PAC····•
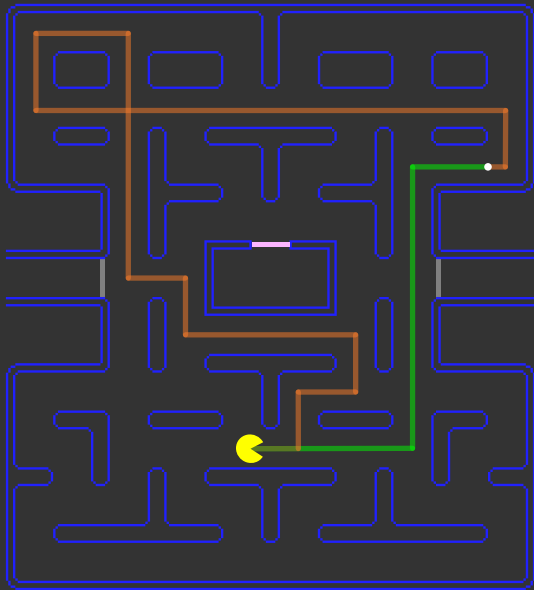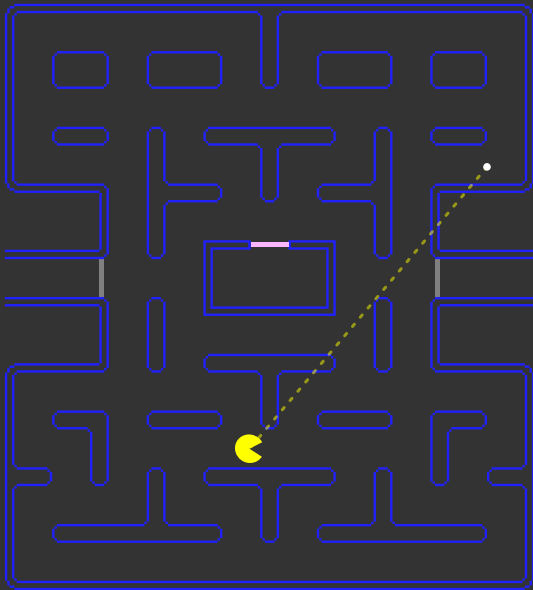
BOT(ᗣ˚)

Let's reduce Pacman to a small case:
One pellet left
No ghosts

Two choices:
Orange Path
Green Path

Why is Green better?

This brings us to the idea of an Objective function
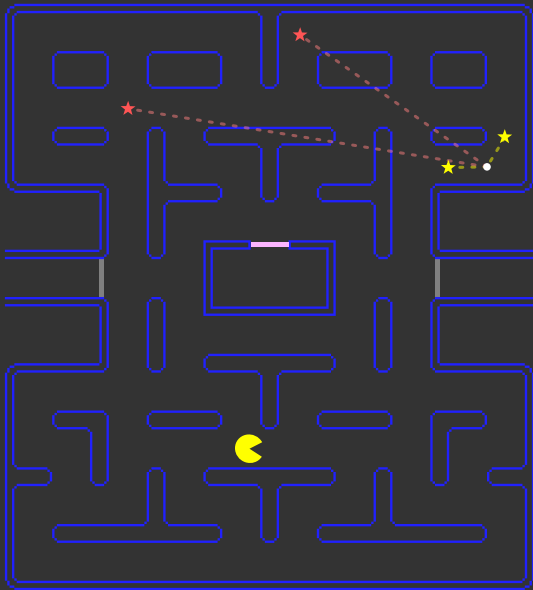
J(path) = sum of costs

What is a simple cost we can minimize?

J(path) = total distance traveled

Realistically, we can't evaluate every path...
but we can rule out bad ones

How can we guess how (un)favorable an
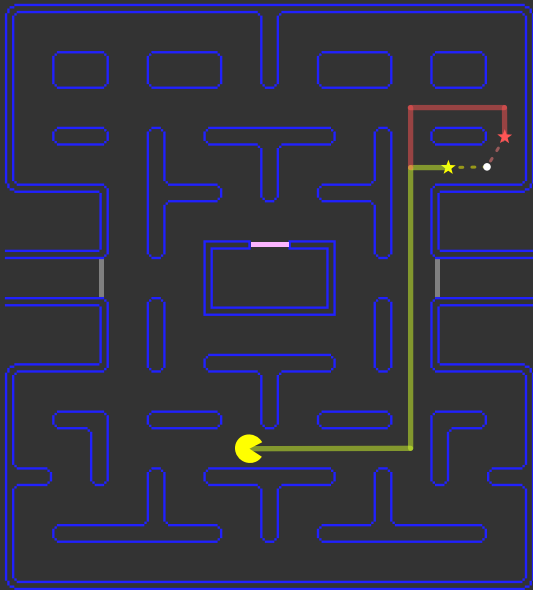intermediate position is?

We can use a heuristic:

H(position) = Euclidean distance to the goal

We can set intermediate goals based on the heuristic

If the heuristic cost H(position) is higher, it's less optimal as an intermediate goal

But that's not the whole story ...

We can set intermediate goals based on the heuristic

If the heuristic cost H(position) is higher, it's less optimal as an intermediate goal

But that's not the whole story ...

... the heuristic ignores the effort to get to that intermediate goal

We can think of another component to the cost, what we'll call the "guaranteed cost":

G(subpath, position) = true minimum cost of subpath to position
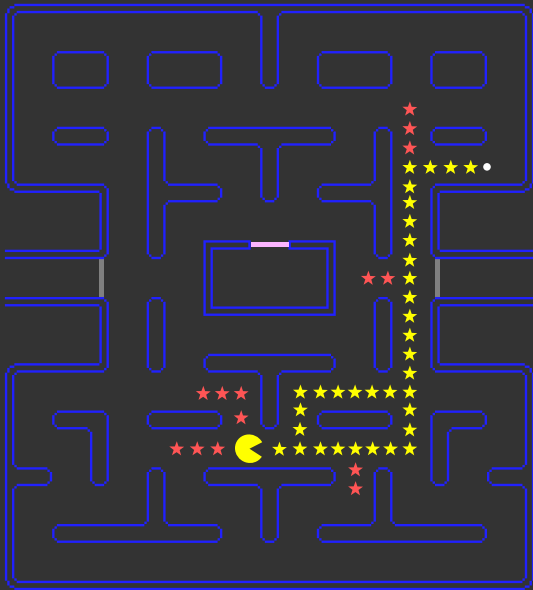
Algorithm (A-Star):

```
initial = pacman.location
positions = [initial]
while positions is not empty:
    current = argmin(cost, positions)
    positions.remove(current)
    if current == goal:
        SUCCESS
    for direction in [UP, LEFT, DOWN, RIGHT]:
        if current.valid_move(direction):
            new = current.move(direction)
            new.G = current.G + 1
            new.H = distance(new, goal)
            new.cost = new.G + new.H
            positions.add(new)
FAILURE
```
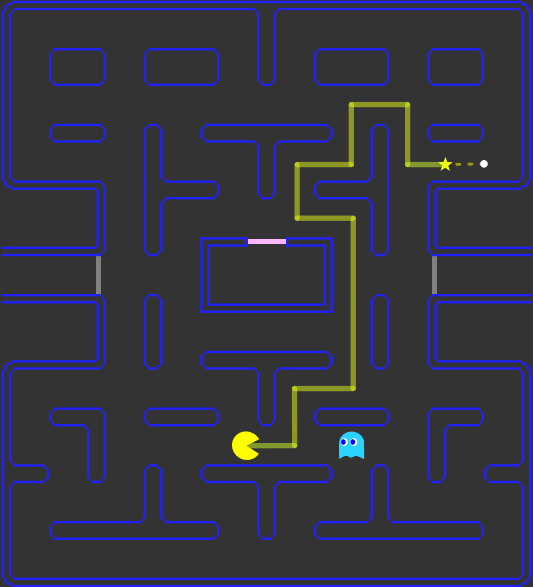
Modifications we used last year:

Ghost Avoidance:
Predict ghost behaviors
Terminate unsafe paths early
Extra cost associated with being near ghosts

Ghost Chasing:
After super pellet, set the target to nearest ghost
"Predict" ghost behaviors
Stay away from the ghost lair

Fruit:
If the fruit is close enough, set the target to it

Targeting:
Choose targets close to super pellets
Eat as many pellets as possible along the way
Plan for when ghosts speed up