- Competitions
- Discussions
- In cooperation with Warlight.net

- avdonface
- Sign out

AI Block Battle

- Home
- Rules
- Getting Started
- Leaderboard
- Game log
- Discussions

Start *coding* right away

You have two options when you want to participate in this competition. The first is to download one of the starterbots from the right. All the basic stuff like communication with the engine is already implemented, but making the bot smart is up to you. The other option is to create your own bot from scratch. Look at the Languages page for a list of all available languages to code your bot in.

All communication between your bot and the engine works through the standard input and output channels. In the section "Communicating with the game engine" you can find a detailed explanation regarding the protocol all bots should implement to communicate with the game server.

The engine source code will be made available as soon as the competition is out of beta, so you can test your bot locally. It will be available on our Github page.

Starter bots

Java
C#   By LithiumToast
Scala   By GreenTea
Haskell   By knullberg
Pascal   By skalogryx
Pascal   By paunovic
Rust   By Pete
Python   By krille
C++   By cube
C   By dtahcs

A simple example

```java
import java.util.Scanner;

public class MyBot {

    private Scanner scan = new Scanner(System.in);

    public void run()
    {
        while(scan.hasNextLine()) {
            String line = scan.nextLine();
            if(line.length() == 0) { continue; }
            String[] parts = line.split(" ");

            switch(parts[0]) {
                case "settings":
                    // store game settings
                    break;
                case "update":
                    // store game updates
                    break;
                case "action":
                    System.out.println("drop");
                    System.out.flush();
                    break;
                default:
                    // error
            }
        }
    }

    public static void main(String[] args) {
        (new MyBot()).run();
    }
}
```

This is a very simple example of a working bot in Java. It reads the engines output and only does something when an action is requested by the engine: it will drop the piece. This bot obviously won't win most of the games, but technically it is doing the correct stuff.

Communicating with the *game* engine

The communication between your bot and the engine goes via the standard input and output channels. Every single line the engine gives is a specific piece of information or a request. The response from your bot should also be just one line. Starting with this competition we have standardized the API, so all new competitions will also work with this API. There are three types of lines, in the following format:

- **settings [type] [value]** given only at the start of the game. General settings of the game are given here.
- **action [type] [time]** indicates a request for an action.
- **update [player] [type] [value]** this is an update of the game state. **[player]** indicates what the update is about, but could also be 'game' to indicate a general update.

In the table below you can find all specific commands your bot can expect for this competition. The format for an action response is also given.

Output from engine	Description
settings timebank *t*	Maximum time in milliseconds that your bot can have in its time bank
settings time_per_move *t*	Time in milliseconds that is added to your bot's time bank each move
settings player_names [*b*,...]	A list of all player names in this match, including your bot's name
settings your_bot *b*	The name of your bot for this match
settings field_width *i*	The width of the field, i.e. number of row-cells
settings field_height *i*	The height of the field, i.e. number of column-cells
update game round *i*	The number of the current round
update game this_piece_type *x*	The type of the piece that has just spawned on the field
update game next_piece_type *x*	The type of the piece that will spawn the next round
update game this_piece_position *i,i*	The starting position in the field for the current piece (top left corner of the piece bounding box)
update *b* row_points *i*	The amount of row points the given player has scored so far
update *b* combo *i*	The height of the current combo for the given player
update *b* skips *i*	The amount of skips the given player has available
update *b* field [*i*;...;...]	The complete playing field of the given player
action moves *t*	Request for the whole set of moves for this round
Output from bot	**Description**
[*m*,...]	A list of moves separated by comma's

Each "i" can be any integer. Each "b" is represents the name of a bot (as far as the engine is concerned) and can be either *player1* or *player2* for this competition. "t" represents a time in milliseconds. A "c" is a cell type of the playing field. This can be a 0, 1, 2 or 3, for *empty, shape, block* or *solid* respectively. Shape means that the cell is part of the current moving piece (shape) in the field, a block is one of the cells that can be set to empty again by clearing a row, and a solid cell is at the bottom of the field, next to you by your opponent. Solid cells cannot change again. The "c" can only be found in the field update; cells in a row are separated by comma's and the rows are separated by semicolons (see the section below for an example). "x" is one of the 7 piece types and can be either *I, J, L, O, S, T* or *Z*. Finally, the "m" in the bot's response can be one of the following moves: down, left, right, turnleft, turnright, drop or skip.
Output "no_moves" if you want your bot to do nothing, rather than letting it time out, so the bot is not shut down by the engine for being defect. For this competition the default action is *drop*.

The engine will always try to perform the action given by the output of your bot. In case the action is not valid in the current game situation, the engine will output a warning and either skip the action (turnleft/turnright), change it to 'down' (left/right) or freeze the piece in place (down).

Input and output between the bot and the engine go via standard input and output channels, but you can use the error output channel to print stuff in the error dump of your game. This will not be read by the engine and can be used for debugging.

By *example*

The game itself is not very difficult to understand, but to make it a bit more clear what the inputs and outputs look like we're providing an example. You can find this in the dump of any of your own games as well.

```
settings timebank 10000
settings time_per_move 500
settings player_names player1,player2
settings your_bot player1
settings field_height 20
settings field_width 10
```

These are the settings of the game. This information is sent at the start of a match only. Your bot now knows the timebank settings, the player names and the field dimensions. Continuing with the rest of the input/output:

```
update game round 1
update game this_piece_type O
update game next_piece_type I
update game this_piece_position 4,-1
```

Your bot gets updates about the game in general. More specific, it gets the round number and information about the current and next piece. Continuing:

```
update player1 row_points 0
update player1 combo 0
update player1 skips 0
update player1 field 0,0,0,0,1,1,0,0,0,0;0,0,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,0,0; ... 0,0,0,0,0,0,0,0,0,0
```

These are the updates about your bot (player1). The current points, skips and combo are 0, because this is just the start of the game. The field is empty, except for the O piece that has spawned at the top.

```
update player2 row_points 0
update player2 combo 0
update player2 field 0,0,0,0,1,1,0,0,0,0;0,0,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,0,0; ... 0,0,0,0,0,0,0,0,0,0
```

Next your bot gets the updates about your opponent. Again the points, combo and field, but at the start of the game they are the same as your own bot's updates.

```
action moves 10000
left,left,left,drop //output from your bot
```

Lastly, your bot gets the request to do his moves. The amount of time your bot has to respond is 10000, which is the full timebank at the start of the game. The moves your bot returns are 4 times left and then a drop. Now the piece fits perfectly in the lower left corner of the field. Nice. The game continues again from the player updates, goes to the player updates and then requests an action again. Seems easy enough, right?