

TP6 : La course de tortue

Lien du repository GitHub : <https://github.com/EC-Nantes/tp-note-2-delepine_pourias_guidon.git>

Table des matières

[1.Conception générale 3](#_Toc129887093)

[2.Conception détaillée 5](#_Toc129887094)

[2.1 Partie 5](#_Toc129887095)

[2.1.1 Méthode preparationDeLaPartie 5](#_Toc129887096)

[2.1.1 Méthode preparationDeLaPartie Initialisation 6](#_Toc129887097)

[2.1.1 méthode deroulementPartie 6](#_Toc129887098)

[2.2 Tortue 7](#_Toc129887099)

[2.3 Carte 7](#_Toc129887100)

[2.3.1 Classe mère 7](#_Toc129887101)

[2.3.2 Classes filles 7](#_Toc129887102)

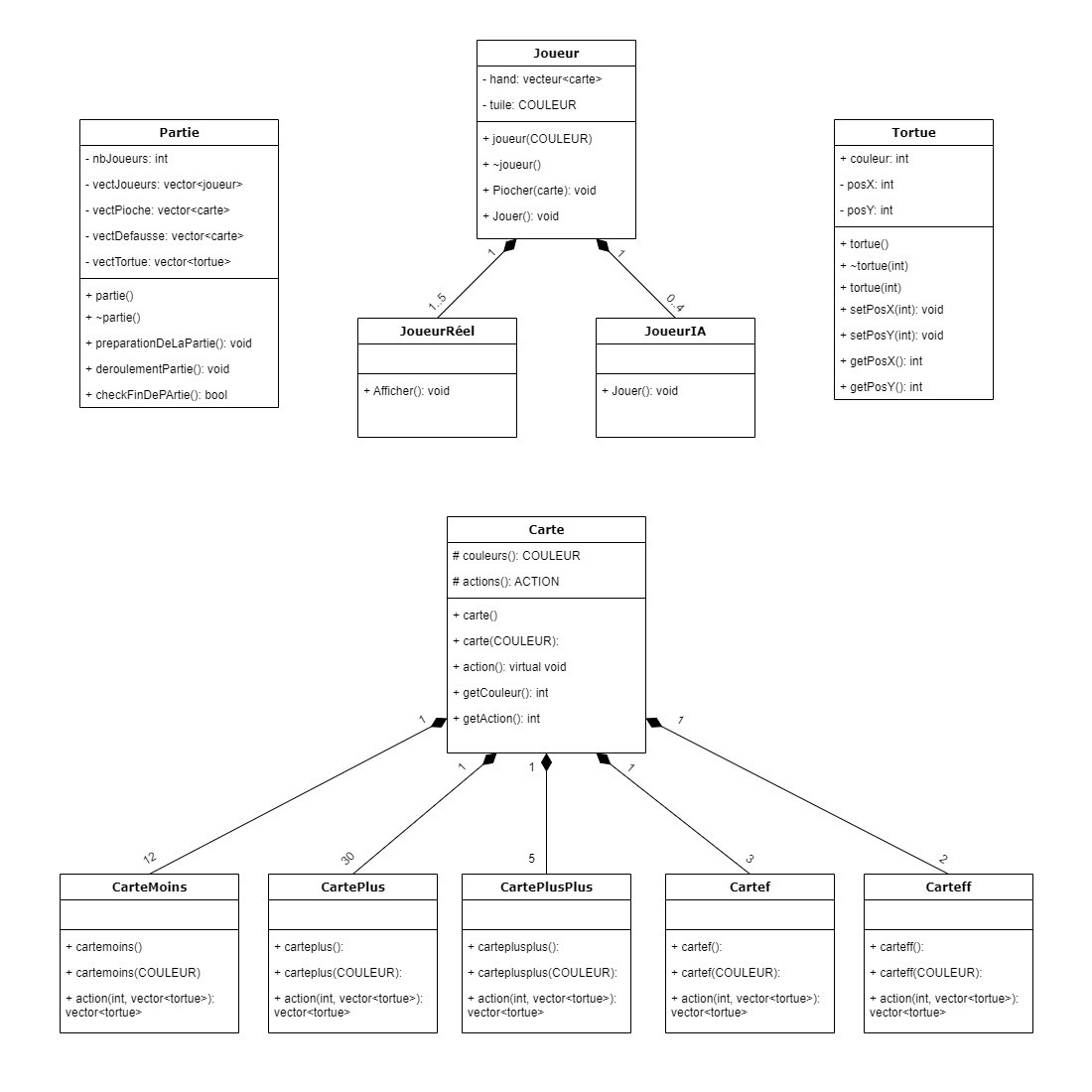
[2.3.2 Tests unitaires 7](#_Toc129887103)

# 1.Conception générale

Pour modéliser le jeu de la course de tortue, on a décidé d’implémenter 4 classes principales:  
 - Joueur,   
 - Carte,   
 - Partie,  
 - Tortue,   
pour matcher le plus possible avec la réalité du jeu.

La classe carte aura 5 classes filles qui correspondant aux différentes cartes existantes dans le jeu.  
Ces classes répondent aux contraintes suivantes :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Classe |  | | | Fonctions à réaliser | Solution |
| Partie | | FP1 | FP1.1 : Initialiser la partie | | Constructeur de partie  Méthode preparationDeLaPartie Attribut nombre de joueurs  Vecteur de joueurs |
| FP1.2 : Déroulement d’un tour | | Méthode deroulementPartie |
| FP1.3 : Gestion de la pioche | | Vecteur de cartes |
| FP1.4 : Gestion de la défausse | | Vecteur de cartes |
| FP1.5 : Appel de l’effet de la carte | | Méthode deroulementPartie |
| FP1.6 : Fin de la partie | | Méthode finDePartie |
| Joueur | | FP2 | FP2.1 : Associer le joueur à sa tortue (tuile) | | Enum COULEUR  Attribut tuile |
| FP2.2 : Gestion de la main | | Méthode piocher  Vecteur de cartes |
| FP2.3 : Choix de carte à jouer (joueur réel) | | Classe fille joueurRéel.h/.cpp  Méthode afficher  Méthode jouer |
| FP2.4 : Choix aléatoire de la carte à jouer (joueur IA) | | Classe fille joueurIA.h/.cpp  Méthode jouer |
| Carte | | FP3 | FP3.1 : Gérer la couleur des cartes | | Enum COULEUR  Attribut couleurs |
| FP3.1 : Gérer les effets des cartes | | Attributs action de chaque classes filles |
| FP3.2 : Gérer la diversité des cartes et de leurs effets | | Classes filles cartes.h |
| FP3.4 : Retourner les modifications à la classe partie | | Vecteur de tortues |
| Tortue | | FP4 | FP4.1 : Associer une tortue à sa couleur | | Enum COULEUR  Attribut couleur |
| FP4.2 : Récupérer la position de la tortue | | Attribut positionX  Attribut positionY  Méthode getCoordoX  Méthode getCoordoY |
| FP4.3 : Modifier la position de la tortue | | Attribut positionX  Attribut positionY  Méthode setCoordoX  Méthode setCoordoY |



*Diagramme de classe de la conception du projet*

# 2.Conception détaillée

Pour commencer, nous avons écrit un fichier ENUM pour simplifier le travail avec les couleurs, qui sont grâce à ce fichier manipulable comme des int ; mais également avec les actions de chaque cartes, qui sont instanciées comme des strings.

## 2.1 Partie

### 2.1.1 Méthode preparationDeLaPartie

Variables :

nbJoueurs : entier

vectJoueurs : tableau de joueurs

vectPioche : tableau de cartes

vectDefausse : tableau de cartes

vectTortue : tableau de tortues

Traitement :

fonction preparationDeLaPartie()

Afficher "Préparation de la partie"

Random\_device rd

mt19937 g(rd())

Mélanger les éléments de vectPioche avec shuffle(vectPioche.begin(), vectPioche.end(), g)

Répéter

Afficher "Entrez le nombre de joueurs (entre 2 et 5) : "

Lire nbJoueurs

jusqu'à ce que nbJoueurs soit compris entre 2 et 5 inclus

Afficher "Vous avez choisi " + nbJoueurs + " joueurs."

Pour i allant de 1 à nbJoueurs

vectJoueurs.ajouter(joueur(COULEUR(i)))

fin pour

Mélanger les éléments de vectJoueurs avec shuffle(vectJoueurs.begin(), vectJoueurs.end(), g)

Pour i allant de 1 à nbJoueurs

vectTortue.ajouter(tortue(i))

fin pour

Pour i allant de 0 à nbJoueurs-1

Pour j allant de 0 à 4

Afficher "Le joueur avec la tuile " + COULEUR\_NOM[vectJoueurs[i].getTuile()] + " pioche une carte"

vectJoueurs[i].Piocher(vectPioche.back())

vectPioche.retirer(vectPioche.back())

fin pour

fin pour

Afficher "Partie prête"

fin fonction

### 2.1.1 Méthode preparationDeLaPartie Initialisation

nbTour = 0

i = 0

Traitement :

Afficher "Lancement de la partie"

Tant que checkFinPartie() est vrai

i = nbTour % (nbJoueurs +1)

Afficher "+ Le joueur avec la tuile " + COULEUR\_NOM[vectJoueurs[i].getTuile()] + " joue"

vectDefausse.ajouter(vectJoueurs[i].Jouer())

vectDefausse.back().action()

Afficher "La carte est défaussée"

vectJoueurs[i].Piocher(vectPioche.back())

vectPioche.retirer(vectPioche.back())

nbTour++

fin tant que

Afficher "Fin de la partie"

fin fonction

### 2.1.1 méthode deroulementPartie

Traitement :

Fonction checkFinPartie() -> bool

finPartie <- vrai

Pour chaque tortue dans vectTortue

Afficher "La tortue " + COULEUR\_NOM[tortue.couleur] + " est sur la case " + tortue.getPosX() + " en étant à la position " + tortue.getPosY()

Si tortue.getPosX() == 10 et tortue.getPosY() == 0

Afficher "La tortue " + COULEUR\_NOM[tortue.couleur] + " a gagné !!!"

finPartie <- faux

Retourner finPartie

Fin fonction

## 2.2 Tortue

Cette classe simple nous permet d’instancier les 5 tortues à chaque début de partie en leur attribuant à chacune une couleur, et aussi de travailler avec des vecteurs de type tortue. Par la suite, nous pouvons éditer sa position.

L’attribut x de la tortue définit sa position de 1 à 10 sur les 10 cases du plateau, tandis que son attribut y définit sa place sur une des cases. Si la tortue rouge est seule sur la case n°2. Ses attributs seront (x=2 ; y=0).

Si 2 tortues se trouvent sur la même case, celle qui se trouve en dessous aura pour attribut y = 0 et celle qui est au dessus aura y = 1. Y peut donc varier entre 0 et 4.

## 2.3 Carte

### 2.3.1 Classe mère

Nous avons décidé d’instancier une méthode abstraite (action) dans la classe mère, qui est redéfinie dans chacune des classes filles pour pouvoir définir les actions des 5 différents types de cartes.

Avec du recul, il aurait été plus simple d’écrire une partie du code de la méthode action dans la classe mère et de ne redéfinir que la partie de code qui diffère suivant les actions effectuées dans chacune des classes filles. Cela nous aurait évité de réécrire une méthode presque identique dans chacune des 5 classes filles.

### 2.3.2 Classes filles

Toutes les classes filles ainsi que leurs méthodes ont été écrites dans le même fichier : cartes.h.

Hormis les constructeurs, elles ne possèdent qu’une méthode, la méthode action. Cette méthode prend en argument le numéro de la tortue sur laquelle elle doit agir ainsi que le vecteur contenant les 5 tortues.

Les méthodes permettent d’exécuter les différentes actions sur les tortues souhaitée, ainsi que sur les tortues qui se trouveraient sur la même case que cette tortue, et au-dessus d’elle.

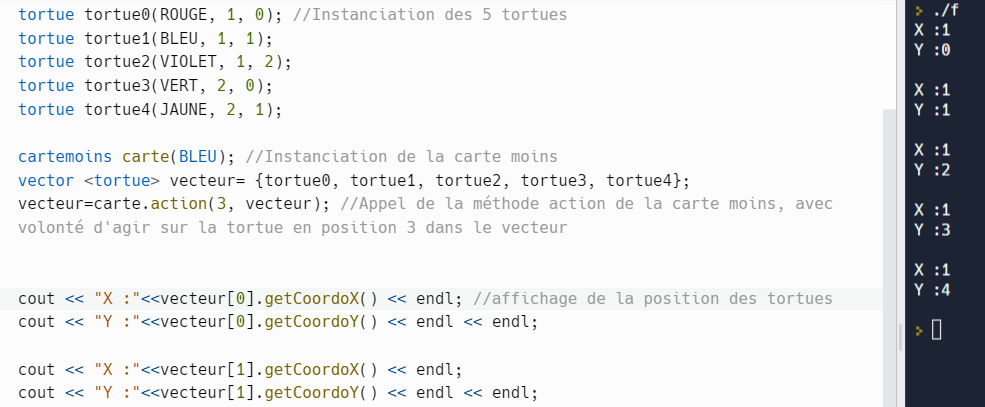
Une fois l’action effectuée, la méthode renvoie ce même vecteur de tortues, qui contient les tortues avec leur position mise à jour.

### 2.3.2 Tests unitaires

Des tests unitaires ont été réalisés pour tester le bon fonctionnement des méthodes actions de toutes les cartes.

Tous les cas de figures ont été testés sur les 5 différentes cartes. La modification de la position en y et en x est bien fonctionnelle.

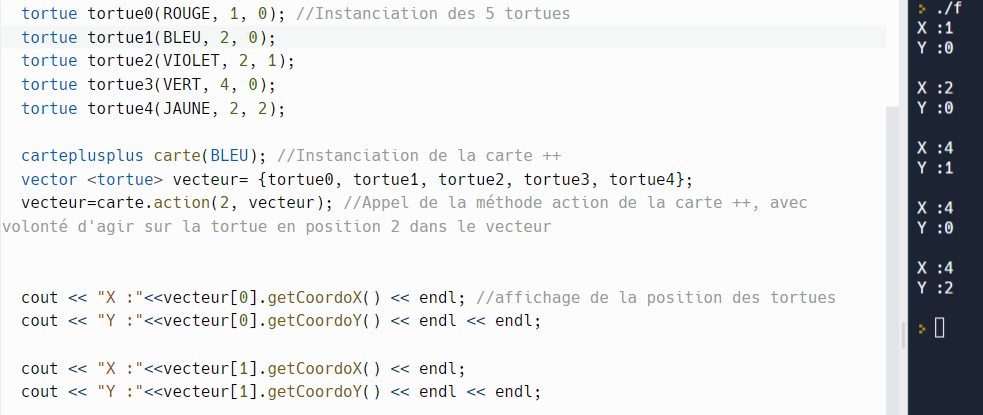
On teste par exemple le bon fonctionnement de la carte moins :



On remarque que la tortue 3 a bien reculée d’une case, et s’est positionnée au-dessus des tortues déjà présentes sur la case (y=3).

La tortue 4, qui était initialement au-dessus de la tortue 3 a également vu sa position être modifié. Elle a reculé d’une case et se place au-dessus de la tortue 3 (y=4).

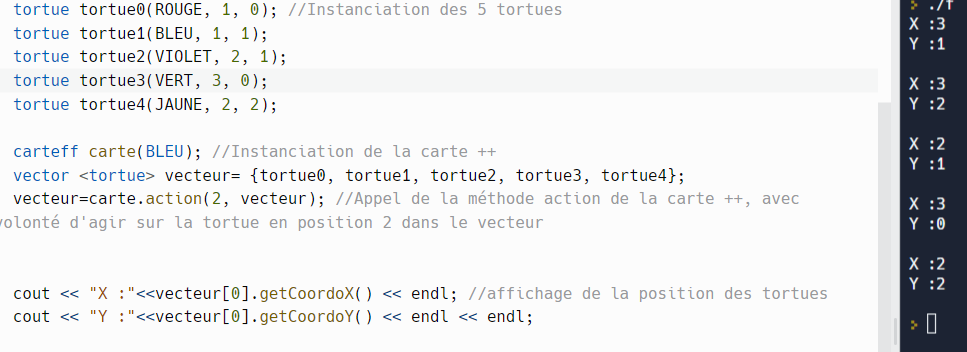
Test du même style avec l’utilisation de la carte plusplus :



On voit que la carte 2 a bien été déplacé de 2 cases vers l’avant (x=4). La tortue 4 qui se trouvait au-dessus l’a suivie et elles se sont toutes les deux placées au-dessus de la tortue3 qui était déjà en position 4.

Quant à la tortue1 qui se trouvait en dessous de la tortue 3, elle a bien gardé sa position.

La carte ff (flèche flèche) fonctionne également :



Elle fait bien avancer de 2 cases les deux dernières tortues, qui se placent au-dessus de la tortue déjà présente sur la case.