Rescue Simulation

RRGGBB

A POSTERIORI

Play · Experience · Learn

RRGGBB

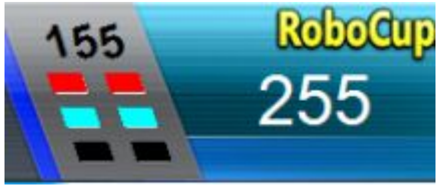
A POSTERIORI
Play · Experience · Learn

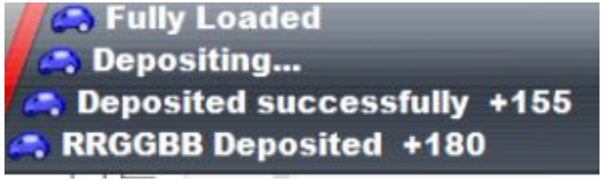
Strategy

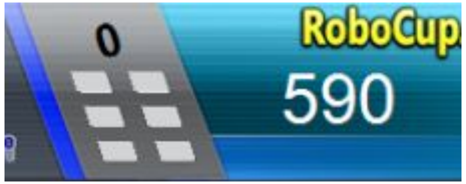
You get more points when you deposit sets of Red, Cyan, and Black objects.

A single set is referred to as RGB and worth 90 points, while 2 sets are called RRGGBB and worth 180 points.

So, **only pickup and deposit RRGGBB sets**.




Fully Loaded
Depositing...
Deposited successfully +155
RRGGBB Deposited +180



State Machines (Recap)

A POSTERIORI
Play · Experience · Learn

We can think of Strategies as Algorithms.
And we can define algorithms in terms of **State Machines**.

Machine **States** are based on the value of one or more variables, and are used to determine the current tactic.

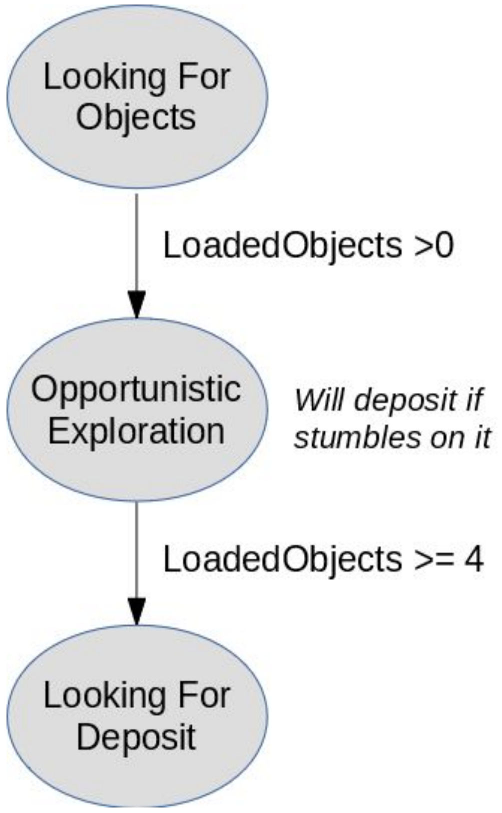
In the case of “Wall-Follow to Deposit” strategy, the algorithm had **2 states** based on LoadedObjects variable:

State	**Behavior / Tactic**
LoadedObjects < 4	Search for Objects
LoadedObjects >= 4	Follow Walls to Collection Box

State Machines - Table or Graph

One way to represent a State Machine is by Table.
Another way is through a Graph.

State	Behavior / Tactic
LoadedObjects == 0	• Search for Objects, • Ignore Collection Box (and Traps)
LoadedObjects < 4	• Search for Objects • Deposit, if you see the Collection Box
LoadedObjects >= 4	• Find & Follow Wall to Collection Box • Pickup Object, if you happen to see one



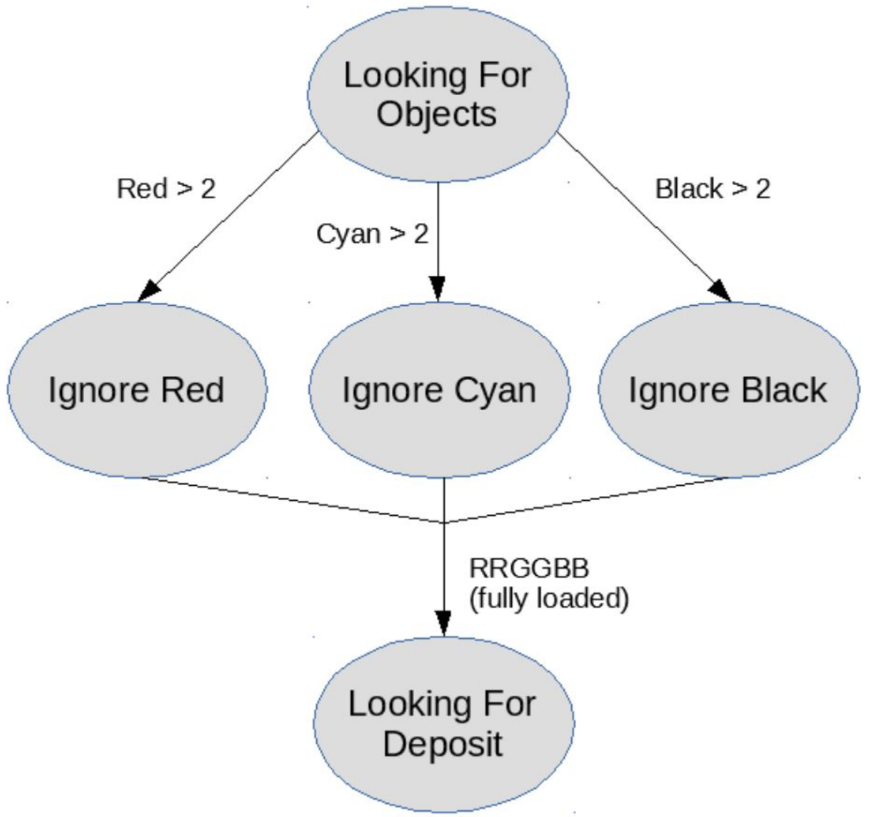
RRGGBB - State/Transition Graph

Algorithm
Always collect exactly 2 Red, 2 Cyan, and 2 Black Objects before making any deposit.

For this algorithm, we need to keep track of:

- **how many objects we pickup for each color**

Not just *LoadedObjects*.



RRGGBB - State/Condition/Action Table

A POSTERIORI
Play · Experience · Learn

State	Condition	Action
Red < 2	CS sees Red	Pickup
Red == 2	CS sees Red	Ignore
Cyan < 2	CS sees Cyan	Pickup
Cyan == 2	CS sees Cyan	Ignore
Black < 2	CS sees Black	Pickup
Black == 2	CS sees Black	Ignore
Red < 2 || Cyan < 2 || Black < 2	CS sees deposit	Ignore
Red == 2 && Cyan == 2 && Black == 2	CS sees deposit	Bank

Creating Variables

As mentioned, for this algorithm, we need to keep track of:

- **how many objects we pickup for each color**

Not just *LoadedObjects*.

Therefore, we need to create new Variables:

- LoadedRed
- LoadedCyan
- LoadedBlack

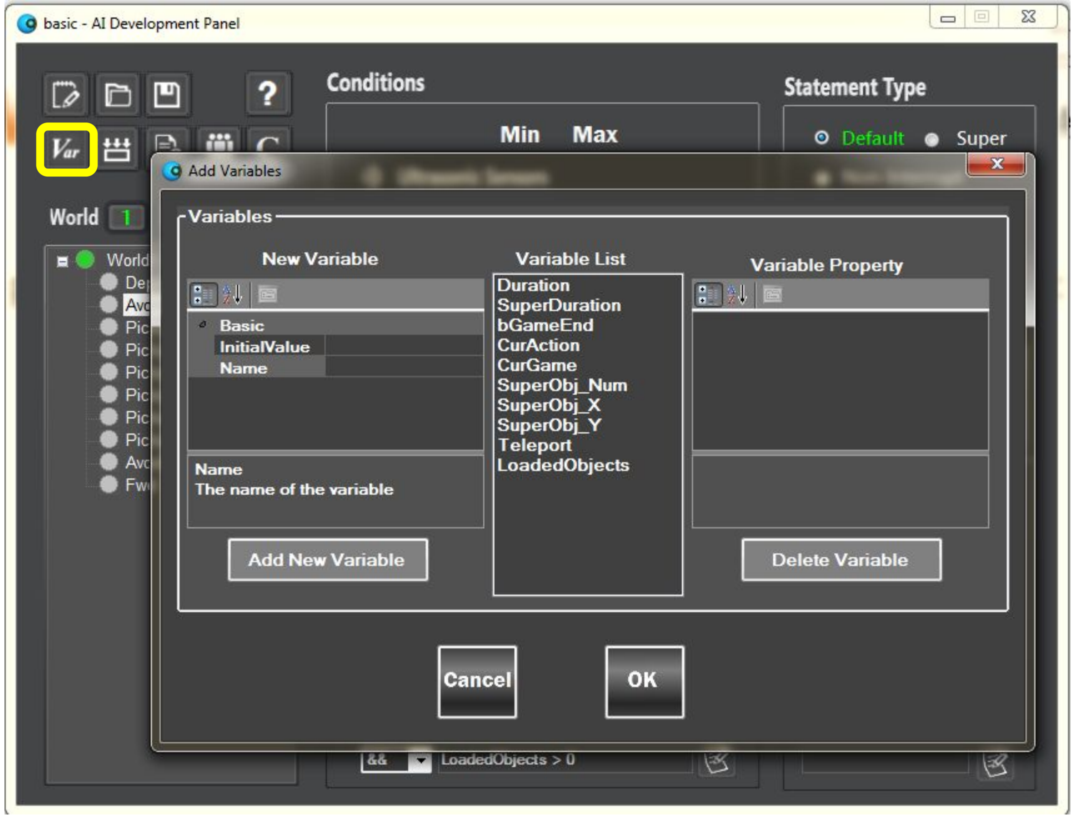
Creating Variables

Under the AI panel, click the **Var** button, to open the:

Add Variables window

You can look at existing variables - we have only used *LoadedObjects* so far, and printed *Duration* for reference.

Variables like SuperObj* will be discussed in other modules.



Variable List:

- Duration
- SuperDuration
- bGameEnd
- CurAction
- CurGame
- SuperObj_Num
- SuperObj_X
- SuperObj_Y
- Teleport
- LoadedObjects

Creating Variables

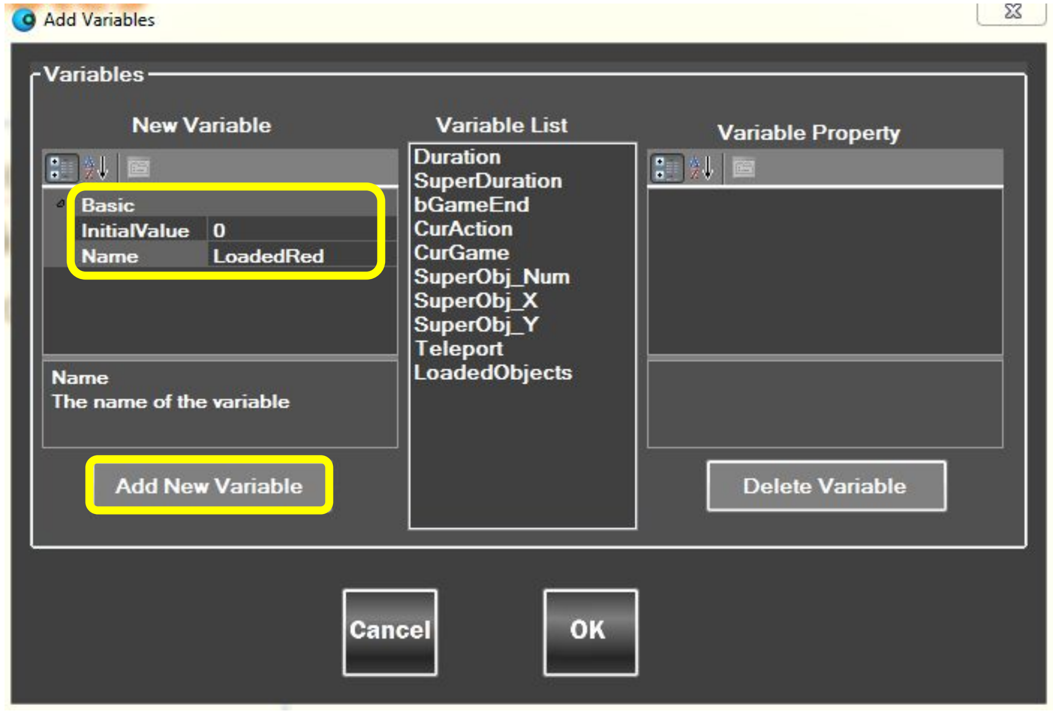
A POSTERIORI
Play · Experience · Learn

Let's create LoadedRed - a new variable to keep track of our # of Red Objects loaded.

Change the Name to our new variable name.

InitialValue should be zero(0), because we start with no Red objects loaded.

Then, click *Add New Variable*.



Add Variables

Variables

New Variable

Basic	
InitialValue	0
Name	LoadedRed

Name
The name of the variable

Variable List

Duration
SuperDuration
bGameEnd
CurAction
CurGame
SuperObj_Num
SuperObj_X
SuperObj_Y
Teleport
LoadedObjects

Variable Property

Creating Variables

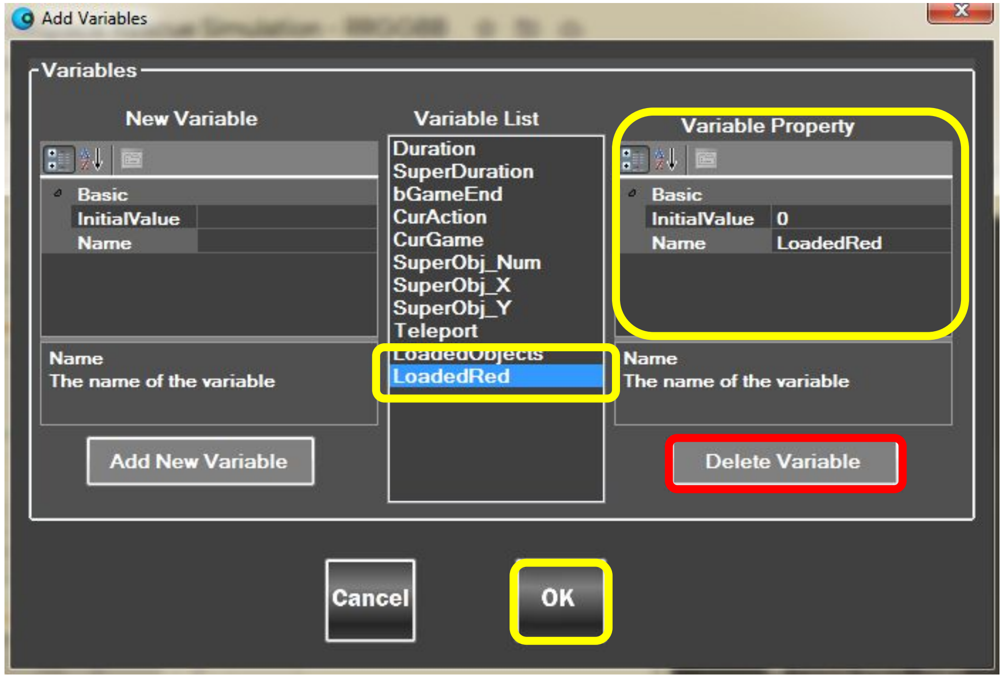
A POSTERIORI
Play · Experience · Learn

Now your new variable, *LoadedRed*, is added to the *Variable List*.

If you **click on it**, you can see its InitialValue setting.

You can also Delete it, if necessary.

Press OK to persist this change to the Variable List.



Creating Variables

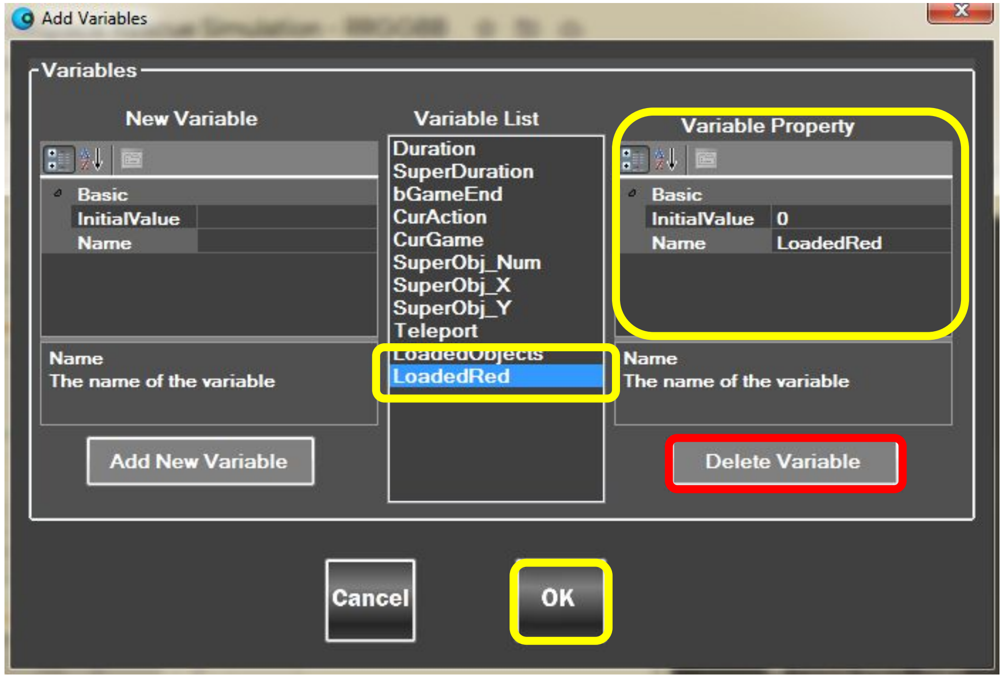
A POSTERIORI
Play · Experience · Learn

Now your new variable, *LoadedRed*, is added to the *Variable List*.

If you **click on it**, you can see its InitialValue setting.

You can also Delete it, if necessary.

Press OK to persist this change to the Variable List.



New Variable:

Basic	
InitialValue	
Name	

Name
The name of the variable

Variable List:

Duration
SuperDuration
bGameEnd
CurAction
CurGame
SuperObj_Num
SuperObj_X
SuperObj_Y
Teleport
LoadedObjects
LoadedRed

Variable Property:

Basic	
InitialValue	0
Name	LoadedRed

Name
The name of the variable

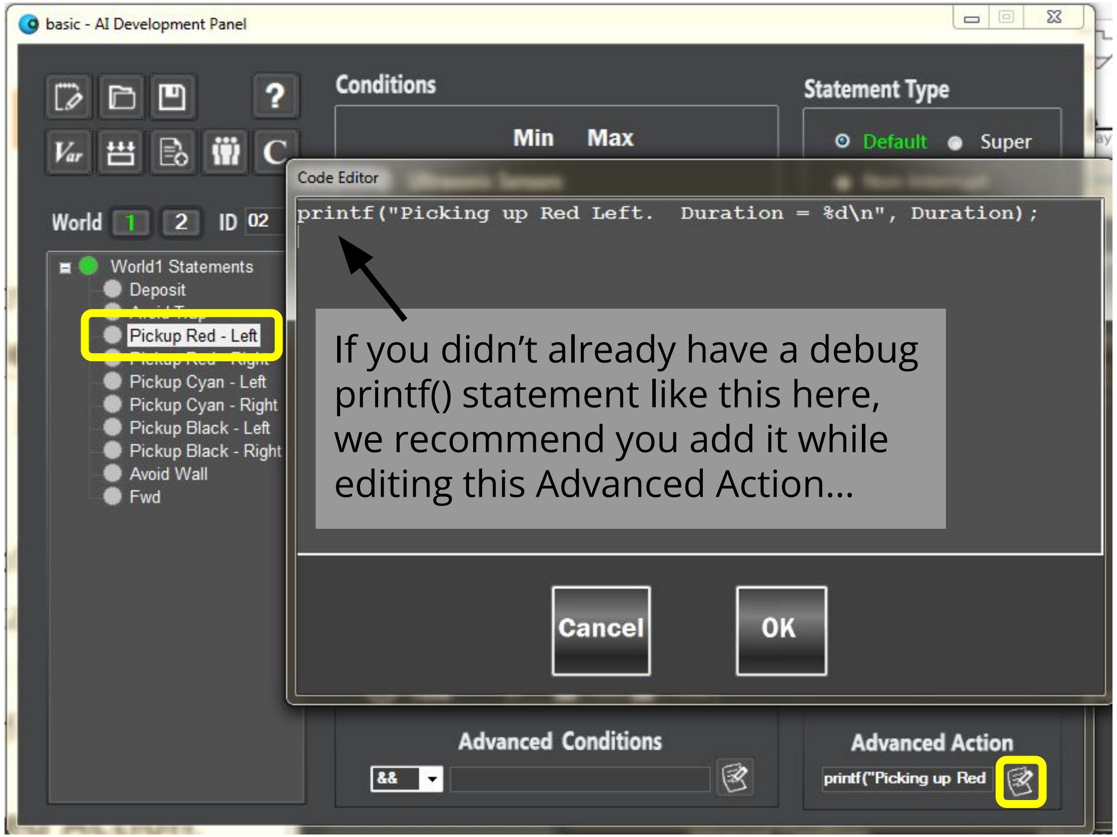
Updating New Variables

A POSTERIORI
Play · Experience · Learn

Now we can try to **update LoadedRed**, whenever we pickup a Red Object.

Choose one of your Pickup Red statements (there should be one for Left and one for Right).

Click **Advanced Action**.



```
printf("Picking up Red Left.  Duration = %d\n", Duration);
```

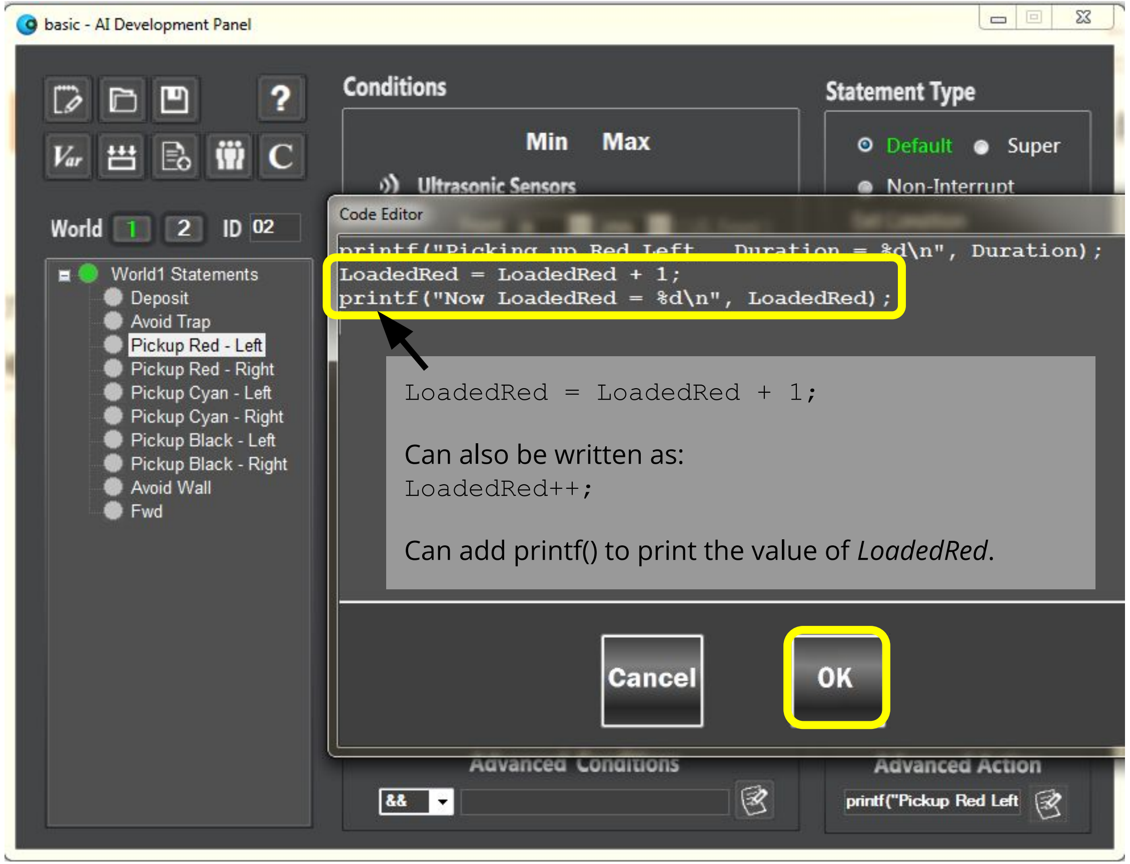
If you didn’t already have a debug printf() statement like this here, we recommend you add it while editing this Advanced Action…

Updating New Variables

Add the appropriate line of code to add 1 to *LoadedRed's* value.

There are two common ways to do that, as shown.

Click OK to save the change.



```
printf("Picking up Red Left    Duration = %d\n", Duration);
LoadedRed = LoadedRed + 1;
printf("Now LoadedRed = %d\n", LoadedRed);
```

```
LoadedRed = LoadedRed + 1;
```

Can also be written as:

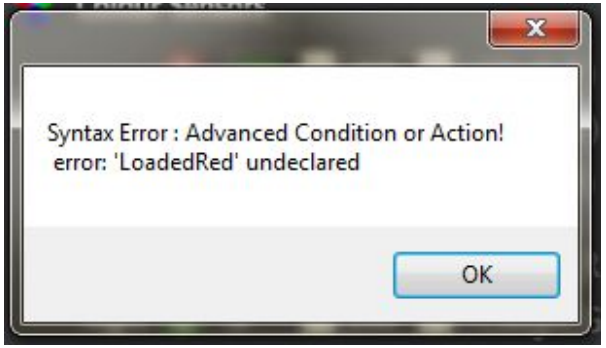
```
LoadedRed++;
```

Can add printf() to print the value of *LoadedRed*.

Syntax Errors

A POSTERIORI
Play · Experience · Learn

If you forget to add a **semicolon(;)** or make some spelling mistake in the code, you will get a Syntax Error popup, and you will not be able to advance - you won't be able to Save, Build, or change other statements in your program.


Syntax Error : Advanced Condition or Action!
error: 'LoadedRed' undeclared

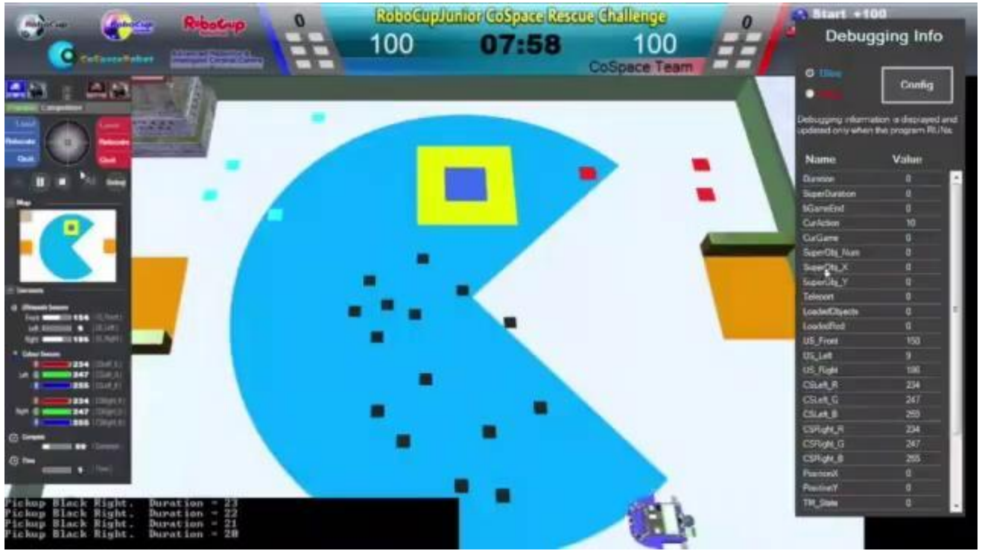
NOTE: I forgot to press OK after Adding New Variable in the Variable List window, so it didn't recognize this variable name.
I had to go back and add it properly, in order to proceed.

TEST TEST TEST

Before doing this for the other Red statement, or the rest of the Objects, let's TEST!

Make sure you can Build successfully.

Then Load your new AI and press Play.



Debugging information is displayed and updated only when the program RUNs

Name	Value
Duration	0
SuperDuration	0
bGameEnd	0
CurAction	10
CurGame	0
SuperObj_Num	0
SuperObj_X	0
SuperObj_Y	0
Teleport	0
LoadedObjects	0
LoadedRed	0
US_Front	150
US_Left	9
US_Right	186
CSLeft_R	234
CSLeft_G	247
CSLeft_B	255
CSRight_R	234
CSRight_G	247
CSRight_B	255
PositionX	0
PositionY	0
TM_State	0

```
Pickup Black Right.   Duration = 23
Pickup Black Right.   Duration = 22
Pickup Black Right.   Duration = 21
Pickup Black Right.   Duration = 20
```

The Bug

A POSTERIORI
Play · Experience · Learn

When you ask your Robot to do something for 3 seconds, it turns that into:

3000 ms / 60 ms per cycle =
50 cycles of 60ms each

For each cycle, it calls the Advanced Action code again…

So we end up executing:

```
LoadedRed = LoadedRed + 1;
```

50 times… **Until Duration is 0!**

(from 49 to 0; 50 total cycles)

```
Picking up Red Left.  Duration = 49
Now LoadedRed = 1
Picking up Red Left.  Duration = 48
Now LoadedRed = 2
Picking up Red Left.  Duration = 47
Now LoadedRed = 3
…
…
Picking up Red Left.  Duration = 3
Now LoadedRed = 47
Picking up Red Left.  Duration = 2
Now LoadedRed = 48
Picking up Red Left.  Duration = 1
Now LoadedRed = 49
Picking up Red Left.  Duration = 0
Now LoadedRed = 50
```

The Fix

Whenever you add FindObject Action to a statement, inside the code internally it is adding this logic:

```
if(Duration == 1) LoadedObjects++;
```

That means, between Duration 49 down to 0, it will only execute LoadedObjects++ (or LoadedObjects = LoadedObjects + 1) **only once!**

When? Only when the Duration is 1, just before the full action is over...

So, let's do the same for LoadedRed...

The Fix

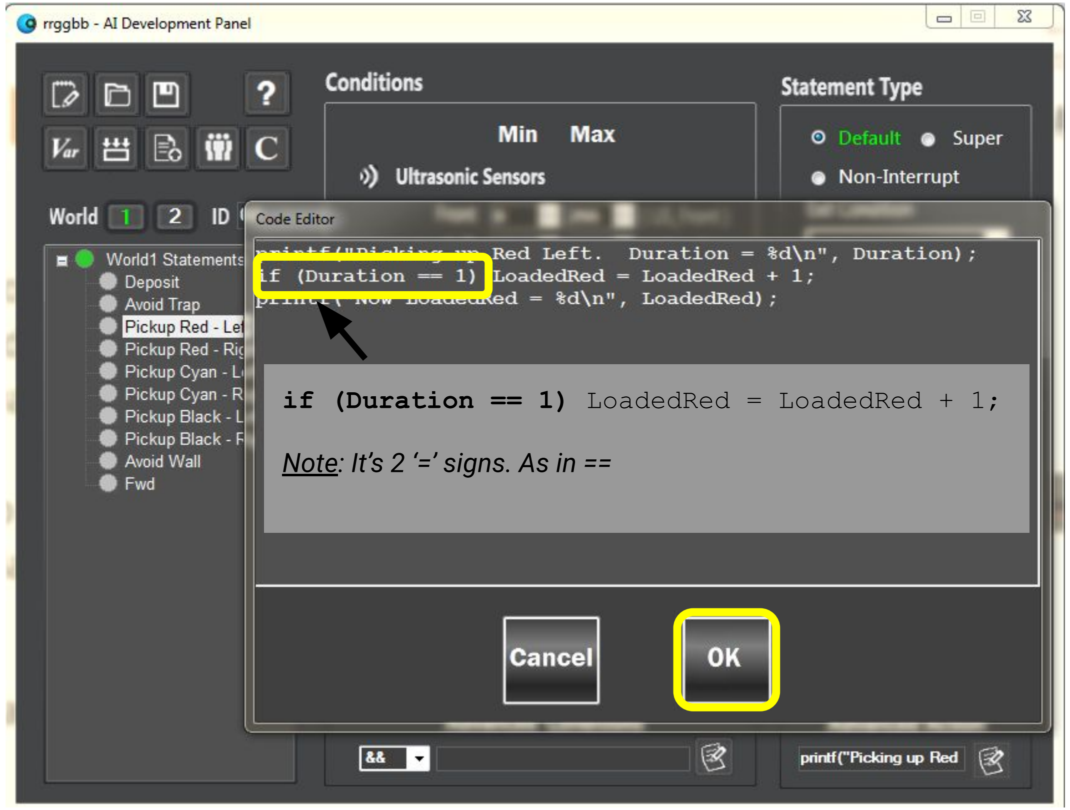
Correct the appropriate line of code to add one more to LoadedRed **only when Duration == 1.**

<u>Coding Notes</u>

= : means “set to”

== : means “is equal?”

Click OK to save



```
printf("Picking up Red Left.  Duration = %d\n", Duration);
if (Duration == 1) LoadedRed = LoadedRed + 1;
printf("Now LoadedRed = %d\n", LoadedRed);
```

```
if (Duration == 1) LoadedRed = LoadedRed + 1;
```

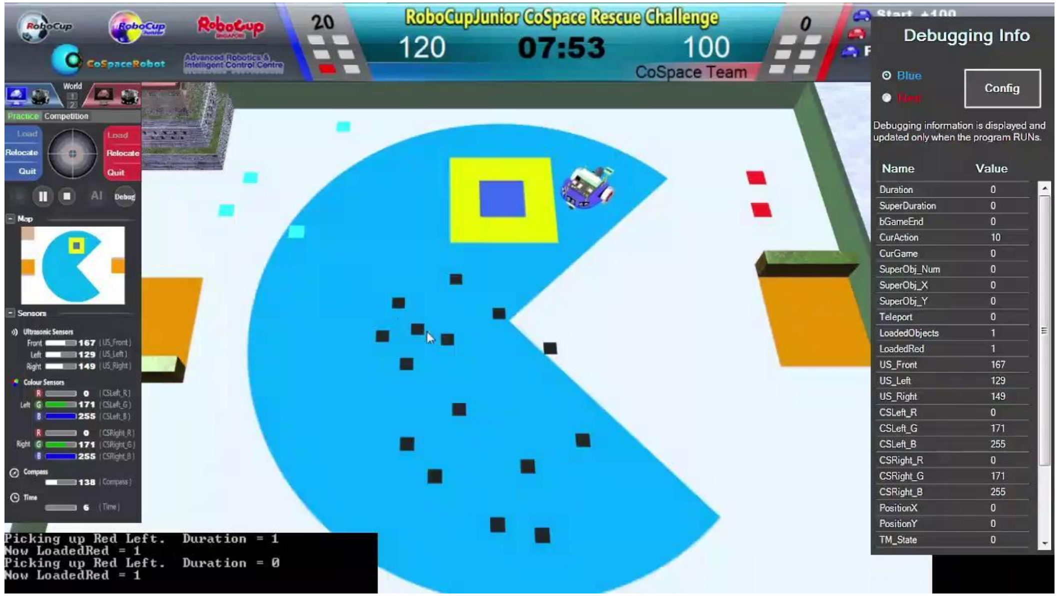
<u>Note</u>: It’s 2 ‘=’ signs. As in ==

printf("Picking up Red

TEST TEST TEST

Before doing this for the other Red statement, or the rest of the Objects, let's TEST!

Now LoadedRed seems to update correctly!



Debugging information is displayed and updated only when the program RUNs.

Name	Value
Duration	0
SuperDuration	0
bGameEnd	0
CurAction	10
CurGame	0
SuperObj_Num	0
SuperObj_X	0
SuperObj_Y	0
Teleport	0
LoadedObjects	1
LoadedRed	1
US_Front	167
US_Left	129
US_Right	149
CSLeft_R	0
CSLeft_G	171
CSLeft_B	255
CSRight_R	0
CSRight_G	171
CSRight_B	255
PositionX	0
PositionY	0
TM_State	0

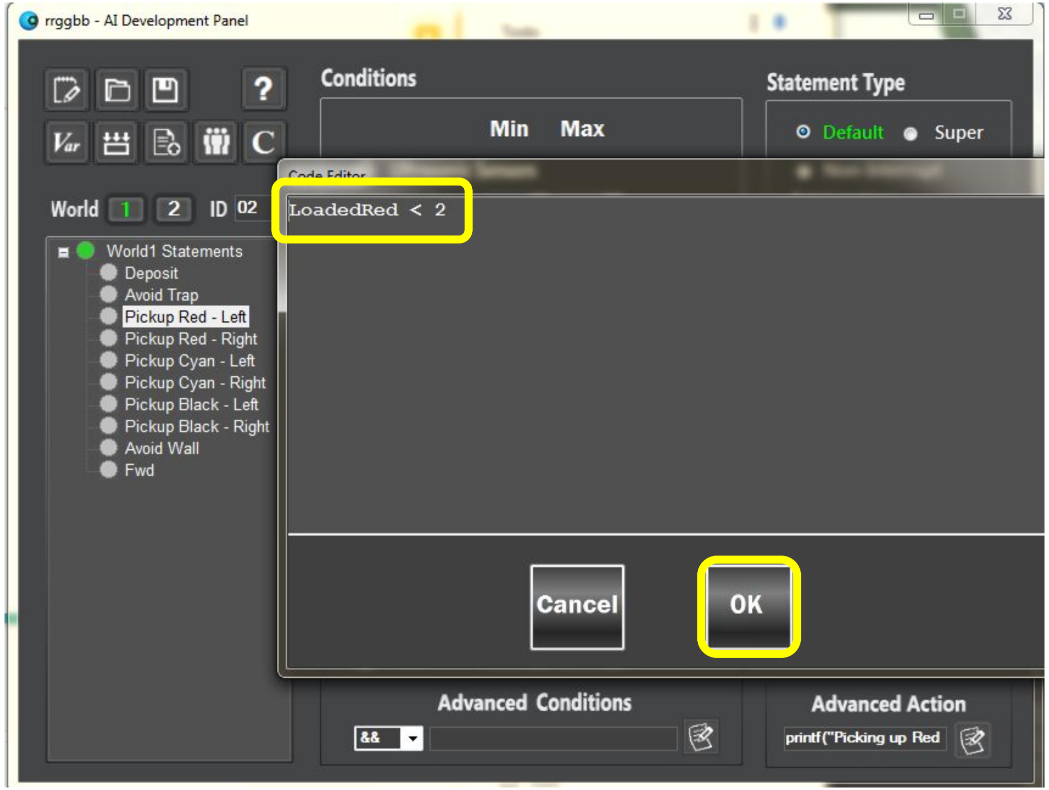
```
Picking up Red Left.  Duration = 1
Now LoadedRed = 1
Picking up Red Left.  Duration = 0
Now LoadedRed = 1
```

Now Add a Condition

We should only pickup Red, if LoadedRed < 2.

So, **add that condition** to the Pickup statement.

Make sure Build succeeds.



```
LoadedRed < 2
```

TEST TEST TEST

A POSTERIORI
Play · Experience · Learn

Build and **TEST**:

- Pickup 2 Reds
- Make sure it won’t pick up another one

If you only coded Left side, make sure you only test from Left side…

Otherwise you will get false results:
Robot won’t check condition, and it won’t update LoadedRed either…

Duplicate Logic to Other Side

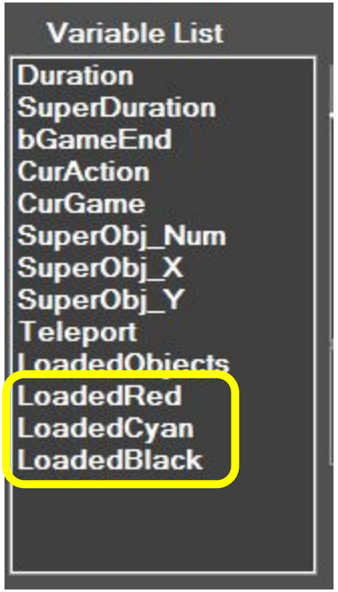
A POSTERIORI
Play · Experience · Learn

Duplicate the LoadedRed **Advanced Condition & Advanced Action** for both sides Pickup statements.

Then **add LoadedCyan** and **LoadedBlack variables**, and add similar logic for those two color Objects.

Test that the Robot will only pickup:

- **2 Red, 2 Cyan, and 2 Black Objects**



Variable List

Duration
SuperDuration
bGameEnd
CurAction
CurGame
SuperObj_Num
SuperObj_X
SuperObj_Y
Teleport
LoadedObjects
LoadedRed
LoadedCyan
LoadedBlack

Deposit Advanced Condition

In our example RRGGBB strategy, you should only Deposit if all 3 are True:

- LoadedRed == 2
- LoadedCyan == 2
- LoadedBlack == 2

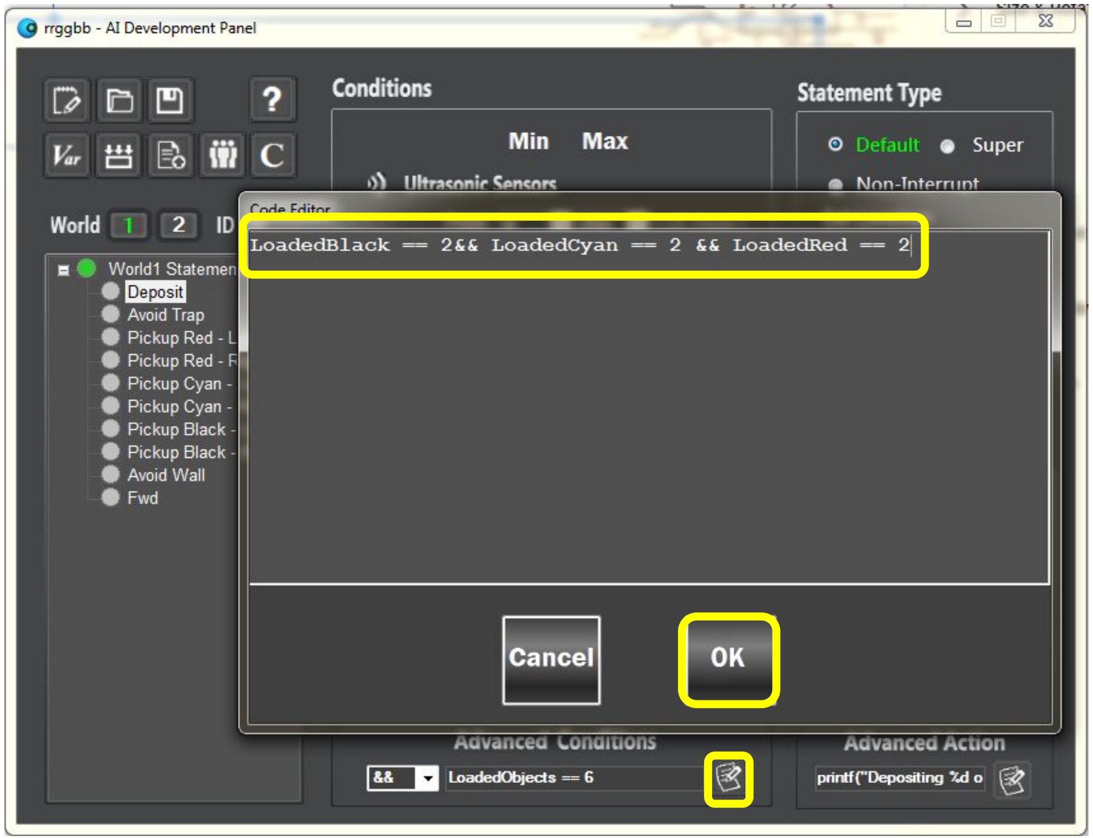
You can add that aggregate condition to Deposit statement using this Advanced Condition:

```
LoadedRed == 2 && LoadedCyan == 2 && LoadedBlack == 2
```

Deposit Advanced Condition

Since the only way you could ever get this condition is when you're also fully loaded, you can also simplify this to:

```
LoadedObjects == 6
```



```
LoadedBlack == 2&& LoadedCyan == 2 && LoadedRed == 2
```

Advanced Conditions: && LoadedObjects == 6

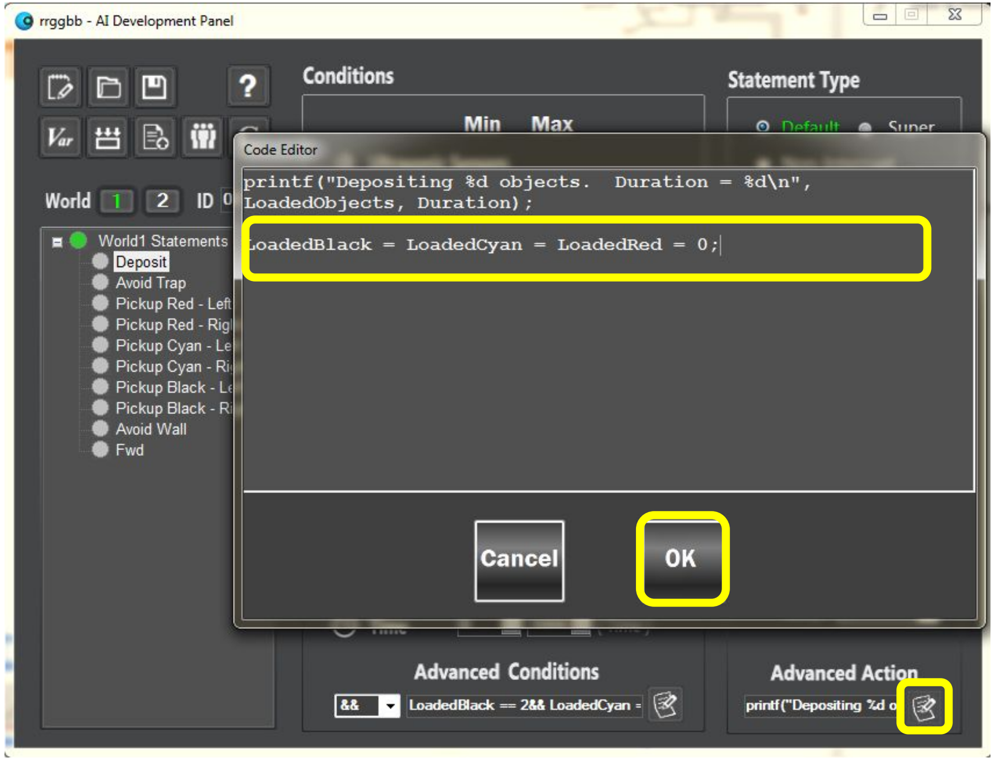
Advanced Action: printf("Depositing %d o

Deposit Advanced Action

Also we need to remember to clear all of our variables after Depositing.

Set all 3 variables back to 0.

```
LoadedBlack = LoadedCyan = LoadedRed = 0;
```



```
printf("Depositing %d objects.  Duration = %d\n",
LoadedObjects, Duration);

LoadedBlack = LoadedCyan = LoadedRed = 0;
```

LoadedBlack == 2&& LoadedCyan =

printf("Depositing %d o

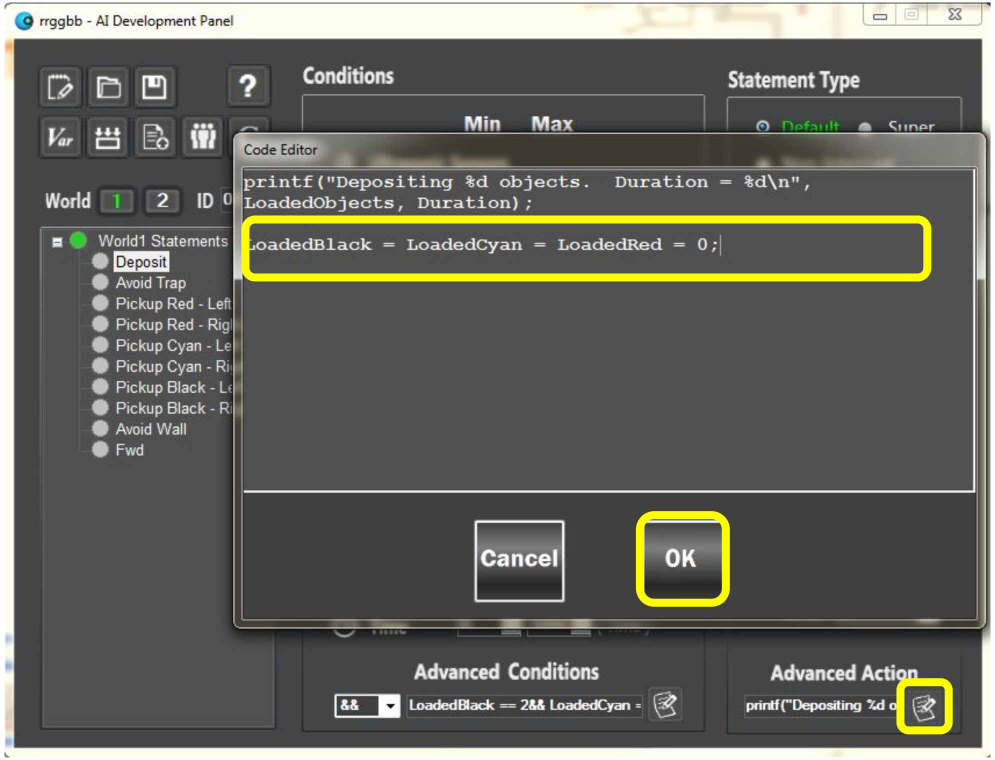
Deposit Advanced Action

A POSTERIORI
Play · Experience · Learn

Do we need
`if (Duration == 1)`?

Test without, and see...

```
if (Duration == 1)
 LoadedBlack = LoadedCyan = LoadedRed = 0;
```



```
printf("Depositing %d objects.  Duration = %d\n", LoadedObjects, Duration);

LoadedBlack = LoadedCyan = LoadedRed = 0;
```

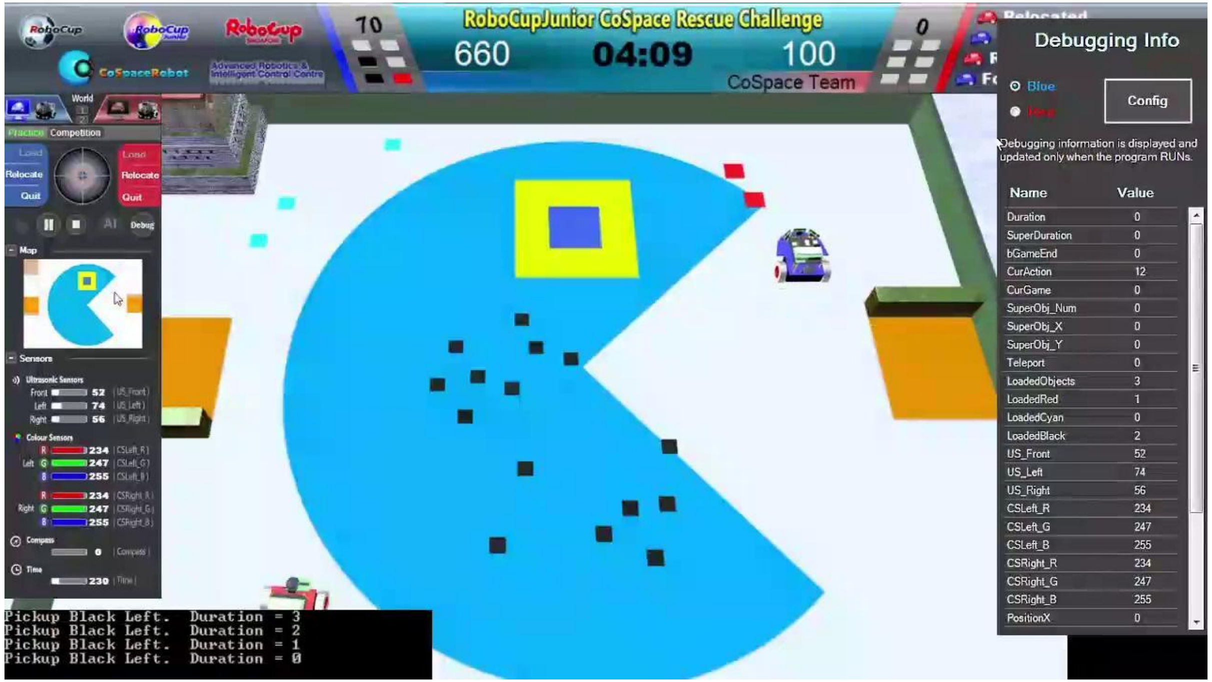
LoadedBlack == 2&& LoadedCyan =

printf("Depositing %d o

Full Example

Build and test the full algorithm.

You can fold the *“Follow Wall When Fully Loaded”* strategy into this algorithm as well.



Debugging Info

Blue
Config

Debugging information is displayed and updated only when the program RUNs.

Name	Value
Duration	0
SuperDuration	0
bGameEnd	0
CurAction	12
CurGame	0
SuperObj_Num	0
SuperObj_X	0
SuperObj_Y	0
Teleport	0
LoadedObjects	3
LoadedRed	1
LoadedCyan	0
LoadedBlack	2
US_Front	52
US_Left	74
US_Right	56
CSLeft_R	234
CSLeft_G	247
CSLeft_B	255
CSRight_R	234
CSRight_G	247
CSRight_B	255
PositionX	0

```
Pickup Black Left.  Duration = 3
Pickup Black Left.  Duration = 2
Pickup Black Left.  Duration = 1
Pickup Black Left.  Duration = 0
```