ZoMbIAGE.

Projet Applications réseau M1 Miage Apprentissage Paris-Dauphine 2014/2015

F. Sikora (Inspiré d'un sujet de S. Thibault)

1 Dates importantes

- Rendu de la RFC: ~~13 avril~~ 17 avril 2015, 23h.
- Rendu du projet: 9 juin 2015, 23h.
- Soutenances: 11 juin 2015, à partir de 13h30 en B028.

2 Versions

Ce sujet est susceptible d'être modifié.

- 30 mars 2015: Mise en ligne du sujet.

3 En quelques mots

Des poules-zombies ont envahi MIAGE City. Chaque joueur contrôle un certain nombre d'éléments présents dans la ville (poules-zombies, coq-zombies, humains, objets statiques...) mais affiche également les élément possédés par les autres joueurs. En d'autres termes, dans un même espace, la ville, plusieurs joueurs possèdent plusieurs éléments. Si un joueur quitte le jeu, les éléments possédés par ce joueur disparaissent de tous les rendus. Si un joueur rejoint le jeu, les éléments possédés par ce joueur apparaissent sur tous les rendus.

4 Règles

Les règles sont les suivantes. Les éléments statiques ne se déplacent pas. Des humains et des poules-zombies (des mâles et des femelles) se déplacent dans la ville, vers une destination. Lorsqu'un zombie rencontre un disque (élément statique), décapité, il meure (ceux qui trouvent la référence peuvent l'indiquer dans la doc utilisateur). Comme c'est aussi une poule, elle pourrait continuer à marcher quelque temps avant de mourir...

Toutes les règles suivantes ne sont pas forcément à coder (mais jouent sur la note finale). Lorsqu'une poule-zombie rencontre un humain, elle le pique et transforme l'humain en poule-zombie. Lorsqu'un zombie rencontre un zombie plus petit que lui, il le mange et grossit. Pour se reproduire, des femelles zombies peuvent déposer des œufs (partagés entre les joueurs) qui peuvent être fécondés par des mâles zombies et qui deviennent des bébés. Les bébés (pouvant être soit humain, soit déjà zombies) grossissent petit à petit jusqu'à une taille aléatoire fixée

à la fécondation. On pourrait aussi imaginer que les poules veuillent manger des vers de terre, devenant alors des godzillas, mais ça va peut être un peu loin... Vous pouvez utiliser http://pixabay.com/ pour des images libres.

5 Protocoles

On demande l'écriture d'un protocole réseau (RFC) gérant les communications entre les joueurs/serveurs (à adapter selon les versions définies plus loin). Le protocole doit définir tous les messages devant être échangés (avant, pendant, après etc). La définition de ce protocole est libre mais doit utiliser TCP.

Les joueurs peuvent vouloir discuter au cours de la partie. On demande également de décrire la manière dont ces communications sont effectuées. Celles-ci doivent se faire en UDP directement de joueur à joueur (le serveur ne relaie pas les messages). Ils doivent cependant passer par le serveur (à adapter selon les versions définies plus loin) pour connaitre l'adresse/port des autres joueurs.

6 Implémentation

On demande l'implémentation en Java des protocoles définis.

6.1 Version 2 joueurs

Je donne sur MyCourse une version de base pour éviter que vous perdiez du temps sur du swing et des éléments non importants dans ce projet en particulier. La ville est affichée toutes les 10ms par un thread périodique. À chaque rafraichissement, les éléments mobiles se déplacent un peu vers leur destination. La vitesse est dépendante de la taille de l'élément (plus il est gros, moins il est rapide). Pour ajouter un nouveau type d'élément, il faut étendre la classe `StaticItem` ou `MoveableItem`. Pour ajouter un nouvel élément, il faut l'ajouter dans le modèle.

Les destinations des éléments peuvent être aléatoire (choix d'un item courant du jeu comme cible). Une poule-zombie vorace peut choisir de viser les poules-zombies plus petites qu'elle. On peut aussi envisager des poules-zombies indécises qui changent d'avis toutes les secondes (et utiliser une tâche périodique pour cela...). Optionnellement, l'utilisateur peut prendre le contrôle de certains éléments.

Une première version simple consiste donc à faire fonctionner le jeu avec deux joueurs. Un premier joueur lance son jeu, et un second joueur rejoint la partie en indiquant l'adresse et le port du premier joueur. Les deux joueurs peuvent alors communiquer (indiquer la position des éléments à afficher etc). Chaque joueur doit voir, sur son propre affichage, l'ensemble des éléments qu'il possède ainsi que l'ensemble des éléments que l'autre joueur possède (affiché avec une autre couleur pour plus de clarté). Par conséquent, les deux joueurs doivent avoir exactement le même affichage, au temps de latence et aux couleurs près.

6.2 Versions à plus

On veut pouvoir gérer plus de 2 "maitre des zombies". Cependant, un joueur ne veut pas envoyer à tout le monde la position de ses éléments: cela créerait trop de connexions et de

messages. Toutes les méthodes suivantes ne sont pas forcements à coder (mais jouent sur la note finale).

6.2.1 Un serveur

Une première version simple consiste à utiliser un unique serveur centralisant les informations reçues par les joueurs puis les diffusant à tous les joueurs.

6.2.2 Plusieurs serveurs

Si le serveur précédent tombe, le jeu ne fonctionne plus. Pour cela, on envisage une version où il existe plusieurs serveurs, chacun gérant plusieurs clients. Les serveurs communiquent alors entre eux pour connaitre leurs adresses mais aussi pour diffuser les informations reçues par les clients et assurer que les messages sont bien diffusés à tous les joueurs. Attention à ne pas envoyer 2 fois la même information à un même serveur ou joueur (attention aux cycles dans le "réseau des serveurs") ! Lorsqu'un serveur est déconnecté, seuls les joueurs qui étaient connectés à se serveur sont déconnectés (et peuvent tenter de se reconnecter à un autre serveur), le reste des joueurs doit pouvoir continuer à jouer. Optionnellement, on peut gérer la charge sur les serveurs automatiquement (pour éviter qu'un serveur gère beaucoup de clients alors que d'autres serveurs contrôlent très peu de clients)...

6.2.3 Peer to peer

Dans cette version, chaque joueur est lui-même une sorte de serveur et informe certains voisins de ses éléments (eux-même vont informer d'autres joueurs et ainsi de suite). Un joueur rejoint le jeu en se connectant via un autre joueur (c.a.d. qu'il indique un couple IP port) et récupère alors des informations sur d'autres joueurs. Lorsqu'un joueur se déconnecte (pas nécessairement de sa propre volonté...), les joueurs qui avaient rejoint le jeu grâce à lui doivent se reconnecter à d'autres joueurs.

7 Conditions de rendu

Le projet est à effectuer en binôme, *i.e.* par 2 personnes. Comme j'ai pu vérifier que vous n'étiez pas un nombre impair pour le projet java, il n'y aura pas d'exception à la règle précédente.

Première étape: description du protocole. Une première RFC de votre protocole (options comprises) est à rendre avant la date définie plus haut sur l'espace MyCourse dédié. Les noms des deux personnes du binôme devront apparaitre dans la RFC, dans le nom du fichier ainsi que dans les commentaires de l'espace MyCourse. Attention, l'espace est fermé automatiquement. La RFC doit être dans le même format que les autres RFC, c'est-à-dire un document texte ASCII rédigé en anglais (voir la RFC 2223 pour le format d'une RFC..., voir aussi ce lien http://www.rfc-editor.org/formatting.html). La RFC doit seulement décrire les méthodes de communications (format des messages, ordres des messages, protocoles utilisés...) entre les différentes entités intervenant dans votre protocole. Aucun détail d'implémentation ne doit être donné. La RFC 1350 (TFTP) peut-être prise en exemple comme une RFC simple décrivant ceci.

Seconde étape: implémentation(s) pour le protocole en langage Java. Théoriquement, deux implémentations de différents binômes pourraient communiquer entre elles si elles suivent la même RFC (théoriquement toujours, une implémentation en langage C++ lancée sur une machine personnelle pourrait communiquer avec une implémentation Java sur un téléphone androïd).

Votre projet est à rendre avant la date précisée plus haut, sur l'espace MyCourse dédié. Un seul envoi par binôme ! Une fois votre fichier envoyé, vous devez avoir un écran de confirmation avec votre fichier et la date de l'envoi. **Il y aura un point en moins par heure de retard, une heure entamée est due.** Le format de rendu est une archive au format **ZIP** contenant:

- Un répertoire *src* avec les sources de votre implémentation java.
- Un répertoire *classes* vide dans l'archive, devant contenir les classes du projet une fois compilé.
- Un répertoire *docs* contenant:
  - La première version de la RFC ainsi qu'une éventuellement mise à jour (les explications sur les changements apportés seront dans la documentation développeur).
  - Une documentation pour l'utilisateur *user.pdf* décrivant à un utilisateur comment se servir de votre projet.
  - Une documentation pour le développeur *dev.pdf*, devant justifier les choix effectués, présenter l'architecture choisie, indiquer quelles ont été les difficultés rencontrées au cours du projet ainsi que la répartition du travail entre les membres du binôme. Ce rapport doit faire le point sur les fonctionnalités apportées, celles qui n'ont pas été faite (et expliquer pourquoi). Il ne doit pas paraphraser le code, mais doit rendre explicite ce que ne montre pas le code. Il doit montrer que le code produit a fait l'objet d'un travail réfléchi et minutieux (comment un bug a été résolu, comment la redondance dans le code a été évitée, comment telle difficulté technique a été contournée, quels ont été les choix, les pistes examinées ou abandonnées...). Ce rapport est le témoin de vos qualités scientifiques mais aussi littéraires (style, grammaire, orthographe, présentation).
  - Un répertoire vide *doc* pour la javadoc.
- Un fichier ant *build.xml* permettant de compiler, créer le jar, générer la javadoc, etc.
- Votre(s) jar(s) exécutable(s) pouvant être lancés au moyen de la commande `java -jar`.

Votre projet doit pouvoir s'exécuter sans utiliser eclipse ! Attention aux images et leur recopie dans le jar. Testez !

L'archive aura pour nom Nom1Nom2.zip, où Nom1 et Nom2 sont les noms des membres du binôme par ordre alphabétique. L'extraction de l'archive devra créer un dossier Nom1Nom2 contenant les élements précisés ci-dessus.

Il va sans dire que les différents points suivants doivent être pris en compte:

- Uniformité de la langue utilisée dans le code (anglais conseillé) et des conventions de nommage et de code (convention Java obligatoire).

- Projet compilant sans erreur et fonctionnant sur les machines de l'université (et en réseau vu l'intitulé de la matière...)
- La RFC et le projet doivent évidement être propres à chaque binôme. Un détecteur automatique de plagiat sera utilisé. Si du texte ou une portion de code a été empruntée (sur internet, chez un autre binome), il faudra l'indiquer dans le rapport. Tout manque de sincérité sera lourdement sanctionné.

La documentation (rapports, commentaires...) compte pour un quart de la note finale. On préférera un projet qui fonctionne bien avec peu de fonctionnalités qu'un projet bancal avec plus de fonctionnalités.

L'utilisation d'un gestionnaire de version type CVS, SVN ou GIT est conseillée (attention toutefois aux sites où le code est public). Riouxsvn semble être gratuit et privé mais je ne le connais pas spécialement.

7.1 Soutenance

Une soutenance d'un quart d'heure aura lieu binôme par binôme à la date précisée plus haut. Elle doit être préparée et menée par le binôme (i.e. fonctionnant parfaitement du premier coup, EN RESEAU, avec des jeux de tests pertinents etc). Des jeux de tests / situations préparées mettant en valeur la correction de votre projet sont grandement recommandée (à fournir au moment du rendu).