TP1 FRAMEWORK

Ahlem KAABI

2.1 Hello World

1/ Téléchargement du projet Atom :

git clone https ://github.com/thevpc/atom.git

2/

L'authentification "anonymous" (anonyme) permet un accès public en lecture seule sans nécessiter d'identifiants.

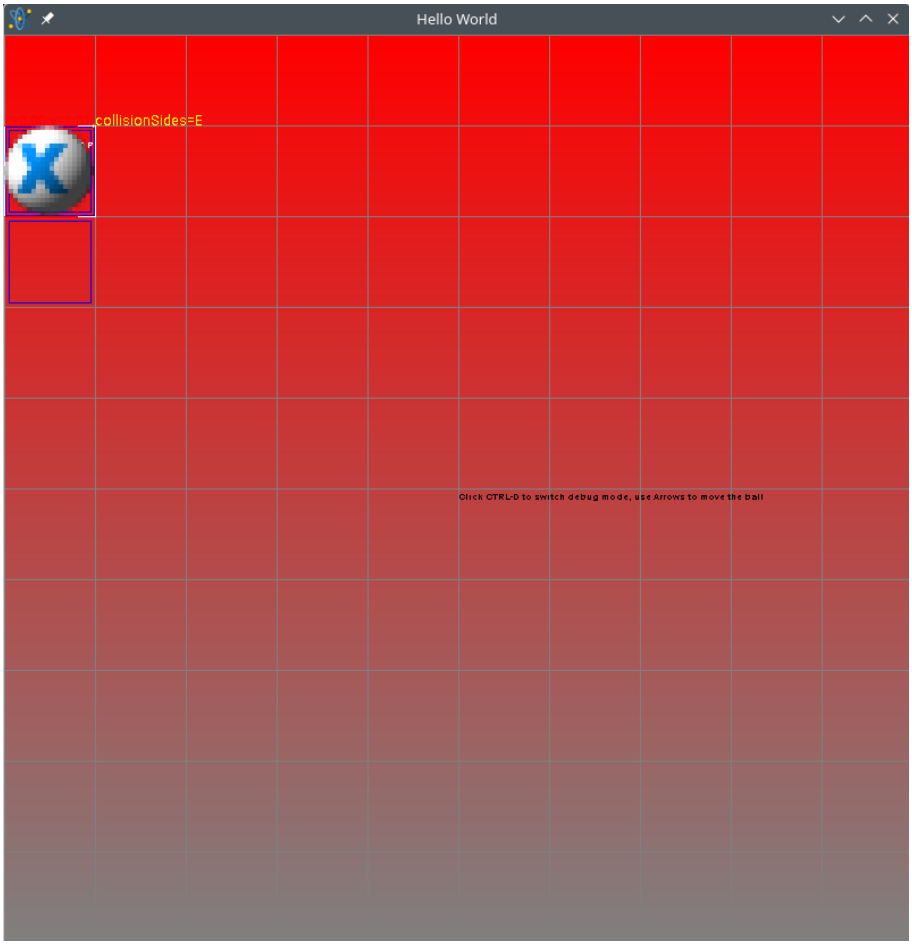
```
git clone https://github.com/thevpc/atom.git
```

- Aucun login/mot de passe requis
- Accès libre au code source
- Téléchargement possible via l'interface web ou ligne de commande

Simplicité : accès au projet en une commande

1. **Uniformité** : Toute la classe travaille sur la même base
2. **Focus sur l'objectif** : L'accent est mis sur l'apprentissage d'Atom, pas sur Git
3. **Respect du cadre open-source** : Le framework reste accessible à tous

3/



2.2 Architecture

Le projet est construit selon les modèles 3tiers et MVC

— **Presentation (P)** : responsable de l'interaction avec l'utilisateur
— **Métier (B)** : reste, c'est les fonctionnalités indépendamment de comment interagir avec user/domaine
— **Accès aux données (D)** : responsable de stockage et de récupération des données

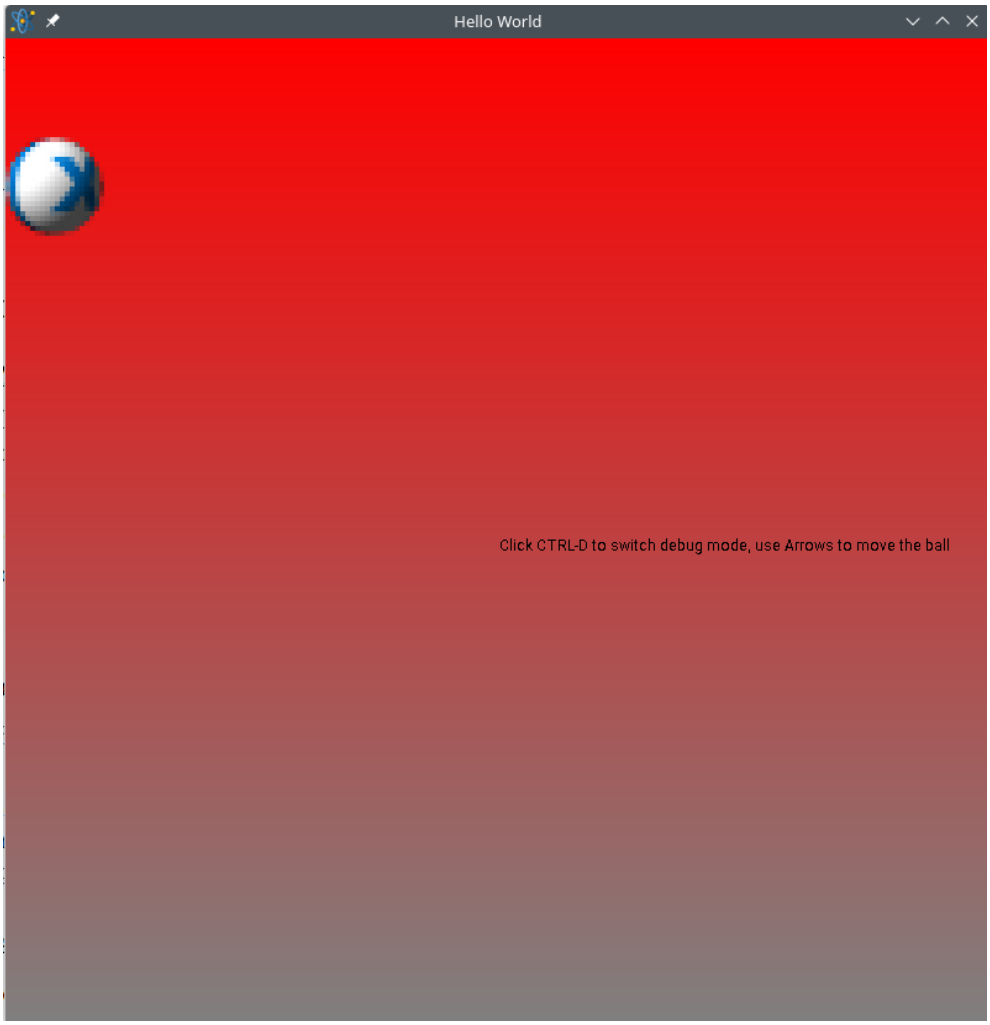
MVC : Model View Controller

— **Model (M)** : responsable de représenter l'état du système
— **Vue (V)** : responsable de la représentation du système pour l'utilisateur (perception), prend connaissance automatiquement des changements du modèle
— **Controleur (C)** : responsable de l'interaction avec le système/user (contrôle de comportement), modife le modèle

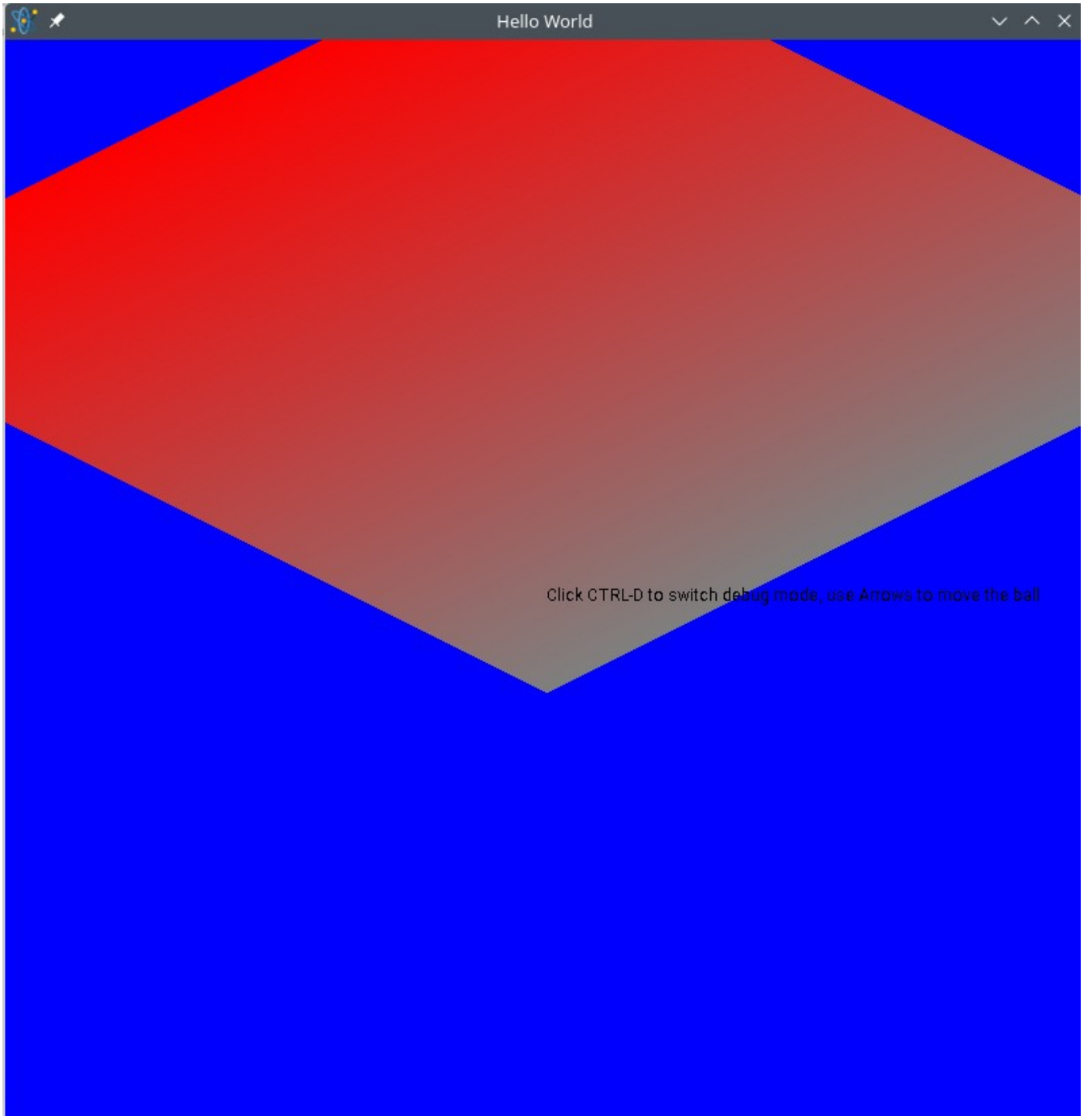
2.3 Composants

Atom introduit plusieurs concepts qui sont décrits ci-après :
— **Layout** : responsable d'afficher un calque et d'intéragir avec ce calque (click/clavier) : View+ Controller
— **SpriteView** : reponsable d'afficher/dessiner un sprite : View/Presentation
— **Scene** : reponsable de dessiner toute une scene (View/Presentation) : ensemble de sprite views + layouts
— **Game** : responsable de dessiner tout le jeu (ensemble de scenes) : View/Presentation
— **Sprite** : responsable de tracer l'état d'un objet mobile dans une scène : Model
— **SceneEngine** : responsable de garantir l'interaction entre l'environnement de la scène et les objets mobiles (collisions,
deplacement, ..) : Business d'une scène
— **GameEngine : Responsable de gérer l'interaction entre les scènes (Business)**
— **Controller** : responsable de gérer l'interaction souris & clavier pour une scène (Controller)
— SpriteCollisionManager : Responsable de traiter les collisions pour un sprite donné (Controller/Business)
— **SceneCollisionManager** : Responsable de traiter les collisions pour di◆érents sprites dans une scène (Controller/Business)
— **SpriteTask** : Tache répetitive réalisée (à chaque tic d'horloge) par un sprite. Par exemple un MoveTask va additionner un delta x,y à la position x,y (Controlller/Business)
Un diagramme de classe est fourni sous "docs" vous permettra de mieux apprécier la décomposition modulaire de Atom.

CTRL-D : Dessine une grille grise ou la supprimer sur toutes les tuiles visibles



CTRL-I: change la scene



..

Fonctionnalités principales de DebugLayer :

1. Affichage d'informations de débogage visuelles :

    a) Grille (ShowGrid) :

    b) Murs et obstacles (ShowWall) :

    c) Tuiles des sprites (ShowSpriteTiles) :

2. Informations sur les sprites :

a) Bornes des sprites (ShowSpriteBounds) :

b) Vélocité et direction (ShowSpriteVelocity) :

c) Chemins de déplacement (ShowSpritePath) :

3. Système de navigation :

a) Zones de défilement (ShowScrollBorders)

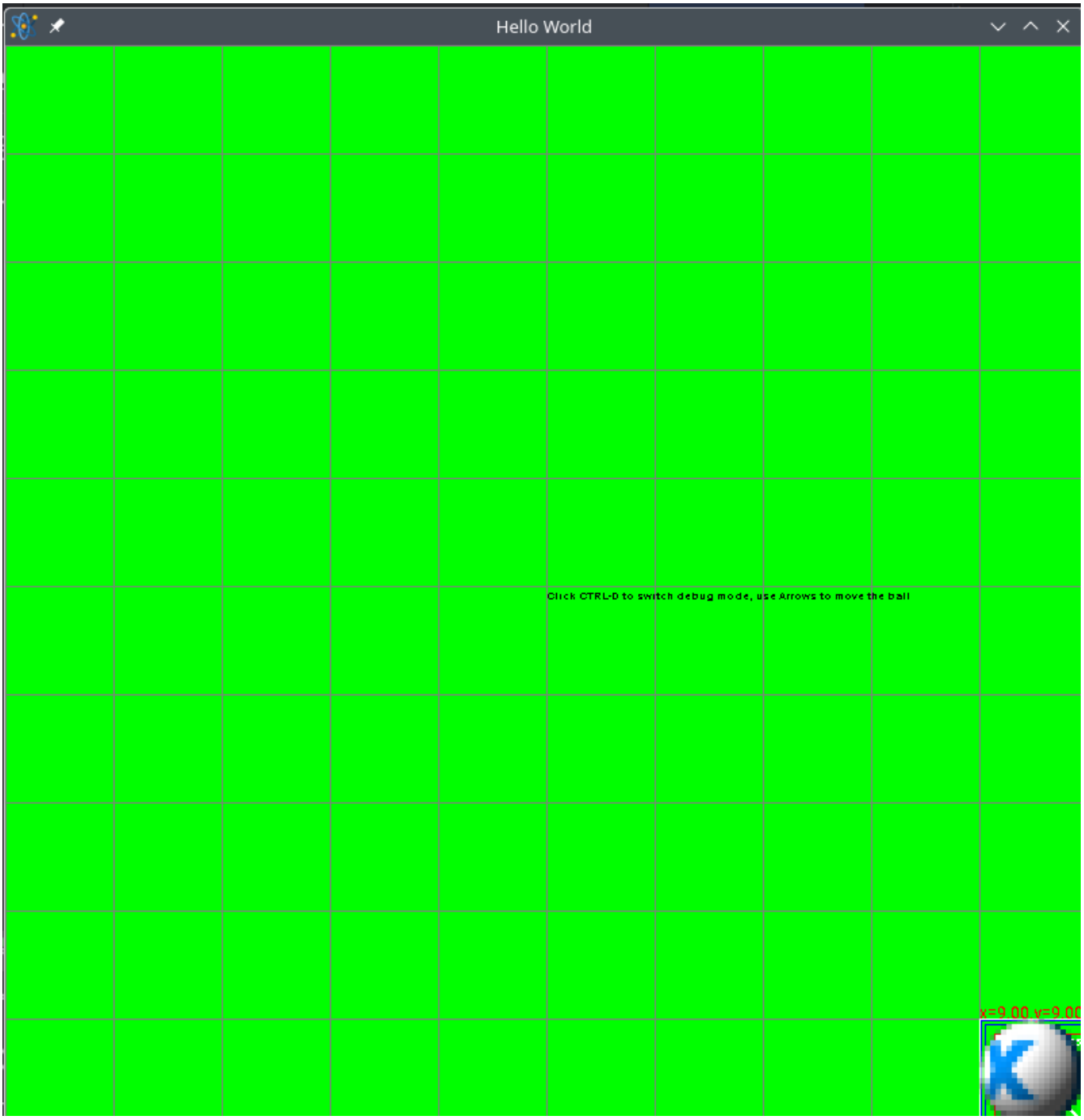
4. Fonctionnalités avancées :

a) HeatMap (ShowHeatMap) :

b) Gestion des composites :

2/ Changer la couleur de l’arrière plan en utilisant FillBoardColorLayer

```
@OnSceneStarted  no usages  thevpc +1 *
private void init() {
    scene.addLayer(new FillBoardColorLayer(Color.GREEN));
```

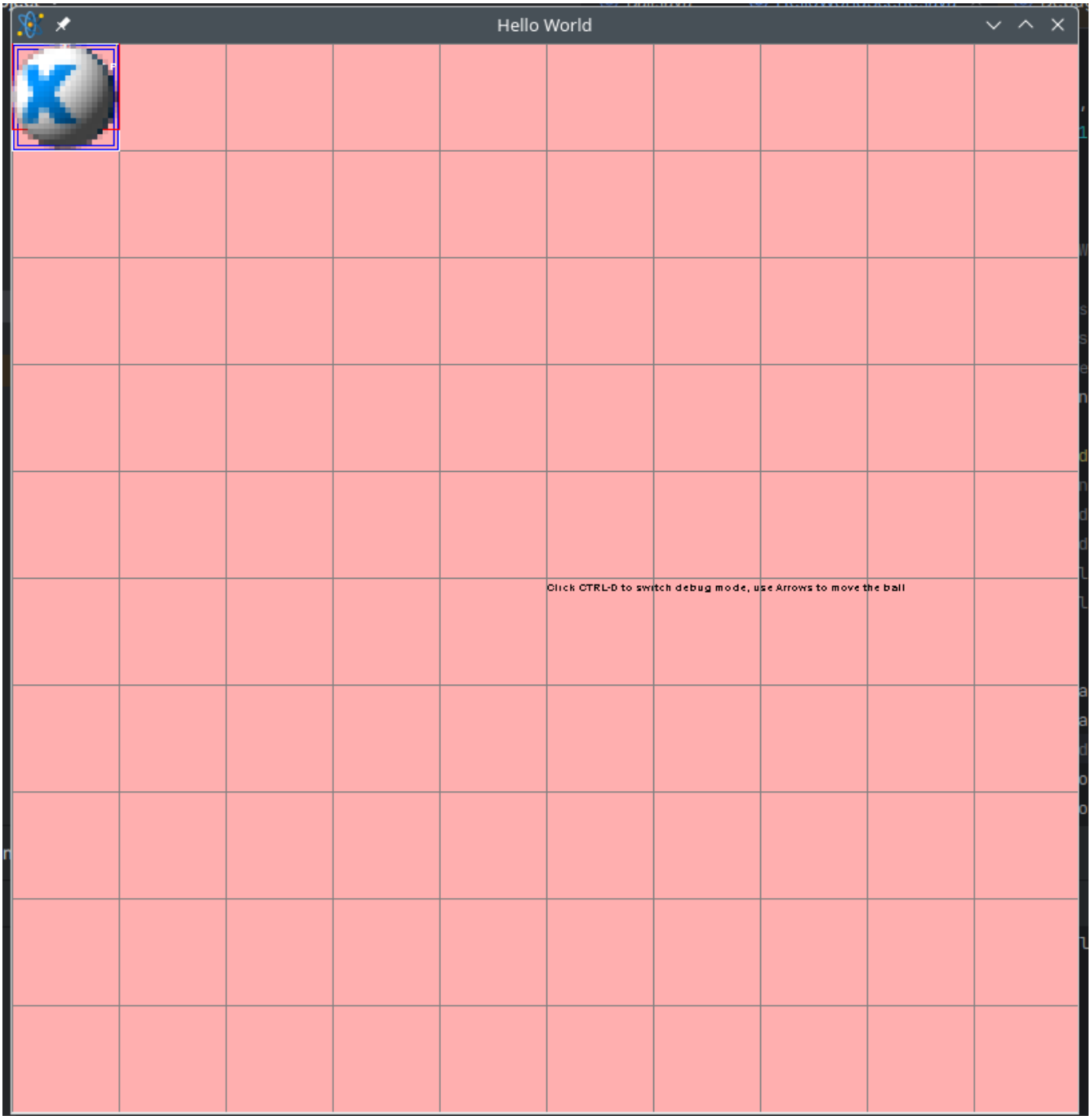


.

3/

Changer la couleur de l’arrière plan en utilisant FillScreenColorLayer

```
scene.addLayer(new FillScreenColorLayer(Color.PINK));
```



4/

5/

```
scene.addLayer(new ImageBoardLayer( imageMap: "/img1.jpg"));
```

3.2 Ajouter un label de score Slabel

1/

```
scene.addComponent(
        new SLabel( value: "Vies : 3")
                .setLocation(Point.ratio( x: 0.1f,  y: 0.1f))

);
```

.

2/

```
@OnNextFrame  no usages  new *
private void aChaqueTic(){
    Sprite ball = sceneEngine.findSpriteByName( s: "Ball0");
    if (ball != null) {
        int vies = ball.getLife();
        viesLabel.setText("Vies : " + vies);
    }

}
```

3.3 Ajouter une balle

1/

Ajouter une seconde balle (Ball2) qui se déplacera initalement dans la direction inverse de la première balle et qui est

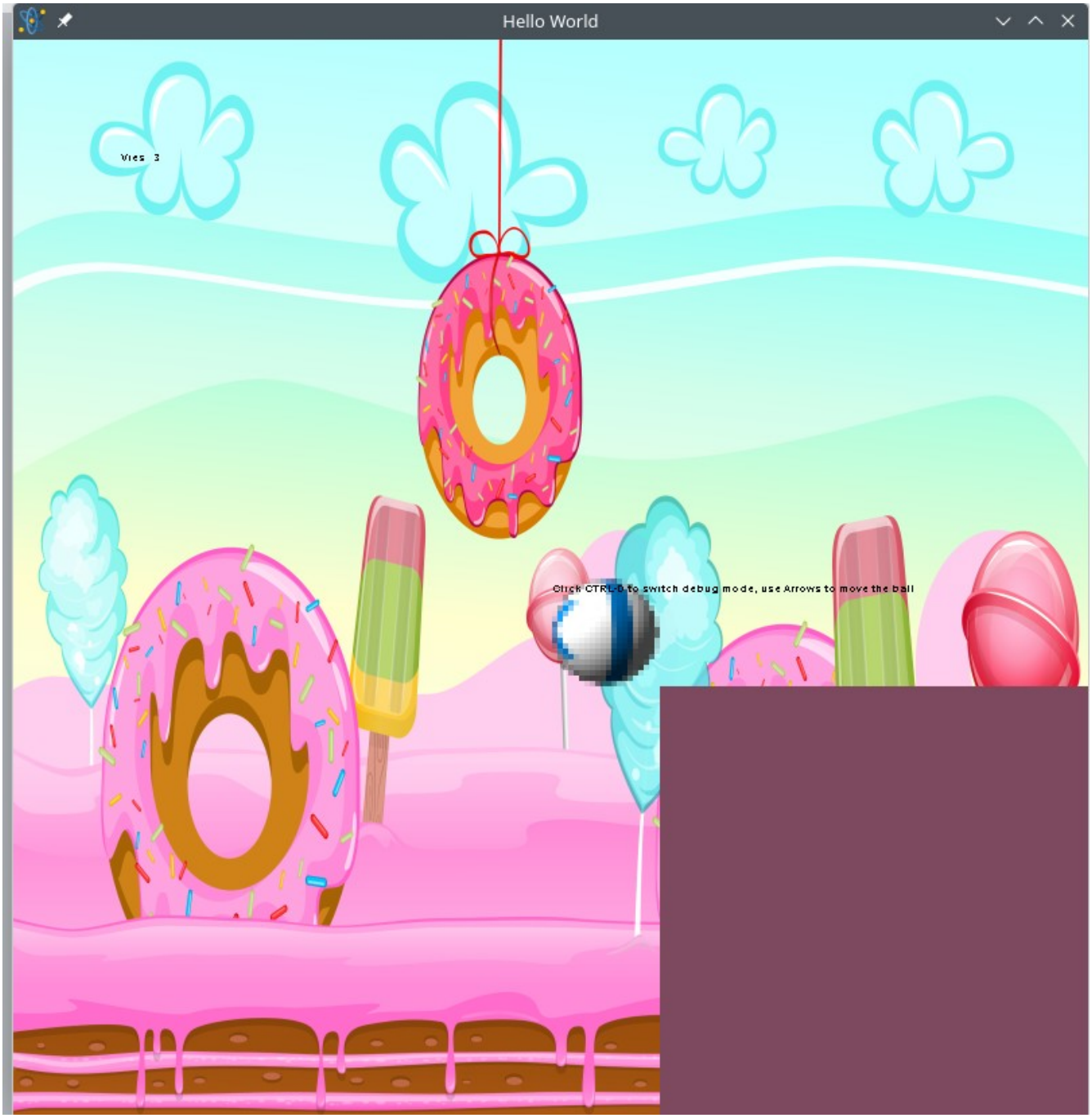
deux fois plus rapide et deux fois plus grosse (en taille). Pour ce faire on créera une nouvelle classe Ball2.

```
@AtomSprite(  no usages  new *
        name = "Ball2",
        kind = "Ball2",
        sceneEngine = "hello",
        x=8,
        y=8,
        direction = -Math.PI/4,
        speed = 0.4,
        mainTask = MoveSpriteMainTask.class,
        collisionTask = StopSpriteCollisionTask.class
//        collisionTask = BounceSpriteCollisionTask.class

)

public class Ball2 {
        @Inject  2 usages
        Sprite sprite;
        @Inject  no usages
        SceneEngine sceneEngine;
        Scene scene;  no usages

        @OnInit  no usages  new *
        private void init(){
            sprite.setLocation( v: 8, v1: 8);
            sprite.setSize( v: 4,  v1: 4);
```

.

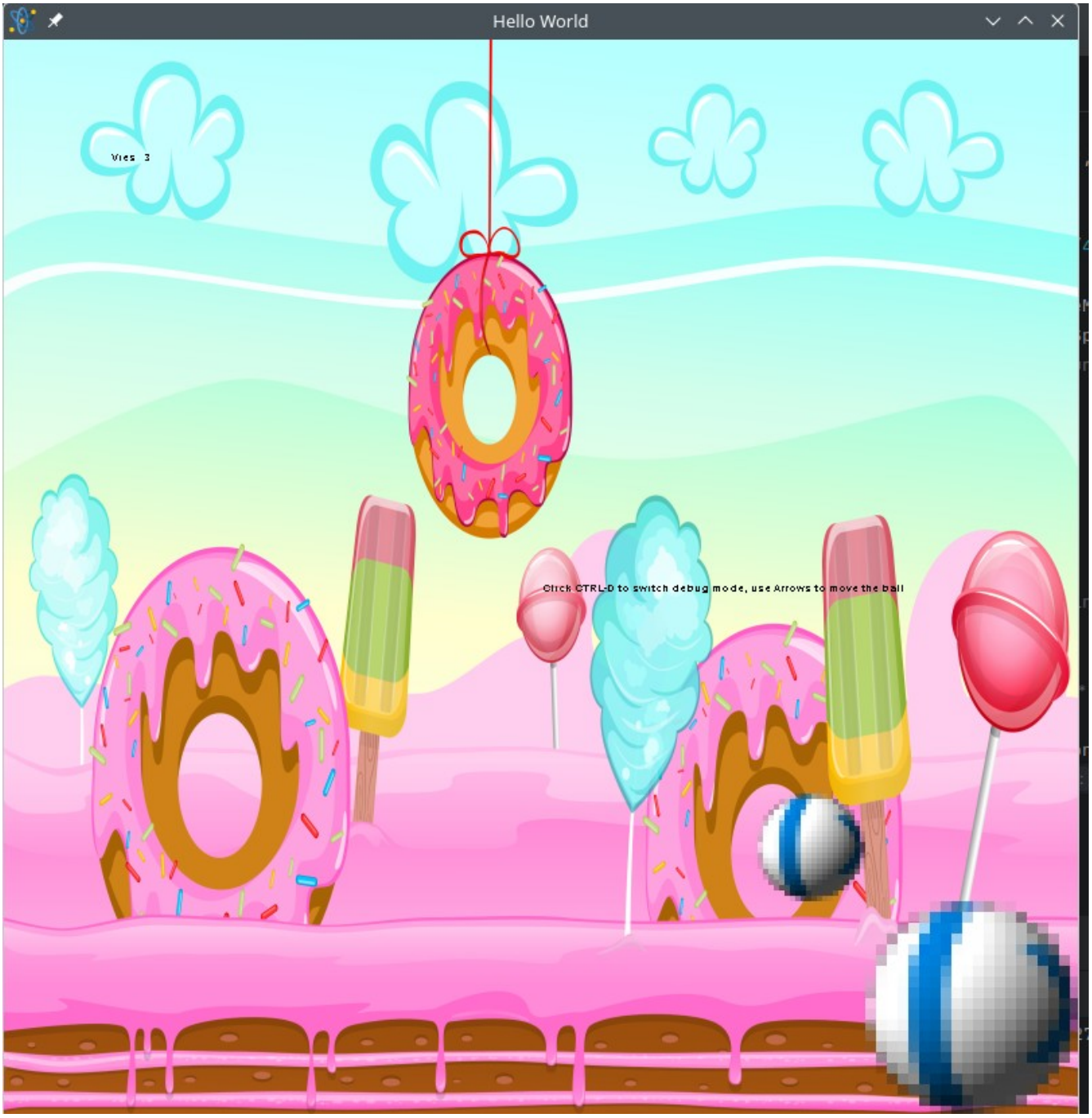
2/

La balle est affiche en un carre car on a definit in nouveau kind “Ball2”



par contre si on met le meme kind “Ball1”

on a :

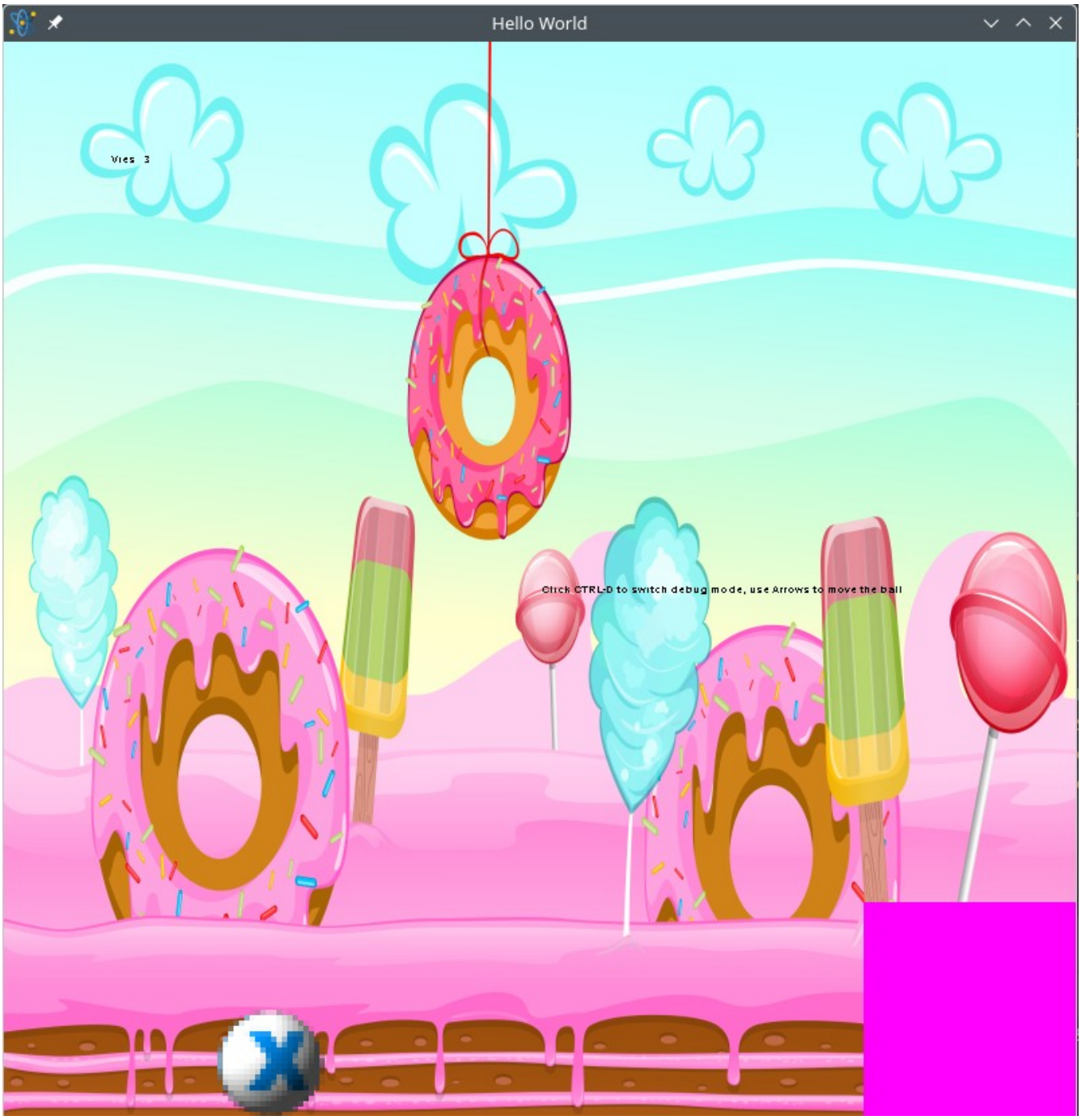

Hello World
Vies : 3
Click CTRL-D to switch debug mode, use Arrows to move the ball

.

3/

```
    @OnInit  no usages  new *
    private void init(){
        sprite.setLocation( v: 8, v1: 8);
        sprite.setSize( v: 2,  v1: 2);
        scene.setSpriteView(SpriteFilter.byName( ...ids: "Ball2"), new DefaultSpriteView() {  new *
            @Override  no usages  new *
            public void draw(SpriteDrawingContext context) {
                Graphics2D g = (Graphics2D) context.getGraphics();
                g.setColor(Color.MAGENTA);
                g.fill(context.getSpriteShape());
            }
        });


    }

}
```

.


Hello World
Vies : 3
Click CTRL-D to switch debug mode, use Arrows to move the ball

4/

```
scene.setSpriteView(SpriteFilter.byKind( ...kinds: "Ball"), new ImageSpriteView( imageMap: "/ball.png",  cols: 8,  rows: 4));
scene.setSpriteView(SpriteFilter.byKind( ...kinds: "Ball2"), new ImageSpriteView( imageMap: "/ball22.png",  cols: 5,  rows: 2));
```

.

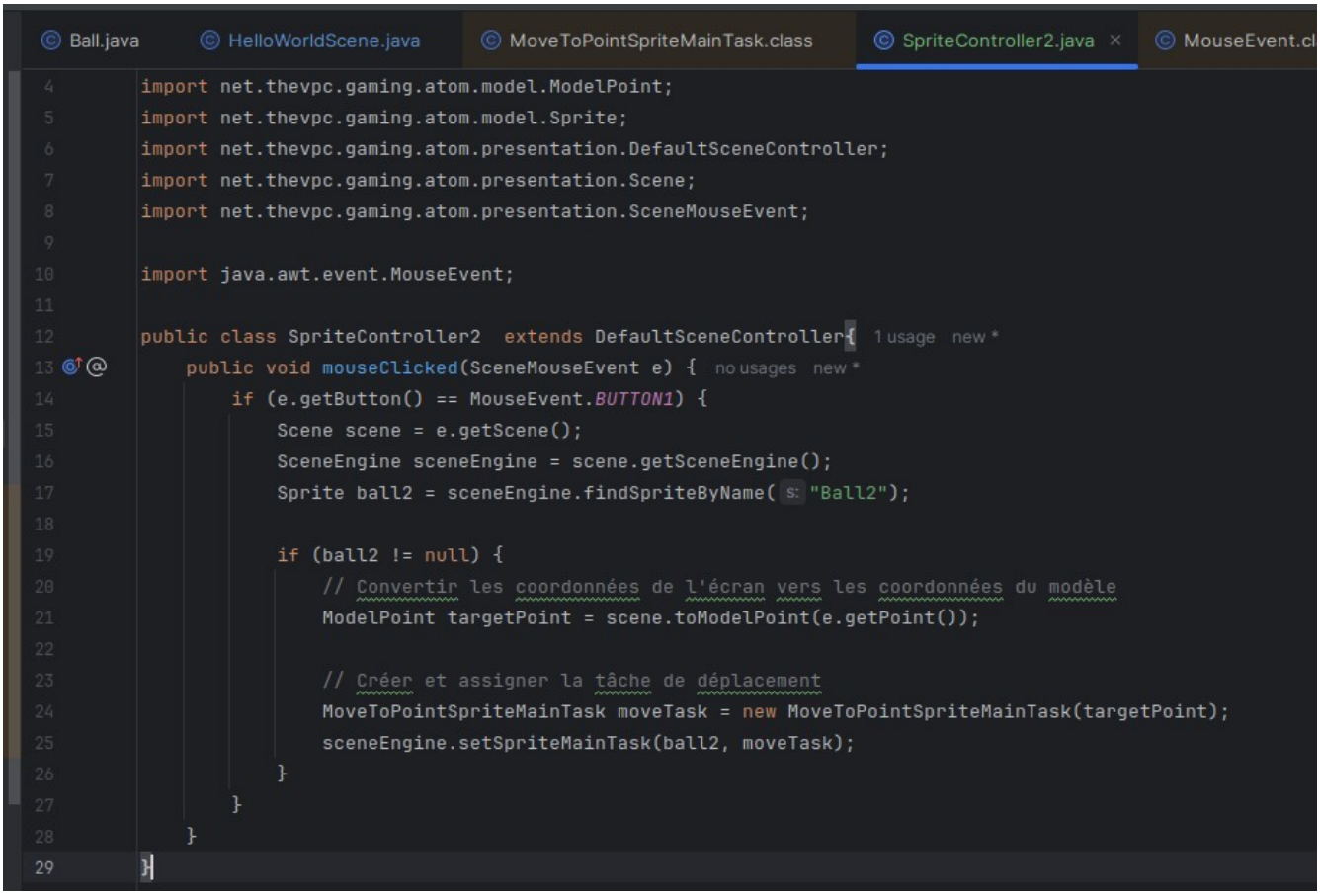
.

4 Commande utilisateur

1/

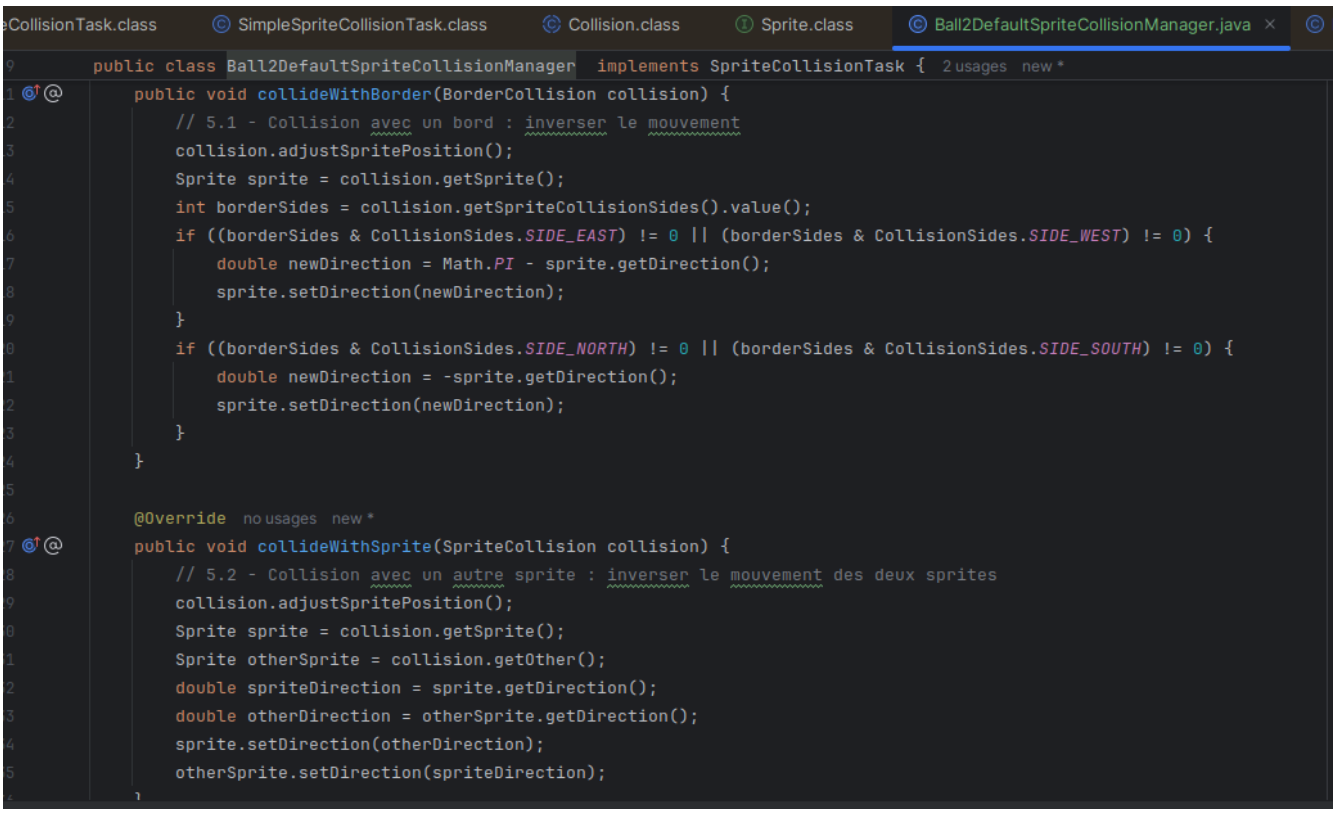
```
scene.addController(new SpriteController(SpriteFilter.byName( ...ids: "Ball2")).setESDFLayout());
```

2/



5 Collision

1/2/


eCollisionTask.class
SimpleSpriteCollisionTask.class
Collision.class
Sprite.class
Ball2DefaultSpriteCollisionManager.java
public class Ball2DefaultSpriteCollisionManager implements SpriteCollisionTask { 2 usages new *
public void collideWithBorder(BorderCollision collision) {
// 5.1 - Collision avec un bord : inverser le mouvement
collision.adjustSpritePosition();
Sprite sprite = collision.getSprite();
int borderSides = collision.getSpriteCollisionSides().value();
if ((borderSides & CollisionSides.SIDE_EAST) != 0 || (borderSides & CollisionSides.SIDE_WEST) != 0) {
double newDirection = Math.PI - sprite.getDirection();
sprite.setDirection(newDirection);
}
if ((borderSides & CollisionSides.SIDE_NORTH) != 0 || (borderSides & CollisionSides.SIDE_SOUTH) != 0) {
double newDirection = -sprite.getDirection();
sprite.setDirection(newDirection);
}
}
@Override no usages new *
public void collideWithSprite(SpriteCollision collision) {
// 5.2 - Collision avec un autre sprite : inverser le mouvement des deux sprites
collision.adjustSpritePosition();
Sprite sprite = collision.getSprite();
Sprite otherSprite = collision.getOther();
double spriteDirection = sprite.getDirection();
double otherDirection = otherSprite.getDirection();
sprite.setDirection(otherDirection);
otherSprite.setDirection(spriteDirection);

.

6 Les scenes

7 Creation de sprites

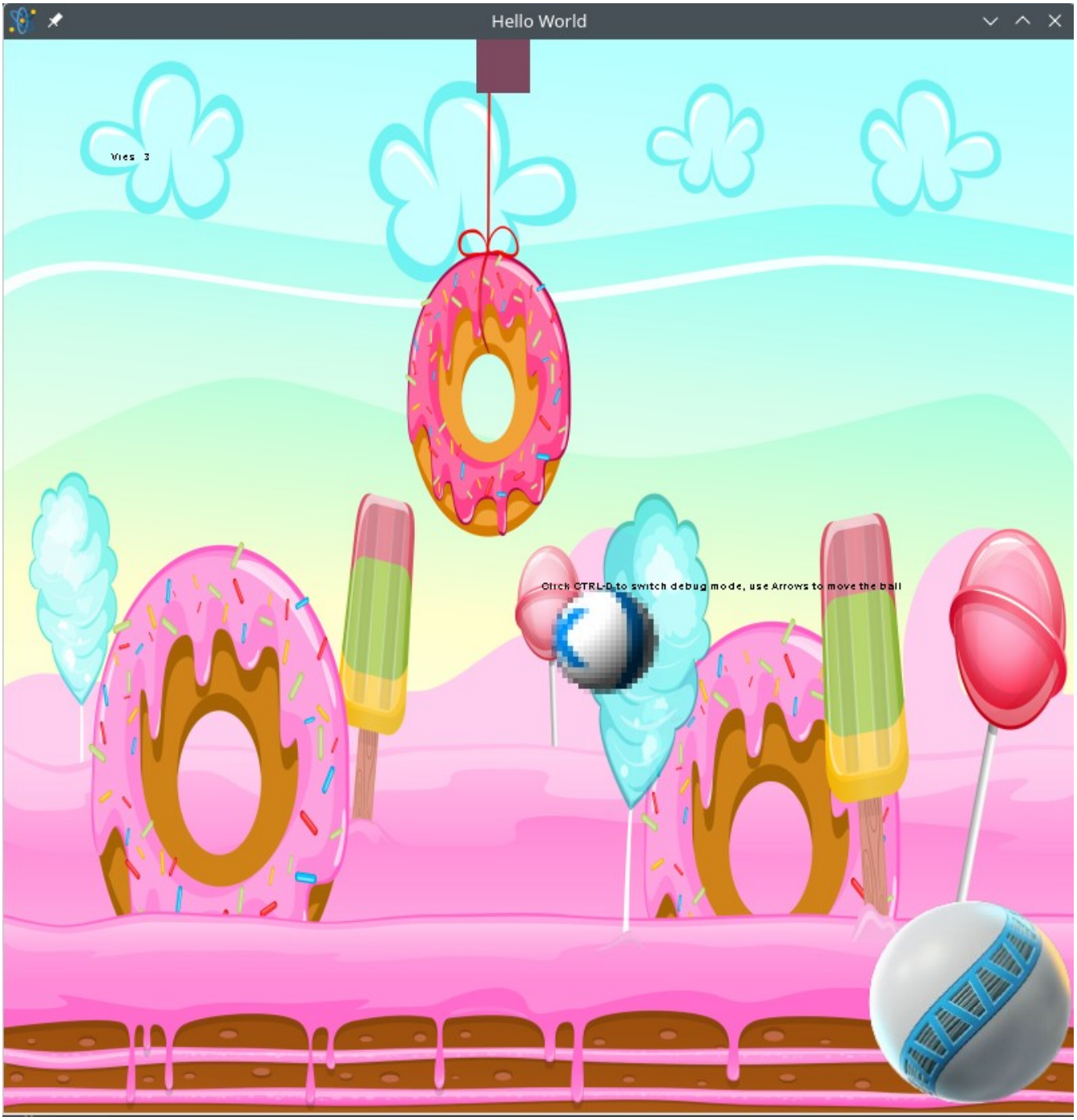
.

```java
import net.thevpc.gaming.atom.model.Sprite;

@AtomSprite(  no usages  new *
        name = "Missile",
        kind = "Missile",
        sceneEngine = "hello",
        speed = 0.2,
        mainTask = MoveSpriteMainTask.class,
        collisionTask = MissileCollisionTask.class
)
public class Missile {
    @Inject  2 usages
    private Sprite sprite;

    @Inject  no usages
    private SceneEngine sceneEngine;

    @OnInit  no usages  new *
    private void init() {
        sprite.setLife(3);
        sprite.setSize( v: 0.5,  v1: 0.5);
    }
}
```

.

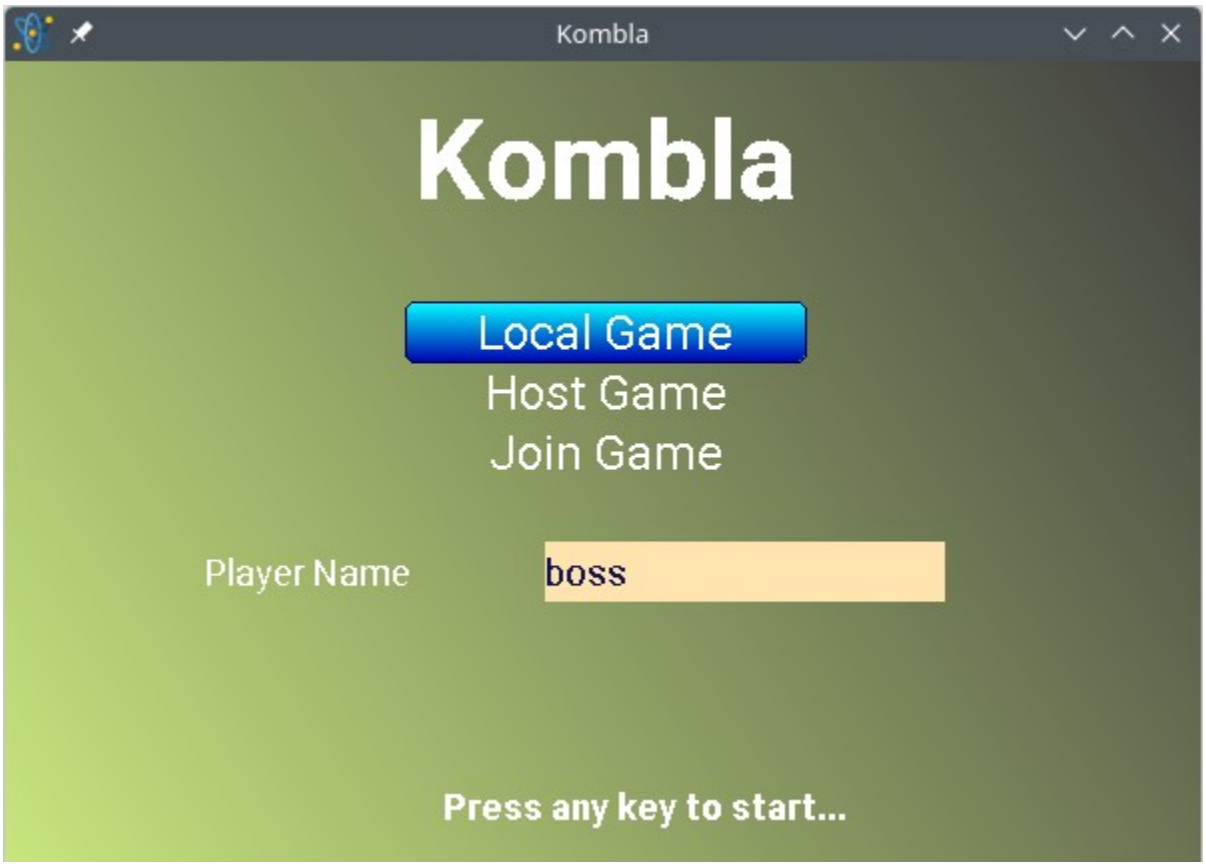
```java
package net.thevpc.gaming.helloworld;

import net.thevpc.gaming.atom.engine.collisiontasks.SpriteCollisionTask;
import net.thevpc.gaming.atom.engine.collisiontasks.SpriteCollision;
import net.thevpc.gaming.atom.engine.collisiontasks.BorderCollision;
import net.thevpc.gaming.atom.model.Sprite;

public class MissileCollisionTask implements SpriteCollisionTask {  1 usage  new *

    @Override  no usages  new *
    public void collideWithBorder(BorderCollision collision) {
        collision.getSceneEngine().removeSprite(collision.getSprite().getId());
    }

    @Override  no usages  new *
    public void collideWithSprite(SpriteCollision collision) {
        Sprite missile = collision.getSprite();
        collision.getSceneEngine().removeSprite(missile.getId());
    }
}
```

.

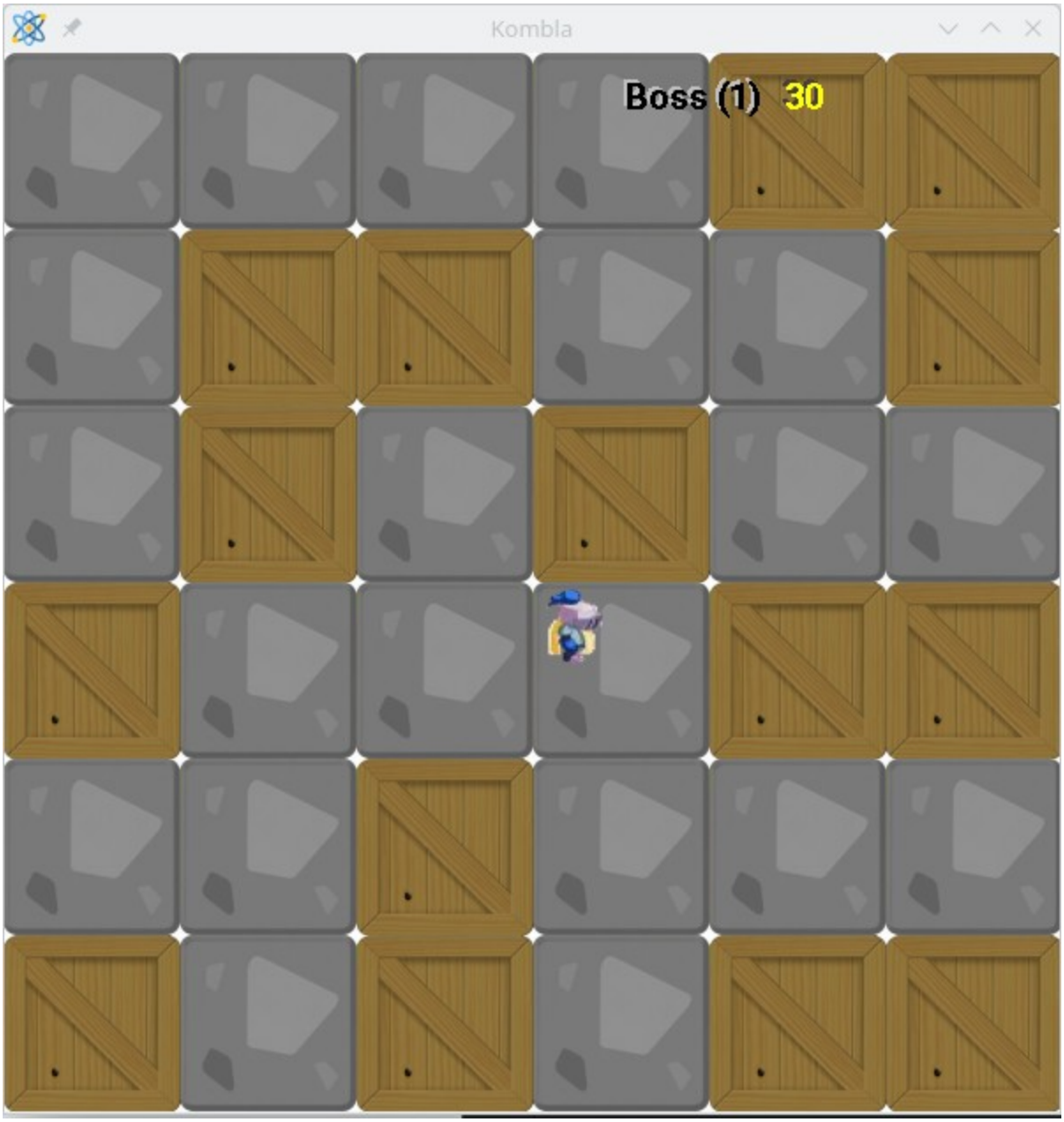

Hello World
Vies : 3
Click CTRL-D to switch debug mode, use Arrows to move the ball

Troisième partie

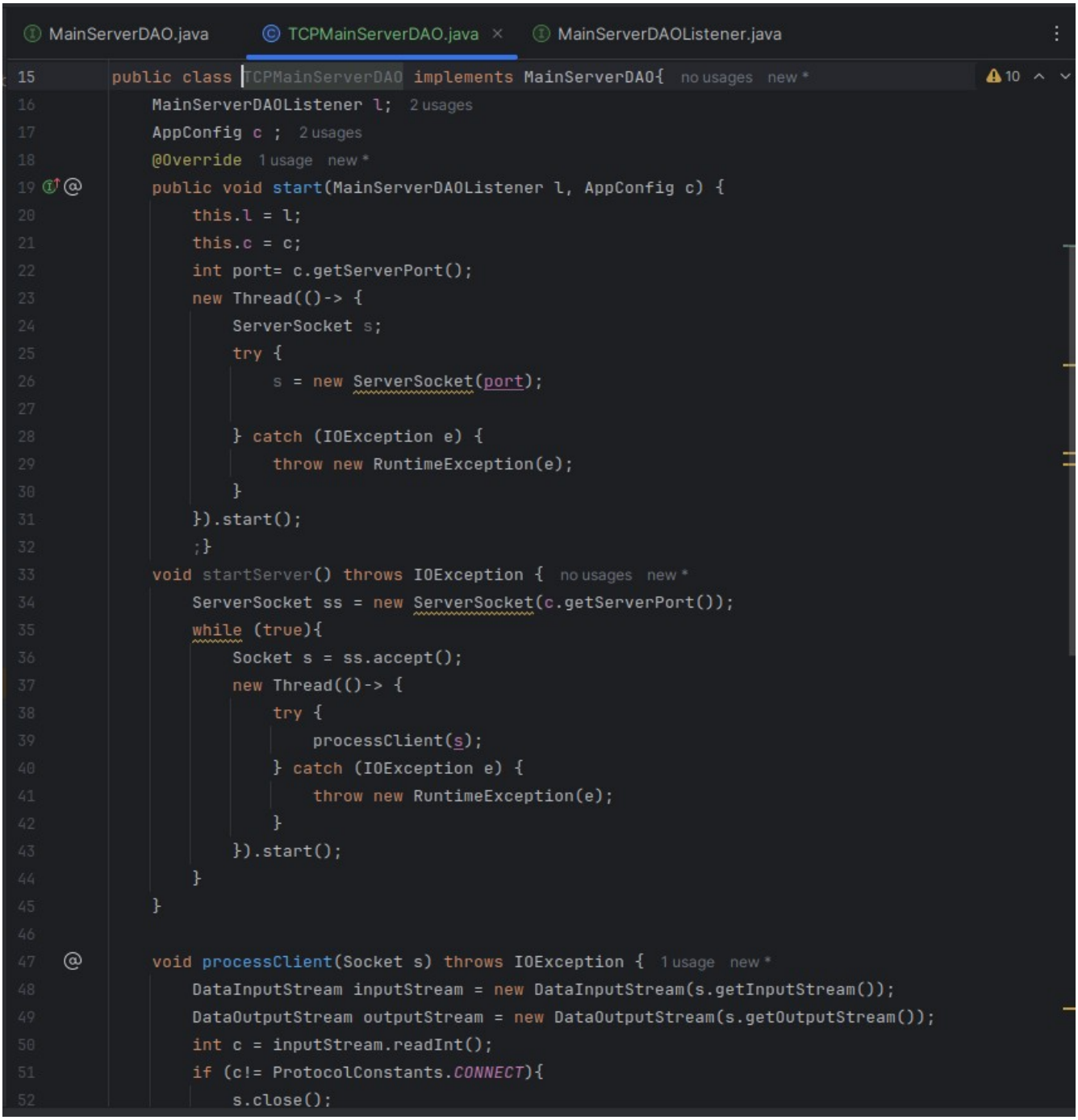
SocketsLe jeu BomberMan



.

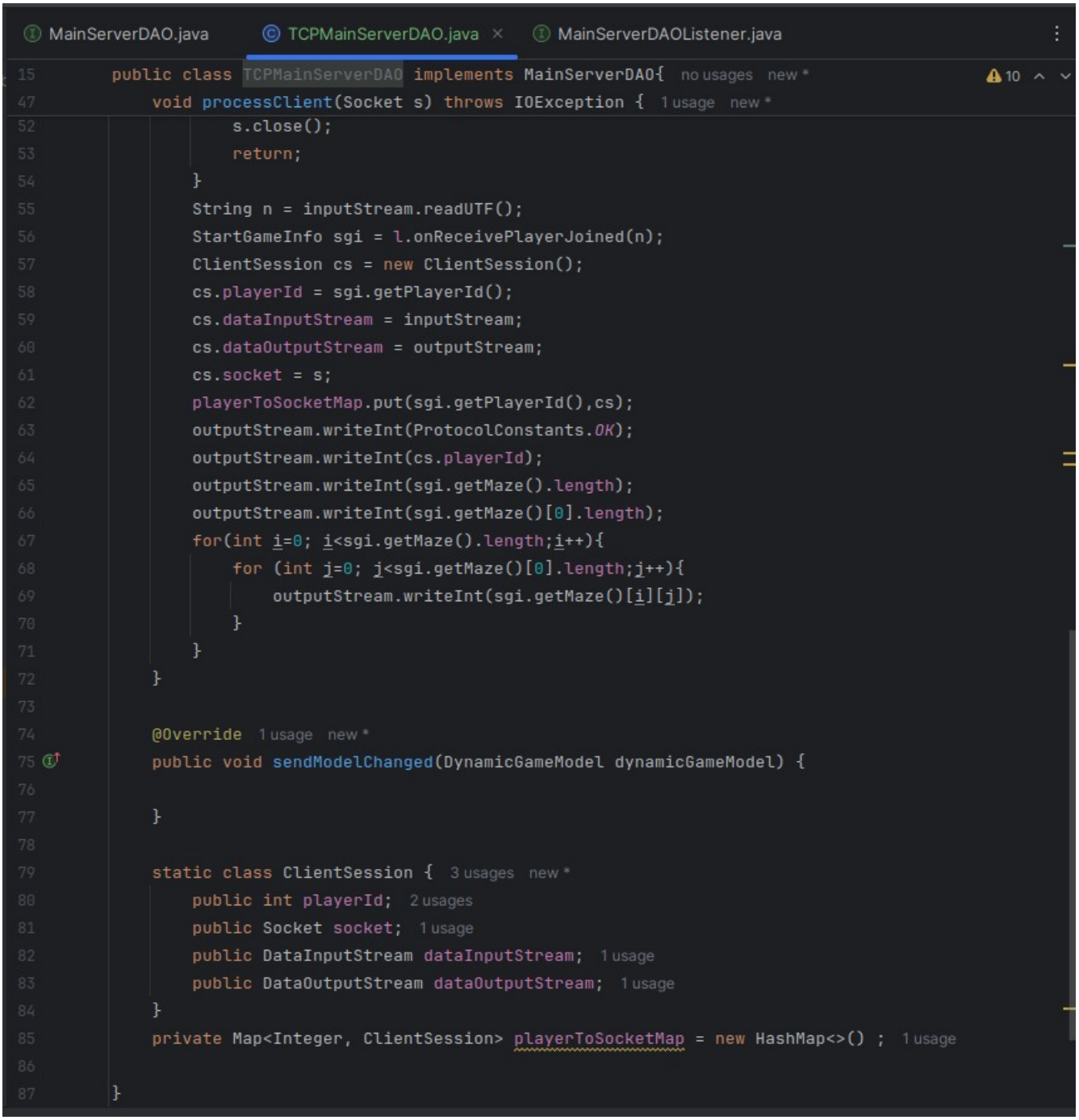


9 Jeu coté Serveur:


MainServerDAO.java
TCPMainServerDAO.java
MainServerDAOListener.java
public class TCPMainServerDAO implements MainServerDAO{ no usages new *
MainServerDAOListener l; 2 usages
AppConfig c ; 2 usages
@Override 1 usage new *
public void start(MainServerDAOListener l, AppConfig c) {
this.l = l;
this.c = c;
int port= c.getServerPort();
new Thread(()-> {
ServerSocket s;
try {
s = new ServerSocket(port);
} catch (IOException e) {
throw new RuntimeException(e);
}
}).start();
;}
void startServer() throws IOException { no usages new *
ServerSocket ss = new ServerSocket(c.getServerPort());
while (true){
Socket s = ss.accept();
new Thread(()-> {
try {
processClient(s);
} catch (IOException e) {
throw new RuntimeException(e);
}
}).start();
}
}
void processClient(Socket s) throws IOException { 1 usage new *
DataInputStream inputStream = new DataInputStream(s.getInputStream());
DataOutputStream outputStream = new DataOutputStream(s.getOutputStream());
int c = inputStream.readInt();
if (c!= ProtocolConstants.CONNECT){
s.close();

-

-



•

1. Créer une classe SocketMainServerDAO qui implémente l’interface MainServerDAO . Proposer un emplacement (package)

logique pour cette classe; → sous Server/dal

2. Créer une classe interne ClientSession stockant l'id du player, la socket du player et les flux d'entrée sortie dans le cas de TCP

```
static class ClientSession {  3 usages  new *
    public int playerId;  2 usages
    public Socket socket;  1 usage
    public DataInputStream dataInputStream;  1 usage
    public DataOutputStream dataOutputStream;  1 usage
}
```

.

3. Ajouter la défnition suivante des sessions clients en tant qu'attribut de la classe. Discuter l'intérêt de la notion de Session dans ce contexte

```
    StartGameInfo sgi = l.onReceivePlayerJoined(n);
    ClientSession cs = new ClientSession();
    cs.playerId = sgi.getPlayerId();
```

la Session ;

1. **Centralise** toutes les informations d'un client
2. **Facilite** la communication ciblée
3. **Améliore** la maintenabilité du code
4. **Permet** une gestion propre des connexions/déconnexions
5. **Sépare** les préoccupations réseau de la logique métier

```
static class ClientSession {  3 usages  new *
    public int playerId;  2 usages
    public Socket socket;  1 usage
    public DataInputStream dataInputStream;  1 usage
    public DataOutputStream dataOutputStream;  1 usage
}
private Map<Integer, ClientSession> playerToSocketMap = new HashMap<>() ;  1 usage
```

.

4. Implémenter la méthode start de sorte à initialiser les informations nécessaires (ci-après) puis de lancer l'écoute Socket dans un nouveau Thread. A la réception de chaque client, la méthode processClient(ClientSession) sera appelée

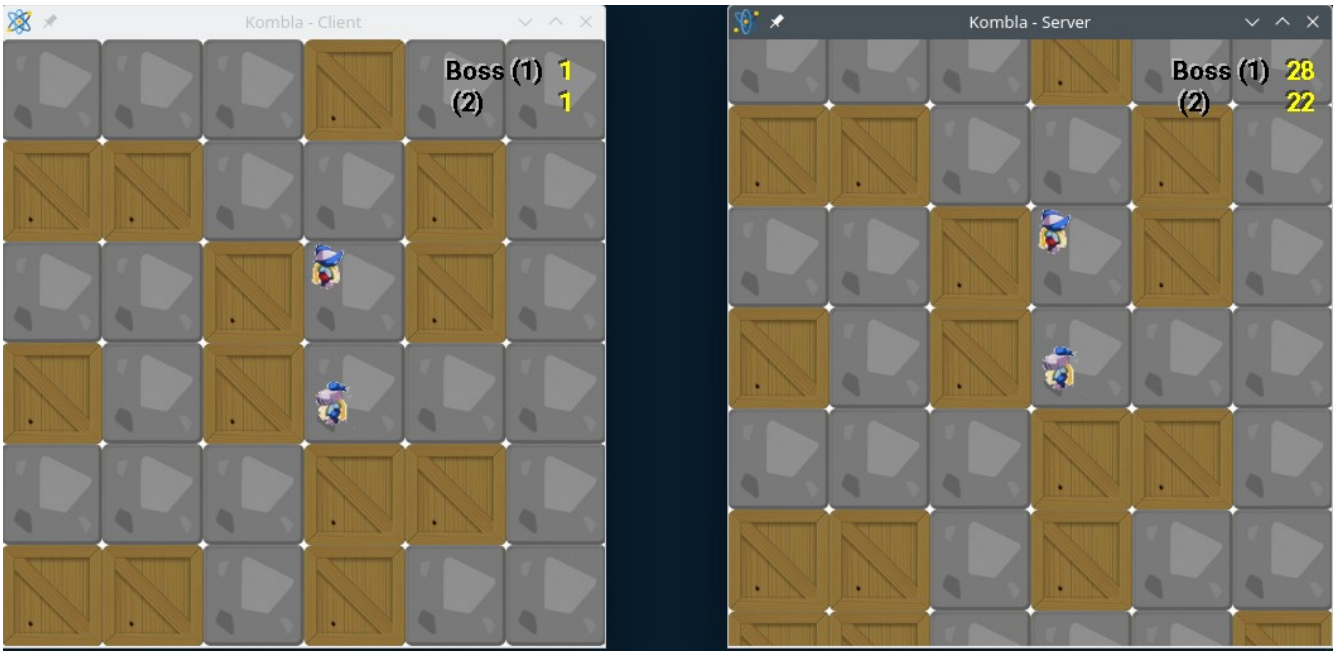
```
public void start(MainServerDAOListener l, AppConfig c) {
    this.l = l;
    this.c = c;
    int port= c.getServerPort();
    new Thread(()-> {
        ServerSocket s;
        try {
            s = new ServerSocket(port);
            startServer();

        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }).start();
    ;}
```

.

..

RMI :

mode Join:



.