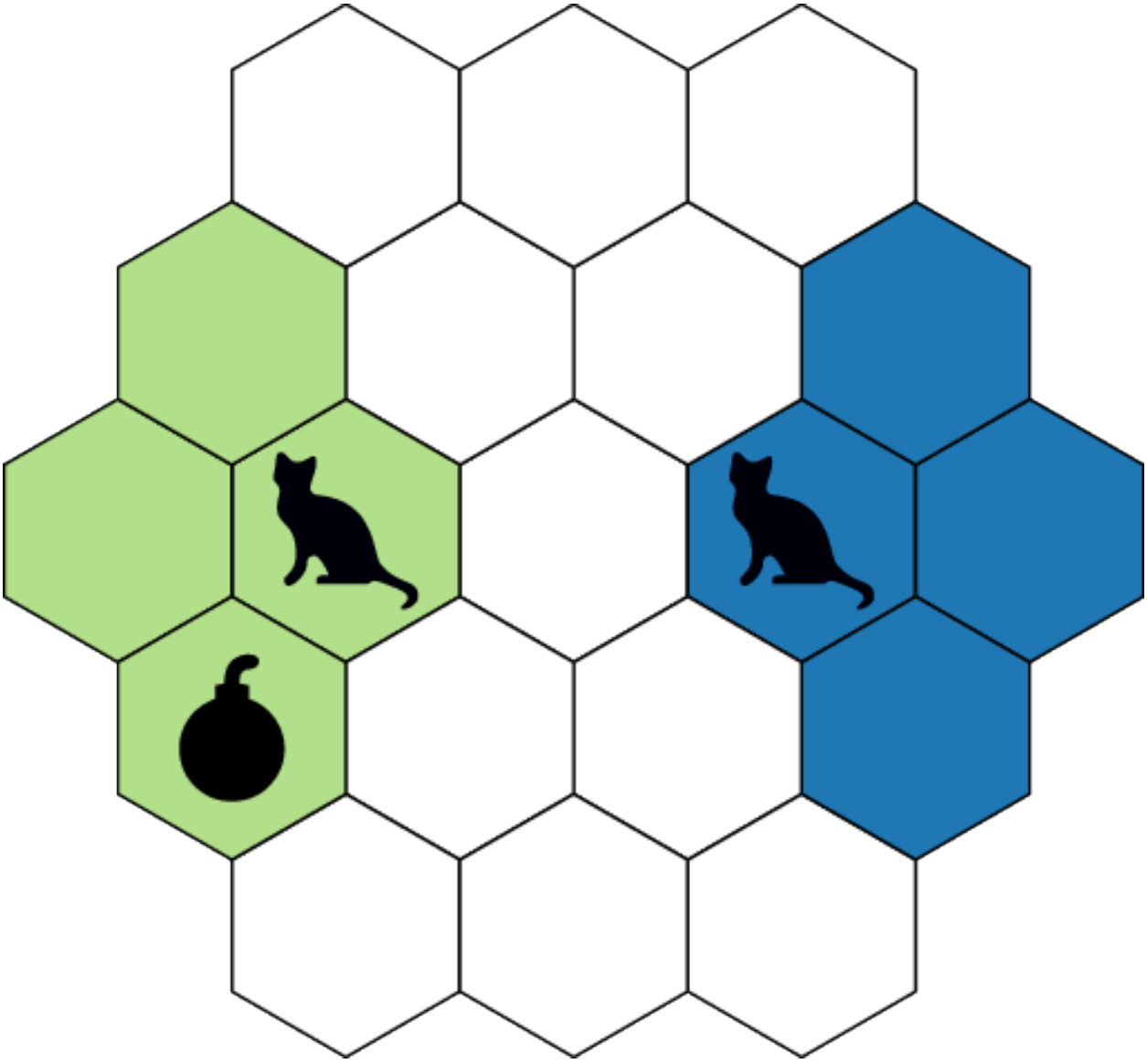
hexabomb Documentation

Millian Poquet

Mar 16, 2019

Contents

hexabomb is a netorcai multi-agent game intended to be played by bots. The game is strongly inspired by Bomberman and Splatoon, with hexagons.



CHAPTER 1

Launching your first game

1.1 Prerequisites

Some programs and libraries must be installed before launching your first game.

- netorcai (network server). Please refer to netorcai's installation documentation for this.
- hexabomb (game logic). See *Installation*.
- hexabomb-visu (visualization client). Please refer to hexabomb-visu's README for this.

You should also decide in which language your bot will be implemented. Some netorcai client libraries have been implemented in several languages to help you build your bot. If you do want to implement your bot in an unsupported language, please refer to netorcai's metaprotocol documentation to implement your own library.

1.2 Running the game server

The netorcai program is in charge of this. This program must be started first, as all the other ones connect to it. If it is installed correctly, just run `netorcai` to run it. Some of the game parameters can be tuned by netorcai's command-line interface, please refer to `netorcai --help` to list the tunable parameters.

1.3 Running the game logic

The hexabomb program is in charge of this. It can be executed with `hexabomb MAP`, where `MAP` is a map filename. Map files are available in hexabomb's git repository. Please refer to `hexabomb --help` for a list of hexabomb command-line options.

1.4 Running the visualization

The hexabomb-visu program does this. It can simply be executed with `hexabomb-visu`. Once again, more options are available (see `hexabomb-visu --help`).

1.5 Running example bots

How to run the bot highly depends on the selected language. Example bots and instructions on how to run them can be found in the `bots` directory of hexabomb's git repository.

Note: You are completely free to hack these example bots as a starting point to implement your own bot, as they are unlicensed.

1.6 Starting the game

The netorcai program has an interactive prompt. You can type `help` in it to list available commands. The `start` command should run the game.

Note: If you want the game to start automatically once all expected clients are connected, you may be interested in netorcai's `--autostart` option.

CHAPTER 2

Installation

This page explains how to install the hexabomb game. To visualize games, please refer to the hexabomb visualization client.

As hexabomb requires netorcai, you should also install netorcai and probably some netorcai client libraries. Please refer to the netorcai documentation for this.

2.1 Build the game

The hexabomb game is developed in D and can be installed with dub. First install a D compiler and dub. You can then directly run the latest release of hexabomb with `dub run hexabomb`. The following commands produce a standalone executable.

```
dub fetch --cache=local hexabomb
cd hexabomb-*/hexabomb
dub build
./hexabomb --help
```

CHAPTER 3

Game description

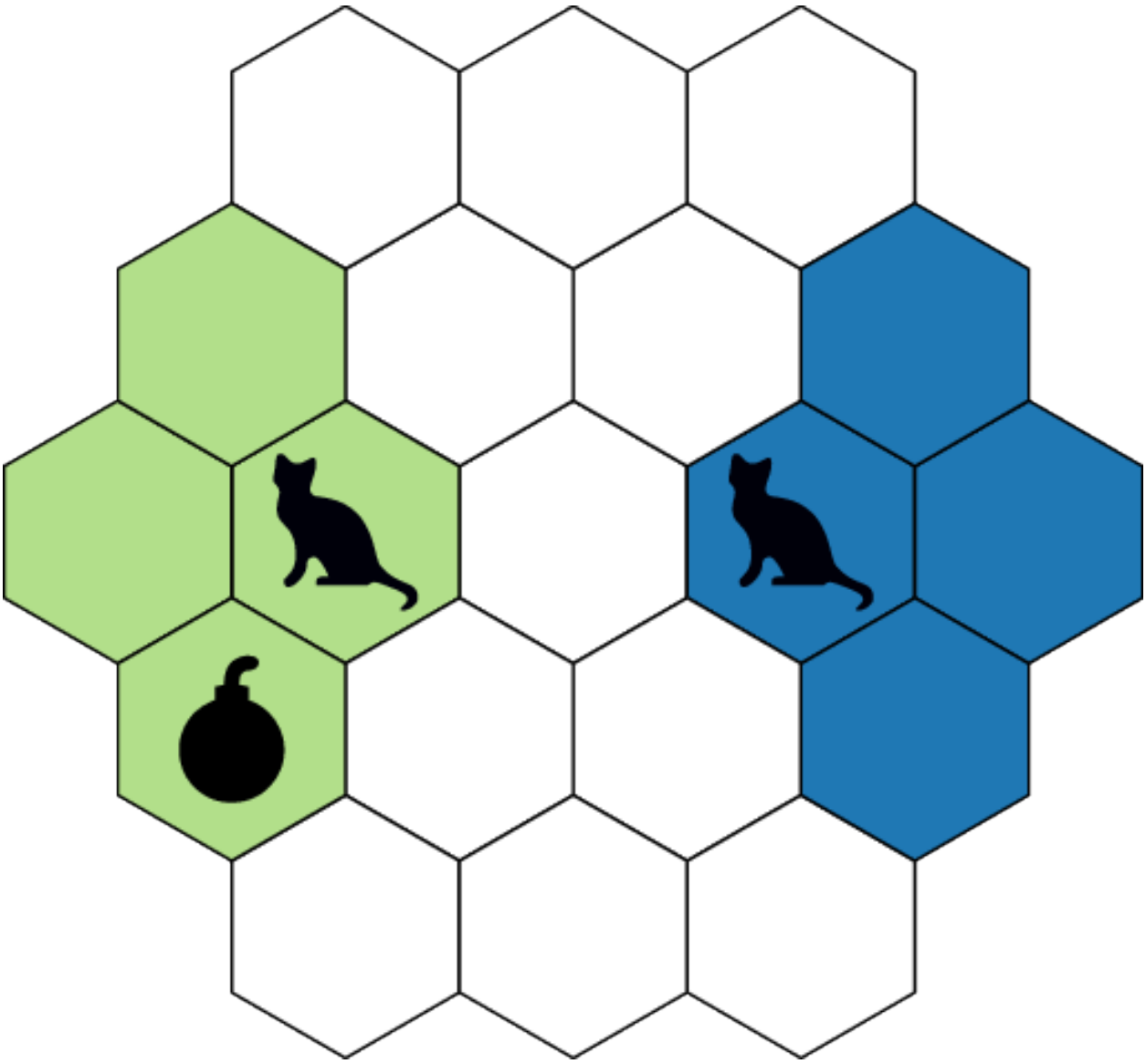
hexabomb is a network multi-player (multi-agent) game, intended to be played by bots. The game is strongly inspired by Bomberman and Splatoon, with hexagons.

Each player controls characters that move on a board. A color is associated to each player. The goal of each player is to have the largest numbers of cell of its color in the board.

For this purpose, the characters color the cells they go through. Additionally, the characters may drop bombs that color surrounding cells when they explode.

3.1 Board

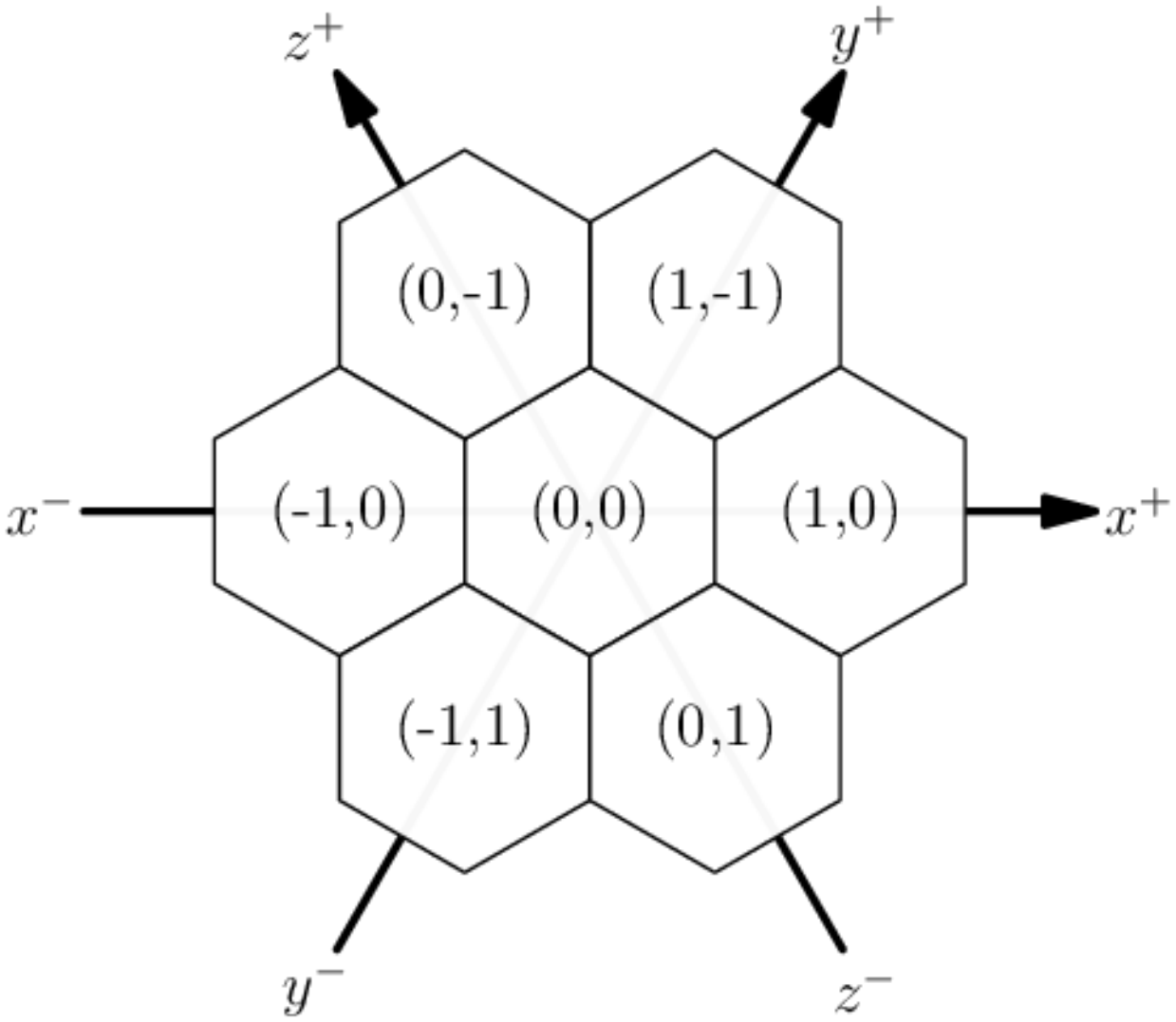
The game board is an hexagonal grid composed of cells.



A cell can have up to 6 neighbors (three axes, two directions per axis). Each cell is identified by its axial coordinates (q, r). This coordinate system makes sure that different cells have different coordinates, and that going into a given direction always results in the same coordinate transformations.

Table 1: Coordinates transformations from cell (q, r) to its neighboring cells.

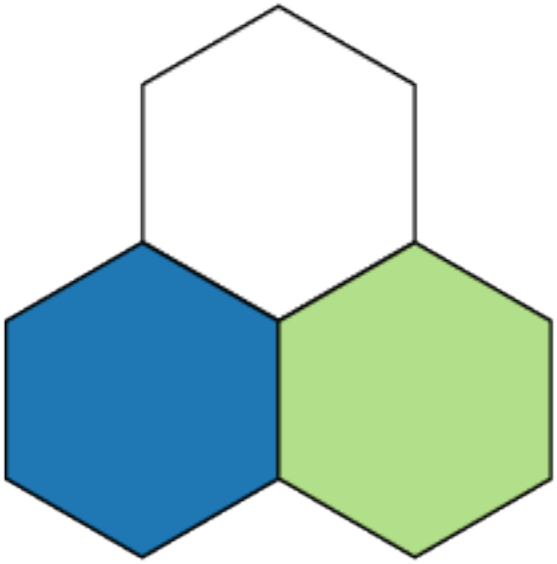
Direction	Meaning	Destination cell
x^+	towards right	$(q+1, r)$
y^+	towards up and right	$(q+1, r-1)$
z^+	towards up and left	$(q, r-1)$
x^-	towards left	$(q-1, r)$
y^-	towards bottom and left	$(q-1, r+1)$
z^-	towards bottom and right	$(q, r+1)$



3.2 Cells

Cells can host characters and bombs, and have a color (that can be neutral). Characters can move into cells if they are empty — i.e., if they do not host characters nor bombs. Cells can be colored by a bomb if they are in the bomb's explosion range.

Traversable cells



Empty (neutral color)

Empty (players' color)

Non-traversable cells

Characters

Bomb

3.3 Turns

The game is turn based. All players can do actions on each turn. All players should respect the following behaviour.

1. Wait for a new turn to start.
2. Receive a new turn (up-to-date board information) from the network.
3. Decide what to do.

4. Send the actions on the network. Start again (goto step 1).

3.4 Actions

On each turn each character can either move or drop a bomb. The exhaustive list of what each character can do is the following.

- Do nothing.
- **move**: Move one cell towards one of the 6 directions. The character must be alive. The target cell must exist and be empty.
- **bomb**: Drop a bomb in the character current cell. The character must be alive and its cell must NOT contain a bomb. The character's bomb count must be 1 or greater.
- **revive**: Revive a dead character on the cell where it died. The character must have been dead for at least 3 turns. The cell be empty.

hexabomb will apply the players' decisions in best effort. In case of conflicts, the faster player is priority.

3.5 Objective and score

At the end of the game, the player with the highest score wins the game.

The score of each player is the cumulated number of cells it controlled throughout the turns. In other words, at the end of each turn, the score of each player is increased by the number of cells of the player's color.

As an example, consider the following 5-cell board on which 2 players (Blue and Green) play.

At the beginning, Blue and Green control the same number of cells (1) and have the same score (1).

On first turn, Blue moves while Green does not. This allows Blue to earn 2 points this turn, while Green only earns 1 point.

Green remains motionless in the next turns, while Blue controls more and more cells. As a result, Blue's score increases way more than Green's.

3.6 Bombs

Bombs can be dropped by characters on their current cell. Bombs explode after a given **delay** and have a **range**. Upon explosion, bombs color the cells of their explosion area with the color of the player that dropped the bomb — killing any character and exploding any bomb present in the explosion area.

Bombs explode in straight lines in all 6 directions and cover up to *range* cells in each direction. A line is stopped if it encounters a non-existing cell — or after *range* cells have been covered.

The animation below shows a simple game scenario involving a bomb.

1. On first turn, Green drops a bomb (delay=3, range=2).
2. On second and third turns, Green moves away from the explosion area.
3. Green does no action during fourth turn. At the end of the turn, the bomb explodes as its delay reaches 0. The explosion area is highlighted in orange. At the end of the fourth turn, all the cells of the explosion range have been colored in green. Blue is killed in the process as it was in the explosion area.

Each character has a bomb count, which is either 0, 1 or 2 (initially, it is 1). Dropping a bomb costs one bomb. The bomb count of every character is increased by 1 every 10 turns (but the bomb count cannot exceed 2).

3.6.1 Simultaneous explosions

Several bombs can explode at the same time. This may happen when the delay of several bombs reaches 0 at the same time or in case of *Chain reaction*.

Simultaneous explosions can lead to conflicts about the coloration of the cells — as some cells can be in the explosion area of several bombs of different colors. The final color of an exploded cells is only determined by the bombs that explode the cell — and by the distance of these bombs to the exploded cell. This is how the color of such a cell is computed.

1. If the cell is strictly closer to one bomb than the others, the cell is colored by the color of the closest bomb.
2. If all the bombs of the set of the closest bombs to that cell have the same color, the cell is colored by the color of the bombs.
3. Otherwise (i.e., if any two bombs of the set of the closest bombs to that cell have different colors), the cell color is turned to neutral.

Simultaneous explosions are figured just below. In this example, all the bombs have a range of 3 cells.

Most of the exploded cells are closer to one bomb from the others and take the bomb's color. The interesting cells are thickly bordered orange.

- Cells at $(1, -3)$, $(2, -2)$ and $(3, -1)$ become green because the two closest bombs that explode each cell are green.
- Cell at $(1, 0)$ stays neutral because the set of the closest bombs that explode the cell contains bombs of different colors.
- Cell at $(-1, 2)$ becomes green because it is only covered by the green bomb at $(2, -1)$. This may seem counterintuitive because of the blue bomb at $(0, 0)$ that prevented cells at $(1, 0)$ and $(0, 1)$ to turn green. **Explosions cannot reduce the explosion range of each other, they can only interfere with the final color of the exploded cells.**

3.6.2 Chain reaction

Without any external influence, a bomb explodes when its delay reaches 0. A bomb can however explode before reaching a delay of 0 because of another bomb. This happens when a bomb is in the explosion area of another bomb (and when the other bomb explodes first). This can lead to a chain reaction where many bombs can explode at the same time.

If a chain reaction involves bombs of different colors, see *Simultaneous explosions* to understand how the cells of the explosion areas are colored.

3.7 Characters life and death

Characters can die because of *Bombs*.

A dead character is removed from the board — thus making the character's cell traversable. Being dead or alive does not impact the bomb count of a character at all.

A character death does not imply any direct score penalty on the dead character's player. However an indirect penalty still exists, as a dead character cannot do any action for at least 3 turns. After these 3 turns, the character can be revived via a *revive* action (see *Actions*).

CHAPTER 4

Sudden death

Sudden death is hexabomb's secondary game mode. It is based on the classical game mode described in *Game description*.

The main difference is the players' objective: They must now survive as long as possible without any consideration about the coloration of cells.

4.1 Game rules modifications

- Character death is permanent: `revive` action is not allowed.
- The score of a player is the number of turns it survived, that is to say the maximum turn number when it still had alive characters.
- There is a special player (identified by $playerID = 0$) that hunts and kills other players' characters. The characters of this special player respect modified rules.
  - Not affected by bomb explosions.
  - Can drop as many bombs as they desire.
  - Can drop bombs with delays and ranges in $[2, 100]$.

4.2 How to detect current game mode?

This depends on the number of special players that is received in the GAME_STARTS message.

- 0 special players means the game mode is classical (described in *Game description*).
- 1 special player means the game mode is sudden death.

4.3 What is the special player strategy?

- Attack close enemies.
- Try to circle enemies.
- Block small paths with bombs that have a long delay.

The following video shows a sudden death game with the special player (ghost characters) and random players (cat characters).

CHAPTER 5

Implementation details

As hexabomb uses netorcai and therefore the netorcai metaprotocol, this page first defines the game-dependent part of the protocol, which is essentially the format of the players' *Actions* and of the *Game state*. This page then answers frequently asked questions.

In netorcai, players are identified by a unique number $playerID$. In hexabomb, the color of a player is set to $playerID + 1$. This is because color 0 is reserved for neutral cells.

5.1 Actions

On each turn, players send their actions in a TURN_ACK netorcai message.

An action is represented as a JSON object with the following fields.

- `id` (number): The character unique identifier.
- `movement` (string): The type of action. Must be one of the following.
  - `move` if one wants the character to move.
  - `bomb` if one wants the character to drop a bomb.
  - `revive` if one wants the character to be revived.

Other fields are required depending on the desired `movement`.

- `direction` (string): The desired direction for `move` movements. Must either be `x+`, `y+`, `z+`, `x-`, `y-`, or `z-`.
- `bomb_range` (number) and `bomb_delay` (number): The desired bomb range and delay for `bomb` movements. Both must be respect $\{n \in \mathbb{N} \mid 2 \leq n \leq 4\}$.
- (No parameters are required for `revive` movements.)

5.1.1 Actions examples

The following example shows the actions taken by one player during one turn.

- Character 0 wants to move one cell towards the x^+ direction.
- Character 1 wants to drop a bomb on its current cell. The bomb should have a delay of 3 turns and a range of 3 cells.
- Character 2 wants to revive (on the cell where it died).

```
[
    {"id":0, "movement":"move", "direction":"x+"},
    {"id":1, "movement":"bomb", "bomb_delay":3, "bomb_range":3},
    {"id":2, "movement":"revive"}
]
```

5.1.2 Application of actions

hexabomb applies the actions in best effort. It will first try to apply each action in a given order, then try to apply failed actions until convergence of the game state.

The order into which the actions are applied is described in the following algorithm. Feel free to read the actual implementation in hexabomb's source code for more details.

```
Do
|   For each player in order of actions reception
|   |   For each action in player-specified order
|   |   |   Try to apply action if it has not been already applied successfully
While (game state has been modified)
```

5.2 Game state

hexabomb describes and sends the game state in DO_INIT_ACK and DO_TURN_ACK netorcai messages. Players receive the game state described here in GAME_STARTS, TURN and GAME_ENDS netorcai messages.

The game state is a JSON object with the following fields.

- `cells`: An array of cells. Each cell is an object with the following fields.
  - `q` (number): The cell q axial coordinate.
  - `r` (number): The cell r axial coordinate.
  - `color` (number): The cell color. 0 means neutral. Otherwise, means the cell belongs to player with $playerID = color - 1$
- `characters`: An array of characters. Each character is an object with the following fields.
  - `id` (number): The character unique identifier.
  - `color` (number). The character color. Means the character belongs to player with $playerID = color - 1$.
  - `q` (number): The q axial coordinate of the cell the player is in.
  - `r` (number): The r axial coordinate of the cell the player is in.
  - `alive` (boolean): Whether the character is alive or not.
  - `revive_delay` (number): The number of turns before which the player can be revived. Dead characters can only be revived when it reaches 0. -1 for alive characters.
  - `bomb_count` (number): The number of bombs the character holds. Either 0, 1 or 2. Increased by 1 every 10 turns (with a maximum of 2).

- `bombs`: An array of bombs. Each bomb is an object with the following fields.
  - `color` (number): The bomb color. Means the bomb belongs to player with $playerID = color - 1$.
  - `range` (number): The bomb range (AKA explosion radius), in number of cells.
  - `delay` (number): The bomb delay. Bombs explode when it reaches 0.
  - `q` (number): The q axial coordinate of the cell the bomb is in.
  - `r` (number): The r axial coordinate of the cell the bomb is in.
- `explosions`: An object of cells that exploded this turn. The key is the color of the exploded cells, while the value is an array of objects with the following fields.
  - `q` (number): The q axial coordinate of the cell that just exploded.
  - `r` (number): The r axial coordinate of the cell that just exploded.
- `cell_count`: An object where keys are **player identifiers** (not colors!) and values are their associated number of cells.
- `score`: An object where keys are **player identifiers** (not colors!) and values are their associated score.

5.2.1 Game state example

The following example shows a game state.

- There are three cells. Two belongs to first player, the last belongs to the other player.
- There are two characters. Only one of them is alive. The other cannot be revived right away, but it will be revivable next turn.
- There is one bomb.

```
{
  "cells":[
    {"q":0, "r":0, "color":1},
    {"q":0, "r":1, "color":2},
    {"q":0, "r":2, "color":2},
    {"q":1, "r":1, "color":2}
  ],
  "characters":[
    {"id":0, "color":1, "q":0, "r":0, "alive": true, "revive_delay":-1},
    {"id":1, "color":2, "q":0, "r":2, "alive":false, "revive_delay": 3}
  ],
  "bombs": [
    {"color":1, "range":3, "delay":2, "q":0, "r":1}
  ],
  "explosions": {
    "2": [
      {"q":0, "r":2},
      {"q":1, "r":1}
    ]
  },
  "cell_count":{
    "0": 2,
    "1": 3
  },
  "score":{
    "0": 8,
```

(continues on next page)

(continued from previous page)

```
    "1": 21
  }
}
```

5.3 How is a turn simulated?

On each turn, hexabomb does the following steps in order. Once again, feel free to read hexabomb's source code in case of doubt.

1. Apply players actions (see *Application of actions*)
2. Reduce the revive delay of dead characters.
3. Increase the bomb count of all characters (every 10 turns).
4. Reduce bomb delays, explode those reaching a delay of 0, compute chain reactions then color exploded cells and kill any character on them.
5. Update the cell count and score of each player.

CHAPTER 6

Example player clients

To avoid the need of implementing bots from scratch, naive bots implementations are provided in several languages. All the bots should be very similar in design.

- Use the netorcai client library of the target language (see netorcai documentation).
- Parse hexabomb game-dependent content in a dedicated *module*, and provide data structures corresponding to hexabomb entities (characters, bombs...).
- Provide a basic *main* file that reads and sends valid netorcai messages, taking random decisions for the characters that belong to the bot.

All example bots are located in the `bots` directory of the hexabomb git repository. The source code of each bot should be documented. Instructions for installation, execution and dependencies installation should be in the `README` of each bot.

CHAPTER 7

Changelog

All notable changes to this project will be documented in this file. The format is based on Keep a Changelog. hexabomb adheres to Semantic Versioning and its public API includes the following.

- hexabomb's command-line interface.
- hexabomb's game-dependent protocol (that uses the netorcai metaprotocol).
- hexabomb's game rules and their implementation.

7.1 Unreleased

- Commits since v1.1.0

7.2 v1.1.0

- Release date: 2019-02-24
- Commits since v1.0.0

7.2.1 Added

- New sudden death game mode, in which the objective of each player is to survive as long as possible. To enable this game mode, run a netorcai game with 1 special player.
- Cells that just exploded are now in the game state sent to clients each turn.

7.3 v1.0.0

- Release date: 2019-01-19
- Commits since v0.1.0

7.3.1 Changed game rules

- Characters can no longer be revived right away after being killed.
- Characters can no longer be revived on a target cell — this is now only possible on the cell where they died.
- Characters now have a bomb count (0, 1 or 2). Dropping a bomb costs one bomb. The bomb count of all characters is increased by 1 every 10 turns (cannot exceed 2). The bomb count initial value is 1.
- Walls have been removed, as they were equivalent to an absence of cell.
- The game state format is now the same in `DO_INIT_ACK` and `DO_TURN_ACK`.

7.3.2 Fixed

- Only one character was allowed per player.

7.4 v0.1.0

- Initial release.
- Release date: 2018-10-30.