Inputs:

While the pythonshell program does not take any inputs, the iterative.lp clingo program does. It accepts two file: one for agents, one for the map the agents will be searching.

Agent files contain the agents in a server, as well as the tasks that these agents need to complete.
Agent files contain the following atoms:

agent(AID, SERVER, X, TYPE)
- -AID is the agent ID (needs to be unique across all servers)
- -SERVER is the server number of the agents origin
- -X is the agents starting location in a given server
- -TYPE is for future implementation of depots

task(TID, SERVER, GROUP, GOAL, TYPE)
- -TID is the task ID (does not need to be unique across all servers)
- -SERVER is the task's server (tasks cannot change servers)
- -GROUP is the task's group
- -GOAL is the location needed to be reached to complete the task
- -TYPE is for future implementation of depots

Map files contain the vertices and edges that make up the search space of a server.
Map files contain the following atoms:

vertice(X, Y)
- -X is the server
- -Y is the vertice's ID (must be unique across all servers)
- NOTE: map filed can be refactored to no longer need X,
  Instead adding a single “server” atom in the file

edge(V, W)
- -V is a vertice
- -W is a vertice

Outputs:

In a single run of our clingo solver, we read in an agents and a map file for each server, and output the agents, the moves and any swaps that have been created.

Our output shows us the following atoms:

agent(AID, SERVER, X, TYPE)

-exactly the same layout as seen in the agents file

move(AID, X, Y, T)

-AID is the agent ID performing the move
-X is the starting vertice
-Y s the ending vertice
-T is the timestep the move is being performed on

swap(TID, SERVER, AID, GOAL, GROUP, Y, T)

-TID is the task ID
-SERVER is the original server of the swap
-AID is the agent performing the task
-GOAL is the location needed to be reached to complete the task
-GROUP is the group of the agent performing the task
-Y is the location of the agent performing the task
-T is the timestep the swap was created (not used, needed by clingo to create the swap)

Communication:

The communication between the two processes is fairly simple. We take the output of the processes and add the following to a list (one for each server):

-Take the last move of each agent, and update each agent so it's starting position Is now its final position in the last run.

-take each swap and split it into an agent and a goal, using the information Packaged in each swap

-remove swapped agents from their previous servers, add them to their new Servers, as well as adding their respective goals

The general flow of the program runs like this:

-Make two child processes, one for each server, and solve them

-Process the output so it can be used as input for the next round of clingo runs

-run two child processes again, one for each server, in order to solve the swapped goals

TODOS:

Use the clingo python API to pause the processes mid run and swap the information,
Rather than waiting until each run is complete.

Cleanup I/O

Implement depots?

Support for more than 2 servers?