CS 386: Lab Assignment 1

(TAs in charge: Siddharth A. and Krishna Murthy Bukkapatnam)

Half Field Offense (HFO) is a multi-agent task that is a part of RoboCup soccer. As apparent from the video below, it is a simplified version of soccer played on one half of the soccer field between an offense team and a defense team. Each coloured circular disk is a soccer player, controlled by a separate client program. The clients connect and communicate with a server, periodically getting snapshots of their state and specifying actions. The server implements the physical simulation, and applies the rules of soccer. The HFO simulator allows us to simulate and evaluate several offensive/defensive strategies involving teams of different sizes.

Demo video

In this assignment, we shall focus solely on the **behaviour of an offense player when it is in possession of the ball**. When not in possession of the ball, offense players play a fixed strategy. The defense players, too, follow a fixed, programmed, strategy. When the offense team does not have possession of the ball, at least one player goes straight to the ball in order to get possession. What must it do once it has possession?

The player represents its state through features such as distances and angles between itself and other players, as well as with static objects such as the goal. The actions available to the player are DRIBBLE, SHOOT, and PASS(k), wherein k is the index of an offense teammate (if there are any). Your task is precisely to code the mapping between state and action: such a mapping is called a *policy*.

Detailed descriptions of the state and action spaces are mentioned in the sections below. A sequence of steps is provided to get you up and running. You will submit agents for playing 1 versus 1 (1v1) and 2 versus 2 (2v2) HFO.

Code

HFO-master.tar.gz

Installation and Execution

First, be sure to be connected to the Internet (log in through https://internet.iitb.ac.in/).

1. Download the tar file and extract it.
2. Run the following commands inside the `HFO-master` directory for installation.
    1. `mkdir build && cd build`
    2. `cmake -DCMAKE_BUILD_TYPE=Release ..`
    3. `make -j4`
    4. `make install`
3. Test the installation by running 1v1 and 2v2 simulations; run the following commands inside the `HFO-master` directory.
    1. `./bin/HFO --offense-npcs=1 --defense-npcs=1 --no-sync`
    2. `./bin/HFO --offense-npcs=2 --defense-npcs=2 --no-sync`

    You must see 1v1 and 2v2 games being simulated.

Here are some of the relevant command line options for running HFO.

- `--no-sync`: Run the simulations in real-time, and not in "synchronised" mode. This option is useful for visualising behaviours. Running with sync-mode on is significantly faster, and can be useful for testing the performance of your agent.
- `--trials=[num_trials]`: Specify the number of episodes by adding this flag.
- `--fullstate`: This option removes noise in the agent's state features, making decision making easier for the agent. Your agent will indeed be tested with full state (without state noise).
- `--no-logging`: No logs will be generated.
- `--offense-agents=[num_agents]`: The server will wait for `num_agents` to connect to it before starting the game.
- `--defense-npcs=[num_agents]`: The server will spawn `num_agents` defense npcs.

You can kill simulations by pressing CTRL+c in the terminal running the server.

States and Actions

The agent's state is represented using a feature vector, which consists of 10 + 6T + 3O floating point numbers, where T is the number teammates (one less than the number of offense players), and O is the number of opponents. The features are listed below in sequence.

Index	Feature	Description
0	x position	Agent's global x coordinate
1	y position	Agent's global y coordinate
2	Orientation	The global direction the agent is facing
3	Ball x position	Ball's global x coordinate
4	Ball y position	Ball's global y coordinate
5	Able to Kick	Boolean indicating if the agent can kick the ball
6	Goal centre proximity	Distance between agent and goal centre
7	Goal centre angle	Angle between x axis and line connecting agent to goal centre
8	Goal opening angle	The magnitude of the largest open angle (between opponents) of the agent to the goal
9	Proximity to opponents	Proximity to the closest opponent
10–(10+T-1)	Teammate goal opening angle	For each teammate, its goal opening angle
(10+T)–(10+2T-1)	Teammate proximity to opponent	For each teammate, its proximity to its closest opponent
(10+2T)–(10+3T-1)	Teammate pass opening angle	For each teammate, the magnitude of the open angle available to pass the ball to the teammate
(10+3T)–(10+6T-1)	Teammate global x position, teammate global y position, teammate uniform number	Three consecutive features for each teammate: global x position, global y position, and uniform number
(10+6T)–(10+6T+3O-1)	Opponent global x position, opponent global y position, opponent uniform number	Three consecutive features for each opponent: global x position, global y position, and uniform number

All non-boolean features except the uniform numbers are normalized to the range [-1, 1]. It is possible that occasionally a feature comes marked as "invalid". If so it is given a value of -2.

The following are the actions available to the agent. These are all "high-level" actions, which are implemented by the agent using lower-level skills (which we need not bother with for this assignment).

1. SHOOT: This action executes the agent's best possible shot to the goal.
2. PASS(k): Passes the ball to the teammate with uniform number k.
3. DRIBBLE: Advances the ball towards the goal using a combination of short kicks and moves.

Observe that for 1v1 HFO, the number of state features is 13 and the number of actions is 2 (PASS is not valid). For 2v2 HFO, the number of state features is 19 and the number of actions is 3. The policies you program need not use all the features available; take some time to think which ones could matter the most.

Task 0 (Ungraded)

This task is to familiarise you with the simulation, the state variables, and the task facing the offense player with the ball. Specifically, look at the `getAction()` function in 1v1agent.cpp. It is given the state features as input; as output it picks a legal action uniformly at random. Verify by running autograder-1v1.sh that this strategy yields a scoring percentage of approximately 5%.

Use the `--no-sync` flag to observe the behaviour of the offense player with the ball. Print out a few relevant state features to observe how they vary as the configuration of players on the field changes. Change `getAction()` such that the action chosen depends on some state feature(s). Visualise the resulting policy.

Task 1 (4 marks)

In this task, you have to program a 1v1 offense agent to maximise its scoring percentage. Revise the `getAction()` function in the file 1v1agent.cpp. Follow these steps to experiment with your code.

1. Start the server by running the following command from the `main` directory.

   ```
   ./bin/HFO --offense-agents=1 --defense-npcs=1 --trials=1000 --fullstate --no-logging --no-sync
   ```

2. In a new terminal from the `example` directory, create the executable by running the following command.

   ```
   ./make_executable1v1.sh
   ```

3. Spawn the agent by running the following command.

   ```
   ./1v1agent --numAgents 1
   ```

To check performance of your agent, run `./autograder-1v1.sh` (in directory `example/`). This script will print out the fraction of episodes that resulted in a goal.

Task 2 (6 marks)

In this task, you have to program a 2v2 offense agent to maximise the scoring percentage of the offense team. Revise the `getAction()` function in the file 2v2agent.cpp. By running the steps below, you will essentially run both offense players with the same binary (albeit the players will have different uniform numbers). Hence, both players will execute the code you fill into `getAction()` whenever they have possession of the ball.

1. Start the server by running the following command from the `main` directory.

   ```
   ./bin/HFO --offense-agents=2 --defense-npcs=2 --trials=1000 --fullstate --no-logging --no-sync
   ```

2. In a new terminal from the `example` directory, create the executable by running the following command.

   ```
   ./make_executable2v2.sh
   ```

3. Spawn the agent by running the following command.

   ```
   ./2v2agent --numAgents 2
   ```

To check performance of your agent, run `./autograder-2v2.sh` (in directory `example/`). This script will print out the fraction of episodes that resulted in a goal.

You are likely to find the PASS action useful for increasing the scoring percentage in 2v2 HFO. Although in general, PASS needs to be passed the uniform number of a teammate as argument, the code you are given is set up such that PASS (without any arguments) in 2v2 HFO will automatically pass the ball to the sole teammate of the offense agent with the ball.

Submission

Place 1v1agent.cpp and 2v2agent.cpp (the two files that you have edited) in a directory named your roll number (say 12345678). Tar and Gzip the directory to produce a single compressed file (12345678.tar.gz). Submit this file on Moodle, under Lab Assignment 1.

Evaluation

Your agents will be evaluated by calling the corresponding autograder scripts.

The 1v1 agent will be evaluated for 4 marks. It will be given 4 marks if its goal scoring percentage exceeds 75%, otherwise 3 marks if it exceeds 60%, otherwise 2 marks if it exceeds 50%, otherwise 0 marks. Extra credit of 2 marks will be awarded if the agent's goal scoring percentage exceeds 95%.

The 2v2 agent will be evaluated for 6 marks. It will be given 6 marks if its goal scoring percentage exceeds 60%, otherwise 4 marks if it exceeds 50%, otherwise 2 marks if it exceeds 40%, otherwise 0 marks. Extra credit of 2 marks will be awarded if the agent's goal scoring percentage exceeds 75%.