

consider a "dog fight" between a **red** fighter and a **blue** fighter (here, fighter = combat aircraft).

If the probability of **red** fighter winning a 1:1 duel is *p*, what is the average number of **blue** aircraft shot down before the **red** is shot down?

In a series of 1:1 duels, it is:

$$p + p^2 + p^3 + \dots = \frac{p}{1 - p}$$

which will be 1 for  $p = \frac{1}{2}$ , 2 for  $p = \frac{2}{3}$ , etc.

## What about n:m fights?

If *n* fighters fight against *m* fighters, the result is more complicated.

This is not a series of 1:1 combats, because all the fighters are engaged at the same time. consider a 1:2 combat. The average number of **blue** aircraft shot down before the (single) **red** fighter is shot is not  $\frac{p}{1-p}$  anymore.

## How to capture the fight?

One way is to divide it into rounds. In each round each side shoots at the other, destroying aircraft if a hit is scored (every hit is fatal).

The rounds continue until one side is wiped out (there is no point to model partial fights, because they may be seen as fights that will resume in the future).

To make it realistic, we must assume that the probability of a hit is small in each round (say  $\Delta p$  and  $\Delta(1-p)$  for the two sides ( $\Delta$  could be 0.01).

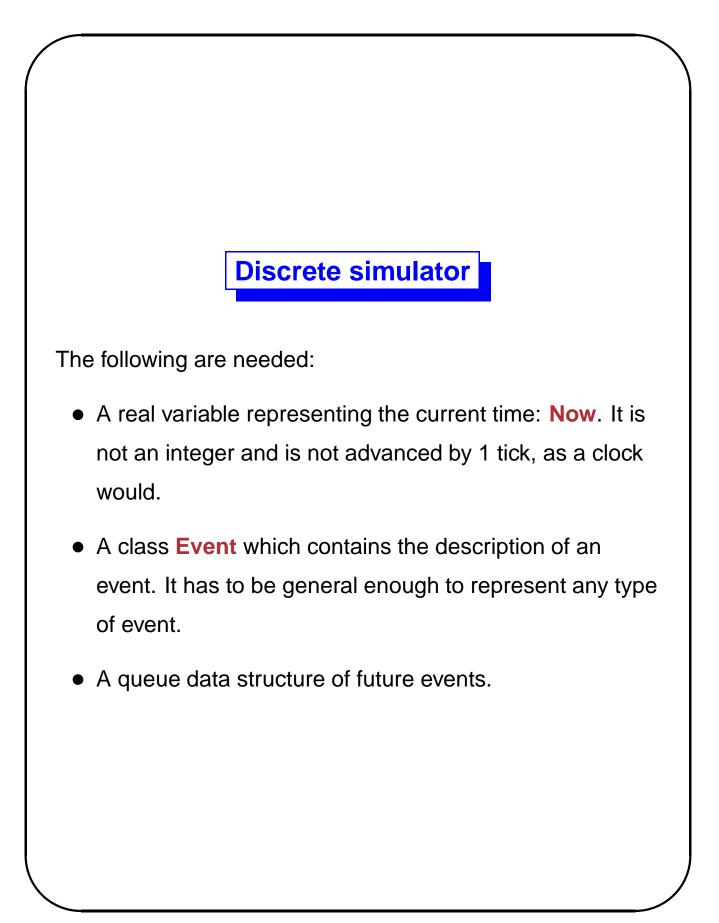
```
bool Step( int &blue , int &red )
{
  for( int i = 0 ; i < blue && red > 0 ; i++ ) {
    if( red > i )
       if( drand48()  )
         blue--;
    if( blue > i )
       if( drand48() < (1-p) * Delta )
         red - -;
  }
  return blue ! = 0 \&\& red ! = 0;
}
```

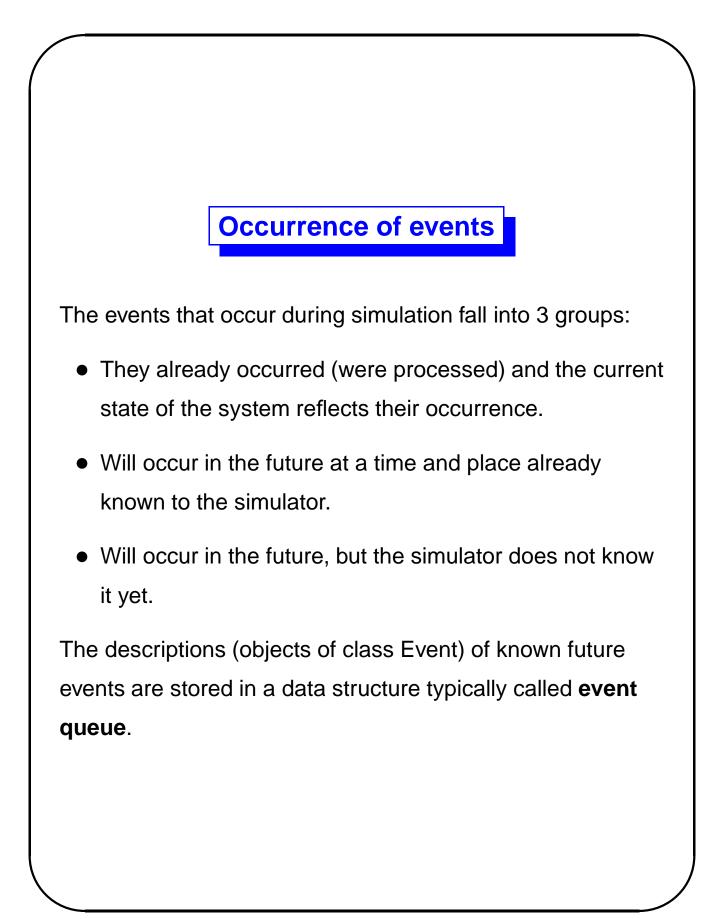
Note the slight advantage given to red (shooting first).

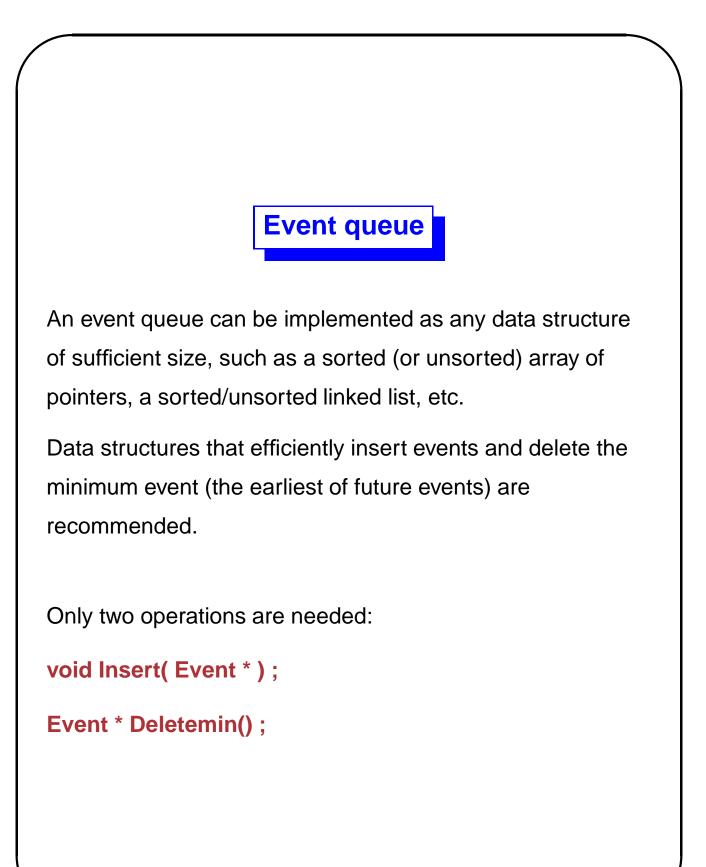
```
for( int i = 0 ; i < N ; i++ ) {
     int red = Reds , blue = Blues ;
     for( int j = 0; Step( blue , red ); j++ );
     redvic[blue]++ ;
  }
  int down = 0;
  for( int i = 0; i < Blues; i++)
     down += redvic[i] * (Blues - i);
  cout \ll (double)down/N \ll " <code>scores\n"</code> ;
}
```

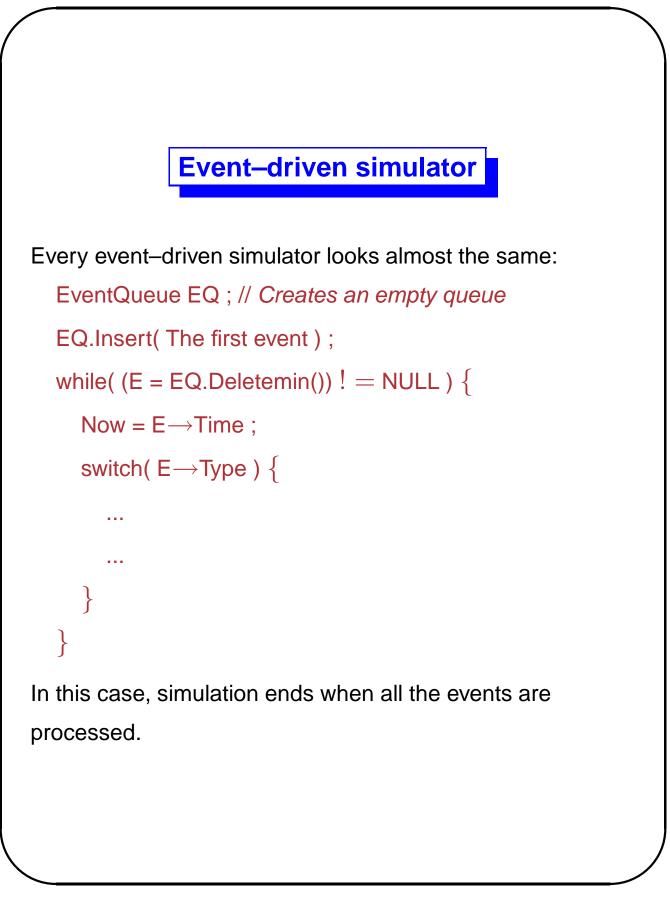
Average yield per plane										
Quality	Original odds in each combat									
	1:1	1:2	2:4	4:8	1:3	2:6	1:4			
$\frac{1}{2}$	1	0.607	0.463	0.373	0.392	0.279	0.283			
$\frac{2}{3}$	2	1.262	1.041	0.879	0.851	0.620	0.607			

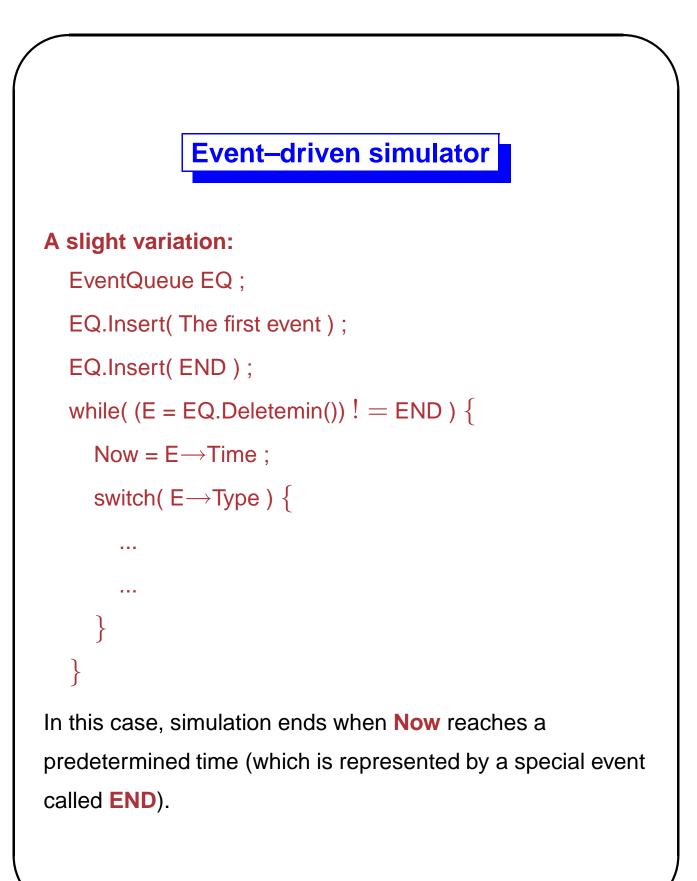
New odds after day of combat											
Quality	Original odds in each combat										
	1:1	1:2	2:4	4:8	1:3	2:6	1:4				
$\frac{1}{2}$	1	0.112	0.034	0.006	0.015	0.0016	0.0023				
$\frac{2}{3}$	2	0.289	0.166	0.077	0.059	0.015	0.0128				

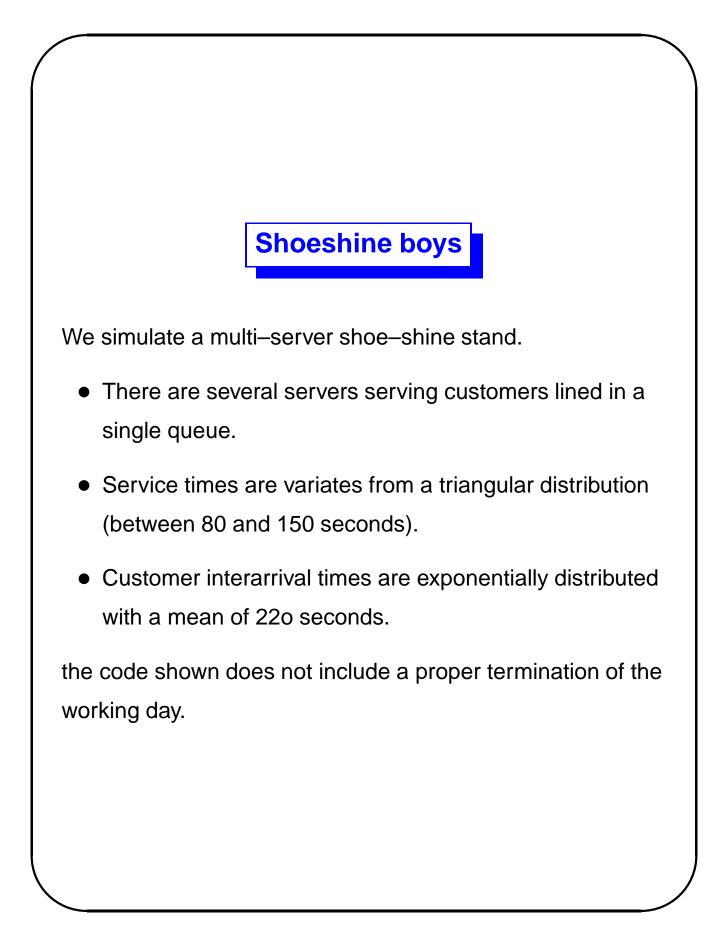












```
EQ.Insert( NextCustomer( Now , waiting ) ) ;
while( (E = EQ.Deletemin()) ! = NULL ) {
  Now = E \rightarrow time;
  if( E \rightarrow Type == DONE ) {
     if( waiting > 0 ) {
       waiting --;
        EQ.Insert(Finish(Now));
     } else
        idle++;
  \} else if( E\rightarrowType == ARRIVAL ) \{
     if( idle > 0 ) {
       idle--:
        EQ.Insert(Finish(Now));
     } else
       waiting++ ;
     EQ.Insert( NextCustomer( Now ) );
  } else ; // trouble
}
```

```
Time Triangular()
{
  double x = drand48();
  return (MAX – MIN) * (x + drand48()) / 2.0 + MIN ;
}
Time Exponential()
{
  return -MEAN * log( drand48() );
}
Event *Finish( time t )
{
  return new Event( t + Triangular(), DONE );
}
Event *NextCustomer( Time t )
{
  return new Event( t + Exponential(), ARRIVAL);
}
```

typedef double Time ; #define EQSize 100 #define MEAN (220) #define MIN (90) #define MAX (150) #define OPEN 0 #define CLOSED 10000000 #define ARRIVAL 1 #define DONE 2

```
class Event {
  public:
     Event(Time T, int t) { time = T; Type = t; }
     Time time ;
     int Type;
};
class EventQueue {
  public:
     EventQueue() { last = -1 ; }
     void Insert( Event * ) ;
     Event * Deletemin() ;
     Event *EP[EQSize] ;
     int last;
};
```

```
void EventQueue::Insert( Event *E )
{
  if( E\rightarrowtime <= CLOSED ) {
     if( last+1 == EQSize ) Overflow() ;
     EP[++last] = E;
  }
}
Event * EventQueue::Deletemin()
{
  if (last == -1)
        return NULL;
  int min = last;
  for( int i = 0; i < last; i++)
     if( EP[i] \rightarrow time < EP[min] \rightarrow time )
        min = i;
  Event *ret = EP[min];
  EP[min] = EP[last - -];
  return ret;
}
```