Cryptron

This document is a step by step guide to Cryptron and mentions the basic game rules. The participants are expected to submit their codes in C/C++/Python.

Introduction

There are two high security data centers (bases) on the opposite ends of the board, with 6 layers of security between them. The goal is to invade the opponents base and steal their data with the help of HackDroids.

The game is played on an arrangement of 8 rows. The aim of the game is to place your HackDroids in the opponent's base (home row) or as near as possible to it.

Basic Terminologies:

Pieces:

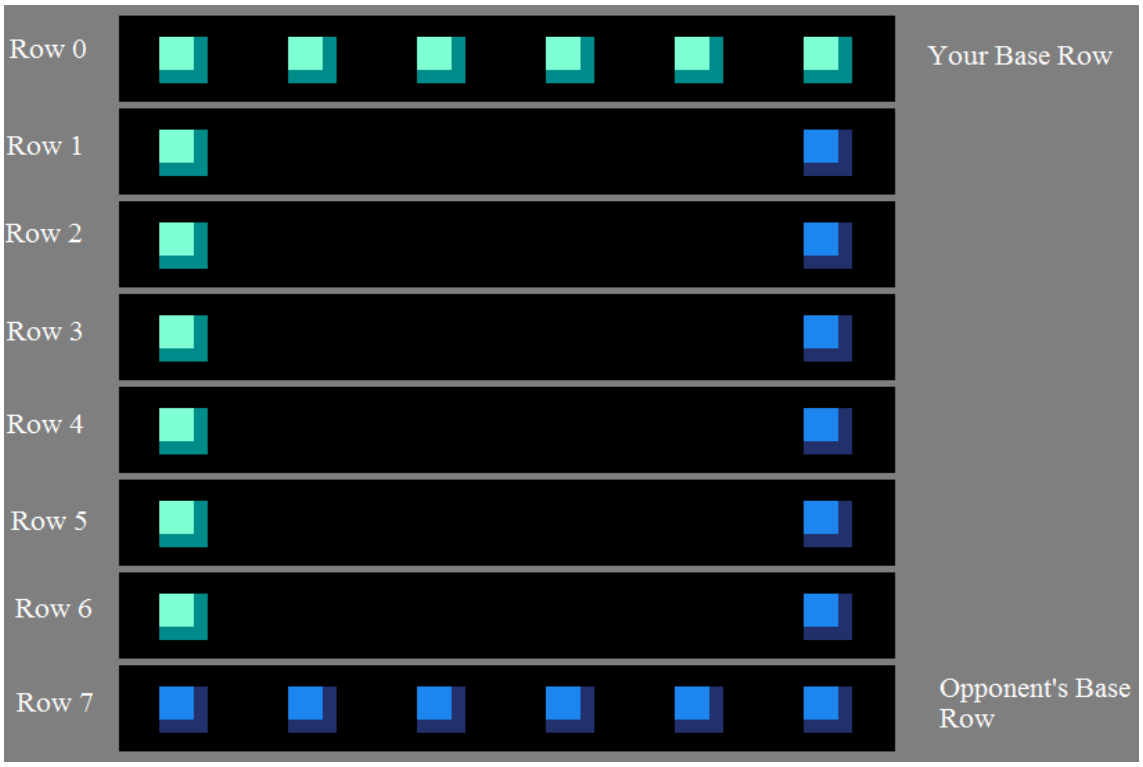
The game consists of 24 HackDroids.
Each player has 12 HackDroids with 6 in the base and 1 HackDroid in each row till the opponent's base.

Rows:

The game consists of 6 rows and 2 bases(home rows). Each row can contain up to 6 HackDroids.

Base:

The rows at the two ends of the board where 6 HackDroids of the respective player are initially placed.
Your base is the 0th row and your opponent's base is the 7th row.
The bases can contain any number of HackDroids.



Game-play:

The game is turn based. Each player gets alternate turns and each turn is composed of 2 moves. Player 1 and 2 will be decided at random.

Initial Input:

The first player will be chosen at random and will receive input as 1 and second player will receive input as 2.
No output is expected at this stage.

Note:
Your base will always be row 0, irrespective of you being player 1 or player 2.

Subsequent Gameplay:

Each player gets to play two moves per turn. The types of moves are :

1. First attack
2. Second attack

1) First Attack:

- The player can move any one of their HackDroids ahead by one row only.
- If the player moves the HackDroid from row i_1 to row f_1,

$$f_1 - i_1 = 1$$

- The total number of tokens in the landing row before the first attack will determine the number of rows the player has to move in the second attack.

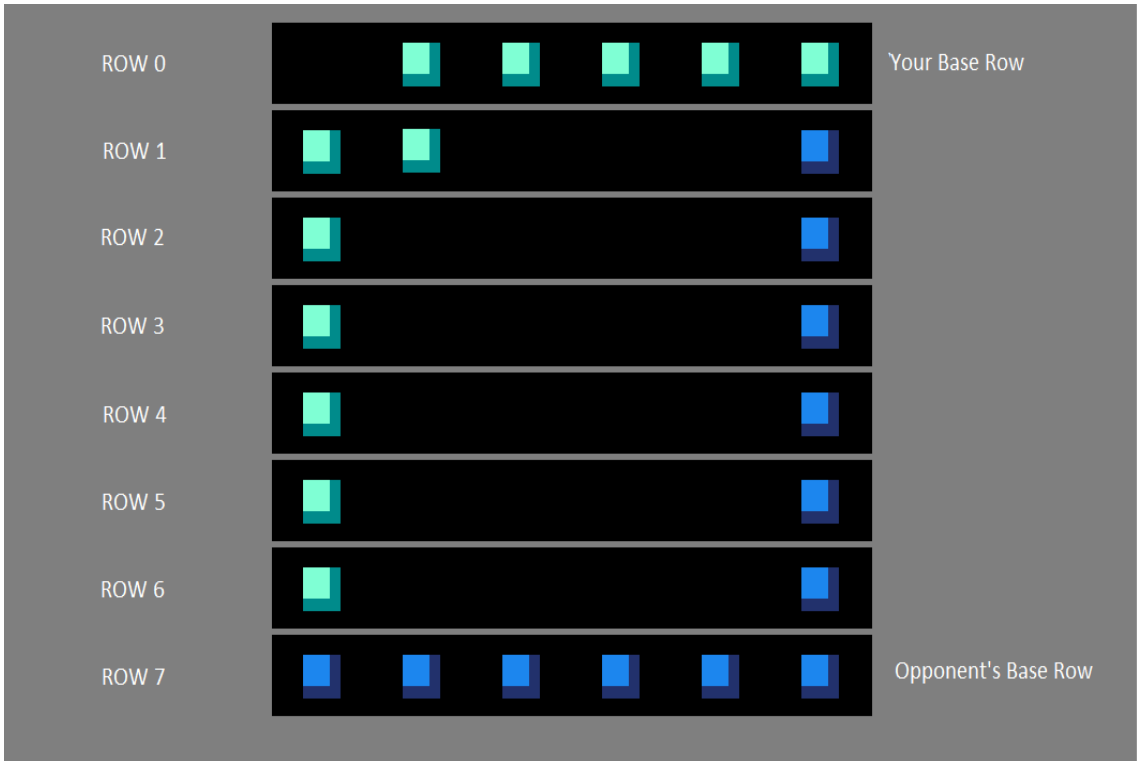
2) Second Attack:

- The player can move any one of their HackDroids ahead by exactly the number of rows as decided by the first attack.
- If the player moves the HackDroid from row i_2 to row f_2,

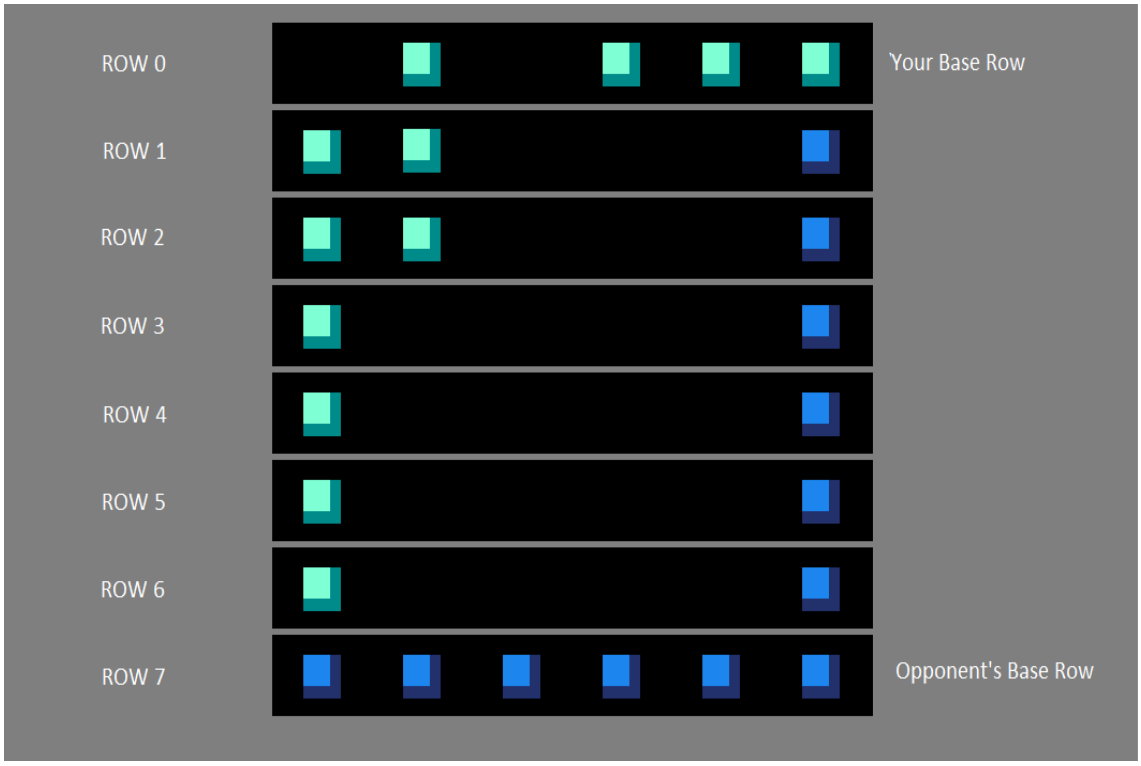
  $f_2 - i_2 =$ number tokens present in f_1 before the first attack.

Example:

Consider the setup as given below. In the first move, a HackDroid is moved from row 0 to row 1, making a jump of length 1.



Before playing the first move, row zero had exactly 2 HackDroids. So the the next move to be played must make a jump of exactly 2, as shown when a HackDroid is move from row 0 to row 2.



Special Case:

If the final position of the first move i.e. f_1 is in the opponent's base, the HackDroid can move exactly one row ahead in its second attack. $f_2 - i_2 = 1$

Input that your code recieves:

The input specifies the opponents previous move.
The input your bot will receive is a 7 character string (including commas) of the format
(i_1, f_1, i_2, f_2) where:
i_1 is the initial position for the 1st move. ($0 <= i_1 < 7$)
f_1 is the final position for the 1st move. ($0 < f_1 <= 7$)
i_2 is the initial position for the 2nd move. ($0 <= i_2 < 7$)
f_2 is the final position for the 2nd move. ($0 < f_2 <= 7$)

Expected Output:

Your bot is expected to give an output as a 7 character string of format (i_1, f_1, i_2, f_2) where:
i_1 is the initial position for the 1st move. $(0 <= i_1 < 7)$
f_1 is the final position for the 1st move. $(0 < f_1 <= 7)$
i_2 is the initial position for the 2nd move. $(0 <= i_2 < 7)$
f_2 is the final position for the 2nd move. $(0 < f_2 <= 7)$

Note:
Player 1 will not receive any input for the first turn as his/her move is the first move of the game.

Bot Format:

The players bot (code) is run only once, with his/her logic executed in an unconditional while loop.
The format of the code should be as follows:

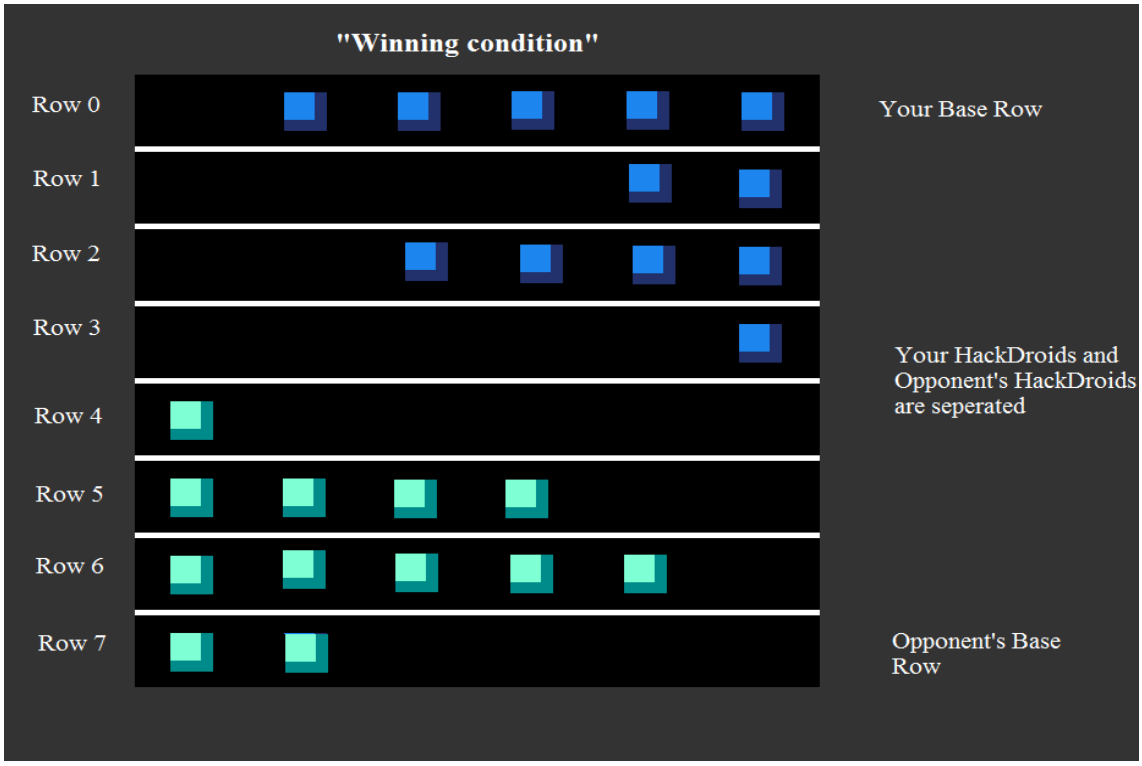
```
main()
{
  /* intial input scanned by the player
  to receive player number: Xi */

  while(1)
  {

    /* if (!(player==1 && turn==1))
    input scanned by the player for every
    iteration specifying the opponent s previous move: X */

    //player's logic


    /*output printed by the player at the end
    of every iteration specifying the current move: Y */
  }
}
```

$x_i = 1 \mid x_i = 2$
$x =$ "i_1, j_1, i_2, j_2"
$y =$ "i_1, j_1, i_2, j_2"

Note:
The moves should be maintained by the players according to their own logic, using an appropriate data structure for the grid. The grid would not be passed to them.
Sample grid initialization is provided in the templates, put up on the website.

End of the game:

The game ends when all your HackDroids cross all of your opponent's HackDroids.



Determination of the winner:

The winner is determined by the calculation of points at the end of the game. The player with more points wins the game.

The marking scheme for you is as follows:

Row Number	Points per HackDroid
7	5
6	3
5	2
4	1

Marking scheme for your opponent:

Row Number	Points per HackDroid
0	5
1	3
2	2
3	1

Judging Criteria:

The overall winner of XOdia will be the one with maximum points. The points are gained as follows:

1. For every match won, the player gets 2 points each.
2. The player who loses will get no points. (no negative marking).
3. In case of a draw, both the players get 1 point each.

Disqualification Criteria:

Bot will be disqualified due to any of the following reasons:

- Bot did not respond within 2 seconds (for C and C++)/6 seconds(for python) of its execution.
- Bot returned an invalid move.
- Syntactical errors in program.
- Excessive resource usage (Bot should consume less then 8MB memory and max file size must be less then 1MB).
- Any malicious activity encountered in the code (The latest version of the bot would be taken into consideration).

In case of any disputes, the decision of the XOdia team will be final.