Worksheet 10.1 / On patrol

Before starting with this worksheet, you should have successfully completed the Ceebot environment "SB 8: Conditional loops" / „Pathfinder" exercise.

Ceebot environment

"SB 10: Arrays and files" / "On patrol"

Mission

Our headquarter inside the crater of an extinct volcano needs to be protected by a robot on guard. The robot perpetually has to patrol along a route right on the crater's rim, indicated by a series of 25 waypoints (category `waypoint`). At the end of each round, the robot needs to reload its battery at the power station (category `PowerStation`).

Concept

In Ceebot environment "SB 8: Conditional loops" / „Pathfinder", we have already programmed the robot to follow a path given by a sequence of waypoints. In our present mission, the task to repeatedly follow this path is slightly more complicated as the waypoints vanish as soon as the robot drives onto them. Thus, after completion of the first round the waypoints are no longer available.
So, the robot has to determine the waypoints' positions during its first round and memorize it for the subsequent rounds.
We already know that all kind of information is stored in

- variables of an adequate type or
- _________________ of an adequate class (see Worksheet 6.2)

In order to store the positions of all waypoints, we could use 25 different instances of the class _____________ and could name them `pos1, pos2, ... pos25`. As all of these 25 instances would have to be treated in the same way – during the first round, the position of a waypoint has to be stored, and during the subsequent rounds, the robot has to drive to the stored positions – we would have to write basically the same commands 25 times. As this method would be is very cumbersome, it is much better to store the positions in an array.

An **array** contains several variables of the same type (or several instances of the same class) and subsumes them under a common identifier.
Just as every variable and every instance of a class, an array has to be **declared** before it can be used in the program:

`Type_of_variable Name_of_array[Size_of_array]` or
`Name_of_class Name_of_array[Size_of_array]`

In our example, we want to store the positions of the 25 waypoints in an array named `pos`, which is declared by

```
    point pos[25];
```

The position of the first waypoint is stored in `pos[0]`, the position of the second waypoint is stored in `pos[1]` and so on. The position of the 25th waypoint is stored in `pos[` _______ `]`.

The individual **elements** of an array are consecutively numbered, beginning with the number 0. This number is set between brackets and is called the **index of the array**.

Beginners in Java und C++ (and thus also in Ceebot) are often confused by the fact that the consecutive numbering of an array's elements starts 0 instead of 1. This often causes bugs in programs. For example, the last element of an array `x` with 20 elements is not `x[20]` but `x[____]`. Arrays offer the possibility to process all elements of the array in the same way by using a loop (usually a `for`-loop) to succesively increase the index of the array. As an example, the loop

```
for (int i=0;i<=99;i=i+1) a[i]=a[i]*3.14159;
```

results in a multiplication of all 100 numbers stored in array `a` by π.

Use an array in a program if several variables of the same type (or several instances of the same class) have to be processed in the same way. Use a loop to process the elements of the array.

Solution

Listing 10.1.1 – On patrol	
Code	**Description**
`extern void object::Patrol() {`	
`  ______________ pos[_____];`	Declaration of the array `pos`
`  int counter=0;`	______________________________
`  object item=radar(______________);`	Locate the first `WayPoint`.
`  while(item!=_______________) {`	Repeat as long as a `WayPoint` exists:
`    pos[counter]=_________________;`	Store the position of the `WayPoint`.
`    goto(item.position);`	______________________________
`    counter=________________________;`	Increase counter by 1.
`    item=____________________________;`	Locate the next `WayPoint`
`  }`	
`  item=_________________________________;`	Locate the `PowerStation`.
`  ____________________________________;`	Move to the power station.
`  wait(7);`	Wait for battery to recharge.
`  for(int i=0;i<=_______;i=______) {`	Second round: Successively drive to...
`    goto(pos[i]);`	... all stored positions.
`  }`	
`  item=radar(_____________________);`	_________________________________
`  goto(_______________________);`	
`  ______________________________;`	Wait for battery to recharge.
`}`	

Explain why we need the variable `counter`:

____

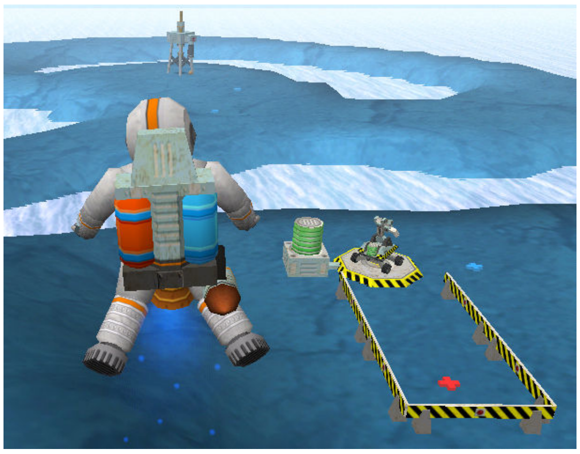
____

Improving the program

1. We have programmed the robot to finish two rounds, but actually it should repeat its patrol over and over again. If we want a certain procedure to be repeated over and over again, we need an ______________________ loop. Insert this loop properly.
2. At the end of each round, the robot waits longer than necessary for the battery to be recharged. Instead of the command `wait(7)`, use a loop to wait just as long as necessary to recharge the battery (which means to wait until `this.`_____________.____________ `== 1`, see Worksheet 7.1).
3. We even can improve the program in another respect: It is not exactly smart to store the positions of the 25 waypoints in the array `pos,` whereas the position of the `PowerStation` is not stored at all. Add one more element to the array which stores this position at the end of the first round. As a consequence, the loop for the consecutive rounds then needs to be executed 26 times.

Ceebot environment

”SB 10: Arrays and files“ / „Maze“



Mission

In the middle of a lake, we have found a mine for titanium ore and use the derrick in the background (categorie `Derrick`) to exploit it. The robot needs to pick up a piece of titanium ore (category `TitaniumOre`) produced by the derrick and to transport it to the red cross at the stockyard. (category `TitaniumSpot`).

Solution

```
Maze
As long as a waypoint exists
    Locate the waypoint and move to its position
    Store the waypoint's position
Locate the derrick and move to its position
Grab the TitaniumOre
Count the index of the stored positions
in reverse order
    Move to the position with the respective index
Locate the power station and move there
Wait until the robot's battery is recharged
Locate the TitaniumSpot and move to its position
Drop the TitaniumOre
```

As in Ceebot exercise ”SB 10: Arrays and files“ / ”On patrol“, the robot drives to the derrick if he moves from one `WayPoint` to the next by use of the `goto` command. The waypoints’ positions have to be stored (because the waypoints vanish). When driving back from the derrick, the robot needs to move from one stored position to next, but **in reverse order**. The program should work with every number of waypoints.

The Nassi-Shneiderman diagram to the left represents a first raw outline for the solution – the first step of a Top-down design process. Refine the diagram using the same phrases as in Listing 10.1.1. The following hints might be helpful.

Hints

- As we want the program to work with an arbitrary number of waypoints, we must not pose a limit to the size of the array `pos`. Just as Java, Ceebot offers an easy possibility to declare an array without predefining its actual size – in the declaration, we just omit the value for the array's size:

  ```
  point pos[];
  ```

  In Listing 10.1.1, the variable ________________ is increased by one each time when the robot passes a `WayPoint`. At the beginning, this variable is set to ______, and thus, at the end of the first round, the variable stores the total number of all waypoints.
- In order to drive from the derrick back to the stockyard, we need to count the index of array `pos` in reverse order. This is best done by the loop

  ```
  for (int i=anzahl-1;i<=0;i= __________)
  ```

  The position of the last `WayPoint` is stored in `pos[anzahl-1]` because the position of the first waypoint is assigned to `pos[0]` (and not `pos[1]`).
- The Nassi-Shneiderman diagram shown above contains several almost identical steps "Localize an object and move to its position". For this procedure, create a function with a single parameter, namely an integer value describing the category of the object to be localized:

  ```
  void goto_next (int cat)
  { … }
  ```

Further exercises

1. Extend the program in Ceebot environment "SB 10: Arrays and files" / "Maze" in such a way that three pieces of titanium ore are transported to the stockyard.
2. Write a program which first stores 10 random numbers in an array and afterwards uses `printf` to print these numbers on the screen.
3. Extension of exercise 2: Additionally print the sum of these 10 random numbers on the screen.
4. In Ceebot environment „SB 10: Arrays and files" / "Cramped confines", you can see three empty batteries behind a small building. They have to be recharged at the `PowerStation` and brought to the stockyard. However, the weeds around the station have proliferated and give the robot a hard time finding its path. The waypoints lead the robot first to the batteries, afterwards to the `PowerStation` and finally back to the stockyard. The robot has to drive this path three times.

   Hints:
   - Due to the narrow gaps between buildings and plants, the `goto` command does not work sufficiently. Instead, the robot needs to drive from one waypoint to the next using the commands

     ```
     turn(direction(...));
     move(distance(...,...));
     ```

   After each movement, the robot has to check out the following two situations:
   - If the robot finds a battery in a distance of less than 5 meters, it needs to grab the battery.
   - If the robot is at the `PowerStation`, it needs to wait for 7 seconds until the empty battery is recharged.