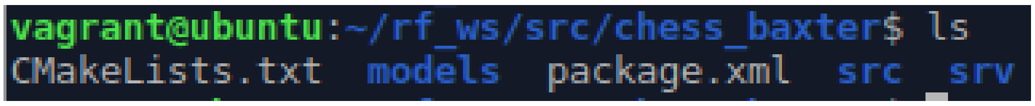
GUID: 2519359j
Name: Omar Jabri

Reflective Analysis and Design Rationale

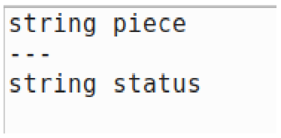
1- Structure Set Up:

First and most important, I started by setting up a working directory.
I started by creating the different folders needed for a good ROS package.
Then, as so to not reinvent the wheel, I have gathered all valuable resources from past labs in order to aid me into developing my own unique solution.



```
vagrant@ubuntu:~/rf_ws/src/chess_baxter$ ls
CMakeLists.txt  models  package.xml  src  srv
```

Service:



```
string piece
---
string status
```

2- ROS Nodes and Roles:

In my solution can be found five ROS nodes. Two of them representing what was already given [spawn_chessboard.py, delete_chessboard.py]. I then developed a service working as a server and middle ware between all other ROS nodes named “spawn_pieces.py”. The latter works on transferring important information between nodes such as the piece to be spawned next and its name following the format [letter, number]. Most importantly, this node spawns each piece in the order of the board setup one at a time, helping Baxter by simplifying the environment and not cause any obstacles. Alongside this server node, we have two nodes gazebo2tfframe.py and pick_and_place.py. The first node is the most important one since it translates the different frames located in the /world environment into transforms that pick_and_place.py node can use in order to confirm which piece at which location to pick next. Gazebo2tfframe.py node gets each piece spawned by spawn_pieces.py and sends the transforms of the spawned piece into a frame given its name. pick_and_place.py will look up this transform and perform operations based on the given parameters. Moreover, this node acts like a client for the service spawn_pieces.py in order to extract the service parameters.

3- Approach and Improvements:

My first approach was simple, using one node that combines all. I followed that as a first step to test if the solution is accessible or not. After succeeding, I decided on following a more structured approach by using the nodes defined above.
I started first by using the positions given by the spawn_chessboard.py, however, this information is not always available for an agent in an environment. Hence, I went on using tf translations which is a much more flexible approach. Finally, I

created a main board setup to explain where the pieces should be and extract a goal position for each piece spawned dynamically.

Improvements can surely be inserted in this solution, by either using a vision-based approach since most of the objects are colour coded, or by improving the planning and execution technique. The latter is crucial since Baxter tends to follow the RRT algorithm, which is in nature random, adding a more structured planning algorithm can help Baxter achieve a smooth and calculated planning.