Flinders University
College of Science & Engineering

COMP1102/8702 & ENGR1721

Computer/Engineering Programming 1

Topic Practical Manual
2017

Topic Coordinator: Dr Carl Mooney
Phone: 8201 5041
Email: carl.mooney@flinders.edu.au

Computer/Engineering Programming 1 – Practical Class 1

Software installation & fun with Robot World

What you will learn?

When you complete this practical you will

1. have the latest Java software installed on your computer
2. have the latest *jGrasp* IDE installed
3. have Java API installed
4. information on the Work Environment

Laboratory Sessions

Structure

Every week, starting from teaching week 1, students will be assigned sets of formative practical exercises to complete. If you do not complete all the exercises during your session in the lab, you can return to the labs and complete the work in your own time or work through the material at home (assuming that you have the appropriate equipment and software). The following table shows the schedule for checkpoints:

Practical:	P01	P02	P03	P04	P05	P06	P07	P08	P09	P10	P11	P12
Checkpoints:		1-5	6-10	11-15	16-20	21-25	26-30	31-35	36-40	41-45	46-50	51-55
Due end of week:		3	4	5	6	7	8	9	10	11	12	13
Additional Checkpoints:		CP 56	CP 57	CP 58	CP 59	CP 60						

In teaching weeks 3, 6, 9, and 12 you will need to complete a quiz. The quiz will be available online (at home or university, unsupervised) and you will be given credit accordingly, as outlined in the relevant topic SAM.

The quizzes will test your knowledge and understanding of the core material covered in the topic (lectures and prescribed textbook reading).

Attendance

You should attend your registered practical session each week. It should not be expected that you will (or should) always complete all lab exercises within your practical session some work to polish your programming skills outside of the lab sessions will be necessary. The time needed to complete laboratory exercises will depend on you prior preparation. To be well prepared for the practical you need to have a reasonable understanding of the material covered in lectures and workshops. It is crucial that you attend each lecture/workshop (or watch it on FLO, if you definitely cannot be physically in class) and complete the required reading/coding exercises at home (details will be provided in each lecture).

For this week you will be getting used to the Java programming environment which includes *jGrasp* Integrated Development Environment (IDE) with the latest release of Java SE. You will also explore some of the initial concepts covered in this weeks lectures.

Laptop/PC Software Download and Installation

It is expected that most students (if not all) will use their own laptops/PC's to practise programming for this topic. Computers in IST laboratories have pre-installed Java software which you are welcome to use but it may not be always possible/convenient to practice in the university labs due to time constraints (labs generally are not open late or on public holidays or weekends) and have a limited number of computers. To complete practical exercises on your laptop, or a PC at home, you may need to download and install some software. If you do not have your own PC/laptop at all or have already installed the latest *jGrasp* , JDK and Java API then you can now jump to the section on Robot World in this document.

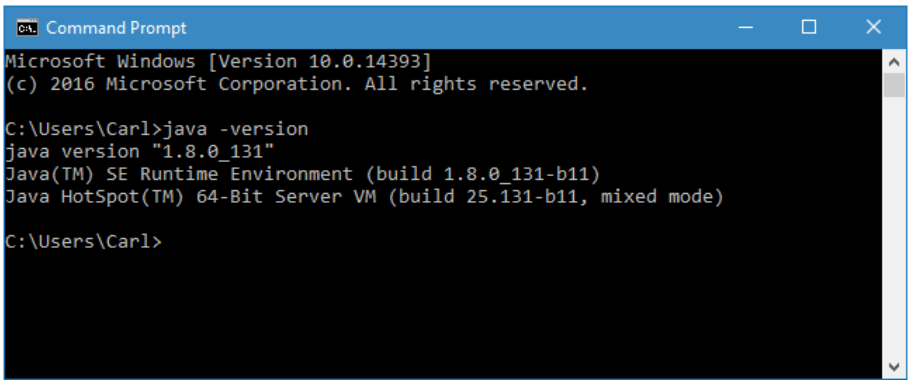
Note that the instructions below are quite detailed but you may still need some help during the software installation. If you are installing on your laptop it is best to bring it to the first laboratory session, in case you need some clarification.

Check what you need

To see whether you need to install Java software open the *Command Prompt* window (in Windows 10 this is located in the `All apps/Windows System` section) and execute the command

```
java -version
```

If Java software is properly installed on your machine you will see a screen similar to the one below.



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Carl>java -version
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)

C:\Users\Carl>
```

If the command `java -version` is not recognised then either Java is not installed on your machine or it cannot be accessed from the command line because the `PATH` variable pointing to java binaries has not been set up. You can fix this by going to the

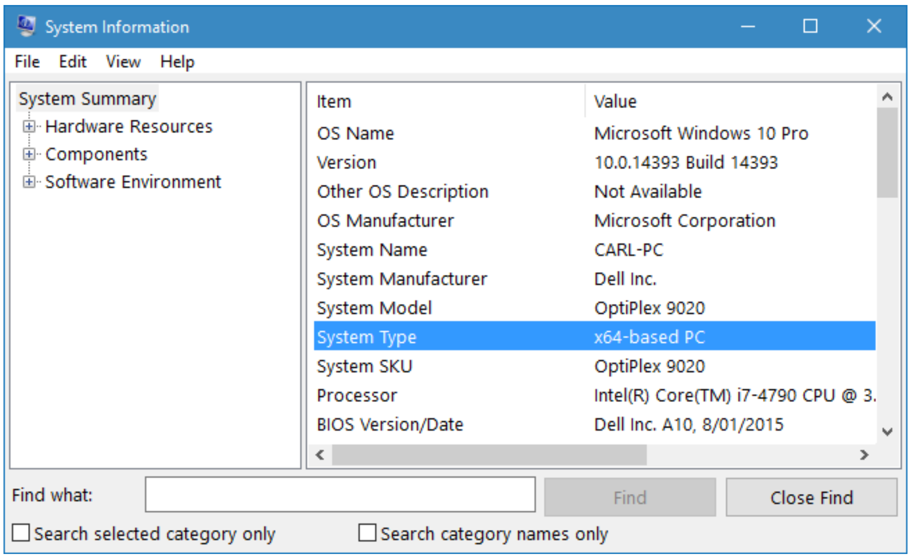
`Control Panel` → `System & Security` → `System` → `Advanced System Settings` → `Environment Variables`

and setting the `PATH` according to your system installation (most likely your `PATH` needs to point to C:\Program Files\Java\jdk1.8.0_121\bin or a similar location).

If you have a Java version earlier than 1.8 then you will need to download the latest one.

Java SE Installation

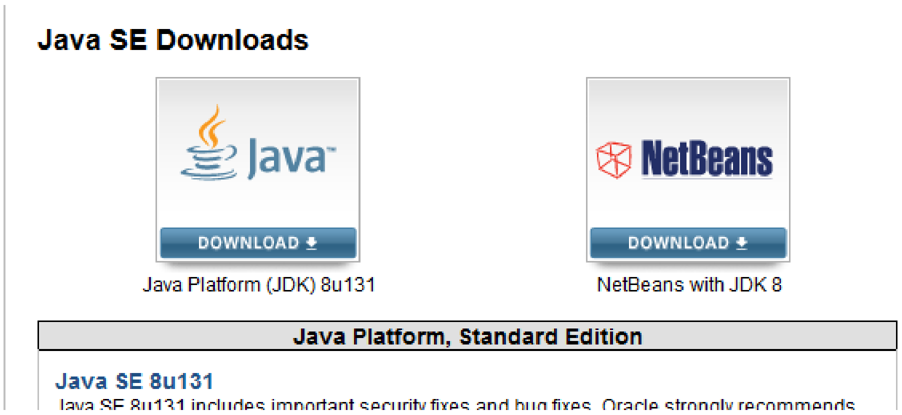
Go to the Oracle website: `http://www.oracle.com/technetwork/java/index.html` to install Java. There are different Java versions for different operating systems like Windows, Mac or Linux. To ensure which one runs on your computer check your system details by looking at the System Information. You should see something like



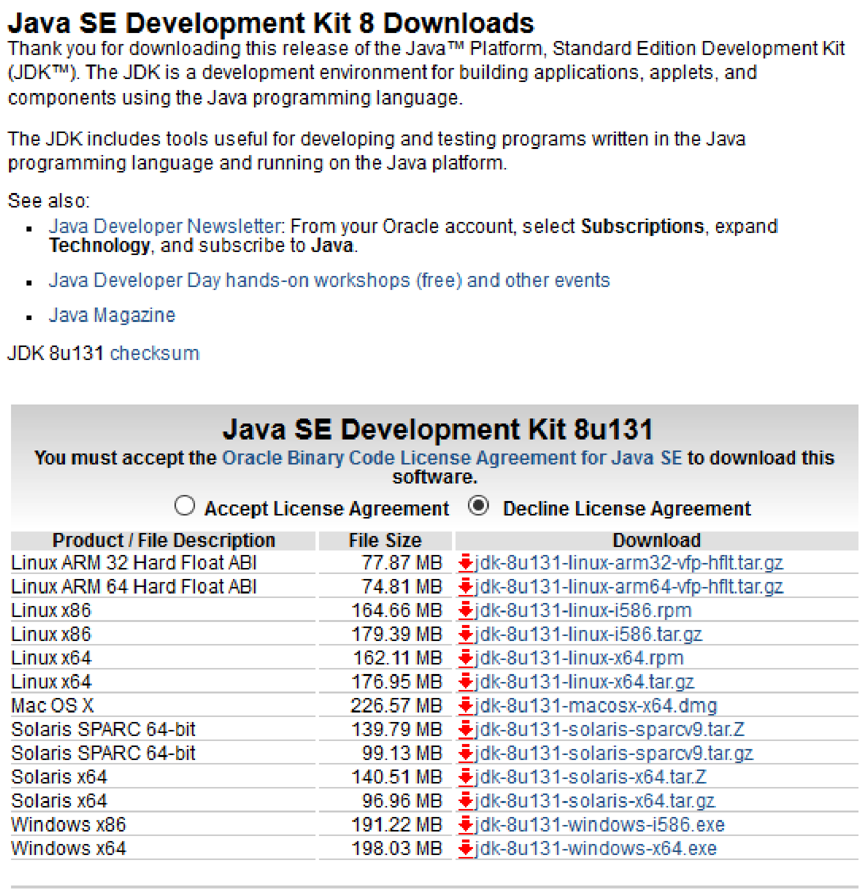
Item	Value
OS Name	Microsoft Windows 10 Pro
Version	10.0.14393 Build 14393
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	CARL-PC
System Manufacturer	Dell Inc.
System Model	OptiPlex 9020
System Type	x64-based PC
System SKU	OptiPlex 9020
Processor	Intel(R) Core(TM) i7-4790 CPU @ 3.
BIOS Version/Date	Dell Inc. A10, 8/01/2015

Identify the line *System Type*, it tells you which system is running your machine. E.g. x64 means that you need Windows x64 version (if you have a fairly recent PC/laptop with Windows 10 this is the most likely version).

On the Oracle web page, which you have opened in your browser, identify the section *Software Downloads* and click on `JavaSE` link. Now, you should see two icons: `Java Platform` and `NetBeans with JDK8 bundle`, like below.


Java SE Downloads
Java
DOWNLOAD
Java Platform (JDK) 8u131
NetBeans
DOWNLOAD
NetBeans with JDK 8
Java Platform, Standard Edition
Java SE 8u131
Java SE 8u131 includes important security fixes and bug fixes. Oracle strongly recommends

Select Java Platform package only. You should see:



Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 8u131 checksum

Java SE Development Kit 8u131

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement [✓ **Decline License Agreement**]

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.87 MB	jdk-8u131-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.81 MB	jdk-8u131-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.66 MB	jdk-8u131-linux-i586.rpm
Linux x86	179.39 MB	jdk-8u131-linux-i586.tar.gz
Linux x64	162.11 MB	jdk-8u131-linux-x64.rpm
Linux x64	176.95 MB	jdk-8u131-linux-x64.tar.gz
Mac OS X	226.57 MB	jdk-8u131-macosx-x64.dmg
Solaris SPARC 64-bit	139.79 MB	jdk-8u131-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.13 MB	jdk-8u131-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u131-solaris-x64.tar.Z
Solaris x64	96.96 MB	jdk-8u131-solaris-x64.tar.gz
Windows x86	191.22 MB	jdk-8u131-windows-i586.exe
Windows x64	198.03 MB	jdk-8u131-windows-x64.exe

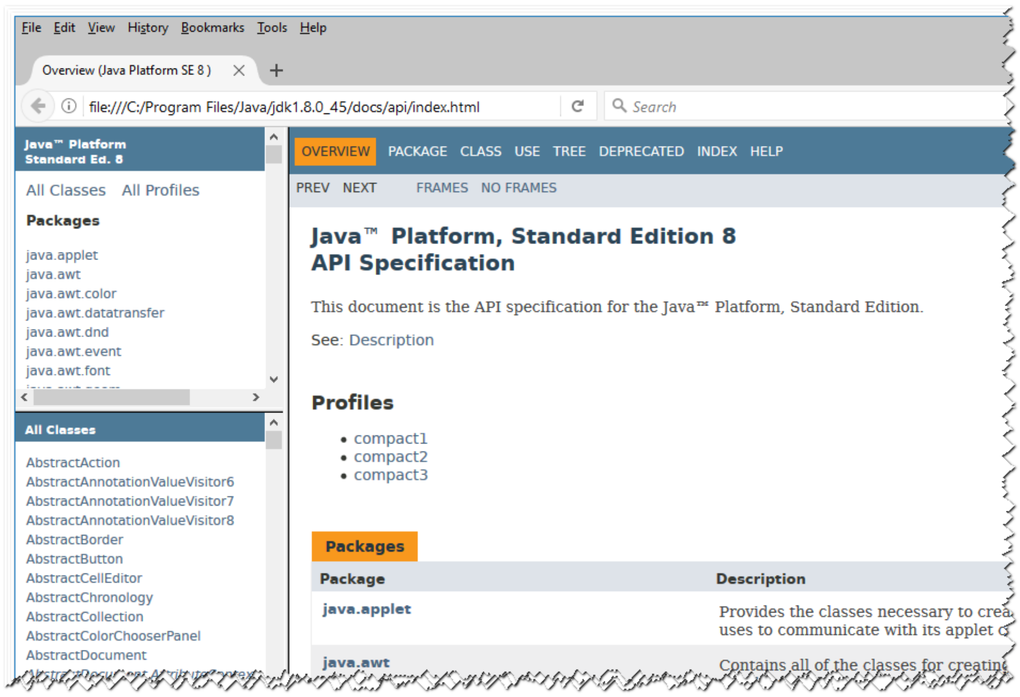
Accept the licence and download the right package for your system (most likely the last one on the list). From that point the installation should run smoothly (just click on the downloaded file). If you are stuck at any point ask your tutor for direction.

Java API Specification

When programming in Java it is very convenient to have the language documentation handy. This is contained in a package called the API (Application Programming Interface). It is a great idea to download it for future reference.

Go to the Oracle website: `http://www.oracle.com/technetwork/java/javase/documentation/jdk8-doc-downloads-2133158.html` and select `Java SE Development Kit 8u131 Documentation` from the list of available API's. Accept the License Agreement and download the `zip` file.

The installation is very simple, just unzip to the Java directory in Program Files. Once you have installed the package, place a link on your desktop for quick access to the documentation (as you may guess you will need it frequently). When you click on that link you should see a page like Note my path to the index file, if you used the default installation folder you may have a very similar path.


File Edit View History Bookmarks Tools Help
Overview (Java Platform SE 8)
file:///C:/Program Files/Java/jdk1.8.0_45/docs/api/index.html
Search
Java™ Platform Standard Ed. 8
All Classes All Profiles
Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
All Classes
AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractAnnotationValueVisitor8
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractChronology
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV NEXT FRAMES NO FRAMES

Java™ Platform, Standard Edition 8
API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

Packages

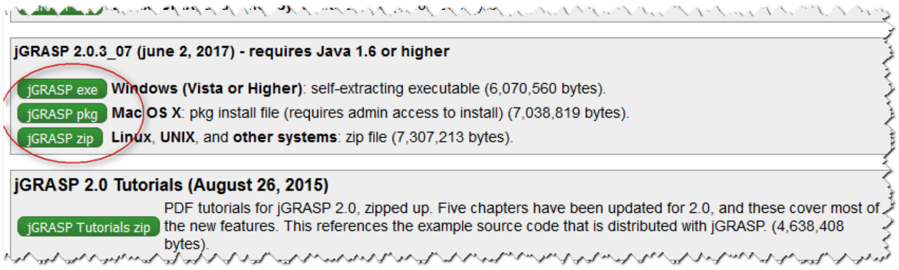
Package	Description
java.applet	Provides the classes necessary to crea uses to communicate with its applet c
java.awt	Contains all of the classes for creatin

jGrasp

Rather than create a large amount of overhead when learning to program, we will use a Integrated Development Environment (IDE), named *jGrasp* , which gives good visual cues on program structure, "friendly" compiler messages, and comes with less baggage than some of the more fully-featured IDE's such as "NetBeans" and "Eclipse".

This can be downloaded from: `http://spider.eng.auburn.edu/user-cgi/grasp/grasp.pl?;dl=download_jgrasp.html`. You can choose to fill out the survey, or not.

Select one of the options circled in the following figure, and also download the tutorial if you are not familiar with the product (it will be helpful).

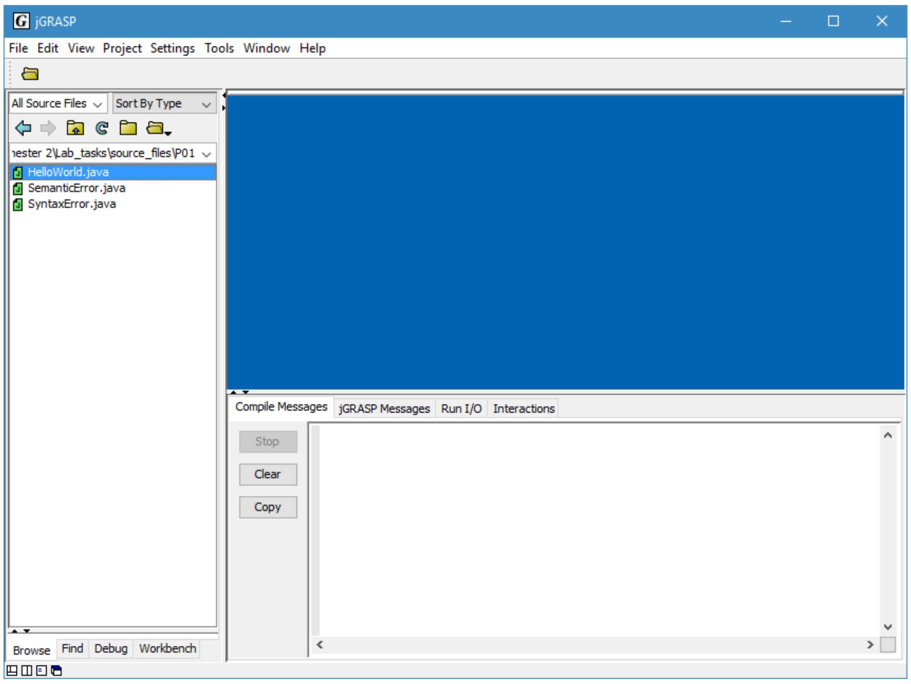

jGRASP 2.0.3_07 (june 2, 2017) - requires Java 1.6 or higher

jGRASP exe **Windows (Vista or Higher)**: self-extracting executable (6,070,560 bytes).
jGRASP pkg **Mac OS X**: pkg install file (requires admin access to install) (7,038,819 bytes).
jGRASP zip **Linux**, **UNIX**, and **other systems**: zip file (7,307,213 bytes).

jGRASP 2.0 Tutorials (August 26, 2015)

jGRASP Tutorials zip PDF tutorials for jGRASP 2.0, zipped up. Five chapters have been updated for 2.0, and these cover most of the new features. This references the example source code that is distributed with jGRASP. (4,638,408 bytes).

Once you have installed the product and opened it you should see something similar to the figure below:



Work Environment

One of the key issues that you should begin with is the development of an appropriate work environment. Losing files, accidentally deleting code, network issues, computer crashes are today's "The dog ate my homework" excuses. To avoid any loss of work you should clearly set up directories to contain your work. You should also develop the habit of regular backup of all your work. This backup should be off the main work computer. Using USBs and cloud storage are great ways to ensure your code is backed up if catastrophe occurs. As discussed in the lectures think about when, where and how many copies of your backups that you need to create.

You should begin by creating folders within your `U: drive` on the University machines. Your `U: drive` is your home drive on the University network and is where all your files will be saved. You should have a separate folder for each of your topics, ie: CP1 (or COMP1102), Fundamentals, ProfSkills, Maths1A, . . . If you work on your PC/laptop you should do similar things. Once you have your topic folders set up you should create sub folders for specific elements of your topics. You may need folders for tutorials, practicals, assignments, readings, group contacts, . . .

For the Computer Programming 1 folder you should have at least; Practicals and Lectures sub folders, again with convenient sub folders.

The Robot World: Its Simulator and Programming Language

Starting the Simulator

The simulator is started using the command line (basic form):

```
java -jar RobotWorld.jar source_file1 source_file2 ...
```

or by double-clicking on the `.jar` file.

The simulator creates three windows: the main window that displays the robot world, a control window that contains buttons and sliders to control the starting/stopping of robots and a trace window that contains a frame for each robot (trace window shown overlaying the main window) see Figure 1:

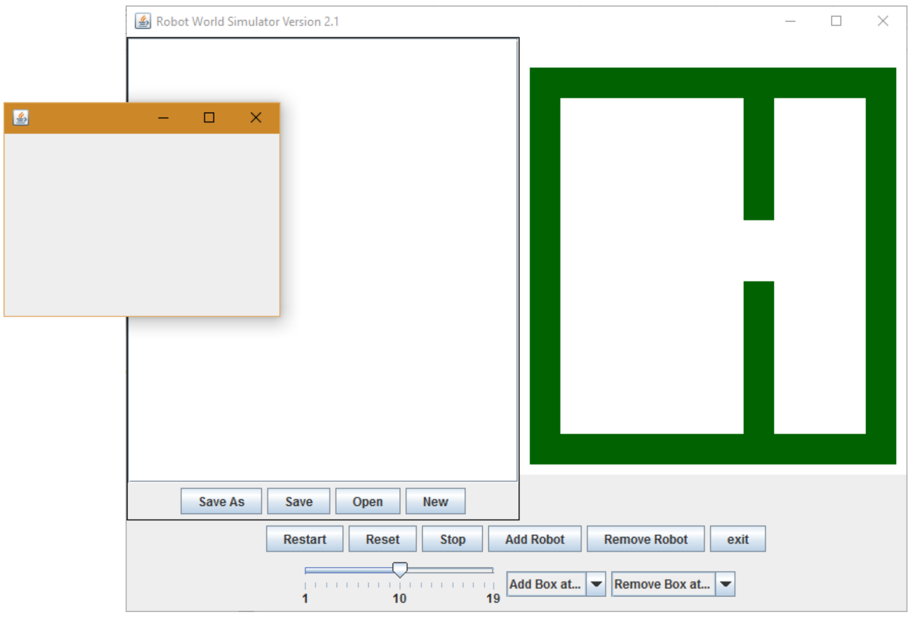


Figure 1: The initial robot world

The buttons act as follows:

Restart : load all robot source files (Robot Control Programs – RCPs) and place the robots at their initial positions.

Stop : stop all robots

Exit : exit the simulator

Add Robot : load another RCP

Add Box at. . . : add a pickup-able obstacle to the robot world at the location specified in the drop-down list

1-19 slider : controls the speed of the robots (1 = slow, 19 = fast)

RCP Syntax

The robot world is divided into a grid. Locations are specified by providing a row number and a column number. Robot control programs must begin with the location of the robot.

For example,

```
/* This is a comment: the robot starts at
     column 5, row 6 */
  start_at 5 6
```

Robots only understand the instructions given in the tables following. The instruction is in **bold** and any other information required by the instruction is in *italics*.

The symbol *expression* stands for any integer or the name of a variable. When it is the name of a variable it refers to the “current” (when the robot is active) value of the variable.

Commands

Instruction	Description	Examples
`start_at` *x y*	Positions the robot at cell (*x*,*y*)	`start_at 7 14`
`step`	Move one step in the direction the robot is facing	`step`
`turn_right`	Turn 90 degrees to the right	`turn_right`
`turn_left`	Turn 90 degrees to the left	`turn_left`

`repeatedly`
    command
    command
    ...
    command
`until` *condition*
	Repeatedly perform the *commands* until the *condition* becomes true. See the table below for valid conditions.	`repeatedly`
		    `step`
		    `step`
		    `turn_right`
		`until the_end_of_time?`

`do_while` *condition*
    command
    command
    ...
    command
`end_do_while`
	While *condition* is true, repeatedly perform the *commands*. See the table below for valid conditions.	`do_while can_move_forward?`
		    `step`
		`end_do_while`

`if` *condition*
    command
    command
    ...
    command
`end_if`
	if *condition* is true, perform each *command*	`if can_move_right?`
		    `turn_right`
		    `step`
		`end_if`

`if` *condition*
    i_command
    i_command
    ...
    i_command
`otherwise`
    o_command
    o_command
    ...
    o_command
`end_if`
	if *condition* is true, perform each *i_command* otherwise perform each *o_command*	`if can_move_forward?`
		    `step`
		`otherwise`
		    `turn_right`
		    `step`
		`end_if`

`quiet` **`noisy`**	Turn diagnostic output off/on. In "noisy" mode, every time the robot performs a command, or tests a condition, a line of output appears in the terminal window from which the "robot world" was started.	`quiet` `noisy`

`trail_off` **`trail_on`**	Turn the robot's trail off/on. In "trail_on" mode, every time the robot moves it leaves a blue square in its previous position - it leaves a trail of blue squares.	`trail_off` `trail_on`
`pick_up` **`put_down`**	Pick up/put down an object directly in front of the robot. Only light green objects can be picked up. An object cannot be put down if there is something else directly in front of the robot (for example, another object, another robot or a wall). The number displayed on a robot (cyan arrow) indicates how many objects a robot has picked up. Objects are obstacles that cannot be stepped over.	`pick_up` `put_down`
`stop`	Stop the simulation (like clicking the "stop" button)	`stop`
`store` *expression* in *variable*.	Store the value of *expression* in *variable*	`store 0 in STEP_COUNT`
`increase` *variable* **`by`** *expression*	Increase *variable* by the value of *expression*	`increase STEP_COUNT by 1`
`decrease` *variable* **`by`** *expression*	Decrease *variable* by the value of *expression*	`decrease SIZE by 10`
`print` *variable*	Produce a line of output that includes the value stored in *variable*	`print STEP_COUNT`

`procedure` *name*
commands
`end_procedure`
 	
`procedure` *name* **`parameter`** *variable*
commands
`end_procedure`
	Declare a procedure (a sequence of commands) that can be called (performed) as a command. Optionally the procedure can take a parameter that specifies an initial value for a local variable. No action is taken by the robot at the time of declaration of the procedure.	`procedure do_a_180`
		`  turn_right`
		`  turn_right`
		`end_procedure`
		
		`procedure step parameter n`
		`   repeatedly`
		`      decrease n by 1`
		`      step`
		`   until equals? n 0`
		`end_procedure`

`perform` *name*
`perform` *name* **`with`** *expression*
	Perform (do the commands in) the procedure *name*. Optionally preform the procedure providing an initial value for its parameter of *expression*.	`perform do_a_180`
		`perform step with 5`

`trigger` *name* **`when`** *character*	Perform (do the commands in) the procedure *name* whenever *character* is typed on the keyboard (asynchronously).	`trigger left when L`

Conditions

Condition	Description
`can_move_forward?` **`cant_move_forward?`**	True if the robot can/can't move one step in the direction it is facing without running into an obstacle (wall, object etc.).
`can_move_left?` **`cant_move_left?`** **`can_move_right?`** **`cant_move_right?`**	True if the robot can/can't turn left/right and move one step in the direction it would be then facing, without running into something (a wall or the goal)
`can_pick_up_object?`	True if there is a light green object directly in front of the robot.
`the_end_of_time?`	Always false
`now?`	Always true
`random` *expression*	Return true with a probability of 1/*expression*.
`equals?` *variable expression*	True if the current value of *variable* is equal to *expression*
`less?` *variable expression*	True if the current value of *variable* is less than *expression*
`greater?` *variable expression*	True if the current value of *variable* is greater than *expression*

Special Variables

Variable	Description
`bag_count`	Always contains the number of things the robot is holding

Comments

Text that will be ignored (human readable comments) can appear in a source file by surrounding the text with **`/*`** and **`*/`**. There must be at least one space or new line before and after the comment start/end symbols. For example,

```
start_at 5,5

/* This program moves the robot to the southern boundary.
   The robot is initially facing north. The first two commands
   will leave it facing south. While there is nothing blocking
   its movement it steps forward.
 */

turn_left
turn_left
do_while can_move_forward?
   step
end_do_while
```

Example RCPs

Some of these RCPs introduce constructs that may not be familiar to you at present (decisions, looping etc.) however, feel free to implement them and have some fun. There will be some other tasks for you to "play with" following this section.

1.
```
/*
Takes three steps forward, turns right and
takes three more steps forward.
*/

start_at 5 8
step
step
step
turn_right
step
step
step
```

2.
```
/* Attempt to move 4 steps forward */

start_at 5 4

if can_move_forward?
  step
end_if
if can_move_forward?
  step
end_if
if can_move_forward?
  step
end_if
if can_move_forward?
  step
end_if
```

3.
```
/* A program that causes the robot to
randomly wander about. Does not terminate.
On average, the robot turns left once in every 5 steps.
*/

start_at 5 6
trail_off
quiet

repeatedly
  if can_move_forward?
    step
  end_if
  if random 5
```

```
    turn_left
  end_if
until the_end_of_time?
```

4.
```
/* Program to move forward until an obstacle
is encountered
*/

start_at 5 6

do_while can_move_forward?
  step
end_do_while
```

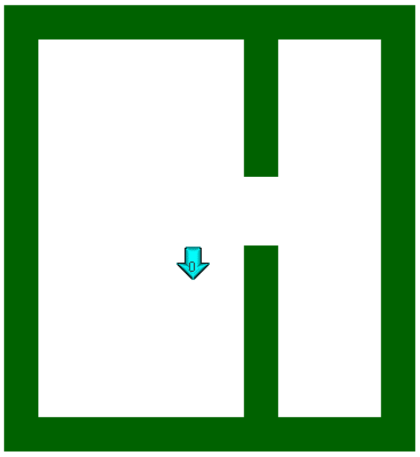
5.
```
/*
Count how many steps it takes until
the robot gets to an obstacle (wall).
*/

start_at 5 11
store 0 in count
repeatedly
  step
  increase count by 1
until cant_move_forward?

/* Print the current value of "count" */

print count
```

6.
```
/* Program to turn the robot left whenever the
'l' key is typed and turn the robot right
whenever the 'r' key is typed
*/
start_at 5 10

procedure left
  turn_left
end_procedure

procedure right
  turn_right
end_procedure

trigger left when l
trigger right when r

repeatedly
  step
until the_end_of_time?
```

7.
```
/*
Shows the use of the pick_up and put_down
commands. The robot moves around the perimeter of the
"world" picking up objects. Once 2 object
have been picked up, they are deposited around
the edge of the world.
*/
start_at 5 10

do_while less? bag_count 2
  do_while can_move_forward?
    step
  end_do_while

  if can_pick_up_object?
    pick_up step
  end_if
  if cant_move_forward?
    turn_left
  end_if
end_do_while

do_while greater? bag_count 0
  do_while can_move_forward?
    step
  end_do_while
  turn_left
  if can_move_forward?
    put_down
    turn_left
    step
    turn_right
    step
    step
    turn_right
    step
    turn_left
  end_if
end_do_while
```

Tasks for you to try

1. Write a RCP that causes the robot to start at position (5 4), to turn to the South (the bottom of the window) and take three steps. The final position of the robot should be:

   This is an example of *sequence* in isolation. Normally sequences will be used as parts of more complex structures.

2. Complete the RCP below by "adding code" (RCP statements) at the indicated place.

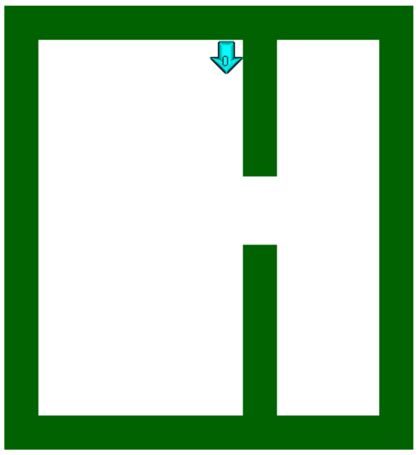
   ```
   /*
   Repeatedly step forward if possible, otherwise turn right.
   That is, move to the facing wall, turn right and then move around
   the perimeter of the "western room" in a clockwise direction.
   */
   start_at 5 4
   repeatedly
   /*
   Add code here to either step forward if possible,
   otherwise turn right. Hint: use an "if/otherwise" statement.
   */
   until the_end_of_time?
   ```

   Since the robot will never stop (the RCP does not terminate) you will need to click the "Stop" button.

   This is an example of *selection* (within iteration) where one of two alternative actions is chosen depending on whether the robot can move forward or not.

3. Modify the RCP from the previous task so that the robot stops once it has moved around the entire perimeter of the "western room" and arrived at the Eastern wall and turned right.

   Hint: use a *variable* to count how many times the robot has turned (start it at 0 and increase it by 1 with every turn). Once it has reached 6 the robot should stop (change the condition after "until" to achieve this). The final position of the robot should be:



4. Modify the RCP from the previous task so that the robot traverses the perimeter of the entire “world” (both rooms) and then stops.