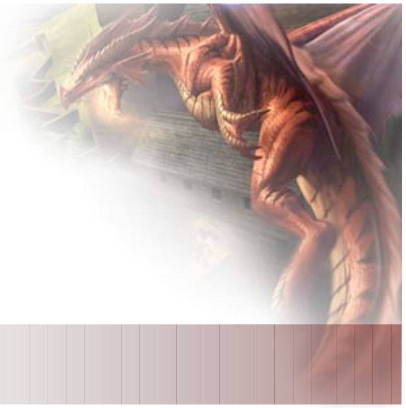
Intro to Programming

CMPUT 299

H. James Hoover

Fall 2005 *2005-10-11*

Version 1.0



Programming is

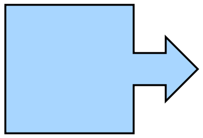
Machine + Instructions

Scripting is

- ♣Programming where the "machine" is often another program or system.
- ♣No real distinction anymore.

Example

- Bob the robot:

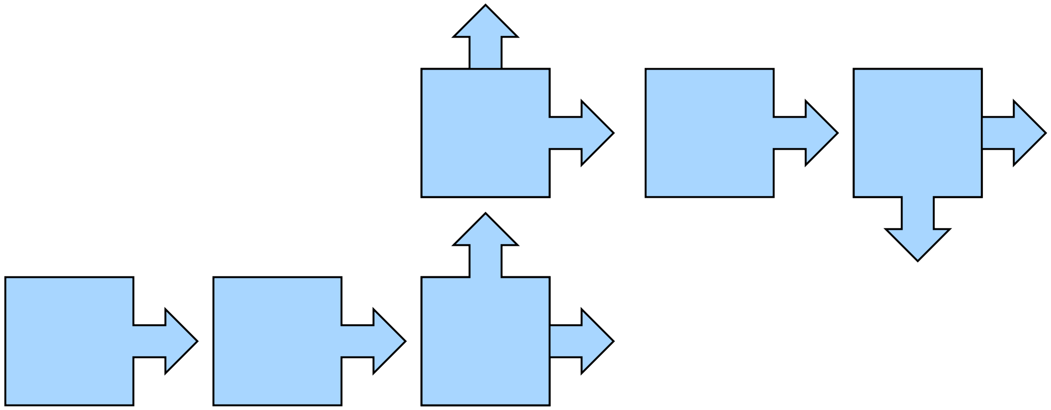


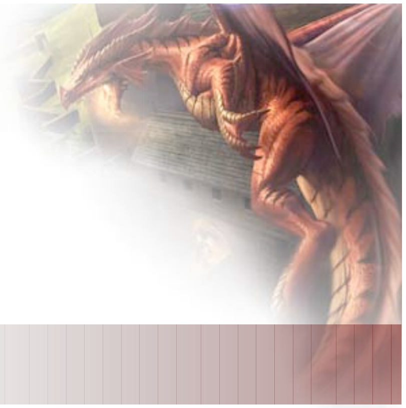
- Instructions:

turn left	L
turn right	R
go forward	F

Straight Line Program

F F L F R F F R



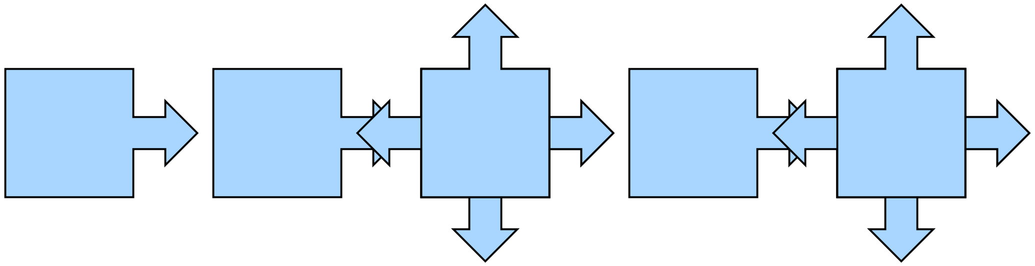


Straight Line Programs

- Simple linear *flow of control*
- Only work in limited, pre-defined contexts
- Building blocks for more complex actions

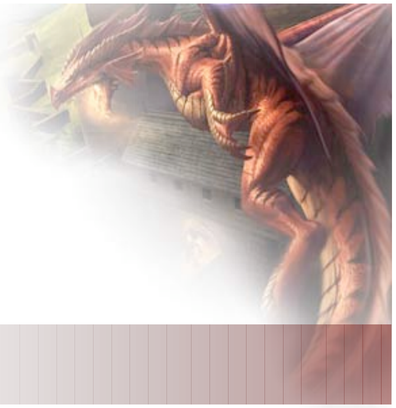
```
Define 2F as F F
Define Spin as R R R R
Define Dance as 2F Spin 2F Spin
```

2F Spin 2F Spin



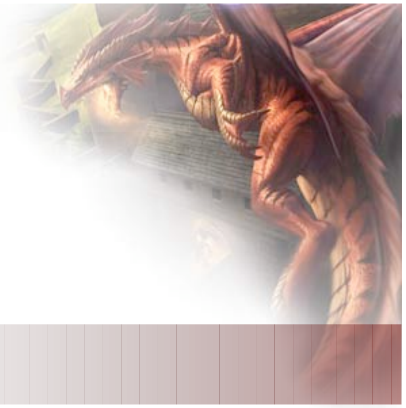
SLP to walk a maze

```
[ ] [ ] [ ] [ ] [ ]
►       [ ]     [ ]
[ ]     [ ]     [ ]        F R F F L F F F F
[ ]
[ ] [ ] [ ] [ ] [ ]
```



Branching Programs

- To adapt to uncertain environment need to have decision ability.
- Decision result causes a branch in the flow of control.

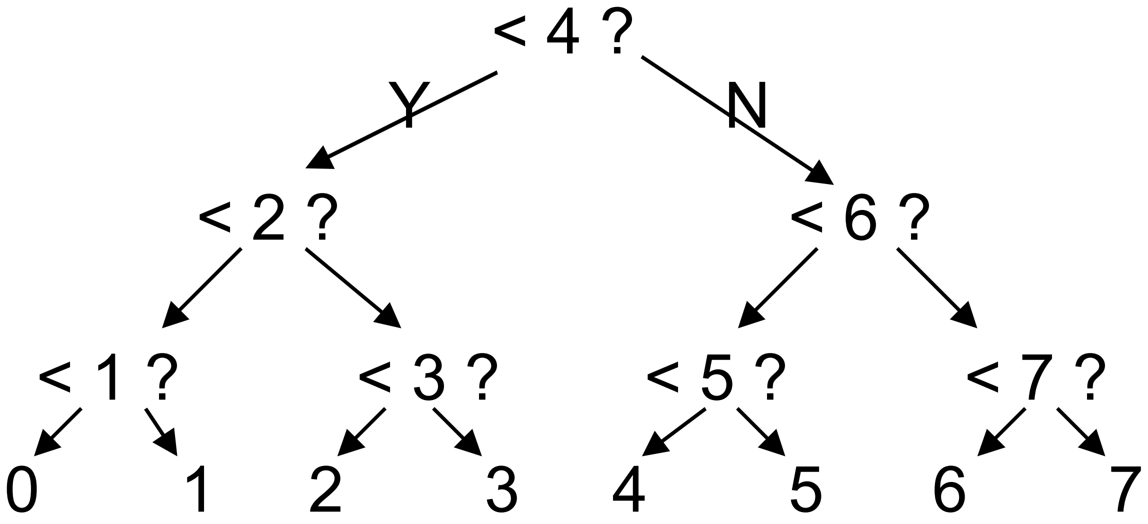


Decision Trees

- Decision trees are a common example of branching programs.
- Appear in many kinds of games and search problems.

Common Decision Tree

- Pick a number in range 0 .. 7



< 4 ?

Y

N

< 2 ?

< 6 ?

< 1 ?

< 3 ?

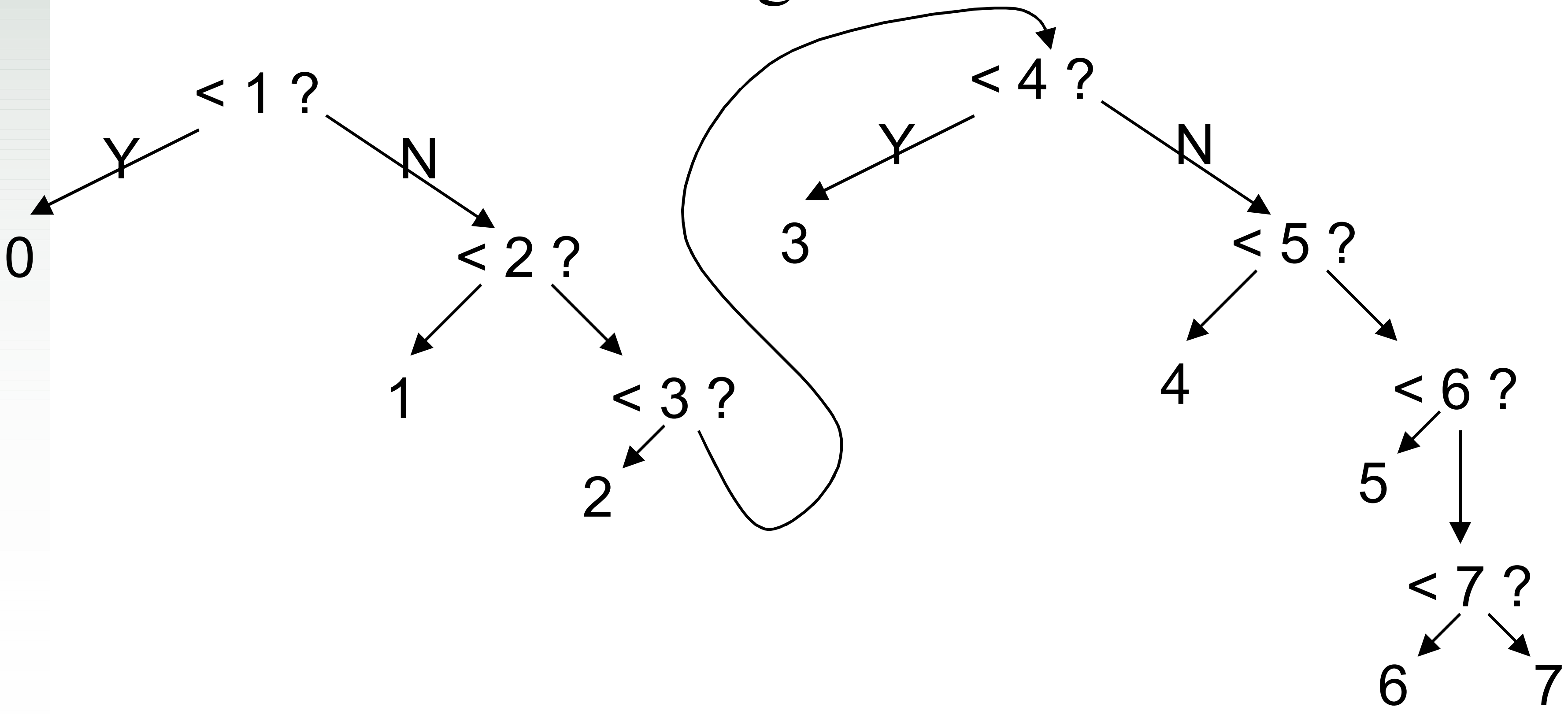
< 5 ?

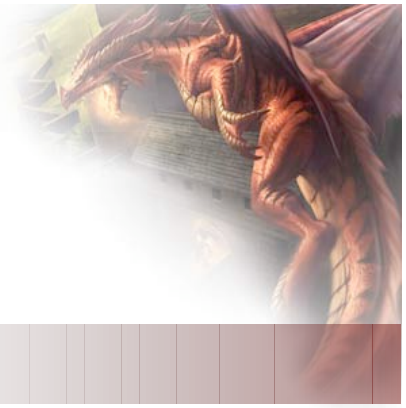
< 7 ?

0 1 2 3 4 5 6 7

Many possible designs …

- Pick a number in range 0 .. 7





So Cost is an Issue

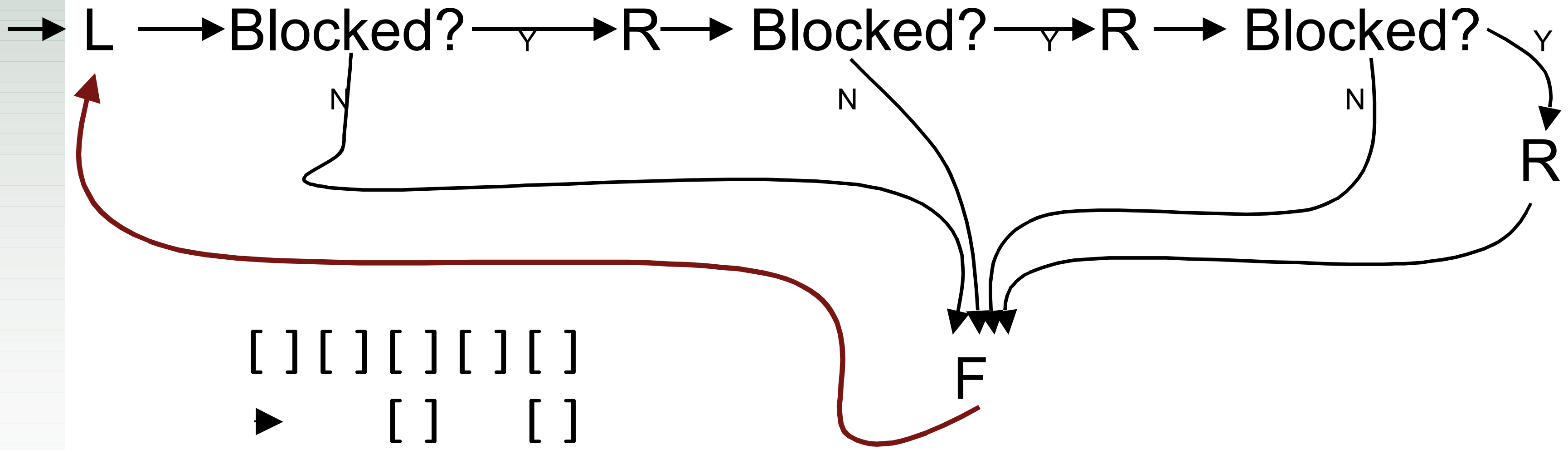
- How much time (e.g. number of steps, decisions)
- How much space (e.g. memory in RAM, on disk)
- How much programmer time?

Looping Programs

- Add decision ability to our robot
- Add Instructions:

  blocked? - which returns Y or N depending on whether can go forward or not.
- Allow branching back to previous point

Maze program again …



L → Blocked? → Y → R → Blocked? → Y → R → Blocked? → Y → R

N

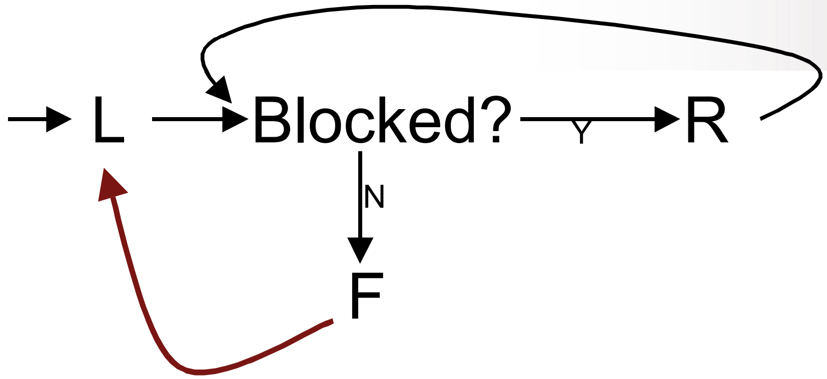
N

N

F

```
[ ][ ][ ][ ][ ]
►     [ ]   [ ]
[ ]   [ ]   [ ]
[ ]
[ ][ ][ ][ ][ ]
```

Simplified …



L

Blocked?

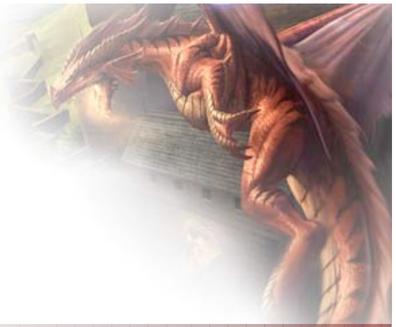
Y

R

N

F

```
[ ][ ][ ][ ][ ]
►     [ ]   [ ]
[ ]   [ ]   [ ]
[ ]
[ ][ ][ ][ ][ ]
```

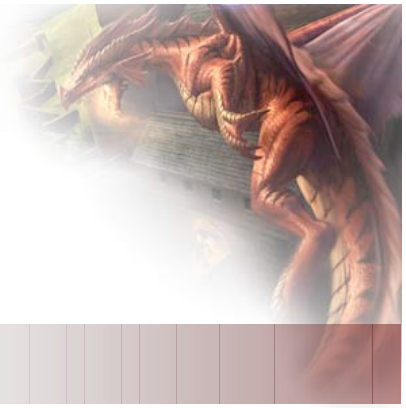


As programs

```
while ( in maze ) {
    L;
    if ( Blocked? ) {
        R;
        if ( Blocked? ) {
            R;
            if ( Blocked? ) {
                R;
                }
            }
        }
    F;
    }
```

```
while ( in maze ) {
    L;
    while ( Blocked? ) {
        R;
        }
    F;
    }
```

Are these equivalent?
I.e. do the same thing?



Key Ideas

- System - all the things that you are interested in. Eg. Maze + Robot
- State - all the dynamic information needed to reconstruct the system at a point in time. E.g. position and orientation of robot.
- If you stop a system at time t, record its state, and then continue you can backtrack back to time t. E.g. Save game.

- Transition - change of a system from one state at time t to another state at time t+1. Transitions are described by rules that say where the current state can go next.
- State space - all the potential states that a system can have. Some of them may never actually occur when a system runs.
- Execution - a sequence of transitions between states, usually starting in some initial state and ending in a final state.

- State Variable - variables capture different parts of the system. They break it into pieces to make it intellectually manageable.
- E.g. for robot in maze have 3 state variables:

orientation o: {N, S, E, W}

position (x,y) where

x: {0, 1, 2, 3, 4}

y: {0, 1, 2, 3, 4}

```
[ ][ ][ ][ ][ ]
 ►    [ ]   [ ]        o: E
[ ]   [ ]   [ ]        x: 0
[ ]                    y: 3
[ ][ ][ ][ ][ ]
```

```
[ ][ ][ ][ ][ ]
      [ ]   [ ]        o: S
[ ]   [ ] ▼ [ ]        x: 3
[ ]                    y: 2
[ ][ ][ ][ ][ ]
```

```
[ ][ ][ ][ ][ ]
      [ ]   [ ]        o: S
[ ]   [▼]   [ ]        x: 2
[ ]                    y: 2
[ ][ ][ ][ ][ ]
```

Not a legit state

♣How big is the state space?

♣For robot in maze have 3 state variables:

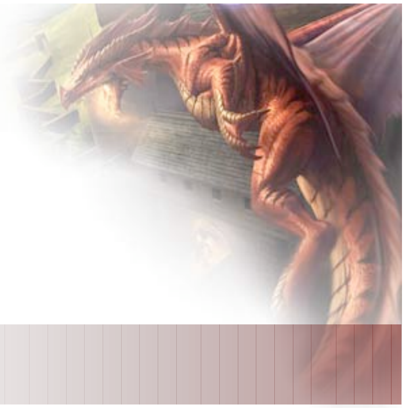
orientation o: {N, S, E, W}

position (x,y) where

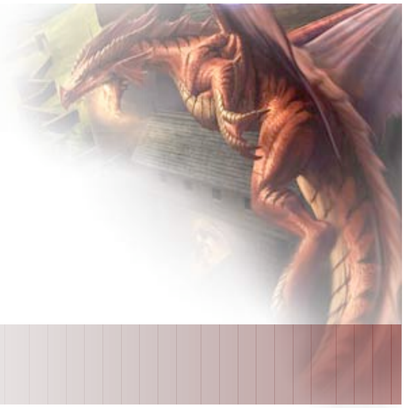
x: {0, 1, 2, 3, 4}

y: {0, 1, 2, 3, 4}

so 4 x 5 x 5 = 100 possible states. Which ones are legal depends on the maze.



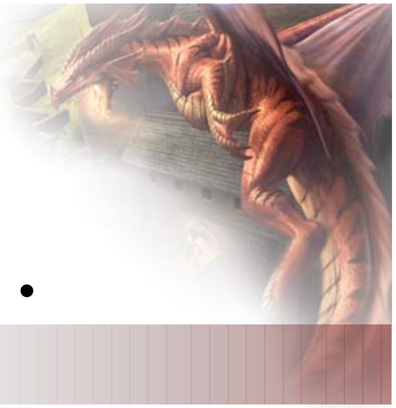
- Actually have a 4th state variable p, the position in the program giving the next instruction the robot is going to execute.



Managing State Space

- The art of programming is managing your state space.
- Total state space is huge (multiply the possible values of all variables).
- Program with just 1000 integers has 2 ** 32000, or about 10 ** 9600 states.

But size doesn’t matter …



- The key to keeping sane is making sure most actions only affect “local” state.
- The narrative guidelines are examples of this.
- Programming guidelines are similar. The main one is:

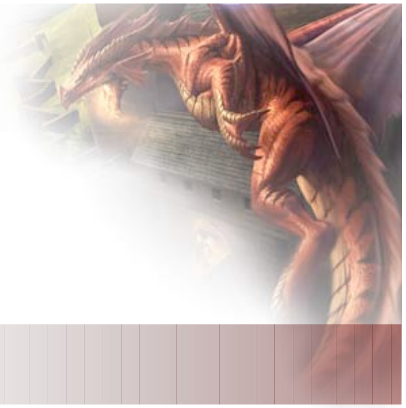
  Don’t talk to too many others.

- The other main one is:

  Just because you can doesn’t mean you should. Aka, Keep it Simple and Stupid.

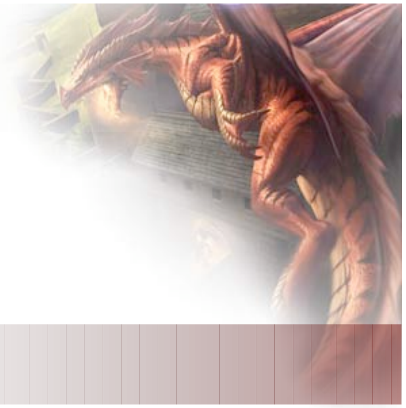
Programming Languages

- Languages are designed for specific purposes, and generally don’t do so well outside their design domain.
- Some are general purpose: Java, Perl
- Others are domain specific: ScriptEase, our toy robot



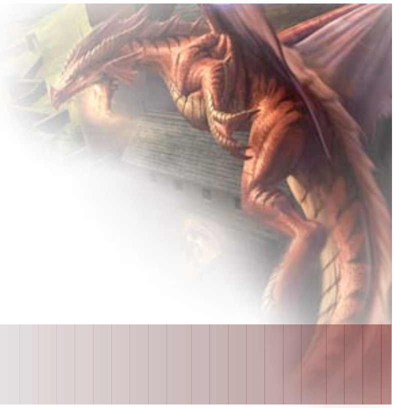
General Purpose

- Have to be able to do almost anything, so tend to be bad at most things.
- Expressing what you want to do is neither easy nor outrageously difficult



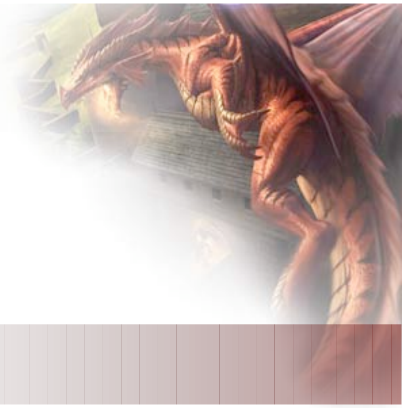
Special Purpose

- Have to be able to do a small number of things well.
- Expressing what you want to do is easy if you are are using it as intended, but often outrageously difficult if you are pushing the boundaries.



ScriptEase

- Special purpose, designed to cover most common activities in a RP game: encounters with other characters and objects.
- Built by looking at common coding patterns in the game engine codes and capturing these in the programming language.



ScriptEase

- ♣Don’t try to make it do what it is not intended to do.