Q-LEARNING AND REAL-TIME STRATEGY

CSCI 8360 Data Science Practicum

Spring 2021

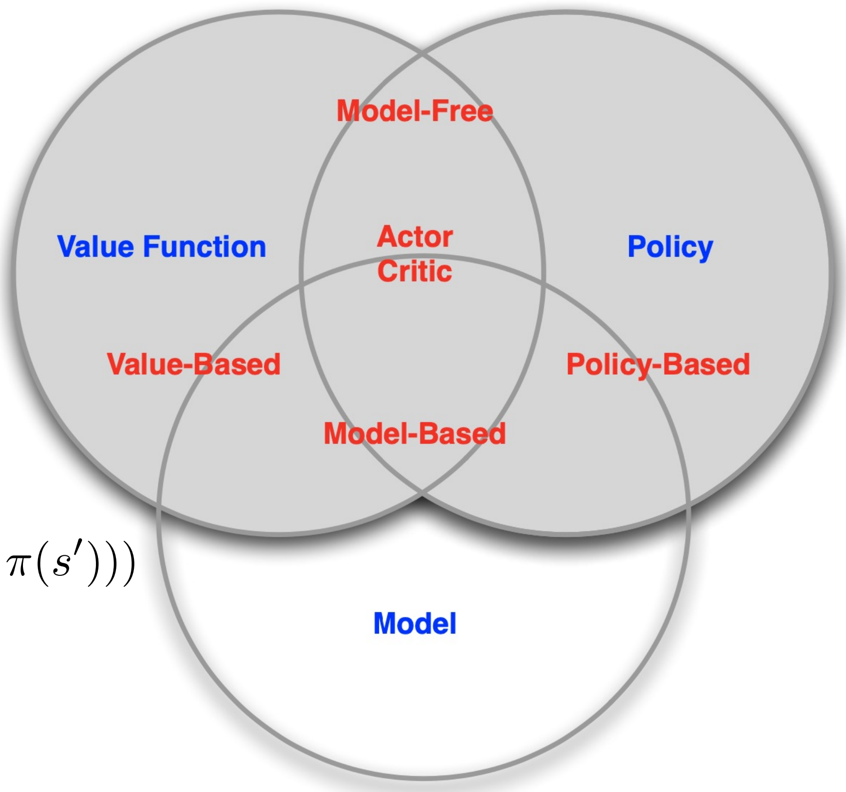
VALUE-BASED METHODS FOR RL

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a)$$

Find the optimal policy π in state s over all possible actions A

What is Q?

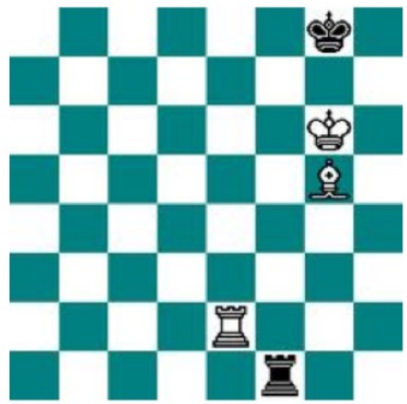
$$Q^\pi(s,a) = \sum_{s' \in S} T(s,a,s')(R(s,a,s') + \gamma Q^\pi(s', a = \pi(s')))$$



PREVIOUSLY ON: THE PREVIOUS LECTURE

Binary-linear value function v(s, w)

- Binary feature vector x(s): one feature per chess piece
- Weight vector w: value of each chess piece
- Position is evaluated by summing weights of current features



$$v(s, \mathbf{w}) = \mathbf{x}(s) \cdot \mathbf{w} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \cdot \begin{array}{cc} \begin{bmatrix} +5 \\ +3 \\ +1 \\ -5 \\ -3 \\ -1 \\ \vdots \end{bmatrix} & \begin{array}{c} \text{♖} \\ \text{♗} \\ \text{♙} \\ \text{♜} \\ \text{♝} \\ \text{♟} \\ \end{array} \end{array}$$

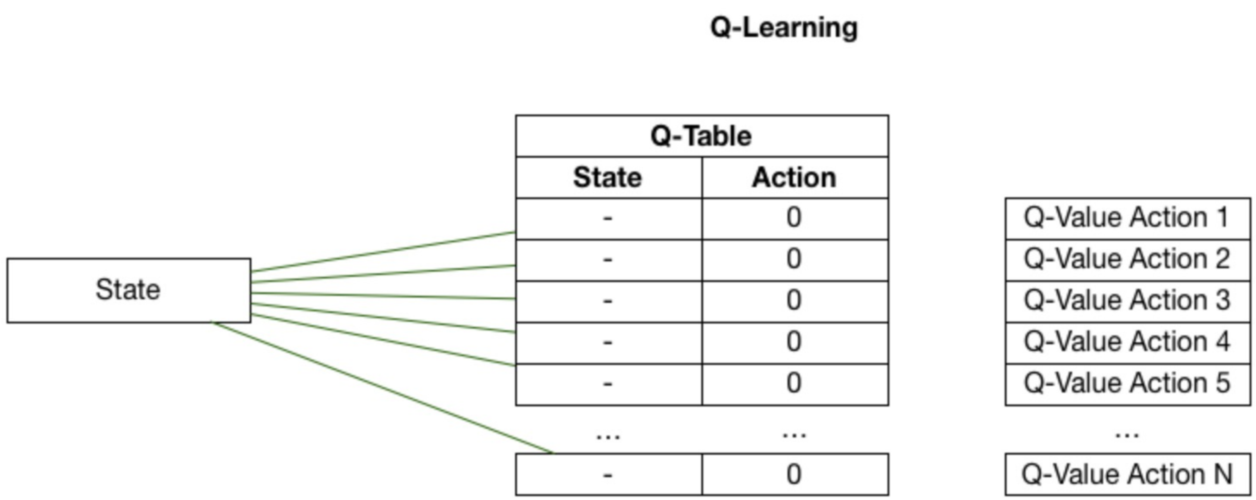
Recall: in a fully observable system, *state* simply becomes *observation*.

$$v(s, \mathbf{w}) = 5 + 3 - 5 = 3$$

Q-LEARNING

For the spreadsheet-o-philes

- Goal of Q-Learning: build a table mapping all possible states to all subsequent estimates of reward for being in that state



Q-LEARNING

$$Q^{\pi}(s,a) = \sum_{s' \in S} T(s,a,s')(R(s,a,s') + \gamma Q^{\pi}(s', a = \pi(s')))$$

The optimal Q* is the *expected discount return* when in state *s* and taking action *a* while following the optimal policy π*

$$w^{l+1} \leftarrow w^{l} + \eta \nabla w^{l}$$

Learning process

Learning rate

Discount factor

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a} Q(s_{t+1}, a_t) - Q(s_t, a_t) \right)$$

Next training iteration

Current estimate

Reward

Estimate of optimal future value

Current estimate

Q-LEARNING

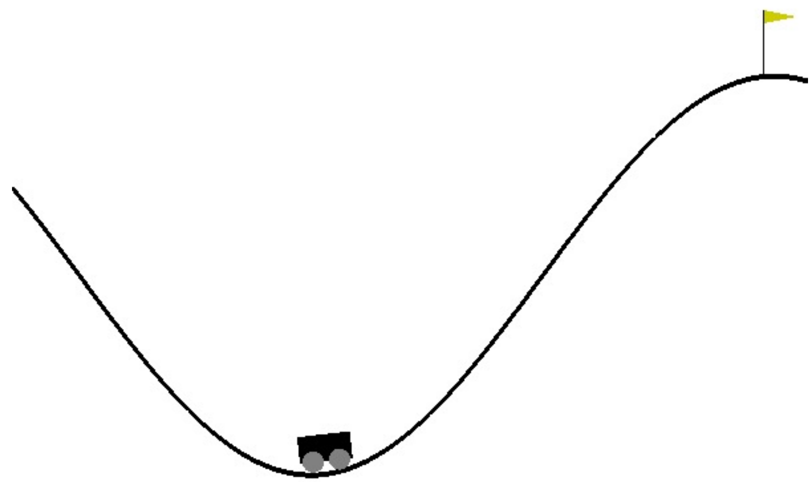
Example: Mountain Car

Actions

- 0: apply left force
- 1: do nothing
- 2: apply right force

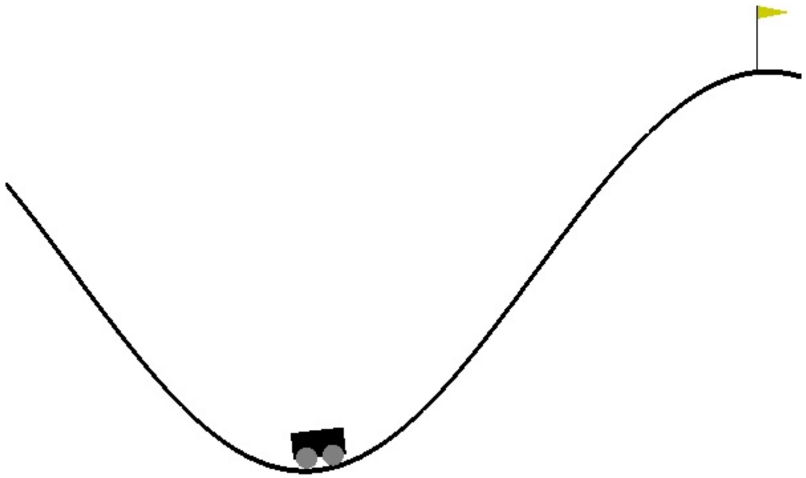
Environment

- State[0]: position
- State[1]: velocity



Car does not have enough force to climb the hill entirely on its own

Q-LEARNING



Example: Mountain Car

You could certainly hard-code this!

- If velocity = 0, apply force in a random direction
- If velocity > 0, apply force in the direction of movement

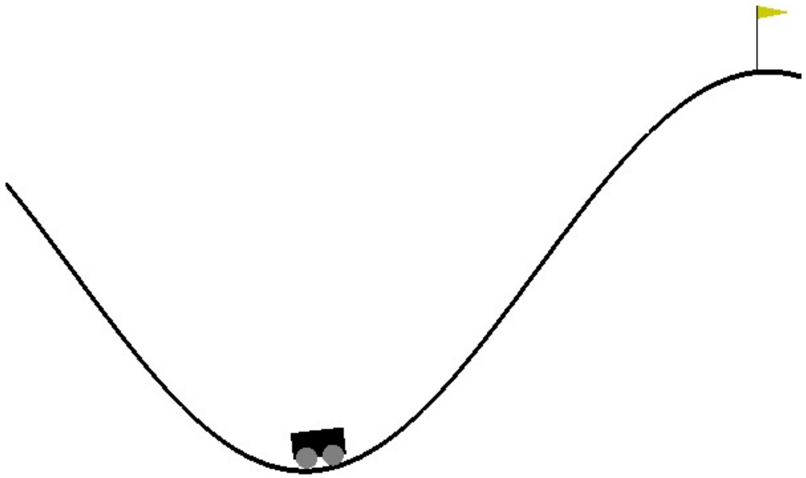
```
done = False

i = 0
while not done:
    i += 1

    if state[1]>0:
        action = 2
    else:
        action = 0

    state, reward, done, _ = env.step(action)
    env.render()
    print(f"Step {i}: State={state}, Reward={reward}")
```

Q-LEARNING

Example: Mountain Car

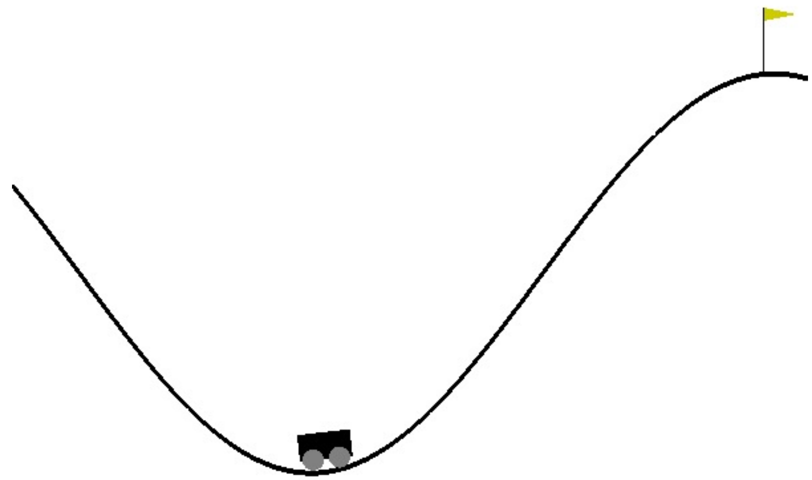
You could simply hard-code this!

- If velocity = 0, apply force in a random direction
- If velocity > 0, apply force in the direction of movement



```
done = False

i = 0
while not done:
    i += 1

    if state[1]>0:
        action = 2
    else:
        action = 0

    state, reward, done, _ = env.step(action)
    env.render()
    print(f"Step {i}: State={state}, Reward={reward}")
```

Q-LEARNING



Example: Mountain Car

…but we'd like something a little more generalizable

Start by discretizing state space

- Binning position/velocity

Randomly initialize Q table

Iterate!

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right)$$

Q-LEARNING

Example: Mountain Car

We'd get a Q table that looks something like this

- Note the discretization of position and velocity into 10 bins
- p0 is far left, p9 far right
- v0 is not moving, v9 is max velocity (magnitude)

After training, values in the table indicate the action that should be taken in a given state

- Yielded the greatest reward in training
- 0: move left, 1: do nothing, 2: move right

	v-0	v-1	v-2	v-3	v-4	v-5	v-6	v-7	v-8	v-9
p-0	2	0	1	1	2	2	2	1	1	1
p-1	0	1	1	2	0	2	2	2	2	2
p-2	0	0	1	2	2	2	2	1	2	1
p-3	0	0	0	0	0	2	2	2	2	2
p-4	0	0	0	0	0	0	2	0	2	2
p-5	1	0	1	0	0	0	2	1	2	2
p-6	2	2	0	0	0	0	2	0	1	0
p-7	1	0	0	0	0	0	0	2	2	0
p-8	1	0	2	0	2	2	2	2	1	0
p-9	2	2	0	2	0	0	1	2	0	2

DEEP Q-LEARNING

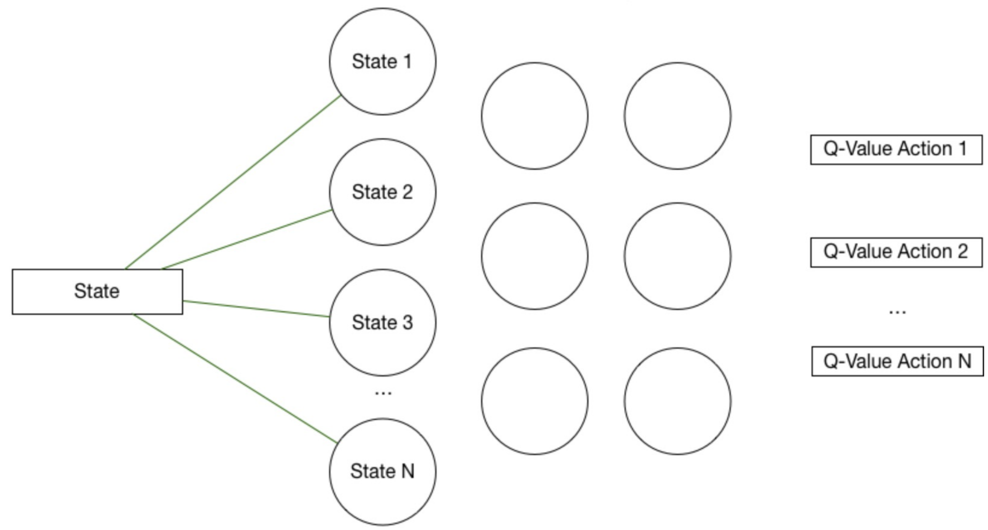
Uses a deep neural network

- Aka, universal function approximator

Also addresses the problem of continuous state values

Input: state

Output: action



PAUSE FOR QUESTIONS

REAL-TIME STRATEGY

RL in video games

A FEW QUESTIONS

Have you heard of “real-time strategy” in the context of video games?

Have you heard of StarCraft (or StarCraft II)?

A FEW ANSWERS

Real-time Strategy (RTS)

Real-time

- As opposed to turn-based
- Time moves forward continuously, without human input (i.e., if you take no action, your in-game avatar will take no action; there's often no option for "pausing")
- First coined to describe *Dune II* in early 1990s
- Really came of age in the late 1990s with *Red Alert*, *WarCraft*, and *StarCraft*

Strategy

- Management of limited resources (including time!)
- Exploitation vs exploration
- Can involve not just military strategy (army composition, unit production, attack vs defense strategies) but also diplomacy, propaganda, economics, culture, or religion
- Video games like *Civilization* or board games like *Risk* and *Settlers of Catan*

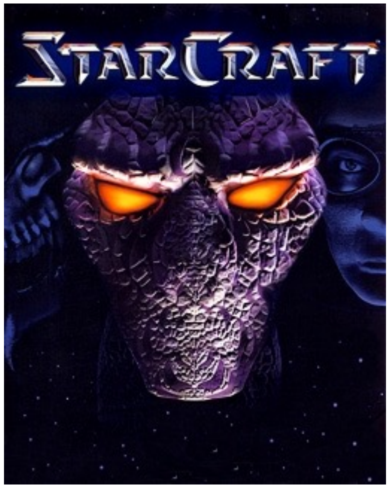
A FEW ANSWERS

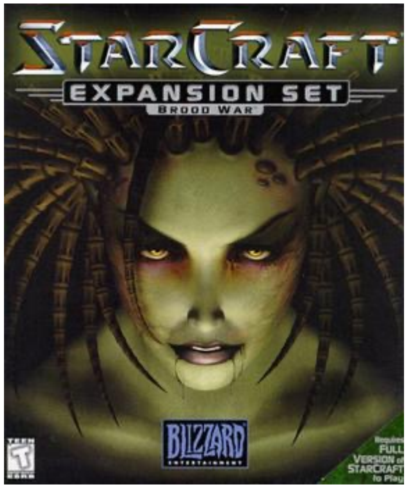
StarCraft II

- Released in three phases: 2010, 2013, and 2015
- Sequel to 1998 *StarCraft* original and *Brood War* expansion

Interstellar war between three factions

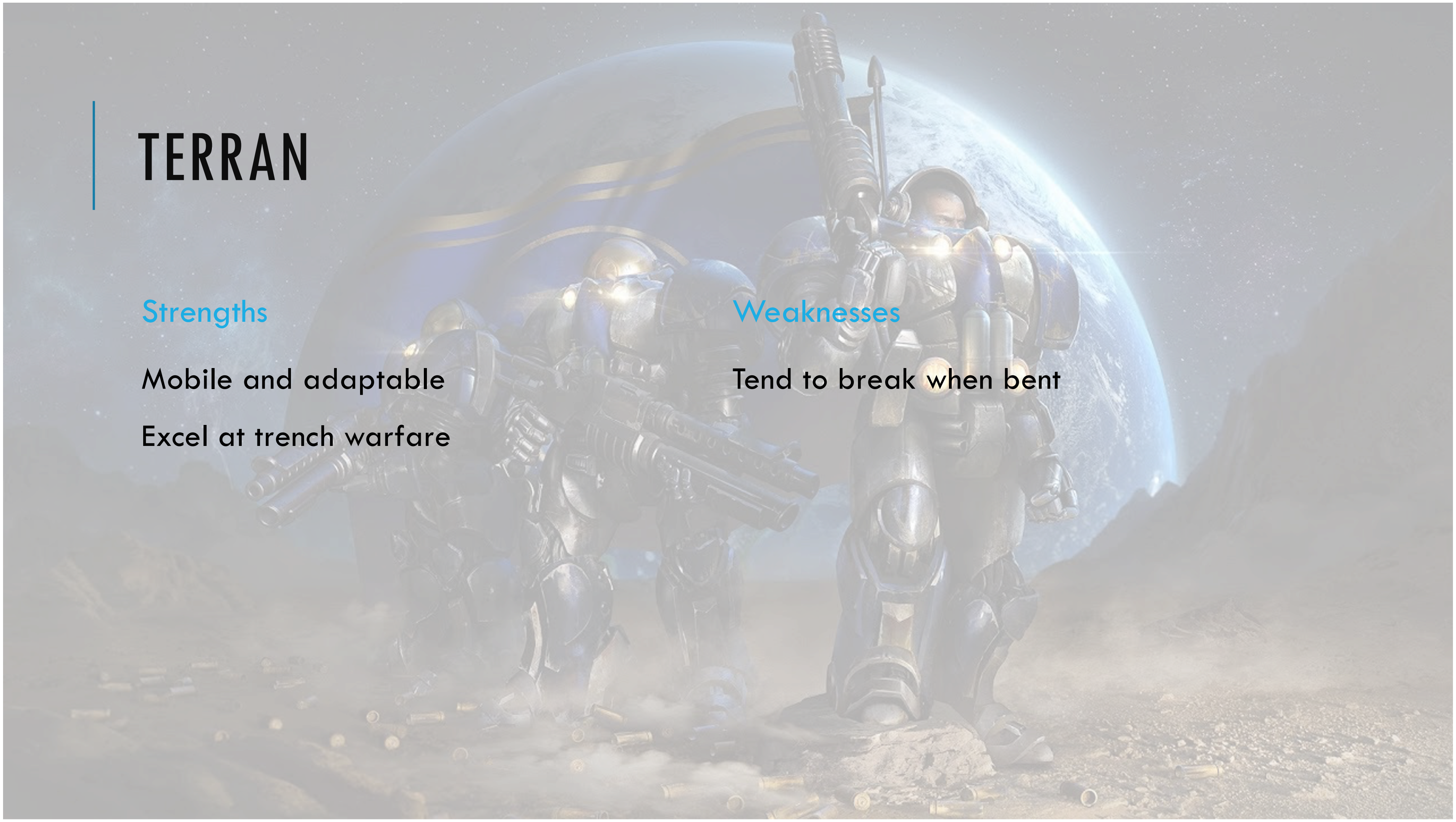
- Terrans (humans)
- Protoss (aliens)
- Zerg (aliens)





LEGACY OF STARCRAFT

StarCraft featured three **wholly and distinctly unique factions** with their own strengths and weaknesses



TERRAN

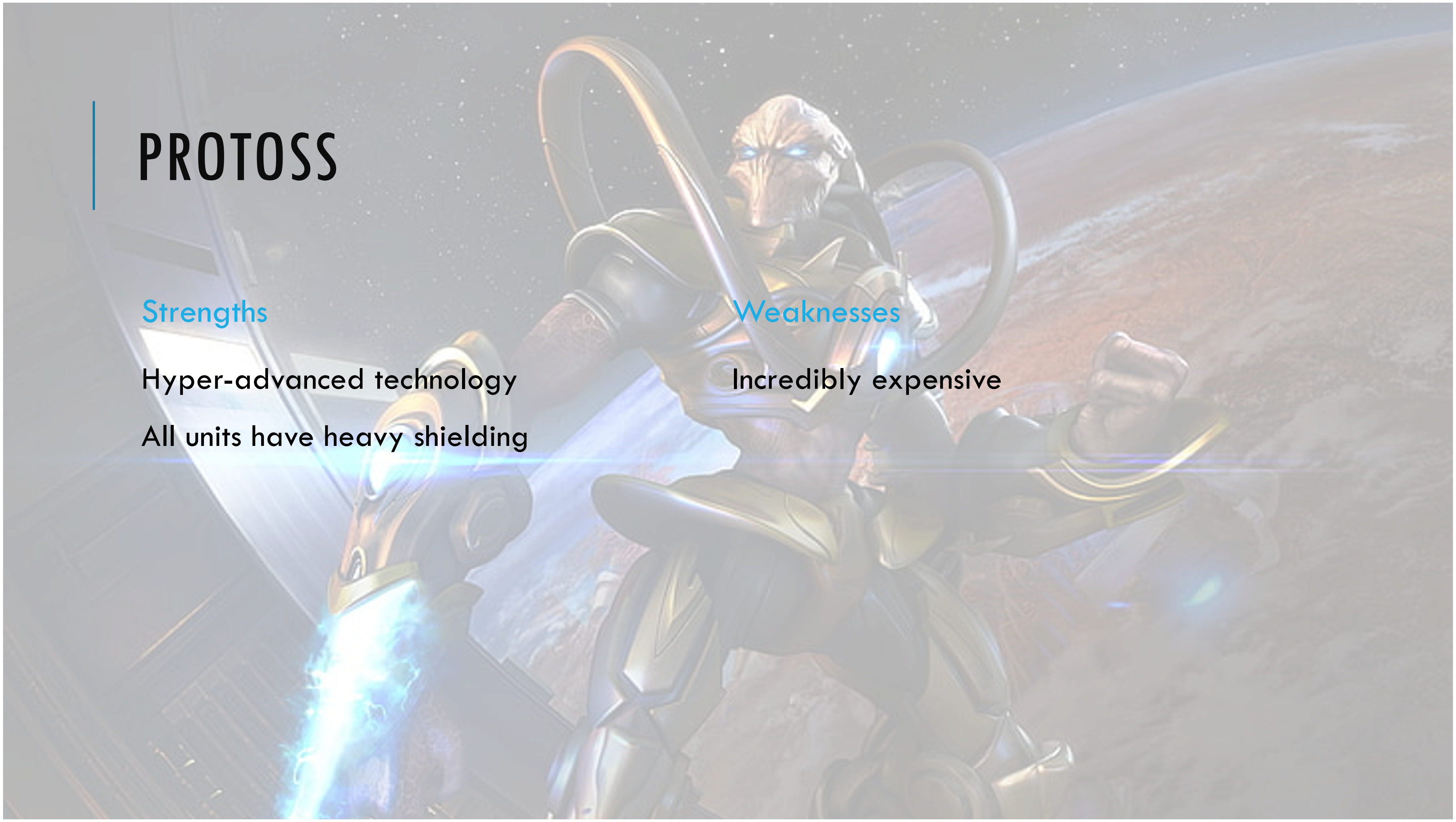
Strengths

Mobile and adaptable

Excel at trench warfare

Weaknesses

Tend to break when bent



PROTOSS

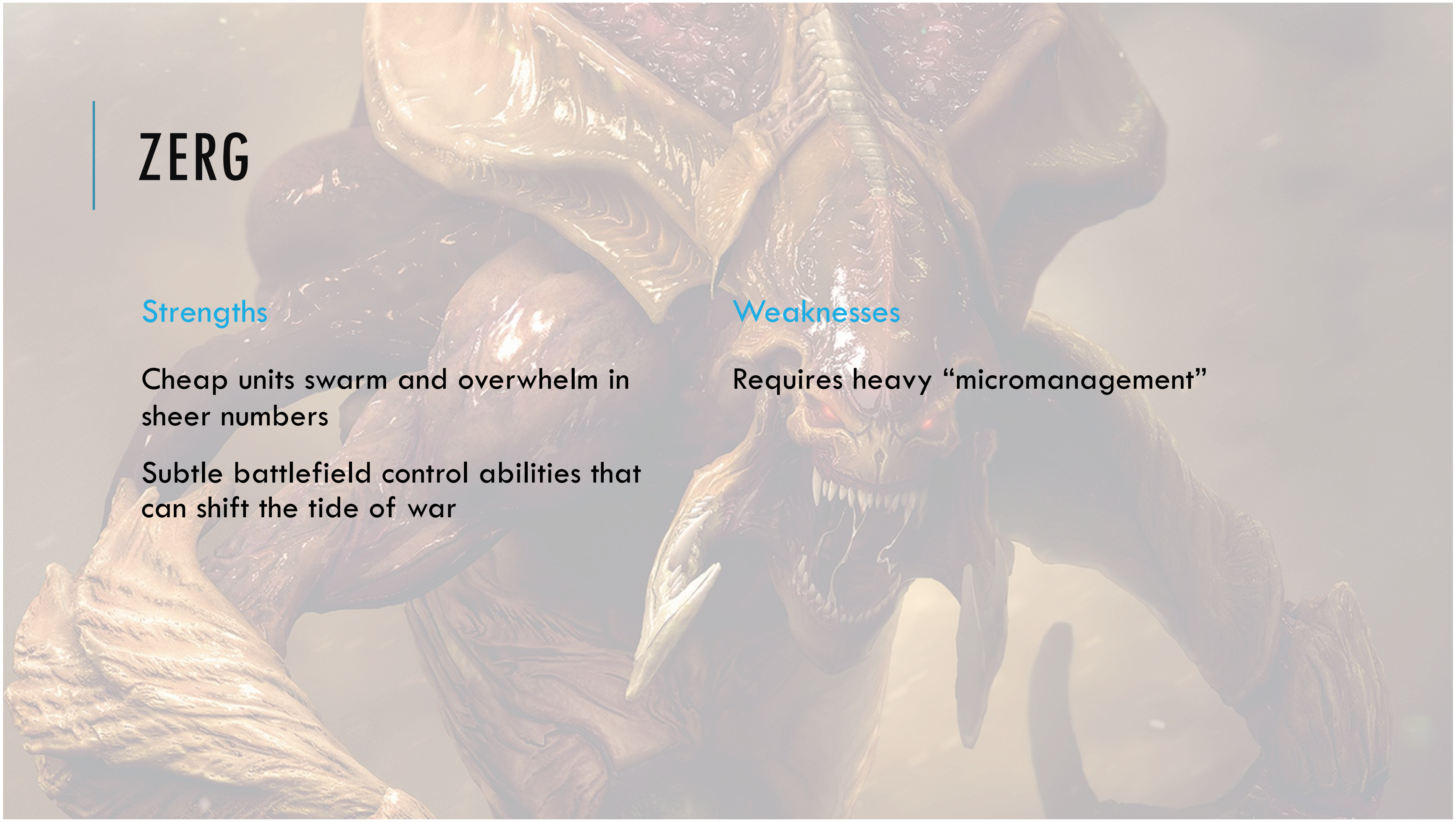
Strengths

Hyper-advanced technology

All units have heavy shielding

Weaknesses

Incredibly expensive



ZERG

Strengths

Cheap units swarm and overwhelm in sheer numbers

Subtle battlefield control abilities that can shift the tide of war

Weaknesses

Requires heavy "micromanagement"

RESOURCES

Two main resources / currencies

Minerals

- Appear in “fields”
- Can have lots of workers mine them simultaneously
- Used for building the basic units of all three factions

Vespene gas

- Must be extracted from geysers
- Can only have 1 worker inside a geyser at a time
- Needed for the upper-tier / advanced units of all three factions (especially Protoss)

CORE STRATEGY (“MACRO”)

Gather resources

- Basic workers gather minerals and vespene

Construct buildings

- Unlocks construction of other buildings and new units
- Some buildings have defensive capabilities (turrets, pillboxes)
- May include expanding to new resource locations to increase rate of income and/or seize certain strategic areas of the map

Build units

- Both for defense and attack
- Want a well-balanced force (air and ground)
- Also want it as fast as possible and as powerful as possible (advanced units, lots of upgrades)

Attack / defend until only one player remains

- Attacks can be well-planned massive sieges or fast hit-and-run raids
- Wait to build up sizeable army, or attack fast (called “rushing”) and knock out opponent before they mount a defense

TRADE-OFFS

Unit production

- Build the less powerful unit now, or save up for the more expensive one later?
- Build more units, or upgrade current ones?
- Research new abilities / units or build more of current ones?

Exploration vs Exploitation

- Expand to a new site (more minerals + vespene) or focus on defending current base?
- Attack enemy or grow army?
- Post units at strategic chokepoints on map or focus on base defense?

DIFFICULTY CURVE

Steep

- For new players: relatively straightforward to get started
- For experts: very, very long and steep climb to the top

Action space is *effectively* infinite

- Any number of actions you could take at any moment (Build? Mine? Upgrade? Research? Scout? Attack? Defend?)

Relatively long game duration dilutes effects of reward on any specific action

- 1v1 games can be as quick as 2-3 minutes, but can go much longer between well-matched players
- FFA (free for all) with 3+ players can last hours

Constant evaluation and re-evaluation of trade-offs

- Game conditions are partially-observable (“fog of war”) so best course of action is not always clear
- Often hedging one’s bets by pursuing multiple strategies, though this also dilutes effect of any one

Factions are unique, but each has a counter for any strategy the others use

- Requires scouting, resource management, and prioritization to effectively counter

MINI GAMES

Google DeepMind (creators of AlphaStar StarCraft II RL bot) created SC2 “mini game” environments for narrow subtasks of SC2

Examples include:

- Build Marines (basic Terran unit)
- Collect minerals and gas
- Defeat Roaches (pernicious Zerg unit)
- Defeat Zerglings and Banelings (core of Zerg overwhelm tactics)
- Move to beacon
- Find and defeat Zerglings

PROJECT 3

Out on **Thursday, April 1** (I promise that's not a harbinger of anything)

- More details to come

In other news:

- P2 peer reviews are due **Tuesday, March 30**
- P2 Lightning Talks are on **Wednesday, March 31** (same format as P1)
- If anyone needs anything, please let me know

QUESTIONS?

REFERENCES

Introduction to Q-Learning for Game Play
https://www.youtube.com/watch?v=A3sYFcJY3lA

Keras Q-Learning in the OpenAI Gym
https://www.youtube.com/watch?v=qy1SJmsRhvM

Atari Games with Keras TF-Agents
https://www.youtube.com/watch?v=co0SwPWoZh0

PyTorch Reinforcement Learning DQN Tutorial
https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html#dqn-algorithm