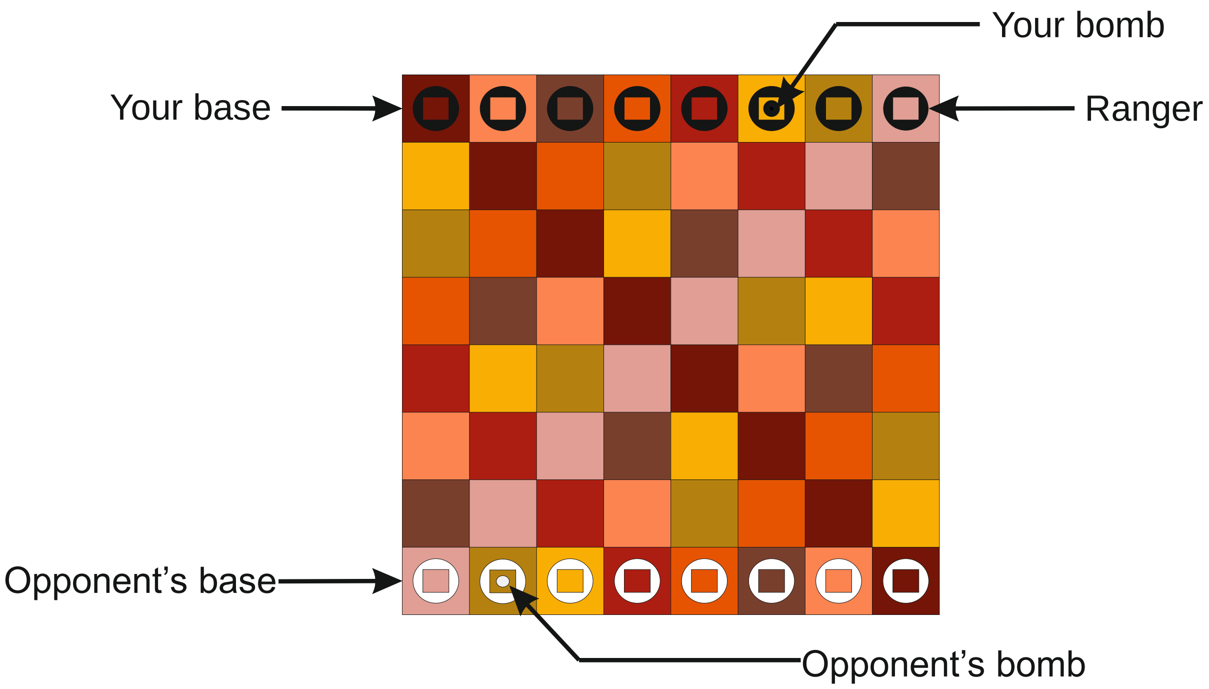
Ka-Boom

This document is a step-by-step guide to Ka-boom. It mentions the basic game rules. Accompanying this file, you will find a standard template. The participants are expected to submit their codes in C/C++/Java/Python.

The aim of this game is to place any one of your rangers along with your bomb at the opponent's base.



The game is played on an 8*8 (row*column) grid. The colours of the grid are in a fixed pattern as shown in the figure.

Every grid position is represented by (i, j) where 0 <=i, j <=7.

Basic Terminologies

1. **Ranger:**

- Each player has 8 uniquely coloured rangers.

- Initially, the players’ rangers are placed in their respective bases.

2.Base:

- The row at one end of the board where all the rangers of a player are initially placed.
- Your base is the row 0 i.e. (0, j) and the opponent’s base is the row 7 i.e. (7, j).

3.Bomb:

- Each player gets one bomb, which he/she can give to any one of his/her rangers at the beginning of the game.
- Your bomb can only be passed between your rangers.

Gameplay and Move format

For the gameplay, there are two types of moves:

- ➢ First move.
- ➢ Subsequent moves.

First move:

- Choose a position from your base where you want to place the bomb i.e. (0, j).
- Your opponent will do the same in his/her base.

Input that your code receives:

- The first player will be chosen at random and will receive input as “1” and second player will receive input as “2”.

- Player 1 will not receive any other input for this turn as his/her move is the first move of the game.
- Player 2 will receive another input specifying the column number in which the opponent has placed his/her bomb, in the following string format: **"j"**

  where j is the column number.

  Expected output:

- The player has to specify the column number in which he/she wants to place the bomb in the base row, in the following string format: **"j"**

  where j is the column number.

Note:

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

Subsequent moves:

For all subsequent moves, each player can either choose to move his/her ranger OR pass the bomb.

- If the player chooses to make a move, he/she must move the ranger that matches the colour of the square on which the opponent's ranger or bomb landed in the previous move. A ranger can move forward or diagonally forward.
- If the player chooses to pass the bomb, he/she must pass it either vertically, horizontally or diagonally to one of his rangers.

Maximum number of consecutive bomb passes allowed for each player: 15.

Input that your code receives for subsequent moves:

A string of 5 characters separated by commas in the following string format: **“x,i1,j1,i2,j2”**

where

- x=0 represents opponent’s bomb movement and x=1 represents opponent’s ranger movement.
- (i1, j1) is the initial position of opponent’s ranger/bomb.
- (i2, j2) is the final position of opponent’s ranger/bomb.

Expected Output for the subsequent moves:

The player has to specify whether he/she wants to move his/her bomb or his/her ranger, along with the initial and final positions of the bomb/ranger. Output string must be in the following format: **“x,i1,j1,i2,j2”**

where

- x=0 represents bomb movement and x=1 represents ranger movement.
- (i1, j1) is the initial position of ranger/bomb.
- (i2, j2) is the final position of ranger/bomb.

BOT FORMAT

The player’s bot (code) is run only once, with his/her logic executed in an unconditional loop of while.

The format of the code should be as follows :

```
main()

{
```

```
//initial input scanned by the player to receive player number: X1
while(1)
{
//if player number==2 and move==first, input scanned by the player to receive player1 move: X2
//if move!=first, 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
} }
```

X1= 1 || X1= 2

X2= { i | 0<= i <=7 }

For player 1's second move, X= { i | 0 <= i <= 7 } .

For all other conditions, X= { "x,i1,j1,i2,j2" | x=0 || x=1, 0 <= i1,j1,i2,j2 <= 7 }

Y ={ "x,i1,j1,i2,j2" | x=0 || x=1, 0 <= i1,j1,i2,j2 <= 7 }

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.

Valid & Invalid moves:

- For rangers:

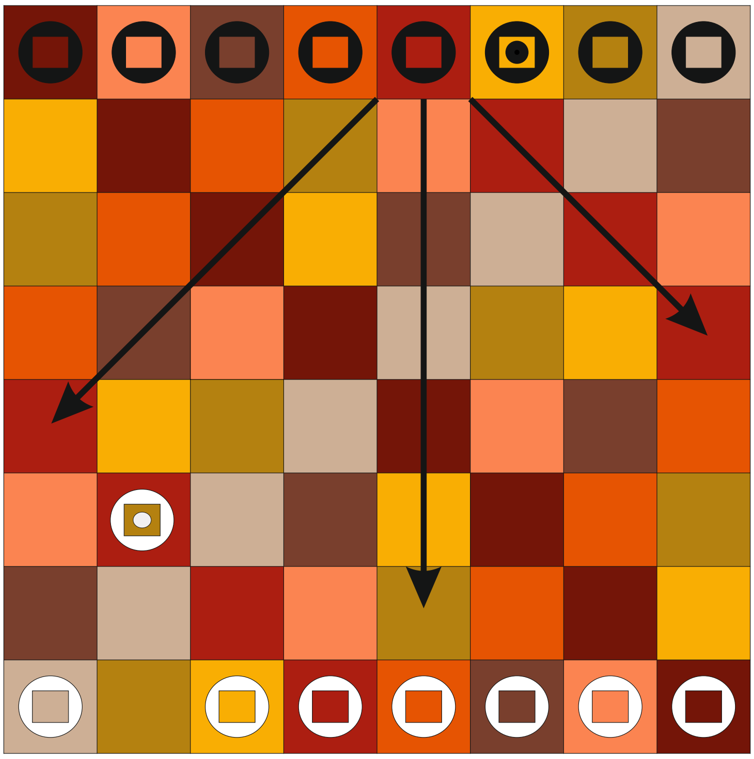
Valid Moves	Invalid moves
If a ranger moves in a straight line, either directly forwards or diagonally forwards.	If a ranger moves sideways or backwards.
A ranger can be moved by any number of squares.	While moving from one square to another, if a ranger passes through a square which already contains another ranger (either belonging to the player or his/her opponent).
If a ranger moves directly to opponent’s base from his base and doesn’t pass through any other ranger while moving.	If a ranger lands on a square which was already occupied.
	If a ranger makes a move which is out of the grid.
	If a ranger of the wrong colour is moved.

	The ranger which a player is supposed to move in the current turn cannot stay in the same position unless and until it is blocked (refer to 'special cases' to see what happens when ranger is blocked).

- For passing the bomb:

Valid Moves	**Invalid Moves**
If the bomb is passed to a square where a ranger (belonging to the player) is present.	If the bomb is passed to an empty square.
If the bomb is passed forward, backward, diagonally forward, diagonally backward or sideways.	If the bomb is passed to a ranger belonging to the opponent.
The bomb can be passed by any number of squares.	If the bomb passes through another ranger before landing on the specified square.

For example:



If your opponent moved onto a red square, you can either move your red ranger in one of the directions shown, or you can pass your bomb.

Special Cases

Ranger is blocked:

- If the ranger which you are supposed to move is completely blocked (cannot be moved forward or diagonally forward), you must pass the bomb. If it is not possible to do so, you must forfeit your turn.
- If you forfeit your turn, your final position=initial position

  ((i1,j1)= (i2,j2)) and your opponent must then move the ranger that matches the colour of the square on which the blocked player's ranger was trapped.

Deadlock:

- If you forfeit your turn and subsequently your opponent does the same, deadlock situation occurs.
- When this happens, the last person to move a ranger before the deadlock occurs loses that round.

End condition

The game will end when

- Any one of the players places any one of his/her rangers along with the bomb in the opponent's base.
- Deadlock situation.
- A bot exceeds the maximum number of total moves allowed i.e. 50.

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 draw, both the players get 1 point each.

Note

Bot will be disqualified due to any of the following reasons:
• Bot did not respond within 2 seconds of its execution.
• Bot returned an invalid move.
• Syntactical errors in program.
• Excessive resource usage (Bot should consume less then 8MB memory and disk usage must be less then 2MB).
• 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.

Happy Coding!