

Content
Introduction
AI chart
Building your bot
Binding
Constructor parameter
Adding a binding
Minimum binding
Binding for the nulls
Replace a bot in a AI-team with your own one
Create a new AI-Team
Write you own AI
Trouble Shooting
Tools

Introduction

Creating and driving you own bot is a real pleasure. But once, you may want to fight against your bot or you would like to enter a championship, you need to Aled your bot. AIng is the process of adding the information to allow the computer to drive your bot. But the job does not stop here, you must also delivers your bot in an existing team (and replace a previous bot) or create your own AI. At last, you may also want to change thoe order of the Aied team so you should be able to meet them in event or, one step to heaven, you would like to create your own strange AI.

This document shows :

- **AI Chart** : describe briefly the common AI
- **Building your bot** : some requirements must be fulfilled when building your bots
- **Binding** : describe the way you attach an AI to a bot
- **Replace a bot in a AI-team with your own** : all is in the title !
- **Create a new AI team**: if you do not want to overwrite an existing AI, this will shows you how to add AI-teams
- **Write you own AI** : Try this to become a guru.
- **Troubleshooting** : Because it may happen, it will happen
- **Tools** : make life easier







The common steps to AI a bot are those :



- Create a bot in the lab
- Export your bot with a bot name like bot0, bot1 ... bot5
- Move your bot (from RA2/robot design) to your target team (you may have create your own team before, or saved a bot in an existing AI team before replacing it)
- Modify the file AI/bindings.py to add your new bot binding
- Launch RA2
- Debug
- Launch RA2
- Debug
- Launch RA2
- Debug again
- It works ? No : go 1 step above / Yes : go to sleep, you're tired.

Before playing the sorcerer, make an archive of your RA2 directory (winzip or winrar your RA2 directory). Sure, you will find trouble and sometime, you may not know how to recover from them. You will be glad to restore everything from your archive. The more you add Aied-bot or try against-nature experiment, the more often you will need to archive your directory from a stable state.




AI Chart




All information are case sensitive. For instance, in a computer view, the word ‘Spin’ does not equal the word ‘spin’
Name of controller start always by uppercase character.
Name of smartzone is always lowercase .

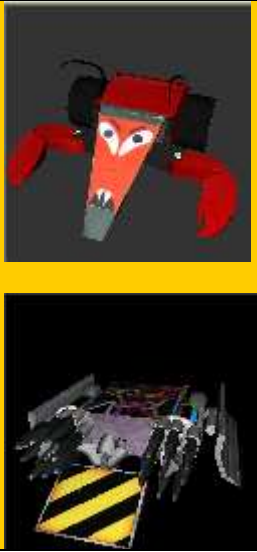

AI.py	AI .py file	SmartZone names	Weapon Control (trigger) names	‘Srimech’
<div>Chopper/Hammer bot</div> <div></div>	Chopper.py	weapon	Fire (button)	(Automatically uses main weapon[s].)
<div>Description: For Chopping or Hammering type bots (like DEADBEAT) : Engage than Fire</div> <div>Bindings Info/Example(s):</div>				
<div>Poker bot</div> <div></div>	Poker.py	weapon	Fire (button)	(Automatically uses main weapon[s].)
<div>Description: For poking/jabbing type bots (like ALARM)</div> <div>Bindings Info/Example(s): list.append(("FireStorm", "Poker", {'nose':math.pi,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(19,)}))</div>				
<div>Flipper bot</div> <div></div>	Flipper.py	flip	Flip (button)	(Automatically uses main weapon[s].)

Description: For lifting/flipping bots (like EMERGENCY), fire weapon if not upside-down. Personnal advice : Poker works as well or better for this kind of bot.				
Bindings Info/Example(s): list.append(("iNsAnItY","Flipper",{ 'nose':math.pi*2,'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(5,)}))				
Popup bot ?????????	Popup.py	weapon	Fire (button)	'Srimech'
Description: For popup wedges that only want to fire their weapons when they can hit the chassis. Components will not trigger the weapon.				
Bindings Info/Example(s): ?????????				
Omni bot 	Omni.py	weapon	Fire, Spin (buttons)	'Srimech'
Description: Very good all around AI.py, for most kinds of bots. Can have a spinning weapon plus a firing weapon.				
Bindings Info/Example(s): list.append(("Behemoth","Omni",{ 'nose':math.pi,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(6,23)}))				
OmniRam bot	OmniRam.py	weapon	Fire, Spin (buttons)	'Srimech'
Description: Omni bot that will back up and ram repeatedly instead of constantly pushing. For rammer/hammer/spinner hybrids and such.				
Bindings Info/Example(s):				
OmniSpin bot	OmniSpin.py	weapon	Fire , Spin (buttons)	'Srimech'
Description: Omni bot that will stop moving when it delivers a big hit, allowing its spinner more time to spin back up. For primarily spinners with srimechs or other weapons.				
Bindings Info/Example(s):				
InvertOmni bot	InvertOmni.py	weapon	Fire , Spin (buttons)	'Srimech'
Description: Same as Omni, but will become invertible once components designated as a srimech break/fail. For bots that are invertible, but fight better when right-side up.				
Bindings Info/Example(s):				
Whipper bot 	Whipper.py WhipperPlus	whipzone	N/A	
Description: Will quickly turn either back and forth (default value) or continuously around (<i>whip</i> parameter = 'around') if bot triggers the smart zone. For sit-and-spinners. WhipperPlus : much like whipper but try to avoid immobility problems				
Bindings Info/Example(s): list.append(("SlapHappy","Whipper",{ 'invertible':True,'nose':math.pi,'whip':"around",'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(9,10)}))				
Frenzy bot (above w/ Hammer strategy)	Frenzy.py	whipzone	Hammer (analog)	




				
Description: - Rotates a weapon (intended for spin motor-powered hammers) back and forth as long as a bot is in the smart zone.				
Bindings Info/Example(s): list.append(("frenZy", "Frenzy", {'nose':math.pi/2,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(26,)}))				
Spinning bot  	Spinner.py DirectionnalSpinner.py	N/A	Spin (button)	
Description: Spins weapon continuously and stops on a big hit to allow spin-up time. Options for full-body spinners or forward-oriented spinners.				
Bindings Info/Example(s): list.append(("Steel Meatball", "Spinner", {'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(11,)}))				
VertSpinner bot 	VertSpinner.py	N/A weapon	Spin (analog) Fire (button)	'Srimech'
Description: Spins its weapon in reverse to self-right. Also becomes invertible when its secondary weapons break. For vertical spinners that can self-right by reversing their weapon and/or firing a srimech.				
Bindings Info/Example(s): list.append(("Layziee", "VertSpinner", {'radius':1,'topspeed':100,'throttle':130,'turn':100,'turnspeed':3.5,'range':99,'weapons':(19,),'sweapons':(19,)}))				
Bee bot 	Bee.py	whipzone	Spin (button)	
Description: Whipper AI (see above) that allows to have a spin motor				
Bindings Info/Example(s):				
Bee CC bot	BeeCC.py	whipzone	N/A	
Description:				
Bindings Info/Example(s):				
Pillar bot	Pillar.py PillarPlus.bot	N/A	Spin (button)	

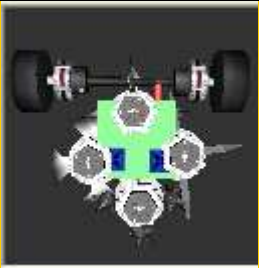
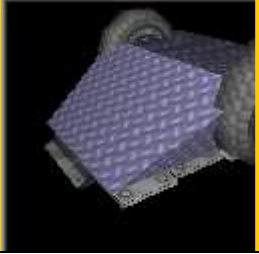
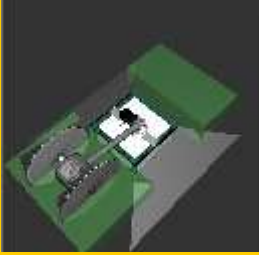

				
Description: - Sits-and-spins in just one direction once opponent is in specified range, and doesn't stop unless immobile counter begins. For sit-and-spinners that only spin one direction. Bindings Info/Example(s):				
Pusher bot 	Pusher.py	N/A	N/A	
Description: A ramming AI that only backs up for another charge when it is at a nearly complete stop. For slow rammers or bots that rely more on pushing than ramming. Bindings Info/Example(s): list.append(("Devil II", "Pusher", {'nose':math.pi*2,'radius':0.1,'range':99,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2}))				
PusherPlus bot	PusherPlus.py	N/A	N/A	'Srimech'
Description: Pusher + Srimech command Bindings Info/Example(s):				
Ram bot 	Rammer.py	N/A	N/A	
Description: A ramming AI that backs up for another ram once it is moving slowly. Bindings Info/Example(s): list.append(("Killdozer", "Rammer", {'invertible':True,'nose':math.pi,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(10,)}))				


Other bot (LittleMetalFriend) (pushes+grabs) 	Other.py	squeeze	Spin (button) Lefthook (analog) Righthook (analog)	‘Srimech’
Description: pushes+grabs Bindings Info/Example(s): list.append(("MyBot", " LittleMetalFriend ",{'weapons':(10,)})				
Smashbox 	Smashbox.py	weapon	Fire (button) Spin (button)	‘Srimech’
Description: Spin than fire with slight delay if bot in range. Distinguish spin weapon (sweapon) , trigger (tweapon) and ? (qweapon) in its parameter. This distinction is coded in a way that the AI will never react as if it loose all its weapon. Bindings Info/Example(s): list.append(("Smashbox", "Smashbox",{'nose':math.pi,'radius':1,'reload':8,'topspeed':18,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(20,),'sweapons':(21,),'tweapons':(22,),'qweapons':(23,)})				
TerraTurtle 	TerraTurtle.py	weapon	Fire (button) Spin (button)	‘Srimech’
Description: much like OmniRam strategy : Ram and spin Bindings Info/Example(s): list.append(("Smoker", "Terraturtle",{'nose':math.pi*2,'invertible':True,'range':50,'radius':1.6,'topspeed':100,'throttle':130,'turn':100,'turnspeed':2,'weapons':(9,10,11,12,13,14,15,16,)})				
Flyer bot	Flyer.py	weapon	Fire (button)	‘Srimech’
Description: omni that define a proximity parameter (like ranger) and a trigger named 'trigger 2' with default value of 'Srimech' that is activated when an enemy is getting close. 'Trigger2' is assumed to make the bot fly. Bindings Info/Example(s):				
Topknot bot	Topknot.py	Zone1: whipzone Zone2: AxeF Zone3: AxeB Zone4: AxeL Zone5: AxeR Zone6: under (fires all)	N/A FireF FireB FireL FireR (buttons)	‘Srimech’
Description: Versatile AI with four independent weapons and whipper function, used for excessively complex designs.				
Simple Weapon bot (fires weapon every 5 sec.)	SimpleWeapon.py	N/A	N/A	
SwitchWep bot	SwitchWep.py	Zone1: PrimaryWep Zone2: SecondaryWep Zone3: OtherWep Zone4: OtherWep2	PrimaryWep SecondaryWep OtherWep OtherWep2 (buttons)	‘Srimech’

				
Description: Once a primary weapon breaks, it will stop firing and a secondary weapon will fire instead. Also has 2 additional independent weapons. For bots that have a sort of emergency backup weapon that interferes with the primary weapon.				
Bindings Info/Example(s):				
InvertSwitchWep bot	InvertSwitchWep.py	Zone1: PrimaryWep Zone2: SecondaryWep Zone3: OtherWep Zone4: OtherWep2	PrimaryWep SecondaryWep OtherWep OtherWep2 (buttons)	‘Srimech’
Description: Same as SwitchWep, but becomes invertible once components designated as a srimech breaks. For SwitchWep bots that are invertible, but fight better right-side up.				
Bindings Info/Example(s):				
FBS 	FBS_2.py		"Ahead" LeftRight Spin	
Description: Sit and Spin, better than Whipper ‘PreSpinEntrance’: 0 This decides how much, if any, your bot will initially enter (go forward) toward the center of the arena, before spinning up. Use this to initially clear walls, or even to ram your opponent). Parameters here are '0' = none (spin immediately in place.); to '1-30' (move toward enemy 'x' amount, then spin). (Note: If your FBS has trouble staying straight during the ‘PreSpinEntrance’ , lower the ‘turn’:xx and ‘turnspeed’:x.x by 20%. If not fixed, repeat until corrected. This will NOT affect 'Spin' speed.) ‘SpinDirection(1/-1)’: 1 1 is clockwise, -1 is counterclockwise. ‘ReMobilizeRoutineTime(10-60)’: 20 This designates how long the bot will try to go 'forward' and then 'backward' to get out of immobilization. 10 is short time, 60 is longer. Adjust for each bot's needs (weight, motors, wheels...). Parameters here are between 10-60.				
Bindings Info/Example(s): list.append(("Bot-1","FBS_1",{ ‘PreSpinEntrance’ :0, ‘SpinDirection(1/-1)’ :1, ‘ReMobilizeRoutineTime(10-60)’ :10, 'range':99, 'radius':0.1, 'topspeed':120, 'throttle':120, 'turn':30, 'turnspeed':1.5, 'weapons':(11,))})				

*** Exotic Ai ****				
AI.py	AI .py file	SmartZone names	Weapon Control (trigger) names	‘Srimech’
SOW (servomotor)	D_SMFE Turret Machinegun_Spin.py	zone1,zone...zone8 whipzone zone also get ‘regroupement zone’ see AI.	Fire1, Fire2...Fire8 Fire	
Description: made to drive a tank-like bot, this AI may be used to drive any SOW bot. ID of servo motor must be given as a binding parameter. Servo motor aiming is calculated and not dependant of any smartzone.				
Bindings Info/Example(s):				

SOW (servomotor) 	Top_Smasher_2R.py	zone1,zone...zone8 whipzone zone also get 'regroupement zone' see AI.	Fire1, Fire2...Fire8 Fire	
Description: one of the best SOW AI. . ID of servo motor must be given as a binding parameter. Brings the bot 'Skinhead_HW_4c to the podium in Gumba challenge'. . Servo motor aiming is calculated and not dependant of any smartzone.				
Bindings Info/Example(s): <code>list.append((" SkinHead_HW_4c","Top_Smasher_2R",{ 'radius':0.1, 'topspeed':130, 'throttle':130, 'turn':60,'turnspeed':1.2, 'TimerSpeed3thru4':5, 'TimerSpeed5thru8':5, 'TimerSpeed1thru7':5, 'SRM_Tlmer':1,'TS_range': 3, 'motor':1, 'weapons':(10,11))})</code>				
SOW (servomotor) 	LaserGuidedV3.py (V4 should come)		Servo Spin Fire	
Description: much like Top_Smasher_2R but the ID of the motor is calculated and not given in parameters -> constraint : just one servo motor in your bot.				
Bindings Info/Example(s): <code>list.append(("TazbotLG","LaserGuidedV3",{ 'nose':math.pi,'range':80, 'radius':3, 'servorange':2,'servonose':-1,'delta':0.15, 'servospeed':60, 'topspeed':99, 'mayFire':True, 'throttle':130,'weapons':(1,2,3,4,5,6,7,8,9))})</code>				
In My Arms 	InMyArms2.py	servozone	Servo Spin Fire	
Description: Open arms when enemy is far, close them when enemy in smart zone, many parameters : open angle, close angle, frequency to check if need to open/close arms/etc. see AI for more details.				
Bindings Info/Example(s): <code>list.append(("testBotX", "InMyArms2", { 'nose': math.pi, 'servonose':1, 'servoopenangle':2.9, 'servocloseangle':0.5, 'servodelta':0.15, 'servospeed':100, 'range': 99,'weapons': (1,2,3,4) }))</code>				
Change frontal weapon from time to time	ShiftWeapon3.py	servozone	Servo Spin Fire	

				
Description: weapons are mounted on a servo motor that switch angle from time to time,				
Bindings Info/Example(s):				
list.append(("shiftweapon II", "ShiftWeapon3", { 'nose': math.pi, 'servospeed':30,'servodelta':0.05, 'servoTimer':35, 'servoNbPos':4,'weapons': (1,2,3,4) }))				
Feel better whene inverted 	OmniInverted.py	weapon	Fire Spin	Grimech
Description: when not, try to become inverted				
Bindings Info/Example(s):				
list.append(("testInverter", "OmniInverted", { 'invertible': True, 'nose': math.pi, 'weapons': (8, 12, 14, 15) }))				
(SOW) Omni with servo mounted weapon 	omniservo.py SpinServo.py	zoneservo1 zoneservo2	Servo Fire Spin	
Description: weapon change position depending on which smartzone foe enter. Less effective than SOW AI that does not rely on smartzone.				
Bindings Info/Example(s):				
Omni that allows multi zone much like topknot 	omnimultizone.py	weapon weapon1 weapon2 weapon3 waepo4	Fire Fire1 Fire2 Fire3 Fire4	Grimech
Description: much like omni but multi zone allowed. Also authorized : selection of the tactic as a parameter.				
Bindings Info/Example(s):				
Poker with no smart zone Omni where firing weapon does not rely on smartzone	RangePoker.py RangeOmni.py			
Description: Poker that does not need smartzone but that relies on its nose direction and range to fire weapon				
Bindings Info/Example(s):				
Try to flank than charge	Plow.py			

				
Description:				
Bindings Info/Example(s):				
FOR EVEN MORE CRAZY AI (Flying bots, etc) see the Madiaba showcase on gametechmods.				

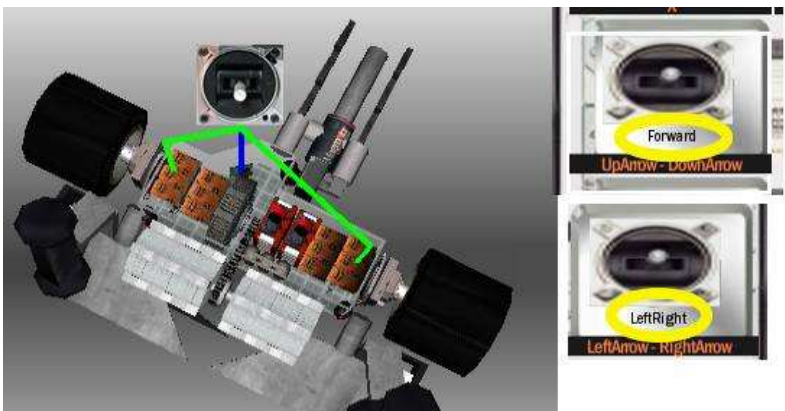
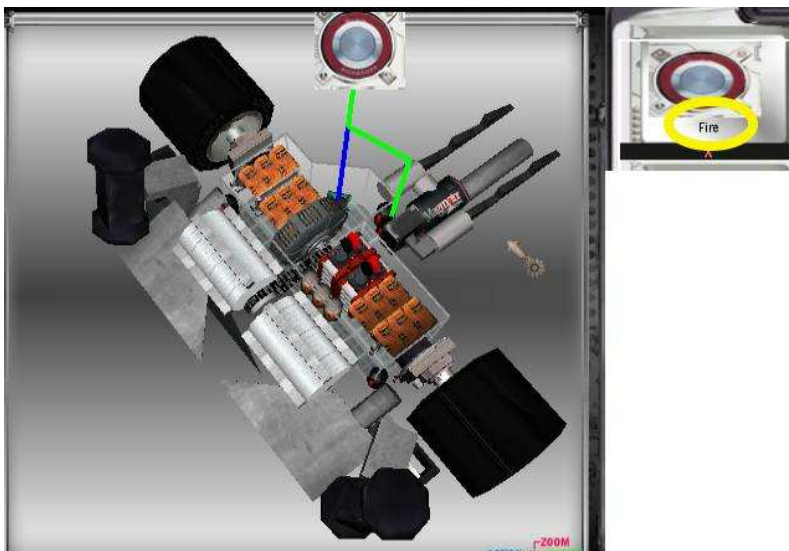
NOTE: 'Srimech' is for triggering the up-righting mechanism(s) in an inverted bot: (Self righting mechanism).

Most useful AI : Spinner, Rammer, Whipper; Omni and Poker.

BUILDING YOUR BOT

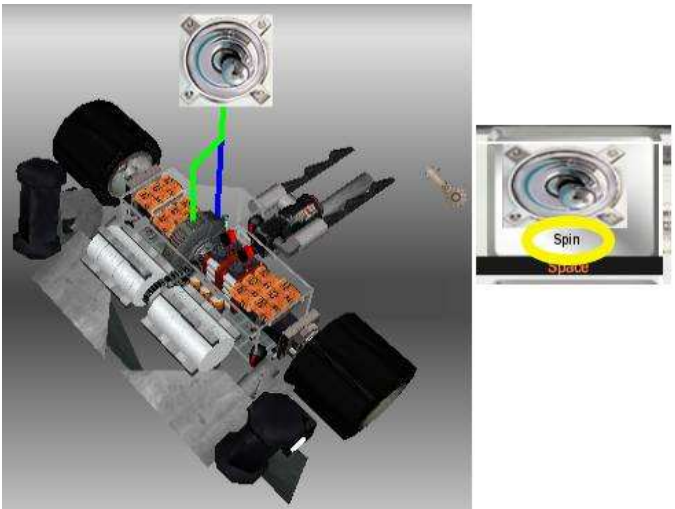
While you build your bot, you must fulfilled some requirements about the name of the controller.

Driving controllers : They must be **Analog controller** and names **Forward** and **LeftRight** (case sensitive) :

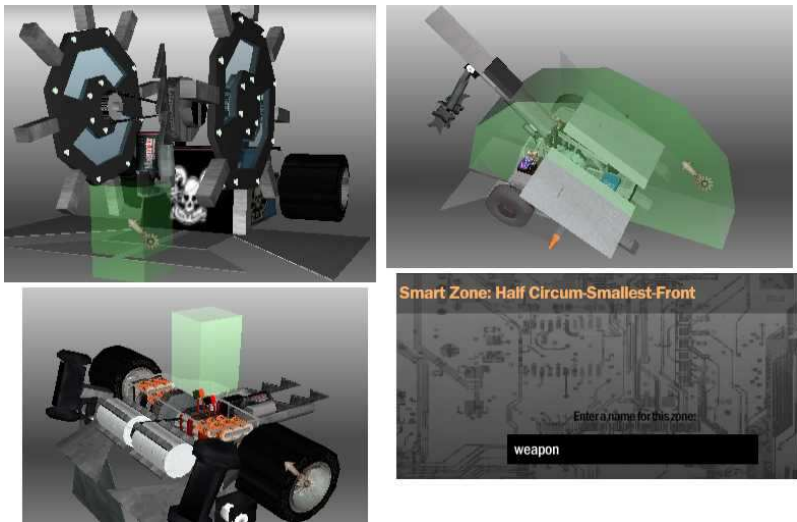


Firing controller : they must be **button**. Most of the time they are called '**Fire**'.

See AI Chart above for special name consideration. Srimech controllers work like Fire controllers but their name.



Spin controller. They must be **button** (some exemption as shown on the AI chart above). Most of the time they are called "**Spin**".



At last : **Firing mechanism** are activated when enemy enter special zones names as '**Custom Zone**'. You must define theses. Most of the time they are called '**weapon**' (and not **Weapon**). Whipper called them '**whipzone**'. See AI chart above for special name requirements. Here are sample placements. A same bot may have many custom zones with the same name.

BINDING:

All these information are case sensitive. Error in typing leads to RA2 crash !

binding format: ('name of robot from .bot file', 'name of AI.py from script class', { 'optional constructor parameter':value, 'another':value})

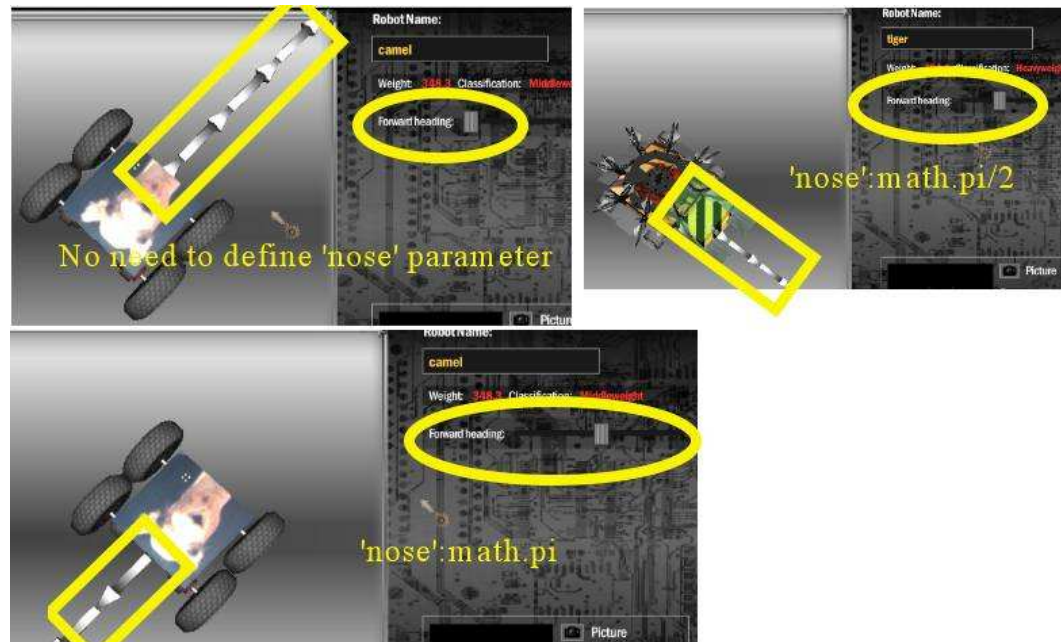
constructor parameters:

nose is "front" of bot- default value is 0 (equals to $2 * \text{math.pi}$).

angles go from $-\text{math.pi}$ through 0 to math.pi

typical nose values may be : 'nose': math.pi , 'nose': $-\text{math.pi} / 2$, 'nose': $\text{math.pi}/2$

you know your nose direction by looking at the profile of your bot :



All about nose :

```
# nose is "front" of bot-
# Nose paradigm: All 360 rotational degrees for "math.pi" are represented by the factor '2'.
# thus: '2'=360degrees, '1'=180degrees, '0.5'=90degrees, '0.25'=45degrees.

#The direction of change(CW/CCW)is determined by the "-" or "+" before the statement.
#thus: '+' causes rotation CCW, '-' causes rotation CW.

#Format examples:
# " 'nose':math.pi*0.25 " will rotate the bot 45degrees/CCW.
# " 'nose':-math.pi*0.25 " will rotate the bot 45degrees/CW.
# " 'nose':math.pi*0.5 " will rotate the bot 90degrees/CCW.
# " 'nose':-math.pi*0.5 " will rotate the bot 90degrees/CW.
# " 'nose':math.pi*0.75 " will rotate the bot 135degrees/CCW.
# " 'nose':-math.pi*0.75 " will rotate the bot 135degrees/CW.
# " 'nose': -math.pi " and math.pi rotate the bot 180 degrees.

(Make sure there is a comma(,)after the the value # (to separate it from the next characteristic and its value).
(-----notes-----)

Note1: "math.pi" AND 'nose':math.pi*1.0 " AND " 'nose': -math.pi*1.0 " will rotate the bot 180 degrees.
(The first two are the same, and the last two just rotate different directions to get 180 degrees.)
Thus there is no real reason to have a factor greater than '1',
since after that you just move into the other half of the circle covered by the other '+' or '-' sign.
Note2: "math.pi*0" or "math.pi*2" = the same as no new heading.
```

invertible - can function upside-down (default False). Once inverted, an invertible bot will try to move to the nearest enemy. Once inverted, a non-invertible bot will try to use its Srimach to go up right. Moreover, some AI will tell a bot not to fight an inverted enemy bot waiting for the countdown to start.

Typical invertible Value may be : 'invertible': True

topspeed - speed in meters/second AI will attempt not to exceed (default 4.0).

throttle - maximum analog value AI will attempt not to exceed (default 100)

turnspeed - turning in radians/second AI will attempt not to exceed (default 2.5)

turn - maximum analog value AI will attempt not to exceed (default 60)

radius - bot radius to used to check distances to hazards, walls, and other bots. (default 1.0). With a high value, your bot will avoid hazards, with a null value, it will ignore them (but hazards will not ignore your bot !). Low value may be useful for high speed bots like pusher or rammer that rely more on their speed than maneuver.

weapon – Id of the weapon. Once all the components with these Id are broken, the strategy of the bot may change to pusher.

Easy usage : put a number for each weapon you have. Sample : 'weapons':(1,2,3,4,5,6)

Adding binding

The file **AI/bindings.py** contains all the bindings of all the bots. **You need to add your line to this file.**

This is a python file which is in fact a program. So you not only need to describe your binding, but you must also need to write Python code to tell the machine to keep your binding. This is easy ; just include you binding in the code below :

```
list.append( your binding )
```

- Warning : **the line containing ‘list append’ must start with exactly 4 blank space** (no tabulation).
- Advice : because bindings.py is a program, each line had a specific role. So, when you add your binding, do it after an existing binding line. It will help you to be sure that indentation is correct
- Advice : most of the time the file bindings.py groups the Ai definition by team, so add your line in the good group.
- Pretty : in this file, **lines strating with # are comments**. You may add your own.

Sample Bindings.py

```
import math

def load(list):
    print "Loading AI bindings"

    # binding format:
    # ( 'name of robot from .bot file', 'name of AI from script class',
    #   { 'optional constructor parameter':value, 'another':value } )

    # constructor parameters:
    # nose - "front" of bot in radians (default 0)
    # invertible - can function upside-down (default False)
    # topspeed - speed in meters/second AI will attempt not to exceed (default 4.0)
    # throttle - maximum analog value AI will attempt not to exceed (default 100)
    # turnspeed - turning in radians/second AI will attempt not to exceed (default 2.5)
    # turn - maximum analog value AI will attempt not to exceed (default 60)
    # radius - bot radius to use for checking for hazards and walls (default 1.0)

    #team0
    list.append(("Steel Meatball", "Spinner", {'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(11,)}))
    list.append(("T O R Q U E", "Spinner", {'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(11,)}))
    list.append(("Facemlikeaman", "Omni", {'invertible':True,'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(11,)}))
    list.append(("Hazardous Contraption", "Omni", {'nose':math.pi,'radius':0.5,'range':50,'topspeed':100,'turn':50,'turnspeed':3,'weapons':(13,14,15,16,17,18,19,20)}))

    #team1
    list.append(("AW-Terminus", "Spinner", {'nose':math.pi*2,'range':30,'radius':0.4,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(6,9,10,11,12)}))
    list.append(("Gammaraze III", "Omni", {'invertible':True,'nose':math.pi*2,'radius':0.1,'range':500,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(12,20,22,36,37,40)}))

    # 39 - Arthrobotics
    list.append(("Wedge of Doom", "Poker", {'nose':math.pi,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(16,)}))
    list.append(("BW-Peer Pressure", "Flipper", {'nose':math.pi*2,'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':4.5,'weapons':(10,)}))
    list.append(("Triclops v2.0", "Omni", {'nose':math.pi*2,'radius':0.1,'topspeed':100,'throttle':130,'turn':130,'turnspeed':2.5,'weapons':(23,24,25)}))

    #team2
    list.append(("Hum-Drums", "Omni", {'invertible':True,'nose':math.pi,'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':50,'turnspeed':1.5,'weapons':(11,)}))
    list.append(("Mako 4", "Omni", {'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(11,)}))
    list.append(("Morse Code Breaker", "Omni", {'nose':math.pi,'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(11,)}))
    list.append(("Mutant Dragonfly III", "Omni", {'invertible':True,'nose':math.pi,'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(11,)}))
    list.append(("obZen", "Omni", {'invertible':False,'nose':math.pi,'range':99,'radius':0.1,'topspeed':100,'throttle':130,'turn':60,'turnspeed':2.5,'weapons':(11,)}))
```

Minimum binding

The minimum information are : the name of the bot, its AI, it’s weapon. So a line like the one above is correct :

```
list.append(("Steel Meatball", "Spinner", {'weapons':(11,)}))
```

Binding for the nulls

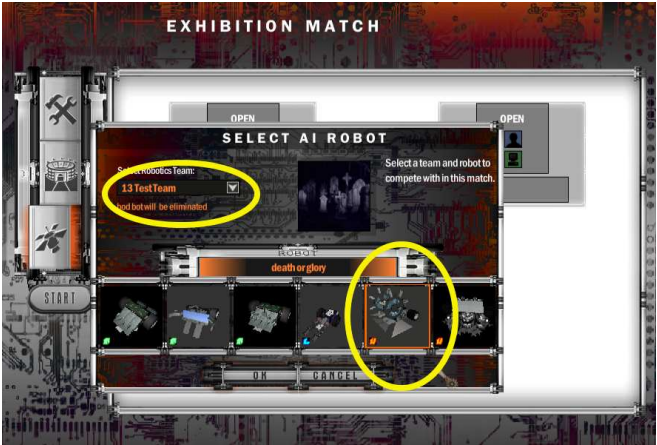
Try this, it should work, as long as you build correctly your bot :

```
list.append(("You Bot Name Here", "Omni", {'weapons':(1)}))
```

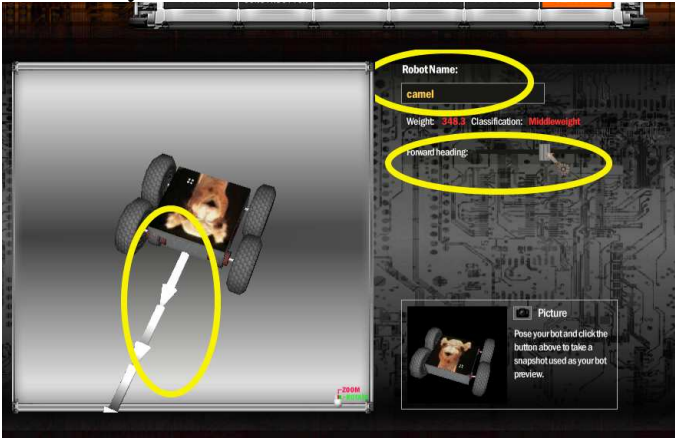

Replace a bot in a AI-team with your own one (from gametechmods forum)

Here are the easy step to replace a bot in a team :

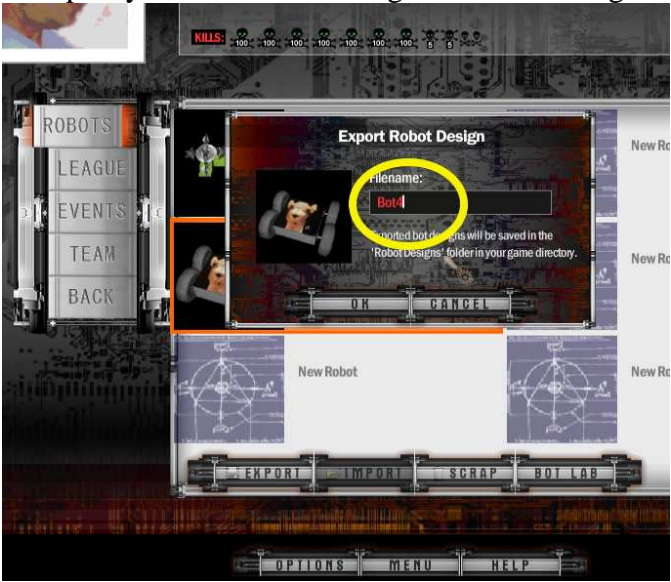
1/ memorize which bot in which team you would like to change :



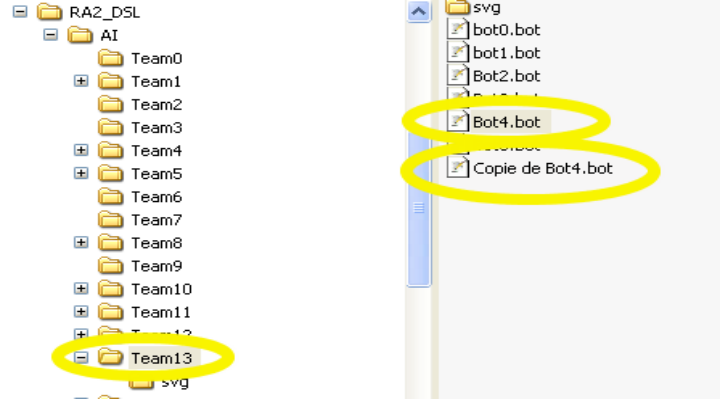
2/ Create your killer bot. Give it a name, be aware of its nose attitude :



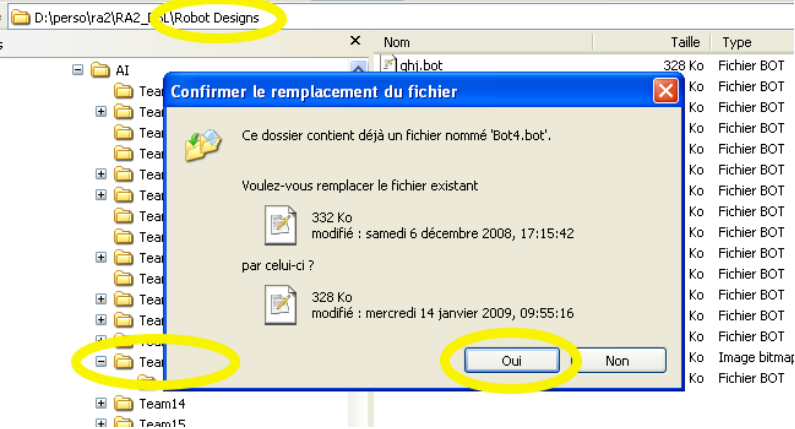
3/ export your bot : For name give it bot0 through bot5 depending of the bot you would like to change in your target team



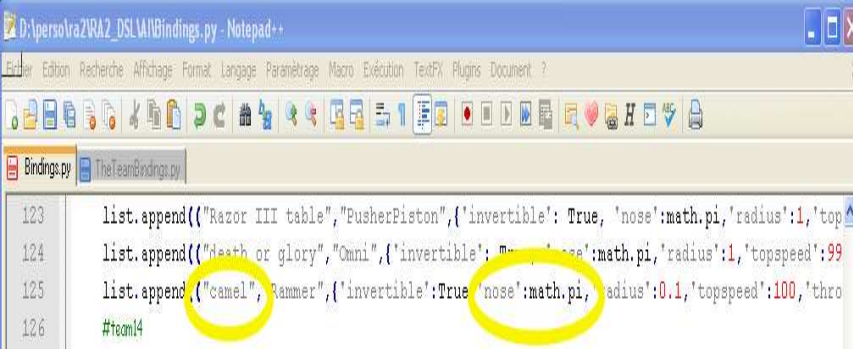
AI-Chart-2.3.doc 2009/06 – GameTechMods-Robot Arena
4/ exit RA2. under your AI directory, select your team and save the bot that you will erase :



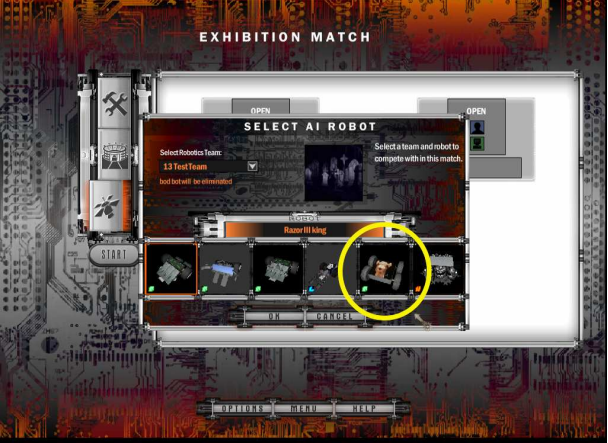
5/ move the bot you have create from RA2/Robot designs to AI/target team. The system should ask you if you really want to erase the bot with the same name (you will) :



6/ add your bindings in AI/bindings.py. It is not necessary to delete the previous line :



7/ launch RA2, enjoy life and drink ... whatever you want :



Create a new AI-Team

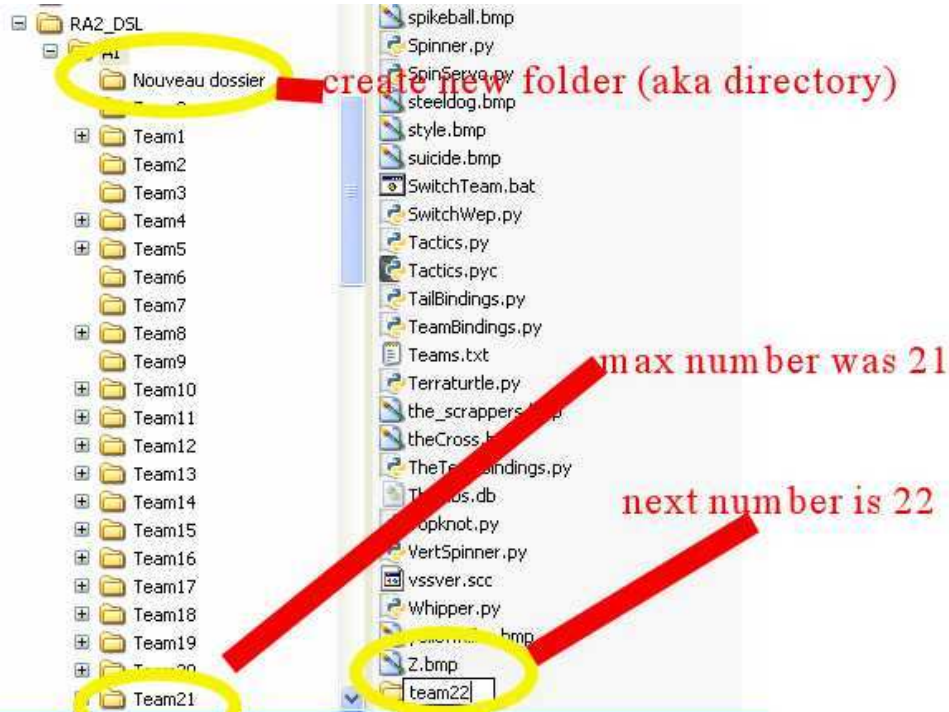
This time, you do not want to erase an existing bot in a team, but you would like to create you own one.
Steps overview :

- 1/ create your bots, export them as bot0,bot1, bot2, ... to bot 5. (a team should have only one bot).
- 2/ create a directory named TeamXXX under RA2/AI
- 3/ add your bindings in RA2/bindings.py
- 4/ copy your bots from robot design to you own team directory (aka TeamXXX)
- 5/ modify the file AI/teams.txt to declare your own team.

In this section, we only focus on steps 2 and 5.

Step 2 : create a directory named teamXXX

Under your RA2/AI directory notice the max number of team, create a new directory with name 'teamXXX' with XXX the max number + 1 :



Step 5 : modify the file AI/teams.txt

The file AI/teams.txt describes for each AIed team, its name, motto, number of bots and its icone.
Other lines describe their state in tourney. Do not bother about them.
What you have to do is to add a set of lines at the end of teams.txt. This set is like this (green words have to be changed with your values) :

```
index XXX the max number of team
The name of your team
Its motto
AI/name of the icone you have previously place under RA2/AI
Robots: 0 1 2 3 4 5 (which means you have 6 bots, it should have been 0 1 4 if you have only defined bot0,bot1 and bot4)
Robot In Event: -1
0
16
16
0
0
0
0
0
0
100000
```

```

true
0
false
0
-1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
0
0

```

Thus, for our new team, the last lines of the file AI/teams.txt may be (added lines in green) :

```

...
index 21
21 Team BBeans Champions
"winners"
AI\bbeans.bmp
Robots: 0 1 2 3 4 5
Robot In Event: -1
0
16
16
0
0
0
0
0
0
100000
true
0
false
0
-1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
0
0
index 22
22 Orcish Power
Nothing but rust and dust
AI/orc.bmp
Robots: 0 1 2 3 4 5
Robot In Event: -1
0
16
16
0
0
0
0
0
0
100000
true
0
false
0
-1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
0
0
0

```

Write you own AI

few words about inheritance and overall view

RA2 give a generic AI called SuperAI coded in RA2/AI/_init_.py. This AI defines many functions. This AI is specialized in named AI like Rammer, Pusher, Spinner etc which inherit from SuperAI. This means that each specialized AI only have to specify its particularity from the generic AI.
While writing a new AI, you just declare the original part of your specialized AI. However, a few constraint must be fulfilled : you must at least declare the name of your new AI, the way to initialized it, and record it in the list of available AI.

As usual, indentation and case sensitive constraints must fulfilled.

The skeleton of a AI script must be like that :

```
from __future__ import generators
import plus
import AI
from AI import vector3
import Arenas
import Gooey
import math
import Tactics

class MyAIName(AI.SuperAI):
    "MyAIName description"
    name = "MyAIName"

... list of functions ...

AI.register(MyAIName)
```

The name of the file must not necessary be the same as the name of the AI but we truly recommend that good practice.

Understanding the main functions of specialized AI.

Here, we present the main function of a AI, their role and what you should use them for. If you do not overwrite them, you will add the standard comportment of SuperAI.

Function name	Role	What to do with	remarks
def __init__(self, **args)	Initialize the AI, its attribute Declare the primary tactics	Parse parameters from bindings.py Declare specific attributes	Must always contains the line <i>AI.SuperAI.__init__(self, **args)</i> Which initialize the generic information of the AI.
def Activate(self, active):	Once All the bots in an event have been built, this script is called after the count down. If the attribute ‘activate’ is true, you know that all the components of the bot have been created At the time this function is called, the screen with the arena is displayed, so information about debugging text zone are also defined here.	Register your smart zone Do work that deal with component ID (ie parse the components to retrieve the Id of a specific one).	Must always call the SuperAI : <i>return AI.SuperAI.Activate(self, active)</i>
def Tick(self):	This function is called 8 time each second.	Short time consuming job, for instance :	Must always end with <i>return AI.SuperAI.Tick(self)</i>

		Calculate enemy range and activate controllers	
def InvertHandler(self):	Function called when the bot is inverted	Call Srimech or other self –upright mechanism of your own	
def LostComponent(self, id):	Function called when a component is lost : usually remove the component Id from the list called ‘weapons’ in the bindings.py . Once this list is empty, change the default tactic to ‘Pusher’	ID : once you lost some weapon, you may require your bot to act like a whipper, a pusher, activate secondary weapon, etc.	Unless you correctly filled the ‘Weapons’ parameter with real ID of weapon component, your list will never be empty and this function usually does nothing.
def DebugString(self, id, string):	Write information on the screen	debug	To activate the debug information while the bot is activated, you must declare thoses lines in your _init_ function : <i>self.debug = True</i> <i>AI.SuperAI.debugging = True</i>
def SmartZoneEvent(self, direction, id, robot, chassis):	Function called when a bot enter in one of your custom zone	Depending of which smartzone is activated, usually trigger a weapon	Smartzone must have been record in the <i>activate</i> function.

Init

In the `_init_` function you declare all the variable you will use further. Default declaration may be seen in the `AI/_init_.py` script.
Before all, call the standard initializations of SuperAI. If you do it later, it may overwrite the job you’ve done.
Typical initialization are :

- Name of controllers
- Name of customZones
- (at last) default tactic

Initialization may be done by defining constant :

```
#declare the name of the custom zone :
self.zone = "weapon"
```

It may be also done by parsing the parameters in the `bindings.py`

```
#read the value of the ‘range’ parameter if defined :
if 'range' in args:
    self.spin_range = args.get('range')
```

Tactics are behaviors that are activated when particular events or context appears. the SuperAI define standards values :

```
self.tactics.append(Tactics.Invert(self, self.InvertHandler()))
self.tactics.append(Tactics.Unstuck(self, self.StuckHandler()))

if plus.getGameType() == "TABLETOP":
    self.tactics.append(Tactics.AvoidEdges(self))
    self.tactics.append(Tactics.PushOffEdge(self))
elif plus.getGameType() == "KING OF THE HILL":
    self.tactics.append(Tactics.Dethrone(self))
    self.tactics.append(Tactics.Reign(self))
```

Default tactics are defined by adding them in a ‘private list’ : You may add more than one tactic. The system will choose for you which one is the best to apply... but its choice should not have been yours, better not add to much tactics.

```
# add default tactic : engage
self.tactics.append(Tactics.Engage(self))
```

For something else than pusher or rammer, select the *Engage* tactics.
Below an array of known tactics :

Name
Dumbcharge

Charge
Shove
Engage
Reorient
Invert
Unstuck
AvoidEdges
PushOffEdge
Dethrone
Reign

Creating new tactics may be done in the file tactics.py. It’s a hard work I do not investigate yet, so I won’t talk much about. Typically a tactic contains 3 functions : the *_init_* to initialize it, the *evaluate* to know between 2 tactics which on is the best to do and the *execute* which is the tactic by itself.

Activate your Bot

Once your bot have been build, the arena displayed on the screen, you may do additional job on the activate function. At least, if you’ve got some, register your custom zone here, and call the SuperAI *Activate* function. :

```
def Activate(self, active):
    if active:
        self.RegisterSmartZone(self.zone, 1)
    return AI.SuperAI.Activate(self, active)
```

We talk about custom zone, later in this document.

Deal with controllers

There are two kind of controllers : analog and button.

Controller type	Command	Effect
Button (used for piston and spin motor)	self.Input(<i>controllerName</i> , 0, 1)	Fire or turn clock
	self.Input(<i>controllerName</i> , 0, 0)	Stop it
	self.Input(<i>controllerName</i> , 0, -1)	Fire or turn counterclock
Analog (used for driving, servo piston and motor, special AI)	self.Input(<i>controllerName</i> ,0, <i>speed</i>)	turn clock, move forward
	self.Input(<i>controllerName</i> ,0, 0)	Stop it
	self.Input(<i>controllerName</i> , 0, - <i>speed</i>)	Turn counter clock, move backward

Speed for analog command may go from 0 to 100

ControllerName Usage

Controller name may be hard-coded :

```
self.Input(“Spin”, 0, 1)
```

You may also declare them in the *_init_* function then reference them :

```
def __init__(self, **args):
    AI.SuperAI.__init__(self, **args)
    self.trigger2 = ["Srimech"]
    # name of srimech controller may be define in bindings.py
    if ‘srimech’ in args: self.triggers2 = args[‘srimech’]
    ...
def InvertHandler(self):
    # fire weapon once per second (until we're upright!)
    while 1:
        self.Input(self.trigger2, 0, 1)

        for i in range(0, 8):
            yield 0
```


Where to use those commands

Typical usage of controller command are made in thoses function : Tick (),InvertHandler (),WhipAround(), WhipBackAndForth(), SmartZoneEvent()

Know the current status

You may wonder which is the current status of a controller : here is a sample usage :

```
currentStatus = self.GetInputStatus("Spin", 0)
```

for a spinner it may be 1,0 or -1

Deal with Custom zones

Custom zone must first be registered in the Activate function. You may hard-code their name or declare them in the _init_ function. Here is an instance of hard coded registration of 2 custom zone associated with the ID and 2 :

```
def Activate(self, active):
    if active:
        # standard smart zone :
        self.RegisterSmartZone("weapon1", 1)
        self.RegisterSmartZone("weapon2", 2)
    return AI.SuperAI.Activate(self, active)
```

Once an enemy enter your custom zone the SmartZoneEvent() function is called. You know which of your smart zone have is concerned by looking at the parameter ‘id’.

In the sample below, if enemy in zone 1 -> fire weapon. If enemy in zone 2 -> turn

```
def SmartZoneEvent(self, direction, id, robot, chassis):
    if id == 1:
        if robot > 0:
            if direction == 1:
                # Fire weapon 1 :
                self.Input("Fire", 0, 0)
    elif id == 2:
        if robot > 0:
            if direction == 1:
                # turn Bot :
                self.Input("LeftRight", 0, 100)

    return True
```

Other functions

To build an AI, you need to overwrite some of the common function describe above. To do this, you have few commands that helps you to act or make good choice. Here is a short array of those command.

Command	Role	Sample usage or file where you may found a useful sample
self.GetNearestEnemy()	Return the Id of the enemy and its range	Omni.py : def Tick()
self.GetNumComponents()	Give the number of components	Here is a sample code to add in the Activate Function : it build a ‘weapon list’ with only the Id of weapons : def Activate(self, active): if active: goon = 1 i = 0 self.weapons = []
self.GetComponentType(i)	Give the kind of component is the one with Id i	

		<pre>while goon == 1: if i == self.GetNumComponents(): break currentType = self.GetComponentType(i) if currentType == "Weapon": self.weapons.append(i) i = i+ 1 return AISuperAI.Activate(self, active)</pre>
self.GetMotorAngle(Id)	Give the current angle of a servo motor with the given Id	
range = self.GetDistanceToID(robot_id)	Give the distance to a given robot identified by its Id	
GetPath(self, from_world, to_world) IsStraightPathClear(self, from_world, to_world) DriveToLocation(...) plus.getLocation(IdRobot)	Travel information	See Tactics.py for usage of theses function
RobotInRange(Id)	Return true if robot in range of our weapon	See Tactics.py for usage. Spiner.py re-defines this function : good tutorial
CheckSensors	Iterate on custom zone to know if there is an enemy, call SmartZoneEvent.	See __init__.py
self.ai.GetSpeed()	Get current speed	See Tactics.py for usage.
angle = self.ai.GetHeading(False)	Get current angle	See Tactics.py for usage.
self.ai.GetEnemies()	Return an array of enemies Id	See Tactics.py for usage.
self.ai.Throttle(100)	Go full power	
damage = self.GetLastDamageReceived()	Return an array where : damage[3] : the robot Id that hurt us damage[2] :the time elapsed damage[1] : amount of damage damage[0] : Id of Weapon (<i>information need validation</i>)	Spinner.py
Damage = self.GetLastDamageDone	As above but what we’ve done	__init__.py
self.GetID()	Get Id of our robot	
self.GetComponentHealth(i)	Give current state of component of Id i	
enemyHead = self.GetHeadingToID(enemyId, False)	Return angle toward enemy	
self.ai.GetHeading(True)	Return current angle	
<i>To complete</i>	<i>To complete</i>	<i>To complete</i>

Trouble Shooting

- **RA2 crashes when launching** : problem with AI/teams.txt or syntax in bindings.py (lack of parenthesis, bad indentation)
- **RA2 crashes when starting a fight** : component not found / component of bot (coming from outer space pack) have not been installed / see below ‘Bindings.py commons error. /
- **RA2 crashes after count down (3-2-1) in a match** : AI is found, but it is not correctly written (case you’ve create your own)
- **Bots don’t move after count down (3-2-1) in a match and than, RA2 crashes** : AI is found, is syntactically correct but use variable that have not been declared of initialized before (problems of upper/lower case – variable declared in a block and referenced outside that block).
- **Black screen when you switch off your computer or your mother disconnected the electrical outlet with a Hoover** :

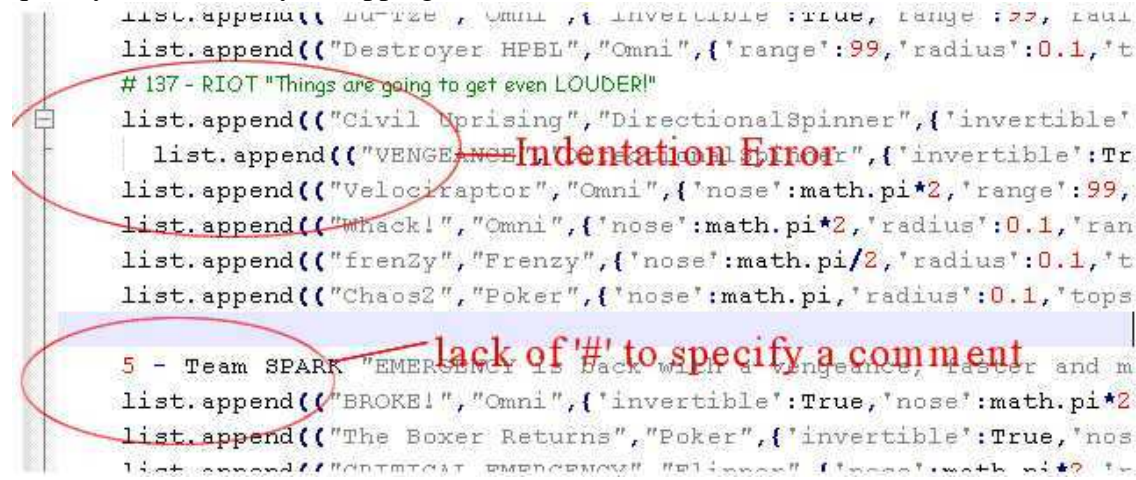
That's normal, do not worry, and tell you it should be worst (listen to the song from the toy dolls 'my girl with a Hoover').

- **Bindings.py common errors :**

- case sensitive problems
- indentation is not good
- Spaces in name of a bot
- Parenthesis unmatched
- duplicate bot names in bindings.py : the AI take one of the definition while you try to correct the other one
- unknown AI

Tools

We encourage the use of the free text editor Notepad++ which gives clever information on python files : with it, you could quickly find error in your tipping :



```
list.append(("DUTIE", "Omni", {'invertible': True, 'range': 99, 'radius': 0.1, 't
list.append(("Destroyer HPBL", "Omni", {'range': 99, 'radius': 0.1, 't
# 137 - RIOT "Things are going to get even LOUDER!"
list.append(("Civil Uprising", "DirectionalSpinner", {'invertible':
list.append(("VENGEANCE", "DirectionalSpinner", {'invertible': Tr
list.append(("Velociraptor", "Omni", {'nose': math.pi*2, 'range': 99,
list.append(("Whack!", "Omni", {'nose': math.pi*2, 'radius': 0.1, 'ran
list.append(("frenzy", "Frenzy", {'nose': math.pi/2, 'radius': 0.1, 't
list.append(("Chaos2", "Poker", {'nose': math.pi, 'radius': 0.1, 'tops

5 - Team SPARK "EMERGENCY is back with a vengeance, faster and m
list.append(("BROKE!", "Omni", {'invertible': True, 'nose': math.pi*2
list.append(("The Boxer Returns", "Poker", {'invertible': True, 'nos
list.append(("OPTIMIST EMERGENCY", "Elixxor", {'nose': math.pi*2, 'r
```

Indentation Error

lack of '#' to specify a comment