CSC466 Project Task 6

Initial Heuristic Player

Abstract

In this project I created the basic structure of rules for my heuristic player. The structure consists of a list of the rule functions, and a function which evaluates the rules in order of significance. Each rule has a special return value if the rule is not applicable so it can be removed from this list and the next rule can be checked. The only rule I implemented in this task was revealing the corner tiles before randomly revealing tiles. I then ran many games with the random player and the heuristic player and compared the results.

Demo

```
[1]> ( load "demos.l" )

;; Loading file demos.l ...

;;  Loading file hms.l ...

;;   Loading file ms.l ...

WARNING: Replacing method #<STANDARD-METHOD (#<STANDARD-CLASS
TILE>)> in #<STANDARD-GENERIC-FUNCTION TILE-INFO>

WARNING: The generic function #<STANDARD-GENERIC-FUNCTION
ADJACENT-TILES> is being modified, but has already been called.

WARNING: Replacing method #<STANDARD-METHOD (#<STANDARD-CLASS
TILE> #<BUILT-IN-CLASS LIST> #<BUILT-IN-CLASS LIST>)> in

         #<STANDARD-GENERIC-FUNCTION ADJACENT-TILES>

;;   Loaded file ms.l

;;  Loaded file hms.l

;; Loaded file demos.l

T
```

```
[2]> ( demo--heuristic-game )

>>> Testing heuristic game player

Playing game on easy with display option


       A B C D E F G H I J

     +---------------------+

   0 | 0 0 0 0 0 1 # 1 0 0 |

   1 | 0 0 0 0 0 1 X 1 0 0 |

   2 | 0 1 1 1 0 1 1 1 0 0 |

   3 | 0 2 X 2 0 0 1 1 1 0 |

   4 | 0 2 X 2 0 0 1 X 1 0 |

   5 | 0 1 1 2 1 1 1 1 2 1 |

   6 | 0 0 0 1 X # # # # X |

   7 | 0 0 1 2 # # # # 2 # |

   8 | 0 0 1 X # X 2 # # X |

   9 | 0 0 1 # # # # X 2 1 |

     +---------------------+


Playing game on easy with stats option

(84 100)


Playing 100 games on easy

Average number of tiles revealed: 61

Number of wins: 8
```

```
Playing 100 games on medium

Average number of tiles revealed: 53

Number of wins: 0


Playing 100 games on hard

Average number of tiles revealed: 25

Number of wins: 0

NIL

[3]>
```

Statistics

	100 Games		1000 Games		10000 Games	
	Avg Rev	Wins	Avg Rev	Wins	Avg Rev	Wins
Random	58	2	56	25	55	382
Heuristic	66	10	63	45	63	524

Code

```lisp
( defun play-heuristic-game ( &optional mode difficulty &aux move )
 ( cond
        ( ( or ( null difficulty ) ( equal difficulty 'easy ) )
            ( generate-board 10 10 10 ) )
        ( ( equal difficulty 'medium ) ( generate-board 16 16 40 ) )
        ( ( equal difficulty 'hard ) ( generate-board 24 20 99 ) )
    )
    ( loop
        ( if ( or ( win-p ) ( heuristic-move ) )
            ( cond
                ( ( equal mode 'display ) ( display-board ) ( return nil ) )
                ( ( equal mode 'stats )
                    ( return ( list ( length ( board-revealed-tiles *board* ) )
                         ( length ( board-tiles *board* ) ) ) ) )
                )
                ( t ( return nil ) )
            )
        )
))
( defun play-n-heuristic( n &optional difficulty &aux result revealed total wins)
    ( setf revealed 0 )
    ( setf total 0 )
    ( setf wins 0 )
    ( dotimes ( i n )
        ( setf result ( play-heuristic-game 'stats difficulty ) )
        ( setf revealed ( + revealed ( car result ) ) )
        ( setf total ( + total ( cadr result ) ) )
        ( if ( win-p ) ( setf wins ( + 1 wins ) ) )
    )
    ( format t "Average number of tiles revealed: ~a~%Number of wins: ~a~%"
        ( floor revealed n ) wins )
)
( defun heuristic-move ( &aux li rule res )
    ( setf li ( rules ) )
    ( loop
        ( setf rule ( pop li ) )
        ( if ( equal rule nil ) ( return nil ) )
        ( setf res ( funcall rule ) )
        ( cond
            ( ( equal res 'na ) ( continue ) )
            ( t ( return res ) )
        )
))
```

```
( defun rules ()
    ( list
        #'corner-r
        #'reveal-random
    )
)
( defun corner-r ( &aux tl tr bl br )
    ( setf tl ( nth 0 ( board-tiles *board* ) ) )
    ( setf tr ( nth ( - ( board-width *board* ) 1 ) ( board-tiles *board* ) ) )
    ( setf bl ( nth ( * ( board-width *board* ) ( - ( board-height *board* ) 1 )
) ( board-tiles *board* ) ) )
    ( setf br ( nth ( - ( length ( board-tiles *board* ) ) 1 ) ( board-tiles
*board* ) ) )
    ( cond
        ( ( not ( tile-revealed tl ) ) ( reveal-tiles ( list ( nth ( tile-name tl
) ( board-tiles *board* ) ) ) '() ) )
        ( ( not ( tile-revealed tr ) ) ( reveal-tiles ( list ( nth ( tile-name tr
) ( board-tiles *board* ) ) ) '() ) )
        ( ( not ( tile-revealed bl ) ) ( reveal-tiles ( list ( nth ( tile-name bl
) ( board-tiles *board* ) ) ) '() ) )
        ( ( not ( tile-revealed br ) ) ( reveal-tiles ( list ( nth ( tile-name br
) ( board-tiles *board* ) ) ) '() ) )
        ( t 'na )
    )
)
```