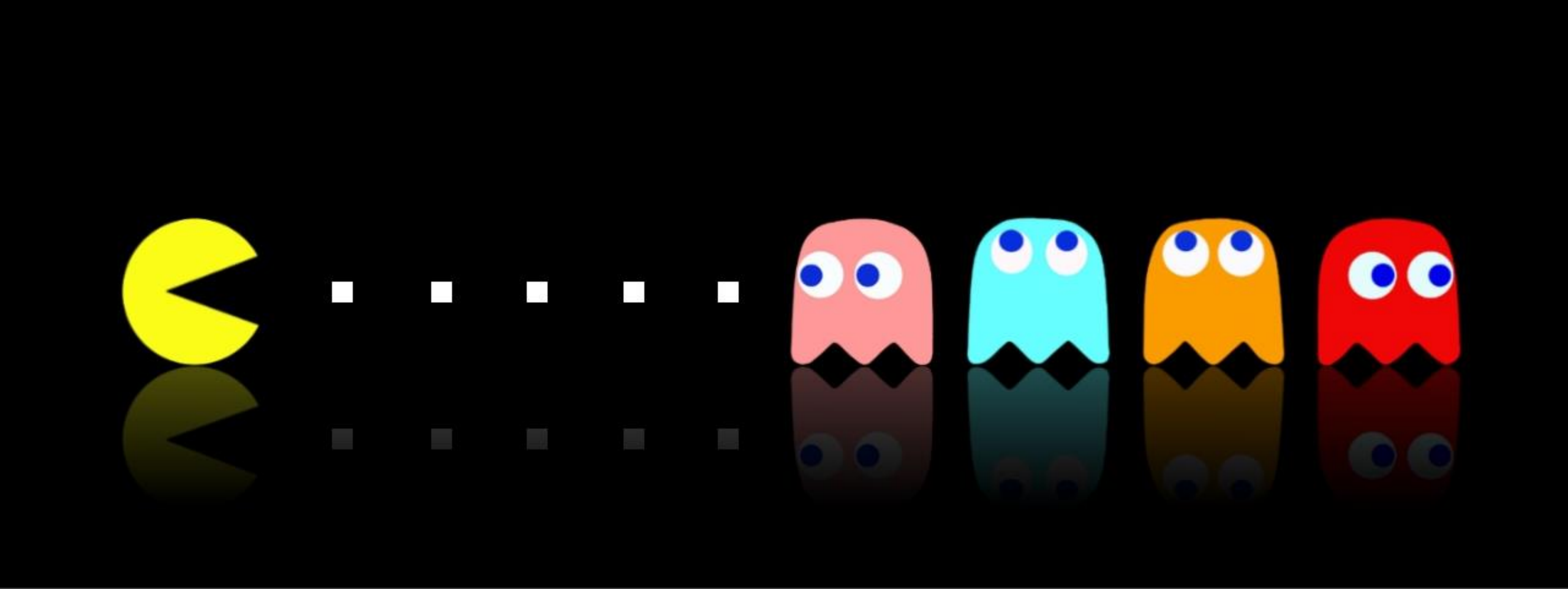
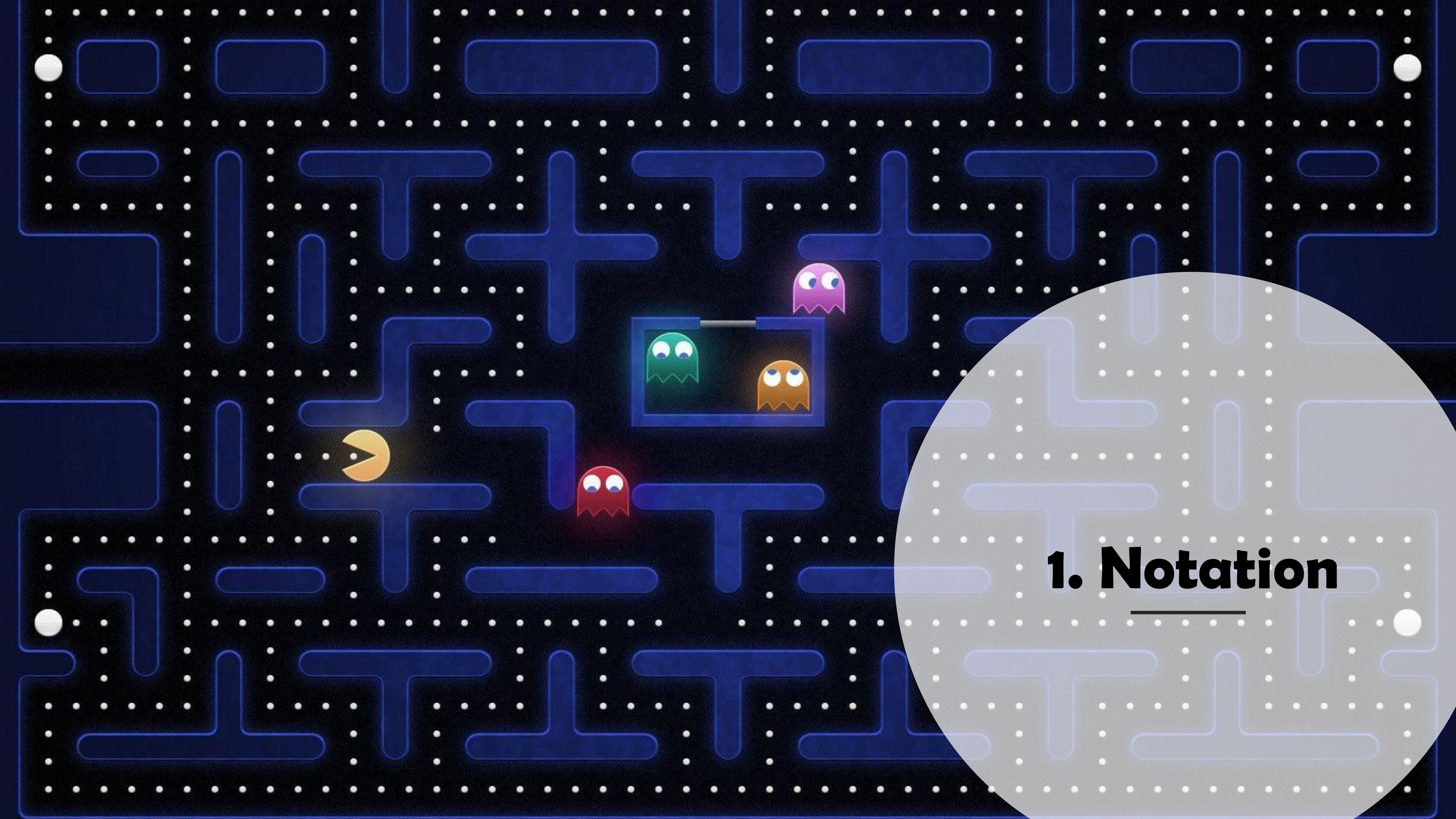
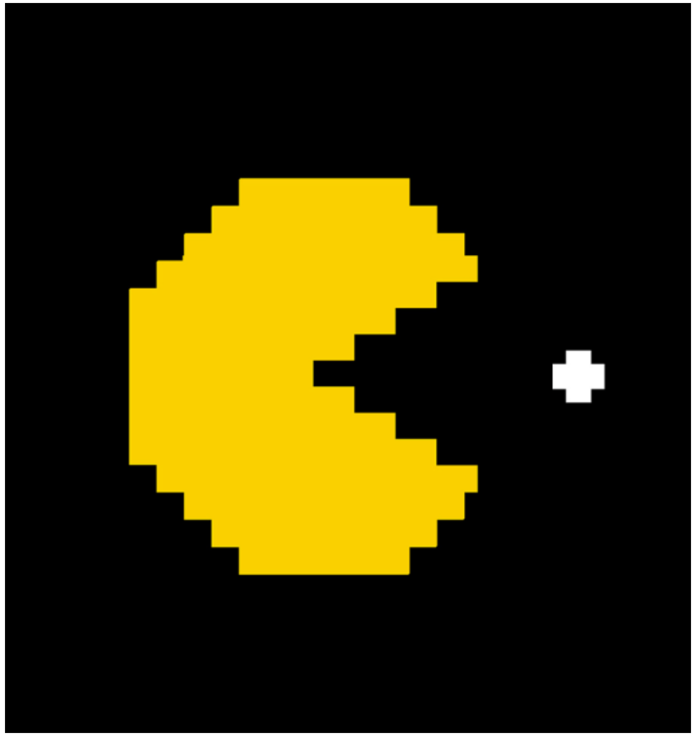


CS 3600 Project 3 Review

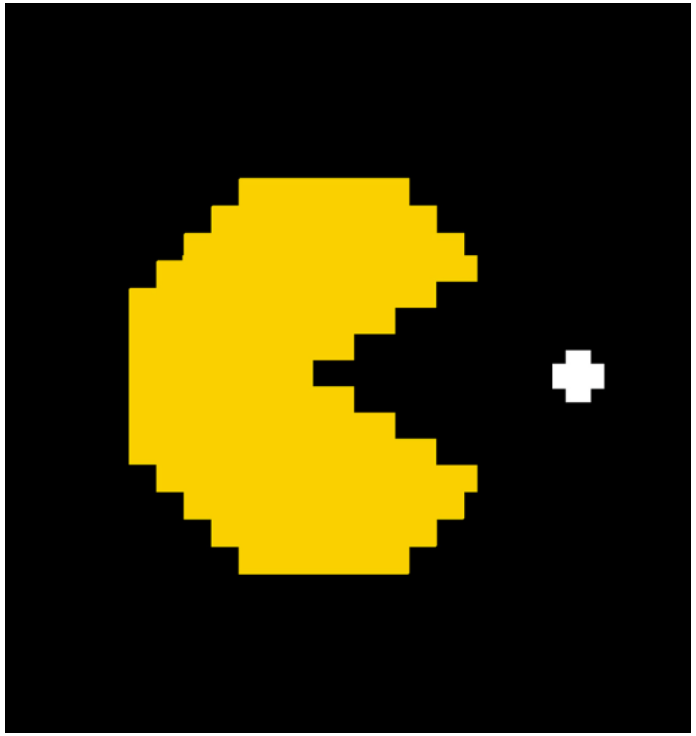


1. Notation



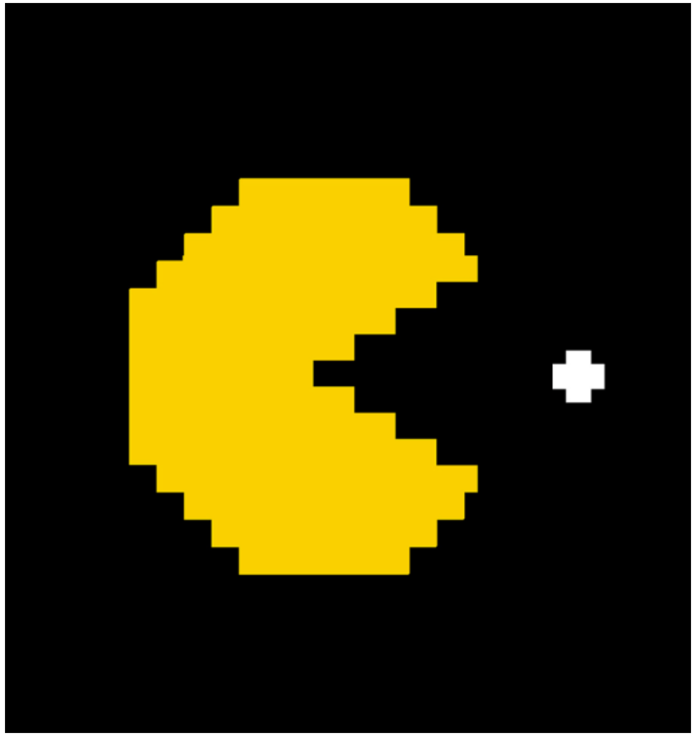
Notation: legalPositions

- Let's call **c** any coordinate which is a legal position
- Access via attribute **self.legalPositions**



Notation: beliefs

- Of the form P(G = c)
  - The probability the random variable representing a ghost's position **G** is equal to position **c**
- Access via **self.beliefs**
  - Should be a counter object



Notation: trueDistance

- trueDistance = ManhattanDistance(Pacman, c)
  - The true distance from Pacman to legal position **c**
- P(G = c) = α P(dist from Pacman to G = trueDistance)
  - Example: [c1] [c2] [c3]
  - If Pacman is at c2 and P(dist from c2 to ghost = 1) = 1:
    - P(G = c1) = ?
    - P(G = c3) = ?

Notation: noisyDistance

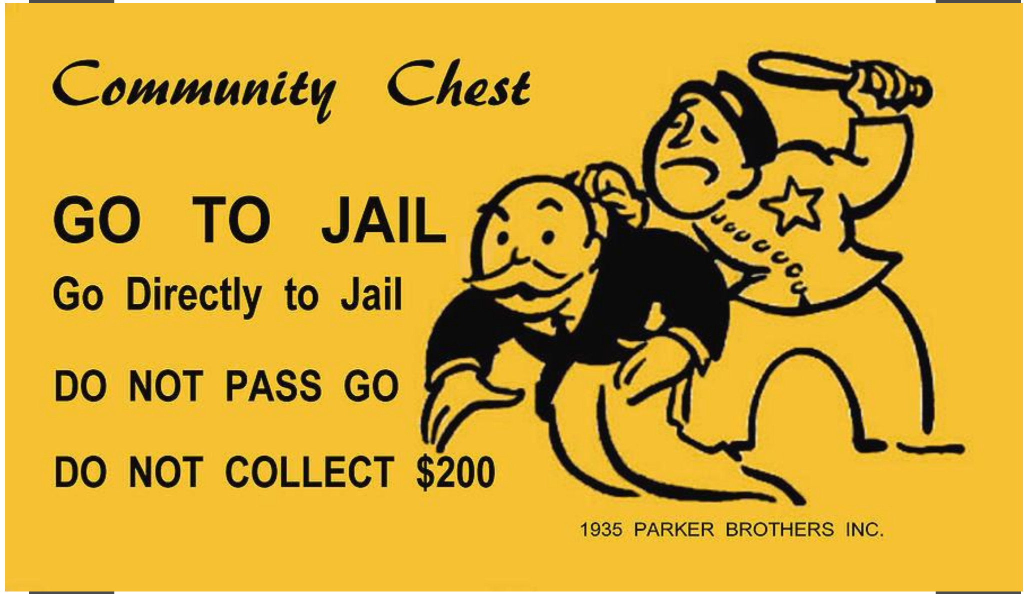
- This is your **observation** parameter
- Remember, real-world sensors are noisy (this is why robots fall down)
- Luckily, we have a probability distribution describing this "noise"

Notation: emissionModel

- Represents:
  - P(noisyDistance|trueDistance)

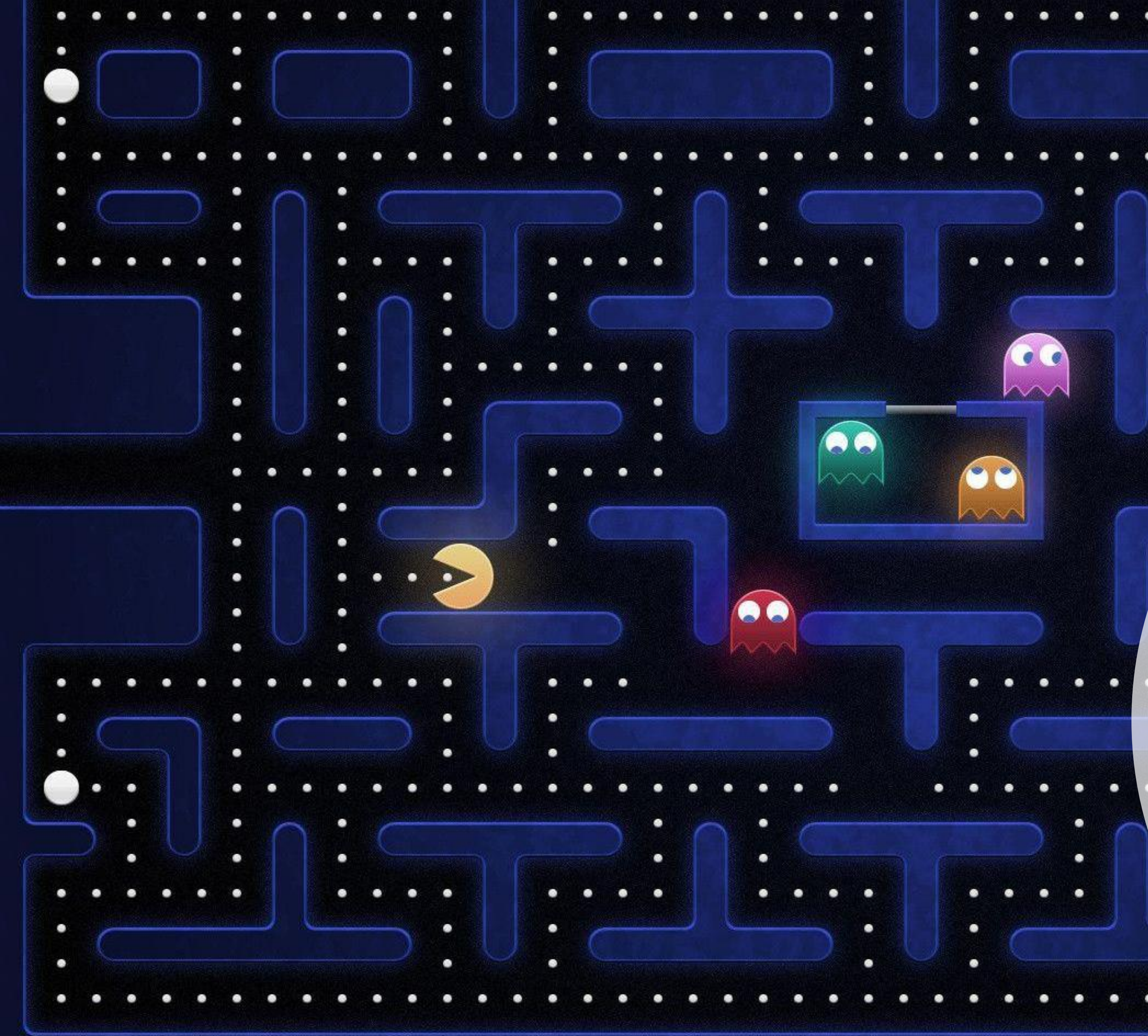
- P(noisyDistance|trueDistance)
  = **emissionModel[trueDistance]**

There is a different emission model for each noisyDistance



Notation: jail

- When a ghost has been captured, the belief that the ghost is anywhere but the jail cell must be 0%, and we must have 100% belief that the ghost is in the jail position
- You can check if a ghost has been captured by Pacman by checking if it has a noisyDistance of None



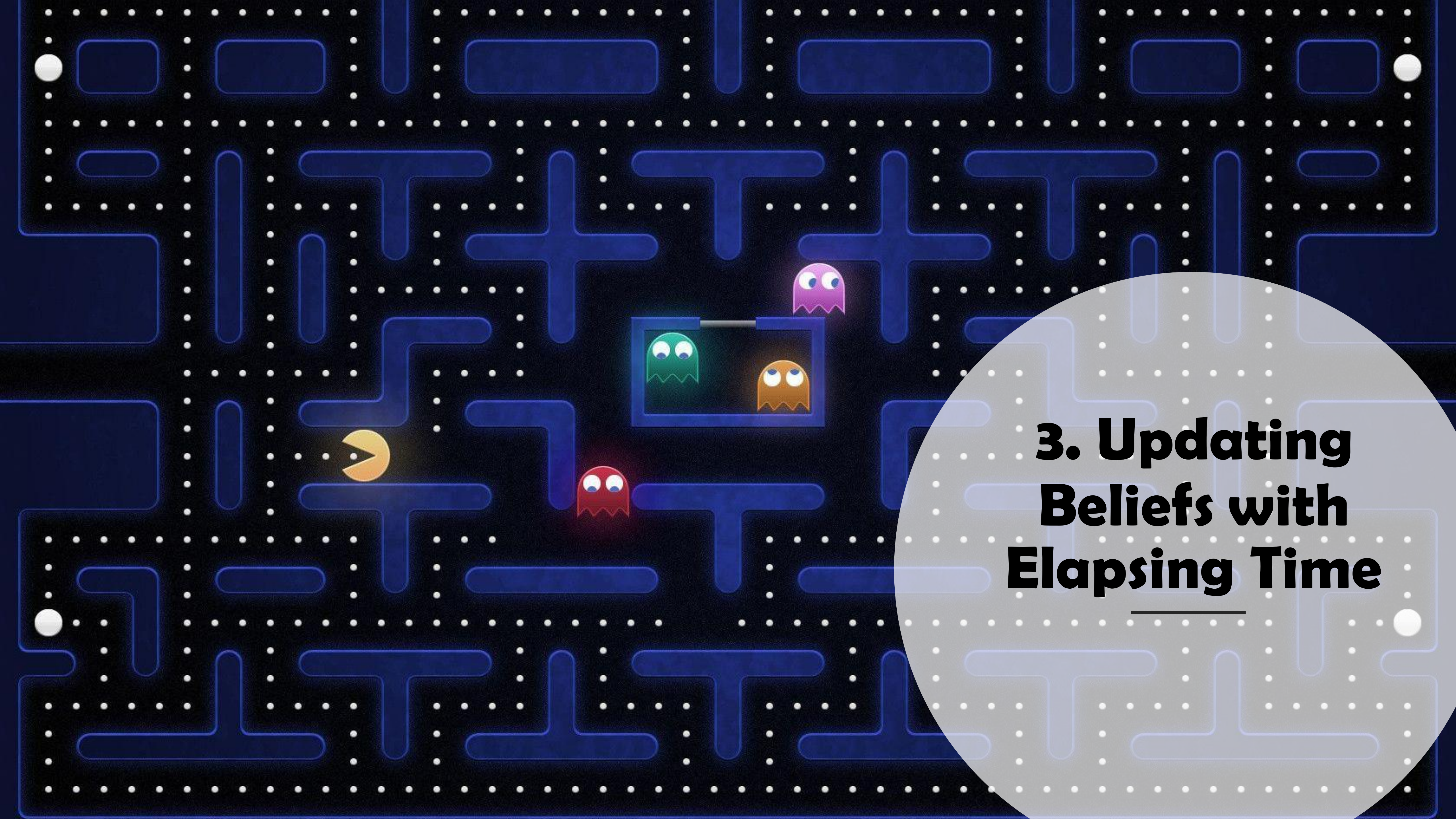
2. Updating Beliefs with Observation (Q1)

Updating Beliefs with Observation (Q1)

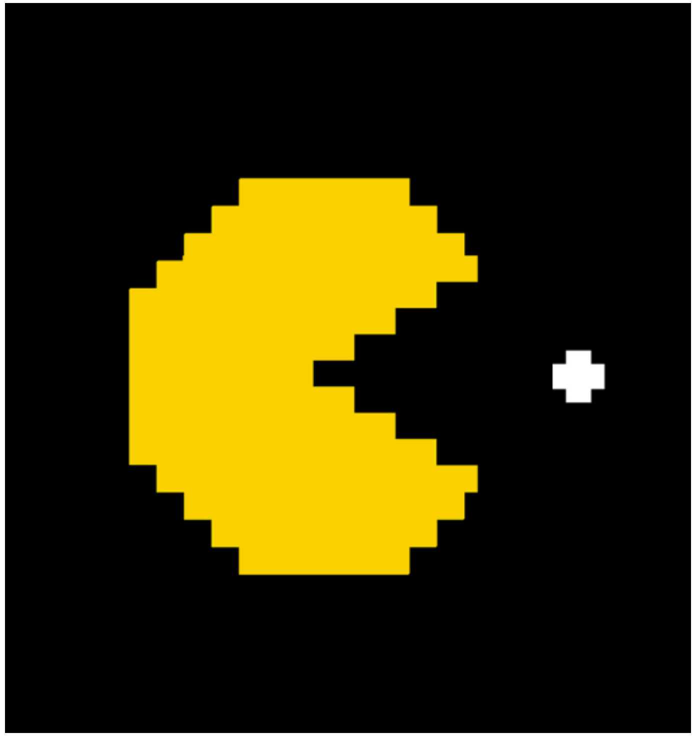
- What we have:
  - P(noisyDistance|trueDistance)
    = **emissionModel[trueDistance]**
  - P(trueDistance)
    - Our prior <u>beliefs</u>

- What we want
  - P(c | noisyDistance)
    = α P (trueDistance to c | noisyDistance)

- Start with math before trying to code



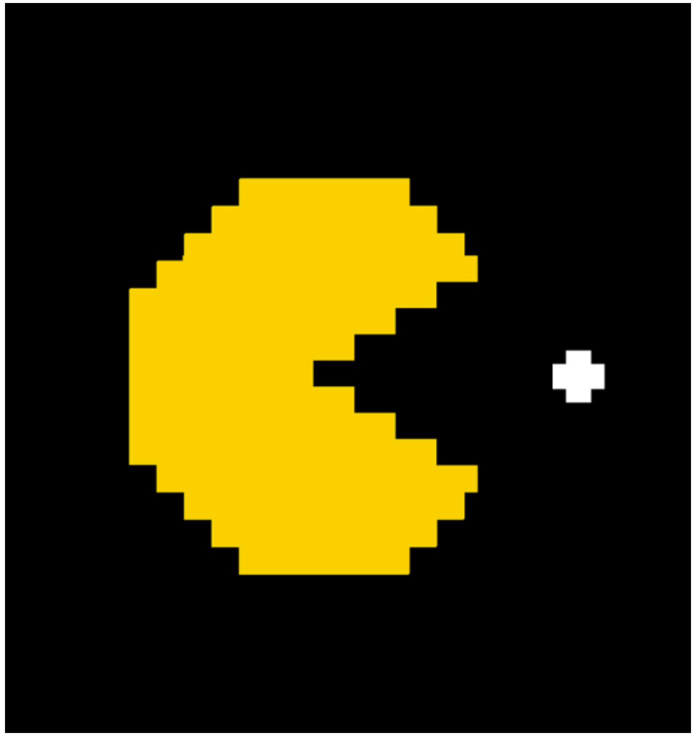
3. Updating Beliefs with Elapsing Time



Updating Beliefs with Time

- **GIVEN: ghost was at another location**
- **`GetPositionDistribution`: gives probability distribution of where ghost may be at <u>next</u> time increment**
- ****Use what we already BELIEVE about ghost's location**

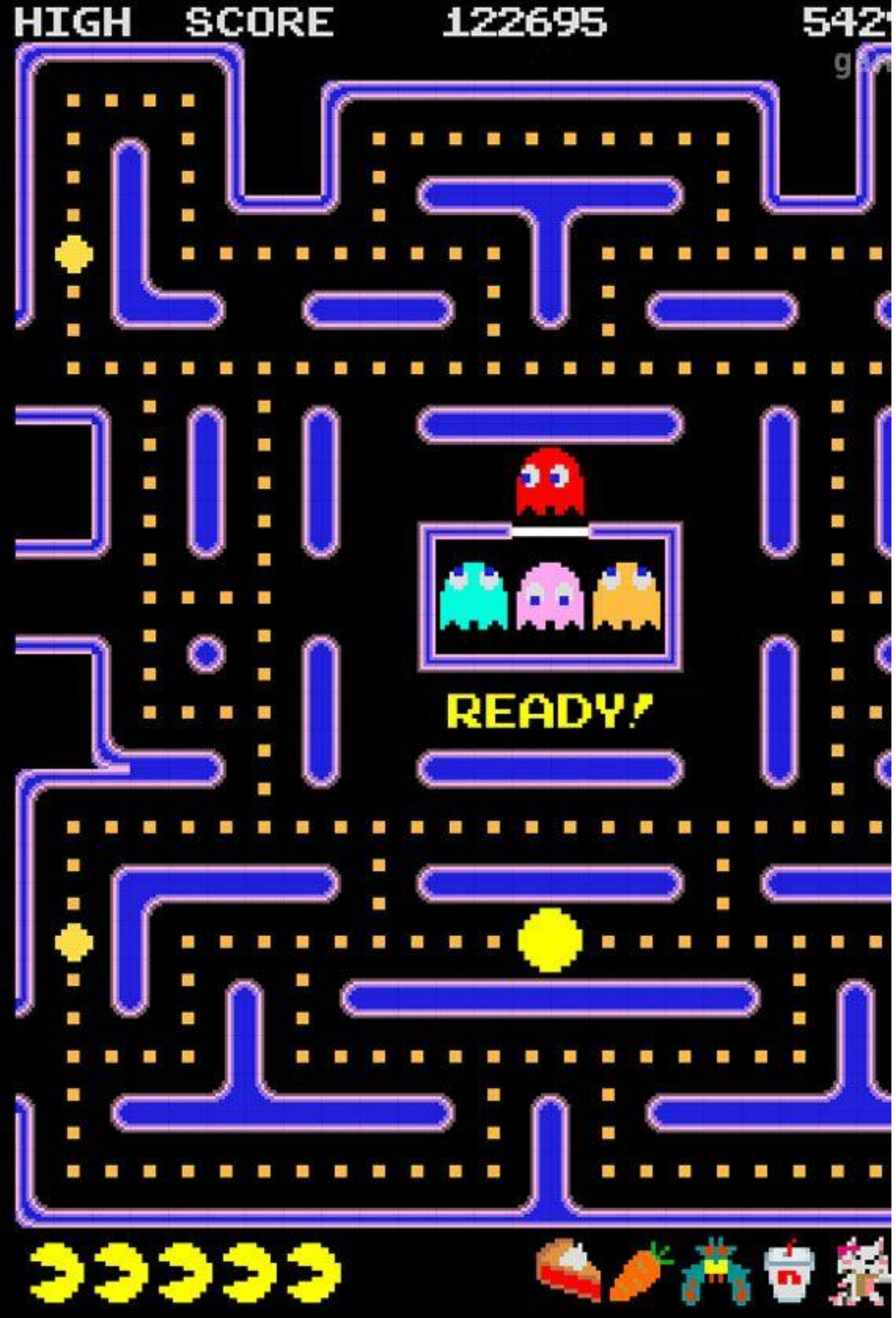
4. Hunting Multiple Ghosts



Hunting Multiple Ghosts

- Must <u>decide</u> how to hunt
- Follow a greedy approach:
  - Determine location ghost will most likely be at (the "true" location)
  - Identify closest ghost to Pacman
  - Choose action to bring Pacman as close as possible to ghost

5. Particle Filtering



Why use Particles?

- Exact filtering is recursive and expensive
- Need to calculate $P(X_t|e_{1:t})$ *for every state* X_t at each time step

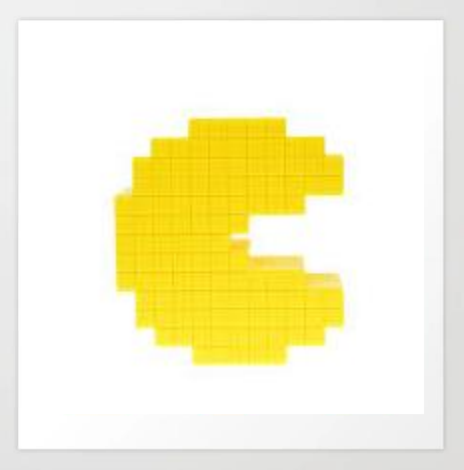
Intuition: Particle Filtering

- Particles are **votes**
- If a state has more particles, it means that it is more likely that Pacman is in this state
- Can translate between belief distributions and particle list

Intuition: Particle Filtering

- Example:
  - You have a bag of 100 particles.
  - 90 of them represent "RAIN RAIN ENDLESS RAIN"
  - 10 of them represent "HOT HOT SUN"
  - So the belief distribution of the current weather
    - 90% "RAIN RAIN ENDLESS RAIN"
    - 10% "HOT HOT SUN"





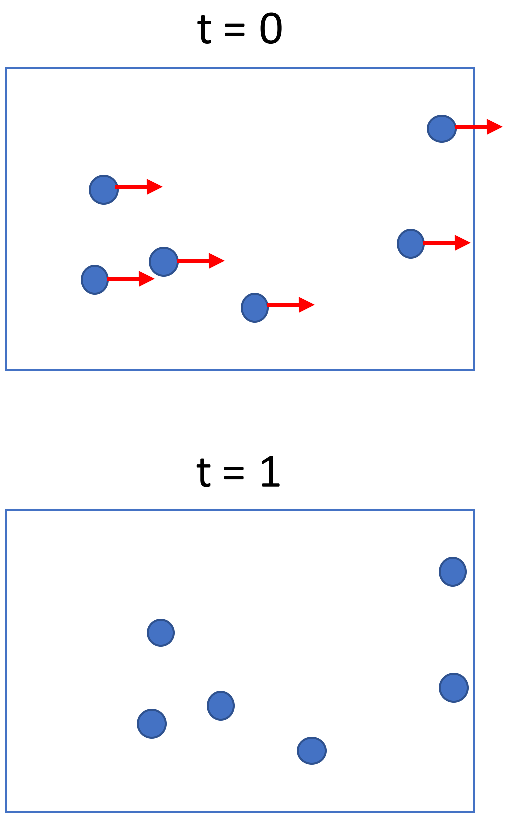
Initializing Uniformly

- We initialize the set of particles according to the prior distribution
- In this case, we do not know where Pacman is at all → Hence a **Uniform Prior**
- Each state should have the **same number of particles** in the beginning
- Try to distribute all particles as evenly as possible, even if the number of particles may not be divisible by the number of states

Updating Particles with Time Elapse

Time Elapse in Particle Filter

- Analogous to time elapse in exact inference
  - Instead of updating the probabilities of being in each state, we sample the transition of each particle
- For each particle:
  1. Find its corresponding transition distribution
  2. Sample a new particle from the distribution



Updating Particles with Observations

Comedian Example

	$P(F_t=T \mid F_{t-1})$	$P(F_t=F \mid F_{t-1})$
$F_{t-1} = T$	0.8	0.2
$F_{t-1} = F$	0.5	0.5

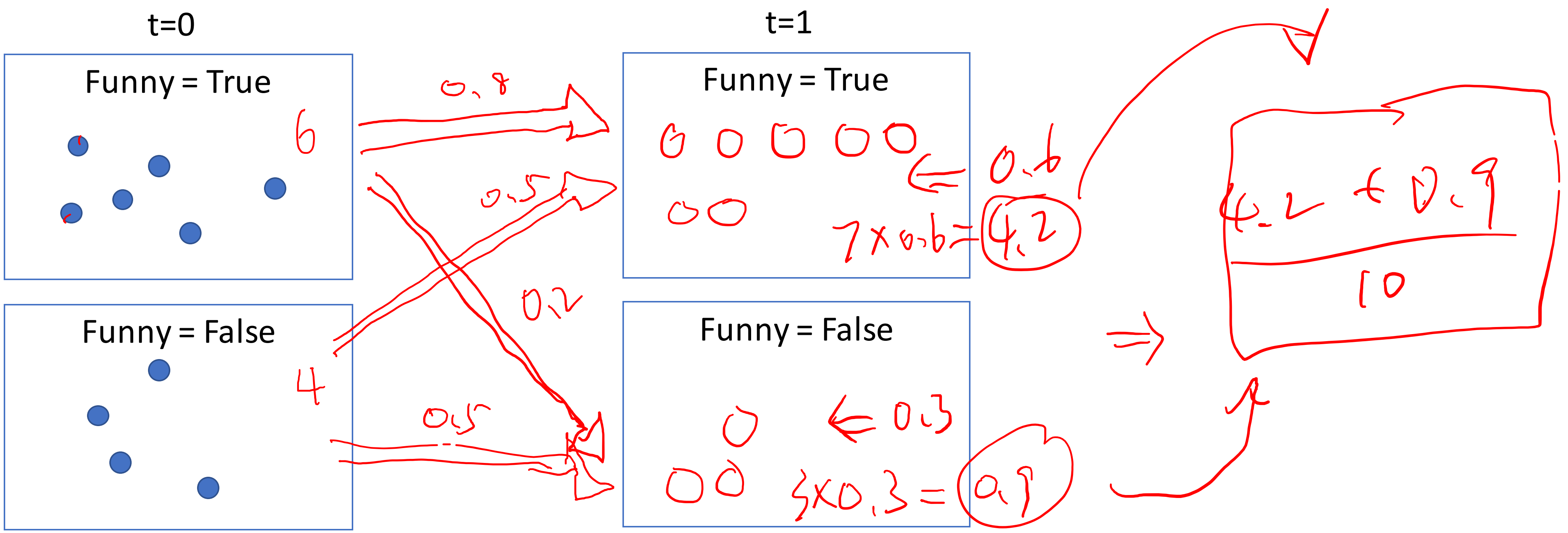
transition model

	$P(V_t=T \mid F_t)$	$P(V_t=F \mid F_t)$
$F_t = T$	0.6	0.4
$F_t = F$	0.3	0.7

sensor model

There's a comedian who is new in town, and he puts on a show at Radio City. When he is funny (denoted by F), he would likely change his voice (denoted by V). We assume that at time t = 0, the probability distribution of him being funny is <0.6, 0.4> (yes, he is often funny).

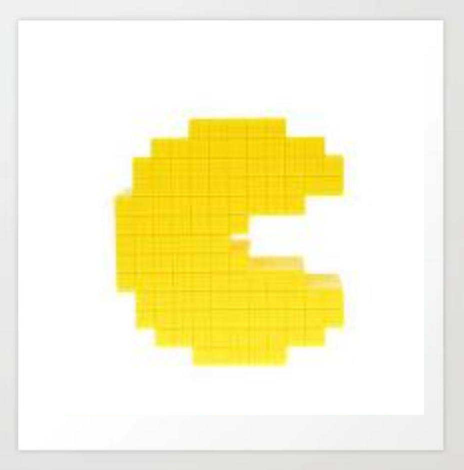
At t = 1, he is observed to change his voice.



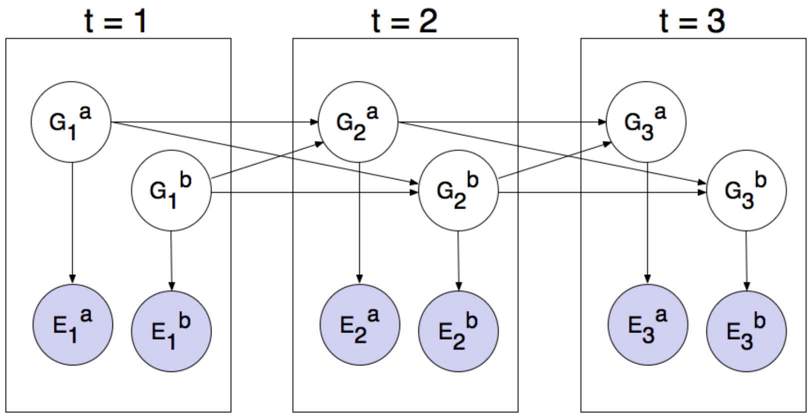
Intuition

- The higher the sum of weights of particles in a state, the more likely Pacman is in that state
- After transitioning, we would need to **re-weigh the particles** based on the emission model
- The **evidence / observation** gives us intuition of whether this state is likely

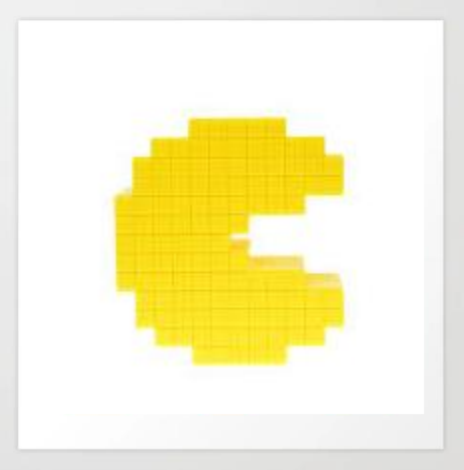
4. Joint Particle Filter



Introduction



- Up to now, we've used separate particle filters for the ghosts
- Now, ghosts do **not** behave independently
  - For example, ghosts may move away from each other if they're too close
- So we need to use one **joint** particle filter
- If we have *k* ghosts, each particle will be a *k*-tuple of ghost positions (c_1, c_2, …, c_k)

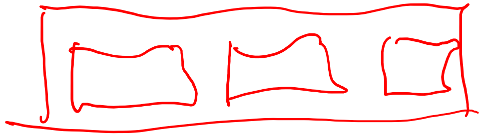


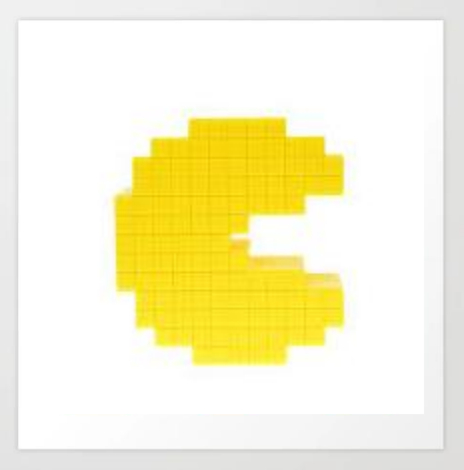
Creating Particles

- All possible particles consist of all legal positions of the *k* ghosts
  - Ends up being all length-*k* permutations of `legalPositions`
- For initialization, we again use a uniform prior
  - Here, uniform over all length-*k* permutations of `legalPositions`
  - These permutations can be created with Python's `itertools.product` module over `legalPositions`
- We won't have enough particles to cover all permutations, so instead we randomly sample a subset from the prior
  - Done in code by taking list of permutations, **shuffling them**, and then taking the first *N* tuples

# of particles

↳ random.shuffle()

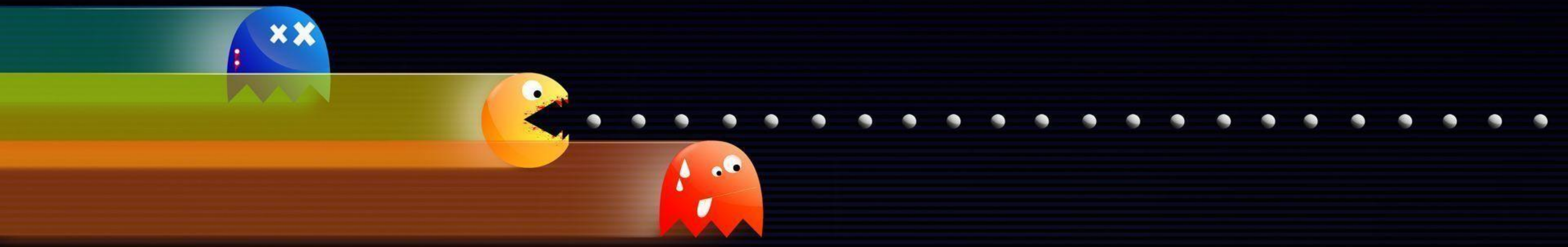




Updating with Observations and Time Elapse

- `observeState` and `elapseTime` will be pretty similar to original particle filter's
- Need to do additional book-keeping since particle is a *tuple of ghosts* instead of a single ghost
  - For example, if some ghost is eaten, all particles must have that ghost go to jail
  - If all particles have zero weight, must reinitialize by calling `initializeParticles`, and then (for each particle) putting any already-eaten ghosts in jail

noisyDist = None



Questions?