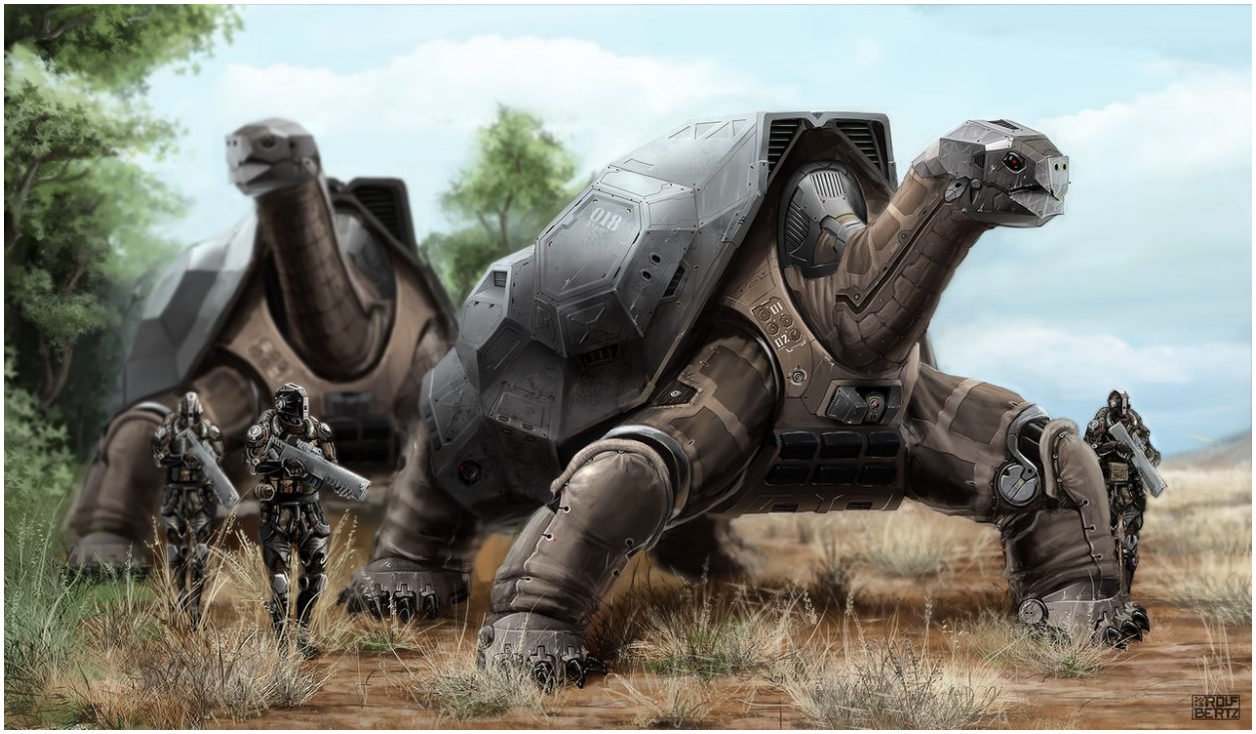
isep
École d'ingénieurs du numérique

Rapport
Projet Robot Turtles

Projet de fin de semestre Algorithmique et Programmation



Rendu pour le mercredi 22 janvier 2020

Alban Erulin, Armand Barral, Alexandre Perbet

Sommaire

Introduction

Lors de ce premier semestre, nous avons découvert le langage Java et la programmation orientée objet. En plus des cours, des travaux dirigés et des travaux pratiques nous avons également eu l'opportunité de développer le projet "Robot Turtles" dont il est question ici.

Dans le cadre de ce projet nous devions recréer le jeu Robot Turtles, les mécaniques de bases du jeu ainsi que des fonctions optionnelles. Nous devions également produire pour ce jeu une interface graphique le rendant jouable pour 2 à 4 joueurs, et enfin un document technique comportant:

- Un bref rappel de ce qui nous est demandé dans ce projet,
- Une description de l'architecture de notre projet et des différents répertoires et fichiers,
- Le diagramme de classe de notre projet,
- Une éventuelle description de l'implémentation d'une fonctionnalité particulière nécessitant d'être mise en avant,
- Une descriptions des bibliothèques externes utilisées.

Ce sont ces derniers éléments qui seront explicités dans la suite de ce rapport.

Description du fonctionnement du projet

Lors du lancement du jeu, on demande le nombre de joueurs, le mode jeu et si les joueurs souhaitent avoir accès à la carte bug. En fonction de la réponse, on initialise d'une part le plateau, les tortues et les joyaux, et d'autre part les attributs des joueurs (nombre de murs de pierre et de glace, éventuellement carte Bug).Chaque joueur dispose d'une pile de cartes dont il tire des éléments.

Une fois la partie créée, on lance effectivement le jeu. Chaque joueur joue à son tour et choisit parmi les actions proposées (utiliser la carte bug, compléter son programme, exécuter le programme ou placer un mur) celle qu'il effectue à ce tour.

Son tour se termine ensuite et il choisit s'il veut défausser tout ou partie de sa main ou la garder pour le prochain tour. Une fois la pile de cartes épuisée, elle est mélangée pour être utilisée au prochain tour du joueur concerné.

Dans le cas d'une partie en mode normal, le jeu se poursuit jusqu'à ce qu'il ne reste plus qu'une tortue n'ayant pas atteint de joyau.

Dans le cas d'une partie en mode "trois à la suite", le jeu se poursuit également jusqu'à ce qu'il ne reste plus qu'une tortue n'ayant pas atteint de joyau. Alors, on réinitialise les piles de deck, les mains et les programmes les positions des tortues et la condition de victoire.

Une fois la manche terminée chaque joueur obtient un nombre de point correspondant au nombre de joueur dont est déduit sa position d'arrivée

Ainsi si on joue a trois joueurs, le premier obtient deux points, le second un points et le derniers n'en obtient pas

On redémarre ensuite une manche et ainsi de suite jusqu'à ce que trois manches aient été jouées; le joueur au score le plus élevé à l'issue gagne la partie.

Manuel d'utilisation du projet

Lancement partie

- La classe principale à compiler est la classe Main
- Une fois l'interface lancée, sélectionnez les animaux à utiliser: un animal différent pour chaque joueur
- Cocher ou non les cases "carte bug" et "mode trois à la suite"
- Appuyer sur "lancer la partie"

Déroulement de la partie

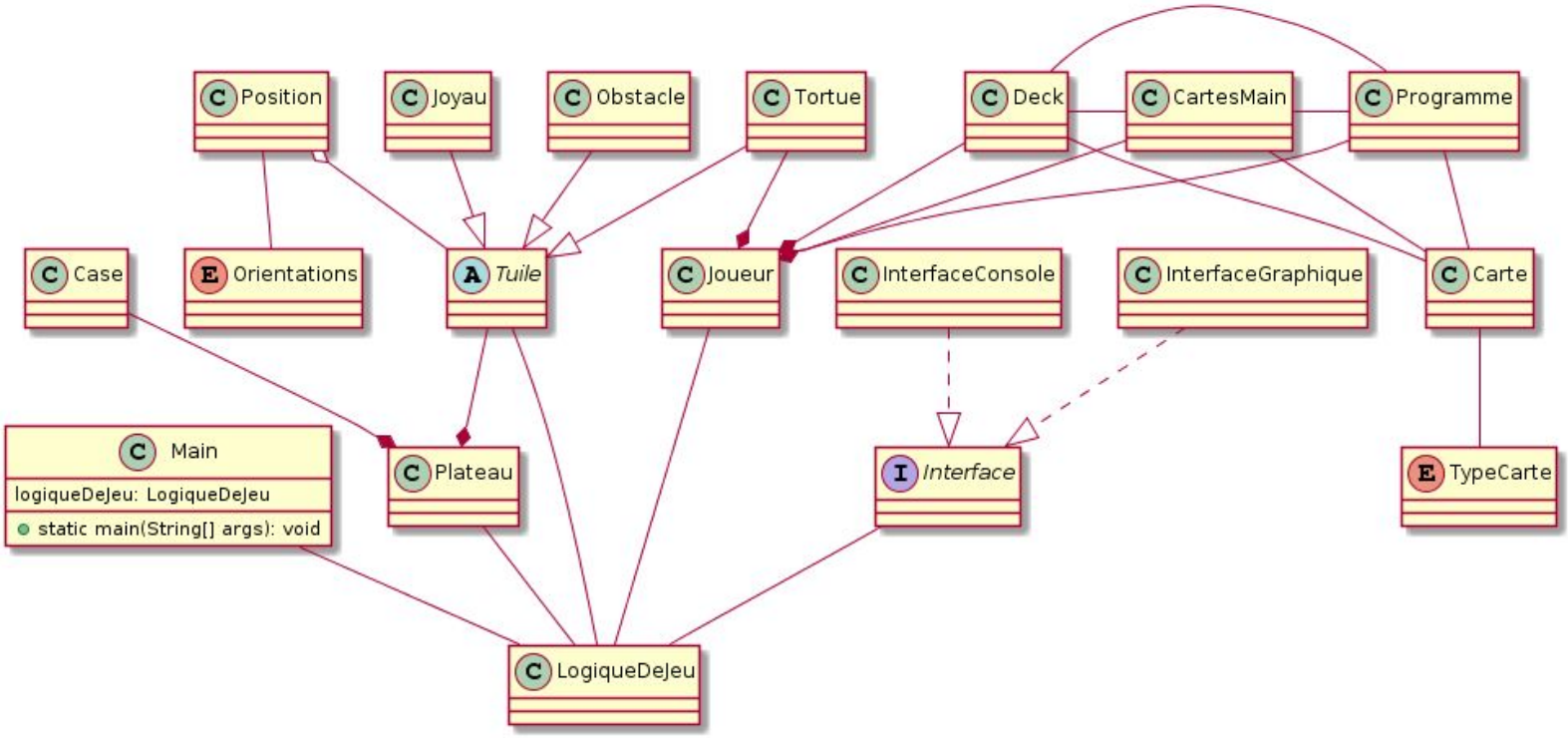
A chaque tour:

- Sélectionner l'action à effectuer (une seule par tour):
    - Placer un mur:
        - Sélectionner le type de mur
        - Choisir les coordonnées de placement du mur
        - Valider
    - Compléter le programme
        - Cliquer sur les cartes à ajouter au programme
        - Valider
    - Exécuter le programme
    - Bug (facultatif)
        - Sélectionner l'animal à cibler
- Sélectionner les cartes à défausser

A la fin de la manche:

- On enchaîne avec la manche suivante en mode trois à la suite
  OU
- On affiche les scores et le programme s'arrête.

Diagramme de classe du projet



Architecture du projet

Notre Programme s'articule autour de la classe LogiqueDeJeu. Cette classe régit l'obtention des données nécessaires à l'initialisation de la partie ainsi que le lancement et l'éventuelle réinitialisation de celle-ci .

La classe Joueur comporte les attributs de chaque joueur: leur programme, leur pioche, leur main et leurs murs et bug restant, et effectue la réinitialisation de leur pioche une fois qu'elle est vide.

Pour chaque joueur est créée une instance de Tortue qui peut effectuer les actions dans le programme du joueur; cette classe hérite donc, ainsi que les autre objets liés au plateau, de la classe Tuile.

L'ensemble de ces tuiles est placé dans le plateau, ce qui permet de vérifier une fois le plateau doté de tous ces éléments qu'aucune tortue n'est bloquée ou n'avance dans des positions qui lui sont interdites.

Description d'une fonctionnalité intéressante

Durant le jeu, les joueurs sont amenés à placer des murs sur le plateau. Afin de décider si le placement d'un mur à des coordonnées demandées par un joueur est valide, il est notamment nécessaire de vérifier que ce placement ne bloquerait l'accès à aucune tortue à aucun joyau. C'est la méthode:

src/Plateau/placementBloquant(int[] coordonnees)

qui effectue cette vérification, en fonction des coordonnées x et y auquelles il est demandé l'autorisation de placer un mur.

Pour chaque joyau, cette méthode appelle à son tour la méthode getCasesAccessibles() qui prend en paramètres les coordonnées de ce joyau, et effectue une recherche récursives de ses cases adjacentes considérées libres, grâce à la méthode getRecursiveCasesAdjacentesLibres().

Ainsi, dès qu'une tortue est trouvée dans une case récursivement adjacente au joyau en cours de test, on sait que relativement à ce joyau, le placement d'obstacle est autorisé: on sort de la boucle et on passe au joyau suivant.

Si relativement à tous les joyaux, le placement a été déterminé non bloquant, alors le placement de l'obstacle est finalement autorisé.

Description des bibliothèques externes utilisées

Swing

Description

Swing est une bibliothèque permettant de créer et d'ouvrir la fenêtre de jeu, d'y placer des boutons, d'importer des fichiers image et de les rendre utilisables, de créer des menus déroulants.

Classes utilisées:

ImageIcon; JButton; JComboBox; JFrame; JOptionPane; JPanel; JToggleButton;

Java.awt

Description

Java.awt est une bibliothèque permettant la création d'une interface graphique stylisée et interactive pour notre projet par l'attribution de fonctions aux boutons, un affichage de texte et sa mise en forme, un affichage d'images sur le plateau, une rotation de celles-ci sur le plateau.

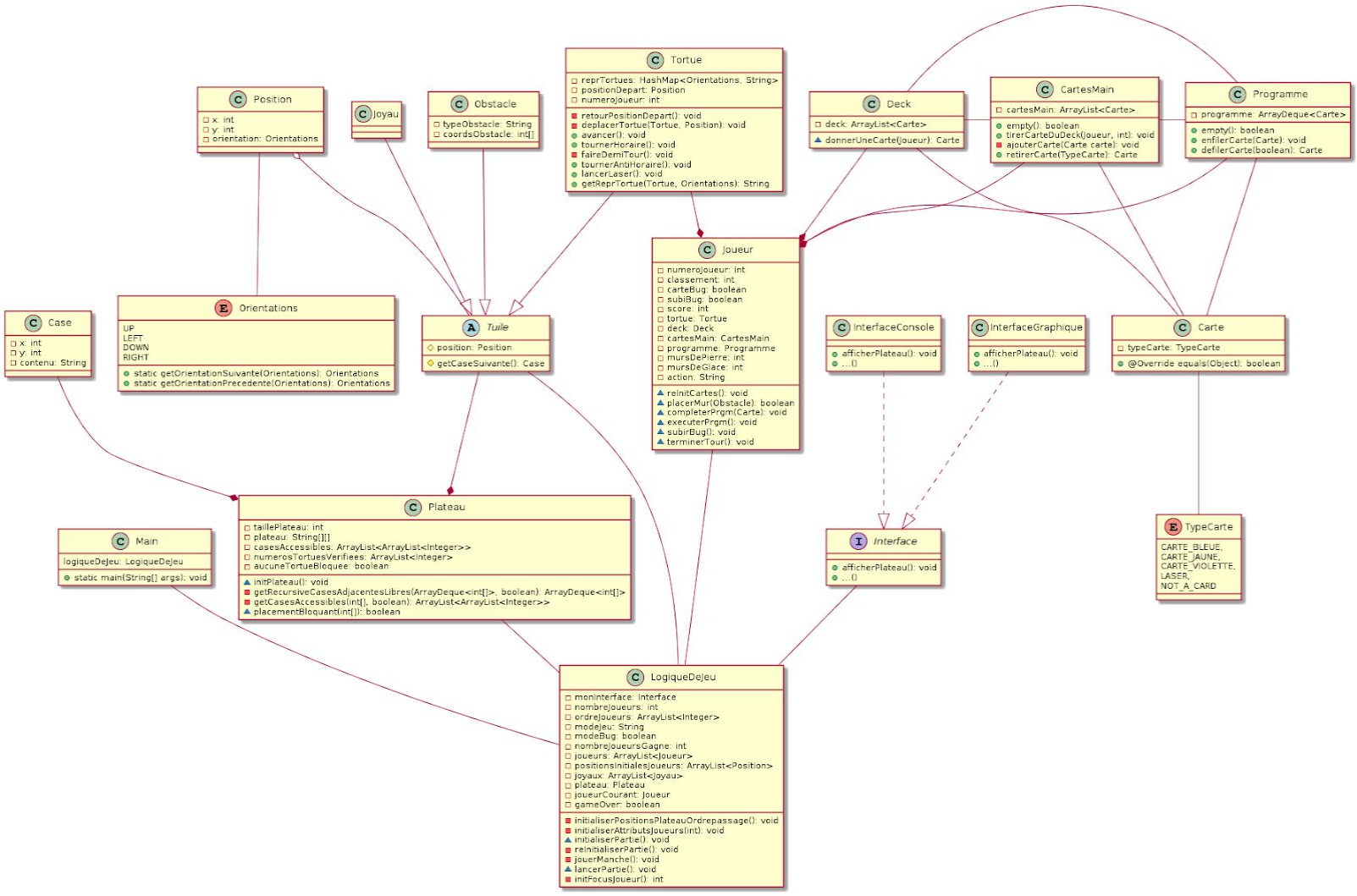
Classes utilisées:

Color; Dimension; FlowLayout; Font; Graphics; Graphics2D; Image; event.ActionEvent; event.ActionListener; event.ItemEvent; event.ItemListener; event.MouseEvent; event.MouseListener

Sources

Images
https://www.deviantart.com/rofelrolf/art/Mech-Turtle-376344017
https://pngtree.com/so/article-6
Apprentissage
http://zetcode.com/tutorials/javagamestutorial/
Son:
https://www.youtube.com/watch?v=WJRoRt155mA&list=PLyo74dkSc2HCKNmORHIXk1J-dnSGYzpqj&index=5
Github
https://github.com/broken-clock/RobotTurtles.git

Annexe


Tortue
reprTortues: HashMap<Orientations, String>
positionDepart: Position
numeroJoueur: int
retourPositionDepart(): void
deplacerTortue(Tortue, Position): void
avancer(): void
tournerHoraire(): void
faireDemiTour(): void
tournerAntiHoraire(): void
lancerLaser(): void
getReprTortue(Tortue, Orientations): String
Position
x: int
y: int
orientation: Orientations
Joyau
Obstacle
typeObstacle: String
coordsObstacle: int[]
Deck
deck: ArrayList<Carte>
donnerUneCarte(Joueur): Carte
CartesMain
cartesMain: ArrayList<Carte>
empty(): boolean
tirerCarteDuDeck(Joueur, int): void
ajouterCarte(Carte carte): void
retirerCarte(TypeCarte): Carte
Programme
programme: ArrayDeque<Carte>
empty(): boolean
enfilerCarte(Carte): void
defilerCarte(boolean): Carte
Joueur
numeroJoueur: int
classement: int
carteBug: boolean
subiBug: boolean
score: int
tortue: Tortue
deck: Deck
cartesMain: CartesMain
programme: Programme
mursDePierre: int
mursDeGlace: int
action: String
reInitCartes(): void
placerMur(Obstacle): boolean
completerPrgm(Carte): void
executerPrgm(): void
subirBug(): void
terminerTour(): void
Case
x: int
y: int
contenu: String
Orientations
UP
LEFT
DOWN
RIGHT
static getOrientationSuivante(Orientations): Orientations
static getOrientationPrecedente(Orientations): Orientations
Tuile
position: Position
getCaseSuivante(): Case
InterfaceConsole
afficherPlateau(): void
...()
InterfaceGraphique
afficherPlateau(): void
...()
Carte
typeCarte: TypeCarte
@Override equals(Object): boolean
Plateau
taillePlateau: int
plateau: String[][]
casesAccessibles: ArrayList<ArrayList<Integer>>
numerosTortuesVerifiees: ArrayList<Integer>
aucuneTortueBloquee: boolean
initPlateau(): void
getRecursiveCasesAdjacentesLibres(ArrayDeque<int[]>, boolean): ArrayDeque<int[]>
getCasesAccessibles(int[], boolean): ArrayList<ArrayList<Integer>>
placementBloquant(int[]): boolean
Main
logiqueDeJeu: LogiqueDeJeu
static main(String[] args): void
Interface
afficherPlateau(): void
...()
TypeCarte
CARTE_BLEUE,
CARTE_JAUNE,
CARTE_VIOLETTE,
LASER,
NOT_A_CARD
LogiqueDeJeu
moninterface: interface
nombreJoueurs: int
ordreJoueurs: ArrayList<Integer>
modeJeu: String
modeBug: boolean
nombreJoueursGagne: int
joueurs: ArrayList<Joueur>
positionsInitialesJoueurs: ArrayList<Position>
joyaux: ArrayList<Joyau>
plateau: Plateau
joueurCourant: Joueur
gameOver: boolean
initialiserPositionsPlateauOrdrepassage(): void
initialiserAttributsJoueurs(int): void
initialiserPartie(): void
reInitialiserPartie(): void
jouerManche(): void
lancerPartie(): void
initFocusJoueur(): int