1. Mini-SHRDLU

SHRDLU is a name of software written in late 1960s by Terry Winograd at MIT Artificial Intelligence (AI) Laboratory. The program was used to demonstrate how AI technologies could be used in natural language processing[1]. Inspired by, but different from, SHRDLU, **Mini-SHRDLU** is a computer game designed by Demis Hassabis, the CEO of Google DeepMind, and his research team for testing their AI algorithms[2].

A Mini-SHRDLU game contains k numbered blocks (from 1 to k respectively) on an $n*n$ vertically suspended board ($k \le n^2$-n). All blocks must be placed in the board, distributed in the columns, next each other down to the bottom in each column. A block on the top of a column can be moved from one column and deposited on the top of another column if space is available. At the beginning of each game, the board is randomly configured, called the *initial state.* A player of the game is given a *goal* or a set of *goals* to achieve by moving the blocks in sequence from the initial state. A goal can be any properties of the positions of blocks. For instance, we can require "*Block 1 to be at the bottom of the first column*" or "*Block 3 on the left of Block 2*". Any board configuration that satisfies the required goals is called a *goal state*. The following two figures show an example of initial state and goal state of a Mini-SHRDLU game with k=*6* and n=*3*.

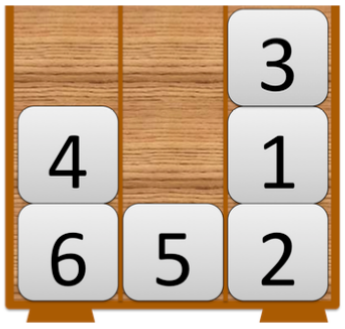

Figure 1: *An example of initial state*

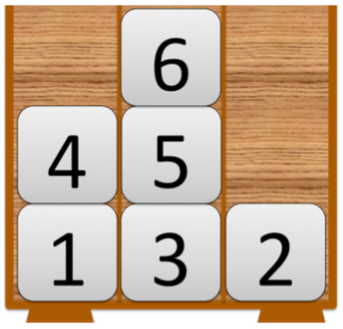

Figure 2: *An example of goal state*

This assignment is to develop a computer program that can host and automatically generate a solution to the Mini-SHRDLU game with any board size, $n,$ and number of blocks, $k,$ as long as n>*2* and $n \le k \le n^2 - n$. However, for the simplicity of description, from now on we assume k=*6* and n=*3*. The general case is quite similar.

[1] https://en.wikipedia.org/wiki/SHRDLU
[2] Alex Graves *et al*. Hybrid computing using a neural network with dynamic external memory, *Nature*, 471-476, 27 October 2016.

2. Game description

For a better understanding, we use the standard AI terminology to explain the game.

2.1 States

Any block configuration of a board is called a *state*. **Figures** 1&2 in the previous page are examples of states. A simple (but not necessarily the best) representation of a state is a two-dimensional array of integers in which zero stands for blank and the numbers 1 to 6 represent the blocks. For instance, the following matrix represents the initial state shown in Figure 1.

$$\begin{array}{c|c|c|c|}
\cline{2-4}
2 & 0 & 0 & 3 \\
\cline{2-4}
1 & 4 & 0 & 1 \\
\cline{2-4}
0 & 6 & 5 & 2 \\
\cline{2-4}
\multicolumn{1}{c}{} & \multicolumn{1}{c}{0} & \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2}
\end{array}$$

Figure 3: *The game state represents the game board of Figure 1. The blue figures are indices, not part of the state.*

Note that we use 0 to index the bottom row and the first column just because it is easier for implementation in an array or vectors.

2.2 Actions

An *action* is a move of the top block on one column to the top of another column. We use a pair of integers, (*source*, *destination*), to represent an action. For instance, (*2, 0*), means to move the top block in Column *2* to the top of Column *0*. An action is *legal* in a state if it is doable in that state. Given the state showing in **Figure** 3, the action (*2,0*) is legal, which moves Block 3 from column 2 to column 0 while (*0,2*) is illegal because column 2 is full. Neither is (1,2). In fact, only four actions are legal in that state: (*0,1*), (*1, 0*), (*2, 0*) and (*2,1*). Once a game player applies a legal action to a state, the state will change to its *next state*. For instance, if we apply the action (*2,0*) to the state shown in **Figure** 3, its next state will be:

$$\begin{array}{c|c|c|c|}
\cline{2-4}
2 & 3 & 0 & 0 \\
\cline{2-4}
1 & 4 & 0 & 1 \\
\cline{2-4}
0 & 6 & 5 & 2 \\
\cline{2-4}
\multicolumn{1}{c}{} & \multicolumn{1}{c}{0} & \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2}
\end{array}$$

Figure 4: *The game state after applying action (2,0) to the state showing in **Figure** 3. Note that actions (1,0) and (2,0) are then not legal in this state.*

2.3 Goals

A *goal* can be any property of block configuration you want a game terminates with. For instance, you can require Block 1 to be located at (*0,0*). In this case, the goal can be represented as (*1, 0, 0*), meaning *Block 1 to be located at coordinate (0,0)*. A *goal state* is any state that satisfies the goal. Normally one goal can have several goal states.

A goal in the form (*block, row, column*) is called an *atom* goal, which means that the *block* must be located at location (*row, column*). Atom goals can be further combined in two different ways: disjunctive and conjunctive. If a set of atom goals {*g1, g2, ..., gm*} are *disjunctive*, any state that satisfies any of the atom goals is a goal state. For instance, the disjunctive goals {(1, 0, 0), (1, 1, 0), (1, 2, 0)} means Block 1 must be in Column 0 no matter which row it is. If a set of atom goals {*g1, g2, ..., gm*} are *conjunctive*, a goal state must satisfy all the atom goals in the set. For instance, the conjunctive goals {(1, 0, 0), (2, 0, 1), (3, 0, 2)} means Blocks 1,2&3 must be all in the bottom row in the goal state.

There are also other ways to represent goals. Let *'b', 'a', 'l'* and *'r'* denote '*below*', '*above*', '*left of*' and '*right of*' respectively. Then '*6b2*' means *Block 6 must be below Block 2*. '*4l1*' means *Block 4 must be at the left of Block 1*. These goals are named *neighbourhood goals.* Obviously neighbourhood goals can be transferred into combinations of disjunctive and conjunctive goals.

2.4 Plans

A *plan* is a sequence of legal actions that can bring a game from the initial state to a goal state. The task of this assignment is to design an algorithm that can automatically generate a plan to execute from any given initial state to a goal state if the goals are achievable. A typical way to find a plan is **game tree search** in term of Artificial Intelligence as shown in **Figure** 5.

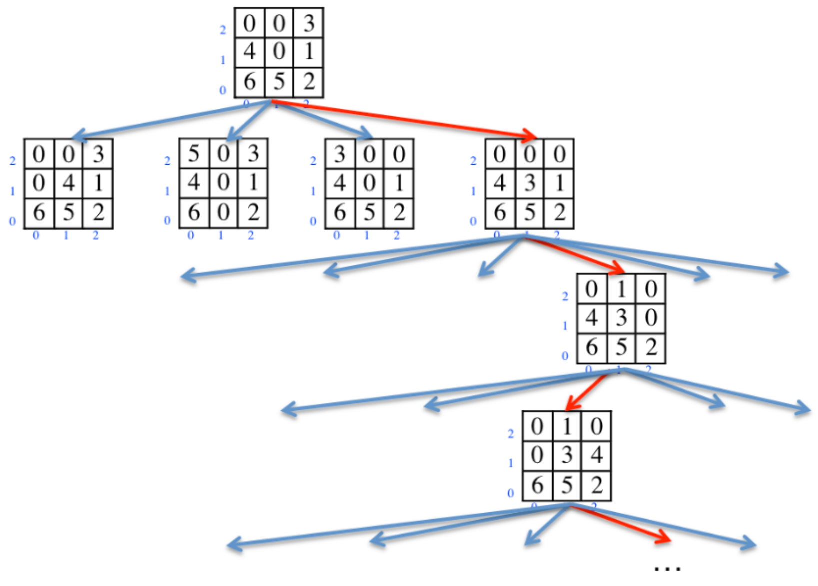

Figure 5: *An example of game tree search (only part of the search tree is displayed)*