

# Enigma Level API II

## task driven approach

---

### Position Tasks

#### Creating Positions

`pos = po(7, 3)` "po()" to generate a position  
`pos = po({7, 3})` using a table as argument  
`pos = obj` every object is a valid position  
`pos = po(12.3, 3.7)` a position within a grid (for an actor)

#### Position Constants

`{7,3}` valid position for all arguments and operations

#### Coordinate Access

`x,y = pos.x, pos.y` member access  
`x,y = pos["x"], pos["y"]` member access  
`x,y = pos:xy()`  
`x,y = obj.x, obj.y` works on objects too  
`x,y = obj:xy()`

#### Position Calculation

`pos = obj + {2,7}` adding offset  
`dpos = obj1 - obj2` difference vector  
`dpos2 = 2 * dpos` scalar multiplication  
`dpos3 = dpos / 2`  
`dpos3 = (obj1 - obj2) / 2` middle between objects

#### Center positions for set actors

`pos_centered1 = pos + {0.5, 0.5}` by offset  
`pos_centered2 = #pos` by special feature  
`pos_centered3 = #obj`

#### Round a position to a grid

`grid_pos = pos:grid()` to integer coordinates  
`grid_pos = ((pos1 - pos2)/2):grid()`

#### Position comparison

`pos_centered1 == pos_centered2` Lua's equality operator  
`pos_centered1 ~= pos_centered2` Lua's inequality operator

#### Position existence

`pos:exists()`

---

### Attribute Tasks

#### Single Attribute Setting

`obj["destination"] = po(7,3)` simple object attribute  
`wo["Brittleness"] = 7.0` global world attribute  
`obj["myattribute"] = "what"` userattribute

#### Multiple Attribute Setting

`obj:set({target=mydoor, action="open"})` set multiple attributes

### Requesting Attributes

`value = obj["attr_name"]` get the value  
`value = obj.attr_name`  
`value = wo["Brittleness"]` get the value of a global level constant  
`if wo["IsDifficult"] then ... end` often used difficult-mode switch

#### Reset Attributes

`obj["length"] = nil` the default length, e.g. '1'  
`obj["color"] = nil` delete color attribute - no color  
`obj["length"] = DEFAULT` the default length, e.g. '1'

---

### Object Tasks

#### Creating Objects

`wo[pos] = {"st_chess", color=WHITE, name="Atrax"}` on grid pos  
`wo[#pos] = {"ac_bug"}` actor centered on grid pos  
`wo[pos] = {"#ac_bug"}` actor centered on grid pos  
`wo[pos] = {"ac_bug", 0.3, 0.7}` actor with offsets to pos  
`wo[my_floor] = {"it_magicwand"}` set an wand on top of a given floor obj  
`wo[pos] = ti["x"]` tile based object definition

#### Object Naming

`no["Atrax"] = obj`  
`wo[pos] = {"st_chess", name="Atrax"}`  
`wo[pos] = {"st_chess", "Atrax", color=WHITE }`

#### Object Autonoming

Each new object will have a unique name.  
`wo[pos] = {"st_chess", name="Atrax#"}` autonamed chesstones

#### Requesting Objects

`obj = no["Atrax"]` named object retrieval from repository  
`obj = fl(pos)` floor at pos  
`obj = it(x,y)` item at pos  
`obj = st(pos)` stone at pos  
`obj = wo:it(pos)` item at pos  
`my_item = it(my_floor)` get the item that is on top of the given floor

#### Killing Objects

`wo[pos] = {"it_nil"}`  
`obj:kill()` be carefull with kill

#### Comparing Objects

`obj1 == obj2`  
`obj1 ~= obj2`

#### Existence of an object

`obj:exists()` object exists?  
`-obj` unary minus operator on object  
`if -obj then ...`

#### Messages

`my_boulder:message("orientate", WEST)`  
`my_boulder:orientate(EAST)`  
`my_door:open()`

#### Object Classification

`obj:is("st_chess")`  
`obj:is("st")`  
`obj:is("st_chess_black")`

---

## Group Tasks

### Creating Groups

```
group = no["Atrax#*"]      a group of all matching objects, wildcards "*" ,"?" allowed
group = grp(obj1, obj2, obj3)  a group of several objects
group = grp({obj1, obj2, obj3}) a group of objects set up in a table
group = grp()                an empty group
```

### Group Usage

```
floor_group["friction"] = 3.2  set attribute on all floors in the group
door_group:message("open")     send message to all members
door_group:open()              open all doors in the group
stone_group:kill()
wo[floor_group] = {"it_coin_m"} add some money on all floor positions
wo[pos] = {"st_switch", target=door_group, action="open"}  multitargets
wo[pos] = {"st_switch", target="door#*", action="close"}
```

### Group Operations

```
doors_lasers = doorgrp + lasergrp  join of two groups
lasergrp      = doors_lasers - doorgrp  difference of two groups
common_doors = doorgrp1 * doorgrp2    intersection of two groups
```

### Group Members

```
count = #mygroup          - number of objects in the group
obj     = mygroup[5]       - 5th object of the group
obj     = mygroup[-1]      - last object of the group
for i = 1, #mygroup do obj = mygroup[i] ... end
for obj in mygroup do ... end
```

### Shuffled Group

```
shuffled_group = sorted_group:shuffle()
shuffled_group = no["Atrax#*"]:shuffle()
```

### Sorted Group

```
sorted_group = group:sort("linear", po(2, 1))
sorted_group = group:sort("linear")
sorted_group = group:sort("circular")
sorted_group = group:sort()
```

### Subset Group

```
sub_group = group:sub(2)      - first two objects
sub_group = group:sub(-2)     - last two objects
sub_group = group:sub(2, 4)   - objects from 2 to 4
sub_group = group:sub(2, -2)  - two objects starting with 2
```

### Nearest Object

```
object = group:nearest(reference)
```

---

## Tiles

### Tiles

```
ti["_"] = {"fl_sahara"}      simple tile
ti["_ "] = {"fl_sahara"}    two char tile
ti[".."] = {"fl_sand"}
ti["###"] = {"st_blocker"}
ti["switch_template"] = {"st_switch"}  tiles can hav arbitrary names too
ti[".."] = {"fl_abys"}      redefinition causes error
ti[" w"] = ti[".."] .. {"it_magicwand"} concatenation of several tiles possible
ti[" w"] = {"fl_abys"} .. ti({"it_magicwand"})
```

---

## Named Positions Tasks

### Named Position Usage

```
obj["name"] = "anchor1"
obj:kill()
pos = po["anchor1"]          position still available
po["anchor2"] = pos
```

### Creating Position Lists

```
polist = po["deepwater#*"]  positionlist with pos
polist = po(grp)             of all group objects
```

### Position List Usage

```
wo[polist] = ti["x"]
grp = fl(polist)
```

### Position List Operations

```
wo[polist .. po["beach#*"]] = {"it_banana"}
```

### Position List Members

```
for i = 1, #pogrp do          iterate over polist
  wo[polist[i]] = {"it_cherry"}
end
```

---

## Other

### Nearest Object

```
ti["F"] = {"st_floppy", target="@door#*"}  target is always the nearest door
ti["B"] = {"st_blocker", name="door#"}     resolved at levelloadtime
ti["o"] = {"#ac_pearl_white", "s#", owner=DEFAULT}  target is always the currently nearest actor
ti["q"] = {"it_rubberband", anchor2="@s#*"}     resolved at runtime when needed
```

### Callbacks from switchlike objects

```
function my_callback(value, sender) ... end Sender is the senderobject, value it's state.
```

### Checkerboard floor

```
ti["x"] = ti({"fl_rough_red", checkerboard=0}) .. {"fl_rough_blue", checkerboard=1}
```

## World

### World Initialization

```
width, height = wo(topresolver, defaultkey, map)
width, height = wo(topresolver, defaultkey, width, height)
```

### World Advanced Methods

```
wo:add(tile_declarations) wo:add(target, tile_declarations)
wo:drawBorder(upperleft_edge, lowerright_edge, tile)
wo:drawBorder(upperleft_edge, width, height, tile)
wo:drawMap(resolver, anchor, ignore, map, [readdir])
wo:drawMap(resolver, anchor, libmap-map, [readdir])
wo:drawRect(upperleft_edge, lowerright_edge, tile)
wo:drawRect(upperleft_edge, width, height, tile)
wo:shuffle0xyd(rules)
wo:shuffle0xyd() wo:shuffle0xyd({no["borderoxyds#*"]:sort("circular"), circular=true})
wo:shuffle0xyd({"leftoxyds#*", "rightoxyds#*", min=3, max=5})
```

---

## Resolvers

### Autotiling

```
res.autotile(subresolver, rules)
res.autotile(ti, {"A", "template_switch"}, {"L", "template_laser"})
res.autotile(ti, {"a", "e", "template_trigger"}, {"A", "E", "template_door"})
```

### Composer

```
res.composer(subresolver) res.composer(subresolver, sequence)
res.composer(ti)
res.composer(ti, "211")    decompose the last two chars together
```

### Puzzler

load the library before use: `<el:dependency el:path="lib/libpuzzle" el:id="lib/libpuzzle" el:release="3" el:preload="true"/>`

```
res.puzzle(subresolver, rules)
res.puzzle(ti, "B", "Y", "I", "M")    Don't forget appr. tile declarations
```

### Random

```
res.random(subresolver, hits, replacements) res.random(ti, "x", {"a", "b"})
res.random(ti, {"x", "y"}, {"i", "j"}, {"a", 2}, {"b", 1})
```

### Custom Resolver

```
tile = myresolver(key, x, y)
```