

Jan 17, 10 11:25

code.txt

Page 1/35

```

1 # ./games.lisp
2 #-----
3
4 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
5 ; This file is part of AIsplash.
6 ;
7 ; AIsplash is free software: you can redistribute it and/or modify
8 ; it under the terms of the GNU General Public License as published by
9 ; the Free Software Foundation, either version 3 of the License, or
10 ; (at your option) any later version.
11 ;
12 ; AIsplash is distributed in the hope that it will be useful,
13 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
14 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
15 ; GNU General Public License for more details.
16 ;
17 ; You should have received a copy of the GNU General Public License
18 ; along with AIsplash. If not, see <http://www.gnu.org/licenses/>.
19 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
20
21 ; disable an irritating SBCL flag
22 ;#+SBCL (DECLAIM (SB-EXT:MUFFLE-CONDITIONS CL:STYLE-WARNING))
23
24 (defun make (&rest files)
25   (handler-bind
26     ((style-warning #'muffle-warning))
27     (dolist (f files)
28       (load f))))
29
30 (defun make-tricks ()
31   "timm tricks"
32   (make "lispl01/deftest" ; must be first
33         "lispl01/caution"
34         "lispl01/strings"
35         "lispl01/hash"
36         "lispl01/list"
37         "lispl01/random"
38         "lispl01/any"
39         "lispl01/reading"
40         "lispl01/normal"
41         "lispl01/number"
42         "lispl01/lispfuns"
43         "lispl01/debug"
44   ))
45
46 (defun make-tables ()
47   "timm's table tricks"
48   (make-tricks)
49   (make "table/structs"
50         "table/header"
51         "table/data"
52         "table/table"
53         "table/xindex"
54   ))
55
56 (defun make-game()
57   "gaming code"
58   (make-tricks)
59   (make "game/globals"
60         "game/macros"
61         "game/defstructs"
62         "game/roaming"
63         "game/types"
64         "game/printing"
65         "game/creation"
66         "game/stagger"
67   ))
68
69 (make-game)
70

```

Jan 17, 10 11:25

code.txt

Page 2/35

```

71 # ./lispl01/number.lisp
72 #-----
73
74 ;this is a comment
75 (defun square (x) (* x x))
76

```

Jan 17, 10 11:25

code.txt

Page 3/35

```

77 # ./lispl01/reading.lisp
78 #-----
79
80 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
81 ; This file is part of AIslash.
82 ;
83 ; AIslash is free software: you can redistribute it and/or modify
84 ; it under the terms of the GNU General Public License as published by
85 ; the Free Software Foundation, either version 3 of the License, or
86 ; (at your option) any later version.
87 ;
88 ; AIslash is distributed in the hope that it will be useful,
89 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
90 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
91 ; GNU General Public License for more details.
92 ;
93 ; You should have received a copy of the GNU General Public License
94 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
95 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
96
97 (defun file->linesOfChar (f)
98   (with-open-file (str f)
99     (stream->linesOfChar str)))
100
101 (defun stream->linesOfChar (str &optional
102   (line (read-line str nil)))
103   (when line
104     (cons (coerce line 'list)
105           (stream->linesOfChar str))))
106

```

Jan 17, 10 11:25

code.txt

Page 4/35

```

107 # ./lispl01/any.lisp
108 #-----
109
110 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
111 ; This file is part of AIslash.
112 ;
113 ; AIslash is free software: you can redistribute it and/or modify
114 ; it under the terms of the GNU General Public License as published by
115 ; the Free Software Foundation, either version 3 of the License, or
116 ; (at your option) any later version.
117 ;
118 ; AIslash is distributed in the hope that it will be useful,
119 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
120 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
121 ; GNU General Public License for more details.
122 ;
123 ; You should have received a copy of the GNU General Public License
124 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
125 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
126
127 (defmethod any ((x list))
128   (nth (my-random-int (length x)) x))
129
130 (defmethod any ((x hash-table))
131   (let ((count (hash-table-count x)))
132     (unless (zerop count)
133       (let ((n (1+ (my-random-int count))))
134         (dohash (key value x)
135                 (if (<= (decf n) 0)
136                     (return-from any value)))))))
137
138 (defun <~ (n1 n2)
139   (if (= n1 n2)
140       (< (my-random 100) 50)
141       (< n1 n2)))
142

```

Jan 17, 10 11:25

code.txt

Page 5/35

```

143 # ./lisp101/caution.lisp
144 #-----
145
146 (defstruct (caution (:print-function caution-print))
147   all (patience 20) killed)
148
149 (defun caution-print (c s depth)
150   (declare (ignore depth))
151   (format s "#(CAUTION :ALL ~a :PATIENCE ~a)"
152     (caution-all c)
153     (caution-patience c)))
154
155 (defun ok (test cautions format-str &rest args)
156   (or test
157     (let ((message (apply #'format `(nil ,format-str ,@args))))
158       (push message (caution-all cautions))
159       (defc (caution-patience cautions))
160       (format t "% ~a~%" message)
161       (when (< (caution-patience cautions) 0)
162         (setf (caution-killed cautions) t)
163         (error "too many warnings"))
164       nil)))
165
166 (defun die (cautions format-str &rest args)
167   (apply #'ok `(nil ,cautions ,format-str ,@args))
168   (setf (caution-killed cautions) t)
169   (error "gasp... wheeze... rosebud... (thud)"))
170
171 (defun test-out-of-patience ()
172   "can't be a deffttest cause it crashes on too many errors"
173   (let ((c (make-caution :patience 5)))
174     (dotimes (i 6) ; patience + 1
175       (ok (= 1 2) c "bad ~a ~a" 1 2)))
176
177 (deftest test-caution ()
178   (let ((c (make-caution)))
179     (ok (= 1 2) c "bad ~a ~a" 1 2)
180     (check
181      (= 19 (caution-patience c) 19)
182      (samep "bad 1 2" (first (caution-all c))))))
183

```

Jan 17, 10 11:25

code.txt

Page 6/35

```

184 # ./lisp101/strings.lisp
185 #-----
186
187 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
188 ; This file is part of AIslash.
189 ;
190 ; AIslash is free software: you can redistribute it and/or modify
191 ; it under the terms of the GNU General Public License as published by
192 ; the Free Software Foundation, either version 3 of the License, or
193 ; (at your option) any later version.
194 ;
195 ; AIslash is distributed in the hope that it will be useful,
196 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
197 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
198 ; GNU General Public License for more details.
199 ;
200 ; You should have received a copy of the GNU General Public License
201 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
202 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
203
204 (defun lt (x y)
205   (if (string-lessp (format nil "~a" x) (format nil "~a" y))
206       t
207       nil))
208
209 (defun nchars (n &optional (char #\Space))
210   (with-output-to-string (s)
211     (dotimes (i n)
212       (format s "~a" char))))
213
214 (defun whitespacep (char)
215   (member char '(#\Space #\Tab #\Newline #\Page) :test #'char=))
216
217 (defun whiteout (seq)
218   (remove-if #'whitespacep seq))
219
220 (defun samep (x y)
221   (string= (whiteout (format nil "~(~a~)" x))
222            (whiteout (format nil "~(~a~)" y))))
223
224

```

Jan 17, 10 11:25

code.txt

Page 7/35

```

225 # ./lisp101/array.lisp
226 #-----
227
228 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
229 ; This file is part of AIslash.
230 ;
231 ; AIslash is free software: you can redistribute it and/or modify
232 ; it under the terms of the GNU General Public License as published by
233 ; the Free Software Foundation, either version 3 of the License, or
234 ; (at your option) any later version.
235 ;
236 ; AIslash is distributed in the hope that it will be useful,
237 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
238 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
239 ; GNU General Public License for more details.
240 ;
241 ; You should have received a copy of the GNU General Public License
242 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
243 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
244
245 (defmacro doarray ((one i a &optional out) &body body )
246   (let ((max (gensym))
247         (b (gensym)))
248     `(let* ((,b ,a)
249            (,max (length ,b)))
250        (dotimes (,i ,max ,out)
251          (let ((one (aref ,b ,i)))
252            ,@body))))))
253
254 (defmacro doarray2 ((one two i a1 a2 &optional out) &body body )
255   (let ((max (gensym))
256         (b1 (gensym))
257         (b2 (gensym)))
258     `(let* ((,b1 ,a1)
259            (,b2 ,a2)
260            (,max (length ,b1)))
261        (dotimes (,i ,max ,out)
262          (let ((one (aref ,b1 ,i))
263                (two (aref ,b2 ,i)))
264            ,@body))))))
265

```

Jan 17, 10 11:25

code.txt

Page 8/35

```

266 # ./lisp101/list.lisp
267 #-----
268
269 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
270 ; This file is part of AIslash.
271 ;
272 ; AIslash is free software: you can redistribute it and/or modify
273 ; it under the terms of the GNU General Public License as published by
274 ; the Free Software Foundation, either version 3 of the License, or
275 ; (at your option) any later version.
276 ;
277 ; AIslash is distributed in the hope that it will be useful,
278 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
279 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
280 ; GNU General Public License for more details.
281 ;
282 ; You should have received a copy of the GNU General Public License
283 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
284 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
285
286 (defmacro geta (key list)
287   `(cdr (assoc ,key ,list)))
288
289 (defmacro last1 (l) `(first (last ,l)))
290
291 (defun accumulate (* first rest)
292   (if (null rest)
293       first
294       (let ((first1 (first rest))
295             (rest1 (rest rest)))
296         (accumulate *
297                     (funcall * first first1)
298                     rest1))))
299
300 (deftest test-accumulate ()
301   (accumulate #'min 10 '(11 12 9 13)))
302
303 (defun shuffle (l)
304   (dotimes (i (length l) 1)
305     (rotatef
306      (elt l i)
307      (elt l (my-random-int (length l))))))
308
309 (defun allbut (l n)
310   (if (zerop n)
311       (values (rest l) (first l))
312       (cons (first l) (allbut (rest l) (1- n)))))
313
314 (defun transpose (x)
315   (apply #'mapcar (cons #'list x)))
316
317 (defmacro doitems ((one n list &optional out) &body body )
318   `(let ((,n -1))
319     (dolist (,one ,list ,out) (incf ,n) ,@body)))
320

```

Jan 17, 10 11:25

code.txt

Page 9/35

```

321 # ./lisp101/debug.lisp
322 #-----
323
324 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
325 ; This file is part of AIslash.
326 ;
327 ; AIslash is free software: you can redistribute it and/or modify
328 ; it under the terms of the GNU General Public License as published by
329 ; the Free Software Foundation, either version 3 of the License, or
330 ; (at your option) any later version.
331 ;
332 ; AIslash is distributed in the hope that it will be useful,
333 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
334 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
335 ; GNU General Public License for more details.
336 ;
337 ; You should have received a copy of the GNU General Public License
338 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
339 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
340
341 <h1><join>Debugging Tricks</join><h1>
342 <h2>Print name of variable, and its bindings</h2><pre>
343 (defmacro o (&rest l)
344   (let ((last (gensym)))
345     `(let (,last)
346        ,@(mapcar #'(lambda(x) `(setf ,last (oprim ,x))) l)
347          (terpri)
348          ,last)))
349
350 (defmacro oprim (x)
351   `(progn (format t "[~a]=[~a] " (quote ,x) ,x) ,x)
352 </pre>
353 <p>E.g.<pre>
354 (deftest test-o ()
355   (let* ((a 1)
356         (b 2)
357         (c (+ a b)))
358     (o a b c)) ; prints [a]=[1] [b]=[2] [c]=[3]
359 </pre>
360 <h2>Profile code</h2><pre>
361 (defmacro watch (code)
362   `(progn
363     (sb-profile:unprofile)
364     (sb-profile:reset)
365     (sb-profile:profile ,@(my-funs))
366     (eval (progn ,code t))
367     (sb-profile:report)
368     (sb-profile:unprofile)))
369
370 (defun my-funs ()
371   (let ((out '()))
372     (do-symbols (s)
373       (if (and (fboundp s)
374               (find-symbol (format nil "~a" s) *package*)
375               (not (member s *lisp-funs*)))
376         (push s out)))
377     out))
378 </pre>
379 <h2>Timing of executions</h2><p>Returns the mean runtime
380 of each <t>n</t> repeats.<pre>
381 (defmacro time-it (n &body body)
382   (let ((n1 (gensym))
383         (i (gensym))
384         (t1 (gensym)))
385     `(let ((,n1 ,n)
386          (,t1 (get-internal-run-time)))
387        (dotimes (,i ,n1) ,@body)
388        (float (/ (- (get-internal-run-time) ,t1)
389                  (* ,n1 internal-time-units-per-second))))))
390 </pre>
391

```

Jan 17, 10 11:25

code.txt

Page 10/35

```

392 # ./lisp101/interact.lisp
393 #-----
394
395 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
396 ; This file is part of AIslash.
397 ;
398 ; AIslash is free software: you can redistribute it and/or modify
399 ; it under the terms of the GNU General Public License as published by
400 ; the Free Software Foundation, either version 3 of the License, or
401 ; (at your option) any later version.
402 ;
403 ; AIslash is distributed in the hope that it will be useful,
404 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
405 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
406 ; GNU General Public License for more details.
407 ;
408 ; You should have received a copy of the GNU General Public License
409 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
410 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
411
412 (let ((goal 10))
413   (defun assess (x)
414     (let (msg status)
415       (cond ((= x goal) (setf msg "just right!"
416                                status t))
417             ((> x goal) (setf msg "too big"))
418             ((< x goal) (setf msg "too small")))
419       (format t "[~a] is ~a~%" x msg
420               status)))
421
422 (defun looping (&key
423                (fn #'assess)
424                (eof :end)
425                (prompt "Guess")
426                (stream *standard-input*)
427                (patience 10))
428   (defc patience)
429   (unless (< patience 0)
430     (if prompt (format t "~a ~a> " patience prompt))
431     (let ((guess (read stream nil eof)))
432       (cond ((eq guess eof) nil)
433             ((funcall fn guess) guess)
434             (t
              (looping
               :fn fn :eof eof :stream stream
               :prompt prompt :patience patience ))))))
438
439 (defun looping-from-string (string)
440   (with-input-from-string (stream string)
441     (looping :stream stream)))
442
443 (defun looping-from-prompt ()
444   (looping))
445
446 (defun looping-from-file (f)
447   (with-open-file (stream f)
448     (looping :stream stream)))
449
450 (defun test-loop1 () (looping))
451
452 (defun test-loop2 ()
453   (looping-from-string
454    "1 2 3 11 23 12 22 10 1 4 100 55 33 66 33 111 44 66"))
455
456 (defun test-loop3 ()
457   (looping-from-file "lisp101/eg/1"))
458

```

Jan 17, 10 11:25

code.txt

Page 11/35

```

459 # ./lisp101/random.lisp
460 #-----
461
462 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
463 ; This file is part of AIslash.
464 ;
465 ; AIslash is free software: you can redistribute it and/or modify
466 ; it under the terms of the GNU General Public License as published by
467 ; the Free Software Foundation, either version 3 of the License, or
468 ; (at your option) any later version.
469 ;
470 ; AIslash is distributed in the hope that it will be useful,
471 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
472 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
473 ; GNU General Public License for more details.
474 ;
475 ; You should have received a copy of the GNU General Public License
476 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
477 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
478
479 <h1><join>Random Numbers in LISP</join></h1>
480 <h2>Pseudo-random Number Generation</h2>
481 <p>The bald guy says <em>when debugging code containing random
482 ;selections, it is useful to
483 ;recreate a
484 ;prior run that lead to an error</em>. Hence, most languages
485 ;offer a pseudo-random number generator which generates a sequence of
486 ;random numbers from some
487 ;seed. Exactly the same sequence of "random" numbers can be recreated by
488 ;resetting the seed and re-running the random sequence.
489 <p>Sadly, I can't make LISP's built in random number generator work that
490 ;way. Yes, I 've tried resetting *random-state* and that did not work.
491 ;Also, I began to worry
492 ;that what ever I did would not be portable to other LISPs.
493 <p>So the following implements a
494 ;simple pseudo-random number generator.
495 <pre>
496 (defparameter *seed0* 10013)
497 (defparameter *seed* *seed0*)
498
499 (defun reset-seed () (setf *seed* *seed0*))
500 </pre>
501 <p>This is the Park-Miller multiplicative congruential randomizer
502 ; (CACM, October 88, Page 1195). Creates pseudo random floating
503 ; point numbers in the range 0.0 &lt; x &lt; 1.0.
504 <pre>
505 (defun park-miller-randomizer ()
506   (let ((multiplier
507         16807.0d0);16807 is (expt 7 5)
508         (modulus
509         2147483647.0d0)) ;2147483647 is (- (expt 2 31) 1)
510     (let ((temp (* multiplier *seed*))
511           (setf *seed* (mod temp modulus))
512           (/ *seed* modulus))))
513 </pre>
514 <h2>My-random</h2> <p> Returns a pseudo random floating-point number
515 ; in range 0.0 &lt;= number < n.
516 ;Note that we subtract the randomly generated number from 1.0
517 ; before scaling so that we end up in the range
518 ; 0.0 &lt;= x < 1.0, not 0.0 <= x &lt; 1.0.<pre>
519 (defun my-random (n)
520   (let ((random-number (park-miller-randomizer))
521         (* n (- 1.0d0 random-number))))
522 </pre>
523 <h2>My-random-int</h2><p> Returns a pseudo random integer
524 ; in range 0 &lt;= number < n-1.<pre>
525 (defun my-random-int (n)
526   (let ((random-number (/ (my-random 1000.0) 1000))
527         (floor (* n random-number))))
528 </pre><H2>Demo</h2> If we've done the above right, then if we
529 ;generate
530 ; two sequences of random numbers, then if we reset the seed between
531 ; sequence one and sequence two, then the two sequences should be the
532 ; same.

```

Jan 17, 10 11:25

code.txt

Page 12/35

```

533 <pre>
534 (deftest test-random ()
535   (check
536     (equalp (random-demo) (random-demo))))
537
538 (defun random-demo (&optional (resetp t))
539   (let (counts out)
540     (labels
541       ((sorter (x y) (< (car x) (car y)))
542        (zap () (setf out nil)
543              (if resetp (reset-seed))
544              (setf counts (make-hash-table)))
545        (inc (n) (setf (gethash n counts)
546                      (1+ (gethash n counts 0))))
547        (cache (k v) (push (list k v) out)))
548       (zap)
549       (dotimes (i 10000) ; 10000 times do
550         (inc (my-random-int 5))) ; generate a num 0..4
551       (maphash #'cache counts) ; hash key/buckets ==> lists
552       (sort out #'sorter)))) ; sort and print list
553 </pre>
554

```

Jan 17, 10 11:25

code.txt

Page 13/35

```

555 # ./lisp101/hash.lisp
556 #-----
557
558 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
559 ; This file is part of AIslash.
560 ;
561 ; AIslash is free software: you can redistribute it and/or modify
562 ; it under the terms of the GNU General Public License as published by
563 ; the Free Software Foundation, either version 3 of the License, or
564 ; (at your option) any later version.
565 ;
566 ; AIslash is distributed in the hope that it will be useful,
567 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
568 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
569 ; GNU General Public License for more details.
570 ;
571 ; You should have received a copy of the GNU General Public License
572 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
573 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
574
575 <h1><join>Hash Functions</join></h1>
576 <p>You can run over lists with dolist:
577 <pre>
578 (dolist (i 1000)
579   (print i))
580 </pre>
581 Why can't you run over hash tables the same way?
582 <h2>Macro Magic: dohash</h2>
583 <p>E.g. print every key value in a hash table
584 <pre>
585 (dohash (k v h)
586   (format t "~a = ~a~%" k v))
587 </pre>
588 <p>Code:
589 <pre>
590 (defmacro dohash ((key value hash &optional end) &body body)
591   `(progn (maphash #'(lambda (,key ,value) ,@body) ,hash)
592     ,end))
593 </pre><p>Short cut: just access the keys or values:<pre>
594 (defmacro dovalues ((value hash &optional end) &body body)
595   (let ((key (gensym)))
596     `(progn (maphash #'(lambda (,key ,value) ,@body) ,hash)
597       ,end)))
598
599 (defmacro dokeys ((key hash &optional end) &body body)
600   (let ((value (gensym)))
601     `(progn (maphash #'(lambda (,key ,value) ,@body) ,hash)
602       ,end)))
603 </pre><p>Q: why <tt>gensym</tt>? Hint: name collision within <tt>@body</tt>.
604 <h2>Pretty print a Hash Table</h2>
605 <p><tt>(dohash (k v h) (format t "~a = ~a~%" k v))</tt> is all very well
606 ; but what does a real pretty print for a hash table look like?
607 <pre>
608 (defun showh (h &key
609   (indent 0) (stream t) (before "") (after ""))
610   (if-empty "empty")
611   (show #'(lambda (x)
612     (format stream "~a~a = ~a~%"
613       (nchars indent) (first x) (rest x))))
614   (lt #'lt))
615   (if (zerop (hash-table-count h))
616     (format stream "~a~a~a" before if-empty after)
617     (let (l)
618       (format stream "~a" before)
619       (maphash #'(lambda (k v) (push (cons k v) l)) h)
620       (mapc show
621         (sort l #'(lambda (a b)
622           (funcall lt (car a) (car b))))))
623       (format stream "~a" after)
624       h)))
625 </pre>
626 <p>If that is too complex, then lets look at an example:
627 <pre>
628 (deftest test-showh ()

```

Jan 17, 10 11:25

code.txt

Page 14/35

```

629 (let ((h (make-hash-table)))
630   (dolist (one '(apple pear banana))
631     (setf (gethash (length (string one)) h) one))
632   (check
633     (samep (with-output-to-string (s) (showh h :stream s))
634       "4 = PEAR
635        5 = APPLE
636        6 = BANANA"))))
637 </pre>
638 <p>(By the same, <tt>samep</tt> is a very relaxed string comparison
639 ; operator. It ignores all white space- which means that a one space
640 ; typo in the test case does not wreck the test.
641 <p>And here's an example for <tt>do-values</tt>
642 <pre>
643 (deftest test-dovalues ()
644   (let (all
645     (h (make-hash-table)))
646     (dolist (one '(apple pear banana))
647       (setf (gethash (length (string one)) h) one))
648     (dovalues (value h)
649       (push value all))
650     (check
651       (equal '(banana pear apple) all))))
652 </pre>
653 <h2>Other Misc Stuff</h2>
654 <pre>
655 (defun keys2sorted-alist (h ranker)
656   (labels ((car-string-lessp (x y)
657     (string-lessp (rest x) (rest y))))
658     (let (all (n -1))
659       (dohash (key value h)
660         (push (cons key
661           (format nil (if (numberp key) ranker "~a") key))
662           all))
663       (mapcar #'(lambda (one) `((, (first one) . ,(incf n)))
664         (sort all #'car-string-lessp))))))
665
666 (defmethod print-object ((object hash-table) stream)
667   (format stream "#<HT ~a>" (hash-table-count object)))
668 </pre>

```

Jan 17, 10 11:25

code.txt

Page 15/35

```

669 # ./lisp101/defitest.lisp
670 #-----
671 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
672 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
673 ; This file is part of AIslash.
674 ;
675 ; AIslash is free software: you can redistribute it and/or modify
676 ; it under the terms of the GNU General Public License as published by
677 ; the Free Software Foundation, either version 3 of the License, or
678 ; (at your option) any later version.
679 ;
680 ; AIslash is distributed in the hope that it will be useful,
681 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
682 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
683 ; GNU General Public License for more details.
684 ;
685 ; You should have received a copy of the GNU General Public License
686 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
687 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
688 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
689 <h1><join>Unit Tests</join></h1>
690 ; <p>(From Peter Seibler's excellent text:
691 ; <a href="http://gigamonkeys.com/book">http://gigamonkeys.com/book</a>).
692 ;<p>
693
694 ; <h2>Q: What is the same across all languages?</h2>
695 ;<p>A: all programs in all languages need testing.
696 ; <p><em>I should be able to write a test in any language.
697 ; --J. Random Hacker</em>
698 ; <p>Every programming has to be debugged:
699 ; <a href="http://unbox.org/wisp/var/timm/09/310/share/img/bug.jpg">
700 ; </a>
702 ; <ul><em> Debugging had to be discovered. I can remember the exact
703 ; instant when I realized that a large part of my life from then on was
704 ; going to be spent in finding mistakes in my own programs.<br>
705 ; - Maurice Wilkes 1949
706 ; </em></ul>
707 ; <br clear=all>
708 ; <p>
709 ; Wilkes is pointing out that the time required to debug a system is
710 ; surprisingly large- over half the effort of building.
711 ; According to Brooks (The Mythical Man Month, 1995), the time
712 ; required to build software divides as follows:
713 ; <ul>
714 ; <li>
715 ; 1/3 in management and planning
716 ; <li> 1/6 in development
717 ; <li> 1/4 in unit test (testing all parts in isolation)
718 ; <li> 1/4 in system testing (testing all parts, in combination)
719 ; </ul>
720 ; </ul>
721 ; <p>Decades later, the same distribution persists. It looks like this:
722 ; <center>
723 ; 
724 ; </center>
725 ; <p>What this diagram is saying is that if you want to drastically change
726 ; the economics of software development, do something about
727 ; <em>testability</em>.
728 ; So dealing with a new language,
729 ; always start with the test engine.
730 ;<p>Peter Seibler's <tt>defitest</tt> test engine is both darn useful
731 ;and a good beginner's
732 ;test for LISP and LISP design. He took a test engine, found common
733 ;patterns, and wrote macros to implement those common features.
734 ;<p>Macros are a little beyond most LISP beginners so, if the following,
735 ;do not try to understand it all- just test yourself to see how much
736 ; you do understand.
737
738 ;<h2>Globals</h2>
739 ;<p>Place to store (1) a list of active tests and (2) a list of all test nam
es.
740 ;<pre>
741 (defparameter *test-name* nil)

```

Jan 17, 10 11:25

code.txt

Page 16/35

```

742 (defparameter *all-tests* nil)
743 ;</pre>
744 ;<p>Q: what is the difference between <tt>defparameter</tt>,
745 ; <tt>defvar</tt> and <tt>defconstant</tt>? Hint: usually use
746 ;<tt>defparameter</tt>
747
748 ;<h2>Running Test(s)</h2>
749 ;<p>Every test does the following..
750 ;<pre>
751 (defmacro run-tests (&body body)
752   `(progn
753     (make)           ; check you have the system updated
754     (tests-reset)   ; reset some counters to zero
755     ,@body           ; run the code
756     (tests-report)) ; report the state of the counters
757   )
758 ;<p>Q: in the above, what does "" and "@" do? Hint1: this is
759 ; really tricky. Hint2: <tt>run-tests</tt>
760 ; does not execute; rather it does some compile-time rewrites.
761 ;<p>Usually, we just run <tt>tests</tt> which
762 ; runs over all all the <tt>*all-tests*</tt> and runs one?
763 ;<pre>
764 (defun tests ()
765   (run-tests
766    (dolist (one (reverse *all-tests*))
767      (funcall one))))
768 ;</pre>
769 ;<p>Q: Why do we call <tt>reverse</tt> in the above? Hint:
770 ; the name of new tests is pushed onto <tt>*all-tests*</tt> in
771 ; the order of their arrival.
772 ;<p>Q: What does <tt>funcall</tt> do? Hint: pointer to functions.
773 ;<h2>Defining One Test</h2>
774 ;<p>When we define tests, we have to store its name in <tt>*all-tests*</tt>
775 ; then write a special <tt>defun</tt> using that test name.
776 ; Inside this <tt>defun</tt>, before we run a test, we push the name
777 ; to the end of a list of active tests.
778 ; <p>(Note that the explanation actually takes more characters
779 ; than the actual code. Tee hee..).
780 ;<pre>
781 (defmacro deftest (name parameters &body body)
782   `(progn
783     (setf *all-tests* (remove-duplicates (cons ',name *all-tests*)))
784     (defun ,name ,parameters
785       (let ((*test-name* (append *test-name* (list ',name))))
786         ,@body))))
787   )
788 ;</pre>
789 ;<h2>Support Tools</h2>
790 ;<p>A little complex. Exercise for the reader.
791 ;<pre>
792 (defmacro check (&body forms)
793   `(combine-results
794     ,(loop for f in forms collect `(report-result ,f ',f))))
795
796 (defmacro with-gensyms ((&rest names) &body body)
797   `(let ,(loop for n in names collect
798         `(,n (make-symbol ,(string n))))
799     ,@body)
800
801 (defmacro combine-results (&body forms)
802   (with-gensyms (result)
803     `(let ((,result t))
804       ,(loop for f in forms collect
805             `(unless ,f (setf ,result nil))))
806     ,result))
807 ;</pre>
808 ;<h2>Handling the Counters</h2>
809 ;<p>Object-oriented encapsulation can be implemented in LISP using
810 ; <em>closures</em>. In the following, only these functions can access
811 ; the number of <tt>passes</tt> and <tt>fails</tt>.
812 ;<pre>
813 (let ((passes 0) (fails 0))
814   (defun tests-reset()
815     (setf passes 0)
816     (setf fails 0))

```

Jan 17, 10 11:25

code.txt

Page 17/35

```

816 (defun tests-report ()
817   (if (zerop (+ passes fails))
818       (format t "no tests~%")
819       (format t "~%PASSES: ~a (~a %)~%FAILS : ~a~%"
820             passes (* 100 (/ passes (+ passes fails)))
821             fails))
822   (>= passes fails))
823 (defun report-result (result form)
824   (if result
825       (and (incf passes)
826            (format t "% ~a~%" *test-name*))
827       (and (incf fails)
828            (format t "~%fail ... ~a: ~a~%" *test-name* form)))
829   result)
830 )
831 ;</pre>
832 ;<p>Q: what do these three functions do?
833 ;<h2>Examples</h2>
834 ;<p>For examples of <tt>deftest</tt> in action, see
835 ;<a href="http://menzies.us/ai/?lib/lisp101/random.lisp">random.lisp</a> and
836 ;<a href="http://menzies.us/ai/?lib/lisp101/hash.lisp">hash.lisp</a>.
837 ;The <a href="http://menzies.us/ai/?lib/lisp101/hash.lisp">hash.lisp</a>
838 ; example shows one interesting feature of <tt>deftest</tt>: it can be
839 ; used to document the expected behaviour.
840

```

Jan 17, 10 11:25

code.txt

Page 18/35

```

841 # ./game/tests.lisp
842 #-----
843
844 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
845 ; This file is part of AIslash.
846 ;
847 ; AIslash is free software: you can redistribute it and/or modify
848 ; it under the terms of the GNU General Public License as published by
849 ; the Free Software Foundation, either version 3 of the License, or
850 ; (at your option) any later version.
851 ;
852 ; AIslash is distributed in the hope that it will be useful,
853 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
854 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
855 ; GNU General Public License for more details.
856 ;
857 ; You should have received a copy of the GNU General Public License
858 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
859 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
860
861

```

Jan 17, 10 11:25

code.txt

Page 19/35

```

862 # ./game/stagger.lisp
863 #-----
864
865 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
866 ; This file is part of AIslash.
867 ;
868 ; AIslash is free software: you can redistribute it and/or modify
869 ; it under the terms of the GNU General Public License as published by
870 ; the Free Software Foundation, either version 3 of the License, or
871 ; (at your option) any later version.
872 ;
873 ; AIslash is distributed in the hope that it will be useful,
874 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
875 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
876 ; GNU General Public License for more details.
877 ;
878 ; You should have received a copy of the GNU General Public License
879 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
880 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
881
882 (defun stagger (board &optional (n 10))
883   (onstream board)
884   (cond ((< n 0) (onstream board))
885         (t      (stagger1 board)
886                 (stagger board (1- n))))))
887
888 (defun stagger1 (board)
889   (move-to (any (empty-neighbors board)) board))
890
891 (deftest test-stagger ()
892   (reset-seed)
893   (let* ((board (readl))
894          (some (empty-neighbors board))
895          (one (any some)))
896     (stagger (readl))))
897

```

Jan 17, 10 11:25

code.txt

Page 20/35

```

898 # ./game/macros.lisp
899 #-----
900
901 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
902 ; This file is part of AIslash.
903 ;
904 ; AIslash is free software: you can redistribute it and/or modify
905 ; it under the terms of the GNU General Public License as published by
906 ; the Free Software Foundation, either version 3 of the License, or
907 ; (at your option) any later version.
908 ;
909 ; AIslash is distributed in the hope that it will be useful,
910 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
911 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
912 ; GNU General Public License for more details.
913 ;
914 ; You should have received a copy of the GNU General Public License
915 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
916 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
917
918 (defmacro piece-at (piece board)
919   `(aref (board-contents ,board)
920          (point-y ,piece)
921          (point-x ,piece)))
922
923 (defmacro walk ((i j cell endp board &optional out) &rest body)
924   (let ((imax (gensym))
925         (jmax (gensym))
926         (contents (gensym)))
927     `(let (,cell
928           (,imax (1+ (board-width ,board)))
929           (,jmax (1+ (board-height ,board)))
930           (,contents (board-contents ,board)))
931       (dotimes (,j ,jmax ,out)
932         (dotimes (,i ,imax)
933           (let ((endp (= ,i (1- ,imax))))
934             (setf ,cell (aref ,contents ,j ,i)
935                    ,@body))))))
936

```

Jan 17, 10 11:25

code.txt

Page 21/35

```

937 # ./game/types.lisp
938 #-----
939
940 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
941 ; This file is part of AIslash.
942 ;
943 ; AIslash is free software: you can redistribute it and/or modify
944 ; it under the terms of the GNU General Public License as published by
945 ; the Free Software Foundation, either version 3 of the License, or
946 ; (at your option) any later version.
947 ;
948 ; AIslash is distributed in the hope that it will be useful,
949 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
950 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
951 ; GNU General Public License for more details.
952 ;
953 ; You should have received a copy of the GNU General Public License
954 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
955 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
956
957 (defun make-grid ()
958   (make-array (list (fixed-height *f*) (fixed-width *f*))))
959
960 (defmethod aspiece ((x piece)) x)
961 (defmethod aspiece ((x symbol)) (char->piece x))
962
963 (defun atp (x)
964   (geta x (fixed-directions *f*)))
965
966 (defmethod piece->char ((x piece))
967   (first (rassoc (type-of x) (fixed-chars *f*))))
968 (defun char->type (x)
969   (geta x (fixed-chars *f*)))
970 (defun char->piece (x)
971   (let* ((type (char->type x))
972          (constructor (intern (string-upcase
973                                (format nil "make--a" type)))))
974     (unless type
975       (error "~a unknown" x))
976     (funcall constructor)))
977
978

```

Jan 17, 10 11:25

code.txt

Page 22/35

```

979 # ./game/printing.lisp
980 #-----
981
982 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
983 ; This file is part of AIslash.
984 ;
985 ; AIslash is free software: you can redistribute it and/or modify
986 ; it under the terms of the GNU General Public License as published by
987 ; the Free Software Foundation, either version 3 of the License, or
988 ; (at your option) any later version.
989 ;
990 ; AIslash is distributed in the hope that it will be useful,
991 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
992 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
993 ; GNU General Public License for more details.
994 ;
995 ; You should have received a copy of the GNU General Public License
996 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
997 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
998
999 (defun onstring (x)
1000   (with-output-to-string (str)
1001     (onstream x str)))
1002
1003 (defmethod onstream ((x blank) &optional (str *standard-output*))
1004   (princ " " str))
1005
1006 (defmethod onstream ((x piece) &optional (str *standard-output*))
1007   (princ (piece->char x) str))
1008
1009 (defmethod onstream ((x board) &optional (str *standard-output*))
1010   (let (sep
1011         (max (1- (fixed-width *f*))))
1012     (format str "~%")
1013     (walk (i j cell endrowp x)
1014           (princ (or sep "") str)
1015           (onstream cell str)
1016           (if endrowp (princ #\Newline str)))))
1017
1018 (defmethod onstream ((x player) &optional (str *standard-output*))
1019   (princ (player-direction x) str))
1020

```

Jan 17, 10 11:25

code.txt

Page 23/35

```

1021 # ./game/defstructs.lisp
1022 #-----
1023
1024 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1025 ; This file is part of AIslash.
1026 ;
1027 ; AIslash is free software: you can redistribute it and/or modify
1028 ; it under the terms of the GNU General Public License as published by
1029 ; the Free Software Foundation, either version 3 of the License, or
1030 ; (at your option) any later version.
1031 ;
1032 ; AIslash is distributed in the hope that it will be useful,
1033 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
1034 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
1035 ; GNU General Public License for more details.
1036 ;
1037 ; You should have received a copy of the GNU General Public License
1038 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
1039 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1040
1041
1042 (defstruct point
1043   "why isn't this a listp build in?"
1044   (x 0) (y 0))
1045
1046
1047 (defstruct board
1048   (contents (make-grid))
1049   g
1050   h
1051   width
1052   height
1053   gold
1054   at)
1055
1056 (defstruct thing)
1057
1058 (defstruct (player (:include thing))
1059   direction
1060   holding)
1061
1062 (defstruct (piece (:include thing)) (cost 0))
1063 (defstruct (blank (:include piece)))
1064 (defstruct (blocks (:include piece (cost (fixed-inf *f*)))) fixedp)
1065 (defstruct (tool (:include piece)))
1066
1067 (defstruct (axe (:include tool)))
1068 (defstruct (key (:include tool)))
1069 (defstruct (dynamite (:include tool)))
1070 (defstruct (gold (:include tool)))
1071
1072 (defstruct (bush (:include blocks)))
1073 (defstruct (door (:include blocks)))
1074 (defstruct (wall (:include blocks)))
1075 (defstruct (water (:include blocks (fixedp t)))
1076

```

Jan 17, 10 11:25

code.txt

Page 24/35

```

1077 # ./game/roaming.lisp
1078 #-----
1079
1080 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1081 ; This file is part of AIslash.
1082 ;
1083 ; AIslash is free software: you can redistribute it and/or modify
1084 ; it under the terms of the GNU General Public License as published by
1085 ; the Free Software Foundation, either version 3 of the License, or
1086 ; (at your option) any later version.
1087 ;
1088 ; AIslash is distributed in the hope that it will be useful,
1089 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
1090 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
1091 ; GNU General Public License for more details.
1092 ;
1093 ; You should have received a copy of the GNU General Public License
1094 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
1095 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1096
1097 (defmethod move-to ((new-pos point) board)
1098   (let* ((new (piece-at new-pos board))
1099          (old-pos (board-at board)))
1100     (setf (board-at board) new-pos)
1101     (rotatef
1102      (piece-at new-pos board)
1103      (piece-at old-pos board))))
1104
1105 (defmethod neighbors ((p point))
1106   (mapcar #'(lambda (q)
1107             (make-point :x (+ (point-x p) (second q))
1108                        :y (+ (point-y p) (third q))))
1109           (fixed-steps *f*)))
1110
1111 (defmethod neighbors ((b board))
1112   (neighbors (board-at b)))
1113
1114 (defun empty-neighbors (board)
1115   (let (out)
1116     (dolist (point (neighbors board) out)
1117       (unless (typep point 'blocks)
1118         (if (inbounds-p board point)
1119             (push point out))))))
1120
1121 (defun inbounds-p (board point)
1122   (let ((xmin 0) (xmax (board-width board))
1123         (ymin 0) (ymax (board-height board)))
1124     (and (<= xmin (point-x point) xmax)
1125          (<= ymin (point-y point) ymax))))
1126

```

Jan 17, 10 11:25

code.txt

Page 25/35

```

1127 # ./game/globals.lisp
1128 #-----
1129
1130 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1131 ; This file is part of AIslash.
1132 ;
1133 ; AIslash is free software: you can redistribute it and/or modify
1134 ; it under the terms of the GNU General Public License as published by
1135 ; the Free Software Foundation, either version 3 of the License, or
1136 ; (at your option) any later version.
1137 ;
1138 ; AIslash is distributed in the hope that it will be useful,
1139 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
1140 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
1141 ; GNU General Public License for more details.
1142 ;
1143 ; You should have received a copy of the GNU General Public License
1144 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
1145 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1146
1147 (defstruct fixed
1148   "fixed stuff"
1149   (inf most-positive-fixnum)
1150   (ninf (* -1 most-positive-fixnum))
1151   (version 0.01)
1152   (width 21) ; counting from zero
1153   (height 10) ; counting from zero
1154   (chars '( (#\a . axe)
1155             (#\k . key)
1156             (#\d . dynamite)
1157             (#\g . gold)
1158             (#\B . bush)
1159             (#\- . door)
1160             (#\ . blank)
1161             (#\* . wall)
1162             (#\ . blank)
1163             (#\~ . water)
1164             (#\^ . player)
1165             (#\> . player)
1166             (#\v . player)
1167             (#\< . player )))
1168   (steps '( (deltax deltax
1169             (#\^ 0 -1)
1170             (#\> 1 0)
1171             (#\v 0 1)
1172             (#\< -1 0))))
1173
1174 (defparameter *f* (make-fixed))
1175
1176 (defstruct wm "runtime working memory"
1177   (toolong 1000)
1178   board)
1179
1180 (defparameter *w* (make-wm))
1181
1182 (defun zap ()
1183   (setf *w* (make-wm)))
1184

```

Jan 17, 10 11:25

code.txt

Page 26/35

```

1185 # ./game/creation.lisp
1186 #-----
1187
1188 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1189 ; This file is part of AIslash.
1190 ;
1191 ; AIslash is free software: you can redistribute it and/or modify
1192 ; it under the terms of the GNU General Public License as published by
1193 ; the Free Software Foundation, either version 3 of the License, or
1194 ; (at your option) any later version.
1195 ;
1196 ; AIslash is distributed in the hope that it will be useful,
1197 ; but WITHOUT ANY WARRANTY; without even the implied warranty of
1198 ; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
1199 ; GNU General Public License for more details.
1200 ;
1201 ; You should have received a copy of the GNU General Public License
1202 ; along with AIslash. If not, see <http://www.gnu.org/licenses/>.
1203 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
1204
1205 (defun init-board (l)
1206   (let ((board (make-board))
1207         (height (length l))
1208         (width (length (first l)))
1209         (i -1) (j -1))
1210     (labels
1211       ((init-cell (char)
1212        (let ((cell (char->piece char))
1213              (pos (make-point :x i :y j)))
1214          (incf i)
1215          (if (gold-p cell)
1216              (setf (board-gold board) pos)
1217              (if (player-p cell)
1218                  (setf (board-at board) pos
1219                        (player-direction cell) char))
1220              cell))
1221        (init-row (row)
1222         (incf j)
1223         (setf i 0)
1224         (if (= width (length row))
1225             (mapcar #'init-cell row)
1226             (error "~a not of length ~a" row width)))
1227        (let* ((rows (mapcar #'init-row l))
1228               (contents (make-array (list height width)
1229                                     :initial-contents rows)))
1230          (setf (board-contents board) contents
1231                (board-width board) (1- width)
1232                (board-height board) (1- height))
1233          (if (board-ok-p board)
1234              board))))))
1235
1236 (defmethod board-ok-p ((x board))
1237   (unless (board-gold x)
1238     (error "missing the gold"))
1239   (unless (board-at x)
1240     (error "missing a player"))
1241   (unless (= (board-height x) (1- (fixed-height *f*)))
1242     (error "board height is ~a, not ~a"
1243            (board-height x)
1244            (1- (fixed-height *f*))))
1245   (unless (= (board-width x) (1- (fixed-width *f*)))
1246     (error "board width is ~a, not ~a"
1247            (board-width x)
1248            (1- (fixed-width *f*))))
1249   (walk (i j cell endp x)
1250         (unless (typep cell 'thing)
1251             (error "~a is not a piece" cell)))
1252   t)
1253
1254 (deftest read1 ()
1255   (let ((f "game/eg/1"))
1256     (init-board (file->linesOfChar f))))
1257
1258 (deftest show1 ()

```

Jan 17, 10 11:25

code.txt

Page 27/35

```

1259 (check (string-equal (onstring (readl))
1260 "
1261 ~~~~~
1262 ~~~~~
1263 ~ B B k ~
1264 ~ *** ** ~
1265 ~*-* v *-*~
1266 ~ ** ** ~
1267 ~ g ** d ** a ~
1268 ~ BB BB ~
1269 ~~~~~
1270 ~~~~~
1271 " )))
1272

```

Jan 17, 10 11:25

code.txt

Page 28/35

```

1273 # ./table/header.lisp
1274 #-----
1275
1276 ;;;; columns isa list of header
1277
1278 (defun columns-new (cols klass)
1279 (let (tmp)
1280 (doitems (col i cols)
1281 (let ((new
1282 (if (numericp col)
1283 (make-numeric :name col :ignorep (ignorep col)
1284 :classp (= i klass))
1285 (make-discrete :name col :ignorep (ignorep col)
1286 :classp (= i klass))))))
1287 (push new tmp)))
1288 (reverse tmp)))
1289
1290 (defun columns-header (cols)
1291 (mapcar #'header-name cols))
1292
1293 (defun numericp (x)
1294 (equal (char (symbol-name x) 0) #\$))
1295
1296 (defun ignorep (x)
1297 (and (symbolp x)
1298 (equal (char (symbol-name x) 0) #\?)))
1299
1300 (defun unknownp (x)
1301 (ignorep x))
1302

```

Jan 17, 10 11:25

code.txt

Page 29/35

```

1303 # ./table/structs.lisp
1304 #-----
1305
1306 (defstruct table
1307   name           ; symbol       : name of the table
1308   columns        ; list of header : one header foreach column
1309   class          ; number       : which column is the header?
1310   (cautions (make-caution)) ; list of caution : any load-time errors?
1311   all          ; list of eg     : all the examples
1312   indexed
1313 )
1314
1315 (defstruct eg features class)
1316
1317 (defstruct header name classp ignorep
1318   (f (make-hash-table :test #'equal)))
1319 (defstruct (numeric (:include header)))
1320 (defstruct (discrete (:include header)) uniques)
1321

```

Jan 17, 10 11:25

code.txt

Page 30/35

```

1322 # ./table/table.lisp
1323 #-----
1324
1325 ;;; a table stores headers and examples
1326
1327 (defun isa (row tbl) (nth (table-class tbl) row))
1328
1329
1330 (defun table-width (tbl)
1331   (length (table-columns tbl)))
1332
1333 (defun table-copy (tbl &optional (new (copy-tree (table-all tbl))))
1334   (data :egs new
1335         :name (table-name tbl)
1336         :klass (table-class tbl)
1337         :columns (columns-header (table-columns tbl))))
1338
1339 (defun classes (tbl)
1340   (discrete-uniques (table-class-header tbl)))
1341
1342 (defun nclasses (tbl)
1343   (1+ (length (classes tbl))))
1344
1345 (defun eggs (tbl)
1346   (table-all tbl))
1347
1348 (defun negs (tbl)
1349   (1+ (length (table-all tbl))))
1350
1351 (defun table-class-header (tbl)
1352   (nth (table-class tbl) (table-columns tbl)))
1353

```

Jan 17, 10 11:25

code.txt

Page 31/35

```

1354 # ./table/data.lisp
1355 #-----
1356
1357 ;;;; the data function returns a table containing some eggs
1358
1359
1360 (defun data (&key name columns eggs (klass -1))
1361   (let* (tmp-egs
1362         (tbl
1363          (make-table
1364           :name name
1365           :columns (columns-new
1366                    columns
1367                    (class-index klass (length columns))))))
1368     (setf (table-class tbl)
1369           (class-index klass (table-width tbl)))
1370     (dolist (eg eggs)
1371       (if (setf eg (datums eg tbl))
1372         (push eg tmp-egs)))
1373     tbl))
1374
1375 (defun datums (one tbl)
1376   (let ((oops (table-cautions tbl)))
1377     (when
1378      (ok (= (table-width tbl) (length one))
1379          oops "~a wrong size" one)
1380       (mapc #'(lambda (column datum)
1381                (datum column datum oops))
1382            (table-columns tbl)
1383            one)
1384       (push (make-eg :class (isa one tbl) :features one)
1385            (table-all tbl))))))
1386
1387 (defmethod datum ((column discrete) datum oops)
1388   "things to do when reading a discrete datum"
1389   (declare (ignore oops))
1390   (unless (member datum (discrete-uniques column))
1391     (push datum (discrete-uniques column)))
1392   t)
1393
1394 (defmethod datum ((column numeric) datum oops)
1395   "things to do when reading a numeric datum"
1396   (ok (numberp datum) oops "~a is not a number" datum)
1397   t)
1398
1399 (defun class-index (klass width)
1400   (if (< klass 0) (+ klass width) klass))
1401
1402 (defun make-datal ()
1403   (data
1404    :name 'weather
1405    :columns '(forecast temp humidty $windy play)
1406    :egs '((sunny hot high FALSE no)
1407          (sunny hot high TRUE yes)
1408          (sunny hot high yes)
1409          )))
1410
1411 (deftest test-table ()
1412   (check
1413    (samep
1414     (make-datal)
1415     "#S(TABLE
1416      :NAME WEATHER
1417      :COLUMNS (#S(DISCRETE
1418                   :NAME FORECAST
1419                   :CLASSP NIL
1420                   :IGNOREP NIL
1421                   :F #<HT 0>
1422                   :UNIQUES (SUNNY))
1423                #S(DISCRETE
1424                   :NAME TEMP
1425                   :CLASSP NIL
1426                   :IGNOREP NIL
1427                   :F #<HT 0>

```

Jan 17, 10 11:25

code.txt

Page 32/35

```

1428         :UNIQUES (HOT))
1429         #S(DISCRETE
1430            :NAME HUMIDTY
1431            :CLASSP NIL
1432            :IGNOREP NIL
1433            :F #<HT 0>
1434            :UNIQUES (HIGH))
1435         #S(NUMERIC :NAME $WINDY :CLASSP NIL :IGNOREP NIL :F #<HT 0>)
1436         #S(DISCRETE
1437            :NAME PLAY
1438            :CLASSP T
1439            :IGNOREP NIL
1440            :F #<HT 0>
1441            :UNIQUES (YES NO)))
1442   :CLASS 4
1443   :CAUTIONS #(:CAUTION :ALL ((SUNNY HOT HIGH YES) wrong size
1444                    TRUE is not a number
1445                    FALSE is not a number) :PATIENCE 17)
1446   :ALL (#S(EG :FEATURES (SUNNY HOT HIGH TRUE YES) :CLASS YES)
1447        #S(EG :FEATURES (SUNNY HOT HIGH FALSE NO) :CLASS NO))
1448   :INDEXED NIL)
1449 "
1450 )))
1451
1452

```

Jan 17, 10 11:25

code.txt

Page 33/35

```

1453 # ./table/xindex.lisp
1454 #-----
1455
1456
1457 ;;; xindex runs over the data. populates the "counts" of each column header.
1458 ;; genrate the indexes
1459 (defun xindex (tbl)
1460   (unless (table-indexed tbl)
1461     (setf (table-indexed tbl) t)
1462     (dolist (row (table-all tbl) tbl) ; for al rows do ...
1463       (xindex1 (eg-class row)
1464                 (eg-features row)
1465                 (table-columns tbl))))
1466   tbl)
1467
1468 (defun xindex1 (class datums columns) ; for all datum in a row do ...
1469   (mapc #'(lambda (column datum)
1470            (unless (ignorep datum)
1471              (xindex-datum column class datum)))
1472         columns
1473         datums))
1474
1475 (defmethod xindex-datum ((column discrete) class datum)
1476   (let* ((key `(,class ,datum))
1477          (hash (header-f column)))
1478     (incf (gethash key hash 0)))
1479
1480 (defmethod xindex-datum ((column numeric) class datum)
1481   (let* ((key class)
1482          (hash (header-f column))
1483          (counter (gethash key hash (make-normal))))
1484     (setf (gethash key hash) counter) ; make sure the hash has the counter
1485     (add counter datum))
1486
1487 (defun make-data2 ()
1488   (data
1489    :name 'weather
1490    :columns '(forecast temp humidty windy play)
1491    :egs '((sunny hot high FALSE no)
1492           (sunny hot high TRUE no)
1493           (rainy cool normal TRUE no)
1494           (rainy mild high TRUE no)
1495           (sunny mild high FALSE no)
1496           (overcast cool normal TRUE yes)
1497           (overcast hot high FALSE yes)
1498           (rainy mild high FALSE yes)
1499           (rainy cool normal FALSE yes)
1500           (sunny cool normal FALSE yes)
1501           (rainy mild normal FALSE yes)
1502           (sunny mild normal TRUE yes)
1503           (overcast mild high TRUE yes)
1504           (overcast hot normal FALSE yes)
1505   )))
1506
1507 (defun make-data3 ()
1508   (data
1509    :name 'weather
1510    :columns '(forecast temp humidty $wind play)
1511    :egs '((sunny hot high 20 no)
1512           (sunny hot high 10 no)
1513           (sunny hot high 30 no)
1514           (sunny hot high 20.2 yes)
1515           (sunny hot high 20.1 yes)
1516           (sunny hot high 20.7 yes)
1517   )))
1518
1519 (deftest test-xindex2 ()
1520   (check
1521    (samep
1522     (with-output-to-string (str)
1523       (dolist (col (table-columns (xindex (make-data2))))
1524         (format str "~%-a~%" (header-name col))
1525         (showh (header-f col) :stream str)))
1526     "FORECAST

```

Jan 17, 10 11:25

code.txt

Page 34/35

```

1527         (NO RAINY) = 2
1528         (NO SUNNY) = 3
1529         (YES OVERCAST) = 4
1530         (YES RAINY) = 3
1531         (YES SUNNY) = 2
1532
1533     TEMP
1534         (NO COOL) = 1
1535         (NO HOT) = 2
1536         (NO MILD) = 2
1537         (YES COOL) = 3
1538         (YES HOT) = 2
1539         (YES MILD) = 4
1540
1541     HUMIDTY
1542         (NO HIGH) = 4
1543         (NO NORMAL) = 1
1544         (YES HIGH) = 3
1545         (YES NORMAL) = 6
1546
1547     WINDY
1548         (NO FALSE) = 2
1549         (NO TRUE) = 3
1550         (YES FALSE) = 6
1551         (YES TRUE) = 3
1552
1553     PLAY
1554         (NO NO) = 5
1555         (YES YES) = 9
1556         ")))
1557
1558 (deftest test-xindex3 ()
1559   (check
1560    (samep
1561     (with-output-to-string (str)
1562       (dolist (col (table-columns (xindex (make-data3))))
1563         (format str "~%-a~%" (header-name col))
1564         (showh (header-f col) :stream str)))
1565     "FORECAST
1566         (NO SUNNY) = 3
1567         (YES SUNNY) = 3
1568
1569     TEMP
1570         (NO HOT) = 3
1571         (YES HOT) = 3
1572
1573     HUMIDTY
1574         (NO HIGH) = 3
1575         (YES HIGH) = 3
1576
1577     $WIND
1578     NO = #S(NORMAL :MAX 30 :MIN 10 :N 3 :SUM 60 :SUMSQ 1400)
1579     YES = #S(NORMAL :MAX 20.7 :MIN 20.1 :N 3 :SUM 61.000004 :SUMSQ 1240.54)
1580
1581     PLAY
1582         (NO NO) = 3
1583         (YES YES) = 3
1584         ")))
1585
1586 (defun make-some-weather-data ()
1587   (data
1588    :name 'weather
1589    :columns '(forecast $temp $humidty wind play)
1590    :egs '((sunny 85 85 FALSE no)
1591           (sunny 80 90 TRUE no)
1592           (overcast 83 86 FALSE yes)
1593           (rainy 70 96 FALSE yes)
1594           (rainy 68 80 FALSE yes)
1595           (rainy 65 70 TRUE no)
1596           (overcast 64 65 TRUE yes)
1597           (sunny 72 95 FALSE no)
1598           (sunny 69 70 FALSE yes)
1599           (rainy 75 80 FALSE yes)
1600           (sunny 75 70 TRUE yes)

```

Jan 17, 10 11:25

code.txt

Page 35/35

```

1601      (overcast 72 90 TRUE yes)
1602      (overcast 81 75 FALSE yes)
1603      (rainy 71 91 TRUE no)))
1604
1605 (deftest test-some-counts ()
1606   (check
1607     (samep
1608       (with-output-to-string (str)
1609         (dolist (col (table-columns (xindex (make-some-weather-data))))
1610           (format str "~%-a~%" (header-name col))
1611           (showh (header-f col) :indent 10 :stream str)))
1612     "
1613     FORECAST
1614       (NO RAINY) = 2
1615       (NO SUNNY) = 3
1616       (YES OVERCAST) = 4
1617       (YES RAINY) = 3
1618       (YES SUNNY) = 2
1619
1620     $TEMP
1621       NO = #S(NORMAL :MAX 85 :MIN 65 :N 5 :SUM 373 :SUMSQ 28075)
1622       YES = #S(NORMAL :MAX 83 :MIN 64 :N 9 :SUM 657 :SUMSQ 48265)
1623
1624     $HUMIDTY
1625       NO = #S(NORMAL :MAX 95 :MIN 70 :N 5 :SUM 431 :SUMSQ 37531)
1626       YES = #S(NORMAL :MAX 96 :MIN 65 :N 9 :SUM 712 :SUMSQ 57162)
1627
1628     WIND
1629       (NO FALSE) = 2
1630       (NO TRUE) = 3
1631       (YES FALSE) = 6
1632       (YES TRUE) = 3
1633
1634     PLAY
1635       (NO NO) = 5
1636       (YES YES) = 9" )))
1637
1638 ;; e.g. query the structures
1639
1640 (defun f (tbl &optional class index range)
1641   (cond ((null class) ; return number of instances
1642         (length (table-all tbl)))
1643         ((null index) ; return number of a certain class
1644         (fl (nth (table-class tbl) (table-columns tbl))
1645             class
1646             class))
1647         (t ; return frequency of a range in a class
1648         (fl (nth index (table-columns tbl))
1649             class
1650             range))))
1651
1652 (defmethod fl ((column discrete) class range)
1653   (gethash `(',class ,range) (header-f column) 0))
1654
1655
1656 (deftest test-f ()
1657   (let ((tbl (xindex (make-data3))))
1658     (check
1659       (samep 6 (f tbl))
1660       (samep 3 (f tbl 'yes))
1661       (samep 3 (f tbl 'yes 1 'hot))
1662     )))
1663

```