

game.py ([original](#))

```
# game.py
# -----
# Licensing Information: Please do not distribute or publish solutions to this
# project. You are free to use and extend these projects for educational
# purposes. The Pacman AI projects were developed at UC Berkeley, primarily by
# John DeNero (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
# For more info, see http://inst.eecs.berkeley.edu/~cs188/sp09/pacman.html
```

```
from util import *
import time, os
import traceback
```

```
#####
# Parts worth reading #
#####
```

```
class Agent:
```

```
    """
```

```
    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
    """
```

```
    def __init__(self, index=0):
        self.index = index
```

```
    def getAction(self, state):
    """
```

```
    The Agent will
    must return an
    """
```

```
    raiseNotDefined()
```

```
class Directions:
```

```
    NORTH = 'North'
    SOUTH = 'South'
    EAST = 'East'
    WEST = 'West'
    STOP = 'Stop'
```

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

```
    RIGHT = dict([(y,x) for x, y in LEFT.items()])
```

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

```
class Configuration:
```

```
    """
```

```
    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).
```

```
    """
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

def __init__(self, pos, direction):
    self.pos = pos
    self.direction = direction

def getPosition(self):
    return (self.pos)

def getDirection(self):
    return self.direction

def isInteger(self):
    x,y = self.pos
    return x == int(x) and y == int(y)

def __eq__(self, other):
    if other == None: return False
    return (self.pos == other.pos and self.direction == other.direction)

def __hash__(self):
    x = hash(self.pos)
    y = hash(self.direction)
    return hash(x + 13 * y)

def __str__(self):
    return "(x,y)="+str(self.pos)+", "+str(self.direction)

```

```

def 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 mo

```

"""
x, y= self.pos
dx, dy = vector
direction = Actions.vectorToDirection(ve
if direction == Directions.SFOR:
    direction = self.direction # There is
return Configuration((x + dx, y+dy), direction)

```

```

class AgentState:

```

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

```

def __init__( self, startConfiguration, isPacman ):
    self.start = startConfiguration
    self.configuration = startConfiguration
    self.isPacman = isPacman
    self.scaredTimer = 0

```

```

def __str__( self ):
    if self.isPacman:
        return "Pacman: " + str( self.configuration )
    else:
        return "Ghost: " + str( self.configuration )

```

```

def __eq__( self, other ):
    if other == None:
        return False
    return self.configuration == other.configuration and self.scaredTimer ==
other.scaredTimer

```

```

def __hash__(self):
    return hash(hash(self.configuration) + 13 * hash(self.scaredTimer))

```

```

def copy( self ):

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

state = AgentState( self.start, self.isPacman )
state.configuration = self.configuration
state.scaredTimer = self.scaredTimer
return state

def getPosition(self):
    if self.configuration == None: return None
    return self.configuration.getPosition()

def getDirection(self):
    return self.configuration.getDirection()

class Grid:
    """
    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.
    """
    def __init__(self, width, height, initialValue=False, bitRepresentation=None):
        if initialValue not in [False, True]: raise Exception('Grids can only contain
booleans')
        self.CELLS_PER_INT = 30

        self.width = width
        self.height = height
        self.data = [[initialValue for y in range(height)] for x in range(width)]
        if bitRepresentation:
            self.unpackBits(bitRepresentation)

    def __getitem__(self, i):
        return self.data

    def __setitem__(self, key):
        self.data[key]

    def __str__(self):
        out = [[str(self.data[x][y]) for x in range(self.width)] for y in
range(self.height)]
        out.reverse()
        return '\n'.join([''.join(x) for x in out])

    def __eq__(self, other):
        if other == None: return False
        return self.data == other.data

    def __hash__(self):
        # return hash(str(self))
        base = 1
        h = 0
        for l in self.data:
            for i in l:
                if i:
                    h += base
                base *= 2
        return hash(h)

    def copy(self):
        g = Grid(self.width, self.height)
        g.data = [x[:] for x in self.data]
        return g

    def deepCopy(self):
        return self.copy()

    def shallowCopy(self):
        g = Grid(self.width, self.height)
        g.data = self.data

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

    return g

def count(self, item =True ):
    return sum([x.count(item) for x in self.data])

def asList(self, key = True):
    list = []
    for x in range(self.width):
        for y in range(self.height):
            if self[x][y] == key: list.append( (x,y) )
    return list

def packBits(self):
    """
    Returns an efficient int list representation

    (width, height, bitPackedInts...)
    """
    bits = [self.width, self.height]
    currentInt = 0
    for i in range(self.height * self.width):
        bit = self.CELLS_PER_INT - (i % self.CELLS_PER_INT) - 1
        x, y = self._cellIndexToPosition(i)
        if self[x][y]:
            currentInt += 2 ** bit
            if (i + 1) % self.CELLS_PER_INT == 0:
                bits.append(currentInt)
                currentInt = 0
    bits.append(currentInt)
    return tuple(bits)

def _cellIndexToPosition(self, index):
    x = index / se
    y = index % se
    return x, y

def _unpackBits(self, bits):
    """
    Fills in data from a bit-level represent
    """
    cell = 0
    for packed in bits:
        for bit in self._unpackInt(packed, self.CELLS_PER_INT):
            if cell == self.width * self.height: break
            x, y = self._cellIndexToPosition(cell)
            self[x][y] = bit
            cell += 1

def _unpackInt(self, packed, size):
    bools = []
    if packed < 0: raise ValueError, "must be a positive integer"
    for i in range(size):
        n = 2 ** (self.CELLS_PER_INT - i - 1)
        if packed >= n:
            bools.append(True)
            packed -= n
        else:
            bools.append(False)
    return bools

def reconstituteGrid(bitRep):
    if type(bitRep) is not type((1,2)):
        return bitRep
    width, height = bitRep[:2]
    return Grid(width, height, bitRepresentation= bitRep[2:])

```

```

#####
# Parts you shouldn't have to read #
#####

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

class Actions:
    """
    A collection of static methods for manipulating move actions.
    """
    # Directions
    _directions = {Directions.NORTH: (0, 1),
                    Directions.SOUTH: (0, -1),
                    Directions.EAST: (1, 0),
                    Directions.WEST: (-1, 0),
                    Directions.STOP: (0, 0)}

    _directionsAsList = _directions.items()

    TOLERANCE = .001

    def reverseDirection(action):
        if action == Directions.NORTH:
            return Directions.SOUTH
        if action == Directions.SOUTH:
            return Directions.NORTH
        if action == Directions.EAST:
            return Directions.WEST
        if action == Directions.WEST:
            return Directions.EAST
        return action
    reverseDirection = staticmethod(reverseDirection)

    def vectorToDirection(vector):
        dx, dy = vector
        if dy > 0:
            return Directions.NORTH
        if dy < 0:
            return Directions.SOUTH
        if dx < 0:
            return Directions.WEST
        if dx > 0:
            return Directions.EAST
        return Directions.STOP
    vectorToDirection = staticmethod(vectorToDirection)

    def directionToVector(direction, speed = 1.0):
        dx, dy = Actions._directions[direction]
        return (dx * speed, dy * speed)
    directionToVector = staticmethod(directionToVector)

    def getPossibleActions(config, walls):
        possible = []
        x, y = config.pos
        x_int, y_int = int(x + 0.5), int(y + 0.5)

        # In between grid points, all agents must continue straight
        if (abs(x - x_int) + abs(y - y_int) > Actions.TOLERANCE):
            return [config.getDirection()]

        for dir, vec in Actions._directionsAsList:
            dx, dy = vec
            next_y = y_int + dy
            next_x = x_int + dx
            if not walls[next_x][next_y]: possible.append(dir)

        return possible

    getPossibleActions = staticmethod(getPossibleActions)

    def getLegalNeighbors(position, walls):
        x, y = position
        x_int, y_int = int(x + 0.5), int(y + 0.5)
        neighbors = []

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

for dir, vec in Actions._directionsAsList:
    dx, dy = vec
    next_x = x_int + dx
    if next_x < 0 or next_x == walls.width: continue
    next_y = y_int + dy
    if next_y < 0 or next_y == walls.height: continue
    if not walls[next_x][next_y]: neighbors.append((next_x, next_y))
return neighbors
getLegalNeighbors = staticmethod(getLegalNeighbors)

def getSuccessor(position, action):
    dx, dy = Actions.directionToVector(action)
    x, y = position
    return (x + dx, y + dy)
getSuccessor = staticmethod(getSuccessor)

class GameStateData:
    """
    """
    def __init__( self, prevState = None ):
        """
        Generates a new data packet by copying information from its predecessor.
        """
        if prevState != None:
            self.food = prevState.food.shallowCopy()
            self.capsules = prevState.capsules[:]
            self.agentStates = self.copyAgentStates( prevState.agentStates )
            self.layout = prevState.layout
            self._eaten = prevState._eaten
            self.score = prevState.score
        self._foodEaten = None
        self._capsuleEaten = None
        self._agentMoved = False
        self._lose = False
        self._win = False
        self.scoreChange = 0

    def deepCopy( self ):
        state = GameStateData( self )
        state.food = self.food.deepCopy()
        state.layout = self.layout.deepCopy()
        state._agentMoved = self._agentMoved
        state._foodEaten = self._foodEaten
        state._capsuleEaten = self._capsuleEaten
        return state

    def copyAgentStates( self, agentStates ):
        copiedStates = []
        for agentState in agentStates:
            copiedStates.append( agentState.copy() )
        return copiedStates

    def __eq__( self, other ):
        """
        Allows two states to be compared.
        """
        if other == None: return False
        # TODO Check for type of other
        if not self.agentStates == other.agentStates: return False
        if not self.food == other.food: return False
        if not self.capsules == other.capsules: return False
        if not self.score == other.score: return False
        return True

    def __hash__( self ):
        """
        Allows states to be keys of dictionaries.
        """

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

for i, state in enumerate( self.agentStates ):
    try:
        int(hash(state))
    except TypeError, e:
        print e
        #hash(state)
    return int((hash(tuple(self.agentStates)) + 13*hash(self.food) + 113*
hash(tuple(self.capsules)) + 7 * hash(self.score)) % 1048575 )

def __str__( self ):
    width, height = self.layout.width, self.layout.height
    map = Grid(width, height)
    if type(self.food) == type((1,2)):
        self.food = reconstituteGrid(self.food)
    for x in range(width):
        for y in range(height):
            food, walls = self.food, self.layout.walls
            map[x][y] = self._foodWallStr(food[x][y], walls[x][y])

    for agentState in self.agentStates:
        if agentState == None: continue
        if agentState.configuration == None: continue
        x,y = [int( i ) for i in nearestPoint( agentState.configuration.pos )]
        agent_dir = agentState.configuration.direction
        if agentState.isPacman:
            map[x][y] = self._pacStr( agent_dir )
        else:
            map[x][y] = self._ghostStr( agent_dir )

    for x, y in self.capsules:
        map[x][y] = 'o'

    return str(map)

def _foodWallStr self, x, y:
    if hasFood:
        return '.'
    elif hasWall:
        return '%'
    else:
        return ' '

def _pacStr( self, dir ):
    if dir == Directions.NORTH:
        return 'v'
    if dir == Directions.SOUTH:
        return '^'
    if dir == Directions.WEST:
        return '>'
    return '<'

def _ghostStr( self, dir ):
    return 'G'
    if dir == Directions.NORTH:
        return 'M'
    if dir == Directions.SOUTH:
        return 'W'
    if dir == Directions.WEST:
        return '3'
    return 'E'

def initialize( self, layout, numGhostAgents ):
    """
    Creates an initial game state from a layout array (see layout.py).
    """
    self.food = layout.food.copy()
    self.capsules = layout.capsules[:]
    self.layout = layout
    self.score = 0

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

self.scoreChange = 0

self.agentStates = []
numGhosts = 0
for isPacman, pos in layout.agentPositions:
    if not isPacman:
        if numGhosts == numGhostAgents: continue # Max ghosts reached already
        else: numGhosts += 1
    self.agentStates.append( AgentState( Configuration( pos, Directions.STOP),
isPacman) )
self._eaten = [False for a in self.agentStates]

try:
    import boinc
    _BOINC_ENABLED = True
except:
    _BOINC_ENABLED = False

class Game:
    """
    The Game manages the control flow, soliciting actions from agents.
    """

    def __init__( self, agents, display, rules, startingIndex=0, muteAgents=False,
catchExceptions=False ):
        self.agentCrashed = False
        self.agents = agents
        self.display = display
        self.rules = rules
        self.startingIndex = startingIndex
        self.gameOver = False
        self.muteAgents = muteAgents
        self.catchExce
        self.moveHisto
        self.totalAgen
        self.totalAgen
        self.agentTimeout = False
        import cStringIO
        self.agentOutput = [cStringIO.StringIO()]

    def getProgress(self):
        if self.gameOver:
            return 1.0
        else:
            return self.rules.getProgress(self)

    def _agentCrash( self, agentIndex, quiet=False):
        "Helper method for handling agent crashes"
        if not quiet: traceback.print_exc()
        self.gameOver = True
        self.agentCrashed = True
        self.rules.agentCrash(self, agentIndex)

OLD_STDOUT = None
OLD_STDERR = None

def mute(self, agentIndex):
    if not self.muteAgents: return
    global OLD_STDOUT, OLD_STDERR
    import cStringIO
    OLD_STDOUT = sys.stdout
    OLD_STDERR = sys.stderr
    sys.stdout = self.agentOutput[agentIndex]
    sys.stderr = self.agentOutput[agentIndex]

def unmute(self):
    if not self.muteAgents: return
    global OLD_STDOUT, OLD_STDERR
    # Revert stdout/stderr to originals

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro


```
sys.stdout = OLD_STDOUT
sys.stderr = OLD_STDERR
```

```
def run( self ):
    """
    Main control loop for game play.
    """
    self.display.initialize(self.state.data)
    self.numMoves = 0

    ###self.display.initialize(self.state.makeObservation(1).data)
    # inform learning agents of the game start
    for i in range(len(self.agents)):
        agent = self.agents[i]
        if not agent:
            self.mute(i)
            # this is a null agent, meaning it failed to load
            # the other team wins
            print "Agent %d failed to load" % i
            self.unmute()
            self._agentCrash(i, quiet=True)
            return
        if ("registerInitialState" in dir(agent)):
            self.mute(i)
            if self.catchExceptions:
                try:
                    timed_func = TimeoutFunction(agent.registerInitialState,
int(self.rules.getMaxStartupTime(i)))
                    try:
                        start_time = time.time()
                        timed_func(self.state.deepCopy())
                        time
                        self
                    except:
                        print "% i
                        self.unmute()
                        self.agentTimeout = True
                        self._agentCrash(i, quiet=True)
                        return
                except Exception, data:
                    self._agentCrash(i, quiet=False)
                    self.unmute()
                    return
            else:
                agent.registerInitialState(self.state.deepCopy())
                ## TODO: could this exceed the total time
                self.unmute()

    agentIndex = self.startingIndex
    numAgents = len( self.agents )

    while not self.gameOver:
        # Fetch the next agent
        agent = self.agents[agentIndex]
        move_time = 0
        skip_action = False
        # Generate an observation of the state
        if 'observationFunction' in dir( agent ):
            self.mute(agentIndex)
            if self.catchExceptions:
                try:
                    timed_func = TimeoutFunction(agent.observationFunction,
int(self.rules.getMoveTimeout(agentIndex)))
                    try:
                        start_time = time.time()
                        observation = timed_func(self.state.deepCopy())
                    except TimeoutFunctionException:
                        skip_action = True
```

```

        move_time += time.time() - start_time
        self.unmute()
    except Exception, data:
        self._agentCrash(agentIndex, quiet=False)
        self.unmute()
        return
    else:
        observation = agent.observationFunction(self.state.deepCopy())
        self.unmute()
    else:
        observation = self.state.deepCopy()

    # Solicit an action
    action = None
    self.mute(agentIndex)
    if self.catchExceptions:
        try:
            timed_func = TimeoutFunction(agent.getAction,
int(self.rules.getMoveTimeout(agentIndex)) - int(move_time))
            try:
                start_time = time.time()
                if skip_action:
                    raise TimeoutFunctionException()
                action = timed_func( observation )
            except TimeoutFunctionException:
                print "Agent %d timed out on a single move!" % agentIndex
                self.agentTimeout = True
                self._agentCrash(agentIndex, quiet=True)
                self.unmute()
                return
            move_time += time.time() - start_time

```

Assignment Project Exam Help

```

        if move_time > self.rules.getMaxMoveTime(agentIndex):
            print "Agent %d exceeded the maximum move time: %f" % (agentIndex, self.rules.getMaxMoveTime(agentIndex))
            self.agentTimeout = True
            self._agentCrash(agentIndex, quiet=True)
            self.unmute()

        self.totalAgentTimes[agentIndex] += move_time
        #print "Agent: %d, time: %f, total: %f" % (agentIndex, move_time, self.totalAgentTimes[agentIndex])
        if self.totalAgentTimes[agentIndex] > self.rules.getMaxTotalTime(agentIndex):
            print "Agent %d ran out of time! (time: %1.2f)" % (agentIndex, self.totalAgentTimes[agentIndex])
            self.agentTimeout = True
            self._agentCrash(agentIndex, quiet=True)
            self.unmute()
            return
        self.unmute()
    except Exception, data:
        self._agentCrash(agentIndex)
        self.unmute()
        return
    else:
        action = agent.getAction(observation)
        self.unmute()

    # Execute the action
    self.moveHistory.append( (agentIndex, action) )
    if self.catchExceptions:
        try:

```

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

        self.state = self.state.generateSuccessor( agentIndex, action )
    except Exception, data:
        self.mute(agentIndex)
        self._agentCrash(agentIndex)
        self.unmute()
        return
    else:
        self.state = self.state.generateSuccessor( agentIndex, action )

    # Change the display
    self.display.update( self.state.data )
    ###idx = agentIndex - agentIndex % 2 + 1
    ###self.display.update( self.state.makeObservation(idx).data )

    # Allow for game specific conditions (winning, losing, etc.)
    self.rules.process(self.state, self)
    # Track progress
    if agentIndex == numAgents + 1: self.numMoves += 1
    # Next agent
    agentIndex = ( agentIndex + 1 ) % numAgents

    if _BOINC_ENABLED:
        boinc.set_fraction_done(self.getProgress())

    # inform a learning agent of the game result
    for agent in self.agents:
        if "final" in dir( agent ) :
            try:
                self.mute(agent.index)
                agent.final( self.state )
                self.unmute()
            except Exception, data:
                if not s:
                    self._ag
                    self.unm
                return
    self.display.finish()

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro