Goal

End the game with a higher **score** than your opponent.
Three players are pitted against one another in the **arcade olympics**.

Each player controls a character in **four** mini-games **simultaneously**. Earn a maximum of **medals** in all four games to acquire the highest **score**.

Rules

Each player is hooked up to **four** different arcade machines, and each of these machines is running a different **mini-game**. Your code can read the 8 **registers** used internally by the machines: `GPU`, containing a string and `reg0` to `reg6` containing integers. What those values represent is different for each game.

The game is played in turns. On each turn, all three players perform one of four possible actions: `UP`, `DOWN`, `LEFT`, or `RIGHT`.

When an action is performed by a player, their agent in **each** mini-game performs that same action, because the controls have been wired to all 4 machines at once.

Earning medals

The four mini-games play on loop throughout the game. In each run of a mini-game you may acquire a gold, silver or bronze **medal**. In between runs is a **reset** turn where the mini-game is inactive.

At the end of the game, each player's score for each mini-game is calculated based on the number of medals earned in total, with this formula:

`mini_game_score = nb_silver_medals + nb_gold_medals * 3`

The scores for all **four** mini-games are **multiplied together** to determine the **final score**.

During a reset turn, the `GPU` register will show `"GAME_OVER"`.

If there are ties in a mini-game, tied players will win the same highest medal. For instance, if two players tie for first place, they will both win gold and the third player will receive **bronze**.

Mini-game 1: Hurdle Race

This mini-game is a race between the three agents. Each agent is on the same randomly generated race track. The racetrack is composed of **30 spaces**, agents start on the first space, and the last space is the finish line. A space may contain a **hurdle** which the agents must **jump** over or else they will **collide** with it and be **stunned** for the next `3` turns. A stunned agent will not move regardless of the action performed.

On each turn, the agents can perform one of the following actions:

- `UP`: jump over one space, ignoring any hurdle on the next space and moving by `2` spaces total.
- `LEFT`: move forward by `1` space.
- `DOWN`: move forward by `2` spaces.
- `RIGHT`: move forward by `3` spaces.

Moving into a hurdle will interrupt the agent's movement, stopping on the same space as the hurdle.

When either agent reaches the **finish**, the run ends. The players are awarded a medal based on their positions in the race, and the next run begins after a **reset** turn.

Register	Description	Example
GPU	ASCII representation of the racetrack. `.` for empty space. `#` for hurdle.	`.....#...#...#.....`
reg0	position of player 1	0
reg1	position of player 2	6
reg2	position of player 3	12
reg3	stun timer for player 1	1

reg4	stun timer for player 2	0
reg5	stun timer for player 3	2
reg6	*unused*	

The **stun timer** is the number of turns remaining of being stunned (`3`, then `2`, then `1`). `0` means the agent is not stunned.

Mini-game 2: Archery

Each player controls a cursor with an x coordinate and a y coordinate. Each turn, players pick a direction, then move their cursor in by the current **wind strength** in that direction. After `12` to `15` turns, the players win medals according to how close they are to coordinate `(0,0)` in Euclidean distance.

The x and y coordinates are capped in within `[-20;20]`.

Register	Description	Example
GPU	A series of integers, indicating the power of the wind for upcoming turns. The integer at index `0` is the current wind strength.	`9914113315261`
reg0	x coordinate for player 1	0
reg1	y coordinate for player 1	-10
reg2	x coordinate for player 2	5
reg3	y coordinate for player 2	8
reg4	x coordinate for player 3	-2
reg5	y coordinate for player 3	20
reg6	*unused*	

Mini-game 3: Roller Speed Skating

Players race on a cyclical track `10` spaces long. Each player will have a `risk` attribute ranging from `0` to `5`.

On each turn, a list of the 4 actions will be provided in a random order in the `GPU`, e.g. `ULDR` (for `UP`, `LEFT`, `DOWN`, `RIGHT`), this is called the **risk order**. Performing the action at a higher index will move the player forward the more spaces. But choosing the fastest move is not without risk...

- The action at index `0` will move your player by `1` space and **decrease** your `risk` by `1`
- The action at index `1` will move your player by `2` spaces
- The action at index `2` will move your player by `2` spaces but increase your `risk` by `1`
- The action at index `3` will move your player by `3` space but increase your `risk` by `2`

What's more, if after a move a player finds themselves on the same space as an opponent, both their `risk` is increased by `2`! If a player risk rises to `5` or more, the player is stunned for the next `2` turns and their `risk` is reset to `0`.

Register	Description	Example
GPU	This turn's risk order	`URLD`
reg0	spaces travelled by player 1	2
reg1	spaces travelled by player 2	9
reg2	spaces travelled by player 3	21
reg3	risk of player 1 or stun timer as a negative number if stunned	4
reg4	risk of player 2 or stun timer as a negative number if stunned	-1
reg5	risk of player 3 or stun timer as a negative number if stunned	0

reg6	turns left	14

You can determine if two players share a space by comparing their **spaces travelled modulo 10**.

Mini-game 4: Diving

The players must match the sequence of directions given at the start of each run, called the **diving goal**.

Each turn where an agent's action matches this turn's diving goal direction, the player will increment their current **combo** multiplier, then earn points equal to its value. The combo multiplier starts at 1 and increases by 1 for each consecutive turn where the player's action matches the diving goal. It also **resets** to 1 when the player's action does not match the diving goal.

Register	Description	Example
GPU	This run's diving goal	UUUDDLLLULDRLL
reg0	player 1 points	7
reg1	player 2 points	4
reg2	player 3 points	0
reg3	player 1 combo	1
reg4	player 2 combo	0
reg5	player 3 combo	9
reg6	*unused*	

Victory Condition

You have a higher **final score** after 100 turns.

Defeat Condition

Your program does not provide a command in the allotted time or it provides an unrecognized command.

Debugging tips

- Press the gear icon on the viewer to access extra display options.
- Use the keyboard to control the action: space to play/pause, arrows to step 1 frame at a time.

Game Protocol

Initialization Input

First line: playerIdx an integer to indicate which agent you control in the mini-games.
Next line: the number of simultaneously running mini-games. For this league it's 4.

Input for One Game Turn

Next 3 lines: one line per player, ordered by playerIdx. A string scoreInfo containing a breakdown of each player's final score. It contains 13 integers. The first integer representing the player's current **final score points** followed by three integers per mini-game: nb_gold_medals, nb_silver_medals, nb_bronze_medals.

Next nbGames lines: one line for each mini-game, containing the eight space-separated registers:

- gpu a string
- reg0 an integer
- reg1 an integer
- reg2 an integer
- reg3 an integer
- reg4 an integer
- reg5 an integer
- reg6 an integer

Their values will depend on the game. Unused registers will always be -1.

Output

One of the following strings:

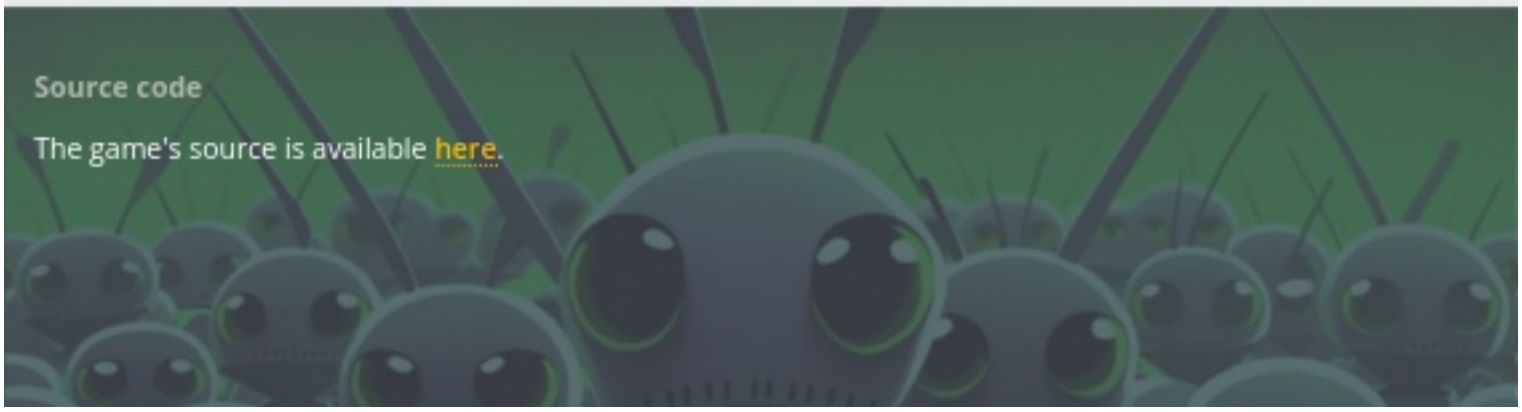
- UP
- RIGHT
- DOWN
- LEFT

The effect will depend on the game.

Constraints

0 ≤ playerIdx ≤ 2
1 ≤ nbGames ≤ 4 *(across all leagues)*

Response time per turn ≤ 50 ms
Response time for the first turn ≤ 1000 ms



Source code

The game's source is available here.