Rule-based programming in Interactive Fiction

Or, How I Learned To Stop Worrying and Love them Declarative Languages*

*...except for Prolog. That’s still too weird.
Inform 7  (released April 30, 2006)

[A fragment of Inform 7 source code...]

Check entering the cell-doors:
   instead say “You are not here to search the cells.”

Instead of climbing something when the location is the Guardroom:
   say “(that is, go up using [the noun])”;
   try moving the Hallway with the noun.

Check moving the Hallway with generic when the location is the Guardroom:
   if the closest enemy is not generic:
      instead say “The opening is above your unaided reach.
      But the guard is your more immediate problem.”;
   else:
      prepare move-with-tutorial;
      instead say “The opening is above your unaided reach.
      You will need to make use of something.”
To stealthily survey the Temple of the Slumbering God, go to slide #4. To march up to the front gate, go to #5. To give up and leave, do neither.

---

### Adventure (Colossal Cave)
Crowther, 1976; FORTRAN

```
67     FORMAT(' THERE ARE ',I2,' THREATENING LITTLE DWARVES IN THE 
       1 ROOM WITH YOU.' ), /
      GOTO 77
75     CALL SPEAK(4)
77     IF(ATTACK.EQ.0)GOTO 71
77     IF(ATTACK.EQ.1)GOTO 79
77     PRINT 78,ATTACK
78     FORMAT(' ',I2,' OF THEM THROW 
           KNIVES AT YOU.' ), /
      GOTO 81
79     CALL SPEAK(5)
79     CALL SPEAK(52+STICK)
79     GOTO(71,83)(STICK+1)
81     IF(STICK.EQ.0) GOTO 69
81     IF(STICK.EQ.1)GOTO 82
81     PRINT 68,STICK
68     FORMAT(' ',I2,' OF THEM GET 
           YOU.' ), /
      GOTO 83
82     CALL SPEAK(6)
83     PAUSE 'GAMES OVER'
```

### Dungeon (Zork)
MIT, 1978; MDL

```
<DEFINE TRAP-DOOR ("AUX" (RM ,HERE))
#DECL ((PRSA) VERB 
   (RM) ROOM (DOOR) OBJECT)
<COND (<AND <VERB?
   "OPEN" "CLOSE">
   ==? .RM <SFIND-ROOM 
   "LROOM">>> 
   <OPEN-CLOSE <PRSO>
   "The door reluctantly opens to reveal 
    a rickety staircase descending 
    into darkness."
   "The door swings shut and closes.">)
   (==? .RM <SFIND-ROOM 
    "CELLA">) 
   <COND (<VERB? "OPEN">
   <TELL
   "The door is locked from above."
   (<TELL <PICK-ONE 
    ,DUMMY>>>)>>>)
```
20 years of IF development systems

- 1981 – DDL
- 1983 – The Quill
- 1985 – GAGS
- 1986 – ADVSYS
- 1987 – AGT
- 1992 – ALAN
- 1992 – TADS 2
- 1994 – Inform 5
- 1995 – Hugo
- 1996 – Inform 6
- 1997 – ADRIFT

To crawl into the muddy hypocaust you discover behind the Temple, go to slide #7. To circle around to the front gate, go to slide #5.
What’s in an IF game?

Living Room

Player

- lantern
- sword

Kitchen

- brown sack
- sandwich
- garlic
- knife

#5 To carefully inspect the hinges of the immense iron gate, go to slide #8. To pound your sword-hilt on the gate until someone opens it, go to #10.
What's in an IF game?

- Room
- Player
- Container
- etc...

To be cursed by the Slumbering God for the thousandth time, go to slide #32.
The object-oriented approach

class Object:
    string description: "You see nothing special."

Object Bottle:
    string description: "A glass bottle."
# The object-oriented approach

```python
class Object:
    function description(): {
        print "You see nothing special.";
    }

Object Bottle:
    function description(): {
        if (self.empty)
            print "An empty glass bottle.";
        else
            print "A bottle of", self.contents, ".";
    }
```

#8  To slip the catch which is so delicately hidden near the gate hinges, go to slide #12. To give up on the front gate and search the rear, go to #4.
The OO IF model in action

Object Flower:
method smell(): {
    print "Sweet as a rose.";
}

"SMELL FLOWER" -> (smell, Flower) -> call Flower.smell();
Where to put more complex conditions?

“LOOK” → Null.look()? Room.look?

“EXAMINE SWORD” (when room is dark) → Object.examine()? Darkness.examine()? 

“PUT MEAT IN BASKET” → Meat.put(Basket)? Basket.put(Meat)?

“TAKE PIE” (when angry orc is in room) → Orc.takeNearby(Pie)?

“PULL LEVER” (when lever is electrified) → Lever.anyTouching()?
Maybe if we put it all in one method...

```java
Player.do(action, object) {
    if (isDark() && action is Examine/Read) {
        print "Too dark."; return;
    }
    if (Orc.nearby && action is Take) {
        print "It stops you."; return;
    }
    switch (action) {
        case Look: ...
        case Take: ...
    }
}
```

...let's not.
How Inform 6 does it

function PerformAction() {
    if player.does_action(...) return;
    foreach O in room:
        if O.nearby_action(...) return;
    if room.contains_action(...) return;
    if object.action(...) return;
    action_default(...);
}

To silently ease open the hidden sally-portal that you have discovered by the gate, go to slide #16. To smash it down, go to slide #10.
What model describes how I really work?

- Code
  - action condition
- Code
  - object condition
- Code
  - game state condition
- Code
  - complex condition with several parameters

To be overwhelmed by the surprisingly well-trained guards, go to slide #14.
The rule: a general template

```
do ATOM as CODE if CONDITION
```
The rule: a general template

```
function do ATOM as CODE if CONDITION
```

```
function do FuncName as {...code...} if true
```
The rule: a general template

```
do ATOM as CODE if CONDITION

  do Description(obj) as "Nothing special."
    if true
  do Description(obj) as "Looks valuable."
    if (obj ofclass Treasure)
  do Description(obj) as "Huge and sparkly!"
    if (obj is HopeDiamond)
```
We could make `atom` just another parameter

```
do (atom, obj) as "Nothing special."
   if (atom is Description)

do (atom, obj) as "Looks valuable."
   if (atom is Description && obj ofclass Treasure)

do (atom, obj) as "Huge and sparkly!"
   if (atom is Description && obj is HopeDiamond)

...let's not.
```
More tricks with rules – constants

```
do Pi as 3.14159

do DarknessMessage as "It’s very dark."
```
More tricks with rules – methods

do Price(obj) as 0

do Price(obj) as 10
  if (obj ofclass Treasure)

do Price(obj) as {
  total = 0;
  foreach X in obj
    total += Price(X);
  return total;
} if (obj ofclass Container)
Assigning values at runtime

do Price(obj) as 10
   if (obj ofclass Treasure)

do Hit(MingVase) as {
   print "It cracks.";
   Price(MingVase) := 0;
}
Assigning values at runtime

```plaintext
do Price(obj) as 10
   if (obj ofclass Treasure)

do Price(obj) as 0
   if (...the assignment below has occurred...)

do Hit(MingVase) as {
   print "It cracks."
   Price(MingVase) := 0;
}
```

To take the key that the decrepit (but unpleasantly familiar) prisoner offers you, and slip out into the hallway, go to slide #22.
Assigning values at runtime

```plaintext
do Score as 0

do Slay(Dragon) as {
    print "The evil dragon is dead!";
    Score := Score + 10;
}

do Rescue(Princess) as {
    print "On the first date?";
    Score := Score + 20;
}
```
Defining classes in terms of rules

```plaintext
do Treasure(obj) as false
do Treasure(HopeDiamond) as true
do Treasure(MingVase) as true

doi Price(obj) as 0

do Price(obj) as 10
  if (Treasure(obj))
```
So why bother with any of this?

Or, let’s back down the garden path a little
Why rules?

- IF is made of exceptions.
- Handle complexity by being able to ignore most of it.
- A simple programming mechanism that can be made complicated later.
- You have to collaborate with the standard library (modify every part of it).
- Third-party libraries have to collaborate with the standard library, and with each other.
- You have to collaborate with yourself.
Resolving rule collisions is the hard part

Some wrong ways to handle it:

- Logical precedence (only works for the easy cases)
- Source code order (drives you insane)
- Ornate table for which ops take precedence (impossible to remember, and then usually wrong)
- Throw an error, require fixing by hand (billions and billions of errors)
Fixing a rule conflict

Rule-X: do Description(person) as {...}
Rule-Y: do Description(obj) as {...}
    if IsDark

Rule-Fix: do Description(person) as {...}
    if IsDark

...or...

Rule-Y ⊇ Rule-X
Resolving conflicts by groups of rules

- Darkness-Extension ⊇ Standard-Library
- Game-Code ⊇ Library-Code
- My-Scenery-Rule ⊇ Darkness-Extension
- Standard-Room-Rule ⊇ Game-Chapter-3
- Standard-Room-Rule ⊇ My-Grue-Rule

To navigate the hypocaust for what seems like the thousandth time, go to slide #20.
Handling groups of rules with more rules

do GameGroup(rule) as true
   if (...rule is defined in game code...)
do LibraryGroup(rule) as true
   if (...rule is defined in library code...)
do Override(GameGroup, LibraryGroup) as true

do Override(LibraryRule27, Chapter3Group) as true
Other difficult problems

- **Complementary properties**
  
  \[
  \text{Parent}(X,Y) \iff \text{Child}(Y,X) \iff \text{Nonempty}(X)
  \]

- **Multi-step operations**
  
  \[
  \{ \text{Perform}(\text{action}); \text{Report}(\text{action}) \}
  \]

- **Adding steps**
  
  do \text{Message} as \{ "Prologue"; \text{Message} \}
  
  do \text{Message} as \{ \text{Message}; "Epilogue" \}

- **Retaining information across rules**
You can’t demonstrate a complexity-management tool on toy problems

Or, I’m hosed
References

Dennis Jerz: Somewhere Nearby is Colossal Cave  
http://www.digitalhumanities.org/dhq/vol/001/2/000009.html

Stephen Granade: A Brief History of Interactive Fiction  
http://www.brasslantern.org/community/history/timeline-c.html

Graham Nelson: Natural Language, Semantic Analysis and Interactive Fiction  

The Interactive Fiction Wiki  
http://ifwiki.org/

Andrew Plotkin: ongoing and completely unorganized notes on rule-based programming  
(includes this presentation)  
http://www.eblong.com/zarf/rule-language.html