CSP301
Design Practices in Computer Science

Design Document for Programming Assignment #2

By
Siddharth Eswaran
2005CS10187

THE PROBLEM

Aim: To design and implement a single player interactive board game- "ROBOTS". The game shall be having a graphical interface with mouse support.

Problem Specification:

1. The game consists of a 31×31 board with 961 cells in 31 rows and 31 columns. Each cell is indexed by *(r, c)* [1≤ r, c ≤ 31].
2. Each cell is either empty or is occupied by the player or a robot or debris.
3. There are R robots initially. [1≤ R ≤ 50]
4. There are T potential teleports on the board. [1≤ T ≤ 20]
5. Rules for the movement of the player:
   (a) move to an adjacent cell (in any of the 8 compass directions),
   (b) teleport to any of the teleport locations provided the destination is empty,
   (c) remain stationary,
   (d) walk to an adjacent cell which contains debris and the debris can be pushed to an adjacent cell along the line of movement provided
      (i) this adjacent cell does not contain debris,
      (ii) If this cell contains a robot, the robot is destroyed.
   (e) The player can't leave the board or push the debris out of the board.
6. Rules for the movement of the robots: Robots can move into any of the adjacent cells (even if they are not empty) in a manner that they get closer to the player with each move. Robots get destroyed if
   (a) two or more robots walk to the same cell,
   (b) A robot walks to a cell containing debris.
7. Destroyed robots become debris.
8. The player wins if all robots are destroyed.
9. The objective is to remain in the game as long as possible (and potentially win).

The player shall make the first move.

THE SOLUTION

Data structures employed:

1. A 31×31 matrix of structures containing a character and a pointer to a node in which *(r-1, c-1)* corresponds to *(r, c)* and the characters are assigned the following symbolic values:
   (a) EMPTY if that cell is empty
   (b) ROBOT if that is occupied by a robot
   (c) DEBRIS if that is occupied by debris
   (d) PLAYER if that is occupied by the player.
   If a cell is occupied by a robot, the pointer holds the address of the node corresponding to that robot and the pointer is set to the null pointer otherwise.

2. A list of nodes of ordered pairs (x, y) containing the locations of the R robots. If a robot gets destroyed, its node in the list is deleted.

3. An array of twenty ordered pairs is maintained for the T teleport locations (if a teleport can be visited any number of times) OR a list of the teleport locations (if a teleport once visited becomes obsolete). [The problem specification is silent on this aspect.]

4. A global variable avaltp for maintaining the currently available teleport locations. (Updated with each move.)

Methods used

1. moverobots: performs the move action for all the robots.

2. destroy: takes an ordered pair (a location on the board) and destroys the robot at that location.

3. statistics: generates game statistics at the end of the game.

4. updatetp: updates the available number of teleports (avaltp).

The Algorithm

Player's move:
1. Suppose the position the player chooses to move is (r_p, c_p). The position (r_p, c_p) is checked if it is empty, has debris or a robot.

   Case 1: It is empty – the player is shifted.
   Case 2: It is debris – it is shifted along the line of movement (if possible) and the method 'destroy' is called if the debris is pushed into a cell containing a robot.
   Case 3: It is a robot – the player will be killed and the game is over.
   Case 4: It is out of bound – the player is not allowed to move and another chance is given to him for making the move.

2. Suppose the player chooses to teleport to any teleport location.
   (a) If 'avaltp' is zero, then the player can't teleport.
   (b) If a teleport can be visited any number of times, then a random position from the array is chosen and the player is sent there (if it is empty), the process is repeated, otherwise.
   (c) If a teleport once visited is to become obsolete, then the player sent to the location in the front of the list and that node is deleted.

3. Suppose the player chooses to teleport to a specific teleport location.
   (a) If that teleport is empty, then the player is teleported otherwise he is not.
   (b) That teleport is deleted if necessary.

Robots' move:
1. Before updating the positions of the robots in the list, all robot positions in the matrix will be made EMPTY. After updating the positions of the robots in the list, their positions are mapped to the matrix.

2. Now for updating the positions of the robots in the list, the list is traversed. Suppose for a particular robot, player is at (r_p, c_p) and robot is at (r_R, c_R).

I define

$A = r_p - r_R$; a = 1 if A>0, 0 if A=0, -1 if A<0

$B = r_p - r_R$; b = 1 if B>0, 0 if B=0, -1 if B<0

The new position of the robot will be $(r_R + a, c_R + b)$. This shall make the robot one step closer to the player.

3. Now this new position is checked in the matrix. If it is debris, then the robot is destroyed and its node be deleted from the list.

4. If the new position is a robot, it means the robot has entered a cell into which a robot had moved in just recently. These two will be destroyed and their nodes be deleted from the list.

5. If a robot hits the player, the player is killed and the game is over.

Updating the number of available teleports:
The data structure storing the teleport locations is traversed and the teleport locations are checked to be empty and the empty ones are counted.

With each iteration, the number of moves, time, and other statistical quantities are recorded.

Input and Output Schemes:

1. The input will be through a file. At the onset of the game, there will be an option 'Load Game…' which will ask the user for a file name and load the game from the file. The file must contain the following:
    (a) The first line contains the integers R and T separated by a space.
    (b) This is followed by R lines containing two integers which indicate the positions of the robots.
    (c) This is then followed by T lines having two integers which indicate the potential teleport locations.

2. The output will be in the form of a graphic window. There will be suitable iconic representations for 'Player', 'Robot' and 'Debris'.

3. There will be an 8-direction button system for navigation of the player.

4. Gtk graphics library will be used for the implementation.

Analysis of the Algorithm:

1. moverobots: O (R) time.
2. destroy: O (1) time.
3. updatetp: O (T) time.

I have designed the algorithm considering time efficiency. The implementation, however, can be made more space efficient for embedded systems (for mobile phones for example) by reducing the matrix to a list of occupied positions (because the matrix is sparse). But this may take a toll on the time complexity.

Testing Mechanisms: [These will be available only in the debug version.]

1. The contents of the lists will be printed simultaneously in the console.
2. The gaming area will be in the form of a grid which will enable me to see whether the shifting of objects takes place properly or not.
3. I will create [about 4] file containing the input. These can be loaded one by one and tested.