An Interactive Adventure Game Engine Built Using Pyparsing

Author: Paul McGuire <ptmcg@austin.rr.com>

Version: 1.1

Agenda

- Two Kinds of Parsing Applications
- Pyparsing Grammar Definition
- Merge Command Interpreter with Parser
- Game Engine Grammar
- Pyparsing BNF Definition
- Finishing Up the Game
  - Define Rooms
  - Define Items
  - Put Items in Rooms
- Running the Game
- Sample Session
- Finishing Touches

Two Kinds of Parsing Applications

Design-driven:

```
language -> BNF -> parser impl --+->
 concept     ^                   |
             |   refine/extend   |
             +---  language    --+
```

Data-driven:

```
gather    determine
sample ->   text    -> BNF -> parser ---+->
inputs     patterns                     |
             ^         gather new       |
             +----- (non-conforming) ---+
                         inputs
```

Pyparsing Grammar Definition

At startup, user program defines the pyparsing grammar, by building up expressions that comprise the supported application grammar

Basic building blocks are Literals and Words

Basics are combined using And, Or, MatchFirst operations

- pyparsing defines operators '+', '^', '|' to simplify this step, and make the code more readable

oneOf("red green blue") is a short-cut for:

```
Literal("red") | Literal("green") | Literal("blue")
```

Pyparsing "Hello World"

Compose grammar:

```
greeting = oneOf("Hello Ahoy Yo Hi") +
           Literal(",") +
           Word( string.uppercase, string.lowercase ) +
           Literal("!")
```

Call parseString():

```
print greeting.parseString("Hello, World!")
print greeting.parseString("Ahoy, Matey !")
print greeting.parseString("Yo,Adrian!")
['Hello', ',', 'World', '!']
['Ahoy', ',', 'Matey', '!']
['Yo', ',', 'Adrian', '!']
```

Pyparsing Grammar Definition (2)

Pyparsing grammar elements can be given results names

At runtime, the matched tokens are assigned these results names

The individual tokens can be retrieved by name from the parsed results:

```
greeting = oneOf("Hello Ahoy Yo Hi").setResultsName("greeting") +
           Literal(",") +
           Word( string.uppercase,
                 string.lowercase ).setResultsName("subject") +
           Literal("!")

results = greeting.parseString("Yo,Adrian!")
print results.greeting   # prints 'Yo'
print results.subject    # prints 'Adrian'
```

We'll use this feature to access command qualifiers

Pyparsing Grammar Definition (3)

Pyparsing grammar elements can keep references to callbacks or "parse actions", executed when a particular grammar element is parsed from the input string

Parse actions are called with 3 arguments:

- the complete input string
- the starting location of the matching text
- the list of tokens found in the input text matching the expression

A parse action can modify the matched tokens by returning a value:

- returned value replaces the matched tokens in the parsed results data
- returned value does not have to be a string
- if no value is returned, tokens are not modified

Merge Command Interpreter with Parser

Define grammar expression for each command

Create command processor class, inherit from base Command class

Attach command processor to grammar expression, using parse action

Compose command parser from Or of all grammar expressions

At runtime:

```
while not game over:
   prompt for command string
   parse command string, Command subclass is returned
   perform Command
```

Game Engine Grammar

Commands:

```
- INVENTORY or INV or I - lists what items you have
- DROP or LEAVE <objectname> - drop an object
- TAKE or PICKUP or PICK UP <objectname> - pick up an object
- USE or U <objectname> [ IN or ON <objectname> ] - use an
  object, optionally IN or ON another object
- OPEN or O <objectname> - open an object
- MOVE or GO - go NORTH, SOUTH, EAST, or WEST
  (can abbreviate as 'GO N' and 'GO W', or even just 'E' and 'S')
- LOOK or L - describes the current room and any objects in it
- DOORS - display what doors are visible from this room
- QUIT or Q - ends the game
- HELP or H or ? - displays this help message
```

Pyparsing BNF Definition

Define verbs for each command:

```
invVerb = oneOf("INV INVENTORY I", caseless=True)
dropVerb = oneOf("DROP LEAVE", caseless=True)
takeVerb = oneOf("TAKE PICKUP", caseless=True) | \
    (CaselessLiteral("PICK") + CaselessLiteral("UP") )
moveVerb = oneOf("MOVE GO", caseless=True) | empty
useVerb = oneOf("USE U", caseless=True)
openVerb = oneOf("OPEN O", caseless=True)
quitVerb = oneOf("QUIT Q", caseless=True)
lookVerb = oneOf("LOOK L", caseless=True)
doorsVerb = CaselessLiteral("DOORS")
helpVerb = oneOf("H HELP ?",caseless=True)
```

Pyparsing BNF Definition (2)

```
itemRef = OneOrMore(Word(alphas)).setParseAction( self.validateItemName )

nDir = oneOf("N NORTH",caseless=True).setParseAction(replaceWith("N"))
sDir = oneOf("S SOUTH",caseless=True).setParseAction(replaceWith("S"))
eDir = oneOf("E EAST",caseless=True).setParseAction(replaceWith("E"))
wDir = oneOf("W WEST",caseless=True).setParseAction(replaceWith("W"))
moveDirection = nDir | sDir | eDir | wDir

invCommand = invVerb
dropCommand = dropVerb + itemRef.setResultsName("item")
takeCommand = takeVerb + itemRef.setResultsName("item")
useCommand = useVerb + itemRef.setResultsName("usedObj") + \
    Optional(oneOf("IN ON",caseless=True)) + \
    Optional(itemRef,default=None).setResultsName("targetObj")
openCommand = openVerb + itemRef.setResultsName("item")
moveCommand = moveVerb + moveDirection.setResultsName("direction")
quitCommand = quitVerb
lookCommand = lookVerb
doorsCommand = doorsVerb
helpCommand = helpVerb
```

Pyparsing BNF Definition (3)

Base Command class:

```
class Command(object):
    "Base class for commands"
    def __init__(self, verb, verbProg):
        self.verb = verb
        self.verbProg = verbProg

    @staticmethod
    def helpDescription():
        return ""

    def _doCommand(self, player):
        pass

    def __call__(self, player ):
        print self.verbProg.capitalize()+"..."
        self._doCommand(player)
```

Pyparsing BNF Definition (4)

```
    class InventoryCommand(Command):
        def __init__(self, quals):
            super(InventoryCommand,self).__init__("INV", "taking inventory")

        @staticmethod
        def helpDescription():
            return "INVENTORY or INV or I - lists what items you have"

        def _doCommand(self, player):
            print "You have %s." % enumerateItems( player.inv )

    def makeCommandParseAction( self, cls ):
        def cmdParseAction(s,l,t):
            return cls(t)
        return cmdParseAction

    invCommand.setParseAction( self.makeCommandParseAction( InventoryCommand ) )
    dropCommand.setParseAction( self.makeCommandParseAction( DropCommand ) )
    takeCommand.setParseAction( self.makeCommandParseAction( TakeCommand ) )
    ... etc. ...
```

Pyparsing BNF Definition (5)

```
    class TakeCommand(Command):
        def __init__(self, quals):
            super(TakeCommand,self).__init__("TAKE", "taking")
            self.subject = quals["item"]

        @staticmethod
        def helpDescription():
            return "TAKE or PICKUP or PICK UP - pick up an object (but some are deadly)"

        def _doCommand(self, player):
            rm = player.room
            subj = Item.items[self.subject]
            if subj in rm.inv and subj.isVisible:
                if subj.isTakeable:
                    rm.removeItem(subj)
                    player.take(subj)
                else:
                    print "You can't take that!"
            else:
                print "There is no %s here." % subj
```

Pyparsing BNF Definition (6)

Complete game grammar - Or of all defined commands:

```
return ( invCommand   |
         useCommand   |
         openCommand  |
         dropCommand  |
         takeCommand  |
         moveCommand  |
         lookCommand  |
         doorsCommand |
         helpCommand  |
         quitCommand ).setResultsName("command") + LineEnd()
```

Define Rooms

```
roomMap = """     # define global variables for referencing rooms
     d-Z          frontPorch = rooms["A"]
     |            garden     = rooms["b"]
   f-c-e          kitchen    = rooms["c"]
   . |            backPorch  = rooms["d"]
   q<b            library    = rooms["e"]
     |            patio      = rooms["f"]
     A
"""

rooms = createRooms( roomMap )
rooms["A"].desc = "You are standing at the front door."
rooms["b"].desc = "You are in a garden."
rooms["c"].desc = "You are in a kitchen."
rooms["d"].desc = "You are on the back porch."
rooms["e"].desc = "You are in a library."
rooms["f"].desc = "You are on the patio."
rooms["q"].desc = "You are sinking in quicksand.  You're dead..."
rooms["q"].gameOver = True
```

Define Items

```
# create items
itemNames = """sword.diamond.apple.flower.coin.n
               shovel.book.mirror.telescope.gold bar""".split(".")
for itemName in itemNames:
    # Item ctor also updates class dict of all items by name
    Item( itemName )

Item.items["apple"].isDeadly = True
Item.items["mirror"].isFragile = True
Item.items["coin"].isVisible = False

Item.items["shovel"].usableConditionTest = ( lambda p,t: p.room is garden )
def useShovel(p,subj,target):
    coin = Item.items["coin"]
    if not coin.isVisible and coin in p.room.inv:
        coin.isVisible = True
Item.items["shovel"].useAction = useShovel

OpenableItem("treasure chest", Item.items["gold bar"])
```

Put Items in Rooms

```
putItemInRoom("shovel", frontPorch)
putItemInRoom("coin", garden)
putItemInRoom("flower", garden)
putItemInRoom("apple", library)
putItemInRoom("mirror", library)
putItemInRoom("telescope", library)
putItemInRoom("book", kitchen)
putItemInRoom("diamond", backPorch)
putItemInRoom("treasure chest", patio)
```

Running the Game

```
def playGame(p,startRoom):
    # create parser
    parser = Parser()
    p.moveTo( startRoom )
    while not p.gameOver:
        cmdstr = raw_input(">> ")
        cmd = parser.parseCmd(cmdstr)
        if cmd is not None:
            cmd.command( p )
    print
    print "You ended the game with:"
    for i in p.inv:
        print " -", aOrAn(i), i

# create player
plyr = Player("Bob")
plyr.take( Item.items["sword"] )

# start game
playGame( plyr, frontPorch )
```

Sample Session

```
You are standing at the front door.
There is a shovel here.
>> take shovel
Taking...
>> n
Moving...
You are in a garden.
There is a flower here.
>> use fhovel
No such item 'fhovel'.
>> use shovel
Using...
>> l
Looking...
You are in a garden.
There are a coin and a flower here.
>> take flower
Taking...
```

```
>> take coin
Taking...
```

Sample Session

```
>> i
Taking inventory...
You have a sword, a shovel, a flower, and a coin.
>> doors
Looking for doors...
There are doors to the north, south, and west.
>> w
Moving...
You are sinking in quicksand.  You're dead...
There is nothing here.
Game over!

You ended the game with:
 - a sword
 - a shovel
 - a flower
 - a coin
```

Finishing Touches

Save game / load game

Fold globals into Game class instance

Mapping

Documentation

Current Code