Métodos de Busca no Pacman

Generated by Doxygen 1.8.13

Contents

Chapter 1

README

Trabalho 1. Métodos de Busca no PacMan

> **Notice to mariners:** Based on Pacman search problems originally from `Berkeley AI CS 188`. Original code has been ported to Python 3.x.

Introdução

Neste trabalho, o agente Pacman tem que encontrar caminhos no labirinto, tanto para chegar a um destino quanto para coletar comida eficientemente. O objetivo do trabalho será programar algoritmos de busca e aplicá-los ao cenário do Pacman. O código desse trabalho consiste de diversos arquivos Python, alguns dos quais você terá que ler e entender para fazer o trabalho.

Arquivos que devem ser editados:

- `search.py` - Onde ficam os algoritmos de busca.
- `searchAgents.py` - Onde ficam os agentes baseados em busca.

Arquivos que devem ser lidos:

- `pacman.py` - O arquivo principal que roda jogos de Pacman. Esse arquivo também descreve o tipo `Game↩State`, que será amplamente usado nesse trabalho.
- `game.py` - A lógica do mundo do Pacman. Este arquivo descreve vários tipos auxiliares como `Agent↩State`, `Agent`, `Direction` e `Grid`.
- `util.py` - Estruturas de dados úteis para implementar algoritmos de busca.

Arquivos que podem ser ignorados:

- `graphicsDisplay.py` - Visualização gráfica do Pacman
- `graphicsUtils.py` - Funções auxiliares para visualização gráfica do Pacman
- `textDisplay.py` - Visualização gráfica em ASCII para o Pacman
- `ghostAgents.py` - Agentes para controlar fantasmas

- `keyboardAgents.py` - Interfaces de controle do Pacman a partir do teclado
- `layout.py` - Código para ler arquivos de layout e guardar seu conteúdo

O que deve ser entregue:

- Os arquivos `search.py` e `searchAgents.py` serão modificados no trabalho.

Cada grupo deve entregar esses dois arquivos e deve entregar também um relatório impresso respondendo as perguntas listadas abaixo. Cada grupo deve ser composto de 2 ou 3 alunos.

Bem-vindo ao Pacman

Depois de baixar o código (search.zip), descompactá-lo e entrar no diretório search, você pode jogar um jogo de Pacman digitando a seguinte linha de comando:

```
python pacman.py
```

O agente mais simples em `searchAgents.py` é o agente `GoWestAgent`, que sempre vai para oeste (um agente reflexivo trivial). Este agente pode ganhar às vezes:

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

Mas as coisas se tornam mais difíceis quando virar é necessário:

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

`pacman.py` tem opções que podem ser dadas em formato longo (por exemplo, `--layout`) ou em formato curto (por exemplo, `-l`). A lista de todas as opções pode ser vista executando:

```
python pacman.py -h
```

Todos os comandos que aparecem aqui também estão em `commands.txt`, e podem ser copiados e colados.

Encontrando comida em um ponto fixo usando algoritmos de busca

No arquivo `searchAgents.py`, você irá encontrar o programa de um agente de busca (`SearchAgent`), que planeja um caminho no mundo do Pacman e executa o caminho passo-a-passo. Os algoritmos de busca para planejar o caminho não estão implementados – este será o seu trabalho.

Para entender o que está descrito a seguir, pode ser necessário olhar esse glossário de objetos. Primeiro, verifique que o agente de busca SearchAgent está funcionando corretamente, rodando:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

O comando acima faz o agente `SearchAgent` usar o algoritmo de busca `tinyMazeSearch`, que está implementado em `search.py`. O Pacman deve navegar o labirinto corretamente.

Agora chegou a hora de implementar os seus algoritmos de busca para o Pacman! Os pseudocódigos dos algoritmos de busca estão no livro.

Lembre-se que um nó da busca deve conter não só o estado mas também toda a informação necessária para reconstruir o caminho (sequência de ações) até aquele estado.Importante: Todas as funções de busca devem retornar uma lista de ações que irão levar o agente do início até o objetivo. Essas ações devem ser legais (direções válidas, sem passar pelas paredes).

Dica 1: Os algoritmos de busca são muito parecidos. Os algoritmos de busca em profundidade, busca em extensão, busca de custo uniforme e A$*$ diferem somente na ordem em que os nós são retirados da borda. Então o ideal é tentar implementar a busca em profundidade corretamente e depois será mais fácil implementar as outras. Uma possível implementação é criar um algoritmo de busca genérico que possa ser configurado com uma estratégia para retirar nós da borda. (Porém, implementar dessa forma não é necessário).

Dica 2: Dê uma olhada no código dos tipo `Stack` (pilha), `Queue` (fila) e `PriorityQueue` (fila com prioridade) que estão no arquivo `util.py`.

Etapa 1 (2 pontos)

Implemente o algoritmo de busca em profundidade (DFS) na função `depthFirstSearch()` do arquivo `search.py`. Para que a busca seja completa, implemente a versão de DFS que não expande estados repetidos (seção 3.5 do livro).

Teste seu código executando:

```
python pacman.py -l tinyMaze -p SearchAgent
python pacman.py -l mediumMaze -p SearchAgent
python pacman.py -l bigMaze -z .5 -p SearchAgent
```

A saída do Pacman irá mostrar os estados explorados e a ordem em que eles foram explorados (vermelho mais forte significa que o estado foi explorado antes).

- **Pergunta 1:** A ordem de exploração foi de acordo com o esperado? O Pacman realmente passa por todos os estados explorados no seu caminho para o objetivo?Dica: Se você usar a pilha Stack como estrutura de dados, a solução encontrada pelo algoritmo DFS para o mediumMaze deve ter comprimento 130 (se os sucessores forem colocados na pilha na ordem dada por getSuccessors; pode ter comprimento 246 se forem colocados na ordem reversa). (Pergunta 2) Essa é uma solução ótima? Senão, o que a busca em profundidade está fazendo de errado?

Etapa 2 (2 pontos)

Implemente o algoritmo de busca em extensão (BFS) na função `breadthFirstSearch` do arquivo `search.py`. De novo, implemente a versão que não expande estados que já foram visitados. Teste seu código executando:

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
```

- **Pergunta 3:** A busca BFS encontra a solução ótima? Senão, verifique a sua implementação. Se o seu código foi escrito de maneira correta, ele deve funcionar também para o quebra-cabeças de 8 peças (seção 3.2 do livro-texto) sem modificações.

```
python eightpuzzle.py
```

A busca BFS vai encontrar o caminho com o menor número de ações até o objetivo. Porém, podemos querer encontrar caminhos que sejam melhores de acordo com outros critérios. Considere o labirinto mediumDotted↩ Maze e o labirinto mediumScaryMaze. Mudando a função de custo, podemos fazer o Pacman encontrar caminhos diferentes. Por exemplo, podemos ter custos maiores para passar por áreas com fantasmas e custos menores para passar em áreas com comida, e um agente Pacman racional deve poder ajustar o seu comportamento.

Etapa 3 (2 pontos)

Implemente o algoritmo de busca de custo uniforme (checando estados repetidos) na função `uniformCost↩ Search` do arquivo `search.py`. Teste seu código executando os comandos a seguir, onde os agentes tem diferentes funções de custo (os agentes e as funções são dados):

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
python pacman.py -l mediumScaryMaze -p StayWestSearchAgentA* search
```

Etapa 4 (2 pontos)

Implemente a busca A$*$ (com checagem de estados repetidos) na função `StarSearch` do arquivo `search.py`. A busca A$*$ recebe uma heurística como parâmetro.

Heurísticas têm dois parâmetros: um estado do problema de busca (o parâmetro principal), e o próprio problema. A heurística implementada na função `nullHeuristic` do arquivo `search.py` é um exemplo trivial.

Teste sua implementação de A$*$ no problema original de encontrar um caminho através de um labirinto para uma posição fixa usando a heurística de distância Manhattan (implementada na função `manhattanHeuristic` do arquivo `searchAgents.py`).

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
```

A busca A$*$ deve achar a solução ótima um pouco mais rapidamente que a busca de custo uniforme (549 vs. 621 nós de busca expandidos na nossa implementação).

Coletando comida

Agora iremos atacar um problema mais difícil: fazer o Pacman comer toda a comida no menor número de passos possível. Para isso, usaremos uma nova definição de problema de busca que formaliza esse problema: `FoodSearchProblem` no arquivo `searchAgents.py` (já implementado). Uma solução é um caminho que coleta toda a comida no mundo do Pacman. A solução não será modificada se houverem fantasmas no caminho; ela só depende do posicionamento das paredes, da comida e do Pacman. Se os seus algoritmos de busca estiverem corretos, A$*$ com uma heurística nula (equivalente a busca de custo uniforme) deve encontrar uma solução para o problema testSearch sem nenhuma mudança no código (custo total de 7).

```
python pacman.py -l testSearch -p AStarFoodSearchAgent
```

Nota: `AStarFoodSearchAgent` é um atalho para `-p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic`. Porém, a busca de custo uniforme fica lenta até para problemas simples como `tinySearch`.

Etapa 5 (2 pontos)

Implemente uma heurística admissível `foodHeuristic` no arquivo `searchAgents.py` para o problema `FoodSearchProblem`. Teste seu agente no problema `trickySearch`:

```
python pacman.py -l trickySearch -p AStarFoodSearchAgent
```

Glossário

Este é um glossário dos objetos principais na base de código relacionada a problemas de busca:

- `SearchProblem` (`search.py`): Um `SearchProblem` é um objeto abstrato que representa o espaço de estados, função sucessora, custos, e estado objetivo de um problema. Você vai interagir com objetos do tipo `SearchProblem` somente através dos métodos definidos no topo de `search.py`.
- `PositionSearchProblem` (`searchAgents.py`): Um tipo específico de `SearchProblem` — corresponde a procurar por uma única comida no labirinto.
- `FoodSearchProblem` (`searchAgents.py`): Um tipo específico de `SearchProblem` — corresponde a procurar um caminho para comer toda a comida em um labirinto.
- **Função de Busca:** Uma função de busca é uma função que recebe como entrada uma instância de `SearchProblem`, roda algum algoritmo, e retorna a sequência de ações que levam ao objetivo. Exemplos de função de busca são `depthFirstSearch` e `breadthFirstSearch`, que deverão ser escritas pelo grupo. A função de busca dada `tinyMazeSearch` é uma função muito ruim que só funciona para o labirinto `tinyMaze`.
- `SearchAgent`: é uma classe que implementa um agente (um objeto que interage com o mundo) e faz seu planejamento de acordo com uma função de busca. `SearchAgent` primeiro usa uma função de busca para encontrar uma sequência de ações que levem ao estado objetivo, e depois executa as ações uma por vez.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 game.Actions Class Reference

Parts you shouldn't have to read #.

Public Member Functions

- def **reverseDirection** (action)
- def **vectorToDirection** (vector)
- def **directionToVector** (direction, speed=1.0)
- def **getPossibleActions** (config, walls)
- def **getLegalNeighbors** (position, walls)
- def **getSuccessor** (position, action)

Static Public Attributes

- int **TOLERANCE** = .001
- **reverseDirection** = staticmethod(reverseDirection)
- **vectorToDirection** = staticmethod(vectorToDirection)
- **directionToVector** = staticmethod(directionToVector)
- **getPossibleActions** = staticmethod(getPossibleActions)
- **getLegalNeighbors** = staticmethod(getLegalNeighbors)
- **getSuccessor** = staticmethod(getSuccessor)

4.1.1 Detailed Description

Parts you shouldn't have to read #.

```
A collection of static methods for manipulating move actions.
```

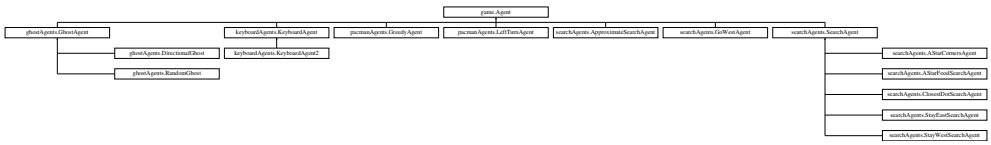
The documentation for this class was generated from the following file:

- game.py

4.2 game.Agent Class Reference

Parts worth reading #.

Inheritance diagram for game.Agent:



Public Member Functions

- def **__init__** (self, index=0)
- def getAction (self, state)

Public Attributes

- **index**

4.2.1 Detailed Description

Parts worth reading #.

```
An agent must define a getAction method, but may also define the
following methods which will be called if they exist:

def registerInitialState(self, state): # inspects the starting state
```

4.2.2 Member Function Documentation

4.2.2.1 getAction()

```
def game.Agent.getAction (
            self,
            state )
```

```
The Agent will receive a GameState (from either {pacman, capture, sonar}.py) and
must return an action from Directions.{North, South, East, West, Stop}
```

The documentation for this class was generated from the following file:

- game.py

4.3 game.AgentState Class Reference

Public Member Functions

- def **__init__** (self, startConfiguration, isPacman)
- def **__str__** (self)
- def **__eq__** (self, other)
- def **__hash__** (self)
- def **copy** (self)
- def **getPosition** (self)
- def **getDirection** (self)

Public Attributes

- **start**
- **configuration**
- **isPacman**
- **scaredTimer**

4.3.1 Detailed Description

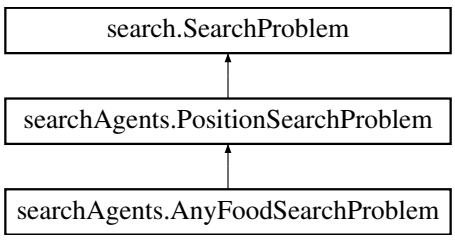
```
AgentStates hold the state of an agent (configuration, speed, scared, etc).
```

The documentation for this class was generated from the following file:

- game.py

4.4 searchAgents.AnyFoodSearchProblem Class Reference

Inheritance diagram for searchAgents.AnyFoodSearchProblem:



Public Member Functions

- def **__init__** (self, gameState)
- def isGoalState (self, state)

Public Attributes

- **food**
- **walls**
- **startState**
- **costFn**

4.4.1 Detailed Description

```
A search problem for finding a path to any food.

This search problem is just like the PositionSearchProblem, but
has a different goal test, which you need to fill in below.  The
state space and successor function do not need to be changed.

The class definition above, AnyFoodSearchProblem(PositionSearchProblem),
inherits the methods of the PositionSearchProblem.

You can use this search problem to help you fill in
the findPathToClosestDot method.
```

4.4.2 Member Function Documentation

4.4.2.1 isGoalState()

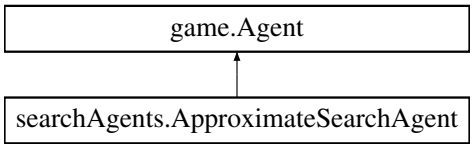
```
def searchAgents.AnyFoodSearchProblem.isGoalState (
              self,
              state )


The state is Pacman's position. Fill this in with a goal test
that will complete the problem definition.
```

The documentation for this class was generated from the following file:

- searchAgents.py

4.5 searchAgents.ApproximateSearchAgent Class Reference

Inheritance diagram for searchAgents.ApproximateSearchAgent:



Public Member Functions

- def **registerInitialState** (self, state)
- def getAction (self, state)

Additional Inherited Members

4.5.1 Member Function Documentation

4.5.1.1 getAction()

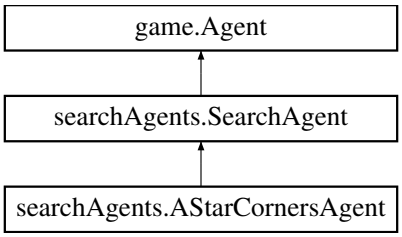
```
def searchAgents.ApproximateSearchAgent.getAction (
              self,
              state )

From game.py:
The Agent will receive a GameState and must return an action from
Directions.{North, South, East, West, Stop}
```

The documentation for this class was generated from the following file:

- searchAgents.py

4.6 searchAgents.AStarCornersAgent Class Reference

Inheritance diagram for searchAgents.AStarCornersAgent:



Public Member Functions

- def **__init__** (self)

Public Attributes

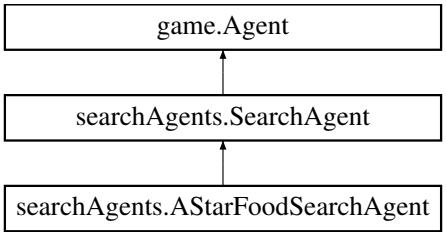
- **searchFunction**
- **searchType**

The documentation for this class was generated from the following file:

- searchAgents.py

4.7 searchAgents.AStarFoodSearchAgent Class Reference

Inheritance diagram for searchAgents.AStarFoodSearchAgent:



Public Member Functions

- def **__init__** (self)

Public Attributes

- **searchFunction**
- **searchType**

The documentation for this class was generated from the following file:

- searchAgents.py

4.8 pacman.ClassicGameRules Class Reference

Public Member Functions

- def **__init__** (self, timeout=30)
- def **newGame** (self, layout, pacmanAgent, ghostAgents, display, quiet=False, catchExceptions=False)
- def process (self, state, game)
- def **win** (self, state, game)
- def **lose** (self, state, game)
- def **getProgress** (self, game)
- def **agentCrash** (self, game, agentIndex)
- def **getMaxTotalTime** (self, agentIndex)
- def **getMaxStartupTime** (self, agentIndex)
- def **getMoveWarningTime** (self, agentIndex)
- def **getMoveTimeout** (self, agentIndex)
- def **getMaxTimeWarnings** (self, agentIndex)

Public Attributes

- **timeout**
- **initialState**
- **quiet**

4.8.1 Detailed Description

```
These game rules manage the control flow of a game, deciding when
and how the game starts and ends.
```

4.8.2 Member Function Documentation

4.8.2.1 process()

```
def pacman.ClassicGameRules.process (
              self,
              state,
              game )
```

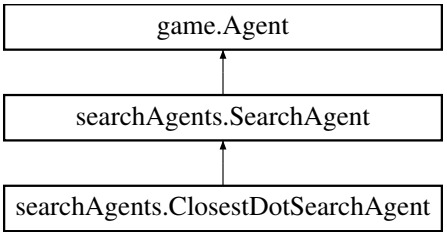
```
Checks to see whether it is time to end the game.
```

The documentation for this class was generated from the following file:

- pacman.py

4.9 searchAgents.ClosestDotSearchAgent Class Reference

Inheritance diagram for searchAgents.ClosestDotSearchAgent:



Public Member Functions

- def **registerInitialState** (self, state)
- def **findPathToClosestDot** (self, gameState)

Public Attributes

- **actions**
- **actionIndex**

The documentation for this class was generated from the following file:

- searchAgents.py

4.10 game.Configuration Class Reference

Public Member Functions

- def **__init__** (self, pos, direction)
- def **getPosition** (self)
- def **getDirection** (self)
- def **isInteger** (self)
- def **__eq__** (self, other)
- def **__hash__** (self)
- def **__str__** (self)
- def generateSuccessor (self, vector)

Public Attributes

- **pos**
- **direction**

4.10.1 Detailed Description

```
A Configuration holds the (x,y) coordinate of a character, along with its
traveling direction.

The convention for positions, like a graph, is that (0,0) is the lower left corner, x increases
horizontally and y increases vertically.  Therefore, north is the direction of increasing y, or (0,1).
```

4.10.2 Member Function Documentation

4.10.2.1 generateSuccessor()

```
def game.Configuration.generateSuccessor (
              self,
              vector )
```

```
Generates a new configuration reached by translating the current
configuration by the action vector.  This is a low-level call and does
not attempt to respect the legality of the movement.

Actions are movement vectors.
```

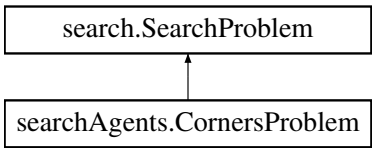
The documentation for this class was generated from the following file:

- game.py

4.11 searchAgents.CornersProblem Class Reference

This portion is incomplete.

Inheritance diagram for searchAgents.CornersProblem:



Public Member Functions

- def __init__ (self, startingGameState)
- def **getStartState** (self)
- def **isGoalState** (self, state)
- def getSuccessors (self, state)
- def getCostOfActions (self, actions)

Public Attributes

- **walls**
- **startingPosition**
- **corners**

4.11.1 Detailed Description

This portion is incomplete.

Time to write code! #

```
This search problem finds paths through all four corners of a layout.

You must select a suitable state space and successor function
```

4.11.2 Constructor & Destructor Documentation

4.11.2.1 __init__()

```
def searchAgents.CornersProblem.__init__ (
             self,
             startingGameState )
```

```
Stores the walls, pacman's starting position and corners.
```

4.11.3 Member Function Documentation

4.11.3.1 getCostOfActions()

```
def searchAgents.CornersProblem.getCostOfActions (
              self,
              actions )
```

```
Returns the cost of a particular sequence of actions.  If those actions
include an illegal move, return 999999.  This is implemented for you.
```

4.11.3.2 getSuccessors()

```
def searchAgents.CornersProblem.getSuccessors (
              self,
              state )
```

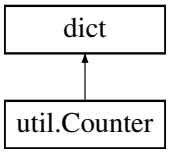
```
Returns successor states, the actions they require, and a cost of 1.

 As noted in search.py:
     For a given state, this should return a list of triples,
 (successor, action, stepCost), where 'successor' is a
 successor to the current state, 'action' is the action
 required to get there, and 'stepCost' is the incremental
 cost of expanding to that successor
```

The documentation for this class was generated from the following file:

- searchAgents.py

4.12 util.Counter Class Reference

Inheritance diagram for util.Counter:



Public Member Functions

- def **__getitem__** (self, idx)
- def incrementAll (self, keys, count)
- def argMax (self)
- def sortedKeys (self)
- def totalCount (self)
- def normalize (self)
- def divideAll (self, divisor)
- def copy (self)
- def __mul__ (self, y)
- def __radd__ (self, y)
- def __add__ (self, y)
- def __sub__ (self, y)

4.12.1 Detailed Description

```
A counter keeps track of counts for a set of keys.

The counter class is an extension of the standard python
dictionary type.  It is specialized to have number values
(integers or floats), and includes a handful of additional
functions to ease the task of counting data.  In particular,
all keys are defaulted to have value 0.  Using a dictionary:

a = {}
print a['test']

would give an error, while the Counter class analogue:

>>> a = Counter()
>>> print a['test']
0

returns the default 0 value. Note that to reference a key
that you know is contained in the counter,
you can still use the dictionary syntax:

>>> a = Counter()
>>> a['test'] = 2
>>> print a['test']
2

This is very useful for counting things without initializing their counts,
see for example:

>>> a['blah'] += 1
>>> print a['blah']
1

The counter also includes additional functionality useful in implementing
the classifiers for this assignment.  Two counters can be added,
subtracted or multiplied together.  See below for details.  They can
also be normalized and their total count and arg max can be extracted.
```

4.12.2 Member Function Documentation

4.12.2.1 __add__()

```
def util.Counter.__add__ (
              self,
              y )
```

Adding two counters gives a counter with the union of all keys and
counts of the second added to counts of the first.

```
>>> a = Counter()
>>> b = Counter()
>>> a['first'] = -2
>>> a['second'] = 4
>>> b['first'] = 3
>>> b['third'] = 1
>>> (a + b)['first']
1
```

4.12.2.2 __mul__()

```
def util.Counter.__mul__ (
              self,
              y )
```

Multiplying two counters gives the dot product of their vectors where
each unique label is a vector element.

```
>>> a = Counter()
>>> b = Counter()
>>> a['first'] = -2
>>> a['second'] = 4
>>> b['first'] = 3
>>> b['second'] = 5
>>> a['third'] = 1.5
>>> a['fourth'] = 2.5
>>> a * b
14
```

4.12.2.3 __radd__()

```
def util.Counter.__radd__ (
              self,
              y )
```

Adding another counter to a counter increments the current counter
by the values stored in the second counter.

```
>>> a = Counter()
>>> b = Counter()
>>> a['first'] = -2
>>> a['second'] = 4
>>> b['first'] = 3
>>> b['third'] = 1
>>> a += b
>>> a['first']
1
```

4.12.2.4 __sub__()

```
def util.Counter.__sub__ (
              self,
              y )
```

Subtracting a counter from another gives a counter with the union of all keys and counts of the second subtracted from counts of the first.

```
>>> a = Counter()
>>> b = Counter()
>>> a['first'] = -2
>>> a['second'] = 4
>>> b['first'] = 3
>>> b['third'] = 1
>>> (a - b)['first']
-5
```

4.12.2.5 argMax()

```
def util.Counter.argMax (
              self )
```

Returns the key with the highest value.

4.12.2.6 copy()

```
def util.Counter.copy (
              self )
```

Returns a copy of the counter

4.12.2.7 divideAll()

```
def util.Counter.divideAll (
              self,
              divisor )
```

Divides all counts by divisor

4.12.2.8 incrementAll()

```
def util.Counter.incrementAll (
              self,
              keys,
              count )

Increments all elements of keys by the same count.

>>> a = Counter()
>>> a.incrementAll(['one','two', 'three'], 1)
>>> a['one']
1
>>> a['two']
1
```

4.12.2.9 normalize()

```
def util.Counter.normalize (
              self )

Edits the counter such that the total count of all
keys sums to 1.  The ratio of counts for all keys
will remain the same. Note that normalizing an empty
Counter will result in an error.
```

4.12.2.10 sortedKeys()

```
def util.Counter.sortedKeys (
              self )

Returns a list of keys sorted by their values.  Keys
with the highest values will appear first.

>>> a = Counter()
>>> a['first'] = -2
>>> a['second'] = 4
>>> a['third'] = 1
>>> a.sortedKeys()
['second', 'third', 'first']
```

4.12.2.11 totalCount()

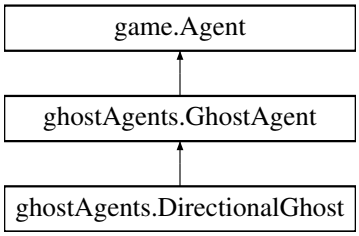
```
def util.Counter.totalCount (
              self )

Returns the sum of counts for all keys.
```

The documentation for this class was generated from the following file:

- util.py

4.13 ghostAgents.DirectionalGhost Class Reference

Inheritance diagram for ghostAgents.DirectionalGhost:



Public Member Functions

- def **__init__** (self, index, prob_attack=0.8, prob_scaredFlee=0.8)
- def **getDistribution** (self, state)

Public Attributes

- **index**
- **prob_attack**
- **prob_scaredFlee**

The documentation for this class was generated from the following file:

- ghostAgents.py

4.14 game.Directions Class Reference

Static Public Attributes

- string **NORTH** = 'North'
- string **SOUTH** = 'South'
- string **EAST** = 'East'
- string **WEST** = 'West'
- string **STOP** = 'Stop'
- dictionary **LEFT**
- **RIGHT** = dict([(y,x) for x, y in list(LEFT.items())])
- dictionary **REVERSE**

4.14.1 Member Data Documentation

4.14.1.1 LEFT

```
dictionary game.Directions.LEFT  [static]
```

Initial value:

```
=        {NORTH: WEST,
                 SOUTH: EAST,
                 EAST:  NORTH,
                 WEST:  SOUTH,
                 STOP:  STOP}
```

4.14.1.2 REVERSE

```
dictionary game.Directions.REVERSE  [static]
```

Initial value:

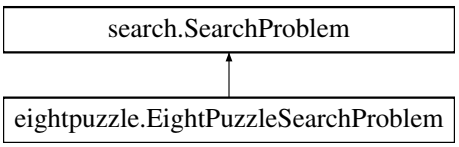
```
=  {NORTH: SOUTH,
              SOUTH: NORTH,
              EAST: WEST,
              WEST: EAST,
              STOP: STOP}
```

The documentation for this class was generated from the following file:

- game.py

4.15 eightpuzzle.EightPuzzleSearchProblem Class Reference

Inheritance diagram for eightpuzzle.EightPuzzleSearchProblem:



Public Member Functions

- def **__init__** (self, puzzle)
- def **getStartState** (self)
- def **isGoalState** (self, state)
- def getSuccessors (self, state)
- def getCostOfActions (self, actions)

Public Attributes

- **puzzle**

4.15.1 Detailed Description

```
  Implementation of a SearchProblem for the  Eight Puzzle domain

  Each state is represented by an instance of an eightPuzzle.
```

4.15.2 Member Function Documentation

4.15.2.1 getCostOfActions()

```
def eightpuzzle.EightPuzzleSearchProblem.getCostOfActions (
              self,
              actions )

 actions: A list of actions to take

This method returns the total cost of a particular sequence of actions.  The sequence must
be composed of legal moves
```

4.15.2.2 getSuccessors()

```
def eightpuzzle.EightPuzzleSearchProblem.getSuccessors (
              self,
              state )

  Returns list of (successor, action, stepCost) pairs where
  each succesor is either left, right, up, or down
  from the original state and the cost is 1.0 for each
```

The documentation for this class was generated from the following file:

- eightpuzzle.py

4.16 eightpuzzle.EightPuzzleState Class Reference

Public Member Functions

- def __init__ (self, numbers)
- def isGoal (self)
- def legalMoves (self)
- def result (self, move)
- def __eq__ (self, other)
- def **__hash__** (self)
- def **__str__** (self)

Public Attributes

- **cells**
- **blankLocation**

4.16.1 Detailed Description

```
The Eight Puzzle is described in the course textbook on
page 64.

This class defines the mechanics of the puzzle itself.  The
task of recasting this puzzle as a search problem is left to
the EightPuzzleSearchProblem class.
```

4.16.2 Constructor & Destructor Documentation

4.16.2.1 __init__()

```
def eightpuzzle.EightPuzzleState.__init__ (
              self,
              numbers )
```

```
  Constructs a new eight puzzle from an ordering of numbers.

numbers: a list of integers from 0 to 8 representing an
  instance of the eight puzzle.  0 represents the blank
  space.  Thus, the list

 [1, 0, 2, 3, 4, 5, 6, 7, 8]

  represents the eight puzzle:
 -------------
 | 1 |   | 2 |
 -------------
 | 3 | 4 | 5 |
 -------------
 | 6 | 7 | 8 |
 -------------

The configuration of the puzzle is stored in a 2-dimensional
list (a list of lists) 'cells'.
```

4.16.3 Member Function Documentation

4.16.3.1 __eq__()

```
def eightpuzzle.EightPuzzleState.__eq__ (
              self,
              other )
```

Overloads '==' such that two eightPuzzles with the same configuration are equal.

```
>>> EightPuzzleState([0, 1, 2, 3, 4, 5, 6, 7, 8]) == \
 EightPuzzleState([1, 0, 2, 3, 4, 5, 6, 7, 8]).result('left')
True
```

4.16.3.2 isGoal()

```
def eightpuzzle.EightPuzzleState.isGoal (
              self )
```

Checks to see if the puzzle is in its goal state.

```
 -------------
 |   | 1 | 2 |
 -------------
 | 3 | 4 | 5 |
 -------------
 | 6 | 7 | 8 |
 -------------
```

```
>>> EightPuzzleState([0, 1, 2, 3, 4, 5, 6, 7, 8]).isGoal()
True

>>> EightPuzzleState([1, 0, 2, 3, 4, 5, 6, 7, 8]).isGoal()
False
```

4.16.3.3 legalMoves()

```
def eightpuzzle.EightPuzzleState.legalMoves (
              self )
```

Returns a list of legal moves from the current state.

Moves consist of moving the blank space up, down, left or right.
These are encoded as 'up', 'down', 'left' and 'right' respectively.

```
>>> EightPuzzleState([0, 1, 2, 3, 4, 5, 6, 7, 8]).legalMoves()
['down', 'right']
```

4.16.3.4 result()

```
def eightpuzzle.EightPuzzleState.result (
              self,
              move )
```

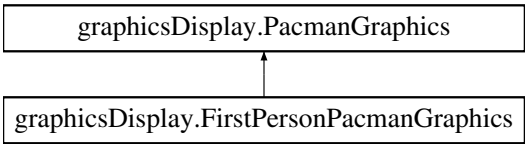
```
  Returns a new eightPuzzle with the current state and blankLocation
updated based on the provided move.

The move should be a string drawn from a list returned by legalMoves.
Illegal moves will raise an exception, which may be an array bounds
exception.

NOTE: This function *does not* change the current object.  Instead,
it returns a new object.
```

The documentation for this class was generated from the following file:

- eightpuzzle.py

4.17 graphicsDisplay.FirstPersonPacmanGraphics Class Reference

Inheritance diagram for graphicsDisplay.FirstPersonPacmanGraphics:



Public Member Functions

- def **__init__** (self, zoom=1.0, showGhosts=True, capture=False, frameTime=0)
- def **initialize** (self, state, isBlue=False)
- def **lookAhead** (self, config, state)
- def **getGhostColor** (self, ghost, ghostIndex)
- def **getPosition** (self, ghostState)

Public Attributes

- **showGhosts**
- **capture**
- **isBlue**
- **layout**
- **distributionImages**
- **previousState**

The documentation for this class was generated from the following file:

- graphicsDisplay.py

4.18 searchAgents.FoodSearchProblem Class Reference

Public Member Functions

- def **__init__** (self, startingGameState)
- def **getStartState** (self)
- def **isGoalState** (self, state)
- def **getSuccessors** (self, state)
- def getCostOfActions (self, actions)

Public Attributes

- **start**
- **walls**
- **startingGameState**
- **heuristicInfo**

4.18.1 Detailed Description

```
A search problem associated with finding the a path that collects all of the
food (dots) in a Pacman game.

A search state in this problem is a tuple ( pacmanPosition, foodGrid ) where
  pacmanPosition: a tuple (x,y) of integers specifying Pacman's position
  foodGrid:       a Grid (see game.py) of either True or False, specifying remaining food
```

4.18.2 Member Function Documentation

4.18.2.1 getCostOfActions()

```
def searchAgents.FoodSearchProblem.getCostOfActions (
              self,
              actions )
```

```
Returns the cost of a particular sequence of actions.  If those actions
include an illegal move, return 999999
```

The documentation for this class was generated from the following file:

- searchAgents.py

4.19 game.Game Class Reference

Public Member Functions

- def **__init__** (self, agents, display, rules, startingIndex=0, muteAgents=False, catchExceptions=False)
- def **getProgress** (self)
- def **mute** (self)
- def **unmute** (self)
- def run (self)

Public Attributes

- **agentCrashed**
- **agents**
- **display**
- **rules**
- **startingIndex**
- **gameOver**
- **muteAgents**
- **catchExceptions**
- **moveHistory**
- **totalAgentTimes**
- **totalAgentTimeWarnings**
- agentTimeout

  self.display.initialize(self.state.makeObservation(1).data) inform learning agents of the game start
- **numMoves**
- **state**

Static Public Attributes

- **OLD_STDOUT** = None
- **OLD_STDERR** = None

4.19.1 Detailed Description

```
The Game manages the control flow, soliciting actions from agents.
```

4.19.2 Member Function Documentation

4.19.2.1 run()

```
def game.Game.run (
              self )
```

```
Main control loop for game play.
```

4.19.3 Member Data Documentation

4.19.3.1 agentTimeout

`game.Game.agentTimeout`

self.display.initialize(self.state.makeObservation(1).data) inform learning agents of the game start

TODO: could this exceed the total time.

The documentation for this class was generated from the following file:

- game.py

4.20 pacman.GameState Class Reference

YOUR INTERFACE TO THE PACMAN WORLD: A GameState #.

Public Member Functions

- def getLegalActions (self, agentIndex=0)

  Accessor methods: use these to access state data #.
- def generateSuccessor (self, agentIndex, action)
- def **getLegalPacmanActions** (self)
- def generatePacmanSuccessor (self, action)
- def getPacmanState (self)
- def **getPacmanPosition** (self)
- def **getGhostStates** (self)
- def **getGhostState** (self, agentIndex)
- def **getGhostPosition** (self, agentIndex)
- def **getGhostPositions** (self)
- def **getNumAgents** (self)
- def **getScore** (self)
- def getCapsules (self)
- def **getNumFood** (self)
- def getFood (self)
- def getWalls (self)
- def **hasFood** (self, x, y)
- def **hasWall** (self, x, y)
- def **isLose** (self)
- def **isWin** (self)
- def __init__ (self, prevState=None)

  `Helper methods: #`

  You shouldn't need to call these directly #
- def **deepCopy** (self)
- def __eq__ (self, other)
- def __hash__ (self)
- def **__str__** (self)
- def initialize (self, layout, numGhostAgents=1000)

Public Attributes

- **data**

4.20.1 Detailed Description

YOUR INTERFACE TO THE PACMAN WORLD: A GameState #.

```
A GameState specifies the full game state, including the food, capsules,
agent configurations and score changes.

GameStates are used by the Game object to capture the actual state of the game and
can be used by agents to reason about the game.

Much of the information in a GameState is stored in a GameStateData object.  We
strongly suggest that you access that data via the accessor methods below rather
than referring to the GameStateData object directly.

Note that in classic Pacman, Pacman is always agent 0.
```

4.20.2 Constructor & Destructor Documentation

4.20.2.1 __init__()

```
def pacman.GameState.__init__ (
              self,
              prevState = None )

        Helper methods:               #
```

You shouldn't need to call these directly #

```
Generates a new state by copying information from its predecessor.
```

4.20.3 Member Function Documentation

4.20.3.1 __eq__()

```
def pacman.GameState.__eq__ (
              self,
              other )

Allows two states to be compared.
```

4.20.3.2 __hash__()

```
def pacman.GameState.__hash__ (
              self )
```

Allows states to be keys of dictionaries.

4.20.3.3 generatePacmanSuccessor()

```
def pacman.GameState.generatePacmanSuccessor (
              self,
              action )
```

Generates the successor state after the specified pacman move

4.20.3.4 generateSuccessor()

```
def pacman.GameState.generateSuccessor (
              self,
              agentIndex,
              action )
```

Returns the successor state after the specified agent takes the action.

4.20.3.5 getCapsules()

```
def pacman.GameState.getCapsules (
              self )
```

Returns a list of positions (x,y) of the remaining capsules.

4.20.3.6 getFood()

```
def pacman.GameState.getFood (
              self )
```

```
Returns a Grid of boolean food indicator variables.

Grids can be accessed via list notation, so to check
if there is food at (x,y), just call

currentFood = state.getFood()
if currentFood[x][y] == True: ...
```

4.20.3.7 getLegalActions()

```
def pacman.GameState.getLegalActions (
              self,
              agentIndex = 0 )
```

Accessor methods: use these to access state data #.

```
Returns the legal actions for the agent specified.
```

4.20.3.8 getPacmanState()

```
def pacman.GameState.getPacmanState (
              self )
```

```
Returns an AgentState object for pacman (in game.py)

state.pos gives the current position
state.direction gives the travel vector
```

4.20.3.9 getWalls()

```
def pacman.GameState.getWalls (
              self )
```

```
Returns a Grid of boolean wall indicator variables.

Grids can be accessed via list notation, so to check
if there is food at (x,y), just call

walls = state.getWalls()
if walls[x][y] == True: ...
```

4.20.3.10 initialize()

```
def pacman.GameState.initialize (
              self,
              layout,
              numGhostAgents = 1000 )
```

Creates an initial game state from a layout array (see layout.py).

The documentation for this class was generated from the following file:

- pacman.py

4.21 game.GameStateData Class Reference

Public Member Functions

- def __init__ (self, prevState=None)
- def **deepCopy** (self)
- def **copyAgentStates** (self, agentStates)
- def __eq__ (self, other)
- def __hash__ (self)
- def **__str__** (self)
- def initialize (self, layout, numGhostAgents)

Public Attributes

- **food**
- **capsules**
- **agentStates**
- **layout**
- **score**
- **scoreChange**

4.21.1 Detailed Description

4.21.2 Constructor & Destructor Documentation

4.21.2.1 __init__()

```
def game.GameStateData.__init__ (
              self,
              prevState = None )
```

Generates a new data packet by copying information from its predecessor.

4.21.3 Member Function Documentation

4.21.3.1 __eq__()

```
def game.GameStateData.__eq__ (
              self,
              other )
```

Allows two states to be compared.

4.21.3.2 __hash__()

```
def game.GameStateData.__hash__ (
              self )
```

Allows states to be keys of dictionaries.

4.21.3.3 initialize()

```
def game.GameStateData.initialize (
              self,
              layout,
              numGhostAgents )
```

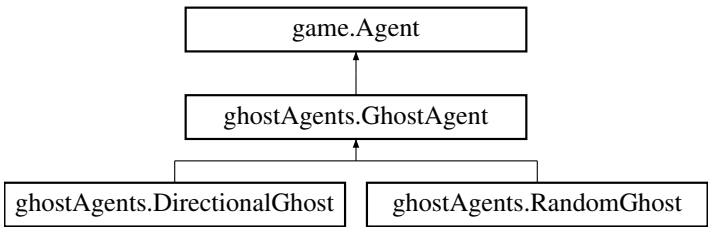
Creates an initial game state from a layout array (see layout.py).

The documentation for this class was generated from the following file:

- game.py

4.22 ghostAgents.GhostAgent Class Reference

Inheritance diagram for ghostAgents.GhostAgent:



Public Member Functions

- def **__init__** (self, index)
- def **getAction** (self, state)
- def **getDistribution** (self, state)

Public Attributes

- **index**

The documentation for this class was generated from the following file:

- ghostAgents.py

4.23 pacman.GhostRules Class Reference

Public Member Functions

- def getLegalActions (state, ghostIndex)
- def **applyAction** (state, action, ghostIndex)
- def **decrementTimer** (ghostState)
- def **checkDeath** (state, agentIndex)
- def **collide** (state, ghostState, agentIndex)
- def **canKill** (pacmanPosition, ghostPosition)
- def **placeGhost** (state, ghostState)

Static Public Attributes

- float **GHOST_SPEED** = 1.0
- **getLegalActions** = staticmethod(getLegalActions)
- **applyAction** = staticmethod(applyAction)
- **decrementTimer** = staticmethod(decrementTimer)
- **checkDeath** = staticmethod(checkDeath)
- **collide** = staticmethod(collide)
- **canKill** = staticmethod(canKill)
- **placeGhost** = staticmethod(placeGhost)

4.23.1 Detailed Description

```
These functions dictate how ghosts interact with their environment.
```

4.23.2 Member Function Documentation

4.23.2.1 getLegalActions()

```
def pacman.GhostRules.getLegalActions (
              state,
              ghostIndex )
```

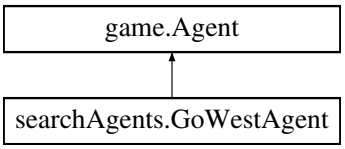
```
Ghosts cannot stop, and cannot turn around unless they
reach a dead end, but can turn 90 degrees at intersections.
```

The documentation for this class was generated from the following file:

- pacman.py

4.24 searchAgents.GoWestAgent Class Reference

Inheritance diagram for searchAgents.GoWestAgent:



Public Member Functions

- def **getAction** (self, state)

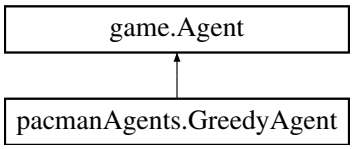
Additional Inherited Members

The documentation for this class was generated from the following file:

- searchAgents.py

4.25 pacmanAgents.GreedyAgent Class Reference

Inheritance diagram for pacmanAgents.GreedyAgent:



Public Member Functions

- def **__init__** (self, evalFn="scoreEvaluation")
- def **getAction** (self, state)

Public Attributes

- **evaluationFunction**

The documentation for this class was generated from the following file:

- pacmanAgents.py

4.26 game.Grid Class Reference

Public Member Functions

- def **__init__** (self, width, height, initialValue=False, bitRepresentation=None)
- def **__getitem__** (self, i)
- def **__setitem__** (self, key, item)
- def **__str__** (self)
- def **__eq__** (self, other)
- def **__hash__** (self)
- def **copy** (self)
- def **deepCopy** (self)
- def **shallowCopy** (self)
- def **count** (self, item=True)
- def **asList** (self, key=True)
- def packBits (self)

Public Attributes

- **CELLS_PER_INT**
- **width**
- **height**
- **data**

4.26.1 Detailed Description

```
A 2-dimensional array of objects backed by a list of lists.  Data is accessed
via grid[x][y] where (x,y) are positions on a Pacman map with x horizontal,
y vertical and the origin (0,0) in the bottom left corner.

The __str__ method constructs an output that is oriented like a pacman board.
```

4.26.2 Member Function Documentation

4.26.2.1 packBits()

```
def game.Grid.packBits (
              self )
```

```
Returns an efficient int list representation

(width, height, bitPackedInts...)
```

The documentation for this class was generated from the following file:

- game.py

4.27 graphicsDisplay.InfoPane Class Reference

Public Member Functions

- def **__init__** (self, layout, gridSize)
- def toScreen (self, pos, y=None)
- def **drawPane** (self)
- def **initializeGhostDistances** (self, distances)
- def **updateScore** (self, score)
- def **setTeam** (self, isBlue)
- def **updateGhostDistances** (self, distances)
- def **drawGhost** (self)
- def **drawPacman** (self)
- def **drawWarning** (self)
- def **clearIcon** (self)
- def **updateMessage** (self, message)
- def **clearMessage** (self)

Public Attributes

- **gridSize**
- **width**
- **base**
- **height**
- **fontSize**
- **textColor**
- **scoreText**
- **ghostDistanceText**
- **teamText**

4.27.1 Member Function Documentation

4.27.1.1 toScreen()

```
def graphicsDisplay.InfoPane.toScreen (
              self,
              pos,
              y = None )
```

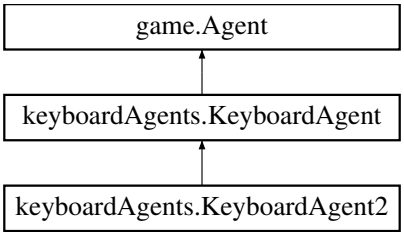
```
  Translates a point relative from the bottom left of the info pane.
```

The documentation for this class was generated from the following file:

- graphicsDisplay.py

4.28 keyboardAgents.KeyboardAgent Class Reference

Inheritance diagram for keyboardAgents.KeyboardAgent:



Public Member Functions

- def **__init__** (self, index=0)
- def **getAction** (self, state)
- def **getMove** (self, legal)

Public Attributes

- **lastMove**
- **index**
- **keys**

Static Public Attributes

- string **WEST_KEY** = 'a'
- string **EAST_KEY** = 'd'
- string **NORTH_KEY** = 'w'
- string **SOUTH_KEY** = 's'
- string **STOP_KEY** = 'q'

4.28.1 Detailed Description

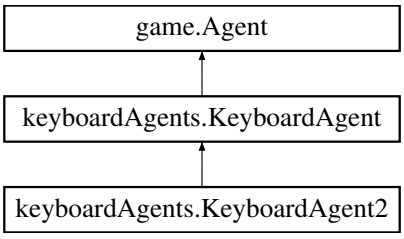
```
An agent controlled by the keyboard.
```

The documentation for this class was generated from the following file:

- keyboardAgents.py

4.29 keyboardAgents.KeyboardAgent2 Class Reference

Inheritance diagram for keyboardAgents.KeyboardAgent2:



Public Member Functions

- def **getMove** (self, legal)

Static Public Attributes

- string **WEST_KEY** = 'j'
- string **EAST_KEY** = "l"
- string **NORTH_KEY** = 'i'
- string **SOUTH_KEY** = 'k'
- string **STOP_KEY** = 'u'

Additional Inherited Members

4.29.1 Detailed Description

```
A second agent controlled by the keyboard.
```

The documentation for this class was generated from the following file:

- keyboardAgents.py

4.30 layout.Layout Class Reference

Public Member Functions

- def **__init__** (self, layoutText)
- def **getNumGhosts** (self)
- def **initializeVisibilityMatrix** (self)
- def **isWall** (self, pos)
- def **getRandomLegalPosition** (self)
- def **getRandomCorner** (self)
- def **getFurthestCorner** (self, pacPos)
- def **isVisibleFrom** (self, ghostPos, pacPos, pacDirection)
- def **__str__** (self)
- def **deepCopy** (self)
- def processLayoutText (self, layoutText)
- def **processLayoutChar** (self, x, y, layoutChar)

Public Attributes

- **width**
- **height**
- **walls**
- **food**
- **capsules**
- **agentPositions**
- **numGhosts**
- **layoutText**
- **visibility**

4.30.1 Detailed Description

```
A Layout manages the static information about the game board.
```

4.30.2 Member Function Documentation

4.30.2.1 processLayoutText()

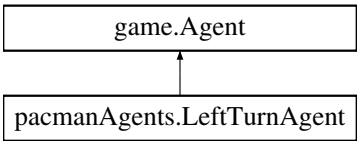
```
def layout.Layout.processLayoutText (
              self,
              layoutText )

Coordinates are flipped from the input format to the (x,y) convention here

The shape of the maze.  Each character
represents a different type of object.
 % - Wall
 . - Food
 o - Capsule
 G - Ghost
 P - Pacman
Other characters are ignored.
```

The documentation for this class was generated from the following file:

- layout.py

4.31 pacmanAgents.LeftTurnAgent Class Reference

Inheritance diagram for pacmanAgents.LeftTurnAgent:



Public Member Functions

- def **getAction** (self, state)

Additional Inherited Members

The documentation for this class was generated from the following file:

- pacmanAgents.py

4.32 textDisplay.NullGraphics Class Reference

Public Member Functions

- def **initialize** (self, state, isBlue=False)
- def **update** (self, state)
- def **pause** (self)
- def **draw** (self, state)
- def **finish** (self)

The documentation for this class was generated from the following file:

- textDisplay.py

4.33 textDisplay.PacmanGraphics Class Reference

Public Member Functions

- def **__init__** (self, speed=None)
- def **initialize** (self, state, isBlue=False)
- def **update** (self, state)
- def **pause** (self)
- def **draw** (self, state)
- def **finish** (self)

Public Attributes

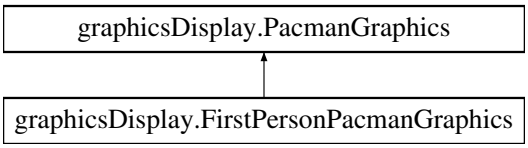
- **turn**
- **agentCounter**

The documentation for this class was generated from the following file:

- textDisplay.py

4.34 graphicsDisplay.PacmanGraphics Class Reference

Inheritance diagram for graphicsDisplay.PacmanGraphics:



Public Member Functions

- def **__init__** (self, zoom=1.0, frameTime=0.0, capture=False)
- def **initialize** (self, state, isBlue=False)
- def **startGraphics** (self, state)
- def **drawDistributions** (self, state)
- def **drawStaticObjects** (self, state)
- def **drawAgentObjects** (self, state)
- def swapImages (self, agentIndex, newState)
- def **update** (self, newState)
- def **make_window** (self, width, height)
- def **drawPacman** (self, pacman, index)
- def **getEndpoints** (self, direction, position=(0, 0))
- def **movePacman** (self, position, direction, image)
- def **animatePacman** (self, pacman, prevPacman, image)
- def **getGhostColor** (self, ghost, ghostIndex)
- def **drawGhost** (self, ghost, agentIndex)
- def **moveEyes** (self, pos, dir, eyes)

- def **moveGhost** (self, ghost, ghostIndex, prevGhost, ghostImageParts)
- def **getPosition** (self, agentState)
- def **getDirection** (self, agentState)
- def **finish** (self)
- def **to_screen** (self, point)
- def **to_screen2** (self, point)
- def **drawWalls** (self, wallMatrix)
- def **isWall** (self, x, y, walls)
- def **drawFood** (self, foodMatrix)
- def **drawCapsules** (self, capsules)
- def **removeFood** (self, cell, foodImages)
- def **removeCapsule** (self, cell, capsuleImages)
- def drawExpandedCells (self, cells)
- def **clearExpandedCells** (self)
- def **updateDistributions** (self, distributions)

Public Attributes

- **have_window**
- **currentGhostImages**
- **pacmanImage**
- **zoom**
- **gridSize**
- **capture**
- **frameTime**
- **isBlue**
- **distributionImages**
- **previousState**
- **layout**
- **width**
- **height**
- **infoPane**
- **currentState**
- **food**
- **capsules**
- **agentImages**
- **expandedCells**

4.34.1 Member Function Documentation

4.34.1.1 drawExpandedCells()

```
def graphicsDisplay.PacmanGraphics.drawExpandedCells (
              self,
              cells )
```

```
Draws an overlay of expanded grid positions for search agents
```

4.34.1.2 swapImages()

```
def graphicsDisplay.PacmanGraphics.swapImages (
              self,
              agentIndex,
              newState )
```

```
  Changes an image from a ghost to a pacman or vis versa (for capture)
```

The documentation for this class was generated from the following file:

- graphicsDisplay.py

4.35 pacman.PacmanRules Class Reference

Public Member Functions

- def getLegalActions (state)
- def applyAction (state, action)
- def **consume** (position, state)

Static Public Attributes

- int **PACMAN_SPEED** = 1
- **getLegalActions** = staticmethod(getLegalActions)
- **applyAction** = staticmethod(applyAction)
- **consume** = staticmethod(consume)

4.35.1 Detailed Description

```
These functions govern how pacman interacts with his environment under
the classic game rules.
```

4.35.2 Member Function Documentation

4.35.2.1 applyAction()

```
def pacman.PacmanRules.applyAction (
              state,
              action )
```

```
Edits the state to reflect the results of the action.
```

4.35.2.2 getLegalActions()

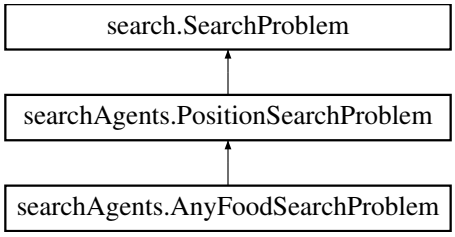
```
def pacman.PacmanRules.getLegalActions (
              state )

Returns a list of possible actions.
```

The documentation for this class was generated from the following file:

- pacman.py

4.36 searchAgents.PositionSearchProblem Class Reference

Inheritance diagram for searchAgents.PositionSearchProblem:



Public Member Functions

- def __init__ (self, gameState, costFn=lambda x:1, goal=(1, 1), start=None, warn=True)
- def **getStartState** (self)
- def **isGoalState** (self, state)
- def getSuccessors (self, state)
- def getCostOfActions (self, actions)

Public Attributes

- **walls**
- **startState**
- **goal**
- **costFn**

4.36.1 Detailed Description

```
A search problem defines the state space, start state, goal test,
successor function and cost function.  This search problem can be
used to find paths to a particular point on the pacman board.

The state space consists of (x,y) positions in a pacman game.

Note: this search problem is fully specified; you should NOT change it.
```

4.36.2 Constructor & Destructor Documentation

4.36.2.1 __init__()

```
def searchAgents.PositionSearchProblem.__init__ (
              self,
              gameState,
              costFn = lambda x: 1,
              goal = (1, 1),
              start = None,
              warn = True )
```

```
Stores the start and goal.

gameState: A GameState object (pacman.py)
costFn: A function from a search state (tuple) to a non-negative number
goal: A position in the gameState
```

4.36.3 Member Function Documentation

4.36.3.1 getCostOfActions()

```
def searchAgents.PositionSearchProblem.getCostOfActions (
              self,
              actions )
```

```
Returns the cost of a particular sequence of actions.  If those actions
include an illegal move, return 999999
```

4.36.3.2 getSuccessors()

```
def searchAgents.PositionSearchProblem.getSuccessors (
              self,
              state )
```

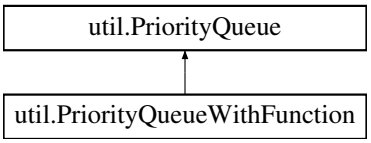
```
Returns successor states, the actions they require, and a cost of 1.

 As noted in search.py:
     For a given state, this should return a list of triples,
 (successor, action, stepCost), where 'successor' is a
 successor to the current state, 'action' is the action
 required to get there, and 'stepCost' is the incremental
 cost of expanding to that successor
```

The documentation for this class was generated from the following file:

- searchAgents.py

4.37 util.PriorityQueue Class Reference

Inheritance diagram for util.PriorityQueue:



Public Member Functions

- def **__init__** (self)
- def **push** (self, item, priority)
- def **pop** (self)
- def **isEmpty** (self)

Public Attributes

- **heap**

4.37.1 Detailed Description

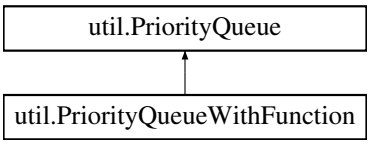
```
Implements a priority queue data structure. Each inserted item
has a priority associated with it and the client is usually interested
in quick retrieval of the lowest-priority item in the queue. This
data structure allows O(1) access to the lowest-priority item.

Note that this PriorityQueue does not allow you to change the priority
of an item.  However, you may insert the same item multiple times with
different priorities.
```

The documentation for this class was generated from the following file:

- util.py

4.38 util.PriorityQueueWithFunction Class Reference

Inheritance diagram for util.PriorityQueueWithFunction:



Public Member Functions

- def **__init__** (self, priorityFunction)
- def **push** (self, item)

Public Attributes

- **priorityFunction**

4.38.1 Detailed Description

```
Implements a priority queue with the same push/pop signature of the
Queue and the Stack classes. This is designed for drop-in replacement for
those two classes. The caller has to provide a priority function, which
extracts each item's priority.
```

The documentation for this class was generated from the following file:

- util.py

4.39 util.Queue Class Reference

Public Member Functions

- def **__init__** (self)
- def **push** (self, item)
- def pop (self)
- def **isEmpty** (self)

Public Attributes

- **list**

4.39.1 Member Function Documentation

4.39.1.1 pop()

```
def util.Queue.pop (
              self )
```

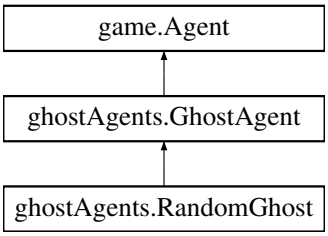
```
  Dequeue the earliest enqueued item still in the queue. This
  operation removes the item from the queue.
```

The documentation for this class was generated from the following file:

- util.py

4.40 ghostAgents.RandomGhost Class Reference

Inheritance diagram for ghostAgents.RandomGhost:



Public Member Functions

- def **getDistribution** (self, state)

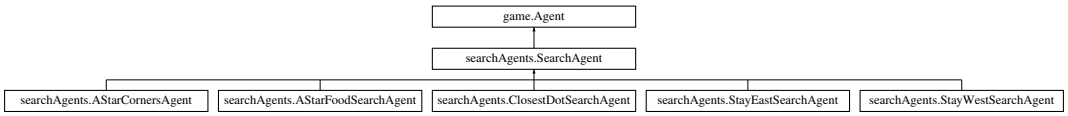
Additional Inherited Members

The documentation for this class was generated from the following file:

- ghostAgents.py

4.41 searchAgents.SearchAgent Class Reference

Inheritance diagram for searchAgents.SearchAgent:



Public Member Functions

- def **__init__** (self, fn='depthFirstSearch', prob='PositionSearchProblem', heuristic='nullHeuristic')
- def registerInitialState (self, state)
- def getAction (self, state)

Public Attributes

- **searchFunction**
- **searchType**
- **actions**
- **actionIndex**

4.41.1 Detailed Description

```
This very general search agent finds a path using a supplied search algorithm for a
supplied search problem, then returns actions to follow that path.

As a default, this agent runs DFS on a PositionSearchProblem to find location (1,1)

Options for fn include:
  depthFirstSearch or dfs
  breadthFirstSearch or bfs


Note: You should NOT change any code in SearchAgent
```

4.41.2 Member Function Documentation

4.41.2.1 getAction()

```
def searchAgents.SearchAgent.getAction (
              self,
              state )


Returns the next action in the path chosen earlier (in registerInitialState).  Return
Directions.STOP if there is no further action to take.

state: a GameState object (pacman.py)
```

4.41.2.2 registerInitialState()

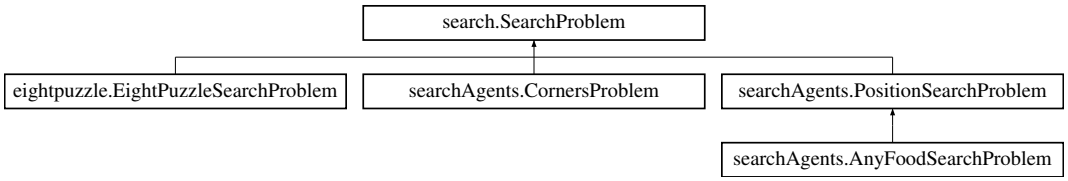
```
def searchAgents.SearchAgent.registerInitialState (
              self,
              state )


This is the first time that the agent sees the layout of the game board. Here, we
choose a path to the goal.  In this phase, the agent should compute the path to the
goal and store it in a local variable.  All of the work is done in this method!

state: a GameState object (pacman.py)
```

The documentation for this class was generated from the following file:

- searchAgents.py

4.42 search.SearchProblem Class Reference

Inheritance diagram for search.SearchProblem:



Public Member Functions

- def getStartState (self)
- def isGoalState (self, state)
- def getSuccessors (self, state)
- def getCostOfActions (self, actions)

4.42.1 Detailed Description

```
This class outlines the structure of a search problem, but doesn't implement
any of the methods (in object-oriented terminology: an abstract class).

You do not need to change anything in this class, ever.
```

4.42.2 Member Function Documentation

4.42.2.1 getCostOfActions()

```
def search.SearchProblem.getCostOfActions (
              self,
              actions )
```

```
 actions: A list of actions to take

This method returns the total cost of a particular sequence of actions.  The sequence must
be composed of legal moves
```

4.42.2.2 getStartState()

```
def search.SearchProblem.getStartState (
              self )
```

```
Returns the start state for the search problem
```

4.42.2.3 getSuccessors()

```
def search.SearchProblem.getSuccessors (
              self,
              state )


  state: Search state

For a given state, this should return a list of triples,
(successor, action, stepCost), where 'successor' is a
successor to the current state, 'action' is the action
required to get there, and 'stepCost' is the incremental
cost of expanding to that successor
```

4.42.2.4 isGoalState()

```
def search.SearchProblem.isGoalState (
              self,
              state )


  state: Search state

Returns True if and only if the state is a valid goal state
```

The documentation for this class was generated from the following file:

- search.py

4.43 util.Stack Class Reference

Public Member Functions

- def **__init__** (self)
- def **push** (self, item)
- def **pop** (self)
- def **isEmpty** (self)

Public Attributes

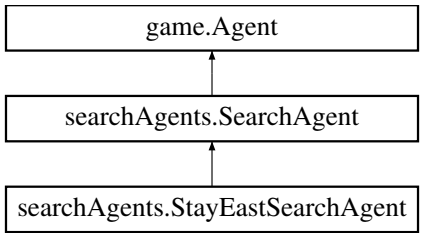
- **list**

The documentation for this class was generated from the following file:

- util.py

4.44 searchAgents.StayEastSearchAgent Class Reference

Inheritance diagram for searchAgents.StayEastSearchAgent:



Public Member Functions

- def **__init__** (self)

Public Attributes

- **searchFunction**
- **searchType**

4.44.1 Detailed Description

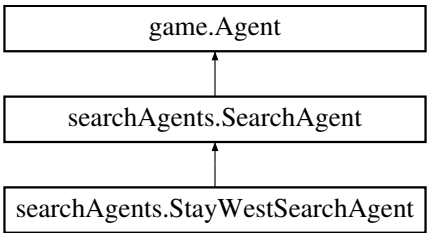
```
An agent for position search with a cost function that penalizes being in
positions on the West side of the board.

The cost function for stepping into a position (x,y) is 1/2^x.
```

The documentation for this class was generated from the following file:

- searchAgents.py

4.45 searchAgents.StayWestSearchAgent Class Reference

Inheritance diagram for searchAgents.StayWestSearchAgent:



Public Member Functions

- def **__init__** (self)

Public Attributes

- **searchFunction**
- **searchType**

4.45.1 Detailed Description

```
An agent for position search with a cost function that penalizes being in
positions on the East side of the board.

The cost function for stepping into a position (x,y) is 2^x.
```

The documentation for this class was generated from the following file:

- searchAgents.py

4.46 util.TimeoutFunction Class Reference

Public Member Functions

- def **__init__** (self, function, timeout)
- def **handle_timeout** (self, signum, frame)
- def **__call__** (self, args)

Public Attributes

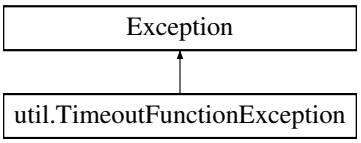
- **timeout**
- **function**

The documentation for this class was generated from the following file:

- util.py

4.47 util.TimeoutFunctionException Class Reference

Inheritance diagram for util.TimeoutFunctionException:



4.47.1 Detailed Description

```
Exception to raise on a timeout
```

The documentation for this class was generated from the following file:

- util.py

Index