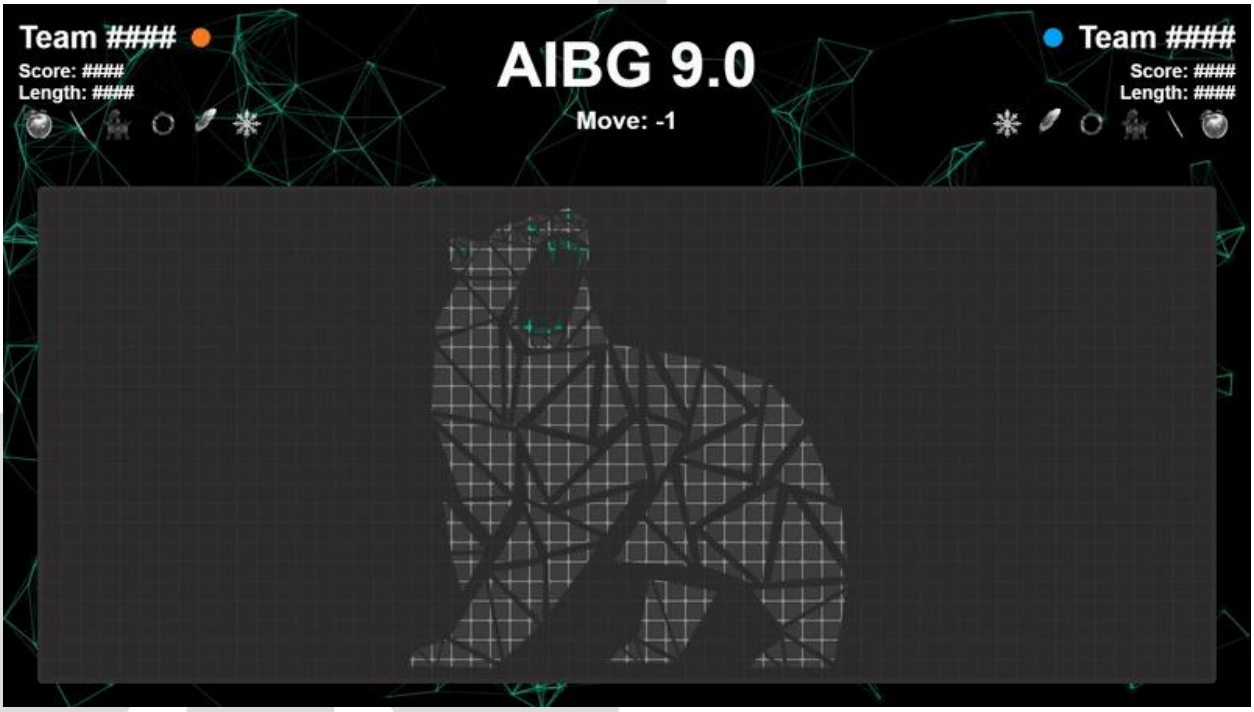
AIBG Topic manual

Introduction

Welcome to the ninth edition of BEST Zagreb's annual Artificial Intelligence Battleground hackathon. As you might have heard, this competition revolves around creating artificial intelligence agents that are used to play a competitive turn-based game for two players. After the programming phase (which lasts for 20 full hours), the agents will face off against each other and the teams that built the 3 highest ranking agents will receive monetary prizes. However, you're not doing this for the prize alone, as tech company representatives will observe and evaluate your work during the hackathon. Play your proverbial cards right and you might just land a job...

Please read this document in its entirety and pay special attention to section 2, General overview, as it contains explanations of (hopefully) everything relevant about the game's internal logic. Of course, if you have any questions, or find any bugs, you can ask the topic team, and we'll try our BEST (pun pun) to answer them.



Game mechanics

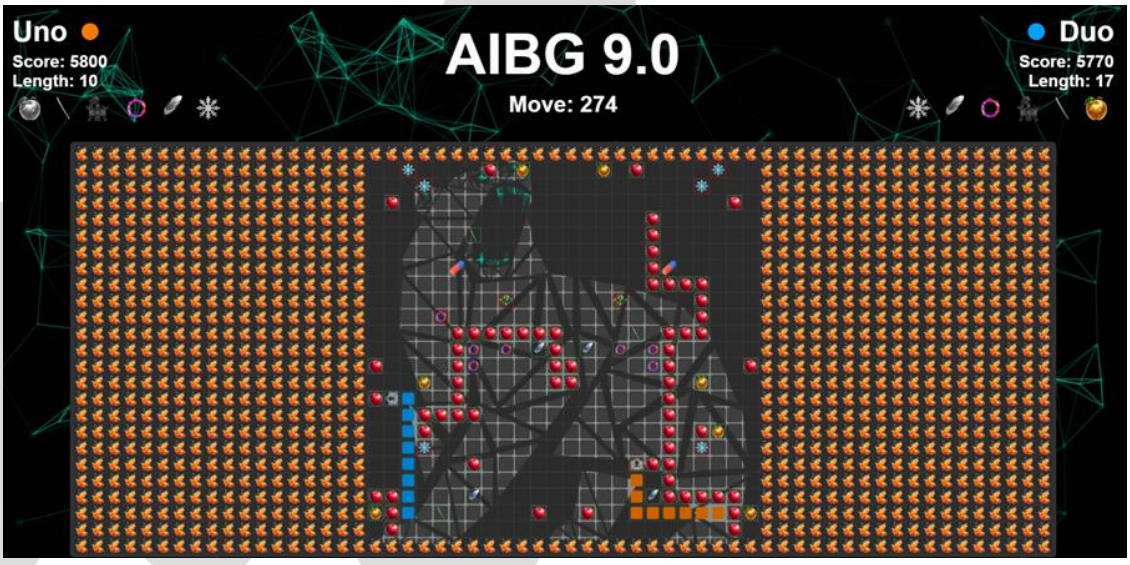
The concepts described here serve to provide you with a clear understanding of how the game itself works. Please read this section carefully, as we've tried to cover all relevant information. For any further questions, just ask away on discord.

General Overview

Each game is a head-to-head battle between two players, each controlling a snake.
Both players **start with a snake of length 9, positioned on opposite sides of a 25-row by 60-column board, facing toward the center. The side each player starts on is determined randomly at the beginning of the game.**
Every player begins with a **score of 1000 points.**

The board itself is dynamic: after 100 moves, the **borders begin to shrink** horizontally **every 10 moves**, and once the board reaches a square aspect ratio (1:1), it also starts shrinking vertically. This process continues **until the playable area is reduced to a minimum of 19 (rows) x 20 (columns)**.
If a snake's body is caught outside the shrinking border, the severed segments are converted into apples, which can then be collected for rewards.

Items, including apples and various modifiers, always spawn in mirrored positions on both sides of the board to ensure fairness.



Win Conditions

Victory can be achieved in several ways. The most direct is to force the opposing player to collide with a wall, their own body, or the other player's body. Any of these collisions results in an immediate win for the surviving player.
Alternatively, if a player's score drops to zero, they lose and the other player wins.

If both players reach zero simultaneously, or if both collide at the same time (such as a head-to-head collision), the winner is determined first by the higher score, then by the longer snake length if scores are tied. If both score and length are equal, the game is declared a draw.
Additionally, if the game reaches the maximum of 900 moves without a winner, the player with the higher score is declared the victor; if scores are tied, snake length is used as a tiebreaker, and if still tied, the game ends in a draw.

Rewards & Penalties

Players accumulate or lose points based on their actions and interactions during the game.

Moving towards the center of the board grants a reward of +20 points, while moving away from the center still provides a smaller reward of +10 points.
However, making an invalid move (such as an illegal direction or exceeding the move timeout of 150ms) results in a penalty of -50 points.
Attempting to reverse direction (moving directly opposite to the previous move) incurs a -30 points penalty.

Additionally, losing body segments, whether due to border shrinkage or certain item effects, costs the player -30 points per segment lost.

These rewards and penalties are designed to encourage strategic movement and penalize careless or risky play.

Items

Items play a crucial role in shaping the flow of the game. Besides apples, which spawn every 5 moves, other items have a 10% chance of appearing on any move. All items spawn in mirrored positions for balance.

Here's a breakdown of each item:

- **Apple** - The most common item, apples spawn every 5 moves. Collecting an apple increases the player's score by 50 and extends the snake's length by 1. Only affects the player who picks it up.
- **Golden Apple** - Grants a reward of 70 points and extends the snake's length by 5 over 5 moves (1 per move). Only affects the player who picks it up.
- **Katana** - Lasts for 10 moves and rewards 60 points. It can cut off the enemy's tail segments (unless they have active armour), turning those segments into apples and penalizing the opponent.
- **Armour** - Provides protection for 15 moves and rewards 60 points. While active, it negates the katana item effect. If the other player collides with the player who has the armour item active, it will result in a collision.
- **Shorten** - Instantly shortens a snake's body by a set amount (affecting either self, enemy, or both, determined randomly upon spawn) and grants 30 points.
- **Tron** - Lasts for 15 moves, rewards 50 points, and can affect self, enemy, or both (determined randomly at spawn). It leaves a trail behind the player for the next 15 moves, after which the whole trail is removed at once.
- **Freeze** - Freezes the enemy for 8 moves, preventing them from making moves, and grants 30 points.
- **Leap** - Lasts for 5 moves, rewards 80 points, and can affect self, enemy, or both. It allows the affected snake to leap forward by repeating the last move direction. You can use this to leap over the enemy in a *slippery* situation.
- **Nausea** - Lasts for 1 move, rewards 90 points, and affects the enemy, causing their next move to be determined at random. Mirrored nausea items have the same predetermined random direction.
- **Reset Borders** - Instantly resets the board's borders to their initial state, and grants 30 points. This affects the map rather than a specific player.

Picking up the same item while it is currently active resets its duration.

Please note that items can interact with each other and the effects can modify each other's behaviors. Each item's effect, duration, and reward are carefully balanced to add depth and unpredictability to the game.

Starting the game

Game Flow:

1. Start the server first
2. Connect two clients using valid player IDs
3. The game starts automatically when both players are connected
4. Server will close automatically when the game ends

Prerequisites

1. Install Node.js and npm (for server and JavaScript clients)
2. Install Python 3.7+ (for Python clients)
3. Install an IDE with Live Server extension (for visuals)
4. Install required dependencies:
    - For server:
      cd server
      npm install

    - For JavaScript client:
      cd clients
      npm install

    - For Python client:
      pip install websockets

If you get any error during the starting of the game, check if you have the correct (usually latest) versions of the runtimes installed, and be sure to ask an LLM for help as the problem is most likely on your computer so we cannot debug it easily.

Running the Server

Create a players.json file in the server directory using the example:

- Copy players.json.example to players.json
- Edit player IDs and names as needed

Then start the server:
cd server
node server.js [port]

The server will run on port defined on start or 3000 by default.

Note that the **server timeout is turned off by default** for testing purposes but will be **turned on during the tournament**.

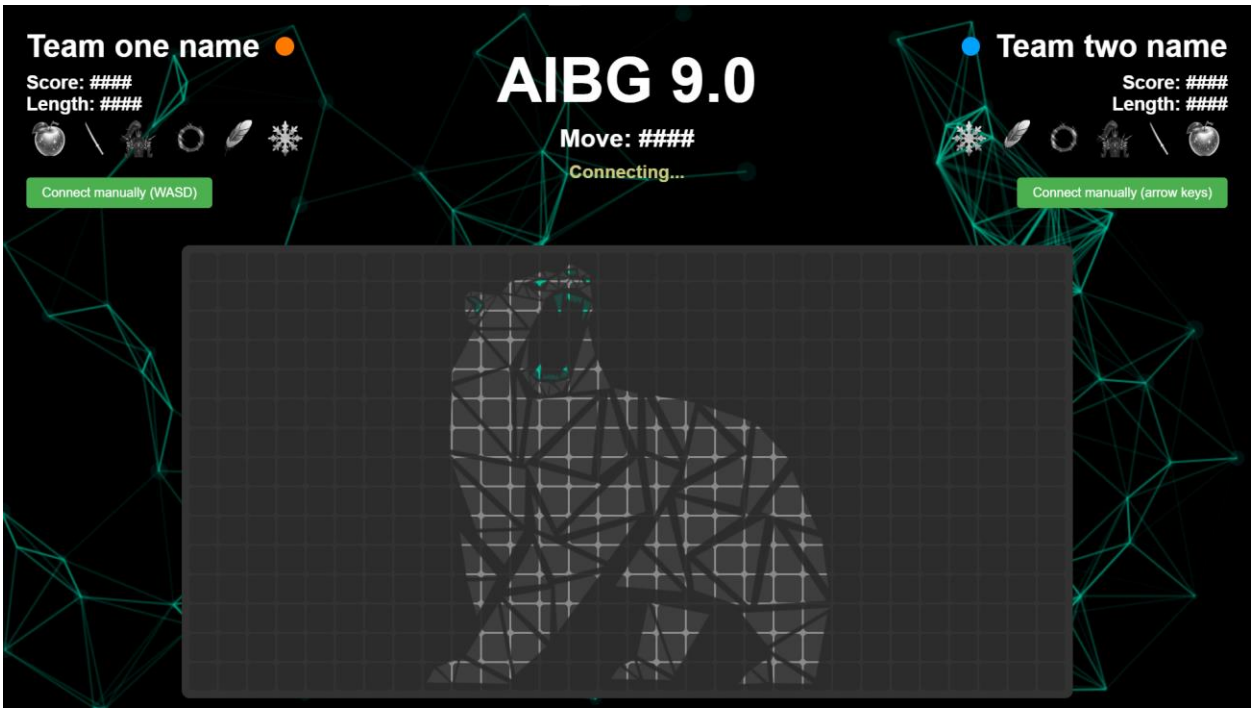
Running the Visuals

Open the project in your IDE (we recommend VS Code or Trae AI). Right-click on visuals/index.html and select "Open with Live Server". If you don't see this option, install the "Live Server" extension first. The game visualization will open in your default browser. The visuals will automatically connect to the server when it's running.

After every processed move the **game state is displayed in the browser console** in inspect mode. You can use this to understand the game better, and to copy example game states to test out your agent.

Debug mode

There is a special **debug mode** integrated that is accessed using the mode=debug query in the visuals URL (*http://127.0.0.1:5500/visuals/index.html**?mode=debug***) with which you can connect manually with ids “k” and “l” to test out the game.



Running example agents

JavaScript Client

Run the JS client (agent.js):

node clients/agent.js [playerID] [mode]

- playerID: Your unique player ID (default: "k")
- mode: Game mode (default: "up")
    - "up", "down", "left", "right": Goes only in that direction
    - "random": Makes random moves
    - "timeout": Progressively increases delay between moves
    - "apple": Seeks and moves toward the nearest apple while avoiding obstacles
    - "survive": Focuses on avoiding collisions and staying alive

Python Client

Run the Python client (agent.py):

python clients/agent.py [playerID] [mode]

- playerID: Your unique player ID (default: "k")
- mode: Game mode (default: "up")
    - "up", "down", "left", "right": Goes only in that direction
    - "random": Makes random moves

- "timeout": Progressively increases delay between moves
- "apple": Seeks and moves toward the nearest apple while avoiding obstacles
- "survive": Focuses on avoiding collisions and staying alive

Connecting your own agents

In the clients folder we have provided basic agents in JS (simpleAgent.js) and Python (simpleAgent.py) you can use as a starting template from which you can build your AI without the hustle of trying to figure out the connections.

In addition to that, there are also more advanced aforementioned agents that use basic algorithms to win the game against whom you can test out your agent.

You may, and we encourage you to, **modify the provided code** in any way you want to test out the functionality of different aspects of the game, and to make it easier to train your agents (such as auto-restarts). Unfortunately, if you have some problems or bugs with the modified code **we cannot help you debug it** so you will have to do it all by yourself, but feel free to **ask for possible modifications** of the server or the visuals and we will consider updating it during the hackathon.
Do note that the game will be **run on the provided code** so any changes you made to it will not be available during the tournament.

The following sections provide a more detailed explanation of the communication between the server and agent.

Connecting to the Server (WebSocket)

Server address when started is *ws://localhost:3000* (for the tournament the agents will have to connect to *ws://topic.aibg.best.hr:3000*).
Agents must connect with a unique player ID (defined in players.json). Example connection URL: *ws://localhost:3000?id=YOUR_PLAYER_ID.*

Initial handshake

On connection, the server responds with a JSON message:

```
{
  "message": "Player connected successfully.",
  "id": YOUR_PLAYER_ID,
  "name": YOUR_TEAM_NAME
}
```

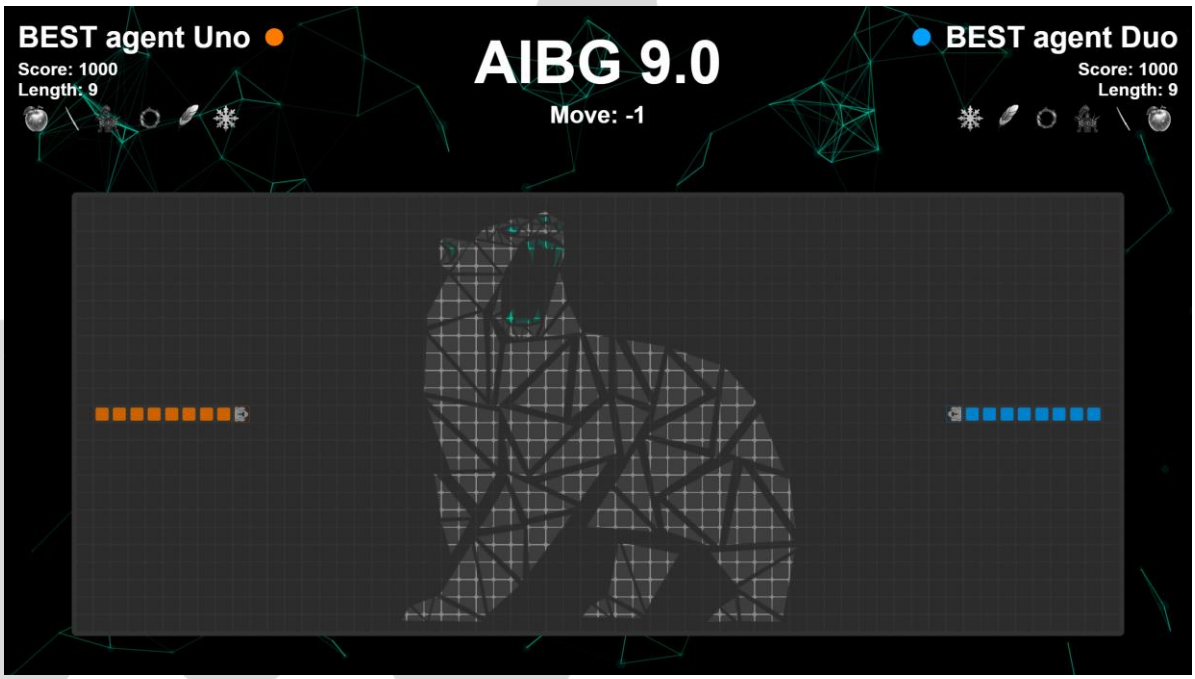
Note that the **player name** received in this connection message is **not the same as the player id**, and it is **how you will discern your agent in the game state**.

If the ID is invalid or already in use, you'll receive an error message and the connection will close.

Please also note that your player ID is secret and only known to your team and must not be disclosed to any other teams. If other teams find out your ID they can sabotage your inputs and make you lose the game. If you think that your ID was compromised, ask the topic team for a new one.

Game state updates

Once both players are connected, the server sends the game state as a JSON object after every move.



Game state outline:

```
{
  "map": [[null, ...], ...], // 2D array representing the board
  "players": [
    {
      "name": "Team K",
      "score": 1030,
      "body": [{"row": 5, "column": 3}, ...],
      "activeItems": [...],
      “lastMoveDirection”: “up”,
      “nextMoveDirection”: “frozen”,
    },
    {
      "name": "Team L",
      "score": 1050,
      "body": [{"row": 7, "column": 3}, ...],
      "activeItems": [...],
      “lastMoveDirection”: “left”,
      “nextMoveDirection”: null,
    },
    ...
  ],
  "moveCount": 420,
  "winner": null // or player name or -1 for draw
}
```

For a more thorough example, check the accompanying gamestateExample.json that represents the game state, or simply test out the game with the debug mode (the game state is printed out in the console after every move).

Sending moves

Each agent must send a JSON message for each move in the following format:

```
{
  "playerId": "k",
  "direction": "up" // or "down", "left", "right"
}
```

Only “up”, “down”, “left”, and “right” directions are valid. If you send an invalid direction or omit it, or your agent takes more than 150 ms to send to move, the server will penalize your agent.

Receiving game state

After both players send their moves, the server processes them and sends the updated game state to all clients and the connected frontend.

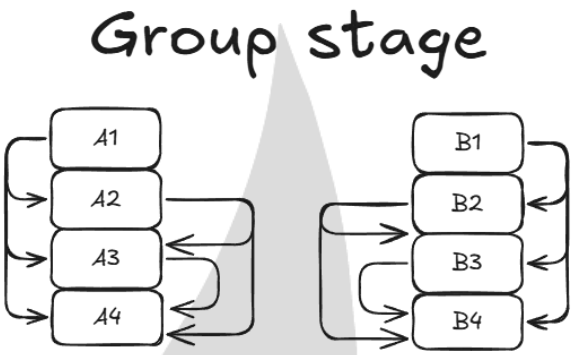
If the game ends, the winner attribute will be set (player name, or -1 for draw).

Tournament

Each matchup in both the group stage and the knockout stage will consist of **3 rounds**. The team that wins **2 or more rounds** will be considered the winner of the matchup.
In the group stage, this result determines how points are awarded. In the knockout stage, it determines which team advances to the next round.

Group stage

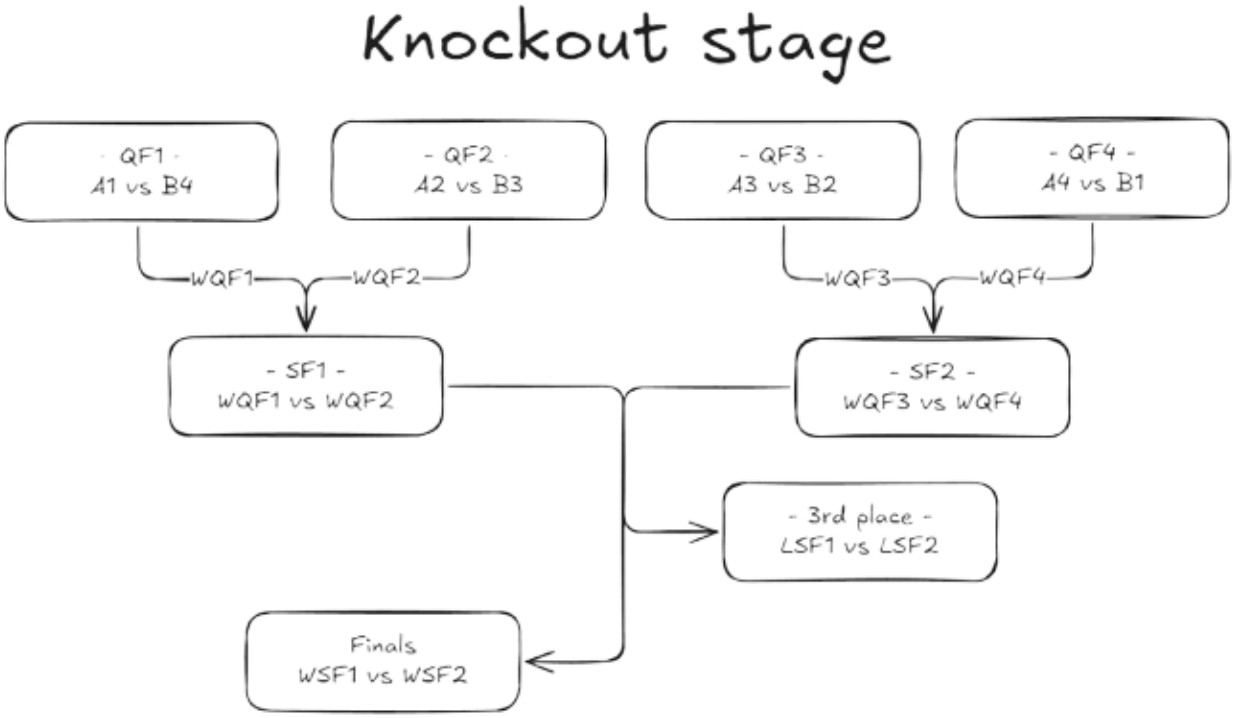
The tournament begins by randomly dividing the 8 teams into two groups of four, called **Group A** and **Group B**. Within each group, every team plays one match against each of the other three teams. Teams earn points based on their performance in these matches: **3 points for a win, 1 point for a draw, and 0 points for a loss**. After all group matches are completed, the teams in each group are ranked from first to fourth based on their total points.



Knockout stage

Although extremely unlikely, in the event of a tie in a knockout match, the winner will be determined by selecting a representative from each team to play the game manually in debug mode, with the server timeout set to 1 second. The victor of this sudden-death match will advance.

Here's a simple diagram representing the matchups of the knockout stage:



Quarter-finals

Once the group stage is completed, all eight teams progress to the knockout phase, starting with the quarter-finals. To reward strong group-stage performance, the top-ranked teams face the lower-ranked teams from the opposite group. The matchups are as follows:

- 1st in Group A vs 4th in Group B
- 2nd in Group A vs 3rd in Group B
- 1st in Group B vs 4th in Group A
- 2nd in Group B vs 3rd in Group A

Semi-finals

The winners of the quarter-final matches progress to the semi-finals. In this round:

- The winner of A1 vs B4 plays the winner of A2 vs B3
- The winner of B1 vs A4 plays the winner of B2 vs A3

These matches determine which two teams advance to the final.

Third place match

The two teams that lose in the semi-finals play one more match against each other to determine which team finishes in **third place**.

Finals

The winners of the semi-final matches face each other in the final to determine the **tournament champion** and the **runner-up**.

Code submission

At the end of the 20-hour hackathon, you will need to submit your source code to the provided google drive folder in your perspective folder.

Final words

We wish you luck! Remember: the first rule of AIBG is to have fun and be yourself!

- AIBG 9.0 Topic team, BEST Zagreb