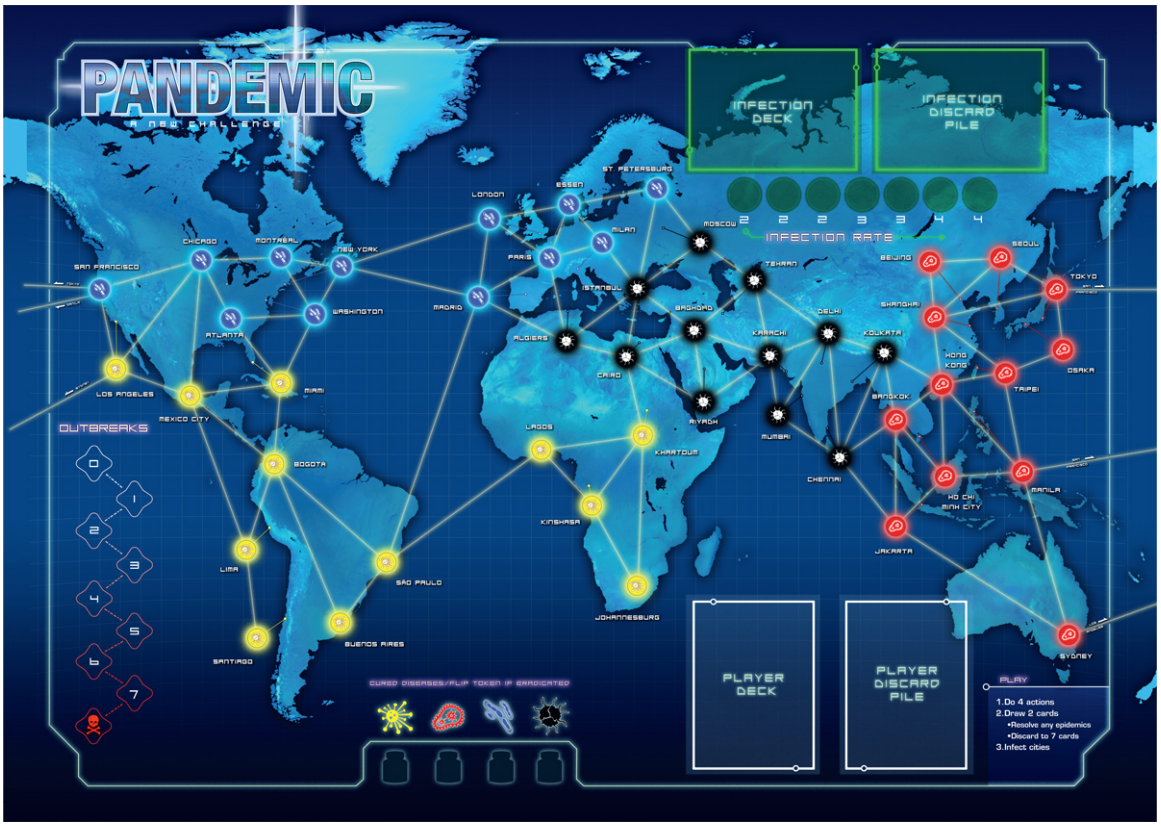
Projet de Pandemic [1]

Le monde est en danger ! Vous seul, ou en binôme, pouvez faire en sorte de sauver l'espèce humaine ... Dans un premier temps nous vous donnons un aperçu du jeu choisi, puis nous aborderons plus précisément point par point les aspects techniques de la modélisation.

Description light du jeu

L'action, simplifiée, se déroule sur la carte des grandes capitales. Des maladies vont y apparaître et évoluer. Elles sont représentées par des nombres colorés déposés sur les villes concernées. Chaque joueur possède un pion, et se trouve dans une ville. Leur objectif est de collaborer pour lutter contre la propagation, et si possible vaincre les maladies. Ils pourront effectuer un certain nombre d'actions : principalement des soins ou des déplacements, mais peut être aussi trouver un remède.



Situation de départ

Le pion de chaque joueur est situé à Paris. Certaines villes sont déjà infectées : 3 par des maladies qui commencent au stade 3, 3 autres par des maladies au stade 2, et encore 3 au stade 1. Il y a donc au départ 9 chiffres colorés situés sur 9 villes. On rappelle que le chiffre indique le stade d'évolution, et la couleur représente une maladie.

Une action d'un joueur consiste soit à déplacer son pion vers une ville voisine soit, si la ville où il est est infectée, à faire diminuer d'une unité le stade d'évolution d'une de ses maladies.

1. *C'est pandémie, pas pain de mie ! (Eric Judor - Problemos - 2017)*

C'est tout pour le moment, un joueur peut tout de même combiner librement jusqu'à 4 de ces actions à son tour.

Puis c'est au tour des maladies de progresser : une ville et une maladie sont choisies au hasard, et à cet endroit la maladie apparaît ou se développe d'un stade supplémentaire.

On passe ensuite au joueur suivant, etc. Nous avons décrit là le grand principe du déroulement du jeu.

Des graphes

Le premier effort de programmation portera sur la modélisations de la carte, qui est essentiellement un graphe. On définit l'interface :

```
public interface Graphe{
 public int getNbNoeud();
 public Iterator<Integer> getNoeudsVoisin(int num);
 public void putEtiquette(String nomVille, int num);
 public void addVoisins(int num1, int num2); // graphe symétrique
 public int getNumNoeud(String nomVille);
 public Iterator<String> getVillesVoisines(String ville);
 public String toString();
}
```

On considère un fichier texte, dont vous trouverez un exemple sur moodle, constitué de la manière suivante :

1. Une ligne $entier_{nb}$
2. Une ligne $entier_L$ # $entier_H$
3. Une suite de lignes de la forme *nom* # *description*
4. Une suite de lignes de la forme nom_1 ## nom_2

Ce fichier décrit un graphe dont $entier_{nb}$ est le nombre de noeuds, chaque noeud a un nom qui apparaît une et une seule fois dans la section numérotée 3. Les liaisons entre les noeuds (les arêtes du graphe) sont données par les paires de la section 4 ; elles doivent être interprétées comme étant symétriques. Pour votre information, mais ce n'est pas utile pour le moment, la description permettra de positionner un noeud sur une fenêtre dont la dimension est celle de la section 2.

Question 1. Ecrivez les deux implémentations classiques des graphes : une représentée par une matrice d'adjacence, l'autre par les listes d'adjacences. Vous écrirez le constructeur qui prend en argument un nom de fichier, ou un graphe de l'autre implémentation demandée.

Question 2. Modifier l'interface pour qu'elle soit générique en fonction d'un type de Noeuds

Séparer la vue du modèle

Votre modélisation sera constituée de diverses classes, on souhaite que certaines soient spécifiquement relatives à la présentation.

Il est toujours intéressant de dissocier dès la conception les aspects graphiques des problèmes spécifiques au modèle. Ainsi lorsqu'il faudra adapter un programme sur une plateforme différente

les enjeux seront bien cernés : on développera de nouvelles vues qui implémenteront une interface et cette vue pourra être substituée à l'ancienne dans le modèle.

L'association entre le modèle et la vue se fera en introduisant un champs correspondant à la vue dans le modèle de votre jeu, et réciproquement s'il y a besoin d'interaction. Pour rendre compte d'une modification graphique le jeu s'adressera toujours à la vue du modèle.

Pour manipuler des images nous vous fournissons une classe `ImageSimple` avec son constructeur `ImageSimple(String img_path, int x, int y)` où `img_path` est un nom de fichier d'extension `.jpg` ou `.png` et les valeurs (x, y) précisent sa nouvelle dimension en pixels.

Nous vous fournissons de quoi faire une vue, à partir de l'interface `VueGenerale`, et d'une implémentation `VueExemple`. Elle définit un certain nombre de méthodes que nous illustrerons par la suite :

- `VueExemple(String bgroundPic, int x, int y)` qui construit une fenêtre de dimensions x,y avec une image de fond dont on précise le chemin.
- `setVille(String ville, int x, int y)` associe une ville à la coordonnée (x, y) en pixels sur une image.
- `positionneEnVille(String ville, ImageSimple pion)` qui place l'image d'un pion sur une ville.
- `deplace(ImageSimple pion,String ville_dep, String ville_arr)`

```
VueGenerale v;
v = new VueExemple("../pandemicExemple.jpg",1200,800);
v.setCase("Paris", 570, 249);
v.setCase("Madrid", 492, 287);
ImageSimple pion=new ImageSimple("../pin.png",25,25);
v.positionneEnVille("Paris", pion);
v.deplace(pion, "Paris", "Madrid");
```

L'explication est la suivante : on crée et affiche la vue d'un plateau de jeu, puis on crée une image pour un pion, on informe la vue que Paris a pour coordonnées $(366, 136)$, on fait de même pour Madrid, on positionne le pion Paris, on le déplace de Paris à Madrid.

Premier Prototype

Question 3. Définissez une classe prototype, qui associe un graphe, une vue, etc .... Elle disposera d'un constructeur permettant la mise en place de la situation de départ décrite dans la section *Description light du jeu*, et vous pourrez éventuellement tester les premières règles avant de commencer la modélisation des joueurs qui sera la prochaine étape.

Joueurs

Dans notre contexte, les joueurs s'exprimeront au clavier, la gestion d'une souris est hors-sujet, ne perdez pas de temps avec cet aspect là.

On souhaite pouvoir paramétrer les compétences d'un joueur. Les actions possibles seront donc des objets à part entière, qui seront affectés à un joueur lors de sa construction. [2]

Voici quelques exemple :

2. Il s'agit là d'un parti-pris. Si vous voulez résoudre le problème de modélisation d'une autre manière vous pouvez le faire.

— le répartiteur est un joueur qui ne peut effectuer que des déplacements. Il choisit n'importe quel pion et le déplace sur une ville voisine. Il peut également amener un pion directement là ou se trouve un autre pion.
— l'infirmier est un joueur qui ne peut que soigner, il diminue le stade d'une maladie dans la ville où il est
— le french doctor est un joueur qui a la double compétence : de soigner et de se déplacer lui même.
— ...

Question 4. Développez les classes joueurs et actions.

Maladies

D'une certaines façon la gestion des maladies peut être vues comme le fait d'un joueur particulier. Nous allons enrichir le jeu en ce sens.

Voici les choix qui ont été fait pour le jeu original :

— Les villes sont regroupées par couleurs, pas tout à fait par continent mais presque. L'idée est que les raisons qui font qu'une maladie apparaisse sont liées à la géographie. Cela nous permettra de limiter géographiquement les apparitions.
— On cherche également à contrôler le timing d'apparition des maladies. Nous allons utiliser pour cela un paquet de carte, contenant initialement toutes les villes. Des opérations simples sur ce paquet permettrons de crédibiliser la chronologie de l'épidémie.

On peut à présent définir au moins deux joueurs différents pour la maladie :

— celui qui fait évoluer une maladie choisie au hasard, dans une ville prise au hasard
— celui qui tire une carte du paquet, et qui fait évoluer la maladie spécifiquement associée à la ville correspondantes
— ...

Question 5. Complétez votre modèle pour introduire la gestion des maladies

Règles plus avancées

Progression de la maladie

On considère que les maladies ne vont pas simplement évoluer en renforçant leur stade indéfiniment. On met donc en place la règle suivante, dite `d'explosion :` si une maladie devait dépasser le stade 3, alors elle reste à ce stade, mais chacune des villes voisines se trouve infectée d'une unité de plus. Pour éviter des situations de blocage, on considère que lors de la résolution d'un tour de cette règle, chaque ville ne peut être infectée par explosion en chaîne qu'une fois au plus.

Question 6. Définissez des jeux qui permettent d'intégrer cette nouvelle règle.

Variations sur les actions des joueurs

Le jeu `Pandemic` cherche à donner la possibilité aux joueurs d'étudier les maladies pour les vaincre. Cette idée est développée à l'aide d'un second paquet de cartes, similaire à celui utilisé pour les maladies. Il sert de support à un certain nombre d'actions qui prisent ensembles permettent de simuler les recherches sur les maladies :

— Lors d'une action d'*étude* un joueur pioche une carte. Il ne peut en conserver qu'un nombre limité 7.
— Si un joueur a en main 5 cartes de la même couleur, il peut s'en défausser et faire une action de *cure*. Alors la maladie correspondante sera vaincue, et ne pourra plus se développer.
— Une action d'*oubli* consiste à se défausser d'une de ses cartes.
— Une action d'*échange* consiste à échanger une carte de son choix avec un autre joueur.

Question 7. Implémentez le cadre pour ces nouvelles actions

Question 8. Nous n'avons pas vraiment précisé les conditions de gain des joueurs ou des maladies. Vous pouvez en choisir quelques unes qui soient crédibles dans le contexte du jeu.

Quelques conseils

Si en cours de réalisation vous décidez de faire des modifications importantes, prenez la précaution de sauvegarder votre ancienne version. Un programme qui ne marcherait pas à cause d'un changement de dernière minute serait très difficile à évaluer.

Evitez les copiés-collés, si vous en faites c'est que probablement il y a un concept commun que vous n'avez pas su dégager.

Rapport et soutenance

— Le projet est à faire seul ou en binômes.
— Les soutenances sont prévues en janvier et seront individuelles.
— Votre travail est à rendre sur `Moodle` sous la forme d'une archive nommée *`nom1-nom2`* selon la constitution de votre binôme, et qui s'extraira dans un répertoire *`nom1-nom2`*. Elle devra contenir :
  — les sources et ressources (images ...) de votre programme. Attention à bien utiliser des noms relatifs pour que nous puissions tester votre code sans avoir à modifier quoi que ce soit.
  — un fichier nommé `README` qui indique comment on se sert de votre programme, et ce qu'il nous faut regarder en premier.
  — un rapport au format *PDF* de quatre ou cinq pages expliquant les parties traitées, les problèmes connus, et les pistes d'extensions que vous n'auriez pas encore implémentées.
  — pensez à préparer tout ce que vous jugerez utile pour rendre la soutenance fluide.
  — la soutenance devra pouvoir se faire sur les machine de TP à partir des sources que vous avez déposées. Elle se déroulera dans un mélange de questions et de tests. Il pourra vous être demandé de modifier votre code pour répondre à une question spécifique.