Rapport - MVC-BAP

ENEE Luc, GERMAIN Adam

C'est un jeu de balle au prisonnier, où deux équipes de 3 joueurs s'affrontent. Chaque équipe a un joueur contrôlé par un humain et deux bots.

Si un joueur tient la balle, il a la possibilité de la lancer dans la direction souhaitée pour éliminer des membres de l'équipe adverse. Dans le cas où un joueur se fait éliminer, il prend le contrôle d'un des bots encore en vie de son équipe et la portie continue. Les joueurs sont cantonnés à leur moitié de terrain.

La balle perd de la vitesse une fois lancée, et s'arrête elle devient trop lente. Elle peut alors être ramassée par un joueur qui passe à proximité. Si la balle touche le côté gauche ou droite du terrain, elle rebondit et gagne aussi de la vitesse. C'est un choix délibéré afin d'encourager des tirs ambitieux et surprenants. Si la balle sort par le haut ou le bas du terrain, elle est remise au centre de la zone de l'équipe correspondante, dans le but d'être ramassée par l'équipe adverse quand un tir est manqué.

Design Pattern :

MVC :

Le modèle-vue-contrôleur (MVC) est un modèle architectural qui sépare une application en trois groupes principaux de composants : les modèles, les vues et les contrôleurs. MVC est abrégé en tant que Model View Controller, et est un modèle de conception créé pour développer des applications.

Il reste assez pratique pour le développement de jeu grâce à la séparation de la vue et du contrôleur. En effet, dans ce développement l’affichage et les contrôles sont amenés à changer en permanence indépendamment du modèle.

Singleton :

Singleton est un modèle de conception de création qui vous permet de d'assurer qu'une classe n'a qu'une seule instance, tout en fournissant un point d'accès global à cette instance.

Ici elle est utilisée pour le Field qui est la zone de dessin du jeu. On aurait pu aussi l'utiliser pour la balle car cette dernière reste unique dans notre version du jeu.

Encapsulation : (Facade)

Facade est un modèle de conception structurelle qui fournit une interface simplifiée à une bibliothèque, un framework ou tout autre ensemble complexe de classes.

Ici nous nous en servons de l'encapsulation (facade) pour les classes Human et Bot qui héritent de la classe Player qu'il vont utiliser comme interface simplifiée.

Description :

Ball : Class

- Ball(double x, double y, double angle, double speed) : constructeur de la balle permet d'initialiser cette dernière
- void setAngle(double angle) : permet de définir l'angle de la balle
- void setSpeed(double speed) : permet de définir la vitesse de la balle
- double getX() : permet de récupérer la coordonnée X de la balle
- double getY() : permet de récupérer la coordonnée Y de la balle
- Image getImage() : permet de récupérer l'image de la balle
- void move() : permet la gestion des déplacements de la balle
- void isOutofBound() : permet à la balle de rester sur le terrain de jeu en la faisant rebondir sur les murs des côtés et de placer la balle au milieu de la zone d'équipe dans le cas où elle sort du terrain par le haut ou le bas
- void bounce() : permet de faire rebondir la balle sur les murs des côtés
- boolean touch(PLayer player) : permet de dire si la balle a touché un joueur
- void update(PlayerTeam teamA, PlayerTeam teamB) : met à jour la balle en fonction de ce qu'elle rencontre sur le terrain

Bot : Class extend Player

- Bot(string color, double xInit, double yInit, string side) : constructeur du bot permet d'initialiser ce dernier
- void move() : permet de faire bouger le bot
- void moveLeft() : gère les déplacement du bot vers la gauche
- void moveRight() : gère les déplacement du bot vers la droite

Human : Class extend Player

- Human(string color, double xInit, double yInit, string side) : constructeur du joueur humain permet d'initialiser ce dernier
- void turnLeft() : permet de modifier l'angle de visée du joueur vers la gauche
- void turnRight() : permet de modifier l'angle de visée du joueur vers la droite
- void shoot() : réalise le tir de la balle par le joueur

Player : Class

- Player(string color, double xInit, double yInit, string side) : constructeur du joueur permet d'initialiser ce dernier
- double getX() : permet de récupérer la coordonnée X du joueur
- double getY() : permet de récupérer la coordonnée Y du joueur
- void moveLeft() : permet au joueur de se déplacer vers la gauche
- void moveRight() : permet au joueur de se déplacer vers la droite
- void moveUp() : permet au joueur de se déplacer vers le haut
- void moveDown() : permet au joueur de se déplacer vers le bas
- void spriteAnimate() : permet d'animer le sprite du joueur quand il court
- boolean isTouched() : permet de dire si un joueur est touché par la balle

PlayerTeam : Class

- PlayerTeam(int botnumber, string color, string side) : constructeur des équipes de joueurs
- Human getHumanPlayer() : permet de récupérer le joueur humain de l'équipe
- string getSide() : permet de récupérer le side de l'équipe
- ArrayList<Player> getPLayer() : permet de récupérer la liste des joueurs de l'équipe
- void checkBallCollision(Ball b) : permet de savoir si la balle a touché un joueur de l'équipe

- boolean allPlayersKilled() : permet de savoir si tous les joueurs de l'équipe sont morts

Field : Class (Singleton)

- Field(double width, double height) : constructeur de la fenêtre ou sera affiché le jeu
- static Field getInstance() : permet de récupérer une instance de Field
- double getTopsideYLimit() : permet de récupérer la limite de déplacement de l'équipe du haut
- double getBopsideYLimit() : permet de récupérer la limite de déplacement de l'équipe du bas
- static void createInstance(double width, double height) : permet de créer l'instance de Field

Controller : Class

- Controller(PlayerTeam teamA, PlayerTeam teamB, Ball balle) : constructeur du contrôleur qui gère tous les contrôles du jeu
- void PlayerMovement() : permet de gérer les déplacements des joueurs en fonction des touches définies

View : Class

- View(Field field) : constructeur de la vue permettant d'afficher le jeu
- void setGameActors(ArrayList<Player> players, Ball balle) : permet de mettre à jour les joueurs encore en vie
- void drawGame() : permet d'afficher le jeu
- void drawEndOfGame(string winnigside) : permet d'afficher l'écran de fin de jeu avec un message de victoire
- void drawArrow() : permet d'afficher la flèche de visée pour les joueurs humains
- void rotate(GraphicsContext gc, double angle, double px, double py) : permet de tourner la flèche de visée

Sprite : Class

- Sprite(Image animationimage, int numCells, int numRows, Duration frametime, string side) : constructeur de sprite permettant la récupération des images correspondant à chaque élément du jeu

- void playContinuously() : permet de jouer une animation de manière permanente
- void playShoot() : permet de jouer l’animation de tir