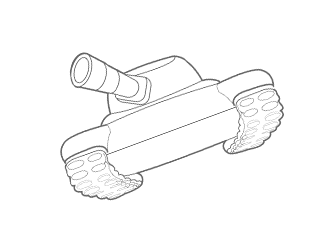
WarBots

Revised June 21, 2002

September 12, 2022



# Introduction

In WarBots, you write a Python program to control a virtual robot. Up to four robots compete in a battle to the death. All robots are identical with the exception of the control programs and option settings.

The WarBots package provides a simple programming interface to your robot, as well as some sample robots that you can use as programming examples and as competitors.

Each robot can move around, search for other robots, and try to destroy them.

There are automation features that you can use to get up and running very quickly with almost no code, or you can carefully program every detail of your robot’s behavior.

# 

**Table Of Contents**

[Introduction](#_mbbazgjsy2rv)

[Overview: Arena And Robots](#_z2ihbspmk0i1)

[Arena and Coordinates](#_njd42gy7pqi4)

[Robot](#_f0w9hh2se0ji)

[Drive System](#_1nte3m1gbjrx)

[Cannon](#_r5zc6c3brc20)

[Radar](#_anf85ufhmibg)

[Radar Detector](#_22ykpld1392v)

[Armor](#_7xnftkbkhyvz)

[Quick Start](#_7azwvuf1if4s)

[Programming Your Own Robot](#_9p3f8b6khra4)

[setup()](#_q2jy5syna7z7)

[ping()](#_6czgtemm5g5i)

[move()](#_bfedwfggmbl0)

[Drive System Details](#_7hzhsqxm1ctb)

[Radar Details](#_l66onljnpag2)

[Radar Accuracy](#_rdhp84jndjnx)

[Cannon Details](#_v1aeg54gsgiu)

[Armor](#_eks71xsta7bw)

[Running a competition](#_vxd9hg7ord2m)

[Starting Position](#_dnqnr13owqej)

[Robot Programming Commands](#_nl7hfd6eerl5)

[Action Methods](#_foyc9i47inbs)

[mybot.drive (speed, direction)](#_beiakv7hprzb)

[mybot.fire (dir, range)](#_3dmo1eec9wgp)

[mybot.scan(dir, resolution)](#_r6byqpw88k8z)

[Data Methods](#_qvaghbcwvvha)

[mybot.x()](#_he27bt3gerg1)

[mybot.y()](#_d8fbesaf1b4s)

[mybot.health()](#_j0szjfpb8gxa)

[mybot.heat()](#_55j143fuqae8)

[mybot.speed()](#_6j1z7zovtuk)

[mybot.direction()](#_8zote1mp9l8w)

[mybot.dsp()](#_b683b8eppyc6)

[mybot.index()](#_cn2x61to8ueb)

[mybot.time()](#_j5ai8dejve2f)

[Utility Methods](#_v85ipyhyn95c)

[mybot.post(“message”)](#_d3awku1cgnz1)

[mybot.set\_name(“name”)](#_qhq8otbo2row)

[mybot.set\_autopilot()](#_e5uutw5iu9h1)

[mybot.set\_autoscan()](#_86sxfslkn42e)

[mybot.set\_armor(value)](#_ybtfro184r9h)

[mybot.set\_scan(value)](#_6n7hlub6eq7u)

[General Programming Considerations](#_c6jqwruf59rb)

[Time](#_ysj97inl537y)

[Armor](#_art74dc3uzi)

[Diminishing Returns](#_ufm4nxeea3mk)

[Debugging](#_hstu607slatz)

[Acknowledgements](#_b89ook4zd4f7)

# 

# Overview: Arena And Robots

## Arena and Coordinates

Robot battles take place in a 1000 meter square walled arena.

The X axis runs horizontally, with the left edge being 0 and the right edge being 999.

The Y axis runs vertically, with 0 at the bottom.

Directions start with 0 pointing to the right and increase counterclockwise. Legal directions are 0 to 359 inclusive.

Each robot’s position is expressed as X and Y axis values, and its current direction (or heading) is a number between 0 and 359.

## Robot

Robots are tracked vehicles. Each robot has the same equipment:

* Drive system with overheat and tip-over protection
* Cannon that fires explosive shells
* Radar to detect other robots
* Built-in GPS system to determine the robot’s exact location, heading, and speed
* Motor temperature sensor
* Health readout
* Radar detector to tell when it has been scanned by another robot

In addition to the standard equipment, you can also choose the amount of armor that you want and the accuracy of your radar. Both of these choices have trade-offs.

All robots are the same. The only differences are the settings you choose and the program that you write.

### Drive System

You can command your robot to travel in any direction at any speed up to the robot’s maximum possible speed, which depends on the amount of armor.

However, at high speeds the motors begin to overheat. If the motor overheats, the robot is restricted to a lower speed until it cools down. There is a temperature sensor in the robot, so you can check your motor temperature.

The drive system also has an automatic protection mechanism to prevent tipping over. If you try to turn too sharply at too high a speed, the drive mechanism will disengage and the robot will coast to a stop in the current direction and will not move until a new drive command is issued.

### Cannon

Each robot has a cannon with a maximum range of 700 meters. The cannon fires an explosive shell which causes damage to any robot within 40 meters of the explosion. Shells travel at 200 meters per second, so it may take as much as 3.5 seconds for the shell to reach the target coordinates. The cannon has perfect accuracy: the shell will explode precisely where it’s aimed. Each shot heats the barrel, which affects the accuracy of the radar.

### Radar

Each robot has a radar which can be used to detect other robots. The radar can be pointed in any direction. The radar detects enemy robots in a wedge-shaped area up to +/- 10 degrees from the radar direction. The radar reading has errors based on the width of the scan and the heat of the cannon barrel.

### Radar Detector

Each robot has a detector that can tell when it is scanned by another robot. The radar detector will automatically execute your ‘ping()' method, passing you the ID of the robot that scanned you. There is no way to determine the distance or heading to the robot that scanned you.

### Armor

Each robot has armor that can reduce the damage from nearby explosions. You can select the amount of armor. More armor increases protection, but reduces the robot’s maximum speed.

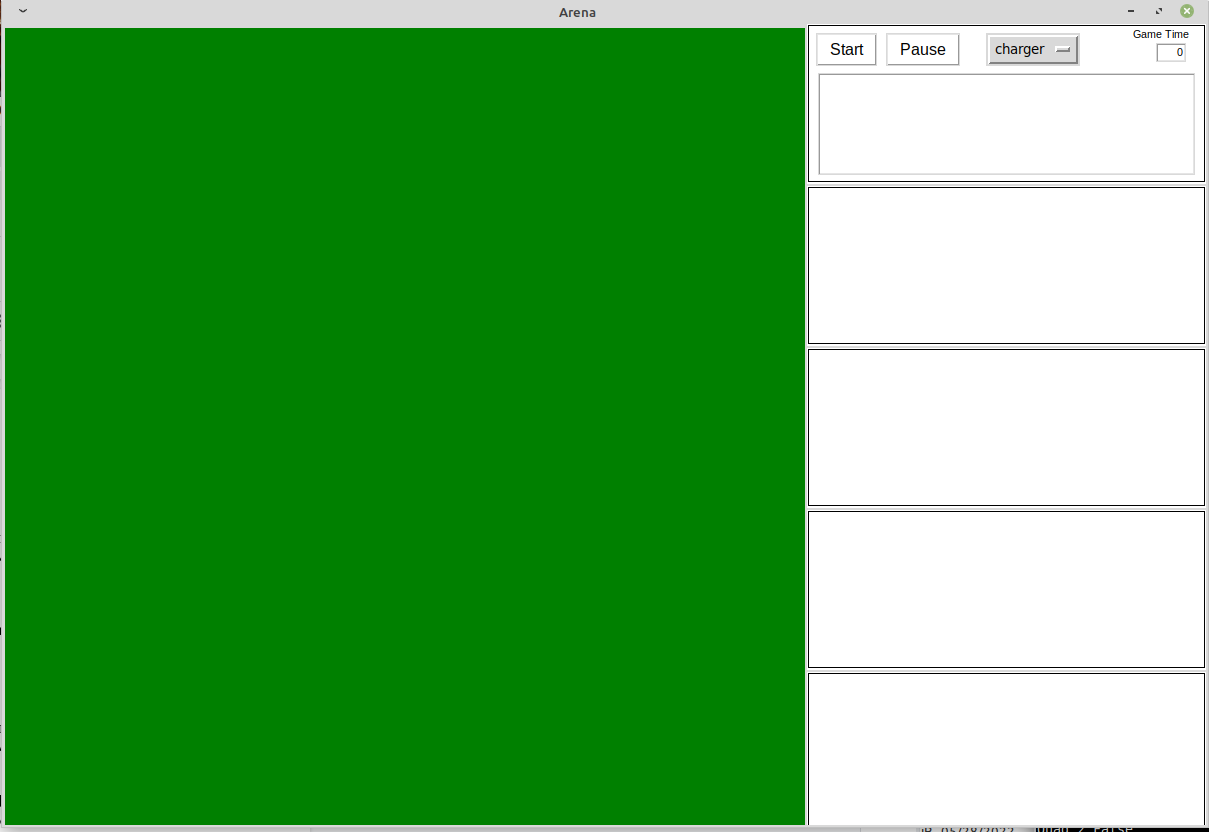
# Quick Start

Make sure you have Python 3, the ‘Warbots’ package, and optionally PyCharm installed - see the ‘Installation’ section of this document. Make note of the directory where the Warbot package is located.

Start a tournament. You can do this one of three ways:

1. Run the ‘arena’ program
2. Edit your robot using pycharm and click the green ‘run’ icon
3. Start any robot from the command line by typing ‘python <robot-file-name>’

This will open a window with the tournament arena and status panels. If you used option 2 or 3, the first robot will already be positioned in the arena.



Add robots. You’ll need at least two, and you can have up to four. In the top right panel, there’s a pull-down menu listing all of the available robots. Choose robots to start them. You can choose the same one more than once. After you’ve selected your competitors, click the ‘Start’ button.

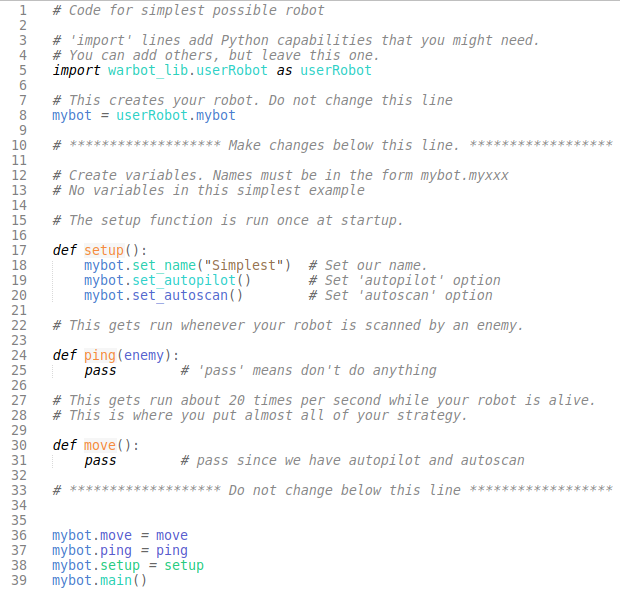
# Programming Your Own Robot



Since your robot is a virtual creation, you won’t have the opportunity to work on the physical components. Instead, you create the program which controls his actions. Since the robot has many limitations, you’ll need to be clever to get the best performance out of him. In fact, you may find it a challenge to keep him from running into the walls!

To make your own robot program, you’ll need to use the Python programming language and a text editor or the PyCharm program. Copy the sample robot ‘simplestBot.py’ and give it your own name. The file name must end in ‘Bot.py’, so ‘firstBot.py’ would be a valid name.

The sample robot that you have copied contains everything you need. In fact, you can run it just as it is. Let’s take a look at the code for this robot. In this example, gray text is comments that explain the code but don’t affect robot behavior. There are sections at the top and bottom that don’t change and can safely be ignored.



The center section has a place to create variables and three functions that define the behavior of the robot. These functions - *‘setup’, ‘ping’*, and ‘*move’* are where you will do your programming.

## setup()

This function is run when the robot is first started, before the tournament begins. It’s where you can define the name and configuration and establish any initial values. In this example, the name is set and autopilot and autoscan are set.

## ping()

This is the ‘radar detector’. This function is run every time your robot is scanned by another robot. You don’t have to do anything as in this example. Just use the ‘pass’ command if you don’t want to take any action when you’ve been scanned.

## move()

This function is run over and over as long as your robot is alive. This is where you control the movement of your robot, scan for enemies, and fire your cannon.

# 

# Drive System Details

You can command your robot to travel using the ‘mybot.drive’ command. This command takes two values: speed and direction. Once the command is issued, the robot will travel at the commanded speed and direction if possible.

Speeds are expressed as a percentage of the robot’s maximum possible speed, which depends on the amount of armor. If you command your robot to travel at a speed of 100, it will move as fast as it can.

However, at speeds above 35%, the motors begin to overheat. When the motor temperature reaches 200 degrees, the robot is restricted to 35% speed until it cools down. There is a temperature sensor in the robot, so you can check your motor temperature.

The drive system also has an automatic protection mechanism to prevent tipping over. Large course changes are not allowed at high speeds.

| **Course Change In Degrees** | **Max Allowable Speed** |
| --- | --- |
| <= 25 | 100% |
| 25-50 | 50% |
| 50-75 | 30% |
| > 75 | 20% |

Any attempt to exceed these limits results in the drive mechanism disengaging. The robot will coast to a stop in the current direction and will not move until a new drive command is issued.

The slower the robot is moving, the faster it can turn.

| **Speed** | **Rate of turn (degrees per second)** |
| --- | --- |
| < 25% | 90 |
| 25 - 50% | 60 |
| 50 - 75% | 40 |
| > 75% | 30 |

If the robot runs into a wall it will stop immediately, suffering five units of damage.

# 

# Radar Details

Each robot has a radar which can be used to detect other robots. The radar can be pointed in any direction. The radar detects enemy robots in a wedge-shaped area up to +/- 10 degrees from the radar direction. Each radar scan takes time, during which nothing else can be done.

The radar will return the distance to the nearest robot in the scan beam area, subject to two error sources:

* The wider the radar beam, the more error. At the widest setting, a random distance error up to +/- 50 meters is introduced.
* Each shell fired from the cannon causes the cannon barrel to heat up. Barrel heat causes a random plus-or-minus error equal to the barrel heat.

## Radar Accuracy

You can specify the accuracy of the radar with a range of 0 to 100. A more accurate radar will provide better measurement of the distance to enemy robots, but each scan takes longer. Just as with armor, the default is 50.

The amount of error and the delay between scan depends on the scan accuracy setting:

| **Scan Accuracy Setting** | **Max Distance Based Error At 10°** | **Heat Based Error Per Degree** | **Delay After Scan** |
| --- | --- | --- | --- |
| 0 | 100 | 2 | 100ms |
| 50 | 50 | 1 | 200ms |
| 100 | 25 | .5 | 400ms |

At barrel temperatures above 35, the radar always returns a range of 0.

The radar can also determine the identity of the last scanned robot.

## Cannon Details

Each robot has a cannon with a maximum range of 700 meters. The cannon fires an explosive shell which causes damage to any robot within 40 meters of the explosion. Shells travel at 200 meters per second, so it may take as much as 3.5 seconds for the shell to reach the target coordinates.

Damage to each nearby robot is based on its armor level and distance from the explosion. A detailed explanation can be found in the ‘Armor’ section of this document, but this table shows typical values.

| **Meters From Explosion** | **Damage at 0% armor** | **Damage at 50% armor** | **Damage at 100% armor** |
| --- | --- | --- | --- |
| 5 | 50% | 22% | 12% |
| 10 | 25% | 11% | 6% |
| 20 | 12% | 6% | 3% |
| 40 | 6% | 3% | 1.5% |

Each robot has an unlimited number of shells. Shells come in clips of four. Reloading takes four seconds, and installing a new clip takes twelve seconds.

Firing a shell heats the cannon barrel by 20 units. The cannon barrel cools by 2 units per second. Barrel heat affects radar performance as described in the radar section. There is no temperature sensor for the cannon barrel.

## Armor

You can specify the amount of armor with a range from 0 to 100. The more armor your robot has, the less damage it will have when shells explode nearby. However, the weight of the armor will reduce the top speed as well at the speed at which the robot can turn. The details are described in the ‘Armor Details’ section of this document. If you do not specify your armor level, the default is 50.

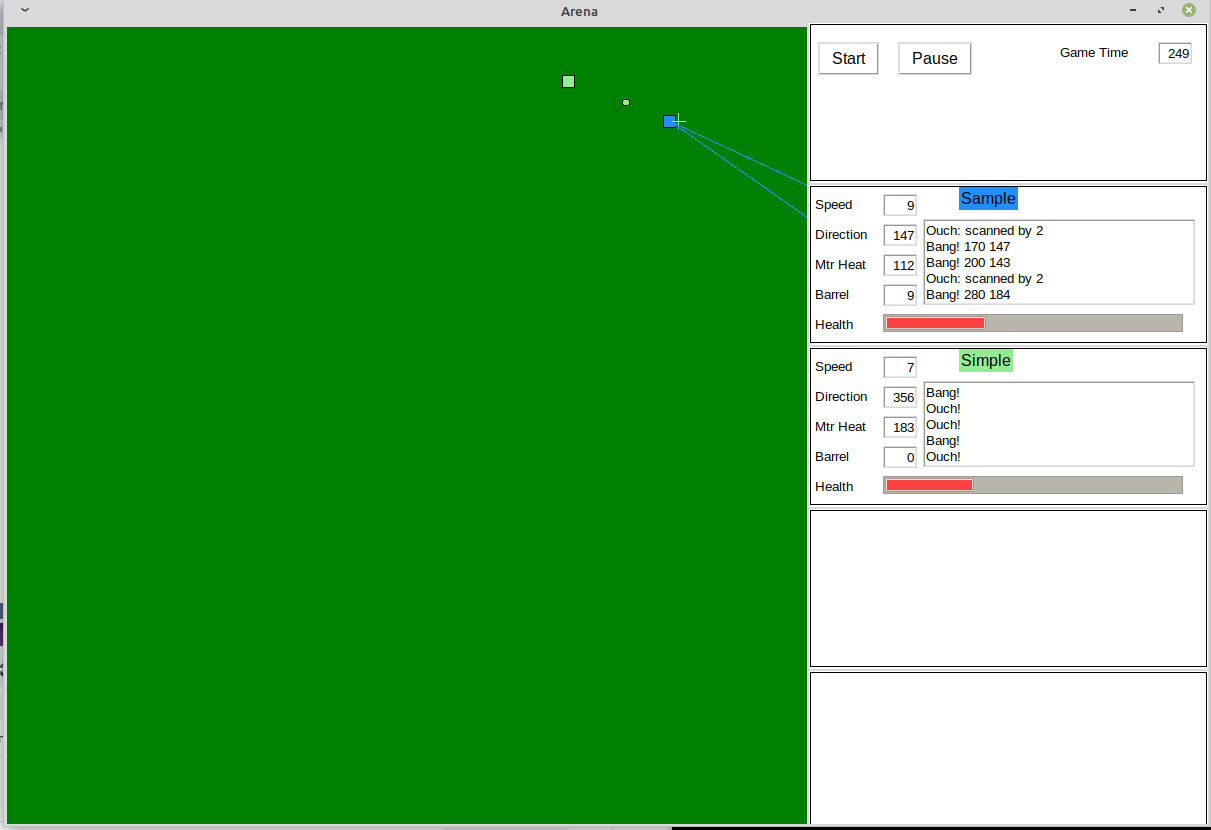
# 

# Running a competition

### Starting Position

Robots start in a random quadrant of the arena, and will always start at least 100 units away from the nearest edge. Robots start with 100% health, and are declared 'dead' when health drops to 0%. Health does not affect robot function - a nearly dead robot still functions exactly like a perfectly healthy one.

To run a competition, start the server program and two or more robot programs. The server provides a display of the combat arena and status panels for the game and for each robot.



This screen shot shows the arena on the left and the status panels on the right. There are two robots named ‘Sample’ and ‘Simple’. Each robot has a server-assigned color.

In the arena, the green robot ‘Simple’ has fired a shell that’s in flight and will explode just to the right of ‘Sample’, the blue robot. At the moment, the blue robot is scanning down and to the right.

The status panels to the right provide information about each robot. Sample’s name is shown at the top of its panel in the server-assigned color. This robot is traveling on a heading of 147 (which is approximately in the direction of the green robot) at a speed of 9 meters per second. The motor is at a temperature of 112 degrees, and the barrel is at a heat of 9. Health is shown as a red bar graph at the bottom.

# Robot Programming Commands

In your code, your robot is called ‘mybot’. There are several commands that you can use in your program. These are called ‘functions’ or ‘methods’ in the programming world. Some methods require values like speed or direction. These values are called ‘arguments’.

## Action Methods

The first group of methods tell the robot to do something. They return a value that you can use to check what happened.

### **mybot.drive (speed, direction)**

Command the robot to go at the specified speed in the specified direction. Returns the maximum speed that the robot is capable of based on motor temperature and commanded turn rate. The direction must be in the range 0 to 359, and speed must be between 0 and 100. This command requires 100 milliseconds to execute.

Example:



This will command the robot to travel at a speed of 100% at a heading of 45 degrees.

### mybot.fire (dir, range)

Attempt to fire a shell. If your robot is not busy reloading and the direction and range are legal values, a shell is fired. Dir must be in the range 0 to 359, and range must be between 40 and 700. The fire method returns:

0 if a shell was fired

-1 if no shell was fired

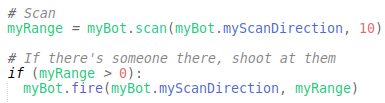
This command requires 100 milliseconds to execute IF a shell is fired, and 200 milliseconds if no shell is fired.

### mybot.scan(dir, resolution)

Use the radar to look for other robots. Scan at dir +/- resolution degrees. Returns the distance to the detected robot, or 0 if no robot is detected. Dir must be in the range 0 to 359, and resolution must be between 1 to 10.

This command requires 200 milliseconds to execute.

Example: Perform a scan 20 degrees wide centered at the direction specified in the myScanDirection variable. If a robot is detected, fire a shell at that same direction to the distance measured by the scan.



Note that in this case, the range contains an error between +50 and –50 meters because of the scan width. Under these conditions, a direct hit may not occur.

## Data Methods

The next group of methods give you information about the robot or battle environment.

### mybot.x()

Returns the horizontal position of the robot. The position will be between 0 (the left edge) and 999 (the right edge) of the arena.

### mybot.y()

Returns the vertical position of the robot. The position will be between 0 (the bottom) and 999 (the top) of the arena.

### mybot.health()

Returns the current value for the health of your robot. 100 is unharmed, and 0 is dead.

### mybot.heat()

Returns the current motor temp for your robot.

### mybot.speed()

Returns the current speed of your robot.

### mybot.direction()

Returns the current direction that your robot is traveling.

### mybot.dsp()

Returns the identity of the robot who last scanned you.

### mybot.index()

Returns the ID number of your robot (1-4) assigned by the server.

### mybot.time()

Returns the current time in seconds/fractional seconds. Maximum useful resolution is 50 milliseconds.

## Utility Methods

There are some methods that allow you to configure your robot and perform non-battle functions.

### mybot.post(“message”)

Display message on the status display for your robot.

### mybot.set\_name(“name”)

Set the robot's name.

### mybot.set\_autopilot()

Enable autopilot mode. This command can only be used in the setup() function.

### mybot.set\_autoscan()

Enable autoscan mode. This command can only be used in the setup() function.

### mybot.set\_armor(value)

Set the robot's armor level. Value must be a number between 0 and 100. This command can only be used in the setup() function.

### mybot.set\_scan(value)

Set the robot's radar accuracy. Value must be a number between 0 and 100. This command can only be used in the setup() function.

## General Programming Considerations

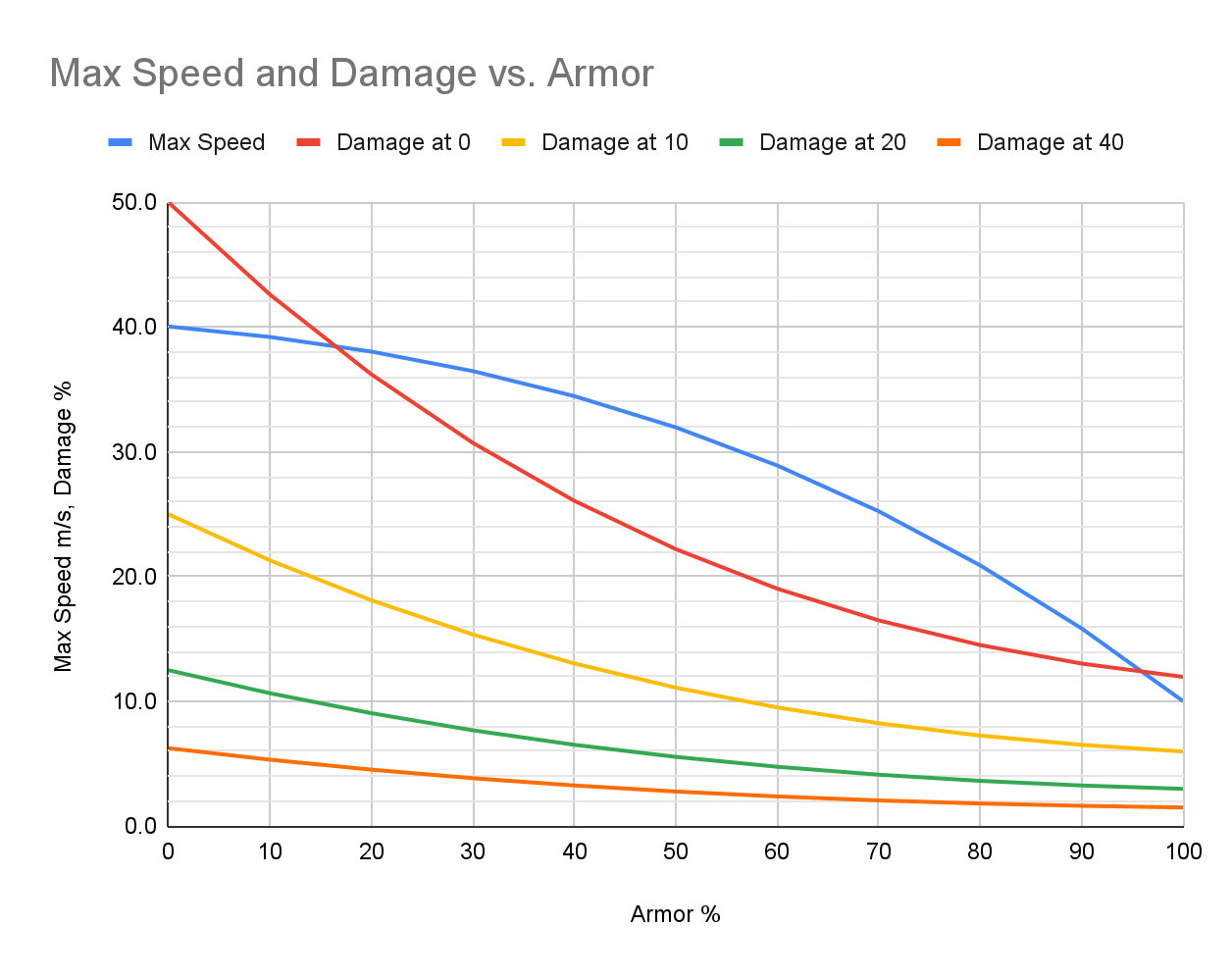
### Time

The server recalculates everything about 20 times per second, or about once every 50 milliseconds. However, your program will wait for commands that take time, such as scanning or firing shells.

# Armor

The amount of armor that you choose has two different effects. First, more armor means less damage from any nearby exploding shells. The chart below shows the amount of damage from explosions at 0, 10, 20, and 40 meters at armor levels from 0% to 100%.

For instance, the yellow line shows damage from explosions 10 meters away. At the left side of the chart, we see that with an armor level of 0%, the robot would suffer 25 units of damage. With 50% armor, damage would be about 11 units, and at 100% armor the damage would be about 6 units.



There is a tradeoff, though. Increased armor reduces the robot’s maximum speed. The blue line shows this relationship as well. At the left side, we see that a robot with 0% armor would have a top speed of 40 meters per second (m/s). In the middle, a robot with 50% armor would have a top speed of about 32m/s, and at the right side we see that a robot with 100% armor would have a top speed of only 10m/s.

## Diminishing Returns

Note that the relationship between damage and armor is not a straight line. The relationship between armor and speed isn’t a straight line either. This means that sometimes adding or removing a bit of armor doesn’t always make the same difference in either speed or protection.

For instance, changing from 0% to 10% armor makes a pretty small difference in maximum speed, but it reduces damage by quite a bit. On the other end, increasing armor from 90% to 100% doesn’t reduce damage much, but it reduces speed by quite a bit.

This effect is called ‘diminishing returns’. As you add armor starting with 0%, each bit that you add gives you less additional protection, but costs you more in lost speed. The first bit of armor is a pretty good deal, but at some point it’s no longer worth it.

### Debugging

There are several commands to support debugging. These commands are typically used when running your robot interactively. These commands do nothing unless the ‘Debug’ button is clicked on the server control panel.

mybot.Pause()

Pause the server. Same effect as clicking the ‘Pause’ button on the server control panel.

mybot.Run()

Same effect as clicking the ‘Run’ button on the server control panel.

mybot.Mark(x as integer, y as integer, c as long)

Make a mark on the arena at x,y in color c.

mybot.whereis(e) as Long

Returns the location of enemy robot e as a long integer in the form 1000x + y. For instance, if the enemy robot is at x=437, y=854 the method will return 437854.

mybot.bheat() as integer

Returns the barrel heat for your robot.

## Acknowledgements

The WarBots package is a programming game based on Tom Poindexter's TclRobots, copyright 1994,1996 Tom Poindexter (http://www.nyx.net/~tpoindex) which is in turn based on an ancient programming game called 'Core Wars'.

Installation

You’ll need Python 3 and tkinter

pip install tk

apt-get install python3-tk

Windows:

Install latest version of Python from the Windows store.

At command line, python -m pip install tk

Download PyCharm

<https://www.jetbrains.com/pycharm/download/#section=windows>

May have to set interpreter in PyCharm - button near bottom right corner.