## expressions:

number: 5, -10.3
boolean: true/false
nested expressions: (expression)
tracery rules: “hello, #/playername#”
function: sin(x), randomInt(5, 10), myCustomFunc(x, y)
path variables: [/foo] [/foo/5/weapon{[/weaponclass]}]
value operations: expression + expression (e.g. + - * / ^ %)
boolean operations: expression == expression (e.g. == != < <=...)
listComprehension([x/xp]) where [x/lvl]>3 for [/characters]

## conditions:

expression booleanOperator expression
    “x==5”  (!=, --, <=, >=, <, >)
tracery rule (for regex matching)
    “#yes# i want #animal#” (saves yes and animal)
    (values saved: INPUT, INPUT_SOURCE, MATCH_0, MATCH_1..)
“wait:expression”
function: hasTag(x, tag)

## actions:

expression actionOperator expression
    “x=5” (for (=, +=, -=, /=, *=, ^=, %=)
tracery rule
    “hello”, “hello, #animal#”, “hello #/playername#”
    send the expanded rule as a msg (chat)
“wait:expression”
function: doComplexThingie(x, tag)

**Map**

```
{
   id: “myID”,
   title: “What a catchy title”,
   states: {
      origin: {
         tags: [”start”, “meta”, “undersea”],
         onEnterSay: “You need an origin”,
         onEnter: “’you can do actions’ x=5”,
         exits: [”wait:5 ->next ’go to the state \‘next’\’’”,
              “x>5 ->@ ‘go back to this state’”,
              “x<2 ->^ ‘go to the previous state’”,
              “x==3 ->.three ‘go to any state with the tag \‘three\’’”,
              “x==4 ->^.three ‘go the most recent state
                        with the tag \‘three\’’”,
              ]

         onExitSay: “Thats enough for now”
      },
      next: {
           onEnterSay: “dead end”
      }
   }
}
```