This is the take-home portion of the ECE 594 final, Fall 2017.

Submit as a word or pdf document that is clearly readable - choose sensible fonnts.
Start each question on a new page.

Submit by email by 5 pm on Thursday 21st December.

5. Referring to the “bouncing balls” example from our exploration of Javascript:
`https://github.com/alexwarth/um-js/blob/master/lec2-balls.js`

Modify this script so that…

a. It does not allow the user to add new balls.

b. It starts with exactly two balls in the simulation, each with a random position and velocity

c. The two balls bounce off each other, not just the sides of the canvas.
I don’t mind if the way the balls bounce is unrealistic, you can make it as simple as detecting overlap and reversing the direction of travel.

6.

In number theory, the *Peano Axioms* are used to define the natural numbers (i.e. positive integers) and the arithmetic operations on them. This is how it goes:

- A constant `z` is used to represent the base case for natural numbers, which we normally refer to as zero.
- A function `s` (for successor) is used to mean "the next natural number after".

So `z`, `s(z)`, `s(s(z))`, `s(s(s(z)))`, `s(s(s(s(z))))`, ... is a list of all the natural numbers, they are the numbers we normally call zero, one, two, three, four, ...

It happens that Prolog is a good language for working with numbers expressed in this way.

This predicate will produce a list of all the natural numbers:

```
    number(z).
    number(s(X)) :- number(X).
```

"Zero is a number. If X is a number then so is the successor of X".

We can define addition like this:

```
    add(z, Y, Y).
    add(s(X), Y, s(Z)) :- add(X, Y, Z).
```

"Zero plus anything is equal to that same thing. if X plus Y is Z, then (the successor of X) plus Y is the successor of Z".

The predecessor function ("the natural number before") also has a simple definition:

```
    pred(s(X), X).
```

"Zero hasn't got a predecessor, so there is no rule for that, but the predecessor of the successor of X is X itself".

Give Prolog definitions in this style for:

a.
Subtraction
(so that `sub(`*X*`,` *Y*`,` *Z*`)` means *X* minus *Y* is *Z*)

b.
Multiplication

c.
Division

d.
Factorial

7.

Referring to the Prolog English grammar example:

```
http://rabbit.eng.miami.edu/class/een594/gr.txt
```

This program already “understands” simple declarative sentences such as

red dogs eat cake
he chased a big hairy cat

Extend it so that it understands statements of belief such as

my cat thinks red dogs eat cake
i believe he chased a big hairy cat