Monte Carlo Simulation

Today

1. Review of programming errors and special floating point values
2. Review of structs
3. Monte Carlo Simulation

Programming Errors Review

- Be aware of how floats are stored
- Never do `if (total == 200.0)`
- Avoid subtractive cancellation
- int overflows, float overflows, NaN, inf, -inf
- In the exam, be prepared to interpret some code, maybe writing small bits of code *hint*

Exam Info

- Questions might be:
  - Asking what happens when a bit of code is executed.
  - What happens when an integer overflows
- Be prepared to write little bits of code
- Such as, write a bit of code to test if a float is NaN
- Exam will be different from last year
- Plenty of review will be done, and sample questions will be given

Struct Review

```
struct dog {
    int age;
    char name[80];
};

void renameDog(dog* d) {
    strcpy(mydog->name, "Spike");
}

int main() {
    dog mydog;

    mydog.age = 3;
    strcpy(mydog.name, "Fido");

    printf("dog name is %s\n", mydog.name);

    renameDog(&mydog);

    printf("dog name is %s\n", mydog.name);
}
// this is the stack version
```

Randomised Algorithms

- There are algorithms that:
  - Take an infinite amount of time
  - Always produce correct results, with different resource demands
  - Produce correct results 75% of the time

Randomised Algorithms

- There are algorithms that:
  - Take an infinite amount of time
    - **(Monte Carlo Algorithms)**
  - Always produce correct results, with different resource demands
    - **(Las Vegas Algorithms)**
  - Produce correct results 75% of the time
    - **(Atlantic City Algorithms)**
- **There are also generally stochastic algorithms**

Monte Carlo Simulation

- Monte Carlo methods are a class of computational algorithms
- Repeated random sampling for results
- No precise definition

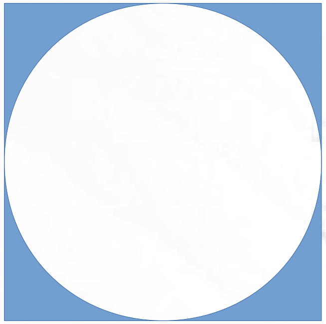
Monte Carlo Simulation

- Typical pattern:
  - Define a domain of possible inputs
  - Generate inputs randomly over the domain
  - Perform a deterministic computation on the inputs
  - Aggregate (combine/average/total) the results

Example: calculating Pi

- Draw a square, and then draw a circle inside the square touching it at 4 points

Example: calculating Pi



- Area of circle = πr^2
- Area of square = $2r \cdot 2r = 4r^2$
- So (area of circle) / (area of square) = $(\pi r^2)/(4r^2) = \pi/4$
- This ratio is what is important

Doing the simulation

- Generate points randomly inside the square
- N = total number of points
- M = number of those points that are also in the circle
- If you generate enough points, then the ratio of M to N would be the same as the ratio of square area vs circle area

$$M/N = \pi/4$$

Results

```
N is 100000000, M is 78542699
Ratio M/N = 0.79, Pi = 3.14170796
```

- 100 million for Pi correct to 3 decimal places...
- Can always make it a 100 billion, but that would take my computer a couple of days
- Can always use a supercomputer?
- Use a better RNG?

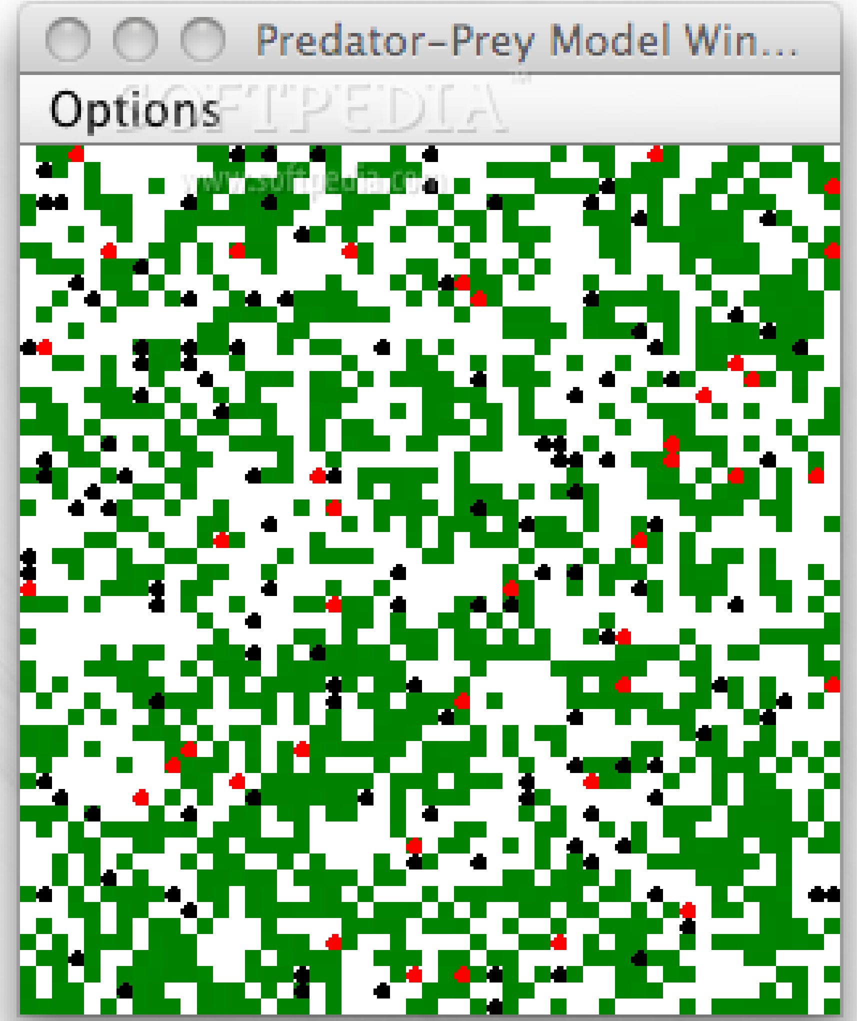
Monte Carlo Conclusions

- It's often useful, but very slow when you need a very precise answer
- Slow because a LOT of random numbers must be generated

Calculating chance events

- Can use Monte Carlo for picking simulation parameters
- Useful for Predator-Prey simulations
- For example, what should the chance be, of a fox catching a rabbit?

Predator-prey simulation



Predator-prey

- First approach:
  - Take all factors into account, eg. speed of animals, fitness, weather, terrain, etc.
  - Set up a complicated formula
  - Calculate if the fox catches the rabbit.

Predator-prey

- Second approach - use chance
  - The fox has a 56% chance of catching a rabbit.
  - Generate a random number in [1,100]. If the number is 56 or less, then the fox catches the rabbit.
  - Very simple, does not require complicated maths
  - Modify chance until it feels right - maybe 48% or 60%
  - Produces surprisingly realistic results