DEVELOPER GUIDE

Subject

The subject of developer is to implement the bot engine for simple soccer game using javascript. It does not require the expert skills in javascript. You have to know the basic statement constructions and standard math function (https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Math).

SDK Structure

SDK has the following folder structure:

doc	
developer-guide.pdf	This document
node_modules	Do not provided, generated after 'npm install' command
public	Root for game web site
Assets	Assets folder
Audio	
Css	
img	
Js	
engine.js	Game engine
game.js	Game controller used to orchestrate the engine and bots
renderer.js	Position renderer
config.js	Config file for game engine, see "Game Settings" section
defaultBot.js	The ready-to-use bot engine used as opponent to debug
favicon.ico	
index.html	
userBot.js	The main implementation of your bot
index.js	
package.json	npm project file for node.js developers

How to start

There are two basic ways to run the project. You can use Node.js infrastructure or your favorite http server.

To developing using Node.js please download and install the latest Node.js at https://nodejs.org/en/ . Then open main project folder in command line and run the following commands:

```
npm install
```

```
npm start
```

The `npm install` downloads and installs all project dependencies required to start to the `node_modules` folder. You should run it only once.

The `npm start` runs the http server with the game engine `public` folder. Then you can open http://localhost:3500 in a browser to see the game. Note: you can change the port at the index.js file if you have port conflict.

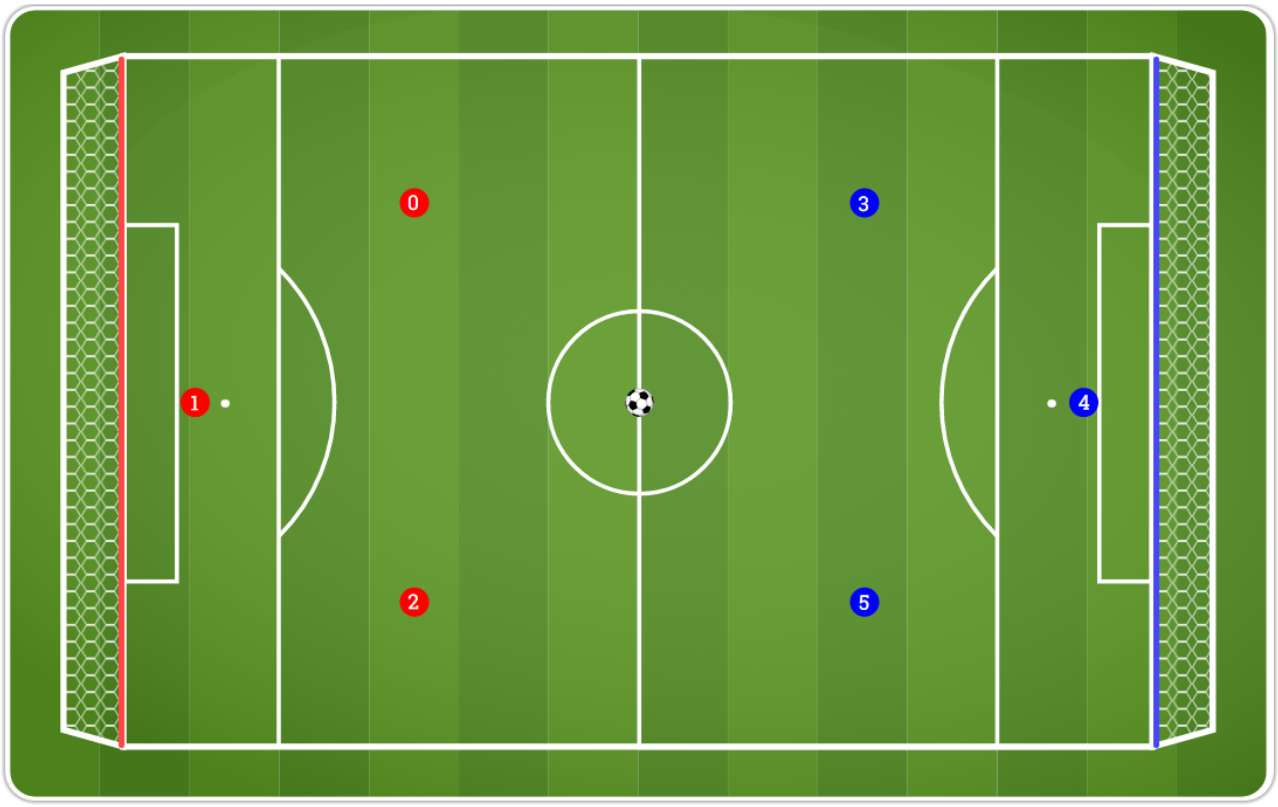
The second way (if you don't want to use Node.js) is just to start your favorite http server (apache, nginx, IIS, IIS Express etc.) using the `public` project folder. For example, a Visual Studio developers can run the IIS Express from the command line:

```
"%ProgramFiles(x86)%\IIS Express\iisexpress.exe" /path:[full path to public folder] /port:3500
```

Then game engine become available at http://localhost:3500

Rules

The soccer game is a real-time game for two teams (red and blue) of three players each. The start position is below:



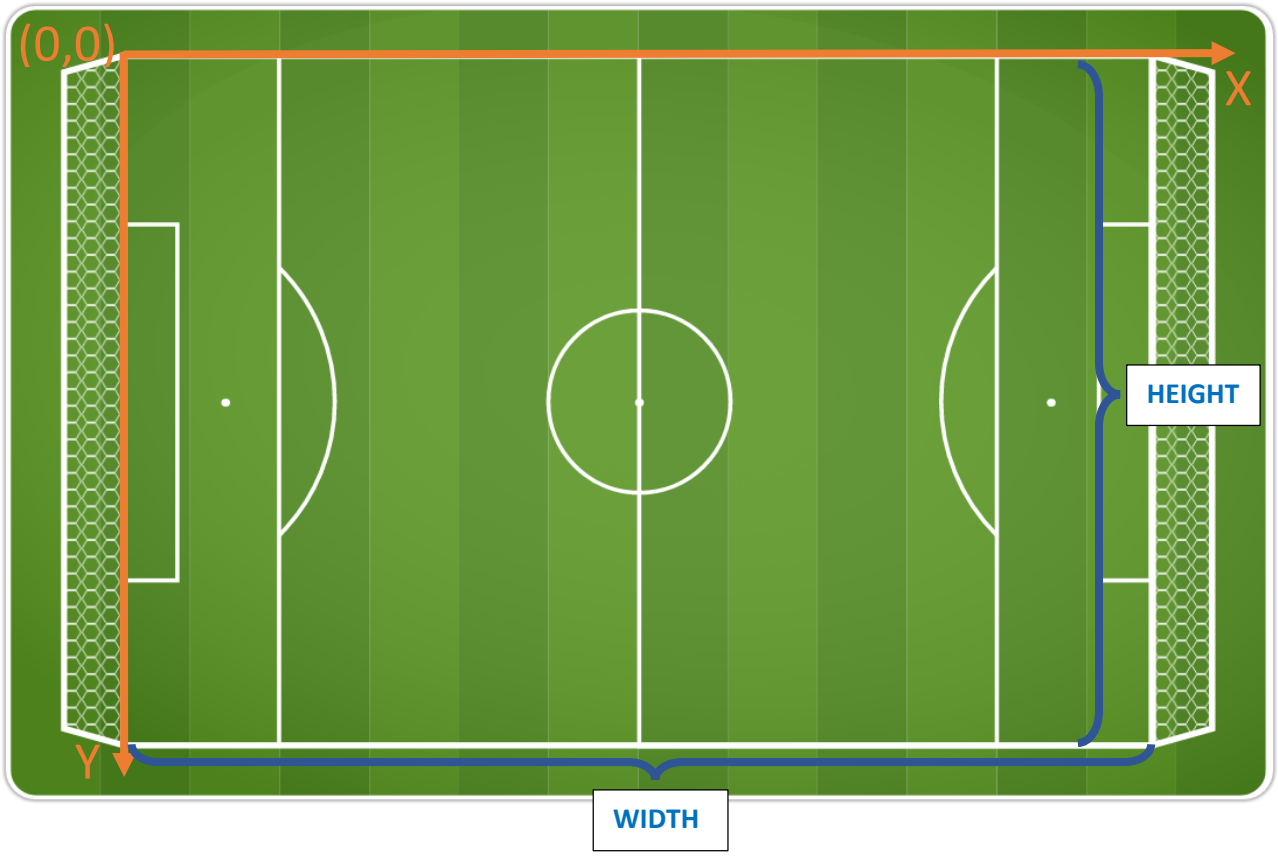
The objective of soccer is to score goals by advancing the ball down field into the opposing team's goalpost. The goalpost takes the whole left or right side (is marked with red and blue line on the picture above) to simplify the rules.

Game last for a total of 2 minutes and are divided into two halves of 1 minute. After the first half (1 minute) the teams are change the sides. The left side team always has the first move.

The game time is split into ticks. The tick last 20ms and on each tick only one team player can change its velocity and direction. If bot does not send the update for required player in 20ms then game engine do not change the properties of player and next tick is started. At the first tick players 0 and 3 can update the move, at the second – players 1 and 4, at the third – 2 and 5. And then again: 0 and 3, 1 and 4, 2 and 5.

The Field

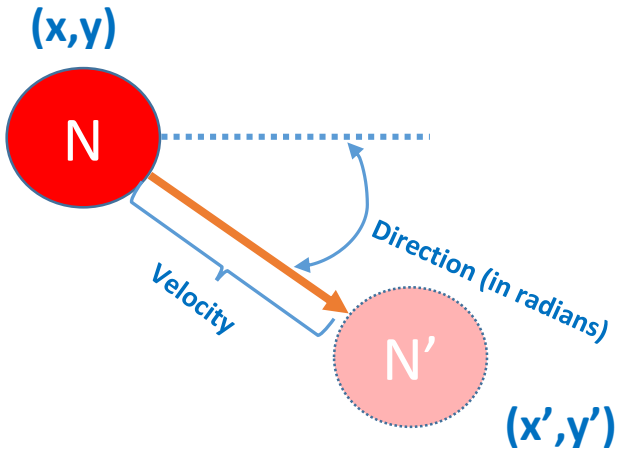
The field is 2D grid with the following system of coordinates:



The origin of this grid is positioned in the top left corner at coordinate (0,0). All elements are placed relative to this origin.

The Player

Each player has velocity and direction that describe the vector of movement. The (x,y) coordinates are related to the center of the player circle.



The next position is calculated by velocity and direction.

$$x' = x + velocity * \cos(direction)$$

$$y' = y + velocity * \sin(direction)$$

The new coordinates are calculated on every tick for each player and ball. The player velocity and direction can be changed by bot only (one team player for tick, as noticed above). Player has no acceleration and deceleration.

The Player movement restrictions

Each player has the momentum. It means it can change the velocity gradually only, increasing and decreasing velocity tick by tick. All players have the maximum allowed velocity. Changing the direction depends on a velocity. If velocity is low, then player can change the direction by more angle. If velocity is high, then player can update the direction in a low range. The restriction formula that describe that direction change is inversely proportional to velocity is

$$\Delta\, direction < \frac{\pi}{1 + velocity} = \Delta_{max}$$

This is the simple artificial formula that is describe the direction momentum of player.

Note. If bot returns the large values of velocity and direction, then game engine automatically corrects these values to the maximum allowed. No exception, error or penalty are occurred. It means you can simplify the bot strategy and use any values, but game engine will track the player's momentum.

If strategy does not update the player velocity and direction, then old values are used. It means player has uniform motion, game engine never stops or accelerates the players. (But player can start to move if another player kicks it!)

The real values of velocity limits are specified below at the "The Bot interface" section.

The Ball physics

The ball movement also has some limits and features. The ball has a velocity and direction as well as a player. Player movement is uniform motion but ball moves using accelerated motion. Ball has deceleration due to physical friction and resistance. From the other side ball has one-time acceleration if player kicks it.

Ball also has a maximum velocity. The real values of velocity limits, kick acceleration and deceleration are specified below at the "The Bot interface" section.

Kicks (Collisions)

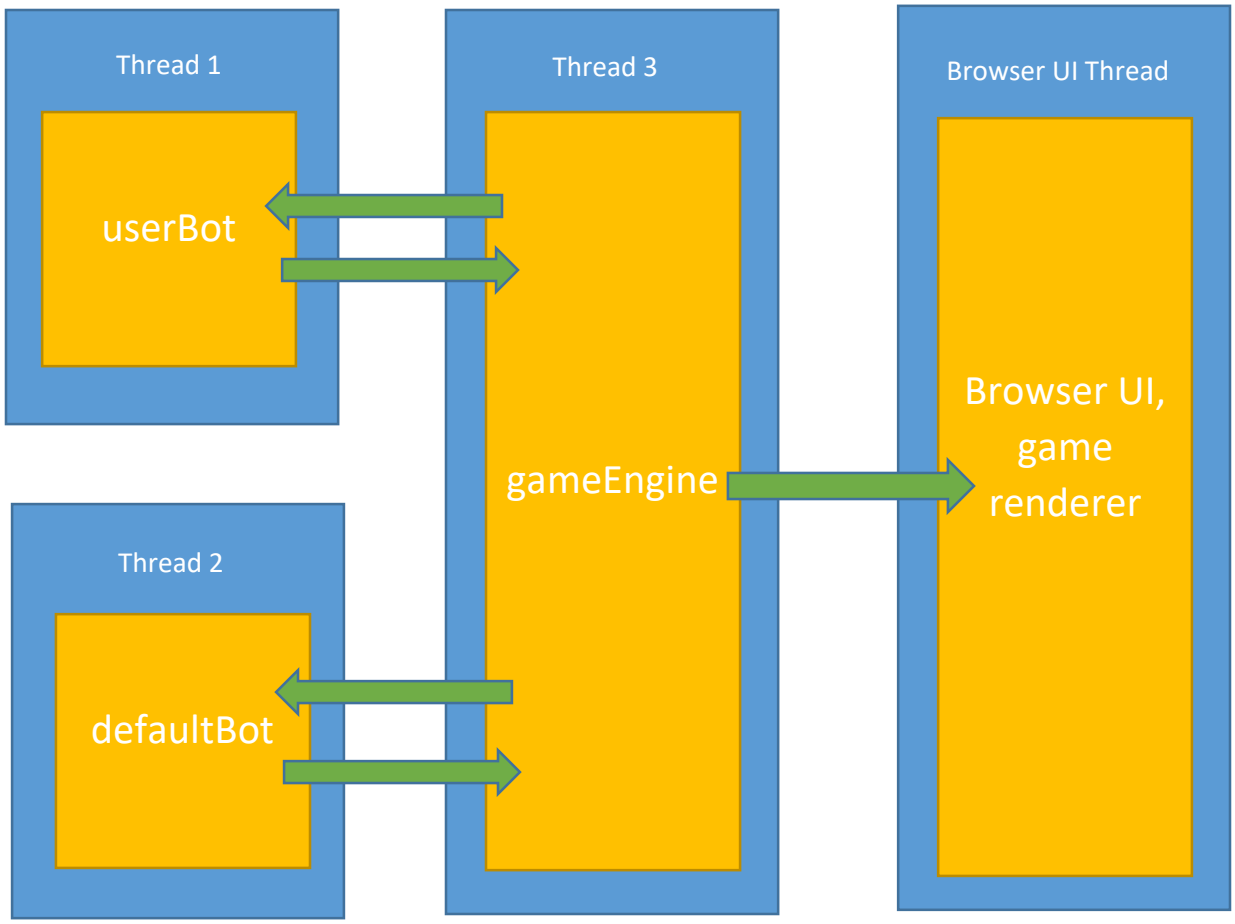
Collisions between players and balls are processed according to simple physics laws using the elastic collision algorithm (https://en.wikipedia.org/wiki/Elastic_collision#Two-dimensional). If you are interested in details, please refer to the function `collisionDetection` at `/public/assets/js/engine.js`.

Collisions with the field sides are determined by the Law of Reflection (the angle of reflection is equal to the angle of incidence). To simplify the game strategy no energy is lost during the collisions with field sides, game engine changes the direction of solid only.

If player kicks the ball, then ball gets the extra kick acceleration.

The Bot interface

The bot is the algorithm to play the soccer team against opponent. The sample of bot is provided at `public/userBot.js` and opponent is `public/defaultBot.js`. Each bot and game engine run in a separate thread (webworker).



Each bot gets a message from game engine when tick starts about the current position and team player number (`playerIndex`) that can update the motion. During this tick (20ms) bot can send back message to the engine with updated velocity and direction for specified player. After 20ms engine calculates the new positions of players and balls (processes collisions, check goals etc.) and send the message to game renderer to re-draw the position. Then new tick starts and engine sends this position to each bot. If bot does not send the update during a tick period, then bot message is ignored by engine.

The minimal simple bot that do nothing is:

```
function getPlayerMove(data) {
    const currentPlayer = data.yourTeam.players[data.playerIndex];
    return {
        direction: currentPlayer.direction,
        velocity: currentPlayer.velocity
    };
}
onmessage = (e) => postMessage(getPlayerMove(e.data));
```

The last line is required to process the communication between bot and game engine. `onmessage` handler receives the message `e` from the game engine and send the result of `getPlayerMove` function to engine via `postMessage`. You have to implement the bot strategy inside `getPlayerMove` function.

The message from engine has the following structure (JSON format):

```
{
    "type":"playerMove", // type of message
    "settings":          // settings data
    {
        "field":         // soccer field
        {
            "width":708, // field width
            "height":473 // field height
        },
        "player":   // player settings
        {
            "maxVelocity":6, // max velocity of player
            "maxVelocityIncrement":0.5, // max allowed increment for velocity
            "radius":10,     // radius or player's round
            "mass":100       // player mass for collision processing
        },
        "ball":   // ball settings
        {
            "maxVelocity":16,  // max velocity of ball
            "radius":10,       // ball radius
            "mass":5,          // ball mass for collision processing
            "moveDeceleration":0.3, // decreased from ball velocity every tick
            "kickAcceleration":8    // increased ball velocity every kick
        },
        "periodDuration":30000, // half-time duration (in ticks)
        "tickDuration":20,      // tick duration (in ms)
    },
    "playerIndex":1,   // player to change the velocity and direction (0,1,2)
    "tick":1,          // # of game tick
    "yourTeam":        // your team data
    {
        "type":"guest",  // "home" – left side player,"guest" - right side
```

```
    "goals":0,       // goals count
    "players":[      // array of players positions
        {
            "x":200.36287699916477,  // x coordinate of player's center
            "y":100.34397715545828,  // y-coordinate of player's center
            "direction":0.7586665645776556, // motion direction (in radians)
            "velocity":0.5                  // motion velocity
        },
        {
            "x":50,
            "y":236.5,
            "direction":6.283185307179586,
            "velocity":0,
        },
        {
            "x":200,
            "y":373,
            "direction":6.283185307179586,
            "velocity":0,
         }
    ]
},
"opponentTeam":  // opponent team data
{
    "type":"home",  // "home" - left side player, "guest" - right side
    "goals":0,      // goals count
    "players":[
        {
            "x":507.63712300083523,  // x-coordinate of player's center
            "y":100.34397715545828,  // y-coordinate of player's center
            "direction":2.3829260890121375, // motion direction (in radians)
            "velocity":0.5                  // motion velocity
        },
        {
            "x":658,
            "y":236.5,
            "direction":3.141592653589793,
            "velocity":0
        },
        {
            "x":508,
            "y":373,
            "direction":3.141592653589793,
            "velocity":0
        }
    ]
},
"ball":  // ball data
{
    "x":354,   // x-coordinate of ball center
    "y":236.5, // y-coordinate of ball center
    "direction":1.5707963267948966, // motion direction (in radians)
```

```
        "velocity":0                    // motion velocity
    }
}
```

Note: The ball and each player object also has a `settings` reference that are identical to `settings.ball` and `settings.player`.

The message from bot to engine has the following structure (JSON format):

```
{
    "direction":1.5707963267948966, // new motion direction (in radians)
    "velocity":0,                   // new motion velocity
}
```

The update data belongs to the player specified as `playerIndex` in the incoming message.

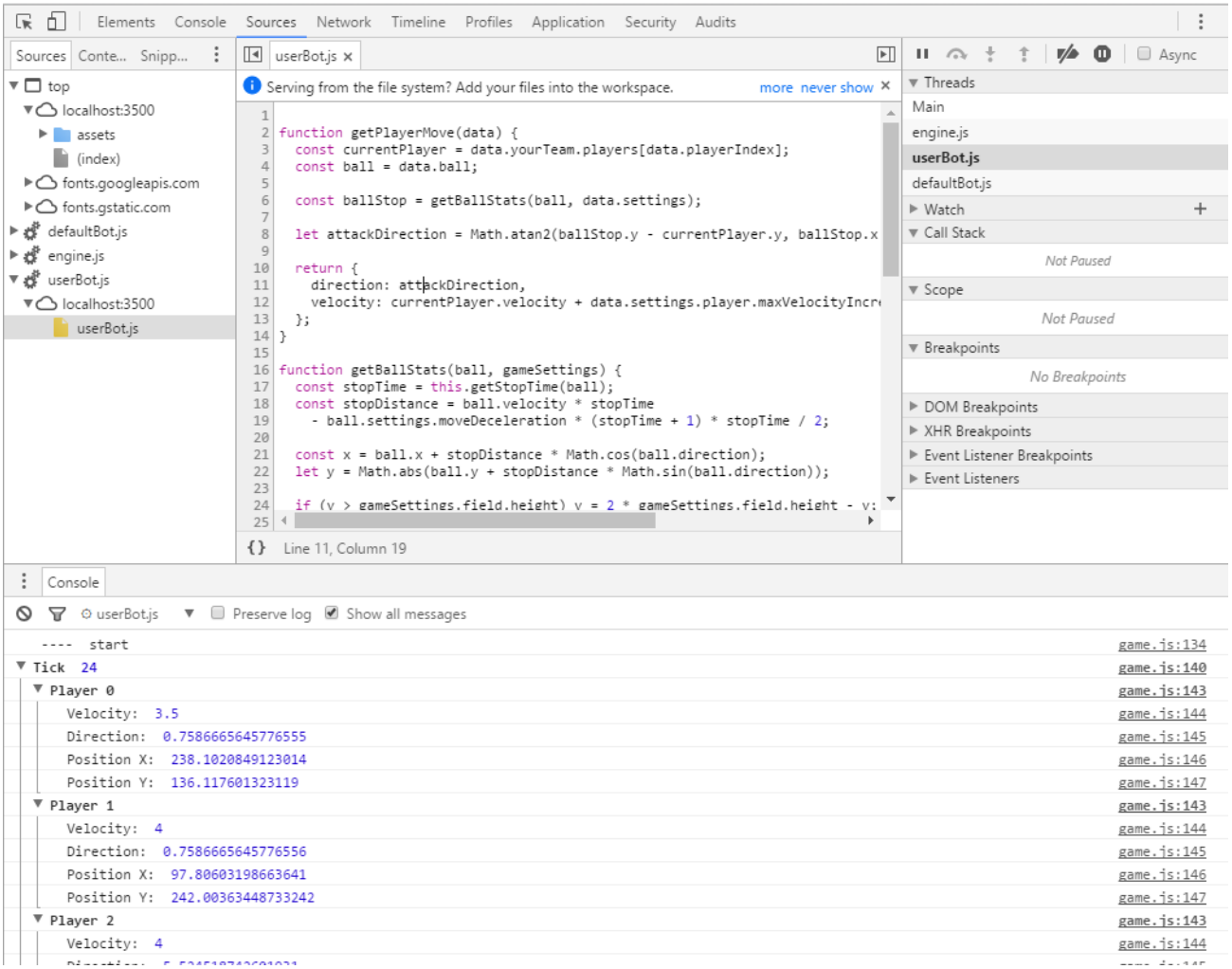
> **!** Please take an attention the engine automatically always translates the team position for bot to use left side. It made for developer convenience to avoid the routine tasks to normalize position and concentrate on strategy only.
> It means that your goalpost is always translated to the left side (`x=0`), and the opponent goalpost is always at the right side (`x=data.settings.field.width`)

Debugging

It is highly recommended to use the Chrome browser for testing and debugging. Open a Developer Tools and click **Sources** tab and select the userBot.js source file. In the source code window you can set the breakpoints and inspect the values of variables. (See screenshot below)

The main html document also has a several options to debug and testing. At first every player has a number (aka `playerIndex`) and visualized vector of velocity and direction. You can use

- **START** button to run game in real-time mode
- **NEXT STEP** button to debug game in step-by-step mode
- **Debug** switch to log the position into DevTools console
- **X1**, **X0.5**, **X0.25** toggles to decrease the real-time speed



You can use `console.log("…Debug info…")` instruction to save the debug into log. You can see the local log at DevTools console and download the log at game page for challenge matches. The size of logs at game server is limited to 1.5 Mbytes for each match.

Game Settings

Game settings are placed at `public/config.json` file. These settings will not be changed during the completion. Settings are passed to the bot per every tick in the engine tick message. You can change the `bot1` and `bot2` values to exchange the side of teams, to test several bot implementations or play locally against colleages.

Submitting the Bot

To submit bot please open https://aichallenge.lab.epam.com/mybot and copy-paste the content of `public/userBot.js` into the text area or drag-n-drop this file. Click Submit button to send the bot implementation to AI Challenge server.

3rd party Libraries and Frameworks

Competition participants are not allowed to include the 3rd party libraries or frameworks in a bot strategy. The bot source code has to be single flat file with size above 10Mb without any dependencies. It means that using javascript `require` or `import` statements is forbidden. Using network, file system or other input-output operations are not allowed. Using the plain vanilla javascript is the only way to make the starting conditionals equal to all participants (who are expert in javascript and who are not).

Fair Play

Players who are deemed to violate the spirit of fair and sportsmanlike competition will be disqualified from competition without any opportunity for appeal. In particular, memory scanning, intentionally losing games, and behavior conditional on the opponent's identity are prohibited.

All types of strategies that deal with visible player's motion are allowed. You can kick the ball, kick your players, kick opponent players, keep all players at goalpost etc. You can use the source code from game engine to predict the future position. These strategies are visually recognized by other participants. Other participants can implement the contra-strategy against you.

> **!** Participants can exploit some game engine funny behavior (for example very nice dribbling feature to move the ball around the player, can you find it?). We intentionally do not fix some nice features that leads to more spectacular matches and attracts more fans.

The fixed matches are not allowed. Game engine do not provide you with the name of opponent to prevent the any kind of fixed games.

We do not restrict the number of developers under the one account. It is allowed to create the bot strategy by the group of colleagues. But login account person will be rewarded only.

Disclaimer

Please be aware of the performance of your bot. If your bot can send messages in 20ms on your local box it does not guarantee that it can do the same on game engine server. Game engine server has a lower CPU capacity. We use the 4-cores Intel i7 CPU boxes to process the games but we run several games simultaneously on the same box at the same time. It affects the real performance. The number of parallel games depends on number of players because we should complete all processing inside reasonable time frame.

All claims about the performance of engine boxes will be rejected. Please make your bot as faster as possible.