

**UNIVERSITE PROTESTANTE AU CONGO**

*Faculté des sciences informatiques*



**B.P 4745**

**KINSHASA/LINGWALA**

## **TP7**

**PROGRAMMATION SYSTEMES SUR LE COMPORTEMENT D'UN  
SIMPLE PLANIFICATEUR DE CPU**

Présenté par :



**NGOMA MAKWAMA ALAIN**

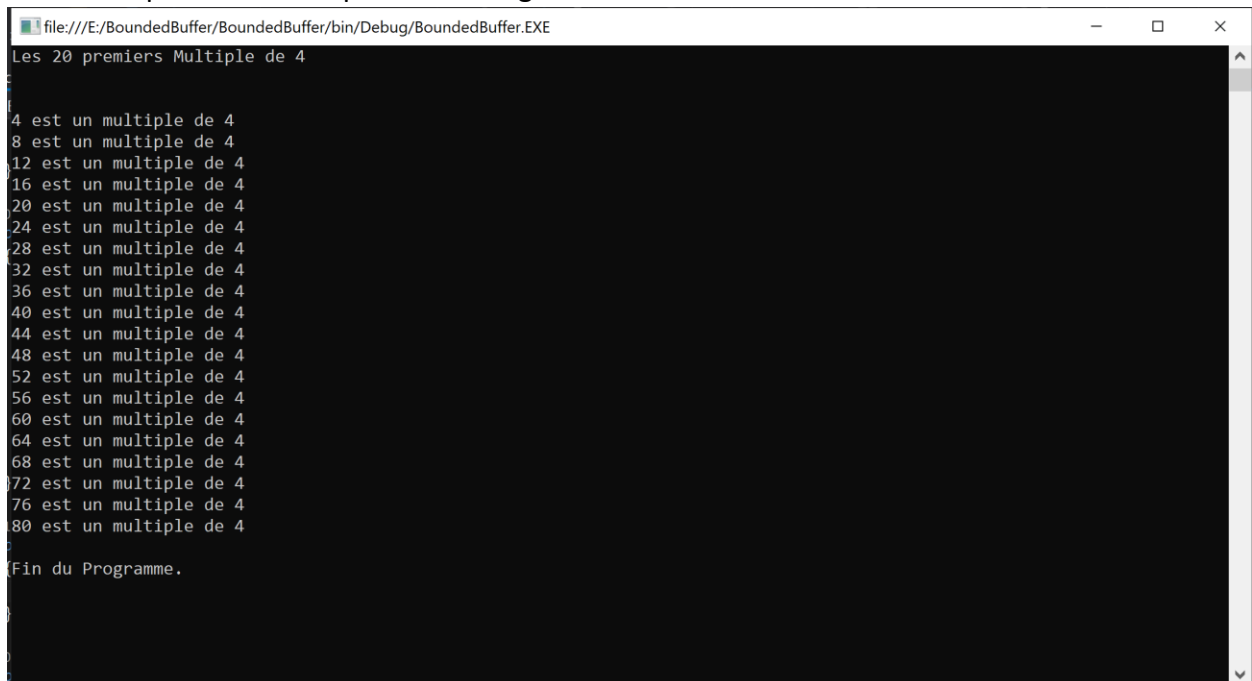
**PROF. DR. Patrick MUKALA**

**ANNEE – ACADEMIQUE 2020 - 2021**

## LA RESOLUTION DU TP 7.

A. Pour cette première question nous avons la charge de pouvoir télécharger et d'implémenter la classe BoundedBuffer, en apportant quelques petites modifications sur le code enfin d'obtenir le résultat que nous attendons qui est l'affichage des 20 premiers multiples de 4. Nous devons également vérifier que toutes les valeurs sont affichées dans le bon ordre, aucune valeur n'est doit être manquante sur la liste mais aussi aucune valeur n'est affiche deux fois.

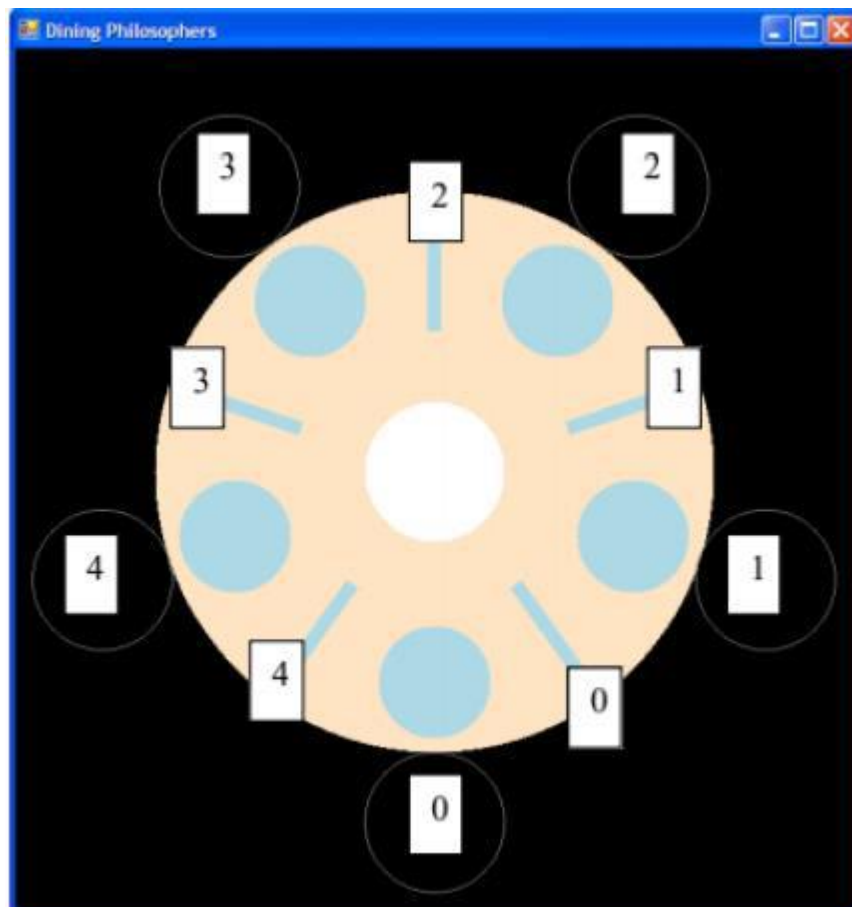
Comme le présente cette première image :



```
file:///E:/BoundedBuffer/BoundedBuffer/bin/Debug/BoundedBuffer.EXE
Les 20 premiers Multiple de 4
4 est un multiple de 4
8 est un multiple de 4
12 est un multiple de 4
16 est un multiple de 4
20 est un multiple de 4
24 est un multiple de 4
28 est un multiple de 4
32 est un multiple de 4
36 est un multiple de 4
40 est un multiple de 4
44 est un multiple de 4
48 est un multiple de 4
52 est un multiple de 4
56 est un multiple de 4
60 est un multiple de 4
64 est un multiple de 4
68 est un multiple de 4
72 est un multiple de 4
76 est un multiple de 4
80 est un multiple de 4
{Fin du Programme.
```

Image 1 : les 20 premiers multiples de 4.

B. Nous devons être sûr que les philosophes ne se mentent pas dans l'impasse en laissant un philosophe ramasser les bâtons dans un autre ordre que les autres philosophes restants.

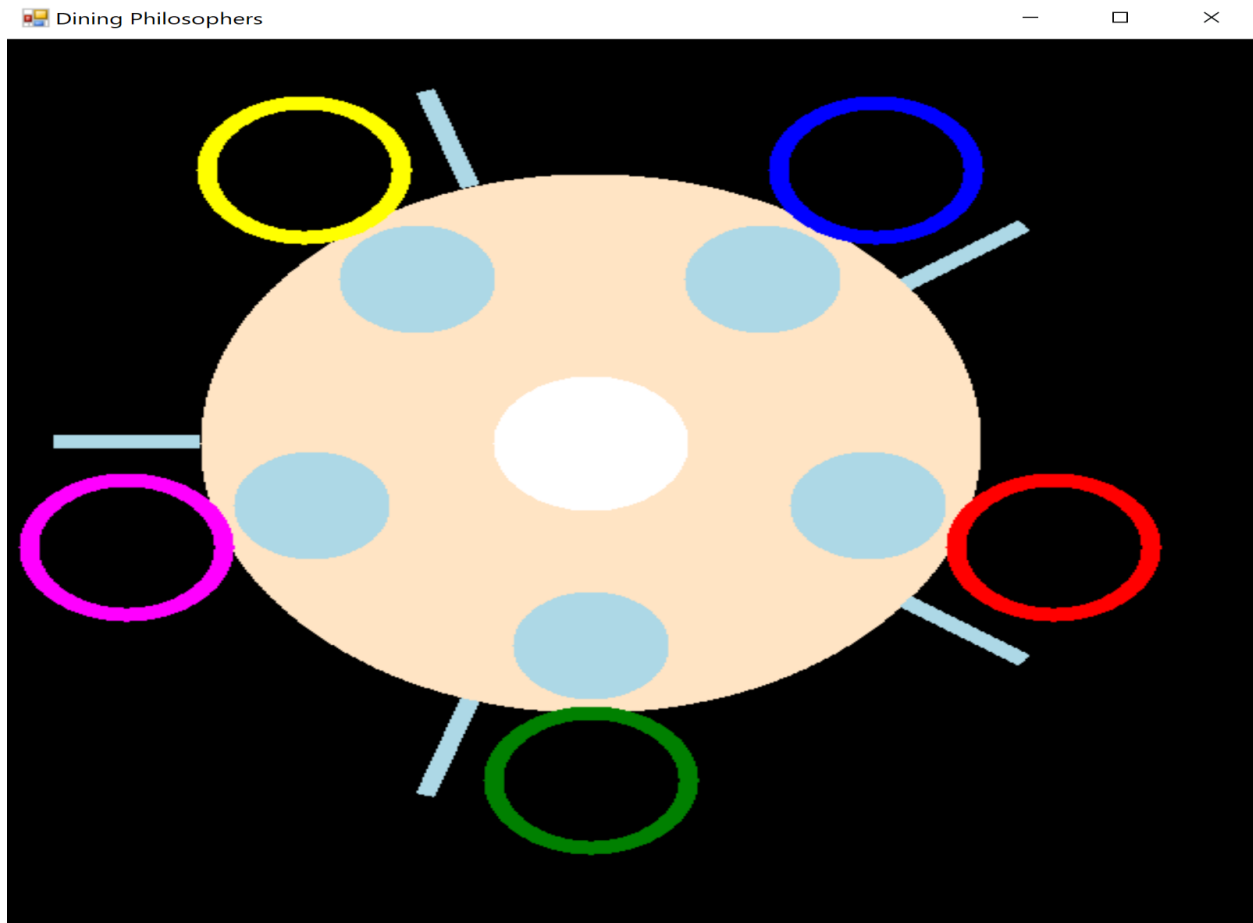


Capture 1 : Voici l'ordre de départ donc chaque philosophe est contraint de pouvoir ramasser la fourchette en étant compte de la numérotation.

Lors de la première exécution de notre programme nous constatons une impasse d'entre de jeux et immédiatement, c'est-à-dire l'erreur résulte du fait que la logique pour le ramassage n'est pas respectée.

Nous voyons par exemple que le philosophe 3 ramasse la fourchette 2 ce qui ne respecte pas du tout l'ordre imposé à la capture 1.

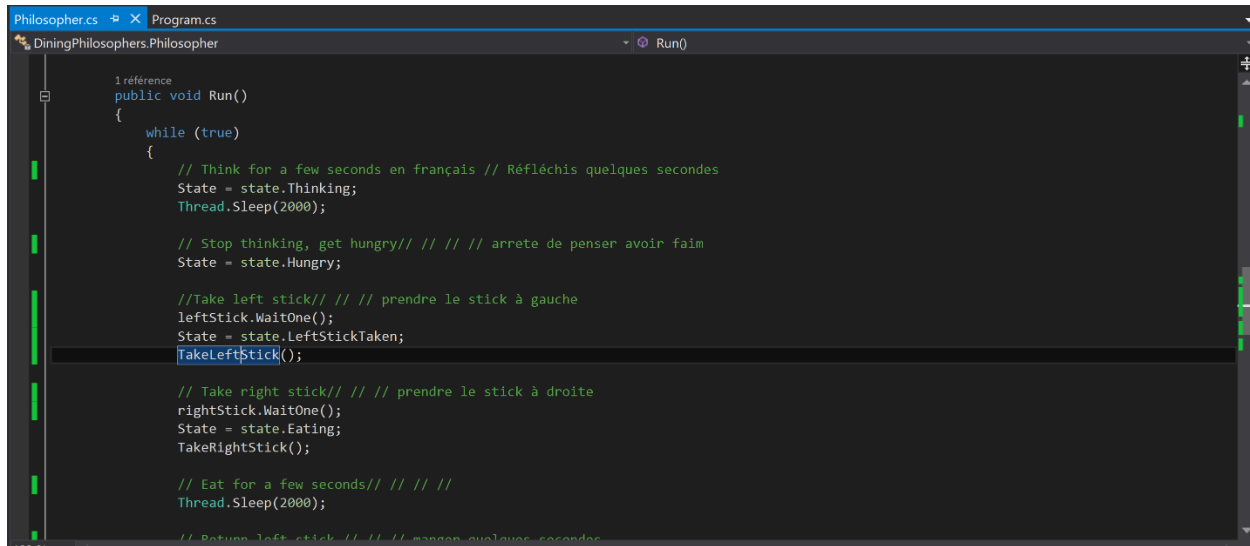
En voici l'exemple de l'exécution :



Capture 2 : L'exécution démonstrative de l'impasse constatée juste après le lancement du programme.

Après exécution voici les différents numéros rattachés entre eux, en rouge nous avons la numérotation des fourchettes et en blanc la numérotation des philosophes. Pour avoir le résultat attendu il fallait que le numéro de la fourchette soit égal au numéro renseignant le philosophe. Par exemple le philosophe 3 devait avoir la fourchette 3 aussi.

Et cela est dû à une partie du code représenté ci-dessous :



```

1 référence
public void Run()
{
    while (true)
    {
        // Think for a few seconds en français // Réfléchis quelques secondes
        State = state.Thinking;
        Thread.Sleep(2000);

        // Stop thinking, get hungry// // // // arrete de penser avoir faim
        State = state.Hungry;

        //Take left stick// // // prendre le stick à gauche
        leftStick.WaitOne();
        State = state.LeftStickTaken;
        TakeLeftStick();

        // Take right stick// // // prendre le stick à droite
        rightStick.WaitOne();
        State = state.Eating;
        TakeRightStick();

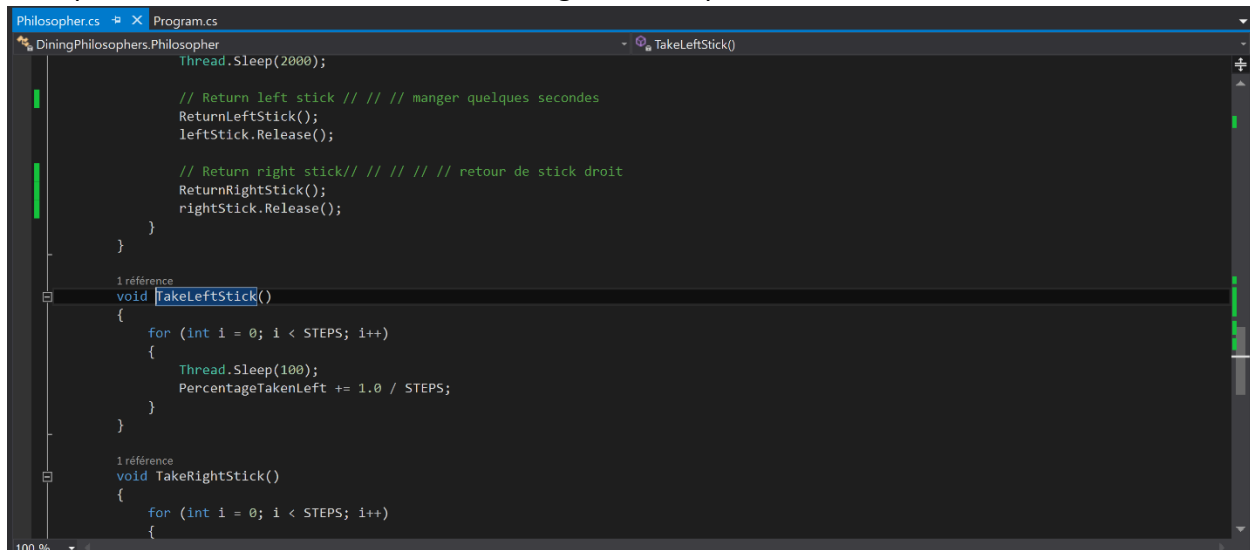
        // Eat for a few seconds// // // //
        Thread.Sleep(2000);

        // Return left stick // // // manger quelques secondes
    }
}

```

Capture 3 : la partie du code ne permettant pas d’obtenir le résultat attendu.

La partie du code encerclé en rouge montre l’appel de la méthode TakeLeftStick qui permet de faire pivoter toutes les fourchettes vers la gauche ce qui fait bouleverser l’ordre de l’exécution.



```

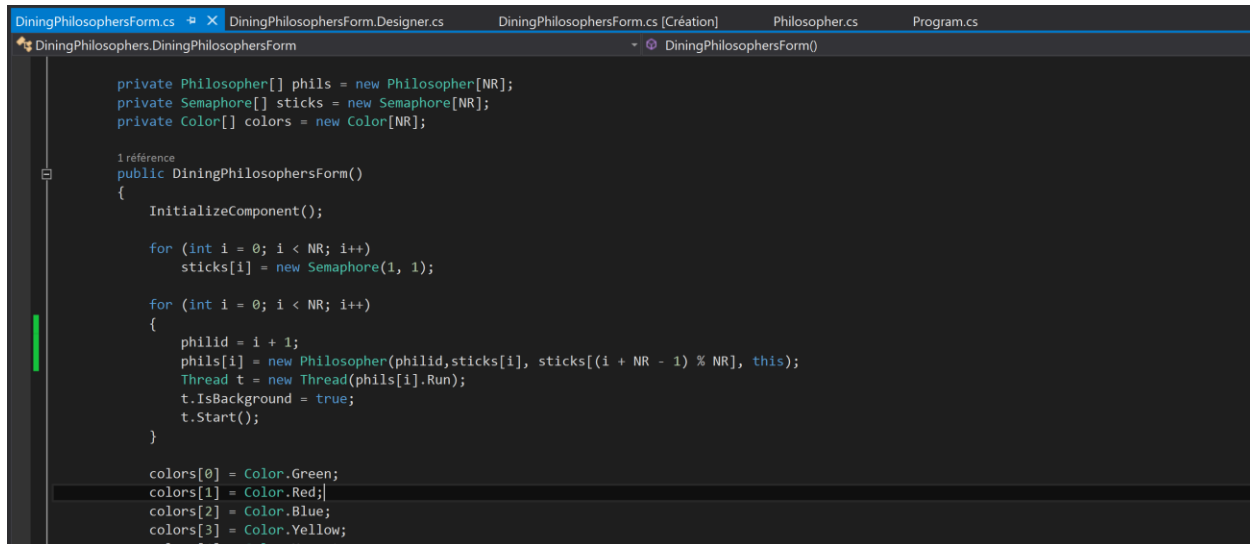
1 référence
void TakeLeftStick()
{
    for (int i = 0; i < STEPS; i++)
    {
        Thread.Sleep(100);
        PercentageTakenLeft += 1.0 / STEPS;
    }
}

1 référence
void TakeRightStick()
{
    for (int i = 0; i < STEPS; i++)
    {

```

Capture 4 : La méthode TakeLeftStick encerclé en rouge également.

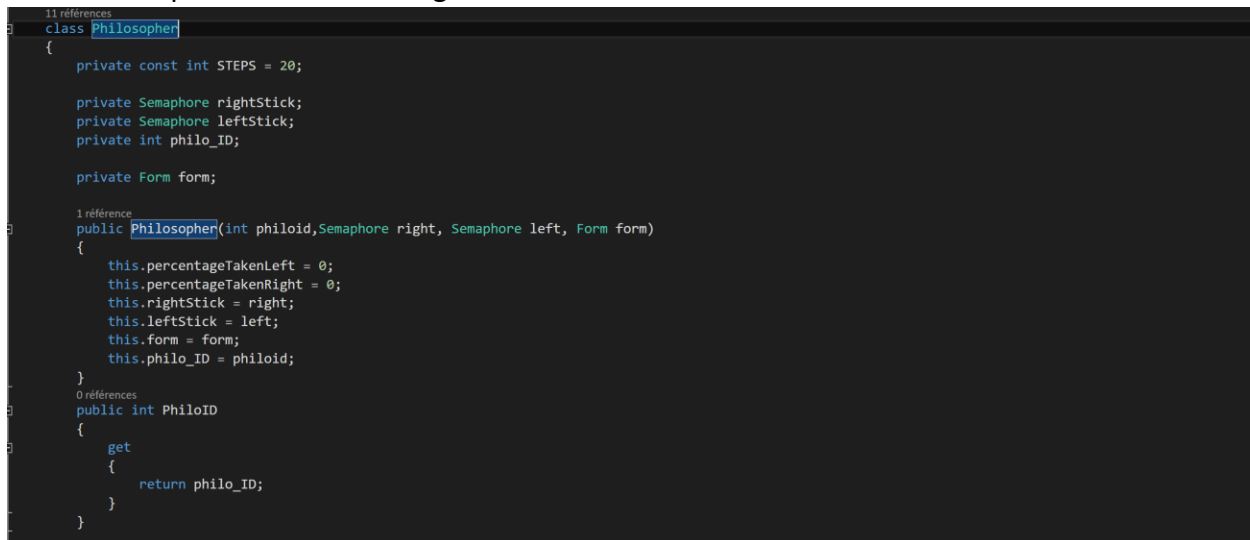
Pour résoudre ce problème nous avons premièrement changé la classe légèrement la classe DiningPhilosophersFrom.cs pour récupérer l’id de chaque philosophe enfin d’en faire quelques opérations dans la classe Philosopher.



Capture 5 : Le nouveau code permet de renvoyer l'id de chaque philosophe.

Dans la classe Philosopher nous devons récupérer l'id de chaque utilisateur, nous allons créer un nouveau paramètre philoid qui sera initialisé dans le constructeur dont nous générons le getter et le setter.

En voici la représentation en image :



Capture 6 : La déclaration du paramètre id ainsi que son initialisation.

Les opérations se font dans la méthode Run qui se trouve dans la classe Philosopher. Grâce à l'id récupérer nous allons faire de test en utilisant la structure conditionnelle if... else pour faire en sorte qu'un philosophe récupère la fourchette dans un autre ordre que les quatre autres.

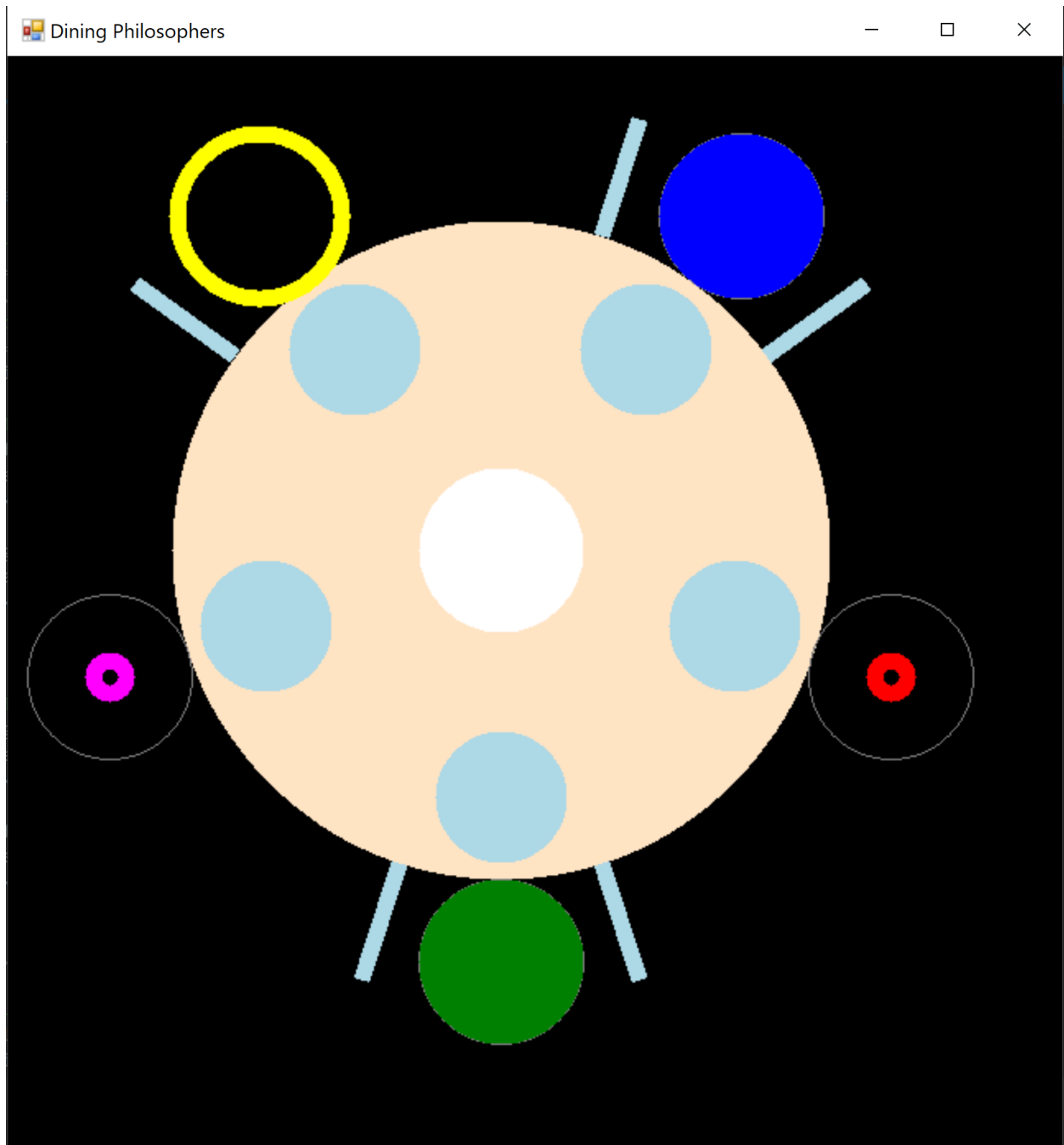
Dans notre exemple les quatre premiers vont ramasser les fourchettes à droite puis celui de la gauche et le dernier celui dont son id est égal à 5 ramasse la fourchette de la gauche et puis celui de droite.

```

public void Run()
{
    while (true)
    {
        // Think for a few seconds en français // Réfléchis quelques secondes
        State = state.Thinking;
        Thread.Sleep(2000);
        // Stop thinking, get hungry// // // // arrete de penser avoir faim
        State = state.Hungry;
        if (philo_ID == 5)
        {
            //Take left stick// // // prendre le stick à gauche
            leftStick.WaitOne();
            State = state.LeftStickTaken;
            TakeLeftStick();
            // Take right stick// // // prendre le stick à droite
            rightStick.WaitOne();
            State = state.Eating;
            TakeRightStick();
        }
        else
        {
            //Take left stick// // // prendre le stick à gauche
            rightStick.WaitOne();
            State = state.RightStickTaken;
            TakeRightStick();
            // Take right stick// // // prendre le stick à droite
            leftStick.WaitOne();
            State = state.Eating;
            TakeLeftStick();
        }
        // Eat for a few seconds// // // //
        Thread.Sleep(2000);
    }
}

```

Capture 7 : La partie du code permettant de faire le test et d'attribuer les fourchettes en tenant compte de la logique donnée.



Capture 8 : La correction de l'impasse en permettant une distribution normale de chaque fourchette à chaque philosophe.

C. Nous devons être sûr que les philosophes ne se mentent pas dans l'impasse en autorisant au plus 4 philosophes à la fois pour ramasser leurs fourchettes.



```

using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Windows.Forms;

namespace DiningPhilosophers
{
    11 références
    class Philosopher
    {
        private const int STEPS = 20;

        private Semaphore rightStick;
        private Semaphore leftStick;
        private Semaphore[] fourchette;
        private int idPhilo;
        private int temps;
        private Form form;

        1 référence
        public Philosopher(Semaphore right, Semaphore left, Form form, int philo_ID, Semaphore [] four, int think_Delay)
        {
            this.percentageTakenLeft = 0;
            this.percentageTakenRight = 0;
            this.rightStick = right;
            this.leftStick = left;
            this.form = form;
            this.idPhilo = philo_ID;
            this.fourchette = four;
            this.temps = think_Delay;
        }
    }
}

```

Photo 1 : La déclaration du sémaphore et son initialisation.

```

1 référence
public void Run()
{
    while (true)
    {
        // Think for a few seconds
        State = state.Thinking;
        Thread.Sleep(temps);
        // Stop thinking, get hungry
        State = state.Hungry;
        test(idPhilo);
        //taking forks
        take_fork(leftStick, rightStick);
        // Eat for a few seconds
        Thread.Sleep(2000);
        release_fork(leftStick, rightStick);
    }
}

1 référence
private void take_fork(Semaphore right, Semaphore left)
{
    // Take left stick
    left.WaitOne();
    State = state.LeftStickTaken;
    TakeLeftStick();

    // Take right stick
    right.WaitOne();
    State = state.Eating;
    TakeRightStick();
}

1 référence
private void release_fork(Semaphore right, Semaphore left)
{
}

```

Photo 2 : Le code permettant de réaliser cette transition.

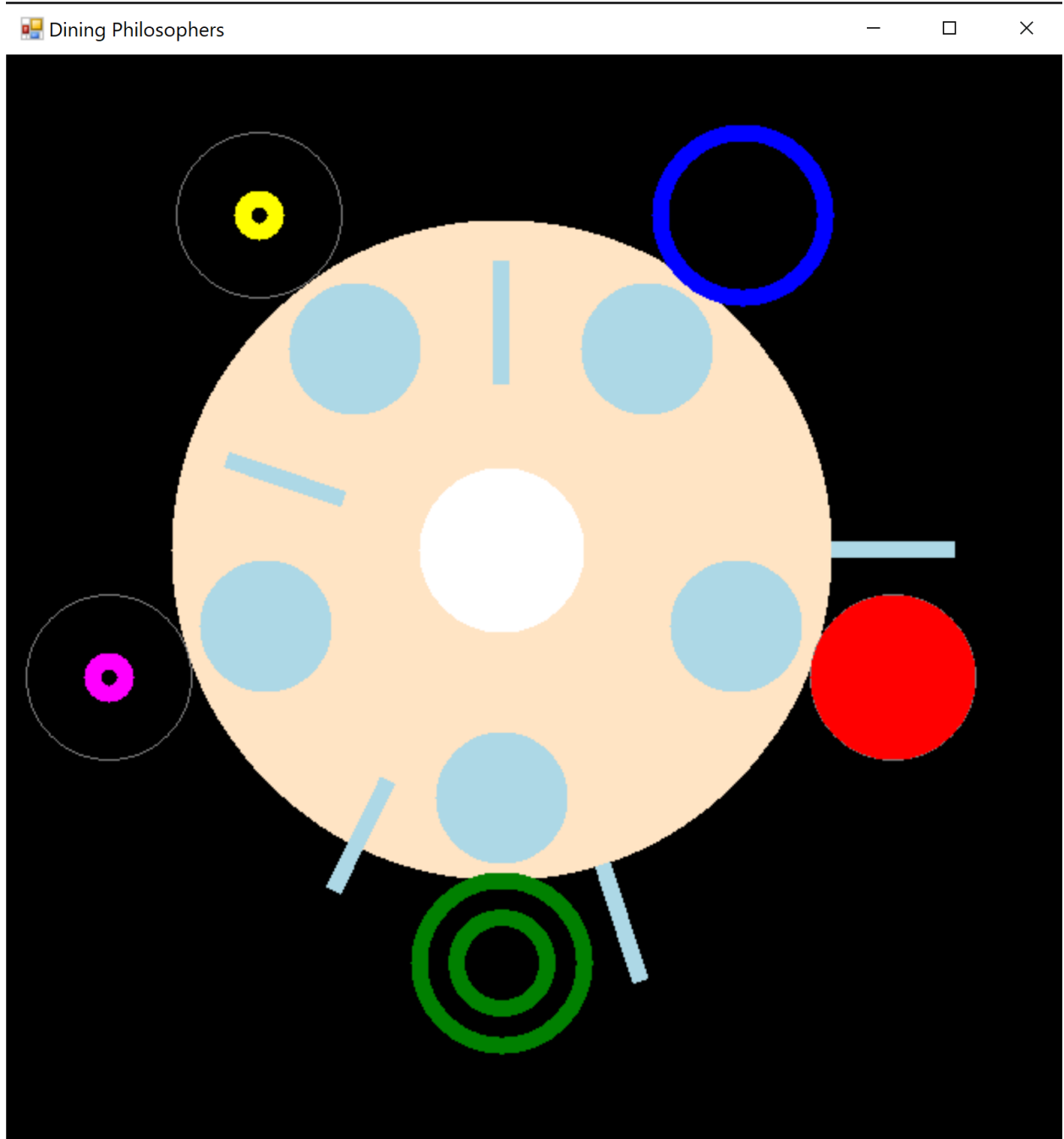


Photo 3 : Exécution du programme.