pacsim

Class PacUtils

java.lang.Object
    pacsim.PacUtils

```
public class PacUtils
extends java.lang.Object
```

Multi-modal AI Simulator Utilities

Constructor Summary

Constructors

Constructor and Description
PacUtils()

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method and Description
static **PacFace**	**avoidTarget**(java.awt.Point p, java.awt.Point t, **PacCell**[][] cell) Choose an available direction that maximizes the distance from a given target
static double	**euclideanDistance**(int x1, int y1, int x2, int y2) Compute the Euclidean distance between two points
static double	**euclideanDistance**(java.awt.Point p1, java.awt.Point p2) Compute the Euclidean distance between two points
static **PacFace**	**euclideanShortestToTarget**(java.awt.Point curr, **PacFace** face, java.awt.Point target, **PacCell**[][] cell) Chose the available direction that most closely approaches a

	target, using the Euclidean distance measure, but not the opposite of the current direction NOTE: This method returns null if the only option is to reverse.
static java.util.List<java.awt.Point>	**findGhosts**(**PacCell**[][] state) Find all the ghosts on the current board
static **PacmanCell**	**findPacman**(**PacCell**[][] state) Find Pac-Man if he is on the board (for simulation experiments)
static **StartCell**	**findStart**(**PacCell**[][] state) Find the start cell, if any (for search problems)
static boolean	**foodRemains**(**PacCell**[][] state) Determine whether any food remains on the board
static boolean	**goody**(int x, int y, **PacCell**[][] c) Determine whether the current cell contains either food or a power pellet
static int	**manhattanDistance**(int x1, int y1, int x2, int y2) Compute the Manhattan distance between two point locations
static int	**manhattanDistance**(java.awt.Point p1, java.awt.Point p2) Compute the Manhattan distance between two point locations
static **PacFace**	**manhattanShortestToTarget**(java.awt.Point curr, **PacFace** face, java.awt.Point target, **PacCell**[][] cell) Chose the available direction that most closely approaches a target, using the Manhattan distance measure
static **GhostCell**	**nearestGhost**(java.awt.Point p, **PacCell**[][] cell) Find the nearest ghost, if any
static java.awt.Point	**nearestGoody**(java.awt.Point p, **PacCell**[][] cell) Find the nearest food or power pellet cell
static java.awt.Point	**nearestGoodyButNot**(java.awt.Point p, java.awt.Point tgt, **PacCell**[][] cell) Find the nearest food or power pellet cell, but not a particular goody
static java.awt.Point	**nearestUnoccupied**(java.awt.Point p, **PacCell**[][] cell)

	Find the nearest unoccupied cell
static **PacCell**	**neighbor**(**PacFace** face, **PacCell** pc, **PacCell**[][] cell) Find the immediate neighbor of a given cell in a particular direction
static boolean	**oppositeFaces**(**PacFace** a, **PacFace** b) Determine whether two facing directions are opposites
static **PacFace**	**randomNotReverse**(java.awt.Point curr, **PacFace** face, java.awt.Point target, **PacCell**[][] cell) Choose a random available direction but not the opposite of the current direction
static **PacFace**	**randomOpenForGhost**(java.awt.Point curr, **PacCell**[][] cell) Choose a random direction where the next cell is not a ghost, wall, or Pac-Man
static **PacFace**	**randomOpenForPacman**(java.awt.Point curr, **PacCell**[][] cell) Choose a random facing direction that is not in the direction of a ghost, house, or wall cell
static **PacFace**	**reverse**(**PacFace** face) Find the opposite facing direction
static boolean	**unoccupied**(int x, int y, **PacCell**[][] c) Determine whether a particular cell is unoccupied

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PacUtils

```
public PacUtils()
```

Method Detail

findStart

```
public static StartCell findStart(PacCell[][] state)
```

Find the start cell, if any (for search problems)

Parameters:

`state - the cell array to examine`

Returns:

`the Start Cell, if any`

findPacman

```
public static PacmanCell findPacman(PacCell[][] state)
```

Find Pac-Man if he is on the board (for simulation experiments)

Parameters:

`state - the cell array to examine`

Returns:

`the Pac-Man cell, if any`

findGhosts

```
public static java.util.List<java.awt.Point> findGhosts(PacCell[][] state)
```

Find all the ghosts on the current board

Parameters:

`state - the cell array to examine`

Returns:

`a list containing the ghost cells, if any`

foodRemains

```
public static boolean foodRemains(PacCell[][] state)
```

Determine whether any food remains on the board

Parameters:

`state - the cell array to examine`

Returns:

`T/F`

neighbor

```
public static PacCell neighbor(PacFace face,
                               PacCell pc,
                               PacCell[][] cell)
```

Find the immediate neighbor of a given cell in a particular direction

Parameters:

`face - the current direction`

`pc - the current cell`

`cell - the cell array to examine`

Returns:

`the immediate neighbor of the cell in the input direction, if any`

manhattanDistance

```
public static int manhattanDistance(java.awt.Point p1,
                                    java.awt.Point p2)
```

Compute the Manhattan distance between two point locations

Parameters:

`p1 - the first point`

`p2 - the second point`

Returns:

`non-negative integer distance`

manhattanDistance

```
public static int manhattanDistance(int x1,
                                    int y1,
                                    int x2,
                                    int y2)
```

Compute the Manhattan distance between two point locations

Parameters:

`x1 - x-coordinate of first point`

`y1 - y-coordinate of first point`

`x2 - x-coordinate of second point`

`y2 - y-coordinate of second point`

Returns:

non-negative integer distance

manhattanShortestToTarget

```
public static PacFace manhattanShortestToTarget(java.awt.Point curr,
                                                PacFace face,
                                                java.awt.Point target,
                                                PacCell[][] cell)
```

Chose the available direction that most closely approaches a target, using the Manhattan distance measure

Parameters:

curr - the current location

face - the current facing direction

target - the target location

cell - the cell array to examine

Returns:

a facing direction

euclideanDistance

```
public static double euclideanDistance(java.awt.Point p1,
                                       java.awt.Point p2)
```

Compute the Euclidean distance between two points

Parameters:

p1 - the first point

p2 - the second point

Returns:

a real-valued distance

euclideanDistance

```
public static double euclideanDistance(int x1,
                                       int y1,
                                       int x2,
                                       int y2)
```

Compute the Euclidean distance between two points

Parameters:

x1 - x-coordinate of first point

y1 - y-coordinate of first point

x2 - x-coordinate of second point

y2 - y-coordinate of second point

Returns:

a real-valued distance

euclideanShortestToTarget

```
public static PacFace euclideanShortestToTarget(java.awt.Point curr,
                                                PacFace face,
                                                java.awt.Point target,
                                                PacCell[][] cell)
```

Chose the available direction that most closely approaches a target, using the Euclidean distance measure, but not the opposite of the current direction NOTE: This method returns null if the only option is to reverse. In such case, it is usually best to reverse direction and then call this method again.

Parameters:

curr - the current location

face - the current facing direction

target - the target location

cell - the cell array to examine

Returns:

a facing direction

avoidTarget

```
public static PacFace avoidTarget(java.awt.Point p,
                                  java.awt.Point t,
                                  PacCell[][] cell)
```

Choose an available direction that maximizes the distance from a given target

Parameters:

p - the current location

t - the target location

cell - the cell array to examine

Returns:

a facing direction

randomNotReverse

```
public static PacFace randomNotReverse(java.awt.Point curr,
                                       PacFace face,
                                       java.awt.Point target,
                                       PacCell[][] cell)
```

Choose a random available direction but not the opposite of the current direction

Parameters:

`curr - the current cell location`

`face - the current facing direction`

`target - this parameter is not used`

`cell - the cell array to examine`

Returns:

`a facing direction`

randomOpenForPacman

```
public static PacFace randomOpenForPacman(java.awt.Point curr,
                                          PacCell[][] cell)
```

Choose a random facing direction that is not in the direction of a ghost, house, or wall cell

Parameters:

`curr - the current cell location`

`cell - the cell array to examine`

Returns:

`a facing direction`

randomOpenForGhost

```
public static PacFace randomOpenForGhost(java.awt.Point curr,
                                         PacCell[][] cell)
```

Choose a random direction where the next cell is not a ghost, wall, or Pac-Man

Parameters:

`curr - the current location`

`cell - the cell array to examine`

Returns:

`a facing direction`

nearestGoody

```
public static java.awt.Point nearestGoody(java.awt.Point p,
                                           PacCell[][] cell)
```

Find the nearest food or power pellet cell

Parameters:

`p - the current location`

`cell - the cell array to examine`

Returns:

`the location of the nearest goody`

nearestGoodyButNot

```
public static java.awt.Point nearestGoodyButNot(java.awt.Point p,
                                                 java.awt.Point tgt,
                                                 PacCell[][] cell)
```

Find the nearest food or power pellet cell, but not a particular goody

Parameters:

`p - the current location`

`tgt - the goody to avoid`

`cell - the cell array to examine`

Returns:

`the location of the nearest goody`

goody

```
public static boolean goody(int x,
                            int y,
                            PacCell[][] c)
```

Determine whether the current cell contains either food or a power pellet

Parameters:

`x - the x-coordinate of the current cell`

`y - the y-coordinate of the current cell`

`c - the cell array to examine`

Returns:

`T/F`

nearestGhost

```
public static GhostCell nearestGhost(java.awt.Point p,
                                     PacCell[][] cell)
```

Find the nearest ghost, if any

Parameters:

`p - the current location`

`cell - the cell array to examine`

Returns:

`the nearest ghost`

nearestUnoccupied

```
public static java.awt.Point nearestUnoccupied(java.awt.Point p,
                                               PacCell[][] cell)
```

Find the nearest unoccupied cell

Parameters:

`p - the current cell location`

`cell - the cell array to examine`

Returns:

`the nearest unoccupied cell`

unoccupied

```
public static boolean unoccupied(int x,
                                 int y,
                                 PacCell[][] c)
```

Determine whether a particular cell is unoccupied

Parameters:

`x - the x-coordinate of the input cell`

`y - the y-coordinate of the input cell`

`c - the input cell array`

Returns:

`T/F`

oppositeFaces

```
public static boolean oppositeFaces(PacFace a,
                                    PacFace b)
```

Determine whether two facing directions are opposites

Parameters:

`a - the first facing direction`

`b - the second facing direction`

Returns:

`T/F`

reverse

```
public static PacFace reverse(PacFace face)
```

Find the opposite facing direction

Parameters:

`face - the input facing direction`

Returns:

`the opposite direction of face`