**CSCE 155N Matlab Programming Project 2 – Summer 2016**

**Assigned: Wednesday 6/22/2015**

**DUE DATE:**

**Friday 7/1/2016 at 11:59 PM**

**(hardcopy in class Monday)**

**Battle of Brains, Blood, and Brawn**

**Problem Statement:**

The trainload of werewolves, vampires, and zombies crashed, releasing the creatures into the night. They made their way to a couple castles that were ruled by dueling wizards who were delighted at the prospects of forming them into armies. A mix of 24 of the creatures were thus inducted at each of the castles, 48 in all. Their wills were bent to fighting the army of the opposing wizard, rather than simply biting each other.

As a reminder, when a werewolf fights with a vampire, the werewolf will invariably bite the vampire; when a vampire fights with a zombie, the vampire will invariably bite the zombie; when a zombie fights with a werewolf, the zombie invariably bites the werewolf. As before, the bitten creature recovers, transformed into its biter’s form, but now will also switch allegiance to the biter’s army. Not previously mentioned because it was not relevant, a creature that attacks another creature of its own kind will utterly destroy its victim, never to return.

The battlefield consists of an 8 x 8 grid, like a chessboard. The wizard easily gets the mix of creatures he wants (If the truck wreck survivors are not of the favored mix, he has them bite and transform each other until the mix is right!) The creatures are lined up on the last 3 rows of each side, arranged however the wizard wishes, but shadows are cast so the opposing wizard cannot observe the constituency and of the other’s army. The type of each creature will not become clear to the opposition until there is an attack upon or by that creature.

Play is similar to that of queens in chess. In alternating fashion the wizards move one of their troops any number of steps forward/backward, sidewise, or diagonal. An attack occurs by moving a troop to an opposing troop’s position. The casualty of the attack is taken to any open position in the winner’s 3 rows where it can transform and recover by the time of the next play. It is possible to attack ones own creature, in which case the transformation takes place and the creature is similarly moved back to anywhere in the first 3 rows to recover. This could be a valid strategic move if one needed more of a certain form of creature, or wanted to relocate it under different constraints.

Play continues until the opposition is wiped out (or the sun rises).

**Problem input:**

You need to get the arrangement of creatures on the board for each wizard, with the constraint that each wizard must be able to do so without the other observing his placements. How you do this is up to you, but you need to justify your design in the report that accompanies your program. You may choose inputting from the keyboard, prompted for each datum, or prompted for an entire matrix. You may choose to input from a data file. Perhaps you can think of other ways to get the data into the program. If you upgrade this project to become project 4, you might use a GUI approach to data entry. For the actual playing of the game, each wizard selects the piece to move and where it is to move. This could be like declaring moves in chess, but does not have to be. Again, you decide and justify.

**Problem output:**

Similarly, you may design how the results are reported. We should see the final pattern and numbers. We may appreciate seeing the progress each night brings. Again, discuss your choices in the report.

**Problem options:**

You are welcome to propose variations, such as having random mutations, etc. Yes, discuss your choices in the report.

**Collaboration:**

Work together as a team on any or all aspects of the research and design. Ideally take advantage of the talents of each member of the team. Use your teams to finalize each of the multiple design options. It is essential to keep track of who did what and where any useful information was found. You need to keep track when you help someone and when you receive help from someone. This includes students from other teams and those outside the class, TAs and me (the instructor). Keeping a log is highly recommended.

**What and How to Submit:**

Read and have your program conform to the “Program Documentation Guidelines” which were provided previously.

As you make progress with the project, periodically handin (electronically) updates numbered as mine are on my site. How many updates really depends on what seems natural. It might be 5 or it might be 15. Anyway, we should see a progression of “working” programs handed in as ***bbbb1.m***, ***bbbb2.m***, etc.

By the deadline hand in electronically the two files, bbbb.m (the Matlab function file for the game), and bbbb.doc (which contains summaries, documentation, and sample runs). In the next class period, hand in hardcopy versions, stapled together with the cover page in front.

Each team member should submit his/her own analysis of the relative contributions of all the members toward the project. This should be submitted electronically using each member’s handin account. This is in addition to the acknowledgement section of the main report. Assuming allocation is fairly even, all will receive the same grade.

The Word document should contain the following, all carefully labeled:

* Cover page with name(s) and the account under which it is submitted, title, date submitted, etc.
* A discussion of the features you implemented in the project. Describe how they work and what Matlab options were used to program them. This should be at a fairly high level, not a line-by-line analysis of the code.
* An “instruction manual” that a non-programmer can use to set up and run the game.
* An annotated cut and paste sample dialog sampling of the running of the simulation. (Hint: Use the ‘diary’ command or cut & paste as appropriate.)
* A discussion of the testing that was performed. This should include testing of each component as it was being built, and testing of the final program ensuring that it works properly under a comprehensive range of conditions.
* An annotated cut and paste of a sample dialog, demonstrating how your program responds to extreme and faulty input. (This could be combined with the previous section.)
* Acknowledge all collaborations (both internal to the team and external), detailing what each person contributed individually, and what was done jointly. Indicate approximate percentages of the work contributed by each person in design, coding, testing, documentation, and report preparation.

**Grading Criteria:**

* Properly running features – 30%
* Program logic is well designed – 20%
* Progress versions – 20%
* Documentation guidelines are followed – 10%
* Handin Documents formatted and arranged as specified – 10%
* Testing is comprehensive – 10%