Projet LU2IN002 - 2020-2021

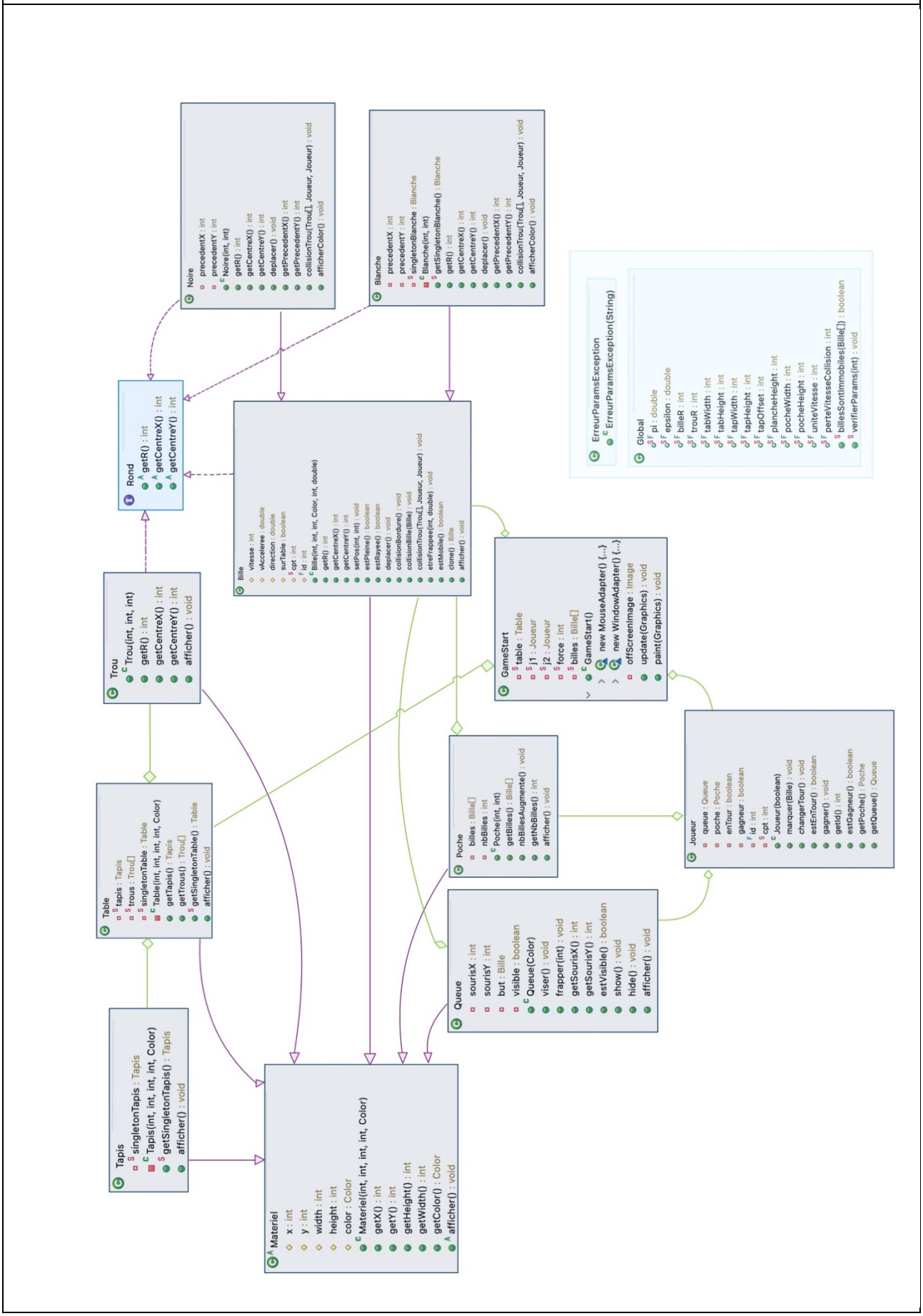
Numéro du groupe de TD/TME : Groupe 11

Nom : Xu	*Nom :* Chen
Prénom : Renwen	*Prénom :* Lucie
N° étudiant : 28705797	*N° étudiant :* 28610408

Thème choisi (en 2 lignes max.)
Nous avons choisi le billard comme thème de notre projet.

Description des classes et de leur rôle dans le programme (2 lignes max par classe)
Class Main : une classe qui permet au client de tester le projet. Nous avons le choix de voir les règles (taper en paramètre 1) ou commencer le jeu (taper en paramètres 2) Class GameStart : Classe qui contient contient tout ce qui est nécessaire à la simulation du jeu. Class Global : Classe qui contient toutes les variables statiques de notre projet plus deux méthodes : une méthode qui vérifie si tous les billes sont immobiles et une méthode qui vérifie les paramètres. Class ErreurParamsExeption : C'est une classe étendant de Exception Class Joueur : Nous avons 2 joueurs dans le projet, chacun possède une queue et une poche de billes. En suivant les règles, le joueur gagne quand sa poche est pleine et la bille noire est rentrée. Class Materiel : Classe mère (abstraite) de notre projet. Elle englobe presque tous les Classes de notre projet. Class Table : Classe chargée de créer la table de billard (singleton) qui est constitué d'un Tapis et 6 trous. Class Tapis : Classe chargée de créer le tapis vert (singleton). Class Poche: Chaque joueur possède une poche qui fait le compte du nombre de billes, distingués en pleine et rayée, qui sont rentrées dans les trous. Class Queue : Classe qui représente la queue dans le jeu de billard, c'est le matériel qu'on utilise pour frapper les billes, et celui-ci change de position selon la souris. Interface Rond : Interface qui donne les propriétés d'un matériel rond. Class Trou : Classe chargé de créer les trous (matériel rond) Class Bille : Les billes (rond) dans le jeu sont soit pleines soit rayées et peuvent se déplacer par frappe, puis il peut subir des collisions avec le bord de la table, les autres billes ou aussi les trous. Class Blanche : Hérite de Class Bille et qui représente la bille blanche dans le jeu, quand elle rencontre un trou, elle revient à sa dernière position immobile. Class Noire : Hérite de Class Bille et qui représente la bille noire dans le jeu. C'est la bille qui détermine le gagneur, le premier à le faire rentrer après que tous les billes soient rentrées est le gagnant.

Schéma UML des classes vision fournisseur (dessin "à la main" scanné ou photo acceptés)



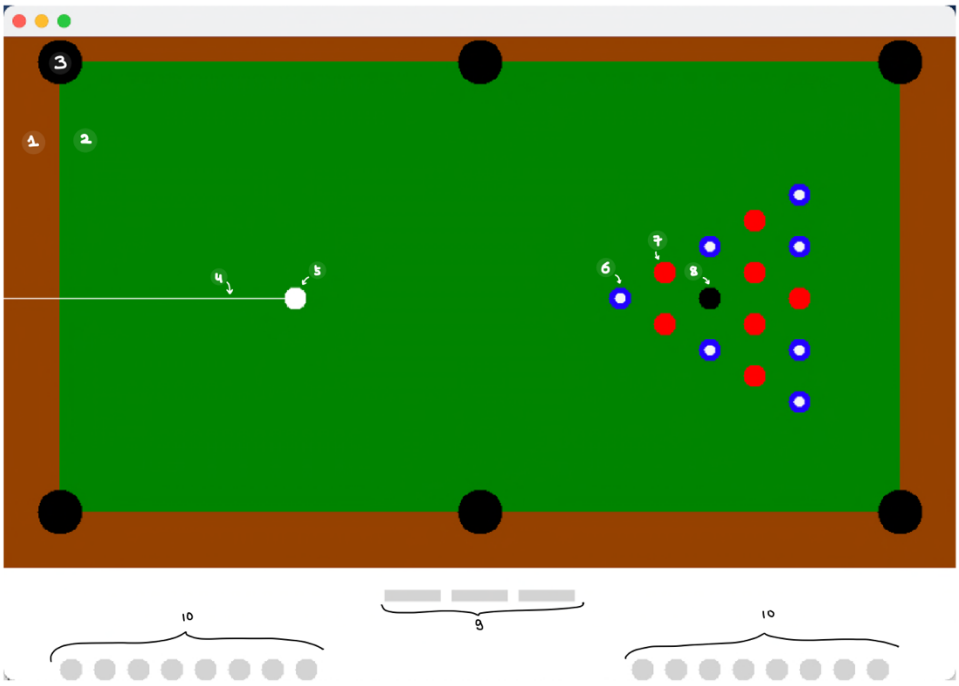
Checklist des contraintes prises en compte:	*Nom(s) des classe(s) correspondante(s)*
Classe contenant un tableau ou une ArrayList	Class Table Class GameStart Class Poche
Classe avec membres et méthodes statiques	Class Global Variable cpt de Class Joueur Variable cpt de Class Bille
Classe abstraite et méthode abstraite	Class Materiel Méthode afficher dans Classe Materiel
Interface	Interface Rond
Classe avec un constructeur par copie ou clone()	Class Bille

Définition de classe étendant Exception	`public class ErreurParamsException extends Exception {`
	`  public ErreurParamsException(String s) {`
	`  super(s);`
	`  }`
	`}`

Gestion des exceptions	Try catch dans Class GameStart et Class Main
Utilisation du pattern singleton	Class Tapis Class Table Class Blanche

Présentation de votre projet (max. 2 pages) : texte libre expliquant en quoi consiste votre projet.
Notre projet consiste de représenter le sport : le billard. Les règles du jeu ont été simplifiés pour qu'on se concentre sur les applications de la POO et de donner une vision générale de la représentation sans rentrer dans les détails. Nous avons choisi de représenter graphiquement le sport, avec une interface graphique où on peut jouer à deux personnes dessus : joueur 1 détient la queue blanche (billes pleines) et joueur 2 détient la queue noire (billes rayées). Leur but est de rentrer toutes les billes leur appartenant dans les trous à l'aide de la queue. Si l'un des joueurs rentre une bille de l'adversaire, il continue à jouer, mais le point est rajouté à l'adversaire, donc la bille rentre dans la poche de l'adversaire. Dès qu'un des joueurs a rentré tous ses billes à lui il peut tenter de rentrer la bille noire pour gagner la partie. Lorsqu'on rentre la bille blanche, ou la bille noir au mal moment, elles reviennent à sa position avant qu'ils aient été frappées.

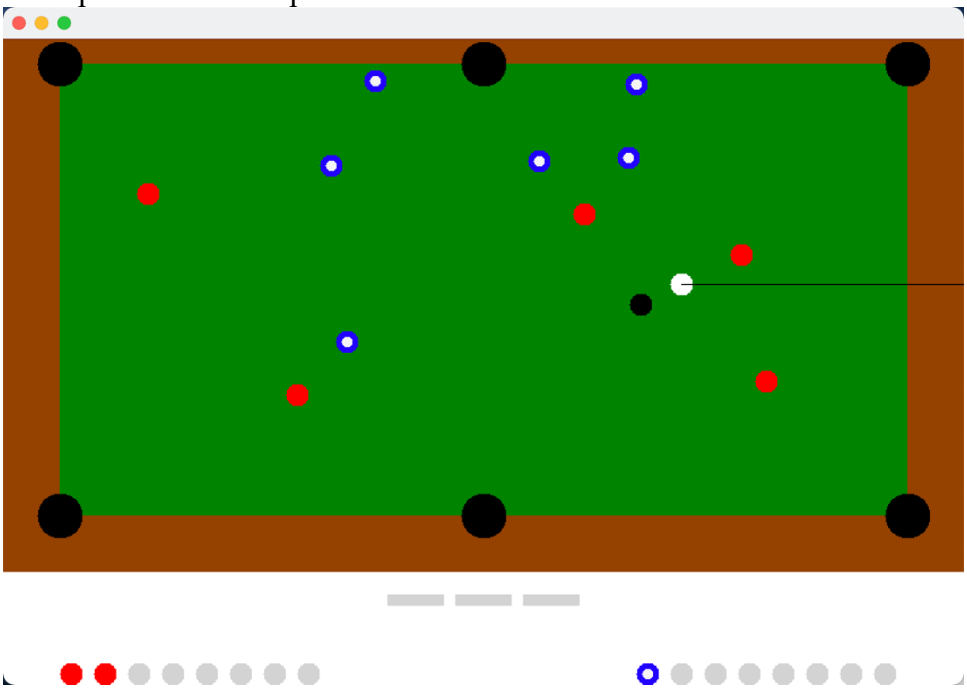
Voici une photo de l'interface graphique :



1. Table
2. Tapis
3. Trou
4. Queue
5. Bille blanche
6. Bille rayée
7. Bille pleine
8. Bille noire

9. Time control : 3 tirets qui représentent l'intensité de la force
10. A gauche : poche du joueur 1. A droite : poche du joueur 2

Exemple d'un tour de partie :



Différences :
La queue est noire car c'est le tour du joueur 2
Le joueur 1 a deux billes dans sa poche
Le joueur 2 a une bille dans sa poche

Copier / coller vos classes et interfaces à partir d'ici :

```
import java.awt.*;

public class Bille extends Materiel implements Rond {
   /** Vitesse de la bille */
   protected int vitesse;
   /** Cette variable représente la vitesse accélérée mais il n'est pas vraiment la vitesse accélérée */
   protected double vAcceleree = 10;
   /** Direction du déplacement */
   protected double direction;
   /** Si surTable est vrai, cette bille est sur table, sinon elle est dans l'une des poches */
   protected boolean surTable;
   /** Compteur des billes */
   protected static int cpt = 0;
   protected final int id;

   public Bille(int x, int y, int r, Color color, int v, double d) {
      super(x, y, r * 2, r * 2, color);
      this.vitesse = v;
      this.direction = d;
      this.vAcceleree = 10;
      this.surTable = true;
      this.id = cpt;
      cpt++;
   }

   public int getR() {
      return width / 2;
   }

   public int getCentreX() {
      return getX() + getR();
   }

   public int getCentreY() {
      return getY() + getR();
   }

   /** Mettre la bille à une certaine position */
   public void setPos(int x, int y) {
      this.x = x;
      this.y = y;
   }

   /** Determiner si cette bille est pleine */
   public boolean estPleine() {
      if (id >= 1 && id <= 7) {
         return true;
      } else {
         return false;
      }
```

```
    }

    /** Determiner si cette bille est rayée */
    public boolean estRayee() {
        if (id >= 9 && id <= 15) {
            return true;
        } else {
            return false;
        }
    }

    /** Déplacer la bille, puis changer sa vitesse avec le variable vAcceleree */
    public void deplacer() {

        x += vitesse * Math.cos(direction);
        y += vitesse * Math.sin(direction);

        if (vitesse > 0) { // La vitesse moins un après dix fois de déplacement, jusqu'à  vitesse == 0
            vAcceleree -= 1;
            if (vAcceleree == 0) {
                vitesse -= 1;
                vAcceleree = 10;
            }
        }
    }

    /** Collision avec les bordures de la table*/
    public void collisionBordure() {
        int suivantX = x + (int)(vitesse * Math.cos(direction));
        int suivantY = y + (int)(vitesse * Math.sin(direction));

        /** Eviter que la bille dépasser le cadre des bordures en cours du déplacement */
        if (suivantX <= Global.tapOffset || suivantX >= Global.tapOffset + Global.tapWidth - width) {
            direction = Global.pi - direction;
        }
        if (suivantY <= Global.tapOffset || suivantY >= Global.tapOffset + Global.tapHeight - height)
{
            direction = -direction;
        }

        /** Repositionner les billes qui sont dehors du cadre */
        if (surTable) {
            if (x < Global.tapOffset) x = Global.tapOffset;
            if (x > Global.tapOffset + Global.tapWidth - width) x = Global.tapOffset + Global.tapWidth
- width;
            if (y < Global.tapOffset) y = Global.tapOffset;
            if (y > Global.tapOffset + Global.tapHeight - height) y = Global.tapOffset +
Global.tapHeight - height;
        }
    }

    /** Collision avec une autre bille */
```

```
    public void collisionBille(Bille b) {
        if (this != b) {

            int x1 = this.getCentreX();
            int x2 = b.getCentreX();
            int y1 = this.getCentreY();
            int y2 = b.getCentreY();
            double d = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
            double v1x = vitesse * Math.cos(direction);
            double v1y = vitesse * Math.sin(direction);
            double v2x = b.vitesse * Math.cos(b.direction);
            double v2y = b.vitesse * Math.sin(b.direction);

            if (Math.abs(d) < Global.epsilon + 2 * getR()) { // Changer ses vitesses et ses directions

                int tmpV;
                double tmpD;

                tmpV = vitesse;
                tmpD = direction;
                vitesse = b.vitesse;
                direction = b.direction;
                b.vitesse = tmpV;
                b.direction = tmpD;

                if (vitesse > 0) {
                    vitesse -= Global.perteVitesseCollision;
                }
                if (b.vitesse > 0) {
                    b.vitesse -= Global.perteVitesseCollision;
                }

                deplacer();
                b.deplacer();
            }

        }
    }

    /** Collision avec les trous */
    public void collisionTrou(Trou[] trous, Joueur j, Joueur j1) {
        for (int i = 0 ; i < trous.length ; i++) {
            double dis = Math.sqrt((trous[i].getCentreX() - this.getCentreX()) * (trous[i].getCentreX() -
this.getCentreX()) +    (trous[i].getCentreY() - this.getCentreY()) * (trous[i].getCentreY() -
this.getCentreY()));

            if (dis < 30) {
                this.vitesse = 0;
                if (estPleine()) { // Si c'est une bille pleine, le joueur j marque
                    j.marquer(this);
                } else if (estRayee()) { // Si c'est une rayee, le joueur j1 marque
                    j1.marquer(this);
```

```
            }

            // Dans GameStart.java les joueurs vont changer leur tour après la frappe, ici on le re-
change, cela signifie que le joueur peut continuer à frapper si il a marqué une bille (même s'il
n'est pas de lui)
            j.changerTour();
            j1.changerTour();

          }
        }
    }

    /**  être frappe par la queue */
    public void etreFrappee(int v, double d) {
        vitesse = v;
        direction = d;
    }

    /** déterminer si la bille est mobile */
    public boolean estMobile() {
        if (vitesse == 0) {
            return false;
        } else {
            return true;
        }
    }

    public Bille clone() {
        return new Bille(x, y, getR(), color, vitesse, direction);
    }

    public void afficher() {
        System.out.println("Bille");
    }

}
```

```
import java.awt.*;

public class Blanche extends Bille implements Rond {
    /** Coordonnées de la dernier position (précédente) de la bille blanche quand elle était immobile */
    private int precedentX;
    private int precedentY;
    /** Créer le singleton */
    private static Blanche singletonBlanche = new Blanche(250, 250);

    private Blanche(int x, int y) {
        super(x, y, Global.billeR, new Color(255, 255, 255), 0, 0);
    }

    public static Blanche getSingletonBlanche() {
        return singletonBlanche;
    }

    public int getR() {
        return width / 2;
    }

    public int getCentreX() {
        return getX() + getR();
    }

    public int getCentreY() {
        return getY() + getR();
    }

    /** Déplacement de la bille blanche */
    @Override
    public void deplacer() {
        x += vitesse * Math.cos(direction);
        y += vitesse * Math.sin(direction);

        if (vitesse > 0) {
           vAcceleree -= 1;
           if (vAcceleree == 0) {
              vitesse -= 1;
              vAcceleree = 10;
           }
        } else { // Renouveler la position précedente
              precedentX = x;
              precedentY = y;
        }
    }

    /** Retourner les coordonnées de la dernier position (précédente) de la bille blanche quand elle était immobile */
    public int getPrecedentX() {
```

```
            return precedentX;
        }

        public int getPrecedentY() {
            return precedentY;
        }

        /** Collision avec les trous */
        @Override
        public void collisionTrou(Trou[] trous, Joueur j, Joueur j1) {
            for (int i = 0 ; i < trous.length ; i++) {
                double dis = Math.sqrt((trous[i].getCentreX()- this.getCentreX()) *
(trous[i].getCentreX() - this.getCentreX()) +    (trous[i].getCentreY() - this.getCentreY()) *
(trous[i].getCentreY() - this.getCentreY()));
                if (dis < 30) { // Repositionner la bille blanche à la position précedente
                   vitesse = 0;
                   x = precedentX;
                   y = precedentY;
                }
            }
        }

        public void afficherColor() {
            System.out.println("Blanche");
        }
}
```

```
public class ErreurParamsException extends Exception {
	public ErreurParamsException(String s) {
		super(s);
	}
}
```

```
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;

public class GameStart extends Frame {

   private static Table table = Table.getSingletonTable();

   /** Les deux joueurs */
   private static Joueur j1 = new Joueur(true);
   private static Joueur j2 = new Joueur(false);

   /** La force de la frappe */
   private static int force;

   /** Les billes */
   private static Bille billes[] = new Bille[16];

   public GameStart() {

      // Créer toutes les billes par une boucle et les positionner
      billes[0] = Blanche.getSingletonBlanche(); // Bille blanche

      billes[1] = new Bille(0, 0, Global.billeR, new Color(255, 0, 0), 0, 0); // Billes pleines
      for (int i = 2 ; i <= 7 ; i++) {
         billes[i] = billes[1].clone();
      }

      billes[8] = new Noire(620, 250); // Bille noire

      billes[9] = new Bille(0, 0, Global.billeR, new Color(0, 0, 255), 0, 0); // Billes rayées
      for (int i = 10 ; i <= 15 ; i++) {
         billes[i] = billes[9].clone();
      }

      billes[1].setPos(700, 250);
      billes[2].setPos(580, 273);
      billes[3].setPos(580, 227);
      billes[4].setPos(660, 273);
      billes[5].setPos(660, 227);
      billes[6].setPos(660, 319);
      billes[7].setPos(660, 181);
      billes[9].setPos(700, 296);
      billes[10].setPos(700, 204);
      billes[11].setPos(700, 158);
      billes[12].setPos(700, 342);
      billes[13].setPos(620, 204);
      billes[14].setPos(540, 250);
      billes[15].setPos(620, 296);
```

```
        // Créer la fenêtre
        setSize(Global.tabWidth, Global.tabHeight + Global.plancheHeight);
        setLocation(50, 50);
        setVisible(true);

        // Ajouter MouseLisener
        addMouseListener(new MouseAdapter() {
            long startTime;
            long endTime;
            long totalTime;

            // On commence à chronométrer dès qu'on appuie sur le bouton gauche de la souris
            @Override
            public void mousePressed(MouseEvent e1) {
                startTime = System.currentTimeMillis();
                force = 0;
            }
            // Des qu'on relâche le bouton gauche de la souris, le chronomètre s'arrête. Ensuite on d
étermine que d'autant le temps d'appuie de la souris est long, d'autant la force qu'on effectue
sur la balle est grande.
            @Override
            public void mouseReleased(MouseEvent e2) {
                endTime    = System.currentTimeMillis();
                totalTime = endTime - startTime;
                if (totalTime > 0) {
                    force = 1;
                }
                if (totalTime > 500) {
                    force = 2;
                }
                if (totalTime > 1000) {
                    force = 3;
                }

                // Frappe
                if (j1.estEnTour()) {
                    j1.getQueue().frapper(force);
                } else if (j2.estEnTour()) {
                    j2.getQueue().frapper(force);
                }
                    // Changer leur tour
                j1.changerTour();
                j2.changerTour();

            }
        });

        // Ajouter WindowListener
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){ // Si on clique le bouton fermé, le jeu s'arrête
                System.exit(0);
```

```
        }});

    }


    private Image offScreenImage = null;
    // Renouveler et tracer le background image qui est statique
    @Override
    public void update(Graphics g) {
        if (offScreenImage == null) offScreenImage = this.createImage(table.getWidth(),
table.getHeight() + Global.plancheHeight);
        Graphics gOffScreen = offScreenImage.getGraphics();
        // On coloris le fond en blanc
        gOffScreen.setColor(new Color(255, 255, 255));
        gOffScreen.fillRect(0, 0, table.getWidth(), table.getHeight() + Global.plancheHeight);
        // Dessiner la table
        gOffScreen.setColor(table.getColor());
        gOffScreen.fillRect(table.getX(), table.getY(), table.getWidth(), table.getHeight());
        // Dessiner le tapis
        gOffScreen.setColor(table.getTapis().getColor());
        gOffScreen.fillRect(table.getTapis().getX(), table.getTapis().getY(),
table.getTapis().getWidth(), table.getTapis().getHeight());
        // Dessiner les trous
        gOffScreen.setColor(table.getTrous()[0].getColor());
        for (int i = 0 ; i < 6 ; i++) {
            gOffScreen.fillOval(table.getTrous()[i].getX(), table.getTrous()[i].getY(),
table.getTrous()[i].getWidth(), table.getTrous()[i].getHeight());
        }


        // Dessiner les cercles des poches qui seront les emplacements des billes apres avoir rentrees
dans les trous
        gOffScreen.setColor(new Color(211, 211, 211));
        for (int i = 0 ; i < 8 ; i++) {
            gOffScreen.fillOval(j1.getPoche().getX() + i * (2 * Global.billeR + 10),
j1.getPoche().getY(), Global.billeR * 2, Global.billeR * 2);
            gOffScreen.fillOval(j2.getPoche().getX() + i * (2 * Global.billeR + 10),
j2.getPoche().getY(), Global.billeR * 2, Global.billeR * 2);
        }

        // Dessiner time control. Nous représentons la force (donc temps) qu'on soumet pour frapper
une bille par les trois tirets en bas de la table
        gOffScreen.setColor(new Color(211, 211, 211));
        gOffScreen.fillRect(425 - 25 - 10 - 50, 520, 50, 10);
        gOffScreen.fillRect(425 - 25, 520, 50, 10);
        gOffScreen.fillRect(425 + 25 + 10, 520, 50, 10);

        paint(gOffScreen);
        g.drawImage(offScreenImage, table.getX(), table.getY(), null);
    }

    // Renouveler et tracer les objets sur table qui sont dynamiques
```

```
    @Override
    public void paint(Graphics g) {
        super.paint(g);

        // Dessiner les billes
        for (int i = 0 ; i < billes.length ; i++) {
            g.setColor(billes[i].getColor());
            g.fillOval(billes[i].getX(), billes[i].getY(), billes[i].getWidth(), billes[i].getHeight());
            //    Dessiner les rayures des billes rayees
            if (billes[i].estRayee()) {
                g.setColor(new Color(255, 255, 255));
                g.fillOval(billes[i].getX() + Global.billeR / 2, billes[i].getY() + Global.billeR / 2,
Global.billeR, Global.billeR);
            }

            // Vérifier la collision avec les boredures
            billes[i].collisionBordure();

            // Vérifier la collision avec les trous
            billes[i].collisionTrou(table.getTrous(), j1, j2);

            // Vérifier la collision entre les billes
            for (int n = 0 ; n + i < billes.length ; n++) {
                billes[i].collisionBille(billes[i + n]);
            }

            // Déplacer
            billes[i].deplacer();
        }

        // Renouveler time control. Nous colorions les trois tirets selon la quantité de temps qu'on a
restés appuyés sur le bouton de la souris, et représente donc aussi la quantité de force appliqué à
la bille
        if (force >= 1) {
            g.setColor(new Color(255, 99, 71));
            g.fillRect(425 - 25 - 10 - 50, 520, 50, 10);
        }
        if (force >= 2) {
            g.setColor(new Color(255,0,0));
            g.fillRect(425 - 25, 520, 50, 10);
        }
        if (force >= 3) {
            g.setColor(new Color(220,20,60));
            g.fillRect(425 + 25 + 10, 520, 50, 10);
        }


        // Vérifier l'état de mouvement des billes, si toutes les billes sont immobiles, on mettre la
queue visible, sinon on le cache
        if (Global.billesSontImmobiles(billes)) {
            if (j1.estEnTour()) {
```

```
            j1.getQueue().show();
        } else if (j2.estEnTour()) {
            j2.getQueue().show();
        }
        // Repaint time control. On ré-initialise les trois tirets en bas en couleur grise.
        g.setColor(new Color(211, 211, 211));
        g.fillRect(425 - 25 - 10 - 50, 520, 50, 10);
        g.fillRect(425 - 25, 520, 50, 10);
        g.fillRect(425 + 25 + 10, 520, 50, 10);
    } else {
        j1.getQueue().hide();
        j2.getQueue().hide();
    }

    // Dessiner les queues
    if (j1.getQueue().estVisible()) {
        g.setColor(j1.getQueue().getColor());
        g.drawLine(j1.getQueue().getX(), j1.getQueue().getY(), j1.getQueue().getSourisX(),
j1.getQueue().getSourisY());
        j1.getQueue().viser();
    }
    if (j2.getQueue().estVisible()) {
        g.setColor(j2.getQueue().getColor());
        g.drawLine(j2.getQueue().getX(), j2.getQueue().getY(), j2.getQueue().getSourisX(),
j2.getQueue().getSourisY());
        j2.getQueue().viser();
    }

    // Déterminer si l'un des deux joueurs a gagné
    if (j1.getPoche().getNbBilles() == 8) {
        j1.gagner();
        setVisible(false);
        System.out.println("Joueur " + j1.getId() + " gagne !");
    }
    if (j2.getPoche().getNbBilles() == 8) {
        j2.gagner();
        setVisible(false);
        System.out.println("Joueur " + j2.getId() + " gagne !");
    }

    if (j1.estGagneur() || j2.estGagneur()) System.exit(0);

    repaint();

    // Dans le but d'afficher et que l'image courant soit visible, faire une pause de 10ms
    try {
        Thread.sleep(10);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
}
```

```
import java.awt.*;

import java.util.ArrayList;

public class Global {
   public static final double pi = 3.14;
   public static final double epsilon = 0.1;

   /** Rayon de la bille */
   public static final int billeR = 10;

   /** Rayon du trou */
   public static final int trouR = 20;

   /** Taille de la table */
   public static final int tabWidth = 850;
   public static final int tabHeight = 500;

   /** Taille et position du tapis */
   public static final int tapWidth = 750;
   public static final int tapHeight = 400;
   public static final int tapOffset = 50;

   /** Taille de la zone blanche en-dessous de la table */
   public static final int plancheHeight = 100;

   /** Taille de la poche */
   public static final int pocheWidth = 240; // (billeR * 2 + 10) * 8
   public static final int pocheHeight = 20; // billeR * 2

   /** Constantes concernantes la vitesse */
   public static final int uniteVitesse = 5;
   public static final int perteVitesseCollision = 1;

   /** Déterminer si toutes les billes sur table sont immobiles */
   public static boolean billesSontImmobiles(Bille[] billes) {
      for (int i = 0 ; i < billes.length ; i++) {
         if (billes[i].estMobile()) return false;
      }
      return true;
   }

   /** Vérifier les paramètres du programme */
   public static void verifierParams(int nb) throws ErreurParamsException {
      if (nb != 1) throw new ErreurParamsException("Nombre du parametre incorrect");
      else return ;
   }
}
```

```
import java.awt.*;

public class Joueur {
	/** Queue et poche du joueur */
	private Queue queue;
	private Poche poche;
	/** Si enTour est vrai, le joueur est en tour */
	private boolean enTour;
	/** Si gagneur est vrai, il est le gagneur */
	private boolean gagneur;
	/** L'id du joueur. On a deux joueurs dans ce jeu */
	private final int id;
	/** Compteur du nombre de joueurs */
	private static int cpt = 0;

	public Joueur(boolean enTour) {
		cpt++;
		id = cpt;
		this.enTour = enTour;
		gagneur = false;

		if (id == 1) { // S'il est le premier, sa queue est blanche, sa poche est à gauche
			queue = new Queue(new Color(255, 255, 255));
			poche = new Poche(50, 580);
		} else if (id == 2) { // S'il est le deuxième, sa queue est noire, sa poche est à droite
			queue = new Queue(new Color(0, 0, 0));
			poche = new Poche(Global.tabWidth - 50 - Global.pocheWidth, 580);
		}
	}

	/** Joueur frappe une bille dans un trou */
	public void marquer(Bille b) {
		b.x = getPoche().getX() + getPoche().getNbBilles() * (2 * Global.billeR + 10);
		b.y = getPoche().getY();
		b.surTable = false;
		getPoche().getBilles()[getPoche().getNbBilles()] = b;
		getPoche().nbBillesAugmente();
	}

	/** Changer leur tour */
	public void changerTour() {
		if (enTour) {
			enTour = false;
		} else {
			enTour = true;
		}
	}

	/** Determiner si ce joueur est en tour */
	public boolean estEnTour() {
		return enTour;
```

```
    }

    public void gagner() {
        gagneur = true;
    }

    public int getId() {
        return id;
    }

    /** Determiner si ce joueur est le gagneur */
    public boolean estGagneur() {
        return gagneur;
    }

    public Poche getPoche() {
        return poche;
    }

    public Queue getQueue() {
        return queue;
    }

}
```

```
public class Main {
	public static void main(String[] argv) {
		int choix;

		try {
			Global.verifierParams(argv.length);
			choix = Integer.parseInt(argv[0]);
			if (choix == 1) {
				GameStart game = new GameStart();
			} else if (choix == 2) {
				System.out.println("Bienvenu au monde du billard.");
				System.out.println("Ceci est un jeu pour deux personnes : ");
				System.out.println("Joueur 1 utilise la queue blanche et a pour objective les
billes pleines");
				System.out.println("Joueur 2 utilise la queue noire et a pour objective les billes
rayees");
				System.out.println("");
				System.out.println("But du jeu : ");
				System.out.println("Rentrer tous les billes de ton champ dans les trous à l'aide
de la bille blanche et de la queue");
				System.out.println("Une fois votre poche est pleine, vous tentez de rentrer la
bille noir dans le trou");
				System.out.println("");
				System.out.println("Instructions : Pour frapper appuyez sur le bouton gauche de
la souris dans l'interface du jeu et controlez la direction de la queue pour marquer.");
				System.out.println("Le temps d'appuie sur le bouton de la souris determine
l'intensite de la force appliquée sur la bille frappée");
				System.out.println("Attention : si vous rentrez une bille de votre adversaire,
vous pouvez continuer à joueur, mais c'est lui qui gagne un point.");
			} else {
				System.out.println("Vous n'avez que deux choix du parametre : 1 ou 2");
			}

		} catch (ErreurParamsException e) {
			System.out.println(e +	"\njava Main 1 : Game start\njava Main 2 : Regles du jeu");

		} catch (NumberFormatException e) {
			System.out.println("Vous n'avez que deux choix du parametre : 1 ou 2");
		}

	}
}
```

```
import java.awt.*;

public abstract class Materiel {
   /** Coordonnées, taille et couleur du matériel */
   protected int x;
   protected int y;
   protected int width;
   protected int height;
   protected Color color;

   public Materiel(int x, int y, int w, int h, Color color) {
      this.x = x;
      this.y = y;
      height = h;
      width = w;
      this.color = color;
   }

   public int getX() {
      return x;
   }

   public int getY() {
      return y;
   }

   public int getHeight() {
      return height;
   }

   public int getWidth() {
      return width;
   }

   public Color getColor() {
      return color;
   }

   /** Pour debug */
   public abstract void afficher();
}
```

```
import java.awt.*;

public class Noire extends Bille implements Rond {
    /** Coordonnées de la dernier position (précédente) de la bille noire quand elle était immobile */
    private int precedentX;
    private int precedentY;

    public Noire(int x, int y) {
        super(x, y, Global.billeR, new Color(0, 0, 0), 0, 0);
    }

    public int getR() {
        return width / 2;
    }

    public int getCentreX() {
        return getX() + getR();
    }

    public int getCentreY() {
        return getY() + getR();
    }

    @Override
    public void deplacer() {
        x += vitesse * Math.cos(direction);
        y += vitesse * Math.sin(direction);

        if (vitesse > 0) {
           vAcceleree -= 1;
           if (vAcceleree == 0) {
              vitesse -= 1;
              vAcceleree = 10;
           }
        } else {
              precedentX = x;
              precedentY = y;
        }
    }

    /** Retourner les coordonnées de la dernier position (précédente) de la bille noire quand elle était immobile */
    public int getPrecedentX() {
        return precedentX;
    }

    public int getPrecedentY() {
        return precedentY;
    }
```

```
    /** Si le joueur a marqué 7 billes, puis il frappe la bille noire dans les trous, il gagne. Sinon on repositionner la bille noire */
    @Override
    public void collisionTrou(Trou[] trous, Joueur j, Joueur j1) {
        for (int i = 0 ; i < trous.length ; i++) {
            double dis = Math.sqrt((trous[i].getCentreX()- this.getCentreX()) * (trous[i].getCentreX() - this.getCentreX()) +    (trous[i].getCentreY() - this.getCentreY()) * (trous[i].getCentreY() - this.getCentreY()));
            if (dis < 30) {
                vitesse = 0;
                if (j.getPoche().getNbBilles() == 7 && j1.estEnTour()) { // Joueur j1 est en tour, cela signifie que c'est joueur j qui a frappé la bille noire dans le trou, car les joueurs vont changer leur tour après la frappe (dans GameStart.java)
                    j.marquer(this);
                } else if (j1.getPoche().getNbBilles() == 7 && j.estEnTour()) { // Avec la meme raison
                    j1.marquer(this);
                } else {
                    x = precedentX;
                    y = precedentY;
                }
            }
        }
    }

    public void afficherColor() {
        System.out.println("Noir");
    }
}
```

```
public class Poche extends Materiel {
    /** Les billes dans la poche */
    private Bille[] billes;
    /** Nombre des billes */
    private int nbBilles;

    public Poche(int x, int y) {
        super(x, y, Global.pocheWidth, Global.pocheHeight, null);
        billes = new Bille[8];
        nbBilles = 0;
    }

    public Bille[] getBilles() {
        return billes;
    }

    public void nbBillesAugmente() {
        nbBilles++;
    }

    public int getNbBilles() {
        return nbBilles;
    }

    public void afficher() {
        System.out.println("Poche");
    }
}
```

```
import java.awt.*;

public class Queue extends Materiel {
	/** Coordonnées de la souris */
	private int sourisX;
	private int sourisY;
	/** La bille pointée par cette queue */
	private Bille but;
	/** Si cette queue est visible */
	private boolean visible;

	public Queue(Color color) {
		super(Blanche.getSingletonBlanche().getX(), Blanche.getSingletonBlanche().getY(), 0, 0,
color);
		sourisX = java.awt.MouseInfo.getPointerInfo().getLocation().x;
		sourisY = java.awt.MouseInfo.getPointerInfo().getLocation().y;
		but = Blanche.getSingletonBlanche();
		visible = false;
	}

	/** Viser selon le changement de position de la souris */
	public void viser() {
		sourisX = java.awt.MouseInfo.getPointerInfo().getLocation().x;
		sourisY = java.awt.MouseInfo.getPointerInfo().getLocation().y;
		x = but.x + Global.billeR;
		y = but.y + Global.billeR;
	}

	/** Frapper le but */
	public void frapper(int force) {
		if (visible && y != sourisY && x != sourisX) {
			double angle = Math.atan((double)(y - sourisY) / (double)(x - sourisX));
			if (sourisX > x) angle += Global.pi;
			but.etreFrappee(1 + Global.uniteVitesse * force, angle);
		}
	}

	public int getSourisX() {
		return sourisX;
	}

	public int getSourisY() {
		return sourisY;
	}

	public boolean estVisible() {
		return visible;
	}

	public void show() {
		visible = true;
	}
```

```
    public void hide() {
        visible = false;
    }

    public void afficher() {
        System.out.println("Queue");
      }
}
```

```
public interface Rond {
   /** Retourner la rayon du objet */
   public int getR();
   /** Retourner la coordonnee x de la centre */
   public int getCentreX();
   /** Retourner la coordonnee y de la centre */
   public int getCentreY();
}
```

```
import java.awt.*;

public class Table extends Materiel {
   /** Créer le tapis, les trous sur la table */
   private static Tapis tapis = Tapis.getSingletonTapis();
   private static Trou[] trous = {
      new Trou(Global.tapOffset - Global.trouR, Global.tapOffset - Global.trouR, Global.trouR),
      new Trou(Global.tapOffset - Global.trouR, Global.tapOffset + Global.tapHeight - Global.trouR,
Global.trouR),
      new Trou(Global.tabWidth / 2 - Global.trouR, Global.tapOffset - Global.trouR, Global.trouR),
      new Trou(Global.tabWidth / 2 - Global.trouR, Global.tapOffset + Global.tapHeight -
Global.trouR, Global.trouR),
      new Trou(Global.tapOffset + Global.tapWidth - Global.trouR, Global.tapOffset - Global.trouR,
Global.trouR),
      new Trou(Global.tapOffset + Global.tapWidth - Global.trouR, Global.tapOffset +
Global.tapHeight - Global.trouR, Global.trouR)
   };
   /** Créer le singleton */
   private static Table singletonTable = new Table(0, 0, Global.tabWidth, Global.tabHeight, new
Color(139,69,19));


   private Table(int x, int y, int w, int h, Color color) {
      super(x, y, w, h, color);
   }

   public Tapis getTapis() {
      return tapis;
   }

   public Trou[] getTrous() {
      return trous;
   }

   public static Table getSingletonTable() {
      return singletonTable;
   }

   public void afficher() {
      System.out.println("Table");
   }
}
```

```
import java.awt.*;

public class Tapis extends Materiel {
	/** Créer le singleton */
	private static Tapis singletonTapis = new Tapis(Global.tapOffset, Global.tapOffset,
Global.tapWidth, Global.tapHeight, new Color(0, 128, 0));

	private Tapis(int x, int y, int w, int h, Color color) {
		super(x, y, w, h, color);
	}

	public static Tapis getSingletonTapis() {
		return singletonTapis;
	}

	public void afficher() {
		System.out.println("Tapis");
	}
}
```

```
import java.awt.*;

public class Trou extends Materiel implements Rond {
	public Trou(int x, int y, int r) {
		super(x, y, r * 2, r * 2, new Color(0, 0, 0));
	}

	public int getR() {
		return width / 2;
	}

	public int getCentreX() {
		return getX() + getR();
	}

	public int getCentreY() {
		return getY() + getR();
	}

	public void afficher() {
		System.out.println("Trou");
	}
}
```