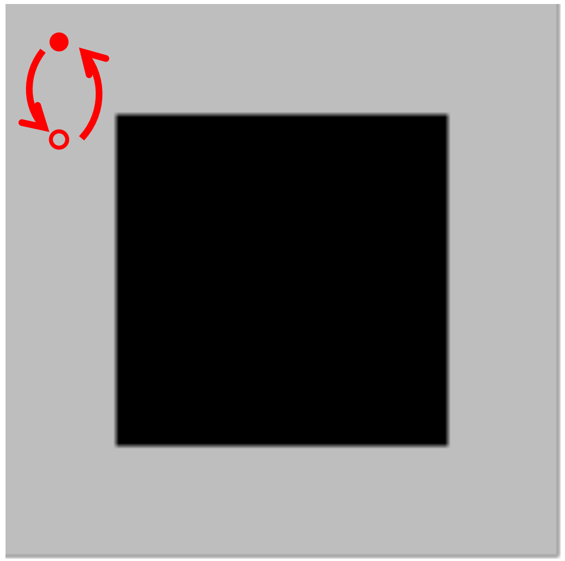
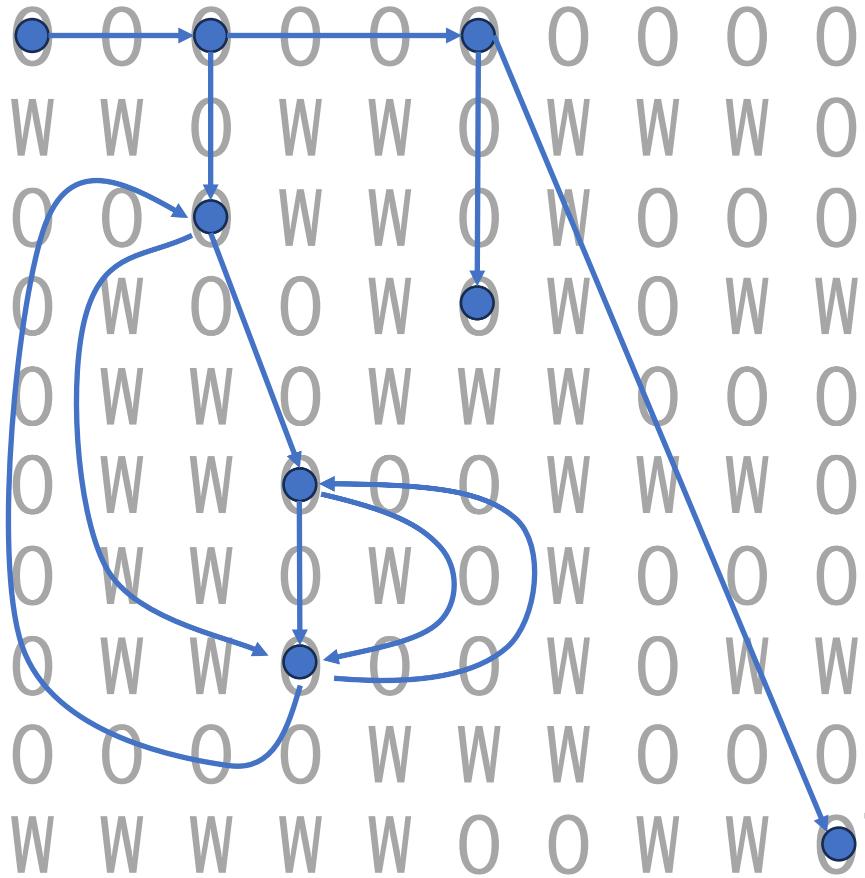
Lecture 21
mazes and graphs



```
(define M7
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W W W O O O
        O W W O O O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

The fine grained maze structure is pretty dense. Not complex really, but just dense.

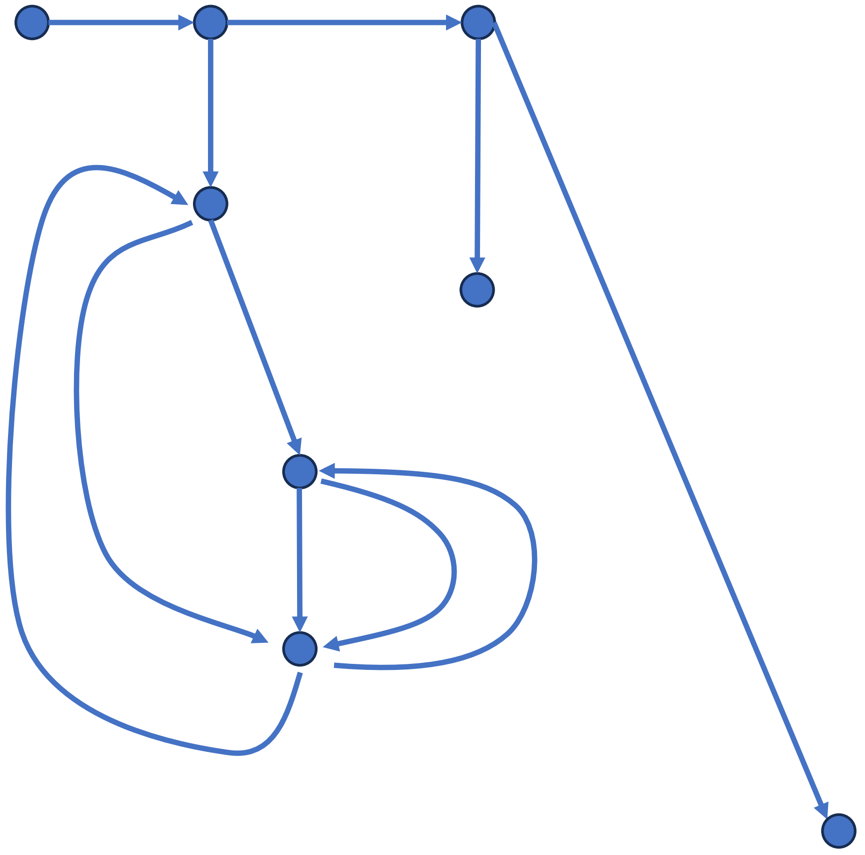
But we can simplify…



```
(define M7
  (list O O O O O O O O O O
        W W O W W O W W W O
        O O O W W O W O O O
        O W O O W O W O W W
        O W W O W W W O O O
        O W W O O O W W W O
        O W W O W O W O O O
        O W W O O O W O W W
        O O O O W W W O O O
        W W W W W O O W W O))
```

The fine grained maze structure is pretty dense. Not complex really, but just dense.

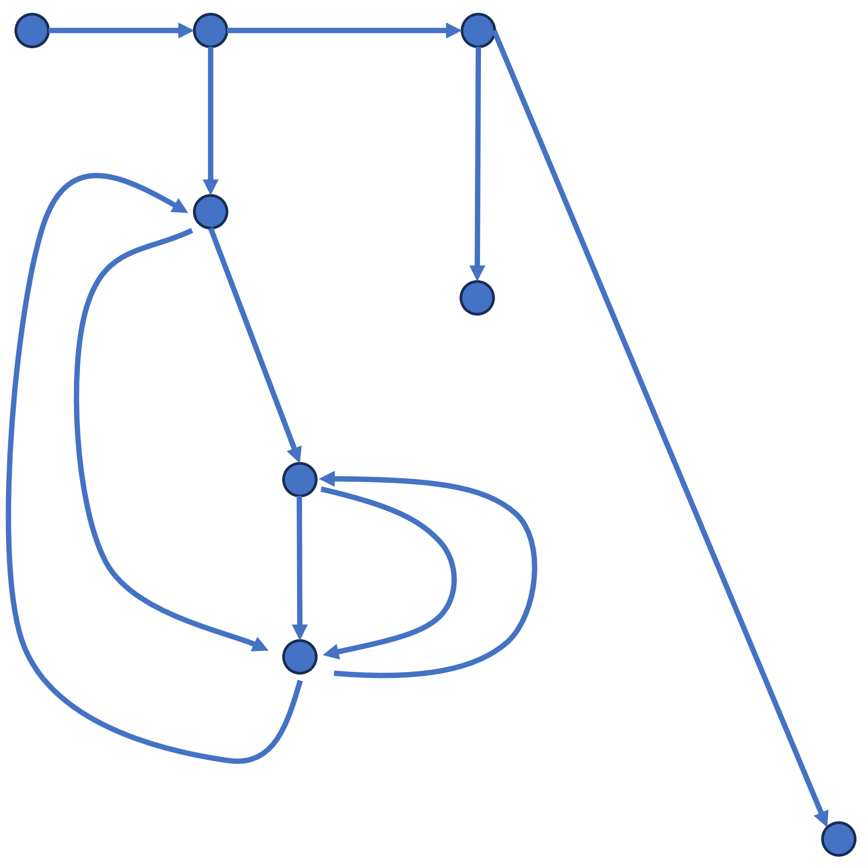
But we can see that among the dense back and forth, some points are more interesting.



The fine grained maze structure is pretty dense. Not complex really, but just dense.

But we can see that among the dense back and forth, some points are more interesting.

Clicker time!



solvable-no-revisits?

graph: yes → path or visited required

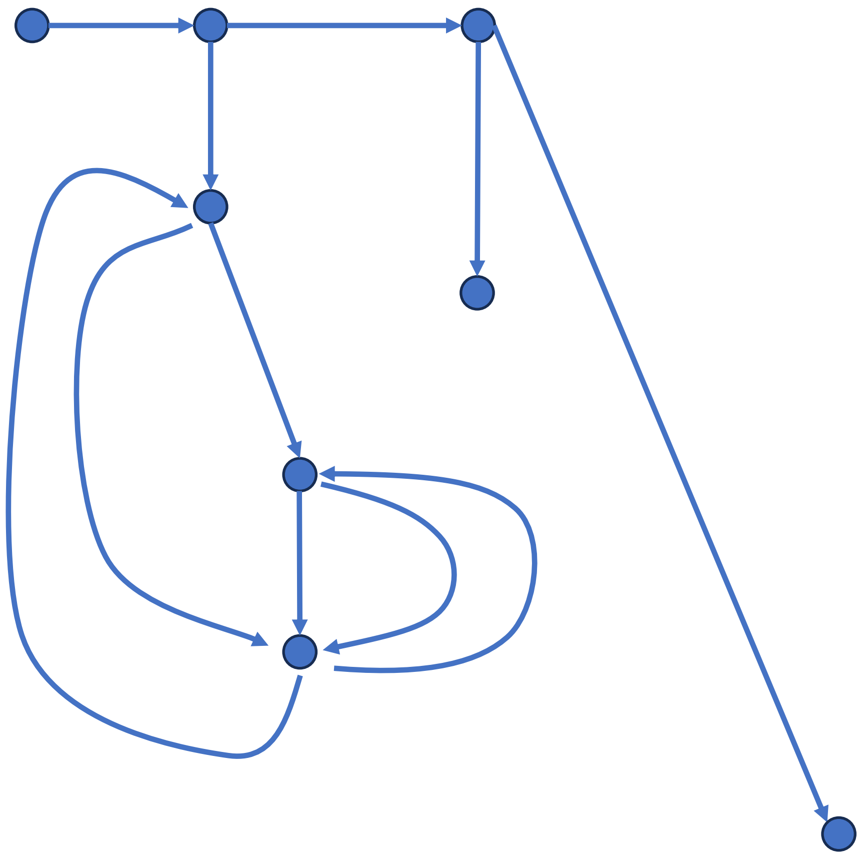
…

```
(define (solvable-no-revisits? m)
  (local [(define R (sqrt (length m)))

          ;; trivial:
          ;; reduction:
          ;; argument:

          (define (fn-for-p p p-wl visited)
            (cond [(solved? p) true]
                  [(member? p visited) (fn-for-lop p-wl visited)]
                  [else
                   (fn-for-lop (append (next-ps p) p-wl) (cons p visited))]))

          (define (fn-for-lop p-wl visited)
            (cond [(empty? p-wl) false]
                  [else
                   (fn-for-p (first p-wl) (rest p-wl) visited)]))
```



distance-from

graph: yes → path or visited required

…

```
(define (distance-from m start end)
  (local [(define R (sqrt (length m)))

          ;; trivial:
          ;; reduction:
          ;; argument:

          (define (fn-for-p p path dist)
            (cond [(equal? p end)   (distance-add1 dist)]
                 ;[(solved? p)      false]
                  [(member? p path) false]
                  [else
                   (if (equal? p start)
                       (fn-for-lop (next-ps p) (cons p path) 0)
                       (fn-for-lop (next-ps p)
                                   (cons p path)
                                   (distance-add1 dist)))]))

          (define (fn-for-lop lop path dist)
            (cond [(empty? lop) false]
                  [else
                   (local [(define try (fn-for-p (first lop) path dist))]
                     (if (not (false? try))
                         try
                         (fn-for-lop (rest lop) path dist)))]))
```