- strategies

① + graph algo. to search board for feature shaping?

+ eute. rewards only based on earlier state + actions, not new state → easier

+ anti-symmetric rewards ✓

② • ε ↓ during training
• strong hardware (graphic card) for NN?
• 'paperspace' to train on hardware
• learn from rule-based agent
• agent-centric + overall view

③ • kamikaze mode, look eute. at scoring system ← i.g. exploit rules ☺
• symmetry
• 1-step Q-learning, store whole Q by reducing features due to symmetry (no regression) ← try n-step

• ε annealing

The current state of the game world is passed in the dictionary `game_state`. It has the following entries:

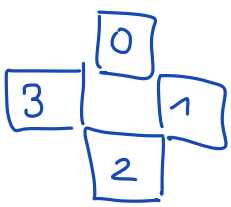
`'round': int`	The number of rounds since launching the environment, starting at 1.
`'step': int` → für modi ?	The number of steps in the episode so far, starting at 1.
`'field': np.array(width, height)` ↳ wie viel davon? + hard code valide aktionen 0	A 2D `numpy` array describing the tiles of the game board. Its entries are 1 for crates, -1 for stone walls and 0 for free tiles. Note that we use image coordinates (x, y), so `print(game_state['field'])` will show the board transposed compared to the GUI.
`'bombs': [(int, int), int)]` ↳ auf 'nur erreichbare' reduzieren?	A list of tuples $((x, y), t)$ of coordinates and countdowns for all active bombs (countdown == 0 means that the bomb is about to explode).
`'explosion_map': np.array(width, height)` → evtl. hard coded als 'no go aktion'?	A 2D `numpy` array stating, for each tile, for how many more steps an explosion will be present. Where there is no explosion, the value is 0.
`'coins': [(x, y)]` → evtl. nur nahe?	A list of coordinates (x, y) for all currently collectable coins.
`'self': (str, int, bool, (int, int))`	A tuple $(n, s, b, (x, y))$ describing your own agent. ~~n~~ is the agent's name, s its current score, b is a boolean indicating whether the `'BOMB'` action is possible (i.e. no own bomb is currently ticking), and (x, y) is the coordinate on the field.
`'others': [(str, int, bool, (int, int))]`	A list of tuples like the one above for all opponents that are still in the game.
~~`'user_input': str|None`~~	User input via the GUI. See Section 7 for details.

Feature design for Task 1:

features ≙ unabhängige Information
Welt dieser ≙ Möglichkeiten dafür

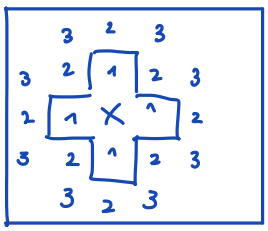
x_{ij} : $i \triangleq$ field

$j = \begin{cases} 2 & \text{free \& coin direction} \\ 1 & \text{free field} \\ 0 & \text{wall, i.g. invalid} \end{cases}$



"coin" direction:

(1) feld ist näher zum nächsten Coin



1. suche nächsten coin
$\vec{y} = \text{coins}[\underset{\text{coins}}{\text{argmin}}(d(\text{coins} - \vec{x}))]$
2. Berechne Entfernung zu Standort & umgebende Felder,

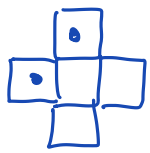
$$d(\vec{x}, \vec{y}) = \sum_{i=1}^{3} |\vec{x}_i - \vec{y}_i|$$

? → + sofern diese nicht invalide

3. vgl. & gebe Empfehlung ab

(Problem: non-reachable coins & andere schneller erreichen)

(2) Feld ist näher zum nächsten bei Bombe immer erreichbarem Coin



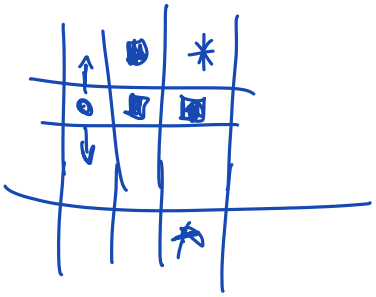
(3) Feld —"—, optimal im Wettkampf

(4) Breadth-first search ?

+ optimaler Weg
+ max. Outcome
...

(wenn komplizierter: walls vs crates)

Verbessern:
- falls non-reachable, do not try but other mode
- Umgang mit "No targets"
- mehrere gleich-nahe coins



Key Components

- Model based approach.
- Linear Regression with mini-batch SGD.
- Prioritized Experience Replay.
- Loss function: N-step TD Q-learning.
- Decision strategy: ε-greedy.
- Training: compete against rule-based agents.
- Handcrafted feature extraction function.
- Reward shaping.

N = 3, discount ca 80%;
train with forbidding invaldi actions by hard coding

Reward Shaping

When LETHAL:

- Reward movement in escape direction, else penalize agent.

When NON-LETHAL:

- If previous action == BOMB:
  - Reward if target reached, else penalize.
- If previous action == WAIT:
  - Reward if there was no better option, else penalize.
- else (previous action was any movement):
  - Prioritize: Others > Coins > Crates
  - Give reward for correctly moving towards goal, else penalize.

Note on reward balancing:

- important to penalize/reward symmetrically
- penalty_incorrect_bomb <= -4 * reward_correct_escape

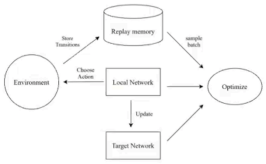
Feature Extraction

- 10 features: 2 binary indicators, 4 directions indicators.
- Based on graph-algorithms, mainly breadth-first-search for traversal to closest tiles.

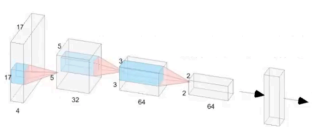
```
#   [DANGER,         ATTACK,              ESCAPE,              OTHERS,          COINS,              CRATES]
#   [lethal, target_aquired, escape_x, escape_y, other_x, other_y, coin_x, coin_y, crate_x, crate_y]
#   [     0,              1,        2,        3,       4,       5,      6,      7,       8,       9]
```

(Double) Deep Q-Learning

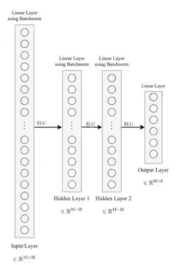
- Instead of manual feature engineering for learning via Q-Table, use Deep Q-Learning (DQL) to learn relationship between observation, action and reward
  - Implicit feature engineering
  - Optimal Q-Function is approximated by a neural network
- Drawback: DQL overestimates Q-Values
- Introduce Double DQL to avoid bias
  - Idea: separate action selection and validation
- Target network is regularly updated
- Off-Policy method allows to sample arbitrary transitions



Regression Models



Concatenate CNN output with agent-centric features to generate MLP input vector

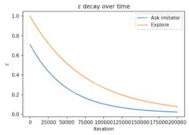


2. Approach: CNN extension

1. Approach: Multilayer Perceptron

Training Techniques

- Optimize local network for 5 epochs after each round
- Clipped gradients to [0,1]
- Sample batches from replay memory for every optimization step
- Train agent using epsilon-greedy strategy
- Imitate rule-based agent to get more meaningful episodes early on
- Implemented in Pytorch using the Cuda Toolkit



Hyperparameter	Value	Comment
History size (k)	1000	
Gamma	0.9	Discount Factor
Learning rate	0.001	
Momentum	0.9	RMSProp (see section 2.3.2 on page 13)
Batch size	128	
Epsilon decay	$0.999 \rightarrow 0.001$	Decay over time
Imitator rate decay	$0.7 \rightarrow 0.0001$	Decay over time
Target network update rate	5	Frequency of target network updates

Table 2: Hyperparameters used during training

Defined rewards

- Manual reward definition
- Oriented on "natural" rewards when playing the game
- Small negative rewards for
  - plain waiting
  - movement
  - dropping bombs
- Only hand out positive rewards sparsely
  - Force agent to be aggressive
- Two auxiliary rewards
  - Avoid Bomb
  - Run in local loop

Event	Reward
Coin collected	70
Killed opponent	500
Survived round	1
Crate destroyed	20
Coin found	15
Got killed	-80
Killed self	-150
Invalid action	-20
Waited	-5
Moving (in each direction)	-2
Bomb dropped	-7
Avoid bomb	3
Run in local loop	-10

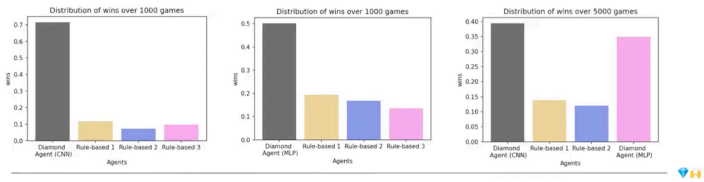
Table 1: Applied event to reward mapping

Evaluation

Two level evaluation

1. Three rule-based agents (*1000 games*)
2. Two rule-based agents + best Diamond agent so far (*5000 games*)

To determine which agent has actually won a round, we just use the maximum score in the last step of a round



Conclusion & Outlook

Persistent Problems during inference

- In exceptionally rare occasions, agent still kills itself immediately
- Agent becoming "stuck" when all crates are destroyed and nothing happens in its direct vicinity
- Too passive gameplay, generating rewards through destroying crates and collecting coins is preferred by the agent over killing opponents

Future Outlook and possible improvements

- Ideas including: normalizing rewards, trying other algorithms like REINFORCE, prioritized replay memory, new regression models (RNNs, LSTM), modifying sample probability for rare events, ...

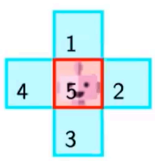
Approaches
we had in the project

- Using the python package: tensorforce
  - training a Neural Network (DQN)
  - did not work (nor training progress)
- most simple approach: Q-learning with own feature reduction
  - extract a few hand-crafted features
  - calculate the Q-values in a Q-table

6 features

$$s_{i'} \mapsto \Psi(s_{i'}) = \vec{x}_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \\ x_{i5} \\ x_{i6} \end{pmatrix}$$

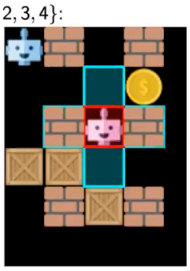
- $x_{i1}, \ldots, x_{i4}$: information on the neighbouring fields or direction
- x_{i5}: information on the current field
- x_{i6}: "game mode"



Neighbouring Fields or Direction

Neighbouring field features x_{ij} for $j \in \{1, 2, 3, 4\}$:

$$x_{ij} = \begin{cases} 0 & \text{if the corresponding adjacent field } j \text{ is free} \\ 1 & \text{depends on the mode value } x_{i6} \text{ (see later)} \\ 2 & \text{if the field } j \text{ is a wall, bomb, opponent, crate or yields safe death} \\ 3 & \text{if the field } j \text{ possibly yields danger} \end{cases}$$



Current Field

Current field feature x_{i5}:

$$x_{i5} = \begin{cases} 0 & \text{if the agent would not die if he would wait and can not place a bomb or placing a bomb would lead to his safe death or not destroy a crate.} \\ 1 & \text{if a bomb placement would not lead to own safe death and would destroy at least 1 crate/opponent} \\ 2 & \text{if a bomb placement would not lead to own safe death and would destroy at least 3 crates/opponents} \\ 3 & \text{if a bomb placement would not lead to own safe death and would destroy at least 6 crates/opponents} \\ 4 & \text{if a bomb is on the current tile or if waiting would lead to own safe death} \end{cases}$$

+ genug „stagische" Informationen?
evtl. weitsichtige re hin zu fügen?

+ schlaue Handlungsstrategien durch Reward shaping

Game Mode

Mode feature x_{i6}:

Problem: nicht immer collecten sinnvoll, wenn # coins > 0

$$x_{i6} = \begin{cases} 0 & \text{if the number of visible coins is greater than 0.} \\ & (\equiv \text{“coin collecting mode”}) \\ 1 & \text{if the number of visible coins is 0} \\ & (\equiv \text{“bombing/crate destroying mode”}) \\ 2 & \text{if there is no collectable coin (all 9 coins are collected)} \\ & \text{and if the number of opponents is not 0} \\ & (\equiv \text{“terminator mode”}) \end{cases}$$

Addendum Neighbouring Field and Game Mode

Neighbouring field feature $x_{ij} = 1$ for $j \in \{1, 2, 3, 4\}$:

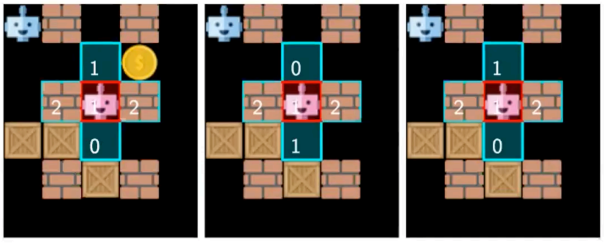
1. necessary condition: field is empty (or coin), no possible danger nor safe death
2. mode-specific condition:

Coin collecting mode field is nearer to the next nearest coin

Bombing/Crate destroying mode a bomb planted this tile would destroy more crates/opponents compared to the current tile and the other neighbouring tiles or if there is no 1 in the neigbouring tiles but 0s then the feature is 1 if the tile is nearer to the next nearest crate.

Terminator mode field is nearer to the next opponent but not within their bomb spread

Examples



(a) Coin collecting mode $x_{i6} = 0$ (b) Crate destroying mode $x_{i6} = 1$ (c) Terminator mode $x_{i6} = 2$

Figure: Feature values for x_{ij} for $j = 1, \ldots, 5$ for different game modes

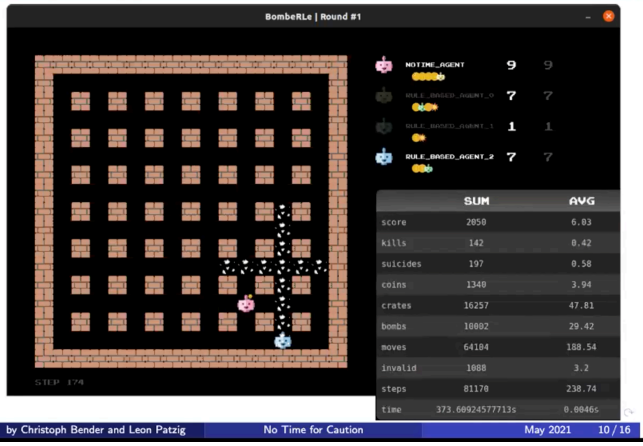
Kamikaze Feature

$$\vec{x_i} = (2, 2, 2, 2, 2, 2)^T$$

condition all coins are collected and agent is at the top of the leader board and no other opponent can catch up with us if our agent kills himself and if he will not die by another bomb on the current tile

until the end positive reward for bombing and a negative reward for all other moves or waits ($\rightarrow$ he will bomb himself up + will safe win the round + destroying the chance of the opponents to win the round)

Kamikaze Example



Using Rotation and Mirror Invariance

- number of possible states:

$$n = |\{x_{i1}\}_i| \cdot |\{x_{i2}\}_i| \cdot |\{x_{i3}\}_i| \cdot |\{x_{i4}\}_i| \cdot |\{x_{i5}\}_i| \cdot |\{x_{i6}\}_i| = 4^4 \cdot 5 \cdot 3 = 3840$$

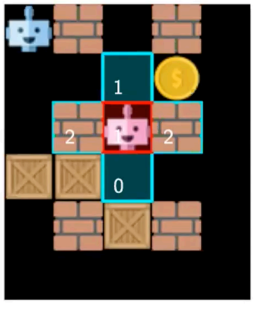
- exploit rotation and mirror invariance where possible: replaces $|\{x_{i1}\}_i| \cdot |\{x_{i2}\}_i| \cdot |\{x_{i3}\}_i| \cdot |\{x_{i4}\}_i|$ with $a(m)$
- final actual number of non-equivalent states = entries in the Q-table:

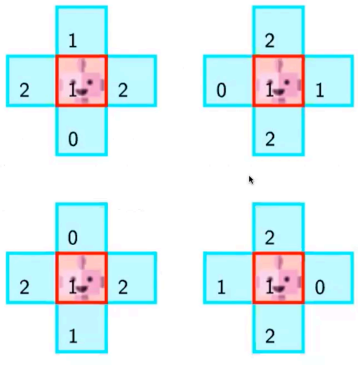
$$\tilde{n} = a(4) \cdot 5 \cdot 3 = 825$$

Actual number of nonequivalent states for the first 4 features

$$a(m) = 3 \cdot \binom{m+2}{4} + \binom{m+1}{2} = m \cdot (m+1) \cdot \frac{m^2 + m + 2}{8}.$$

See https://oeis.org/A002817.





General Tipps:

+ find your own way, there is no one approach to this the 'right way'
+ try to be good but rather learn & implement well-thoughtey interesting ideas for grade

- simple but carefull design

Things to keep in mind

- < 0,5s action
- upload be 4 deadline ± join on MampF
- use ML :-
- at least 2 models, with one limited to techniques of lecture
- plan in Bugs to fix

Umsetzungs-Plan Idee 6.03.2022

2 Agents: 1 lecture-only techniques!

- automatic vs. hand-designed feature design
  - automatic: regression forests, clustering, PCA, ... vgl. lecture
  - hand-designed: last year inspiration
- evtl. model-based vs. -free

train

- model based vs model free? compare ✓, but how?
  - train or hard-code prob?
  - problem: learn only rule-based actions? evtl. nur coins seeking

- TD $\underline{n}$? - step Q-learning vs SARSA vs (linear regr. +) gradient descent on loss fct (← which ?) compare (3)

- feature engineering - symmetries
  1. schau dir obs. an
  2. lasse dich von last year inspiration
  3. regr. forrest, clustering, ...

- reward shaping introduce auxiliary rewards & ○ potential fct & theory ?

- hyperparameter optimization $\varepsilon, \alpha, \gamma$

- custom environment = evtl. modify framework for training
  ○ training settings für subtasks coden
- create own events

game modes depending on # players ?

- train model parameters
- long enough training to make sure agents knows all situations (important!)

- Q: basierend auf rule-based-imitation?
- train agent for competition (ig)
  - create training data using rule-based agent
  - evtl. self-play training
  - evtl. download other teams agent to train against.
- evtl. train with last time winners :-) ?

act

- make sure agent reacts fast enough

Learn from last year: - very hard:

- decide to go to right direction if multiple bombs placed + care about bomb behind crates
- no suicide, e.g. kill other agents with bomb
- no illegal moves

- ϵ-greedy vs softmax + hyperpar. opt.

- spiel-modi: evtl. player modus basierend auf task 1-4 + hard-code switch + später auch noch trainieren?

- feature design welches nur für training gebraucht auch nur dort einfügen

Roadmap:

4 tasks

1. On a game board without any crates or opponents, collect a number of revealed coins as quickly as possible. This task does not require dropping any bombs. The agent should learn how to navigate the board efficiently.

2. On a game board with randomly placed crates yet without opponents, find all hidden coins and collect them within the step limit. The agent must drop bombs to destroy the crates. It should learn how to use bombs without killing itself, while not forgetting efficient navigation.

3. On a game board with crates, try to hunt and blow up our predefined `peaceful_agent` (easy) and the `coin_collector_agent` (hard). The former agent does not drop bombs and moves randomly. The latter agent will drop bombs only for the purpose of collecting coins.

4. On a game board with crates, hold your own against one or more opposing agents (e.g. the full-strength `rule_based_agent`) and fight for the highest score.

To dos:

- study & get inspiration from more RL or internet
- customize agent + team name
- submit test by given times

- A directory containing the agent code of your best performing model, including all trained parameters. This is the subdirectory of `agent_code` that you are developing your agent in (see details below).

Report

Your second submission (report) for the final project should consist of the following:

- A PDF report of your approach. The report should comprise at least 10 pages per team member (and please not much more) and – for legal reasons – indicate after headings who is responsible for each subsection.
- The URL of a public repository containing your entire code base (including all models you developed), which must be mentioned in the report. Please **do not upload your report** to this repository.

- indicate additional libraries clearly at begin of report

- evtl private GitHub

→ report + agent + repository

- 3er Gruppe ??

Zeitplan - Idee:

- Plan in Bugs to fix!
- Lang enough training to know whole world