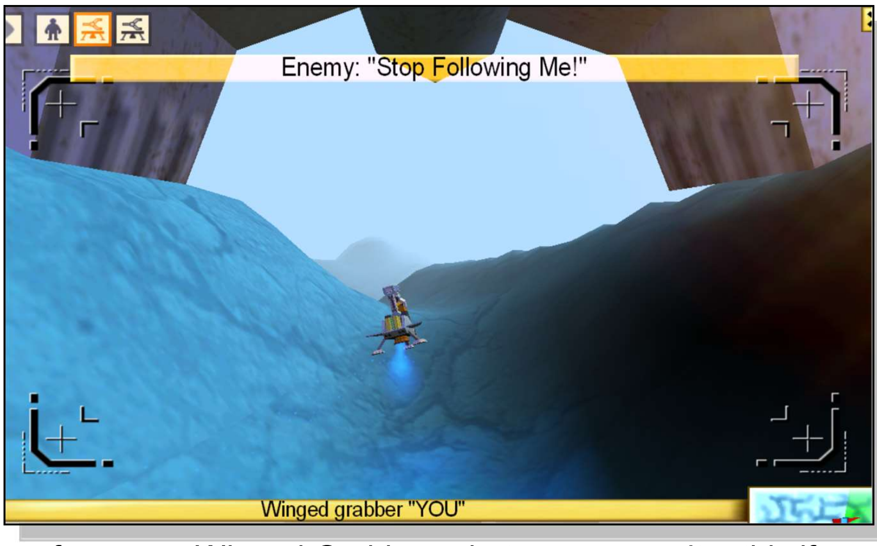
Week 8

Project: Robot Chase

Overall Task (Ceebot Task 24.2)

- This exercise starts with an enemy robot stealing the BlackBox and flying off to a secret area of the planet.
- The **BlackBox** (which is actually an orange-brown colour!) holds important expedition details and must be returned.
- Your task is to devise a program that will allow you to control your winged robot
- Then you must fly your robot, follow the enemy, retrieve the BlackBox and return it to the SpaceShip.
- You start with a 'cockpit' view from your Winged Grabber robot .. you can alter this if you prefer, by clicking the camera icon.



Basic Program: Follow the Enemy

The basic program to design will allow you to use your keyboard to control your flying robot.
You will need to be able to continually do the following:

- Fly the robot higher
- Fly the robot lower
- Turn left
- Turn right

by pressing different keys on the keyboard.

The keypushed() instruction

- It is no good using the **dialog()** instruction for this project because it will keep stopping the action.
- If you have not met the **keypushed()** instruction before, you need to know about it now. It simply allows you to detect the pressing of a key and then to do something quickly without stopping the action.
- Here is an example: detecting a left arrow key press (called a Virtual Key)

```
if  (keypushed(VK_LEFT))            // if left arrow key is pushed
{
        turn(2);                    // turn slightly left
}
```

- You can also use it to detect ordinary keys, for example:

```
if  (keypushed("a"))                // if the a key is pushed
```

Program Help

There are various ways to tackle this project, but if you are stuck, here are a few suggestions.
First .. plan your program by designing an **algorithm** for it.

- I hope you can see that you will need an infinite **loop**
- Inside the loop you will need to drive the robot forward at high speed. You can try using **move(..)** but it is better to use the **drive(..)** instruction.

```
drive(1,0);        //  moves forward at top speed
```

- You should also pause very briefly (0.01 seconds is about right) to allow the robot time to move a bit before anything else happens.
- Then inside the loop you should also detect appropriate keypushes (see above) and use these to control the robots 4 main movements (up, down, left, right)
- Hint: you need to use the **jet(..)** instruction.

If you can put all this together, you should now be able to fly and follow the Enemy!

Additional thoughts

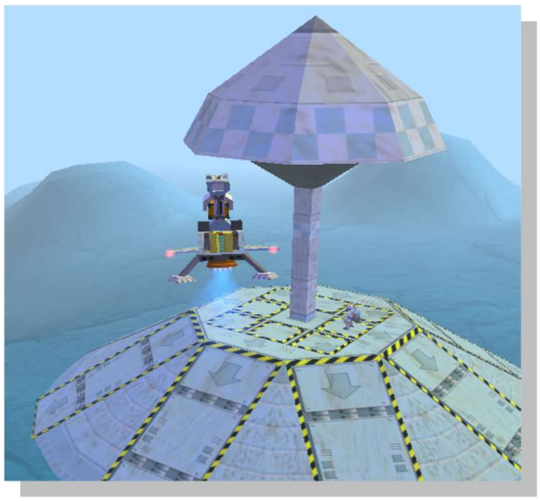
- It is a good idea to start your program by displaying some instructions to remind the user what the controls do.
- Put all this in a separate function that is called from your main program.

Extension Work 1: Returning the BlackBox to the SpaceShip

1. **Slowing Down**
    - If you follow the enemy robot, you will eventually see it drop the BlackBox in a secret location and then fly off.
    - You must land your robot to pick up the box, but at the moment you are flying forward at top speed so you will probably just crash!
    - You must provide a way of slowing down or stopping the robot, before you try to land.
    - Hint: you could use a variable for the speed setting in the **drive(..)** instruction.

2. **Picking Up and Returning the BlackBox**
    - Now expand your control panel by adding the ability to grab an item and to drop an item using keyboard control again
    - Then finish the exercise by actually picking up the dropped BlackBox and taking it to the SpaceShip and then dropping it there.
    - Where is the SpaceShip? You can look at the minimap at the bottom right of the screen to find it.

Extension Work 2: AutoPilot

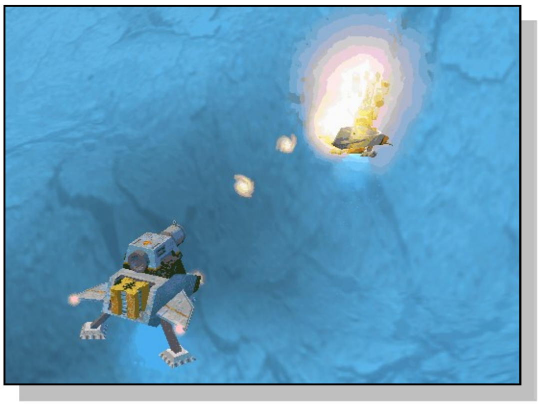


There is another way to tackle this exercise.
You can provide an autopilot program that uses the robot's radar to follow the enemy robot.
Design another program, with at least 3 **functions**:

1. **TakeOff()**
    - The robot should ascend to a height of about 10 metres
2. **Follow()**
    - Using the radar, follow the enemy until it drops the BlackBox.
    - Note: the BlackBox will not be detected while it is being carried .. the enemy robot will mask it. So **radar(BlackBox)** will return a **null** value until the box is dropped.
3. **ReturnHome()**
    - The robot should descend and pick up the BlackBox.
    - Then use the radar again to find the SpaceShip, go there and drop the box.

Start the program by giving the user a choice of a manual control or autopilot
You can get more help using **[F1]**

Extension Work 3 : Finish off the Enemy!



- The Enemy robot flew off after stealing and dumping the BlackBox. You have (hopefully) been able to retrieve the box but you still have this pesky robot hanging around waiting to do more mischief!
- Notice that there is a powerful **WingedShooter** robot near the SpaceShip. You can select its icon at the top of the screen.

Your final task is to program this robot to automatically seek and destroy the enemy WingedGrabber robot.

Hints

- You can modify some of the code that you have already written
- The enemy robot is hovering so you will need to be able to reach the same height before firing (remember to use the z coordinate here)
- Note that **radar("Enemy")** can be used instead of **radar(WingedGrabber)** .. why could the normal method be a problem?
- Note also that there is a **TargetBot** on a nearby hill .. so you can practice your program using this instead of waiting for the other program to finish.

Project Deliverables

In your log book, you should include:

- Algorithms
- All your commented Source Code
- Test Plan
- Screenshots
- Everything should be clearly labelled and contain a report summarising how you tackled the project and any problems along the way.

Appendix A: Examples of Important Ceebot Instructions

1. Ceebot Specific

move (20);	.. move 20 metres
turn(180);	.. turn 180 degrees anticlockwise
wait(1.5);	.. wait for 1.5 seconds
fire(2.5);	.. fire cannon for 2.5 seconds
aim(10);	.. aim cannon 10 degrees up (use values from -20 to +20)
grab();	.. grab the item directly in front
drop();	.. drop the item being carried
pendown();	.. put the pen against the floor ready for drawing
penup();	.. lift the pen to stop drawing
red();	.. select a red pen for drawing (various colours available)
drive(1, 0);	.. drive forward at full speed
jet(1);	.. fly upwards at full speed (use values -1 to +1)

2. Input and Output

name = dialog("Enter Name");	.. show dialog to input name and store it in name variable
message("I am " + name);	.. output a message with text joined to a name variable
num = strval (dialog("Enter number"));	.. enter string and convert to its number value
item = radar(WheeledShooter);	.. get details of the nearest WheeledShooter robot
keypushed(VK_UP);	.. detects pressing of a key (e.g the UP arrow key) .. used with if()

3. Variables

int count;	.. define a variable called count to store an integer number
float num;	.. define a variable called num to store a float (decimal) number
string name;	.. define a variable called name to store a string (text or words)
object item;	.. define a variable called item to store an object's details
point here;	.. define a variable called here to store a position (x and y)

4. Assignments to Variables (must be defined first)

count = 0;	.. put 0 into the count variable (previously defined as int)
num = 5.67;	.. put 5.67 into the num variable (previously defined as float)
name = "Fred Bloggs";	.. put these 11 characters in the name variable (defined as string)
here = this.position;	.. store current position of robot in the here variable (a point)
angle = direction(item.position);	.. find angle of item from you (after using radar)
dist = distance (item.position, this.position);	.. find distance from item to your position.

5. Calculations

count ++;	.. add 1 to the value of the count variable
count --;	.. subtract 1 from the value of the count variable
count = count + 3;	.. add 3 to value of the count variable (or use **count += 3;**)
count = count - 6;	.. subtract 6 from the count variable (or use **count -= 6;**)
av = (num1 + num2 + num3 + num4) / 4;	.. work our average of 4 numbers
tax = bill * 17.5 / 100;	.. work out 17.5 percent tax on your bill

6. Loops (iteration)

a. The while loop

```
int count = 0;      // initialise a loop counter to zero

while (count < 10)     // continue while loop counter is less than 10
{
      message ("The count is " + count);   // repeated message
      count ++;      // keep loop going by adding 1 to counter
}
```

b. The for loop

```
// initialise loop counter; continue while count less than 10 ;  add 1 at end of loop

for (int count = 0; count < 10; count ++)
{
        message (“The count is " + count);   // repeated message (10 times)
}
```

c. The do while loop

```
int  count = 0;      // initialise a loop counter to zero

do
{
        count ++;       // keep loop going by adding 1 to loop counter
          message (“The count is " + count);   // repeated message
}
while (count < 10);     // continue while loop counter is less than 10
```

7. Selection

a. The if statement

```
int  count = 0;

while (count < 10)
{
        if (count == 4)  // if count is equal to 4
        {
                message (“We are half way" );
        }

        count ++;
}
```

b. The if else statement

```
        if (count >= 4)  // if count is greater or equal to 4
        {
                message (“We have reached half way" );
        }
        else
        {
                message ("We are NOT half way yet");
        }
```

c. The switch statement

```
switch(count)  // use count value to switch to various cases below:
{
        case 1:                              // i.e. if  count value = 1
                message ("We are just starting" );      break;
        case 2:  case 3: case 4:
                message ("We are on our way" );         break;
        case 4:
                message ("We are half way" );           break;
        default:
                // do nothing for any other values
}
```

8. Conditions

(a == b) .. a is equal to b ?
(a > b) .. a is greater than b ?
(a < b) .. a is less than b ?
(a >= b) .. a is greater or equal to b ?
(a <= b) .. a is less than or equal to b ?
(a != b) .. a is NOT equal to b ?

(a == b **||** a == c**)** .. a is equal to b **OR** a is equal to c ?
(a == b **||** a == c **||** a == d**)** .. a is equal to b **OR** a is equal to c **OR** a is equal to d ?

(a == b **&&** a == c**)** .. a is equal to b **AND** a is equal to c ?
(a <= 100 **&&** a >= 0**)** .. a is less than or equal to 100 **AND** a is greater or equal to 0 ?

9. Functions

```
void  object::myFunc()
{
    message ("I am now inside the myFunc function");
}
```

*// this defines a function called **myFunc** which has no parameters and returns nothing (void)*
*// to use it, you 'call' it using its name : i.e. **myFunc();***

10. Functions with parameters

```
float  object::Tax(float  amount)
{
     float  taxAmount;                // local variable
      taxAmount = amount * 17.5/100;
      return  taxAmount;
}
```

*// this defines a function called **Tax** which has 1 parameter and returns a float value*
*// to use it, you can 'call' it like this: **vat = Tax(Bill);***
// this passes the Bill value into the function and picks up the returned tax value from it.
([F2] key for more)

Appendix B: The Basics of C# (Console)

1. Input and Output

name = Console.ReadLine(); .. store input in a name variable (defined as string)
Console.WriteLine("I am " + name); .. output a message with text joined to a name variable
num1 = Convert.ToDouble (Console.ReadLine()); .. enter string and convert to a double
num2 = Convert.ToInt32 (Console.ReadLine()); .. enter string and convert to an integer

2. Variables

int count;	.. define a variable called count to store an integer number
double num;	.. define a variable called num to store a double (decimal) number
string name;	.. define a variable called name to store a string (text or words)

3. Assignments to Variables (must be defined first)

count = 0;	.. put 0 into the count variable (previously defined as int)
num = 5.67;	.. put 5.67 into the num variable (previously defined as double)
name = "Fred Bloggs";	.. put these 11 characters in the name variable (defined as string)

4. Calculations

count ++;	.. add 1 to the value of the count variable
count --;	.. subtract 1 from the value of the count variable
count = count + 3;	.. add 3 to value of the count variable (or use **count += 3;**)
count = count - 6;	.. subtract 6 from the count variable (or use **count -= 6;**)
av = (num1 + num2 + num3 + num4) / 4;	.. work our average of 4 numbers
tax = bill * 17.5 / 100;	.. work out 17.5 percent tax on your bill

5. Loops (iteration)

a. The while loop

```
int count = 0;      // initialise a loop counter to zero

while (count < 10)     // continue while loop counter is less than 10
{
        Console.WriteLIne ("The count is " + count);   // repeated
        count ++;       // keep loop going by adding 1 to counter
}
```

an infinite loop

```
while (true)     // continue the while loop forever
{
        Console.WriteLIne ("Yippeeee!!");   // repeated forever
}
```

b. The for loop

```
// initialise loop counter; continue while count less than 10 ;  add 1 at end of loop

for (int count = 0; count < 10; count ++)
{
        Console.WriteLine ("The count is " + count);   // repeated 10 times
}
```

c. The do while loop

```
int count = 0;      // initialise a loop counter to zero

do
{
        count ++;       // keep loop going by adding 1 to loop counter
        Console.WriteLine (“The count is " + count);   // repeated message
}
while (count < 10);     // continue while loop counter is less than 10
```

6. Selection

a. The if statement

```
if (count == 4)  // if count is equal to 4
{
        Console.WriteLine (“We are half way" );
}
```

b. The if else statement

```
if (count >= 4)  // if count is greater or equal to 4
{
        Console.WriteLine (“We have reached half way" );
}
else
{
        Console.WriteLine ("We are NOT half way yet");
}
```

c. The switch statement

```
switch(count)  // use count value to switch to various cases below:
{
        case 1:                               // i.e. if  count value = 1
                Console.WriteLine (“We are just starting" );            break;
        case 2:  case 3: case 4:
                Console.WriteLine (“We are on our way" );               break;
        case 4:
                Console.WriteLine (“We are half way" );         break;
        default:
                // do nothing for any other values                      break;
}
```

7. Conditions

(a == b) .. a is equal to b ?
(a > b) .. a is greater than b ?
(a < b) .. a is less than b ?
(a >= b) .. a is greater or equal to b ?
(a <= b) .. a is less than or equal to b ?
(a != b) .. a is NOT equal to b ?

8. Multiple Conditions

```
(a == b  || a == c)           .. a is equal to b OR a is equal to c  ?
(a == b  || a == c || a == d)   .. a is equal to b OR a is equal to c OR a is equal to d ?

(a == b  && a == c)           .. a is equal to b AND a is equal to c  ?
(a <= 100  && a >= 0)         .. a is less than or equal to 100 AND a is greater or equal to 0  ?
```

9. Classes, Objects and Methods

```
class Meal                    // define a class called Meal
{
      private string food;    // the class has one class variable (attribute or field)

        public static void Main()        // program starts executing here
        {
             Meal myMeal = new Meal();           // create a new myMeal object
           myMeal.getFood();                   // call the object's getFood() method
      }

      public Meal()                        // this is the Meal class constructor
      {
            food = "Fish and Chips";    // this sets the default food
      }

      public void getFood()           // define a method getFood()which returns nothing
(void)
      {
            Console.WriteLine("What would you like to eat?");
            food = Console.ReadLine();             // input into the class variable food
      }
}
```

*// this defines a simple class called **Meal** which has one variable, one method, one constructor*

10. Methods with parameters

```
public double setTax(double  amount)
{
    double  taxAmount;          // local variable
      taxAmount = amount * 17.5/100;
      return  taxAmount;
}
```

*// this defines the method **setTax()** which has 1 parameter (amount) and <u>returns</u> a <u>double</u> value*
// this method will be defined inside a class e.g the Meal class above
// to use it, you can 'call' it like this:

vat = *myMeal*.**setTax(Bill);** *// assume myMeal is the object created from Meal*

// this passes the value of <u>Bill</u> into the method and picks up the returned tax value from it.

Assessment of CO452 Programming Concepts

1. This module is assessed by coursework. There are 3 parts to this coursework (Part A, B and C). There are study packs for each of the 3 parts. The study packs contain both class exercises (for practice in the sessions) and independent exercises relating to the programming concept being taught that week. There is a project week in Part B that includes a series of tasks that link with each other.

2. **Class exercises will not be assessed**. These are exercises to help you understand the concepts introduced each week and prepare you for the independent exercises. It is a good idea to work together in pairs or groups on the class exercises.

3. **Independent studies (and project tasks in Part B) will be assessed.** The code for these tasks will be assessed on their efficiency, syntax, correct use of concept, and whether the code fulfils the requirements of the task. Some tasks may also require additional documentation such as test plans and algorithms. Please include screenshots of your code running and comments where relevant**. You must complete these independent exercises on your own outside of the session.**

4. Create a logbook (for example: an MS Word document) to document your code. The logbook should contain your designs, algorithms, test plans, source code and results of your work. **This must be submitted electronically through the designated TurnItIn submission point** (your tutor will show you). **If there is a technical problem and you cannot submit through TurnItIn, please email Debra.Harper@bucks.ac.uk explaining the issue, and attaching your logbook. Alternatively speak to someone from the DMM administration office (E4.08).**

5. Your mark for this module will be based on your grades for each of the parts (A, B, C). Below shows the weighting of each of the parts:

Part A: 30% of module mark

Week 1 (4 independent exercises); Week 2 (4 ind. ex.); Week 3 (4 ind. ex.)

Part B: 40% of module mark

Week 5 (TBD), Week 6 (TBD), Week 7 (TBD), Week 8 (Project)

Part C: 30% of module mark

Week 10 (TBD), Week 11 (TBD), Week 12 (TBD)

Grade related criteria for Programming - CO452

A	Where the student has demonstrated clear evidence of an excellent understanding of the theories and principles together with a high degree of analytical accuracy, good design skills, implementing fully tested solutions that show reliability, maintainability, readability and minimal complexity and correct form of presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the seminar sessions and attempt at least 85% of independent studies.*
B	Where the student has demonstrated clear evidence of a good understanding of the theories and principles together with a good analytical ability, good design skills, implementing solutions that show reliability, maintainability, readability and minimal complexity and correct form of presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the seminar sessions and attempt at least 75% the independent studies.*
C	Where the student has demonstrated a reasonable understanding of the theories and principles together with a reasonable analytical ability, design skills, implementing solutions that appreciate the need for reliability, maintainability, readability and minimal complexity and reasonable presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the seminar session and attempt at least 66% of the independent studies.*
D	Where the student has demonstrated an understanding of the theories and principles of analysis, design, implementation and presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the seminar session and attempt at least 50% of the independent studies.*
E	Where the student has made a genuine attempt to acquire the knowledge and skills but requires further application and study to demonstrate an understanding of the theories and principles of analysis, design, implementation and presentation skills. *In order to demonstrate a genuine attempt the student will normally be expected to attend the seminar sessions and attempt at least 40% of the independent studies.*
F	Where the student has clearly not acquired sufficient knowledge and skills and not attempted or coped with the directed study with any degree of competence regarding theories, principles, analysis, design, implementation and presentation skills or where the student has NOT attended for assessment or where the student has copied work from an alternative source.

Module Name and code	**Programming Concepts CO452**
Staff:	Kevin Maher, Carlo Lusuardi, Richard Jones, Nick Day, Based on original work by Brian Ward

Learning Outcomes:

- Analyse a simple requirement in a structured manner
- Design, document, implement and test reliable, maintainable programs as solutions to simple problems
- Use structured techniques of design and implementation and good documentation practice.
- Use software development tools.

WK		LECTURE/TUTORIAL	PRACTICAL
1		**INTRO to Ceebot, VARIABLES, INPUT and OUTPUT**	Ceebot Chapters 1-6
2		**ITERATION**	Ceebot Chapters 7-8; 12-15
3		**SELECTION**	Ceebot Chapters 9-10
4		**WORKSHOP for CW 1 Part A submission next week**	
5		**FUNCTIONS**	Ceebot Chapters 18-19
6		**PARAMETERS**	Ceebot Chapters 20-21
7		**ARRAYS**	Ceebot Chapters 22-23
8		**Ceebot PROJECT**	Ceebot Chapter 24
9		**WORKSHOP for CW 1 Part B submission next week**	
10		**C# 1 Input and Output**	C# Intro Directed Study Pack: Unit 1
11		**C# 2 Sequence, Selection, Iteration**	C# Intro Directed Study Pack: Unit 2
12		**C# 3 Classes, Objects and Methods**	C# Intro Directed Study Pack: Unit 3
		Christmas Break	
13		**WORKSHOP for CW 1 Part C submission next week**	
14			

Course Texts:
Comprehensive Course Notes are provided

Bradley & Millspaugh, ***Programming in C#***, 2010, pub: McGraw Hill
Deitel & Deitel, ***Visual C# 2010 How to Program***, 2011, pub: Pearson