## Defining the BLOXORZ problem

### State Representation

The Bloxorz game is simulated in the application via the following components:

Bloxorz World Map

World map is defined by a static m \* n square matrix containing tiles (represented by ‘1’) and holes or no-tile spaces (represented by ‘0’). The holes or no-tile spaces are needed to represent the irregularly shaped world map into a square matrix.

The world map also contains a 1\*1 block represented by value ‘9’ called the ‘target’ or the ‘goal’ block.

The world map is static in the nature that the map itself does not change throughout the course of the game.

Brick

A brick is a 1\*1\*2 sized 3-d structure, that occupies either 1 or 2 blocks depending upon the orientation. The orientation can be standing (occupying 1 block on the 2-dimensional world map) or horizontally/vertically lying (occupying 2 blocks on the world map)

Brick component in the app holds a Position object, and offers methods for moving the brick or identifying the number and positions of the occupied blocks.

Position

A position object holds the brick’s x, y coordinates as well as its current orientation.

The position object offers low level methods for comparing current position with the target block’s position.

TreeNode

TreeNode is the unit used to represent a node in the graph. Each node contains a brick object (which composites a Position object) and some properties/attributes to link the nodes together in order to create a graph.

The nodes can be connected from parent to child node via 4 directions, namely:

left, right, up and down.

For easily navigating through the graph/tree, we also use additional properties / links like:

parent that connects a child node back to its parent, and

dir\_from\_parent to suggest what direction was picked from parent to reach a child node.

The TreeNode also contains properties specific to A\* algorithm. The properties are named:

cost - is the actual cost to reach up to the node from the head of the tree.

f\_score - is the sum of g-cost and h-cost, used for computing the estimated cost.

### States and Tree Representation

The Bloxorz game can be though of as a background layer composed of the world map, with a series of movements of Brick. Each of the nodes in the graph thus only needs to represent the brick position with respect to the map. The directional movements between the nodes thus represent the edge between the nodes.

The application, thus uses a head TreeNode, consisting of a Brick object initialised at position 2,2 (or 1,1 in a 0-based world).

Each of feasible movements form a new node connected to the head or it’s child nodes, thereby making a graph ( or a tree since most of the search algorithms maintain a visited node list to restrict movements terminating on an already visited node, thereby preventing any loops.)

The world map combined with the brick position and orientation can be though of as one state, that can transition to another state via brick movement in one of the four possible directions.

## 2. BFS and DFS implementation

BFS Search

The BFS being an uninformed search traverses through each and every state in a tree order traversal. For this, the algorithm needs to maintain a queue (nodes\_queue), using which it can expand nodes one level at a time.

BFS algorithm also maintains a visited nodes/positions list to prevent the brick from moving back to its earlier position.

|  |
| --- |
|  |

Fig. 1 - Initial moves of the BFS search tree. (order of directions - LRUD)

DFS Search

Similar to BFS, The DFS search tree is also implemented in the application. The DFS algorithm maintains a stack (internally managed via recursive function calls). It also maintains a list of visited nodes to prevent loops.

Unless BFS and A\*, DFS is highly sensitive to the order of search directions.

A detailed analysis of the same is presented later in Section 6.

|  |
| --- |
| Right  Right  Down  Left  Left  Left  Left  Left  Right  invalid  visited  visited  invalid |

Fig. 2 - Initial moves of DFS Search tree (order of directions - LRUD)