# Artificial Intelligence Spring 2022
# Group Game Project (Pacman)

Team23 0816175

## Strategy:

In my pacman, according to the distance from different object, the Pacman I designed will do the respective action, so first step is to determine the distance of different objects pass in the function getStep(playerStat,ghostStat,propsStat):

```
def getStep(playerStat, ghostStat, propsStat):
```

So I use BFS to search each object's position, and first step is to simplify the coordinates of original pass-in coor, cause it would cost too much time on calculating distance, so I reduced 400x400 points coordinates to 16x16 field, although it will reduce the precision of the calculation and Pacman's action. To achieve the time limit of 0.04s, I divide each coordinate by 25, to make them transform to coordinates of 16x16

```
P_start_x = int(playerStat[0]/25)
P_start_y = int(playerStat[1]/25)
```

```
int(prop[1]/25),int(prop[2]/25))
```

So now I have each of props, players, ghost, coordinates, which is presented as [x,y] of [16,16] present.

Next step, in each iteration, I use four list to store different object's cooridinates, and I'll use them to record any object on the map , now.

```
pellet_cooridinates = []
power_cooridinates = []
bomb_cooridinates = []
ghost_cooridinates = []
```

Then, I use four variable to obtain the closest different four distance of 4 objects on maps now. And init the bombdis = 999, because there's not always having bombs on the maps. To prevent error occur, we set the distance in advanced.

## How to get closest distance by cooridinates:

So, I use BFS to search distance between two coordinates, parameter are two coordinates,

```
def BFS_Player(start,prop_coo):
```

And it will start at player's position, and do search until it find the closest object that pass in, and prop_coo is actually a list of props coordinates, it's not coordinates of one point, instead, it's all of coordinates of all the objects on the map. To da so, BFS can search until it get the closest spot, and return the distance & direction toward this position.

```
queue = []
direction = {}
direction[(start[0],start[1])] = 'right'
queue.append(start)
```

But, however can we get the direction toward the position? We have to also record each step's direction, and use **backtrace**.

So we use queue (actually a list in python data structure) to do BFS, and a direction to record all of the direction on the way to the closest spot.

```
# left
if(((node[0]-1,node[1]) not in direction) and (vertical_wall[node[0]][node[1]] != 1)):
    # visit.append([node[0],node[1]-1])
    queue.append([node[0]-1,node[1]])
    # print('queue插入',[node[0],node[1]-1])
    # print('他的方向是 left\n')
    direction[(node[0]-1,node[1])] = 'left'
```

So take direction ‘left’ as example, we have to check if the position is traverse already, and if there’s a wall to block our BFS, combine two above condition, we append the next spot into our queue and record the direction of the position.

## Backtrace:

So, once the BFS reach the closest spot, it will halt to add more position into queue, and start to da traceback.


```
# left
if(((node[0]-1,node[1]) not in direction) and (vertical_wall[node[0]][node[1]] != 1)):
    # visit.append([node[0],node[1]-1])
    queue.append([node[0]-1,node[1]])
    # print('queue插入',[node[0],node[1]-1])
    # print('他的方向是 left\n')
    direction[(node[0]-1,node[1])] = 'left'
```


So once the loop stop, which means the BFS get the closest spot, and we have to traceback to original point that pass in BFS initially.

```
if((node[0],node[1]) in prop_coo):
    # do backtrace
    now_point_x = node[0]
    now_point_y = node[1]
    first_action = 'right'
    distance = 0
```

So node is the point now we traverse, and if it’s in prop_coo, which means BFS reach the closest spot. And we start to traceback.

```
while((now_point_x!=start[0] or now_point_y !=start[1])):
    # print((now_point_x,now_point_y),'\'s direction is',
    if(direction[(now_point_x,now_point_y)] == 'right'):
        now_point_x -= 1
        first_action = 'right'
    elif(direction[(now_point_x,now_point_y)] == 'left'):
        now_point_x += 1
        first_action = 'left'
    elif(direction[(now_point_x,now_point_y)] == 'up'):
        now_point_y += 1
        first_action = 'up'
    elif(direction[(now_point_x,now_point_y)] == 'down'):
        now_point_y -= 1
        first_action = 'down'
    distance += 1
return distance,first_action,node
```

So, in traceback, we traverse from destination back to start, and if the direction is ‘right’, which means we just move to right to get to the spot, so we have to move out x = x-1, vice versa. And also to record the distance while we traverse. So until both x & y coordinates match the start, we return the distance now, and the first step toward this spot, and also the node (which is the props’s coordinates for debug usage)

## Last step:

So, now we have each direction,distance of all the objects on the map. And our team decide to chase the powerball as the first priority, so we use variable finaldir as the final direction we’re going to pass into Pacman, and set rules below:

```
if len(power_cooridinates)>0:
    finaldir = powerdir
if(playerStat[3] > 1500 and ghostdis < 20):
    finaldir = ghostdir
```

These two lines of code is quite easy to understand, if there's power_ball, we chase them, if we're in power mode, and ghost distance is close enough, we chase the ghost, because it gets more score to eats ghost in power mode.

```
if(ghostdis < 4 and (ghost_position[0] == P_start_x or ghost_position[1] == P_start_y)):
    if ghostdis == finaldir:
        if finaldir == 'right' or finaldir == 'left':
            # 上面不是牆
            if(parallel_wall[P_start_x][P_start_y] != 1):
                finaldir = 'up'
            # 下面不是牆
            if(parallel_wall[P_start_x][P_start_y+1] != 1):
                finaldir = 'down'
        else :
            # 左邊不是牆
            if(vertical_wall[P_start_x][P_start_y] != 1):
                finaldir = 'left'
            # 右邊不是牆
            if(vertical_wall[P_start_x+1][P_start_y] != 1):
                finaldir = 'right'
```

Then we check the bomb & ghost dis, if they're too close and there's no wall along side, we turn the way to hide from ghost & bomb.

Last one, is our Pacman get stuck (turn 180°)on the corner or a wall, we determine the direction of Pacman now, and set it not to turn 180°, but 90°, and we wouldn't turn around once we detect possible stuck, but to count the timestep, to prevent it turn immediately even Pacman is actually not stuck in corner.

```
elif(finaldir == 'left' and  (vertical_wall[P_start_x][P_sta
    finaldir = random.choice(['up','down'])
    count += 1
else:
    count = 0
if(count >= 2):
    finaldir = random.choice(['left','right','up','down'])
    count = 0
```