UM EECS 281 Fall 2004

Programming Assignment 1
Mission to Mars

The Game

Your job is to program a new Mars rover named the **Vanguard**. The Vanguard is being deployed on a remote location on the planet Mars. The purpose of the rescue mission is to find the lost British rover **Beagle 2** and help Beagle 2 transmit its data back to Earth. Shortly after landing, unintelligible data was received from Beagle 2 and suddenly all contact was lost. It is believed that Beagle 2's video recorders are still active and could provide key information regarding what went wrong. Satellite reconnaissance has located Beagle 2, and revealed that several key components of Beagle 2 are scattered about the surface of the Red Planet. Among these parts are the Apex Transmogrifier Unit (**ATU**) and the Wavelet Transponder Dish (**WTD**). Vanguard must gather these pieces and repair Beagle 2, so that Beagle 2 may beam its information back to Earth.

Vanguard must also discover a way to provide enough power for Beagle 2 to successfully send its data back to Earth. It is believed that a rare mineral, Praesodymium (aka **Crystal**) is located on Mars in crystal form. This crystal may be processed by the ATU and provide enough power for Beagle 2 to operate and send home its collected data. Therefore, part of the mission is to assist Vanguard in finding crystal to help power Beagle 2. Is Summary, your mission is to program the Vanguard to find the ATU, WTD, and Crystal, and proceed to Beagle 2 for repairs after gathering these objects.

A Harsh and Barren World

The game world is divided into multiple *maps* where pieces of Beagle 2 might be located. A map may be thought about as a level in the game world. The number of maps in any particular game world may vary from 1 to 10.

The Vanguard can travel between these maps by use of new NASA technology, called *teleporters*, that instantly transport the rover to the specific, associated map. Teleporters, signified by grey tiles in a map, transport the Vanguard from the current location to the associated tile on another map specified by the grey teleporter tile. Teleporters are bidirectional; that is, they can transport from map X to map Y and from map Y to map X.

All maps are accessible by at least one teleporter, however, the {ATU, WTD, Crystal, and Beagle 2} may be distributed between multiple maps. **Figure 1** shows an example game world with three maps. In the first map there are two teleporters, one teleports between map one and map two, and the other teleports between map one and map three.

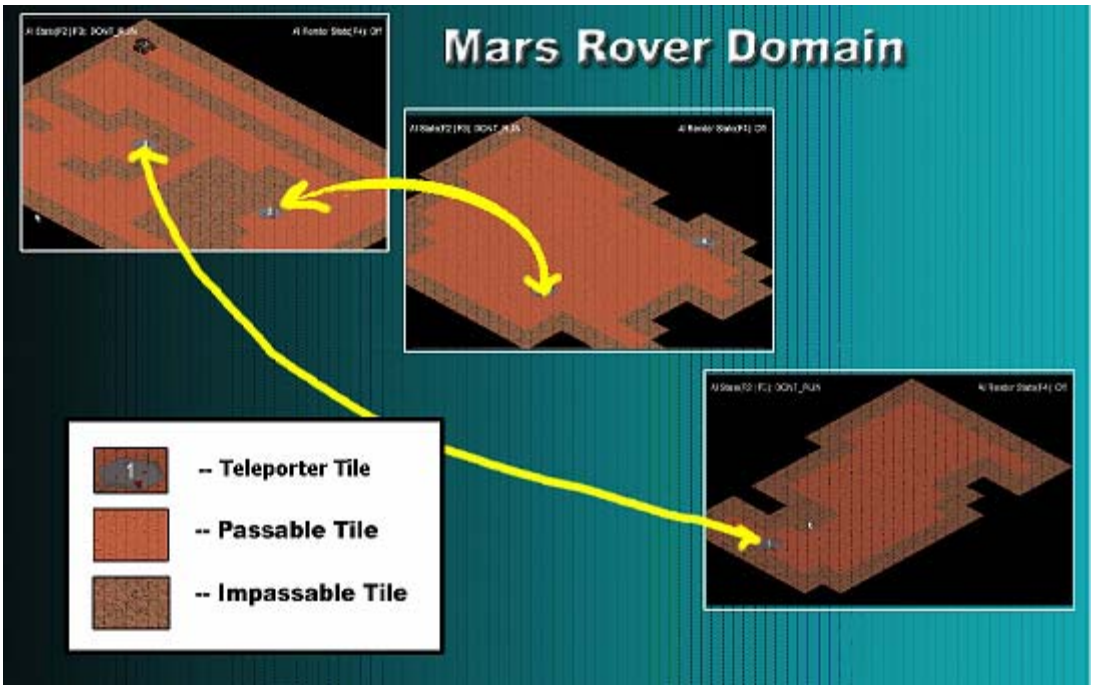

Figure 1. The world of the Mars Rovers.

Your Tasks

You are to develop two path finding algorithms:

- A stack-based path finding algorithm;
- A queue-based path finding algorithm.

Your stack must use a *linked list* as its underlying data structure. Your queue must use an *array* as its underlying data structure.

You MUST develop your own stack and queue data structures for this programming assignment. That is, aside from the string class, you may not use the STL for any portion of this programming assignment. You can use the codes provided in the textbook, but you MUST include the textbook in your references if you use.

Grade Composition

Form

5%: Properly developed and commented code. For details, see Coding Style section near the end of this document.

Documentation

7.5%: Timing Analysis

7.5%: Test Report

Performance

40%: Working Stack algorithm

40%: Working Queue algorithm

What to Turn in

For the project, each student is required to turn in documentation in hardcopy, and source code to the autograder. Details about both types of submission follow.

Documentation

You must turn in a hard-copy of your *Timing Analysis* and *Test Report* results to the eecs281 drop box in 2420 EECS by the due date/time. Make sure your uniqname is on the hard-copy.

Timing Analysis

For timing analysis, we require that you run both algorithms on the given example test cases. Results will be turned in the form of an Excel graph. Collect the timing information for each of the given example test cases, for each of your algorithms. For each algorithm sort the timing numbers by the testcase map size. Then plot both sets of numbers on a single Excel graph. (If your algorithm detects some fatal errors on a given test case and exits with proper EXIT_CODE explained in File Output section, you don't need to collect timing information for the test case.)

The timing information is available via the console command **time**. To display the current timing information, press tab to open up the console, then type in **time** and press enter. This will display the time that Vanguard spends executing the path-finding algorithm, including returning the solution tiles, but not including movement time.

To determine the time efficiency of your code, we will compare your run time with an average of the TA's and instructor's runtimes. Performance points will be assigned based on this comparison, using a sliding scale. If your runtime is not within X times the base runtime, you will lose points. The value of X will be announced soon.

Test Report

We will be grading for correctness primarily by running your program on a number of test cases. If you have a small, silly bug that causes most of the test cases to fail, then you may get a very low score on that part of the programming assignment *even though you completed 95% of the work.* As mentioned above, most of your grade will come from correctness testing. Therefore, it is imperative that you test your code thoroughly. To encourage some degree of correctness testing, we require that you hand in your test cases in the form of a 1-2 page Test Report. Your Test Report should follow the template that can be found on http://www.eecs.umich.edu/courses/ eecs281/f04/projects.html

Source code

You need to submit source code to the autograder. The files you must submit are: ai.cpp, ai.h, and any other files you use.

The Autograder

You can submit your code to the autograder a total of 10 times per day. For the first 3 submissions per day you will receive correctness feedback by email. After your first 3

submissions, you will simply receive a response indicating that your submission has been received. **You should NOT use the autograder as a debugger!** You will only get 10 total submissions per day, so make sure you have tested your code thoroughly before even contemplating submission to the autograder. The autograder has only enough functionality to ensure that you are getting the basics right (compiles, runs, solves some trivial cases, etc.) **but you still need to test your code carefully and thoroughly on your own.**

If any code takes too long to run, then the autograder will simply terminate it. 'Too long' will be determined based upon the average run time of the best performing projects, including that of the staff. (Hint: your program should not take more than 10 seconds in path-finding for any of the given example test cases.) Every time you submit your code to the autograder, it will log your run time. We will periodically post the top 10 performance numbers on the course web site or forum.

Despite its name, the autograder does not assign grades, it merely tests for correctness against previously created test cases. Your grade will be assigned after all final submissions are received and graded.

Every time code is submitted to the autograder, the old copy is **overwritten**. We *do not* keep backup copy of anyone's code. It is the student's responsibility to back up code. If you have working code, make sure you make a backup copy before making any more "improvements". The adventurous among you may try to learn how to use version control software for this purpose. Your last submission to the autograder before the deadline is considered your final submission and is the one used to assign a grade.

Submission to the autograde

To submit your code to the autograder, use the submit281 script. An instruction explaining how to use the submit281 script can be found on http://www.eecs.umich.edu/courses/eecs281/f04/projects.html