CEEBOT

1 Introduction

1.1 Minimal requirements

CeeBot requires an up-to-date and powerful computer. It is particularly important to have a good 3D graphic adapter for maximum performance.

- 300 MHz CPU, 64 Mb RAM
- 3D graphic card with 16 Mb RAM
- 100 Mb free disk space
- Windows XP, 2000, ME, 98 or 95
- DirectX 8 (If necessary, CeeBot will install DirectX 8)

Recommended graphic cards :

- nVidia GeForce 2, 3 or 4
- Matrox G400, G450, G550 or Parhelia

3dfx Voodoo graphic adapters are not officially supported, but they may work (no guarantee).

Some graphic chips in notebooks will not provide satisfactory performance.

1.2 To install CeeBot

- Insert the CeeBot-A CD-ROM in the drive

Normally, it should start automatically after some ten seconds or so.
Click **Install**

If the autorun doesn't start, follow these steps :

- Double-click **My computer**
- Open the **D**: drive (where D: is the letter of your CD-ROM drive)
- Double-click **Install**

The installation process will check if you have DirectX 8a. If this is not the case, you'll be prompted to install it.

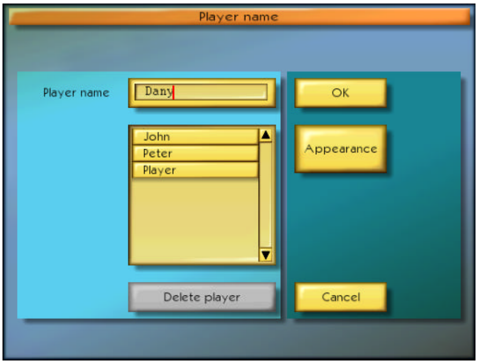
If you have a previous version of CeeBot, a new installation in the same folder will overwrite all data, saved games or programs.

1.3 To uninstall CeeBot

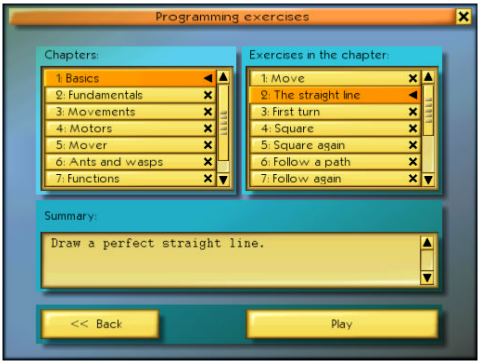
- Double-click **My computer.**
- Double-click **Control panel**.
- Double-click **Add or Remove programs**.
- Double-click **CeeBot-A** in the list.

2 First approach

In this section, we're going to tke a close look at the second exercise. The detailed explanations will be found in the follwing chapters. This is your very first contact, to help you get familiar with CeeBot.



The first thing CeeBot needs is the player's name. It is better if each player gives a separate name, because CeeBot saves your programs and keeps track o your progress.
So just type in your pseudo or name and it will then appear in the list. You can select it and click OK.
Then click



Programming exercises

Chapters:

1: Basics
2: Fundamentals ✗
3: Movements ✗
4: Motors ✗
5: Mover ✗
6: Ants and wasps ✗
7: Functions ✗

Exercises in the chapter:

1: Move ✗
2: The straight line
3: First turn ✗
4: Square ✗
5: Square again ✗
6: Follow a path ✗
7: Follow again ✗

Summary:

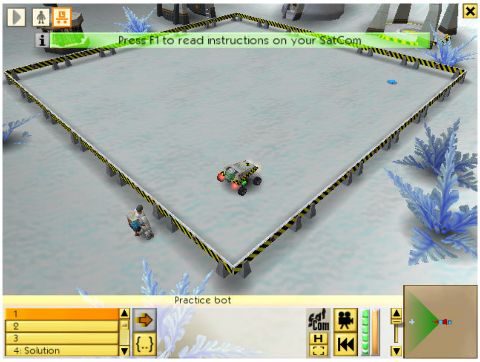
```
Draw a perfect straight line.
```

CeeBot now lists the various exercises. The left column shows the different chapters, and the right column is the list of the exercises in the selected chapter.

The tick means the exercise has been done successfully.

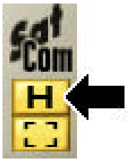
The exercises are progressively more difficult. We suggest you try them in that order, even though this isn't compulsory.

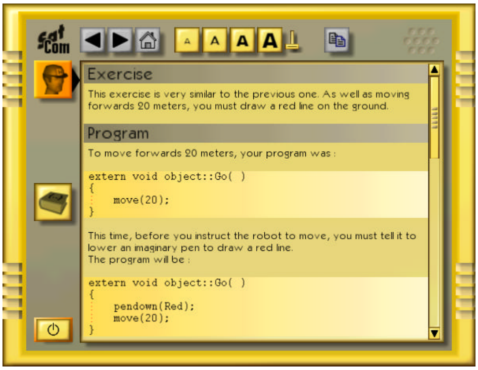
Choose « **1: Basics** » and « **2: The straight line** », click «**Play**».



The green text at the top of the screen asks you to press « **F1** » to check your ***SatCom***. That's where you'll find all the instructions you need to complete the exercise.

You can also press this button, at the bottom of the screen :





Exercise

This exercise is very similar to the previous one. As well as moving forwards 20 meters, you must draw a red line on the ground.

Program

To move forwards 20 meters, your program was :

```
extern void object::Go( )
{
    move(20);
}
```

This time, before you instruct the robot to move, you must tell it to lower an imaginary pen to draw a red line.
The program will be :

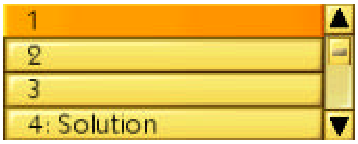
```
extern void object::Go( )
{
    pendown(Red);
    move(20);
}
```

Read this carefully, as it explains exactly what you must do. When you've finished, press the button in the bottom left corner.

If necessary, you can come back to this screen at any time by pressing « **F1** ».

You can tell the exercise bot is already selected by the orange icon in the top left corner. If this wasn't the case, all you need to do is to click it.

You can also click on the robot itself.

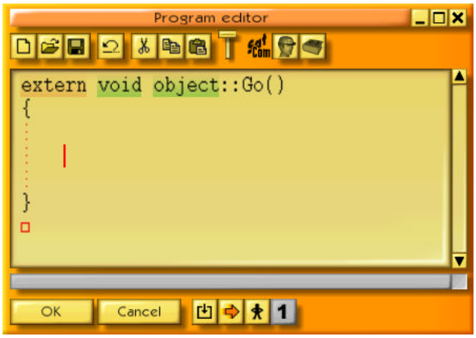


1
2
3
4: Solution

Select the first program in the list. Each bot can have up to 10 programs. This can be handy when you're testing different solutions. All these programs are saved automatically in each player's profile.

Program nr 4 is the solution so you'll never be stuck. But try not to give in too soon!

To enter the program editor, click **{..}**.

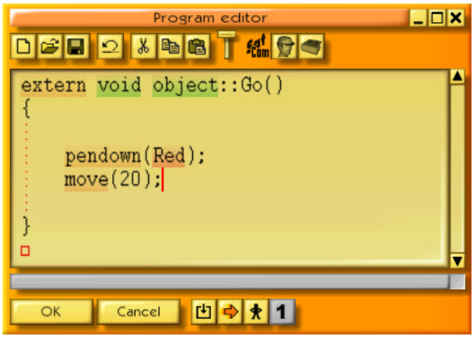


```
extern void object::Go()
{
    |
}
```

The editor comes up with an empty shell ready for you to write your new program.

The text cursor is between the braces.

Do not alter the first line or the braces.



```
extern void object::Go()
{
    pendown(Red);
    move(20);
}
```

Your job is to draw a red line 20 meters long, right up to the blue cross. Your first instruction is to lower the virtual red pen :

```
pendown(Red);
```

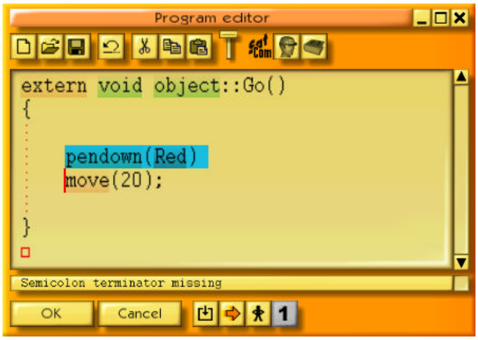
Make sure you type a capital "R" for Red. The rest is in lower case.

Notice the **;** at the end of the instruction.

Now you need a second instruction telling the bot to move forwards 20 meters. Your program will be easier to read if you go to the next line.

```
move(20);
```

Don't forget the **;**



```
extern void object::Go()
{
    pendown(Red)
    move(20);
}
```

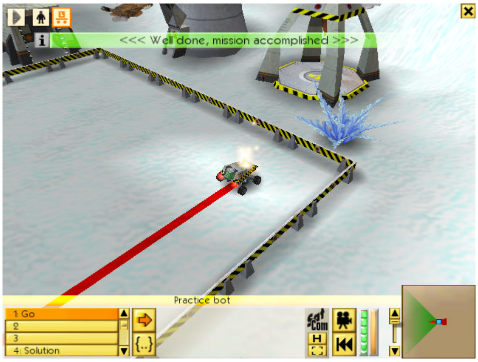
Semicolon terminator missing

Click **OK**

If you've made a mistake, it will be highlighted in blue, and a message will be displayed at the bottom of the screen.

In this example, we forgot the **;** at the end of the first line.

If everything is correct, when you click **OK** the editor is closed.

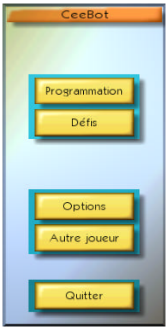


<<< Well done, mission accomplished >>>

Click this icon to run your new program. The robot should drive up to the blue cross, leaving a red trail behind.

Congratulations, you've just completed your very first program.

3 Main menu



3.1 Exercises

In this part of CeeBot-A, you can learn how to program the bots, even if you don't know anything about programming. The exercises are gathered into chapters, and progressively introduce the basic concepts of programming in a structured and object-oriented language. You can do the exercises in any order, but it's better to start with the easier ones.

3.2 Challenges

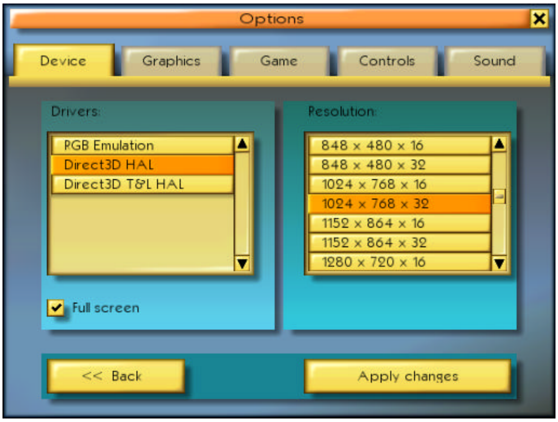
Each challenge is a mission you must try to fulfill. You need a pretty good knowledge of the CeeBot programming language to succeed. Challenges are a good way of checking the skills acquired in the programming exercises.

3.3 Options

Click « **Options** » to access various settings separated into 5 categories.

3.3.1 Display

The first time you run CeeBot, it is set to use a 640 x 480 x 16 display. On many computers, you can get a much better graphic result by changing the default settings.



Drivers:

It's best to choose a HAL driver (Hardware Abstraction Layer), and avoid "Emulation" or "T&L" drivers.

Resolution:

The first and the second figure stand for the size of the image on your screen.
The third number influences the number of colors used:

16 bits will display 65,000 colors, whereas 32 bits will display 4 million colors. The higher you set these parameters, the more detailed the scene will be on the screen. However, high parameters need a lot more processor and graphic power, and the frame-rate will be lower.

To find the best settings for your computer, we suggest you start off with 640 x 480 x 16 and climb slowly higher. Most modern graphic cards support at least 1024 x 768 x 16.

If the display seems to be a little jerky, try a lower quality.

✍ Full screen

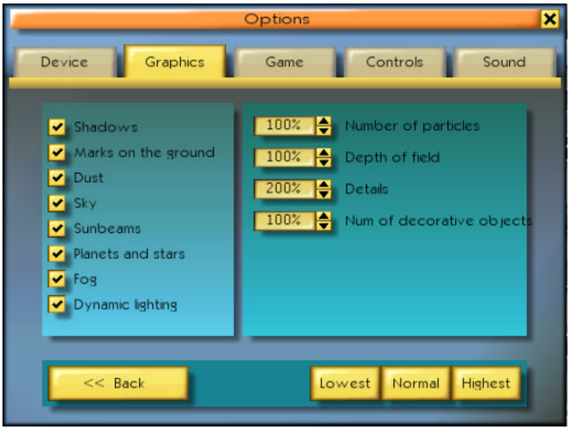
Normally CEEBOT runs in full screen mode, whatever resolution you choose. If you deactivate this flag, CeeBot will be displayed in a fixed size window of approximately 640 x 480 pixels.

[Apply changes]

Hit this key to use the new settings.

3.3.2 Graphics

If the game is too slow, or if the frame-rate is too low, you can tune down some of the graphic options. Obviously, the higher the settings the more beautiful the game!



- ✓ Shadows
- ✓ Marks on the ground
- ✓ Dust
- ✓ Sky
- ✓ Sunbeams
- ✓ Planets and stars
- ✓ Fog
- ✓ Dynamic lighting

100%	Number of particles
100%	Depth of field
200%	Details
100%	Num of decorative objects

Number of particles (0% - 200%)

Particles are used to simulate dust, smoke, fire, explosions, etc.

Depth of field (50% - 200%)

This feature determines up to what distance the scenery is displayed. This distance varies from planet to planet. A high value close to 200% allows you to see very far, but requires a powerful 3D graphic card.

Details (0% - 200%)

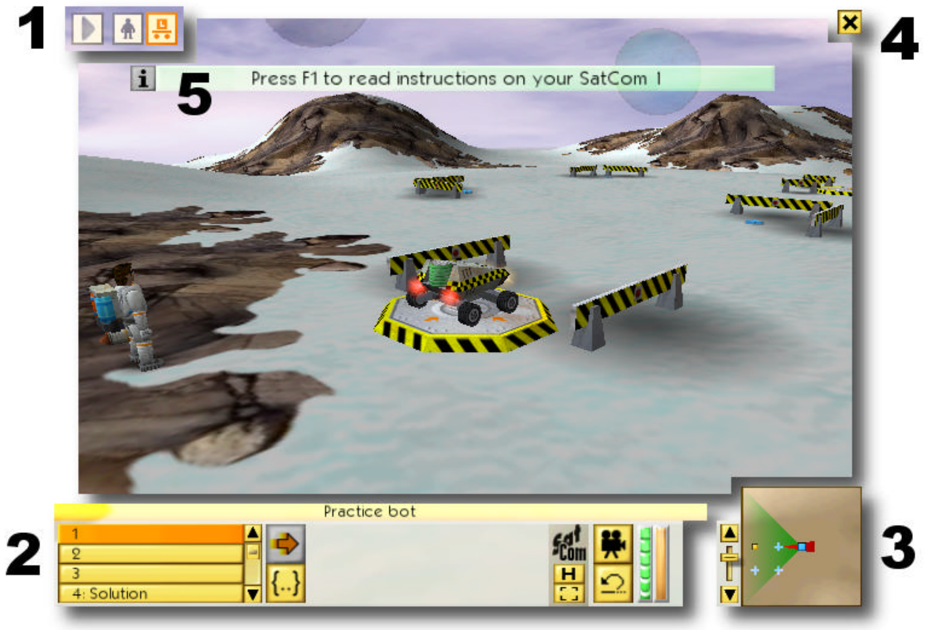
Sets the visual quality of 3D objects according to their distance.

Number of decorative objects (0% to 100%)

Sets the number of purely ornamental objects like plants, trees, crystals, etc.

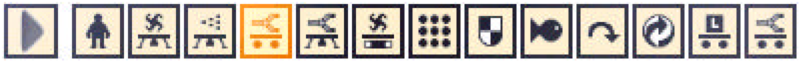
4 The screen

This is what the screen looks like during an exercise :



1 Summary of available robots and buildings
2 Control panel of the selected item
3 Mini map
4 Access to the menu
5 Temporary messages

4.1 The summary



The top of the screen displays a summary of the robots and buildings in the exercise. The currently selected item shows up in orange. When a robot is running its program, the frame of the corresponding icon flashes.

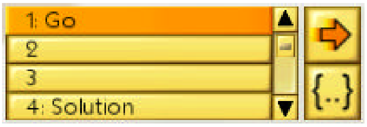
You can select an item by clicking an icon, or by pressing the « Tab » key.

At the beginning of the exercise the astronaut and the robots are displayed. Click the arrow to show the buildings. Click again to switch back.

4.2 *The control panel*

The bottom of the screen displays all the controls of the selected item. The buttons will therefore be different if you've selected the cosmonaut, a robot or an exchange post.

4.2.1 *The ten programs*



The left bottom corner gives access to 10 program slots for each robot. The 4th program usually is our suggestion for the **solution** of the exercise. Use the scroll bar to access programs 4 to 10..

Launch or stop the execution of the selected program.

Launch the built-in program editor for the selected program (see section **Erreur ! Source du renvoi introuvable.**). When you edit a program, the game pauses so you can take all the time you need to modify programs.

4.2.2 *The SatCom*

The ***Sat*****Com** is a watch-like apparatus the cosmonaut wears on his wrist. It is used to communicate with the main operating center and displays all information and data about the exercise (see section 5).

Click the **H** icon or press « **F1** » to activate the ***Sat*****Com** and display the object of the exercise.

Click the **[ ]** icon to activate the ***Sat*****Com** and display information about the selected item.

4.2.3 *The camera*

You can change the viewpoint of the camera by clicking this button or by pressing the space bar of your keyboard. Usually this switches between a rear sight and the onboard camera.

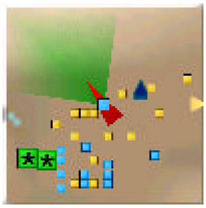
4.2.4 *Reset*

This button sets the items and surroundings to their initial state. Any program you wrote won't be modified. Use this feature when you want to change and run your program from the beginning again.

4.2.5 *Gauges*

Shows the state of the selected object. From left to right: energy level of the power cell, the shield level and the jet's temperature (only for flying robots and the astronaut).

4.3 The mini map



The mini map in the lower right corner offers an overview of the situation. The darker the zones, the lower the altitude. Robots are yellow, buildings are blue and enemies are green.

Enemies are only displayed if a radar is available. When you pass over an object on the mini map, the name of the object is shown in a small popup window. You can click on an object on the mini map to select it.

The slider on the left side of the map allows you to zoom in or out.

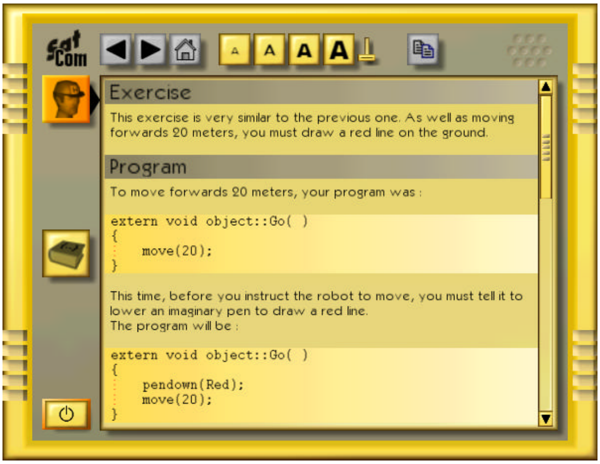
4.4 The menu

If you click this button in the top right corner of the screen, the menu is displayed:

Continue	Continue the current exercise.
Options	Access certain game options. You don't have access to all options from here. If you want to have access to all options use the Options button of the main menu (see section 3.3).
Restart	Abort the current exercise and restart from the beginning.
Abort	Abort the current exercise.

5 The *Sat*Com

The ***Sat*Com** is a very crafty instrument that displays all the instructions for the exercise. It also holds information about the CeeBot programming language.



Exercise

This exercise is very similar to the previous one. As well as moving forwards 20 meters, you must draw a red line on the ground.

Program

To move forwards 20 meters, your program was :

```
extern void object::Go( )
{
    move(20);
}
```

This time, before you instruct the robot to move, you must tell it to lower an imaginary pen to draw a red line.
The program will be :

```
extern void object::Go( )
{
    pendown(Red);
    move(20);
}
```

It works rather like an Internet browser. Whenever a word is underlined, you can click it to display the corresponding page.

Previous page.

Next page.

Display the page that came up when you turned the ***Sat*Com** on.

Copies the highlighted characters to the clipboard. You can use this feature to copy all or part of a program and paste it into the editor (see section **Erreur ! Source du renvoi introuvable.**).

Displays the instructions for the current exercise. « F1 » always accesses this page directly.

Displays overall help for the CeeBot programming language. You'll find detailed explanations for each instruction. « F2 » always accesses this page directly.

Turns the ***Sat*Com** off and closes the window.

If one of the icons described here is gray, it can't be used in the current context.

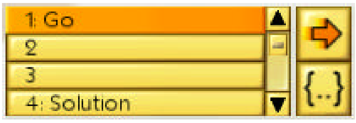
6 Programming

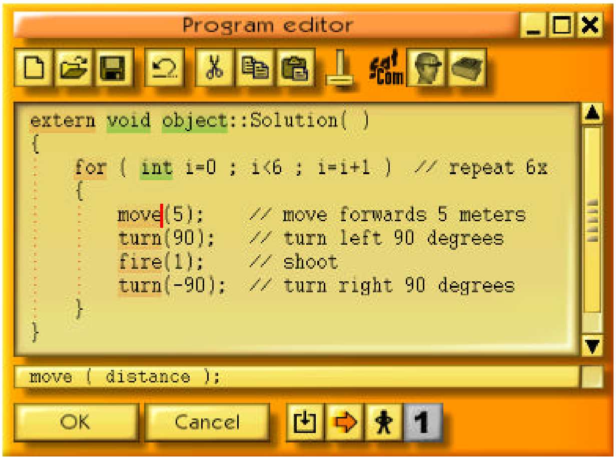
All the robots are programmable. In some exercises, one of the robots is pre-programmed and cannot be accessed. For instance in exercise « **4.4: Tom thumb** ».

6.1 *Program editor*

To enter the program editor proceed as follows:

- Select a robot
- Choose one of the 10 program slots in the bottom left corner of the screen
- Click the « Edit the selected program » **{..}** button





```
extern void object::Solution( )
{
    for ( int i=0 ; i<6 ; i=i+1 )  // repeat 6x
    {
        move(5);     // move forwards 5 meters
        turn(90);    // turn left 90 degrees
        fire(1);     // shoot
        turn(-90);   // turn right 90 degrees
    }
}
```

```
move ( distance );
```

During program edition the game is paused. If you move the mouse pointer close to the left or the right edge of the screen, the camera will rotate. If you move the mouse pointer close to the upper or the lower edge of the screen, the viewpoint moves forward or backward. Use this feature to have a good look at the surroundings. It can be necessary to reduce the editing window.

Reduced size, to reduce the window to a bar at the bottom of the screen.

Maximum size, to enlarge the window to full screen.

Close, equivalent to « OK ».

If you enter the program editor while the program is being executed the game is not paused but you can observe the progress of the program (see section 6.1.11).

The program editor window can be moved by dragging the title bar. You can also change its size by dragging the edges or the corners. Next time you call up the editor, it will be exactly as you left it.

The keywords of the language are displayed in different colors:

Color	***Kind***	***Example***
Orange	Instruction	`aim`, `fire`, `turn`, `goto`, `while`, etc.
Green	Variable type	`object`, `float`, `int`, etc.
Red	Constants, Category	`TitaniumOre`, `AlienAnt`, etc.

When the cursor is on a keyword, the status bar at the bottom of the program editor window displays the word and some information. You can then click on the status bar or press « F3 » to bring up the **SatCom** , which will display further information.

You can double click a word to select the whole word. Shift-arrow also selects text. Ctrl-arrow moves word by word and Shift-Ctrl-arrow selects text word by word just like any standard editor. The usual Ctrl-X, Ctrl-V, Ctrl-C shortcuts for copying and pasting text are also available.

6.1.1 New

Clears the whole program and creates an empty skeleton program :

```
extern void object::New( )
{

}
```

`New` is the default name of the program. You can change it to whatever you like, but you musn't use blank spaces or special characters, only letters and digits. For example `Search`, `BringTitanium`, `ComeHome`, etc.

Note: You can revert to the previous situation by pressing Ctrl-Z or by clicking the Undo button (see below).

6.1.2 Open and save

All programs you create are automatically saved within the mission or the exercise. On the other hand, if you want to reuse a program in another mission you must save it explicitly.

Private The program is saved in a private folder attached to the current player.

Public The program is saved in a public folder and will be available to all other players.

The name of the folder where the program will be saved appears in the title bar of the Save dialog. For example, **Savegame\Player1\Program\** means that the program will be saved in **c:\Program Files\CeeBot-A\Savegame\Joueur\Program\** (assuming CeeBot has been installed in the folder **c:\Program Files\ CeeBot-A**.

Shortcuts: « Ctrl+O » and « Ctrl+S ».

6.1.3 Undo

Cancels the last modification. You can Undo the 20 last modifications.

Shortcut: « Ctrl+Z ».

6.1.4 Cut, Copy and Paste

Cuts or *copies* the selected text to the clipboard. If no text is selected, the whole line will be taken. The contents of the clipboard can be pasted into the program you are editing.

Shortcuts: « Ctrl+X », « Ctrl+C » and « Ctrl+V ».

6.1.5 Font size

Use this slider to change the font size for the editor and the **SatCom**.

6.1.6 SatCom: Instructions

Shows information about the goal of the mission or the exercise.

Shortcut: « F1 ».

6.1.7 SatCom: Programming help

Shows general information about programming robots. Use the hypertext links (underlined in blue) to navigate through the pages.

Shortcut: « F2 ».

6.1.8 OK

Compiles the program and quits the editor. If there is a syntax error in your program, the error is shown and an appropriate error message is displayed in the status bar.

6.1.9 Cancel

Quits the editor without compiling but all modifications are saved.

6.1.10 Compile

Compiles the program without quitting the editor. This is useful to check your program for syntax errors.

6.1.11 Execute/stop

Starts or stops program execution without quitting the editor. This is useful for debugging purposes as you can follow the progress of your program. If the button on the right side depicts a standing man, the program will be executed step by step.

Note: During program execution the content is displayed in orange and you cannot modify the program.

6.1.12 Pause/continue

Switches from step by step execution to continuous execution and back.

6.1.13 One step

Executes the next instruction in step by step mode. The lower part of the program editor shows the contents of the different variables which change during the progress of the program.

6.2 The CeeBot programming language

The CeeBot programming language is very close to C++, C# and Java? . It has been designed specially for CEEBOT and is very well adapted to learning. In this manual, we only describe some very simple examples. For a more complete description, use the **SatCom** by pressing « F2 ».

The CeeBot language is case sensitive, for example, `Radar` is not the same as `radar`.

Each instruction must be terminated by a semicolon "`;`".

Comments start with `//` and end at the end of the line.

6.2.1 Instruction `Radar`

With this instruction, you can look for objects like enemies, bots, buildings or raw material.

Write the name of the object that you are looking for between brackets. Put the result in a variable of the type object. Here is an example that looks for the closest ant:

```
        // At the beginning of the program:
        object  item; // variable declaration

        // Look for the closest ant
        item = radar(AlienAnt);
```

6.2.2 Instruction `goto`

`goto()` instructs the bot to reach a given position.

The most current use consists in moving the bot to an object located with the instruction `radar()`. If the information returned by `radar()` has been stored in a variable, write the name of the variable followed by `.position` in order to get the position of the object. Here is an example of a program that looks for a titanium cube and goes to its position:

```
        object item;
        item = radar(Titanium);
        goto(item.position);
```

6.2.3 Instructions `grab` and `drop`

The instruction `grab()` tells the bot to use its operating arm to grab an object located on the ground, on the platform of a building or on the power cell location of a bot. The instruction `drop()` on the other hand instructs the bot to put whatever the operating arm is carrying on the ground, on the platform of a building or on the power cell location of a bot.

After finding titanium in the previous example, this is how to tell your bot to pick it up and put it down 5 meters further:

```
        grab();   // grab the object
        move(5);  // move forwards 5m
        drop();   // drop it
```

6.2.4 Instruction `fire`

Use this instruction to fire the bot's onboard cannon. You can specify how long the burst must last. Generally, this instruction is used to shoot one-second bursts:

```
	fire(1);
```

6.2.5 Instruction `while`

`while(){}` is used to repeat a set of instructions several times. The most frequent use of while consists in repeating a set of instructions again and again. In order to achieve this, write `while(true){}` and put the instructions to be repeated in braces `{}`. As an example, here is a program that repeats the following actions:

?? look for a spider,
?? turn towards it,
?? shoot.

```
	while (true)
	{
		item = radar(AlienSpider);
		turn(direction(item.position));
		fire(1);
	}
```

Just execute this program once, and the shooter will kill all spiders around it.

6.2.6 Instruction `distance`

With `distance( , )` you can reckon the distance between two positions. If you use `position` alone, this gives you the position of the bot that is executing the program. A variable followed by `.position`, gives you the position of the object described in the variable.

Here is a program that moves forward, covering exactly the distance between the bot and the closest ant:

```
	item = radar(AlienAnt);
	move(distance(position, item.position));
```

This is of course pure suicide. Better to stop 40 meters away, in order to be within shooting range:

```
	item = radar(AlienAnt);
	move(distance(position, item.position) - 40);
```

6.2.7 Instruction `if`

Use `if(){}` to execute a set of instructions only if a certain condition is true. Write the condition in brackets (), and the instructions in braces {}.

Here is an example: The bot will shoot only if the target is closer than 40 meters:

```
	item = radar(AlienAnt);
	if (distance(position, item.position) < 40)
	{
		fire(1);
	}
```

You can also test if an object exists at all. If the instruction `radar()` does not find the requested object, it returns the special value `null`. So you can test if an object does not exist with the condition `(item == null)`, or test if it exists with `(item != null)`. A double equal == tests equality, an exclamation mark followed by an equal sign `!=` tests inequality. Here is a test that will go to load the power cell up only if there is a power station:

```
        station = radar(PowerStation);
        if (station != null)
        {
                goto(station.position);
                wait(5);
        }
```

6.2.8 *Instruction* `motor`

The instruction `motor( , );` sets the speed for the left-hand and the right-hand motor of the bot. The speed given to the motors will remain constant during the execution of the next instructions and therefore it is possible to perform a rotation during the instruction `fire();`. This will sweep a whole zone with only one burst. Here is an example that will sweep the zone in front of the bot:

```
        turn(45);          // turns 45 degrees left
        motor(0.5, -0.5); // slow rotation to the right
        fire(2);           // fire
        motor(0,0);        // stops the rotation
```

With the left-hand motor turning half-speed forwards and the right-hand motor turning half-speed backwards, the bot will pivot slowly during the 2-second-burst.

6.2.9 *Instruction* `turn`

Use the instruction `turn()` to instruct the bot to perform a rotation of a certain number of degrees. 90 degrees is a quarter turn, 180 degrees is a half turn. A positive angle will perform a counterclockwise rotation, a negative angle means a clockwise rotation. Here are some examples with `turn()`:

```
        turn(90);    // quarter turn to the left
        turn(-90);   // quarter turn to the right
        turn(180);   // half turn
```

In order to turn the bot towards an object found with the instruction `radar()`, you must calculate the rotation angle with the instruction `direction()`:

```
        TheThing = radar(AlienSpider);
        turn(direction(TheThing.position));
```

After these lines, just fire the cannon, and you'll have gotten rid of a hostile element.

7 Development team

Daniel Roux, Denis Dumoulin, Otto Kölbl, Michael Walz

EPSITEC SA, Mouette 5, CH-1092 Belmont
epsitec@epsitec.ch
www.epsitec.com