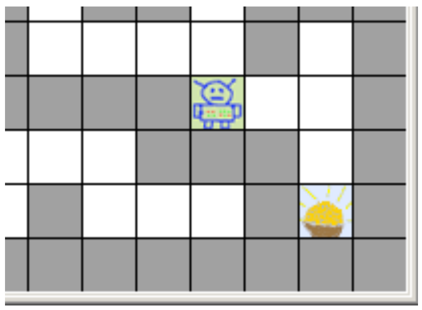
EEN118 LAB TWELVE

This is a continuation of lab eleven. Remind yourself of the assignment for that lab, and the instructions that came with it, and carry on here.

7. *Better Graphics*

There are two useful library functions that will display graphics from image files. You should have proper pictures of the robot and the treasure. You can even have nicer backgrounds for the maze squares if you want:



One function is called `image_from_file()`. It takes as its one parameter the name of a file, which may be any `.gif`, `.png`, or `.bmp` file, and extracts the image data from it. It returns a strange value whose type in C++ is "`image *`".

The second function is called `draw_image`. It takes three parameters: one of those "`image *`" things I just mentioned, and two `int`s that specify the x and y coordinates of the position in the window where the image should be drawn (top left corner).

It is a very simple process. When the program is first started, it might have a statement or two that look something like this:

```
image * robot_icon = image_from_file("robot.gif");
image * gold_icon = image_from_file("c:/progs/lab12/gold.bmp");
```

and later, when it is time to draw the robot, you would simply write

```
draw_image(robot_icon, x, y);
```

where of course `x` and `y` are replaced by the appropriate position coordinates.

8. *A Foolish Robot*

Your robot does not move except under your direct command. Add another command letter, perhaps '`A`', that puts him in Automatic mode. When in automatic mode, the robot should repeatedly select a random neighbouring empty square, and move into it, until he happens upon the treasure.

One way to achieve this would be simply to stop calling `wait_for_key_typed()` once the robot is in automatic mode, but then you would never be able to get him out of automatic mode. Fortunately, the `wait_for_key_typed()` has a little bit of extra functionality. It can take one optional parameter, a floating point number telling it how long (in seconds) to wait for the user to type a key.

This statement

```
        char c = wait_for_key_typed(0.0);
```

will not wait at all. If the user has already pressed a key, then everything is fine, and `c` is set to that key's ASCII code as usual. Otherwise, `c` is just set to zero instantly.

Add another letter command, perhaps '`M`' for Manual, that turns off automatic mode, and puts the robot back obediently under your command. You will probably find that 0.0 seconds isn't the ideal maximum wait for keyboard input.

9. *Retracing Steps*

In order to avoid getting lost, it is important for explorers to remember the path they followed, so that they can always reliably retrace their steps. Just like the Apollo Eleven astronauts who left a trail of breadcrumbs behind them so they could find their way back home.

Give your explorer this ability. The easiest way is to create a second array, the same size as the maze map array. Every time the robot explorer moves into a square, how he got there (which direction he moved in) is stored in the corresponding position in the second array. If ever the robot needs to go back, he just looks in the new array to see what direction he came in, and moves in the opposite direction.

Add a '`B`' command (for Back) to your program. Each time '`B`' is typed, the robot should retrace his steps by one square. If you keep typing '`B`', he should eventually get all the way back to his starting position. Of course it won't work if you've gone in a circle. Important: the *how-I-got-here* array should only be updated after normal movements, not after '`B`' commands. Otherwise two '`B`'s in a row will have no overall effect.

10. *Semi-Intelligent Behaviour*

It is fairly easy to make your robot explore the maze totally unaided, and find a path to the treasure (if a path exists). This is the strategy:

- Whenever the robot finds himself anywhere in the maze, he should examine each of the four neighbouring squares. If there is any neighbouring square that he has never been in before, he should move into it.
- If there is more than one unexplored neighbouring square, just pick any one.
- If ever the robot finds there are no neighbouring squares that he has never visited before, then he should retrace his steps by one square, and continue the procedure from there.
- If ever the robot finds himself at the treasure square, he has solved the maze, and should stop and be happy.

- If ever the robot finds himself trying to retrace his steps back from the starting square, that means that the maze has no solution, and he would probably go on a rampage.

Automate your robot, so that he follows this strategy whenever you put him in automatic mode (you can make up your own strategy if you like, but it won't be easy to find another one that always works) and finds his way from '+' to '$' without any help.

This strategy is what makes this a reasonable robotics experiment. Even though the program has posession of a complete map of the maze, it doesn't cheat by using it too much. At every step it only considers the immediately neighbouring squares when deciding what to do next, working under the same restrictions as a real robot lost in a maze, only having the information that its own sensors can gather.

11. *The Yellow Brick Road*

Once the robot successfully reaches the treasure square, it would be nice to make the whole path he took clearly visible, perhaps by colouring those squares in a way that stands out. Of course, we would only want to see the squares he visited that were on the successful path, not ones that he later backed out of by retracing his steps.

The information required to do this very easily is already recorded in your program. Try to work out what to do, and then do it.

12. *Monsters*

Put a monster at a random position in the maze. Every time the robot makes a move, the monster also makes a move, heading towards the robot's position one square at a time, not being allowed to walk through walls of course. If the monster reaches the robot, you can decide what happens.

To make it more of an interesting game, you could make the monster move at its own pace instead of just taking turns. You could perhaps have the monster move one square every second, regardless of what your robot has done.

You could have different monsters with different degrees of intelligence: the dimmest ones could just move at random, cleverer ones could rush for the treasure and try to ambush your robot on the way. You can think up other monster plans too.

You probably have a better idea of what makes a good video game than I have, so just get on with it. Make it good. Try to make it into something that a person could enjoy playing with for a little while.