The Survivor

Définition du fonctionnement global du jeu.

Table des matières

I) Début du programme : l'initialisation

1) La classe Survivor : début du programme

Le programme lance la classe Survivor qui ne fait que crée une instance de la classe MainWindow dans le package Graphics.

2) La classe MainWindow: gestion de la fenêtre

La classe MainWindow va crée les layout pour le placement des éléments, elle va ensuite crée les éléments :
-ile
-labelVie
-vie
-les quatre boutons de direction
-Le bouton option

La classe MainWindows va aussi crée les classes qui s'occupe des événements clavier GestionClavier et la classe qui s'occupe des événements de la fenêtre : Event

3) La classe principale : Ile

L'ile va faire plusieurs choses, premièrement elle va initialiser la mémoire pour les zones, le tableau d'occupation avec le nombre de cases passer en paramètre. Une fois les initialisations faites, elle fait appel a la méthode generateMap(true) , le paramètre de cette méthode définie si l'on recrée les personnages , ici il n'ont jamais été crée donc on lui envoi true.

Cette méthode va crée les zones et les placer sur le terrain de manière aléatoire
Le nombre de ZM est défini à l'initialisation puis modifié via le menu option plus tard.

Après avoir créée les zones la fonction va crée les créatures et les placer toujours de manière aléatoire sur une case du terrain.

Âpres si le paramètre de la fonction est vrai alors elle va crée un rimbo et le placer sur le terrain, sinon la fonction le placera juste aléatoirement sur une zone du terrain.

Une fois la création du monde terminé, un appel static à la classe VerifierCarte permettra de savoir si le terrain est valide (si on peut arriver la Zone d'extraction) sinon on régénérera la carte jusqu'à trouver une carte valide.

II) La création et fonctionnement de la classe Zone

Il y a trois types de Zone, toutes les Zones on un accès static a la classe Image pour avoir un accès a toutes les images du jeu.

Chaque Zone a une fonction resizeImage cette fonction lorsqu'elle est appeler va vérifier plusieurs choses afin de définir correctement l'affichage de la carte.

-Si la case est non découverte et quelle n'est pas dans le champ de vision de rimbo : on lui met une image noir redimensionner la la taille de la case et on arrête la fonction.

-Si la case est non découverte, mais dans le champ de vision de rimbo, on lui applique une image noir et s'il y a une créature sur la case, on l'affiche par-dessus l'image noir (fusion des images)

-Si la case est découverte, mais pas dans le champ de vision de rimbo, on affiche se qui doit apparaitre (en fonction de la case il y a des fusions) + l'image griser

-Si la case est découverte et dans le champ de vision de rimbo on l'affiche normalement

Liste des fusions :

ZP = imageHerbe
ZM = imageHerbe + imageForet
ZE = imageHerbe + imageExtration
Griser = Zone (ZP ou ZM ou ZE) + imageGriser
Non découverte = Zone (ZP ou ZM ou ZE) = imageNoir
Rimbo = Zone (ZP ou ZM ou ZE) + imageRimbo
Prédateur visible = Zone (ZP ou ZM ou ZE) + imagePredateur

À chaque fois que rimbo marche sur une case qu'il n'a jamais visiter, la case met le token exploré à 1 qui indique que la case est visitée.

Une autre fonction se chargera de placer les tokens d'image grisé correctement dans la fonction Ile.

III) La gestion de déplacement de rimbo et des prédateurs

Les déplacements de rimbo s'effectuent comme suit:

Lorsque la touche des flèches directionnelle a été presser ou que l'on a appuyé sur un des boutons de direction la fonction seDelacer de la classe Rimbo est appeler avec comme paramètre la direction que doit prendre rimbo et si les créatures joueront après.

Lorsque nous appelons cette fonction on récupère les limites de la carte (pour ne pas en sortir :))

Si on essaye de sortir, on considère qu'on a joué sans se déplacer.

On sauvegarde l'ancienne position de rimbo.

On incrémente la position de rimbo dans la direction voulue

On met le token exploré à 1 sur la case visiter

On appel la fonction resizeOneImg(coordonner de l'ancienne case de rimbo) qui va appeler la fonction de rezize image de la zone aux coordonner cela aura pour effect d'effacer l'image de rimbo de l'ancienne case

On appelle la fonction resizeOneImg(coordonné de la nouvelle case de rimbo) qui aura pour effet d'afficher rimbo sur la nouvelle case.

Si c'est une ZM ... on fait machine arrière, on revient sur l'ancienne case et on fait attaquer la Zone

Une fois le déplacement effectuer, on appel gestionVisibilite de la classe Ile qui parcourra toutes les cases de la carte pour définir quelle case seront griser ou pas et place leur token correctement.

Ensuite vérifie que la case ou l'on se trouve est pas la ZE, si c'est le cas on a gagner ! On affiche un message pour savoir si on veut rejouer

Si on n'a pas gagné on appel la fonction faireJouerPredateur de l'ile, cette fonction appellera faireJouerPredateur de la fonction IA cette fonction calculera en fonction du niveau de diffilcuter de l'ile les différentes stratégies des créatures.

La fonction va vérifier que Rimbo n'est pas sur la position d'un prédateur (avant de les faire jouer) ça voudra dire que rimbo attaque !

Donc on appelé la fonction attaquer de rimbo et définie le blocage de la créature concerner a 3 puis appel a la fonction fuir de rimbo (cette fonction prend un nombre aléatoire pour la direction jusqu'a trouver un endroit ou rimbo pourrait fuir puis la fonction réappelle la fonction seDelacer de rimbo mais sans faire jouer les créatures après.

Ensuite après les éventuels combats, on fait jouer les créatures

La fonction de déplacement des créatures marche globalement pareil que celle de rimbo a la différence qu'eux peuvent marcher sur les ZM et vérifie si la créature est bloquée ou pas.

Si après le déplacement, une créature est sur rimbo, alors c'est la créature qui attaque !

On appel attaquer de la créature et comme tout à l'heure on fait fuir rimbo.

Voilà, le déplacement est terminé, jusqu'à l'appui d'une prochaine touche.

IV) La gestion du redimensionnement

Pour pouvoir gérer les redimensionnements, je fais appel à deux classes.

L'une est la classe Event qui va effectivement recevoir les demandes de rezize de la fenêtre

L'autre est la classe Rezizer qui va effectivement redimensionner la fenêtre.

Pourquoi deux classes ? Explication ...

La classe Event va recevoir un événement a chaque redimensionnement, par redimensionnement java entent "a chaque pixel bouger par le redimensionnement j'envoie un signal".

Donc je me retrouvai avec 200 appels à la fonction componentResized d'Évent.

Sauf que 200 redimensionnements globaux de toutes les cases fessai ramer l'application et mettait 1 minute pour réafficher les cases redimensionner.

La solution est celle de la classe Resizer, cette classe possède 2 méthodes,

L'une est addDemande qui va s'incrémenter a chaque fois que componentResized va être appelé.

Le nombre de demandes va être très grand.

La deuxième méthode est un timer réglé sur 500 ms qui va appeler la seconde méthode en boucle.

La seconde méthode définie que si le nombre de demandes n'est pas égal à 0 alors je redimensionne et je place le nombre de demandes à 0.

Du coup a la place de faire 200 redimensionnements je n'en est que 2 ou 3 en fonction du temps que l'on met a redimensionner la fenêtre. Puis je n'en est pas du tout quand la fenêtre bouge pas.

V) Fonctionnement de l'attaque

Les attaques sont gérées par une interface, l'interface prend en paramètre rimbo.

L'attaque enlève de la vie a rimbo puis si la vie de rimbo est négative ou égale a 0 on définie sa vie a 0 (pour éviter la vie a -1) et on termine le jeu, si la vie est supérieur a 0 on fait appelé a la fonction fuir de rimbo.

VI) Gestion des options

La gestion des options se fait grasse à une nouvelle fenêtre, cette fenêtre contiens un sélecteur pour le niveau de difficulté, et deux cases qui corresponde au nombre de cases X et Y que l'on souhaite.

Quand les changements sont validés, cela modifie toutes les variables de l'ile avec les nouvelles valeurs, cela purge les anciennes cases et régénère une nouvelle carte.

En fonction du niveau de difficulté choisi, les valeurs peuvent changer, nombre de monstres, vie de rimbo, intelligence des monstres, nombre de ZM.