

Tempo Decorrido - OnBot

Java

Introdução ao tempo decorrido

Uma maneira de criar um código autônomo é usar um temporizador para definir quais ações devem ocorrer quando. Dentro do SDK, as ações podem ser programadas para um temporizador usando o `ElapsedTime`. Os temporizadores consistem em duas categorias principais: contagem regressiva e contagem progressiva. Na maioria das aplicações, um temporizador é considerado um dispositivo que conta regressivamente a partir de um intervalo de tempo especificado, como o temporizador em um telefone ou micro-ondas. No entanto, alguns temporizadores, como cronômetros, contam progressivamente a partir de zero. Esses tipos de temporizadores medem o tempo decorrido. O `ElapsedTime` é um temporizador de contagem progressiva, registrando o tempo decorrido desde o início de um evento definido, como o início de um cronômetro. Neste caso, é o tempo decorrido desde quando o temporizador é instanciado ou redefinido no código. O temporizador `ElapsedTime` começa a contar o tempo decorrido a partir do ponto de sua criação dentro de um código. Por exemplo, nesta seção, o `ElapsedTime` será criado (ou instanciado) na seção de código que ocorre quando o modo operacional é inicializado. Não há opção de parar o temporizador `ElapsedTime`. Em vez disso, a função `reset()` pode ser usada dentro do seu código para reiniciar o temporizador em vários intervalos. Depois que o temporizador é redefinido, o tempo decorrido pode ser consultado chamando métodos como `time()`, `seconds()` ou `milliseconds()`. O tempo fornecido pelos métodos consultados pode ser usado em loops para determinar quanto tempo uma ação específica deve ocorrer.

Para obter mais informações sobre o objeto `ElapsedTime`, consulte a [documentação do Java](#) (Java Docs).

Seções	Objetivos das seções
Noções básicas de programação com tempo	Aprender a lógica para fazer um código autônomo por tempo

Programação com tempo decorrido

Para começar, crie um novo modo operacional (op mode) chamado HelloWorld_ElapsedTime utilizando o exemplo BasicOpMode_Linear. Existem outras características que você pode selecionar para facilitar as coisas ao começar a desenvolver seus modos operacionais autônomos. Por exemplo, como você pode se lembrar, selecionar "Setup Code for Configured Hardware" cria as referências necessárias para o mapa de hardware. Outra opção que você pode escolher é configurar o código como um modo operacional autônomo. Isso adiciona a anotação @Autonomous, que distingue o código como um modo operacional autônomo na aplicação Driver Station.

Ao criar um modo operacional (op mode), é necessário decidir se ele deve ser configurado como modo autônomo. Para aplicações com duração inferior a 30 segundos, geralmente necessárias para jogabilidade competitiva, é recomendado alterar o tipo de op mode para autônomo. Para aplicações com duração superior a 30 segundos, configurar o código como o tipo de op mode autônomo limitará o tempo de execução do seu código autônomo a 30 segundos. Se você planeja ultrapassar os 30 segundos incorporados no SDK, é recomendável manter o código como um tipo de op mode teleoperado. Para obter informações sobre como os op modes funcionam, visite a seção de Introdução à Programação.

The screenshot shows the 'New File' dialog box. The 'File Name' field is 'HelloWorldElapsedTime' with a file type of 'java'. The 'Location' field is 'org.firstinspires.ftc.teamcode'. The 'Sample' dropdown is 'BasicOpModeLinear'. Under the 'Autonomous' section, the 'Autonomous' radio button is selected. The 'Setup Code for Configured Hardware' checkbox is checked. The 'Cancel' and 'OK' buttons are at the bottom right.

A seleção das características discutidas acima permitirá que você comece com o seguinte código:

```
package org.firstinspires.ftc.teamcode;  
  
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
```

```
import com.qualcomm.robotcore.hardware.AnalogInput;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.robotcore.hardware.ColorSensor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.hardware.DigitalChannel;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;
```

```
@Autonomous
```

```
public class HelloWorld_ElapsedTime extends LinearOpMode {
    private DcMotor leftMotor;
    private DcMotor rightMotor;
    private DcMotor arm;
    private Servo claw;
    private DigitalChannel touch;
    private Gyroscope imu;
```

```
@Override
```

```
public void runOpMode() {
    imu = hardwareMap.get(Gyroscope.class, "imu");
    leftMotor = hardwareMap.get(DcMotor.class, "leftmotor");
    rightMotor = hardwareMap.get(DcMotor.class, "rightmotor");
    arm = hardwareMap.get(DcMotor.class, "arm");
    claw = hardwareMap.get(Servo.class, "claw");
    touch = hardwareMap.get(DigitalChannel.class, "touch");

    telemetry.addData("Status", "Initialized");
    telemetry.update();

    // Wait for the game to start (driver presses PLAY)
    waitForStart();

    // run until the end of the match (driver presses STOP)
    while (opModelsActive()){
        telemetry.addData("Status", "Running");
```

```
        telemetry.update();
    }
}
}
```

Como o foco desta seção é o Tempo Decorrido, é necessário criar uma variável de ElapsedTime e uma instância de ElapsedTime. Para fazer isso, a seguinte linha é necessária:

```
private ElapsedTime runtime = new ElapsedTime();
```

A linha acima realiza duas ações. Uma variável privada ElapsedTime chamada "runtime" é criada. Uma vez que "runtime" é criado e definido como uma variável ElapsedTime, ele pode armazenar as informações e dados de tempo relevantes. A outra parte da linha runtime = new ElapsedTime(); cria uma instância do objeto temporizador ElapsedTime e a atribui à variável "runtime".

Adicione essa linha ao op mode junto com as outras variáveis privadas.

```
public class HelloWorld_ElapsedTime extends LinearOpMode {
    private DcMotor leftMotor;
    private DcMotor rightMotor;
    private DcMotor arm;
    private Servo claw;
    private DigitalChannel touch;
    private Gyroscope imu;
    private ElapsedTime runtime = new ElapsedTime();
```

O objetivo deste exemplo é realizar uma série de ações em intervalos de tempo, como avançar por três segundos. Outra maneira de pensar é que o robô avança enquanto o temporizador ElapsedTime for menor ou igual a três segundos, ou seja, runtime.seconds() <= 3.0. Para este exemplo em particular, a melhor maneira de atingir esse objetivo é usar um loop while. Substitua o loop while padrão do op mode pelo seguinte loop:

```
waitForStart();
while (runtime.seconds() <= 3.0) {

}
```

É importante saber que, dentro de um modo operacional linear, um loop while deve sempre ter a condição Booleana opModelsActive(). Essa condição garante que o loop while será encerrado quando o botão de parada for pressionado.

Os loops while executam quando a condição é verdadeira e param quando a condição é falsa. Neste caso, o loop while deve iniciar apenas se ambas as condições (opModelsActive() e runtime.seconds() <= 3.0) forem verdadeiras. O loop while deve terminar quando runtime.seconds() > 3 for maior que três segundos ou quando o botão de parada na estação do piloto for pressionado. Para realizar isso, o operador lógico && precisa ser utilizado.

O operador && é um operador lógico em Java. Este símbolo é equivalente a "e" em Java. Utilizar isso em uma instrução condicional requer que ambas as declarações precisem ser verdadeiras para que a condição geral seja verdadeira.

```
waitForStart();
while (opModelsActive() && (runtime.seconds() <= 3.0)) {

}
```

Lembre-se de que o temporizador ElapsedTime começa a contar quando é instanciado ou resetado. Como o temporizador está sendo instanciado quando a variável runtime está sendo criada, e as criações de variáveis estão ocorrendo antes do comando waitForStart(); ser executado; o temporizador começará a contar quando o modo operacional for inicializado, em vez de quando o modo operacional for iniciado. Isso pode causar problemas na consistência do desempenho do robô, dependendo do atraso entre a inicialização e o início.

Considere o seguinte cenário: Em um ambiente de competição, equipes frequentemente são obrigadas a inicializar seu robô antes do início de uma partida. Isso significa que um robô pode permanecer na fase de inicialização por alguns segundos a alguns minutos. Se um código autônomo é baseado no uso de um temporizador ElapsedTime que começa ao ser instanciado, quanto mais tempo um robô passa na fase de inicialização, menos provável é que ele funcione conforme esperado.

Para evitar problemas decorrentes de um atraso de tempo entre a inicialização e o início, é possível adicionar um reset do temporizador ao código. Adicione a linha runtime.reset(); entre o comando waitForStart(); e o loop while.

```
waitForStart();
runtime.reset();
while (opModelsActive() && (runtime.seconds() <= 3.0)) {

}
```

Agora que o temporizador está resetado, vamos em frente e adicionar o código relacionado aos motores. Se você se lembra do artigo "Programming Drivetrain Motors", os motores no trem de força (drivetrain) se espelham entre si. A natureza espelhada da montagem dos motores faz com

que eles girem em direções opostas. Para corrigir essa discrepância, a direção do motor direito precisa ser invertida. Adicione as seguintes linhas de código ao op mode acima do comando `waitForStart()`;

```
rightMotor.setDirection(DcMotor.Direction.REVERSE);
```

Agora, dentro do loop `while`, adicione as linhas `leftmotor.setPower(1);` e `rightmotor.setPower(1);` para definir ambos os motores para funcionar em velocidade máxima na direção para frente.

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.AnalogInput;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.robotcore.hardware.ColorSensor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.hardware.DigitalChannel;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;
import com.qualcomm.robotcore.util.ElapsedTime;
```

```
@Autonomous
```

```
public class HelloWorld_ElapsedTime extends LinearOpMode {
    private DcMotor leftMotor;
    private DcMotor rightMotor;
    private DcMotor arm;
    private Servo claw;
    private DigitalChannel touch;
    private Gyroscope imu;
    private ElapsedTime runtime = new ElapsedTime();

    @Override
    public void runOpMode() {
        imu = hardwareMap.get(Gyroscope.class, "imu");
        leftMotor = hardwareMap.get(DcMotor.class, "leftmotor");
        rightMotor = hardwareMap.get(DcMotor.class, "rightmotor");
```

```

arm = hardwareMap.get(DcMotor.class, "arm");
claw = hardwareMap.get(Servo.class, "claw");
touch = hardwareMap.get(DigitalChannel.class, "touch");

rightMotor.setDirection(DcMotor.Direction.REVERSE);

telemetry.addData("Status", "Initialized");
telemetry.update();
// Wait for the game to start (driver presses PLAY)

waitForStart();
// run until the end of the match (driver presses STOP)

runtime.reset();
while (opModelsActive() && (runtime.seconds() <= 3.0)) {
    leftMotor.setPower(1);
    rightMotor.setPower(1);
}

```

Agora você tem o código básico necessário para fazer com que seu robô avance por três segundos. Isso deve proporcionar a você uma noção básica de codificação com ElapsedTime. Outras ações, como abrir e fechar uma garra ou levantar um braço, podem ser codificadas em seu programa autônomo.

Como aconselhado nas seções anteriores, é benéfico adicionar telemetria a determinado código para obter os dados de feedback que você deseja ou precisa. Para este exemplo, a telemetria mostrará quantos segundos se passaram para cada etapa da jornada do robô.

```

while (opModelsActive() && (runtime.seconds() <= 3.0)) {
    leftMotor.setPower(1);
    rightMotor.setPower(1);
    telemetry.addData("Leg 1", runtime.seconds());
    telemetry.update();
}

```

Para este guia específico, o objetivo final é testar a precisão de um robô avançando do ponto A para o ponto B e, em seguida, recuando de volta para o ponto A. Para fazer isso, é necessário escrever outra seção de código com base no temporizador. Uma maneira de fazer isso é copiar o loop while que você já criou e fazer as edições necessárias, como alternar a direção de energia para os motores.

```
runtime.reset();
while (opModelsActive() && (runtime.seconds() <= 3.0)) {
    leftMotor.setPower(1);
    rightMotor.setPower(1);
    telemetry.addData("Leg 1", runtime.seconds());
    telemetry.update();
}
```

```
runtime.reset();
while (opModelsActive() && (runtime.seconds() <= 3.0)) {
    leftMotor.setPower(-1);
    rightMotor.setPower(-1);
    telemetry.addData("Leg 2", runtime.seconds());
    telemetry.update();
}
```

Observe que um `runtime.reset()`; adicional foi adicionado ao código acima. A outra opção para um segundo `while` loop teria envolvido adicionar uma condição adicional ao `while` loop, como:

- `while(opModelsActive() && (runtime.seconds() > 3.0) && runtime.seconds() <=6.0)`

A escolha de redefinir o temporizador antes de iniciar uma nova etapa da jornada do robô foi feita para reduzir a quantidade de alterações de código que podem ser necessárias durante os testes do código.

Exemplo de código completo]

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.AnalogInput;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.robotcore.hardware.ColorSensor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.hardware.DigitalChannel;
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
```

```
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;
import com.qualcomm.robotcore.util.ElapsedTime;
```

```
@Autonomous
```

```
public class HelloWorld_ElapsedTime extends LinearOpMode {
    private DcMotor leftMotor;
    private DcMotor rightMotor;
    private DcMotor arm;
    private Servo claw;
    private DigitalChannel touch;
    private Gyroscope imu;
    private ElapsedTime runtime = new ElapsedTime();
```

```
@Override
```

```
public void runOpMode() {
    imu = hardwareMap.get(Gyroscope.class, "imu");
    leftMotor = hardwareMap.get(DcMotor.class, "leftmotor");
    rightMotor = hardwareMap.get(DcMotor.class, "rightmotor");
    arm = hardwareMap.get(DcMotor.class, "arm");
    claw = hardwareMap.get(Servo.class, "claw");
    touch = hardwareMap.get(DigitalChannel.class, "touch");
    leftMotor.setDirection(DcMotor.Direction.FORWARD); // Set to REVERSE if using AndyMark motors
    rightMotor.setDirection(DcMotor.Direction.REVERSE);

    telemetry.addData("Status", "Initialized");
    telemetry.update();
    // Wait for the game to start (driver presses PLAY)
    waitForStart();

    // run until the end of the match (driver presses STOP)

    runtime.reset();
    while (opModelsActive() && (runtime.seconds() <= 3.0)) {
        leftMotor.setPower(1);
        rightMotor.setPower(1);
        telemetry.addData("Leg 1", runtime.seconds());
```

```
telemetry.update();  
}  
  
runtime.reset();  
while (opModelsActive() && (runtime.seconds() <= 3.0)) {  
    leftMotor.setPower(-1);  
    rightMotor.setPower(-1);  
    telemetry.addData("Leg 2", runtime.seconds());  
    telemetry.update();  
}  
  
}  
}
```

Revisão #1

Criado 18 dezembro 2023 19:19:37 por Enzo

Atualizado 18 dezembro 2023 19:44:05 por Enzo