OPTUM TECH UNIVERSITY

Capstone Project

To complete the course, we will introduce the final project: 2048. Before the dive into the technical details, we'll provide an overview of the game:

- The gameboard consists of a 4 X 4 grid. Each cell in the grid can hold a number in the order of 2^x.
- The gameboard starts by adding two '2' values randomly to the board.
- The you have the option of moving in four directions: Up, Down, Left, Right. When you move a given direction, blocks of like value combine (and double). Two '2' values combine into '4'. All whitespace gets shifted so that the blocks are as far towards the direction as possible. During every move, a '2' is placed in a blank space randomly on the board.
- You win the game by obtaining 2048, or lose it by filling up the gameboard to the point where no blocks can be combined

Give the **game** a shot if you have never played it.

Start off by downloading the most recent version of the repo. Navigate to /SOAR/2048 within terminal. You can run it by typing 'make run'. Your rules should be in a file called 2048.Soar in the same folder. Update the makefile to point to your Soar directory.

Input

The gameboard is inputted into Soar automatically at the beginning of each decision cycle. It utilizes the following format:

At io.input-link there are 16 variables representing each on the blocks on the board.

<u>Each block has a name and a value:</u>

io.input-link.<blockX>.name <val> (*example*: io.input-link.<blockX>.name block1-1)

io.input-link.<blockX>.value <val> (*example*: io.input-link.<blockX>.value 0)

<u>Each block has values for up/down/left/right of the block:</u>

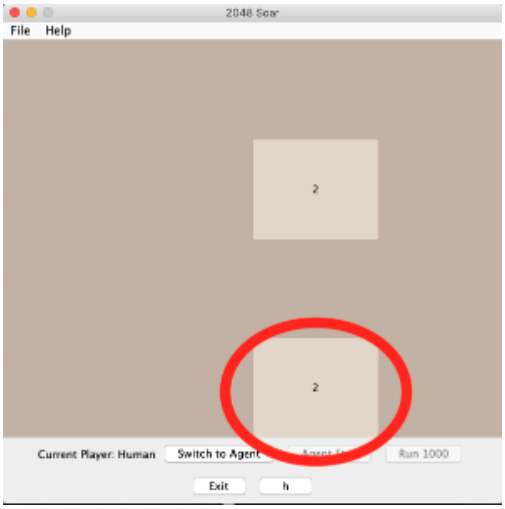
io.input-link.<blockX>.u <val> (*example*: io.input-link.<blockX>.u 0)

io.input-link.<blockX>.d <val> (*example*: io.input-link.<blockX>.d 4)

io.input-link.<blockX>.l <val> (*example*: io.input-link.<blockX>.l 16)

io.input-link.<blockX>.r <val> (*example*: io.input-link.<blockX>.r 0)

Important note: The block values aren't necessarily the adjacent values. This value will represent the closest value that isn't 0, or 0 if none exists. As an example, the u value for the red block would be '2', since it is the closest none-zero value:



Each block also has the names of the adjacent blocks. These will only exist if the block exists, and isn't a wall:

io.input-link.<blockX>.uBlock <val> (*example*: io.input-link.<blockX>.uBlock block0-2)

io.input-link.<blockX>.dBlock <val> (*example*: io.input-link.<blockX>.dBlock block2-2)

io.input-link.<blockX>.lBlock <val> (*example*: io.input-link.<blockX>.lBlock block1-1)

io.input-link.<blockX>.rBlock <val> (*example*: io.input-link.<blockX>.rBlock block1-3)

If a given direction has a wall, it will be represented as:

io.input-link.<blockX>.wall <dir> (example: io.input-link.<blockX>.wall u)

If the board has not changed, a variable will be added with the number of times it has remained unchanged:

io.input-link.<blockX>.change.count <val>

Output

The only thing required for the output is the direction that the agent has chosen:

io.output-link.otherlink.direction u/d/l/r

The output link must be cleansed at every cycle, or the old moves will remain.

```
#CLEAN Output Link
sp {top*apply*cleanupOutput
   (state <s> ^operator <op>
              ^superstate nil
              ^io.output-link <out>)
   (<out> ^<cmd> <id>)
   (<id> ^status)
-->
   (<out> ^<cmd> <id> -)
}
```

The goal of this project is not for you to obtain a perfect score. Although this game can come off as simple, it is quite complex. The main goal is to obtain a result that is based off of some reasoning that you determine. One recommended path is to simply propose an operator for each adjacent pair of blocks and select the highest.