Mudra

A Declarative Approach for Gesture Programming

"a scheme+sql for gestures"

“say hello to each person”

```
(defrule hello_world
  (Person (name ?name))
 =>
  (printout t “Hello “ ?name))
```

“say hello to each person”

```
(defrule hello_world
  (Person (name ?name))
 =>
  (printout t “Hello “ ?name))
```

conditional element

action

“say hello to each person”

```
(defrule hello_world
  (Person (name ?name))
  (test (= ?name “Bruno”))
 =>
(printout t “Hello “ ?name))
```

conditional element

Test function

action

“say hello to each person”

For each “Person” in the fact base

```
(defrule hello_world
  (Person (name ?name))
 =>
  (printout t “Hello “ ?name))
```

“say hello to each person”

unbound variable

```
(defrule hello_world
  (Person (name ?name))
 =>
  (printout t “Hello “ ?name))
```

id	name
0	Lode
1	Bruno

"say hello to each person that said hello to me"

Person

id	name
0	Lode
1	Bruno

Speech

userid	word
1	hello
1	fun

“say hello to each person that said hello to me”

Person

id	name
0	Lode
1	Bruno

Speech

userid	word
1	hello
1	fun

```
(defrule hello_world
  (Person (id ?id) (name ?name))
  (Speech (userid ?id) (word “hello”))
 =>
  (printout t “Hello “ ?name))
```

“say hello to each person that said hello to me”

Person

id	name
0	Lode
1	Bruno

Speech

userid	word
1	hello
1	fun

literal constraint

```
(defrule hello_world
  (Person (id ?id) (name ?name))
  (Speech (userid ?id) (word “hello”))
 =>
  (printout t “Hello “ ?name))
```

“say hello to each person that said hello to me”

Person

id	name
0	Lode
1	Bruno

Speech

userid	word
1	hello
1	fun

unification

```
(defrule hello_world
  (Person (id ?id) (name ?name))
  (Speech (userid ?id) (word “hello”))
 =>
  (printout t “Hello “ ?name))
```

"... and record my answer in the db"

```
(deftemplate (Answer (slot userid) (slot word)))
```

Answer

userid	word

“... and record my answer in the db”

```
(deftemplate (Answer (slot userid) (slot word)))
```

```
(defrule hello_world
  (Person (id ?id) (name ?name))
  (Speech (userid ?id) (word “hello”))
 =>
  (printout t “Hello” ?name crlf)
  (assert (Answer (userid ?id) (word “Hello “))))
```

“... and record my answer in the db”

```
(deftemplate (Answer (slot userid) (slot word)))
```

Person

id	name
0	Lode
1	Bruno

Speech

userid	word
1	hello
1	fun

Answer

userid	word
1	hello

```
(defrule hello_world
  (Person (id ?id) (name ?name))
  (Speech (userid ?id) (word “hello”))
 =>
  (printout t “Hello” ?name crlf)
  (assert (Answer (userid ?id) (word “Hello “))))
```

Let’s apply this to gestures...

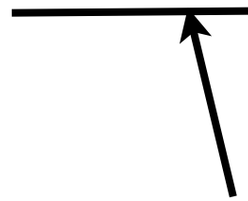
Gesture 1: Move right

Gesture 1: Move right

"Find a combination of points where the first is left of the second point"

Gesture 1: Move right

"Find a combination of points where the first is left of the second point"



x1 < x2

Gesture 1: Move right

"Find a combination of points where the first is left of the second point"

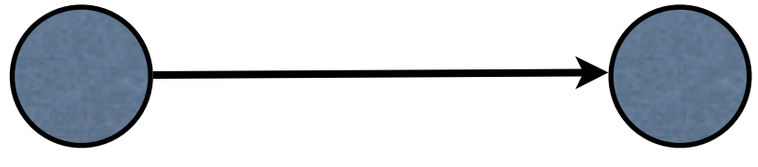
t1 < t2

Gesture 1: Move right

```
(defrule move_right
   ...


=>
   (printout t “Moved right!”))
```

Gesture 1: Move right

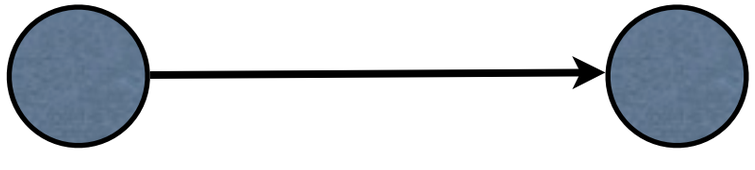


```
(defrule move_right
  (Tuio2Dcur (x ?x1) (time ?t1))
  ....


=>
  (printout t “Moved right!”))
```

```
(< ?t1 ?t2)
(< ?x1 ?x2)
```

Gesture 1: Move right

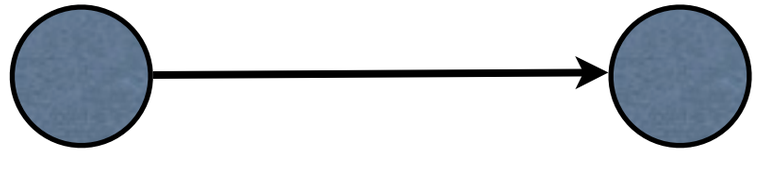


```
(defrule move_right
  (Tuio2Dcur (x ?x1) (time ?t1))
  (Tuio2Dcur (x ?x2) (time ?t2))
  (test (< ?t1 ?t2))

=>
  (printout t “Moved right!”))
```

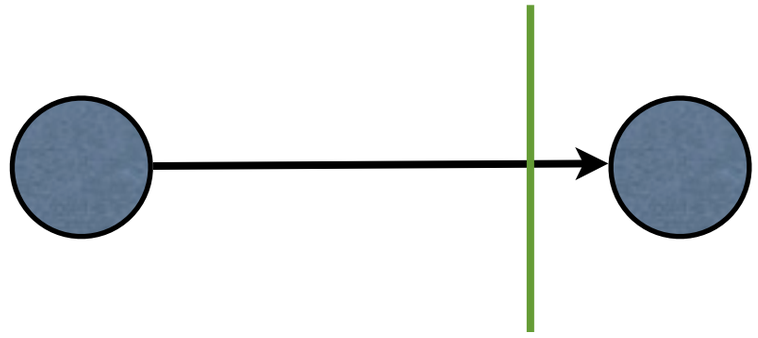
(< ?x1 ?x2)

Gesture 1: Move right

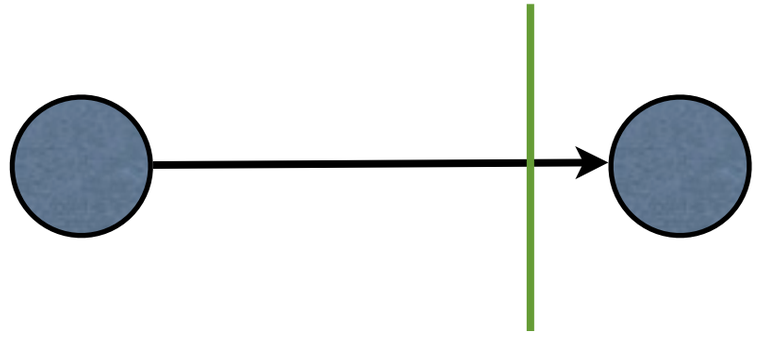


```
(defrule move_right
  (Tuio2Dcur (x ?x1) (time ?t1))
  (Tuio2Dcur (x ?x2) (time ?t2))
  (test (< ?t1 ?t2))
  (test (< ?x1 ?x2))
=>
  (printout t “Moved right!”))
```

Gesture 1: Move right

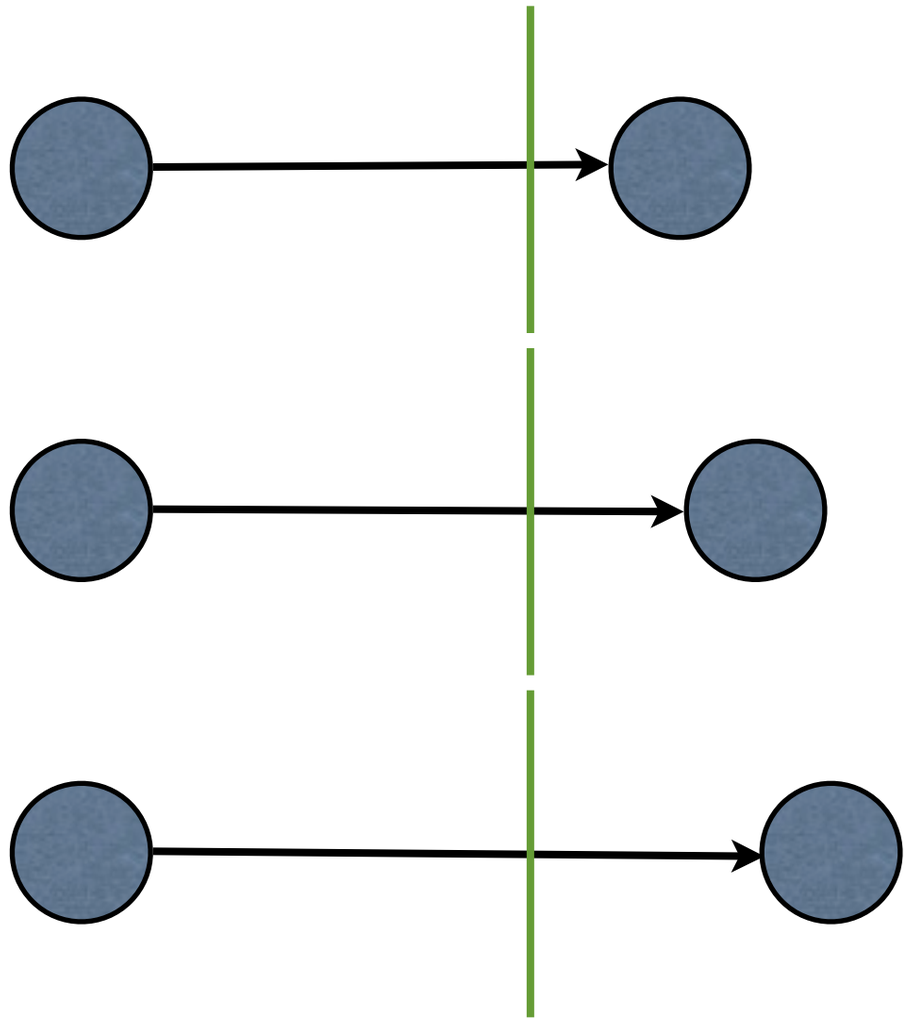


Gesture 1: Move right



```
(defrule move_right
  (Tuio2Dcur (x ?x1) (time ?t1))
  (Tuio2Dcur (x ?x2) (time ?t2))
  (test (< ?t1 ?t2))
  (test (> ?x2 (+ ?x1 0.2)))
=>
  (printout t “Moved right!”))
```

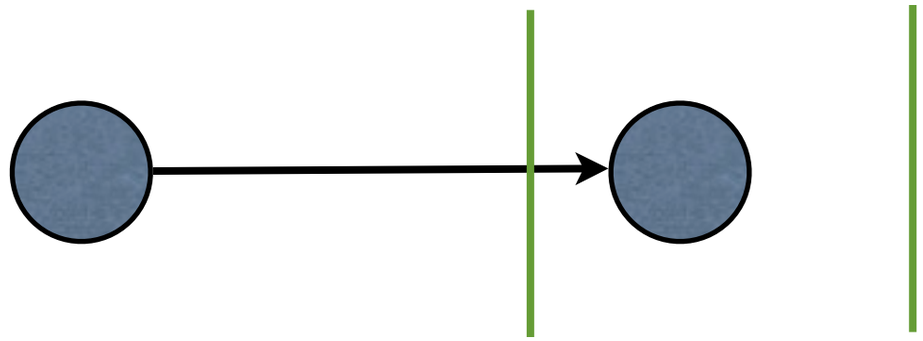
Gesture 1: Move right



Multiple triggers

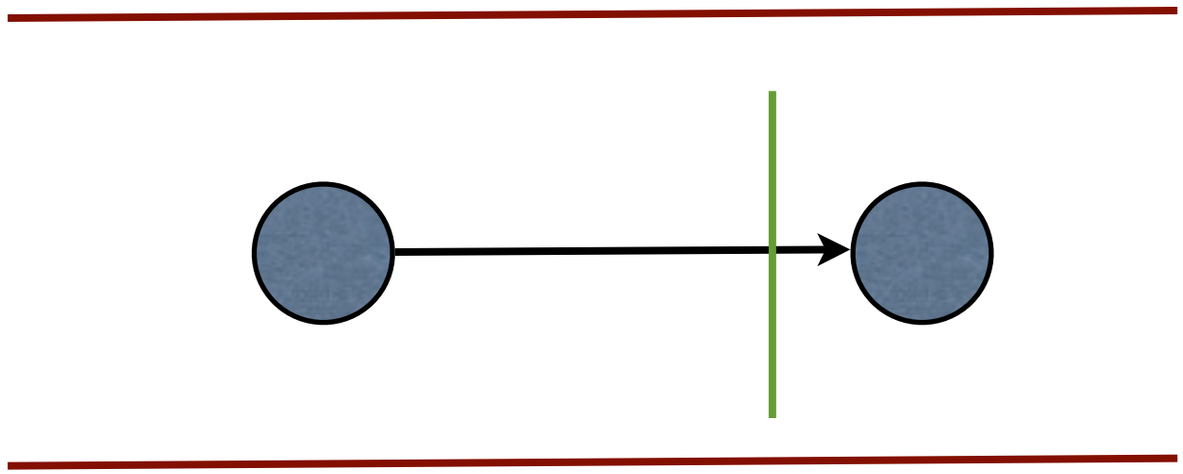
- Exhaustive search
- Fits your definition

Gesture 1: Move right

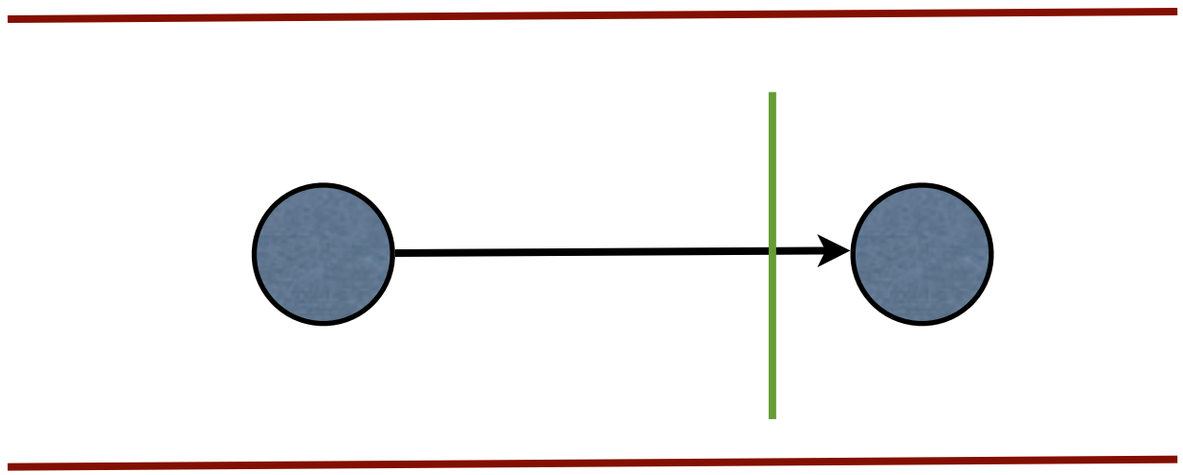


```
(defrule move_right
  (Tuio2Dcur (x ?x1) (time ?t1))
  (Tuio2Dcur (x ?x2) (time ?t2))
  (test (< ?t1 ?t2))
  (test (> ?x2 (+ ?x1 0.2)))
  (test (< ?x2 (+ ?x1 0.3)))
=>
  (printout t “Moved right!”))
```

Gesture 1: Move right



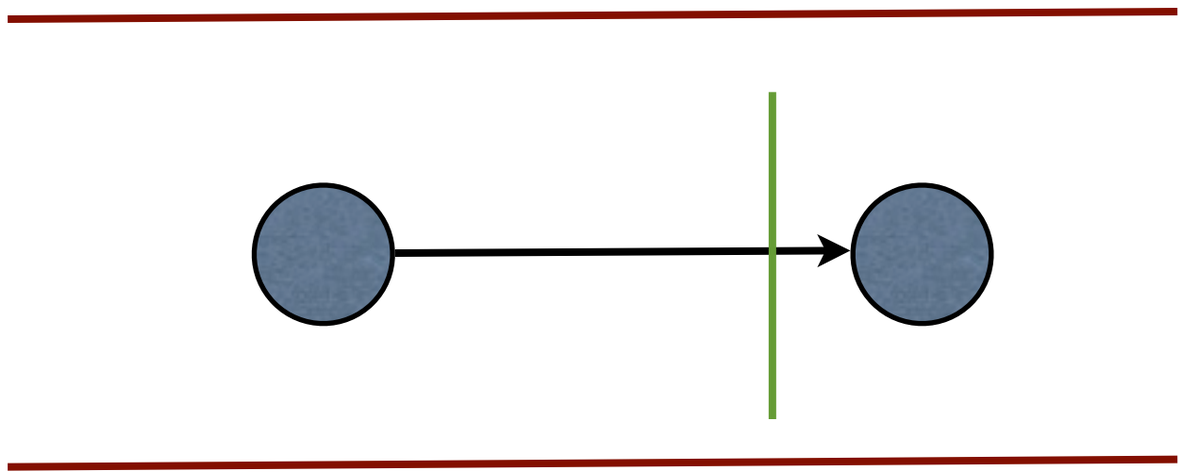
Gesture 1: Move right



```
(defrule move_right
  (Tuio2Dcur (x ?x1) (time ?t1))
  (Tuio2Dcur (x ?x2) (time ?t2))
  (test (< ?t1 ?t2))
  (test (> ?x2 (+ ?x1 0.2)))
  (test (< ?x2 (+ ?x1 0.3)))
  ...
=>
  (printout t “Moved right!”))
```

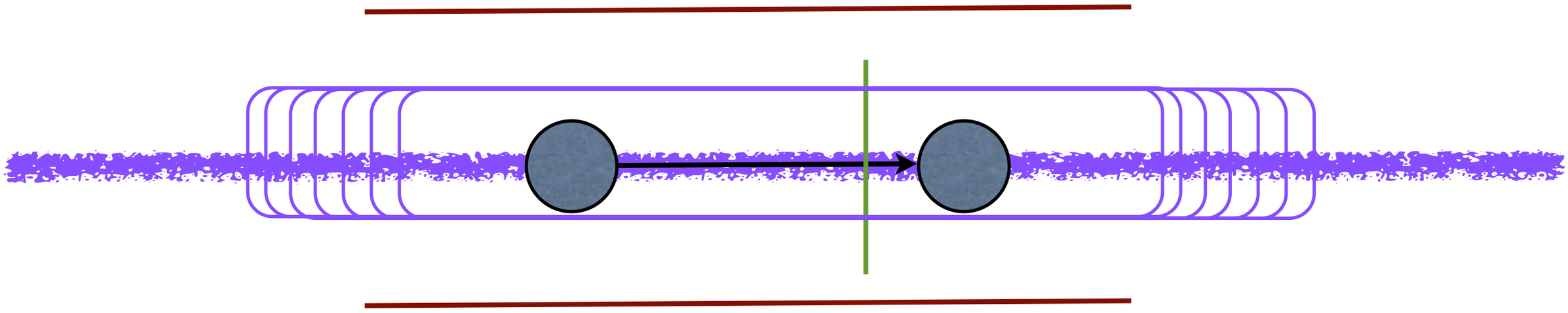
(feql a b diff)

Gesture 1: Move right

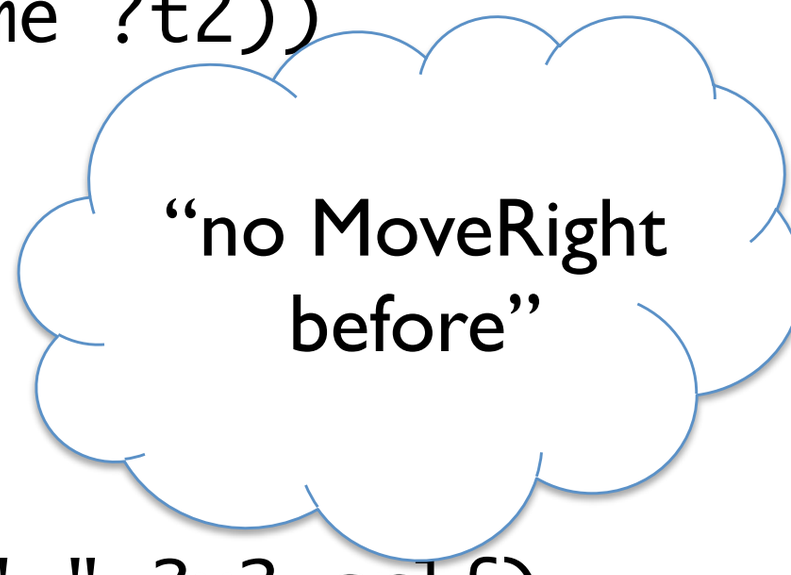


```
(defrule move_right
  (Tuio2Dcur (x ?x1) (y ?y1) (time ?t1))
  (Tuio2Dcur (x ?x2) (y ?y2) (time ?t2))
  (test (< ?t1 ?t2))
  (test (> ?x2 (+ ?x1 0.2)))
  (test (< ?x2 (+ ?x1 0.3)))
  (test (feql ?y1 ?y2 0.025))
=>
  (printout t “Moved right!”))
```

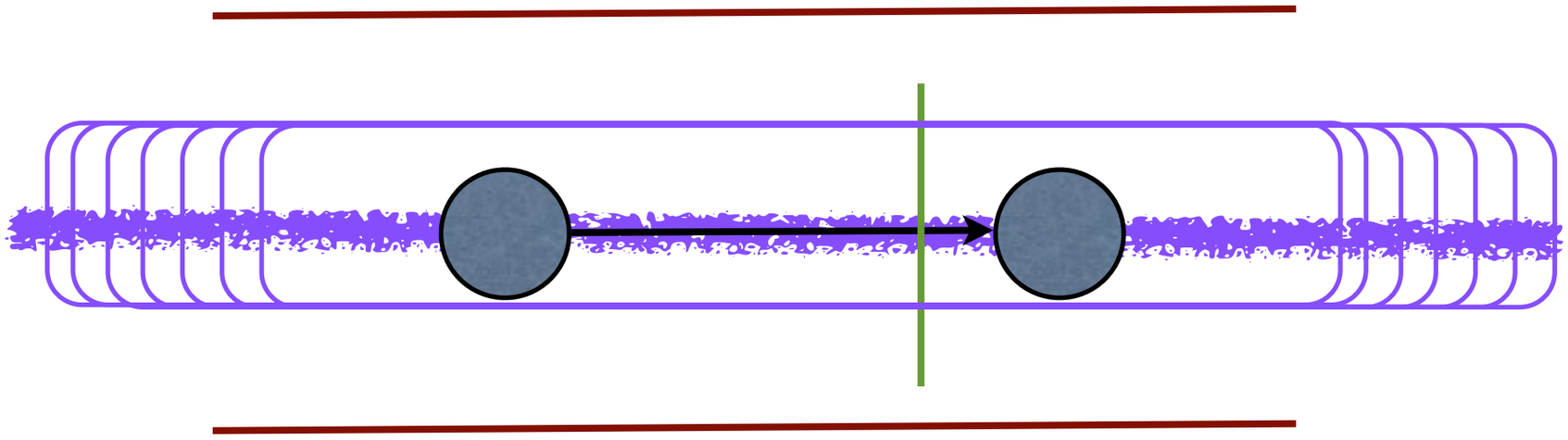
Gesture 1: Move right



```
(defrule move_right
  (Tuio2Dcur (x ?x1) (y ?y1) (time ?t1))
  (Tuio2Dcur (x ?x2) (y ?y2) (time ?t2))
  (test (< ?t1 ?t2))
  (test (> ?x2 (+ ?x1 0.248)))
  (test (< ?x2 (+ ?x1 0.3)))
  (test (feql ?y1 ?y2 0.025))
=>
  (printout t "MoveRight! " ?x1 " " ?x2 crlf)
  (assert (MoveRight (x ?x2))))
```



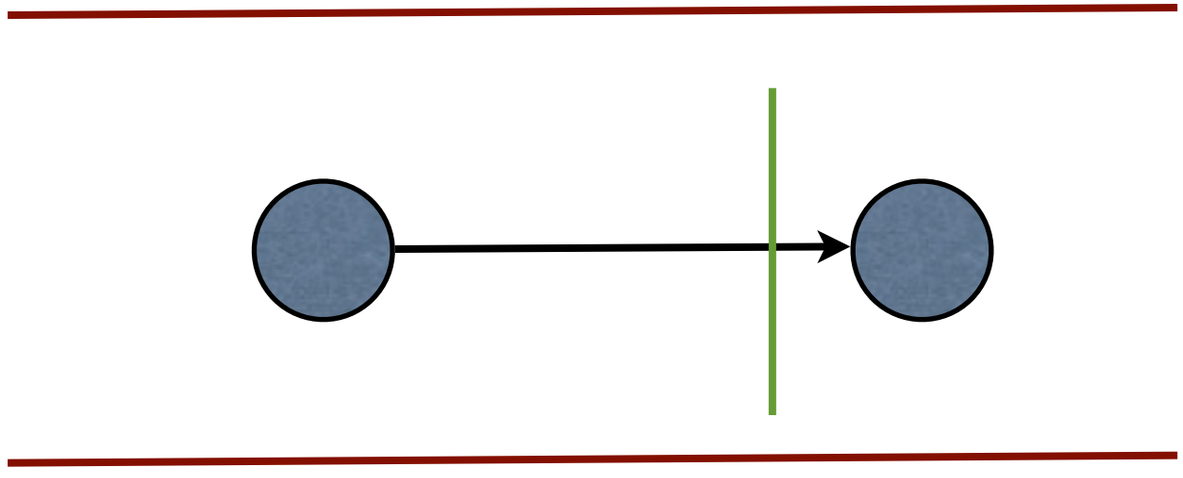
Gesture 1: Move right



```
(defrule move_right
  (Tuio2Dcur (x ?x1) (y ?y1) (time ?t1))
  (not (and (MoveRight (x ?nx))
            (test (> ?nx ?x1))))
  (Tuio2Dcur (x ?x2) (y ?y2) (time ?t2))
  (test (< ?t1 ?t2))
  (test (> ?x2 (+ ?x1 0.248)))
  (test (< ?x2 (+ ?x1 0.3)))
  (test (feql ?y1 ?y2 0.025))
=>
  (printout t "MoveRight! " ?x1 " " ?x2 crlf)
  (assert (MoveRight (x ?x2))))
```

Gesture 1: Move right

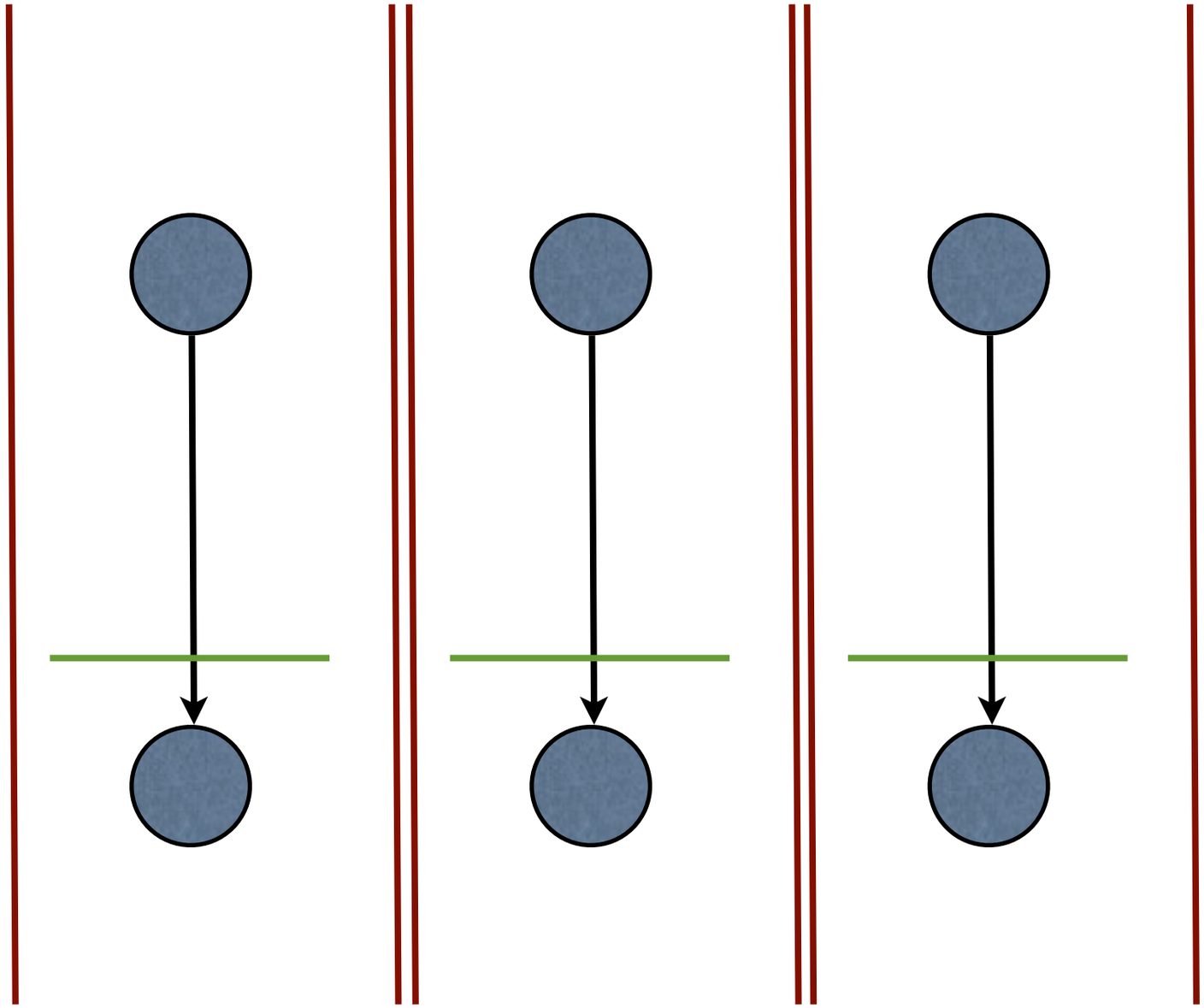
Discarding history



```
(defrule move_right
  (Tuio2Dcur (x ?x1) (y ?y1) (time ?t1))
  (not (and (MoveRight (time ?tn))
            (test (> ?tn ?t1))))
  (Tuio2Dcur (x ?x2) (y ?y2) (time ?t2))
  (test (< ?t1 ?t2))
  (test (> ?x2 (+ ?x1 0.248)))
  (test (< ?x2 (+ ?x1 0.3)))
  (test (feql ?y1 ?y2 0.025))
  (not (and (Tuio2Dcur (state 0|1) (time ?tn))
            (test (> ?tn ?t1))
            (test (< ?tn ?t2))))
=>
  (printout t "MoveRight! " ?x1 " " ?x2 crlf)
  (assert (MoveRight (x ?x2))))
```

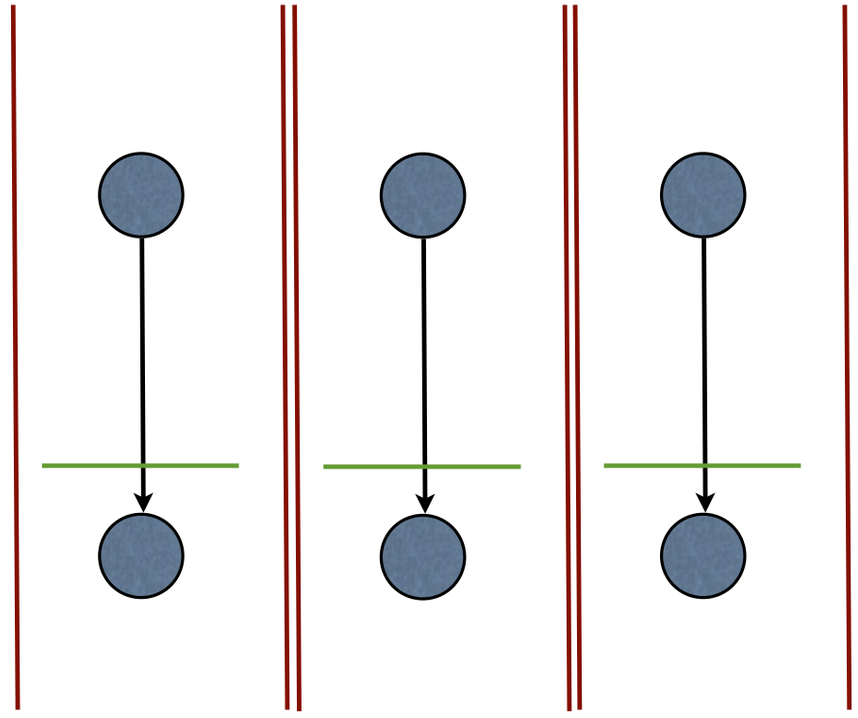
"no up|down in between"

Gesture 2: Supporting multiple “move down”s



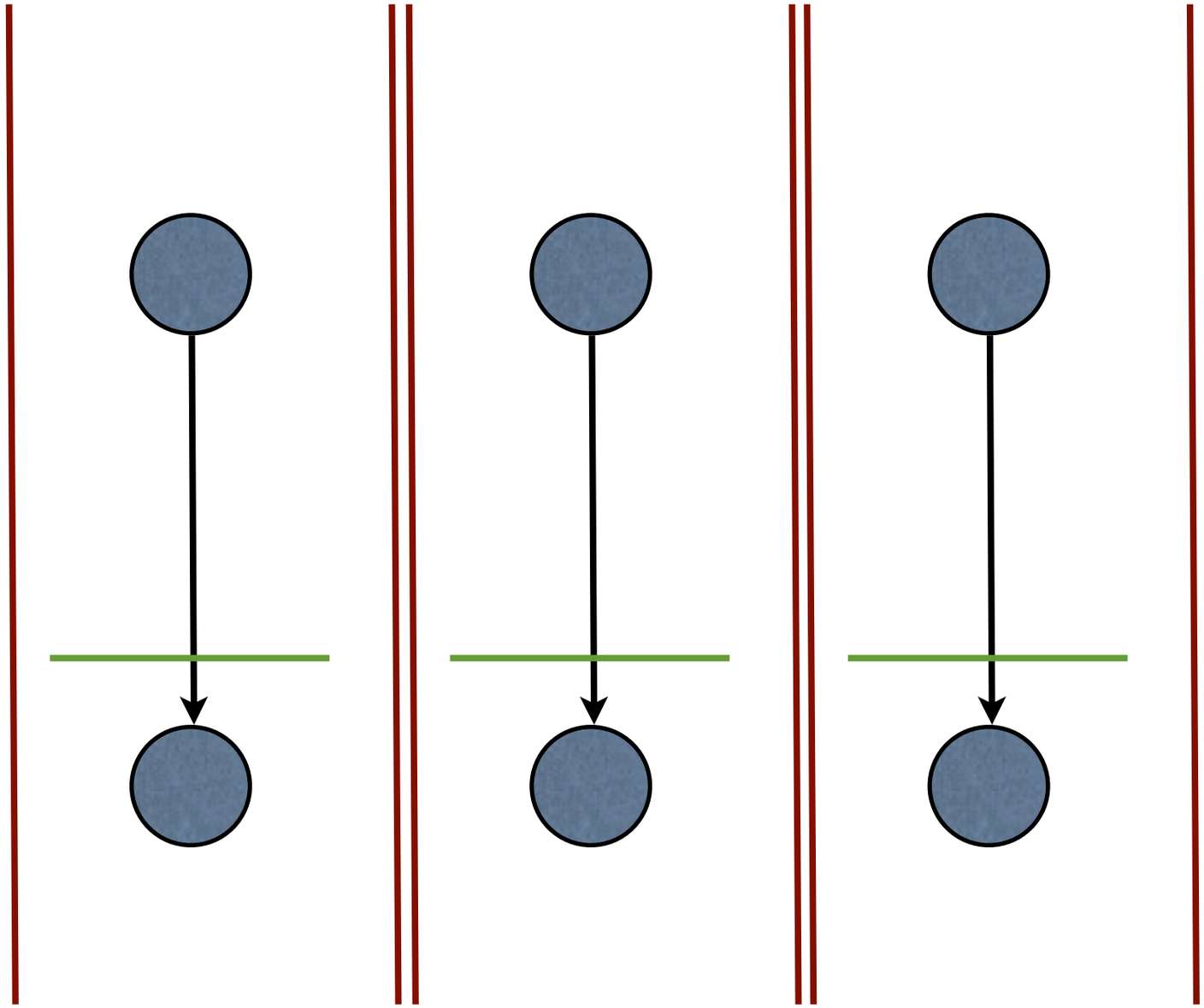
Gesture 2: Supporting multiple “move down”s

Unification

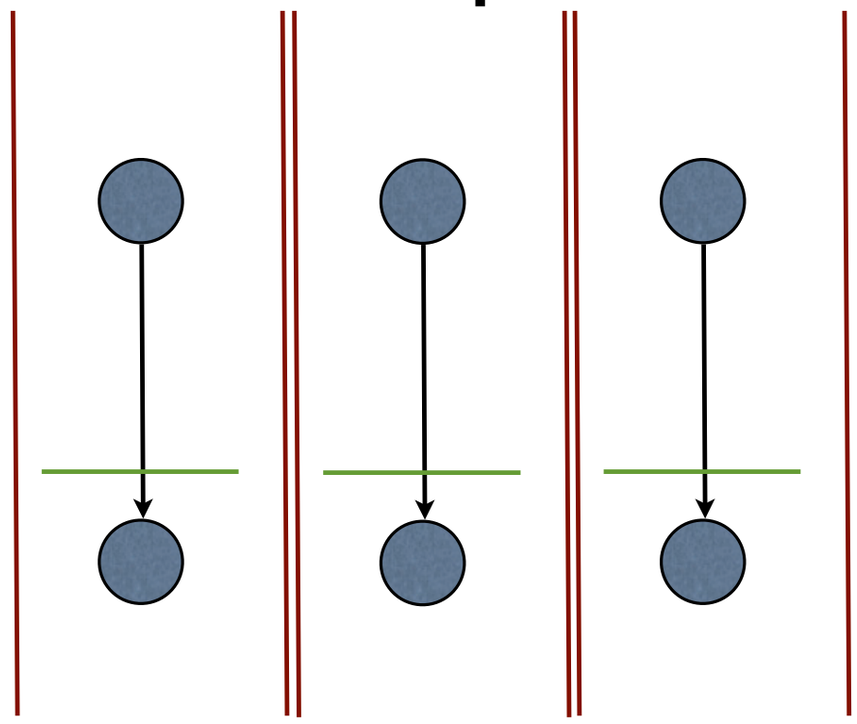


```
(defrule move_down
   (Tuio2Dcur (id ?id)) (x ?x1) (y ?y1) (time ?t1))
   (not (and (MoveDown (id ?id) (time ?nt))
             (test (> ?nt ?t1))))
   (Tuio2Dcur (id ?id) (x ?x2) (y ?y2) (time ?t2))
   (test (< ?t1 ?t2))
   (test (> ?y2 (+ ?y1 0.2)))
   (test (< ?y2 (+ ?y1 0.3)))
   (test (feql ?x1 ?x2 0.02))
=>
  (printout t "MoveDown! " ?id " " ?y1 " " ?y2 " " ?t2 crlf)
  (assert (MoveDown (id ?id) (time ?t2))))
```

Gesture 3: Composed 3 down

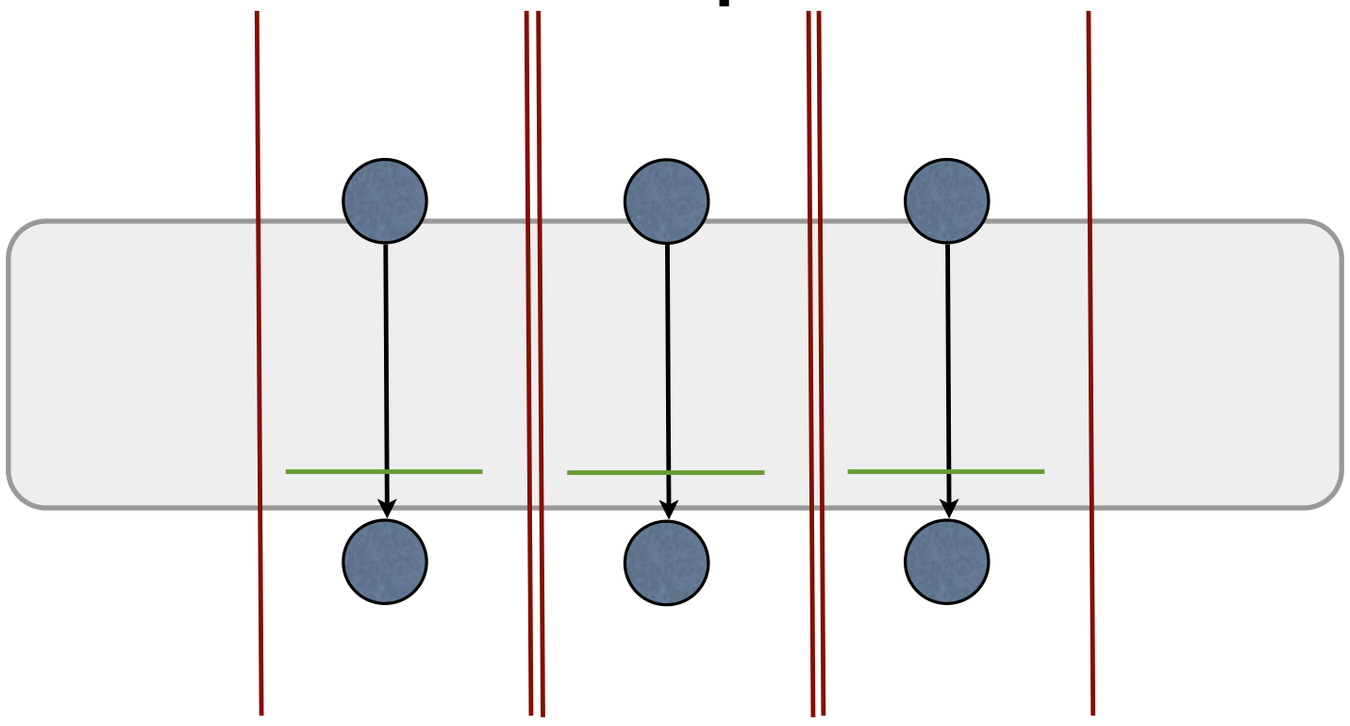


Gesture 3: Composed 3 down



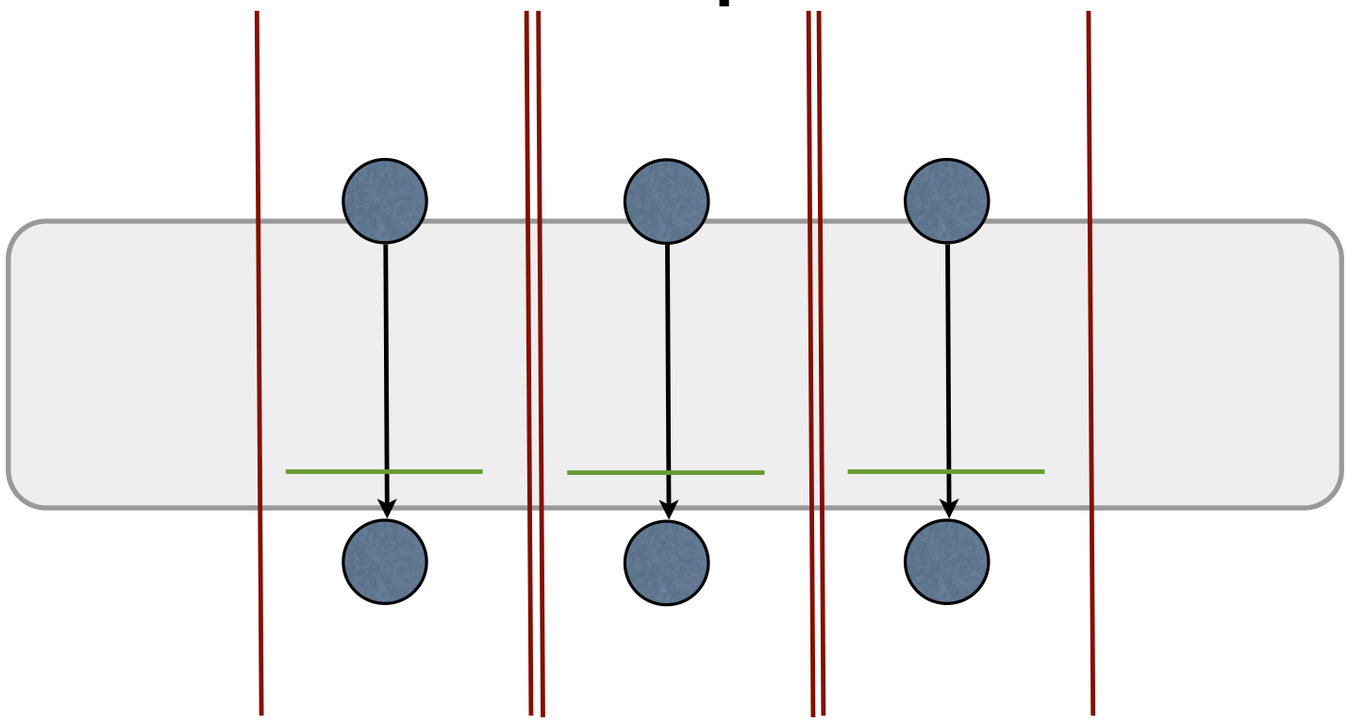
```
(defrule move3Down
  (MoveDown (x ?x1) (y ?y1) (time ?t1))
  (MoveDown (x ?x2) (y ?y2) (time ?t2))
  (MoveDown (x ?x3) (y ?y3) (time ?t3))
  ...
=>
  (assert (Move3Down (time ?t1)))
```

Gesture 3: Composed 3 down



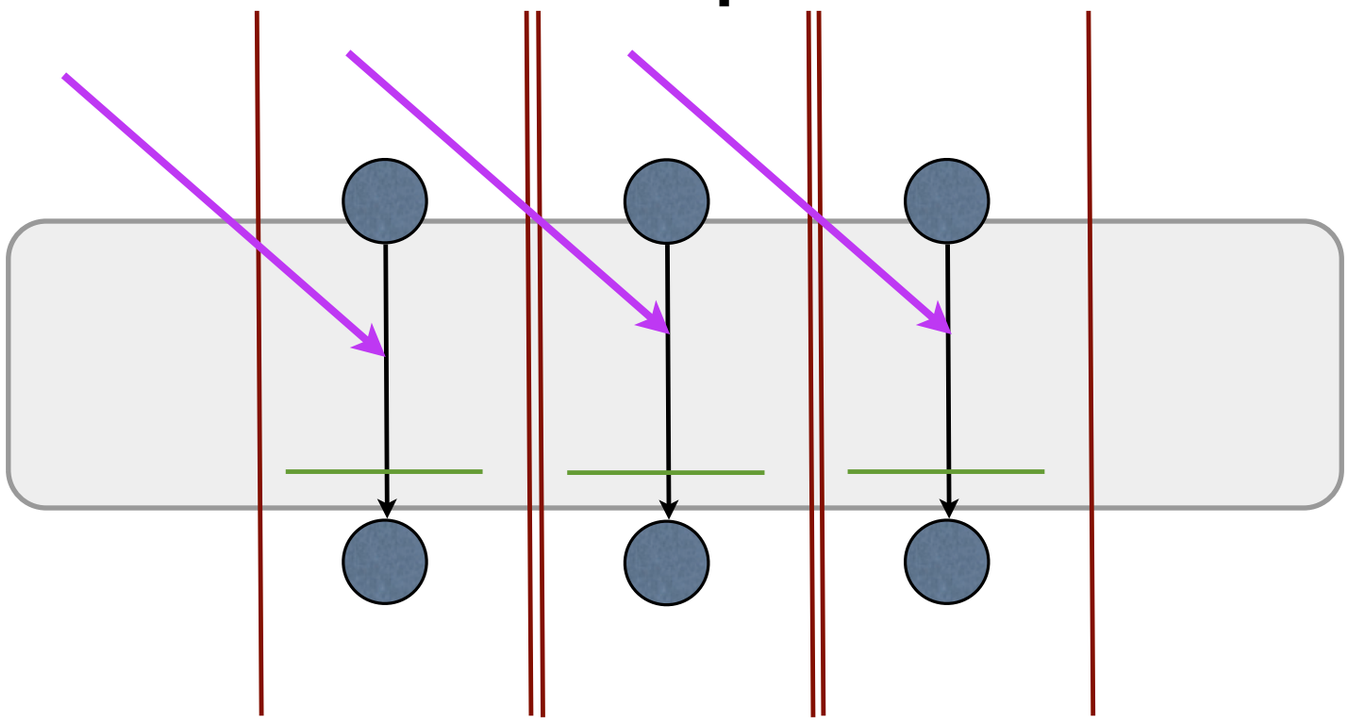
```
(defrule move3Down
  (MoveDown (x ?x1) (y ?y1) (time ?t1))
  (MoveDown (x ?x2) (y ?y2) (time ?t2))
  (MoveDown (x ?x3) (y ?y3) (time ?t3))
  ...
=>
  (assert (Move3Down (time ?t1)))
```

Gesture 3: Composed 3 down



```
(defrule move3Down
  (MoveDown (x ?x1) (y ?y1) (time ?t1))
  (MoveDown (x ?x2) (y ?y2) (time ?t2))
  (MoveDown (x ?x3) (y ?y3) (time ?t3))
  (test (< ?x1  ?x2 ?x3))
  (test (feql ?y1  ?y2 0.05))
  (test (feql ?y2 ?y3 0.05))
=>
  (assert (Move3Down (time ?t1)))
```

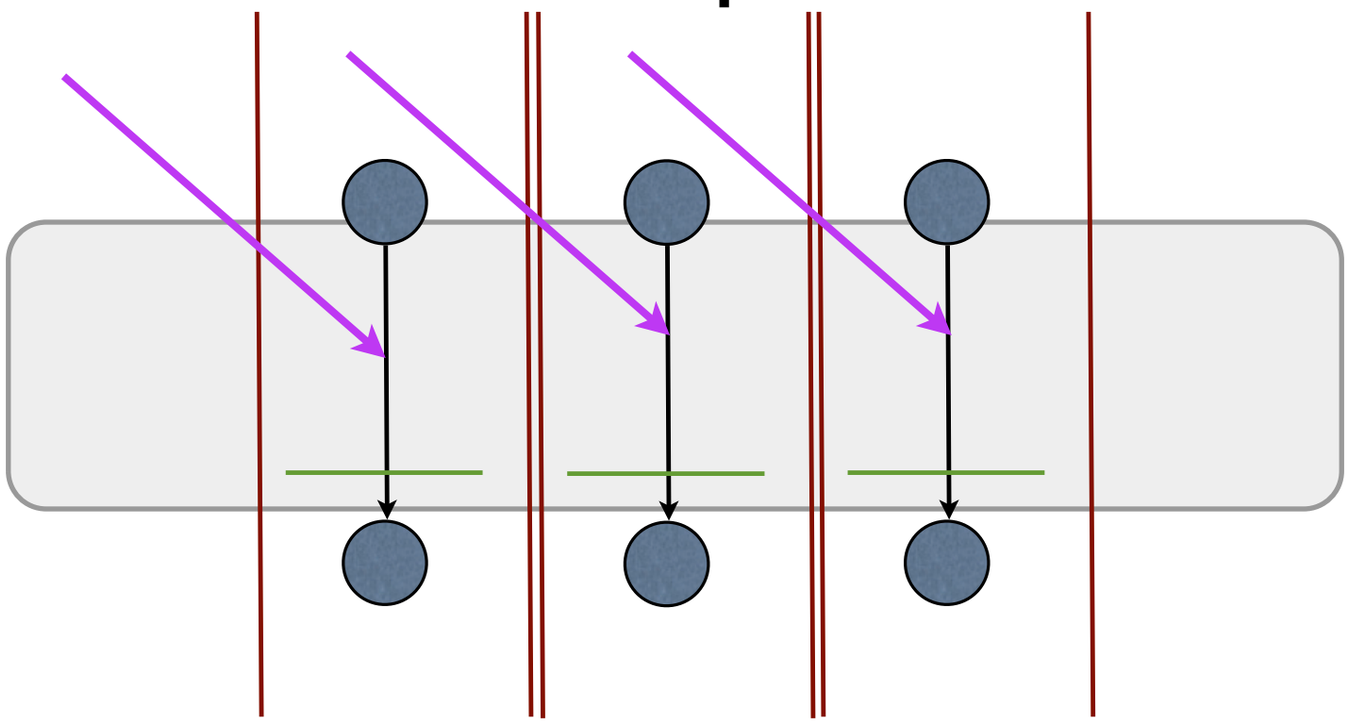
Gesture 3: Composed 3 down



```
(defrule move3Down
  (MoveDown (x ?x1) (y ?y1) (time ?t1))
  (MoveDown (x ?x2) (y ?y2) (time ?t2))
  (MoveDown (x ?x3) (y ?y3) (time ?t3))
  (test (< ?x1  ?x2 ?x3))
  (test (feql ?y1  ?y2 0.05))
  (test (feql ?y2 ?y3 0.05))
=>
  (assert (Move3Down (time ?t1)))
```

From different Fingers (IDs)

Gesture 3: Composed 3 down



```
(defrule move3Down
  (MoveDown (id ?i1) (x ?x1) (y ?y1) (time ?t1))
  (MoveDown (id ?i2) (x ?x2) (y ?y2) (time ?t2))
  (MoveDown (id ?i3) (x ?x3) (y ?y3) (time ?t3))
  (test (<> ?i1 ?i2 ?i3))
  (test (< ?x1 ?x2 ?x3))
  (test (feql ?y1 ?y2 0.05))
  (test (feql ?y2 ?y3 0.05))
=>
  (assert (Move3Down (time ?t1)))
```

Mudra

1. Identification
2. Spatio-temporal parameters
3. GUI-symbiosis
4. Segmentation

Multimodal Fusion