Contest 13: Sumobot

Start date: 19 October 2017
Due: 19 October 2017

The Sumobot Contest will be held on Thursday, the 19th of October, from 6:00 pm in SR1, COM1. The sequence of events are as follows:

Time	Event
18:15 - 18:30	Introduction of Contest Rules
18:30 - 19:30	Group Stage I
19:30 - 20:00	Break with catered dinner and Redbull
20:00 - 20:45	Group Stage II
20:45 - 21:00	Quarter-final
21:00 - 21:15	Semi-final
21:15 - 21:30	Final
21:30 - 21:45	Prize presentation

Rules and Regulations

Playing Field

The Dohyō will be a circle with diameter of 140 cm, with 7 concentric circles with different colors (magenta, blue, cyan, green, yellow, red and white) painted on it as illustrated by the image below. The Dohyō will be elevated about 20cm above the ground.

Robot Specifications

Contesting robots are required to follow these constraints:

- Only parts from one Lego EV3 set (with extension kit) will be allowed on the robots.
- The physical dimensions of the robot should not exceed 45cm x 45cm. There is no height limit and the robot can take any shape and size once the match begins.
- Robots should run autonomously once started. There should not be remote control of any kind in battle.

Match and Game

Each match will consist of a number of games, depending on the stage of the tournament. Each game starts with the 2 robots at one of the starting configurations. The game ends when one of the robots has been defeated. A robot deemed to be defeated when it, or any part thereof, touches the ground. The other robot is the winner of the game. If both robots touch the ground in quick succession, the referee decides which one was first or calls a tie. Each game also has a **time limit of 60 seconds**. If both robots are still on the dohyo after this time limit, the game is a tie.

Starting orientation of robots:

We will rotate among the following 3 different configurations, in that order:

- The robots are facing each other head-on $\rightarrow\leftarrow$
- The robots are parallel and facing the same direction $\uparrow\uparrow$
- The robots are facing away from each other $\leftarrow\rightarrow$

Tournament format

Group Stage I : The 3-Way fighting

The 30 teams will be divided into 10 groups each consisting of 3 teams. They will compete (within each group) in a 3-way fight manner (two matches per group) and the winner will proceed to next stage.

For each 3-way match, the 3 contesting teams will play until only one team left. And get 1, 2, 3 points according to the order of being knocked out. The winner will proceed to next stage.

Group Stage II : Repechage

For the 10 teams in this round, 2 teams will be randomly selected and give a direct pass. The rest 8 teams will be randomly divided into four groups and compete against each other and the winner will proceed to the Semi-final Stage.

Apart from the 6 teams, We will have a secret Respechage and two more teams can join the 6 teams and enter the Quarter-final Stage!
(The details about the respechage will be anounced during the contest)

Quarter-final and Semi-final Stage

The 8 teams will enter the Quarter-final Stage.
We will have 2 groups, each consisting of 2 teams. They will compete (within each group)

against each other and the winner will proceed to the Semi-final Stage.

For each Quarterfinals and Semifinals match, the 2 contesting teams will play until one team wins twice. If no team has won twice after 5 games, we will use the tiebreaker to determine the team to advance to the next round.

Final Stage

The final match will be a best of 5, i.e. the first team to win 3 rounds will win the tournament. If no team has won after 10 games, both will be declared champions.

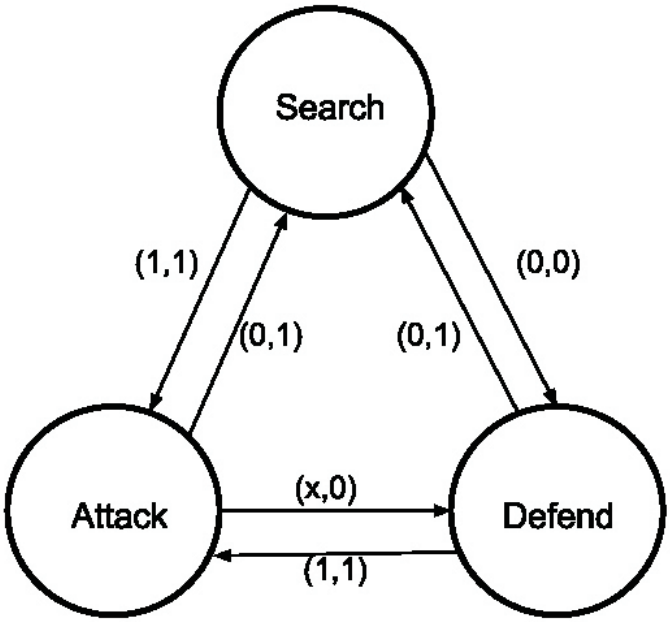
Tiebreaker: The Pillar Chase

Tied robots will be placed around a green pillar of A4 size in the foyer of SR1. Their initial position is 10cm away from the pillar, pointing in counter-clockwise direction. The winner is the robot that manages to touch its opponent by going counter-clockwise around the pillar, or by lapping its opponent by going counter-clockwise. A team laps its opponent when any part of it crosses the imaginary line between the centre of the pillar and any part of the opponent.

Appendix A: Designing a Program Using State Diagrams

This section will introduce to you how state diagrams can be used to design programs for the Sumobot contest.

Your robot will need at least 3 basic capabilities: searching, attacking, and defending. We will call each of these capabilities a state. Your robot should be able to switch from one state to another based on the situation. For example, in the beginning your robot may try to search for another robot (the "searching" state), but when a target is detected, your robot should transition from the "searching" state to the "attacking" state. If we try to enumerate all possible transitions among these three states, we will get a state diagram, as shown below.



It would be a good idea to implement transition conditions based on the design of your robot. You should define the precise meaning of "safe position" and "target detected" for your robot. Because your robot can only know its environment through its different sensors, its state and transitions should be contingent on the information these sensors give it.

Different states can have different behaviours. Here are some simple examples. You can define any behaviour you want.

- Search: the robot will repeatedly spin round in circles until an object is detected within 20cm from its front
- Attack: rush towards the target at full speed
- Defend: turn around and move forward for a distance of 30cm (may apply if the robot is at the edge of the Dohyō)

Appendix B: Documentations for the EV3 robot functionalities

Remember to put these three lines at the top of your program

```
#!/usr/bin/env node
var ev3 = require('./node_modules/ev3source/ev3.js');
```

```
var source = require('./node_modules/ev3source/source.js');
```

TIPS: The robot takes quite some time to compile your program into machine instructions, in order to execute it. During this time, other robots could have started their program and push you off the ring! One way is to mitigate this is to put this at the start of your program:

```
ev3.waitForButtonPress();
```

You can then start your program beforehand. Once the referee announces the start of the match, press any button to get your robot into action! In this way, your robot does not need to waste precious time compiling the program when the match has already started!

The list and miscellaneous functions available in Source Academy (Week 8) are available for use. However, you will need to put the keyword `source.` in front of the functions that you are normally used to.

```
// To create a pair(3, 4)
source.pair(3, 4).

// To obtain the head of a pair
source.head(source.pair(9, 10)).
```

Here is the list of all the functions available, you may use it as reference when you are working on this mission:

Motors:

- `ev3.motorA()`: returns the motor connected to the port A.
- `ev3.motorB()`: returns the motor connected to the port B.
- `ev3.motorC()`: returns the motor connected to the port C.
- `ev3.motorD()`: returns the motor connected to the port D.
- `ev3.runForDistance(motor, distance, speed)`: causes the motor (which you get from the previous functions) to rotate at the specified speed for a specific number of rotations; if you pass a negative distance, the motor will rotate backward.
- `ev3.runForTime(motor, time, speed)`: causes the motor (which you get from the previous functions) to rotate at the specified speed for a specific duration.
- `ev3.motorGetPosition(motor)`: gets the current position of the motor in pulses of the rotary encoder. When the motor rotates clockwise, the position will increase. Likewise, rotating counter-clockwise causes the position to decrease. This function, together with the next 2 functions, are useful for precise movement.
- `ev3.runToAbsolutePosition(motor, position, speed)`: Runs the motor to an absolute position
- `ev3.runToRelativePosition(motor, position, speed)`: Runs the motor to a position relative to the current position value, i.e. `current_position + position`
- `ev3.motorGetSpeed(motor)`: gets the current motor speed in tacho counts per second
- `ev3.motorSetSpeed(motor, speed)`: sets the current motor speed in tacho counts per second; if you pass a negative speed, the motor will rotate backward.
- `ev3.motorStart(motor)`: causes the motor (which you get from the previous functions) to rotate at the speed specified earlier by `ev3.motor_set_speed` until it is stopped.

- `ev3.motorStop(motor)`: causes the motor (which you get from the previous functions) to stop rotating.
- `ev3.motorSetStopAction(motor, stopAction)`: specifies the way the motor will be stopped. Possible values of `stopAction` are `"coast"`, `"brake"`, and `"hold"`. `"coast"` means that power will be removed from the motor and it will freely coast to a stop. `"brake"` means that power will be removed from the motor and a passive electrical load will be placed on the motor. This is usually done by shorting the motor terminals together. This load will absorb the energy from the rotation of the motors and cause the motor to stop more quickly than coasting. `"hold"` does not remove power from the motor. Instead it actively tries to hold the motor at the current position. If an external force tries to turn the motor, the motor will â€~push backâ€™ to maintain its position (maybe good for defense). Note that this function will not make the motor stop. Use `ev3.stop` for that.

Color sensor:

- `ev3.colorSensor()`: returns the connected color sensor.
- `ev3.colorSensorRed(colorSensor)`: returns the red value read from the `colorSensor`, it's a number within the range of 0 to 1020.
- `ev3.colorSensorGreen(colorSensor)`: returns the green value read from the `colorSensor`, it's a number within the range of 0 to 1020.
- `ev3.colorSensorBlue(colorSensor)`: returns the blue value read from the `colorSensor`, it's a number within the range of 0 to 1020.
- `ev3.reflectedLightIntensity(colorSensor)`: return the percentage of the reflected light intensity.

Touch Sensor

- `ev3.touchSensor1()`: returns the touch sensor connected to the port 1.
- `ev3.touchSensor2()`: returns the touch sensor connected to the port 2.
- `ev3.touchSensor3()`: returns the touch sensor connected to the port 3.
- `ev3.touchSensor4()`: returns the touch sensor connected to the port 4.
- `ev3.touchSensorPressed(touchSensor)`: returns whether the touch sensor is pressed.

Ultrasonic Sensor

- `ev3.ultrasonicSensor()`: returns the connected ultrasonic sensor.
- `ev3.ultrasonicSensorDistance(ultrasonicSensor)`: returns the distance between the sensor and an object in centimeters (cm).

Gyro Sensor

- `ev3.gyroSensor()`: returns the connected gyro sensor.
- `ev3.gyroSensorRateMode(gyroSensor)`: switches the gyro sensor to rate mode.
- `ev3.gyroSensorRate(gyroSensor)`: Returns the value of the gyro sensor.
- `ev3.gyroSensorAngleMode(gyroSensor)`: switches the gyro sensor to angle mode.
- `ev3.gyroSensorAngle(gyroSensor)`: Returns the number of degrees that the sensor has been rotated since it was put into this mode

Sound

- `ev3.playSequence(beeps)`: play the sequence of beeps. `beeps` is a list of beeps. A beep is a `list(frequency, length, delay)` which specifies the frequency, length of the beep and the delay after the beep (times are measured in miliseconds).
- `ev3.speak(script)`: speak the string `script`

Buttons

- `ev3.waitForButtonPress()`: pauses execution until a button is pressed
- `ev3.buttonEnterPressed()`: returns whether the last button pressed is Enter
- `ev3.buttonUpPressed()`: returns whether the last button pressed is Up
- `ev3.buttonDownPressed()`: returns whether the last button pressed is Down
- `ev3.buttonLeftPressed()`: returns whether the last button pressed is Left
- `ev3.buttonRightPressed()`: returns whether the last button pressed is Right

Miscellaneous

- `ev3.pause(time)`: Pauses the program for *at least* the specified amount of time in milliseconds.
- `ev3.runUntil(terminatingCondition, task)`: repeatedly executes `task()` until `terminatingCondition()` is satisfied, note that both of the two arguments are **functions**.
- `ev3.runForever(task)`: repeatedly executes `task()` forever.

Part of the fun is learning how to troubleshoot. If you have difficulties, start by googling your problems. `source.alert()` is a useful function to output to the screen.