Container_Environment

February 21, 2023

[18]:

```python
# import necessary libraries - (CELL 1)
import os
import itertools
import numpy as np
import gym
from gym import spaces
from stable_baselines3.common.env_checker import check_env
from stable_baselines3.common.env_util import make_vec_env
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3 import PPO, A2C
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from mpl_toolkits.mplot3d import Axes3D
```

[3]:

```python
# block on the container yard - (CELL 2)
class Block:
    id_obj = itertools.count()

    # block consists of locations, block size can be specified (max: rows,␣
 ↪bays, tiers)
    def __init__(self, rows, bays, tiers):
        self.block_id = "B" + str(next(Block.id_obj) + 1)
        self.rows = rows
        self.bays = bays
        self.tiers = tiers
        self.locations = [Location(row, bay, tier) \
                          for row in range(1, rows + 1) \
                          for bay in range(1, bays + 1) \
                          for tier in range(1, tiers + 1)]

    # 3D array representation of wether location has a container (1) or not (0)
    def current_state(self):
        state = np.array([], dtype=np.int32)
        for location in self.locations:
            if location.container is None:
                state = np.append(state, [0])
            else:
                state = np.append(state, [1])
        state = state.reshape(self.rows, self.bays, self.tiers)
        return state

    # find location by (row, bay, tier) coordinate
    def location_by_coordinate(self, row, bay, tier):
        target_location = None
        for location in self.locations:
            if location.row == row and location.bay == bay and location.tier == tier:
                target_location = location
                break
        return target_location

    # delete all containers from locations
    def empty_out(self):
        for location in self.locations:
            location.container = None

    # shows all information
    def info(self):
        return(f"block id: {self.block_id}, "\
               f"maximum rows: {self.rows}, "\
               f"maximum bays: {self.bays}, "\
               f"maximum tiers: {self.tiers}, "\
               f"location amount: {len(self.locations)}\n")


# location in a block
class Location:
    id_obj = itertools.count()

    # locations have no container in them upon creation
    def __init__(self, row, bay, tier, container=None):
        self.location_id = "L" + str(next(Location.id_obj) + 1)
        self.row = row
        self.bay = bay
        self.tier = tier
        self.container = container

    # give (row, bay, tier) coordinate of location in block
    def coordinate(self):
        return self.row, self.bay, self.tier

    def has_container(self):
        return self.container is not None
    # shows all information
    def info(self):
        container_info = None
        if self.container is not None:
            container_info = self.container.container_id
        return(f"location id: {self.location_id}, "\
               f"row: {self.row}, "\
               f"bay: {self.bay}, "\
               f"tier: {self.tier}, "\
               f"container: {container_info}\n")


# container on a vessel or location
class Container:
    id_obj = itertools.count()

    # container has an origin vessel and destination vessel
    def __init__(self, origin_vessel_id, destination_vessel_id):
        self.container_id = 'C' + str(next(Container.id_obj) + 1)
        self.origin_vessel_id = origin_vessel_id
        self.destination_vessel_id = destination_vessel_id

    # shows all information
    def info(self):
        return(f"container id: {self.container_id}, "\
               f"origin vessel id: {self.origin_vessel_id}, "\
               f"destination vessel id: {self.destination_vessel_id}\n")


# vessel with containers
class Vessel:
    id_obj = itertools.count()

    # call containers upon creating a vessel have the same destination
    def __init__(self, max_containers, container_destination_vessel_id, dock):
        self.vessel_id = 'V' + str(next(Vessel.id_obj) + 1)
        self.max_containers = max_containers
        self.container_destination_vessel_id = container_destination_vessel_id
        self.containers = [Container(self.vessel_id,␣
 ↪container_destination_vessel_id) \
                           for container in range(0, max_containers)]
        self.dock = dock
        self.dock.vessels.append(self)

    # delete all containers and recreate them
    def regenerate_containers(self):
        self.containers = [Container(self.vessel_id, self.
 ↪container_destination_vessel_id) \
                           for container in range(0, self.max_containers)]

    # returns the amount of containers in the vessel
    def container_amount(self):
        return len(self.containers)

    # shows all information
    def info(self):
        return(f"vessel id: {self.vessel_id}, "\
               f"maximum containers: {self.max_containers}, "\
               f"container amount: {self.container_amount()}, "\
               f"docked at: {self.dock.dock_id}\n")


# dock in which vessels are stored
class Dock:
    id_obj = itertools.count()

    # dock contains a list of vessels
    def __init__(self):
        self.dock_id = 'D' + str(next(Dock.id_obj) + 1)
        self.vessels = []

    # call the delete and reacreate containers for every vessel in the dock
    def regenerate_containers(self):
        for vessel in self.vessels:
            vessel.regenerate_containers()

    # returns the sum of all containers of every vessels in the dock
    def container_amount(self):
        container_amount = 0
        for vessel in self.vessels:
            container_amount += vessel.container_amount()
        return container_amount

    # shows all information
    def info(self):
        return(f"dock id: {self.dock_id}, "\
               f"vessel amount: {len(self.vessels)}, "\
               f"container amount: {self.container_amount()}\n")
```

[4]:

```
# custom environment with programmed class components - (CELL 3)
class CustomEnv(gym.Env):
    metadata = {'render.modes': ['human']}
    def __init__(self, rows=3, bays=3, tiers=1, containers_per_vessel=4):
        super(CustomEnv, self).__init__()
        # objects needed in the environment
        self.block = Block(rows, bays, tiers)
        self.dock = Dock()
        self.vessel1 = Vessel(containers_per_vessel, 'V98', self.dock)
        self.vessel2 = Vessel(containers_per_vessel, 'V99', self.dock)

        # keep track of total rewards in a single episode
        self.score = 0

        # early termination if too many illegal moves were made
        self.illegal_moves = 0

        # map every location as an action
        self.action_dict = {action: location for action, location in␣
↪enumerate(self.block.locations)}

        # define action space and observation space
        n_actions = len(self.action_dict)
        self.action_space =  spaces.Discrete(n_actions)
        self.observation_space = spaces.Box(low=0, high=1, \
                                            shape=(self.block.rows, self.block.
↪bays, self.block.tiers), dtype=np.int32)

    # move container from vessel to location
    def move_container(self, vessel, location):
        container = vessel.containers.pop(-1)
        location.container = container

    # get the neighboring location of a location based on direction
    def neighbor_location(self, location, direction):
        row, bay, tier = location.coordinate()
        location_neighbor = None
        if direction == 'right':
            location_neighbor = self.block.location_by_coordinate(row + 1, bay,␣
↪tier)
        elif direction == 'left':
            location_neighbor = self.block.location_by_coordinate(row - 1, bay,␣
↪tier)
        elif direction == 'front':
            location_neighbor = self.block.location_by_coordinate(row, bay + 1,␣
↪tier)
        elif direction == 'back':
            location_neighbor = self.block.location_by_coordinate(row, bay - 1,␣
↪tier)
```

```
        elif direction == 'up':
            location_neighbor = self.block.location_by_coordinate(row, bay, tier + 1)
        elif direction == 'down':
            location_neighbor = self.block.location_by_coordinate(row, bay, tier - 1)
        return location_neighbor

    # check if 2 location's containers have the same destination
    def locations_have_same_destination(self, location_1, location_2):
        destination_1 = location_1.container.destination_vessel_id
        destination_2 = location_2.container.destination_vessel_id
        return destination_1 == destination_2

    def step(self, action):
        reward = 0
        done = False

        # get location to be used
        location = self.action_dict.get(action)

        # get vessel which still has a container in it
        vessel = None
        for ves in self.dock.vessels:
            if ves.container_amount() > 0:
                vessel = ves
                break

        # reward if container got placed in a available location
        container_is_placed = False
        if location.has_container() == True:
            self.illegal_moves += 1
            reward -= 20
        else:
            self.move_container(vessel, location)
            container_is_placed = True
            reward += 20

        # reward if container got placed adjacent to a container with similar destination
        if (container_is_placed == True):
            directions = ['right', 'left']
            # get neighboring location
            for direction in directions:
                location_neighbor = self.neighbor_location(location, direction)
                # neighboring location must exist
                if location_neighbor is not None:
                    # neighboring location must have a container
                    if location_neighbor.has_container() == True:
                        # containers must have same destination
                        if self.locations_have_same_destination(location,␣
↪location_neighbor) == True:
                            reward += 10
                        else:
                            reward -= 10

        # update score
        self.score += reward

        # new observation
        observation = self.block.current_state()
        info = {
            'score': self.score,
            'illegal moves': self.illegal_moves
        }

        # done if all containers have been moved out of the vessels in dock
        all_containers_placed = (self.dock.container_amount() == 0)

        # done if too many illegal moves were made
        to_many_illegal_moves = (self.illegal_moves >= 4)

        done = all_containers_placed or to_many_illegal_moves

        return observation, reward, done, info

    def reset(self):
        self.block.empty_out()
        self.dock.regenerate_containers()
        observation = self.block.current_state()
        self.score = 0
        self.illegal_moves = 0
        return observation

    def render(mode='rgb_array'):
        pass

    def close (self):
        pass
```

[5]:

```
# create environment object for testing - (CELL 4)
test_env = CustomEnv()
```

[7]:

```
# check if custom environment meets gym requirements - (CELL 5)
check_env(test_env, warn=True)
```

[15]:

```
# visualization function - (CELL 6)
class Visualizer:
    def __init__(self):
        pass

    def render(self, block):
        # clear any figure and axes if present
        plt.clf()

        # create axes and get data
        axes = [block.rows, block.bays, block.tiers]
        data = block.current_state()

        # map each color string to a RGBA array
        alpha = 0.6
        colors = ['magenta', 'cyan', 'yellow', 'red', 'green', 'blue']
        RGBAs = [[1, 0, 1, alpha], [0, 1, 1, alpha], [1, 1, 0, alpha], \
                 [1, 0, 0, alpha], [0, 1, 0, alpha], [0, 0, 1, alpha]]
        color_RGBA = {}
        for i in range(0, len(colors)):
            color_RGBA.update({colors[i]: RGBAs[i]})
        #print("color_RGBA: ", color_RGBA)

        # get container destination per location in block
        locations = block.locations
        destinations = []
        for location in locations:
            if location.container is not None:
                destinations.append(location.container.destination_vessel_id)
            else:
                destinations.append(None)
        #print("destinations: ", destinations)

        # get all unique destinations, first remove None then remove duplicates
        unique_destinations = list(filter(lambda element: element is not None,␣
 ↪destinations))
        unique_destinations = list(set(unique_destinations))
        #print("unique_destinations: ", unique_destinations)

        # map each unique destination to a color string
        destination_color = {}
        for i in range(0, len(unique_destinations)):
            destination_color.update({unique_destinations[i]: colors[i]})
        #print("destination_color: ", destination_color)
        # map each unique destination to a RGBA array
        destination_RGBA = {}
        for destination in unique_destinations:
            color = destination_color.get(destination)
            RGBA = color_RGBA.get(color)
            destination_RGBA.update({destination: RGBA})
        #print("destination_RGBA: ", destination_RGBA)

        # create 4D facecolor array
        facecolors = np.array([], dtype=np.float32)
        for destination in destinations:
            if destination is not None:
                RGBA = destination_RGBA.get(destination)
                facecolors = np.append(facecolors, RGBA)
            else:
                RGBA = [None, None, None, None]
                facecolors = np.append(facecolors, RGBA)
        #print("facecolors: ", facecolors)
        facecolors = facecolors.reshape(block.rows, block.bays, block.tiers, 4)

        # plot figure
        fig_size = (7, 7)
        fig = plt.figure(figsize=fig_size)
        ax = fig.add_subplot(111, projection='3d')

        # customize axes
        font_size = 12
        ax.set_xlabel('row', fontsize=font_size, rotation=0)
        ax.set_xticks([n for n in range(0, block.rows)])
        ax.set_ylabel('bay', fontsize=font_size, rotation=0)
        ax.set_yticks([n for n in range(0, block.bays)])
        ax.set_zlabel('tier', fontsize=font_size, rotation=0)
        ax.set_zticks([n for n in range(0, block.tiers)])

        # add legend showing which color is which destination
        patches = []
        for destination in destination_color:
            patch = mpatches.Patch(color=destination_color.get(destination),␣
    ↪label=destination)
            patches.append(patch)
        ax.legend(handles=patches)

        # change aspect ratio create container shape
        aspect_ratio = (4, 8, 1)
        ax.set_box_aspect(aspect_ratio)
```

```
        # use voxels to portray containers
        ax.voxels(data, facecolors=facecolors, edgecolors='k')
        plt.show()

visualizer = Visualizer()
```

[9]:

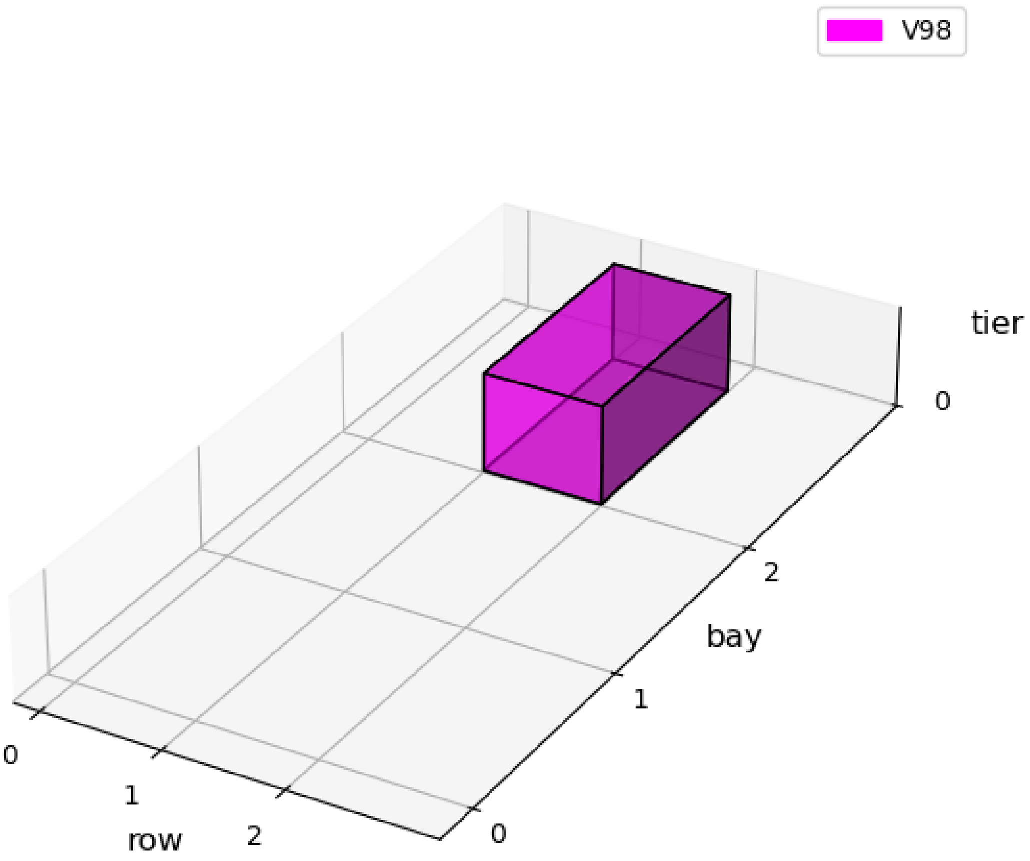
```
# test out the environment action space and observation space - (CELL 7)
print(test_env.observation_space)
print(test_env.observation_space.sample())
print(test_env.action_space)
print(test_env.action_space.sample())
```

```
Box([[[0]
  [0]
  [0]]

 [[0]
  [0]
  [0]]

 [[0]
  [0]
  [0]]], [[[1]
  [1]
  [1]]

 [[1]
  [1]
  [1]]

 [[1]
  [1]
  [1]]], (3, 3, 1), int32)
[[[0]
  [1]
  [0]]

 [[0]
  [1]
  [1]]

 [[1]
  [1]
  [1]]]
Discrete(9)
2
```

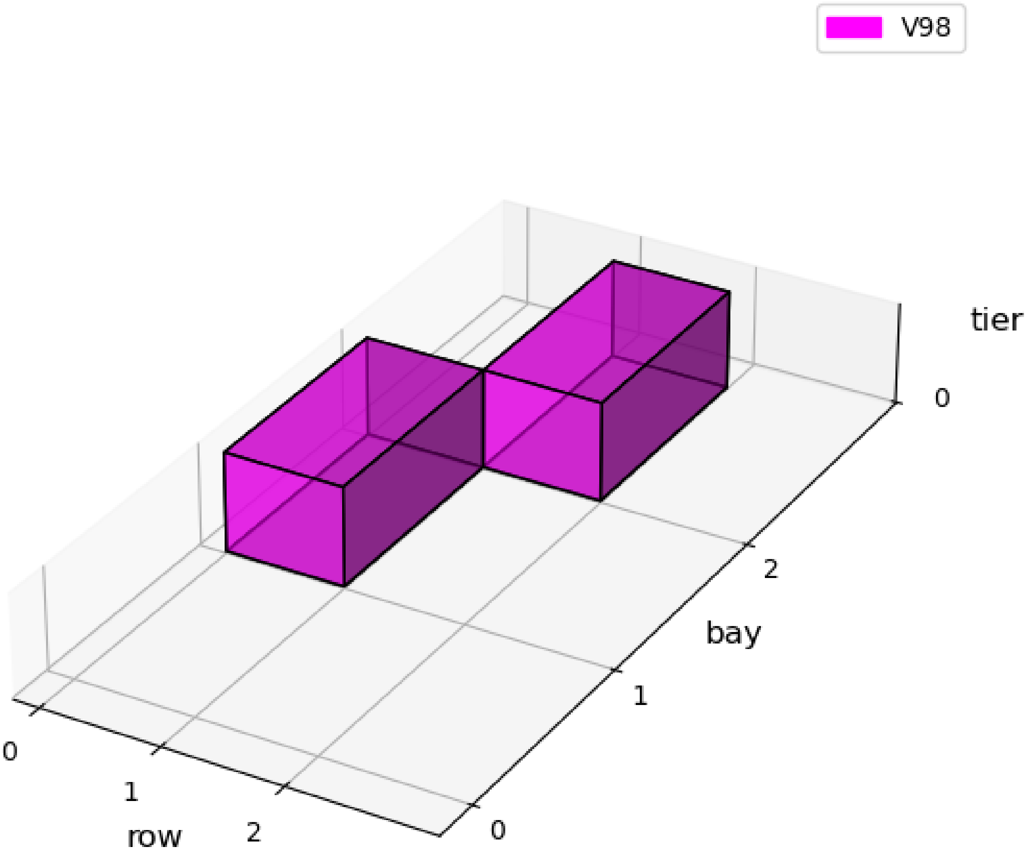
[27]:

```python
# simulate placing of containers and visualizing - (CELL 8)
test_env.reset()
n_steps = 20
for step in range(0, n_steps):
    print(f"step: {step + 1}")
    action = np.random.randint(low=0, high=len(test_env.block.locations) - 1)
    obs, reward, done, info = test_env.step(action)
    print(f"reward: {reward}")
    print(f"score: {info.get('score')}")
    print(f"observation: {obs.tolist()}")
    print(f"illegal moves: {info.get('illegal moves')}")
    print(f"done: {done}\n")
    visualizer.render(test_env.block)
    if done:
        break
```

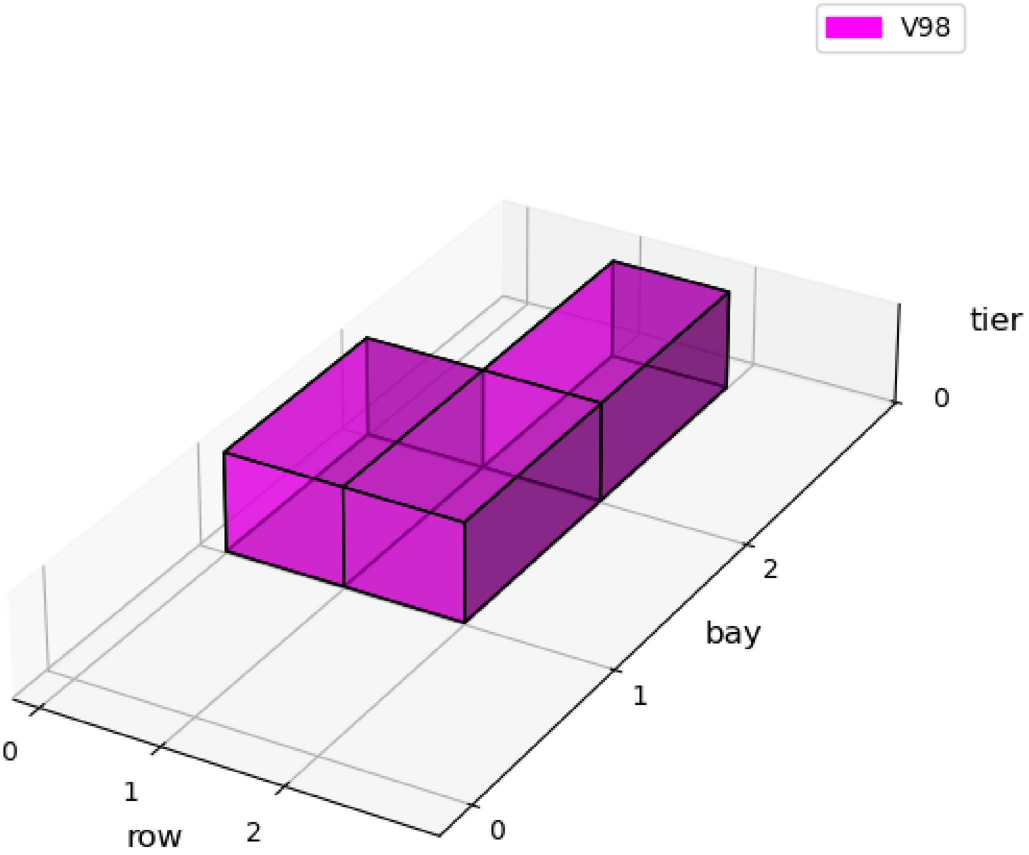
```
step: 1
reward: 20
score: 20
observation: [[[0], [0], [0]], [[0], [0], [1]], [[0], [0], [0]]]
illegal moves: 0
done: False


<Figure size 640x480 with 0 Axes>
```



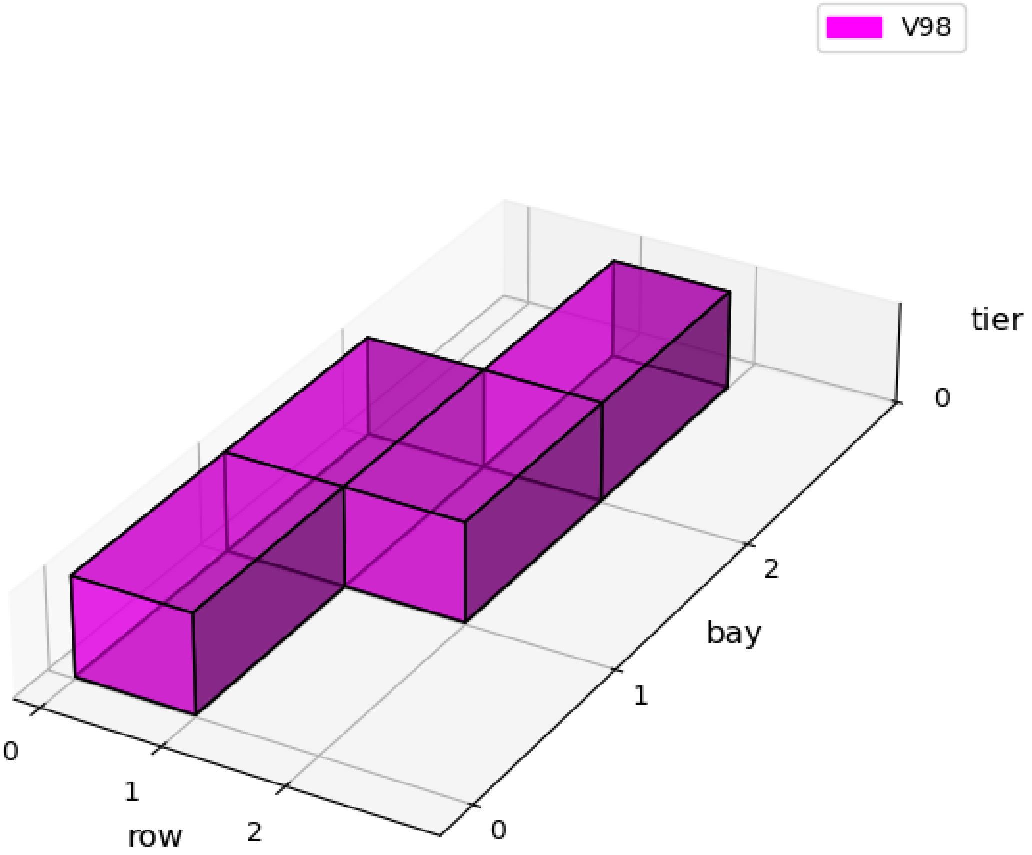
```
step: 2
reward: 20
score: 40
observation: [[[0], [1], [0]], [[0], [0], [1]], [[0], [0], [0]]]
illegal moves: 0
done: False


<Figure size 640x480 with 0 Axes>
```



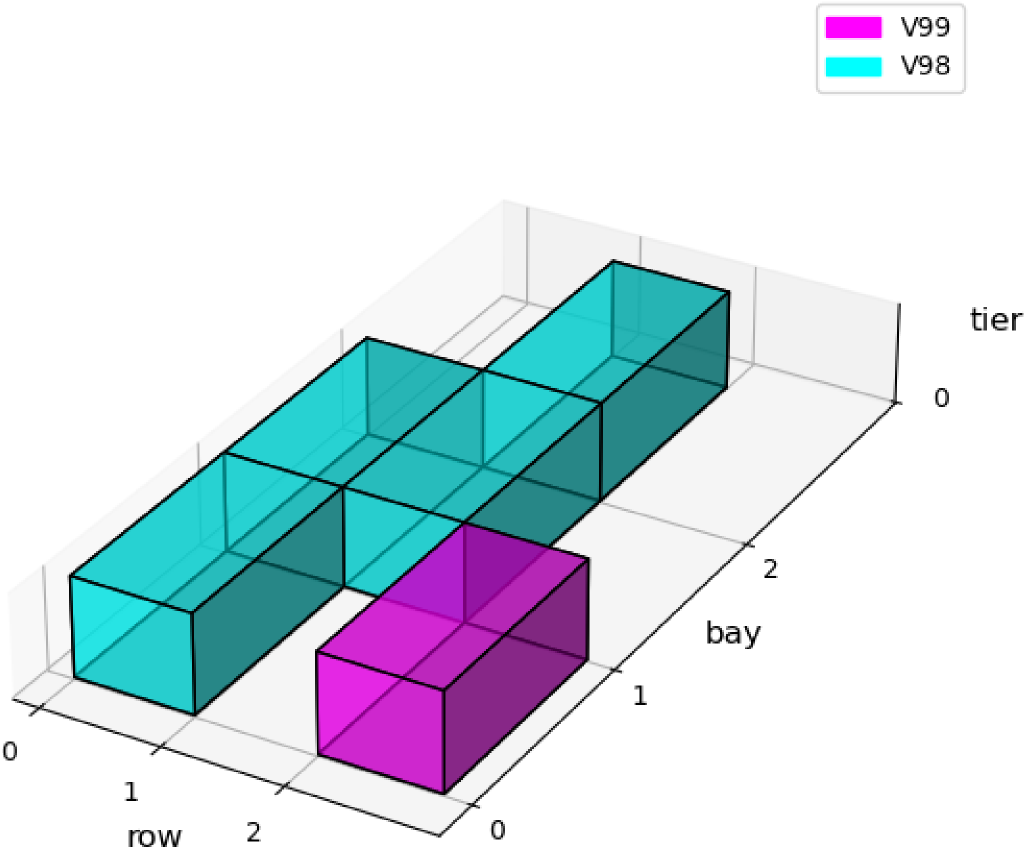
```
step: 3
reward: 30
score: 70
observation: [[[0], [1], [0]], [[0], [1], [1]], [[0], [0], [0]]]
illegal moves: 0
done: False


<Figure size 640x480 with 0 Axes>
```



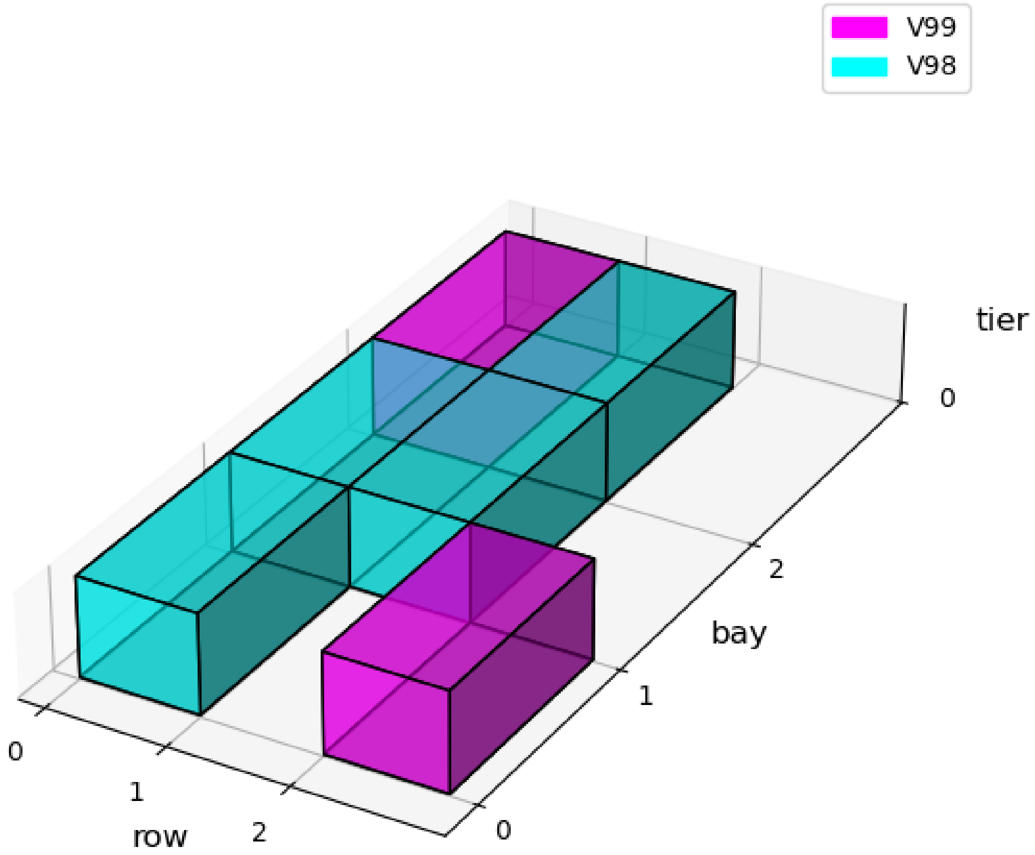
```
step: 4
reward: 20
score: 90
observation: [[[1], [1], [0]], [[0], [1], [1]], [[0], [0], [0]]]
illegal moves: 0
done: False


<Figure size 640x480 with 0 Axes>
```



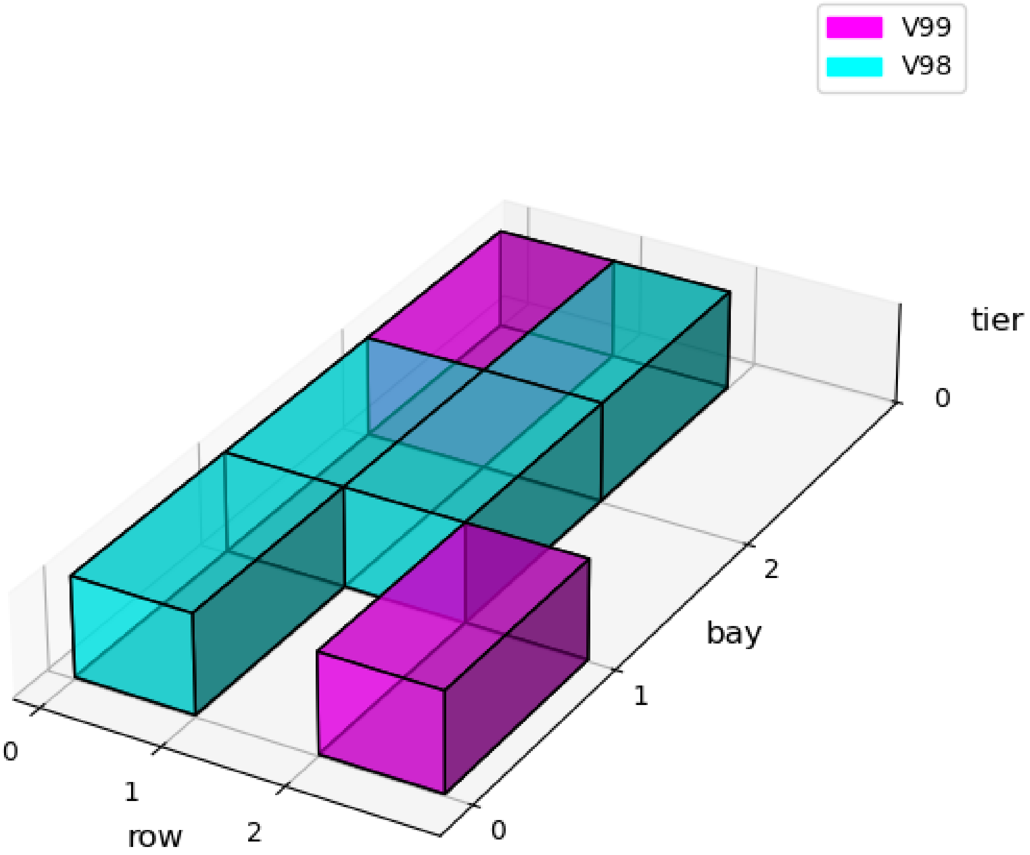
```
step: 5
reward: 20
score: 110
observation: [[[1], [1], [0]], [[0], [1], [1]], [[1], [0], [0]]]
illegal moves: 0
done: False


<Figure size 640x480 with 0 Axes>
```



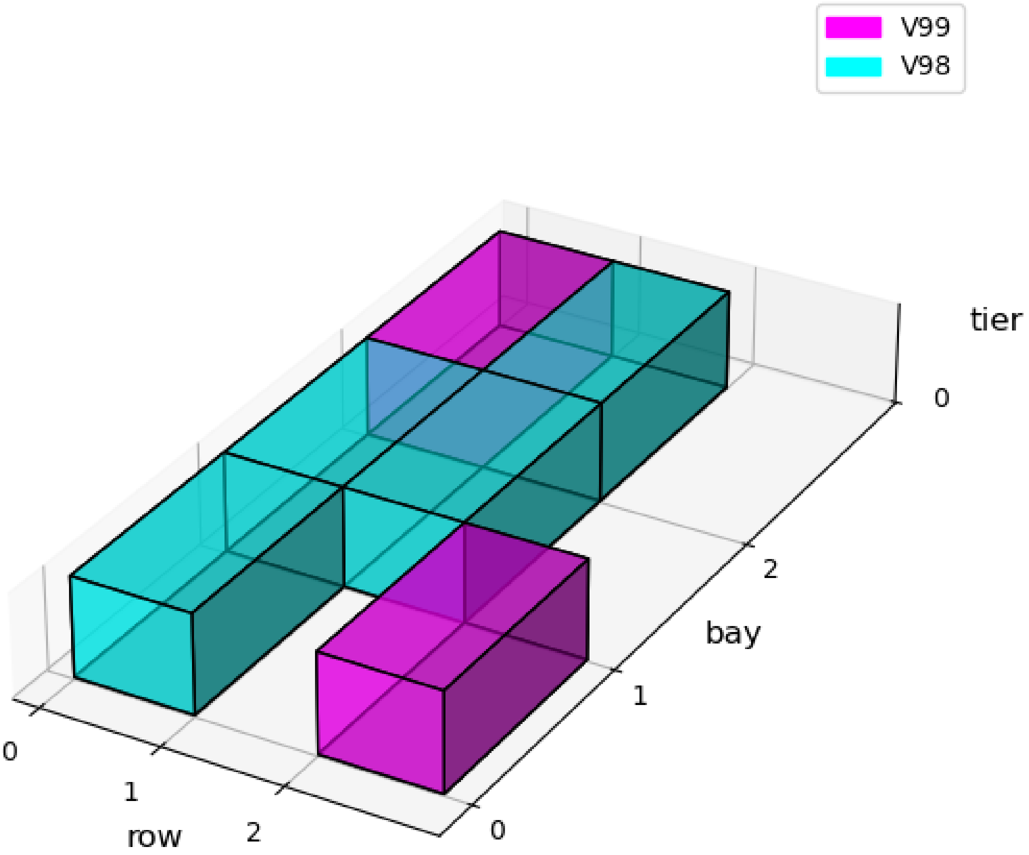
```
step: 6
reward: 10
score: 120
observation: [[[1], [1], [1]], [[0], [1], [1]], [[1], [0], [0]]]
illegal moves: 0
done: False


<Figure size 640x480 with 0 Axes>
```



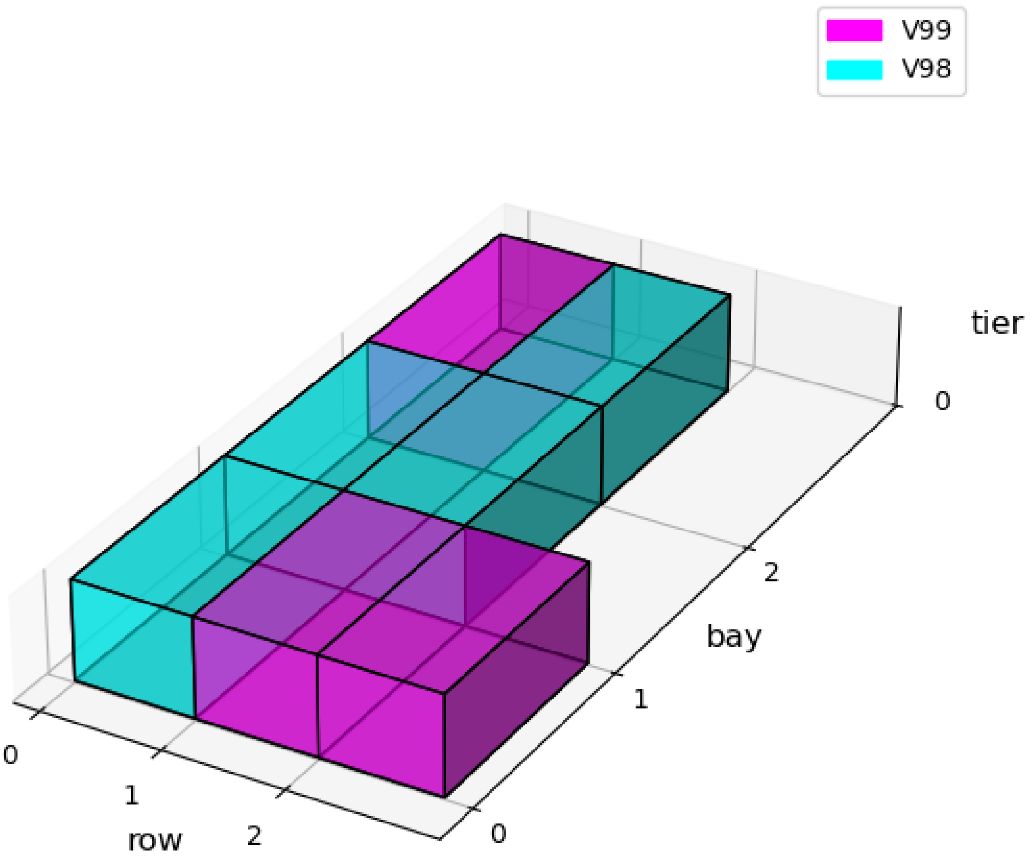
```
step: 7
reward: -20
score: 100
observation: [[[1], [1], [1]], [[0], [1], [1]], [[1], [0], [0]]]
illegal moves: 1
done: False


<Figure size 640x480 with 0 Axes>
```



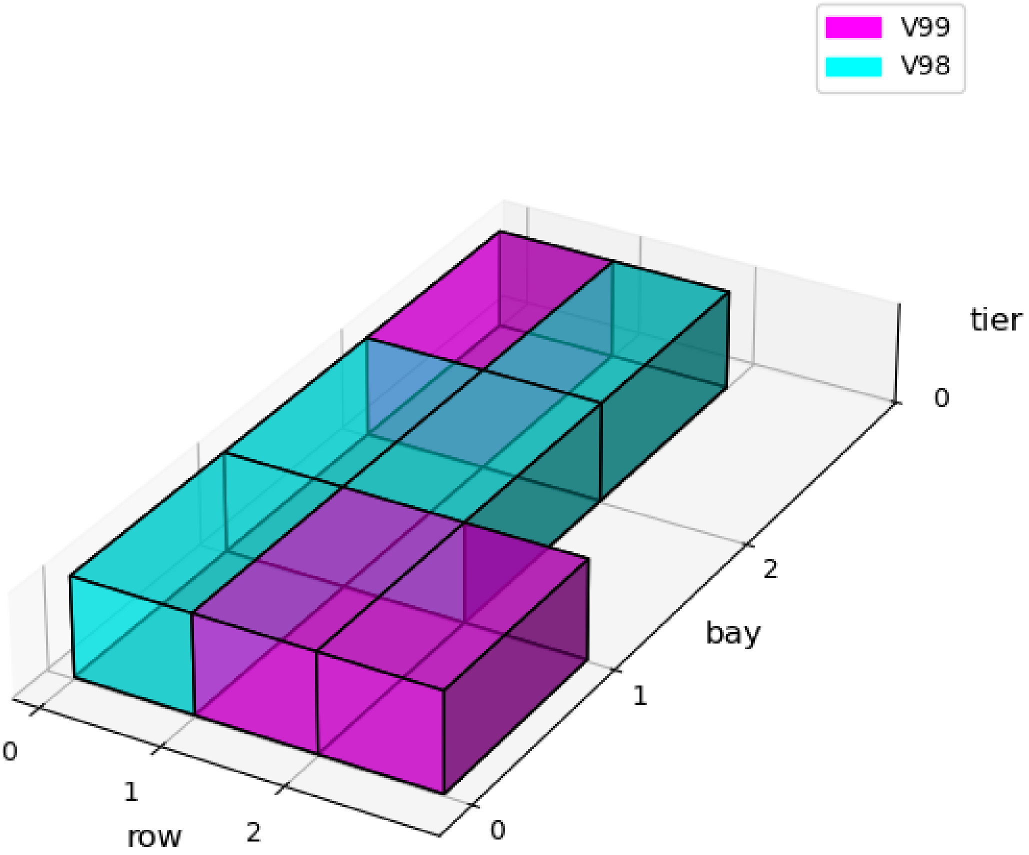
```
step: 8
reward: -20
score: 80
observation: [[[1], [1], [1]], [[0], [1], [1]], [[1], [0], [0]]]
illegal moves: 2
done: False


<Figure size 640x480 with 0 Axes>
```



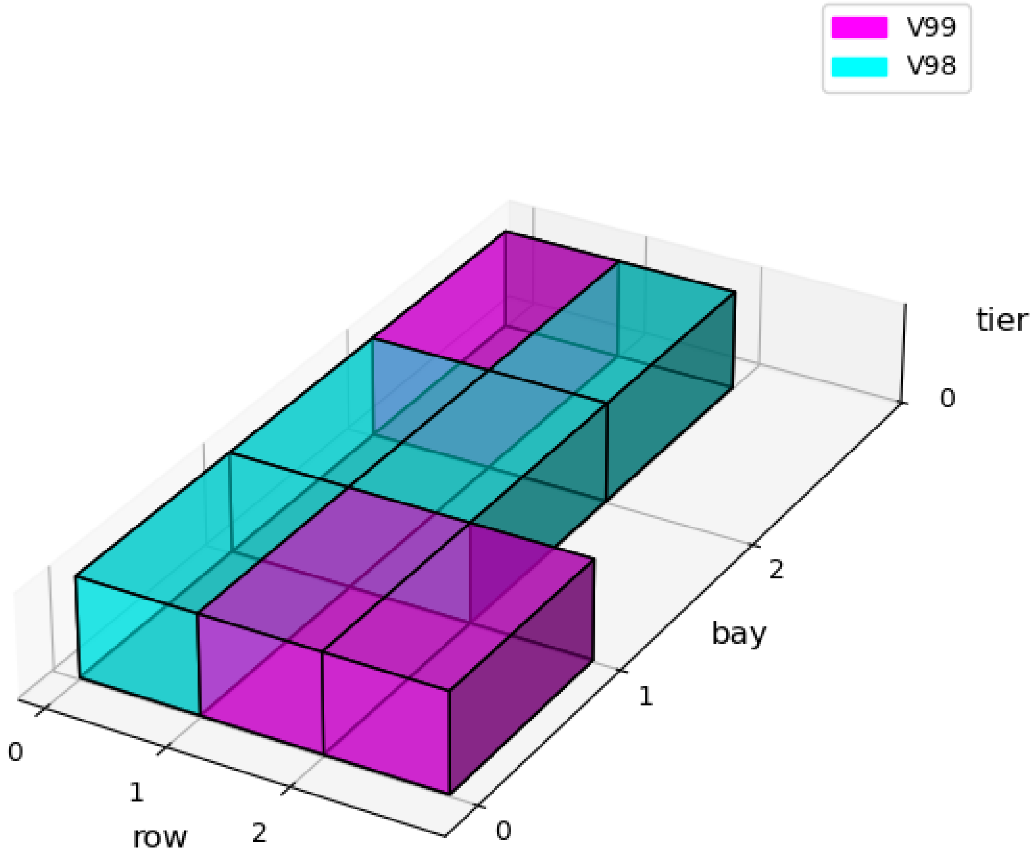
```
step: 9
reward: 20
score: 100
observation: [[[1], [1], [1]], [[1], [1], [1]], [[1], [0], [0]]]
illegal moves: 2
done: False


<Figure size 640x480 with 0 Axes>
```



```
step: 10
reward: -20
score: 80
observation: [[[1], [1], [1]], [[1], [1], [1]], [[1], [0], [0]]]
illegal moves: 3
done: False


<Figure size 640x480 with 0 Axes>
```



```
step: 11
reward: -20
score: 60
observation: [[[1], [1], [1]], [[1], [1], [1]], [[1], [0], [0]]]
illegal moves: 4
done: True


<Figure size 640x480 with 0 Axes>
```



[11]:

```
# create directories to save trained models and logs in - (CELL 9)
models_dir_ppo = "models/PPO"
models_dir_a2c = "models/A2C"
logdir = "logs"

os.makedirs(models_dir_ppo, exist_ok=True)
os.makedirs(models_dir_a2c, exist_ok=True)
os.makedirs(logdir, exist_ok=True)
```

[50]:

```
# wrap environment to vectorized environments - (CELL 10)
n_envs = 1
env = CustomEnv()
env = make_vec_env(lambda: env, n_envs=n_envs)
```

[52]:

```
# create and train PPO model with different hyperparameter values - (CELL 11)
ppo_timesteps = 500000
```

[39]:

```
# TRAINING STARTS HERE - (CELL 12)
```

[1]:

```
# hyperparameter tuning, use logging and tensorboard to see how different␣
 ↪values performed - (CELL 13)
# learning rate
ppo_learning_rates = [0.03, 0.003, 0.0003, 0.00003, 0.000003]
for learning_rate in ppo_learning_rates:
    model_ppo = PPO('MlpPolicy', env, learning_rate=learning_rate,␣
 ↪batch_size=32, verbose=1, tensorboard_log=logdir)
    model_ppo.learn(total_timesteps=ppo_timesteps, reset_num_timesteps=False, \
    tb_log_name=f"PPO-learning_rate={learning_rate}")
    model_ppo.save(f"{models_dir_ppo}/PPO-learning_rate={format(learning_rate,␣
 ↪'f')}")

# gamma
ppo_gammas = [0.90, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99]
for gamma in ppo_gammas:
    model_ppo = PPO('MlpPolicy', env, gamma=gamma, batch_size=32, verbose=1,␣
 ↪tensorboard_log=logdir)
    model_ppo.learn(total_timesteps=ppo_timesteps, reset_num_timesteps=False,␣
 ↪tb_log_name=f"PPO-gamma={gamma}")
    model_ppo.save(f"{models_dir_ppo}/PPO-gamma={gamma}")

# gae_lambda
ppo_gae_lambdas = [0.80, 0.85, 0.90, 0.95]
for gae_lambda in ppo_gae_lambdas:
    model_ppo = PPO('MlpPolicy', env, gae_lambda=gae_lambda, batch_size=32,␣
 ↪verbose=1, tensorboard_log=logdir)
    model_ppo.learn(total_timesteps=ppo_timesteps, reset_num_timesteps=False,␣
 ↪tb_log_name=f"PPO-gae_lambda={gae_lambda}")
    model_ppo.save(f"{models_dir_ppo}/PPO-gae_lambda={gae_lambda}")

# ent_coef
ppo_ent_coefs = [0.0, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001]
for ent_coef in ppo_ent_coefs:
    model_ppo = PPO('MlpPolicy', env, ent_coef=ent_coef, batch_size=32,␣
 ↪verbose=1, tensorboard_log=logdir)
    model_ppo.learn(total_timesteps=ppo_timesteps, reset_num_timesteps=False,␣
 ↪tb_log_name=f"PPO-ent_coef={ent_coef}")
    model_ppo.save(f"{models_dir_ppo}/PPO-ent_coef={format(ent_coef, 'f')}")

# vf_coef
ppo_vf_coefs = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
```

```
for vf_coef in ppo_vf_coefs:
    model_ppo = PPO('MlpPolicy', env, vf_coef=vf_coef, batch_size=32,␣
 ↪verbose=1, tensorboard_log=logdir)
    model_ppo.learn(total_timesteps=ppo_timesteps, reset_num_timesteps=False,␣
 ↪tb_log_name=f"PPO-vf_coef={vf_coef}")
    model_ppo.save(f"{models_dir_ppo}/PPO-vf_coef={vf_coef}")
```

[13]:

```
# amount of environments running parallel - (CELL 14)
n_final_envs = 1
final_env = CustomEnv()
final_env = make_vec_env(lambda: final_env, n_envs=n_final_envs)

# hyperparameters
final_policy = 'MlpPolicy'
final_batch_size = 32
final_learning_rate = 0.00003
final_gamma = 0.94
final_gae_lambda = 0.95
final_ent_coef = 0.1
final_vf_coef = 0.2

final_ppo = PPO(policy=final_policy, env=final_env,␣
 ↪learning_rate=final_learning_rate, batch_size=final_batch_size, \
                gamma=final_gamma, gae_lambda=final_gae_lambda,␣
 ↪ent_coef=final_ent_coef, vf_coef=final_vf_coef, \
                verbose=1, tensorboard_log=logdir)
```

```
Using cpu device
```

[14]:

```
# train final model with best performing hyperparameters - (CELL 15)
final_timesteps = 3000000
final_ppo.learn(total_timesteps=final_timesteps, reset_num_timesteps=False,␣
 ↪tb_log_name="PPO-final")
final_ppo.save(f"{models_dir_ppo}/PPO-final")
```

```
Logging to logs/PPO-final_0
---------------------------------
| rollout/           |          |
|    ep_len_mean     | 9.87     |
|    ep_rew_mean     | 45.2     |
| time/              |          |
|    fps             | 2210     |
|    iterations      | 1        |
|    time_elapsed    | 0        |
|    total_timesteps | 2048     |
---------------------------------
-------------------------------------------
```

```
| rollout/                 |               |
|    ep_len_mean           | 9.9           |
|    ep_rew_mean           | 48.7          |
| time/                    |               |
|    fps                   | 1311          |
|    iterations            | 2             |
|    time_elapsed          | 3             |
|    total_timesteps       | 4096          |
| train/                   |               |
|    approx_kl             | 0.00035778174 |
|    clip_fraction         | 0             |
|    clip_range            | 0.2           |
|    entropy_loss          | -2.2          |
|    explained_variance    | 0.00587       |
|    learning_rate         | 3e-05         |
|    loss                  | 227           |
|    n_updates             | 10            |
|    policy_gradient_loss  | -0.00407      |
|    value_loss            | 1.18e+03      |
--------------------------------------------
--------------------------------------------
| rollout/                 |               |
|    ep_len_mean           | 9.71          |
|    ep_rew_mean           | 47.3          |
| time/                    |               |
|    fps                   | 1155          |
|    iterations            | 3             |
|    time_elapsed          | 5             |
|    total_timesteps       | 6144          |
| train/                   |               |
|    approx_kl             | 0.00033839108 |
|    clip_fraction         | 0             |
|    clip_range            | 0.2           |
|    entropy_loss          | -2.2          |
|    explained_variance    | 0.0315        |
|    learning_rate         | 3e-05         |
|    loss                  | 304           |
|    n_updates             | 20            |
|    policy_gradient_loss  | -0.00405      |
|    value_loss            | 1.15e+03      |
--------------------------------------------
-------------------------------------------
| rollout/                 |              |
|    ep_len_mean           | 10           |
|    ep_rew_mean           | 52.6         |
| time/                    |              |
|    fps                   | 1090         |
|    iterations            | 4            |
|    entropy_loss         | -0.474      |
|    explained_variance   | 0.929       |
|    learning_rate        | 3e-05       |
|    loss                 | 32.2        |
|    n_updates            | 14610       |
|    policy_gradient_loss | -0.0302     |
|    value_loss           | 174         |
-----------------------------------------
------------------------------------------
| rollout/                |              |
|    ep_len_mean          | 8.11         |
|    ep_rew_mean          | 198          |
| time/                   |              |
|    fps                  | 940          |
|    iterations           | 1463         |
|    time_elapsed         | 3187         |
|    total_timesteps      | 2996224      |
| train/                  |              |
|    approx_kl            | 0.0018498615 |
|    clip_fraction        | 0.0228       |
|    clip_range           | 0.2          |
|    entropy_loss         | -0.435       |
|    explained_variance   | 0.967        |
|    learning_rate        | 3e-05        |
|    loss                 | 11.9         |
|    n_updates            | 14620        |
|    policy_gradient_loss | -0.0153      |
|    value_loss           | 76.4         |
------------------------------------------
------------------------------------------
| rollout/                |              |
|    ep_len_mean          | 8.07         |
|    ep_rew_mean          | 199          |
| time/                   |              |
|    fps                  | 940          |
|    iterations           | 1464         |
|    time_elapsed         | 3189         |
|    total_timesteps      | 2998272      |
| train/                  |              |
|    approx_kl            | 0.0026460278 |
|    clip_fraction        | 0.0198       |
|    clip_range           | 0.2          |
|    entropy_loss         | -0.428       |
|    explained_variance   | 0.986        |
|    learning_rate        | 3e-05        |
|    loss                 | 1.93         |
|    n_updates            | 14630        |
|    policy_gradient_loss | -0.0105      |
|    value_loss           | 29.9         |
------------------------------------------
------------------------------------------
| rollout/                |              |
|    ep_len_mean          | 8.16         |
|    ep_rew_mean          | 197          |
| time/                   |              |
|    fps                  | 940          |
|    iterations           | 1465         |
|    time_elapsed         | 3191         |
|    total_timesteps      | 3000320      |
| train/                  |              |
|    approx_kl            | 0.0023121561 |
|    clip_fraction        | 0.0211       |
|    clip_range           | 0.2          |
|    entropy_loss         | -0.417       |
|    explained_variance   | 0.987        |
|    learning_rate        | 3e-05        |
|    loss                 | 3.81         |
|    n_updates            | 14640        |
|    policy_gradient_loss | -0.0108      |
|    value_loss           | 28.7         |
------------------------------------------
```

[19]:

```
# evaluate the model - (CELL 16)
mean_reward, std_reward = evaluate_policy(final_ppo, final_ppo.get_env(),␣
 ↪n_eval_episodes=100)
print(f"mean reward: {mean_reward}")
print(f"std reward: {std_reward}")
```

```
mean reward: 200.0
std reward: 0.0
```

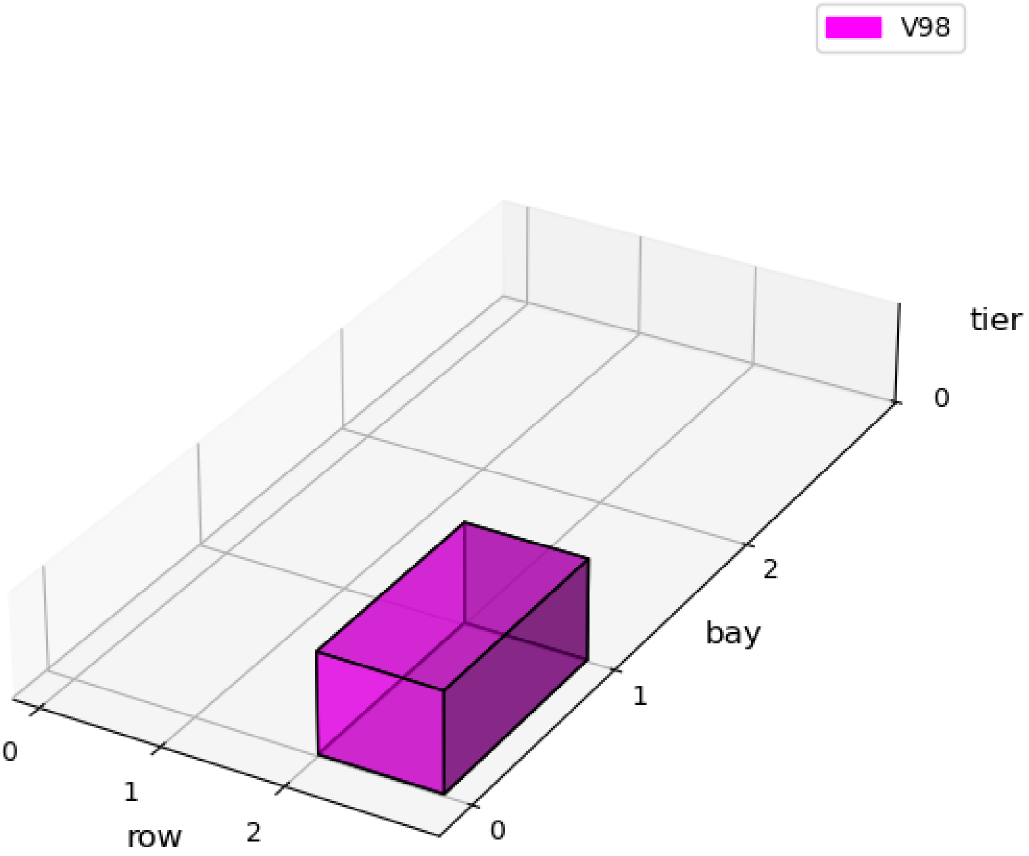
[21]:

```
# simulate a single episode - (CELL 17)
eval_env = CustomEnv()
obs = eval_env.reset()
```

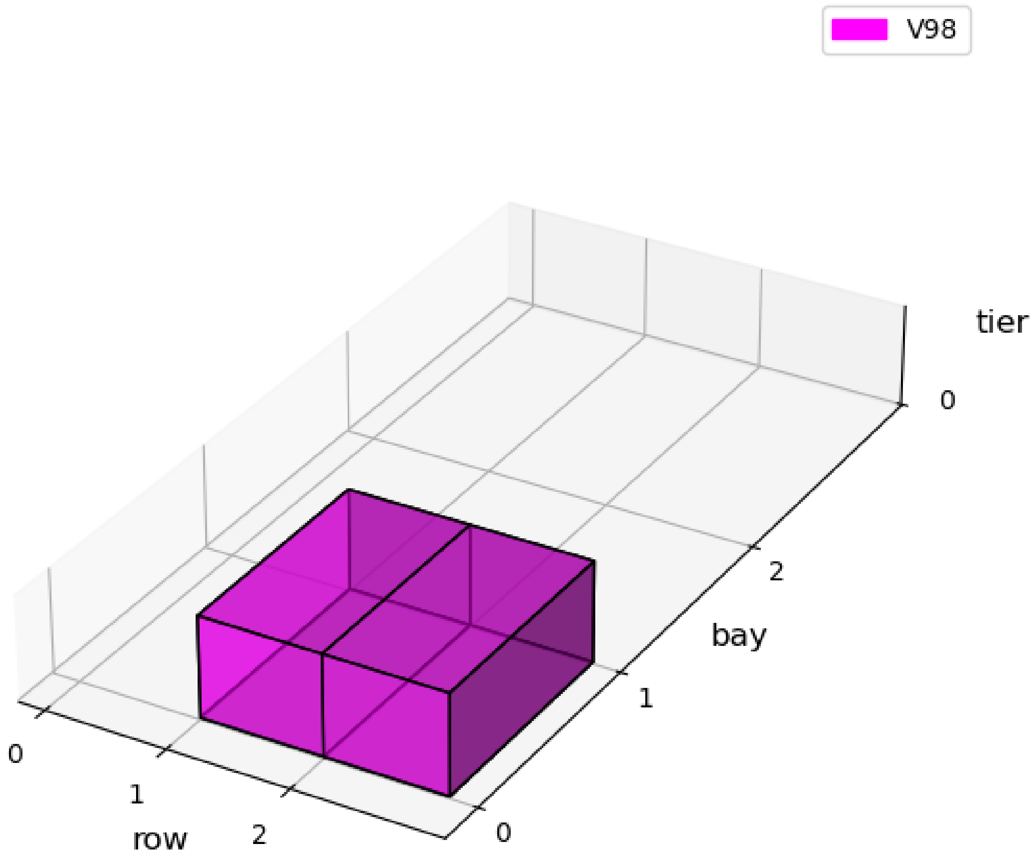
[31]:

```
# use trained model to make predictions - (CELL 18)
steps_taken = 0
while (True):
    steps_taken += 1
    print(f"step: {steps_taken}")
    action, _state = final_ppo.predict(obs)
    obs, reward, done, info = eval_env.step(action)
    print(f"reward: {reward}")
    print(f"score: {info.get('score')}")
    print(f"observation: {obs.tolist()}")
    print(f"illegal moves: {info.get('illegal moves')}")
    print(f"done: {done}\n")
    visualizer.render(eval_env.block)
    if done:
        break
obs = eval_env.reset()
```

```
step: 1
reward: 20
score: 20
observation: [[[0], [0], [0]], [[0], [0], [0]], [[1], [0], [0]]]
illegal moves: 0
done: False


<Figure size 640x480 with 0 Axes>
```



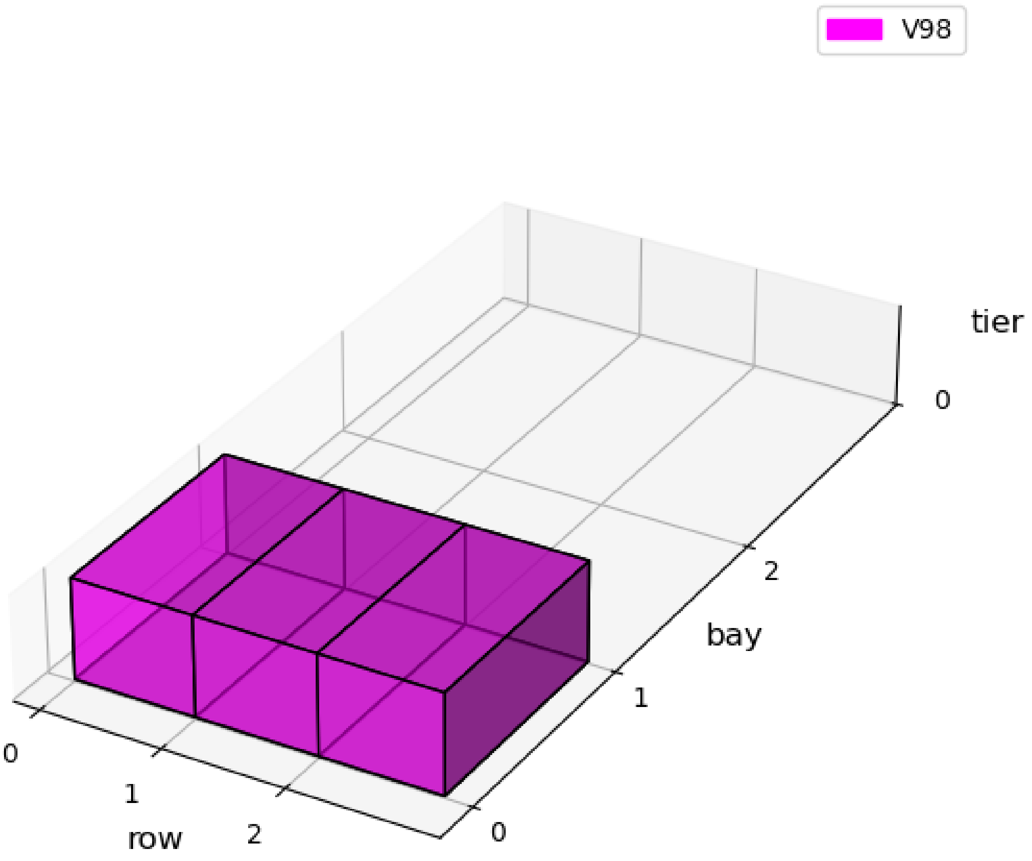
```
step: 2
reward: 30
score: 50
observation: [[[0], [0], [0]], [[1], [0], [0]], [[1], [0], [0]]]
illegal moves: 0
done: False

<Figure size 640x480 with 0 Axes>
```



```
step: 3
reward: 30
score: 80
observation: [[[1], [0], [0]], [[1], [0], [0]], [[1], [0], [0]]]
```

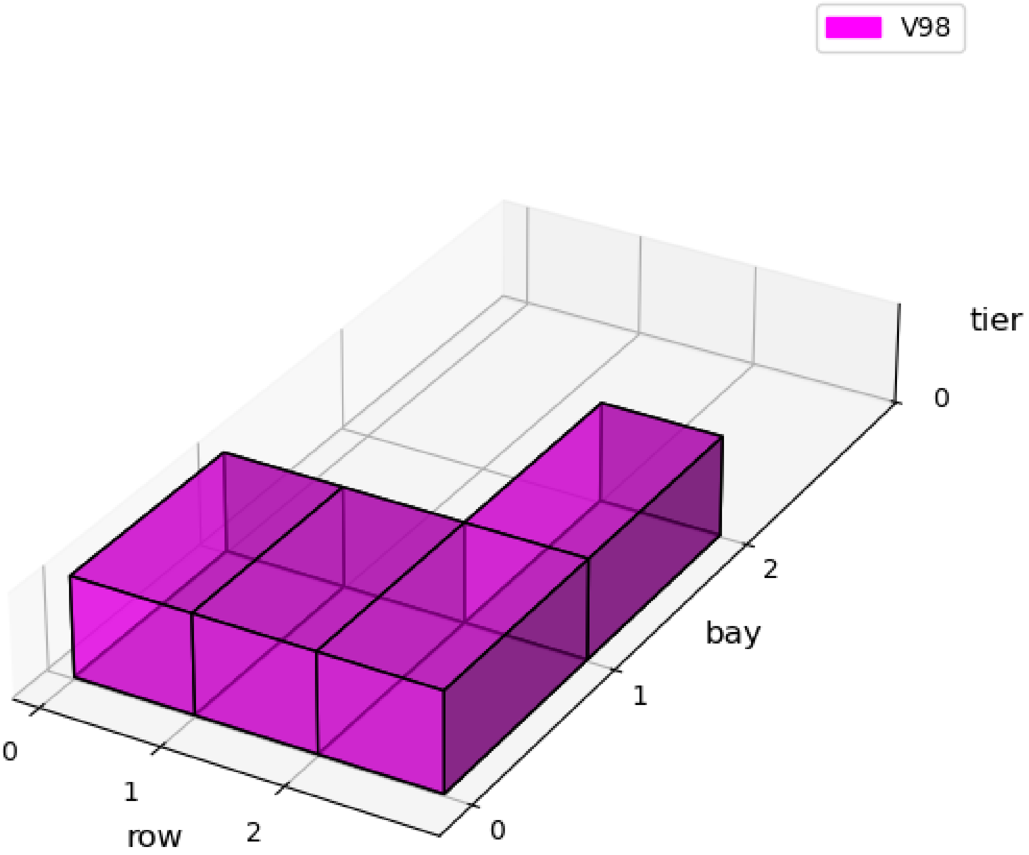
```
illegal moves: 0
done: False
```

```
<Figure size 640x480 with 0 Axes>
```

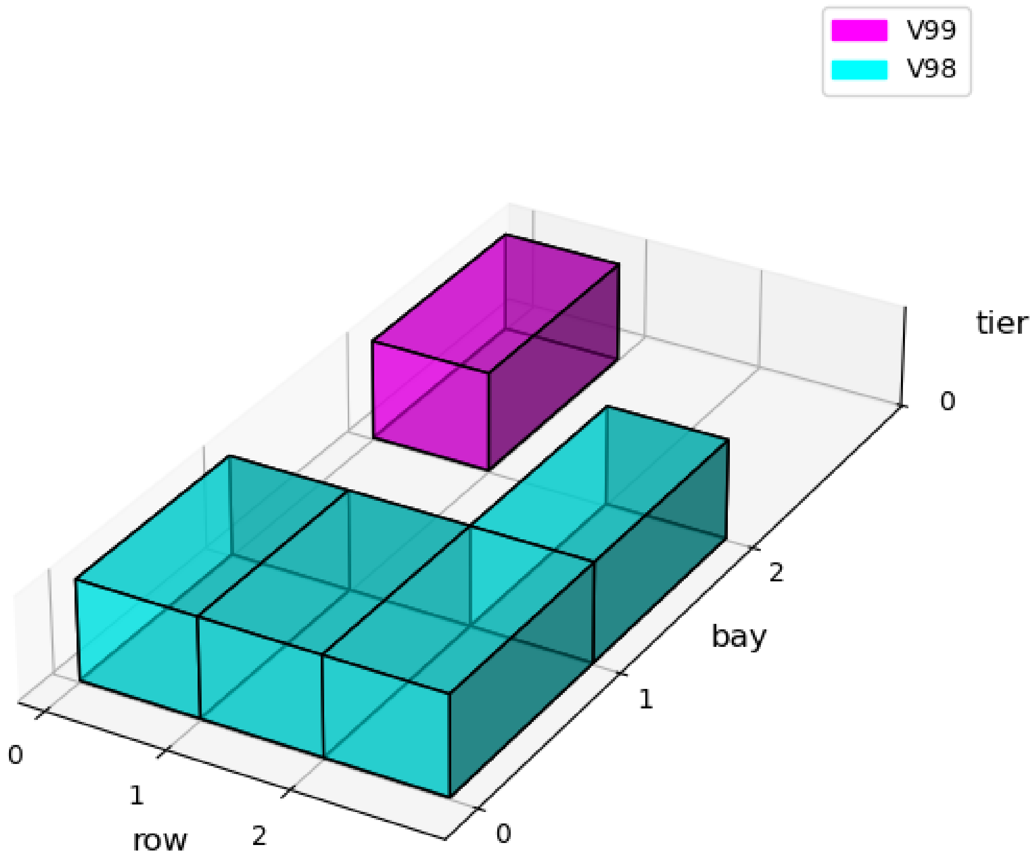


```
step: 4
reward: 20
score: 100
observation: [[[1], [0], [0]], [[1], [0], [0]], [[1], [1], [0]]]
illegal moves: 0
done: False
```

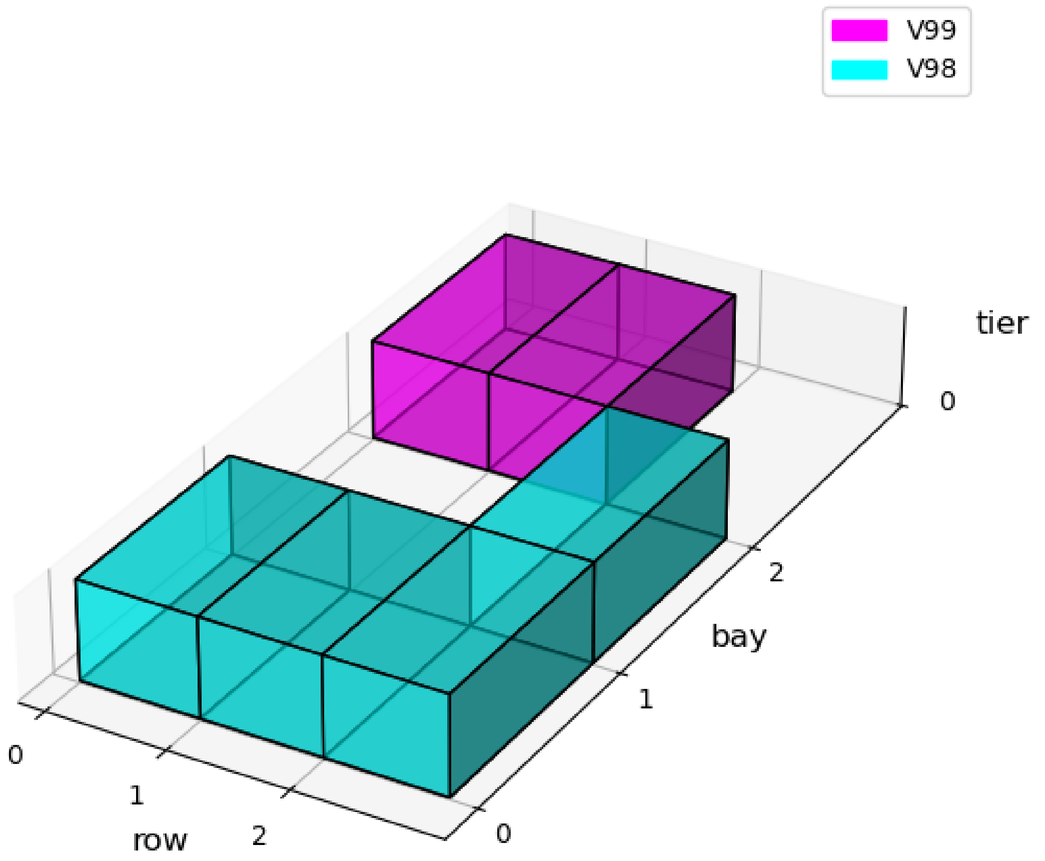
```
<Figure size 640x480 with 0 Axes>
```



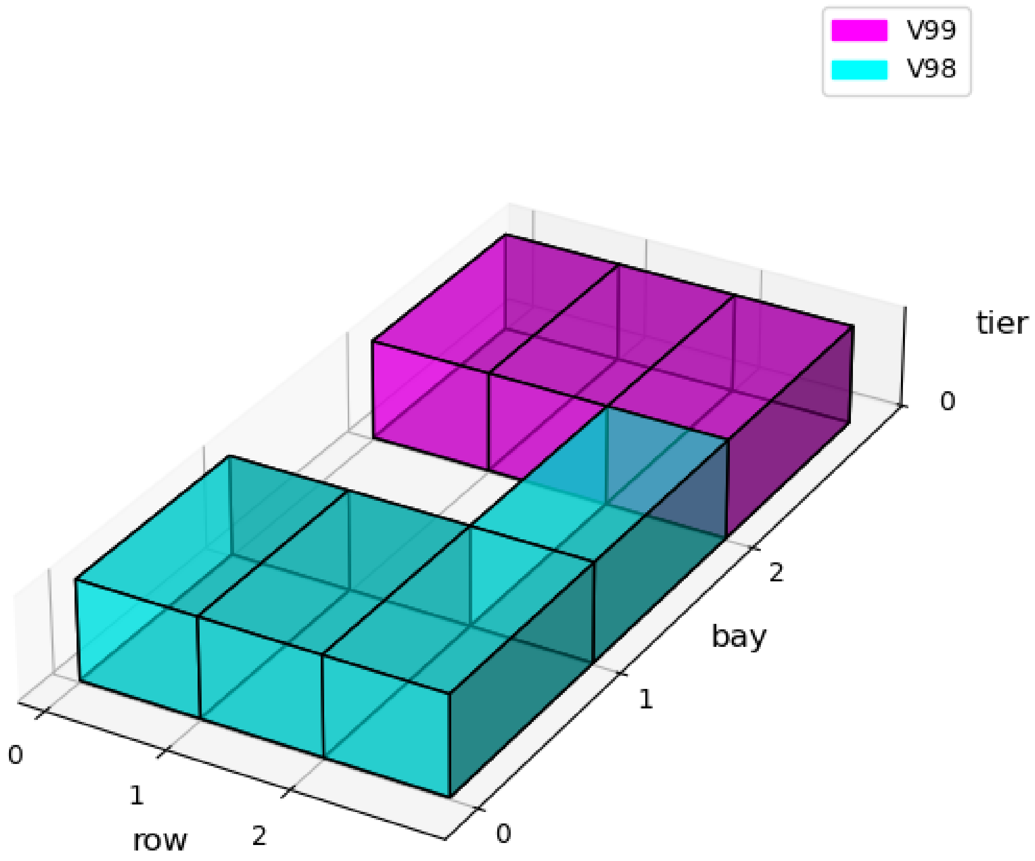
```
step: 5
reward: 20
score: 120
observation: [[[1], [0], [1]], [[1], [0], [0]], [[1], [1], [0]]]
illegal moves: 0
done: False


<Figure size 640x480 with 0 Axes>
```



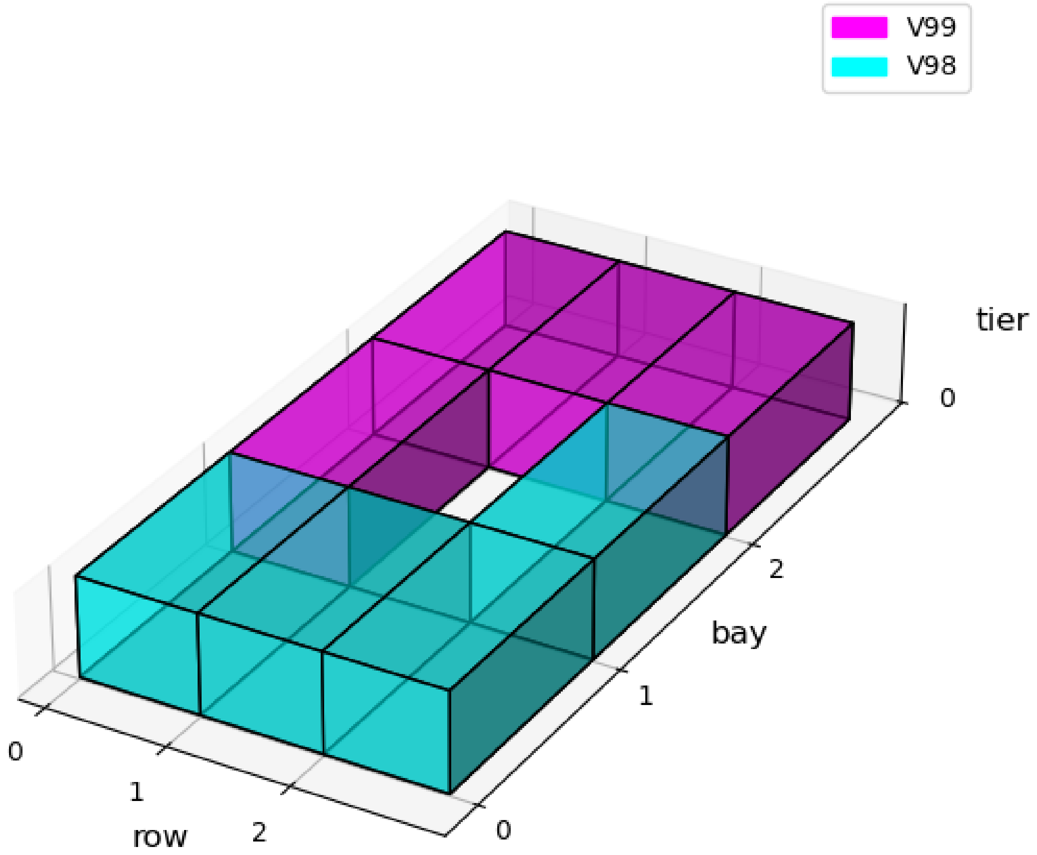
```
step: 6
reward: 30
score: 150
observation: [[[1], [0], [1]], [[1], [0], [1]], [[1], [1], [0]]]
illegal moves: 0
done: False


<Figure size 640x480 with 0 Axes>
```



```
step: 7
reward: 30
score: 180
observation: [[[1], [0], [1]], [[1], [0], [1]], [[1], [1], [1]]]
illegal moves: 0
done: False


<Figure size 640x480 with 0 Axes>
```



```
step: 8
reward: 20
score: 200
observation: [[[1], [1], [1]], [[1], [0], [1]], [[1], [1], [1]]]
illegal moves: 0
done: True


<Figure size 640x480 with 0 Axes>
```



```
[12]: # block with locations - (CELL 19)
      block = Block(3, 3, 1)
      print(block.info())
      for location in block.locations:
          print(location.location_id, location.row, location.bay, location.tier,␣
        ↪location.container is not None)
      print()

      # location in a block
      print(block.locations[0].info())

      # dock which can have vessels
      dock = Dock()
      print(dock.info())
```

```python
# vessel with containers
vessel1 = Vessel(4, 'V9', dock)
vessel2 = Vessel(4, 'V10', dock)
print(vessel1.info())
print(dock.info())

# container in a vessel
print(vessel1.containers[0].info())

# check if location has a container
print(block.locations[0].has_container())
```

```
block id: B2, maximum rows: 3, maximum bays: 3, maximum tiers: 1, location
amount: 9

L10 1 1 1 False
L11 1 2 1 False
L12 1 3 1 False
L13 2 1 1 False
L14 2 2 1 False
L15 2 3 1 False
L16 3 1 1 False
L17 3 2 1 False
L18 3 3 1 False

location id: L10, row: 1, bay: 1, tier: 1, container: None

dock id: D2, vessel amount: 0, container amount: 0

vessel id: V3, maximum containers: 4, container amount: 4, docked at: D2

dock id: D2, vessel amount: 2, container amount: 8

container id: C9, origin vessel id: V3, destination vessel id: V9

True
```