CADIOT Vincent

Rapport Projet Programmation Impérative

Projet « RISK »

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

OSHIRO Ken

M2 CCI

# Sommaire :

Table des matières

[Sommaire : 2](#_Toc472269343)

[1. Introduction 2](#_Toc472269344)

[2. Le jeu « Risk » 2](#_Toc472269345)

[1) Présentation historique du jeu 2](#_Toc472269346)

[2) Les règles 2](#_Toc472269347)

[3. Algorithmes et programme 4](#_Toc472269348)

[1) Les objets 4](#_Toc472269349)

[a) Pays 4](#_Toc472269350)

[b) Joueur 4](#_Toc472269351)

[2) Les fonctions 5](#_Toc472269352)

[a) Fonction « AfficherConditionVictoire » 5](#_Toc472269353)

[b) Fonction « CreationPays » 5](#_Toc472269354)

[c) Fonctions « Afrique », « AmeriqueNord », etc… 6](#_Toc472269355)

[d) Fonction « PlacementArmees » 6](#_Toc472269356)

[e) Fonction « renforts » 7](#_Toc472269357)

[f) Fonction « PlacementRenfort » 8](#_Toc472269358)

[g) Fonction « combat » 8](#_Toc472269359)

[h) Fonction « DeplacementTroupes » 9](#_Toc472269360)

[i) Fonction « verifiervictoire1 », « verifiervictoire2 », etc… 10](#_Toc472269361)

[j) Fonction « VictoireFinale » 11](#_Toc472269362)

[3) Programme principal 11](#_Toc472269363)

[4. Conclusion 12](#_Toc472269364)

# 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. Nous avons décidé de partir sur une application ludique à travers la réalisation d’un programme permettant de jouer au fameux jeu de plateau « Risk ».

# Le jeu « Risk »

## Présentation historique du jeu

Ce jeu a été créé en 1957 par le français Albert Lamorisse tout d’abord sous le nom de « Conquête du monde ». Le jeu sera remanié la même année par Jean-René Vernes pour le rendre plus rapide et plus ludique et prendra alors le nom de « **Risk ». Il** est aujourd'hui édité par Parker, une société du groupe Hasbro.

## 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.

Les joueurs vont ensuite devoir s’affronter pour la domination d’un certain nombre de territoires à l’aide de pions représentant des armées. L’issu des combats étant déterminée par des jets de dés à 6 faces.

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.

Il existe dans la version de base 14 conditions de victoire différentes.

Exemples :

* Vous devez conquérir 18 territoires et occuper chacun d'eux avec deux armées au moins
* 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, 30 à 4, 25 à 5 et 20 à 6). Le joueur doit ensuite répartir ses pions sur ses territoires. Attention tout territoire doit obligatoirement être occupé par au moins 1 pion.

Dès que tout les joueurs ont placé leurs pions la partie peut vraiment commencer.

**Tour de jeu :**

*Renforts*

Au début de son tour chaque joueur se voit attribué un certain nombre de pions en renfort. Le joueur obtient autant d'armées qu'il a de multiples de 3 territoires sur le planisphère sans distinction de continents (arrondi à l’inférieur). Le joueur gagne des renforts supplémentaires s'il possède tous les territoires d'un continent (Afrique : 3, Amérique du Nord : 5, Amérique du Sud : 2, Asie : 7, Europe : 5, Océanie : 2).

*Combats*

Le joueur peut ensuite décider d’attaquer des territoires à l’aide de ses armées. Pour pouvoir attauquer un pays le joueur doit déplacer ses pions depuis un pays voisin. Le joueur peut attaquer avec maximum 3 pions à la fois et doit toujours laisser au moins 1 pion sur le pays depuis lequel il attaque. Le joueur attaquant jette ensuite autant de dés que d’armées utilisées, le joueur défenseur lui ne jette 2 dés même si il possède plus de 2 armées sur son territoire et 1 dé si il ne possède qu’une armée pour se défendre. On compare ensuite les résultats des jets de dés et on détermine les pertes de la manière suivante :

On commence par comparer le dé le plus fort des 2 côtés :

Si le dé de l’attaquant est stictement supérieur à celui du défenseur ce dernier perd 1 pion. En cas d’égalité ou de dé inférieur de la part de l’attaquant c’est ce dernier qui perd 1 pion.

On regarde ensuite si il y en a le 2ème dé le plus fort des 2 côtés et on applique le même raisonnement.

Lorque 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.

Le joueur est libre d'attaquer dans le même tour successivement et sans ordre établi, d'autres adversaires en d'autre points du planisphère.

*Déplacements*

Qu’il ai attaqué avant ou non le joueur peut ensuite déplacer ses troupes à l’intérieur de ses territoires en respectant le fait que les territoires doivent être adjacents et qu’au moins un pion est laissé sur tout les territoires.

A la fin de cette action on passe au tour du joueur suivant.

Ces actions se répètent jusqu’à ce qu’un des joueurs réalise la condition de victoire qui lui a été attribué et l’annonce.

Nous avons ici présenté les règle de base du jeu. Il existe en effet différentes variantes dans les règles, notamment pour les parties à 2 joueurs. D’autres règles supplémentaires sont venus s’ajouter au fil du temps pour enrichir l’expérience de jeu.

# Algorithmes et programme

## Les objets

Pour notre programme nous avons été amené à créer 2 objets : l’objet « Pays » et l’objet « Joueur ».

### Pays

Cet objet va nous permettre, comme son nom l’indique, de gérer les pays du plateau de jeu. Il possède 5 champs :

|  |
| --- |
| private Joueur joueur;  private int id;  private String nom ;  private int occupepar;  private Pays[] tabdeplacements= new Pays[6]; |

Tableau 1 : champs de l’objet « Pays »

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 tout les pays voisins (pays vers lequel il est possible de déplacer les troupes).

Cet objet est créé à l’aide du constructeur suivant :

|  |
| --- |
| public Pays (int id,String nom,int occupepar ) |

Tableau 2 : constructeur de « Pays »

Le champ joueur est rempli lors de l’attribution aléatoire des pays en début de jeu et la liste des voisins dans la fonction qui va permettre la construction des pays.

Des « getters » et « setters » classiques sont ajoutés afin de modifier le propriétaire d’un pays ou le nombre d’armées occupantun 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 »

### Joueur

Cet objet va nous permettre de gérer les joueurs. Il contient 3 champs :

|  |
| --- |
| private String couleur;  private int nConditionDeVictoire;  private int id; |

Tableau 4 : champs de l’objet « Joueur »

Les champ « 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.

L’objet « Joueur » est créé à l’aide du constructeur suivant :

|  |
| --- |
| public Joueur (String c, int n){  couleur = c;  nConditionDeVictoire = n;  id = ++n; |

Tableau 5 : constructeur de l’objet « Joueur »

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.

## Les fonctions

Afin de clarifier le programme et éviter la répétition d’actions similaires nous avons créé plusieurs fichiers contenant des fonctions.

### Fonction « AfficherConditionVictoire »

|  |
| --- |
| public static void AfficherConditionVictoire(Joueur j1) |

Tableau  : signature de la fonction « AfficherConditionVictoire »

Cette fonction prend donc en paramètre un joueur et affiche à l’écran la condition de victoire qu’il doit réaliser :

|  |
| --- |
| if (j1.getNConditionDeVictoire() == 1){  System.out.println("Joueur " +j1.getCouleur()+" : Vous devez conquerir 18 territoires et occuper chacun d'eux avec deux armees au moins");  } |

Tableau  : affichage de la condition de victoire n°1

### Fonction « CreationPays »

|  |
| --- |
| public static Pays[] CreationPays(Pays[] tabPays) |

Tableau  : 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);  p1.addVoisin(p29);  p1.addVoisin(p32);  p1.addVoisin(p12);  p1.addVoisin(p22); |

Tableau  : exemple de création d’un Pays

### Fonctions « Afrique », « AmeriqueNord », etc…

|  |
| --- |
| public static Pays[] Afrique (Pays[] tabPays) |

Tableau  : signature de la fonction « Afrique »

Cette fonction prend en paramètre le tableau de tout les pays et va retourner un tableau qui contiendra les pays se situant en Afrique.

|  |
| --- |
| Afrique[0]=tabPays[1]; |

Tableau  : remplissage du tableau de l’Afrique

Les autres fonctions des différents continents fonctionnent sur le même principe.

### Fonction « PlacementArmees »

|  |
| --- |
| public static void PlacementArmees(Joueur j1, ArrayList<Pays> TabJoueurPays, Pays[] tabPays, Pays[] Continent1, Pays[] Continent2, Pays[] Continent3, Pays[] Continent4, Pays[] Continent5, Pays[] Continent6) |

Tableau  : signature de la fonction « PlacementArmees »

Cette fonction prend en paramètres un joueur, le tableau des pays qu’il possède, le tableau de tout 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.

Dans cette fonction nous appelons la fonction « AfficherConditionVictoire » pour permettre au joueur de positionner judicieusement ses armées.

|  |
| --- |
| Fonctions.AfficherConditionVictoire(j1); |

Tableau  : appel de la fonction « AfficherConditionVictoire »

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.

|  |
| --- |
| int nbpions=21;  for(i=0;i<14;i++){  TabJoueurPays.get(i).setOccupepar(1);  } |

Tableau  : 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 tout ces pions.

|  |
| --- |
| do {  //menu  }while (nbpions>0); |

Tableau  : répétition du menu

|  |
| --- |
| case 1:  System.out.println("\n\nAfrique:");  for(i=0;i<=5;i++){  System.out.println("Pays: "+Continent1[i].getNom()+" n°: "+Continent1[i].getId()+" Détenu par le joueur :"+Continent1[i].getJoueur().getCouleur()+" Occupé par: "+Continent1[i].getOccupepar()+" armées");  }  …  break; |

Tableau  : choix n°1 affichage de l’état du monde

Pour chaque continent on vient afficher le pays (nom +n°), le joueur le possédant et le nombre d’armées positionnées dessus.

|  |
| --- |
| case 2:  //affichage force joueur (1)  do{ (2)  System.out.println("\nsaisissez le numéro du pays sur lequel vous voulez placer une/des armée(s)");  num=sc.nextInt();  }while(num<1 || num>=43); (3)  if (true==TabJoueurPays.contains(tabPays[num-1])){ (3)  System.out.println("saisissez le nombre de troupe à placer");  nbarme=sc.nextInt();  if(nbarme<=nbpions && nbarme>=0){ (4)  tabPays[num-1].setOccupepar(nbarme+tabPays[num-1].getOccupepar());  nbpions=nbpions-nbarme;  }else  System.out.println("Vous ne disposez pas d'assez de troupes"); (5)  }else  System.out.println("Ce pays n'est pas disponible"); (5)  break; |

Tableau  : choix n°2 placement des armées

Pour ce choix nous commençons par afficher les forces du joueur sur le même principe que dans le tableau n°17 mais avec le tableau des pays du joueur (1) puis dans le (2) 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 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).

### Fonction « renforts »

|  |
| --- |
| public static int renforts (Joueur j1, ArrayList<Pays> TabJoueurPays, Pays[] tabPays, Pays[] Continent1, Pays[] Continent2, Pays[] Continent3, Pays[] Continent4, Pays[] Continent5, Pays[] Continent6) |

Tableau  : signature de la 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 tout 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 = (int)(Math.floor(TabJoueurPays.size()/3)); |

Tableau  : 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 :

|  |
| --- |
| for(i=0;i<7;i++){  if(Continent5[i].getJoueur() == j1){  cptEurope = cptEurope+1;  }  }  if ( cptEurope == 6 ){  pionsdispo += 5;  } |

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

### Fonction « PlacementRenfort »

|  |
| --- |
| public static void PlacementRenfort(int renforts, Joueur j1, ArrayList<Pays> TabJoueurPays, Pays[] tabPays) |

Tableau  : signature de la 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 tout les pays.

Elle sert tout comme la fonction « PlacementArmees » à placer les renforts.

|  |
| --- |
| while (renforts>0){  //actions  } |

Tableau  : tant que le nombre de renforts disponible est supérieur à 0  on répète nos actions

Les actions à répéter sont similaires à la fonction « PlacementArmees » (e)).

### Fonction « combat »

|  |
| --- |
| static int[] combat (int att, int def) |

Tableau  : signature de la 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 2.2)Combats).

|  |
| --- |
| resultatatt[i] =(1+(int)(Math.random() \* ((6-1) +1))); |

Tableau  : 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 :

|  |
| --- |
| for (i = 0; i < jetsatt ; i++){  if(resultatatt[i] > maxatt){  maxatt = resultatatt[i];  }  } |

Tableau  : calcul du dé le plus fort de l’attaquant

On calcul alors les pertes :

|  |
| --- |
| if (maxatt > maxdef){  tabRes[0]= 0;  tabRes[1]= 1;  }  else {  tabRes[0]= 1;  tabRes[1]= 0;  } |

Tableau  : calcul des pertes

Si 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 :

|  |
| --- |
| for (i = 0; i < jetsatt ; i++){  if(resultatatt[i] == maxatt){  resultatatt[i]=0;  maxatt=0;  }  } |

Tableau  : 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.

### Fonction « DeplacementTroupes »

|  |
| --- |
| public static boolean DeplacementTroupes(Joueur j1, ArrayList<Pays> TabJoueurPays, ArrayList<Pays> TabJoueurPays1, ArrayList<Pays> TabJoueurPays2, Pays[] tabPays, Pays[] Continent1, Pays[] Continent2, Pays[] Continent3, Pays[] Continent4, Pays[] Continent5, Pays[] Continent6) |

Tableau  : signature de la fonction « DeplacementTroupes »

Cette fonction prend en paramètre le joueur dont c’est le tour, les tableaux des pays de tout les joueurs, le tableau de tout les pays et les tableaux des continents et renvoie un boolean 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 enemis 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 :

|  |
| --- |
| for(j=0;j<TabJoueurPays.size();j++){  //Affichage du pays (j) du joueur  voisin=TabJoueurPays.get(j).getDeplacements();  for(i=0;i<voisin.length;i++){  if(voisin[i] != null){  //Affichage des infos sur le pays voisin (i)  }  }  } |

Tableau  : 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 ») :

|  |
| --- |
| if(nbarme<=tabPays[numPays-1].getOccupepar() && nbarme>=0 && nbarme<=3 && ((tabPays[numPays-1].getOccupepar()-nbarme)>=1)){ |

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

On recupère ensuite le numéro du pays à attaquer en vérifiant qu’il est bien voisin du pays attaquant et qu’il apprtient à un autre joueur. Des messsages d’erreurs sont affichés sinon.

:

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

Tableau  : 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 :

|  |
| --- |
| if(tabPays[numPays1-1].getOccupepar()==0) |

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

L’état des troupes est modifié et le pays conquis est suprimé 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]);  for(j=0;j<TabJoueurPays1.size();j++){  if(TabJoueurPays1.get(j).getJoueur().getCouleur() == j1.getCouleur()){  TabJoueurPays1.remove(tabPays[numPays1-1]);  }  } |

Tableau  : 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 ce 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.

|  |
| --- |
| tabPays[numPays-1].setOccupepar(tabPays[numPays-1].getOccupepar()-nbarme);  tabPays[numPays1-1].setOccupepar(tabPays[numPays1-1].getOccupepar()+nbarme); |

Tableau  : résultats d’un déplacement interne

Cette opération de déplacements internes est répétée jusqu’à ce que le joueur signal par une saisie d’un nombre qui sera convertit en booléen qu’il a bien fini cette action. Ma fin de cette opération ou la saisie d’un nombre « hors-menu » entraine le renvoie d’un booléen « true ». Les autres actions renvoient le booleen « false ».

### Fonction « verifiervictoire1 », « verifiervictoire2 », etc…

|  |
| --- |
| public static boolean verifiervictoire1 (Joueur j1, Pays[] tabPays)  public static boolean verifiervictoire2 (Joueur j1, Pays[] AmeriqueNord, Pays[] Afrique)  public static boolean verifiervictoire3 (Joueur j1, Pays[] Afrique, Pays[] AmeriqueNord, Pays[] AmeriqueSud, Pays[] Asie, Pays[] Europe, Pays[] Oceanie) |

Tableau  : signature de la fonction « verifiervictoire1 », « verifiervictoire2 » et « verifiervictoire3 »

Ces fonctions prennent en paramètre le joueur dont on souhaite vérifier si il a rempli sa condition de victoire et soit les tableaux des continents à conquérir ou tout les tableaux des continents selon le type de victoire à vérifier. 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 :

|  |
| --- |
| for(i=0;i<9;i++){  if(AmeriqueNord[i].getJoueur() == j1){  cptAmNord = cptAmNord+1;  }  } |

Tableau  : conquérir l’Amérique du Nord

### Fonction « VictoireFinale »

|  |
| --- |
| public static boolean VictoireFinale (Joueur j1, Pays[] tabPays, Pays[] Afrique, Pays[] AmeriqueNord, Pays[] AmeriqueSud, Pays[] Asie, Pays[] Europe, Pays[] Oceanie) |

Tableau  : signature de la 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.

## Programme principal

Dans ce programme on comence 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  : 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 ensuite aléatoirement les pays aux joueurs en se servant d’un tableau de pays non attribué. Ce tableau associe au n° d’un pays un booléen « false » si le pays est non attribué, « true » sinon. On tire au sort un numéro de pays que l’on attribue au joueur si il est présent dans le tableau des pays non attribué, puis on le retire de ce tableau.

|  |
| --- |
| for(i=0;i<3;i++){  for (j=0;j<nbpays;j++){  test=false;  while(test==false){  y=((int)(Math.random() \* ((41) +1)));  if (tableauAttribution[y]==false){  tabPays[y].setJoueur(tabJoueur[i]);  if(i==0){  Joueur1Pays.add(tabPays[y]);  }else{  …  }  }  tableauAttribution[y]=true;  test=true;  }  }  } |

Tableau  : attribution des pays

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

|  |
| --- |
| for(i=0;i<3;i++){  Fonctions2.PlacementArmees(tabJoueur[i], TabJoueurPays[i], tabPays, Afrique, AmeriqueNord, AmeriqueSud, Asie, Europe, Oceanie);  } |

Tableau  : mise en place des armées

On gère ensuite les tours de jeu :

|  |
| --- |
| do{  for(i=0;i<3;i++){  nouveauxpions=Renfort.renforts(tabJoueur[i], TabJoueurPays[i],tabPays, Afrique, AmeriqueNord, AmeriqueSud, Asie, Europe, Oceanie);  Fonction3.PlacementRenfort(nouveauxpions,tabJoueur[i], TabJoueurPays[i], tabPays);  do {  finTour=Deplacement.DeplacementTroupes(tabJoueur[i], TabJoueurPays[i], TabJoueurPays[j], TabJoueurPays[k],tabPays, Afrique, AmeriqueNord, AmeriqueSud, Asie, Europe, Oceanie);  }while(finTour !=true);  finJeu = Victoire.VictoireFinale(tabJoueur[i], tabPays, Afrique, AmeriqueNord, AmeriqueSud, Asie, Europe, Oceanie);  if(finJeu == true){  break; // on sort de la boucle "for" car un joueur à réalisé sa condition de victoire  }  }  }while(finJeu != true); |

Tableau  : 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 « true ». 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.

Voilà le jeu est fini !!

# Conclusion

Pour réaliser ce projet nous avons utiliser tout ce que nous avons vu en programmation impérative, fonctions, tableaux, structures conditionnelles, boucles…

Ce qui nous a poser le plus de problèmes sont la gestion des tableaux, l’accès aux variables qu’ils contiennent.

Elargissement du projet :

* Modifier le nombre de joueur
* Rajouter des règles, variantes…
* Implémenter le tout sur une interface graphique