CISC-121 Winter 2013

Week 3 Lab Assignment: Hammurabi

This is the assignment for the labs in Week 3 (January 21 – 25). This assignment will be graded, and is worth 5% of the course.

Deadline:

Please submit your solution by 8:00 AM Monday January 28. I will give you a 24-hour grace period that ends at 8:00 AM Tuesday January 29. Assignments submitted during the grace period will be penalized 10% for lateness. Assignments will not be accepted after the end of the grace period.

Submitting Your Solution:

You are permitted (but not required) to work in pairs on this assignment. Your submission should be in the form of a single file containing your Python program for this lab. At the top of the submission you **MUST** include the following information:

- **full name** and **student number** of the person(s) who worked on this solution.
- the date on which you are submitting.

If you work with another person, **only one of you** should submit the solution. It does not matter which of you submits.

Submit your solution using the CISC-121 Moodle page. This assignment is listed as an activity for the third week of the course.

Introduction: In class we looked at the BASIC version of the Hammurabi game, and a fairly literal translation into Python. Your job is to improve the coding style of the Python version, and to add some features to the game.

Details:

1. I have uploaded two Python versions of the game, one using simple “input” and “print”, and the other using EasyGUI. You are free to use whichever you like for this assignment.

2. Download whichever Python version of the game you choose, and make sure it runs.

3. Replace all the meaningless variable names with meaningful ones. Python allows

long variable names – one popular convention for choosing good variable names is to connect the words of a descriptive phrase with "_" (underscore) characters. For example, a variable that holds the price of land might be named price_per_acre . Be careful – as we saw in class, the original version of this program sometimes uses the same name for different purposes. If you are using SciTE, you can save yourself some tedious typing with the "Complete Word" function (Ctrl-Enter).

4. The main body of the program consists of a **while** loop, with a **break** part way through. This is clumsy and a bit confusing. Replace this with a better structure that does not contain a **break.** This may require some thought.

5. Modify the program so that after the game ends, the user is asked if she wants to play again (if you are using the EasyGUI version, explore the E**asyGUI** tutorial at http://easygui.sourceforge.net/doc/tutorial/index.html to learn about yesno boxes). The user should be able to play as many times as she wants.

6. Modify the program so that before the game commences, the user is asked if she wants to play at the "Easy", "Standard", or "Challenge" level:
   Easy: People eat 15 bushels of grain each, the yield is between 2 and 8 bushels per acre, and there are no rats or plagues.
   Standard: As is.
   Challenge: People eat 25 bushels of grain each, the yield is between 1 and 4 bushels per acre, rats attack with probability 2/3, and if your population drops below 40 you immediately lose the game.

7. Modify the program as follows:
   1. In the year immediately after a plague, the people can only plant 7 acres each rather than the normal 10.
   2. If your ratio of people to acres ever drops below ½, you are immediately conquered by a neighbouring city and you lose the game.
   3. The value of land is currently the same for buying and selling. Modify the program so that the price to buy land is always 1 or 2 greater than the profit from selling land.

8. Add a feature of your own choosing to the program.

Dealing with Errors: In the real world, the game would need to be robust enough to not crash if the user doesn't enter a value when required. For this assignment you can assume that the user will not do this.

Comments in Your Program: Each function and each major block of code must have a comment describing its purpose. Further comments should be added as you feel appropriate to clarify the operation of your program.

Style: Good programming style is essential to writing programs that others can understand. Good style includes (but is not limited to) effective use of comments, good layout on the page, good variable names, and clear flow of events within your program.

Marking Scheme:

This assignment will be marked out of 100:

1. PART 3 : 10
2. PART 4 : 15
3. PART 5 : 10
4. PART 6 : 15
5. PART 7 : 20
6. PART 8 : 10
7. COMMENTS : 10
8. GENERAL STYLE : 10