STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Software Modeling Tools

SYS 611: Systems Modeling and Simulation

Paul T. Grogan, Ph.D.
Assistant Professor
School of Systems and Enterprises

Agenda

1. Spreadsheet Modeling Tools (Excel)
2. Scripted Modeling Tools (Python)

Optional Reading: Python Tutorial

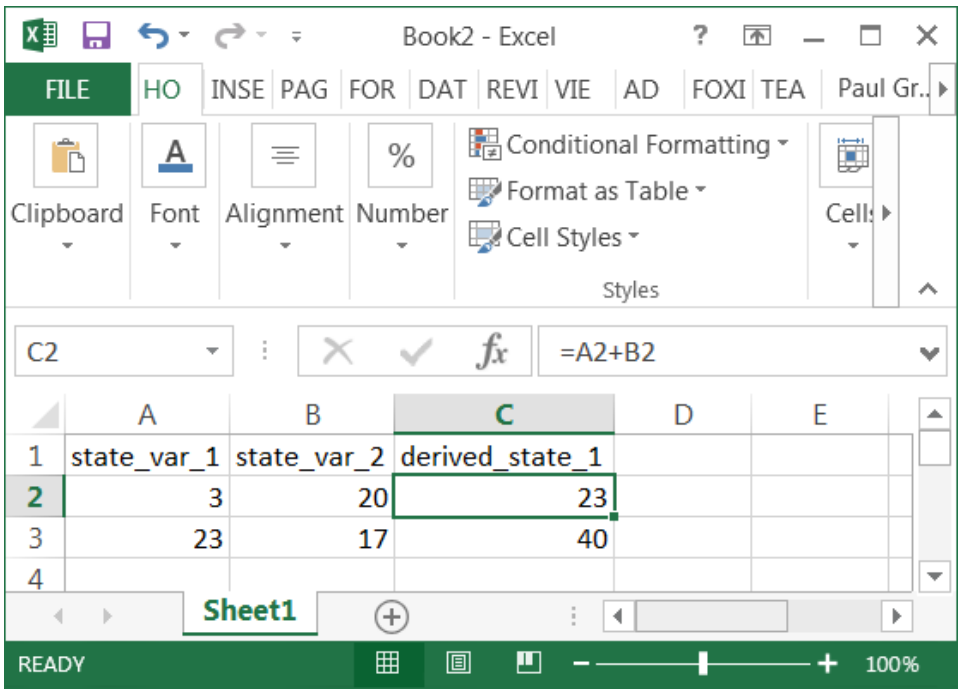
SYS 611 GitHub Repository:

https://github.com/code-lab-org/sys611

Spreadsheet Modeling Tools

Spreadsheet Modeling Tools

- Several variations
  - Microsoft Excel
  - Google Sheets
- Cells contain either elementary state variables or derived state/state transition functions
- Use relative cell equations to easily repeat functions



Spreadsheet Modeling Tools

Advantages

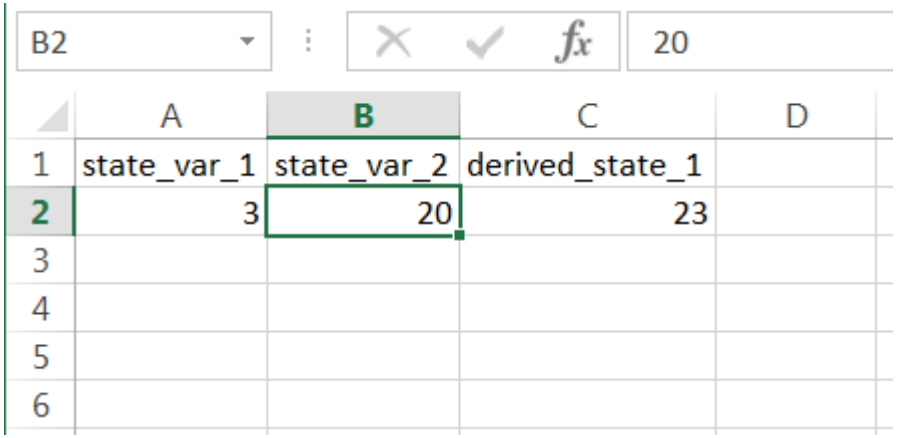
- Widely available
- Low barrier to entry
- Visual state representation
- Powerful computational engine

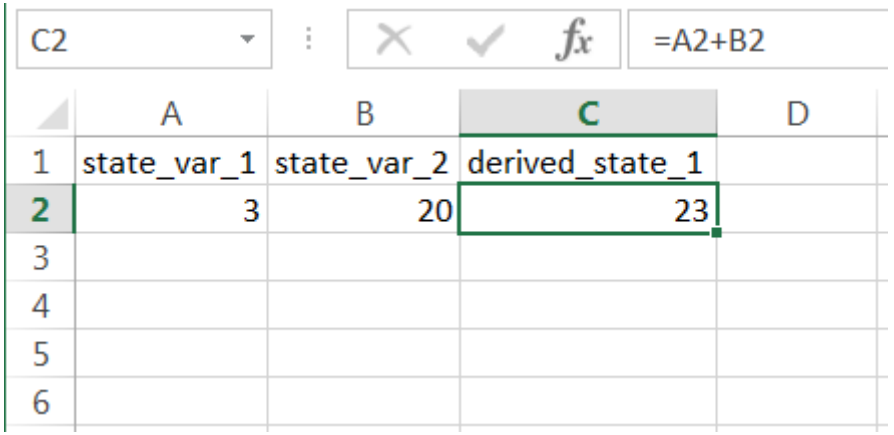
Disadvantages

- State often limited to one-dimensional vectors
- Function chains can be difficult to debug
- Limited ability to document model

Representing State in Excel

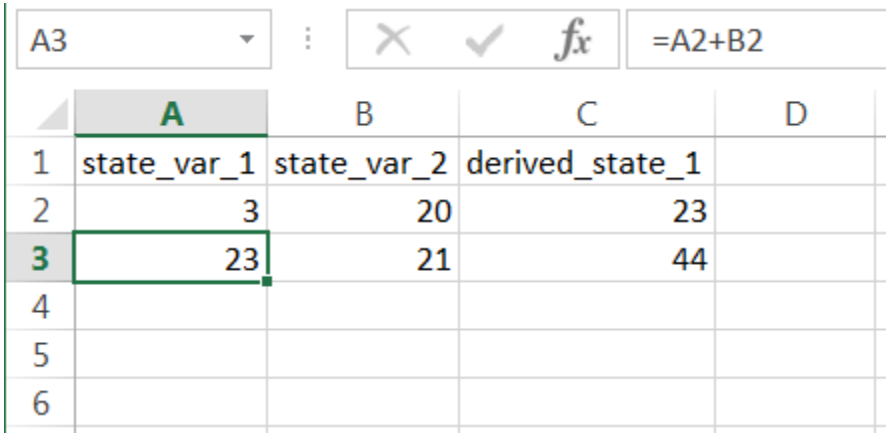
- Use columns for each state variable
- First non-header row is “initial state”
- Elementary state variables stored as raw data
- Derived state variables stored as functions

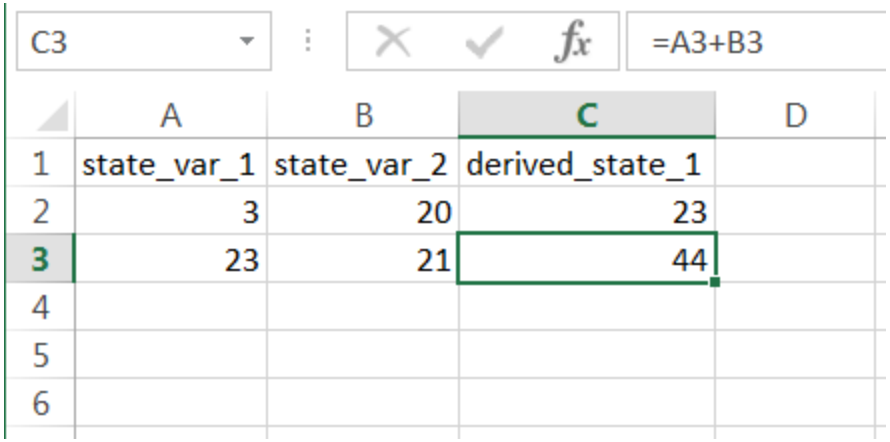




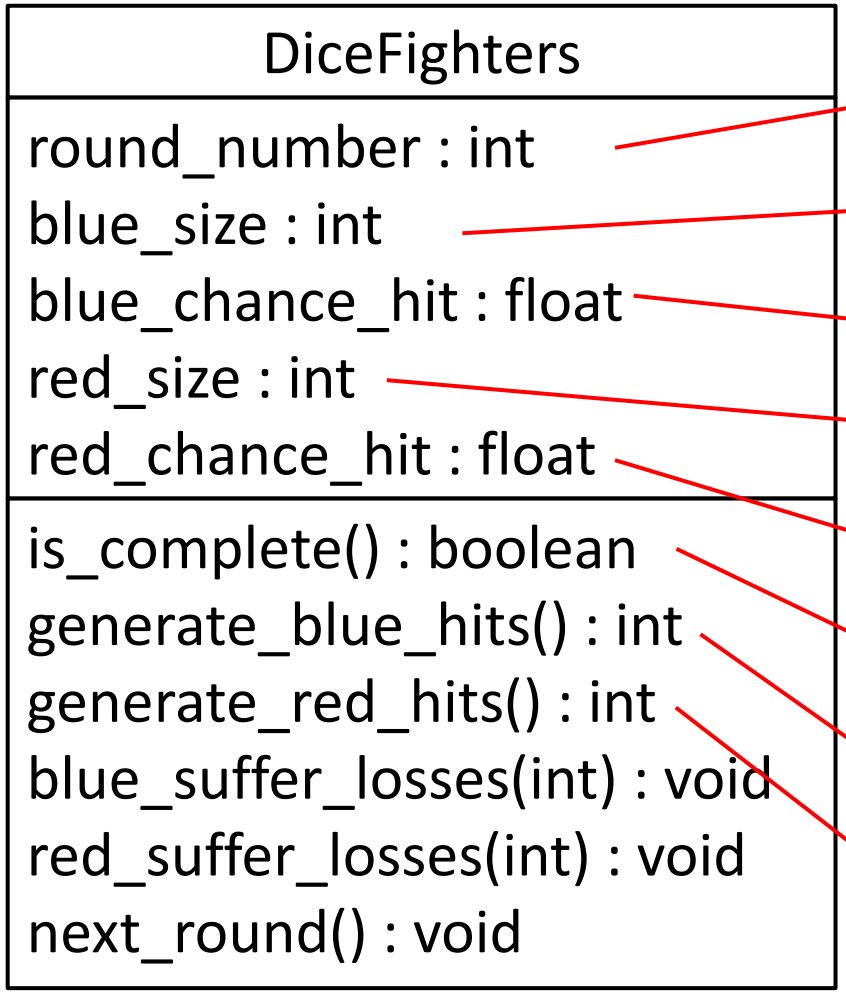
State Changes in Excel

- State changes represented in subsequent rows
- Elementary state changes use functions based on previous row
- Derived state variables can copy-paste original functions
- Can usually fill down equations into lower rows





State Variables




```
A2 = 0
B2 = 10
C2 = 3/6
D2 = 20
E2 = 1/6
F2 = OR(B2<=0,D2<=0)
G2: ???
H2: ???
```

`OR(B2<=0,D2<=0)` checks if either `red_size` or `blue_size` is less than or equal to 0 as `is_complete`

Derived State Variables

Each of these column labels serve as the index of dice for the blue player (up to 20)

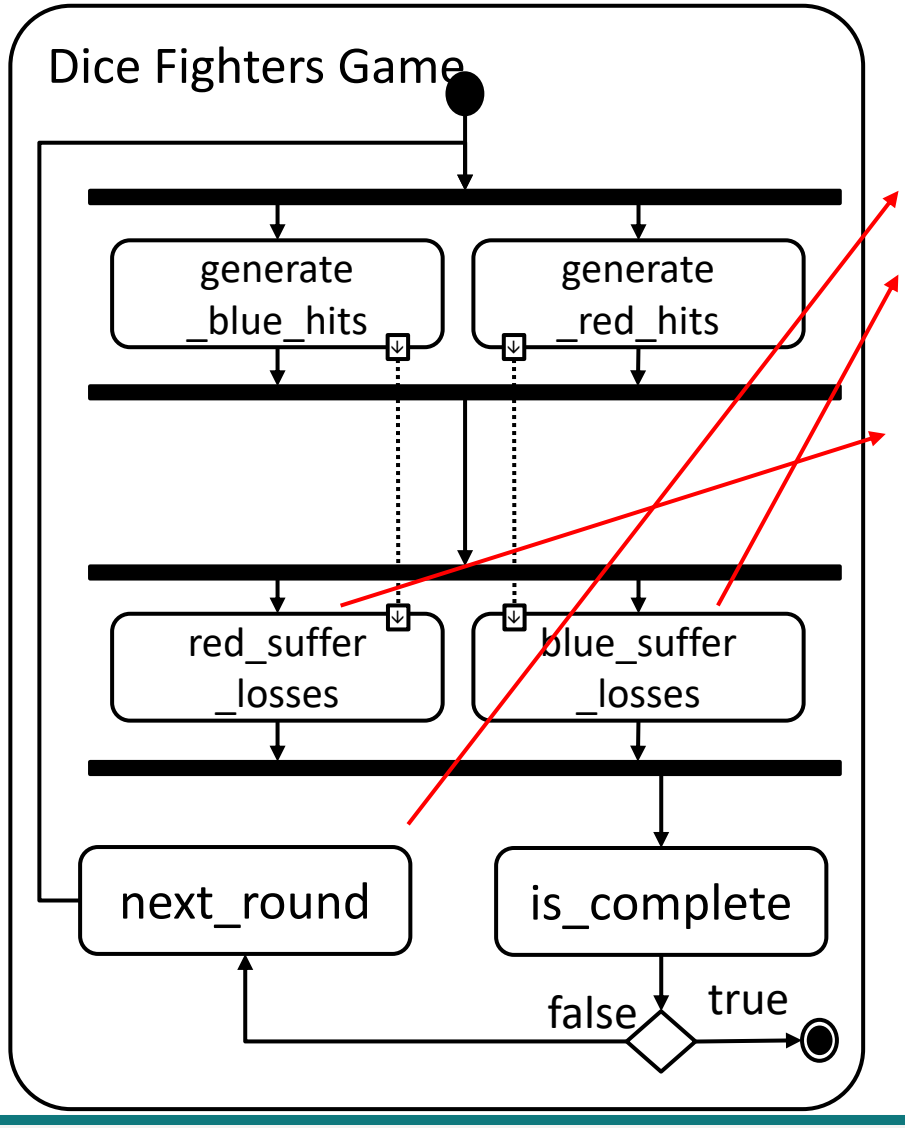
- I1 = 1, J1 = 2, ..., AB1 = 20
- I2 = IF(I$1<=B$2,RAND()<$C2,)
- ...
- AC1 = 1, AD1 = 2, ..., AV1 = 20
- AC2 = IF(AC$1<=$D2,RAND()<$E2,)
- ...
- G2 = COUNTIF(I2:AB2,TRUE)
- H2 = COUNTIF(AC2:AV2,TRUE)

`RAND()<$C2` checks if a random number between 0 and 1 is less than **`blue_chance_hit`**

`IF(I$1<=B$2,...` checks if blue team has large enough size for this dice index

`COUNTIF(I2:AB2,TRUE)` counts how many blue dice generate a hit as derived state **`generate_blue_hits`**

State Changes




```
A3 = A2 + 1
B3 = B2 - H2
C3 = C2
D3 = D2 - G2
E3 = E2
F3 = OR(B2<=0,D2<=0)
G3 = COUNTIF(I3:AB3,TRUE)
H3 = COUNTIF(AC3:AV3,TRUE)
I3 =
IF(I$1<=B$3,RAND()<$C3,)
...
```

Spreadsheet Model

	A	B	C	D	E	F	G	H	I	J	K
1	round_number	blue_size	blue_chance_hit	red_size	red_chance_hit	is_complete	generate_blue_hits	generate_red_hits	1	2	3
2	0	10	0.5	20	0.166666667	FALSE	7	7	TRUE	FALSE	FALSE
3	1	3	0.5	13	0.166666667	FALSE	3	2	TRUE	TRUE	TRUE
4	2	1	0.5	10	0.166666667	FALSE	1	1	TRUE	0	0
5	3	0	0.5	9	0.166666667	TRUE	0	0	0	0	0
6	4	0	0.5	9	0.166666667	TRUE	0	1	0	0	0
7	5	-1	0.5	9	0.166666667	TRUE	0	1	0	0	0
8	6	-2	0.5	9	0.166666667	TRUE	0	2	0	0	0
9	7	-4	0.5	9	0.166666667	TRUE	0	1	0	0	0
10	8	-5	0.5	9	0.166666667	TRUE	0	0	0	0	0
11	9	-5	0.5	9	0.166666667	TRUE	0	0	0	0	0
12	10	-5	0.5	9	0.166666667	TRUE	0	4	0	0	0
13	11	-9	0.5	9	0.166666667	TRUE	0	1	0	0	0
14	12	-10	0.5	9	0.166666667	TRUE	0	0	0	0	0
15	13	-10	0.5	9	0.166666667	TRUE	0	2	0	0	0
16	14	-12	0.5	9	0.166666667	TRUE	0	2	0	0	0
17	15	-14	0.5	9	0.166666667	TRUE	0	0	0	0	0
18	16	-14	0.5	9	0.166666667	TRUE	0	2	0	0	0
19	17	-16	0.5	9	0.166666667	TRUE	0	2	0	0	0
20	18	-18	0.5	9	0.166666667	TRUE	0	1	0	0	0
21	19	-19	0.5	9	0.166666667	TRUE	0	1	0	0	0
22	20	-20	0.5	9	0.166666667	TRUE	0	4	0	0	0
23	21	-24	0.5	9	0.166666667	TRUE	0	2	0	0	0
24	22	-26	0.5	9	0.166666667	TRUE	0	1	0	0	0
25	23	-27	0.5	9	0.166666667	TRUE	0	4	0	0	0
26	24	-31	0.5	9	0.166666667	TRUE	0	2	0	0	0
27	25	-33	0.5	9	0.166666667	TRUE	0	2	0	0	0

Red Victory

Make any change to re-generate random numbers!

Scripted Modeling Tools

Scripted Modeling Tools

- Infinite variations
  - Python, JavaScript
  - Java, C++, C
- **Variables** contain state
- **Functions** operate on state
- Object-oriented scripts define **classes** to combine variables and functions


```
1 # -*- coding: utf-8 -*-
2 """
3 Test Snippit
4
5 @author: Paul T. Grogan, pgrogan@stevens.edu
6 """
7
8 state_1 = 3
9 state_2 = 20
10
11 def derived_state_1():
12     return state_1 + state_2
13
```

Scripted Modeling Tools

Advantages

- Widely available
- Many high-quality third-party libraries
- State can be organized in any structure
- Can document code with inline comments

Disadvantages

- Higher barrier to entry
- Scripts can be difficult to debug without integrated development environment (IDE)
- Function chains can be difficult to debug

Representing State in Python

- Declare a variable to store data in memory
- Primitive data types:
  - Boolean (bool)
  - Integer (int)
  - Floating point (float)
  - Text (str)
- Simple data types:
  - Tuple (tuple)
  - List (list), Matrix

```
state_bool = True
state_int = 3
state_float = 3.
state_str = 'test'
state_tuple = (0,1,2)
state_tuple(0) = 9
state_list = [0,1,2]
state_list[0] = 9
state_mat = [[0,1,2],
             [3,4,5]]
state_mat[0][1] = 9
```

Representing State in Python

- Complex data types:
  - Numpy arrays
  - Dictionary (dict)
  - Object (object)

```
import numpy as np
state_array =
    np.array([0,1,2])
state_dict = {
    'name': 'test',
    'mass': 2.0
}
state_obj = object()
```

- Derived state expressed using functions

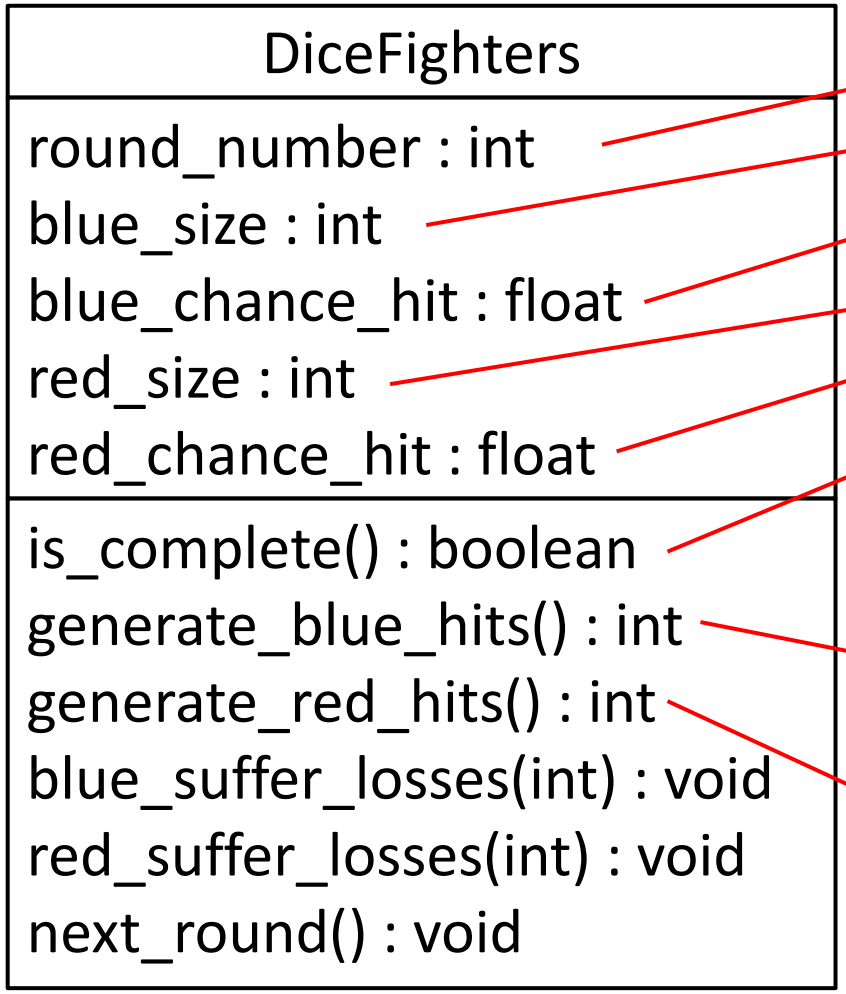
```
state_1 = 3
state_2 = 20
def derived_state_1():
    return state_1 + state_2
```

State Changes in Python

- Declare a function to operate on data
  - Input arguments
  - Output values
- Note: must declare any primitive state variables to be changed as **global** inside function

```
state_int = 0

def add_one():
  global state_int
  state_int += 1

def add_more(num):
   global state_int
   state_int += num

add_one()
add_more(5)
```

State Variables




```
round_number = 0
blue_size = 10
blue_chance_hit = 3/6
red_size = 20
red_chance_hit = 1/6
def is_complete():
    return (red_size<=0
        or blue_size<=0)
def generate_blue_hits():
    pass # see next slide
def generate_red_hits():
    pass # see next slide
```

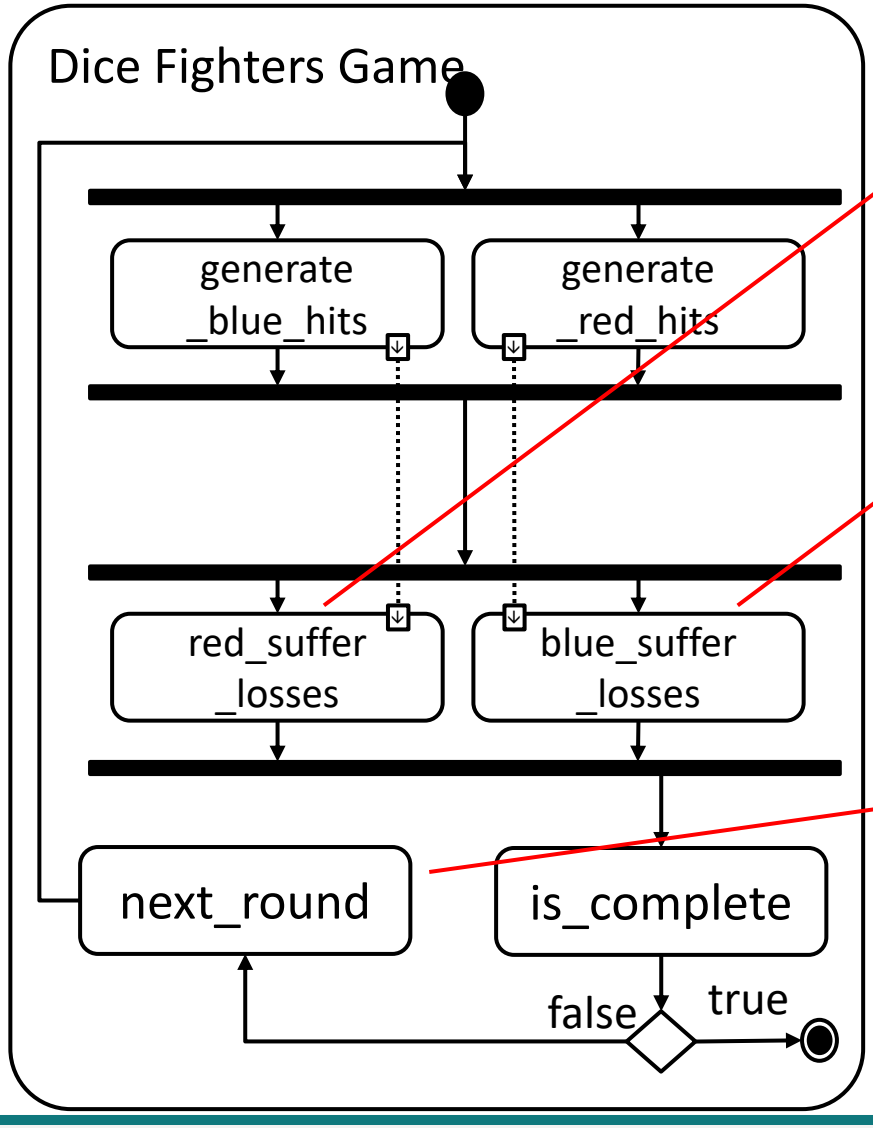
Derived State Variables

```
def generate_blue_hits():
    attacks = np.random.rand(blue_size)
    num_hits = sum(attacks < blue_chance_hit)
    return num_hits

def generate_red_hits():
    attacks = np.random.rand(red_size)
    num_hits = sum(attacks < red_chance_hit)
    return num_hits
```

Generate random numbers between 0 and 1 for each of **`blue_size`**

Count how many attacks are less than **`blue_chance_hit`**

State Changes




```
def red_suffer_losses(hits):
    global red_size
    red_size -= hits
```

```
def blue_suffer_losses(hits):
    global blue_size
    blue_size -= hits
```

```
def next_round():
    global round_number
    round_number += 1
```

Dice Fighters Python Model

```
import numpy as np

round_number = 0
red_size = 20
blue_size = 10
red_chance_hit = 1/6
blue_chance_hit = 3/6

def generate_red_hits():
      attacks = np.random.rand(red_size)
      return sum(attacks < red_chance_hit)
def generate_blue_hits():
      attacks = np.random.rand(blue_size)
      return sum(attacks < blue_chance_hit)
def red_suffer_losses(opponent_hits):
      global red_size
      red_size -= opponent_hits
def blue_suffer_losses(opponent_hits):
      global blue_size
      blue_size -= opponent_hits
def is_complete():
      return (red_size <= 0 or blue_size <= 0)
```

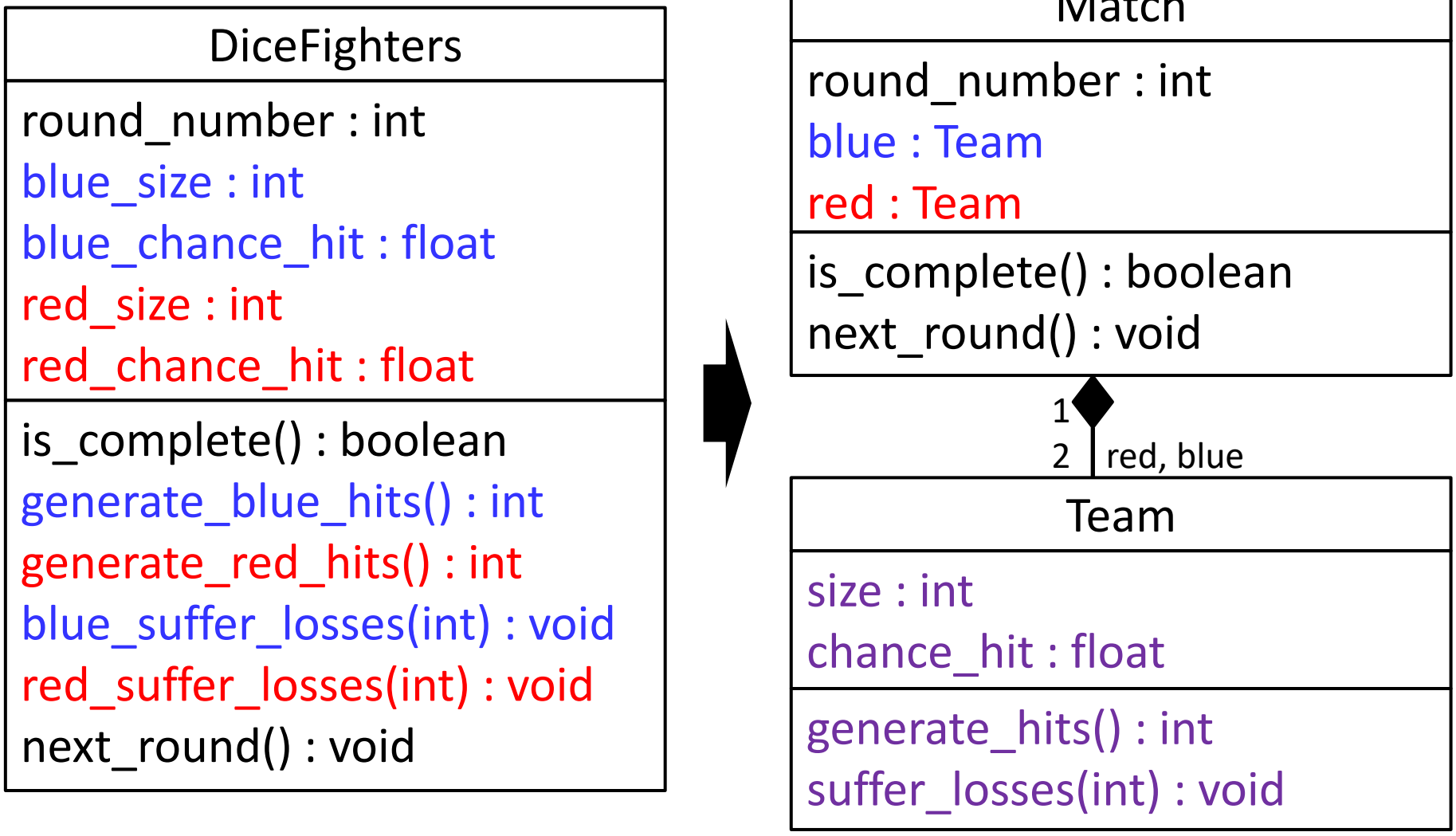
```
def next_round():
      global round_number
      round_number += 1

while not is_complete():
      red_hits = generate_red_hits()
      blue_hits = generate_blue_hits()
      red_suffer_losses(blue_hits)
      blue_suffer_losses(red_hits)
      next_round()
      print("Round {}: {} Red, {} Blue".format(
                  round_number,
                  red_size,
                  blue_size
            ))
if red_size > 0:
      print("Red Wins")
elif blue_size > 0:
      print("Blue Wins")
else:
      print("Tie - Mutual Destruction!")
```

Object-oriented Modeling

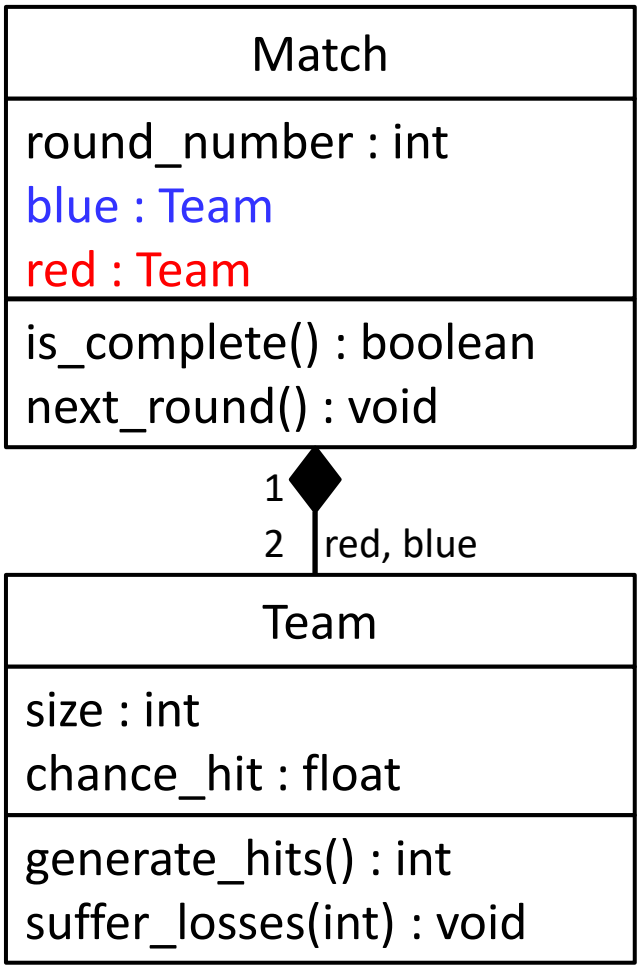
- Define **classes** to combine state with functions
  - Functions have access to local state
  - Eliminates global variable problems
- Useful to organize a complex script
- Useful to reuse code

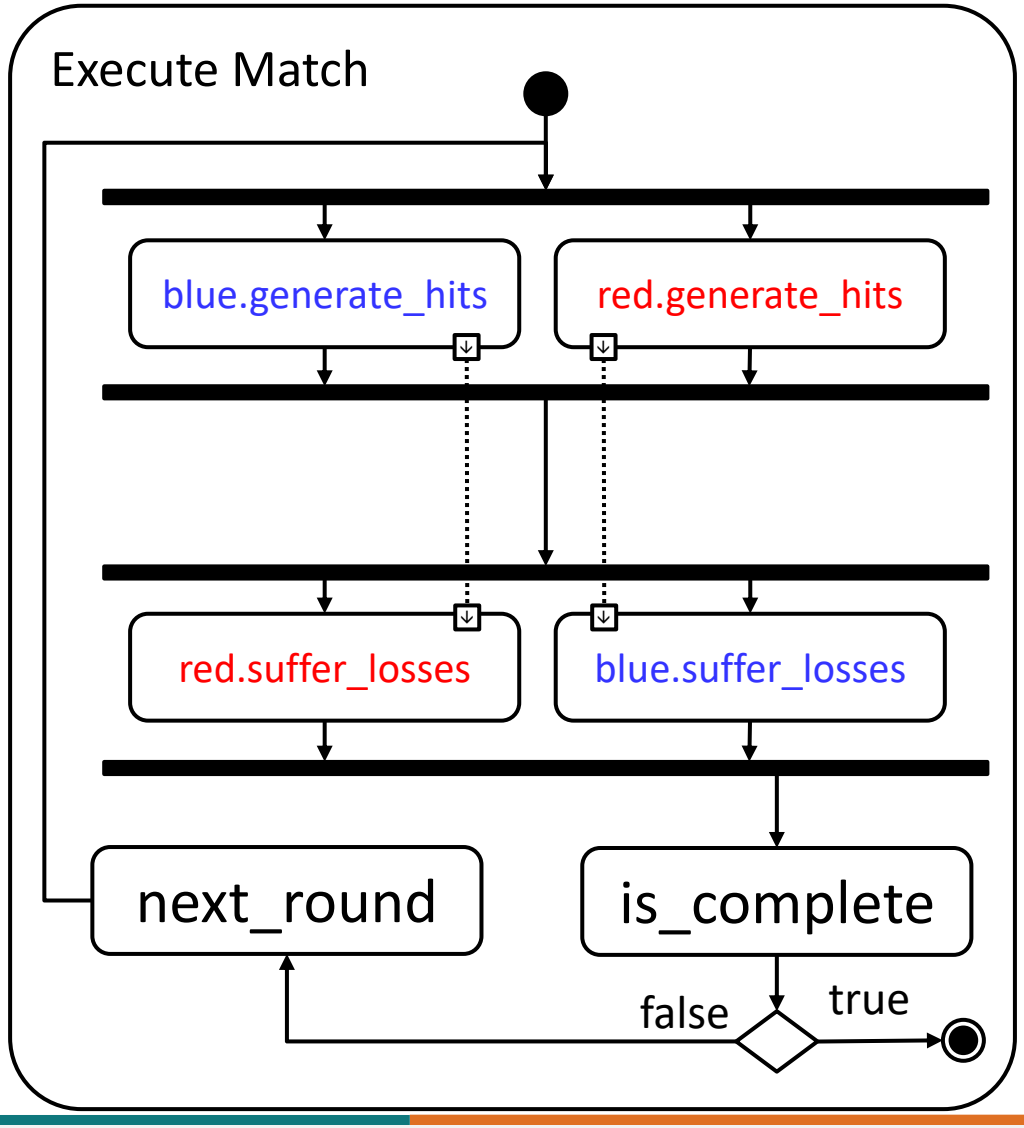
```
class Model:
    def __init__(self, state_1):
        self.state_1 = state_1
        self.state_2 = 3
    def derived_state_1():
        return (self.state_1 +
                self.state_2)
    def add_one():
        self.state_1 += 1
    def add_more(num):
        self.state_2 += num
model = Model(2)
model.state_1
model.add_more(5)
```

Obj.-Orient. State Diagram



Obj.-Orient. Activity Diagram





Obj.-Orient. Python Model

```
import numpy as np

class Team:
      def __init__(self, size, chance_hit):
            self.size = size
            self.chance_hit = chance_hit
      def generate_hits(self):
            attacks = np.random.rand(self.size)
            return sum(attacks < self.chance_hit)
      def suffer_losses(self, opponent_hits):
            self.size -= opponent_hits

round_number = 0
red = Team(20, 1./6)
blue = Team(10, 3./6)
def is_complete(self):
      return (red.size <= 0 or blue.size <= 0)
def next_round(self):
      self.round_number += 1
```

```
while not is_complete():
      red_hits = red.generate_hits()
      blue_hits = blue.generate_hits()
      red.suffer_losses(blue_hits)
      blue.suffer_losses(red_hits)
      next_round()
      print("Round {}: {} Red, {} Blue".format(
                  round_number,
                  red.size,
                  blue.size
            ))
if red.size > 0:
      print("Red Wins")
elif blue.size > 0:
      print("Blue Wins")
else:
      print("Tie - Mutual Destruction!")
```