale des points de vie du héro
8. SpeedItem : Des items qui augmentent la vitesse du héro pendant un certain temps.

Les maps :

Il existe 4 maps dans le jeu, chaque map se termine par une action donnée déclarée dans le scénario dédié à la map, parmi les map on cite :

1. Salle de théâtre, c'est la map initiale. Le joueur est exposé à un scénario qui va l'introduire au jeu. Dans ce scénario : Le héro doit atteindre une case donnée, à côté d'un NPC, en parlant avec ce dernier, Il lui indiquera sa prochaine destination (Monter sur scène).
2. Forêt : Dans cette forêt le héro doit premièrement observer deux monstres, sans que ces monstres puissent l'attaquer. Pour réussir cela, il faudra faire en sorte que la portée du monstre soit égale à 1. La portée ne doit être 0 car le personnage ne pourra pas être à côté des ennemis sans qu'ils puissent

l'attaquer (ce n'est pas le but de jeu). Il n'existe qu'un chemin ouvert qui emmène vers deux items. Quand l'un des deux sera collecté, le boss va apparaître en haut de la map, et le héro devra l'attaquer.

3. Le dungeon, la map du boss où y'aura pas d'énigmes ni de collecte d'item. Dès que le boss meurt le héro changera à nouveau de map vers la map de départ, mais avec un scénario différent.

Une map pourra à tout moment subir un changement qui pourra être :

-Un changement d'une case donnée : cela sera notifié à la vue par un integerProperty qui contiendra l'identifiant de la case. Un listener est ajouté à cette propriété. Si la même case a été changé deux fois successive, alors l'IntegerProperty est mise à -1, ce qui permettra à la vue de récupérer le changement de l'ancienne case (oldValue).

-Un changement de map signifiera un rechargement de la map au niveau de la vue, cela va exiger de supprimer les personnages courants sur l'anchorpane et de clear la tileMap.

-Vu que tous personnage est ajouté à la map, il est donc obligatoire d'avoir une liste de personnage par map. Pour éviter le problème de lecture et écriture en même temps sur une liste qui est dû au fait que la gameloop est un thread qui tourne en parallèle, on a décidé de séparer les listes, d'avoir une liste des éléments qui sont ajoutés et une liste des éléments qui seront ajouté au prochain tour de jeu. Cela nous évitera tout conflit de synchronisation. La liste des nouveaux entités à ajouter est vider après avoir ajouter ses entités à la liste principale.

-La liste principale des joueurs définit les entités qui ont un tour et qui vont jouer leurs tour à la prochaine itération de la gameloop. Donc pour que le scénario fasse disparaître un monstre sans qu'il le tue, il suffira de demander sa suppression dans la liste principale. En cas de réapparition, il faudra juste le rajouter à la liste principale

Fonctionnalité centrale : Les mouvements

Il existe une classe de mouvements qui nous permettra de définir les directions possibles. Tout déplacement ne devra avoir qu'une seule direction, c'est à dire que pendant le même déplacement (en même tour), on pourra pas se déplacer

verticalement puis horizontalement, mais si on veut avoir un tel mouvement, il va falloir effectuer un mouvement en diagonale.

La classe Movement définit les mouvements basique vers 9 directions possibles : Haut, bas, à gauche, à droite, en haut à droite, en haut à gauche, en bas à droite et en bas à gauche et le stay qui signifie ne rien faire.
En utilisant ses mouvements, aucun mouvement de case par case n'est impossible. Il est toujours possible d'avoir un mouvement quelconque. Pour un mouvement de 5 cases horizontalement, il faudra effectuer le mouvement 5 fois de suite.

La classe Movement est aussi capable de nous indiquer la bonne direction vers une zone donnée, elle recevra bien sûr la case de départ, cela est très util pour une tourelle intelligente par exemple. Pour ne pas juste limiter le choix de cette fonctionnalité à la tourelle et pour laisser la possibilités aux autres objets de l'utiliser, on a décidé que c'est mouvement qui doit s'en occuper.

GameLoop

Chaque tour de jeu est lancée par une gameloop, la gameloop tournera en continue tant que le héro n'est pas mort. La gameLoop est un élément centrale dans le jeu, C'est elle qui exécute le tour de chaque personnage. Pendant chaque tour, la gameloop communique la liste des actions effectués à la vue. Cette liste est récupérée par la GameMap en utilisant sa liste principale pour faire jouer un tour à chaque personnage du jeu. Ce actions sera nommée Move et ne contiendra que deux paramètres essentiels : la case d'arrivée, l'identifiant du personnage et la vitesse de déplacements. Cet identifiant du personnage est partagé entre la liste principale de GameMap et la liste des éléments visuels de la gameLoop.

La gameloop est aussi responsable de l'ajout des nouvelles entités en créant l'objet visuel qui leurs correspondent. Elle est aussi responsable de la suppression des éléments retirés de la map.

La gameloop exécute aussi un tour aux éléments visuels.

Scénario

Le scénario est un objet très important, il est obligatoire pour le bon déroulement du jeu. Pour éviter d'écrire trop de condition et de cadrer tous les cas possibles dans une classe, notre scénario sera écrit sous forme d'un texte qui sera convertit en fonctions.
Par définition un événement de scénario est composé de :

- Des conditions : Qui peuvent être liée par un lien logique
- Des actions : Qui sont successives et s'exécutent l'une après l'autre

Conditions :

Les conditions peuvent être relative à :

La case : ->Si la case comporte un item
->Si la case est traversable

Le héro : ->Si le héro est sur une case : On pourra pas contrôler qui est sur une case directement, On sait que c'est un GameCharacter mais ça nous indique pas le sous type (type dynamique). Tandis que le héro sait très bien sa position et peut être récupérée à n'importe quel moment.

Ennemie:->Si l'ennemi est au dessous d'un nombre k de PV. Avec cette condition on pourra contrôler la mort de l'ennemie en mettant que k égal à 0.

Ancienne Condition:->Il existe aussi une condition relative à une autre précédente, les conditions exécutée avec succès seront enregistré dans une liste des événements réussis et seront supprimés de la liste des événements à faire.

Gestion des Conditions :

Une condition pourra être vrai ou faux, cela est passée comme étant le 3ème caractère de la chaîne, les différents conditions peuvent être séparés par des liens logiques : OR, AND. et la négation se fait par !. Deux petits exemples :

Si le héro est sur la case 840, alors on écrira : H:840
Si un monstre est mort : M:idMonstre: :0
Si un monstre est mort et que la case 840 ne contient pas l'item : C:840:
:I-AND-M:idMonstre: :0
Si la case 840 ne contient pas d'item : C:840:!:I

Actions :

Il existe plusieurs type d'actions qui se résument en :

- Création : créé un item, ennemie ou NPC
- Suppression : Supprimer un monstre de la map, rendre une case traversable en supprimant l'obstacle, enlever le NPC situé sur une case donnée.
- Affichage : afficher un message à l'utilisateur
- Ajout : ajouter une attaque au héro, ou lui ajouter un item
- Changer de Map .

Gestion des attaques :

Si l'attaque est une création, alors l'élément créé sera enregistré dans la liste qui contient les éléments du même type, cette liste sera une HashMap qui sera identifié par l'identifiant du monstre, si on crée un monstre ou par l'identifiant de la case si on crée un NPC ou un item. Ces listes avec la liste des événements réussis seront utilisées pour la sauvegarde de la partie.

Au drop, les éléments sont supprimées pour éviter de recréer un monstre à chaque chargement de partie, même après que le joueur l'a tué.

Sauvegarde

La sauvegarde se fera en deux niveau :

1. La sauvegarde des propriété actuelle du héro, l'item courant et ses points d'attaque, mp, pv …
2. La sauvegarde des éléments créé par le scénario : afin de revenir exactement au même état où on était lors de la sauvegarde.

Les sauvegardes doivent être séparés, vu que les événements du scénario doivent être stockés et lû de la même façon qu'un scénario normal. D'ailleurs une sauvegarde de scénario comporte un petit scénario représentant les éléments à créer, ceux qui sont restés sur la map lors de la sauvegarde.

Les fichiers de sauvegarde sont enregistrés dans le répertoire home de l'utilisateur vu qu'il est souvent accessibles en écriture.

Vue :

Au niveau visuel, un personnage est représenté par un MovableView, qui représente un movable quelconque (le héro inclus). Cette classe pourra se déplacer d'une case à une autre en changeant les coordonnées du personnage sur l'anchorPane qui sera superposé au tilePane.

Le mouvement d'un personnage ne se fait par une animation, mais c'est la gameloop qui est responsable de ce mouvement. C'est à dire que le MovableView reçoit un mouvement, le répartie dans le temps par un avancement de x pixels par tour, cet avancement est définie par le nombre de tour du mouvement (dans Move). Cela nous évitera d'avoir des animations qui tournent en parallèle et qui peuvent ne pas être synchronisés.

La différence entre un MovableView et un HeroView est que l'on contrôle pas le scroll de la map en étant un movable normal. Seule le héro est capable de scroller la map. De plus on est obligé d'afficher les propriétés du héro, ce qui fait en sorte que l'on doit séparés les deux classes. La façon de reconnaître si c'est un héro ou un movable normale, c'est que le héro est le premier à être ajouté sur une map, tandis que les autres movables viennent après

Affichage des Messages :

Les messages sont affichés à l'aide d'un MessageView. Le jeu comporte un messageProperty qui représente une propriété contenant les messages à afficher. Le changement de cette propriété indique la récéption d'un nouveau message, ce qui implique l'arrêt de la gameloop et l'affichage du message. Pour afficher tous les messages demandés, chaque changement de cette propriété est récupérer, ils sont alors ajouté à une queue et afficher dans l'ordre de réception, du plus ancien au plus nouveau.

Menus De Jeu:

Notre jeu Zelda-Like comporte plusieurs menus :

- Menu d’Accueil :
  Il s’agit du premier menu que rencontre le joueur lors du lancement de l'exécutable.
  Ce Menu comporte plusieurs boutons avec leur propres fonctions:
    - Nouvelle Partie : Permet de créer une nouvelle partie, cette nouvelle partie initialise le jeu sur la map Salle de Théâtre.
    - Charger une Partie : Permet de charger le MenuChargerMap
    - Quitter : Permet de fermer l’exécutable

- Menu de ChargerMap :
  Il s’agit du menu affiché lorsque le joueur appuie sur un bouton “Charger une Partie”.
  L’utilisateur a la possibilité de sélectionner un nom de sauvegarde dans un tableau et de charger la partie correspondante via le bouton Charger la Partie.
  Il existe un bouton permettant à l’utilisateur de retourner vers le menu précédent.

- Menu de Pause :
  Il s’agit du menu affiché lorsque le joueur est en jeu et qu’il appuie sur le bouton ESCAPE(Echap). Ce menu est affiché pour signifier au joueur que le jeu est en pause cela stoppe donc la Gameloop momentanément.
  Ce menu comporte plusieurs boutons :
    - Continuer la Partie : Permet au joueur de reprendre la partie et donc à la Gameloop de reprendre son cours.
    - Sauvegarder la Partie : Permet de sauvegarder les données (liées a la map et ses occupants y compris le joueur) de la partie en cours.
    - Charger une Partie : Permet de charger le MenuChargerMap.
    - Quitter : Permet de fermer l’exécutable.

- GameOver Menu :
  Il s’agit du menu affiché lorsque le héro a été pourfendu, il comporte plusieurs boutons:
    - Rejouer : Propose au joueur de retenter sa chance, il n’amorce pas une nouvelle partie mais reprend le jeu au début de la dernière map active, en somme il permet de rejouer un niveau de map.
    - Menu Principal : Redirige le joueur vers MenuAccueil.
    - Quitter : Permet de fermer l’exécutable.
    - Chaque entité visuelle (Menu ou map) seront affichée en les mettant sur un StackPane ayant un controlleur GroundController qui permettra :

- Ajouter une autre vue sur le StackPane, ce qui permettra de superposer les éléments.
- Nous renvoyer vers l'élément précédent ce qui nous permettra de revenir sur l'état courante de la vue. Par exemple, en affichant le menu de pause, on ne doit pas perdre ce qui a déjà été réalisé en jeu. Donc on doit sauvegarder l'instance de AnchorPane contenant le tilePane (pour ne pas devoir la recrée si l'utilisateur décide de reprendre la partie)