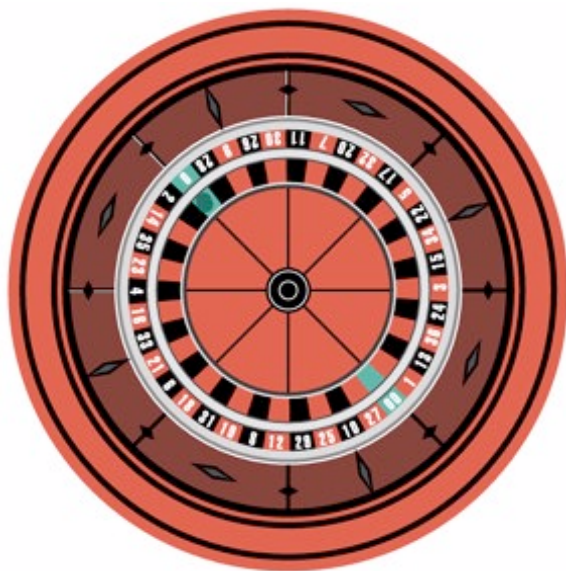


Term Project Ideas

A term project is required of all CSCI E-50a students who have registered for **graduate credit**! The completed project is due in your TF's mailbox no later than 6:00 PM on **Thursday, May 15, 2003**.

We will suggest 3 possible ideas in this document, but you are free to come up with your own. If you choose to do a project that is significantly different from the ones outlined below, you must first submit *a detailed project proposal* to your TF who will advise you as to whether your idea is of reasonable size and complexity. This proposal should be turned in no later than **Monday night, April 14**.

I. Project #1: ROULETTE



A simplified version of *roulette* is very easy to simulate if you know how to use “random” numbers (which you **should**)! The roulette wheel your program will “spin” contains numbers between 0 and 36. Each of these 37 numbers is equally likely to occur. However, whenever the number that comes up is **zero**, the “house” automatically wins (i.e., the human player loses).

The human player is allowed to bet on an actual number between 1 and 36. If he or she guesses right, then the player wins ten times the amount of the bet (i.e., putting down \$25 will result in getting back \$250).

Alternatively, the human player can bet on the winning number being "low" (between 1 and 18), "high" (between 19 and 36), "even" (either 2, 4, 6, ... 36) or "odd" (either 1, 3, 5, ... 35). If he or she guesses right, then the player wins twice the amount of the bet (i.e., a \$25 bet will result in getting back \$50).

Here is what your program might look like in action. User input has been put in a **bold underlined** typeface:

```
OK -- you have $1000 to gamble away.
If you bet correctly on a number between 1 and 36
  you win TEN times the amount of your bet.
If you bet correctly on the number coming up
  HIGH (between 19 and 36), LOW (between 1 and 18),
  EVEN (either 2, 4, 6, ... ) or ODD (either 1, 3, 5, ... )
  then you win TWICE the amount of your bet.
Note:  If the winning number is 0, you always lose.
```

```
How much do you want to bet?  $50
Do you want to bet on an actual number?  yes
```

```
What's your lucky number (1 - 36) ? 47
Don't try to cheat!  You input a bad value!
```

```
What's your lucky number (1 - 36) ? 17
```

```
The wheel goes round and round and round ...
  and it comes up with ... 23
```

```
You lose!  You now have $950.  Want to keep going?  yes
How much do you want to bet?  $150
Do you want to bet on an actual number?  no
```

```
OK -- you can bet on an even, odd, high, or low
  number.  What's your choice?  odd
```

```
The wheel goes round and round and round ...
  and it comes up with ... 11
```

```
You win $300!  You now have $1100.
Want to keep going?  yes
```

```
How much do you want to bet?  $550
Do you want to bet on an actual number?  no
```

```
OK -- you can bet on an even, odd, high, or low
  number.  What's your choice?  low
```

```
The wheel goes round and round and round ...  
and it comes up with ... 20
```

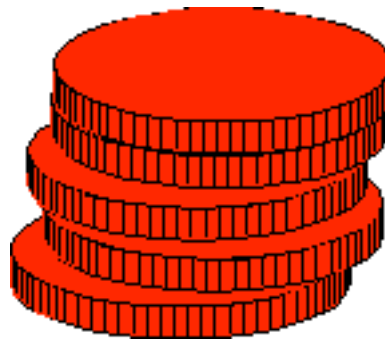
```
You lose! You now have $550.  
Want to keep going? no
```

```
Too bad -- sore-loser.
```

Of course, the precise values which your program prints out will depend upon the sequence of random numbers that gets generated. Refer to the *Unit Three Lecture Notes* for the details on how to generate random integers; you may wish to define a `rint` method similar to the one discussed in lecture or even a *RouletteWheel* kind of Java class. Your program's output need not be identical to what is shown above. However, the program should have a very clear user interface. Make sure you "idiot-proof" your code so the user can't cheat by typing in strange values.

You might also consider how to write this program so that it plays with a *biased* roulette wheel? For example, one might be interested in being able to throw a "high" number approximately 70% of the time, rather than the usual 50% of the time. Be sure to describe your proposed solution in simple, clear, unambiguous English.

You should also consider adding at least one additional feature of your own choice to this project to make it a bit more exciting or interesting.



II. Project #2: Pig-Latin Translation of Text Files



The goal here is to read in the characters stored in a text file, and create a second text file that contains the original text but with every word converted to “Pig-Latin.”

As you probably remember from your youth, U.S. Pig Latin is formed by moving the initial consonant or consonant cluster to the end of the word and adding **-ay**.¹ When the word in question begins with a vowel, dialect differences arise; depending on the region you live in, add **-hay**, **-way**, **-yay** or just plain **-ay**.

Regional differences in US Pig Latin

| | |
|---------------------|---------------------------------|
| <i>English</i> | I speak Pig Latin. |
| <i>Antediluvian</i> | I-ay eak-spay Ig-pay Atin-lay. |
| <i>Paleologic</i> | I-hay eak-spay Ig-pay Atin-lay. |
| <i>Precambrian</i> | I-way eak-spay Ig-pay Atin-lay. |
| <i>Xenophonic</i> | I-yay eak-spay Ig-pay Atin-lay. |

There are even sites on the Internet devoted to Pig-Latin! If you have access to a Web browser such as Netscape, you might want to check out

http://www.girltech.com/HTMLworksheets/GC_Piglatin.html
<http://voyager.cns.ohiou.edu/~jrantane/menu/pigframe2.html>
<http://www2.vip.net/~robertja/piglatin.htm>
<http://www-oss.fnal.gov/~baisley/pigisms.html>

Here’s what your program might look like in action, assuming we stick with the *Xenophonic* dialect:

```
What is the name of your input file: foo.old  
What is the name of your output file: foo.new
```

```
Original File (foo.old):
```

```
Alice was beginning to get very tired of sitting by her  
sister on the bank, and of having
```

¹ The only exception to this simple rule would be words that begin with “qu” — in which case both letters are considered to be consonants; thus, “quick” translates into “ickquay”.

nothing to do. Once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

New File (foo.new):

```
Aliceyay asway eginningbay otay etgay eryvay iredtay ofyay ittingsay
byay erhay istersay onyay ethay ankbay, andyay ofyay avinghay
othingnay otay oday. Onceyay oryay icetway eshay adhay eepedpay
intoyay ethay ookbay erhay istersay asway eadingray, utbay ityay
adhay onay icturespay oryay onversationscay inyay it,yay 'and
atwhay isyay ethay useyay ofyay ayay ookbay,' oughtthay Aliceyay
'ithoutway icturespay oryay onversationcay?'
```

Some tricky things to watch out for:

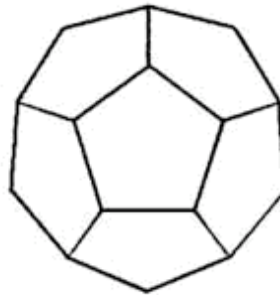
- handle uppercase characters intelligently. For example, “Jack” should appear as “Ackjay” in Pig-Latin, and not as “ackJay”.
- punctuation should end up in the right place — either just before the beginning of a word or right at the end of the word. Notice in the above example, how the comma, question-mark, apostrophe and period were handled.
- since you should not use arrays in solving this problem, you may safely assume that no word will begin with more than 3 consecutive consonants (e.g., “stretch” should be correctly handled).



III. Project #3: Hunt the Wumpus

One of my favorite old computer games is “*Hunt the Wumpus.*” Here’s a brief description:

The mythical Wumpus lives in a cave of 20 rooms. Each room has 3 tunnels leading to other rooms. (You might want to look at a *dodecahedron* to



see how this works;
vertices.)

it’s a shape with 20

The player’s objective is to find and slay the Wumpus.

Unfortunately, there are hazards:

- (a) *Bottomless Pits* — Two rooms have bottomless pits in them. If you go there, you fall into the pit (and lose!)
- (b) *Super Bats* — Two rooms contain super bats. If you go there, a bat grabs you and takes you to some other room at random (which might be troublesome).

The Wumpus is not bothered by the hazards (he has sucker feet and is too big for a bat to lift.) Usually he is asleep. Two things wake him up: your entering his room or your shooting an arrow.

If the Wumpus wakes, he sometimes runs to another room via a tunnel. If you happen to be in the same room with him, you lose.

You (the player) can at each turn either *move* or *shoot* a crooked arrow. If you decide to move, you can move one room (through one tunnel.) if you decide to shoot, know that you have only 5 arrows. You lose when you run out. Each arrow can be shot from 1 to 5 rooms away. You aim by telling the

computer the room #'s you want the arrow to go to. If the arrow can't go that way, it moves at random to the next room.

If the arrow hits the Wumpus, you win. If the arrow hits you, you lose.

Warnings: When you are one room away from Wumpus or hazard, the computer says:

| | |
|-------------------------|-------------------|
| <i>I smell a Wumpus</i> | or |
| <i>Bats Nearby</i> | or |
| <i>I feel a draft</i> | (when near a pit) |

Note that initially when you start the game your Java program will create a brand-new maze, linking the rooms in some “random” fashion, locating the hazards randomly, and placing you into one of the rooms at random.

To get a good idea of how the game is played, visit one of these websites

<http://www.taylor.org/~patrick/wumpus/>
<http://scv.bu.edu/htbin/wcl/>
<http://www.mssc.edu/compsci/Wumpus.htm>

IV. Choosing Your Own Topic

If you don't like any of our suggestions, try to think of areas in which you have experience — there really is not enough time during the last few weeks of this course in which to learn a new subject and to then write a program in that area. For example, if you do not know how to play checkers, it would be unwise to attempt to write a program which plays an intelligent game of checkers. Furthermore, if you are unfamiliar with the area you choose to work in, you may very well start on a project which is simply impossible! If you are bewildered, sit down with your teaching fellow and brainstorm — some of the best projects have had the most unlikely beginnings, I assure you.

One of the most difficult — and most important — aspects of the topic-choosing stage is to formulate a set of limited objectives. We have learned that students can drastically underestimate the difficulty of the problem they choose to solve, and the amount of time necessary to complete the project. If you insist on undertaking a grandiose project, be sure to plan it in stages — i.e., as a basic program, plus some “frills” — so that if you run out of time or patience (or computer memory), you will still have a complete project to

hand in. Do not attempt to shoot the moon, and miss. I repeat, your teaching fellow can probably help you pick something of the right size — so I urge you to discuss your ideas as soon as possible.

Note that the specifications of the four possible projects we suggest were left somewhat vague and a bit open-ended in order to get you to think through the problems in your own way and to encourage creative implementations.

V. Doing the Project

Your goal is, of course, to construct a program or set of programs which runs. But that goal will not be attained by sitting down at the computer as soon as you think your idea (or one of our suggestions) is clear in your own mind!

You must plan your program[s] carefully on paper! Think about your hand-written version before you begin typing it in. One hour of solitary reflection can save you five or more hours in the terminal room, as you haplessly try to debug illogical Java code.

Avoid writing long, repetitive programs! A 50-line program which exhaustively checks 50 similar cases is no more interesting than a 5-line program which checks 5 cases. Avoid programs which require typing lots of data — your time using the computer is probably severely limited, and you could better spend it working on the design of the program itself. Once again — see your teaching fellow early and often for help — 15 minutes of our guidance could save hours of your own time.

The size of your finished product is not especially critical. A small, but well-planned, elegant, ingenious and well-documented program is to be preferred to a long, rambling, poorly structured monstrosity (as if you didn't already know). Even a three or four page Java program could be an excellent project, IF the code is tightly knit, and well thought out. Think about using parameter passing, and keep the number of global/static variables to an absolute minimum. Be adventurous and free to use any features of the Java language which we have not yet covered. We encourage you to make use of Java classes and objects if you feel at all comfortable using this feature of the language.

If you find it absolutely, positively impossible to finish your work by the due date, please remember that a program can be interesting and useful without being a finished product, so long as it was well thought out to begin with. Your project is not going to be evaluated simply on the basis of how many hours we think you spent in the terminal room, nor on the sheer size of the program[s].

VI. What To Hand In

Your project should not be regarded as a puzzle for the course staff to figure out. It must be well-commented and annotated, of course. But it takes far more than this to communicate a term project's structure and usefulness, so you must give some thought to the write-up which will accompany your program listings and "monitor sessions" — well in advance of the due date.

We expect at least a couple of pages of written explanation and some examples of the program's operation, in addition to the program listings (with their associated comments and other documentation). Flowcharts can be very useful. Many of the best write-ups in past years have carefully incorporated all of the aforementioned elements into a single, unified presentation — as opposed to a disorganized document containing haphazard program listings separate from descriptions of what the programs do, etc. Make certain that your write-up does justice to your programs!

If you feel you have done something which is just too dynamic for the printed page — e.g., a particularly clever "graphics" project — you may want to arrange for an appointment with your teaching fellow so that you can demonstrate your program. But don't do this as a substitute for a lucid write-up.²

² You should actually supply clear, detailed instructions so that your TF can figure out how to run your program without your needing to be around.

VII. Lest There Be Any Confusion ...

The publication *Rules Relating to College Studies* explicitly prohibits students from submitting "the same paper in substance in two or more courses without the prior written permission of the instructors involved". Sometimes, joint projects with other courses have been approved, but they must be cleared with Dr. Eigen and the instructor of the other course in advance . If your term project idea is based on a published article, please be certain to acknowledge the reference[s]!

Beware of plagiarism: the CSCI E-50a course staff is familiar with most Java textbooks and related publications, so don't even think of trying to pass off someone else's work as your own.

As mentioned, completed term projects are due the last week of classes - i.e., the week before the final. There will be positively no exceptions made! You have had more than enough experience by now to imagine just how overcrowded the computer facility can be just when you need it most; or how it can be "unavailable" unexpectedly — sometimes destroying an hour or more of work — and preventing you from running your programs just when you had planned. We hope that most of you will attempt to complete your projects early, so that you can enjoy a peaceful and boring end of semester, and leave the competition for those precious hours on the computers to your less far-sighted peers. Although we will not be offering any material incentives for early term projects, we think the advantages are evident.

VIII. Grading the Term Project

Your project will be graded on a scale of 0 to 10 points. As we have been saying, the size of your project is not really crucial. In fact, you should probably try to prevent your programs from getting much longer than around 7 to 10 printed pages, in total.

Your project will be evaluated in terms of the following criteria:

- (a) *Does the program do what it is supposed to do?* (5 points)

In other words, if you make some claims about what your program does, be certain it works the way you say. We occasionally receive projects which have been tested out in a grossly incomplete manner. Though it might appear to the student that his or her program runs correctly, it is quite possible that when we try it out, we will discover some simple cases which cause one or more procedures to "bomb out".

In addition to working for "normal" inputs, you will be graded on how well your program is *idiot-proofed* and how *user-friendly* your program appears to be. While it is unreasonable to expect you to completely foolproof a term project, you should nonetheless include tests which ensure that the user will be prevented from typing in information that causes the program to malfunction. If, for example, there is a part of your program that expects the user to input a value between 1 and 5, then your program should test whether, in fact, the person using the program did type in a proper value — and if [s]he did not, to report this condition to the user, and take appropriate action!

Try to do exhaustive testing of the program as part of your term project. Any features not tested will not be counted as part of the project. Of course, your program should not contain any syntax errors, nor execution errors with normal inputs.

(b) *Presentation* (2 points)

Your write-up **is** of importance to us! Your presentation should include a clear description of the project and how it works. This can be, to some extent, an updated and more specific version of your proposal (which we will describe shortly). Also included under this “presentation” heading are such factors as the clarity with which you documented your programs (don't bother writing foolish and unnecessary comments next to every *Java* statement). Remember, *flowcharts* can be of great help in your presentation.

Your write-up should be typewritten (or very neatly hand-printed). They are generally anywhere from 2 to 8 pages in length. But be reminded once again: We are not asking for quantity, so keep the “bull” to a minimum.

(c) *Programming Sophistication and Style* (3 points)

Yes, friends, you will also be graded on programming *style*. Please try to obey the rules of “object-oriented” and structured programming that we have emphasized in this course. In particular, this involves modularizing your project by using methods. It really is unreasonable for any individual *Java* method to be much longer than 20 or so lines. Break up your programs into smaller routines. Form the habit of writing methods to do small tasks so that your main program will be uncluttered, highly readable and easily modifiable.

A well-structured main program might simply consist of four or five statements — each of which is a “call” upon a *Java* method (which, in turn, calls upon other methods) to accomplish various tasks. Particularly cunning data structure representation of information is something to watch out for. The many examples we will have done in lecture and in section should provide ample motivation.