

- strategies

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

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

+ anti-symmetric rewards ✓

- ② • $\epsilon \downarrow$ during training

• strong hardware (graphic card) for NN?

• 'paperspace' to train on hardware

• learn from rule-based agent

• agent-centric + overall view

- ③

• Kamikaze Model, look evte. 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
'step': int → für Modi ?
'field': np.array(width, height)
↳ wie viel davon?
+ hard code valide
  aktionen O
'bombs': [(int, int), int]
↳ auf 'hur erreichbare'
  reduzieren?
'explosion_map': np.array(width,
height) → evtl. hard coded
  als 'no go action'?
'coins': [(x, y)] → evtl. hur habe?
'self': (str, int, bool, (int,
int))
'others': [(str, int, bool, (int,
int))]
'user_input': str|None

```

The number of rounds since launching the environment, starting at 1.

The number of steps in the episode so far, starting at 1.

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.

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).

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.

A list of coordinates (x, y) for all currently collectable coins.

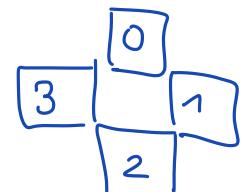
A tuple $(n, s, b, (x, y))$ describing your own agent. n is the agent's name, s its current score, b is a boolean indicating whether the 'BOMB' action is possible (i.e. no own bomb is currently ticking), and (x, y) is the coordinate on the field. A list of tuples like the one above for all opponents that are still in the game.

User input via the GUI. See Section 7 for details.

Feature design for Task 1 :

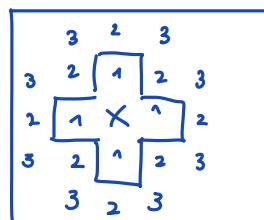
features = unabhängige Information
Welt dieser = Möglichkeiten dafür

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



"coin" direction:

① feld ist näher zum nächsten Coin



1. Suche nächsten Coin

$$\vec{i} = \text{coins}[\arg\min_{\text{coins}} (\vec{x} - \vec{i})]$$

2. Berechne Entfernung zu Standort & umgebende Felder,

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

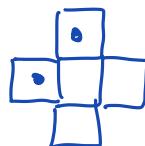
? → + sofern diese nicht invalide

3. vgl. & gebe Empfehlung
ab

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

② Feld ist näher zum nächsten erreichbaren Coin
ber Bombe immer

x .



③ Feld -||-, optimal im Wettkampf

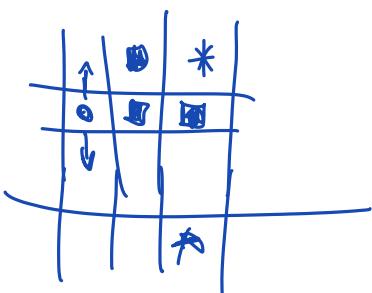
④ Breadth-first search ? + optimierter Weg
+ max. outcome

(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.

?

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
- $\text{penalty_incorrect_bomb} \leq -4 * \text{reward_correct_escape}$

Feature Extraction

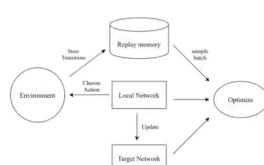
- 10 features: 2 binary indicators, 4 directions indicators.
- Based on graph-algorithms, mainly breadth-first-search for traversal to closest tiles.

```
# [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]
```

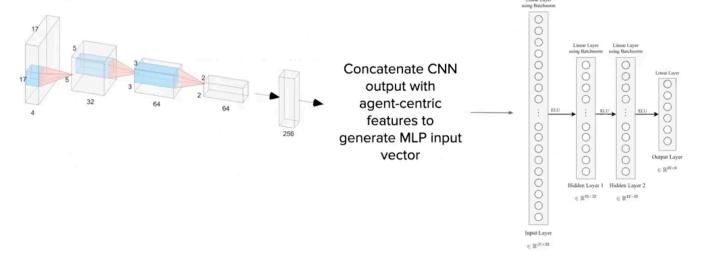


(Double) Deep Q-Learning

- 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

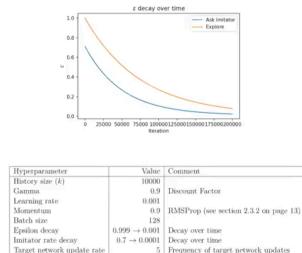


2. Approach: CNN extension

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

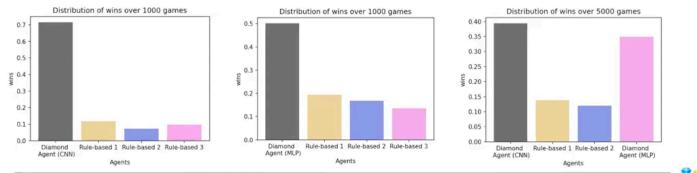


Evaluation

Two level evaluation

- Three rule-based agents (1000 games)
- 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



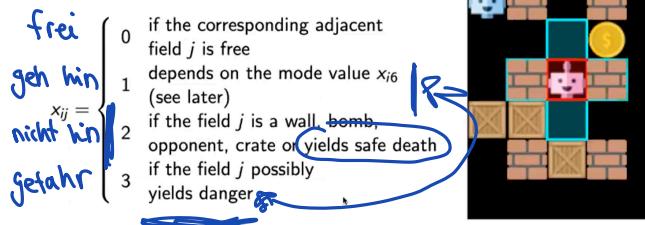
Approaches

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

Neighbouring Fields or Direction

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



+ genug „statische“ Informationen?
evtl. weitsichtige re hin zufügen?

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

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

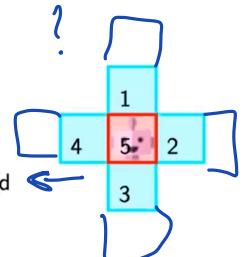
Future Outlook and possible improvements

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

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



Current Field

Current field feature x_{i5} :

- $x_{i5} =$
- 0 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 if a bomb placement would not lead to own safe death and would destroy at least 1 crate/opponent
 - 2 if a bomb placement would not lead to own safe death and would destroy at least 3 crates/opponents
 - 3 if a bomb placement would not lead to own safe death and would destroy at least 6 crates/opponents
 - 4 if a bomb is on the current tile or if waiting would lead to own safe death

"kann warten, soll keine Bombe"

Bombe macht immer mehr Sinn

weg hier

+ schlaue Handlungsstrategien durch Reward shaping

Game Mode

Addendum Neighbouring Field and Game Mode

Mode feature x_{i6} :

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

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

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

- necessary condition: field is empty (or coin), no possible danger nor safe death
- mode-specific condition:
 - Coin collecting mode field is nearer to the next nearest coin
 - Bombing/Create 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 neighbouring 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

↳ o parallel oder exclusive modes ? + danger vs rest

Examples

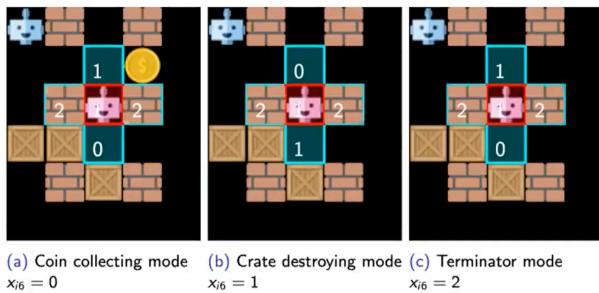


Figure: Feature values for x_{ij} for $j = 1, \dots, 5$ for different game modes

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)

Using Rotation and Mirror Invariance

- number of possible states:

$$n = |\{x_1\}_i| \cdot |\{x_2\}_i| \cdot |\{x_3\}_i| \cdot |\{x_4\}_i| \cdot |\{x_5\}_i| \cdot |\{x_6\}_i| = 4^4 \cdot 5 \cdot 3 = 3840$$

- exploit rotation and mirror invariance where possible: replaces $|\{x_1\}_i| \cdot |\{x_2\}_i| \cdot |\{x_3\}_i| \cdot |\{x_4\}_i|$ with $a(m)$

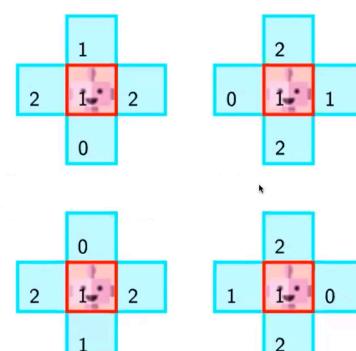
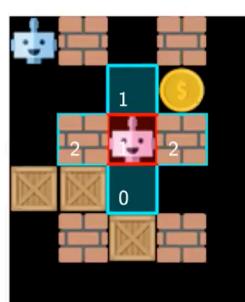
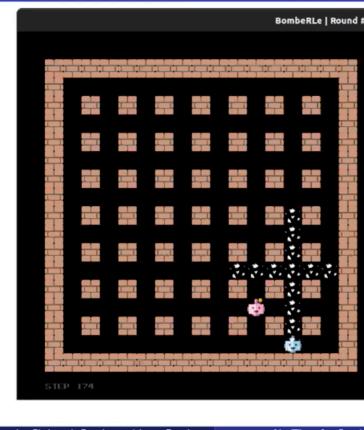
• final actual number of non-equivalent states = entries in the Q-table:

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

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>.



① \exists :

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>.

② Finde Q-table, welches Symmetrien ausnutzt, d.h. $\hat{a} = Q(s) = \operatorname{argmax}_a Q_{\{i(s), a\}}$

	$x_1=0, x_2=0$	$x_1=0, x_2=1$	$x_1=1, x_2=0$	$x_1=1, x_2=1$
a_1				
a_2				



A

2 Möglichkeiten: Symmetrie bei feature design

Für Form lernen da oder erst bei Q berücksichtigen
Sonst schon act
(O_{SS} Problem)

B

Birk's Idee: Q mit $\# \dim \hat{=} \text{features} + a$
& Eintrag $\hat{=} \text{Feature Wert}$
↳ am einfachsten..

→ wie wäre mit z.B. $n = 1$:

$\# \dim \hat{=} \text{'features ohne Symmetrie'} + a + n$

& eindeutigen (surjektiv & injektiv 'außer Symmetrien') code

$$C: \{x_1, x_2, x_3, x_4\} \rightarrow \{0, 1, \dots, a(n)-1\}$$

Wobei $x_i \in X^3$ mit X beliebig \uparrow $i \in \{1, \dots, 4\}$
köönnte n oder \exists, \dots

Oder: Über Datenstruktur automatisch, z.B.
 da Reihenfolge egal
 \uparrow
 fürs Suchen ja,
 aber fürs Agieren nicht

$$\begin{aligned} \text{i.A. } f &= \text{feature_design (game_state)} \\ \tilde{f} &= T(f) \quad \text{Symmetrie - Trafo} \\ \tilde{a} &= \underset{a'}{\operatorname{argmax}} Q(\tilde{f}, a') \\ a &= D(\tilde{a}) \quad \text{Symmetrie - Detrafo.} \end{aligned}$$

ZV kleiner: T, Q, D , wobei feature_design
 gewisse Variabilität

$$\hookrightarrow \# Q = \frac{n^4}{4!} \cdot 3 \cdot 5$$

Bsp.: Reihenfolge egal

$$\begin{aligned} T(f) &= \text{sort}(f) = \tilde{f} \quad \text{mit ind_sort}(f) = \\ &\quad (i_1, i_2, i_3, i_4) \xrightarrow{} (0, 1, 2, 3) \\ \tilde{a} &= Q(\tilde{f}) \in \{0, 1, 2, 3\} * \\ D(\tilde{a}) &= f [\text{ind_sort}(f)[a]] \end{aligned}$$

?

0		00	11	22		000	011	111	222	?
		01	12			001	012	112		
		02				002	022	122		

Vergleiche exp. zu Rotation-/ Spiegelsymmetrie



General Tipps:

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

Things to keep in mind

- < 0,5s action
- upload by 4 deadline \pm join on MAMPF
- use ML \subset
- 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
↑
Regression forests,
clustering, IAA,...
vgl. Lecture
- ↑
 best year inspiration

- evtr. model-based vs. - free

- invalid actions hard-code vs.
nicht

train

- Model based vs model free ? compare ①,
but how ?



train or hard-code prob ?

- problem: learn only rule-based actions?
evtl. nur coins seeking

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

- feature engineering
 - 1. Schau dir obs. an
 - 2. lasse dich von east year inspiration
 - 3. regr. forest, clustering, ...
- symmetries

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

- hyper parameter optimization ϵ, α, γ

- custom environment = ext. modity framework for
training

○ training settings for subtasks coden

- create own events

game nodes depending on # players?

- ϵ -annealing + invalid & BS moves ver-
hindern?
↳ invalid configurations

- from model parameters

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

- train over episode = 10 games to learn long term behaviour
- Q⁰ basierend auf rule-based-initiation?
- 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 := ?

= bisher: ϵ -greedy nur aus a aktionen beschränkt

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: evte. 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 do's:

- Study & get inspiration from more RL on internet
- customize agent + team name
- submit test b4 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

○ Zer Gruppe ?

Testplan - Idee:

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

- viel trainieren
 - + davor jetzige Probleme (Algorithm Bias) lösen
 - + Training Analyse lernen: Q - Konvergenz? Coin-Collecting speed (e.g. coins / step)
 - + Q-Matrizen speichern
 - + Simulation enden sobald alle coins weg
 - * Ugl. mit "Optimum": Coin-Collector Agent
 - + invalide movements:
 - hard-coden oder lassen
 - ↑
 - analyseren
 - ↑
 - auch ϵ -greedy:
 - nur valid movements
 - + $Q^{(0)} = 0$
 - (+ ϵ -annealing)

- Auxiliary Rewards : für valide Aktionen & Coins +
 - Symmetrie Problem
-

- feature design: mehrere coins anzeigen
- Hyperparameter optimieren : $\gamma \downarrow$

$$\text{Probs} = 0$$

$$\alpha < 1$$

ϵ -annealing

- Potential fct for reward shaping
- evtl n-step learning