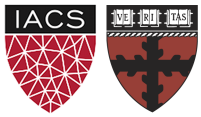


HARVARD
School of Engineering
and Applied Sciences

ComputeFest 2013 Computational Challenge

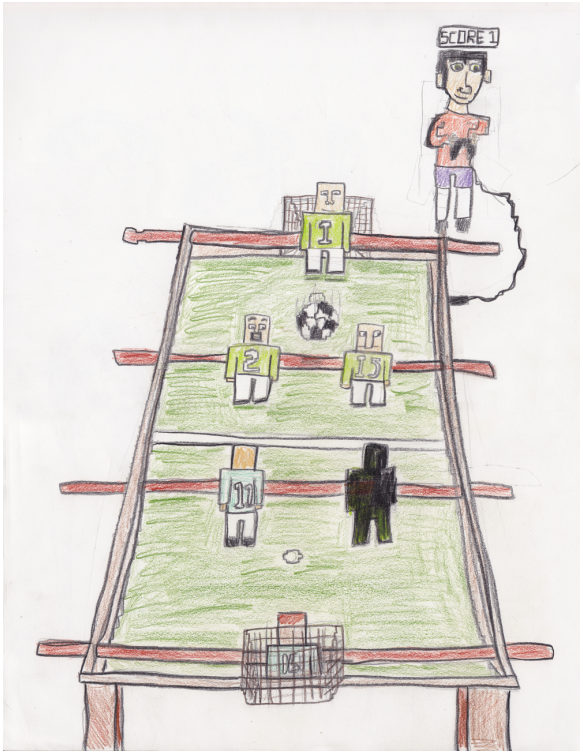
ComputeFest is an annual extravaganza of skill- and knowledge-building activities for students and researchers interested in computational science and engineering, organized by the Institute for Applied Computational Science at SEAS.

In January 2012, IACS organized the first Computational Challenge. Two student teams competed on an optimization problem: how to get medical aid to the maximum number of victims stranded by a hurricane striking the city of Cambridge. The arrival of superstorm Sandy in October 2012 showed that such a problem definition is hardly far-fetched.

Chris Beaumont of the Harvard-Smithsonian Center for Astrophysics and Blessing Okeke of SEAS were the team whose approach saved the most hypothetical victims, winning them a pair of iPads.

This year, in place of a single problem judged at the end of a computing run, we invite students to participate in a game. This year's challenge will be a little more light-hearted, but equally challenging: a computational foosball game. Here are the rules of the game. May the best team win!

2013 World Cup comFoosball Competition



Foosball is a table-top game invented in 1923. Two teams, from opposite sides of the table, operate plastic figures ("foosplayers"), which are attached to rotating and sliding axles, in order to kick the ball into the opponent's goal. After a point is scored, play is resumed from mid-field. Foosplayers are also used to block and pass the ball. In this computational challenge, we define a simple game modeled after soccer and foosball.

Your goal is to provide an AI that can play our version of foosball/soccer by strategically optimizing player positions. Your computational strategy for determining the rosters of your foosplayers (deemed your "foosteam") will be pitted against other foosteams in the foosplay-off tournament.

In the following sections, we define the game that you will be playing in this competition. There are many possible approaches to the problem that speak to the core of computational science: stochastic analysis, optimization, software design, high performance computing, etc. We hope that you use your talents to crush your opposition.

Definitions

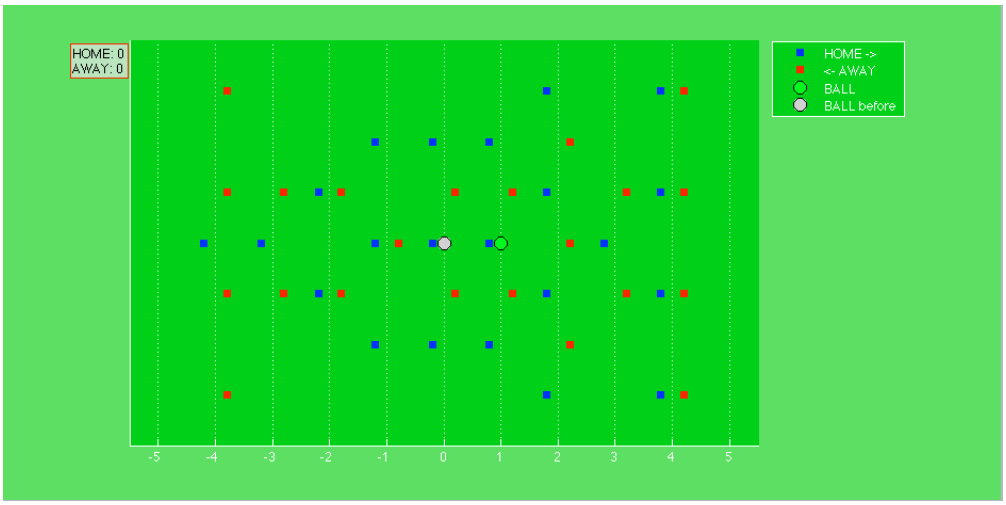


Figure 1: A typical starting line-up. Blue players are the *HOME* team and red players are the *AWAY* team. *HOME* team moves the ball to the right and *AWAY* team moves the ball to the left. The current ball position is shown as a green circle. Also shown is the previous ball position shown as light gray circle.

Field: The field consists of nine rows: -4, -3, -2, -1, 0, 1, 2, 3, and 4. The positive rows are the offensive side, the negative rows are the defensive side, and zero is the *midfield*. Rows -5 and 5 are the "goals" of the board.

Teams: The game is played with two teams, the *HOME* team and the *AWAY* team. Each team is composed of 26 foosplayers. At any given moment, 22 foosplayers from each team must be on the field.

Foosplayer: Each fielded foosplayer is placed in a row. Foosplayers of the *HOME* team always try to kick the ball to the right (towards row 5), while foosplayers of the *AWAY* team always try to kick the ball left (towards row -5). Foosplayer i also has a fatigue level e_i. The fatigue increases as the play goes on (see Section *Gameplay* below for details).

Ball: The ball begins in the mid-field and is advanced once per round in a direction determined by the foosplayers on the ball's current row (see Section *Gameplay*). If the ball reaches row 5, the *HOME* team scores a point. If the ball reaches row -5, the *AWAY* team scores a point. When a point is scored, the ball is reset to midfield.

Gameplay

At the start of each game, each team submits its initial roster to distribute the foosplayers to the rows. Each foosplayer i has initial fatigue $e_i = 0$.

Play begins by placing the *ball* at mid-field (row zero). Then,

1. Let the ball be in row B. All foosplayers on row B compete to determine the direction the ball is moved: Let $\mathcal{H}$ and $\mathcal{A}$ be the set of foosplayers from the *HOME* and *AWAY* teams on row B and let $\mathcal{F} = \mathcal{H} \cup \mathcal{A}$. Let $r \in [0, 1)$ be a uniform random number. Then,

$$\text{If } r \begin{Bmatrix} < \\ > \\ = \end{Bmatrix} \frac{\sum_{i \in \mathcal{H}} 0.99^{e_i}}{\sum_{i \in \mathcal{F}} 0.99^{e_i}} \text{ then the ball is } \begin{cases} \text{moved to row } B+1 \\ \text{moved to row } B-1 \\ \text{kept at row } B \end{cases}$$

   That is, the fatigue e_i of the foosplayers on row B determine the probability that the *HOME* team (and the *AWAY* team) advances the ball.

   In the case that row B contains no foosplayers ($\mathcal{F} = \emptyset$) then the ball is advanced to row $B + 1$ or $B - 1$ with probability 0.5.

2. If the new ball position is a goal row, the appropriate team earns a goal and the ball is reset to midfield.

3. Once the new ball position is determined, every foosplayer in row B that competed to advance the ball is hit with fatigue:

$$\text{For all } i \in \mathcal{F}, \quad e_i \leftarrow e_i + 1$$

4. After the ball is advanced and the players fatigued, each team may move any one foosplayer from its current row to an adjacent row.

5. Repeat for 200 rounds.

6. Every 200 rounds, a new quarter begins. To begin a new quarter, each team may submit am entirely new roster. That is, foosplayers may be placed in any row. Note that all fooplayers that were "benched" in the previous quarter have no fatigue ($e_i = 0$).

7. Repeat for 4 quarters.

8. The team with the most goals wins the game.

Roster

A roster is an array that maps foosplayer number to foosplayer row position. Each foosteam has 26 foosplayers with exactly 22 foosplayers on the field during any round. The ith value is the ith foosplayers' row position. For example:

```
-4 -4 3 3 100 4 -3 -3 -2 -2 -2 -1 -1 -1 0 2 2 0 1 1 1 1 100 100 4 100
```

Row positions that are outside the range $[-4, 4]$ are considered "benched" and not in play. Negative row numbers are always defensive positions and positive row numbers are always offensive positions.

Foosteams submit the entire roster each round. If an invalid roster is submitted, the game will print a warning, reject the play, and accept the null move – no change to foosplayers' positions will be made.

Most rounds, the roster can only change by a single foosplayer's row position to an adjacent row. However, when the round number is 0, 200, 400, and 600, a new quarter is about to begin and any valid roster will be accepted.

Game State

When both foosteams have submitted their rosters (valid or invalid) the game will continue to the next round by updating the foosplayer positions, the foosplayer fatigues, the ball position, the foosteam scores, and the round number. These are communicated back to the foosteam as a single array (**Note:** 0**-indexed**):

```
game_state[0]: team score
game_state[1]: opponent team score
game_state[2]: game round number
game_state[3]: row number of the ball
game_state[ 4]-[ 29]: team foosplayer row positions
game_state[30]-[ 55]: team foosplayer fatigues
game_state[56]-[ 81]: opponent foosplayer row positions
game_state[81]-[107]: opponent foosplayer fatigues
```

This game state should then be used to choose a roster for the next round.

Game Engine

We will be using a centralized game engine that keeps track of the game state and allows any programming language to communicate with it. This makes it easy to pit foosteams of different types and languages against each other.

You may download a selection of skeleton code that has been written to get you started communicating with this game engine:

Download codes here

or

`wget http://crisco.seas.harvard.edu/ComputeFest2013/Codes.zip`

These codes provide communication clients and skeleton foosteams in Python, Matlab, C++, and Java.

To play two foosteams against each other, launch both foosteams with the same commandline game ID. For example:

```
$ python foosteam.py 1234
Waiting for game 1234
```

launches a foosteam into a game with ID 1234. Another foosteam can join this game by launching with the same game ID:

```
# After compiling with 'javac *.java'
$ java foosteam 1234
Waiting for game 1234
READY
```

The second foosteam is in Java here, but could also be the same Python foosteam from above.

Once two foosteams have joined a game, `READY` is broadcast to both foosteams and the game loop is entered. Both teams submit their first roster, receive the game state, decide on their next roster, etc, until the game is over.

You will be editing the `foosteam` file in the language of your choice: specifically, the `new_move` function, which takes a game state and returns a roster for that game round.

Competition

Time/Location

The final competition will be held on Wednesday at 1:30pm at MD119.

Roud-robin phase

There are 10 teams.

Each team will face all other teams in 45 *matches* (see Table 1). A match is composed of 5 games.

The team that wins the most games in a match is awarded 2 points. If the two teams win the same number of games, 1 point is awarded to each team. The losing team is awarded zero points.

Table 1: The round-robin competition

Round 1	1-10	2-9	3-8	4-7	5-6
Round 2	1-9	2-7	3-6	4-5	8-10
Round 3	1-8	2-5	3-4	6-10	7-9
Round 4	1-7	2-3	4-10	5-9	6-8
Round 5	1-6	2-10	3-9	4-8	5-7
Round 6	1-5	2-8	3-7	4-6	9-10
Round 7	1-4	2-6	3-5	7-10	8-9
Round 8	1-3	2-4	5-10	6-9	7-8
Round 9	1-2	3-10	4-9	5-8	6-7

At the end of the round-robin tournament the two teams with the most total points move on to the **final**. In the case of a tie (teams with the same number of points), the teams will play each other in tie-breaker matches, 5 games per match.

The final

The final match is composed of 9 games and will happen at the end of the round-robin phase. In case of a tie, there will be an additional 3 game match. If there is no winner at the end of that match, there will be a sudden death session (the team that wins the first game is the winner).

Time duration

Each team is given 1 minute to make their moves and substitutions. This is the total accumulating time during the game and is calculated at server side. A team that exceeds the limit automatically loses the game. If both teams exceed the limit, the game is considered as a tie.

Contact Information

Game Engine:
Cris Cecka, Cruft 402, ccecka@seas.harvard.edu.

Rules of the game:
Pavlos Protopapas, Cruft 402, pprotopapas@cfa.harvard.edu

SEAS computational facilities:
Kumar Indireshkumar, kindires@seas.harvard.edu
Robert Parrott (for Monday 1/21), parrott@seas.harvard.edu

Competition:
Cris Cecka, Cruft 402, ccecka@seas.harvard.edu.
Pavlos Protopapas, 402 pprotopapas@cfa.harvard.edu

Disputes:
Mauricio Santillana, msantill@fas.harvard.edu
Daniel Weinstock, dweinsto@seas.harvard.edu

All other:
Rosalind Reid, rreid@seas.harvard.edu

Fair-play

No external collaborations are allowed. This includes consulting forums, friends, family or other students. Abuse of the game server is not allowed. This includes excessive connections or purposely invalid moves.

Acknowledgement

This game was inspired by the AM207 course project of Chase Hu and Ashin Shah.