



# RoboCup Junior Australia

MAZE SPRINT RESCUE FOR  
EV3 AND SPIKE PRIME



# Overview of Competition



“Real world” robot challenge



Modular, flexible game design



Accessible to a wide range of robot platforms



Many possible solutions



Easy for beginners but with a high ceiling



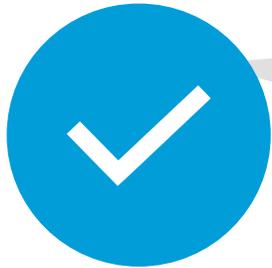
Simple scoring



Engaging and fun.



# Now even easier for beginners!



CAN BE COMPETITIVE  
WITH AN EV3



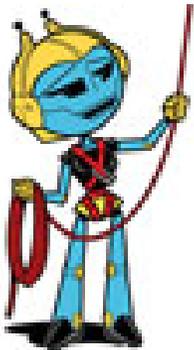
CAN USE THE EV3  
CLASSROOM SCATCH  
PROGRAMMING



PATHWAY TO MORE  
ADVANCED  
SOLUTIONS



LOTS OF DESIGN  
FLEXIBILITY



# Scenario

- There has been a disaster in a factory building.
- Several workers are trapped inside but it is not safe to send in rescue teams. Tokens have been placed to represent trapped (green) and seriously injured (red) victims
- The role of your sprint robot is to enter the building, identify the number and classification of victims, and exit the building as quickly as possible.
- Your robot will then report on the victims found.



# The Competition



You have 2 minutes to calibrate your robot, find the victims and exit the maze.

There is a minimum of 5 victims in the maze. You get points for each one you find. The red victims are worth more points than the red

Bonus points for exiting the maze before the time has elapsed and for accurately reporting the type of victims

The robot must avoid black holes

The robot can be return to last found victim if stuck without penalty but program cannot be restarted

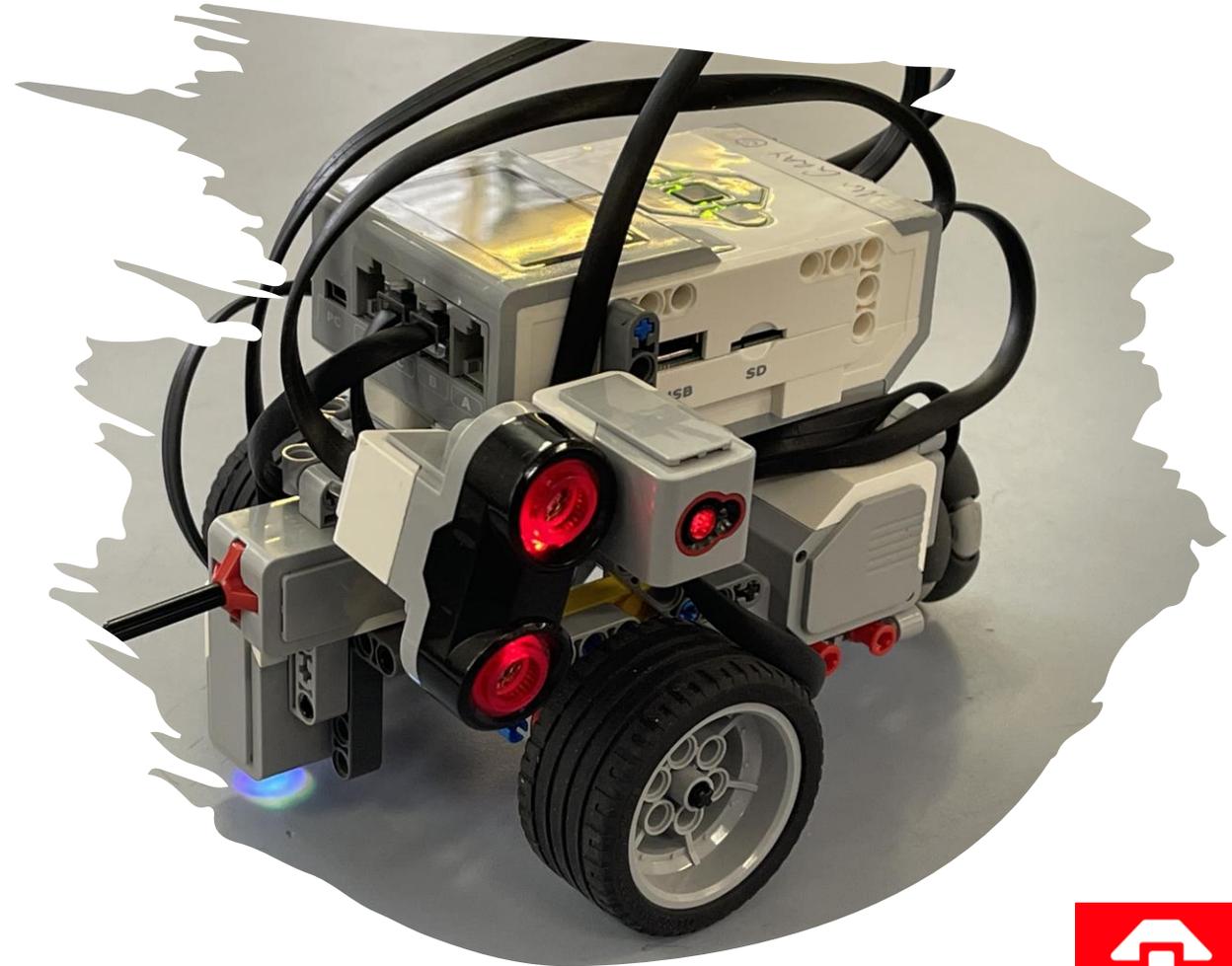
Robot run can be restarted with loss of points at any time

Two divisions

- Standard for EV3 and Spike - with or without a robot!
- Virtual for any robot design running through set mazes.

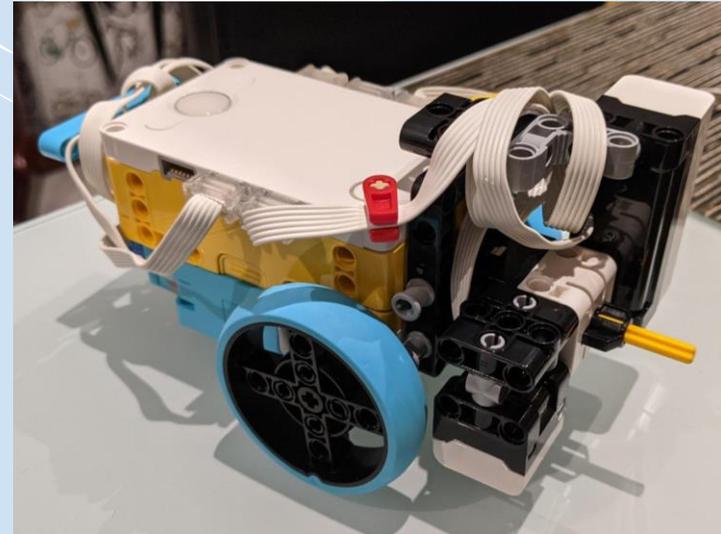
# *The Robot – what you need*

- Basic EV3 robot
- 1 X Colour sensors (pointing down). You can just start with the one facing the wall first
- 1 X Ultrasonic (facing wall)
- 1 X Touch sensor on front of



# *The Robot – what you need*

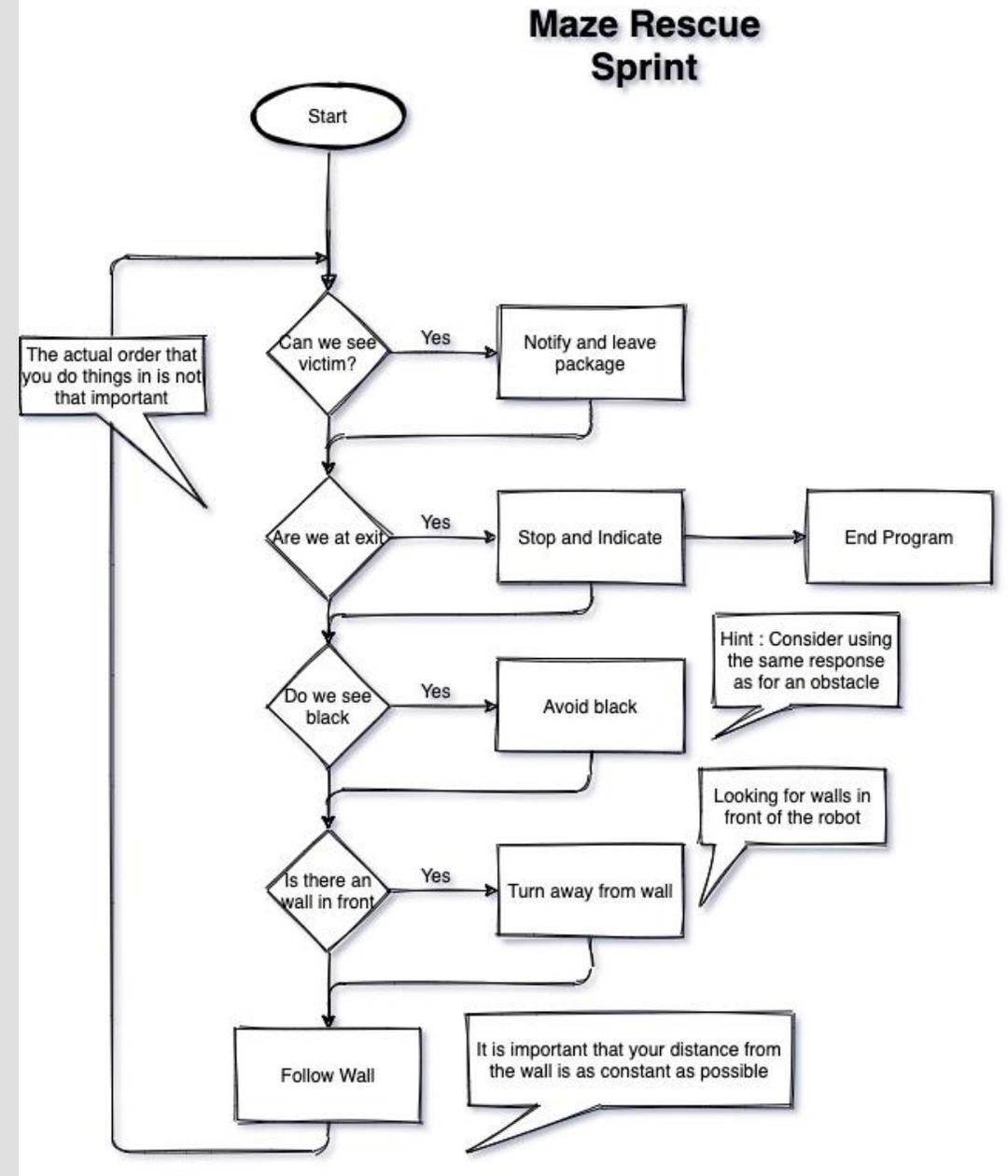
- Basic Spike Prime robot
- 1 X Colour sensors (pointing down)
- 1 X Ultrasonic (facing side)
- 1 X Touch sensor on front of robot



# Maze Algorithm

- Keep it simple and work on one element at a time.
- Make your wall follower as smooth as possible. You want to stay at a constant distance from the wall.
- The rate at which the robot turns is important for reliable performance
- A good development strategy is
  - ✓ Follow a wall
  - ✓ Detect an obstacle in front
  - ✓ Detect a black hole
  - ✓ Find a victim
  - ✓ Find exit and report

We “nest” a series of if ... else statements one at a time to get the robot behavior that we are after.



# Simple Wall Following

- Wall following is simple. The robot either steers towards the wall if too far away or away from the wall if too close.
- The key issue is carefully calibrating your robot turn rates. The robot should follow a constant arc around a wall divider.
- Individual motor speed control works better than steering

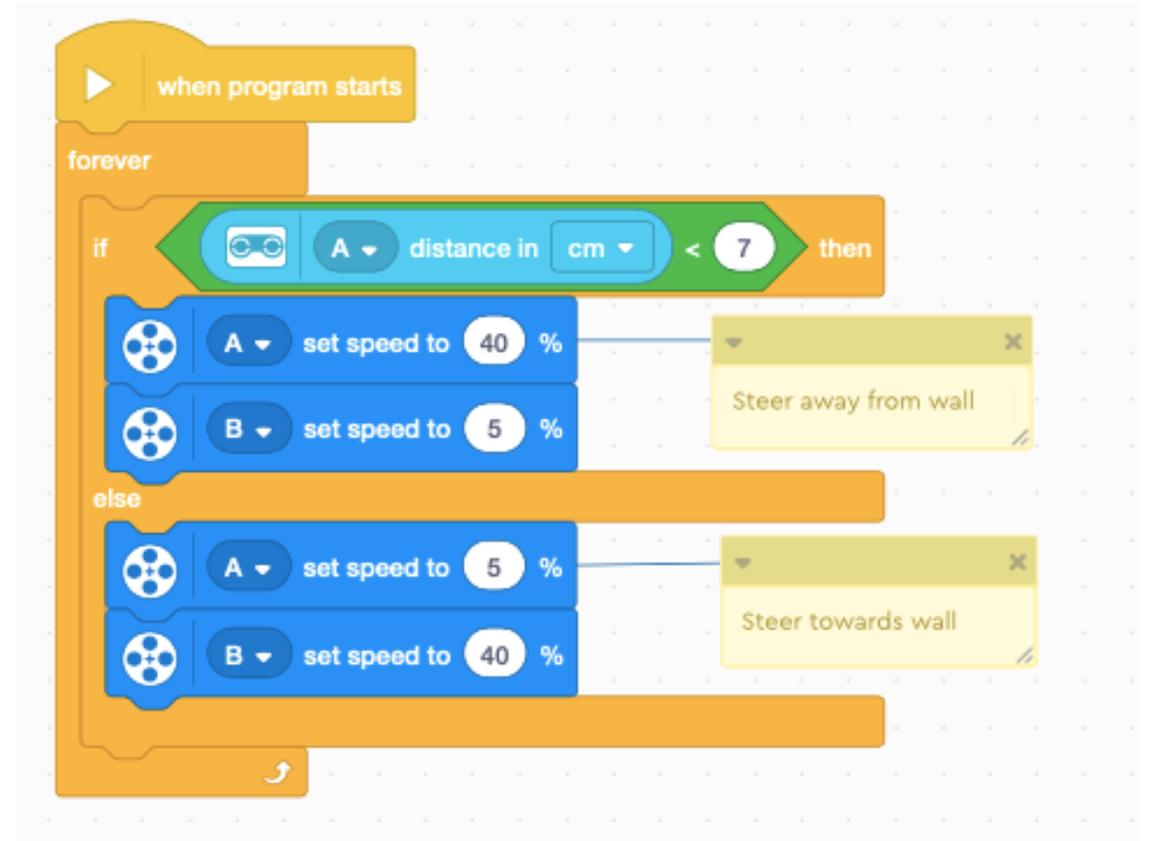
```
graph TD
    Start[when program starts] --> Forever[forever]
    Forever --> If{if distance in cm < 6}
    If -- then --> B15[start motor B at 15% speed]
    If -- then --> C35[start motor C at 35% speed]
    If -- else --> B35[start motor B at 35% speed]
    If -- else --> C15[start motor C at 15% speed]
    B15 --> Forever
    C35 --> Forever
    B35 --> Forever
    C15 --> Forever
```

Simple wall following  
Turn towards the wall if too far away, away from the wall if too close

Tune up the speeds so that the robot turns nicely around corners

# Simple Wall Following

- Wall following is simple. The robot either steers towards the wall if too far away or away from the wall if too close.
- The key issue is carefully calibrating your robot turn rates. The robot should follow a constant arc around a wall divider.
- Individual motor speed control works better than steering



ON/OFF



PROPORTIONAL

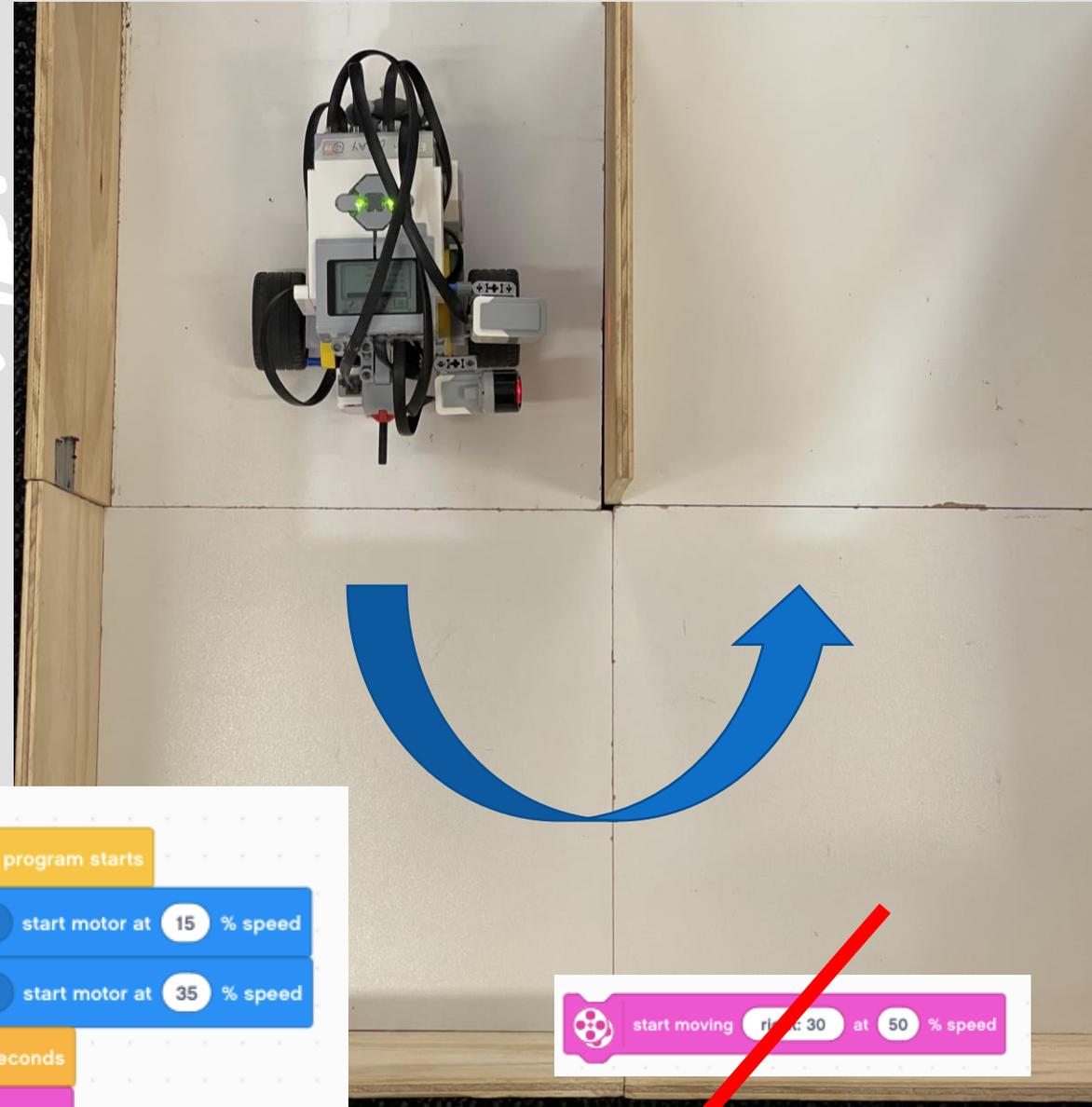
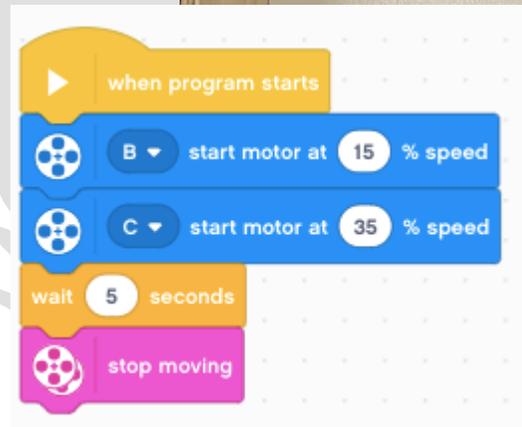


FAST  
PROPORTIONAL



# Setting the Turning radius

- When your robot comes to the end of a wall it will try to turn towards where it thinks the wall is.
- Setup your motor speeds so that your robot will do a smooth arc around the wall.
- There is a relationship between speed and turning plus every robot has different characteristics. Play around until you get it right.
- Individual motor speed works much better than using “steering”.



# Obstacle Avoidance

- When you see an obstacle in front of the robot you need to react in some way.
- This can be a wall or an obstruction. Even a black hole can be treated as an obstacle as seen in the example on the right.
- If you see a wall in front of you for example you could reverse a little and then turn away from the wall.
- Keep the movements small – it is not unusual to take a few attempts at getting around the obstacle.
- With this simple program you have a robot that can solve a basic maze!

The image shows a Scratch script for obstacle avoidance. It starts with a 'when program starts' block, followed by a 'forever' loop. Inside the loop, there are two 'if' blocks. The first 'if' block checks if button 2 is pressed or if sensor 4 is black. If true, it moves the robot backward by 0.25 rotations and then turns left by 100 degrees at 25% speed. The second 'if' block checks if the distance to sensor 1 is less than 6 cm. If true, it starts motor B at 15% speed and motor C at 35% speed. If false, it starts motor B at 35% speed and motor C at 15% speed. Three callout boxes provide additional context: the first asks 'Have we bumped into anything? Put some code in here to get your robot to execute a turn'; the second explains 'Simple wall following: Turn towards the wall if too far away, away from the wall if too close'; the third notes 'Tune up the speeds so that vthe robot turns nicely around corners'.

# Obstacle Avoidance

- When you see an obstacle in front of the robot you need to react in some way.
- This can be a wall or an obstruction. Even a black hole can be treated as an obstacle as seen in the example on the right.
- If you see a wall in front of you for example you could reverse a little and then turn away from the wall.
- Keep the movements small – it is not unusual to take a few attempts at getting around the obstacle.
- With this simple program you have a robot that can solve a basic maze!

```
when program starts
  forever loop
    if (C is pressed or E is colour black) then
      move down for 2 cm
      move left for 1 seconds
      Reverse and little spin
    else
      if (A distance in cm < 7) then
        A set speed to 40 %
        B set speed to 5 %
        Steer away from wall
      else
        A set speed to 5 %
        B set speed to 40 %
        Steer towards wall
```

The code is a Scratch-style block-based program for obstacle avoidance. It starts with a 'when program starts' block, followed by a 'forever' loop. Inside the loop, there is an 'if' block with two conditions: 'C is pressed?' or 'E is colour black?'. If either condition is true, the robot moves down 2 cm, then moves left for 1 second, and then performs a 'Reverse and little spin' action. If the conditions are false, there is an 'else' block containing another 'if' block: 'A distance in cm < 7'. If this is true, motor A is set to 40% speed and motor B to 5% speed, with the instruction 'Steer away from wall'. If false, motor A is set to 5% speed and motor B to 40% speed, with the instruction 'Steer towards wall'. Three yellow callout boxes provide context: 'Looking for wall in front or black tile' points to the first 'if' block, 'Reverse and little spin' points to the corresponding action block, and 'Steer away from wall' points to the second 'if' block's 'then' branch.

# Finding the Victim

- The victims are represented by 50mm square markers in the middle of each passage through the maze. They are coloured red or green to indicate if they are in critical condition or just need to be located.
- This means we can use the Colour Sensor to identify the victim.
- When you find a victim you need to stop for 1 second and indicate in some clear way.
- It may be a good idea to move off the victim before continuing – why?
- A constant wall following distance is critical

The image shows a Scratch script for a robot's behavior. The script is contained within an 'if' block with a condition: '2 is colour green?' or '2 is colour red?'. The 'then' branch of this 'if' block contains the following actions: 'clear display', 'stop moving', 'write VICTIM at 1, 5 with font large black', and 'play sound Expressions / Fanfare until done'. A 'wait 1 seconds' block follows. After the wait, there are two nested 'if' blocks. The first 'if' block has the condition '2 is colour green?' and the 'then' branch contains 'change GREEN by 1'. The second 'if' block has the condition '2 is colour red?' and the 'then' branch contains 'change RED by 1'. Below these, there is a 'move forward for .25 rotations' block. The 'else' branch of the main 'if' block is currently empty. Several yellow callout boxes are present: 'Seen a victim' points to the main 'if' block; 'Notifying' points to the 'write' block; 'Incrementing count' points to the 'change GREEN by 1' block; and 'Moving away from victim' points to the 'move forward' block.

# Finding the Victim

- The victims are represented by 50mm square markers in the middle of each passage through the maze. They are coloured red or green to indicate if they are in critical condition or just need to be located.
- This means we can use the Colour Sensor to identify the victim.
- When you find a victim you need to stop for 1 second and indicate in some clear way.
- A constant wall following distance is critical

```
when program starts
  set redV to 0
  set greenV to 0
  forever
    if E is colour red or E is colour green then
      stop moving
      wait 1 seconds
      play sound Alert until done
      if E is colour red then
        change redV by 1
      else
        change greenV by 1
      move up for 1 cm
    else
      if C is pressed or E is colour black then
        move down for 2 cm
        move turn left for 1 seconds
      else
```

# *Finding the Exit tile*

The exit tile is shiny and highly reflective

See if you can work out some way to sense it

Hint ..... put in another nested if ... then based on reflected light to notify your success and turn the robot off.

Now it is time to display how many and what type of victims you have found.



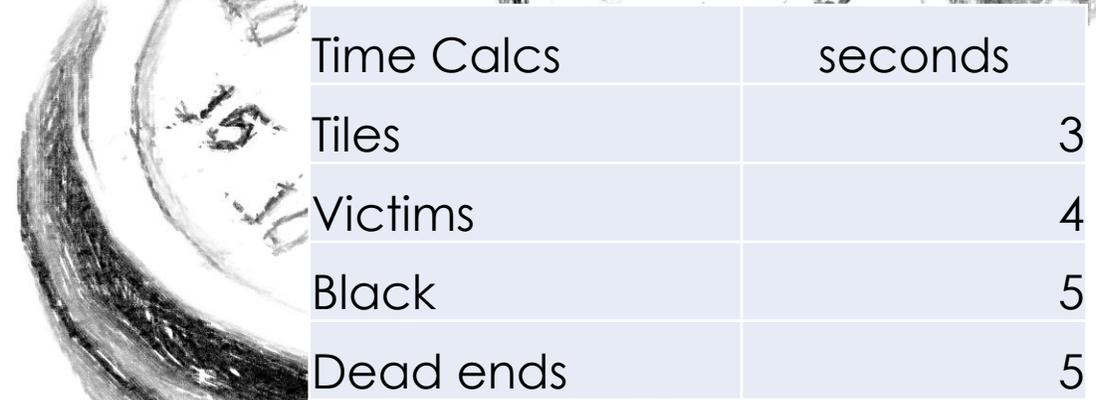
# SCORING

ACTION	POINTS	COMMENTS
Found green victim	10	Robot must stop and clearly signal
Found red victim	20	Robot must stop and clearly signal
Time Bonus	2 points for each second less than Target Time	Robot must stop on exit within the Total Time (Target Time plus 60 seconds)
Reporting Bonus green victims	25 Points	Robot must stop on exit and display the correct number of green victims
Reporting Bonus red victims	50 points	Robot must stop on exit and display the correct number of red victims

The champion teams is decided by the cumulative total over several rounds

# Target Time

- The Target Time is calculated by an algorithm that considers the following
  - Number of 30cm X 30cm “tiles” X time calc
  - Number of victims X time calc
  - Number of Black tiles X time calc
  - Number of Dead Ends on shortest route X time calc
  - A Challenge Factor (0.8-1.2) for each round.
  - Add up the times and divide by the Challenge Factor
  - 0.8 or early rounds rising ads we proceed.



Time Calcs	seconds
Tiles	3
Victims	4
Black	5
Dead ends	5

For a maze with 20 tiles, 5 victims, 2 Black tiles and 3 Dead ends would be calculated as 110 seconds. This is then divided by the Challenge Factor to give the Target Time

# A better Wall Follower

- As already noted an accurate wall follower is essential to increasing the reliability of your victim detection.
- The best way to do this is to use a proportional control algorithm.
- Seems tricky at first but pretty easy once you get the hang of it.
- Essentially you speed up the wall side motor and reduce the speed to the other to move away from the wall.
- You do the opposite if you get too far away.
- **Hints : Make sure you put in something to limit the rate at which the robot turns**
- **Play with the Kp and basespeed until you get the response that you want.**

The image shows a Scratch script for a wall follower robot. The script starts with a 'when program starts' block, followed by four 'set' blocks: 'set setpoint to 6', 'set Kp to 5', 'set maxturn to 10', and 'set basespeed to 25'. A 'forever' loop contains the following blocks: 'set error to 1 \* distance in cm - setpoint', 'set steer to error \* Kp', an 'if' block 'if steer > maxturn then' with 'set steer to maxturn' inside, another 'if' block 'if steer < -1 \* maxturn then' with 'set steer to -1 \* maxturn' inside, and two 'start motor at' blocks: 'B start motor at basespeed + steer % speed' and 'C start motor at basespeed - steer % speed'. There are four callout boxes: 1. 'Calculating the error' pointing to the error calculation block. 2. 'Calculating the correction' pointing to the 'set steer' block. 3. 'ESSENTIAL - Limiting the rate of turn in either direction' pointing to the two 'if' blocks. 4. 'Setting the motor speeds. We have found that individual motor control works better than Steering.' pointing to the two 'start motor at' blocks. A larger callout box at the top right explains the parameters: 'For this example # We are trying to stay 6cm away from th wall (setpoint) # Our reponse rate (Kp) is set to 5 - adjust until you get a smooth response # Our maximum turn rate is set to 10. When applied top our motor speeds this limits us to 35 on motor B and 15 on motor A # basespeed is how fast our ropbot will travel at exactly 6 cm from wall.'

# Important Links

Everything you need will be accessible through the RoboCup junior Australia web site.

I can be contacted at [neil.gray@robocupjunior.org.au](mailto:neil.gray@robocupjunior.org.au)

- <https://www.robocupjunior.org.au/>
- <https://www.robocupjunior.org.au/rescue-maze/>
- <https://www.robocupjunior.org.au/wp-content/uploads/2021/02/Official-2021-RCJA-Rescue-Maze-Rules-NGTC.pdf>
- <https://www.robocupjunior.org.au/wp-content/uploads/2021/02/Maze-Full-Field-Cutting-Layout.pdf>