

# Rapport DAC

## Groupe 01

Les étudiants:

- ❖ Wassim Oubaziz.
- ❖ Mohamed Zine Eddine Larkem.
- ❖ Houcine Ghedjati.

Enseignant de module: **Meriem Belguidoum**



## **Aperçu du projet:**

Ce code représente un jeu où deux équipes de joueurs (appelées "PlayerA" et "PlayerB") passent un ballon entre eux. Chacune des équipes est composée de trois joueurs identifiés par un numéro (0, 1 et 2).

Les joueurs utilisent des sémaphores pour gérer l'accès au ballon et pour simuler les actions de jeu telles que la possession de balle, la perte de balle et le marquage. Les threads sont utilisés pour permettre à chaque joueur de jouer simultanément. En général, l'objectif du code est de simuler une partie de basket entre deux équipes avec des joueurs qui ont la possibilité de marquer des points, de perdre le ballon et de le passer à leurs coéquipiers.

### ***1. Expliqué les Threads et les sémaphores:***

Dans ce code, il y a une classe appelée "Player" qui étend la classe Thread de Java. Il y a également une classe appelée "Ball" qui contient des informations sur l'équipe qui possède le ballon. La méthode principale de la classe Player crée deux objets Ball, "ball1" et "ball2", et lance six threads, chacun associé à un des joueurs de l'équipe (0, 1 ou 2) et au ballon correspondant. Chacun des threads utilise des sémaphores pour synchroniser l'accès au ballon. Un sémaphore est un objet qui peut être utilisé pour contrôler l'accès à une ressource partagée, en garantissant qu'un seul thread à la fois peut accéder à cette ressource. Les sémaphores utilisés dans ce code sont "sema0", "sema1" et "sema2".

Dans le corps de la méthode run() de chaque thread, il y a une boucle infinie qui répète les étapes suivantes: l'acquisition d'un sémaphore, la récupération du ballon, une pause pour simuler une action de jeu, une vérification aléatoire de la perte du ballon et la libération du sémaphore correspondant au prochain joueur. L'utilisation des sémaphores dans ce code permet de garantir qu'un seul joueur à la fois peut avoir le ballon, ce qui évite les

conflits entre les threads et permet de simuler une partie de football.

Les sémaphores sont des objets de la classe Semaphore de Java qui sont utilisés pour synchroniser les threads. Ils peuvent être utilisés pour limiter l'accès à une ressource partagée en permettant seulement un certain nombre de threads d'accéder à la ressource à la fois. Dans ce cas, chaque joueur a un sémaphore qui est utilisé pour synchroniser l'accès à la balle. Lorsqu'un joueur veut avoir la balle, il appelle la méthode `acquire()` sur son sémaphore. Cela bloque le thread jusqu'à ce que le sémaphore soit disponible. Une fois qu'un joueur a la balle, il peut appeler la méthode `release()` sur le sémaphore pour permettre à un autre joueur de l'acquérir.

## ***2.Expliqué les méthodes utiliser:***

**1.** La méthode `Player(int playerId, Ball ball)`: C'est le constructeur de la classe `Player`. Il initialise les variables d'instance de la classe avec les valeurs passées en paramètres.

**2.** La méthode `run()` : C'est la méthode principale exécutée par chaque thread de joueur. Elle contient un bloc if qui détermine quelles actions doivent être effectuées par chaque joueur en fonction de son numéro. Chaque joueur utilise des sémaphores pour gérer l'accès au ballon et pour simuler les actions de jeu telles que la possession de balle, la perte de balle et le marquage.

**3.** La méthode `acquire()` de `Sémaphore` : Cette méthode est utilisée pour acquérir ou bloquer l'accès à un sémaphore. Cela signifie que si un autre thread a déjà acquis ce sémaphore, le thread appelant cette méthode sera bloqué jusqu'à ce que le sémaphore soit libéré.

**4.** La méthode `release()` de `Sémaphore` : Cette méthode est utilisée pour libérer ou débloquent un sémaphore. Cela signifie que si un autre thread est bloqué en attendant d'acquérir ce sémaphore, il sera débloquent et pourra continuer son exécution.

**5.** La méthode `sleep()` : Cette méthode est utilisée pour faire une pause dans l'exécution d'un thread pendant un certain temps (en millisecondes). Elle est utilisée ici pour simuler le temps de jeu.

**6.** La méthode `Math.random()`: Cette méthode est utilisée pour générer un nombre aléatoire compris entre 0 et 1. Elle est utilisée

ici pour simuler l'action aléatoire de perdre ou de garder le ballon lors de la passe.

**7.** La méthode `start()` démarre le jeu en créant deux instances de la classe `Ball`, une pour `"PlayerA"` et une pour `"PlayerB"`, et trois instances de la classe `Player` pour chaque équipe. Il démarre également un nouveau thread qui met à jour la position des deux instances de balles sur le plateau de jeu et met à jour le score pour chaque équipe.

**9.** La méthode `reset()` efface le plateau de jeu, définit la variable `Player.gameOverOver` à `true` et définit la variable `inited` à `false`, réinitialisant ainsi efficacement le jeu.

**10.** La méthode `init()` initialise le jeu en créant des instances de la classe `Gamer` et de la classe `BallG`, et en les ajoutant au plateau de jeu. Il définit également la variable `inited` à `true`, indiquant que le jeu a été initialisé.

### ***3.Expliqué la class HelloApplication :***

Cette classe `"HelloApplication"` étend la classe `Application` de `JavaFX`, qui est utilisée pour créer des applications graphiques en utilisant `Java`. La méthode `start()` charge un fichier `FXML` qui contient la vue de l'application, crée une scène à partir de cette vue, définit le titre de la fenêtre de l'application, affiche la scène sur la stage et gère l'événement de fermeture de la fenêtre. Lorsque l'utilisateur ferme la fenêtre, la variable `gameOverOver` de la classe `Player` est définie sur `true` et le programme se termine en appelant la méthode `Platform.exit()`. La méthode `main()` est utilisée pour lancer l'application en appelant la méthode `launch()` de la classe `Application`.

#### **code de cet classe:**

```
package app.dacproject;
import app.dacproject.classes.Player;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;
```

```

import java.io.IOException;

public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("/hello-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load());
        stage.setResizable(false);
        stage.setTitle("Front Back FootBall Game!");
        stage.setScene(scene);
        stage.setOnCloseRequest(e -> {
            Player.gameOverOver = true;
            Platform.exit();
        });
        stage.show();}
    public static void main(String[] args) {
        launch(); }}

```

#### ***4.Expliqué la class Gamer:***

La classe Gamer étend la classe ImageView de JavaFX. Cela signifie qu'elle a toutes les fonctionnalités de la classe ImageView de base, mais peut également inclure des fonctionnalités supplémentaires spécifiques aux joueurs.

La classe contient un constructeur qui prend en entrée 4 paramètres : x, y, z et image. Les paramètres x et y définissent les coordonnées de l'image (c'est-à-dire où elle doit être affichée sur la fenêtre de jeu). Le paramètre z définit la rotation de l'image et le paramètre image définit l'image à afficher.

Le constructeur utilise ensuite les méthodes setX, setY, setRotate et setImage pour définir les propriétés de l'objet ImageView.

#### ***Le code de cet class:***

```

package app.dacproject.classes;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
public class Gamer extends ImageView {
    public Gamer(double x, double y, double z, Image image) {

```

```
setX(x);
setY(y);
setRotate(z);
setImage(image);
}}
```

### ***5.Expliqué la class BallG:***

Cette classe, appelée "BallG", est une sous-classe de la classe "ImageView" de JavaFX. Elle est utilisée pour créer une instance de la balle de jeu qui sera affichée sur l'interface graphique. Elle prend en entrée 4 paramètres : x, y, z et image. Les coordonnées x et y déterminent la position de la balle sur l'interface graphique, z détermine l'angle de rotation de la balle, et l'image est l'image qui sera utilisée pour représenter la balle.

Dans le constructeur de cette classe, les valeurs des paramètres sont affectées aux propriétés correspondantes de l'objet : la position (x, y), la rotation (z) et l'image (image).

#### **code de cet class:**

```
package app.dacproject.classes;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
public class BallG extends ImageView {
    public BallG(double x, double y, double z, Image image) {
        setX(x);
        setY(y);
        setRotate(z);
        setImage(image);
    }
}
```

### ***6.Expliqué la class Player:***

La classe Player étend la classe Thread, ce qui signifie qu'elle peut être utilisée pour créer des threads. La classe Player a deux variables d'instance: playerId et ball. playerId stocke l'ID du joueur et ball stocke une référence à un objet de la classe Ball qui est utilisé pour stocker les informations sur le match.

**code de cet class:**

```
package app.dacproject.classes;
public class Player extends Thread{
    int playerId;
    Ball ball;
    public static boolean gameOverOver= false;
    public Player(int playerId, Ball ball) {
        this.playerId = playerId;
        this.ball = ball;    }
    @Override
    public void run(){
        //boolean gameOver = false;
        while(!gameOverOver) {
            if (playerId == 0) {
                try {
                    ball.sema0.acquire();
                    ball.ballOwner = 0;
                    System.out.println(ball.teamName + " " + playerId + " has the ball");
                    sleep(1000);
                    ball.state = Math.random();
                    if (ball.state < 0.2) {
                        System.out.println(ball.teamName + " " + playerId + " lost the
ball");
                        ball.ballOwner = 3;
                        sleep(1000);
                        ball.sema0.release();
                        continue;    }
                    ball.sema1.release();
                    ball.sema0.acquire();
                    if (ball.state < 0.2) {
                        ball.sema0.release();
                        continue;    }
                    ball.ballOwner = 0;
                    System.out.println(ball.teamName + " " + playerId + " has and shot
the ball");
                    sleep(1000);
                    ball.ballOwner = 4;
                    sleep(1000);
                    ball.score++;
                    if (ball.score == 10) {
```

```

        gameOverOver = true;
        break; }
    System.out.println(ball.teamName + " scored butt " + ball.score);
    ball.sema0.release();
} catch (InterruptedException e) {
    throw new RuntimeException(e); }
} else if (playerId == 1) {
    try {
        ball.sema1.acquire();
        ball.ballOwner = 1;
        System.out.println(ball.teamName + " " + playerId + " has the ball");
        sleep(1000);
        ball.state = Math.random();
        if (ball.state < 0.2) {
            System.out.println(ball.teamName + " " + playerId + " lost the
ball");

            ball.ballOwner = 3;
            sleep(1000);
            ball.sema0.release();
            continue; }
        ball.sema2.release();
        ball.sema1.acquire();
        if (ball.state < 0.2) {
            ball.sema0.release();
            continue; }
        ball.ballOwner = 1;
        System.out.println(ball.teamName + " " + playerId + " has the ball");
        sleep(1000);
        ball.sema0.release();
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
} else if (playerId == 2) {
    try {
        ball.sema2.acquire();
        ball.ballOwner = 2;
        System.out.println(ball.teamName + " " + playerId + " has the ball");
        sleep(1000);
        ball.state = Math.random();
        if (ball.state < 0.2) {
            System.out.println(ball.teamName + " " + playerId + " lost the
ball");

            ball.ballOwner = 3;
            sleep(1000);

```



```

        ball.sema1.release();
        continue;
    }
    ball.sema1.release();
} catch (InterruptedException e) {
    throw new RuntimeException(e);
} } }
ball.sema0.release(2);
ball.sema1.release(2);
ball.sema2.release(1);
}}

```

## ***7.Expliqué la classe Ball:***

La classe Ball représente un objet ballon dans une application de jeu. Elle contient des informations sur l'équipe propriétaire de la balle (ballOwner), le nom de l'équipe (teamName) et le score actuel (score). Elle utilise également trois semaphores (sema0, sema1 et sema2) pour gérer la synchronisation des threads. La variable state est utilisée pour stocker l'état actuel de la balle, mais il n'est pas clair à partir du code fourni comment elle est utilisée. La classe possède un constructeur qui prend en paramètre le nom de l'équipe et l'utilise pour initialiser la propriété teamName.

### **code de cet classe:**

```

package app.dacproject.classes;

import java.util.concurrent.Semaphore;

public class Ball {
    int ballOwner;
    String teamName;
    Semaphore sema0 = new Semaphore(1);
    Semaphore sema1 = new Semaphore(0);
    Semaphore sema2 = new Semaphore(0);
    int score = 0;

    double state= 0;

    public Ball(String teamName){
        this.teamName = teamName;
    }
}

```

```
}  
}
```

## **8.Expliqué la class Images:**

Classe utilitaire appelée Images qui contient des variables statiques de type Image. Ces variables sont des images de joueurs verts et rouges et une image de balle de soccer. Ces images sont chargées à partir des emplacements spécifiés dans le chemin d'accès de l'image (par exemple "/characterGreen.png") et ont des tailles de 45x70 pour les joueurs et 35x35 pour la balle. La méthode privée Images() empêche l'instanciation de cette classe.

### **code de cet classe:**

```
package app.dacproject.utiles;  
import javafx.scene.image.Image;  
public final class Images {  
    public static final Image GreenPLAYER    = new  
Image("/characterGreen.png", 45, 70, false, false);  
    public static final Image RedPLAYER      = new Image("/characterRed.png",  
45, 70, false, false);  
    public static final Image BALL          = new Image("/ball_soccer1.png", 35, 35,  
false, false);  
    private Images() {  
    }  
}}
```

## **9.Expliqué la classe HelloController:**

*La classe HelloController est responsable de la logique de jeu et de la mise à jour de l'affichage du jeu. La méthode start() démarre le jeu en créant de nouvelles instances de la classe Ball et Player pour chaque équipe et en démarrant leurs threads. La méthode reset() arrête le jeu en effaçant le plateau et en mettant la variable initied à false. La méthode init() initialise le jeu en créant de*

*nouvelles instances de la classe Gamer et BallG pour chaque équipe et en les ajoutant au plateau. Elle met également la variable initied à true, de sorte que le jeu puisse démarrer. Le jeu est contrôlé par les boutons start et reset dans l'interface utilisateur graphique.*

**code cet classe:**

```
package app.dacproject.classes;
import javafx.animation.AnimationTimer;
import javafx.fxml.FXML;
import javafx.scene.layout.Pane;
import javafx.scene.text.Text;
import static app.dacproject.utiles.Images.*;
public class HelloController {
    @FXML
    private Pane board;
    @FXML
    private Text score1;
    @FXML
    private Text score2;
    private BallG ballG1;
    private BallG ballG2;
    private Gamer playerA1;
    private Gamer playerA2;
    private Gamer playerA3;
    private Gamer playerB1;
    private Gamer playerB2;
    private Gamer playerB3;

    private boolean initied = false;

    public HelloController(){

    }
    @FXML
    public void start() {
        if(initied){
            Player.gameOverOver = false;
            Ball ball1 = new Ball("PlayerA");
            (new Player(0, ball1)).start();
            (new Player(1, ball1)).start();
        }
    }
}
```

```
(new Player(2, ball1)).start();
```

```
Ball ball2 = new Ball("PlayerB");
```

```
(new Player(0, ball2)).start();
```

```
(new Player(1, ball2)).start();
```

```
(new Player(2, ball2)).start();
```

```
new Thread(new Runnable() {
```

```
    @Override
```

```
    public void run() {
```

```
        while(true) {
```

```
            if(ball1.ballOwner == 0){
```

```
                ballG1.setX(162);
```

```
                ballG1.setY(149);
```

```
            }else if(ball1.ballOwner == 1){
```

```
                ballG1.setX(335);
```

```
                ballG1.setY(237);
```

```
            }else if(ball1.ballOwner == 2){
```

```
                ballG1.setX(510);
```

```
                ballG1.setY(149);
```

```
            }else if(ball1.ballOwner == 3){
```

```
                ballG1.setX(0);
```

```
                ballG1.setY(0);
```

```
            }else if(ball1.ballOwner == 4){
```

```
                ballG1.setX(335);
```

```
                ballG1.setY(0);
```

```
        }
```

```
        score1.setText("" + ball1.score);
```

```
        if(ball2.ballOwner == 0){
```

```
            ballG2.setX(162);
```

```
            ballG2.setY(523);
```

```
        }else if(ball2.ballOwner == 1){
```

```
            ballG2.setX(335);
```

```
            ballG2.setY(439);
```

```
        }else if(ball2.ballOwner == 2){
```

```
            ballG2.setX(510);
```

```
            ballG2.setY(523);
```

```
        }else if(ball2.ballOwner == 3){
```

```
            ballG2.setX(0);
```

```
            ballG2.setY(660);
```

```
        }
```

```
    }else if(ball2.ballOwner == 4){
```

```
        ballG2.setX(335);
```

```
        ballG2.setY(660);
```

```
    }
```

```

        score2.setText("" + ball2.score);
        if(Player.gameOverOver){
            break;
        }
    }

    }).start();
}

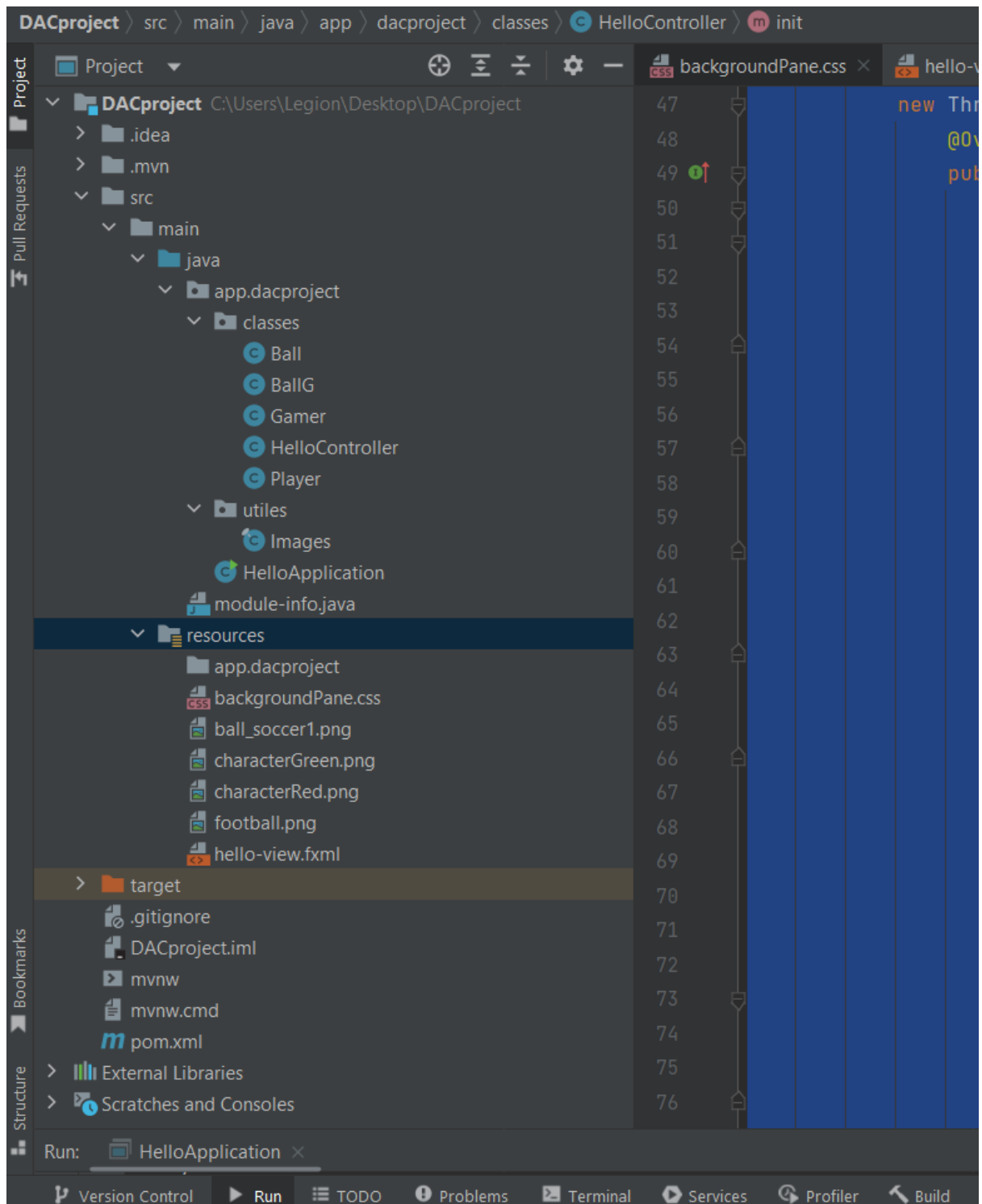
@FXML
public void reset()
{
    board.getChildren().clear();
    Player.gameOverOver = true;
    initied = false;
}

@FXML
private void init()
{
    initied = true;
    board.getChildren().removeAll(playerA1, playerA2, playerA3, ballG1);
    board.getChildren().removeAll(playerB1, playerB2, playerB3, ballG2);
    playerA1 = new Gamer(121 , 134, 0 , GreenPLAYER);
    playerA2 = new Gamer(330 , 253, 270 , GreenPLAYER);
    playerA3 = new Gamer(538 , 134, 180 , GreenPLAYER);
    ballG1 = new BallG(162, 149, 0, BALL);
    board.getChildren().addAll(ballG1,playerA1,playerA2,playerA3);

    playerB1 = new Gamer(121 , 508, 0 , RedPLAYER);
    playerB2 = new Gamer(330 , 388, 90 , RedPLAYER);
    playerB3 = new Gamer(538 , 508, 180 , RedPLAYER);
    ballG2 = new BallG(162, 523, 0, BALL);
    board.getChildren().addAll(ballG2,playerB1,playerB2,playerB3);
}
}

```

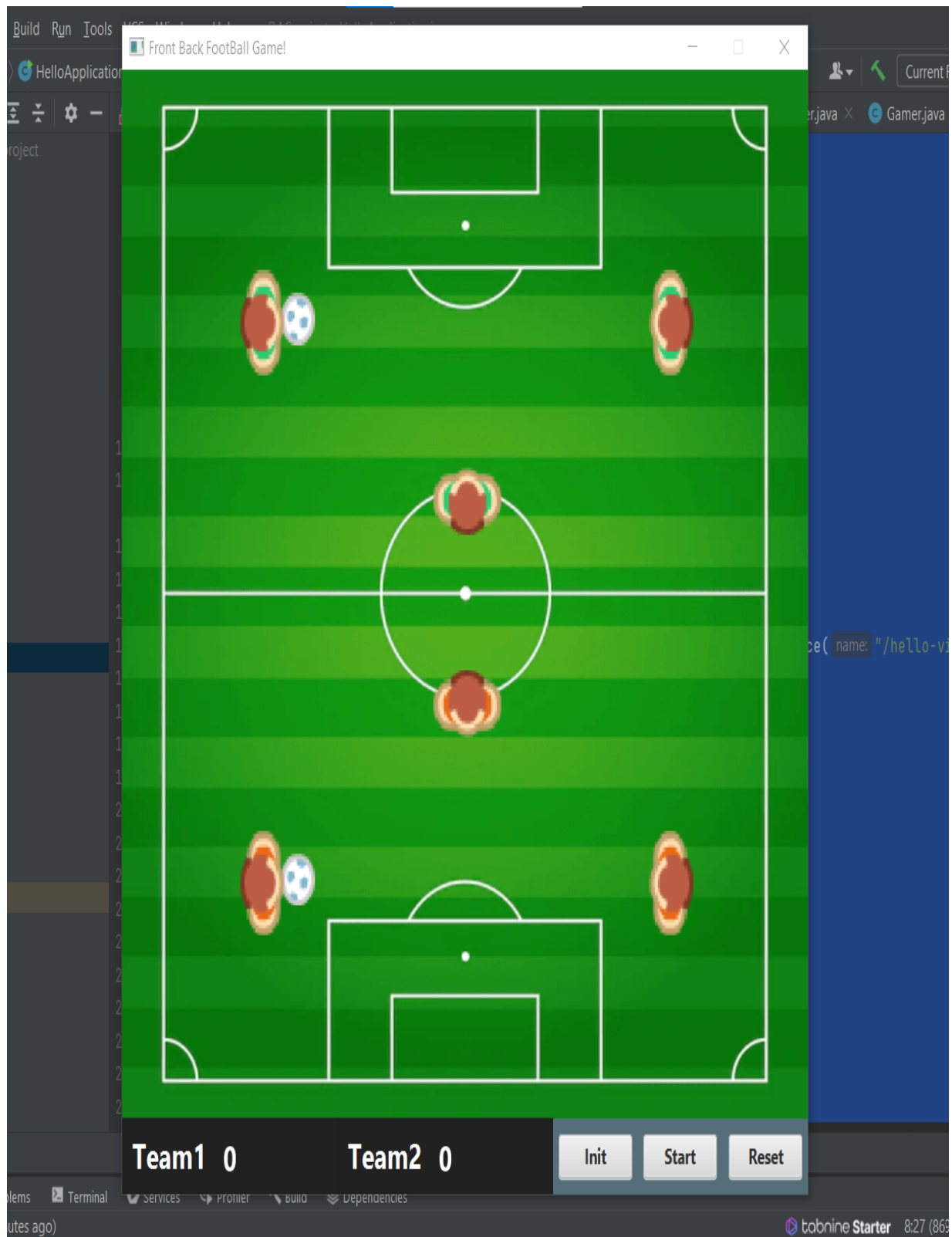
- **La structure de cet jeux :**



- *quand tu lances le jeu:*



- ***lorsque vous cliquez sur le bouton init:***

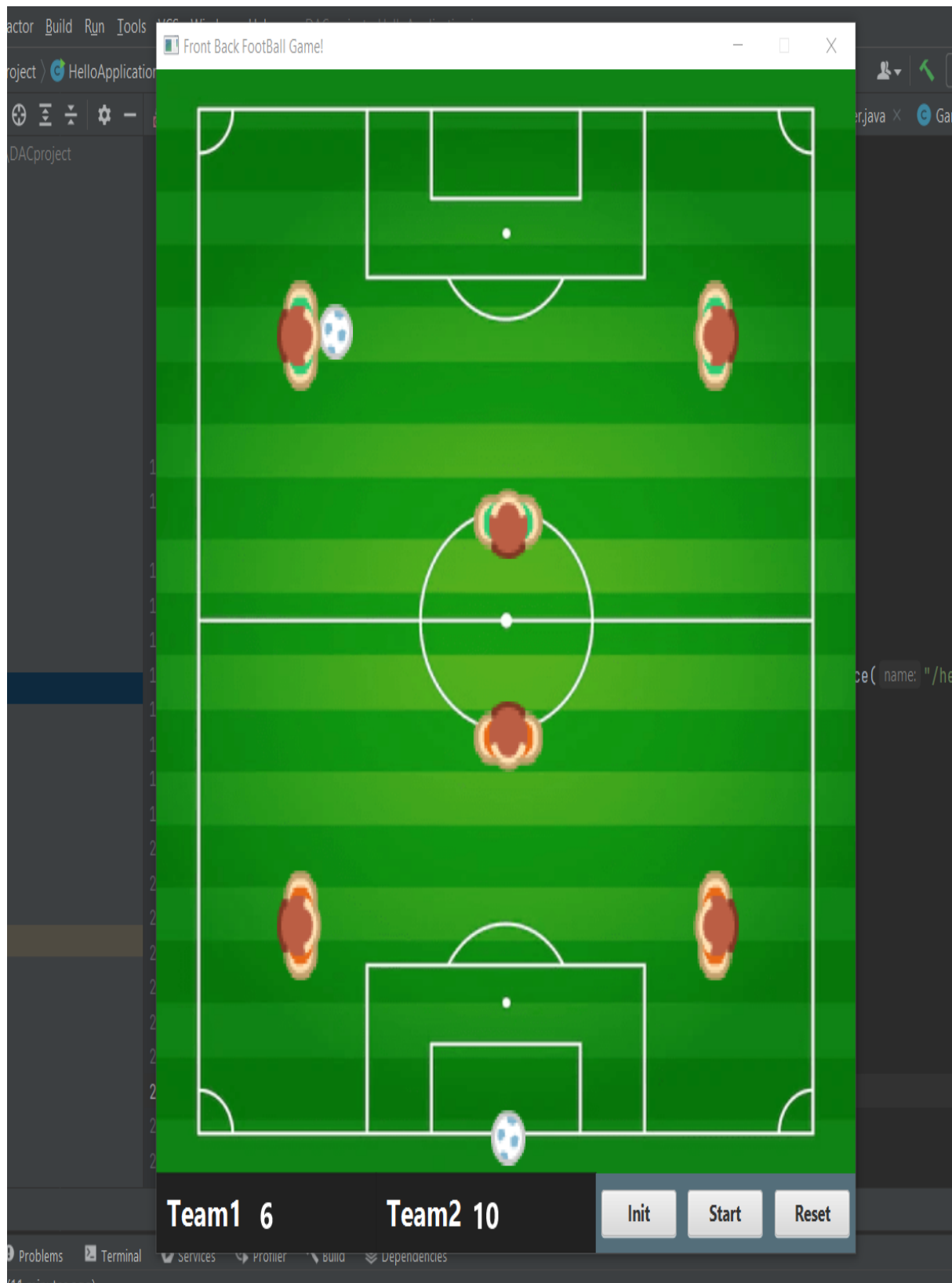




- *lorsque vous cliquez sur le bouton de démarrage:*



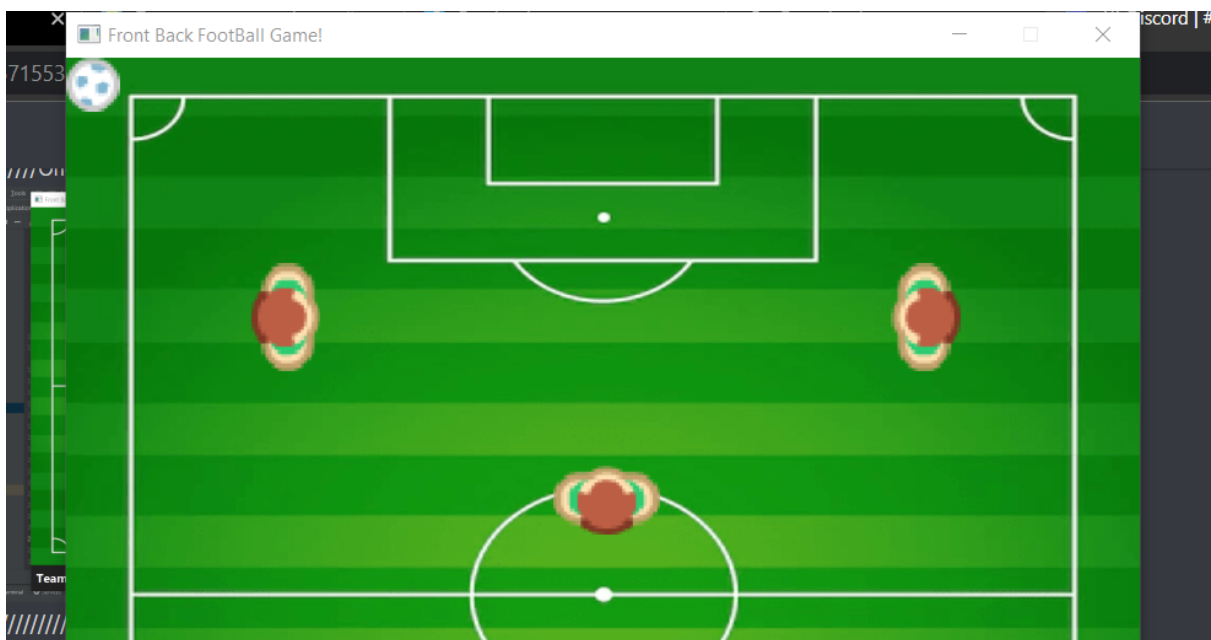
- **Une des équipes Obtenez 10 buts le jeu s'arrêtera automatiquement:**



- **Lorsque vous cliquez sur le bouton Reset:**



- **Lorsque l'équipe A perd le ballon, le ballon ira dans le coin:**



- *quand l'équipe B perd le ballon:*



- *lorsque vous quittez le programme, tout le fil s'arrête et tue le jeu:*

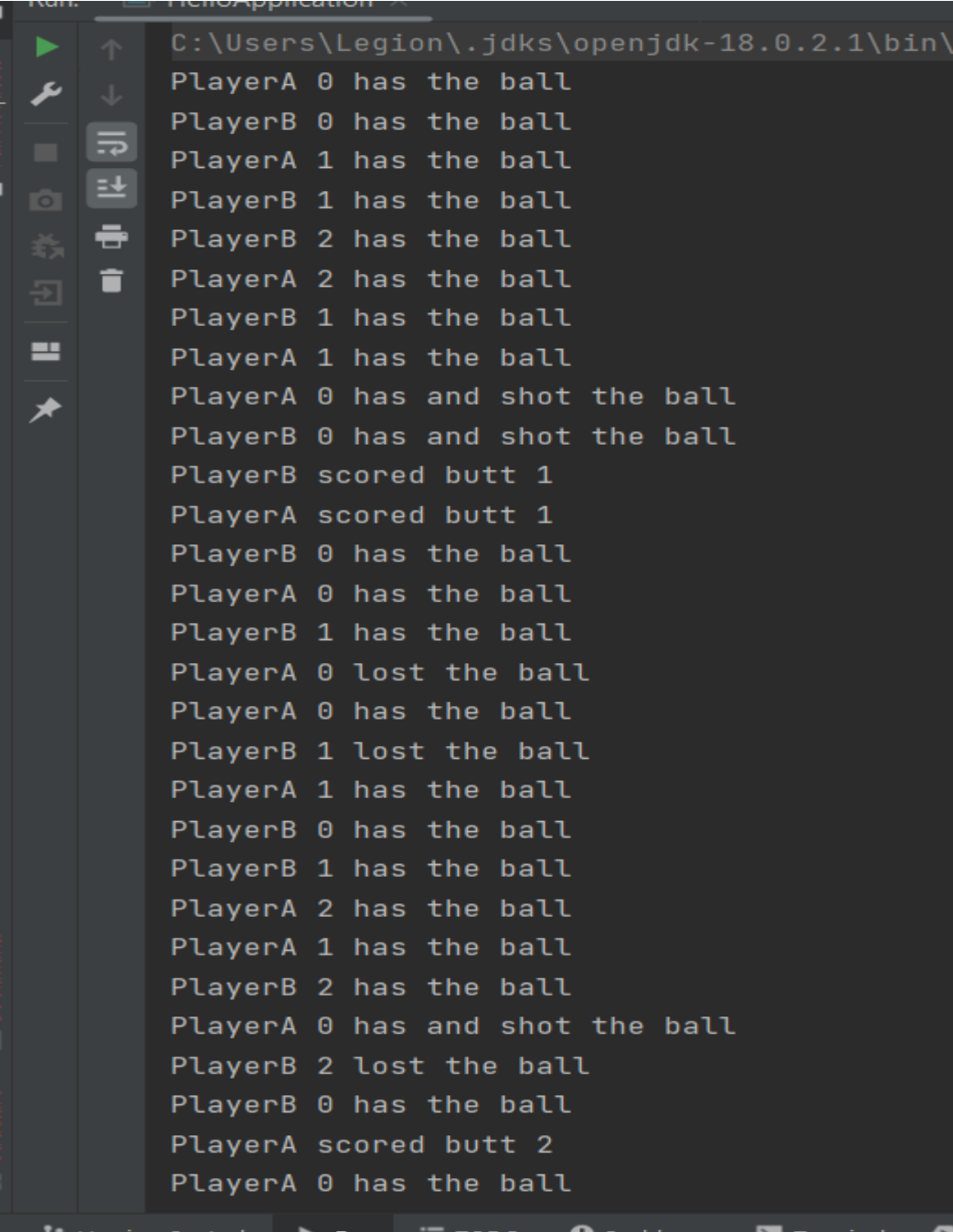
```
PlayerA 0 has and shot the ball
PlayerB 0 lost the ball
PlayerB 0 has the ball
PlayerA 1 has the ball
PlayerA 2 has the ball
PlayerB 1 has the ball
PlayerA 1 has the ball
PlayerB 2 has the ball
PlayerB 0 has and shot the ball
PlayerB 1 has the ball
PlayerB scored butt 9

Process finished with exit code 0
```

Version Control Run TODO Problems Terminal Services Profiler Build De

Build completed successfully in 1 sec, 174 ms (20 minutes ago)

- **Le console :**



```
Run: HelloApplication
C:\Users\Legion\.jdk\openjdk-18.0.2.1\bin\
PlayerA 0 has the ball
PlayerB 0 has the ball
PlayerA 1 has the ball
PlayerB 1 has the ball
PlayerB 2 has the ball
PlayerA 2 has the ball
PlayerB 1 has the ball
PlayerA 1 has the ball
PlayerA 0 has and shot the ball
PlayerB 0 has and shot the ball
PlayerB scored butt 1
PlayerA scored butt 1
PlayerB 0 has the ball
PlayerA 0 has the ball
PlayerB 1 has the ball
PlayerA 0 lost the ball
PlayerA 0 has the ball
PlayerB 1 lost the ball
PlayerA 1 has the ball
PlayerB 0 has the ball
PlayerB 1 has the ball
PlayerA 2 has the ball
PlayerA 1 has the ball
PlayerB 2 has the ball
PlayerA 0 has and shot the ball
PlayerB 2 lost the ball
PlayerB 0 has the ball
PlayerA scored butt 2
PlayerA 0 has the ball
```

**//Erreur Photos il n'y a pas d'erreur //**