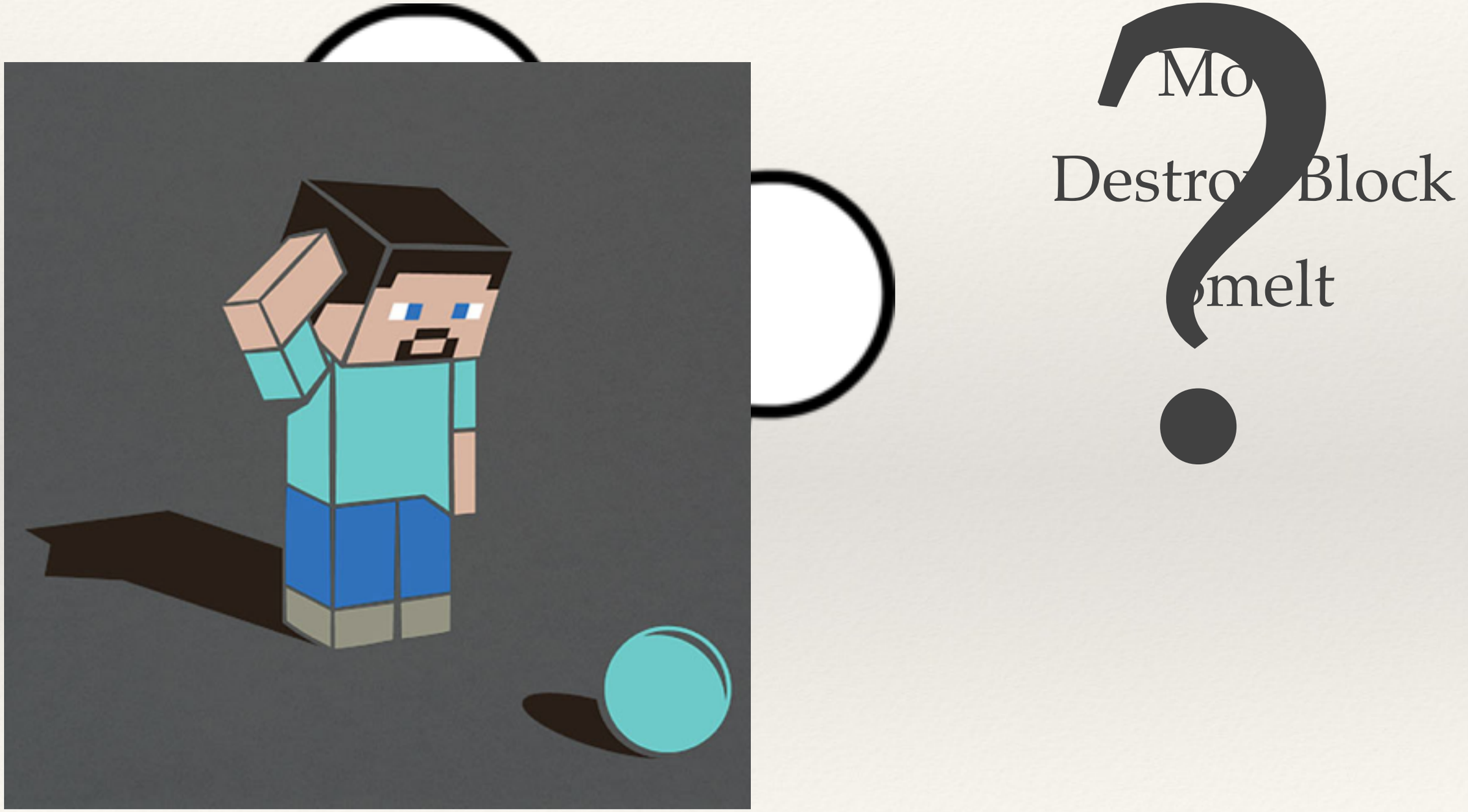
D Ellis Hershkowitz

Reinforcement Learning and Pals

Motivations



Motivations

Motor actions

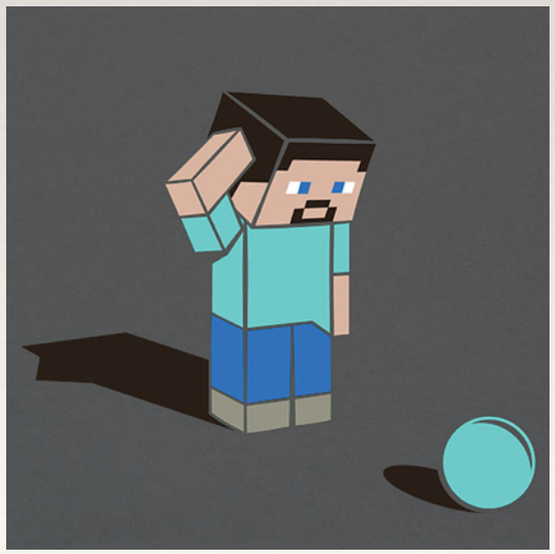
Put ingredient in container

Put object A in place B

?

Transition Dynamics

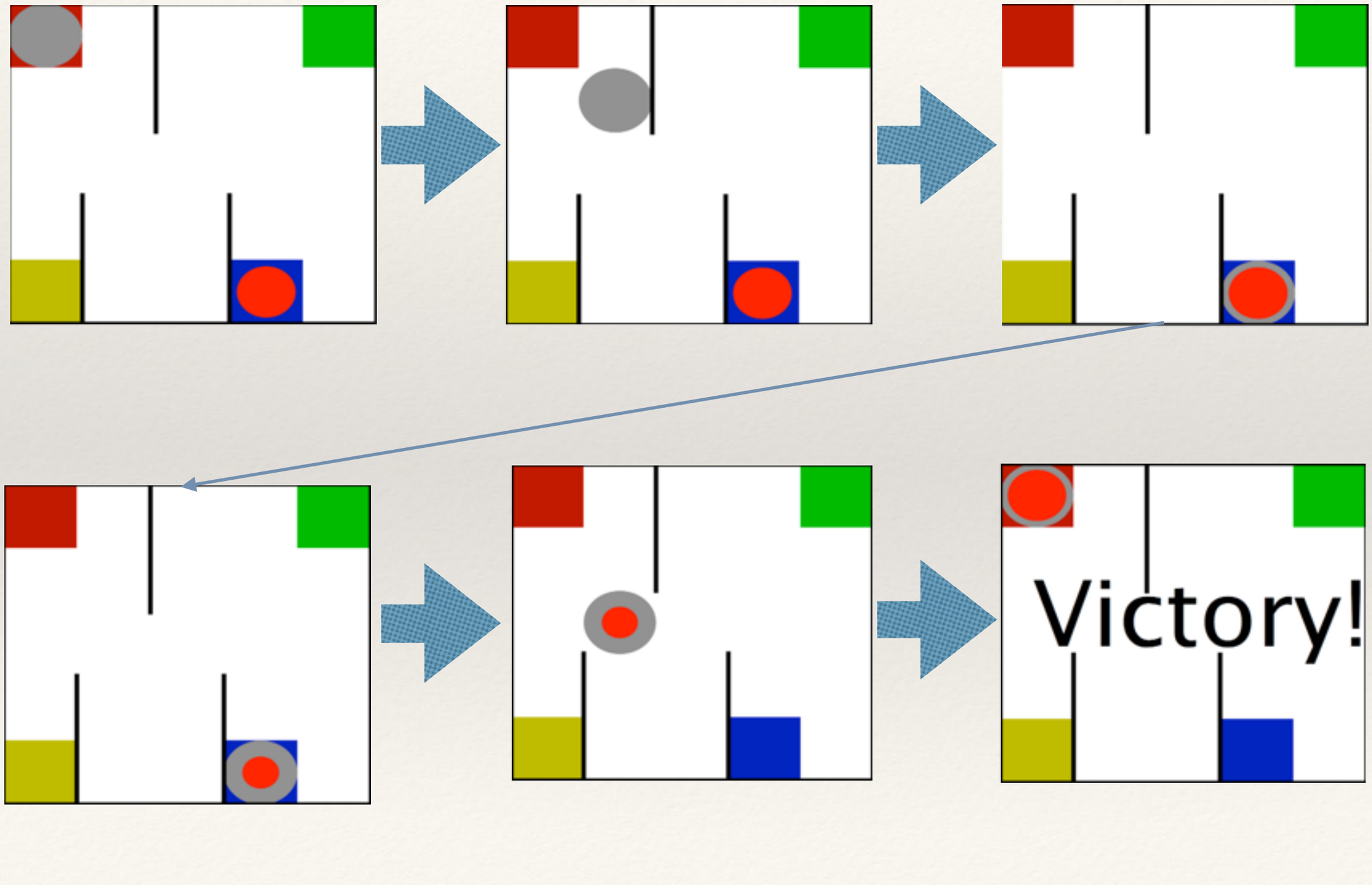
- Solving problems requires understanding how one's actions affect the world around oneself (transition dynamics).
- Often these are obfuscated and so we can't hope to solve our planning problems.



Goal

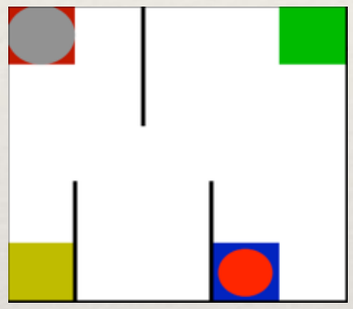
- Learn complex transition dynamics in a family of large state-space MDPs.

Taxi Domain



Markov Decision Processes (MDPs)

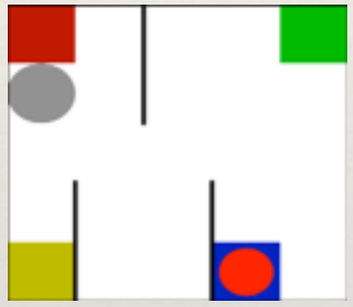
- An MDP is a five tuple of:
  - 1. States (some terminal)



  - 2. Actions

North, east, south, west, pickup, drop off

  - 3. Transition Dynamics



  - 4. Reward Function

(uniform negative)

  - 5. Discount Factor

RMAX

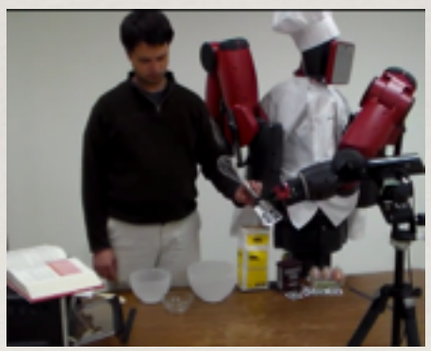
- Input: (state, actions, resulting state) tuples.
- Output: transition dynamics.
- Algorithm idea: remember what actions landed you where for each state. Hallucinate amazing states if transition dynamics not known.

Object-Oriented Thinking

- A natural object-oriented, exploitable structure to problems, think:

- Minecraft

- Robots



Object-Oriented MDPs (OO-MDPs)

- An OO-MDP specifies object classes each of which have attributes that range over some domain.

  E.g. taxi (xAtt, yAtt), passenger (xAtt, yAtt, locationAtt, inTaxiAtt)

- An OO-MDP state is defined by instantiations of these objects.

```
taxi (taxi)
        xAtt:    0
        yAtt:    3
```

```
passenger0 (passenger)
        xAtt:    3
        yAtt:    0
        locationAtt:     red
        inTaxiAtt:       false
```

- An OO-MDP also defines predicates over its states.

  E.g. wallToNorthOfTaxi, wallToSouthOfTaxi, passengerInTaxi etc.

What now?

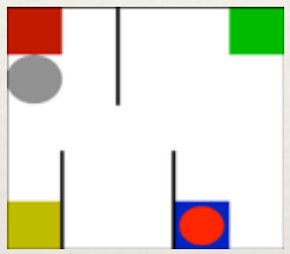
- Have nice OO formulations of problems in OOMDPs.
- RMAX doesn't leverage OOMDPs.
- We want to leverage OO thinking for large problems.

DOORMAX

- OOMDP take on RMAX.
- Input: (state, action and resulting state) triples and PFs.
- Output: transition dynamics.
- A Knows-What-It-Knows (KWIK) algorithm.

DOORMAX Lingo

- Conditions



Wall to west

↓

000100

- Xor of conditions

110 xor 010 = *10

- Matching of conditions

**0 matches 101

**0 does not match 100

- Effects

```
Assigmenteffect of setting value to 3.0 for xAtt of passengers
```

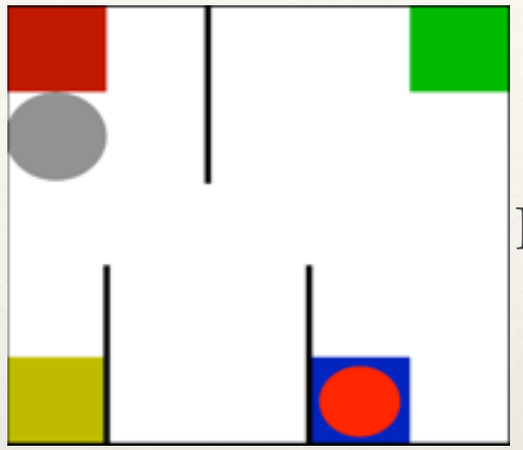
```
ArithmeticEffect of adding -1.0 to xAtt of taxis
```

- Predictions

```
Prediction for west's effect on xAtt of passenger
        condition:(011011)
        effectLearner: ArithmeticEffect of adding -1.0 to xAtt of passengers
```

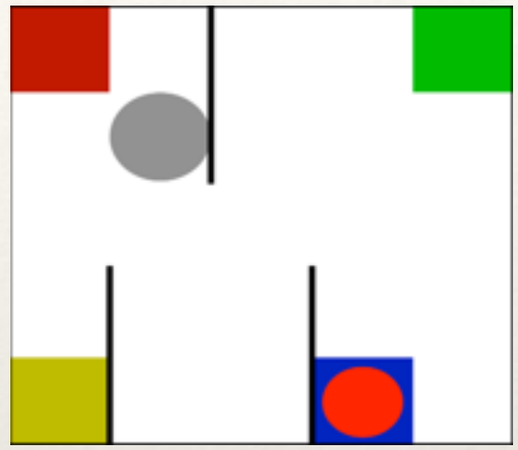
DOORMAX Learning

- Given a state, an action and a resulting state:
- If a failure condition, store condition for action and return.
- Hypothesize all possible effects.
- <u>For each hypothesized effect:</u>
  - If there is no prediction that predicts the effect instantiate one.
  - If there is a prediction, update its condition.
- Rule out contradictory effect types (overlapping conditions or more than k of an effect type for an action).



(000100)

East->



<u>Knowledge of East</u>

Pred: Add 1 to Agent X on (0*0***)

Pred: Assign 1 to Agent X on (010***)

Pred: Assign 1 to Agent X on (***011)

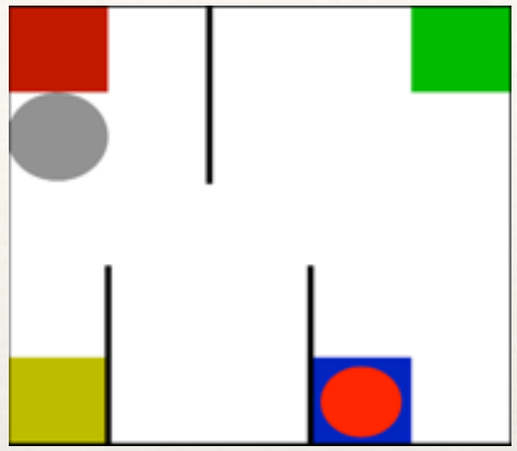
Pred: Assign 1 to Agent X on (000100)

Failure: Assign 1 to Agent X on (010000), (111000)

Failure Conditions: (010000), (111000)

DOORMAX Prediction

- Given a state and action:
- If failure condition for the action, predict current state.
- else pool the effects of all predictions whose conditions match the current state into a pool of effects.
- If any effects contradict or pool is empty, return don't know.
- else return the state that results from applying all effects.

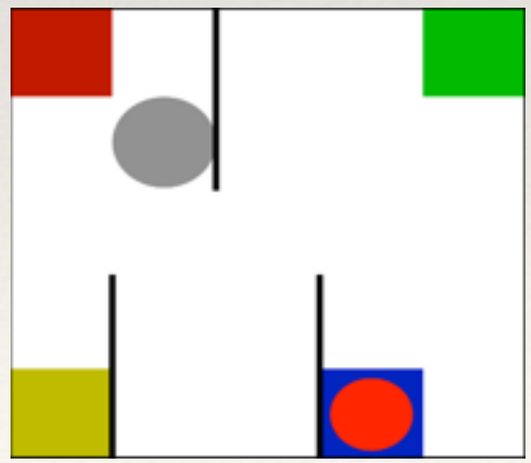


(000100)

```
Prediction for east's effect on xAtt of taxi
        condition:(000100)
        effectLearner: Assigmenteffect of setting value to 1.0 for xAtt of taxis
```

```
Prediction for east's effect on xAtt of taxi
        condition:(*0****)
        effectLearner: ArithmeticEffect of adding 1.0 to xAtt of taxis
```

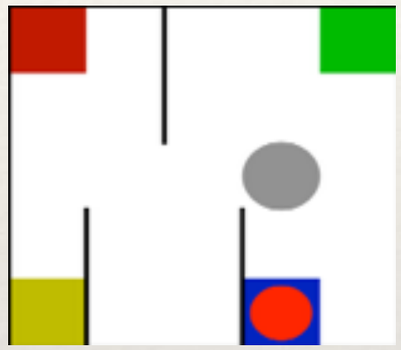
Failure Conditions: (none)



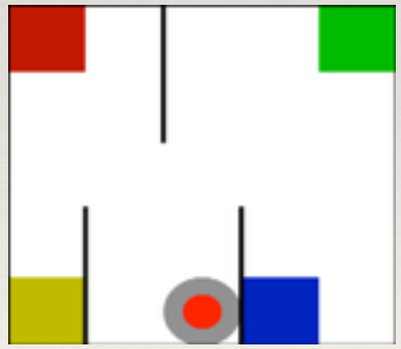
DOORMAX Problem(1/2)

- Suppose taxi has moved around but hasn't encountered the passenger much

```
Prediction for west's effect on xAtt of taxi
        condition:(***0**)
        effectLearner: ArithmeticEffect of adding -1.0 to xAtt of taxis
```



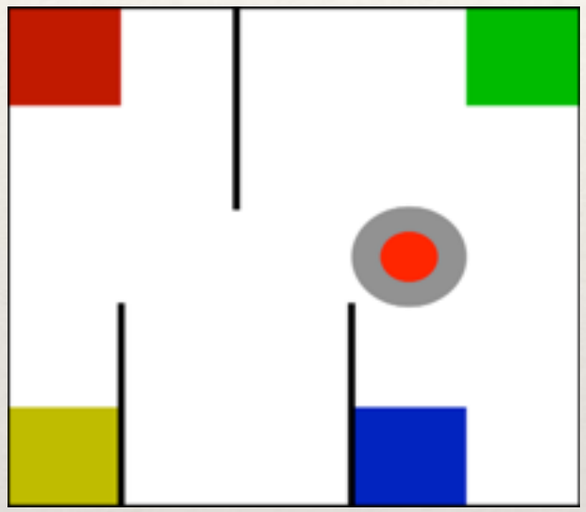
```
Prediction for west's effect on xAtt of passenger
        condition:(011011)
        effectLearner: ArithmeticEffect of adding -1.0 to xAtt of passengers
```



Failure Conditions: (none)

DOORMAX Problem(2/2)

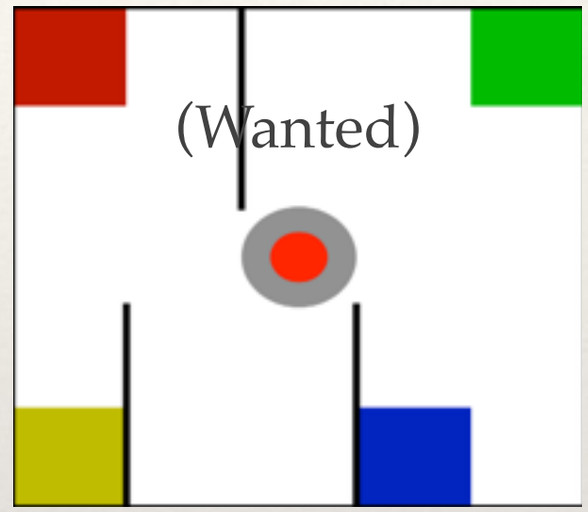
- Consider the following scenario:



Passenger in taxi

(000011)

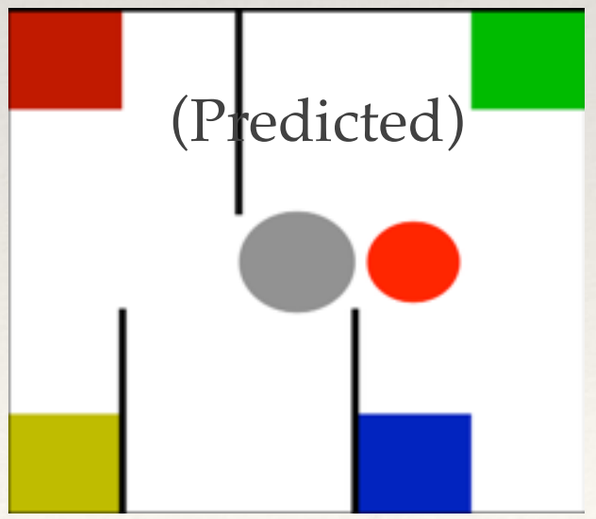
Taxi at passenger




```
Prediction for west's effect on xAtt of taxi
        condition:(***0**)
        effectLearner: ArithmeticEffect of adding -1.0 to xAtt of taxis
```

```
Prediction for west's effect on xAtt of passenger
        condition:(011011)
        effectLearner: ArithmeticEffect of adding -1.0 to xAtt of passengers
```

Predicted Effect: subtract 1 from agent's X



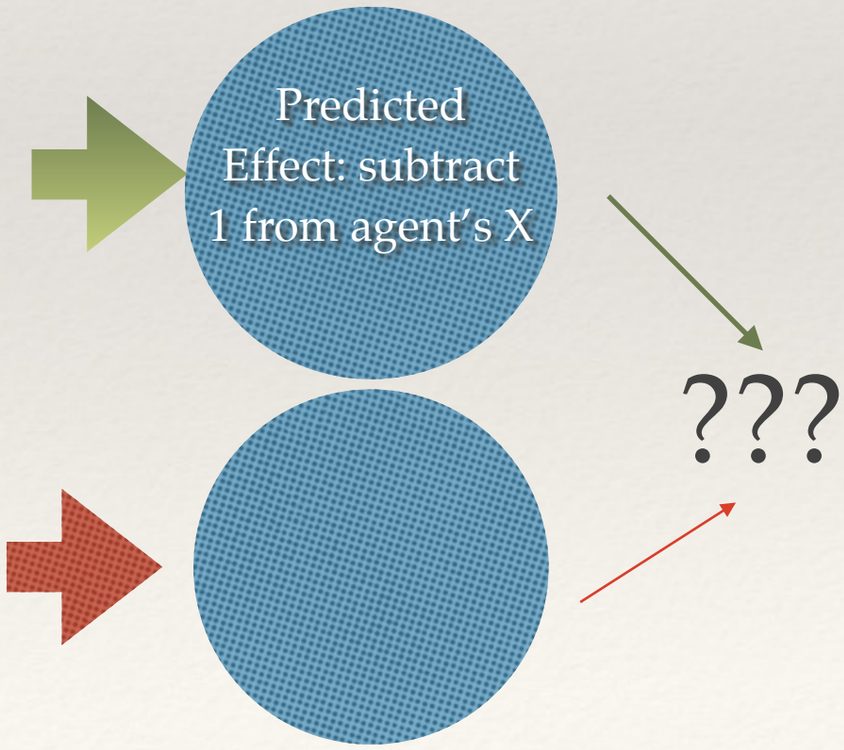
Failure Conditions: (none)

DOORMAX Solutions (1/2)

- Store individual pools and failure conditions for each attribute.
- Return don't know if a pool that is both empty and for which the corresponding action is not a failure condition.

```
Prediction for west's effect on xAtt of taxi
        condition:(***0**)
        effectLearner: ArithmeticEffect of adding -1.0 to xAtt of taxis
```

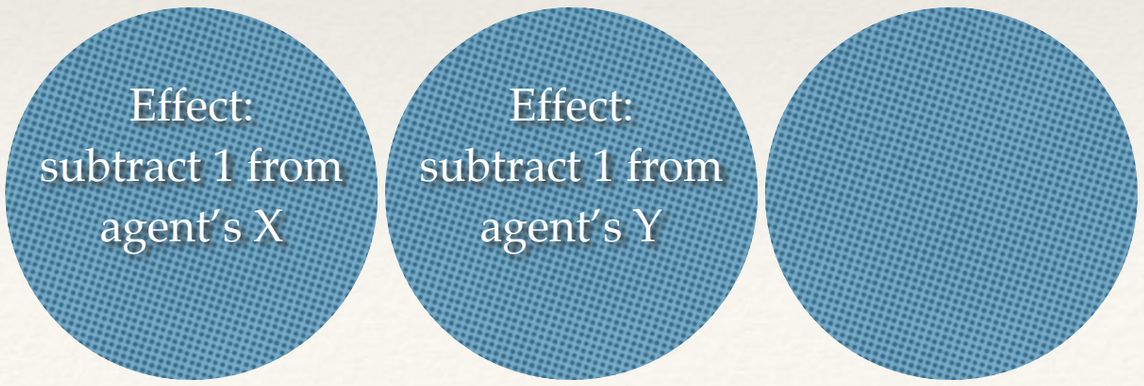
```
Prediction for west's effect on xAtt of passenger
        condition:(011011)
        effectLearner: ArithmeticEffect of adding -1.0 to xAtt of passengers
```



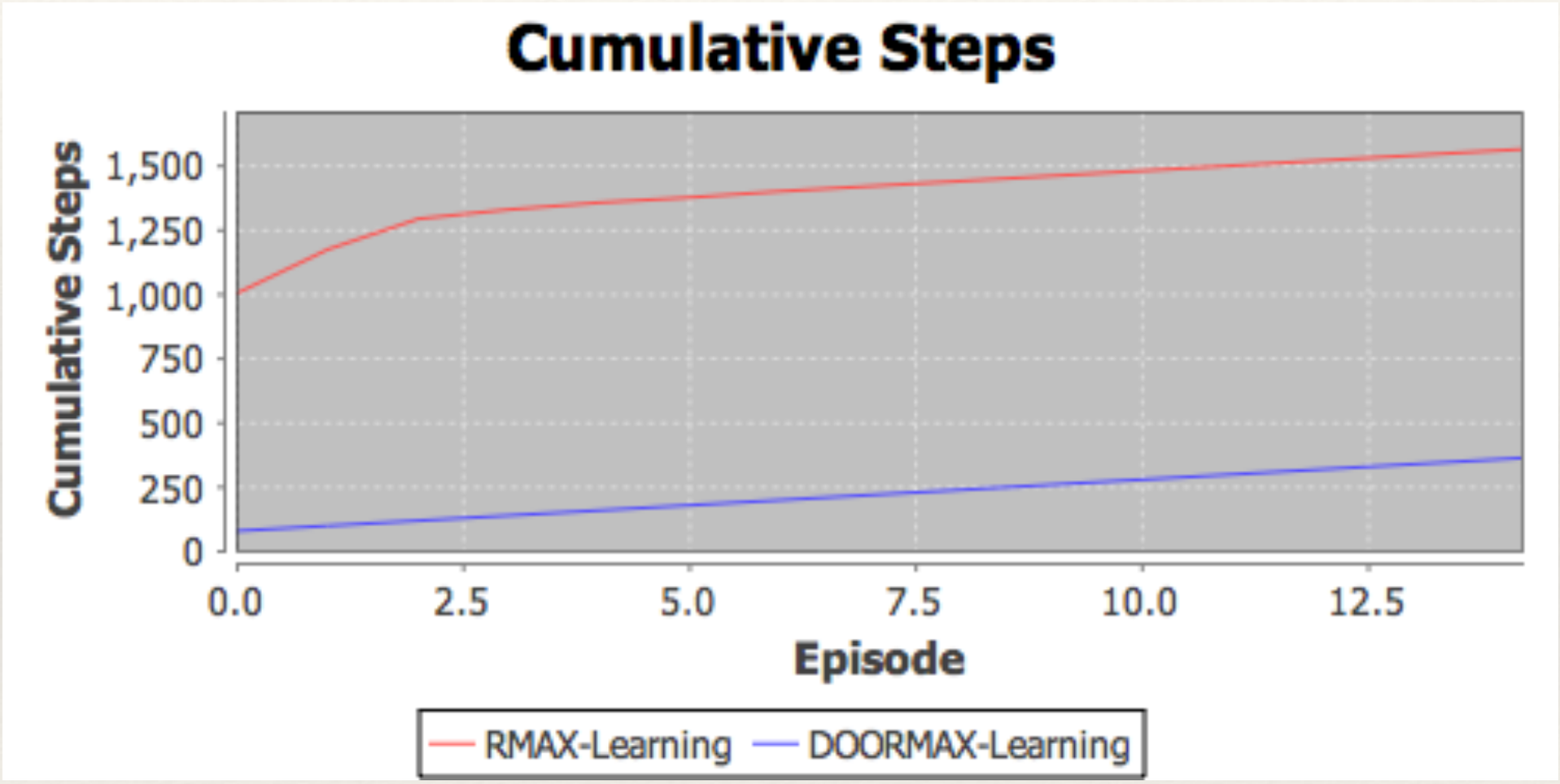
DOORMAX Solutions (2/2)

- End result is a prediction pool for each attribute — we return "don't know" if there is a pool that is both empty and for which the action is not a known failure condition on the attribute.

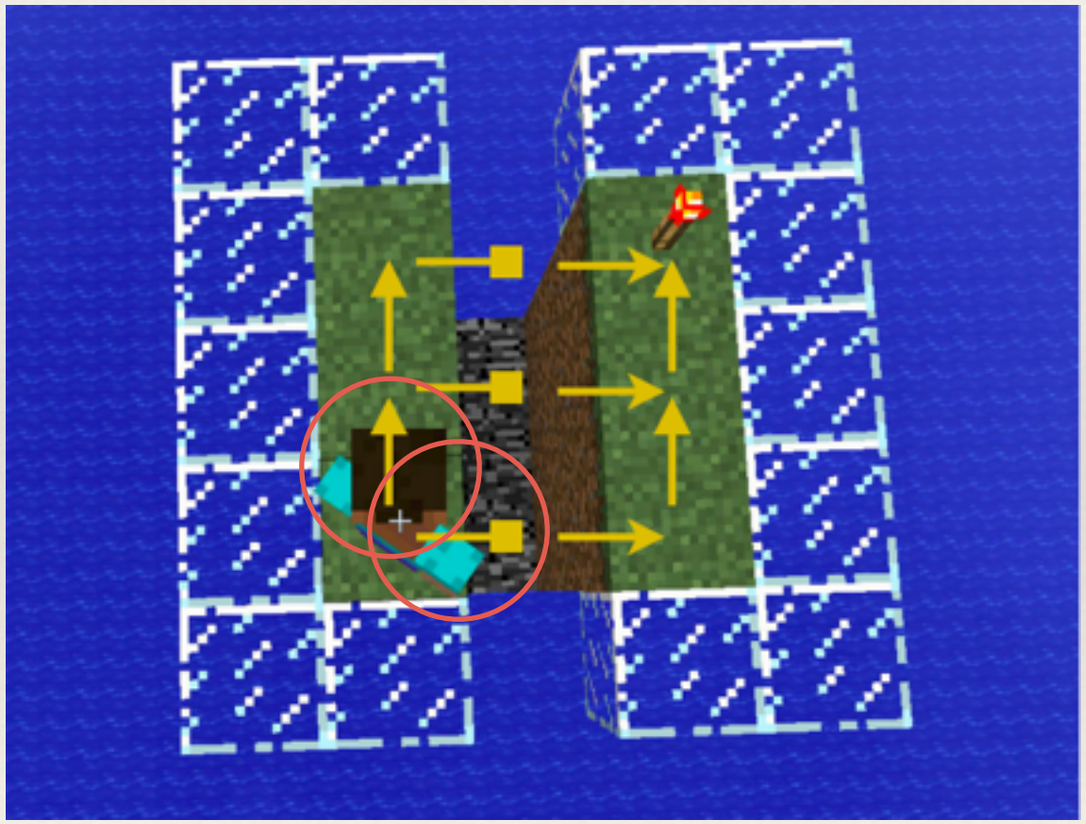
```
Failure condition for east on taxi's xAtt: (010001)(010011)(110011)(110001)(011000)(110000)(010000)(010100)(011100)
Failure condition for east on taxi's yAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(100011)(010001)(
Failure condition for east on location's xAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(100011)(0100
Failure condition for east on location's yAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(100011)(0100
Failure condition for east on location's locationAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(10001
Failure condition for east on passenger's xAtt: (001101)(000001)(000101)(100001)(010001)(010011)(110011)(100000)(110001)(011
Failure condition for east on passenger's yAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(100011)(010
Failure condition for east on passenger's locationAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(1000
Failure condition for east on passenger's inTaxiAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(100011
Failure condition for east on horizontalWall's wallMinAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(
Failure condition for east on horizontalWall's wallMaxAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(
Failure condition for east on horizontalWall's wallOffsetAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(10000
Failure condition for east on verticalWall's wallMinAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(10
Failure condition for east on verticalWall's wallMaxAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)(10
Failure condition for east on verticalWall's wallOffsetAtt: (001101)(001111)(000111)(000001)(100111)(000101)(000011)(100001)
```



DOORMAX Results



What about Minecraft?



Y

X

Predictions for Move

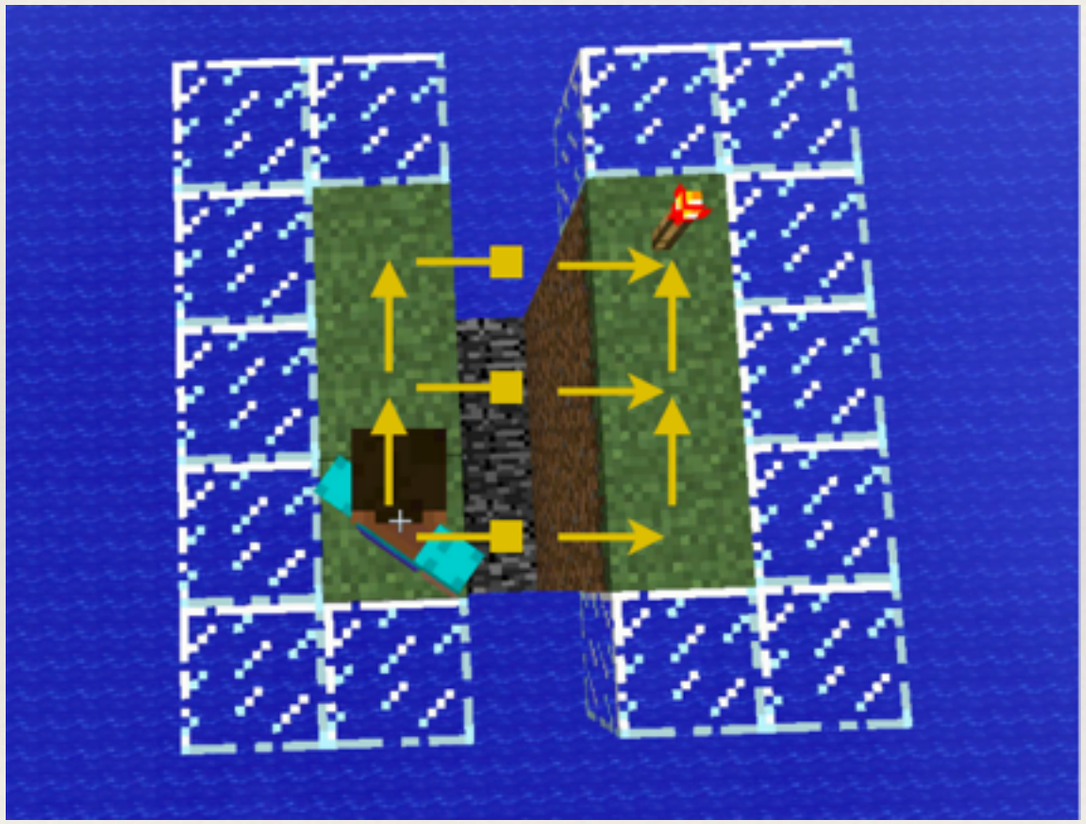
Add 1 to Agent Y on condition1

???

Add 1 to Agent X on condition1

We Need PFs

- If only we had the right propositional function:



Y

X

Predictions for Move

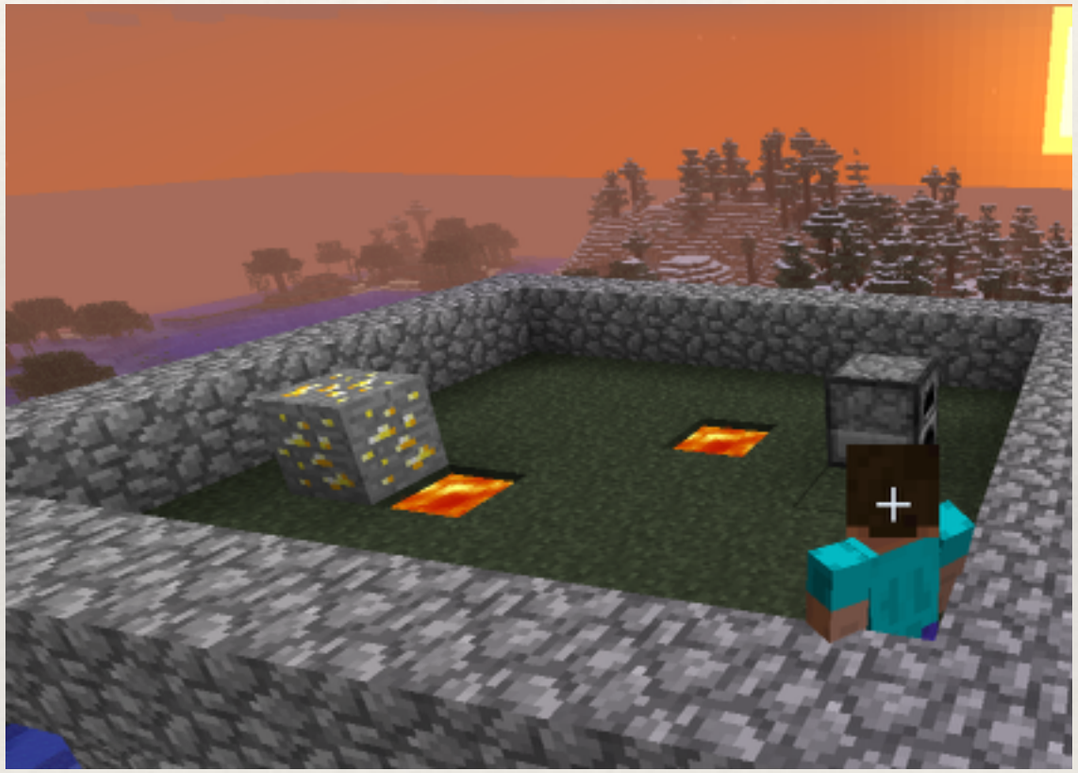
Add 1 to AgentY if facingNorth

Add 1 to AgentX if facingSouth

Specifying PFs

- Too many propositional functions.

AgentLookingAtWall
AgentInMidAir
AgentCanJump
HurdleInfrontOfAgent
TrenchInFrontOfAgent
AgentInLava
AgentCanWalk



```
condition:(*0100000000000111001110000*0000)
```

We Need PFs

- Complex transition dynamics require many hard to specify PFs.

- Wouldn't it be great if we could learn these PFs?…

Future Work on Learning PFs

- Rather than strictly learn exact conditions, why not use supervised learning?

East affects agent X: S1, S2

```
taxi (taxi)
        xAtt:   3
        yAtt:   2

location0 (location)
        xAtt:   0
        yAtt:   0
        locationAtt:    yellow

location1 (location)
        xAtt:   0
        yAtt:   4
        locationAtt:    red

location2 (location)
        xAtt:   3
        yAtt:   0
        locationAtt:    blue
```

```
taxi (taxi)
        xAtt:   4
        yAtt:   2

location0 (location)
        xAtt:   0
        yAtt:   0
        locationAtt:    yellow

location1 (location)
        xAtt:   0
        yAtt:   4
        locationAtt:    red

location2 (location)
        xAtt:   3
        yAtt:   0
        locationAtt:    blue
```

North affects agent Y: S1, S37

```
taxi (taxi)
        xAtt:   3
        yAtt:   2

location0 (location)
        xAtt:   0
        yAtt:   0
        locationAtt:    yellow

location1 (location)
        xAtt:   0
        yAtt:   4
        locationAtt:    red

location2 (location)
        xAtt:   3
        yAtt:   0
        locationAtt:    blue
```

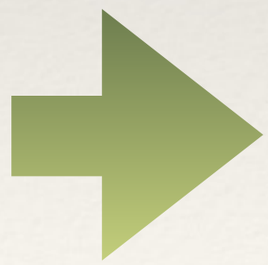
```
taxi (taxi)
        xAtt:   0
        yAtt:   3

location0 (location)
        xAtt:   0
        yAtt:   0
        locationAtt:    yellow

location1 (location)
        xAtt:   0
        yAtt:   4
        locationAtt:    red

location2 (location)
        xAtt:   3
        yAtt:   0
        locationAtt:    blue
```

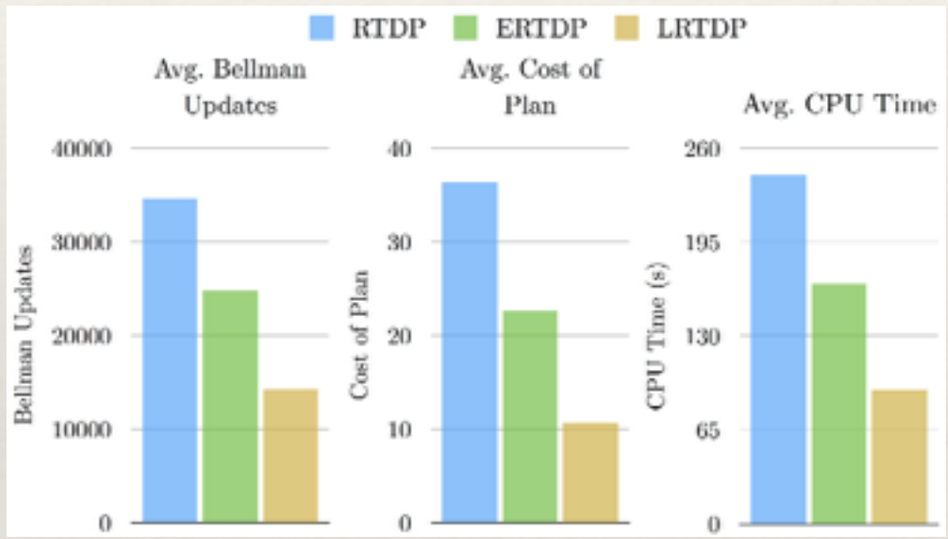
- else pool the effects of all predictions whose conditions match the current state into a pool of effects.



- else pool the effects of all predictions whose conditions are similar to the current state (via classification).

Learning PFs for Affordances

- Earlier work on limiting large state-spaces using propositional functions relevant to .



- Correspondence between PFs that indicate action optimality and transition dynamics is good news!

Scaffolding

Questions and Comments