Initial Submission Due April 13, 2016 at 10AM

This Initial Submission project description is an excerpt from a larger specification being developed for Project 2.

Model-View-Controller

The project is organized around the Model-View-Controller design pattern. There is a distinct division of labor between the Model, which keeps track of everything in the simulated world; the View, which shows the contents of the world; and the Controller, which is how the user interacts with the Model. For the initial submission, we just implement one part of the Controller—the command line interpreter.

Command Line Interpreter

Present the user with a command prompt “Enter command: ” after which the user will type a series of whitespace-delimited strings or numbers.

Each command line is processed as follows:

1. Read the first word.
2. The first word should be either the name of a human, the name of a robot, or a command word.
   - If it is the name of a human, then the next word should be a human command word.
   - If it is the name of a robot, then the next word should be a robot command word.
   - It should otherwise be a specifically-recognized word command.
3. If the first word is not the name of a human or robot, and it is not a valid command, then an "Unrecognized command: <line>" error message is returned, in which <line> is the line that was entered.

The various commands that the user can type in will be as follows:

**Model commands**

Each command is shown below first in its general form, and then in example usage. All locations specified in command strings must be integer pairs and must exist in the world; otherwise, output “Error: Invalid location.” All humans and objects must be placed on routes; otherwise, output “Error: Must be placed on a route.” No check is made of whether the name of a new human or robot is the same name as one of the commands; the result is undefined.

```
create world <n>
create world 10
```

Creates a world of size n by n. The world is a grid that ranges horizontally from 0 to n, and vertically from 0 to n. n must each an integer >= 10 and <= 30. Only one world can be created for each run of the simulation. Until it is created, every other command (other than quit) responds with “Error: You need to create a world first.” Once the world is created, attempting to create it again results in the message “Error: A world already exists.”

```
create waypoint a <x> <y>
create waypoint A 0 0
```

Creates a waypoint named “A” at location (<x>, <y>). A waypoint is a point on the grid that is given a name in the form of a single uppercase letter (other than 'H', 'R', or 'F'). All waypoint letter-names typed in as lowercase are converted to uppercase.

```
create route <w1> <w2> ... <wn>
create route A B C
```

Creates a route from waypoint A to B to C. A is the start of the route and C is the end of the route.

```
create human <name> <x> <y>
create human Jerry 10 10
```

Creates a human at location (10, 10) on the map. <name> must be alphanumeric and there must be no other humans in the model with a name that starts with the same letter or numerical digit.

```
create robot <name> <x> <y>
```

Same functionality as the “create human” command but it creates a robot.
status
All of the humans, robots, and routes describe themselves.

quit
Quits the simulation after prompting the user with “Are you sure you want to quit? (Y/N)” and quitting only if ‘Y’ or ‘y’ is entered.

View Commands

show <optional parameter>
show waypoints
Shows the map. The optional parameter can be “humans”, “robots”, “fires”, or “waypoints”. If a parameter is provided, then only that particular set of objects, plus the waypoints, are shown. The drawing of the map is detailed below. If no parameter is provided, everything is shown.

Human and Robot Commands

Project 2 will give the humans and robots just a single command, as follows:

:name goto <destination>
:name goto <x> <y>
Jerry goto A
Jerry goto 10 20
This will move the human or the robot named <name> from its current location to <destination> in which <destination> is either a single letter describing a waypoint or a point that the human can reach from its current location by following routes on the map. Given the restrictions on creating routes, there should be exactly 0 or 1 ways for the human or robot to get to the destination.

Initial Submission

For the initial submission, create a simple command interpreter in a function called run( ) in a file called P1_Initial.py. Only implement the recognition of the following commands, with just the arguments that are shown, and provide the output that is shown.

create human <name> Correct output should be: Human <name> is created.
create robot <name> Correct output should be: Human <name> is created.
show humans Correct output should be: Humans: <name1>, <name2>, ...
show robots Correct output should be: Robots: <name1>, <name2>, ...
:name goto Correct output should be: Human <name> will goto
or: Robot <name> will goto

Implement three possible error messages:
(1) If the user attempts to create a robot or human, but a robot or human with that name already exists, output the following:
   Error: <name> already exists in the world.
(2) If the user enters a goto command for a human or robot that does not exist:
   Error: <name> is not in the world.
(3) If the user enters additional arguments beyond those permitted above, or extra words after those above:
   Unrecognized command: <line>
in which <line> is the line that was entered.

Additional Specifications
• You may create additional “helper” functions in addition to run( ) which will should contain your main loop.
• You may submit your project via Canvas as many times as you like. We will grade your final submission.
• Do not add any import statements to the program.
• Do not the “try” and “except” statements.
• Comment your code. Your solution must provide an equal number of lines of useful, meaningful comments as it includes lines of Python code.

Grading Criteria
• Are project files submitted on time and with the specified filenames and functions?
• Is the submission complete?
• Does the project correctly implement the specification.
• Is the code well-commented, with a 1:1 ratio between lines of code and lines of useful comments?

Attribution
This project is derived from Kieras’ EECS 381 Project 4 at http://umich.edu/~eecs381/