Outplaying humans in bomberman using Deep Q-learning

AML project presentation by Frank de Morrée,
Ida Stoustrup and Iestyn Watkin.

12th of June 2019

Outline

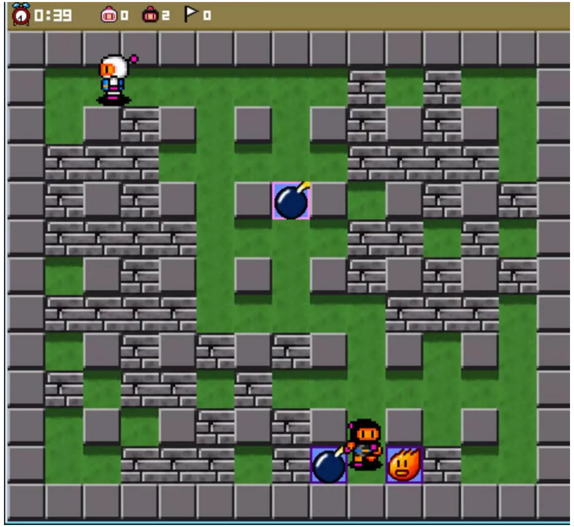
The Problem

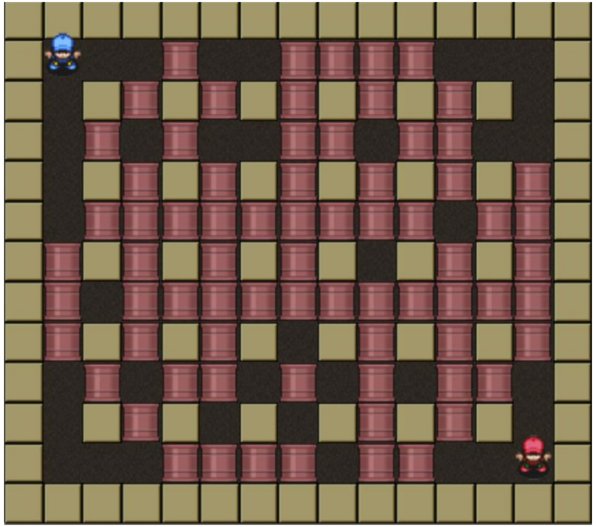
Train an ANN agent to play Bomberman successfully

- ➔ Decided to build lightweight version of the game
- ➔ Train using Deep Q-Reinforcement-Learning
- ➔ Evaluate and tweak until our own skills are laughable

Bomberman

“2D checkerboard space with N players whose goal is to blow up enemy players with bombs while avoiding them. The last man standing wins. Bricks may block paths and contain powerups. Originally 1P”



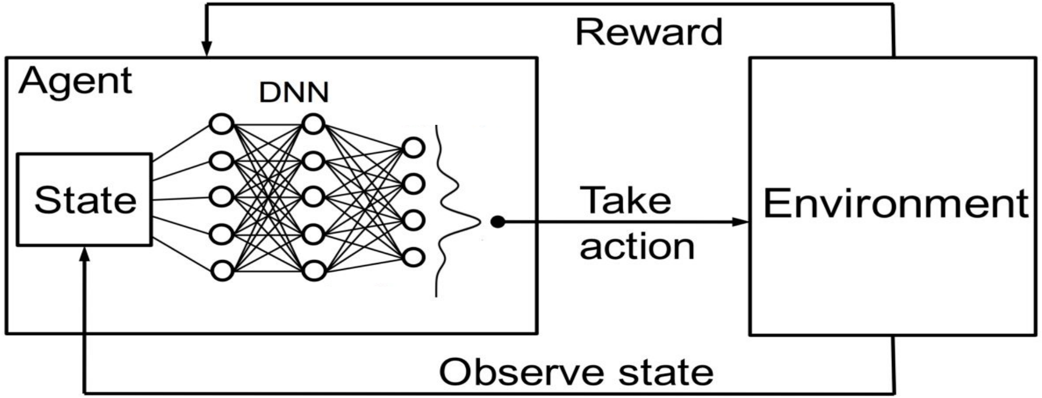


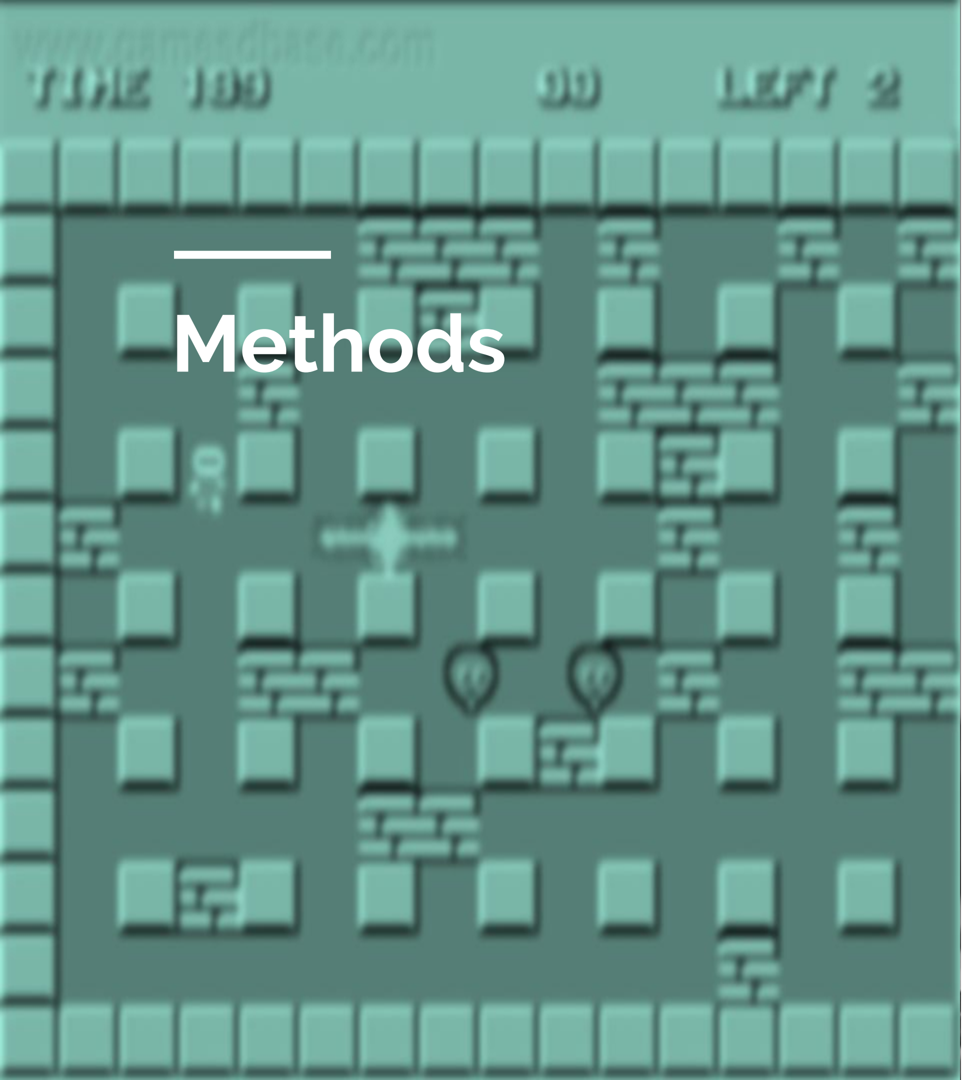
Game Development Phase

- Player class
  - Full moveset + scoring system
- Board class
  - Dynamic board size: walls, bricks, players, spawns and bombs
- Bomb class
  - Timer updates + explosion behaviour
- Result is an N x M x 6 tensor that is the abstract game

Deep Q-learning

"Program AI agents to act effectively in an discrete action environment. The input is a state, after which the agent chooses an action based on weights. Subsequently, the action is rewarded or punished, the weights updated, and the state of the game updated and given to the agent."





Methods

Let’s go in depth, easy as ABC

A. Network architecture

B. Q-function & reward values

C. Training

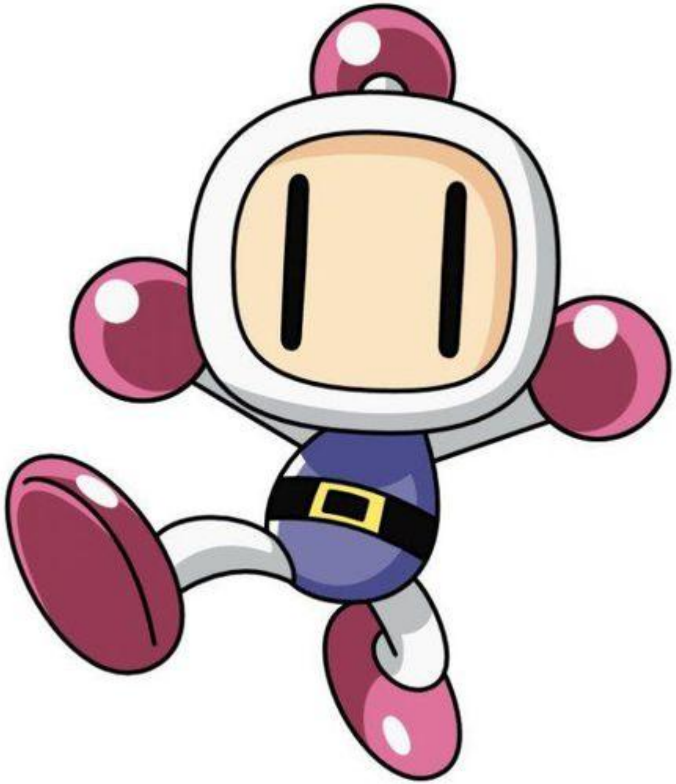
Meet A.L.A.N.

Advanced Learning Artificial Network

Absolutely Lit Axploding Neurons

Anachronistic Liability Adulterating Net

Angry Lemony Ankle-biting Nonsense



Network Architecture A.

Conv2D → Maxpool → Conv2D → Fully connected

5x5 2x2 3x3 2x

A.L.A.N.'s Input

- **Input** is tensor board N x M x 6
- The full board state consisting of 6 object layers
  - Walls
  - Bricks
  - Players
  - Bombs
  - Enemies
  - Powerups

Network Architecture
A.

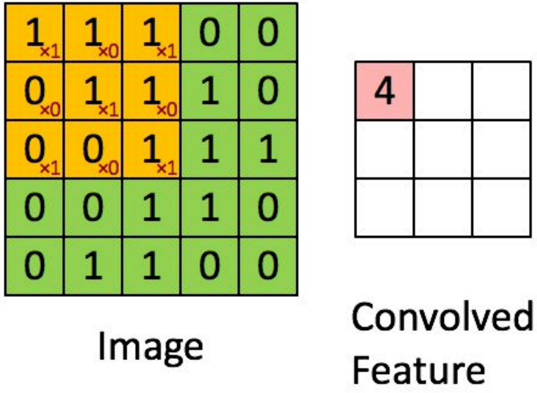
Conv2D → Maxpool → Conv2D → Fully connected
5x5 2x2 3x3 2x

A.L.A.N.'s first convolutional layer

➔ **Conv2D** identifies features

➔ Scans input and convolves

➔ Padding



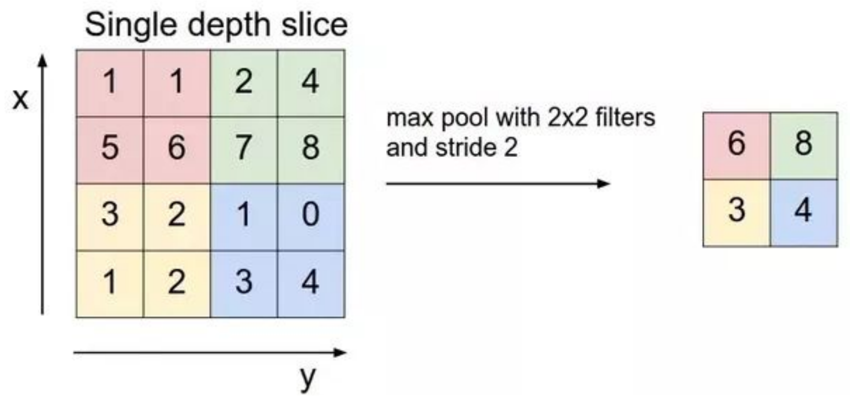
Network Architecture A.

Conv2D → **Maxpool** → Conv2D → Fully connected

5x5 2x2 3x3 2x

A.L.A.N.'s Maxpool layer

➔ **Maxpool** identifies the highest valued features, reducing dimensionality



Network Architecture A.

Conv2D → Maxpool → **Conv2D** → Fully connected

5x5 2x2 3x3 2x

A.L.A.N.'s second convolutional layer

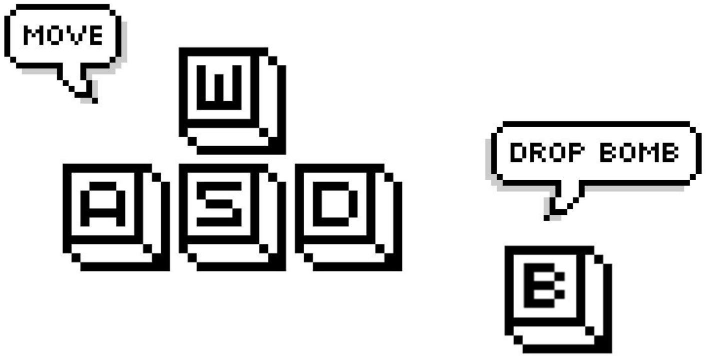
- ➔ **2nd Conv2D**
- ➔ Used to find right representation of the game state

Network Architecture
A.

Conv2D →	Maxpool →	Conv2D →	**Fully connected**
5x5	2x2	3x3	2x

A.L.A.N.'s brain: neurons and output

➔ **Fully connected** layers provides output moves.



Deep Q-Learning

B.

The Q-function tells the agent the quality of a possible action **a** in a particular state **s**

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

$$\text{New } Q(s,a) = Q(s,a) + \alpha\,[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)]$$

Game Board:

Current state (s): 000 010

Q Table: $\gamma = 0.95$

	000 100	000 010	000 001	100 000	010 000	001 000
⇧	0.2	0.3	1.0	-0.22	-0.3	0.0
⇩	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
⇨	0.21	0.4	-0.3	0.5	1.0	0.0
⇦	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Deep Q-Learning

B.

The reward function defines the reward maximising behaviour of Alan

Reinforcement learning rewards

- ➔ Reward shaping
- ➔ Robot arm

```
self.rewards = {
            "kaboomed_brick":       50,
            "kaboomed_player":      200,
            "lost_life":            -45,
            "died":                 0,
            "invalid_move":         -5,
            "spawned_bomb":         -1,
            "do_nothing":           -3,
            "valid_move":           -1,
            "invalid_spawn_bomb":   -5 }
```

- ➔ Exploration vs exploitation

Deep Q-Learning

B.

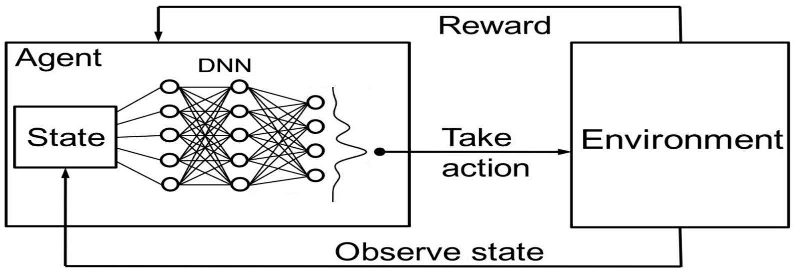
The replay memory class stabilises the learning procedure and removes correlation

Replay memory

➔ Tuples of [State, Action, Observe state, Reward]

➔ Batch of 16 states

➔ Random update removes



Training A.L.A.N
C.

The work before starting the training process summed up

Setup

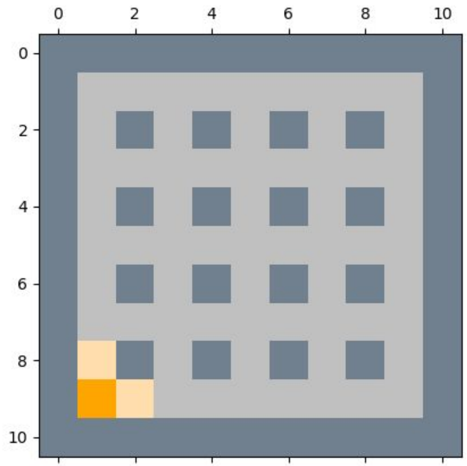
- ➔ A.L.A.N. has moveset
- ➔ A.L.A.N. gets rewards
- ➔ A.L.A.N. maximises rewards
- ➔ Loss function is to be minimised

Training Alan C.

Stage 1 - Teenager in room

Process

- ➔ Initial guess at rewards
- ➔ Solved by tweaking rewards and hyperparameters

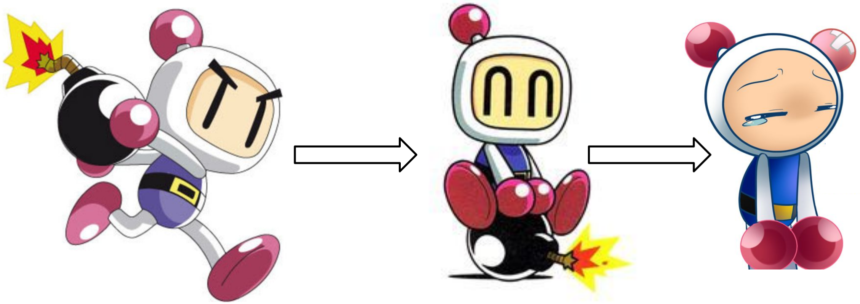


Training
C.

Stage 2 - A.L.A.N. gets PTSD

Process

- ➔ Many negative rewards?
- ➔ After loss of live stopped all movement
- ➔ Solved by tweaking hyperparameters
  - ◆ Less punishment



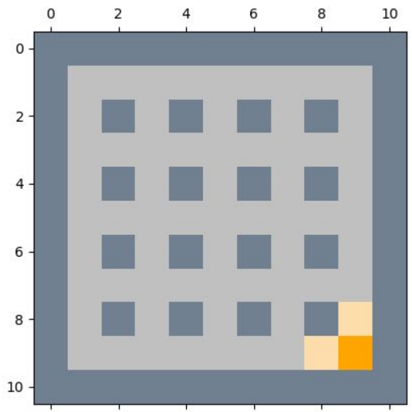
Training

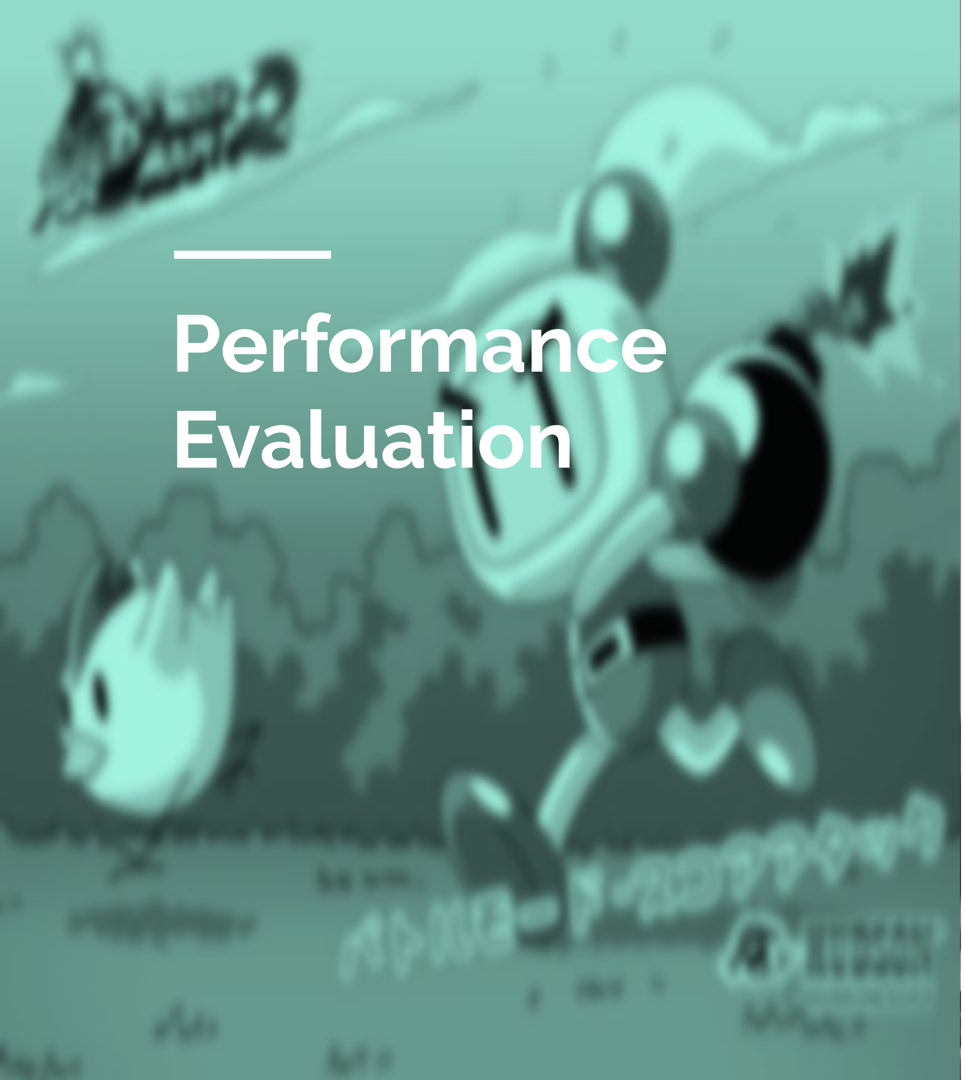
C.

Stage 3 - Lazy Alan

Process

- ➔ Rewarding valid moves?
- ➔ Repeat moves forever
- ➔ Solved by more training and/or tweaking rewards



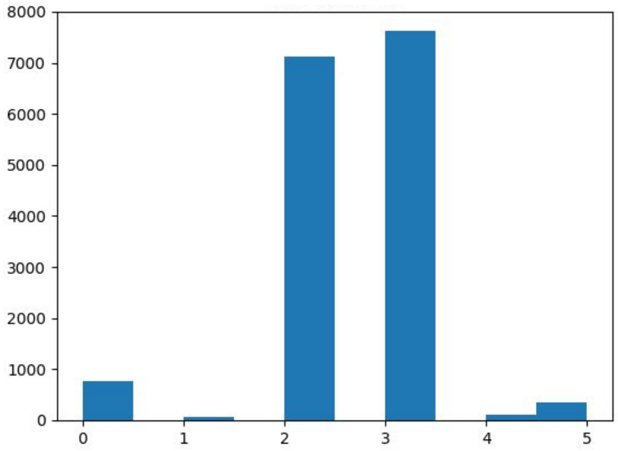


Performance Evaluation

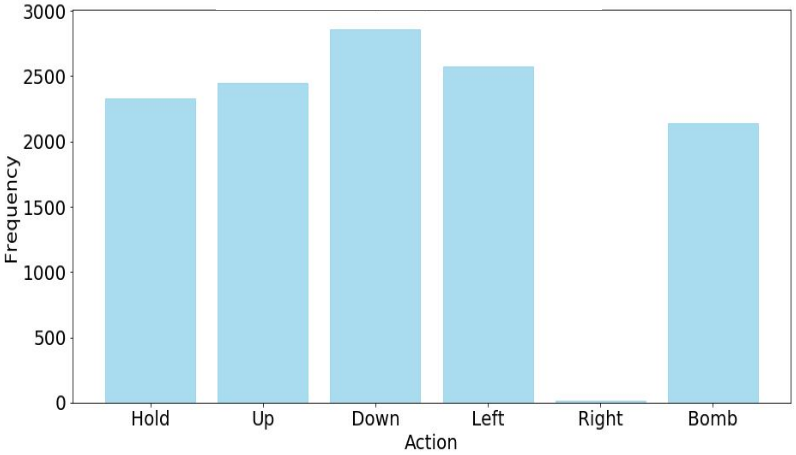
So how did we improve?

- ➔ Alan’s performance is evaluated based on score
- ➔ Not dying
- ➔ Successfully blowing up bricks
- ➔ Gaining as much points as possible

Moves

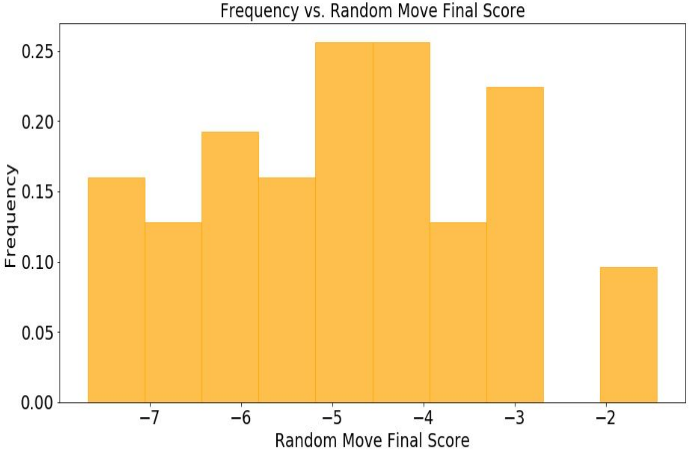


Earlier version ‘motion loop’

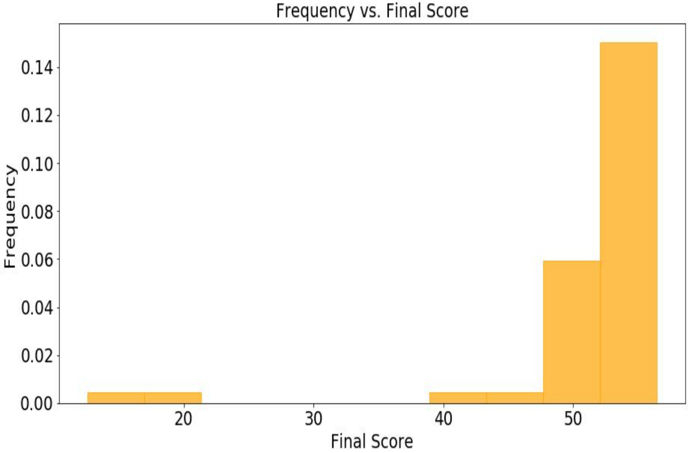


Final version applies all moves

Scoring

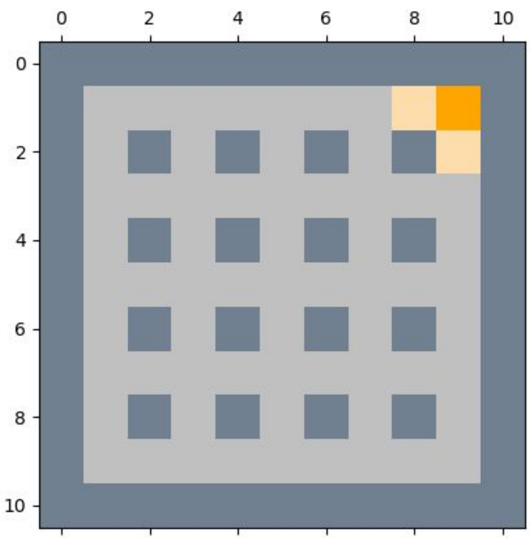


Random moves = bad score

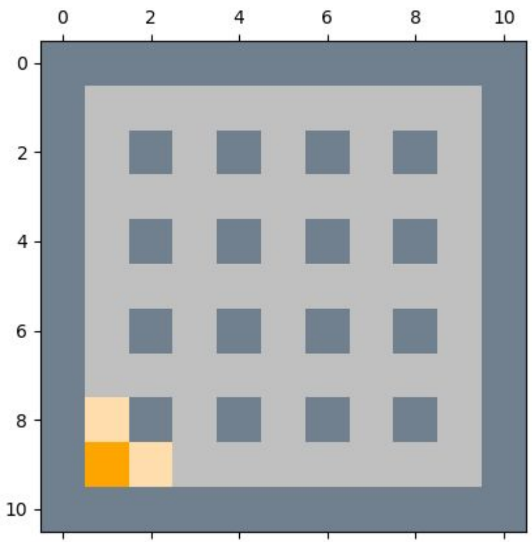


Play from memory = high score

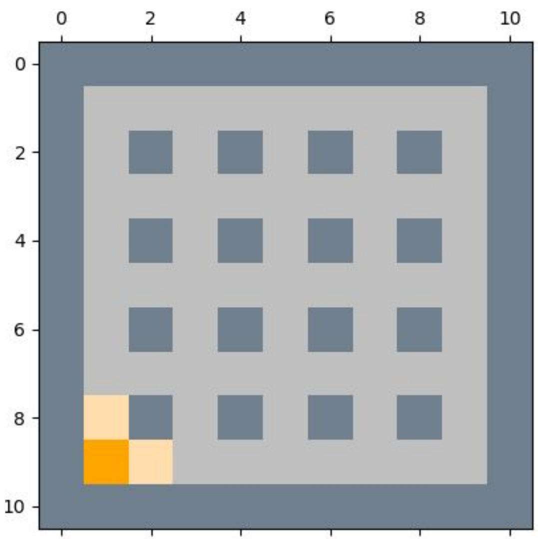
A well trained model is consistent.



Game 11



Game 20



Game 51

Discussion and Conclusions

What did we learn and what's next?

Deep Q-Reinforcement-Learning of Alan the Bomberman ANN

There are still some difficulties but the behaviour is according to plan!

Improvements and lessons learned

- ➔ Thesis sized project = sleepless nights
- ➔ Hyperparameters
  - Trial and error and many configurations work
- ➔ Alan loves loopholes
  - Infini bomb drop
  - Suicidal to avoid further punishment
- ➔ More training is more better
  - Validates the agent and

Support our team!

DISCOUNT!

$5,99

with

PROMO CODE

A.L.A.N.OPE

References

[Reinforcement Learning with Pytorch]

[Human-level control through deep reinforcement learning]

[Deep Reinforcement Learning Course]

All group members have contributed evenly to the project, but Ida deserves extra credit for being awesome and managing to figure out a bunch of both the game and the ANN.