Open in Colab

T81-558: Applications of Deep Neural Networks

Module 12: Reinforcement Learning

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 12 Video Material

- Part 12.1: Introduction to the OpenAI Gym [Video] [Notebook]
- **Part 12.2: Introduction to Q-Learning** [Video] [Notebook]
- Part 12.3: Keras Q-Learning in the OpenAI Gym [Video] [Notebook]
- Part 12.4: Atari Games with Keras Neural Networks [Video] [Notebook]
- Part 12.5: Application of Reinforcement Learning [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [1]:
```
try:
    from google.colab import drive
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: using Google CoLab
```

In [2]:
```
# HIDE OUTPUT
if COLAB:
  !sudo apt-get install -y xvfb ffmpeg x11-utils
  !pip install -q gym
  !pip install -q 'imageio==2.4.0'
  !pip install -q PILLOW
  !pip install -q 'pyglet==1.3.2'
  !pip install -q pyvirtualdisplay
```

```
!pip install -q tf-agents
!pip install -q pygame
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
ffmpeg is already the newest version (7:3.4.8-0ubuntu0.2).
Suggested packages:
  mesa-utils
The following NEW packages will be installed:
  libxxf86dga1 x11-utils xvfb
0 upgraded, 3 newly installed, 0 to remove and 39 not upgraded.
Need to get 993 kB of archives.
After this operation, 2,982 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/main amd64 libxxf86dga1 amd64
2:1.1.4-1 [13.7 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/main amd64 x11-utils amd64 7.7
+3build1 [196 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 xvfb am
d64 2:1.19.6-1ubuntu4.10 [784 kB]
Fetched 993 kB in 1s (1,252 kB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based fr
ontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 7
6, <> line 3.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package libxxf86dga1:amd64.
(Reading database ... 156210 files and directories currently installed.)
Preparing to unpack .../libxxf86dga1_2%3a1.1.4-1_amd64.deb ...
Unpacking libxxf86dga1:amd64 (2:1.1.4-1) ...
Selecting previously unselected package x11-utils.
Preparing to unpack .../x11-utils_7.7+3build1_amd64.deb ...
Unpacking x11-utils (7.7+3build1) ...
Selecting previously unselected package xvfb.
Preparing to unpack .../xvfb_2%3a1.19.6-1ubuntu4.10_amd64.deb ...
Unpacking xvfb (2:1.19.6-1ubuntu4.10) ...
Setting up xvfb (2:1.19.6-1ubuntu4.10) ...
Setting up libxxf86dga1:amd64 (2:1.1.4-1) ...
Setting up x11-utils (7.7+3build1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.3) ...
/sbin/ldconfig.real: /usr/local/lib/python3.7/dist-packages/ideep4py/lib/lib
mkldnn.so.0 is not a symbolic link

     |████████████████████████████████| 3.3 MB 5.1 MB/s
  Building wheel for imageio (setup.py) ... done
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.
2.9 which is incompatible.
     |████████████████████████████████| 1.0 MB 5.2 MB/s
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
```

```
gym 0.17.3 requires pyglet<=1.5.0,>=1.4.0, but you have pyglet 1.3.2 which i
s incompatible.
     |████████████████████████████████| 1.3 MB 5.0 MB/s
     |████████████████████████████████| 1.0 MB 29.8 MB/s
     |████████████████████████████████| 21.8 MB 1.2 MB/s
```

Part 12.2: Introduction to Q-Learning

Q-Learning is a foundational technology upon which deep reinforcement learning is based. Before we explore deep reinforcement learning, it is essential to understand Q-Learning. Several components make up any Q-Learning system.

- **Agent** - The agent is an entity that exists in an environment that takes actions to affect the state of the environment, to receive rewards.
- **Environment** - The environment is the universe that the agent exists in. The environment is always in a specific state that is changed by the agent's actions.
- **Actions** - Steps that the agent can perform to alter the environment
- **Step** - A step occurs when the agent performs an action and potentially changes the environment state.
- **Episode** - A chain of steps that ultimately culminates in the environment entering a terminal state.
- **Epoch** - A training iteration of the agent that contains some number of episodes.
- **Terminal State** - A state in which further actions do not make sense. A terminal state occurs when the agent has one, lost, or the environment exceeds the maximum number of steps in many environments.

Q-Learning works by building a table that suggests an action for every possible state. This approach runs into several problems. First, the environment is usually composed of several continuous numbers, resulting in an infinite number of states. Q-Learning handles continuous states by binning these numeric values into ranges.

Out of the box, Q-Learning does not deal with continuous inputs, such as a car's accelerator that can range from released to fully engaged. Additionally, Q-Learning primarily deals with discrete actions, such as pressing a joystick up or down. Researchers have developed clever tricks to allow Q-Learning to accommodate continuous actions.

Deep neural networks can help solve the problems of continuous environments and action spaces. In the next section, we will learn more about deep reinforcement learning. For now, we will apply regular Q-Learning to the Mountain Car problem from OpenAI Gym.

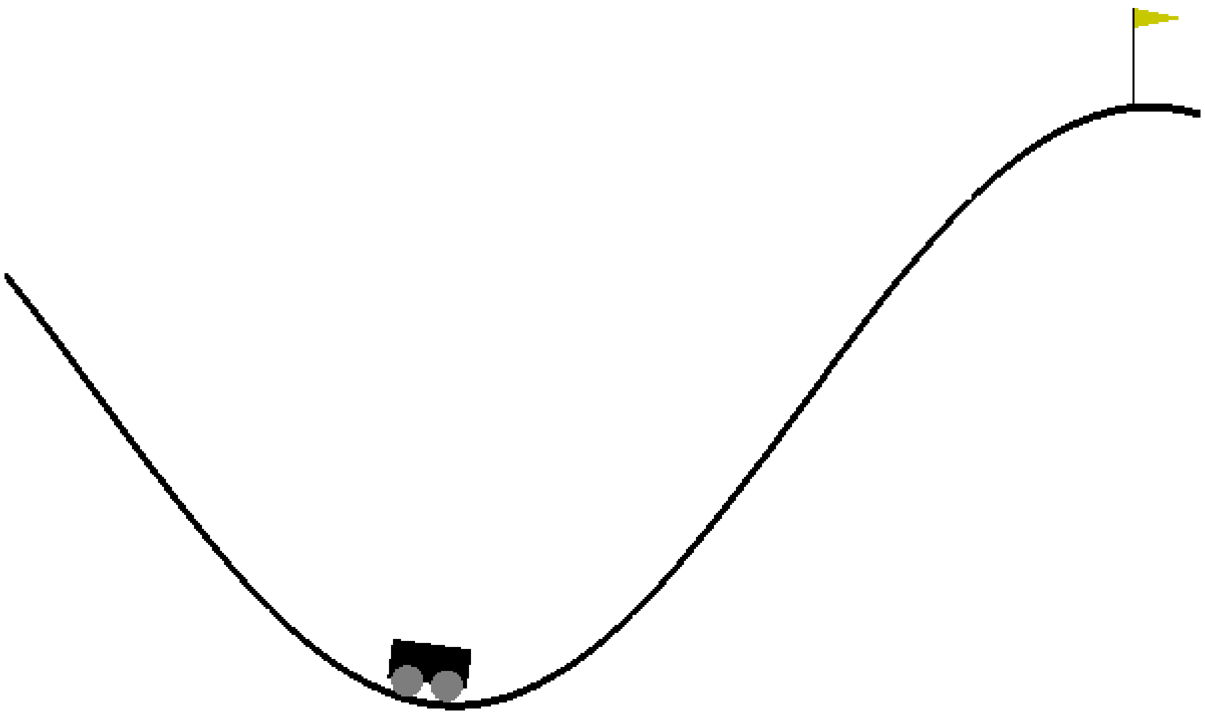
Introducing the Mountain Car

This section will demonstrate how Q-Learning can create a solution to the mountain car gym environment. The Mountain car is an environment where a car must climb a mountain. Because gravity is stronger than the car's engine, it cannot merely accelerate up the steep slope even with full throttle. The vehicle is situated in a valley and must learn to utilize potential energy by driving up the opposite hill before the car can make it to the goal at the top of the rightmost hill.

First, it might be helpful to visualize the mountain car environment. The following code shows this environment. This code makes use of TF-Agents to perform this render. Usually, we use TF-Agents for the type of deep reinforcement learning that we will see in the next module. However, TF-Agents is just used to render the mountain care environment for now.

In [3]:
```
import tf_agents
from tf_agents.environments import suite_gym
import PIL.Image
import pyvirtualdisplay

display = pyvirtualdisplay.Display(visible=0, size=(1400, 900)).start()

env_name = 'MountainCar-v0'
env = suite_gym.load(env_name)
env.reset()
PIL.Image.fromarray(env.render())
```

Out[3]:



The mountain car environment provides the following discrete actions:

- 0 - Apply left force

- 1 - Apply no force
- 2 - Apply right force

The mountain car environment is made up of the following continuous values:

- state[0] - Position
- state[1] - Velocity

The cart is not strong enough. It will need to use potential energy from the mountain behind it. The following code shows an agent that applies full throttle to climb the hill.

In [4]:
```
import gym
from gym.wrappers import Monitor
import glob
import io
import base64
from IPython.display import HTML
from pyvirtualdisplay import Display
from IPython import display as ipythondisplay

display = Display(visible=0, size=(1400, 900))
display.start()


def show_video():
    mp4list = glob.glob('video/*.mp4')
    if len(mp4list) > 0:
        mp4 = mp4list[0]
        video = io.open(mp4, 'r+b').read()
        encoded = base64.b64encode(video)
        ipythondisplay.display(HTML(data='''<video alt="test" autoplay
                 loop controls style="height: 400px;">
                 <source src="data:video/mp4;base64,{0}"
                 type="video/mp4" />
              </video>'''.format(encoded.decode('ascii'))))
    else:
        print("Could not find video")


def wrap_env(env):
    env = Monitor(env, './video', force=True)
    return env
```

We are now ready to train the agent.

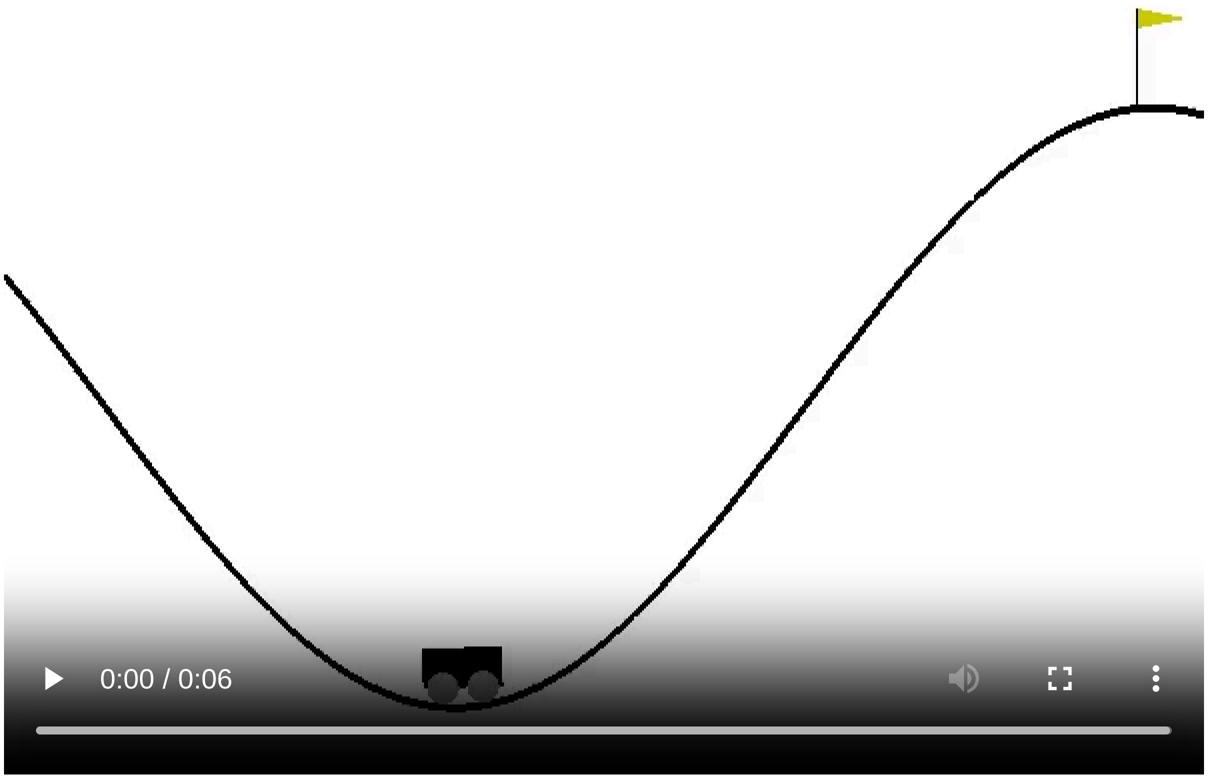
In [5]:
```
import gym

if COLAB:
    env = wrap_env(gym.make("MountainCar-v0"))
else:
    env = gym.make("MountainCar-v0")

env.reset()
```

```
done = False

i = 0
while not done:
    i += 1
    state, reward, done, _ = env.step(2)
    env.render()
    print(f"Step {i}: State={state}, Reward={reward}")

env.close()
```

```
Step 1: State=[-0.50905189  0.00089766], Reward=-1.0
Step 2: State=[-0.50726329  0.00178859], Reward=-1.0
Step 3: State=[-0.50459717  0.00266613], Reward=-1.0
Step 4: State=[-0.50107348  0.00352369], Reward=-1.0
Step 5: State=[-0.4967186   0.00435488], Reward=-1.0
Step 6: State=[-0.4915651  0.0051535], Reward=-1.0
Step 7: State=[-0.48565149  0.00591361], Reward=-1.0
Step 8: State=[-0.47902187  0.00662962], Reward=-1.0
Step 9: State=[-0.47172557  0.00729629], Reward=-1.0
Step 10: State=[-0.46381676  0.00790881], Reward=-1.0
Step 11: State=[-0.45535392  0.00846285], Reward=-1.0
Step 12: State=[-0.44639934  0.00895458], Reward=-1.0
Step 13: State=[-0.4370186   0.00938074], Reward=-1.0
Step 14: State=[-0.42727993  0.00973867], Reward=-1.0
Step 15: State=[-0.41725364  0.01002629], Reward=-1.0
Step 16: State=[-0.40701147  0.01024216], Reward=-1.0
Step 17: State=[-0.396626    0.01038548], Reward=-1.0
Step 18: State=[-0.38616995  0.01045604], Reward=-1.0
Step 19: State=[-0.37571567  0.01045428], Reward=-1.0
Step 20: State=[-0.36533449  0.01038118], Reward=-1.0
Step 21: State=[-0.35509619  0.0102383 ], Reward=-1.0
Step 22: State=[-0.34506852  0.01002767], Reward=-1.0
Step 23: State=[-0.33531672  0.0097518 ], Reward=-1.0
Step 24: State=[-0.32590314  0.00941358], Reward=-1.0
Step 25: State=[-0.31688687  0.00901627], Reward=-1.0
Step 26: State=[-0.30832346  0.00856341], Reward=-1.0
Step 27: State=[-0.30026469  0.00805876], Reward=-1.0
Step 28: State=[-0.2927584   0.00750629], Reward=-1.0
Step 29: State=[-0.2858483  0.0069101], Reward=-1.0
Step 30: State=[-0.27957395  0.00627436], Reward=-1.0
Step 31: State=[-0.27397063  0.00560332], Reward=-1.0
Step 32: State=[-0.26906936  0.00490127], Reward=-1.0
Step 33: State=[-0.26489689  0.00417247], Reward=-1.0
Step 34: State=[-0.26147568  0.00342121], Reward=-1.0
Step 35: State=[-0.25882396  0.00265172], Reward=-1.0
Step 36: State=[-0.25695571  0.00186825], Reward=-1.0
Step 37: State=[-0.25588073  0.00107498], Reward=-1.0
Step 38: State=[-0.25560462  0.00027611], Reward=-1.0
Step 39: State=[-0.25612883 -0.00052421], Reward=-1.0
Step 40: State=[-0.25745062 -0.00132179], Reward=-1.0
Step 41: State=[-0.25956309 -0.00211247], Reward=-1.0
Step 42: State=[-0.26245514 -0.00289205], Reward=-1.0
Step 43: State=[-0.26611148 -0.00365634], Reward=-1.0
Step 44: State=[-0.27051257 -0.00440109], Reward=-1.0
Step 45: State=[-0.27563463 -0.00512205], Reward=-1.0
Step 46: State=[-0.28144957 -0.00581494], Reward=-1.0
Step 47: State=[-0.28792506 -0.00647549], Reward=-1.0
Step 48: State=[-0.29502448 -0.00709942], Reward=-1.0
Step 49: State=[-0.30270698 -0.0076825 ], Reward=-1.0
Step 50: State=[-0.31092755 -0.00822057], Reward=-1.0
Step 51: State=[-0.31963713 -0.00870957], Reward=-1.0
Step 52: State=[-0.32878273 -0.0091456 ], Reward=-1.0
Step 53: State=[-0.33830768 -0.00952495], Reward=-1.0
Step 54: State=[-0.34815185 -0.00984416], Reward=-1.0
Step 55: State=[-0.35825194 -0.0101001 ], Reward=-1.0
Step 56: State=[-0.36854191 -0.01028996], Reward=-1.0
Step 57: State=[-0.37895331 -0.0104114 ], Reward=-1.0
Step 58: State=[-0.38941582 -0.01046252], Reward=-1.0
Step 59: State=[-0.39985775 -0.01044193], Reward=-1.0
Step 60: State=[-0.41020657 -0.01034882], Reward=-1.0
Step 61: State=[-0.42038952 -0.01018295], Reward=-1.0
Step 62: State=[-0.43033423 -0.00994471], Reward=-1.0
Step 63: State=[-0.43996933 -0.0096351 ], Reward=-1.0
Step 64: State=[-0.4492251  -0.00925577], Reward=-1.0
Step 65: State=[-0.45803405 -0.00880895], Reward=-1.0
Step 66: State=[-0.46633157 -0.00829752], Reward=-1.0
Step 67: State=[-0.47405649 -0.00772492], Reward=-1.0
Step 68: State=[-0.48115161 -0.00709512], Reward=-1.0
Step 69: State=[-0.48756422 -0.00641261], Reward=-1.0
Step 70: State=[-0.49324656 -0.00568234], Reward=-1.0
Step 71: State=[-0.49815623 -0.00490967], Reward=-1.0
Step 72: State=[-0.50225654 -0.00410031], Reward=-1.0
Step 73: State=[-0.5055168  -0.00326026], Reward=-1.0
Step 74: State=[-0.50791261 -0.00239581], Reward=-1.0
Step 75: State=[-0.50942603 -0.00151341], Reward=-1.0
Step 76: State=[-0.5100457  -0.00061968], Reward=-1.0
Step 77: State=[-5.09767002e-01  2.78702550e-04], Reward=-1.0
Step 78: State=[-0.50859201  0.00117499], Reward=-1.0
Step 79: State=[-0.50652953  0.00206248], Reward=-1.0
Step 80: State=[-0.50359501  0.00293452], Reward=-1.0
Step 81: State=[-0.49981043  0.00378458], Reward=-1.0
Step 82: State=[-0.49520411  0.00460632], Reward=-1.0
Step 83: State=[-0.48981049  0.00539362], Reward=-1.0
Step 84: State=[-0.48366986  0.00614064], Reward=-1.0
Step 85: State=[-0.47682797  0.00684189], Reward=-1.0
Step 86: State=[-0.46933572  0.00749226], Reward=-1.0
Step 87: State=[-0.46124864  0.00808708], Reward=-1.0
Step 88: State=[-0.45262646  0.00862217], Reward=-1.0
Step 89: State=[-0.44353257  0.00909389], Reward=-1.0
Step 90: State=[-0.43403342  0.00949915], Reward=-1.0
Step 91: State=[-0.42419795  0.00983547], Reward=-1.0
Step 92: State=[-0.41409699  0.01010096], Reward=-1.0
Step 93: State=[-0.40380259  0.01029439], Reward=-1.0
Step 94: State=[-0.39338746  0.01041514], Reward=-1.0
Step 95: State=[-0.38292426  0.0104632 ], Reward=-1.0
Step 96: State=[-0.37248508  0.01043918], Reward=-1.0
Step 97: State=[-0.36214083  0.01034425], Reward=-1.0
Step 98: State=[-0.35196071  0.01018012], Reward=-1.0
Step 99: State=[-0.34201175  0.00994897], Reward=-1.0
Step 100: State=[-0.33235831  0.00965343], Reward=-1.0
Step 101: State=[-0.32306179  0.00929653], Reward=-1.0
Step 102: State=[-0.31418019  0.0088816 ], Reward=-1.0
Step 103: State=[-0.30576792  0.00841226], Reward=-1.0
Step 104: State=[-0.29787557  0.00789236], Reward=-1.0
Step 105: State=[-0.29054969  0.00732588], Reward=-1.0
Step 106: State=[-0.28383272  0.00671697], Reward=-1.0
Step 107: State=[-0.27776289  0.00606983], Reward=-1.0
Step 108: State=[-0.27237418  0.00538871], Reward=-1.0
Step 109: State=[-0.26769627  0.00467791], Reward=-1.0
Step 110: State=[-0.26375458  0.00394169], Reward=-1.0
Step 111: State=[-0.26057026  0.00318432], Reward=-1.0
Step 112: State=[-0.25816021  0.00241005], Reward=-1.0
Step 113: State=[-0.25653713  0.00162309], Reward=-1.0
Step 114: State=[-0.25570949  0.00082763], Reward=-1.0
Step 115: State=[-2.55681628e-01  2.78670044e-05], Reward=-1.0
Step 116: State=[-0.25645367 -0.00077204], Reward=-1.0
Step 117: State=[-0.25802161 -0.00156793], Reward=-1.0
Step 118: State=[-0.26037723 -0.00235562], Reward=-1.0
Step 119: State=[-0.26350814 -0.00313091], Reward=-1.0
Step 120: State=[-0.26739774 -0.0038896 ], Reward=-1.0
Step 121: State=[-0.27202516 -0.00462742], Reward=-1.0
Step 122: State=[-0.2773653  -0.00534014], Reward=-1.0
Step 123: State=[-0.28338876 -0.00602346], Reward=-1.0
Step 124: State=[-0.29006186 -0.0066731 ], Reward=-1.0
Step 125: State=[-0.29734667 -0.00728481], Reward=-1.0
Step 126: State=[-0.30520105 -0.00785438], Reward=-1.0
Step 127: State=[-0.31357871 -0.00837766], Reward=-1.0
Step 128: State=[-0.32242935 -0.00885064], Reward=-1.0
Step 129: State=[-0.33169883 -0.00926948], Reward=-1.0
Step 130: State=[-0.34132937 -0.00963053], Reward=-1.0
Step 131: State=[-0.35125981 -0.00993044], Reward=-1.0
Step 132: State=[-0.36142598 -0.01016617], Reward=-1.0
Step 133: State=[-0.37176102 -0.01033504], Reward=-1.0
Step 134: State=[-0.38219587 -0.01043485], Reward=-1.0
Step 135: State=[-0.39265972 -0.01046385], Reward=-1.0
Step 136: State=[-0.40308055 -0.01042083], Reward=-1.0
Step 137: State=[-0.41338571 -0.01030515], Reward=-1.0
Step 138: State=[-0.42350248 -0.01011677], Reward=-1.0
Step 139: State=[-0.43335875 -0.00985626], Reward=-1.0
Step 140: State=[-0.44288357 -0.00952483], Reward=-1.0
Step 141: State=[-0.45200787 -0.00912429], Reward=-1.0
Step 142: State=[-0.46066497 -0.00865711], Reward=-1.0
Step 143: State=[-0.46879128 -0.00812631], Reward=-1.0
Step 144: State=[-0.4763268  -0.00753552], Reward=-1.0
Step 145: State=[-0.48321567 -0.00688887], Reward=-1.0
Step 146: State=[-0.48940667 -0.006191  ], Reward=-1.0
Step 147: State=[-0.49485367 -0.00544699], Reward=-1.0
Step 148: State=[-0.49951598 -0.00466232], Reward=-1.0
Step 149: State=[-0.50335876 -0.00384278], Reward=-1.0
Step 150: State=[-0.50635325 -0.00299449], Reward=-1.0
Step 151: State=[-0.50847702 -0.00212377], Reward=-1.0
Step 152: State=[-0.50971416 -0.00123714], Reward=-1.0
Step 153: State=[-5.10055410e-01 -3.41248589e-04], Reward=-1.0
Step 154: State=[-0.50949821  0.0005572 ], Reward=-1.0
Step 155: State=[-0.50804672  0.00145148], Reward=-1.0
Step 156: State=[-0.50571184  0.00233488], Reward=-1.0
Step 157: State=[-0.50251105  0.0032008 ], Reward=-1.0
Step 158: State=[-0.4984683   0.00404274], Reward=-1.0
Step 159: State=[-0.49361386  0.00485444], Reward=-1.0
Step 160: State=[-0.487984    0.00562986], Reward=-1.0
Step 161: State=[-0.48162074  0.00636326], Reward=-1.0
Step 162: State=[-0.47457149  0.00704925], Reward=-1.0
Step 163: State=[-0.46688862  0.00768287], Reward=-1.0
Step 164: State=[-0.45862902  0.0082596 ], Reward=-1.0
Step 165: State=[-0.44985362  0.0087754 ], Reward=-1.0
Step 166: State=[-0.44062681  0.00922681], Reward=-1.0
Step 167: State=[-0.43101588  0.00961093], Reward=-1.0
Step 168: State=[-0.42109043  0.00992545], Reward=-1.0
Step 169: State=[-0.41092173  0.0101687 ], Reward=-1.0
Step 170: State=[-0.4005821   0.01033962], Reward=-1.0
Step 171: State=[-0.3901443  0.0104378], Reward=-1.0
Step 172: State=[-0.37968088  0.01046342], Reward=-1.0
Step 173: State=[-0.36926363  0.01041726], Reward=-1.0
Step 174: State=[-0.35896297  0.01030066], Reward=-1.0
Step 175: State=[-0.34884748  0.01011548], Reward=-1.0
Step 176: State=[-0.33898342  0.00986407], Reward=-1.0
Step 177: State=[-0.32943426  0.00954916], Reward=-1.0
Step 178: State=[-0.32026037  0.00917389], Reward=-1.0
Step 179: State=[-0.31151868  0.00874169], Reward=-1.0
Step 180: State=[-0.30326242  0.00825625], Reward=-1.0
Step 181: State=[-0.29554096  0.00772147], Reward=-1.0
Step 182: State=[-0.28839957  0.00714139], Reward=-1.0
Step 183: State=[-0.28187941  0.00652016], Reward=-1.0
Step 184: State=[-0.27601738  0.00586203], Reward=-1.0
Step 185: State=[-0.27084613  0.00517125], Reward=-1.0
Step 186: State=[-0.26639402  0.00445211], Reward=-1.0
Step 187: State=[-0.26268515  0.00370887], Reward=-1.0
Step 188: State=[-0.25973934  0.00294581], Reward=-1.0
Step 189: State=[-0.25757219  0.00216715], Reward=-1.0
Step 190: State=[-0.25619508  0.00137711], Reward=-1.0
Step 191: State=[-0.25561521  0.00057987], Reward=-1.0
Step 192: State=[-2.55835595e-01 -2.20385847e-04], Reward=-1.0
Step 193: State=[-0.25685509 -0.0010195 ], Reward=-1.0
Step 194: State=[-0.25866838 -0.00181329], Reward=-1.0
Step 195: State=[-0.26126596 -0.00259758], Reward=-1.0
Step 196: State=[-0.26463414 -0.00336818], Reward=-1.0
Step 197: State=[-0.26875498 -0.00412085], Reward=-1.0
Step 198: State=[-0.27360632 -0.00485134], Reward=-1.0
Step 199: State=[-0.27916172 -0.0055554 ], Reward=-1.0
Step 200: State=[-0.28539045 -0.00622873], Reward=-1.0
```

It helps to visualize the car. The following code shows a video of the car when run from a notebook.

In [6]:
```
# HIDE OUTPUT
show_video()
```



Programmed Car

Now we will look at a car that I hand-programmed. This car is straightforward; however, it solves the problem. The programmed car always applies force in one direction or another. It does not break. Whatever direction the vehicle is currently rolling, the agent uses power in that direction. Therefore, the car begins to climb a hill, is overpowered, and turns backward. However, once it starts to roll backward, force is immediately applied in this new direction.

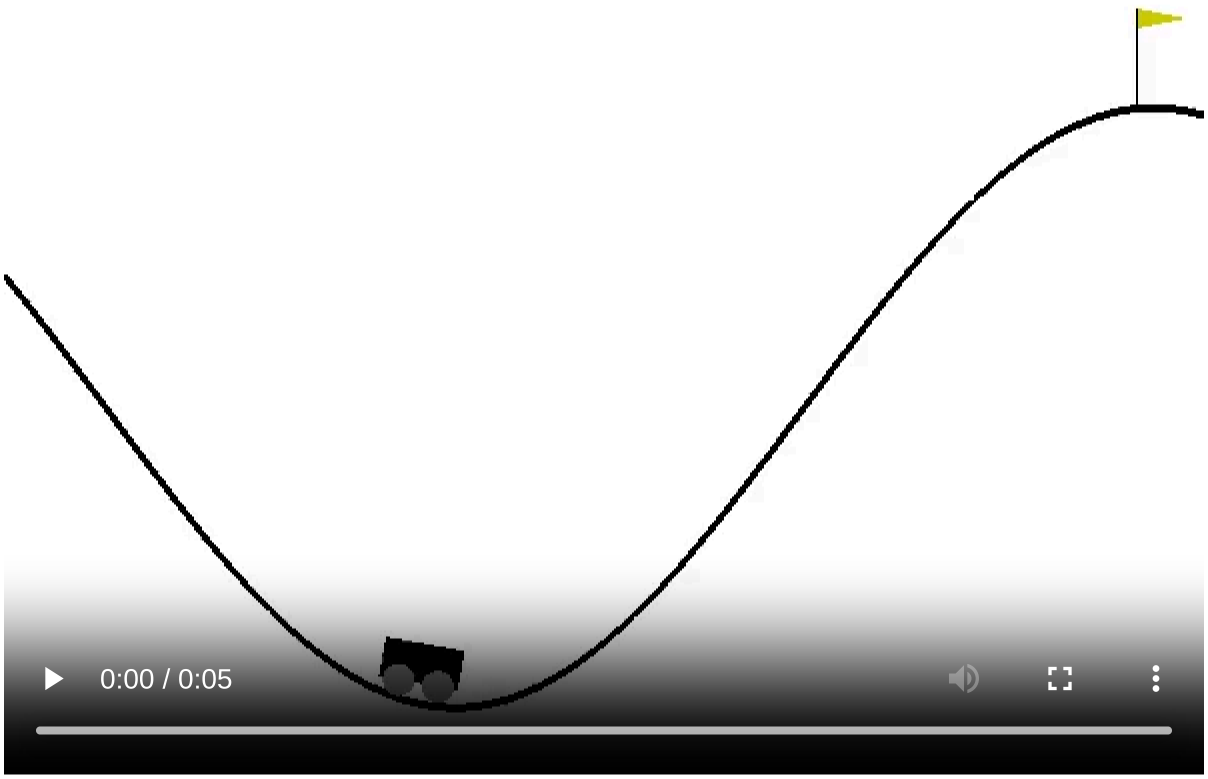
The following code implements this preprogrammed car.

In [7]:
```
import gym

if COLAB:
    env = wrap_env(gym.make("MountainCar-v0"))
else:
    env = gym.make("MountainCar-v0")

state = env.reset()
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}")

env.close()
```

```
Step 1: State=[-5.84581471e-01 -5.49227966e-04], Reward=-1.0
Step 2: State=[-0.58567588 -0.0010944 ], Reward=-1.0
Step 3: State=[-0.58730739 -0.00163151], Reward=-1.0
Step 4: State=[-0.58946399 -0.0021566 ], Reward=-1.0
Step 5: State=[-0.59212981 -0.00266582], Reward=-1.0
Step 6: State=[-0.59528526 -0.00315545], Reward=-1.0
Step 7: State=[-0.5989072  -0.00362194], Reward=-1.0
Step 8: State=[-0.60296912 -0.00406192], Reward=-1.0
Step 9: State=[-0.60744137 -0.00447225], Reward=-1.0
Step 10: State=[-0.61229141 -0.00485004], Reward=-1.0
Step 11: State=[-0.61748407 -0.00519267], Reward=-1.0
Step 12: State=[-0.62298187 -0.0054978 ], Reward=-1.0
Step 13: State=[-0.62874529 -0.00576342], Reward=-1.0
Step 14: State=[-0.63473313 -0.00598783], Reward=-1.0
Step 15: State=[-0.64090281 -0.00616968], Reward=-1.0
Step 16: State=[-0.64721076 -0.00630795], Reward=-1.0
Step 17: State=[-0.65361272 -0.00640196], Reward=-1.0
Step 18: State=[-0.66006412 -0.00645139], Reward=-1.0
Step 19: State=[-0.66652037 -0.00645626], Reward=-1.0
Step 20: State=[-0.67293726 -0.00641689], Reward=-1.0
Step 21: State=[-0.6792712  -0.00633394], Reward=-1.0
Step 22: State=[-0.68547958 -0.00620838], Reward=-1.0
Step 23: State=[-0.69152102 -0.00604144], Reward=-1.0
Step 24: State=[-0.69735564 -0.00583462], Reward=-1.0
Step 25: State=[-0.7029453  -0.00558966], Reward=-1.0
Step 26: State=[-0.70825383 -0.00530853], Reward=-1.0
Step 27: State=[-0.7132472  -0.00499337], Reward=-1.0
Step 28: State=[-0.71789372 -0.00464651], Reward=-1.0
Step 29: State=[-0.72216414 -0.00427042], Reward=-1.0
Step 30: State=[-0.72603185 -0.00386771], Reward=-1.0
Step 31: State=[-0.72947294 -0.00344108], Reward=-1.0
Step 32: State=[-0.73246627 -0.00299334], Reward=-1.0
Step 33: State=[-0.73499362 -0.00252735], Reward=-1.0
Step 34: State=[-0.73703966 -0.00204604], Reward=-1.0
Step 35: State=[-0.73859207 -0.00155241], Reward=-1.0
Step 36: State=[-0.73964152 -0.00104945], Reward=-1.0
Step 37: State=[-7.40181738e-01 -5.40214614e-04], Reward=-1.0
Step 38: State=[-7.40209487e-01 -2.77484127e-05], Reward=-1.0
Step 39: State=[-7.39724603e-01  4.84883491e-04], Reward=-1.0
Step 40: State=[-0.73672998  0.00299462], Reward=-1.0
Step 41: State=[-0.73124359  0.00548639], Reward=-1.0
Step 42: State=[-0.72329865  0.00794494], Reward=-1.0
Step 43: State=[-0.71294396  0.01035469], Reward=-1.0
Step 44: State=[-0.70024433  0.01269963], Reward=-1.0
Step 45: State=[-0.685281    0.01496333], Reward=-1.0
Step 46: State=[-0.66815204  0.01712895], Reward=-1.0
Step 47: State=[-0.6489726   0.01917944], Reward=-1.0
Step 48: State=[-0.62787487  0.02109773], Reward=-1.0
Step 49: State=[-0.60500776  0.02286711], Reward=-1.0
Step 50: State=[-0.58053614  0.02447162], Reward=-1.0
Step 51: State=[-0.55463956  0.02589658], Reward=-1.0
Step 52: State=[-0.52751051  0.02712905], Reward=-1.0
Step 53: State=[-0.49935212  0.02815839], Reward=-1.0
Step 54: State=[-0.47037542  0.0289767 ], Reward=-1.0
Step 55: State=[-0.44079621  0.02957922], Reward=-1.0
Step 56: State=[-0.41083164  0.02996456], Reward=-1.0
Step 57: State=[-0.38069679  0.03013485], Reward=-1.0
Step 58: State=[-0.35060117  0.03009562], Reward=-1.0
Step 59: State=[-0.32074557  0.0298556 ], Reward=-1.0
Step 60: State=[-0.29131919  0.02942639], Reward=-1.0
Step 61: State=[-0.26249729  0.02882189], Reward=-1.0
Step 62: State=[-0.23443946  0.02805783], Reward=-1.0
Step 63: State=[-0.20728838  0.02715108], Reward=-1.0
Step 64: State=[-0.18116928  0.0261191 ], Reward=-1.0
Step 65: State=[-0.15618993  0.02497935], Reward=-1.0
Step 66: State=[-0.13244112  0.02374881], Reward=-1.0
Step 67: State=[-0.10999756  0.02244356], Reward=-1.0
Step 68: State=[-0.08891911  0.02107845], Reward=-1.0
Step 69: State=[-0.06925224  0.01966687], Reward=-1.0
Step 70: State=[-0.05103161  0.01822063], Reward=-1.0
Step 71: State=[-0.03428174  0.01674987], Reward=-1.0
Step 72: State=[-0.01901866  0.01526308], Reward=-1.0
Step 73: State=[-0.00525151  0.01376715], Reward=-1.0
Step 74: State=[0.00701595 0.01226746], Reward=-1.0
Step 75: State=[0.01778397 0.01076801], Reward=-1.0
Step 76: State=[0.02705554 0.00927157], Reward=-1.0
Step 77: State=[0.03483534 0.0077798 ], Reward=-1.0
Step 78: State=[0.04112878 0.00629344], Reward=-1.0
Step 79: State=[0.04594123 0.00481245], Reward=-1.0
Step 80: State=[0.04927738 0.00333615], Reward=-1.0
Step 81: State=[0.05114081 0.00186342], Reward=-1.0
Step 82: State=[0.05153359 0.00039279], Reward=-1.0
Step 83: State=[ 0.0504562 -0.0010774], Reward=-1.0
Step 84: State=[ 0.04590739 -0.00454881], Reward=-1.0
Step 85: State=[ 0.03788225 -0.00802514], Reward=-1.0
Step 86: State=[ 0.02637324 -0.01150901], Reward=-1.0
Step 87: State=[ 0.01137205 -0.01500119], Reward=-1.0
Step 88: State=[-0.00712768 -0.01849973], Reward=-1.0
Step 89: State=[-0.02912685 -0.02199916], Reward=-1.0
Step 90: State=[-0.05461647 -0.02548963], Reward=-1.0
Step 91: State=[-0.08357261 -0.02895614], Reward=-1.0
Step 92: State=[-0.11595059 -0.03237798], Reward=-1.0
Step 93: State=[-0.15167884 -0.03572825], Reward=-1.0
Step 94: State=[-0.1906527  -0.03897386], Reward=-1.0
Step 95: State=[-0.23272866 -0.04207597], Reward=-1.0
Step 96: State=[-0.27771965 -0.04499099], Reward=-1.0
Step 97: State=[-0.32539199 -0.04767234], Reward=-1.0
Step 98: State=[-0.37546482 -0.05007283], Reward=-1.0
Step 99: State=[-0.42761244 -0.05214762], Reward=-1.0
Step 100: State=[-0.48147006 -0.05385761], Reward=-1.0
Step 101: State=[-0.5366428  -0.05517274], Reward=-1.0
Step 102: State=[-0.59271773 -0.05607493], Reward=-1.0
Step 103: State=[-0.64927797 -0.05656025], Reward=-1.0
Step 104: State=[-0.7059178  -0.05663983], Reward=-1.0
Step 105: State=[-0.7622574 -0.0563396], Reward=-1.0
Step 106: State=[-0.81795612 -0.05569872], Reward=-1.0
Step 107: State=[-0.8727231  -0.05476698], Reward=-1.0
Step 108: State=[-0.92632481 -0.0536017 ], Reward=-1.0
Step 109: State=[-0.97858908 -0.05226427], Reward=-1.0
Step 110: State=[-1.02940612 -0.05081704], Reward=-1.0
Step 111: State=[-1.07872672 -0.0493206 ], Reward=-1.0
Step 112: State=[-1.1265585  -0.04783178], Reward=-1.0
Step 113: State=[-1.1729608 -0.0464023], Reward=-1.0
Step 114: State=[-1.2  0. ], Reward=-1.0
Step 115: State=[-1.1987581  0.0012419], Reward=-1.0
Step 116: State=[-1.19427021  0.0044879 ], Reward=-1.0
Step 117: State=[-1.18652173  0.00774848], Reward=-1.0
Step 118: State=[-1.17548846  0.01103326], Reward=-1.0
Step 119: State=[-1.16113808  0.01435038], Reward=-1.0
Step 120: State=[-1.14343234  0.01770574], Reward=-1.0
Step 121: State=[-1.12233007  0.02110228], Reward=-1.0
Step 122: State=[-1.09779103  0.02453904], Reward=-1.0
Step 123: State=[-1.06978073  0.0280103 ], Reward=-1.0
Step 124: State=[-1.03827616  0.03150456], Reward=-1.0
Step 125: State=[-1.0032725   0.03500367], Reward=-1.0
Step 126: State=[-0.9647905   0.03848199], Reward=-1.0
Step 127: State=[-0.92288452  0.04190598], Reward=-1.0
Step 128: State=[-0.87765038  0.04523414], Reward=-1.0
Step 129: State=[-0.82923273  0.04841765], Reward=-1.0
Step 130: State=[-0.77783078  0.05140195], Reward=-1.0
Step 131: State=[-0.72370164  0.05412914], Reward=-1.0
Step 132: State=[-0.66716026  0.05654138], Reward=-1.0
Step 133: State=[-0.60857514  0.05858511], Reward=-1.0
Step 134: State=[-0.54835959  0.06021555], Reward=-1.0
Step 135: State=[-0.4869585   0.06140109], Reward=-1.0
Step 136: State=[-0.42483166  0.06212684], Reward=-1.0
Step 137: State=[-0.36243478  0.06239688], Reward=-1.0
Step 138: State=[-0.30020009  0.06223469], Reward=-1.0
Step 139: State=[-0.23851824  0.06168185], Reward=-1.0
Step 140: State=[-0.17772322  0.06079502], Reward=-1.0
Step 141: State=[-0.1180812   0.05964202], Reward=-1.0
Step 142: State=[-0.05978395  0.05829725], Reward=-1.0
Step 143: State=[-0.0029466   0.05683735], Reward=-1.0
Step 144: State=[0.05239085 0.05533745], Reward=-1.0
Step 145: State=[0.10625911 0.05386826], Reward=-1.0
Step 146: State=[0.15875332 0.05249421], Reward=-1.0
Step 147: State=[0.21002575 0.05127242], Reward=-1.0
Step 148: State=[0.26027822 0.05025247], Reward=-1.0
Step 149: State=[0.30975487 0.04947665], Reward=-1.0
Step 150: State=[0.35873547 0.0489806 ], Reward=-1.0
Step 151: State=[0.40752939 0.04879392], Reward=-1.0
Step 152: State=[0.45647027 0.04894088], Reward=-1.0
Step 153: State=[0.50591109 0.04944082], Reward=-1.0
```

We now visualize the preprogrammed car solving the problem.

```
In [8]: # HIDE OUTPUT
show_video()
```

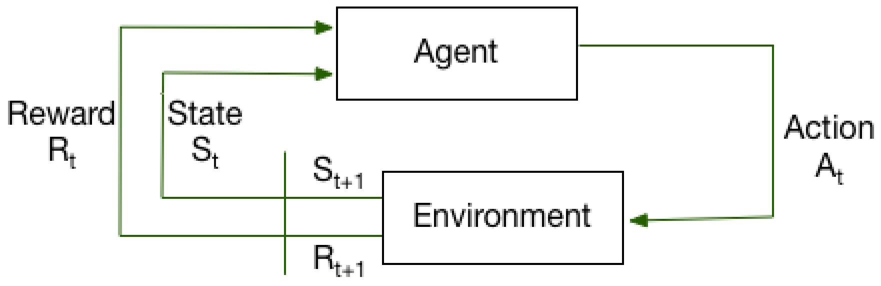


Reinforcement Learning

Q-Learning is a system of rewards that the algorithm gives an agent for successfully moving the environment into a state considered successful. These rewards are the Q-values from which this algorithm takes its name. The final output from the Q-Learning algorithm is a table of Q-values that indicate the reward value of every action that the agent can take, given every possible environment state. The agent must bin continuous state values into a fixed finite number of columns.

Learning occurs when the algorithm runs the agent and environment through episodes and updates the Q-values based on the rewards received from actions taken; Figure 12.REINF provides a high-level overview of this reinforcement or Q-Learning loop.

Figure 12.REINF:Reinforcement/Q Learning



The Q-values can dictate action by selecting the action column with the highest Q-value for the current environment state. The choice between choosing a random action and a Q-value-driven action is governed by the epsilon (ϵ) parameter, the probability of random action.

Each time through the training loop, the training algorithm updates the Q-values according to the following equation.

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}}}^{\text{temporal difference}}$$

There are several parameters in this equation:

- alpha (α) - The learning rate, how much should the current step cause the Q-values to be updated.
- lambda (λ) - The discount factor is the percentage of future reward that the algorithm should consider in this update.

This equation modifies several values:

- $Q(s_t,a_t)$ - The Q-table. For each combination of states, what reward would the agent likely receive for performing each action?
- s_t - The current state.
- r_t - The last reward received.
- a_t - The action that the agent will perform.

The equation works by calculating a delta (temporal difference) that the equation should apply to the old state. This learning rate (α) scales this delta. A learning rate of 1.0 would fully implement the temporal difference in the Q-values each iteration and would likely be very chaotic.

There are two parts to the temporal difference: the new and old values. The new value is subtracted from the old value to provide a delta; the full amount we would change the Q-value by if the learning rate did not scale this value. The new value is a summation of the reward received from the last action and the maximum Q-values from the resulting state when the client takes this action. Adding the maximum of action Q-values for the new state is essential because it estimates the optimal future values from proceeding with this action.

Q-Learning Car

We will now use Q-Learning to produce a car that learns to drive itself. Look out, Tesla! We begin by defining two essential functions.

In [9]:
```
import gym
import numpy as np

# This function converts the floating point state values into
# discrete values. This is often called binning.  We divide
# the range that the state values might occupy and assign
# each region to a bucket.
def calc_discrete_state(state):
    discrete_state = (state - env.observation_space.low)/buckets
    return tuple(discrete_state.astype(int))

# Run one game.  The q_table to use is provided.  We also
# provide a flag to indicate if the game should be
# rendered/animated.  Finally, we also provide
# a flag to indicate if the q_table should be updated.
def run_game(q_table, render, should_update):
    done = False
    discrete_state = calc_discrete_state(env.reset())
    success = False

    while not done:
        # Exploit or explore
        if np.random.random() > epsilon:
            # Exploit - use q-table to take current best action
            # (and probably refine)
            action = np.argmax(q_table[discrete_state])
        else:
            # Explore - t
            action = np.random.randint(0, env.action_space.n)

        # Run simulation step
        new_state, reward, done, _ = env.step(action)

        # Convert continuous state to discrete
        new_state_disc = calc_discrete_state(new_state)

        # Have we reached the goal position (have we won?)?
        if new_state[0] >= env.unwrapped.goal_position:
            success = True

        # Update q-table
        if should_update:
            max_future_q = np.max(q_table[new_state_disc])
            current_q = q_table[discrete_state + (action,)]
            new_q = (1 - LEARNING_RATE) * current_q + LEARNING_RATE * \
                (reward + DISCOUNT * max_future_q)
            q_table[discrete_state + (action,)] = new_q

        discrete_state = new_state_disc

        if render:
                env.render()

    return success
```

Several hyperparameters are very important for Q-Learning. These parameters will likely need adjustment as you apply Q-Learning to other problems. Because of this, it is crucial to understand the role of each parameter.

- **LEARNING_RATE** The rate at which previous Q-values are updated based on new episodes run during training.
- **DISCOUNT** The amount of significance to give estimates of future rewards when added to the reward for the current action taken. A value of 0.95 would indicate a discount of 5% on the future reward estimates.
- **EPISODES** The number of episodes to train over. Increase this for more complex problems; however, training time also increases.
- **SHOW_EVERY** How many episodes to allow to elapse before showing an update.
- **DISCRETE_GRID_SIZE** How many buckets to use when converting each continuous state variable. For example, [10, 10] indicates that the algorithm should use ten buckets for the first and second state variables.
- **START_EPSILON_DECAYING** Epsilon is the probability that the agent will select a random action over what the Q-Table suggests. This value determines the starting probability of randomness.
- **END_EPSILON_DECAYING** How many episodes should elapse before epsilon goes to zero and no random actions are permitted. For example, EPISODES//10 means only the first 1/10th of the episodes might have random actions.

In [10]:
```
LEARNING_RATE = 0.1
DISCOUNT = 0.95
EPISODES = 50000
SHOW_EVERY = 1000

DISCRETE_GRID_SIZE = [10, 10]
START_EPSILON_DECAYING = 0.5
END_EPSILON_DECAYING = EPISODES//10
```

We can now make the environment. If we are running in Google COLAB, we wrap the environment to be displayed inside the web browser. Next, create the discrete buckets for state and build Q-table.

In [11]:
```
if COLAB:
    env = wrap_env(gym.make("MountainCar-v0"))
else:
    env = gym.make("MountainCar-v0")

epsilon = 1
epsilon_change = epsilon/(END_EPSILON_DECAYING - START_EPSILON_DECAYING)
buckets = (env.observation_space.high - env.observation_space.low) \
    / DISCRETE_GRID_SIZE
```

```
q_table = np.random.uniform(low=-3, high=0, size=(DISCRETE_GRID_SIZE
                                                  + [env.action_space.n]))
success = False
```

We can now make the environment. If we are running in Google COLAB, we wrap the environment to be displayed inside the web browser. Next, create the discrete buckets for state and build Q-table.

In [12]:
```
episode = 0
success_count = 0

# Loop through the required number of episodes
while episode < EPISODES:
    episode += 1
    done = False

    # Run the game.  If we are local, display render animation
    # at SHOW_EVERY intervals.
    if episode % SHOW_EVERY == 0:
        print(f"Current episode: {episode}, success: {success_count}" +
              f" {(float(success_count)/SHOW_EVERY)}")
        success = run_game(q_table, True, False)
        success_count = 0
    else:
        success = run_game(q_table, False, True)

    # Count successes
    if success:
        success_count += 1

    # Move epsilon towards its ending value, if it still needs to move
    if END_EPSILON_DECAYING >= episode >= START_EPSILON_DECAYING:
        epsilon = max(0, epsilon - epsilon_change)

print(success)
```

```
Current episode: 1000, success: 0 0.0
Current episode: 2000, success: 0 0.0
Current episode: 3000, success: 0 0.0
Current episode: 4000, success: 31 0.031
Current episode: 5000, success: 321 0.321
Current episode: 6000, success: 602 0.602
Current episode: 7000, success: 620 0.62
Current episode: 8000, success: 821 0.821
Current episode: 9000, success: 707 0.707
Current episode: 10000, success: 714 0.714
Current episode: 11000, success: 574 0.574
Current episode: 12000, success: 443 0.443
Current episode: 13000, success: 480 0.48
Current episode: 14000, success: 458 0.458
Current episode: 15000, success: 327 0.327
Current episode: 16000, success: 323 0.323
Current episode: 17000, success: 295 0.295
Current episode: 18000, success: 314 0.314
Current episode: 19000, success: 362 0.362
Current episode: 20000, success: 488 0.488
Current episode: 21000, success: 566 0.566
Current episode: 22000, success: 591 0.591
Current episode: 23000, success: 441 0.441
Current episode: 24000, success: 385 0.385
Current episode: 25000, success: 1000 1.0
Current episode: 26000, success: 1000 1.0
Current episode: 27000, success: 993 0.993
Current episode: 28000, success: 67 0.067
Current episode: 29000, success: 0 0.0
Current episode: 30000, success: 39 0.039
Current episode: 31000, success: 204 0.204
Current episode: 32000, success: 429 0.429
Current episode: 33000, success: 345 0.345
Current episode: 34000, success: 970 0.97
Current episode: 35000, success: 583 0.583
Current episode: 36000, success: 752 0.752
Current episode: 37000, success: 955 0.955
Current episode: 38000, success: 997 0.997
Current episode: 39000, success: 1000 1.0
Current episode: 40000, success: 1000 1.0
Current episode: 41000, success: 1000 1.0
Current episode: 42000, success: 1000 1.0
Current episode: 43000, success: 1000 1.0
Current episode: 44000, success: 1000 1.0
Current episode: 45000, success: 1000 1.0
Current episode: 46000, success: 1000 1.0
Current episode: 47000, success: 1000 1.0
Current episode: 48000, success: 1000 1.0
Current episode: 49000, success: 1000 1.0
Current episode: 50000, success: 1000 1.0
True
```

As you can see, the number of successful episodes generally increases as training progresses. It is not advisable to stop the first time we observe 100% success over 1,000 episodes. There is a randomness to most games, so it is not likely that an agent would retain its 100% success rate with a new run. It might be safe to stop training once you observe that the agent has gotten 100% for several update intervals.

Running and Observing the Agent

Now that the algorithm has trained the agent, we can observe the agent in action. You can use the following code to see the agent in action.

In [13]:
```
# HIDE OUTPUT

run_game(q_table, True, False)
show_video()
```

Inspecting the Q-Table

We can also display the Q-table. The following code shows the agent's action for each environment state. As the weights of a neural network, this table is not straightforward to interpret. Some patterns do emerge in that direction, as seen by calculating the means of rows and columns. The actions seem consistent at both velocity and position's upper and lower halves.

In [14]:
```
import pandas as pd

df = pd.DataFrame(q_table.argmax(axis=2))
```

```
df.columns = [f'v-{x}' for x in range(DISCRETE_GRID_SIZE[0])]
df.index = [f'p-{x}' for x in range(DISCRETE_GRID_SIZE[1])]
df
```

Out[14]:

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

In [15]:
```
df.mean(axis=0)
```

Out[15]:
```
v-0    1.4
v-1    1.0
v-2    0.8
v-3    0.9
v-4    1.0
v-5    1.1
v-6    1.7
v-7    1.5
v-8    1.8
v-9    1.4
dtype: float64
```

In [16]:
```
df.mean(axis=1)
```

Out[16]:
```
p-0    1.6
p-1    1.3
p-2    1.1
p-3    1.3
p-4    1.0
p-5    1.2
p-6    1.2
p-7    1.2
p-8    1.2
p-9    1.5
dtype: float64
```