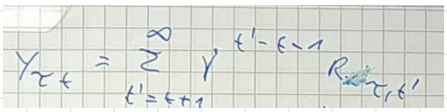
Thoughts on how to improve the Q-Learning algorithm:

Maybe also test models on a crate free, coin-poor scenario (Task 1.5 ?)

Maybe test with recreating bomb and wait to evaluate wether Q-learning can learn those two as well?

Things to maybe try out

- mc value estimation



- „offline" q-update:
  - either update q every K games, while accumulating a K-game TS
  - or update q every game with a TS going back K games

- look into other learning approaches for Q-learning regression
  - linear value approx
  -

Feature Design Task 2:

Goal: Learn to collect all coins by blowing up crates safely
- maximise coin-collecting speed
- minimise deadly accidents

Approach: Develop features that
- make it as easy as possible for the agent to learn good behaviour.
- make it able to also handle complex/edge cases well.

dangerous/safe zones

How to show info about ↓ to agent in time and space?

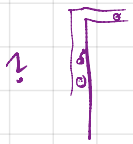
What is required for good behaviour?
- If a bomb is placed, always move to safety in time → more precisely: if a bomb is in proximity of agent (time/space-wise) move away to safe zone
- Idea: place bombs where the expected number of destroyed crates is highest or such that the crate destruction speed is maximised.
- Idea: Have a high priority on collecting accessible coins

Key tasks:
1. collect coins
2. when no coins reacheable, place bomb where many crates are destroyed.
3. when bomb placed, move to safety (and stay away from explosion while it lasts)

Idea: Learn

Pseudo-algorithm:

if not in_safety: — not in_safety = within bomb explosion radius
    move to safety
else if no_reachable_coins: — no_reachable_coins = no visible coins or coins out of reach (path blocked by crates)
    place bomb at good_spot
else: — Idea: good_spot = det. algorithmically at what spot a bomb placed results in the highest coin collecting speed
    collect coins as in Task 1



factor in: time to get to good_spot, time to place bomb and run to safety, time to return to new paths, coin density in crates, multi-bomb strategies?

expected coins = total_time to reach × coin_density × crates_destroyed (remaining_hiddencoins / remaining_crates)

Advantages:
- feature architecture with "recommended direction" can be used again, with additional feature „mode" and action „bomb" and „wait" enabled → do these actions require new features?
  5 modes, 6 actions ⇒ Q-matrix ∈ $\mathbb{R}^{48\times 6}$
- modes maybe individually trainable and probably individually analyze, debug and incentivise
- provided learning of task 1 has become ✓ good and „recommender algorithms" are good, this may actually be very easy to learn.

Disadvantages
- strategy heavily dependant on quality of algorithms
- may prevent agent from employing simultaneous or hybrid strategies *
- (quite like 2019 winner approach)
- learning not really necessary here

*Possible advanced strategies

- Ignore/Deprioritize coins that can be more easily reached by opponents
- place one bomb. Then move in a direction to safety where you (after the first one is gone) can place another bomb. Move to safety exploiting the blast of the first bomb while the second one explodes.
- When going after an accessible coin, place bombs along the way to find more ones
- when a coin is inaccessible behind a few crate, place a bomb specifically to get to that coin
- Further strategies likely exist.
- Hybrid strategies where opponents are present may become even more important.

other ideas:

- Have the features of multiple modes separate from each other
  + potentially allows for learned prioritizing between modes

  ? how to make/incentivize/inform agent when to prioritize one set of features over another?
  - big Q matrix (maybe unnecessary)
  - seems not to increase likelihood of advanced strategies

- Learn when to switch between modes
  + let the low level stuff be handled by algorithms
  + focus „thinking"/learning on deciding between high level strategies