CS330 Project Assignments

as of Wed. Oct. 4, 2006

1.1 About EF Games

EF Games have been introduced in the lecture of Sept.20. There are various books introducing the game rules; unfortunately, these are all rather deep and difficult. If you know what graphs, isomorphisms, and induced isomorphisms are, it may be worth looking at p.16 of the slides at

http://www.cse.ucsc.edu/~kolaitis/talks/essllif.ps

The material on these slides is *NOT* part of the course and you do not need to study the slides for the exam. However, if you are curious what EF games are used for, the slides are a good source of information. In short, EF games characterize the expressive power of query languages such as relational algebra and relational calculus. They can be used to prove mathematically that certain questions about the data in a database cannot be formulated as queries in relational algebra.

1.2 The EF Human-vs-Computer Game Application

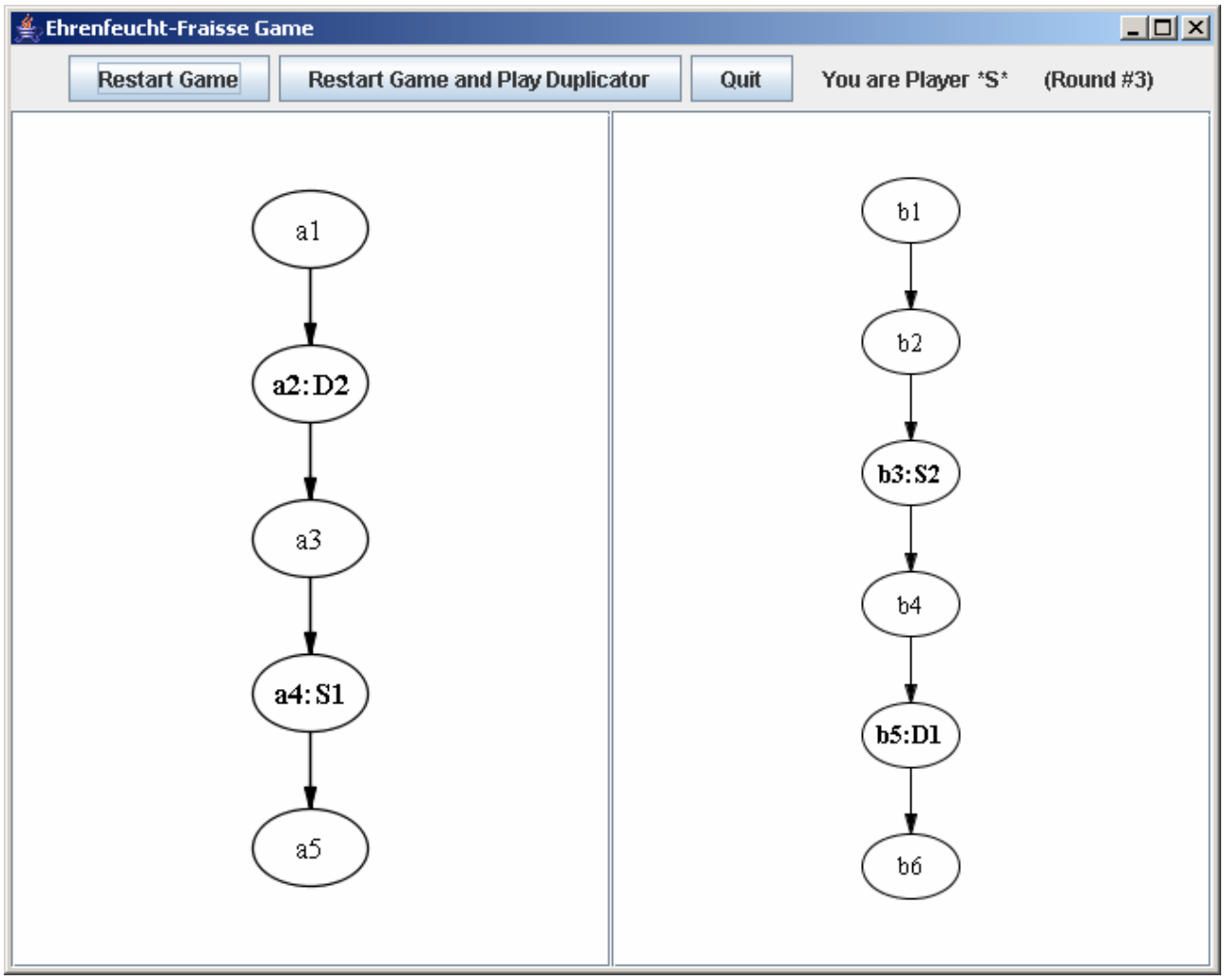
I recommend trying out the EF Game Application that is available on the course homepage:

http://www.cs.cornell.edu/courses/cs330/2006fa/efgame.zip

This archive contains a Java application (the .jar file, which you can start by double-clicking in case a Java VM is installed on your computer) and a number of graph files. (These were created using a program called GraphViz that you can download from the Web. If you like to.)

The application allows you to play against the computer and after very few games played I am quite sure you will understand the rules. When starting up, the application will ask you for two graphs to play on and for the maximal number of

rounds to play. Since the computer's strategy computation was a very quick hack and is very inefficient, the strategy computation, which is performed just after you have made your choices, may take a few seconds and may even crash if the main memory granted to the Java VM is exceeded. This is also the reason why I have limited the maximum number of rounds to four. For learning the game this should not matter, except that each game will be rather short. On startup, you will play Spoiler and the computer will play Duplicator. There is a button for switching the roles.



Screenshot from the EF Game Application.

1.3 Goal of the Project: a Web-based Game Server for EF Games.

Note: the following is intentionally a mixture of requirements affecting the database structure and a description of the features of the Game Server to be implemented. The underlined portions have been added in this version of the specification.

The finished EF Game Server allows for the playing of EF games by a community of players. Each player can pose challenges to play EF games and can accept challenges from other players.

Each *player* is identified by her netid. Authentication can be achieved using a password. No name or further personal information is stored. A player's record is created on first login and further comprises a flag indicating whether the player is currently logged in and the date and time of the last login (null on the first login). The game server provides a logout option on each page. There is no automatic logout after a timeout. If a player leaves the game server (e.g. closes the browser) without logging out, the system simply updates the player's "last login date" on the next login. Until then, the game server will wrongly believe that the player is around.

On successful login, the game server displays the player's customized "main page", to be described later.

The game server stores *graphs* to be played on. These are uniquely identified by their name. Furthermore, a short description of the graph, a .jpg image file of the graph, and a .dot file containing the specification of the graph can be stored.

When posing a *challenge*, the player can choose the parameters of the game, i.e. the two graphs on which to play, the maximum number of rounds after which the game ends, and the role to play (Spoiler or Duplicator). A challenge can be accompanied by a short text message. The challenger may or may not choose an opponent from the players known to the system. If she does, the challenge is called direct; otherwise it is open and may be accepted by any player except the challenger. However, an open challenge can only be accepted by one player, not several, on a first-come-first-serve basis. Each challenge can be accepted, refused, reversed or ignored. By accepting, the game starts and the challenge is not shown anymore. After refusing, the challenge is not shown on the refusing player's main page anymore. However, only a directed challenge can be refused. Reversing a challenge means to refuse a challenge but to pose the same challenge to the challenger in return. (Thus if Player A challenges Player B as duplicator and if Player B reverses the challenge, this means that Player B challenges Player A as duplicator, using the graphs and the maximal number of rounds from the original challenge. This could be implemented by simply switching the netids in the challenger and opponent fields of the tuple representing the challenge to be reversed.) Ignoring a challenge of course means to do nothing about it. A challenger can withdraw her challenge if it has not been accepted yet.

Each *game* follows the acceptance of exactly one challenge. Each challenge is followed by at most one game. The player who accepts another player's challenge is called her opponent. Accepting a challenge means to play according to exactly the parameters fixed in the challenge, i.e. two graphs, a maximum number of rounds, and to take the role (Spoiler or Duplicator) not chosen by the challenger. For each game we also store who has won (Spoiler, Duplicator, or no one yet), and each of the moves played. (Note: the information who has won a game is not redundant with the information about moves stored in the database since players can give up games; in this case they lose those games.)

For each *move of a game*, we store which move within the game it was (out of the sequence, S1, D1, S2, D2, S3, D3, S4, …, i.e., who – Spoiler or Duplicator -- played and out of which round the move was), whether the left ("A") or right graph ("B") was chosen, and which node within that graph was chosen. Each node within a graph is assumed to be identified by a positive integer (1, 2, 3, 4, …). We also store a timestamp indicating when the move was played.

While playing a game, the game screen is shown. This page displays the graphs, the previous moves, the choices of moves the player has (in case it is her turn to play), and additional information such as the netids of the opponents, their roles, and the maximum number of rounds to be played. A player can give up a game (losing it) or return to the main page to resume the game later (without giving up). A player can invite her previous opponent to play a finished game again (i.e. create a challenge from a finished game). One can chose to look at a game that is over – in this case the graphs and the history of moves of the game is displayed.

2.1 Assignment 1 (due Wed. Oct.4, 2006 11:59pm):

- Make sure you understand the rules of the game and the specification of the game server as provided so far. Use the EF game application provided on the course page to practice playing against the computer.
- Draw an entity-relationship (ER) diagram of the game server's database. This diagram is for your own use only and is not to be submitted.
- Translate this diagram into SQL create table-statements to build this database. Provide these statements in a text file. Note: test-build the tables using these statements/this file. The statements must work on the J2EE/Pointbase setup in the CSUGLab.

Of course each table needs a key. Decide whether a subset of the attributes required for each table above is appropriate or whether a special-purpose key-field (with generated keys) should be used. You do not have to avoid null values where they are helpful for representing data, but please try to avoid a bad schema design.

- Add SQL insert-statements to create a database with 10 challenges and 5 games. You may create graph records with null-values for the image fields.
- Add the following SQL queries:

  (1) For each player currently logged in, return the player's netid, the total number of games she has won, and the total number of games she has lost.

  (2) Select all challenges relevant to the current player, i.e. those that have not been accepted yet and which either were posed by or are directed at the current player, or are open.

Observe:

- Only the file with the SQL statements – the create table, insert, and select statements -- is to be handed in. This must be a plain text file and these commands have to work in the CSUGLab (i.e., on PointBase).
- The assignment has to be handed in via CMS by Wed. Oct 4. 11:59pm.
- This is hard deadline and **no** exceptions can be made.

2.2 Assignment 2 (due Sunday Oct. 22, 2006 11:59pm):

Develop a J2EE Web application using JSP and/or servlets that implements the display and summary page (the "main page") of the game server. This page must, *after logging in* (you must implement a way of logging in as well), display the following information:

- As the title of the page, display "<your netid>'s EF Game Server". That is, here we display the netid of the developer of the Game Server rather than its current user.

Moreover, in the body of the page,

- The netid of the current player (i.e. the player to whom the page is shown).

- A timestamp showing when the page was created. (This will remind the player of the age of the information displayed on the page and will cause her to refresh the page from time to time.)
- An overview of the players currently logged in according to the database: For each player show
    - the netid,
    - the number of games won and lost overall and,
    - for all players other than the current player, the number of games won and lost against the current player.

  (Hint: Adapt and extend query (1) from Assignment 1.)
- A tabular overview of the relevant (i.e. not yet accepted) challenges either
    - posed by the current player,
    - directed at the current player, or
    - open.

  (Hint: Adapt and extend query (2) from Assignment 1.)
- A tabular overview of the all current and past games (not just those involving the current player – in assignment 3 you will extend your application for control elements to switch between these alternatives).

- Please look at the general specification of the game server in Section 1.3. Your implementation has to be consistent with that specification and the previous bullet points. Beyond that, you can make your own choices. You may, but do not have to, decide to make your main page look like in the screen shot at the end of this document.
- If necessary to test your Web application, insert further tuples into your database.

For an example of what such a page could look like see the picture "main page of the game server" at the end of this document. Note that this picture already shows various links/buttons for functionality that will be part of later assignments.

Since we have not covered enough security background yet, we will ignore security concerns for now. In particular, logging in (for Assignment 2) means to provide a netid but no password. The schema contains a *Player.password* field for later, but we currently leave it null and neither ask for nor check passwords.

IMPORTANT: Do not implement Assignment 2 using the database schema that you have created for Assignment 1. Instead, use the schema that you are provided with

http://www.cs.cornell.edu/courses/cs330/2006fa/assignments/efgs_schema.txt

(We will all use the same database schema because this will later allow us to run all Game Servers against the same database – with better opportunities for testing your code and playing with your colleagues.)

Observe:

- You can download efgs_schema.txt from the course Web page. Please have a look at it and check how it differs from your solution. If my solution differs from yours it is not necessarily better than yours; in fact, the schema of efgs_schema.txt is not optimal in terms of design. However, this schema will allow us to practice a few things that I think are important later on and will make other aspects of the game server rather easy to implement.
- Note that efgs_schema.txt contains a number of insert statements. While using the schema (i.e., the create table statements) is mandatory, you may remove the insert statements and insert different tuples.
- Decide for yourself how you will later (for Assignment 3) implement the different actions that can be performed on challenges, for instance refusals. Figure out in which cases tuples should be deleted from the database.
- Assignment 2 is to be built using J2EE. You have to provide
  - a zipped version of all of your source code (preferably a zipped version of an Eclipse project directory) plus
  - a J2EE Web Application .war file that can be deployed on the CSUGLab installation of the J2EE Application Server.

  In case you do your development at home, please make sure that you have tested your implementation in the CSUGLab. In particular, you have to change your database URI and Driver class to work there (with PointBase). Please use the database URI

  ‘jdbc:pointbase:<your netid>_efgs_db’

  (for me, this is ‘jdbc:pointbase:cek37_efgs_db’) – this should locate your database in the default location C:\pointbase\databases.
- The assignment has to be handed in via CMS by Sun. Oct 22. 11:59pm.
- This is hard deadline and **no** exceptions can be made.

Notes:

- Your java code can use the lines

```
String url = "jdbc:pointbase:<your netid>_efgs_db;create=true";
Class.forName("com.pointbase.jdbc.jdbcUniversalDriver");
con = DriverManager.getConnection(url, "PBPUBLIC", "PBPUBLIC");
```

to connect to the database.

- If you want to use the pointbase console (C:\Sun\AppServer\pointbase\tools\serveroption\startconsole.bat) then you first have to set the database path using the console's menu item DBA/Database Properties/database/home to C:\pointbase\databases (default is C:Sun\AppServer\pointbase\databases).

- Development at home: Apache Derby uses the default location C:\Sun\AppServer\domains\domain1\config for database files. So if you use 'jdbc:derby: <your netid>_efgs_db' as URI, the database will be looked for in that directory. But please don't forget to modify your application to run on Pointbase in the end, and test it in the CSUGLab!

2.3 Future assignments.

What follows is a short description of the future assignments. More detailed specifications of these assignments will be posted later.

Assignment 3:

Add support for creating assignments and playing games (see the mockup screens at the end of this document). The game playing page must display at least the following information:

- The two graphs and the maximal number of rounds to be played.
- The netids of the opponents and their roles.
- The history of the game (previous moves).
- The information whether it is the current user's turn or her opponent's.
- In the case that it is the opponent's turn, a button or link for refreshing the page.
- Otherwise, control elements that allow the user to specify the next move and a button to commit the next move. Only valid options for the next move should

be displayed. (This is important if the current player plays Duplicator! – You should know why.)
- A button or link for returning to the main page of the game server. This does not end the game; instead, the user can return to playing the game from the main page.

Please make sure that you have studied and understood the screenshots of the main page and the game page below. While you do not need to make your pages look exactly alike, you should support at least the same functionality as suggested by the screenshots. Also look again at the specification of Assignment 1 to understand the general requirements on the game server.

Before starting to work on the code required for this assignment, it is recommended that you make clarify to yourself which forms you need, which information you will pass to the server, and whether you will pass this information using GET or POST requests. Deciding this beforehand will make your development much easier!

IMPORTANT: This specification is still unfinished.

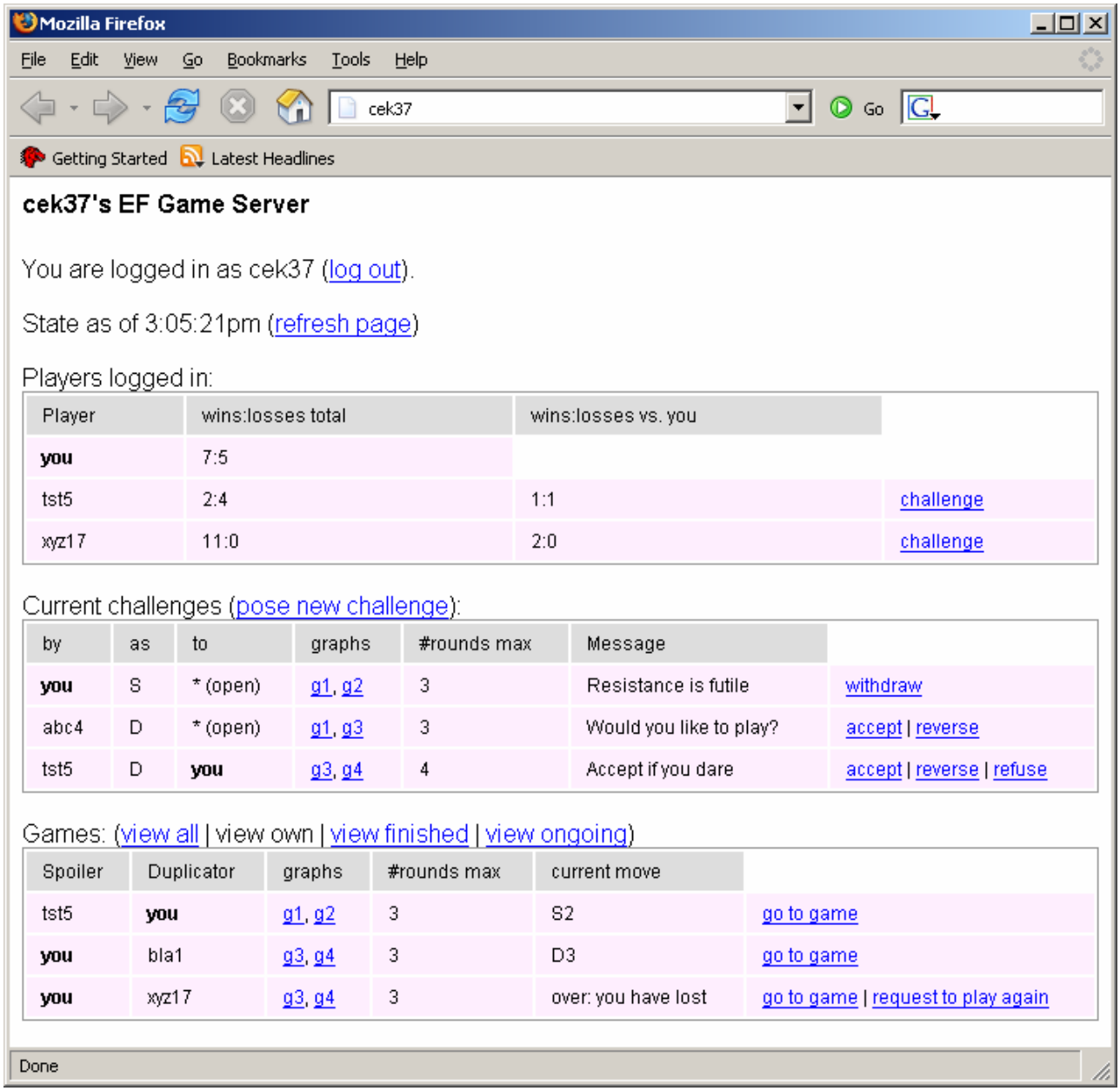
Observe:
- You will be able to download code for checking the winning condition for an EF game (i.e. whether the moves of the game form a partial isomorphism on the game graphs) from the course Web page.
- We will use a class library called Grappa for managing graphs. Example code for reading graphs from .dot files and checking partial isomorphisms on graphs will be provided.

Assignment 4:

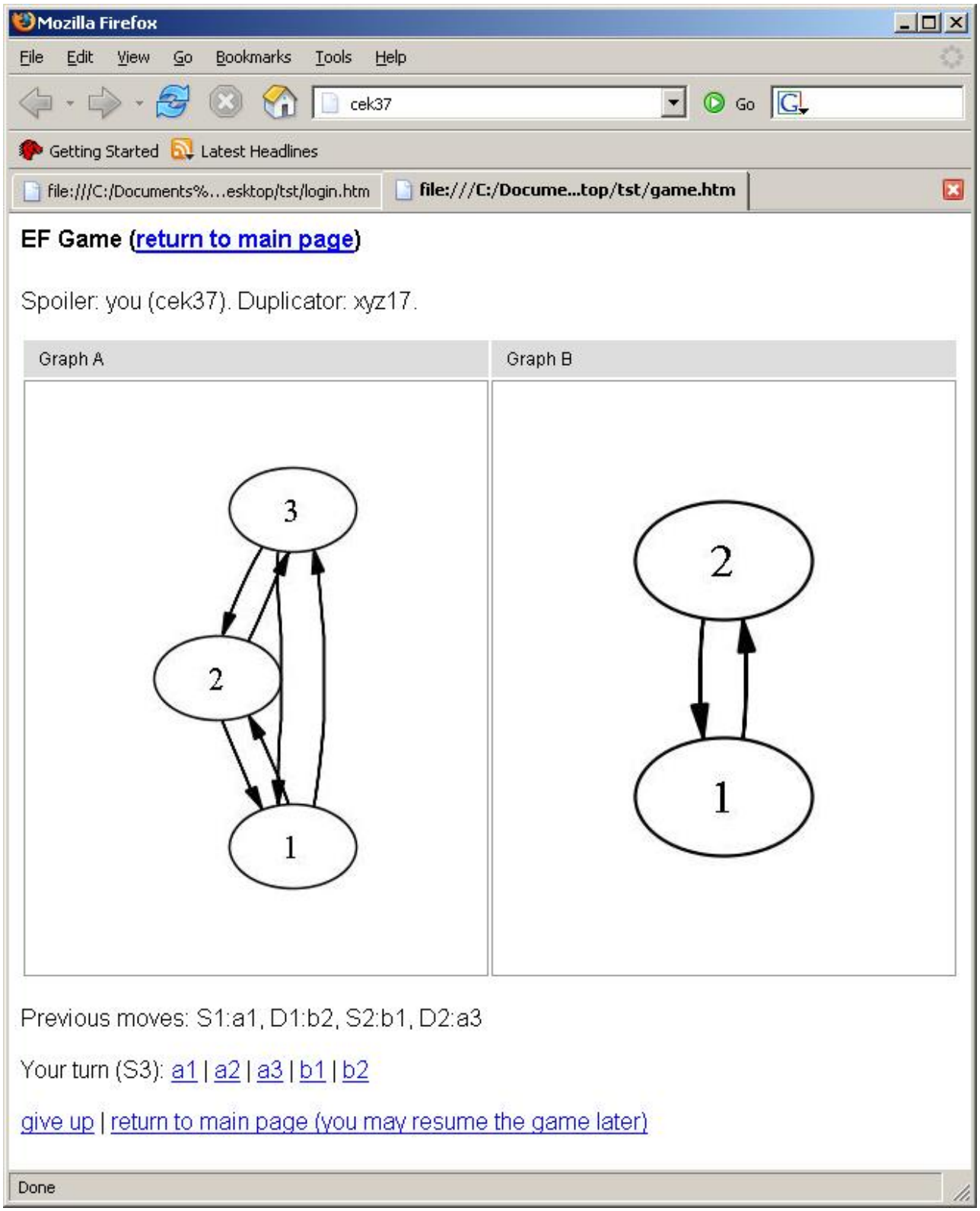
Implement a computer player that human players can challenge via the game server using a Web service that suggests moves. (The Web service will be provided.)

3.1 Screen shots.

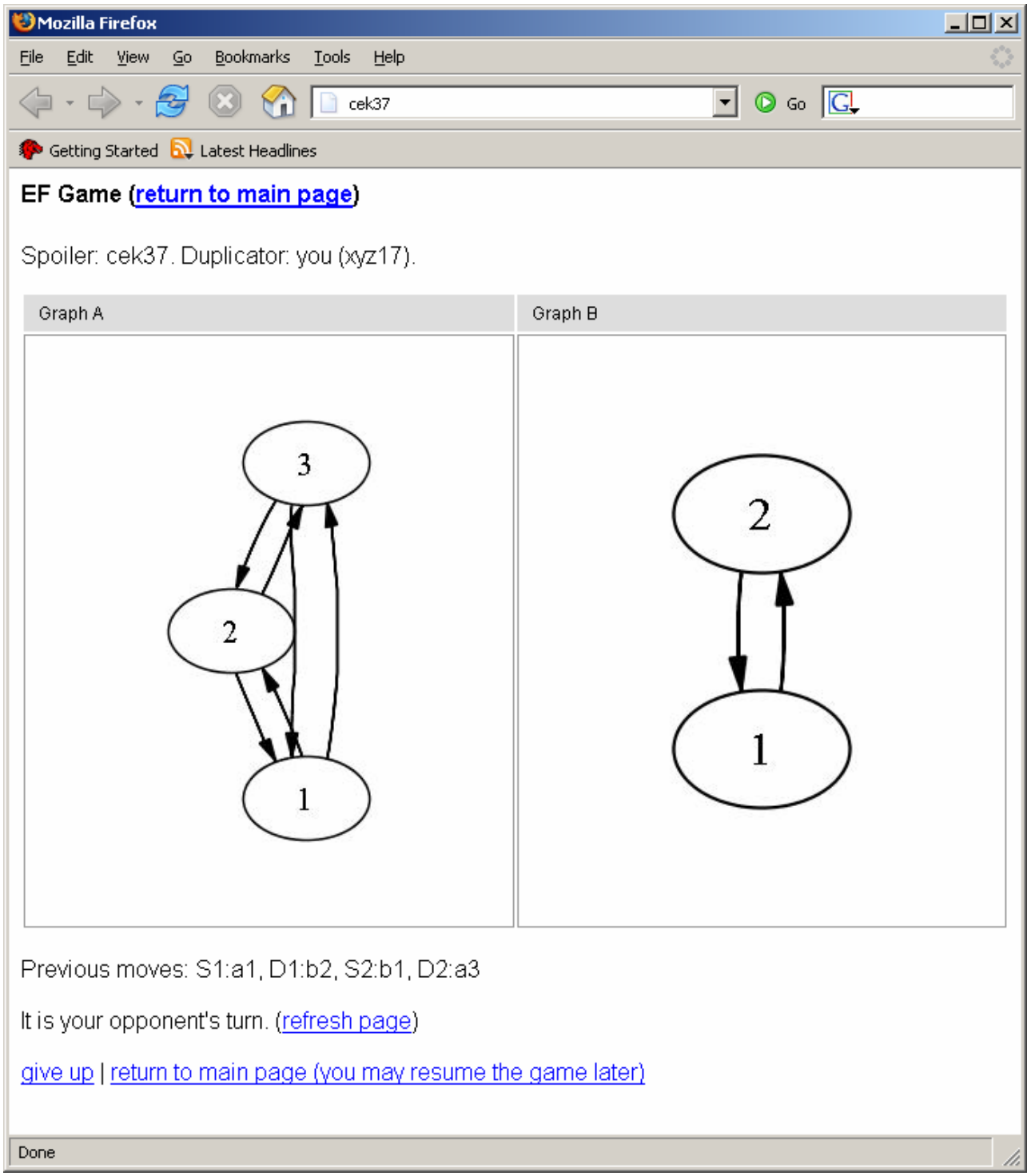
Screen shot: Main page of the game server.



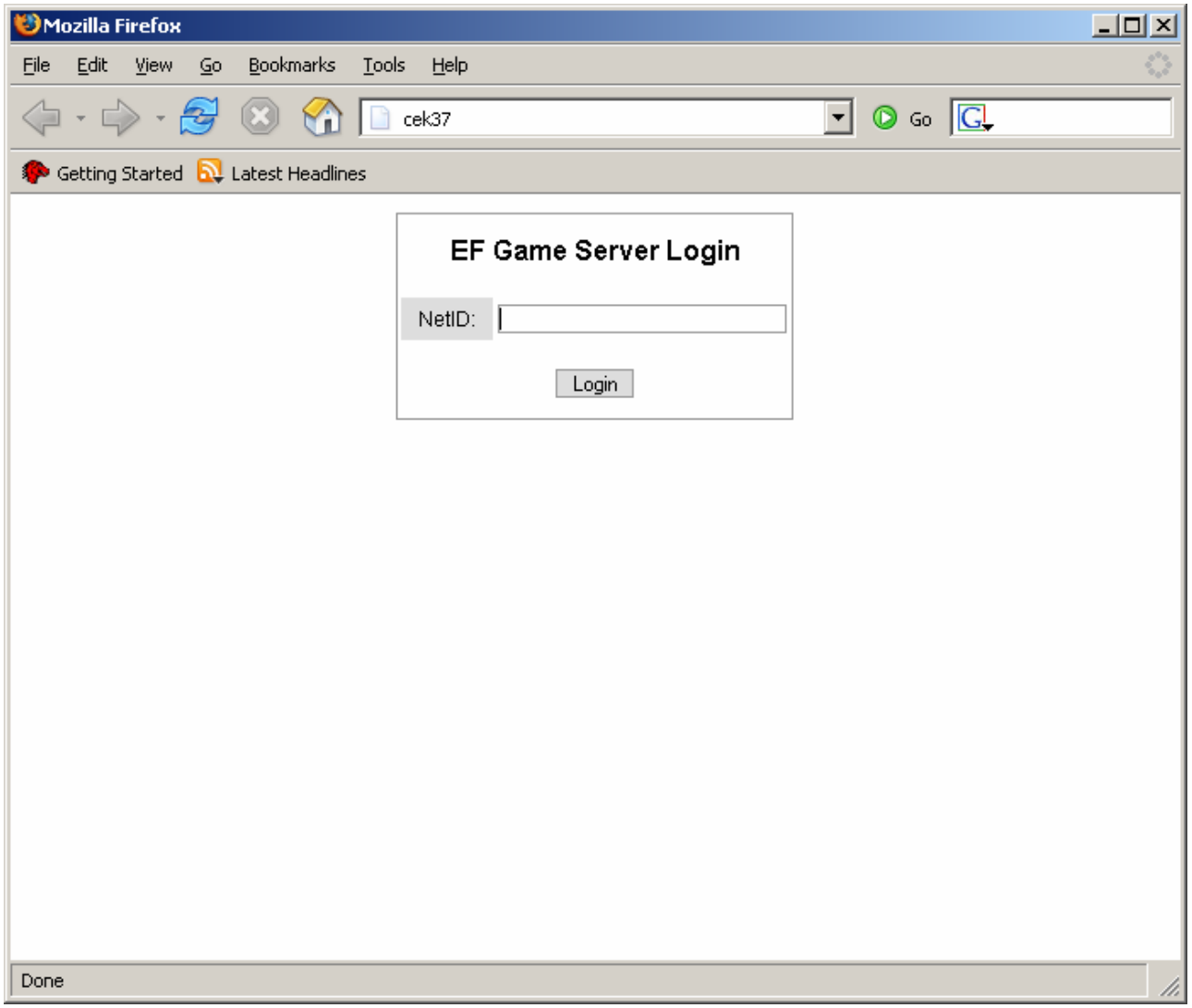
Screen shot: Page for EF game playing (current player's turn).



Screen shot: Page for EF game playing (opponent's turn).



Screen shot: Login screen for Assignment 2 (no passwords).



Screen shot: Creating a new challenge.