Cédric Ponçot
Nicolas Courvoisier

Rapport du projet chasse au trésor

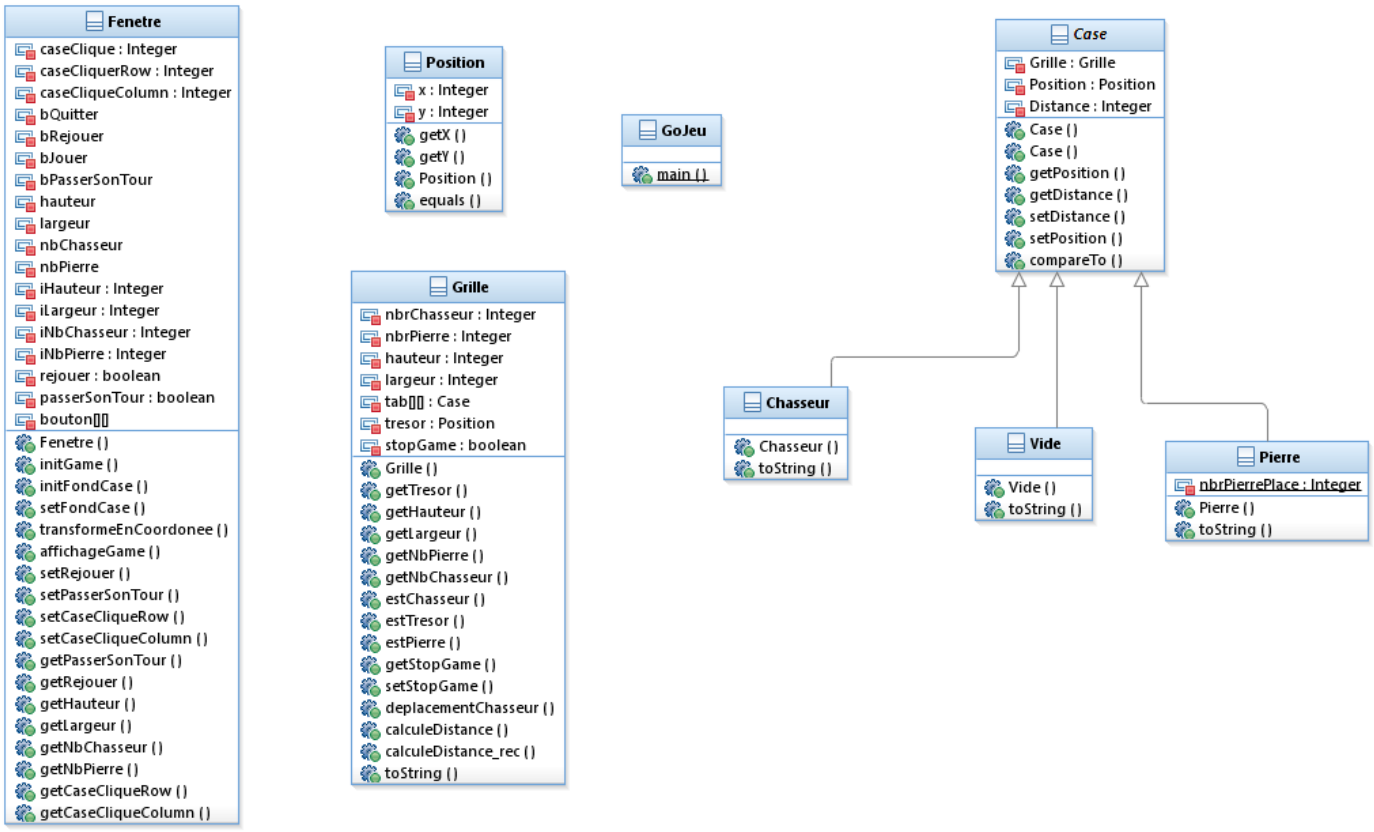
Table des matières

1- Résumé de l'application :

L'application est un jeu nommé chasse au trésor, où des chasseurs cherchent un trésor caché sur une île déserte. Le but du jeu est d'empêcher les chasseurs de trouver le trésor, pour cela le joueur va pouvoir poser et déplacer des pierres dans un nombre limité déterminé en début de partie. Les chasseurs peuvent se tromper dans leur direction, le choix d'un chasseur est fiable à 75% donc dans 25% des cas il n'est pas fiable.

Le joueur ne peut pas déplacer les pierres. Au début de partie, il doit appuyer sur entrée après avoir saisie chaque valeur dans les paramètres. Ensuite la partie se lance, le joueur peut alors cliquer sur les cases pour placer une pierre, passer son tour ou encore quitter la partie. Un message de fin de partie s'affiche quand le joueur a perdu ou gagné et propose de quitter ou de rejouer.

2- Diagramme de classe :



3- Explications des classes :

-Case :

Attribut : distance (distance de la case jusqu'au trésor)
position (position de la case actuelle)
grille (grille sur laquelle nous sommes en train de jouer)
Constructeur : Public Case(int x, int y, int distance, Grille grille);
Public Case(int x, int y, Grille grille);
Prototype : public Position getPosition(); (retourne la position de la case)
public int getDistance(); (retourne la distance de la case au trésor)
public void setDistance(int distance) ; (met une distance)
public void setPosition(int x, int y) ; (ajoute une position)
public int compareTo(Case o); (compare la distance au trésor)

La classe Case est abstraite et implémente Comparable avec 'Case implements Comparable<Case>', cela permettra de comparer la distance de la case jusqu'au trésor en utilisant la fonction compareTo.

-Chasseur :

Attribut : il n'y en a pas.

Constructeur : public Chasseur(int x,int y,Grille grille); (on utilisera ici le mot clé super pour appeler le constructeur de la classe étendu).

Prototype : public String toString(); (retourne le caractere du chasseur sur la grille, ici X).

La classe Chasseur étend de la classe Case 'Chasseur extends Case', on a fait ce choix car une case peut être occupé par un chasseur.

-Vide :

Attribut : il n'y en a pas.

Constructeur : public Vide(int x,int y,Grille g); (on utilisera ici le mot clé super pour appeler le constructeur de la classe étendu).

Prototype : public String toString(); (retourne le caractere du chasseur sur la grille, ici le caractère vide).

La classe Vide étend de la classe Case, c'est case n'est donc pas occupé puisqu'elle est vide.

-Pierre :

Attribut : nbrPierrePlace (nombre de pierre placé sur la grille de jeu, attribut statique)

Constructeur : public Pierre(int x,int y,Grille g); (on utilisera ici le mot clé super pour appeler le constructeur de la classe étendu).

Prototype : public String toString(); (retourne le caractere du chasseur sur la grille, ici le caractère vide).

-Position :

Attribut : x (position x de la case)
Y (position en y de la case)

Constructeur : public Position(int x, int y) ;

Prototype : public int getX(); (retourne la position en X)
public int getY(); (retourne la position en Y)
public boolean equals(Position o); (regarde si une position est égale à une autre)

La classe position permet d'avoir une position sur la grille.

-Grille :

Attribut : nbrChasseur (nombre de chasseurs maximum dans la partie)
nbrPierre (nombre de pierres maximum dans la partie)
hauteur (hauteur de la grille)
largeur (largeur de la grille)
tab[][] (tableau à deux dimensions de case)
tresor (position du trésor)
stopGame (booléen pour déterminer la fin d'une partie)

Constructeur : public Grille(int nbrChasseur,int nbrPierre,int hauteur,int largeur);

Prototype : public Position getTresor(); (retourne la position du trésor)

public int getHauteur(); (retourne la hauteur de la grille)
public int getLargeur(); (retourne la largeur de la grille)
public int getNbrChassuer(); (retourne le nombre de chasseur de la grille)
public int getNbrPierre(); (retourne le nombre de pierre de la grille)
public boolean estPierre(int x,int y); (regarde si la case est une pierre)
public boolean estTresor(int x,int y); (regarde si la case est le trésor)
public boolean estChasseur(int x,int y); (regarde si la case est un chasseur)
public int getStopGame(); (retourne si oui ou non le jeu est finit)
public int setStopGame(); (modifie la valeur de stopGame)
public void deplacementChasseur(int x, int y); (déplace un chasseur)
public void calculDistance(); (calcule la distance jusqu'au trésor)
public void calculDistance_rec(int x,int y,int distance); (calcule récurssivement la distance)
public String toString(); (retourne une chaine de caractère pour afficher la grille)

La classe grille gère les contraintes de la grille (sortie, trouver le trésor…), génère aléatoirement le trésor et initialise les autres cases à des cases vides.

-GoJeu :

La classe GoJeu contient la méthode main qui permet de lancer le jeu. Elle permet aussi l'initialisation de la partie en demandant à l'utilisateur les dimensions de jeu et le nombre de chasseurs.

-Fenetre :

Attribut : caseClique (recupère le numéro de la case cliqué par le joueur)
caseCliqueRow (coordonnée de la ligne cliqué)
caseCliqueColumn (coordonnée de la colonne cliqué)
bQuitter (bouton quitter, type JButton)
bPasserSonTour (bouton option, type JButton)
bRejouer (bouton rejouer, type JButton)
bJouer (bouton jouer, type JButton)
hauteur (hauteur de la grille saisi par le joueur, type JTextField)
largeur (largeur de la grille saisi par le joueur, type JTextField)
nbChasseur (nombre de chasseur saisi par le joueur, type JTextField)
nbPierre (nombre de pierre saisi par le joueur, type JTextField)
iHauteur (hauteur saisi par le joueur converti en type entier)
iLargeur (largeur saisi par le joueur converti en type entier)
iNbChasseur (nombre de chasseur saisi par le joueur converti en type entier)
iNbPierre (nombre de pierre saisi par le joueur converti en type entier)
rejouer (booléen déterminant si le joueur rejoue ou non)
passerSonTour (booléen déterminant si le joueur passe son tour ou non)
bouton[][] (tableau de bouton pour la grille de jeu, type JButton)
Constructeur : public fenetre();
Prototype : public void initGame(); (demande les informations pour créer la fenêtre de jeu)
public void initFondCase(int height, int width, int caseCliqueRow, int caseCliqueColumn, Grille gr); (initialise la grille de jeu)
public void setFondCase(int width, int height, Grille gr); (met à jour la grille, avec les déplacement des chasseurs et les pierres)
public void transformeEnCoordonnee(int nb, int height); (reçoit la case clique et transforme celle-ci en coordonnée)
public void affichageGagne(boolean gagne); (affiche un message si le joueur a gagné ou perdu)

public void setRejouer(boolean rejouer); (met a jour la valeur de la variable rejouer)
public boolean getPasserSonTour(); (retourne la valeur de passerSonTour)
public boolean getRejouer(); (retourne la valeur de rejouer)
public int getHauteur(); (retourne la valeur de hauteur)
public int getLargeur(); (retourne la valeur de largeur)
public int getNbChasseur(); (retourne la valeur de nbChasseur)
public int getNbPierre(); (retourne la valeur de nbPierre)
public int getCaseCliqueRow(); (retourne la valeur de caseCliqueRow)
public int getCaseCliqueColumn(); (retourne la valeur de caseCliqueColumn)
public boolean setPasserSonTour(boolean passerSonTour); (met à jour passer passerSonTour)
public int setCaseCliqueRow(int caseCliqueRow); (met à jour caseCliqueRow)
public int setCaseCliqueColumn(int caseCliqueColumn); (met à jour caseCliqueColumn)

Structure de données et justification :

Toutes nos données sont de simples variables, mise à part la grille qui elle est un tableau à deux dimensions de cases et une case est identifié par une position x et y.

Nous avons créé une classe positon pour pouvoir se repérer dans la grille.

La grille est un tableau à deux dimensions car nous sommes dans un espace à deux dimensions.

Toutes les autres données sont stockées dans des variables simples.

Pour la partie graphique nous avons un tableau de bouton pour détecter les cliques et des zones de saisis pour simplifier le code et l'usage par l'utilisateur.

Algorithme intéressant et capture d'écran :

Calcule des distances :

```
/**
* Applique a une case de faÃ§on rÃ©cursive sa distance actuelle en fonction des pierres posÃ©es.
* @param int x: CoordonnÃ©e en x de la case.
* @param int y: CoordonnÃ©e en y de la case.
* @param int distance: Distance de la case.
*/
public void calculDistance_rec(int x,int y,int distance){
  if(distance<tab[x][y].getDistance()){
    tab[x][y].setDistance(distance);
    boolean gauche=true;
    boolean droite=true;
    boolean haut=true;
    boolean bas=true;

    if(x==0||estPierre(x-1,y))gauche=false;
    if(x==largeur-1||estPierre(x+1,y))droite=false;
    if(y==0||estPierre(x,y-1))bas=false;
    if(y==hauteur-1||estPierre(x,y+1))haut=false;

    if(gauche)calculDistance_rec(x-1,y,distance+1);
    if(droite)calculDistance_rec(x+1,y,distance+1);
    if(bas)calculDistance_rec(x,y-1,distance+1);
    if(haut)calculDistance_rec(x,y+1,distance+1);
  }
}

/**
* Appelle la fonction calculDistance_rec apres avoir initialisee toutes les cases a  hauteur*largeur
*/
public void calculDistance(){
  for(int i=0;i<largeur;i++){
    for(int j=0;j<hauteur;j++){
      if(!estPierre(i,j)){
        tab[i][j].setDistance(hauteur*largeur);
      }
    }
  }
  calculDistance_rec(getTresor().getX(),getTresor().getY(),0);
}
```

Itérateur :

```
//DEPLACEMENT DES CHASSEURS
Iterator<Chasseur> it=chasseurs.iterator();
int nbrChasseurBloque=0;
while(it.hasNext()){
  Chasseur c=it.next();
  if(c.getDistance()<hauteur*largeur){
    g.deplacementChasseur(c.getPosition().getX(),c.getPosition().getY());
    g.calculDistance();
  }else{
    nbrChasseurBloque++;
  }
  //On marque une seconde d'attente entre le deplacement des chasseurs
  try{
    Thread.sleep(1000);
  }
  catch(InterruptedException ex){
    Thread.currentThread().interrupt();
  }
  System.out.println(g);
  f.setFondCase(largeur, hauteur, g);
}
```

4- Conclusion bilan :

Sur le produit :

L'application est terminé et jouable, toutes les fonctionnalités demandées ont été traités. Il y a un trésor posé aléatoirement sur une grille à paramètre évoluant suivant les données saisies par le joueur en début de partie. On peut rejouer sans relancer l'application, l'utilisateur rejoue et saisi à nouveau les paramètre pour une nouvelle partie. On peut quitter à tout moment la partie grâce au bouton quitter et on est informé via divers message si on a perdu, gagné et du nombre de pierres restantes, on peut aussi passer notre tour si l'utilisateur le souhaite.

Sur les outils utilisés :

Pour la partie graphique, nous avons utilisés la collection swing et awt, la classe Fenetre est étendu de JFrame.

Pour le déplacement des chasseurs, ils sont dans une PriorityQueue, appartenant à la collection util de java, cela permet de déterminer quel chasseur va se déplacer en premier (celui le plus éloigné du trésor).

La classe Case implémente Comparable pour pouvoir créer la méthode compareTo() qui détermine la distance au trésor.

Nous avons aussi utilisé un itérateur pour accéder à tous les chasseurs et réaliser les calculs et déplacement possible, (Voir photo).

Améliorations possibles :

Pour améliorer l'application, on pourrait rajouter un menu expliquant les règles du jeu, réaliser des cartes qui se génèrent aléatoirement (sans demander de saisi au joueur, comme une génération automatique). Egalement, quand la partie est terminé et que le joueur veut rejouer, on pourrait réaliser un menu qui demande si on conserve les paramètres de jeu actuel ou si le joueur veut en saisir de nouveaux.

Aussi, le joueur ne peut pas déplacer les pierres.

On pourrait aussi, créer une succession de cartes aléatoires (genre 10) où le joueur gagnerait des vies quand il empêcherait les chasseurs de trouver le trésor et le joueur gagnerait s'il parvient à protéger tous les trésors et ayant encore une vie sinon le jeu se terminerait sur une défaite.

Sur le travail de groupe :

Cédric a principalement travaillé sur la partie algorithmique du projet, correctif de bug et teste de l'application.

Nicolas a principalement travaillé sur la partie graphique, teste de l'application et la rédaction du rapport.

Nous avons tous les deux réalisés la JavaDoc.