TP 11 noté - Ricochet Robots

Contributeurs

- Jack HASCOËT
- Natha ANDRE
- Eugénie ROQUAIN
- Eloi TOURANGIN

Professeur

- Myriam Servieres

Table des matières

Explication des règles

Objectif du jeu

Ricochet Robots est un jeu de plateau où 4 robots de couleurs différentes (rouge, jaune, bleu, vert) se déplacent sur une grille de 16x16 cases. Le but est de déplacer les robots pour amener l'un d'eux sur une case cible spécifique en respectant les règles de déplacement. Le joueur qui trouve la solution en utilisant le moins de mouvements remporte la manche.

Éléments du jeu

- **Robots** : 4 robots de couleurs différentes.
- **Objectifs** : 17 tuiles cibles réparties en 4 groupes de 4 tuiles de couleur identique à celle des robots, et une tuile multicolore.
- **Plateau** : Une grille de 16x16 cases avec des murs et obstacles.
- **Sablier** : Un sablier d'une heure pour limiter le temps de réflexion.

Déroulement d'un tour

1. **Tirage de la tuile objectif** :
   Le plateau ainsi que la tuile objectif sont affiché. Si c'est une tuile de couleur, le but est d'amener le robot de cette couleur sur la case cible correspondante. Si c'est la tuile multicolore, n'importe quel robot peut atteindre la case multicolore.

2. **Réflexion simultanée** :
   Tous les joueurs réfléchissent en même temps pour trouver une solution en utilisant le moins de mouvements possible.

3. **Annonce des solutions** :
   Lorsqu'un joueur trouve une solution, il appui sur espace et rentre le nombre de mouvements nécessaires. Les autres joueurs ont une minute pour proposer de meilleures solutions.

4. **Validation par déplacement** :
   Le joueur ayant proposé la solution avec le moins de mouvements montre sa solution. Si elle est correcte, il remporte la partie. Sinon, le joueur suivant (ayant proposé le nombre de mouvements immédiatement supérieur) montre sa solution, et ainsi de suite.

Règles de déplacement

- Les robots se déplacent en ligne droite et avancent jusqu'à rencontrer un obstacle.
- Les obstacles peuvent être :

    - Les bords du plateau.
    - Les murs présents sur le plateau.
    - Un autre robot.
- Une fois en mouvement, un robot ne peut s'arrêter ou changer de direction avant de rencontrer un obstacle.
- Chaque déplacement compte pour un mouvement, quel que soit le nombre de cases parcourues.

Remarque

Si une solution est atteignable en un seul mouvement après le tirage d'une tuile objectif, les joueurs doivent ignorer cette solution et chercher une alternative.

Build & Compilation

Build

Pour construire le projet, exécutez les commandes suivantes dans un terminal :

```
mkdir build
cd build
cmake ..
```

Lancer le jeu

A la racine du projet, dans un terminale

```
make
```

Lancer le jeu de test

A la racine du projet, dans un terminale

```
cd ./build/
ctest
```

Compilation

Compiler et lancer le jeu

```
make
```

Classe Board

La classe `Board` représente une grille de jeu de 16x16 cases utilisée pour un jeu de type puzzle. Elle contient des méthodes pour générer la grille, placer des murs, des angles, des robots, et des cibles.

Fonctionnement classe Board

1. **Attributs principaux :**
    - `board[16][16]` : Tableau 2D de cases représentant la grille de jeu. Chaque case peut contenir des murs, un robot, ou une cible.
2. **Méthodes privées :**
    - `generateBoardStep1` : Ajoute les murs extérieurs de la grille et les murs formant un carré central.
    - `generateBoardStep2` : Ajoute deux murs extérieurs aléatoires (un vertical et un horizontal) dans chaque quart de la grille.
    - `generateBoardStep3` : Ajoute 4 "angles" (deux murs formant un coin) dans chaque quart de la grille, en s'assurant qu'ils ne touchent pas d'autres murs ou angles.
    - `generateBoardStep4` : Ajoute un angle supplémentaire dans un quart choisi aléatoirement, en respectant les mêmes contraintes.
3. **Méthodes publiques :**
    - `generateBoard` : Génère la grille complète en appelant les étapes 1 à 4.
    - `placeRobots` : Place aléatoirement 4 robots sur la grille, ainsi qu'une cible dans un angle de deux murs.
    - `getBoard` : Retourne la grille de jeu.
    - Constructeur et destructeur : Initialisent et nettoient la grille.

Algorithmes classe Board

Voici l'algorithme en langage naturel pour générer une grille complète et gérer les fonctionnalités de la classe `Board` :

```
type Case
    entier nord
    entier sud
    entier est
    entier ouest
    lien Robot robot
    lien Target cible
fin type

type Board
    tableau[16][16] de Case grille
    tableau[16][16] de Case grilleInitiale
    tableau dynamique de paires (entier, entier) anglesCoordinates
    Target objectifCible
fin type
```

Génération de la grille

```
fonction ø <- generateBoard(B)
algorithme
    appeler generateBoardStep1(B)
    appeler generateBoardStep2(B)
    appeler generateBoardStep3(B)
    appeler generateBoardStep4(B)
fin fonction
```

```
fonction ø <- generateBoardStep1(B)
    paramètre lien Board B
    résultat ø
algorithme
    pour x allant de 0 à 15 faire
        B.grille[x][0].nord <- 1
        B.grille[x][15].sud <- 1
    fin pour

    pour y allant de 0 à 15 faire
        B.grille[0][y].ouest <- 1
        B.grille[15][y].est <- 1
    fin pour

    B.grille[7][7].nord <- 1
    B.grille[7][7].ouest <- 1
    B.grille[7][8].ouest <- 1
    B.grille[7][8].sud <- 1
    B.grille[8][7].nord <- 1
    B.grille[8][7].est <- 1
    B.grille[8][8].est <- 1
    B.grille[8][8].sud <- 1
fin fonction
```

```
fonction ø <- generateBoardStep2(B)
    paramètre lien Board B
    résultat ø
algorithme
    pour chaque quart de la grille faire
        générer x et y aléatoires dans les limites du quart
        si les murs générés ne touchent pas d'autres murs alors
            placer un mur vertical ou horizontal
        fin si
    fin pour
fin fonction
```

```
fonction ø <- generateBoardStep3(B)
    paramètre lien Board B
    résultat ø
algorithme
    pour chaque quart de la grille faire
        compteur angles <- 0
        tant que compteur angles < 4 faire
            générer x et y aléatoires dans les limites du quart
            si la case (x, y) n'est pas occupée par un mur ou un angle
alors
                générer un type d'angle aléatoire
                si l'angle est valide (ne touche pas un autre angle ou mur
extérieur) alors
                    placer l'angle en activant deux murs adjacents
                    ajouter (x, y) à anglesCoordinates
                    incrémenter compteur angles
                fin si
            fin si
        fin tant que
    fin pour
fin fonction
```

```
fonction ø <- generateBoardStep4(B)
    paramètre lien Board B
    résultat ø
algorithme
    répéter
        générer x et y aléatoires dans toute la grille (hors murs
extérieurs)
        si la case (x, y) n'est pas occupée par un mur ou un angle alors
            générer un type d'angle aléatoire
            si l'angle est valide (ne touche pas un autre angle ou mur
extérieur) alors
                placer l'angle en activant deux murs adjacents
                ajouter (x, y) à anglesCoordinates
                arrêter la boucle
            fin si
        fin si
    jusqu'à ce qu'un angle soit placé
fin fonction
```

Placement des robots et des cibles

```
fonction ø <- placeRobots(B, robots)
    paramètre lien Board B
    paramètre tableau[4] de Robot robots
    résultat ø
algorithme
```

```
    compteur robots placés <- 0
    tant que compteur robots placés < 4 faire
        générer x et y aléatoires dans toute la grille (hors carré central)
        si la case (x, y) est libre (pas de robot, pas de cible) alors
            placer un robot sur la case
            mettre à jour ses coordonnées initiales
            incrémenter compteur robots placés
        fin si
    fin tant que
fin fonction
```

```
fonction ø <- placeTargets(B, targets)
    paramètre lien Board B
    paramètre tableau dynamique de Target targets
    résultat ø
algorithme
    pour chaque couleur de cible faire
        pour chaque cible de cette couleur faire
            choisir un angle aléatoire dans anglesCoordinates
            si l'angle est libre (pas de robot, pas de cible) alors
                placer la cible sur la case
            fin si
        fin pour
    fin pour

    si une cible multicolore doit être placée alors
        choisir un angle aléatoire dans anglesCoordinates
        si l'angle est libre alors
            placer la cible multicolore
        fin si
    fin si
fin fonction
```

Déplacement des robots

```
fonction booléen <- moveRobot(B, robot, direction)
    paramètre lien Board B
    paramètre Robot robot
    paramètre caractère direction
    résultat booléen
algorithme
    récupérer les coordonnées actuelles du robot
    tant que le robot peut avancer dans la direction donnée faire
        vérifier les murs, les bords du plateau et les autres robots
        mettre à jour les coordonnées du robot
    fin tant que
    retourner vrai si le robot a bougé, sinon faux
fin fonction
```

Vérification de l'objectif

```
fonction booléen <- targetReached(B, robot)
    paramètre lien Board B
    paramètre Robot robot
    résultat booléen
algorithme
    récupérer la cible sur la case actuelle du robot
    si la cible existe et correspond à l'objectif alors
        retourner vrai
    sinon
        retourner faux
    fin si
fin fonction
```

Sauvegarde et réinitialisation de la grille

```
fonction ø <- saveBoard(B)
    paramètre lien Board B
    résultat ø
algorithme
    copier la grille actuelle dans grilleInitiale
fin fonction
```

```
fonction ø <- reinitBoard(B, robots)
    paramètre lien Board B
    paramètre tableau dynamique de Robot robots
    résultat ø
algorithme
    restaurer grilleInitiale dans grille
    pour chaque robot faire
        réinitialiser ses coordonnées à ses coordonnées initiales
        remettre le robot sur la grille
    fin pour
fin fonction
```

Tests classe Board

Les tests de la classe `Board` permettent de valider les fonctionnalités principales de cette classe, notamment la génération de la grille, le placement des éléments (robots et cibles), les déplacements des robots, et la gestion des objectifs.

1. Test du constructeur (`ConstructorTest`)

- **Objectif** : Vérifier que la grille de jeu est correctement initialisée.
- **Vérifications** :
  - Les murs extérieurs sont correctement placés (les bords de la grille).
  - Les murs formant le carré central sont correctement générés.
  - Les cases sans murs ne contiennent ni robots ni cibles.

2. Test du placement des cibles (`PlaceTargetsTest`)

- **Objectif** : Vérifier que les cibles sont placées correctement sur la grille.
- **Vérifications** :
  - Les cibles sont placées uniquement dans des angles valides (cases avec deux murs adjacents).
  - Le nombre de cibles placées est compris entre 4 (minimum) et 17 (maximum, incluant la cible multicolore).

3. Test du déplacement des robots (`MoveRobotTest`)

- **Objectif** : Vérifier que les robots se déplacent correctement selon les règles du jeu.
- **Vérifications** :
  - Un robot peut se déplacer dans une direction donnée (N, S, E, W) tant qu'il ne rencontre pas :
    - Un mur.
    - Un autre robot.
    - Les bords de la grille.
  - Si un déplacement est possible, les coordonnées du robot sont mises à jour.
  - Si un déplacement n'est pas possible, les coordonnées du robot restent inchangées.

4. Test de sauvegarde et réinitialisation de la grille (`SaveAndReinitBoardTest`)

- **Objectif** : Vérifier que la grille peut être sauvegardée et réinitialisée correctement.
- **Vérifications** :
  - Après avoir sauvegardé l'état initial de la grille, un robot est déplacé.
  - Lors de la réinitialisation, le robot revient à sa position initiale, et la grille retrouve son état sauvegardé.

5. Test de la vérification de l'objectif (`TargetReachedTest`)

- **Objectif** : Vérifier si un robot atteint une cible correspondant à l'objectif.
- **Vérifications** :
  - Si un robot est placé sur une case contenant une cible correspondant à l'objectif (même couleur et forme), la fonction retourne `true`.
  - Sinon, la fonction retourne `false`.

6. Test des méthodes `setTargetObjectif` et `getTargetObjectif` (`TargetObjectifTest`)

- **Objectif** : Vérifier que l'objectif (cible à atteindre) est correctement défini et récupéré.
- **Vérifications** :
  - La méthode `setTargetObjectif` permet de définir une cible comme objectif.
  - La méthode `getTargetObjectif` retourne correctement la cible définie comme objectif.

Classe Display

La classe `Display` sert à générer un affichage visuel en console d'un plateau de jeu, avec une représentation graphique ASCII des murs, des robots et des cibles (les targets). Elle utilise des codes ANSI pour afficher les couleurs et des caractères Unicode pour les formes.

Fonctionnement classe Display

1. **Attributs principaux :**
    - `board[SIZE_BOARD][SIZE_BOARD]` : Tableau 2D de cases représentant la grille de jeu. Chaque case peut contenir des murs, un robot, ou une cible. SIZE_BOARD = int(16)
    - `dispBoard[BOARD_DISP_SIZE][BOARD_DISP_SIZE]` : Affichage visuel du plateau.
    - `DispCaseDir_t` : enum privé pour gérer la direction des cases.
2. **Méthodes privées :**
    - `put_walls` : Parcourt chaque Case pour afficher les murs selon les directions -> Nord, Sud, Est, Ouest.
    - `put_robots` : Vérifie si un robot est présent dans la case. Si oui, récupère sa couleur et sa forme, prépare une chaîne à afficher.
    - `put_targets` : Affichage des cibles.
3. **Méthodes publiques :**
    - `update(Case board[SIZE_BOARD][SIZE_BOARD])` : Met à jour le plateau. Dans l'ordre : 1 - Génère ligne par ligne les caractères ASCII à afficher. 2 - Appelle trois fonctions privées pour : afficher les murs, afficher les robots, afficher les cibles.
    - `print` : Place aléatoirement 4 robots sur la grille, ainsi qu'une cible dans un angle de deux murs.
    - Constructeur et destructeur `Display()`: initialise l'affichage (mise en forme console avec ANSI) et réinitialise les couleurs et efface l'écran.

Algorithmes classe Display

```
fonction ø <- put_walls
    paramètre curCase, x, y
    résultat ø
algorithme
    pour chaque case du plateau
        curCase predn la valeur de la case actuelle
        x et y prennent les coordonnées du plateau

        si curCase est faux
            continuer
        si curcase.getNorth est vrai
            afficher un mur horizontal au coordonnées : dispBoard[x - 1][y]
        si curcase.getSouth
            afficher un mur horizontal au coordonnées : dispBoard[x + 1][y]
        si curCase.getEast est vrai
```

```
            afficher un mur horizontal au coordonnées : dispBoard[x][y + 1]
        si curCase.getWest est vrai
            afficher un mur horizontal au coordonnées : dispBoard[x][y - 1]
        fin si
    fin pour

    initialisation des valeurs booléenne wbot, wtop, wrgt, wlft
    pour chaque case
        to_put prend un espace vide
        si la case = coin supérieur gauche du plateau alors
            wrgt : recherche d'un mur horizontal à droite de ce coin
            wbot : recherche d'un mur vertical en dessous
            si wrgt && wbot alors
                affiche coin complet
            si wbot alors
                Un coin avec seulement un mur vertical
            si wrgt alors
                Un coin avec seulement un mur horizontal
            fin si
        fin si
    fin pour
fin fonction
```

Classe Game

Fonctionnement classe Game

Notre classe Game comprend deux classes : Board et Display. Elle permet d'initialiser le jeu et ainsi démarrer une partie.

Dans notre classe Board on a 4 attributs :

- moveRobot() qui permet d'avancer les pions Robot en ligne jusqu'à un obstacle
- generateBoard() qui permet de générer de facon aléatoire notre plateau de jeu avec des obstacles, les cibles et les pions Robots
- CheckWall(x,y) qui prend en argument les coordonnées de la case et qui vérifie si notre pion robot est face à un mur
- CheckTarget(x,y) qui prend en argument les coordonnées de la case et qui vérifie si la case x,y à un pion robot ou non

La classe Board utilise les enum Colors et Shapes qui ont respectivement chacune des couleurs des cibles et robots ainsi que les formes deux cibles

Dans notre classe Display on a 2 arguments :

- Updates()
- Print()

Notre classe Robot possède deux arguments :

- getShape()

- getColor()
- Robot(Enum Colors) qui lui attribut une couleur
- Robot(Color c, Shape s)

Algorithmes classe Game

La fonction play() est celle qui orchestre le jeu, permet à tel ou tel joueur de jouer, de faire l'attribution des scores et de relancer une tuile.

```
fonction booléen <- play()
    paramètre aucun
    résultat booléen

algorithme
    créer une nouvelle interface d'affichage
    initialiser le plateau temporaire de 16x16 cases
    initialiser les joueurs, robots et cibles
    sauvegarder l’état initial du plateau
    initialiser l’index de la cible à -1

    boucle faire tant que keepPlaying() retourne vrai
        réinitialiser l'affichage des robots et du pseudo
        désactiver les événements clavier

        réinitialiser la position des robots sur le plateau
        incrémenter l’index de la cible
        définir la nouvelle cible sur le plateau
        sauvegarder l’état actuel du plateau

        afficher l’état du plateau
        réinitialiser le nombre de coups des joueurs

        si aucun joueur ne propose de solution alors
            afficher un message d’échec collectif
            retourner vrai
        fin si

        permettre au premier joueur d’annoncer son nombre de coups
        donner aux autres joueurs une minute pour proposer le leur
        ordonner les joueurs par ordre croissant de coups annoncés

        activer la gestion des événements clavier pour les mouvements et la
sélection des robots
        sauvegarder les événements précédents

        mettre à jour l’affichage des robots disponibles

        démarrer la boucle de jeu joueur par joueur
            pour chaque joueur
                initialiser l’état du jeu pour ce joueur
                afficher le plateau actuel
```

```
                remettre les compteurs de mouvement à zéro
                choisir le robot de départ
                afficher l’indicateur de mouvements

                si le joueur a annoncé un nombre de coups
                    afficher à qui c’est le tour

                    tant que le joueur joue
                        attendre un petit délai
                        si un nouvel événement de sélection de robot alors
                            traiter l’événement
                        sinon si un nouvel événement de mouvement alors
                            traiter l’événement
                        fin si

                        si aucun mouvement n’a eu lieu alors
                            continuer à la prochaine itération
                        fin si

                        désactiver le drapeau de rafraîchissement
                        si le joueur a atteint le nombre de coups annoncés
alors
                            continuer à la prochaine itération
                        fin si

                        incrémenter le nombre de coups réalisés
                        mettre à jour l'affichage des coups

                        rafraîchir le plateau

                        si la cible a été atteinte alors
                            si c’est au 1er coup, alors
                                annuler la victoire et réinitialiser la
partie pour ce joueur
                            sinon
                                marquer le joueur comme gagnant
                                mettre fin à son tour
                                mettre à jour son score (2 si exact, 1
sinon)
                                afficher le message de victoire
                        sinon si le joueur a atteint son nombre de coups
sans succès alors
                            afficher le message d’échec
                            mettre fin à son tour
                        fin si
                    fin tant que
                fin si

                si un joueur a gagné alors
                    sortir de la boucle principale
                fin si

                réinitialiser le plateau
            fin pour chaque joueur
```

```
        fin de boucle joueur

    fin faire tant que keepPlaying() est vrai

    afficher le score final
    retourner vrai
fin fonction
```

Classe Robot

La classe `Robot` représente ses caractéristiques : couleur (Color), forme (Shape), coordonnées en 2D (x, y)

Fonctionnement classe Robot

1. **Attributs principaux :**
2. **Méthodes privées :**
    - `color` : couleur du robot
    - `shape` : forme du robot
    - `x` : position en x du robot
    - `y` : position en y du robot
3. **Méthodes publiques :**
    - Constructeur et destructeur : `Robot()` : Crée un robot vert de forme RobotSign, `Robot(Color c)` crée le robot avec la couleur de notre choix.
    - `getColor()`, `getShape()`, `getX()`, `getY()` : Retourne respectivement la couleur, la forme, ses coordonnées en x et y.
    - `setColor(Color)`, `setShape(Shape)`, `setX(int)`, `setY(int)` : Change respectivement la couleur, la forme, ses coordonnées en x et y.