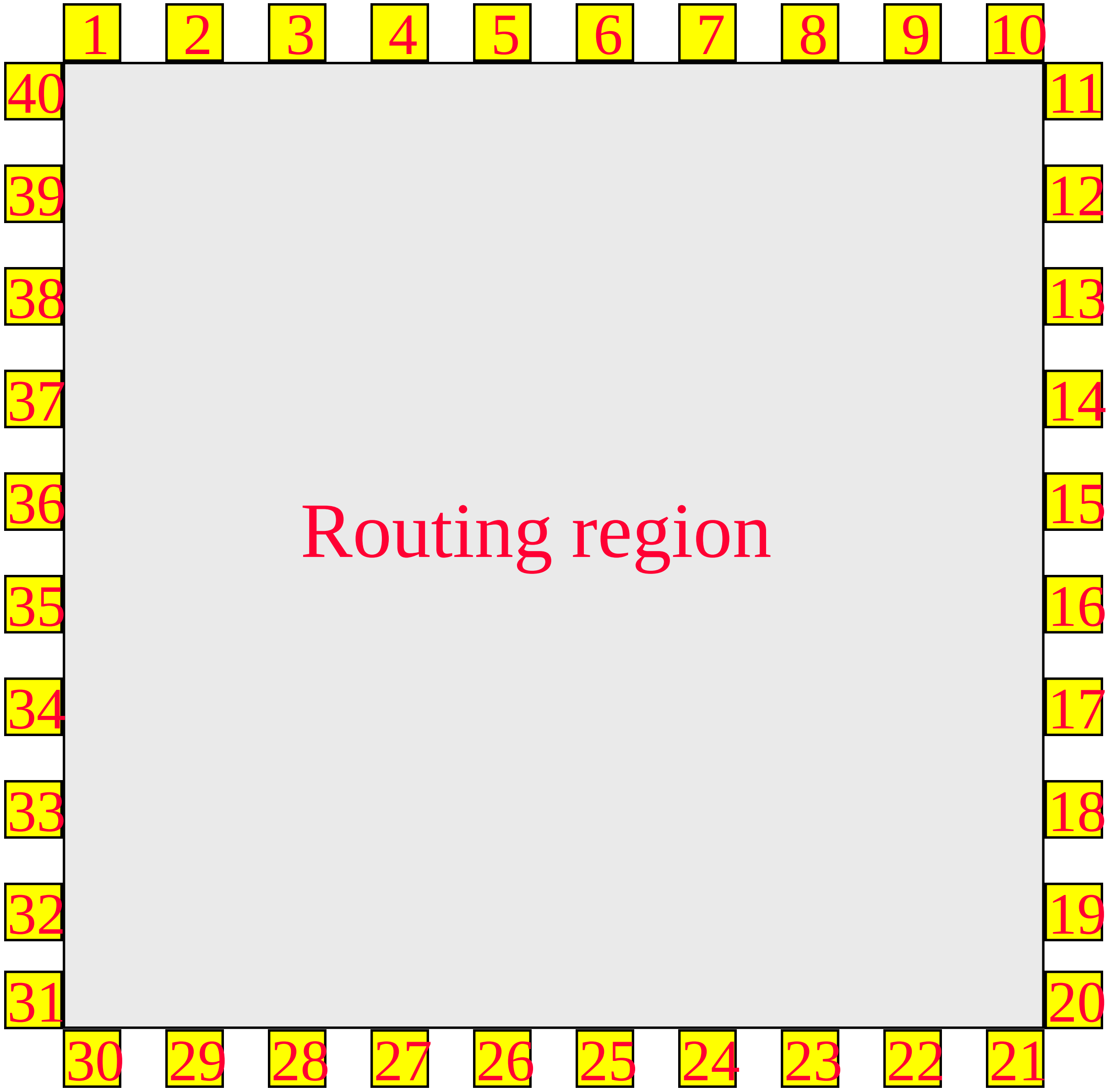
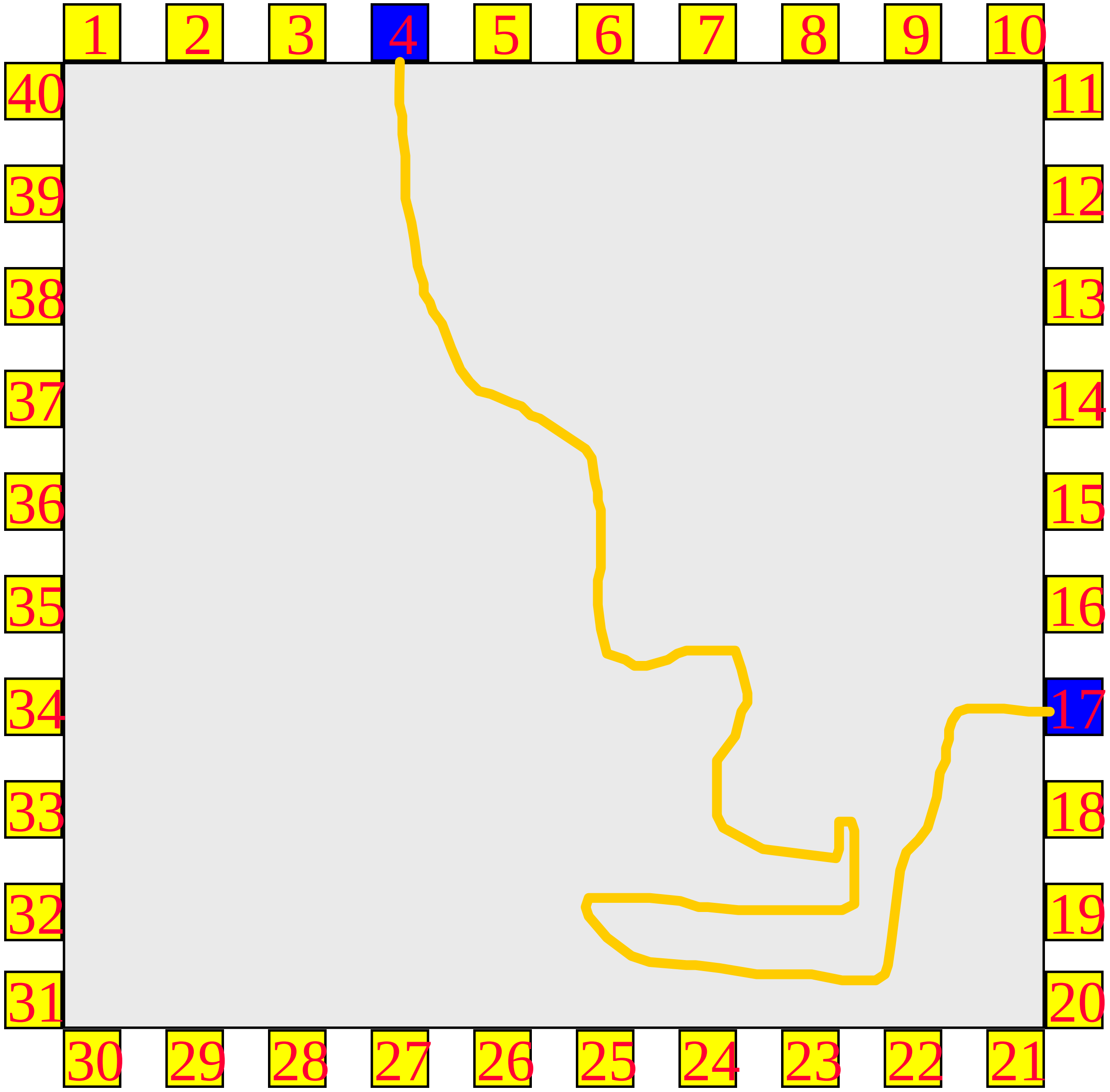
Switch Box Routing



Routing A 2-pin Net

Routing for pins 1-3 and 18-40 is confined to lower left region.



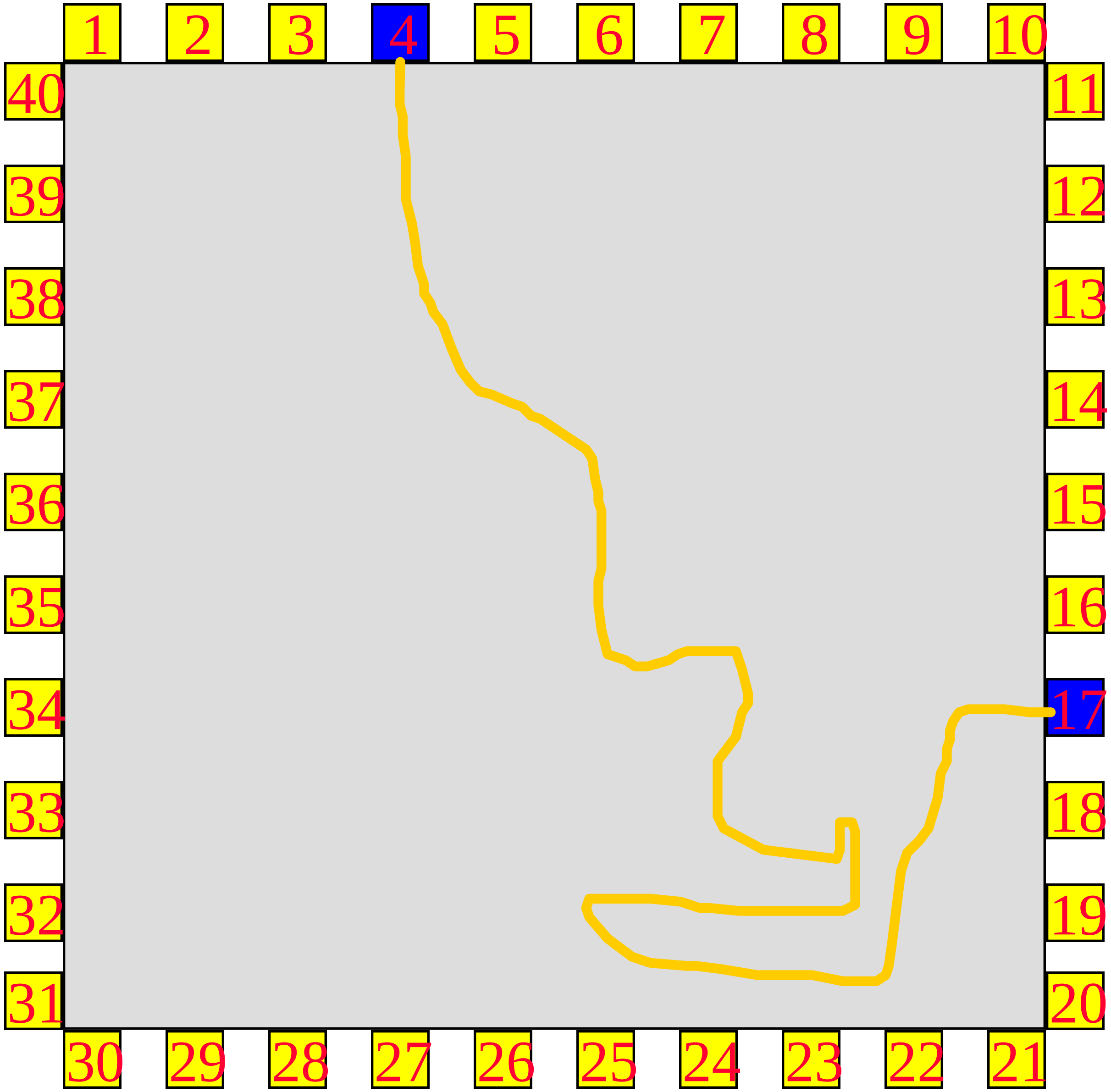
Routing for pins 5 through 16 is confined to upper right region.

Routing A 2-pin Net

(u,v), u<v is a 2-pin net.

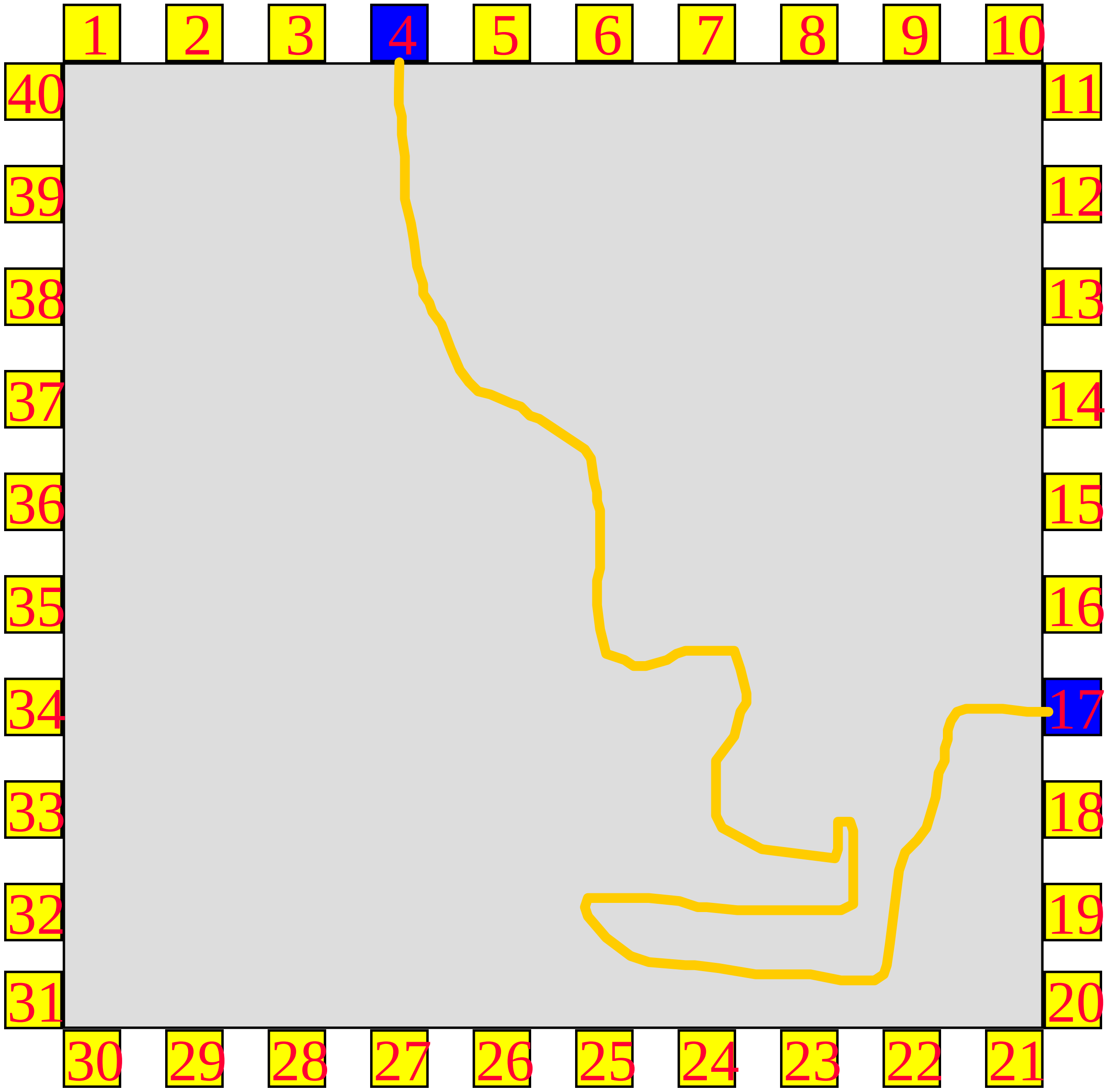
u is start pin.

v is end pin.



Examine pins in clock-wise order beginn-ing with pin 1.

Routing A 2-pin Net



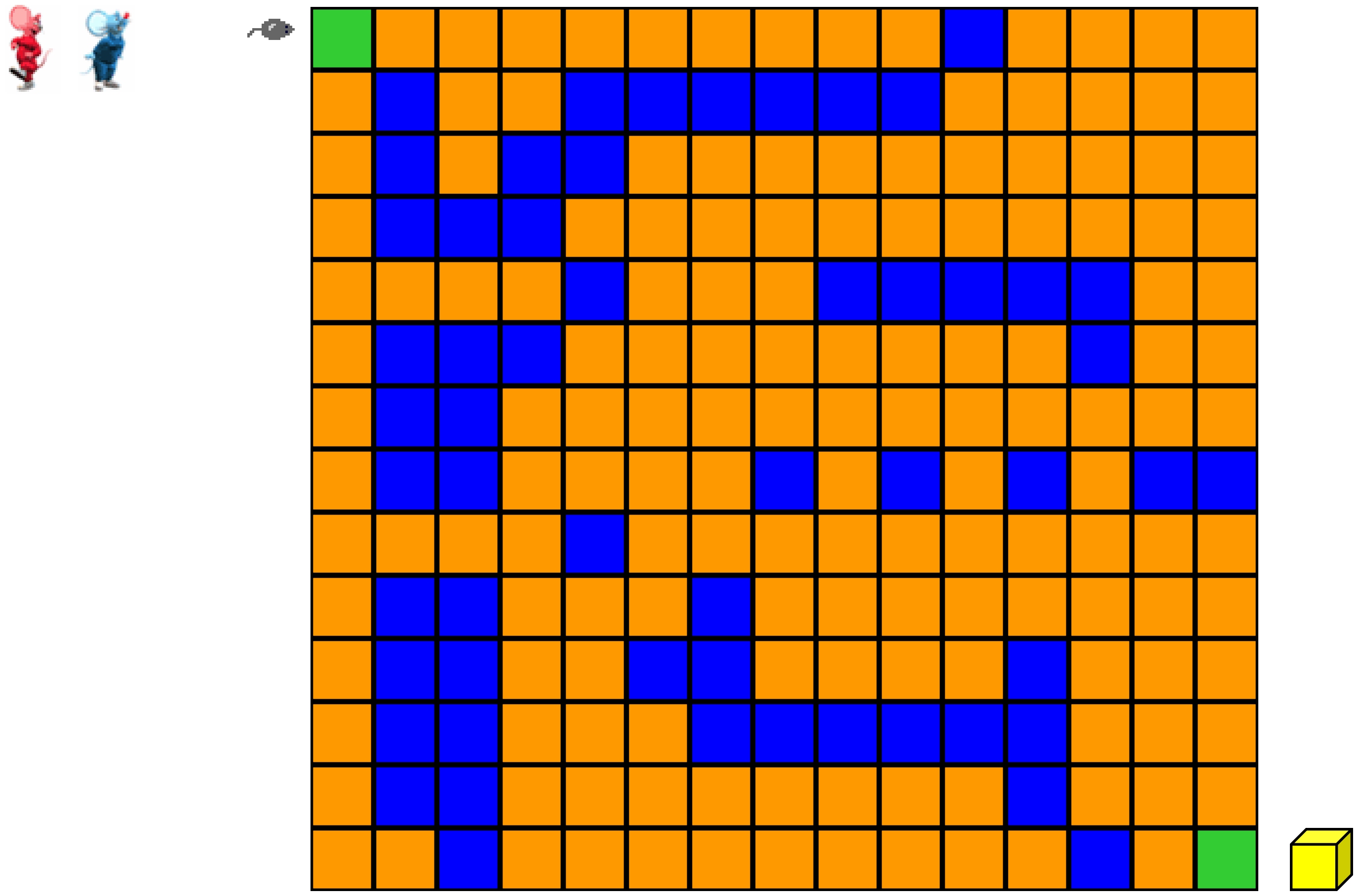
Start pin => push onto stack.

End pin => start pin must be at top of stack.

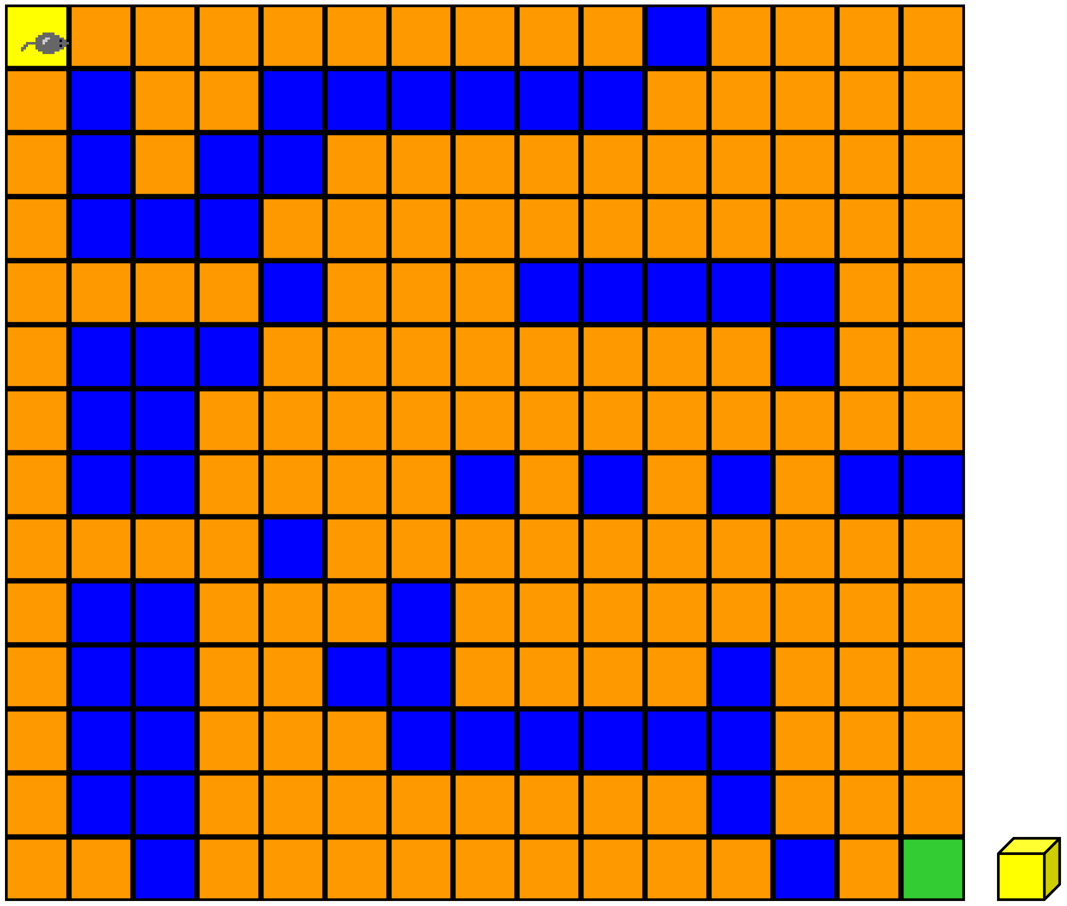
Once, long ago in a land far away, there lived four little characters who ran through a maze looking for cheese to nourish them and make them happy.

Who moved my cheese?

Rat In A Maze

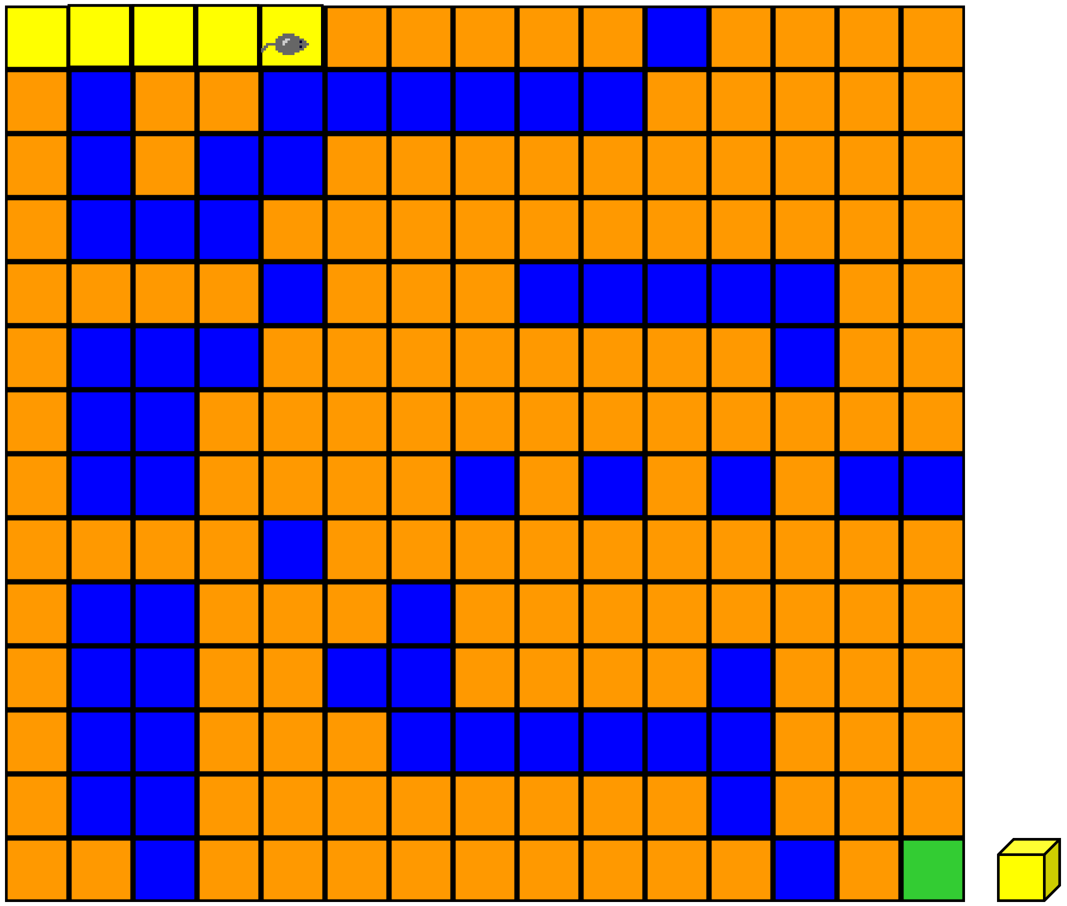


Rat In A Maze



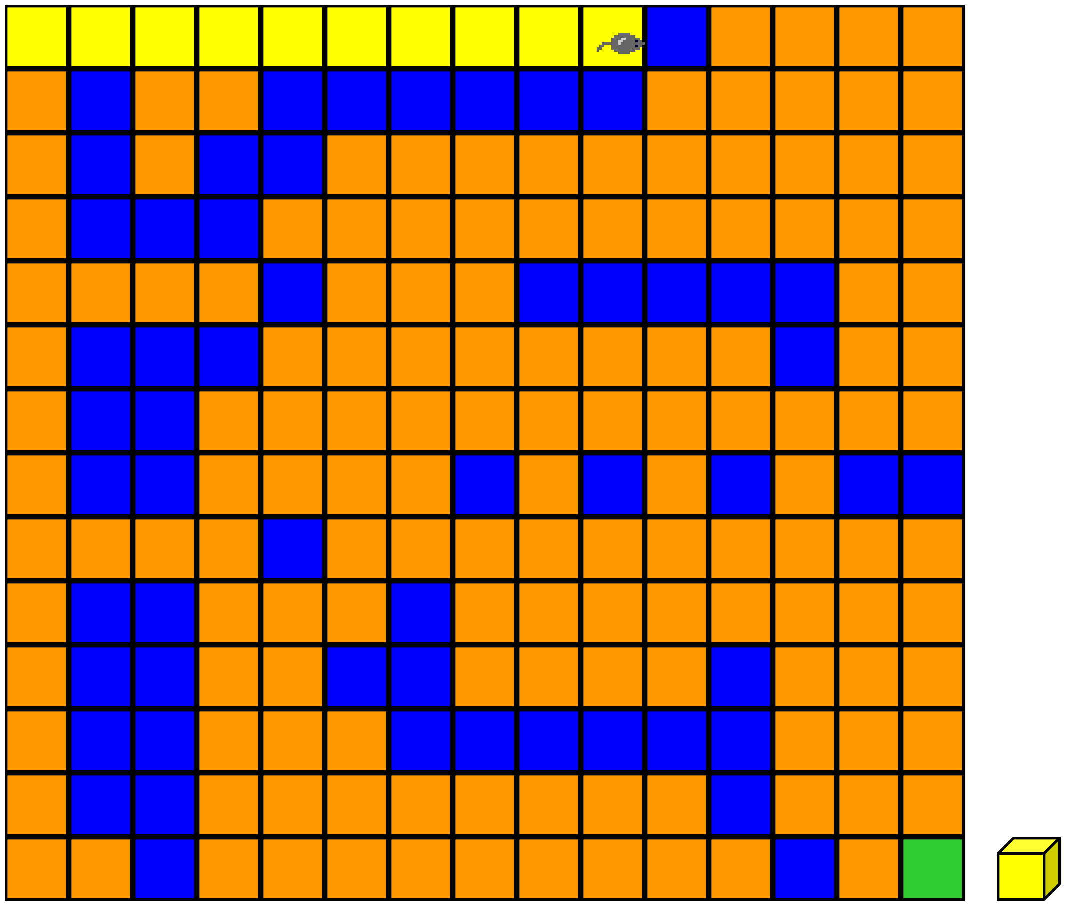
- Move order is: right, down, left, up
- Block positions to avoid revisit.

Rat In A Maze



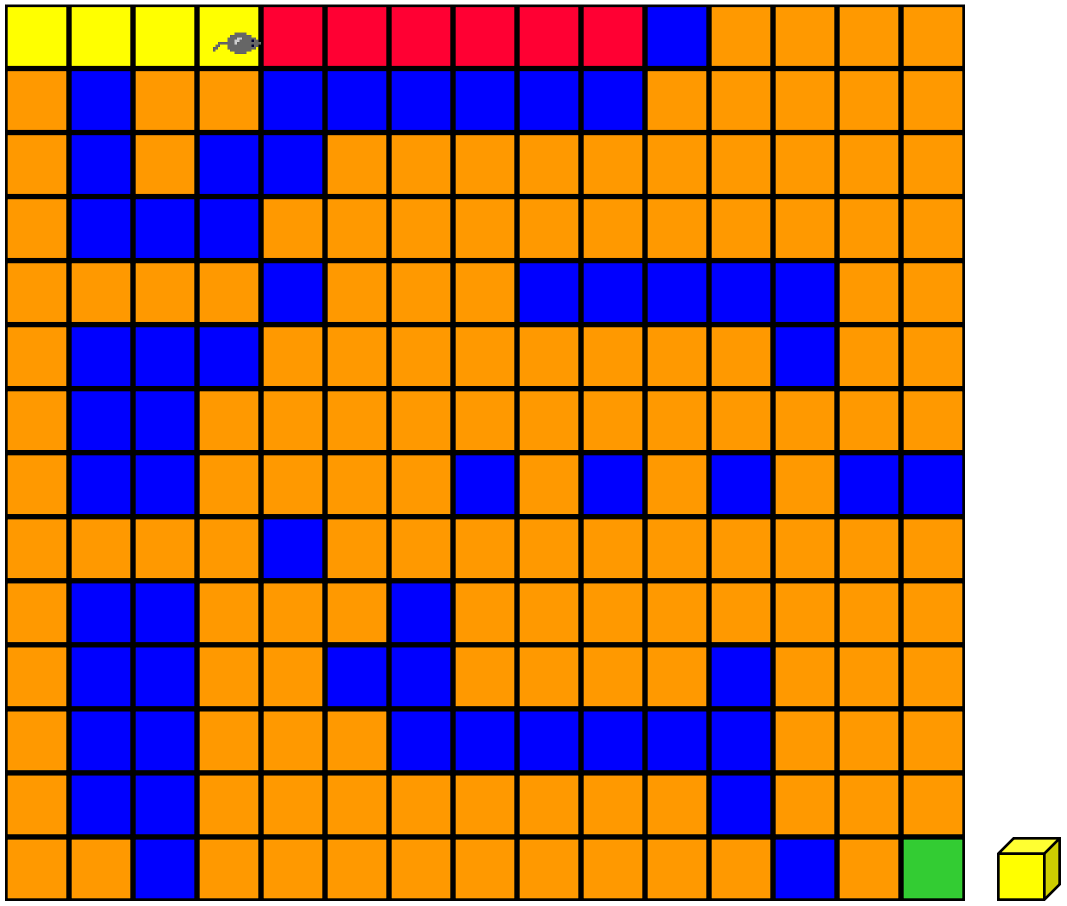
- Move order is: right, down, left, up
- Block positions to avoid revisit.

Rat In A Maze



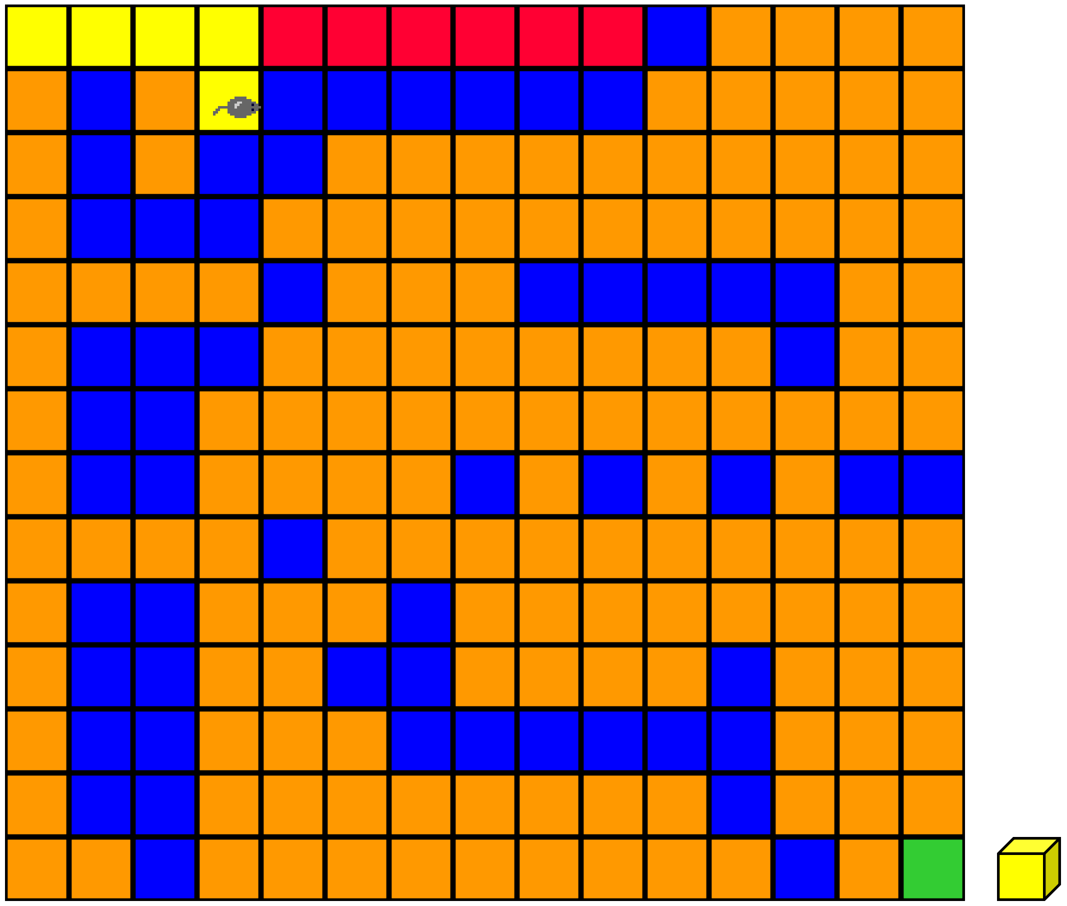
- Move backward until we reach a square from which a forward move is possible.

Rat In A Maze



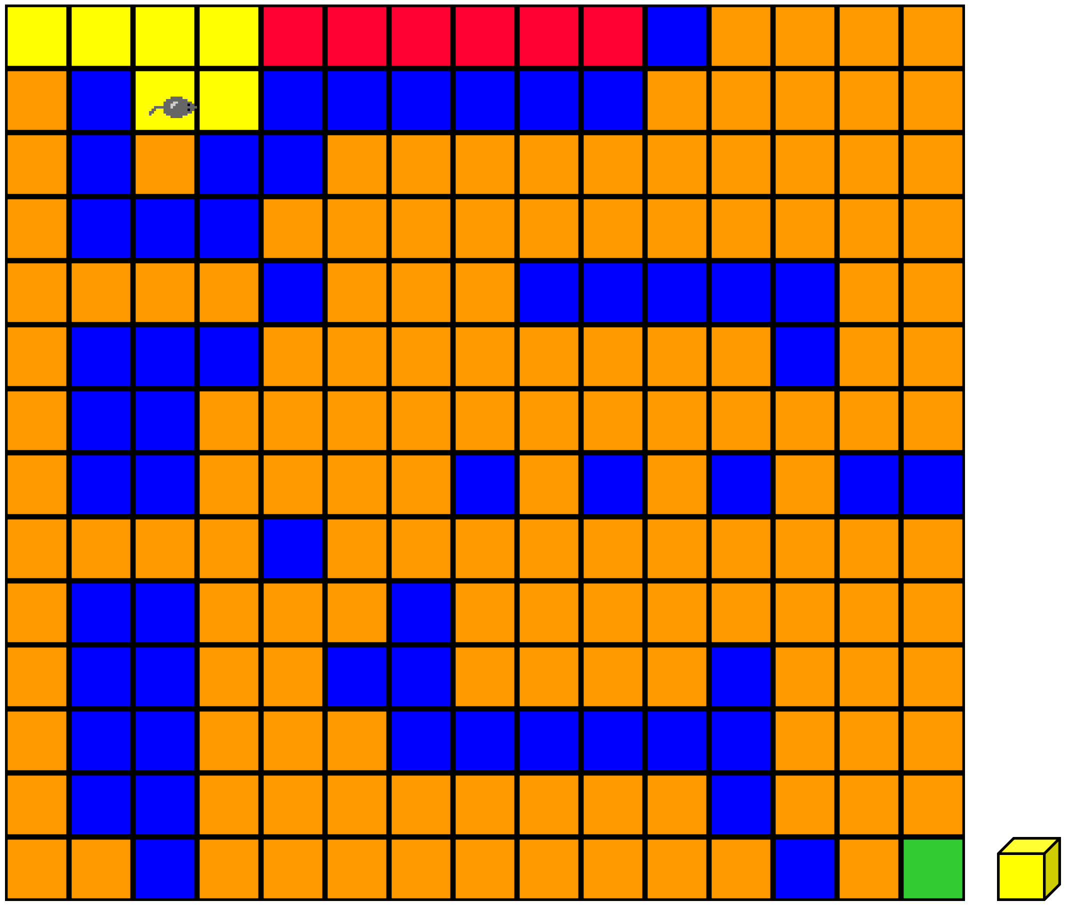
- Move down.

Rat In A Maze



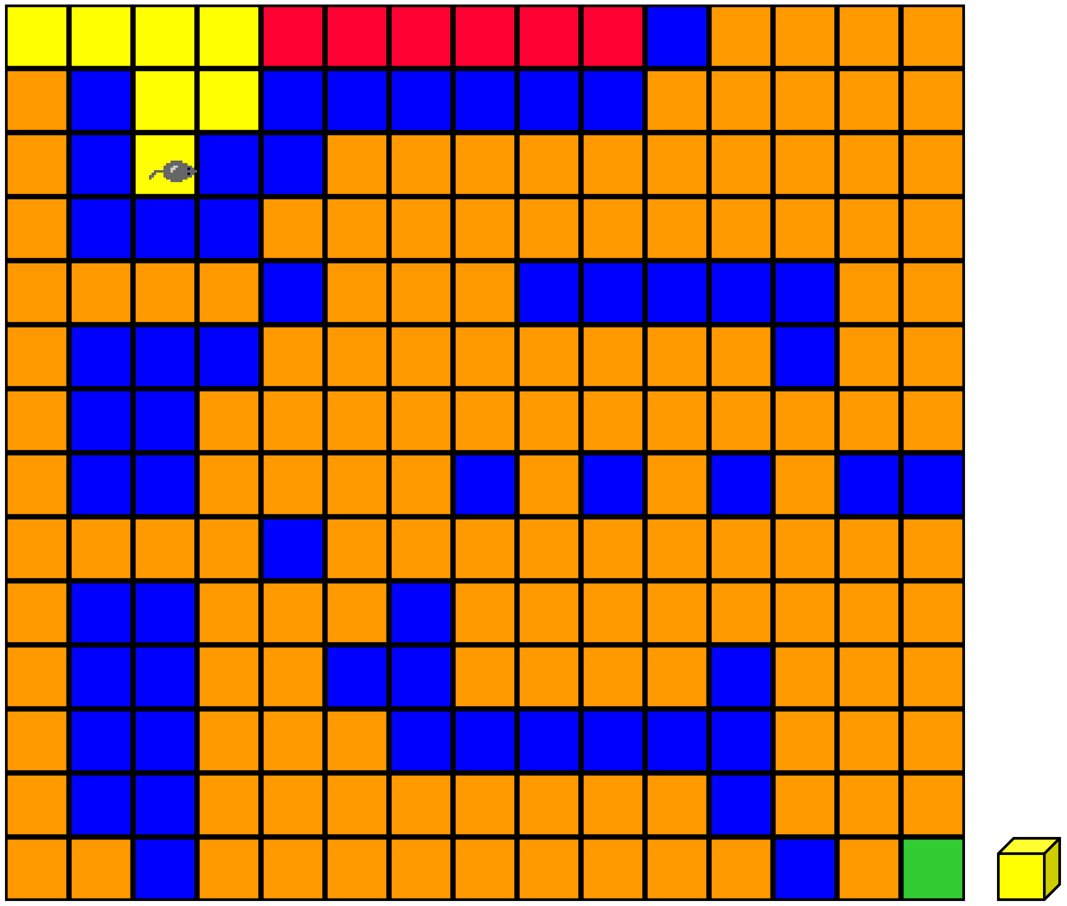
- Move left.

Rat In A Maze



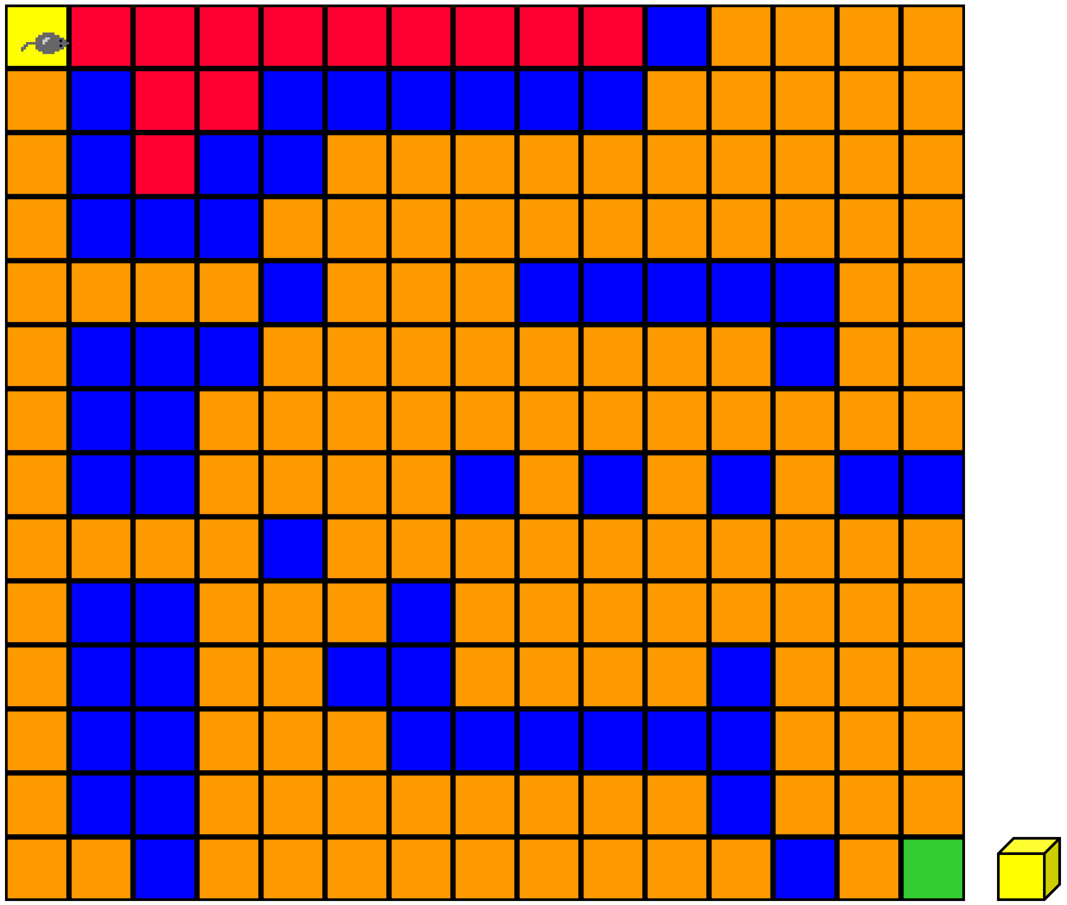
- Move down.

Rat In A Maze



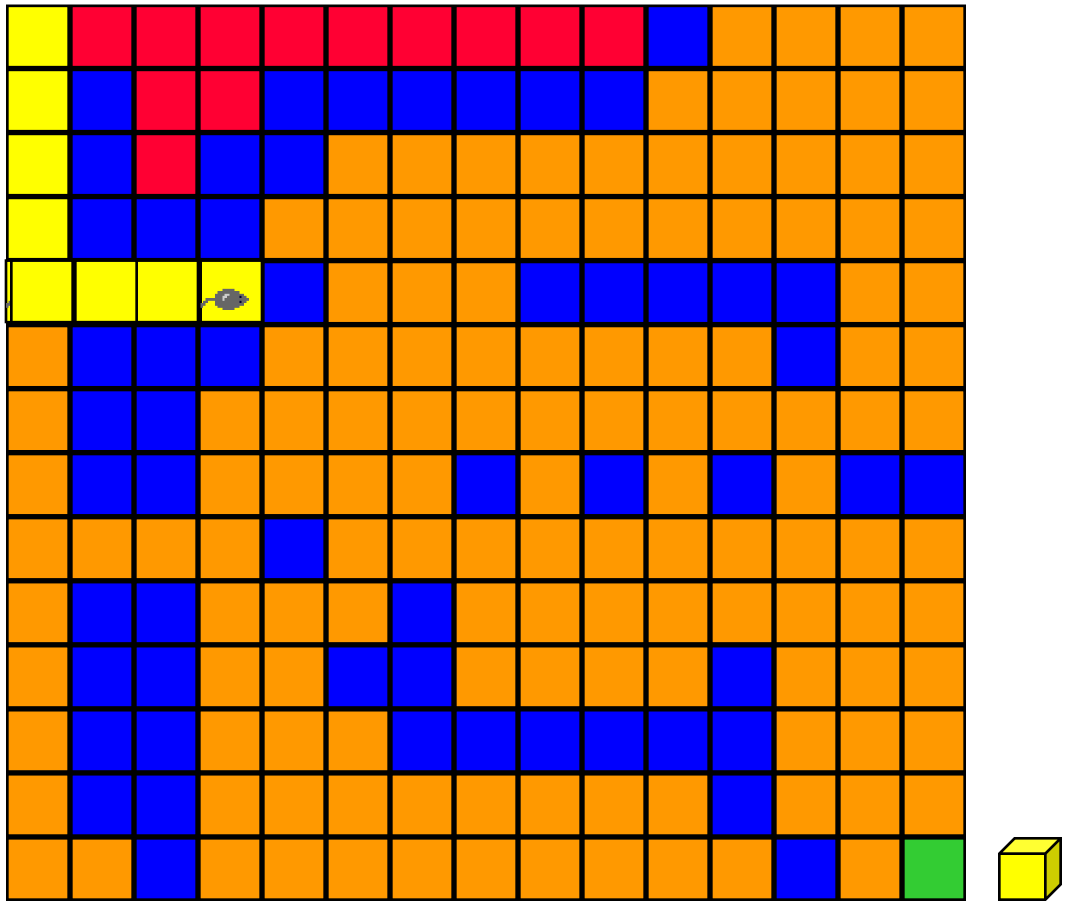
- Move backward until we reach a square from which a forward move is possible.

Rat In A Maze



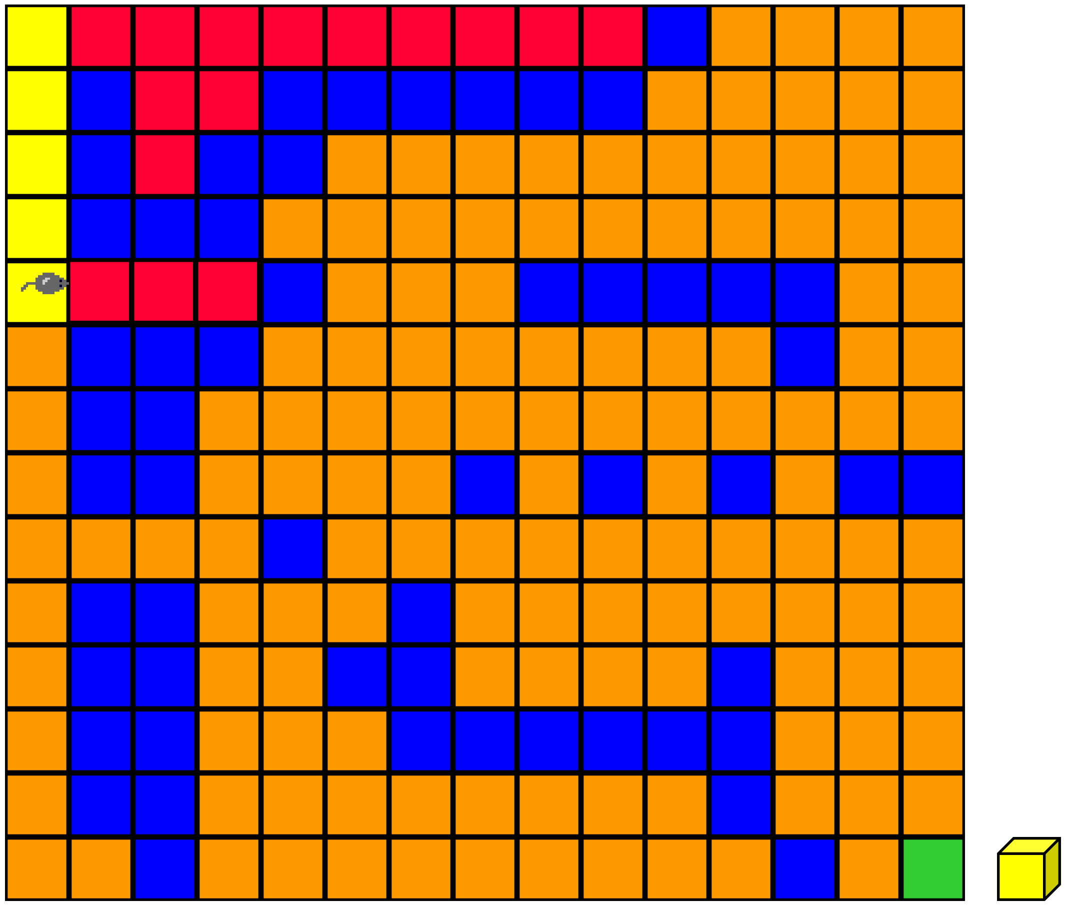
- Move backward until we reach a square from which a forward move is possible.
- Move downward.

Rat In A Maze



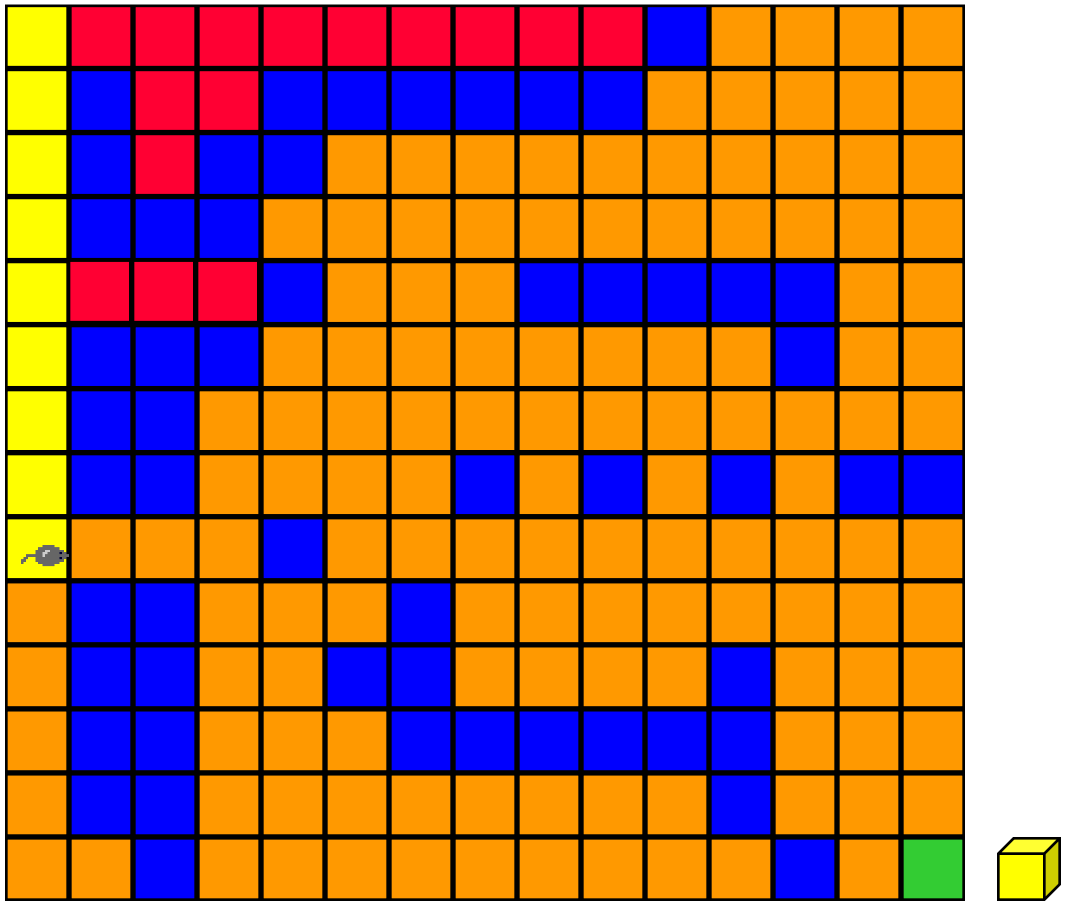
- Move right.
- Backtrack.

Rat In A Maze



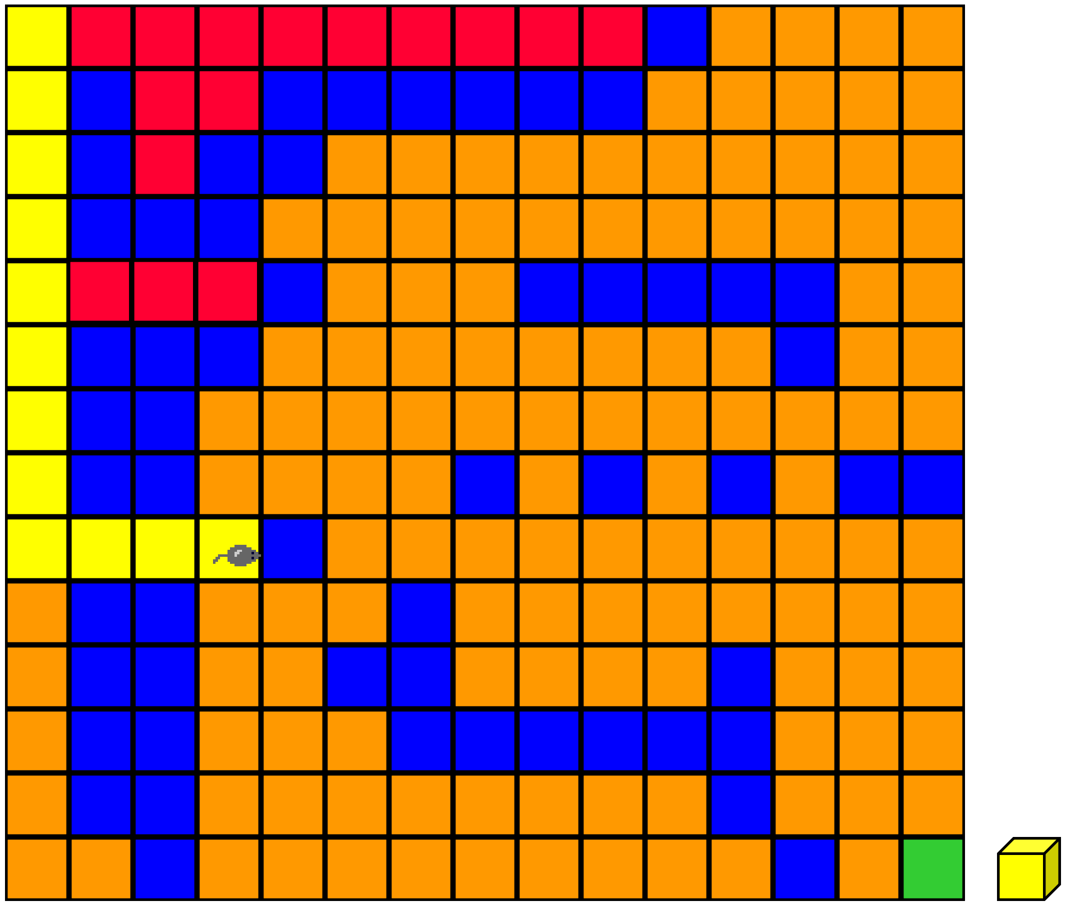
- Move downward.

Rat In A Maze



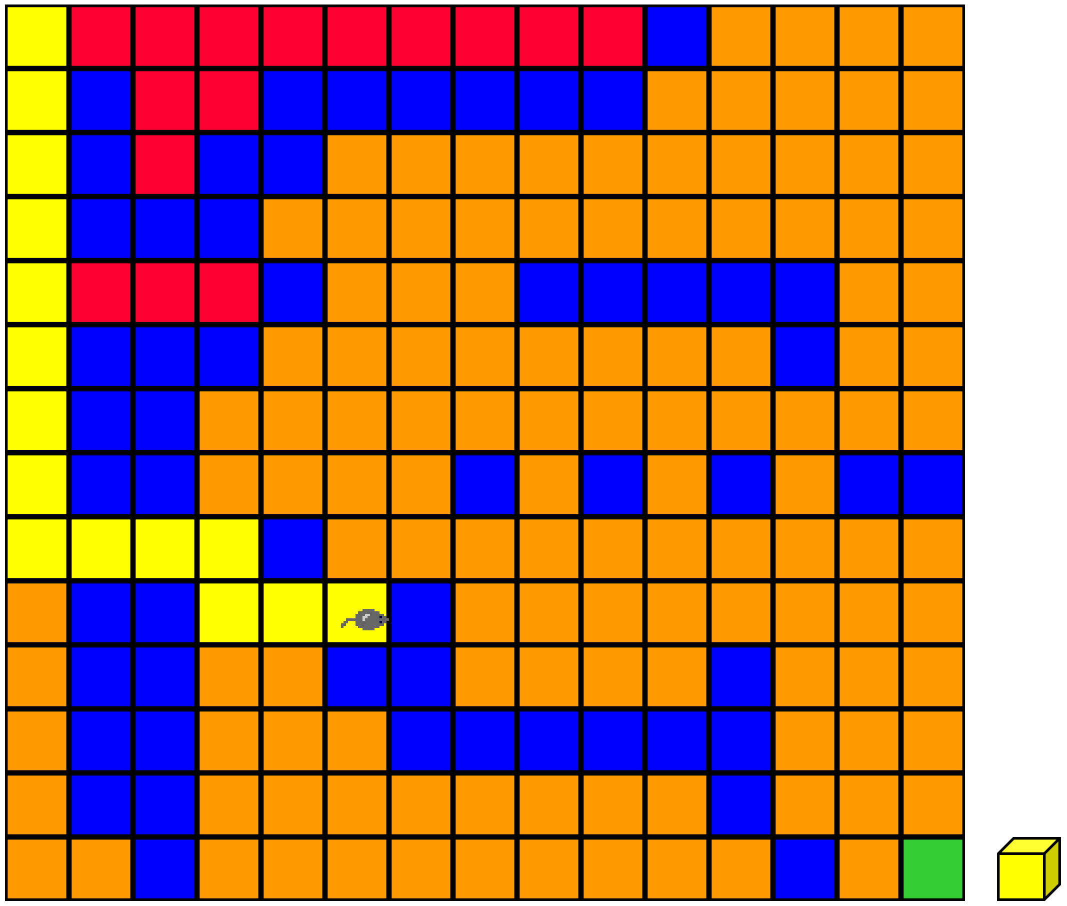
- Move right.

Rat In A Maze



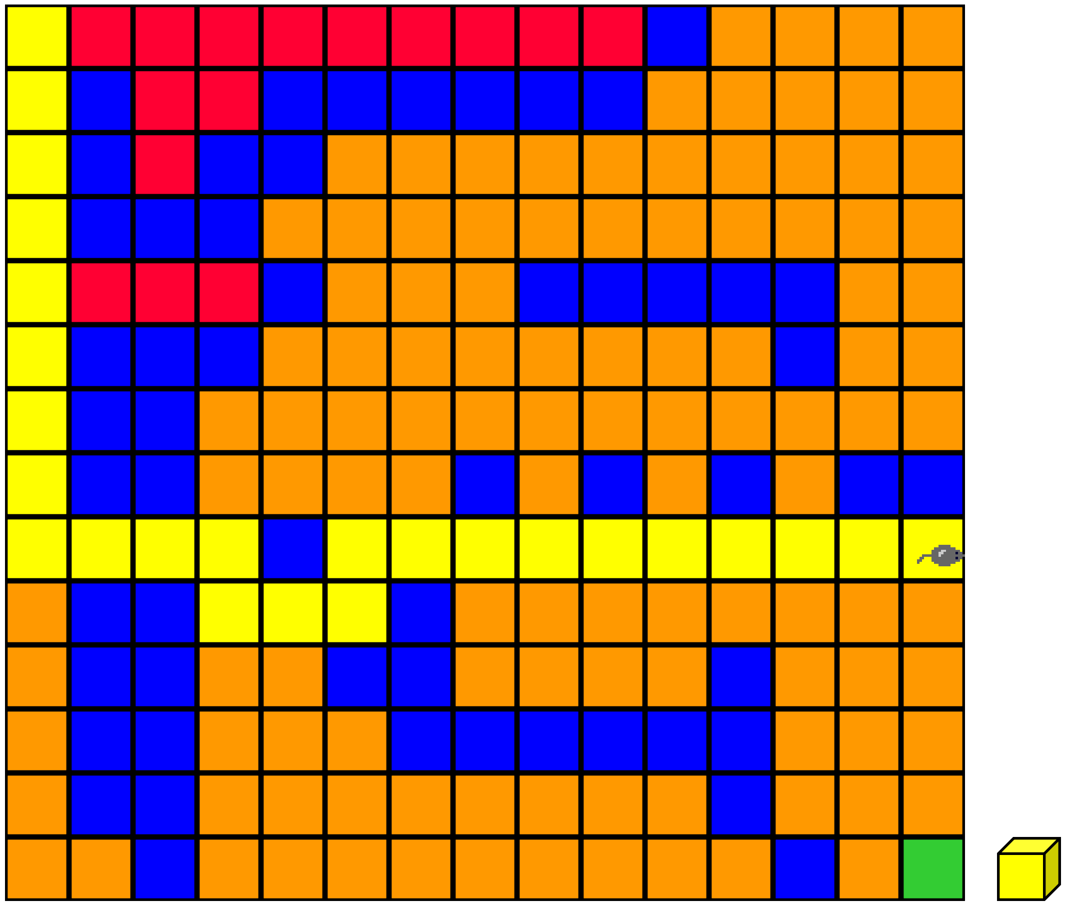
- Move one down and then right.

Rat In A Maze



- Move one up and then right.

Rat In A Maze



- Move down to exit and eat cheese.
- Path from maze entry to current position operates as a stack.

Modular structure of program

- Welcome
- Input
- Find path
- Output

InputMaze()

```
bool InputMaze()
{// Input the maze.
   cout << "Enter maze size" << endl;
   cin >> m;
   Make2DArray(maze, m+2, m+2);
   cout << "Enter maze in row major order" << endl;
   for (int i=1; i<=m; i++)
      for (int j=1; j<=m; j++) cin >> maze[i][j];
   return true;
}
```

FindPath()

```
bool  FindPath()
{// Find a path from (1,1) to the exit (m,m).
 // Return true if successful, false if impossible.
 // Throw NoMem exception if inadequate space.
   path = new Stack<Position>(m * m - 1);
   // initialize offsets
   Position offset[4];
   offset[0].row = 0; offset[0].col = 1; // right
   offset[1].row = 1; offset[1].col = 0; // down
   offset[2].row = 0; offset[2].col = -1; // left
   offset[3].row = -1; offset[3].col = 0; // up

   // initialize wall of obstacles around maze
   for (int i = 0; i <= m+1; i++) {
      maze[0][i] = maze[m+1][i] = 1; // bottom and top
      maze[i][0] = maze[i][m+1] = 1; // left and right
      }
```

```
  Position here;
  here.row = 1;
  here.col = 1;
 maze[1][1] = 1; // prevent return to entrance
  int option = 0; // next move
  int LastOption = 3;

// search for a path
  while (here.row != m || here.col != m) {// not exit
     // find a neighbor to move to
     int r, c;
     while (option <= LastOption) {
        r = here.row + offset[option].row;
        c = here.col + offset[option].col;
        if (maze[r][c] == 0) break;
        option++; // next option
        }
// was a neighbor found?
    if (option <= LastOption) {// move to maze[r][c]
      path->Add(here);
      here.row = r; here.col = c;
      // set to 1 to prevent revisit
      maze[r][c] = 1;
      option = 0;
      }
    else {// no neighbor to move to, back up
      if (path->IsEmpty()) return false;
      Position next;
      path->Delete(next);
      if (next.row == here.row)
        option = 2 + next.col - here.col;
      else option = 3 + next.row - here.row;
      here = next;
      }
  }
```

OutputPath()

```
void OutputPath()
{// Output path to exit.
  cout << "The path is" << endl;
  Position here;
  while (!path->IsEmpty()) {
    path->Delete(here);
    cout << here.row << ' ' << here.col << endl;}
}
```

main()

```
void main(void)
{
   welcome();
   InputMaze();
   if (FindPath()) OutputPath();
   else cout << "No path" << endl;
}
```