

Initiation C#

06 – Robocode

FRANCK JUBIN

2019

Installation des composants

Installation

Installation de java

<https://www.java.com/fr/download/manual.jsp>

Installation de Robocode

<https://sourceforge.net/projects/robocode/files/robocode/1.9.3.3/> fichier : robocode-1.9.3.3-setup.jar

Installation du plugin dotnet

<https://sourceforge.net/projects/robocode/files/robocode/1.9.3.3/> fichier : robocode.dotnet-1.9.3.3-setup.jar

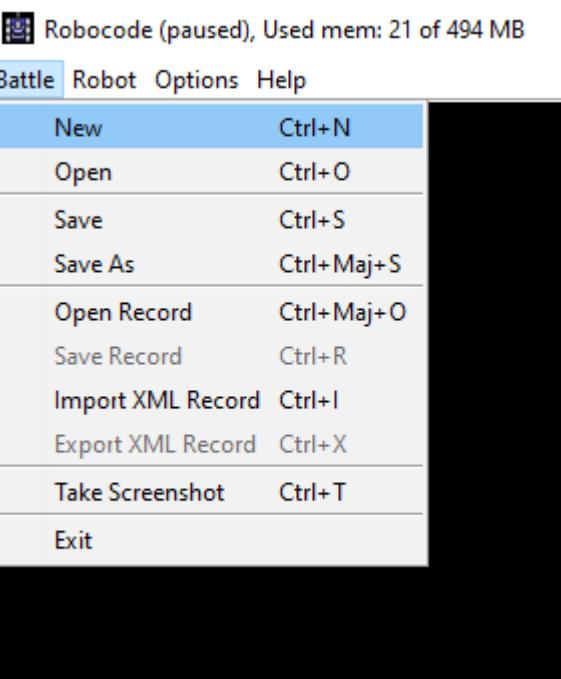
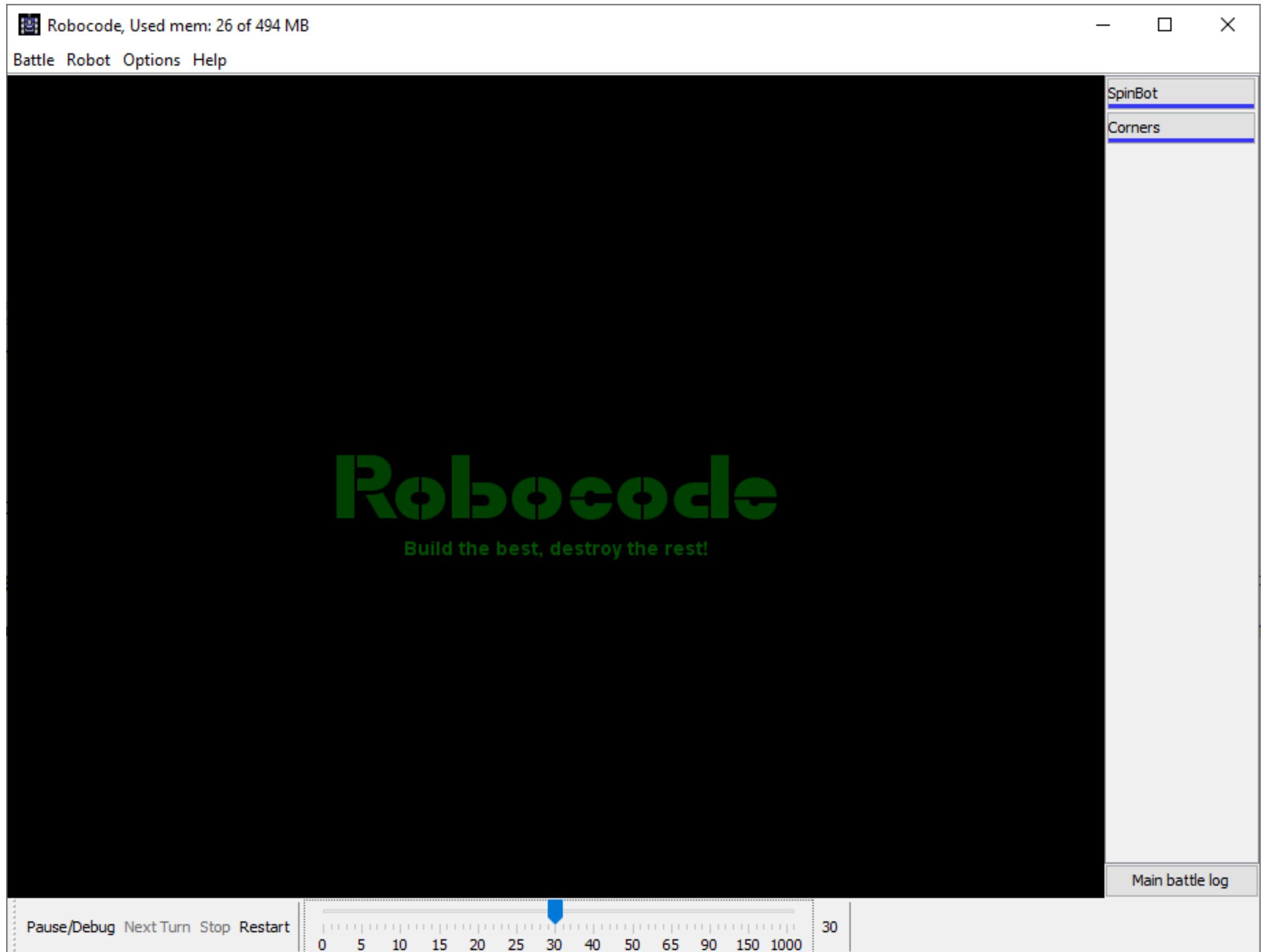
Doc

<https://robocode.sourceforge.io/>

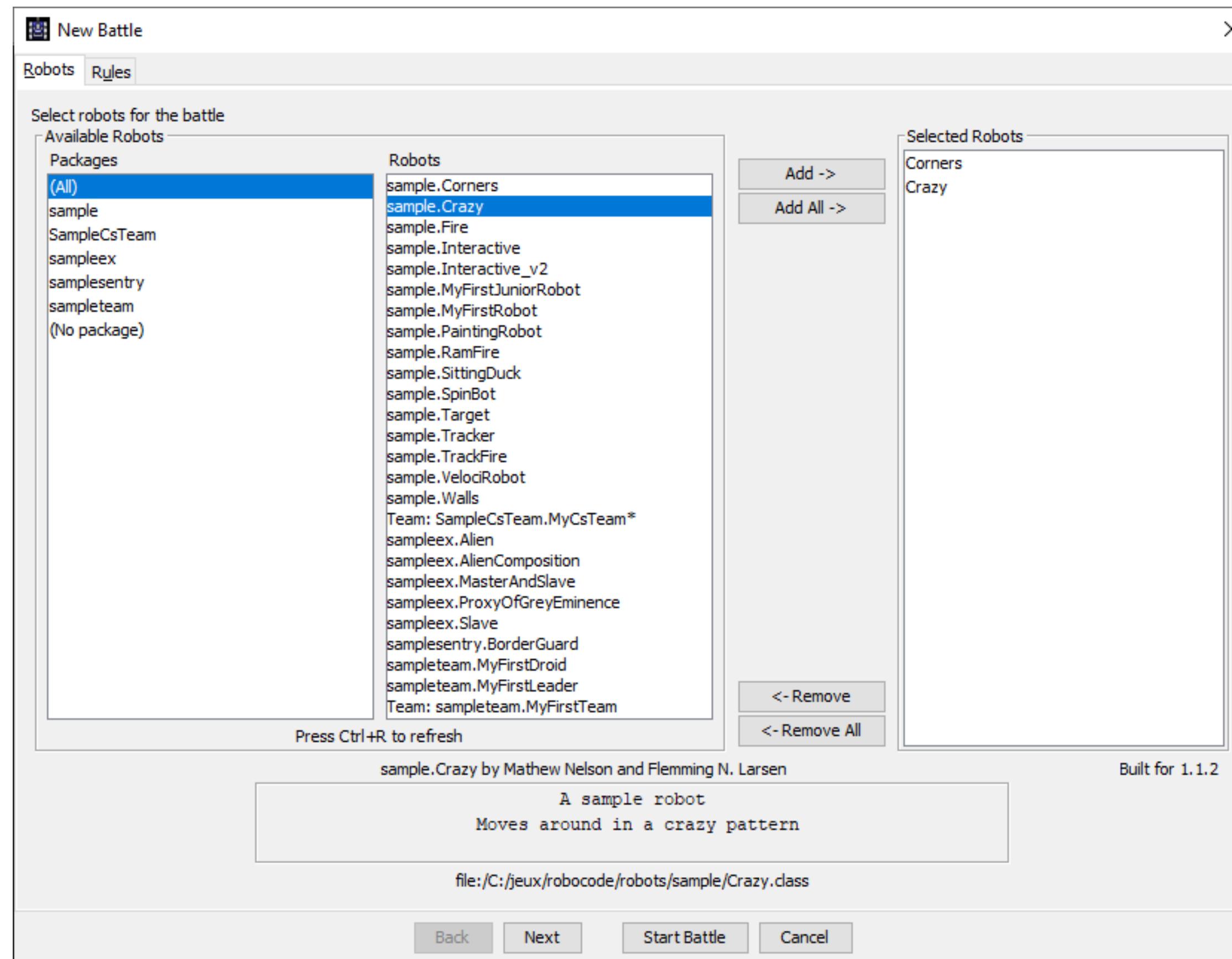
<https://robocode.sourceforge.io/docs/robocode.dotnet>

Robocode

Démarrage

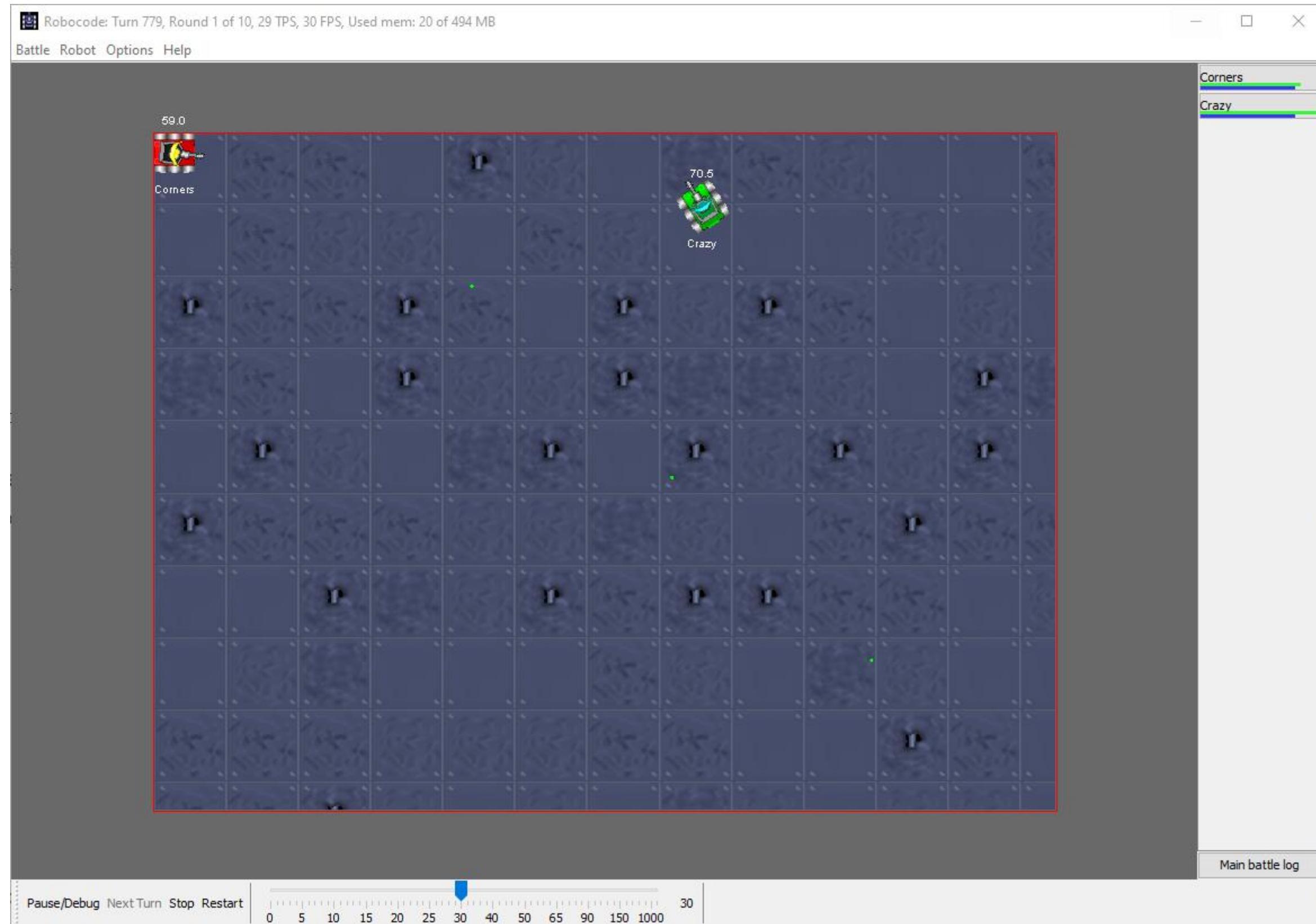


Menu battle / new



Go !

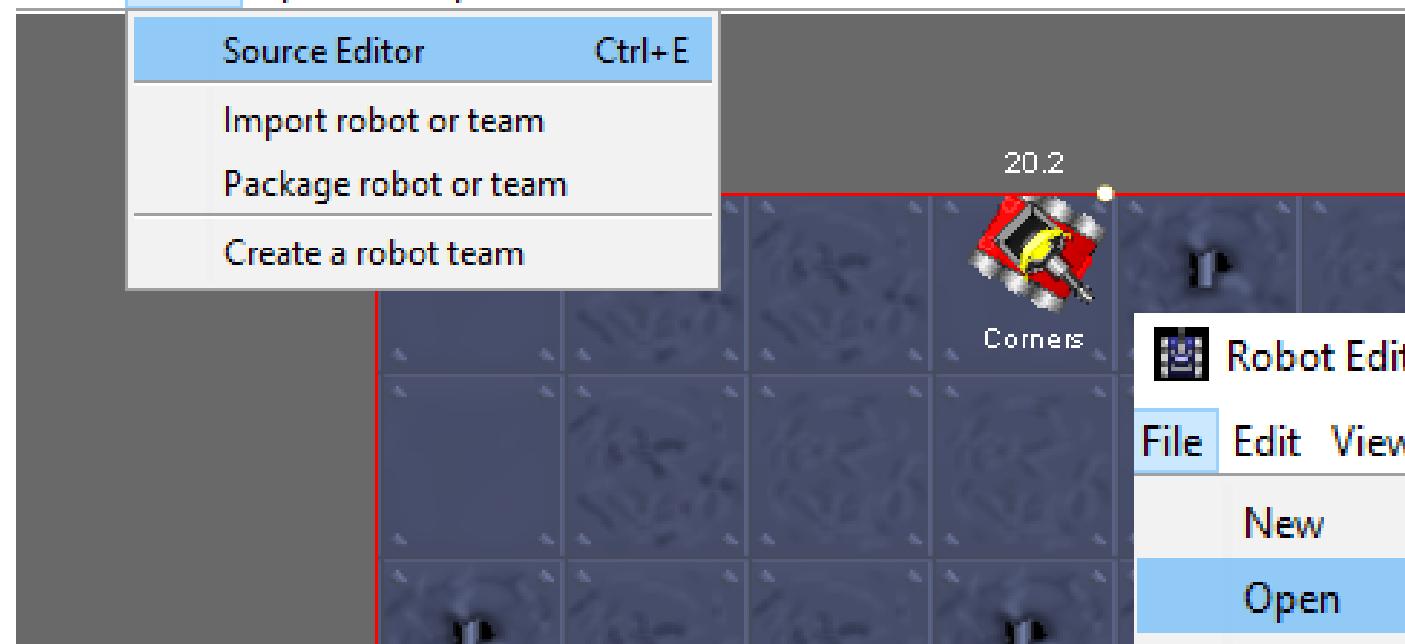
7



Voir la source d'un robot

Robocode: Turn 1554, Round 2 of 10 (paused), Used mem: 24 of 494 MB

Battle Robot Options Help



Robot Editor
File Edit View Compiler Window Help

New
Open Ctrl+O
Extract downloaded robot for editing
Save Ctrl+S
Save As Ctrl+Maj+S
Exit

```
1 package sample;
2 import robocode.*;
3 //import java.awt.*;
4 //import java.awt.event.*;
5 //import java.util.*;
6 //import java.util.*;
7 /**
8 * Test - a robot by (your name here)
9 */
```

Robot Editor
File Edit View Compiler Window Help

Editing - Test

```
1 package sample;
2 import robocode.*;
3 //import java.awt.*;
4 //import java.awt.event.*;
5 //import java.util.*;
6 //import java.util.*;
7 /**
8 * Test - a robot by (your name here)
9 */
```

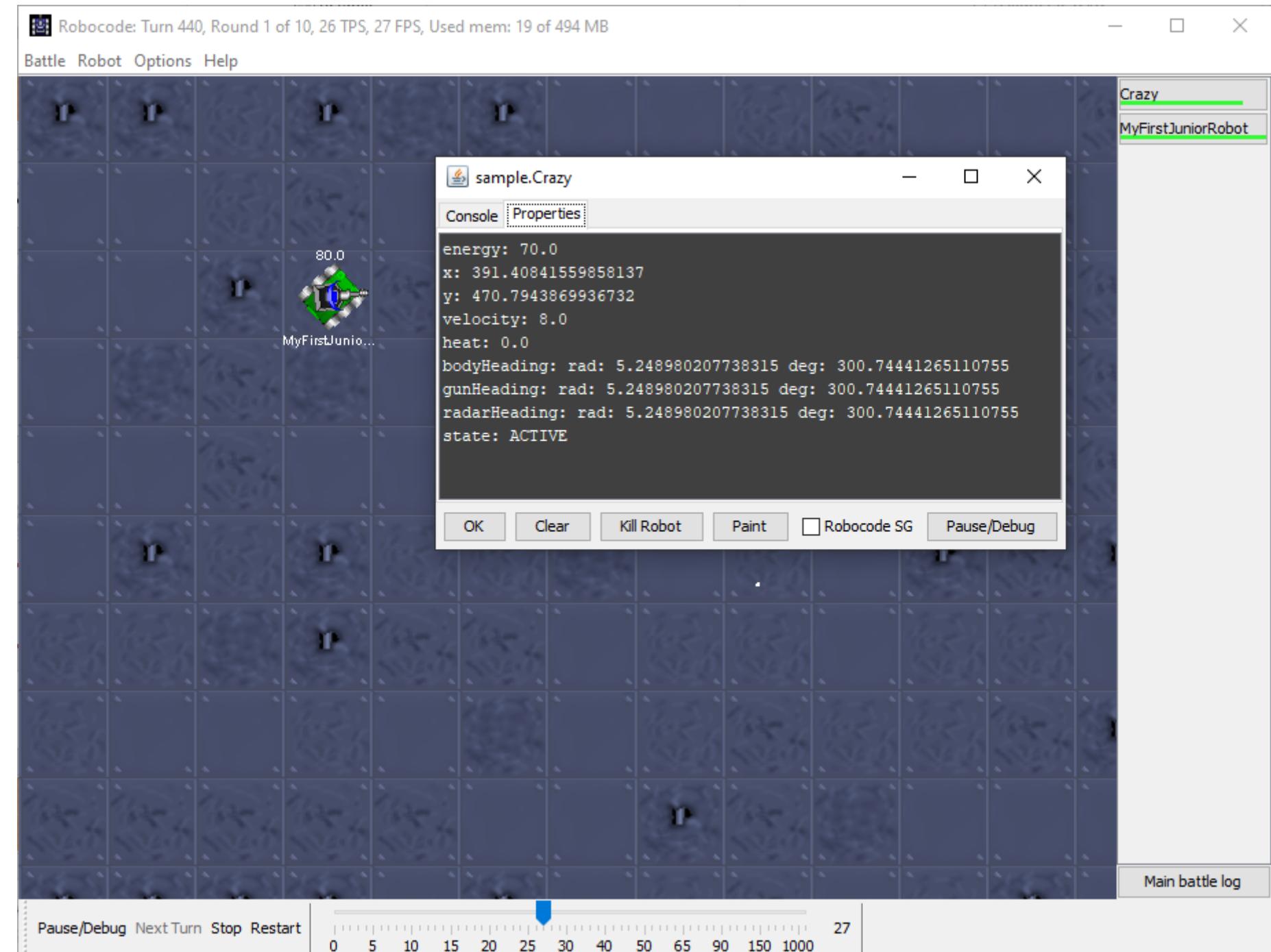
Ouvrir Rechercher dans : sample

Documents r... Bureau Documents

- SittingDuck.data
- Walls.java
- Corners.java
- Crazy.java
- Fire.java
- Interactive.java
- Interactive_v2.java
- MyFirstJuniorRobot.java
- MyFirstRobot.java
- PaintingRobot.java
- RamFire.java
- SittingDuck.java
- SpinBot.java

Propriétés

9



Création d'un robot avec VS

Opérations

Créer une nouvelle solution Robocode avec un projet Console Robocode

Ajouter un projet bibliothèque de classe appelée Robots

Ajouter une référence à ce projet : c:\robocode\libs\robocode.dll et ne pas copier le fichier en local

Ajouter une classe MonPremierRobot.cs

Déclarer le using : using Robocode;

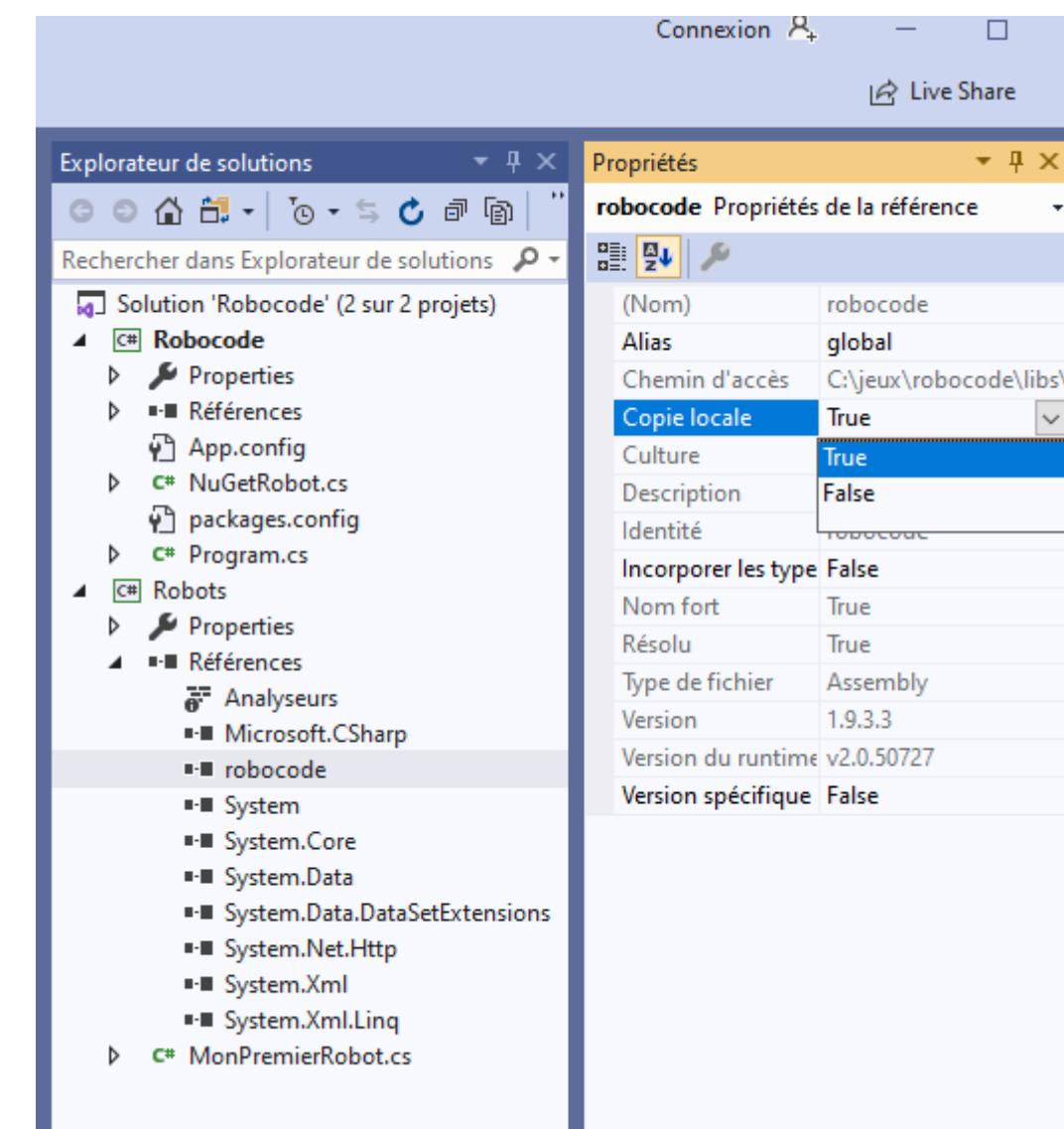
Changer le namespace (utilisez vos initiales) : namespace FJU

Faire hériter votre classe de la classe Robot

Ajouter cette méthode

```
public override void Run()
{
    // Perform your initialization for your robot here

    while (true)
    {
        // Perform robot logic here calling robot commands etc.
    }
}
```



Exemple de code

```
// The main method of your robot containing robot logics
Orréférences
public override void Run()
{
    // -- Initialization of the robot --

    // Here we turn the robot to point upwards, and move the gun 90 degrees
    TurnLeft(Heading - 90);
    TurnGunRight(90);

    // Infinite loop making sure this robot runs till the end of the battle round
    while (true)
    {
        // -- Commands that are repeated forever --

        // Move our robot 5000 pixels ahead
        Ahead(5000);

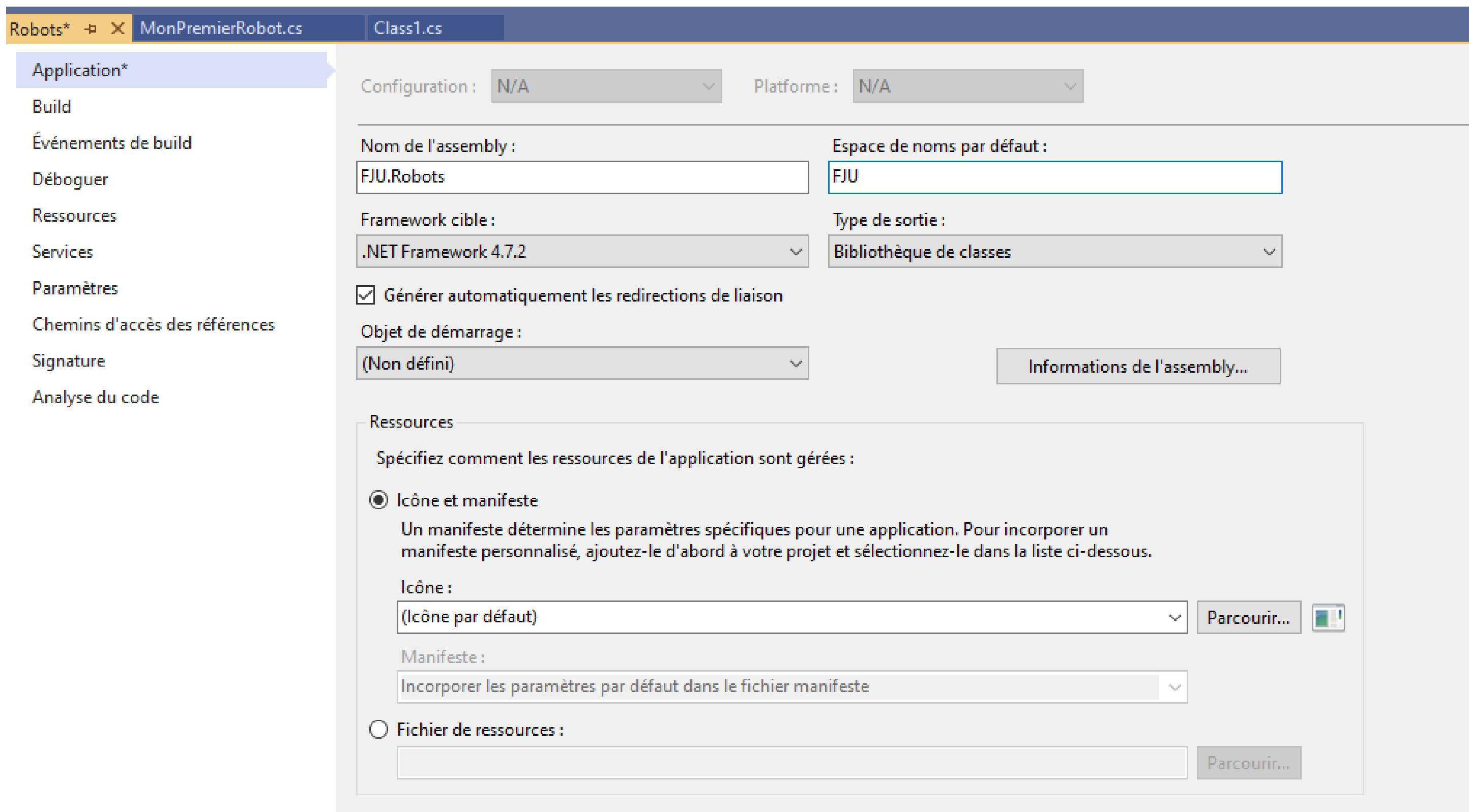
        // Turn the robot 90 degrees
        TurnRight(90);

        // Our robot will move along the borders of the battle field
        // by repeating the above two statements.
    }
}

// Robot event handler, when the robot sees another robot
Orréférences
public override void OnScannedRobot(ScannedRobotEvent e)
{
    // We fire the gun with bullet power = 1
    Fire(1);
}
```

Propriétés du projet

Changer le nom de l'assembly et de namespace par défaut



Votre robot dans l'arène

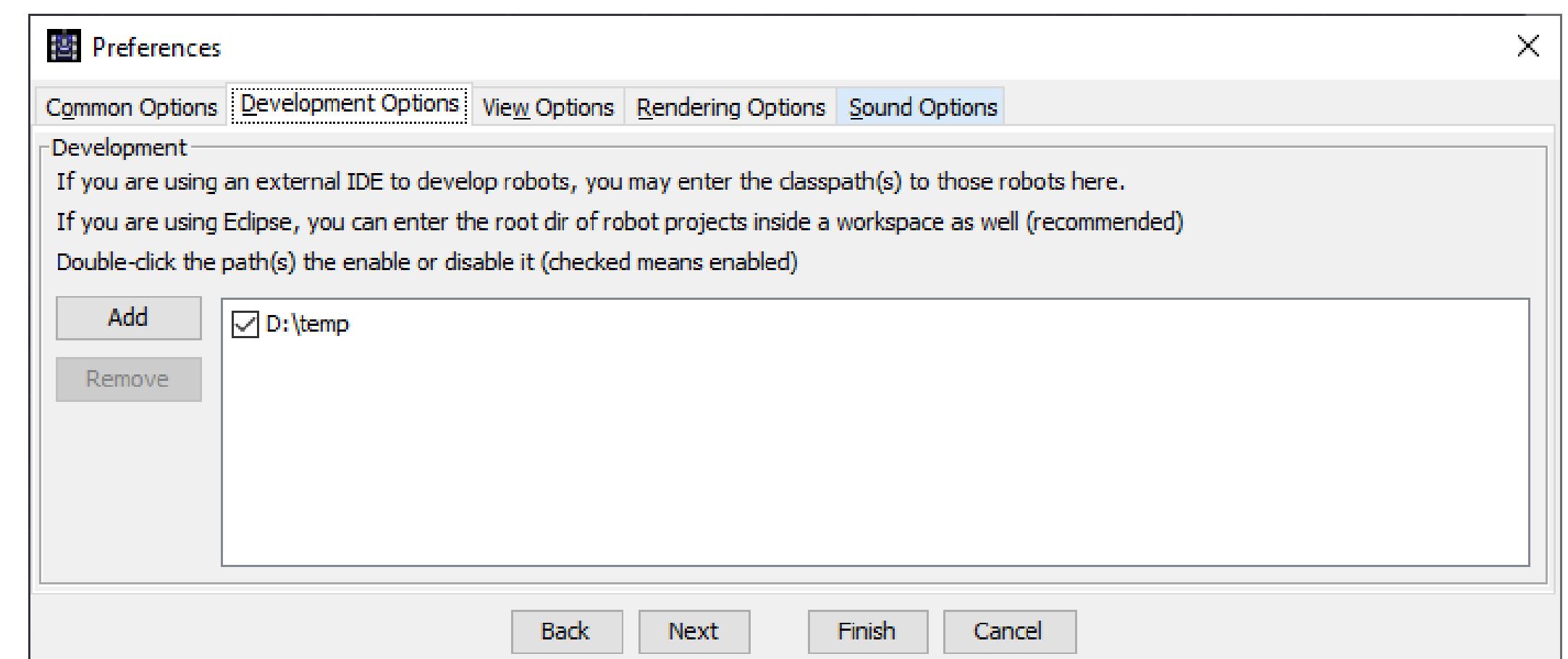
Compiler votre projet

	<RÉP>
dll	4 608
pdb	15 872
dll	114 688
xml	446 816

Sur robocode, dans préférences, développement options

Cliquer sur add et ajouter le chemin vers votre build

!! Attention au C# dans le nom du chemin, ça plante.



Changer le robot

```
// Set colors
BodyColor = (Color.Red);
GunColor = (Color.Black);
RadarColor = (Color.Yellow);
BulletColor = (Color.Green);
ScanColor = (Color.Green);

Energy           VelocityRate

BodyColor = (Color.FromArgb(0, 200, 0));
GunColor = (Color.FromArgb(0, 150, 50));
RadarColor = (Color.FromArgb(0, 100, 100));
BulletColor = (Color.FromArgb(255, 255, 100));
ScanColor = (Color.FromArgb(255, 200, 200));

// Tell the game we will want to turn right 90
SetTurnRight(90);
// At this point, we have indicated to the game that *when we do something*,
// we will want to move ahead and turn right. That's what "set" means.
// It is important to realize we have not done anything yet!
// In order to actually move, we'll want to call a method that
// takes real time, such as WaitFor.
// WaitFor actually starts the action -- we start moving and turning.
// It will not return until we have finished turning.
WaitFor(new TurnCompleteCondition(this));
// Note: We are still moving ahead now, but the turn is complete.
// Now we'll turn the other way...
SetTurnLeft(180);
// ... and wait for the turn to finish ...
WaitFor(new TurnCompleteCondition(this));
// ... then the other way ...
SetTurnRight(180);
// .. and wait for that turn to finish.
WaitFor(new TurnCompleteCondition(this));
// then back to the top to do it all again

Stop(); Scan(); Resume();

TurnRight(Utils.NormalRelativeAngleDegrees(corner - Heading));
```

Environnement

```
private int others; // Number of other robots in the game  
// Save # of other bots  
others = Others; BattleFieldWidth, BattleFieldHeight  
  
IsAdjustGunForRobotTurn = (true); // Keep the gun still when we turn  
Fire(l);  
  
public override void OnHitByBullet(HitByBulletEvent e)  
    e.Heading HitByBulletBearing;  
  
public override void OnHitWall(HitWallEvent e)  
  
public override void OnScannedRobot(ScannedRobotEvent e)  
    e.Distance TurnGunTo(ScannedAngle); e.Bearing  
  
public override void OnDeath(DeathEvent e)  
  
public override void OnHitRobot(HitRobotEvent e)  
    e.IsMyFault e.Energy  
    e.Bearing
```

Environnement

```

public void onStatus(StatusEvent e) {}

public void onBulletHit(BulletHitEvent e) {}

public void onBulletHitBullet(BulletHitBulletEvent e) {}

public void onBulletMissed(BulletMissedEvent e) {}

public void onDeath(DeathEvent e) {}

public void onHitRobot(HitRobotEvent e) {}

public void onHitWall(HitWallEvent e) {}

public void onRobotDeath(RobotDeathEvent e) {}

public void onWin(WinEvent e) {}

```

```

private int trigger; // Keeps track of when to move
trigger = 80;
// Add a custom event named "trigger hit",
AddCustomEvent(
    new Condition("triggerhit",
        (c) =>
    {
        return Energy <= trigger;
    }));

```

```

public override void OnCustomEvent(CustomEvent e)
{
    // If our custom event "triggerhit" went off,
    if (e.Condition.Name == "triggerhit")
    {
        // Adjust the trigger value, or
        // else the event will fire again and again and again...
        trigger -= 20;
        Out.WriteLine("Ouch, down to " + (int) (Energy + .5) + " energy.");
        // move around a bit.
        TurnLeft(65);
        Ahead(100);
    }
}

```

Clavier

```

public override void OnKeyPressed(KeyEvent e)
{
    switch (e.KeyCode)
    {
        case Keys.VK_UP:
            // Arrow up key: move direction = forward (infinitely)
            moveDirection = 1;
            moveAmount = Double.PositiveInfinity;
            break;

        case Keys.VK_DOWN:
            // Arrow down key: move direction = backward (infinitely)
            moveDirection = -1;
            moveAmount = Double.PositiveInfinity;
            break;

        case Keys.VK_RIGHT:
            // Arrow right key: turn direction = right
            turnDirection = 1;
            break;

        case Keys.VK_LEFT:
            // Arrow left key: turn direction = left
            turnDirection = -1;
            break;
    }
}

// Called when a key has been released (after being pressed)
public override void OnKeyReleased(KeyEvent e)
{
    switch (e.KeyCode)
    {
        case Keys.VK_UP:
        case Keys.VK_DOWN:
            // Arrow up and down keys: move direction = stand still
            moveDirection = 0;
            moveAmount = 0;
            break;

        case Keys.VK_RIGHT:
        case Keys.VK_LEFT:
            // Arrow right and left keys: turn direction = stop turning
            turnDirection = 0;
            break;
    }
}

```

Souris

```
// Called when the mouse wheel is rotated
public override void OnMouseWheelMoved(MouseWheelMovedEvent e)
{
    // If the wheel rotation is negative it means that it is moved forward.
    // Set move direction = forward, if wheel is moved forward.
    // Otherwise, set move direction = backward
    moveDirection = (e.WheelRotation < 0) ? 1 : -1;

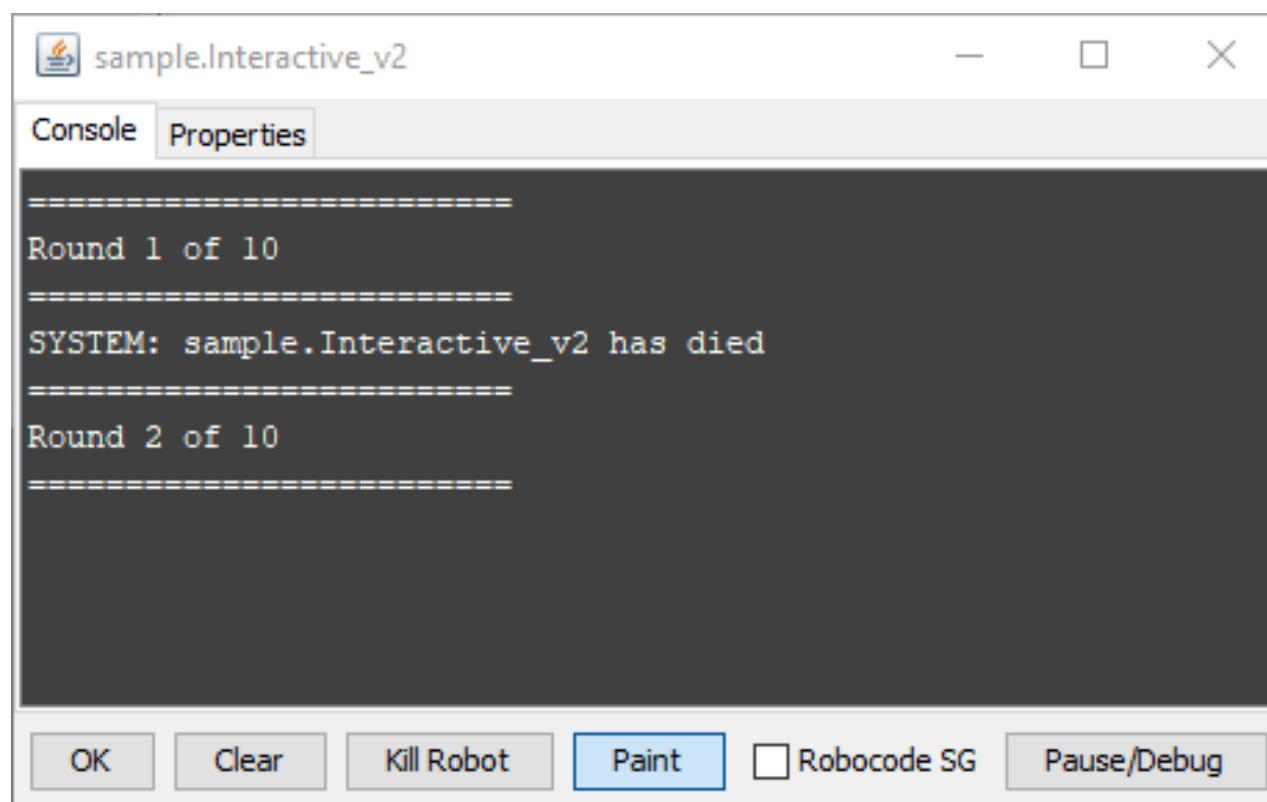
    // Set the amount to move = absolute wheel rotation amount * 5 (speed)
    // Here 5 means 5 pixels per wheel rotation step. The higher value, the
    // more speed
    moveAmount += Math.Abs(e.WheelRotation)*5;
}

// Called when the mouse has been moved
public override void OnMouseMoved(MouseEvent e)
{
    // Set the aim coordinate = the mouse pointer coordinate
    aimX = e.X;
    aimY = e.Y;
}

// Called when a mouse button has been pressed
public override void OnMousePressed(MouseEvent e)
{
    if (e.Button == Keys.BUTTON3)
    {
        // Button 3: Fire power = 3 energy points, bullet color = red
        firePower = 3;
        BulletColor = (Color.Red);
    }
    else if (e.Button == Keys.BUTTON2)
    {
        // Button 2: Fire power = 2 energy points, bullet color = orange
        firePower = 2;
        BulletColor = (Color.Orange);
    }
    else
    {
        // Button 1 or unknown button:
        // Fire power = 1 energy points, bullet color = yellow
        firePower = 1;
        BulletColor = (Color.Yellow);
    }
}

// Called when a mouse button has been released (after being pressed)
public override void OnMouseReleased(MouseEvent e)
{
    // Fire power = 0, which means "don't Fire"
    firePower = 0;
}
```

Paint



```
public override void OnPaint(IGraphics graphics)
{
    var transparentGreen = new SolidBrush(Color.FromArgb(30, 0, 0xFF, 0));
    graphics.FillEllipse(transparentGreen, (int)(X - 60), (int)(Y - 60), 120, 120);
    graphics.DrawEllipse(Pens.Red, (int)(X - 50), (int)(Y - 50), 100, 100);
}
```



```
public override void OnPaint(IGraphics g)
{
    // Draw a red cross hair with the center at the current aim
    // coordinate (x,y)
    g.DrawEllipse(Pens.Red, aimX - 15, aimY - 15, 30, 30);
    g.DrawLine(Pens.Red, aimX, aimY - 4, aimX, aimY + 4);
    g.DrawLine(Pens.Red, aimX - 4, aimY, aimX + 4, aimY);
}
```



```
public void onHitByBullet(HitByBulletEvent e) {
    // debugging by painting to battle view
    Graphics2D g = getGraphics();

    g.setColor(Color.orange);
    g.drawOval((int)(getX() - 55), (int)(getY() - 55), 110, 110);
    g.drawOval((int)(getX() - 56), (int)(getY() - 56), 112, 112);
    g.drawOval((int)(getX() - 59), (int)(getY() - 59), 118, 118);
    g.drawOval((int)(getX() - 60), (int)(getY() - 60), 120, 120);
    g.fillOval((int)(getX() - 60), (int)(getY() - 60), 120, 120);
```

Track this robot

```
private String trackName;  
  
trackName = null; // Initialize to not tracking anyone  
  
public override void OnScannedRobot(ScannedRobotEvent e)  
{  
    // If we have a target, and this isn't it, return immediately  
    // so we can get more ScannedRobotEvents.  
    if (trackName != null && e.Name != trackName)
```

Danse

```
public override void OnWin(WinEvent e)
{
    for (int i = 0; i < 50; i++)
    {
        TurnRight(30);
        TurnLeft(30);
    }
}
```

Trucs utiles

<https://robocode.sourceforge.io/docs/robocode.dotnet/html/ced5662b-8084-48d8-dd83-20fc3665d3be.htm>

```
// demonstrate feature of debugging properties on RobotDialog  
setDebugProperty("lastHitBy", e.getName() + " with power of bullet " + e.getPower() + " at time " + getTime());  
  
// show how to remove debugging property  
setDebugProperty("lastScannedRobot", null);
```

Exercice n°1 : Robot simple

Première compétition

Score total - C'est l'addition de tout le reste, et détermine le rang de chaque robot dans cette bataille.

Score de survie - Chaque robot qui est encore vivant marque 50 points à chaque fois qu'un autre robot meurt.

Bonus du dernier survivant - Le dernier robot vivant gagne 10 points supplémentaires pour chaque robot mort avant lui.

Dommages par balle - Les robots marquent 1 point pour chaque point de dégâts qu'ils infligent à leurs ennemis.

Bonus de dégâts de balles - Lorsqu'un robot tue un ennemi, il marque 20% de plus de tous les dégâts qu'il a infligés à cet ennemi.

Dégâts au bélier - Les robots marquent 2 points pour chaque point de dégâts qu'ils causent en éperonnant leurs ennemis.

Bonus de dégâts de bélier - Lorsqu'un robot tue un ennemi en l'éperonnant, il marque 30% de plus de tous les dégâts qu'il a infligés à cet ennemi.

1sts, 2nds, 3rds - Ceux-ci ne contribuent pas réellement au score, mais sont là pour montrer combien de temps le robot a survécu, c'est-à-dire le nombre de tours où le robot a été placé 1er, 2ème, et 3ème.

Programmer votre premier robot

Vous devez créer une classe contenant un premier robot

Compiler la classe

Indiquer le chemin à robocode

Charger votre robot et le mettre en compétition avec des robots exemples

Exercice n°2 : Robot avancé

Héritage classe JuniorRobot

Une interface robot pour créer le type de robot le plus primitif, qui est un JuniorRobot.

Un robot junior est plus simple que la classe Robot.

Un robot junior a un modèle simplifié, dans le but d'enseigner les techniques de programmation à des étudiants inexpérimentés en programmation.

Le modèle de robot simplifié empêchera le joueur d'être submergé par les règles, la syntaxe de programmation et le concept de programmation de Robocode.

Au lieu d'utiliser des getters et des setters, des champs publics sont fournis pour recevoir des informations comme le dernier robot scanné, les coordonnées du robot, etc.

Toutes les méthodes d'un robot junior bloquent les appels, c'est-à-dire qu'elles ne retournent pas avant que leur action soit terminée et qu'elles prennent au moins un tour pour s'exécuter.

Héritage classe AdvancedRobot

Un type de robot plus avancé que Robot qui permet de ne pas bloquer les appels, de personnaliser les événements et d'écrire dans le système de fichiers.

Héritage classe RateControlRobot

Ce type de robot avancé vous permet de définir un taux pour chacun des mouvements du robot.

Vous pouvez définir le taux pour :

- vitesse - pixels par tour
- rotation du robot - radians par rotation
- rotation des canons - radians par tour
- rotation du radar - radians par tour

Lorsque vous définissez un taux pour l'un des mouvements ci-dessus, le mouvement continuera à un taux spécifié pour toujours, jusqu'à ce que le taux soit modifié. Pour avancer ou reculer, le taux doit être réglé sur une valeur positive. Si une valeur négative est utilisée à la place, le mouvement reviendra en arrière ou vers la gauche. Pour arrêter le mouvement, la vitesse doit être réglée sur 0.

Note : Lors de l'appel de setVelocityRate(), setTurnRate(), setGunRotationRate(), setRadarRotationRate(), setRadarRotationRate() et variantes, tout appel précédent à des fonctions "movement" en dehors de RateControlRobot, comme setAhead(), setTurnLeft(), setTurnRadarRightRadians() et similaires sera écrasé lorsque l'on appelle execute() dans cette classe de robot.

Consultez le code source de l'exemple VelociRobot pour voir comment utiliser ce type de robot.

Héritage IBorderSentry

Un robot qui implémente BorderSentry est un type de robot utilisé pour tenir les autres robots à l'écart des frontières, c'est-à-dire pour protéger les frontières afin d'empêcher les "chenilles murales".

Les robots qui implémentent BorderSentry ont 400 vies/énergies supplémentaires (500 au total), mais sont placés à la limite du champ de bataille lorsque le jeu commence.

Les robots sentinelles frontaliers ne peuvent pas s'éloigner de la zone frontalière, et ils ne peuvent endommager que les robots qui se déplacent dans la zone frontalière.

La taille de la zone frontalière est déterminée par les règles de combat.

Ce type de robot est destiné à être utilisé dans les batailles où les robots doivent être éloignés des frontières afin d'éviter les "rampes murales".

Les robots sentinelles frontaliers n'obtiennent pas de points, et n'apparaîtront pas dans les résultats des batailles ou dans les classements.

Héritage IPaintRobot

Une interface robot qui permet à un robot de recevoir des événements de peinture.

```
void onPaint(Graphics2D g)
```

Cette méthode est appelée à chaque fois que le robot est peint. Vous devez remplacer cette méthode si vous voulez dessiner des objets pour votre robot sur le champ de bataille, par exemple des cibles, des balles virtuelles, etc.

Cette méthode est très utile pour déboguer votre robot.

Notez que le robot ne sera peint que si la fonction "Peindre" est activée sur la fenêtre de la console du robot, sinon le robot ne sera jamais peint (la raison étant que tous les robots peuvent avoir des éléments graphiques qui doivent être peints, et alors vous ne serez peut-être pas capable de dire quels éléments graphiques ont été peints pour votre robot).

Notez également que le système de coordonnées du contexte graphique dans lequel vous peignez les éléments correspond au système de coordonnées Robocode où (0, 0) se trouve dans le coin inférieur gauche du champ de bataille, où X est vers la droite et Y est vers le haut.

BorderGuard.cs

Analyse du code du robot

```
public class BorderGuard
    : AdvancedRobot, IBorderSentry
{
    // Constants
    const double FIREPOWER = 3; // Max. power => violent as this robot can afford it!
    const double HALF_ROBOT_SIZE = 18; // Robot size is 36x36 units, so the half size is 18 units

    // Dictionary containing data for all scanned robots.
    // The key to the dictionary is a robot name and the value is an object containing robot data.
    IDictionary<string, RobotData> enemyDictionary;
    // List containing robot names where the order of names reflects the when these names was accessed the last time
    // within the enemy dictionary. The first in the list is the name of the robot that was accessed the last time,
    // and the last in the list is the oldest one.
    IList<string> accessedEnemyList;

    // Scanning direction, where the radar turns to the right with positive values, and turns
    // to the left with negative values.
    double scanDir = 1;

    // Oldest scanned robot. Can be null.
    RobotData oldestScanned;

    // Target robot for the gun. Can be null meaning that there is currently no target robot.
    RobotData target;

    // Last time when the robot shifted its direction
    long lastDirectionShift;

    // Current direction, where 1 means ahead (forward) and -1 means back
    int direction = 1;

    /// <summary>
    /// Constructs this robot.
    /// </summary>
    public BorderGuard()
```

Programmer votre deuxième robot

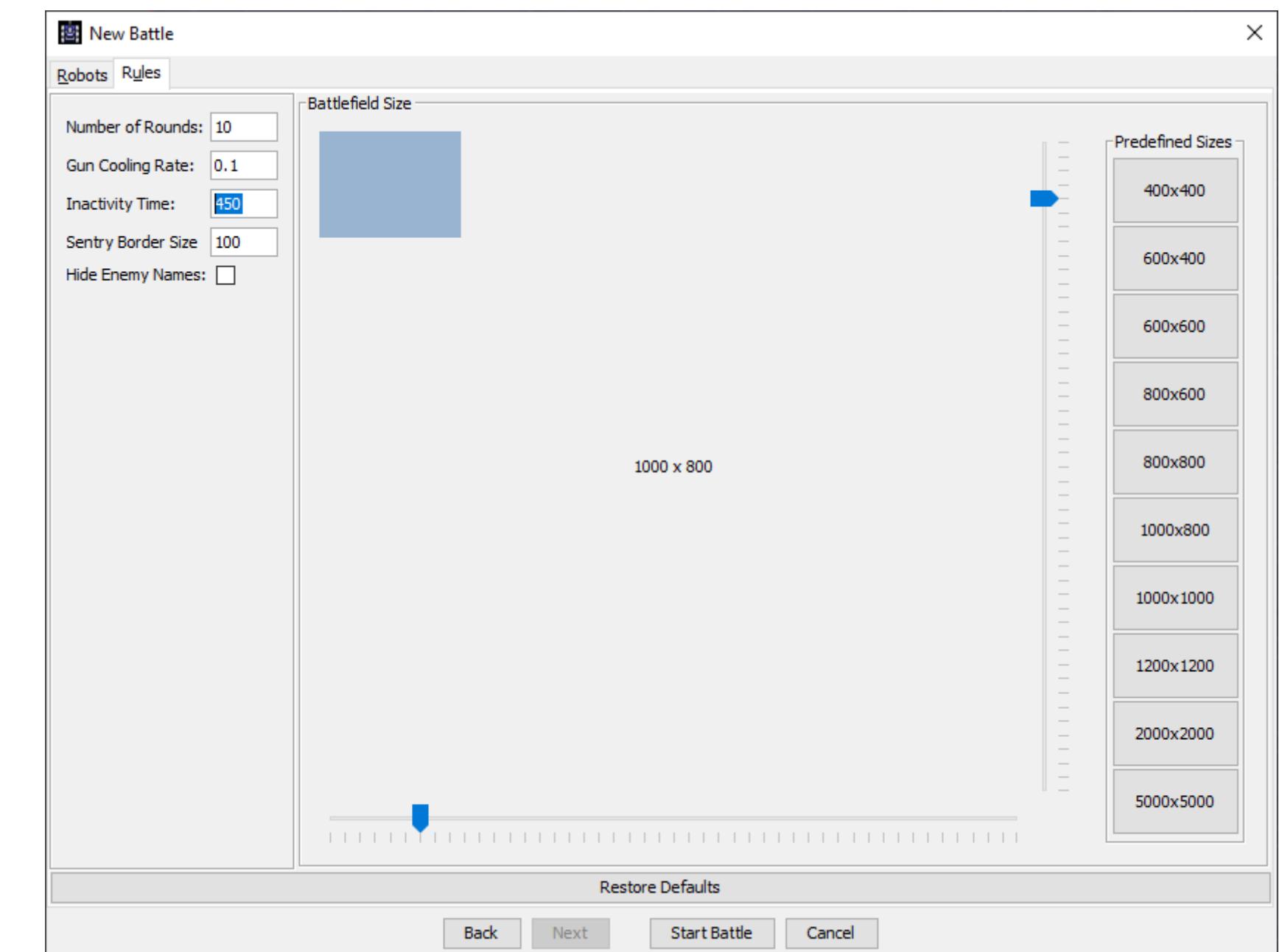
- Vous devez créer un robot avancé
- Compiler la classe
- Indiquer le chemin à robocode
- Charger votre robot et le mettre en compétition avec des robots exemples

Compétition n°1 : Big war

Construire votre robot pour la compétition

36

- Créer un nouveau projet de type bibliothèque de classe «CompetitionRobots»
- Utiliser le namespace M1IL
- Créer une classe qui contient votre robot. Cette classe doit avoir comme nom vos initiales (1^{ère} lettre du prénom, 2 ières lettre du nom) et le nom de votre robot (exemple : FJU_TheDestructor)
- Envoyer le fichier source sur :
- <http://edu.franck-jubin.fr/csharp/robocode/>
- Que le meilleur gagne



Exercice n°3 Équipe de Robots

Héritage classe TeamRobot

Un type avancé de robot qui prend en charge l'envoi de messages entre les coéquipiers d'une équipe de robots.

<https://robocode.sourceforge.io/docs/robocode/robocode/TeamRobot.html>

Modifier and Type	Method and Description
void broadcastMessage(Serializable message)	broadcastMessage(Serializable message) Broadcasts a message to all teammates.
Vector<MessageEvent> getMessageEvents()	getMessageEvents() Returns a vector containing all MessageEvents currently in the robot's queue.
ITeamEvents getTeamEventListener()	getTeamEventListener() Do not call this method!
String[] getTeammates()	getTeammates() Returns the names of all teammates, or null there is no teammates.
boolean isTeammate(String name)	isTeammate(String name) Checks if a given robot name is the name of one of your teammates.
void onMessageReceived(MessageEvent event)	onMessageReceived(MessageEvent event) This method is called when your robot receives a message from a teammate.
void sendMessage(String name, Serializable message)	sendMessage(String name, Serializable message) Sends a message to one (or more) teammates.

Héritage classe TeamRobot

2 fichiers java

```
package sampleteam;

import java.awt.*;

/**
 * RobotColors - A serializable class to send Colors to teammates.
 *
 * @author Mathew A. Nelson (original)
 * @author Flemming N. Larsen (contributor)
 */
public class RobotColors implements java.io.Serializable {

    private static final long serialVersionUID = 1L;

    public Color bodyColor;
    public Color gunColor;
    public Color radarColor;
    public Color scanColor;
    public Color bulletColor;
}
```

```
package sampleteam;

/**
 * Point - a serializable point class
 */
public class Point implements java.io.Serializable {

    private static final long serialVersionUID = 1L;

    private double x = 0.0;
    private double y = 0.0;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }
}
```

Héritage classe TeamRobot

```

public override void Run()
{
    // Prepare RobotColors object
    var c = new RobotColors();

    c.bodyColor = Color.Red;
    c.gunColor = Color.Red;
    c.radarColor = Color.Red;
    c.scanColor = Color.Yellow;
    c.bulletColor = Color.Yellow;

    // Set the color of this robot containing the RobotColors
    BodyColor = c.bodyColor;
    GunColor = c.gunColor;
    RadarColor = c.radarColor;
    ScanColor = c.scanColor;
    BulletColor = c.bulletColor;
    try
    {
        // Send RobotColors object to our entire team
        BroadcastMessage(c);
    }
    catch (Exception)
    {
    }

    // Normal behavior
    while (true)
    {
        SetTurnRadarRight(10000);
        Ahead(100);
        Back(100);
    }
}

```

```

public override void OnMessageReceived(MessageEvent e)
{
    // Fire at a point
    if (e.Message is PointD)
    {
        var p = (PointD)e.Message;
        // Calculate x and y to target
        double dx = p.X - X;
        double dy = p.Y - Y;
        // Calculate angle to target
        double theta = Utils.ToDegrees(Math.Atan2(dx, dy));

        // Turn gun to target
        TurnGunRight(Utils.NormalRelativeAngleDegrees(theta - GunHeading));
        // Fire hard!
        Fire(3);
    }
    else if (e.Message is RobotColors)
    {
        // Set our colors
        var c = (RobotColors) e.Message;

        BodyColor = c.bodyColor;
        GunColor = c.gunColor;
        RadarColor = c.radarColor;
        ScanColor = c.scanColor;
        BulletColor = c.bulletColor;
    }
}

```

Héritage classe TeamRobot

```

public override void OnScannedRobot(ScannedRobotEvent e)
{
    // Don't Fire on teammates
    if (IsTeammate(e.Name))
    {
        return;
    }
    // Calculate enemy bearing
    double enemyBearing = Heading + e.Bearing;
    // Calculate enemy's position
    double enemyX = X + e.Distance*Math.Sin(Utils.ToRadians(enemyBearing));
    double enemyY = Y + e.Distance*Math.Cos(Utils.ToRadians(enemyBearing));

    try
    {
        // Send enemy position to teammates
        BroadcastMessage(new PointD(enemyX, enemyY));
    }
    catch (Exception ex)
    {
        Out.WriteLine("Unable to send order: ");
        Out.WriteLine(ex);
    }
}

/// <summary>
///     OnHitByBullet: Turn perpendicular to bullet path
/// </summary>
public override void OnHitByBullet(HitByBulletEvent e)
{
    TurnLeft(90 - e.Bearing);
}

```

```

public override void OnMessageReceived(MessageEvent e)
{
    // Fire at a point
    if (e.Message is PointD)
    {
        var p = (PointD)e.Message;
        // Calculate x and y to target
        double dx = p.X - X;
        double dy = p.Y - Y;
        // Calculate angle to target
        double theta = Utils.ToDegrees(Math.Atan2(dx, dy));

        // Turn gun to target
        TurnGunRight(Utils.NormalRelativeAngleDegrees(theta - GunHeading));
        // Fire hard!
        Fire(3);
    }
    else if (e.Message is RobotColors)
    {
        // Set our colors
        var c = (RobotColors) e.Message;

        BodyColor = c.bodyColor;
        GunColor = c.gunColor;
        RadarColor = c.radarColor;
        ScanColor = c.scanColor;
        BulletColor = c.bulletColor;
    }
}

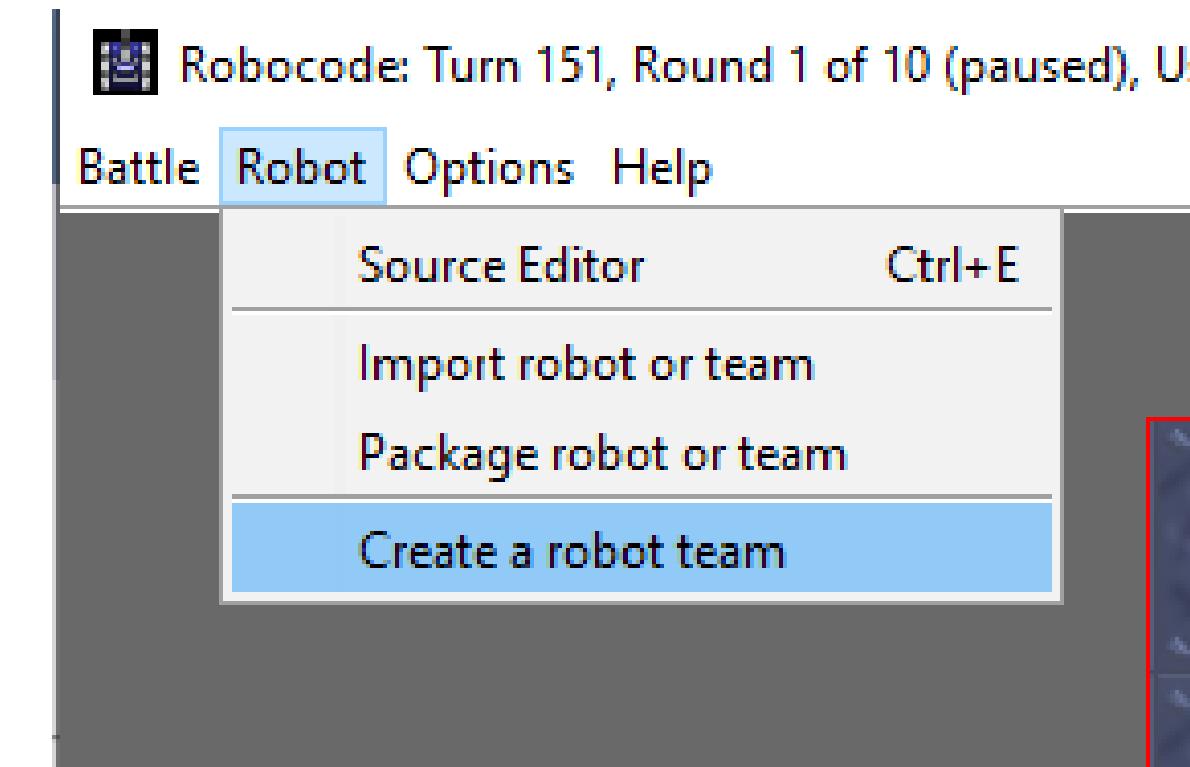
```

Héritage classe Droid

Les robots qui implémentent Droid n'ont pas de scanner, mais 20 vies/énergie supplémentaires.
Cette classe est destinée à être utilisé en équipe.

Construire votre robot pour la compétition⁴³

- Dans bibliothèque de classe «CompetitionRobots»
- Créer 2 robots qui doivent coopérer à l'aide de message et qui prenne les mêmes couleurs toujours en gardant la même convention de nom
- Créer une équipe avec ces 2 robots
- Lancer une bataille sur 10 rounds



Compétition n°2 : 2 camps

Construire votre robot pour la compétition⁴⁵

- Les 2 équipes sont définis par la ligne centrale (Team1 et Team2); On équilibre en même nombre de joueurs
- Chaque joueur crée un robot pour son équipe
 - Utiliser le namespace M1IL
 - Qui prend les couleurs choisies par l'équipe ou vous pouvez désigner un leader qui envoie l'ordre de changement de couleur
 - Nom du robot Team1_FJU_NomDuRobot
 - Définir la ou les stratégies de son robot (vous pouvez être autonome, prendre des décisions et envoyez des messages)
- Si un joueur décide de trahir son équipe (changement de couleur, tire sur les coéquipiers, blocage, kamikaze), le robot sera automatiquement exclu et l'étudiant sera pénalisé sur l'appréciation du contrôle continu.
- Envoyer le fichier source sur : <http://edu.franck-jubin.fr/csharp/robocode/>
- Que la meilleure équipe gagne

FIN