CADIOT Vincent
OSHIRO Ken
M2 CCI

Rapport Projet Programmation Impérative

Projet « RISK »

(Ou la supériorité de l'imagination sur la banalité d'une interface graphique)

1. Introduction

Dans le cadre du cours de programmation impérative nous avons été amenés à réaliser un projet mettant en œuvre les notions d'algorithmique et de programmation abordées. Notre choix s'est porté sur le jeu « RISK » pour deux raisons : la difficulté de réalisation, de telle sorte qu'il ne soit ni trop facile ni trop difficile à programmer, et l'aspect ludique associé, afin de trouver le projet plaisant à réaliser et à tester.

2. Le jeu « Risk »

1) Les règles

Le jeu « Risk » est à la base un jeu plateau qui se joue au tour par tour dans des parties de 2 à 6 joueurs. Le plateau représente la carte du monde constituée de 42 pays regroupés en 6 continents. Chaque joueur reçoit un objectif de conquête qu'il doit remplir.

Déroulement d'une partie :

Mise en place du jeu :
Chaque joueur se voit attribué en début de partie un certain nombre de pays, des pions représentants des armées ainsi qu'une carte précisant sa condition de victoire.
Exemple :
- Vous devez conquérir en totalité l'Amérique du Nord et l'Afrique

Le nombre de pions reçu par les joueurs en début de partie dépend du nombre de joueurs (40 à 2 joueurs, 35 à 3,...). Le joueur doit ensuite répartir ses pions sur ses territoires. Attention tout territoire doit obligatoirement être occupé par au moins 1 pion (et ce à tout moment du jeu).
Lorsque chaque joueur a terminé de placer ses pions, le jeu commence.

Tour de jeu :

Un tour de jeu se déroule en trois phases :

Phase de renforts
Le joueur reçoit un nombre de pions égal à ses territoires contrôlés /3 (arrondi à l'entier inférieur) + des renforts bonus s'il possède en intégralité un ou plusieurs continents. Il les dispose où il le souhaite sur ses territoires.

Phase d'attaque
Le joueur peut attaquer autant de fois qu'il souhaite durant un tour. Il ne peut attaquer qu'un territoire voisin avec un maximum de 3 armées. Il déclare d'où et où il attaque avec combien d'armées, puis lance 1 dé pour chaque armée engagée. Le défenseur lance 1 ou 2 dé (si deux armées ou plus sont présentes). La plus grande valeur de l'attaque est comparée à la plus grande valeur de la défense, celui ayant la plus grande l'emporte et détruit une armée ennemie. Si le défenseur et l'attaquant ont jeté au moins deux dés, alors la seconde plus grande valeur de l'attaque est comparée à la seconde valeur de la défense, et une perte est infligée à celui ayant la plus faible. En cas d'égalité c'est le défenseur qui l'emporte
Lorsque l'assaillant détruit toutes les armées du défenseur sur un territoire il occupe alors ce territoire avec les armées qu'il a engagé lors de l'attaque décisive. Il peut ensuite transporter sur le territoire conquis, en partant du territoire attaquant, autant de troupes qu'il le désire en laissant au moins 1 pion sur le territoire. Il a aussi le droit de se servir immédiatement du territoire nouvellement acquis comme point de départ pour l'attaque des pays voisins.

Phase de déplacements internes
Le joueur peut déplacer autant de fois qu'il le souhaite ses troupes entre deux pays voisins lui appartenant.
Lorsqu'il a validé son tour prend fin et le joueur suivant commence.

Fin du jeu
Le jeu prend fin lorsqu'un des joueurs réussit son objectif.

3. Spécification du problème

Le programme fonctionne via l'utilisation de la console et les saisies clavier des joueurs. Il a été codé de façon modulaire, c'est-à-dire qu'il consiste en une fonction 'main' qui appelle d'autres fonctions nécessaires pour le déroulement du jeu. Deux objets ont aussi été créés : des joueurs et des pays. Une partie importante du travail a donc été de gérer les interactions entre les différents objets, leurs champs, les fonctions et leurs retours, ainsi que la vérification des saisies des joueurs afin qu'elles ne soient pas aberrantes.

Le projet doit pouvoir résoudre les éléments suivants :
- la possibilité de jouer à 3 joueurs partageant la même console
- le respect des règles du jeu de plateau
- le jeu s'arrête lorsque l'un des joueurs a atteint sa condition de victoire.

4. Algorithmes et programme

1) Les objets

a) Pays

Cet objet va nous permettre, comme son nom l'indique, de gérer les pays du plateau de jeu. Il possède 5 champs :
Le champ « joueur » permet de renseigner le joueur qui possède le pays, « id » correspond au n° du pays, « nom » à son nom, « occupepar » au nombre de pions placé sur le pays et « tabdeplacements » est un tableau de « Pays » qui regroupe tous les pays voisins (pays vers lequel il est possible de déplacer les troupes).
Des « getters » et « setters » classiques sont ajoutés afin de modifier le propriétaire d'un pays ou le nombre d'armées occupant un pays et d'obtenir les informations sur les pays.

```
public  void setJoueur(Joueur joueur){
        This.joueur=joueur;}
```

```
public  Joueur getJoueur() {
        return joueur;}
```

Tableau 3 : exemple d'un « setter » et d'un « getter » pour le champ « joueur »

b) Joueur

Cet objet va nous permettre de gérer les joueurs. Il contient 3 champs :
Les champs « couleur » et « id » vont permettre de différencier les joueurs et le champ « nConditionDeVictoire » permettra de stocker le numéro associer à la condition de victoire tirer au sort par le joueur dans le programme principal.
Le champ « id » est incrémenté automatiquement à chaque création de joueur.
Seuls des « getters » sont créés afin d'obtenir les informations sur les joueurs. Les « setters » sont ici inutiles puisque les informations contenues par l'objet « Joueur » n'ont pas besoin d'être modifié en cours de partie.

2) Les fonctions

Afin de clarifier le programme et éviter la répétition d'actions similaires nous avons créé plusieurs fichiers contenant des fonctions. En voici quelques exemples (non détaillés) :

a) Fonction « CreationPays »

```
fonction CreationPays(tabPays : tableau de Pays) : tableau de Pays
```

Tableau 1 : signature de la fonction « CreationPays »

Cette fonction prend en paramètre un tableau de pays et le retourne avec les pays créés.

```
Pays : p1 = new Pays(1,"Afghanistan",0);
tabPays[0] ← p1;
p1.addVoisin(p40); //ajout du pays 40 comme voisin du pays 1
...
```

Tableau 2 : exemple de création d'un Pays

b) Fonction « PlacementArmees »

Cette fonction prend en paramètres un joueur, le tableau des pays qu'il possède, le tableau de tous les pays et les tableaux des continents et permet au joueur de positionner les troupes qui lui ont été attribuées en début de partie sur ses pays.
Pour permettre de bien respecter la règle stipulant qu'au moins une armée doit être positionnée sur un pays nous effectuons-nous même cette opération et retirons les pions utilisés des pions restants à placer.

```
entier nbpions ←21;
pour i de 0 à 14 par pas de 1 faire {
        TabJoueurPays.get(i).setOccupepar(1);
}
```

Tableau 3 : placement d'une armée sur tout les pays du joueur

Un menu est ensuite présenté au joueur lui présentant les différentes actions possibles dans la suite de cette fonction (voir la carte du monde (choix 1) et placer ses armées (choix 2)).
Ce menu est répété jusqu'à ce que le joueur ai placé tous ces pions.

```
répéter {
        //menu
}jusqu'à (nbpions <= 0);
```

Tableau 4 : répétition du menu

```
case 2:
        //affichage force joueur
        Répéter {(1)
                Afficher ("saisissez le numéro du pays sur lequel vous voulez placer
une/des armée(s)");
                Num ← saisie() ;
        } tant que (num<1 ou num>=43); (2)
        Si (TabJoueurPays .contains(tabPays[num-1]) = vrai){ (3)
                Afficher ("saisissez le nombre de troupe à placer");
                Nbarme ←saisie() ;
                Si (nbarme<=nbpions et nbarme>=0) { (4)
```

```
                tabPays[num-1].setOccupepar(barme+tabPays[num-1].getOccupepar());
                        nbpions ← nbpions-nbarme;
                } Sinon
                        Afficher ("Vous ne disposez pas d'assez de troupes"); (5)
        } Sinon
                Afficher ("Ce pays n'est pas disponible"); (5)
        break;
```

Tableau 5 : choix n°2 placement des armées

Pour ce choix nous commençons par afficher les forces du joueur, puis dans le (1) nous permettons au joueur de placer ses troupes. Il doit commencer par saisir le n° du pays sur lequel placer ses troupes, ce n° doit être compris entre 1 et 42 (2) pour que le pays existe et doit appartenir au tableau des pays du joueur (3). Il doit ensuite saisir le nombre de pions qu'il souhaite placer, ce nombre devant être compris entre 0 et le nombre de pions dont il dispose (4). On vient ensuite à l'aide du « setter » approprié modifier l'état du pays et décrémenter le nombre de pions restants. Des messages s'affichent si le joueur rentre des valeurs incohérentes (5).

c) Fonction « renforts »

Cette fonction prend en paramètres le joueur dont c'est le tour, le tableau de la liste de ses pays, le tableau de tous les pays et les tableaux des continents et retourne le nombre de renforts que le joueur doit obtenir au début de son tour.
On commence par calculer le nombre de renforts à fournir au joueur en fonction du nombre de territoire qu'il possède :

```
pionsdispo = floor(TabJoueurPays.size()/3)); //la fonction « floor » réalise l'arrondi
```

Tableau 6 : calcul du multiple de 3 territoires (arrondi à l'inférieur).

Puis on vient ajouter à ce nombre les renforts bonus pour la possession d'un continent, ex :

```
Pour i de 0 à 6 par pas de 1 faire {
        Si (Continent5[i].getJoueur() = j1){
                cptEurope ← cptEurope+1;
        }
}
Si  (cptEurope = 6) {
      pionsdispo += 5;
}
```

Tableau 7 : calcul des renforts supplémentaires pour la possession de l'Europe

d) Fonction « PlacementRenfort »

Cette fonction prend en paramètres un entier correspondant au nombre de renforts reçu par le joueur, le joueur dont c'est le tour, le tableau de ses pays et le tableau de tous les pays. Elle sert tout comme la fonction « PlacementArmees » à placer les renforts.

e) Fonction « combat »

Cette fonction prend en paramètres le nombre de troupes envoyées par l'attaquant et le nombre de défenseurs présents sur le pays attaqué et renvoie un tableau contenant les pertes des 2 joueurs.

On commence par remplir des tableaux contenant les résultats des lancers de dés de l'attaquant et du défenseur. Le nombre de jets est déterminé au préalable en fonction des paramètres entrés pour la fonction (cf règles) Combats).

```
resultatatt[i]  = (1+random() * ((6-1) +1)));
```

Tableau 8 : calcul d'un nombre aléatoire entre 1 et 6 correspondant à un jeté de dé à 6 faces pour l'attaquant

On obtient ensuite le dé le plus fort de chaque joueur :

```
Pour i de 1 à jetsatt par pas de 1 faire {
        Si (resultatatt[i] > maxatt) {
        maxatt ← resultatatt[i];
        }
}
```

Tableau 9 : calcul du dé le plus fort de l'attaquant

S'il y a plus de 1 lancer de la part de l'attaquant il faut comparer les 2èmes dés les plus forts. On commence donc par supprimer les dés déjà utilisés :

```
Pour i de 1 à jetsatt par pas de  1 faire {
        Si (resultatatt[i] = maxatt) {
        resultatatt[i] ← 0;
        maxatt ← 0;
        }
}
```

Tableau 10 : suppression du dé déjà utilisé par l'attaquant

On vient ensuite comme précédemment recalculer les nouveaux dés les plus forts et les pertes des 2 joueurs. Le tableau des pertes est ensuite retourné.

f) Fonction « DeplacementTroupes »

Cette fonction prend en paramètre le joueur dont c'est le tour, les tableaux des pays de tous les joueurs, le tableau de tous les pays et les tableaux des continents et renvoie un booléen qui signifie que le joueur à fini ou non son tour.
Elle permet au joueur dont c'est le tour de visualiser l'état de ses troupes, l'état du monde, d'afficher les déplacements possibles des troupes du joueur, d'attaquer ses ennemis et de réaliser les déplacements internes de ses troupes à travers un menu. Les 2 premières options sont identiques à celles de la fonction « PlacementArmées ».
Affichage des déplacements possibles :

```
Pour j de 1 à longueur(TabJoueurPays) par pas de 1 faire {
        //Affichage du pays (j)  du joueur
        Voisin ← TabJoueurPays.get(j).getDeplacements(); //get(j) permet d'accéder à
//l'élément j de TabJoueurPays
        Pour i de 1 à longueur(voisin) par pas de 1 faire{
                Si (voisin[i] != null){
                //Affichage des infos sur le pays voisin (i)
                }
        }
}
```

Tableau 11 : affichage des déplacements possibles

On récupère pour chaque pays du joueur le tableau de leurs voisins et on affiche les infos sur ces derniers (nom, n°, armées, joueur le possédant).
Attaque :
On commence par récupérer le pays de départ pour l'attaque (cf fonction « PlacementArmées ») puis le nombre de troupes à envoyer («nbarme ») :

```
Si (nbarme<=tabPays[numPays-1].getOccupepar() et nbarme>=0 et nbarme<=3 et
((tabPays[numPays-1].getOccupepar()-nbarme)>=1)){
```

Tableau 12 : vérification du respect des règles lors du choix du nombre de troupes à envoyer

On récupère ensuite le numéro du pays à attaquer en vérifiant qu'il est bien voisin du pays attaquant et qu'il appartient à un autre joueur. Des messages d'erreurs sont affichés sinon.

```
nbDefenseurs ← tabPays[numPays1-1].getOccupepar();
pertesCombat ← Combat.combat(nbarme, nbDefenseurs);
```

Tableau 13 : on récuère le nombre de défenseurs et on lance la fonction « combat »

Les pertes sont prises en compte pour les 2 joueurs et on regarde si l'attaquant conquiert le pays :

```
Si (tabPays[numPays1-1].getOccupepar() = 0)
```

Tableau 14 : le défenseur n'a plus de troupes, l'attaquant conquiert le pays

L'état des troupes est modifié et le pays conquis est supprimé du tableau des pays du défenseur et ajouté à celui de l'attaquant :

```
tabPays[numPays1-1].setOccupepar(nbarme-pertesCombat[0]);
tabPays[numPays-1].setOccupepar(tabPays[numPays-1].getOccupepar()-(nbarme-
pertesCombat[0]));
TabJoueurPays.add(tabPays[numPays1-1]);
Pour j de 1 à longueur(TabJoueurPays1 par pas de 1 faire {
        Si (TabJoueurPays1.get(j).getJoueur().getCouleur() = j1.getCouleur()){
                TabJoueurPays1.remove(tabPays[numPays1-1]);
        }
}
```

Tableau 15 : modification de l'état des troupes et changement du propriétaire pour le pays (exemple avec le joueur 2 comme défenseur)

Ensuite le joueur peut déplacer des troupes depuis le pays attaquant vers le pays conquis en respectant les règles.
Différents messages avertissent des résultats du combat.

Déplacements internes :
Cette option est similaire à celle du dessus si ce n'est que le pays vers lequel le joueur souhaite se déplacer est bien voisin du pays de départ et appartient à ce joueur et qu'il n'y a pas de combat. On se contente de modifier l'occupation des 2 pays à l'aide des « setters ».
Cette opération de déplacements internes est répétée jusqu'à ce que le joueur signale par une saisie d'un nombre (converti en booléen) qu'il a bien fini cette action. La fin de cette opération ou la saisie d'un nombre « hors-menu » entraîne le renvoi d'un booléen « vrai ». Les autres actions renvoient le booléen « faux ».

g) Fonction « verifiervictoire1 », « verifiervictoire2 », etc…

Ces fonctions prennent en paramètre le joueur dont on souhaite vérifier s'il a rempli sa condition de victoire, ainsi que les tableaux des continents utiles à cette vérification. Ces fonctions renvoient toutes un booléen signifiant que le joueur à oui ou non rempli sa condition de victoire. Toutes les conditions sont vérifiées à l'aide de compteurs :

```
Pour i de 1 à 9 par pas de 1 faire {
        Si (AmeriqueNord[i].getJoueur() = j1){
        cptAmNord ← cptAmNord+1;
        }
}
```

Tableau 16 : conquérir l'Amérique du Nord

h) Fonction « VictoireFinale »

Cette fonction prend en paramètres le joueur dont c'est le tour, le tableau de pays et les tableaux des continents. Elle fait appel à toutes les fonctions « verifiervictoire(i) » et renvoie un booléen qui indique si la condition de victoire du joueur passé en paramètre est remplie.

3) Programme principal

Dans ce programme on commence par obtenir un tableau des conditions de victoire en vérifiant que chaque joueur aura bien une condition différente. Le numéro de la condition est tiré aléatoirement (cf fonction « combat »).
On créé ensuite les joueurs :

```
Joueur j1=new Joueur("Bleu",cond[0]);
```

Tableau 17 : création du joueur 1

Les joueurs sont placés dans un tableau de joueur qui servira pour gérer les joueurs dans les tours de jeu.
On créé ensuite les pays, le tableau des pays et les tableaux des continents.
On attribue aléatoirement les pays aux joueurs en se servant d'un tableau de pays non attribués. Ce tableau associe au n° d'un pays un booléen « faux » si le pays est non attribué, « vrai » sinon. On tire au sort un numéro de pays que l'on attribue au joueur s'il est présent dans le tableau des pays non attribués, puis on le retire de ce tableau.

```
Pour i de 1 à 3 par pas de 1 faire {
        Pour j de 1 à nbpays par pas de 1 faire {
                Test ← faux;
                Tant que (test = faux) {
                        Y ← (random() * ((41) +1)));
                        Si (tableauAttribution[y] = faux){
                                tabPays[y].setJoueur(tabJoueur[i]);
                                Si (i=0){
                                        Joueur1Pays.add(tabPays[y]);
                                }Sinon{
                                        …
                                        }
                        }
                        tableauAttribution[y] ← vrai;
                        test ← vrai;
                }
        }
}
```

Tableau 18 : attribution des pays

On gère ensuite la phase de mise en place du jeu en appelant pour chaque joueur la fonction « PlacementArmees » :

```
Pour i de 1 à 3 par pas de 1 faire {
        PlacementArmees(tabJoueur[i], TabJoueurPays[i], tabPays, Afrique, AmeriqueNord,
AmeriqueSud, Asie, Europe, Oceanie);
                }
```

Tableau 19 : mise en place des armées

On gère ensuite les tours de jeu :

```
Répéter {
        Pour i de 1 à 3 par pas de 1 faire {
                Nouveauxpions ← Renfort.renforts(tabJoueur[i], TabJoueurPays[i],tabPays,
Afrique, AmeriqueNord, AmeriqueSud, Asie, Europe, Oceanie);
                PlacementRenfort(nouveauxpions,tabJoueur[i], TabJoueurPays[i], tabPays);
                Répéter {
                        finTour ← DeplacementTroupes(tabJoueur[i], TabJoueurPays[i],
TabJoueurPays[j], TabJoueurPays[k],tabPays, Afrique, AmeriqueNord, AmeriqueSud, Asie,
Europe, Oceanie);
                } jusqu'à (finTour = vrai);
                finJeu ← VictoireFinale(tabJoueur[i], tabPays, Afrique, AmeriqueNord,
AmeriqueSud, Asie, Europe, Oceanie);
                Si (finJeu = vrai) {
                        break; // on sort de la boucle "for" car un joueur  à réalisé sa condition de
victoire
                }
        }
} jusqu'à (finJeu = vrai);
```

Tableau 20 : résumé d'un tour de jeu avec appel des différentes fonctions

Dans cette phase on commence par récupérer les renforts et les placer puis on passe à la phase de choix (vue des forces, attaques, déplacements internes) jusqu'à ce que la fonction « DeplacementTroupes » renvoie « vrai ». On regarde alors si le joueur à rempli sa condition de victoire et si c'est le cas on met fin à cette phase, sinon on passe au tour de jeu du joueur suivant.

5. Conclusion

Pour réaliser ce projet nous avons utilisé tout ce que nous avons vu en programmation impérative, fonctions, tableaux, structures conditionnelles, boucles…
Nous avons rencontré plusieurs difficultés lors de la réalisation du projet. Tout d'abord la maîtrise des tableaux a été cruciale : il a été nécessaire d'utiliser des tableaux de tableaux de Pays, ainsi que des Arraylist. Nous avons rencontré de nombreuses erreurs liées à l'utilisation de ceux-ci, notamment des dépassements de la taille du tableau et des pointeurs nuls. Cela nous a amené à reconsidérer plusieurs fois le code.
La seconde difficulté a été de maîtriser les appels et retours de fonctions. Puisqu'il est impossible de retourner plus qu'une seule valeur, nous avons dû créer des tableaux ou réorganiser le code.

Le programme pourrait être amélioré par l'ajout des éléments suivants :

- Modifier le nombre de joueurs, la carte du monde...
- Rajouter des règles, variantes, nouvelles conditions de victoire…
- Implémenter le tout sur une interface graphique

6. Manuel d'utilisation – RISK

1. Présentation du jeu

Cette version de Risk se joue à 3 joueurs au tour par tour. Vous devez conquérir des territoires à l'aide d'armées que vous déplacez à travers le monde. Vous devez compléter l'objectif qui vous sera attribué en début de partie. Cet objectif est secret, les joueurs non actifs ne doivent pas regarder l'écran de jeu lors des tours des autres joueurs.

2. Lancement

Ouvrez un terminal et accédez au dossier qui contient le fichier Risk.java. Entrez ensuite la commande < java Risk > afin de lancer le jeu.

3. Déroulement

Attention : lorsque le jeu vous demande de saisir un nombre, ne saisissez pas un caractère sinon le programme vous renverra une erreur et s'arrêtera.

Tour de placement

Le premier joueur reçoit son objectif de victoire (secret), 14 territoires, et 21 armées. Il doit placer tous ses pions sur ses territoires.
Lorsqu'il n'a plus d'armées à placer, le second joueur peut jouer. Une fois que le troisième joueur a terminé de placer ses armées, le vrai tour de jeu commence.

Tour de jeu

Le joueur commence par recevoir des renforts (= ses territoires / 3 et des renforts bonus s'il possède l'intégralité d'un continent ou plus). Il doit tous les placer sur ses territoires.
Une fois ceci fait, il passe à la phase d'attaque. Chaque joueur peut attaquer autant de fois qu'il le souhaite tant qu'il respecte ces conditions : il ne peut attaquer qu'un territoire voisin, il doit laisser au minimum une armée sur chaque territoire contrôlé, et il ne peut pas déplacer des armées d'un de ses territoires à un autre.
Pour afficher les territoires et les forces du joueur actif, saisissez 1. Pour connaître l'état général de la carte du monde, saisissez 2. Pour connaître vos voisins (déplacements possibles), saisissez 3. Pour attaquer, saisissez 4. (Note : Il est obligatoire d'afficher d'abord vos déplacements avant d'attaquer, sinon vous ne saurez pas quel territoire attaquer!).
Lorsqu'un joueur attaque il peut choisir d'engager de 1 à 3 armées, ce qui lui permet de faire 1 à 3 jets de dés. Le défenseur jette automatiquement 1 à 2 dés (s'il a plus d'une armée sur le territoire). Le plus haut résultat de l'attaquant et du défenseur sont comparés, et une perte est attribuée à celui qui a la valeur la plus faible. S'il y a eu deux armées ou plus engagées par l'attaquant et le défenseur, la seconde plus haute valeur de l'attaque et la défense sont également comparées et une perte est attribuée. Une égalité est à l'avantage du défenseur.
Lorsqu'il n'y a plus de troupes défendant un territoire celui-ci passe sous le contrôle du joueur attaquant. Il peut choisir de déplacer autant de troupes qu'il souhaite du territoire d'où est venue l'attaque vers le territoire conquis (tant qu'il laisse au moins une armée sur chaque territoire).
Lorsqu'un joueur estime avoir terminé ses attaques, il saisit 5. Il passe alors à la phase de déplacement (saisir 6 ou plus saute la phase de déplacement et termine le tour immédiatement).
Lors de la phase de déplacement un joueur peut choisir de déplacer ses armées d'un de ses territoires vers un autre, sans limite si ce n'est de laisser une armée par territoire. A la fin de ses mouvements, le jeu vérifie si les conditions de victoire sont remplies. Si ce n'est pas le cas, le tour du joueur suivant commence. Le jeu continue jusqu'à ce qu'un des joueurs atteigne son objectif.