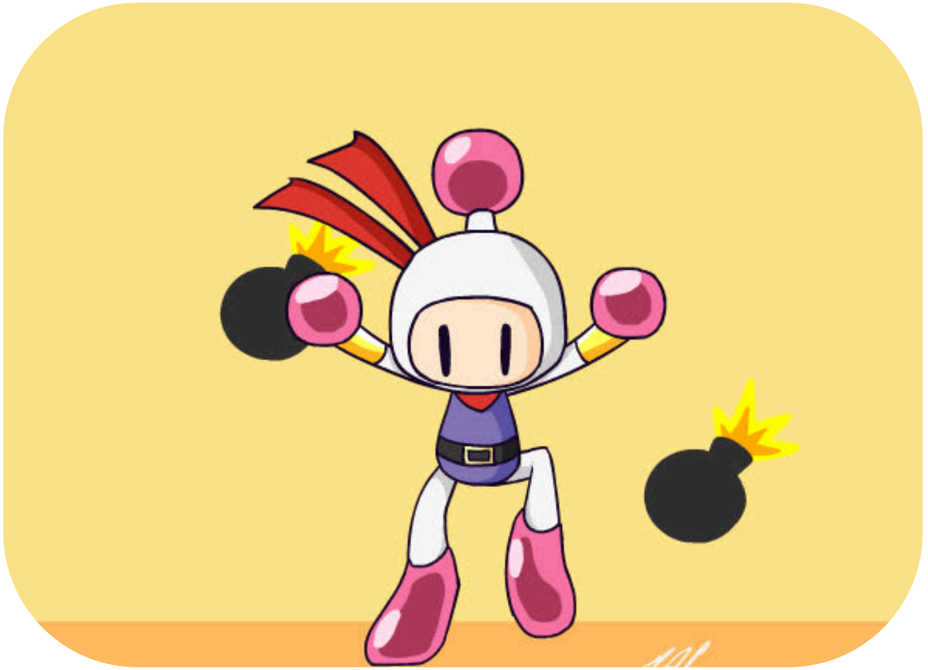
Survey

REINFORCEMENT LEARNING

in BOMBERMAN



CS106 - Artificial Intelligence

22520004 - Tran Nhu Cam Nguyen

22520593 - Nguyen Thanh Hy

22520946 - Le Tin Nghia

Table of CONTENT

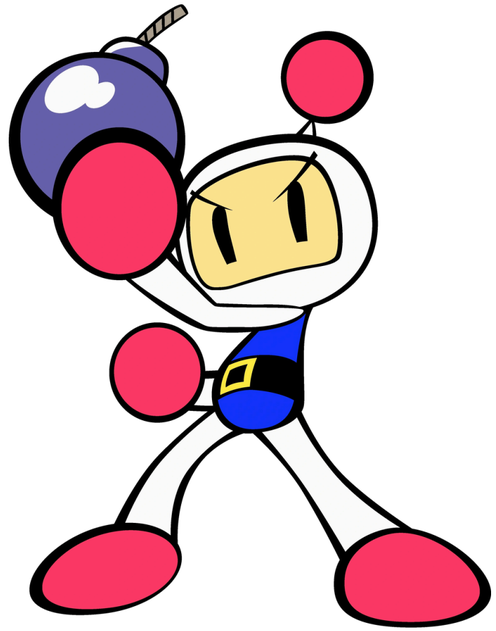
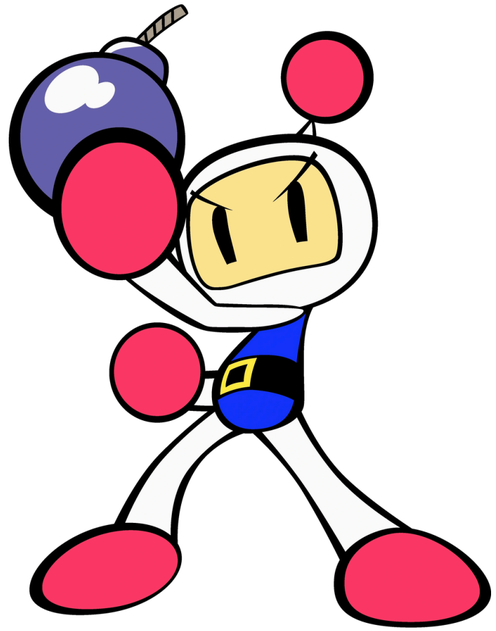
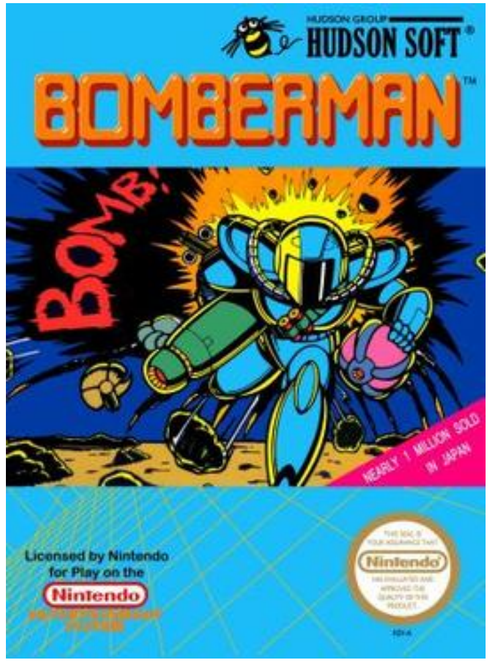


Table of CONTENT



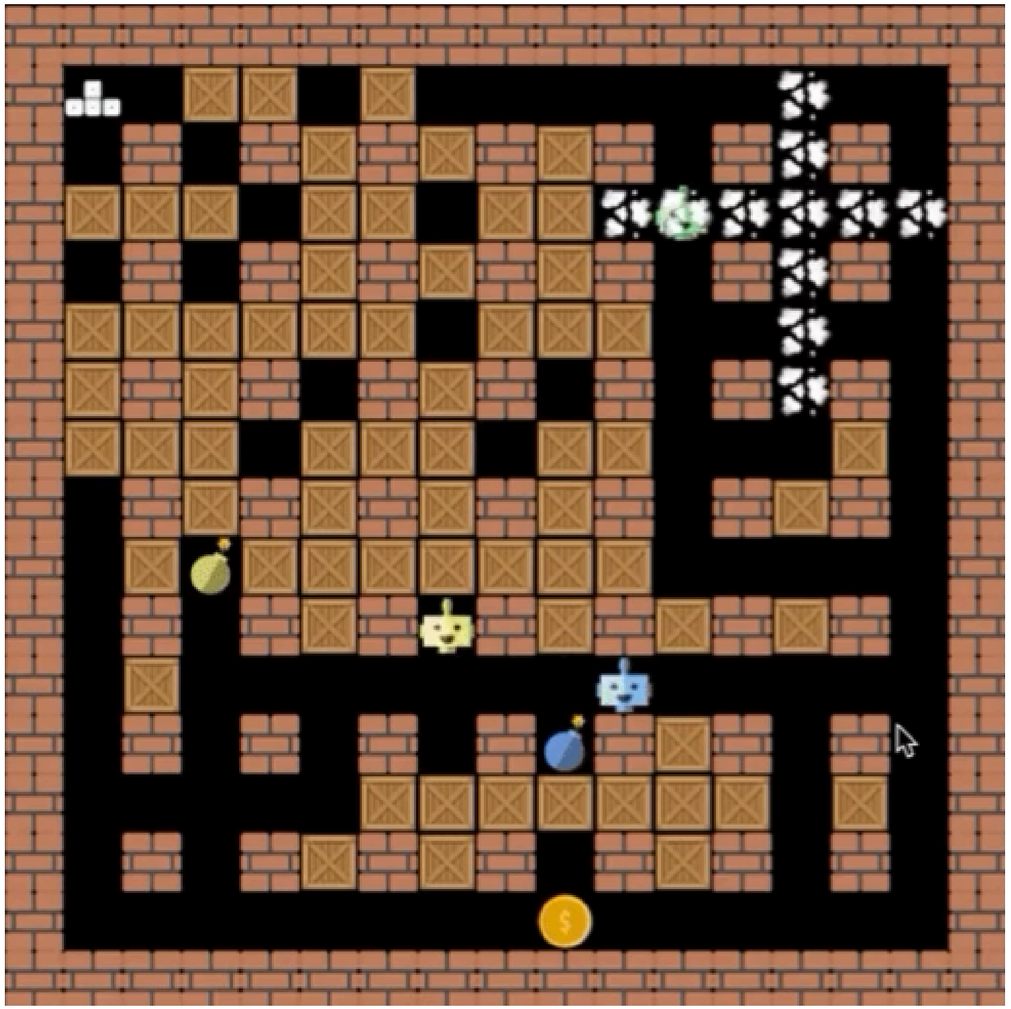
BOMBERMAN GAME

- Video game developed by Hudson Soft
- First released in 1983
- Quickly became an icon and was released on various platforms

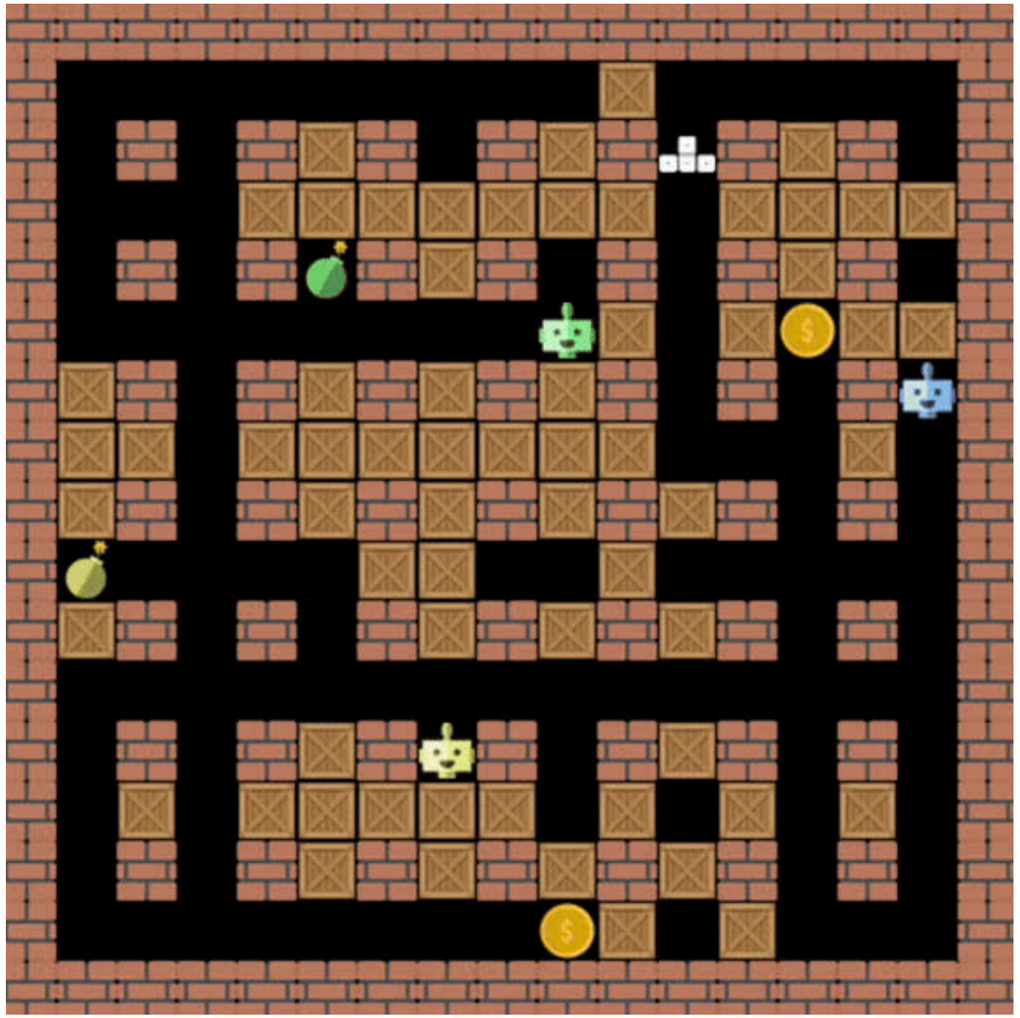


BOMBERMAN GAME

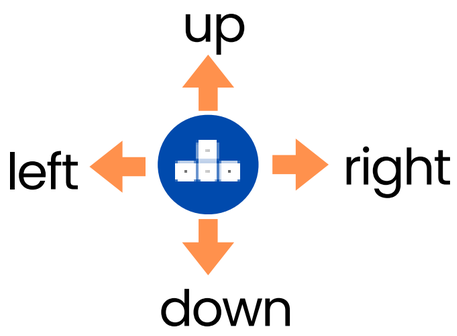
Player control Bomberman, place bombs to destroy crates and enemies, aims to **eliminate all enemies and collect items**



BOMBERMAN GAME



- **AGENT**
  - Bomberman
- **ACTION**
  - Represents a decision made by agent in game



place bomb

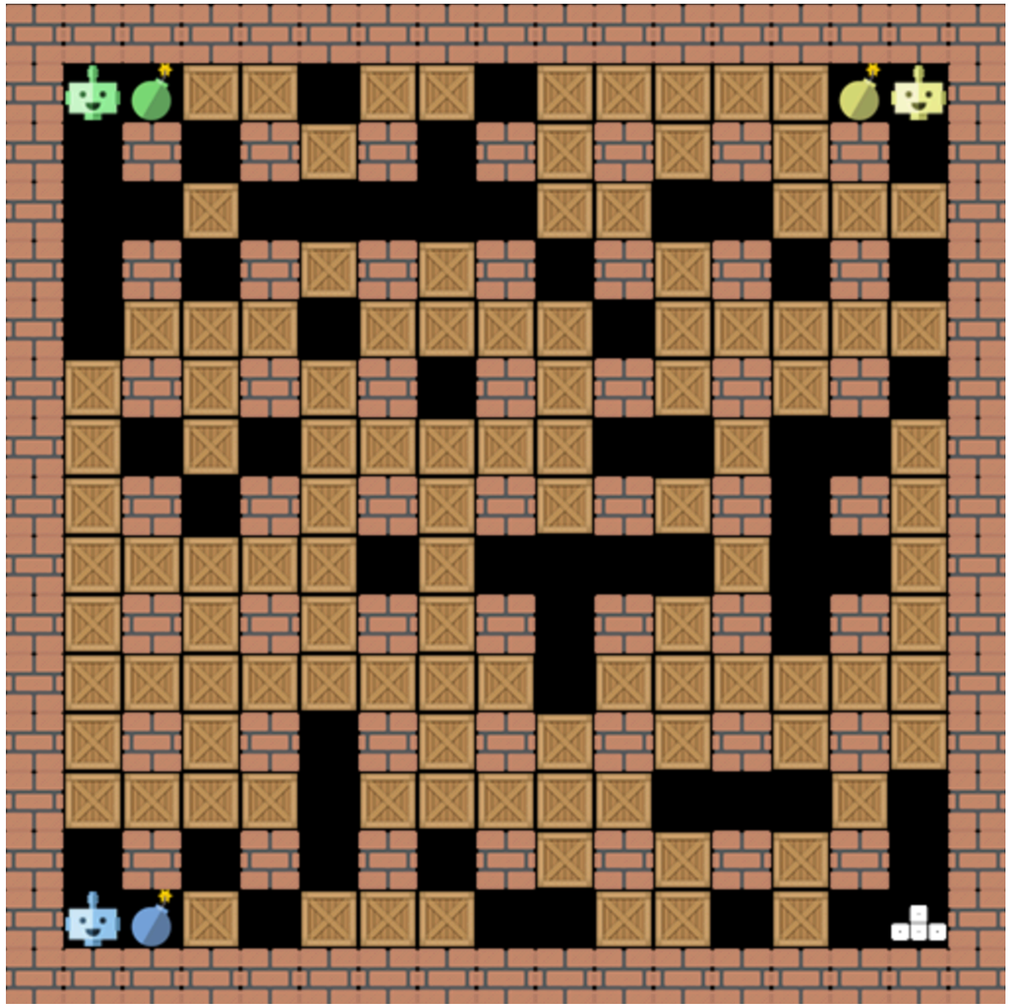
wait

BOMBERMAN GAME

environment.py > GenericWorld

```
30   class GenericWorld:
128      def perform_agent_action(self, agent: Agent, action: str):
129          # Perform the specified action if possible, wait otherwise
130          if action == 'UP' and self.tile_is_free(agent.x, agent.y - 1):
131              agent.y -= 1
132              agent.add_event(e.MOVED_UP)
133          elif action == 'DOWN' and self.tile_is_free(agent.x, agent.y + 1):
134              agent.y += 1
135              agent.add_event(e.MOVED_DOWN)
136          elif action == 'LEFT' and self.tile_is_free(agent.x - 1, agent.y):
137              agent.x -= 1
138              agent.add_event(e.MOVED_LEFT)
139          elif action == 'RIGHT' and self.tile_is_free(agent.x + 1, agent.y):
140              agent.x += 1
141              agent.add_event(e.MOVED_RIGHT)
142          elif action == 'BOMB' and agent.bombs_left:
143              self.logger.info(f'Agent <{agent.name}> drops bomb at {(agent.x, agent.y)}')
144              self.bombs.append(Bomb((agent.x, agent.y), agent, s.BOMB_TIMER, s.BOMB_POWER, agent.bomb_sprite))
145              agent.bombs_left = False
146              agent.add_event(e.BOMB_DROPPED)
147          elif action == 'WAIT':
148              agent.add_event(e.WAITED)
149          else:
150              agent.add_event(e.INVALID_ACTION)
```

BOMBERMAN GAME



- **ENVIRONMENT**
  - Game board: 17x17 grid of cells (walls, empty cells, crates)

wall impassable and unbreakable

free can place bomb on it

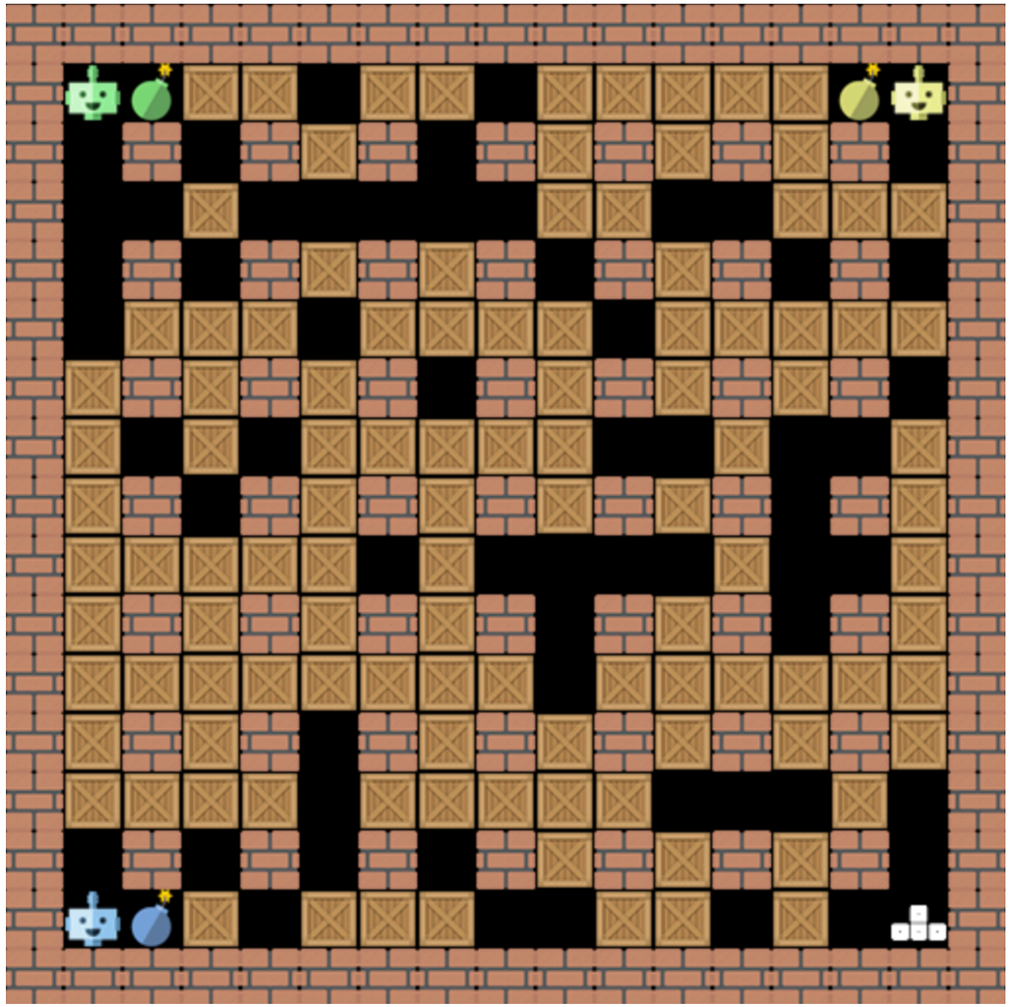
crate can be destroyed by bomb

BOMBERMAN GAME

environment.py > BombeRLeWorld > build_arena

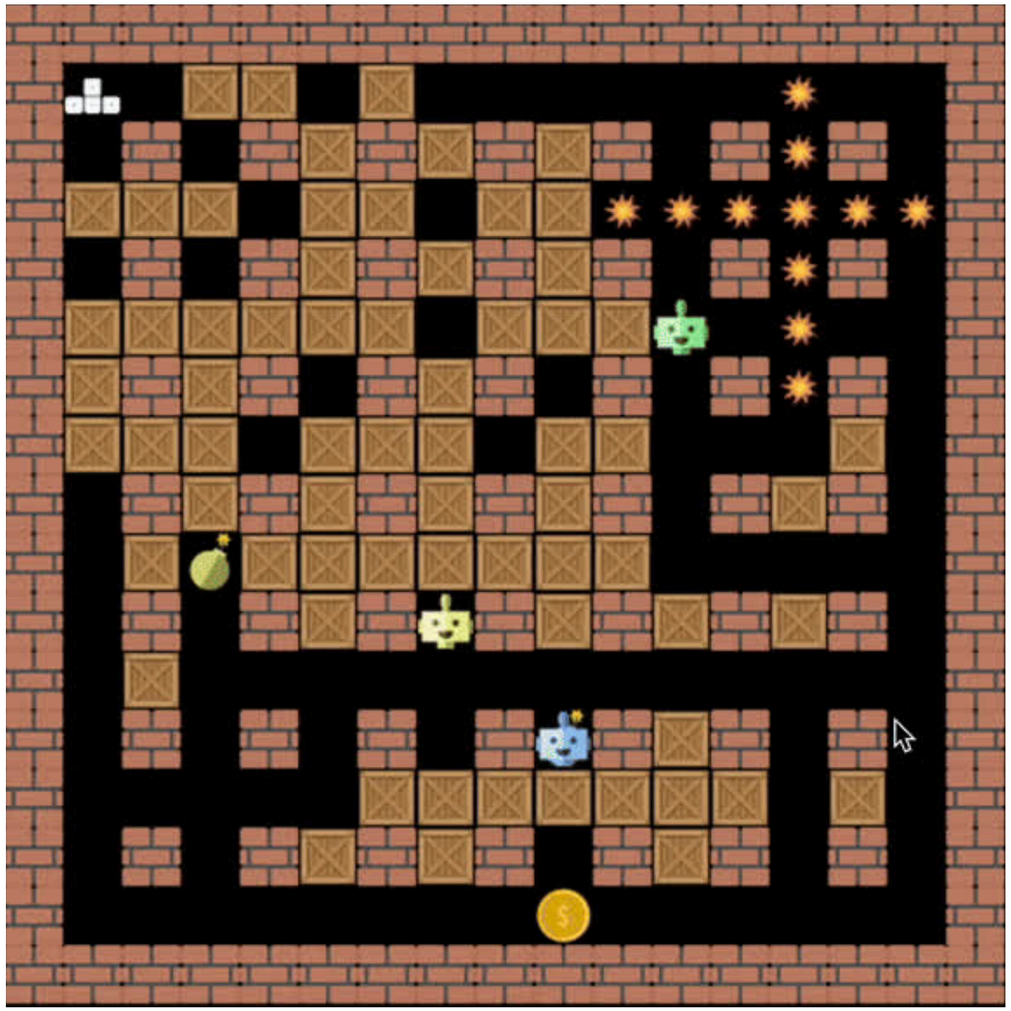
```
331  class BombeRLeWorld(GenericWorld):
348      def build_arena(self):
349          WALL = -1
350          FREE = 0
351          CRATE = 1
352          arena = np.zeros((s.COLS, s.ROWS), int)
353
354          scenario_info = s.SCENARIOS[self.args.scenario]
355
356          # Crates in random locations
357          arena[self.rng.random((s.COLS, s.ROWS)) < scenario_info["CRATE_DENSITY"]] = CRATE
358
359          # Walls
360          arena[:1, :] = WALL
361          arena[-1:, :] = WALL
362          arena[:, :1] = WALL
363          arena[:, -1:] = WALL
364          for x in range(s.COLS):
365              for y in range(s.ROWS):
366                  if (x + 1) * (y + 1) % 2 == 1:
367                      arena[x, y] = WALL
368
369          # Clean the start positions
370          start_positions = [(1, 1), (1, s.ROWS - 2), (s.COLS - 2, 1), (s.COLS - 2, s.ROWS - 2)]
371          for (x, y) in start_positions:
372              for (xx, yy) in [(x, y), (x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)]:
373                  if arena[xx, yy] == 1:
374                      arena[xx, yy] = FREE
```

BOMBERMAN GAME

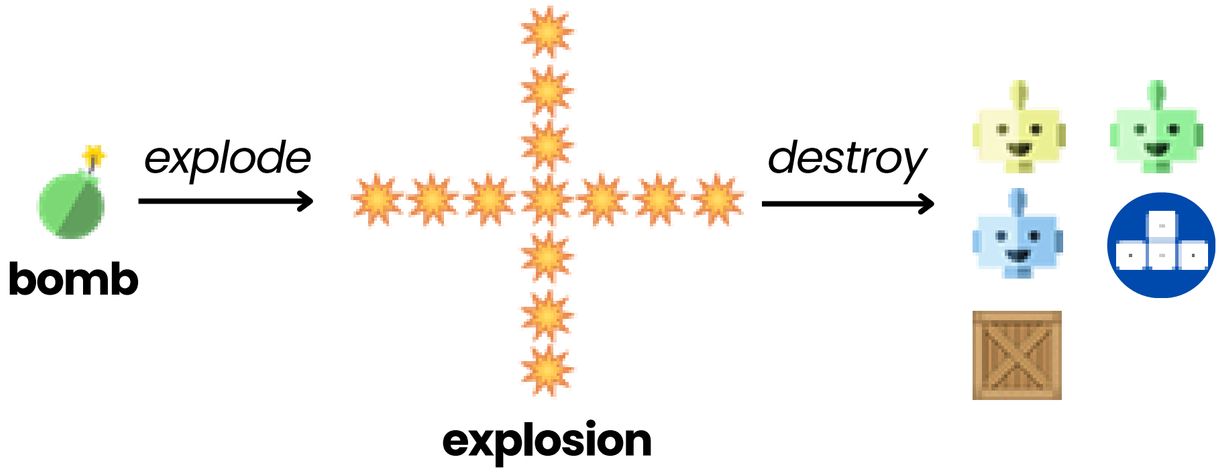


- **ENVIRONMENT**
  - Game board: 17x17 grid of cells (walls, empty cells, crates)
  - Enemies: can move and place bomb like Bomberman

BOMBERMAN GAME



- **ENVIRONMENT**
  - Game board: grid of cells (walls, empty cells, crates)
  - Enemies: can move and place bomb like Bomberman
  - Bombs and Explosions

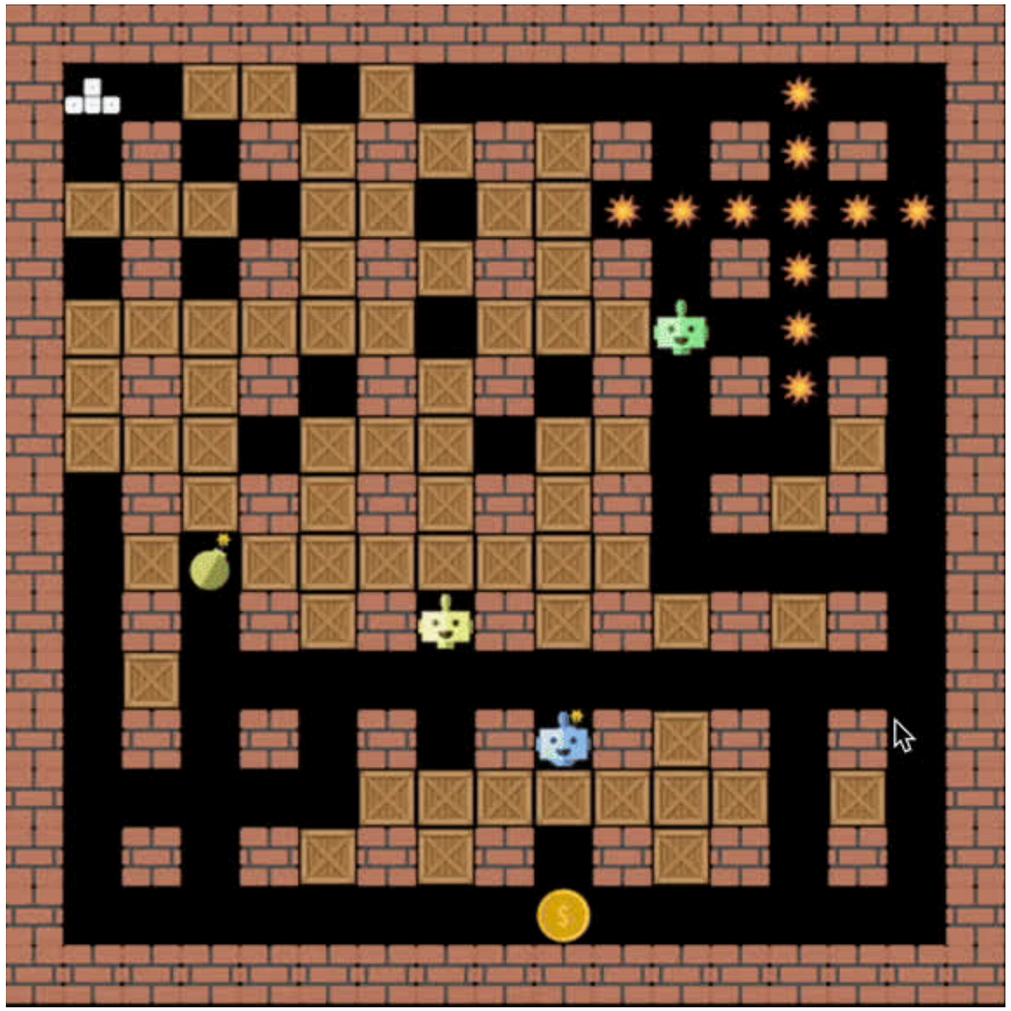


BOMBERMAN GAME

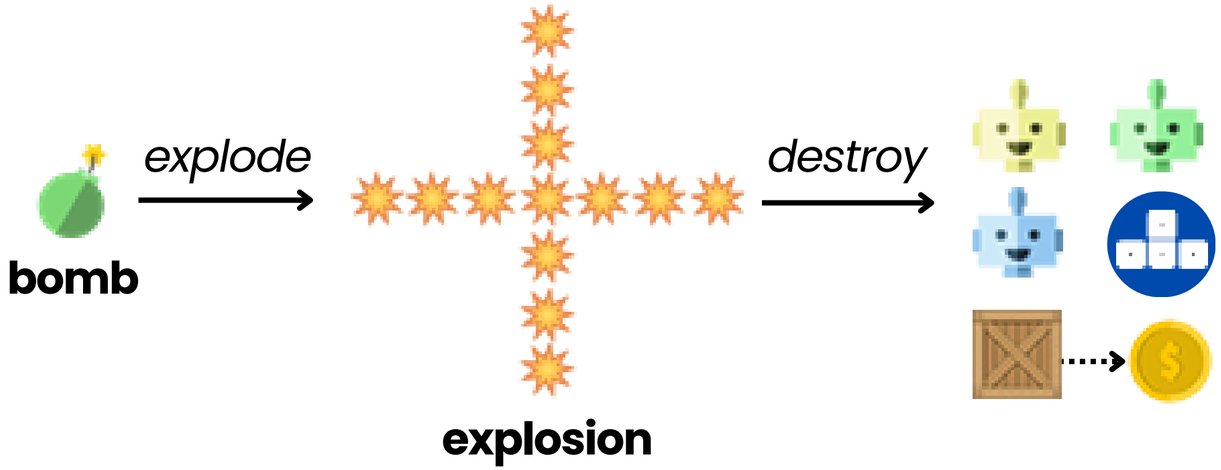
environment.py > GenericWorld > evaluate_explosions

```
30   class GenericWorld:
203      def update_bombs(self):
210          for bomb in self.bombs:
211              if bomb.timer <= 0:
212                  # Explode when timer is finished
213                  self.logger.info(f'Agent <{bomb.owner.name}>\'s bomb at {(bomb.x, bomb.y)} explodes')
214                  bomb.owner.add_event(e.BOMB_EXPLODED)
215                  blast_coords = bomb.get_blast_coords(self.arena)
216
217                  # Clear crates
218                  for (x, y) in blast_coords:
219                      if self.arena[x, y] == 1:
220                          self.arena[x, y] = 0
221                          bomb.owner.add_event(e.CRATE_DESTROYED)
222                          # Maybe reveal a coin
223                          for c in self.coins:
224                              if (c.x, c.y) == (x, y):
225                                  c.collectable = True
226                                  self.logger.info(f'Coin found at {(x, y)}')
227                                  bomb.owner.add_event(e.COIN_FOUND)
228
229                  # Create explosion
230                  screen_coords = [(s.GRID_OFFSET[0] + s.GRID_SIZE * x, s.GRID_OFFSET[1] + s.GRID_SIZE * y) for (x, y) in
231                                   blast_coords]
232                  self.explosions.append(Explosion(blast_coords, screen_coords, bomb.owner, s.EXPLOSION_TIMER))
233                  bomb.active = False
234              else:
235                  # Progress countdown
236                  bomb.timer -= 1
237          self.bombs = [b for b in self.bombs if b.active]
```

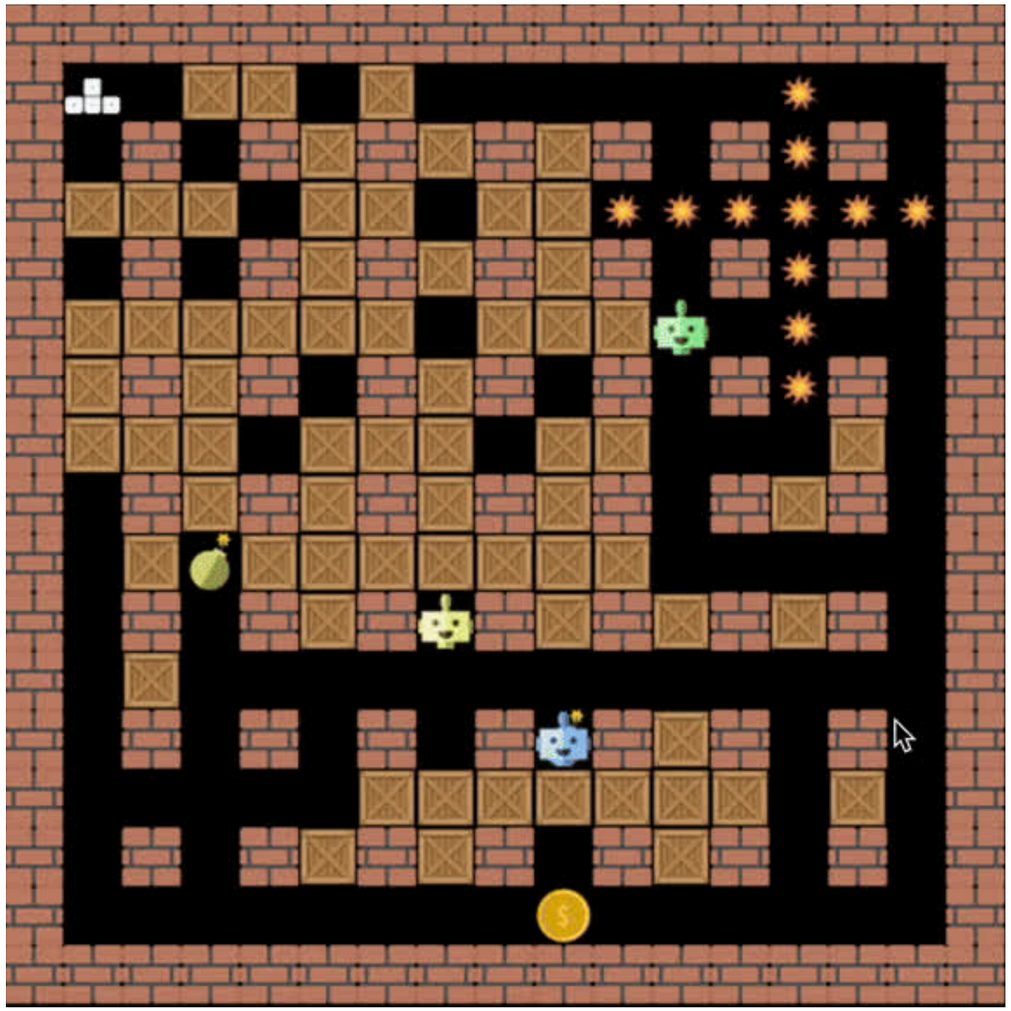
BOMBERMAN GAME



- **ENVIRONMENT**
  - Game board: grid of cells (walls, empty cells, crates)
  - Enemies: can move and place bomb like Bomberman
  - Bombs and Explosions



BOMBERMAN GAME

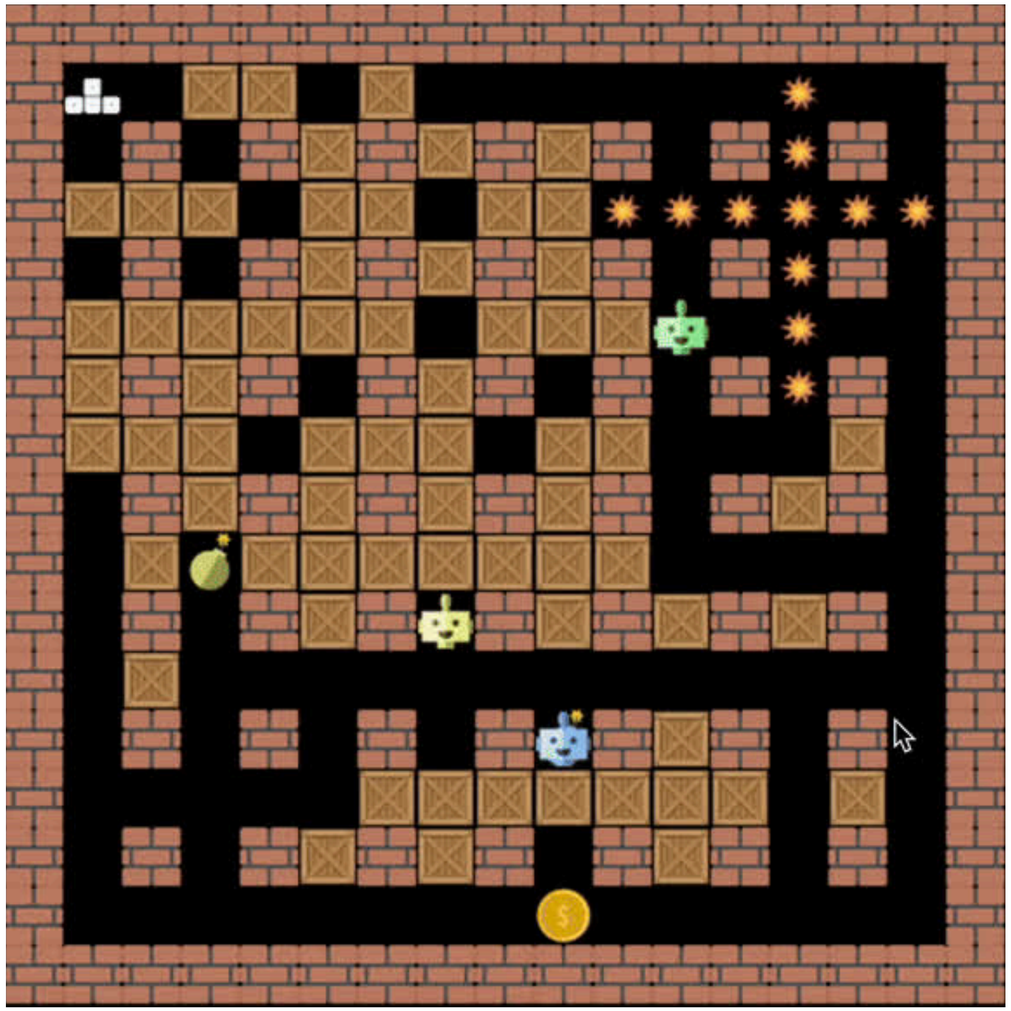


- **ENVIRONMENT**
  - Game board: grid of cells (walls, empty cells, crates)
  - Enemies: can move and place bomb like Bomberman
  - Bombs and Explosions
  - Coins

BOMBERMAN GAME

```
331  class BombeRLeWorld(GenericWorld):
348      def build_arena(self):
376          # Place coins at random, at preference under crates
377          coins = []
378          all_positions = np.stack(np.meshgrid(np.arange(s.COLS), np.arange(s.ROWS), indexing="ij"), -1)
379          crate_positions = self.rng.permutation(all_positions[arena == CRATE])
380          free_positions = self.rng.permutation(all_positions[arena == FREE])
381          coin_positions = np.concatenate([
382              crate_positions,
383              free_positions
384          ], 0)[:scenario_info["COIN_COUNT"]]
385          for x, y in coin_positions:
386              coins.append(Coin((x, y), collectable=arena[x, y] == FREE))
```

BOMBERMAN GAME



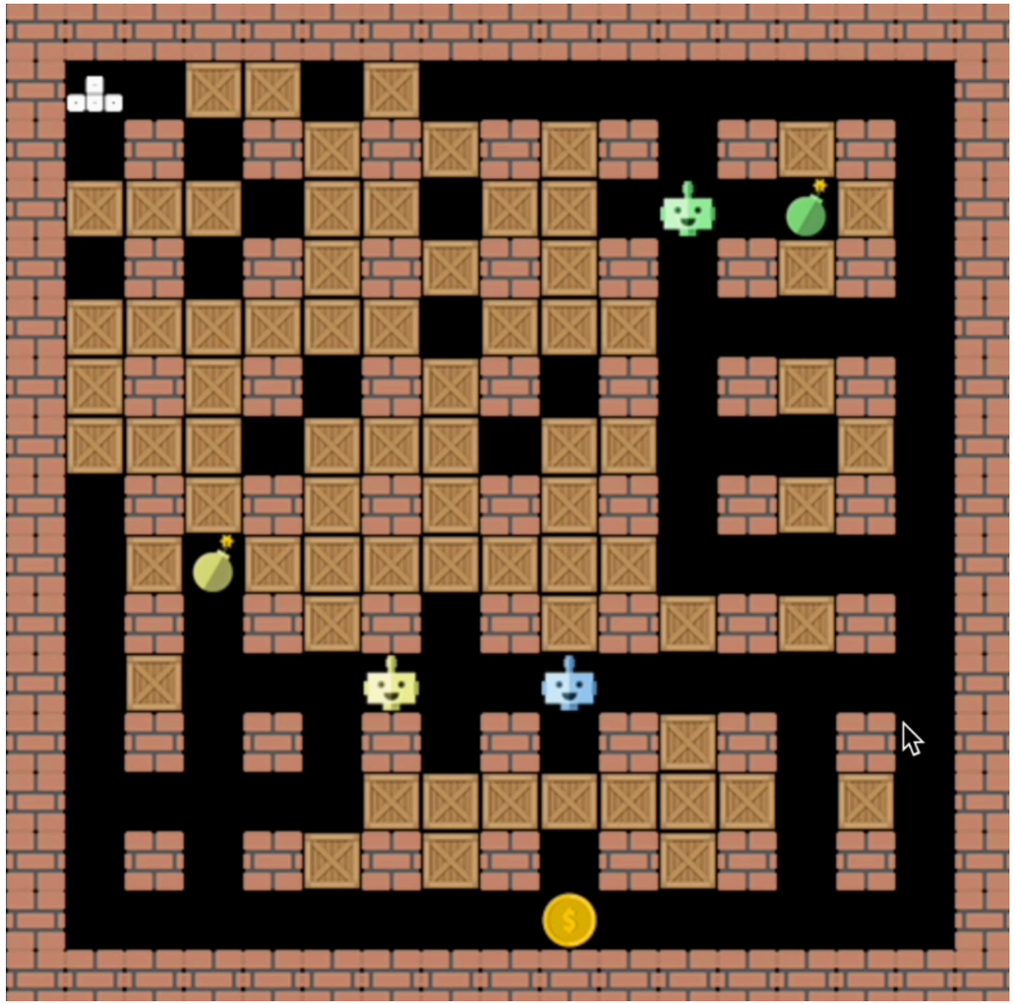
- **ENVIRONMENT**
  - Game board: grid of cells (walls, empty cells, crates)
  - Enemies: can move and place bomb like Bomberman
  - Bombs and Explosions
  - Coins
- **REWARD**
  - Collect a coin: +1
  - Kill an enemy: +5

BOMBERMAN GAME

```
30   class GenericWorld:
179      def collect_coins(self):
180          for coin in self.coins:
181              if coin.collectable:
182                  for a in self.active_agents:
183                      if a.x == coin.x and a.y == coin.y:
184                          coin.collectable = False
185                          self.logger.info(f'Agent <{a.name}> picked up coin at {(a.x, a.y)} and receives 1 point')
186                          a.update_score(s.REWARD_COIN)
187                          a.add_event(e.COIN_COLLECTED)
```

```
30   class GenericWorld:
239      def evaluate_explosions(self):
248                          # Note who killed whom, adjust scores
249                          if a is explosion.owner:
250                              self.logger.info(f'Agent <{a.name}> blown up by own bomb')
251                              a.add_event(e.KILLED_SELF)
252                              explosion.owner.trophies.append(Trophy.suicide_trophy)
253                          else:
254                              self.logger.info(f'Agent <{a.name}> blown up by agent <{explosion.owner.name}>\'s bomb')
255                              self.logger.info(f'Agent <{explosion.owner.name}> receives 5 point')
256                              explosion.owner.update_score(s.REWARD_KILL)
257                              explosion.owner.add_event(e.KILLED_OPPONENT)
258                              explosion.owner.trophies.append(pygame.transform.smoothscale(a.avatar, (15, 15)))
```

BOMBERMAN GAME



- **STATE**
  - Represents current snapshot of game world
  - Information about Bomberman, enemies, coins, bombs, explosions, round, step...

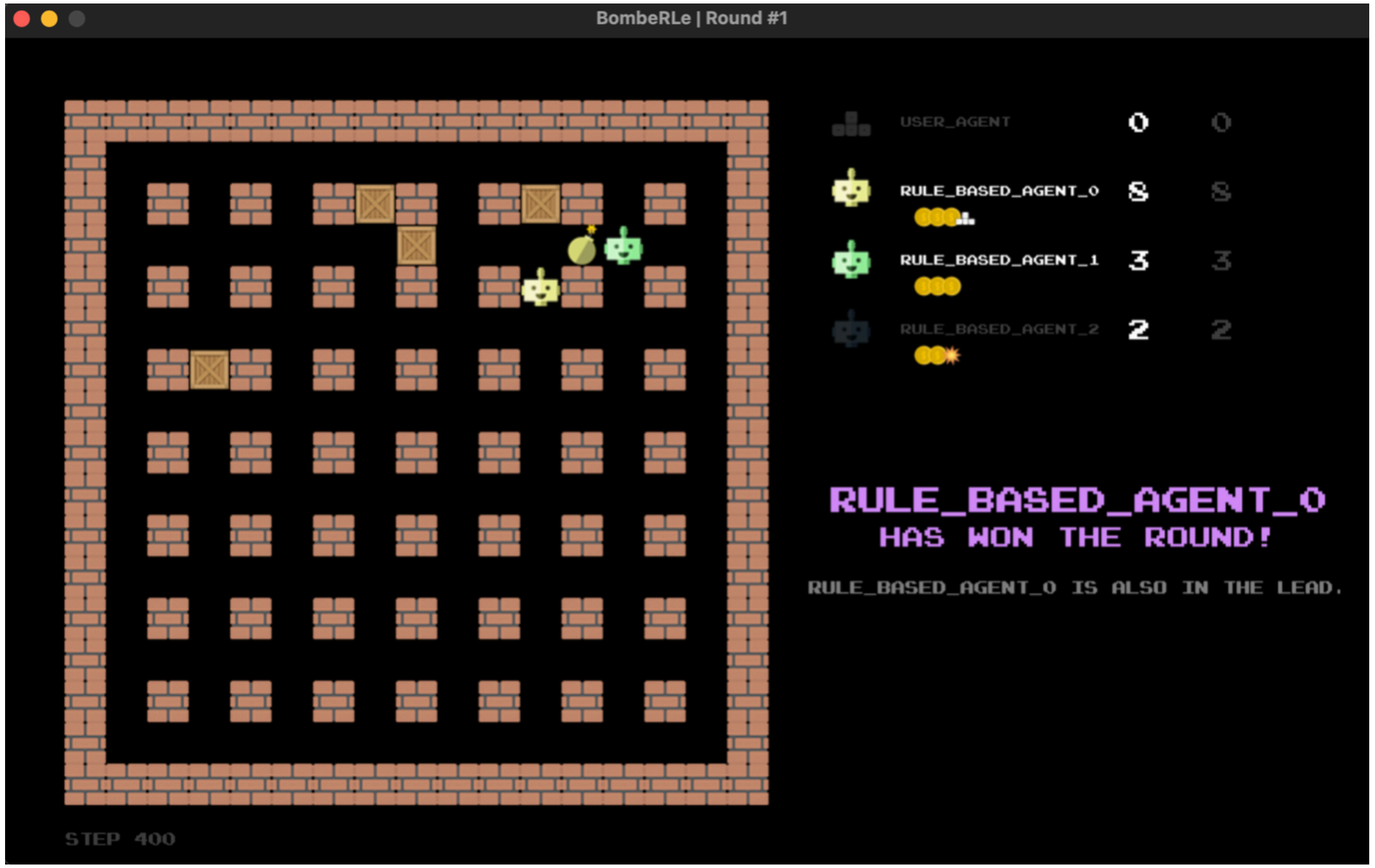
BOMBERMAN GAME

environment.py > GenericWorld

```
30    class GenericWorld:
33        running: bool = False
34        step: int
35        replay: Dict
36        round_statistics: Dict
37
38        agents: List[Agent]
39        active_agents: List[Agent]
40        arena: np.ndarray
41        coins: List[Coin]
42        bombs: List[Bomb]
43        explosions: List[Explosion]
```

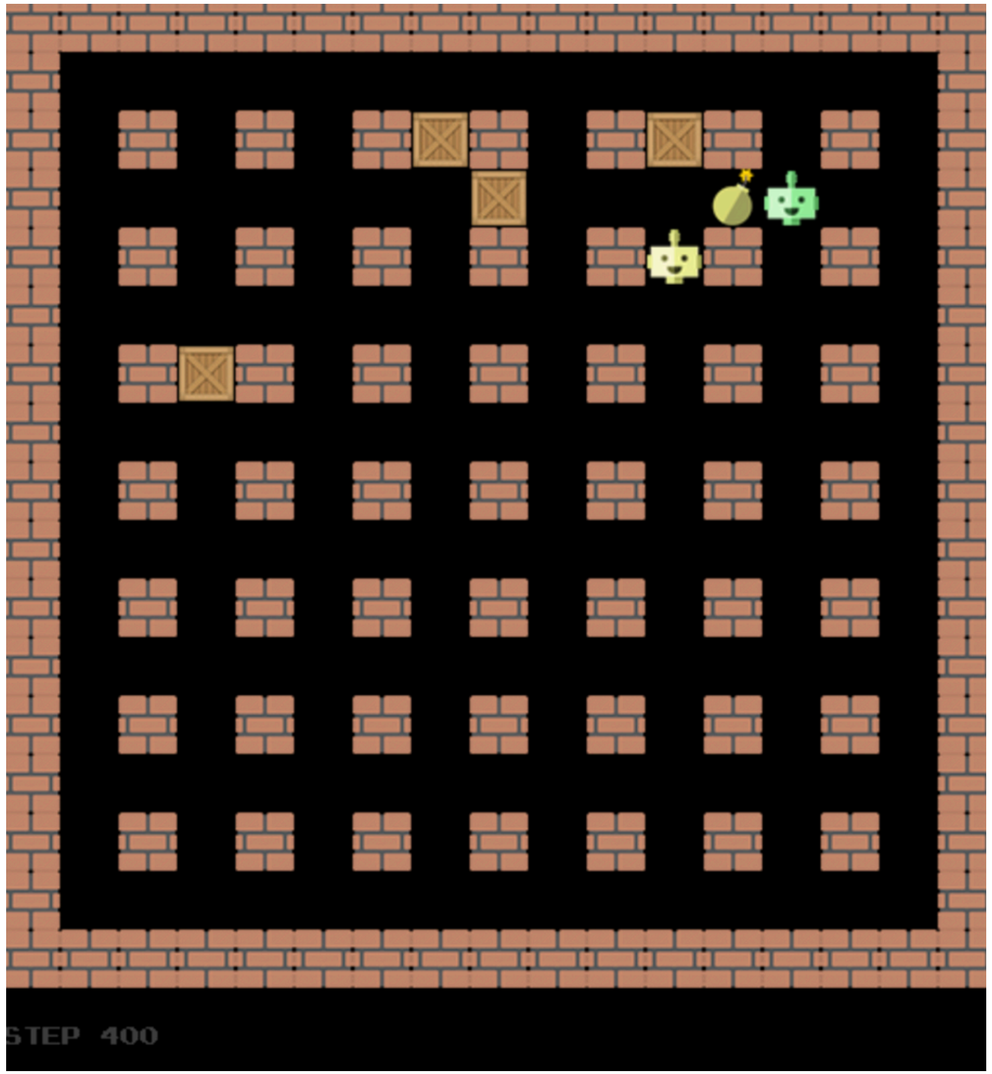
BOMBERMAN GAME

- **GOAL**
  - Kill enemies and collect coins.
  - Achieve the **highest possible score**.



BOMBERMAN GAME

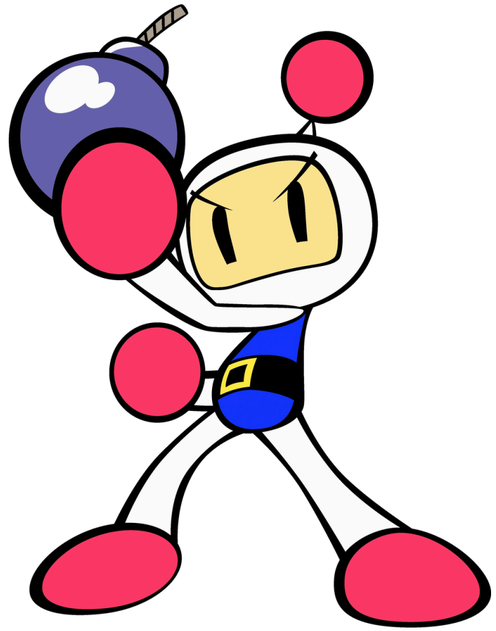
- **TERMINAL STATE**
  - Bomberman and enemies are all defeated.
  - Only Bomberman or one enemy remains, with no crates, coins, bombs, or explosions.
  - Trained agent is eliminated.
  - Exceeding step limit.



BOMBERMAN GAME

```
30  class GenericWorld:
283     def time_to_stop(self):
284         # Check round stopping criteria
285         if len(self.active_agents) == 0:
286             self.logger.info(f'No agent left alive, wrap up round')
287             return True
288
289         if (len(self.active_agents) == 1
290                 and (self.arena == 1).sum() == 0
291                 and all([not c.collectable for c in self.coins])
292                 and len(self.bombs) + len(self.explosions) == 0):
293             self.logger.info(f'One agent left alive with nothing to do, wrap up round')
294             return True
295
296         if any(a.train for a in self.agents) and not self.args.continue_without_training:
297             if not any([a.train for a in self.active_agents]):
298                 self.logger.info('No training agent left alive, wrap up round')
299                 return True
300
301         if self.step >= s.MAX_STEPS:
302             self.logger.info('Maximum number of steps reached, wrap up round')
303             return True
304
305         return False
```

Table of CONTENT



STATE REPRESENTATION

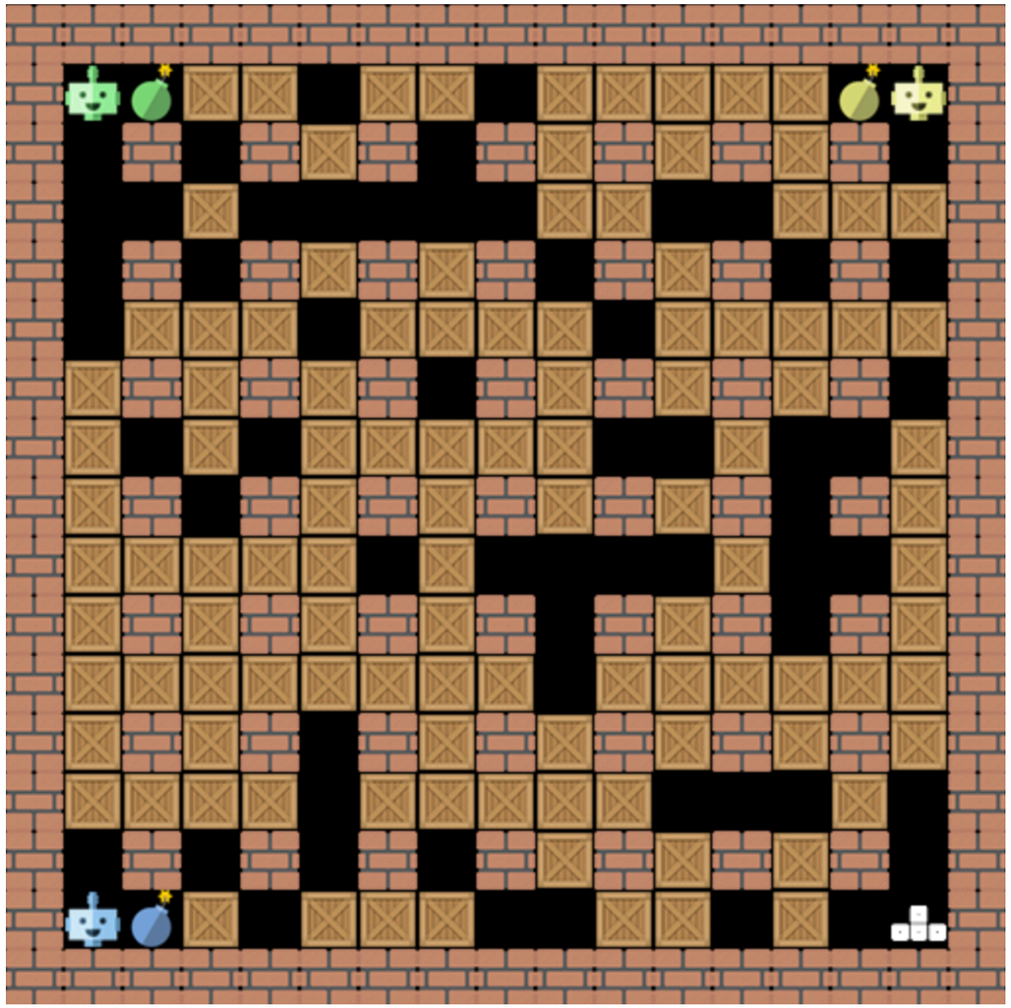
- 4 kinds of representation:
  - Grid (Wall, Empty, Crate, Coin)
  - Agent position
  - Opponent positions
  - Bombs

- Each of the four kinds of representation is a matrix of the same size as the map.

- The state vector is created by concatenating these matrices, making the size of the observation vector four times larger than the grid size

STATE REPRESENTATION

1. Grid:

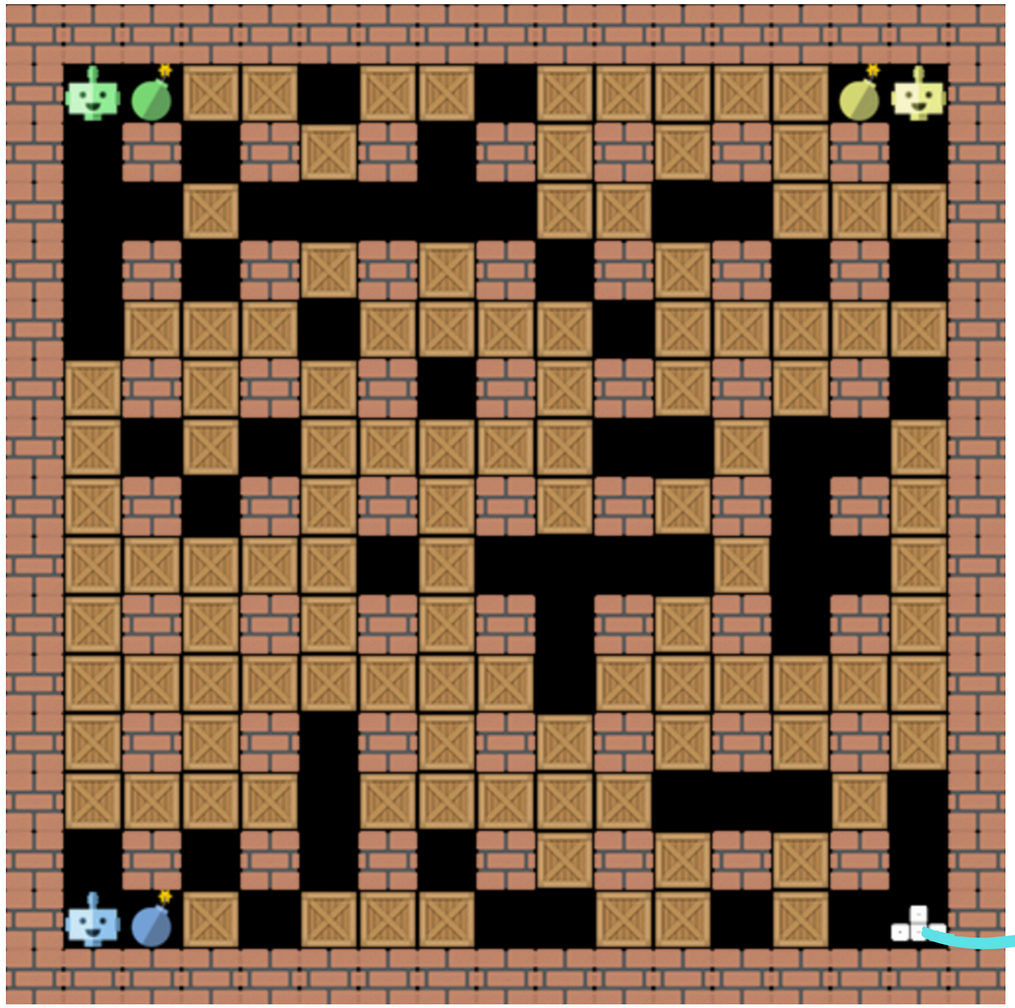


A 17x17 matrix where:

- Empty Cell: 0 ( )
- Wall Cell: -1 ( )
- Crate Cell: 1 ( )
- Coin Cell: 2 ( )

STATE REPRESENTATION

2. Agent Position:

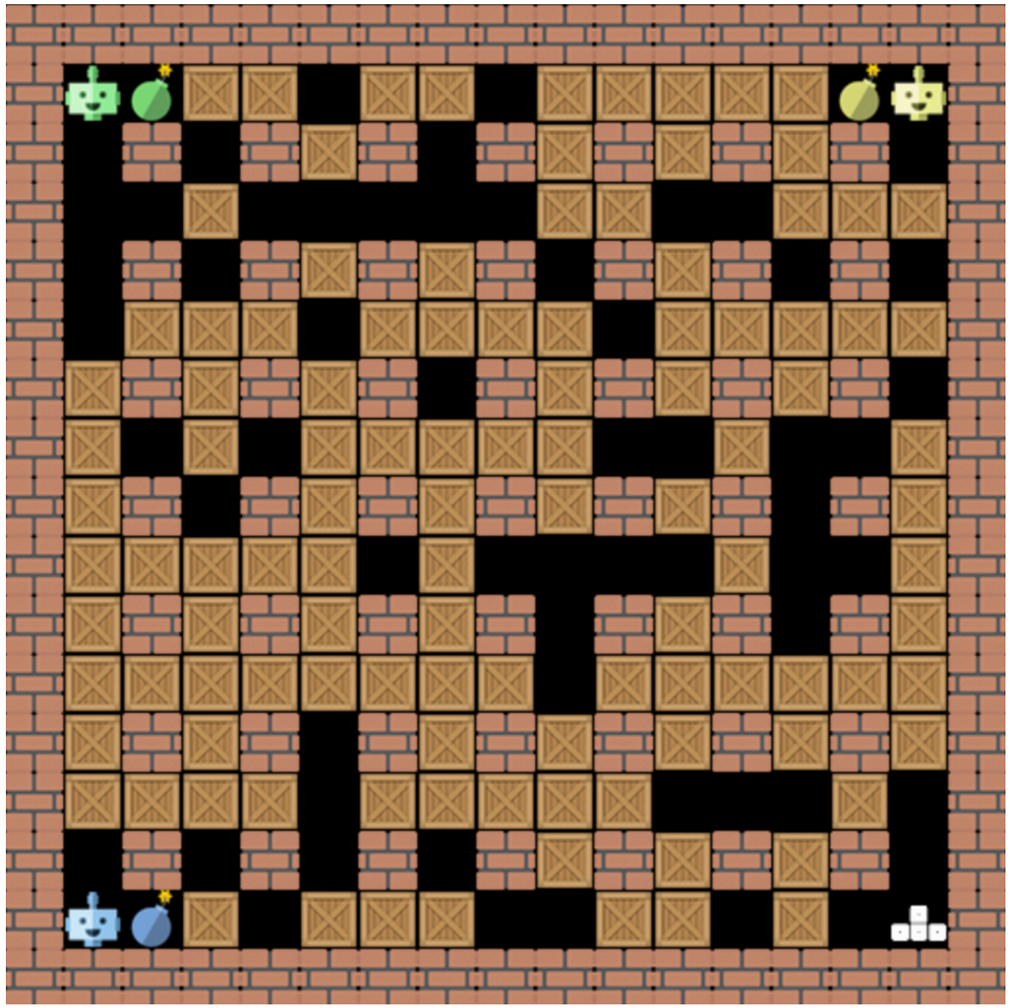


A 17x17 matrix where:

- Agent Present: 1
- Agent Absent: 0

STATE REPRESENTATION

3. Opponent Positions:

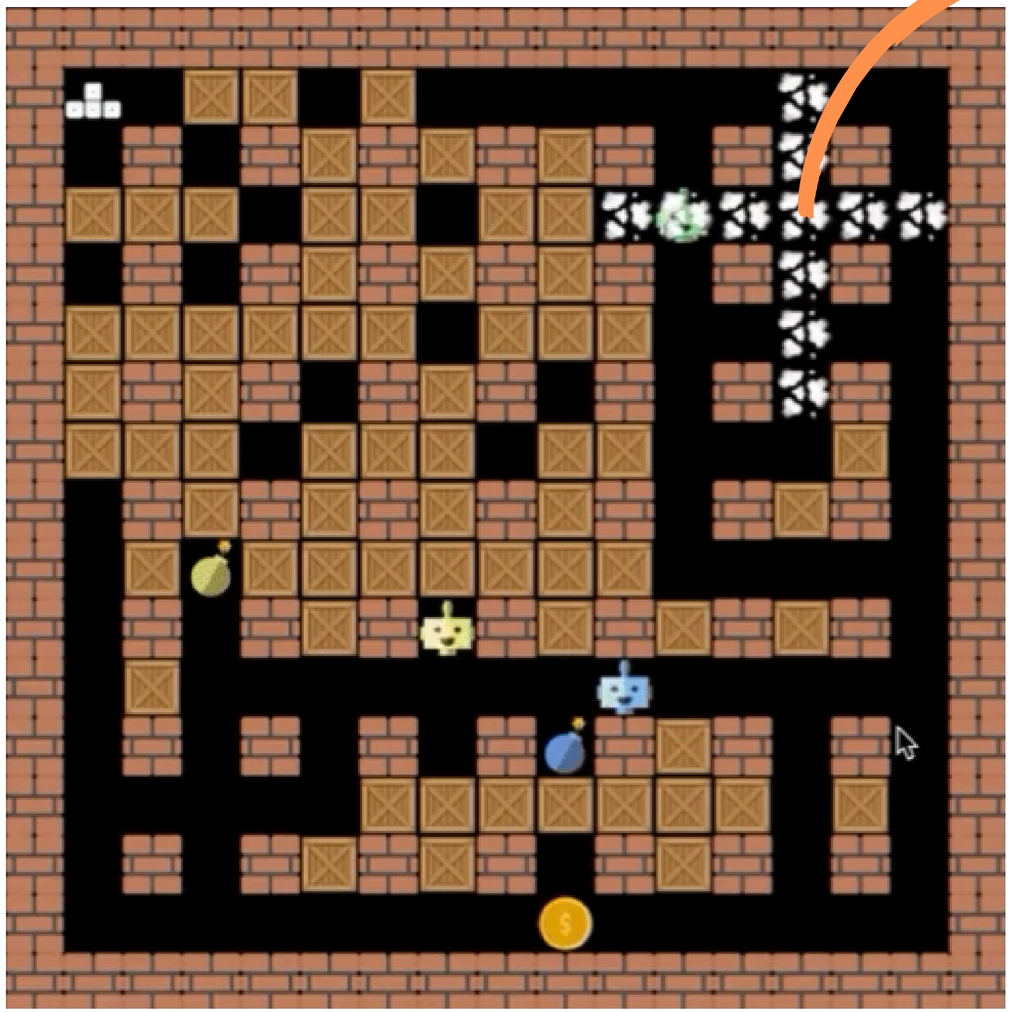


A 17x17 matrix where:

- Opponent Present: 1
- Opponent Absent: 0

STATE REPRESENTATION

4. Bombs



Explosion cells

A 17x17 matrix where the value of:

- Agent's bomb explosion cells = -v
- Opponents' bomb explosion cells = v
- Non-explosion cells = 0

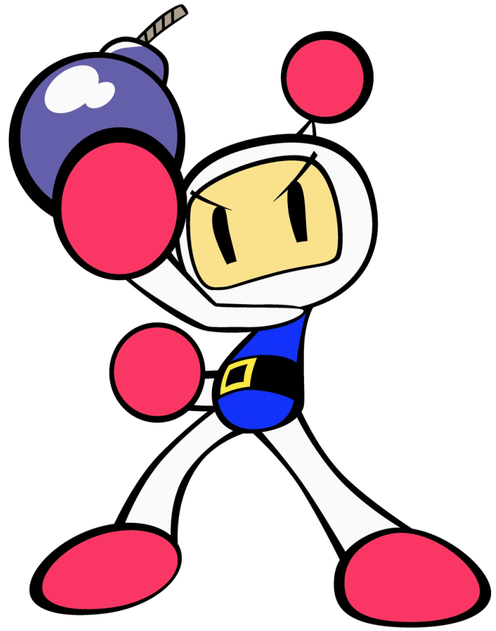
Danger value: v = curr timer / max timer

REWARD REPRESENTATION

WAITED
INVALID_ACTION
BOMB_DROPPED
CRATE_DESTROYED
COIN_FOUND
COIN_COLLECTED
KILLED_OPPONENT
KILLED_SELF
GOT_KILLED

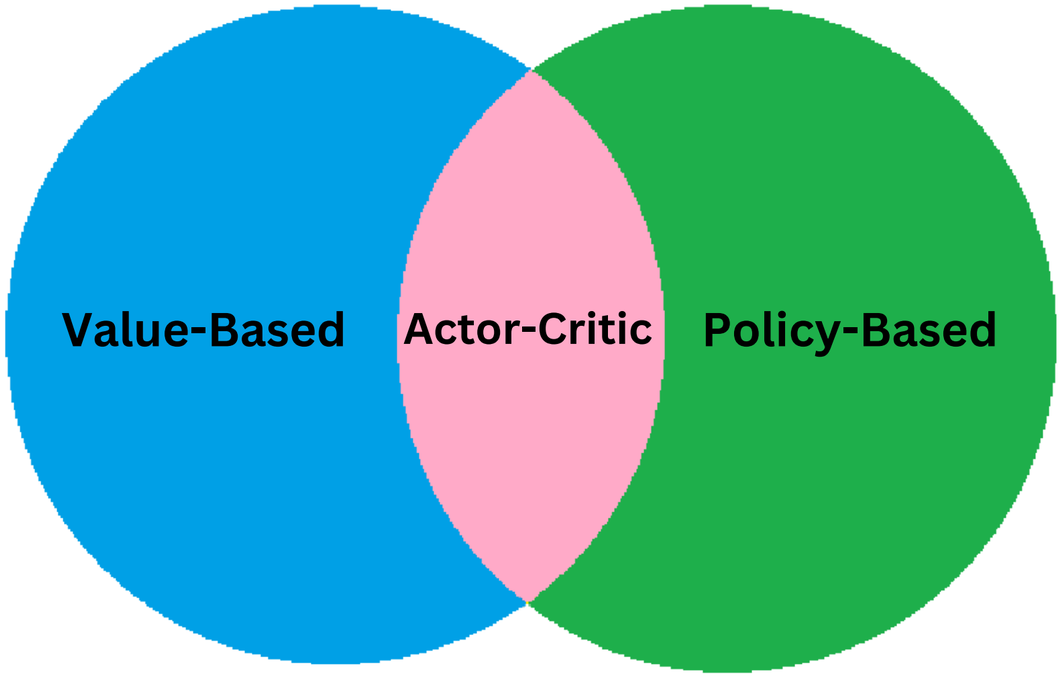
MOVED_TO_COIN
MOVED_AWAY_FROM_COIN
MOVED_TO_BOMB
MOVED_AWAY_FROM_BOMB
CORNER_BOMB
SMART_BOMB
DUMB_BOMB
. . .

Table of CONTENT



ACTOR - CRITIC

Reinforcement Learning



- Two Main Components:
  - Actor: Learns the policy to decide actions.
  - Critic: Evaluates actions using a value function.
- Policy and Value Function:
  - Actor updates the policy based on critic's feedback, while the critic updates the value function based on received rewards.
- Improved Stability:
  - Combines advantages of both value-based and policy-based methods, leading to better stability and performance.

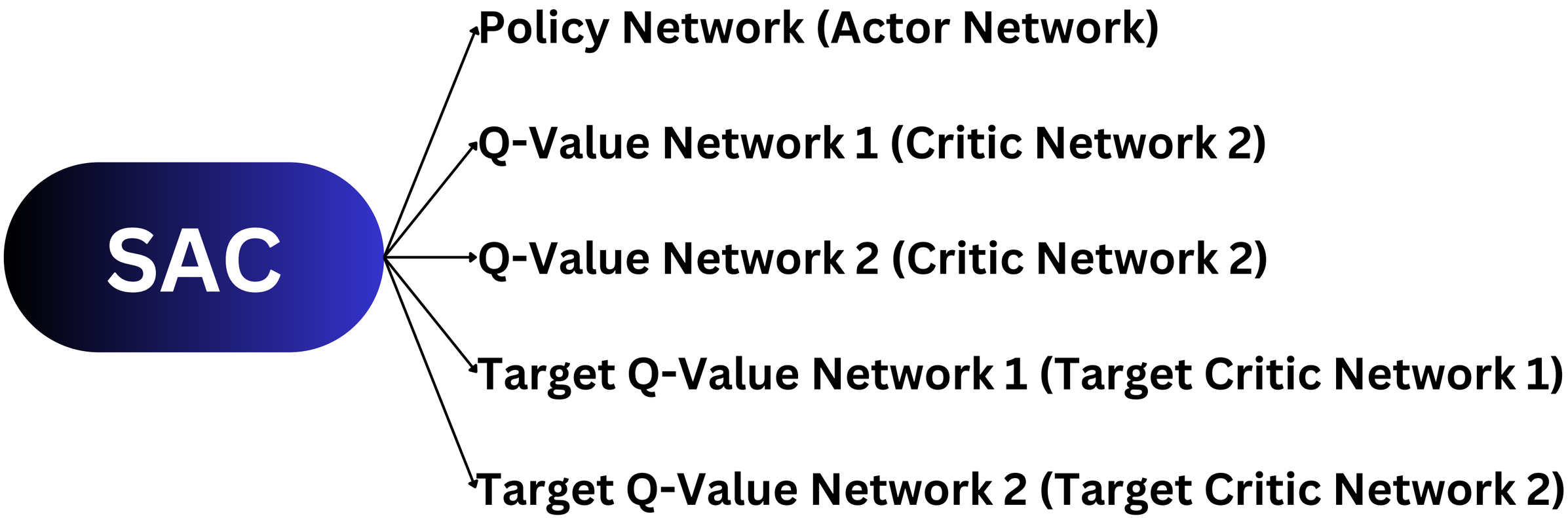
SAC

- SAC is an algorithm belonging to the actor-critic framework. In SAC:
  - They use more networks than previous algorithms.
  - They use entropy as a tool to control the behavior of agent in the future.
  - They use a difference method to update parameters called soft-update.
  - SAC balances exploration and exploitation by optimizing both the expected reward and the policy's entropy.
  - It excels in continuous action spaces and achieves state-of-the-art performance on challenging benchmarks.

NETWORKS

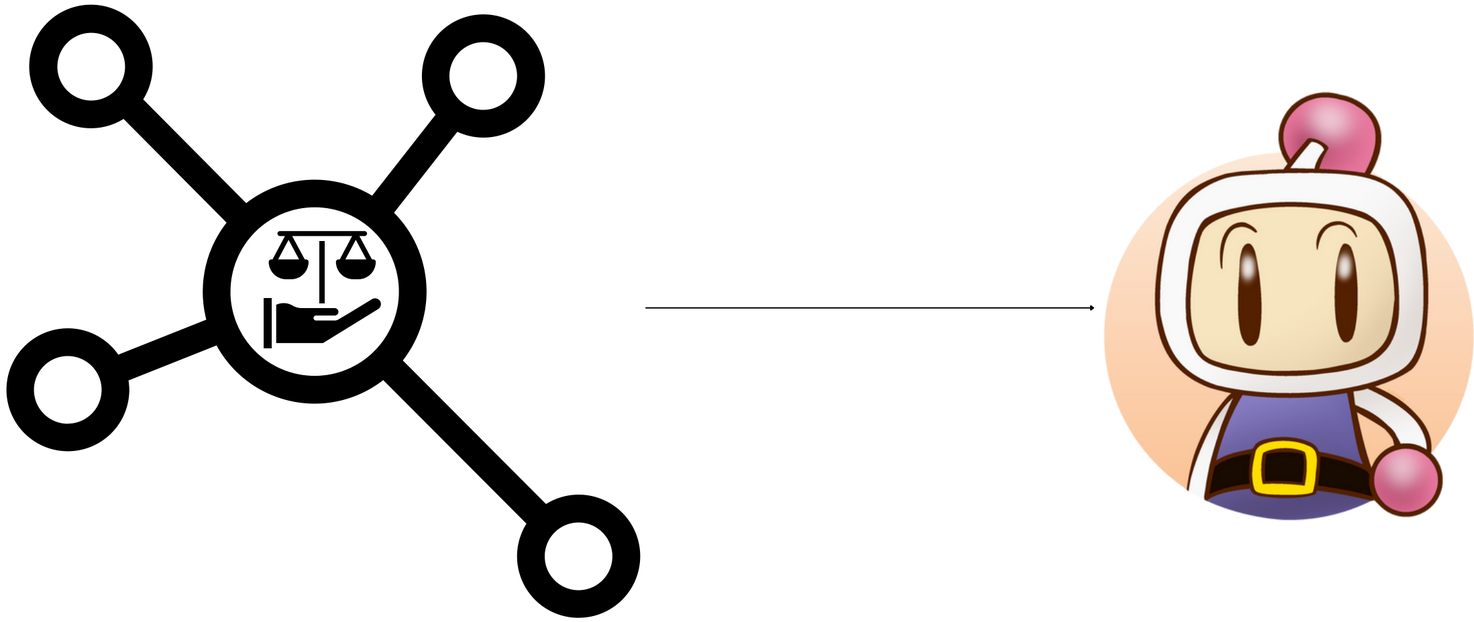

PPO
Policy Network (Actor Network)
Value Network (Critic Network)

NETWORKS


SAC
Policy Network (Actor Network)
Q-Value Network 1 (Critic Network 2)
Q-Value Network 2 (Critic Network 2)
Target Q-Value Network 1 (Target Critic Network 1)
Target Q-Value Network 2 (Target Critic Network 2)

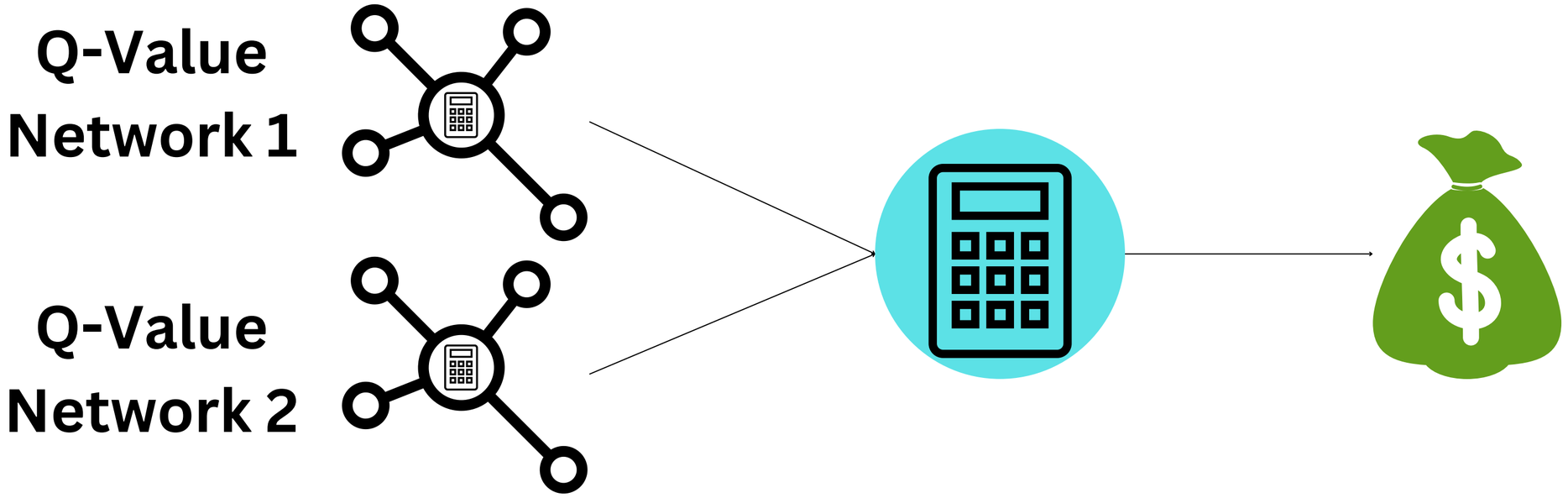
NETWORK

- Policy network:
  - Purpose: Determine the next action of agent.
  - Input: Environment current state.
  - Output: A probability distribution over actions (Gaussian)



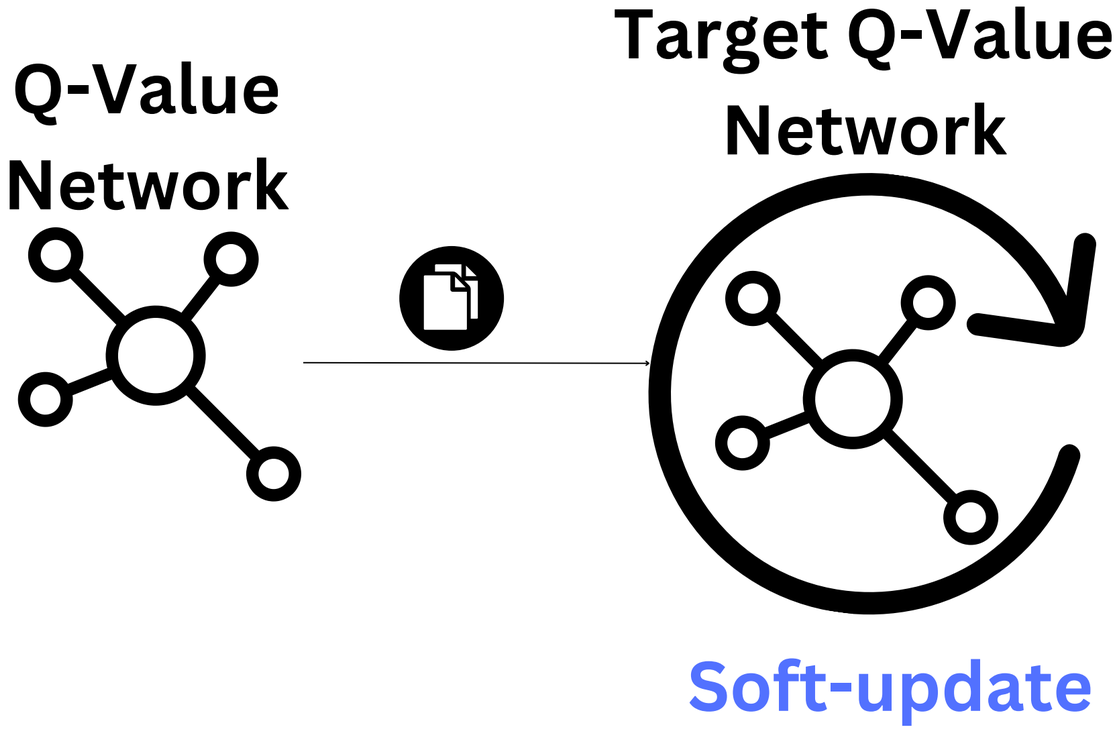
NETWORKS

- Critic networks:
  - Purpose: Estimate the value of an action in a determined state.
  - Input: A pair of action and state.
  - Output: A value

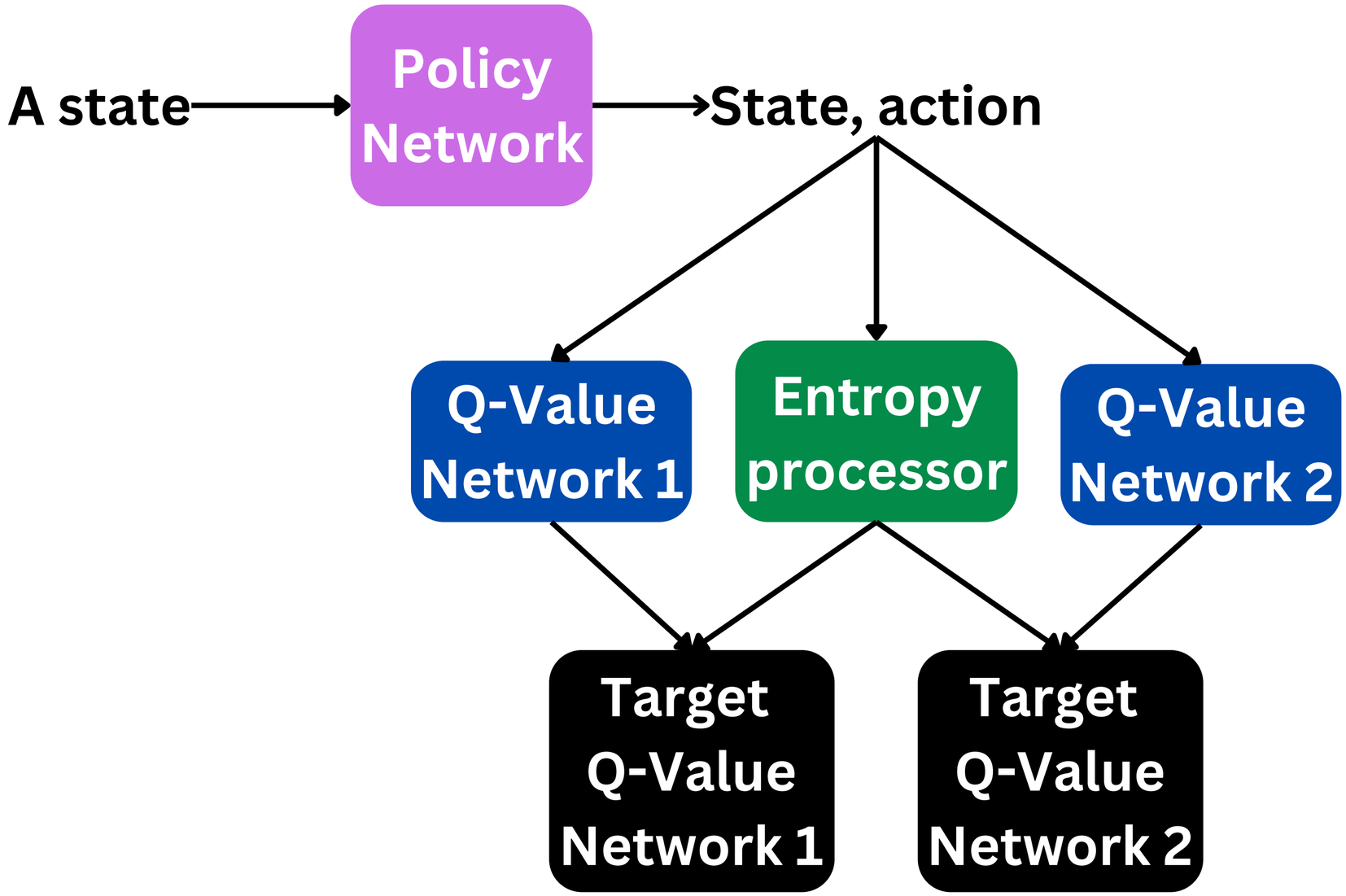


NETWORKS

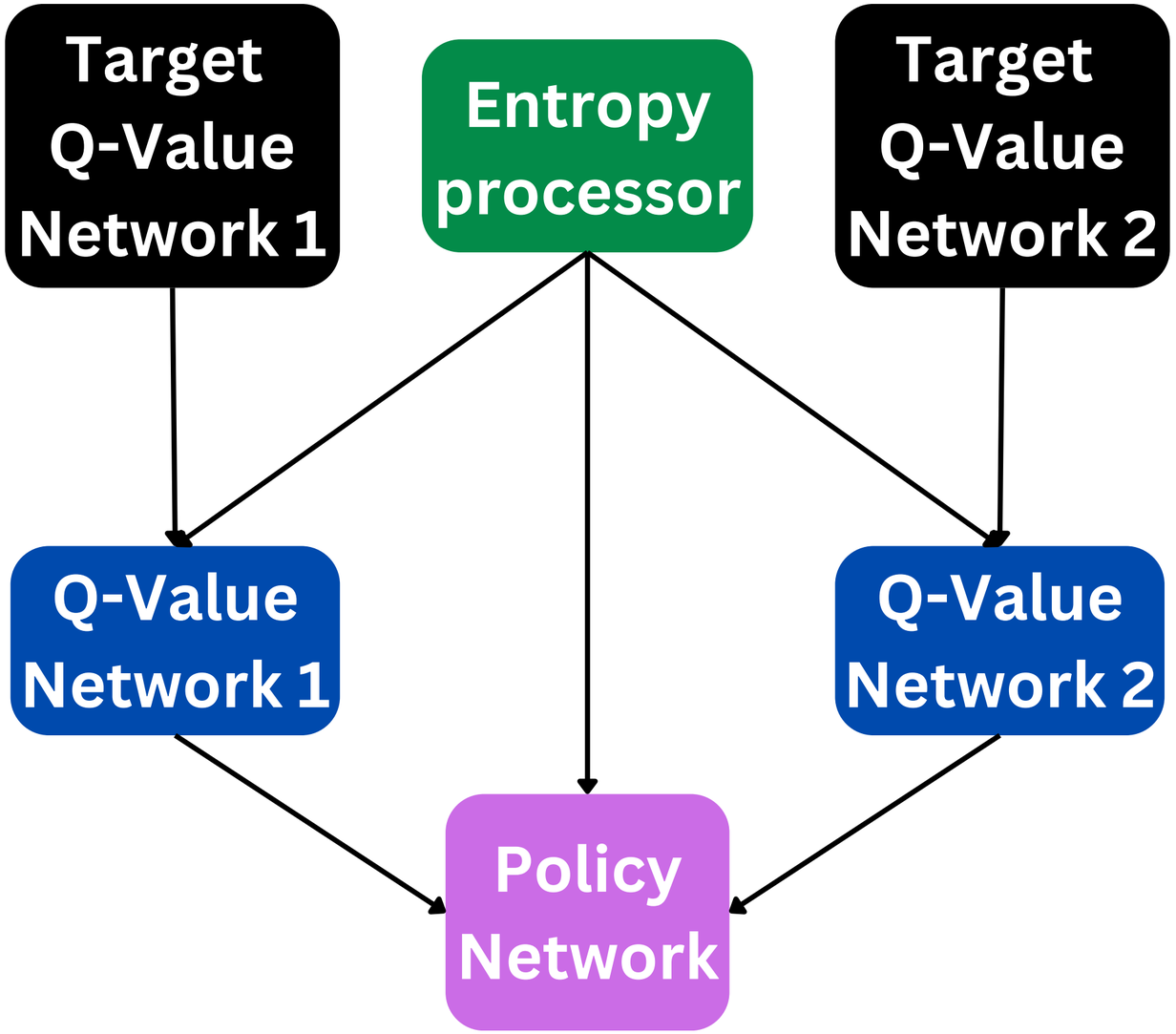
- Target Q-Value Networks (Target Critic Network):
  - Purpose: Provide stable values for Q-Value Network in update progress.
  - Structure: Parameters are copied from Q-Value Network when initialization, updated by soft-update.



FORWARD


A state
Policy Network
State, action
Q-Value Network 1
Entropy processor
Q-Value Network 2
Target Q-Value Network 1
Target Q-Value Network 2

BACKWARD


Target Q-Value Network 1
Entropy processor
Target Q-Value Network 2
Q-Value Network 1
Q-Value Network 2
Policy Network

ENTROPY

- Entropy
  - This is a tool to measure the randomness of decisions made by a policy.

- Policy with high entropy value → more randomness in choosing future action.

- On the other hand → less randomness in choosing.

- In SAC, entropy is calculated by:

$$\mathcal{H}(\pi(\cdot|s_t)) = E_{a_t \sim \pi(\cdot|s_t)} \left[-\log \pi(a_t|s_t) \right]$$

that equivalent to:

$$\mathcal{H}(\pi(\cdot|s_t)) = -\sum_{a_t} \pi(a_t|s_t) \log \pi(a_t|s_t)$$

ENTROPY IN SAC

- SAC try to maximize the entropy value. That help agent can explore and experience more situation.

- Some benefits:
  - That can balance between exploration and choosing actions which is known as good actions.

  - Algorithm value can escape from the local optima.

  - The training progress will smoothy and stable, it also easy to converging.

ENTROPY IN SAC

- SAC try to maximize the entropy value. That help agent can explore and experience more situation.

- Some benefits:
  - That can balance between exploration and choosing actions which is known as good actions.

  - Algorithm value can escape from the local optima.

  - The training progress will smoothy and stable, it also easy to converging.

HOW ?

ENTROPY IN SAC

- In SAC entropy is modeled and adjusted by a parameter called entropy parameter (α)

- Target value function:

$$J(\pi) = E_{s_t \sim \mathcal{D}, a_t \sim \pi} \left[Q(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right]$$

  - α with high value:
    - Prioritize exploration
  - On the other hand:
    - Use prior knowledge

ENTROPY IN SAC

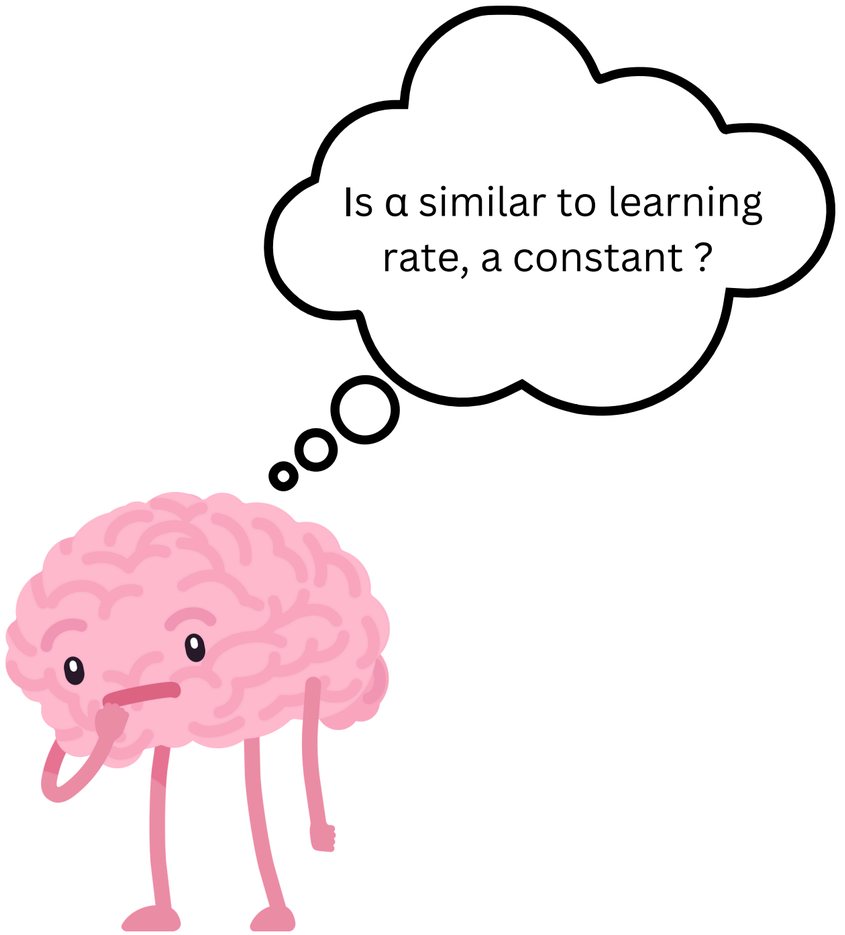
$$J(\pi) = E_{s_t \sim \mathcal{D}, a_t \sim \pi} \left[Q(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right]$$

- Expected reward: $Q(s_t, a_t)$
- Entropy value: $\alpha \mathcal{H}(\pi(\cdot|s_t))$
- We need to maximize the function J and maximize the entropy, and in the SAC algorithm we accept the trade-off of the value of the expected reward to maximize the above two things.

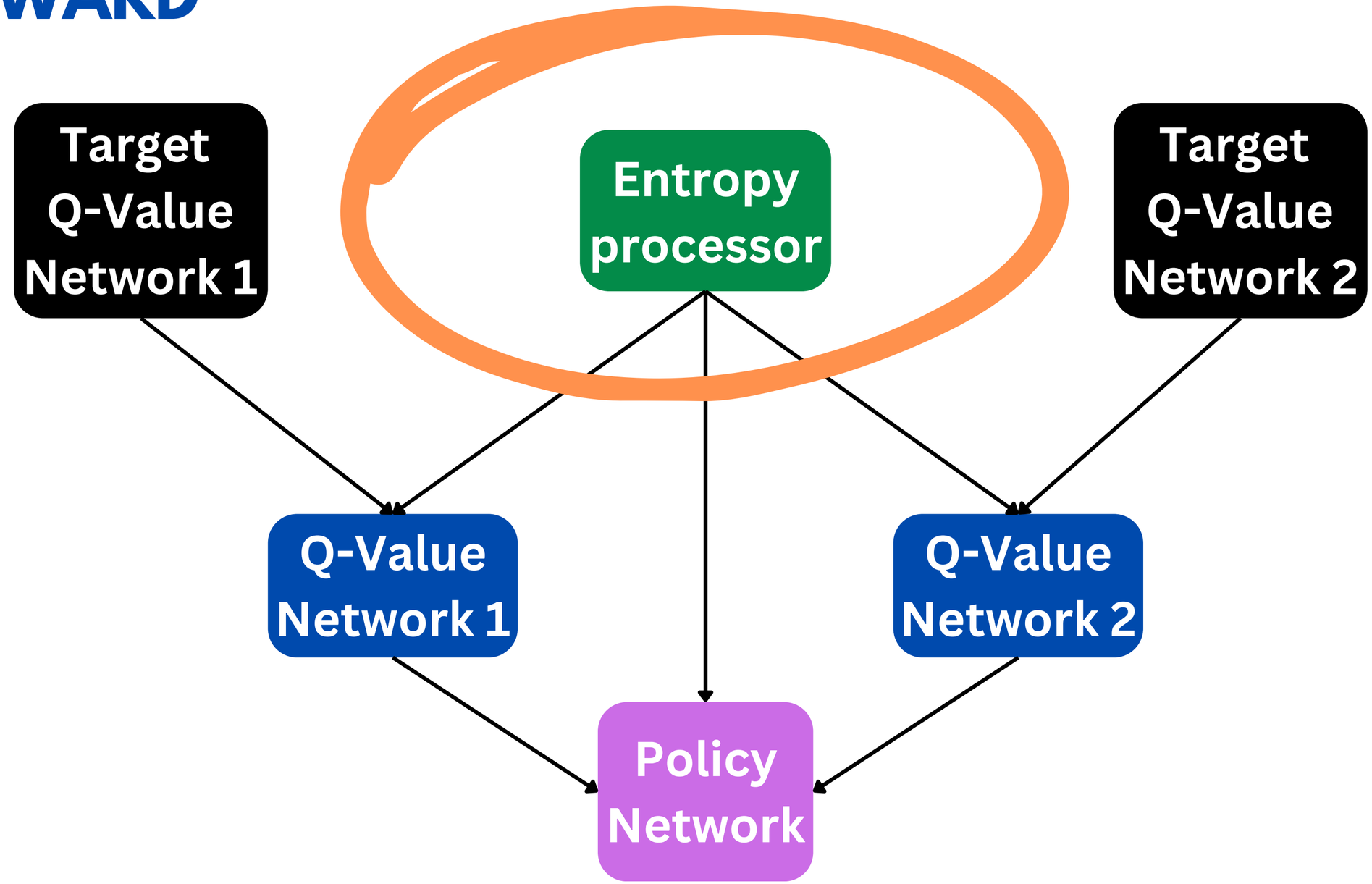
ENTROPY IN SAC

$$J(\pi) = E_{s_t \sim \mathcal{D}, a_t \sim \pi} \left[Q(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right]$$

- Expected reward: $Q(s_t, a_t)$
- Entropy value: $\alpha \mathcal{H}(\pi(\cdot|s_t))$
- We need to maximize the function J and maximize the entropy, and in the SAC algorithm we accept the trade-off of the value of the expected reward to maximize the above two things.



BACKWARD


Target Q-Value Network 1
Entropy processor
Target Q-Value Network 2
Q-Value Network 1
Q-Value Network 2
Policy Network

ENTROPY IN SAC

$$J(\pi) = E_{s_t \sim \mathcal{D}, a_t \sim \pi} \left[Q(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right]$$

- α is a dynamic value, when training this value is updated by:

$$\alpha \leftarrow \alpha - \lambda \left(E_{s_t \sim \mathcal{D}, a_t \sim \pi_\alpha} \left[-\log \pi_\alpha(a_t|s_t) \right] - \mathcal{H}_{target} \right)$$

- Where:
  - α is the temperature parameter.
  - λ is the learning rate.
  - $\pi_\alpha(a_t|s_t)$ is the policy parameterized by α, which determines the probability of action a_t given state s_t.
  - $\mathcal{H}_{target}$ is the target entropy, a hyperparameter that defines the desired level of entropy.

ENTROPY IN SAC

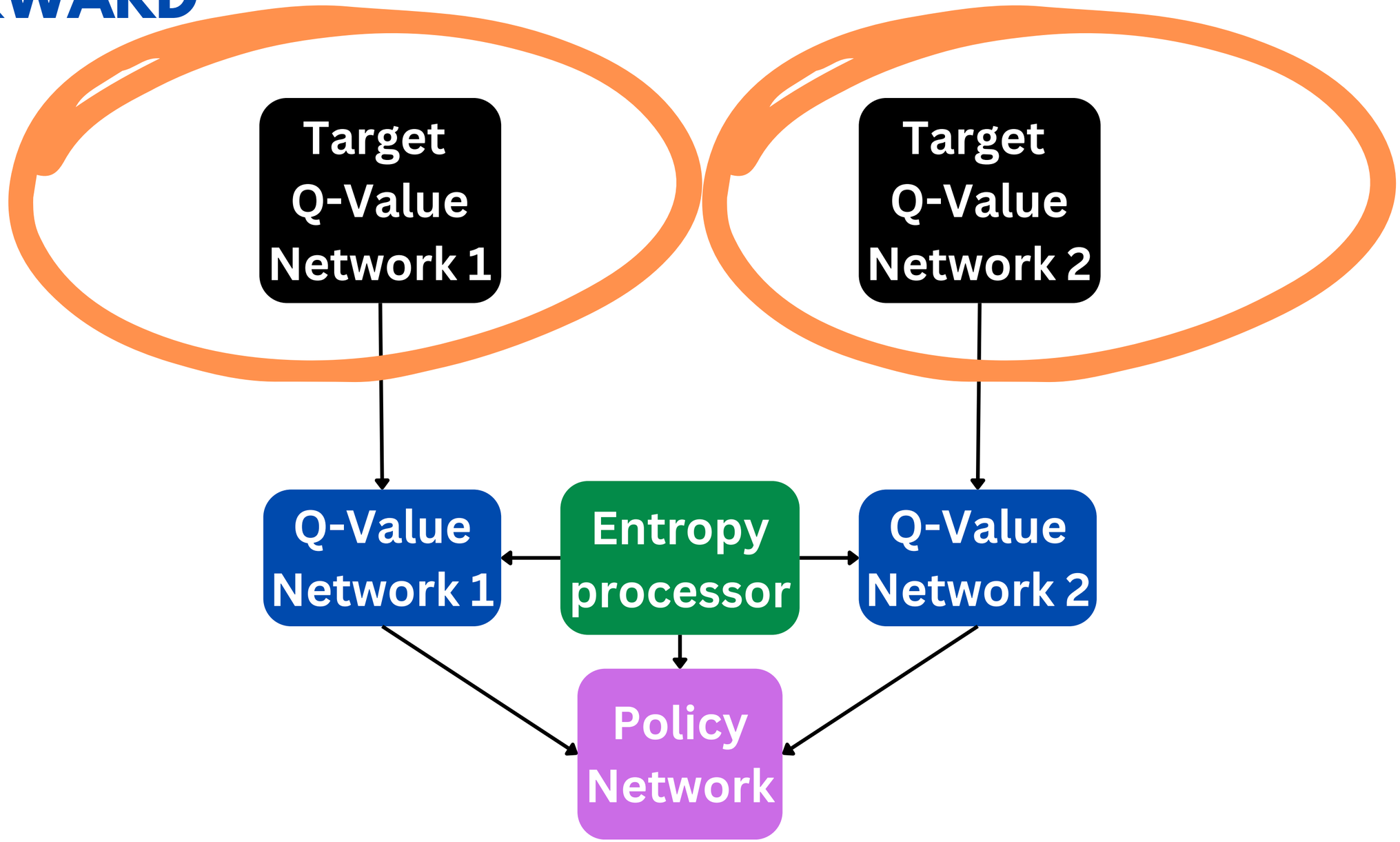
$$\alpha \leftarrow \alpha - \lambda \left(E_{s_t \sim \mathcal{D}, a_t \sim \pi_\alpha} \left[-\log \pi_\alpha(a_t|s_t) \right] - \mathcal{H}_{target} \right)$$

- Where:
  - α is the temperature parameter.
  - λ is the learning rate.
  - $\pi_\alpha(a_t|s_t)$ is the policy parameterized by α, which determines the probability of action a_t given state s_t.
  - $\mathcal{H}_{target}$ is the target entropy, a hyperparameter that defines the desired level of entropy.

- $\mathcal{H}_{target}$ usually < 0 and = -|A| with A is the number of actions.

TARGET Q-VALUE NETWORK

BACKWARD


Target Q-Value Network 1
Target Q-Value Network 2
Q-Value Network 1
Entropy processor
Q-Value Network 2
Policy Network

BACKWARD

- Update target Q-value parameters:

$$\theta_{target} \leftarrow \tau\theta + (1 - \tau)\theta_{target}$$

- Where:
  - θ: Q-Value network parameters.
  - θ_{target}: Target Q-Value network parameters.
  - τ : Learning rate.

BACKWARD

- Update target Q-value parameters:

$$\theta_{target} \leftarrow \tau\theta + (1 - \tau)\theta_{target}$$

- Where:
  - θ: Q-Value network parameters.
  - θ_{target}: Target Q-Value network parameters.
  - τ : Learning rate.

SOFT-UPDATE

SOFT-UPDATE

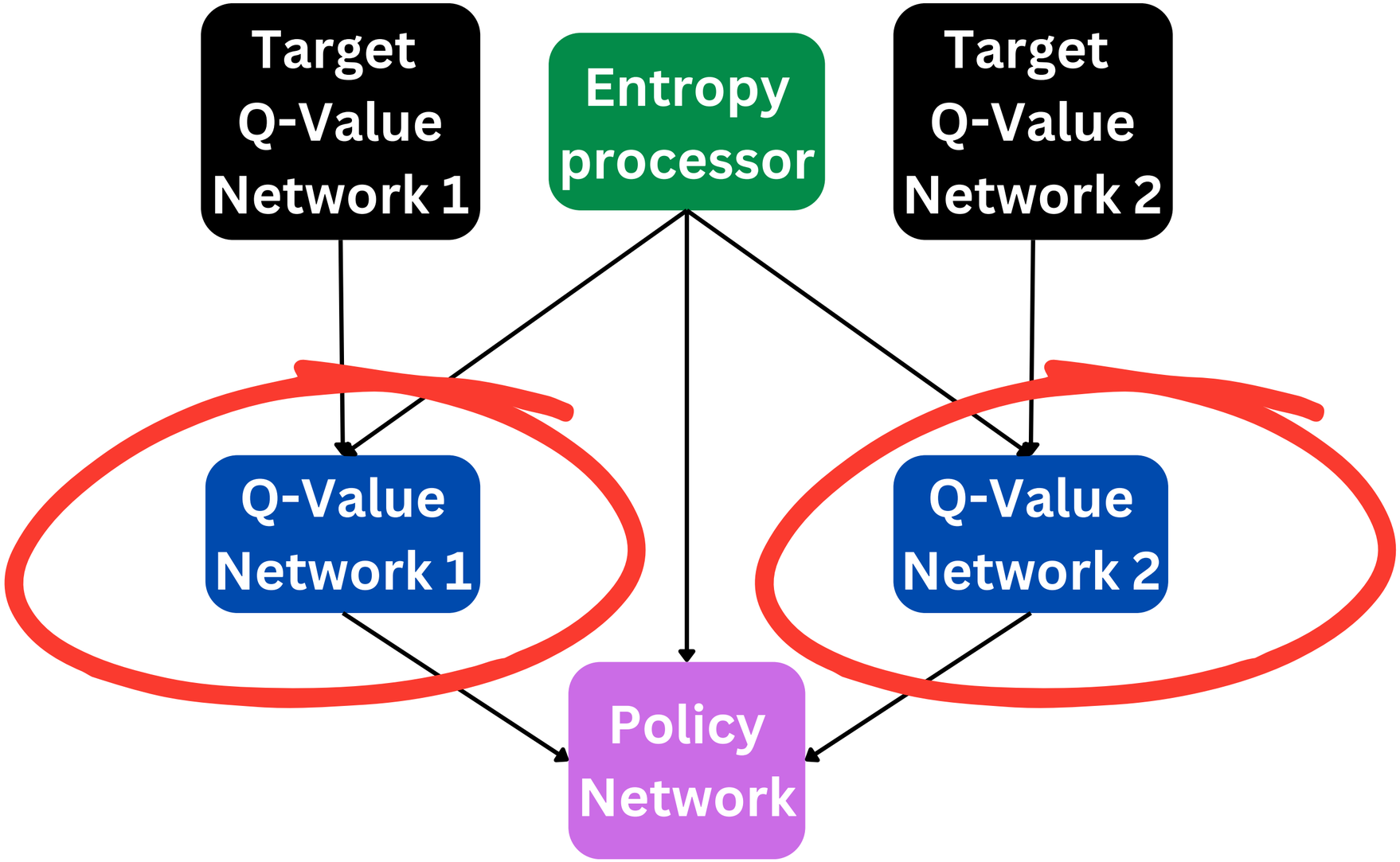
$$\theta_{target} \leftarrow \tau\theta + (1 - \tau)\theta_{target}$$

HARD-UPDATE

$$\theta_{target} = \theta$$

Q-VALUE NETWORK

BACKWARD


Target Q-Value Network 1
Entropy processor
Target Q-Value Network 2
Q-Value Network 1
Q-Value Network 2
Policy Network

BACKWARD

- Calculate target value ***y***:

$$y = r + \gamma E_{a' \sim \pi} \left[Q_{\theta_{target}}(s', a') - \alpha \log \pi(a'|s') \right]$$

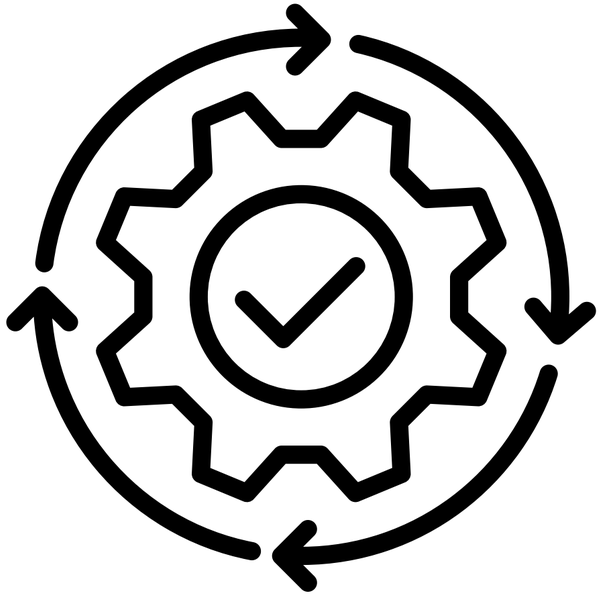
- Where:
  - r: Reward.
  - γ: Discount factor.
  - $Q_{\theta_{target}}(s', a')$: Result of target Q-Value network.
  - α: Entropy factor.

BACKWARD

- Q-value loss function:

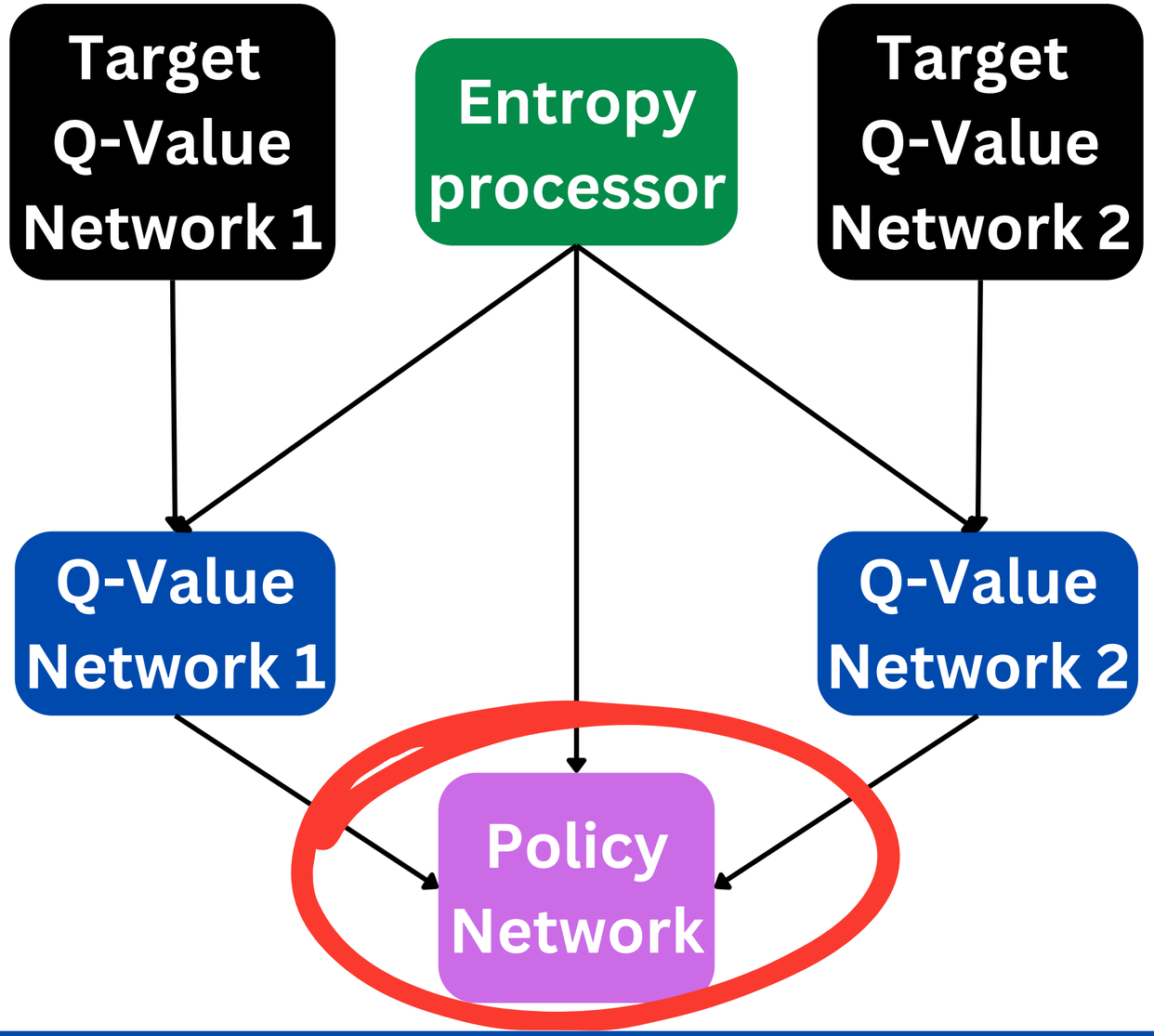
$$\mathcal{L}(\theta_Q) = E_{(s,a,r,s')\sim\mathcal{D}}\left[(Q_\theta(s,a) - y)^2\right]$$

- SAC try to minimize the value of this loss function.
  - Gradient descent



POLICY NETWORK

BACKWARD


Target Q-Value Network 1
Entropy processor
Target Q-Value Network 2
Q-Value Network 1
Q-Value Network 2
Policy Network

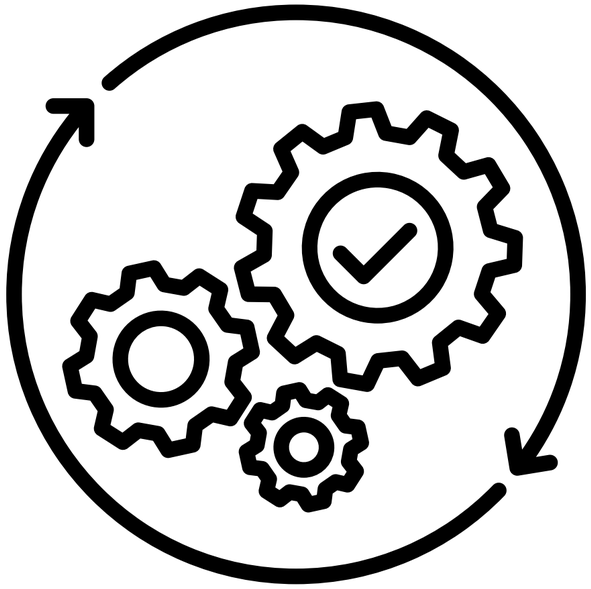
BACKWARD

- Policy network uses KL-divergence loss function:

$$\mathcal{L}(\phi) = E_{s\sim\mathcal{D}}\left[E_{a\sim\pi_\phi}\left[\alpha \log \pi_\phi(a|s) - Q(s,a)\right]\right]$$

- Where:
  - $\mathcal{D}$: Replay buffer.
  - α: Entropy factor.
  - $Q(s,a)$: Q-value of this action.

- Replay buffer is an explored state archive.



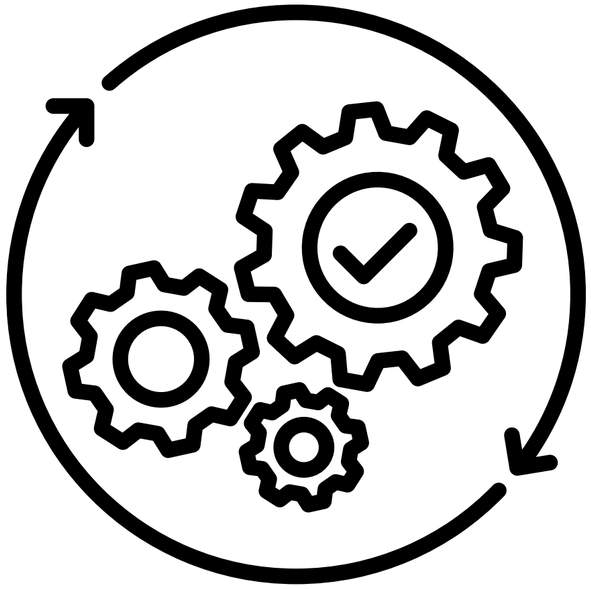
BACKWARD

- Policy network uses KL-divergence loss function:

$$\mathcal{L}(\phi) = E_{s\sim\mathcal{D}}\left[E_{a\sim\pi_\phi}\left[\alpha \log \pi_\phi(a|s) - Q(s,a)\right]\right]$$

- Where:
  - $\mathcal{D}$: Replay buffer.
  - α: Entropy factor.
  - $Q(s,a)$: Q-value of this action.

- SAC try to minimize the value of this loss function.
  - Gradient descent



THANK YOU

22520004@gm.uit.edu.vn
22520593@gm.uit.edu.vn
22520946@gm.uit.edu.vn