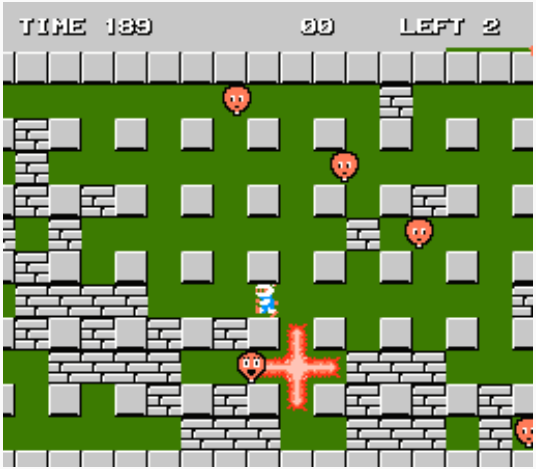
Developing a Successful Bomberman Agent

Dominik Kowalczyk, Jakub Kowalski, Hubert Obrzut, Michał Maras, Szymon Kosakowski, **Radosław Miernik**

5 February 2021

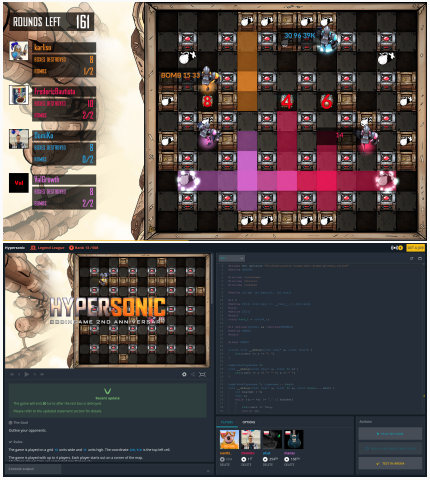
Motivation

Bomberman (Hudson Soft, 1983) is a popular, arcade-style game.



Test bed

Hypersonic is an adaptation of Bomberman as a programming game available on CodinGame[1].

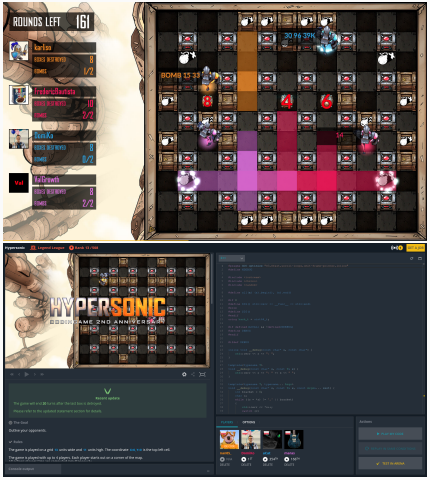


[1] https://www.codingame.com/multiplayer/bot-programming/hypersonic

Test bed

Hypersonic is an adaptation of Bomberman as a programming game available on CodinGame[1].

During the competition held in September 2016, it attracted over 2700 participants. When it concluded, it became one of the multiplayer games available on the platform. At the moment there are more than 2300 agents in the public arena.



[1] https://www.codingame.com/multiplayer/bot-programming/hypersonic

Test bed

The game is played by 2 to 4 players on a 13x11 grid. Grid cells may be floors (passable), walls, or boxes (impassable). Walls' positions are always the same (every odd cell in both directions). Only boxes can be destroyed (with bombs).

Test bed

The game is played by 2 to 4 players on a 13x11 grid. Grid cells may be floors (passable), walls, or boxes (impassable). Walls' positions are always the same (every odd cell in both directions). Only boxes can be destroyed (with bombs).

Players start in corners. Each turn a player may place a bomb (up to a limit) and move to the nearest cell (if it is free), thus there are at most 10 actions. Multiple players can occupy the same cell.

Test bed

The game is played by 2 to 4 players on a 13x11 grid. Grid cells may be floors (passable), walls, or boxes (impassable). Walls' positions are always the same (every odd cell in both directions). Only boxes can be destroyed (with bombs).

Players start in corners. Each turn a player may place a bomb (up to a limit) and move to the nearest cell (if it is free), thus there are at most 10 actions. Multiple players can occupy the same cell.

Bombs explode after 8 turns, creating a blast in both cardinal directions up to *range* cells. If it reaches another bomb, it causes a chain reaction.

Test bed

The game is played by 2 to 4 players on a 13x11 grid. Grid cells may be floors (passable), walls, or boxes (impassable). Walls' positions are always the same (every odd cell in both directions). Only boxes can be destroyed (with bombs).

Players start in corners. Each turn a player may place a bomb (up to a limit) and move to the nearest cell (if it is free), thus there are at most 10 actions. Multiple players can occupy the same cell.

Bombs explode after 8 turns, creating a blast in both cardinal directions up to *range* cells. If it reaches another bomb, it causes a chain reaction.

Initially players may place one bomb with a *range* of 3. Boxes may drop items that increase one statistic when picked up. (Box contents are visible to all players.)

Test bed

The game ends when there is at most one player is left or 20 turns after all boxes are destroyed (with a hard limit of 200 turns). Players are ranked by the order of elimination (last one standing wins). Ties are resolved by the number of destroyed boxes.

Test bed

The game ends when there is at most one player is left or 20 turns after all boxes are destroyed (with a hard limit of 200 turns). Players are ranked by the order of elimination (last one standing wins). Ties are resolved by the number of destroyed boxes.

Top-tier agents usually end the game after 90-120 turns for two and 70-100 turns for four players.

Test bed

The game ends when there is at most one player is left or 20 turns after all boxes are destroyed (with a hard limit of 200 turns). Players are ranked by the order of elimination (last one standing wins). Ties are resolved by the number of destroyed boxes.

Top-tier agents usually end the game after 90-120 turns for two and 70-100 turns for four players.

Time limits are 1000ms for the first and 100ms for each of the following turns. The communication is based on standard input/output streams.

Engine

As usual, the engine is supposed to do two things: compute legal actions of a given state and apply them later. We have implemented two versions of it – one that is very straightforward and the other that heavily relies on preprocessing, bitwise operations, and other low-level programming techniques. For example, it calculates the bomb's blast radius in constant time.

Engine

As usual, the engine is supposed to do two things: compute legal actions of a given state and apply them later. We have implemented two versions of it – one that is very straightforward and the other that heavily relies on preprocessing, bitwise operations, and other low-level programming techniques. For example, it calculates the bomb's blast radius in constant time.

Number of actions performed by a random agent in $500ms$, starting from example midgame situation with 2, 3, and 4 players, resetting every 15 actions or death.

Number of players	2	3	4
`Naive engine`	$90k$	$79k$	$80k$
`Bitwise engine`	$1.45m$	$1.3m$	$1.28m$

Algorithms

We have implemented three different playing algorithms: MCTS, RHEA, and Beam Search.

Algorithms

We have implemented three different playing algorithms: MCTS, RHEA, and Beam Search.

All three algorithms try to predict opponents' behavior by applying a similar strategy. They spend some time performing search as each of the opponents with an assumption that all other players do nothing.

Algorithms

We have implemented three different playing algorithms: MCTS, RHEA, and Beam Search.

All three algorithms try to predict opponents' behavior by applying a similar strategy. They spend some time performing search as each of the opponents with an assumption that all other players do nothing.

Due to the quality of the search, both MCTS and RHEA spend 10ms on each opponent, while 5ms turned out to be enough for Beam Search.

Algorithms – MCTS

There are multiple variants of MCTS for multiplayer games. We have decided to use Single Player MCTS and rely on the opponents' predictions.

Algorithms – MCTS

There are multiple variants of MCTS for multiplayer games. We have decided to use Single Player MCTS and rely on the opponents' predictions.

Additionally, we prune the actions of the initial state, by doing a two turns deep exhaustive search of actions eliminating the player.

Algorithms – MCTS

There are multiple variants of MCTS for multiplayer games. We have decided to use Single Player MCTS and rely on the opponents' predictions.

Additionally, we prune the actions of the initial state, by doing a two turns deep exhaustive search of actions eliminating the player.

Within the time limit, agent reached depth of 12 for own actions and 9 for the opponents. Every playout was limited by a given depth (15), and if the game was not finished, a heuristic function was used instead.

Algorithms – RHEA

As the individual is a vector of actions to perform, it may be the case that some of the actions are not possible, or result in an immediate agent's death. We have decided to ignore both during the evaluation.

Algorithms – RHEA

As the individual is a vector of actions to perform, it may be the case that some of the actions are not possible, or result in an immediate agent's death. We have decided to ignore both during the evaluation.

All of the parameters, including the best chromosome length, were found experimentally. The optimal chromosome length turned out to be 17. It makes sense, as it is enough for the agent to "see" two bombs exploding one after the other.

Algorithms – RHEA

As the individual is a vector of actions to perform, it may be the case that some of the actions are not possible, or result in an immediate agent's death. We have decided to ignore both during the evaluation.

All of the parameters, including the best chromosome length, were found experimentally. The optimal chromosome length turned out to be 17. It makes sense, as it is enough for the agent to "see" two bombs exploding one after the other.

We have considered (μ, λ) and $(\mu + \lambda)$ replacement with full elitism. It seems that $(\mu + \lambda)$ replacement is much more efficient as it keeps previously found good solutions.

Algorithms – Beam Search

We extended a vanilla Beam Search schema by adding a few enhancements:

Algorithms – Beam Search

We extended a vanilla Beam Search schema by adding a few enhancements:

- **Zobrist hashing** (`ZH`). Deduplicate states by using Zobrist hashing.

Algorithms – Beam Search

We extended a vanilla Beam Search schema by adding a few enhancements:

- **Zobrist hashing** (`ZH`). Deduplicate states by using Zobrist hashing.
- **Opponent prediction** (`OP`). Approximate enemies' as described previously.

Algorithms – Beam Search

We extended a vanilla Beam Search schema by adding a few enhancements:

- **Zobrist hashing** (`ZH`). Deduplicate states by using Zobrist hashing.
- **Opponent prediction** (`OP`). Approximate enemies' as described previously.
- **Local beams** (`LB`). Limit the number of states in the same position.

Algorithms – Beam Search

We extended a vanilla Beam Search schema by adding a few enhancements:

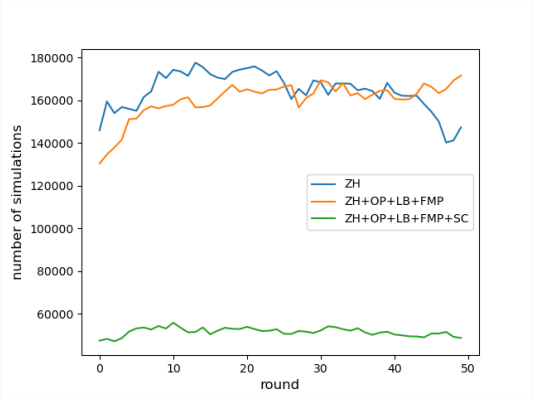
- **Zobrist hashing** (`ZH`). Deduplicate states by using Zobrist hashing.
- **Opponent prediction** (`OP`). Approximate enemies' as described previously.
- **Local beams** (`LB`). Limit the number of states in the same position.
- **First move pruning** (`FMP`). Prune all the unsurvivable actions of the initial state. If there is an action that leads to enemy death, we prune the rest.

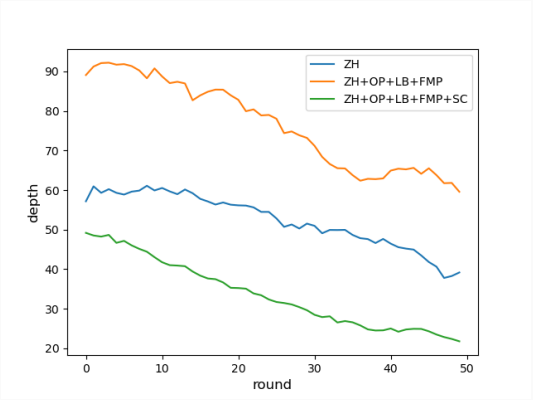
Algorithms – Beam Search

We extended a vanilla Beam Search schema by adding a few enhancements:

- **Zobrist hashing** (`ZH`). Deduplicate states by using Zobrist hashing.
- **Opponent prediction** (`OP`). Approximate enemies' as described previously.
- **Local beams** (`LB`). Limit the number of states in the same position.
- **First move pruning** (`FMP`). Prune all the unsurvivable actions of the initial state. If there is an action that leads to enemy death, we prune the rest.
- **Survivability checking** (`SC`). Highly discourage states in the beam that are not survivable, by decreasing their scores.

Results





Number of simulations and the depth reached by Beam Search within 100ms.

Influence of Beam Search improvements on agent's strength compared to the vanilla (i.e., using only ZH) version. 500 games per each pair.

Enhancement	WIN	LOSE
ZH	19.00%	19.00%
ZH+OP	44.80%	19.60%
ZH+LB	45.20%	36.20%
ZH+FMP	45.00%	29.20%
ZH+OP+LB+FMP	57.60%	22.60%
ZH+OP+LB+FMP+SC	59.40%	23.40%

Win percentages for each algorithm in 1 vs 1 setting. A single cell shows the win ratio of the row agent versus the column agent.

Agents were evaluated on 500 matches.

	MCTS	RHEA	Beam Search
MCTS	—	67.80%	2.20%
RHEA	22.40%	—	1%
Beam Search	96.60%	99%	—

Win percentages of the algorithms in 1 vs 1 vs 1 setting. A single cell shows in how many games the row agent obtained a higher score than the column agent.

Agents were evaluated on 1000 matches.

	MCTS	RHEA	Beam Search
MCTS	—	68.70%	10.30%
RHEA	23.80%	—	3.70%
Beam Search	84.20%	94.90%	—

1	DomiKo	C++	35.25	University of Wrocław	
2	karliso	C++	34.60	N/A	
3	DrFekalus	C#	34.42	DreamVis	
4	ValGrowth	C++	34.06	N/A	
5	FredericBautista	C++	33.84	Self-education / Retired	
6	MSmits	C++	33.73	Petrus Canisius College - Alkmaar / Petrus Canisius College	
7	Marchete	C++	33.34	Universidad Politécnica de Madrid	

A screenshot from the CodinGame Hypersonic leaderboard[2] (taken 27.01.2022), with our Beam Search algorithm variant on the first position.

[2] https://www.codingame.com/multiplayer/bot-programming/hypersonic/leaderboard

A screenshot from CGStats[3], showing detailed win-rates depending on the number of players.

Pseudo	Rank	Score	Language	Progress	Agent id
DomiKo	1	35.25	C++	100%	3695980

2 players (30 games)	Count	Percentage
Victory	22	73%
Defeat	8	27%

3 players (41 games)	Count	Percentage
Victory	18	44%
2nd position	15	37%
Defeat	8	20%

4 players (37 games)	Count	Percentage
Victory	23	62%
2nd position	9	24%
3rd position	3	8%
Defeat	2	5%

[3] http://cgstats.magusgeek.com/app/multi-hypersonic/domiko

Thank you!