

CS330 Project Assignments

as of Mon. Nov. 20, 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

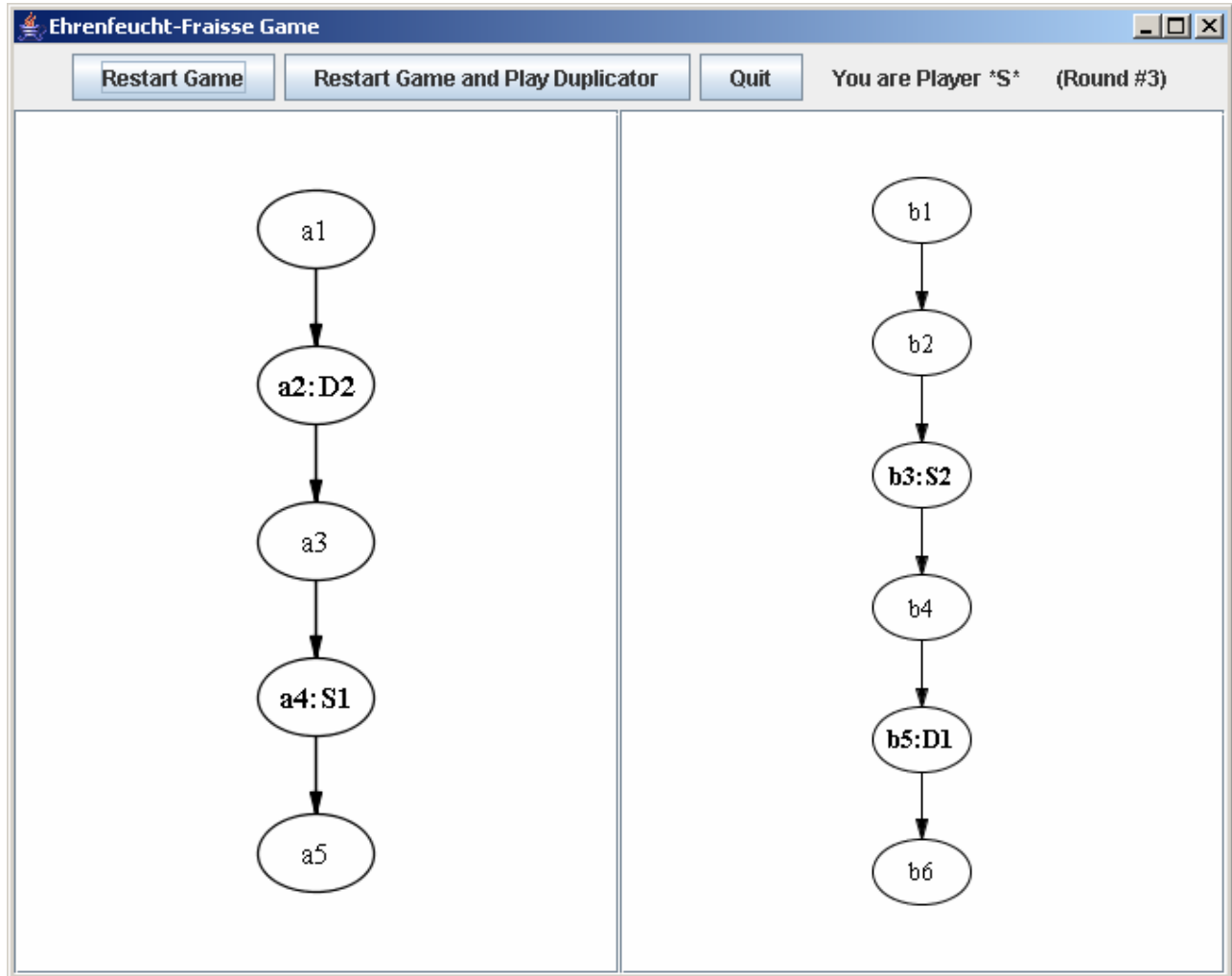
I recommend trying out the EF Game Application that is available on the course homepage:

<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 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 a role (Spoiler or Duplicator) – the one 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). You will have to close and reopen the console for this to work.
- 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!

Some factoids to make sure we all use the schema in the same way:

- Open challenge can be recognized by the NULL value in the opponent_netid field. When such a challenge is accepted, the game tuple will hold the opponent's netid; the opponent_netid of the challenge tuple will remain NULL.
 - If a player gives up a game, s/he loses; for that reason, there can be games that have been won without a single move played.
 - A move fixes a choice of graph and of node in that graph. 'A' corresponds to the graph identified by Challenge.graph1_name and 'B' to Challenge.graph2_name.
-

2.3 Assignment 3 (due Sunday Nov. 12, 2006 11:59pm):

Extend your Web application of Assignment 2 as follows.

Add support for *sessions*, logging in and logging out.

Extend your Web application to allow for the creation of assignments and the playing of games (see the mockup screens at the end of this document). The game playing page must display at least the following information:

- Pictures of the two graphs, labeled “A” and “B”.
- 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. (The player viewing this page is waiting for her opponent to make a move!)
- 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 later.
- A button/link for giving up the game (and thus losing).

Add support for importing graphs. In Assignment 2 we did not have to worry about graphs very much, but now we need two things apart from the name of a graph: its structure (to check the winning condition of the EF game, more about this below) and a nice picture that we can display on the game page. **IMPORTANT NOTE:** You may import the graphs in any way you like; This does not have to be done through a servlet (you need third-party java classes to support file upload in servlets) and cannot be done using the console application. I suggest to do the upload using a classical Java application.

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 like mine, 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.

It is required from you that your Game Server works correctly even if several users are playing concurrently using your Web application. Make sure that you use JDBC transaction support and thread-safe code where needed. Make sure you completely understand which kinds of conflicts may arise where in the Game Server. Please take particular care of RW conflicts. (Making this problem reasonably easy is the reason for certain choices I made in `efgs_schema`.) I will also discuss these issues in class, please attend!

Before starting to work on the code required for this assignment, it is recommended that you first try to understand 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! Also, you may have realized that testing Web applications is more involved than testing classical Java applications. I recommend to test your queries and updates (a main novelty of Assignment 3) in the database console (the Pointbase console or Derby's `ij` class) first; JDBC code that does not essentially require to run in a servlet could also be tested in a normal Java application first. I can also recommend the Eclipse plugins for J2EE development, they are really helpful; if you are working in the CSUGLab, you need to first install your own copy of Eclipse on the Z drive, however, to be able to add plugins.

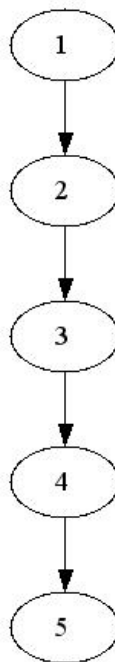
This is the first assignment in which you need to understand EF Games. If you don't, please play the game in the standalone application that I have provided or talk to me.

Observe:

- The file `MyGraphUtil.java` contains code for reading in a graph from a `.dot` file (stored as a BLOB in the graph table) checking the winning condition of an EF game (i.e. whether the moves of the game form a partial isomorphism on the game graphs) can be downloaded from the course Web page. All you need to do to support game playing is to
 - provide the user interfaces for users to see the game state and to play moves, and
 - to check after each round whether the game is over already.
- We will use a class library called Grappa
<http://www.research.att.com/~john/Grappa/>
for managing graphs. You may either copy the `grappa1_2.jar` file to a place where your application server can find it or include it into the `.war` file. (One way to do this is to unzip `grappa1_2.jar` and add it to the classes directory.) If you download and unpack Grappa, you can find that jar file in the "archives" subfolder.

- The Grappa homepage also provides documentation, but basically you should not need to read this; all you need to know is that graphs can be read in using `MyGraphUtil.readGraph()` and that partial isomorphisms can be checked with `MyGraphUtil.isomorphism()`. `MyGraphUtil` is packaged into the project directory `EFminimal` together with a main class to illustrate its use.
- The file `dot_graphs.zip` contains dot-files and corresponding images to import into the database.
- You are free to use either J2EE 1.4 with Pointbase or J2EE 5 with Derby for this assignment. You may create additional views if you need them, but note that for Assignment 4 we will all use strictly the same schema, actually even the same database.

Aside note: “.dot” files are the data format of the graph drawing tool Graphviz (see Google) and its Java API Grappa for storing graphs (this is not about Microsoft Word document template files). You do not need to know it, but there is also no real difficulty related to it. For example, the graph



is represented as a .dot file as

```
digraph "g1" {
    1 [label="1"];
    2 [label="2"];
```

```
3 [label="3"];
4 [label="4"];
5 [label="5"];
1->2; 2->3; 3->4; 4->5;
}
```

You can generate images from .dot files using Graphviz, or you can just simply use the .dot files and images I have provided in dot_graphs.zip. Actually, you can also create images out of Grappa graph objects, but I do not require this from you as it involves AWT/Swing programming. You don't have to do it.

2.4 Assignment 4 (due Wed. Dec. 6, 2006 11:59 pm).

In Assignment 4 we will move from local deployment in the CSUGlab to a common server (dbdevel-dmz.cs.cornell.edu). This server runs Microsoft SQL Server as the DBMS. My suggestion is to do as much of your testing locally before you deploy on dbdevel-dmz. Remember that the database on the server will be used concurrently by many users. If you insert incorrect data or do not manage your transactions well, you may cause substantial frustration to the other students in CS330 and to yourself.

The assignment comprises the following tasks:

1. Implement the remaining functionality that is described in Section 1.3. This in particular includes
 - Reversing challenges
 - Giving up games
 - Refusing direct challenges
 - Withdrawing challenges before they have been accepted.
2. Implement support for showing, in the Games table on the main page, either all games, the current user's own games, the finished games, or the ongoing games. (See the screen shot of the main page below for one possible way of implementing the user interface.)
3. When creating a new challenge, the current user must be able to select the two graphs and the opponent from among the choices available in the database. (See the screenshot below.)
4. On the main page, whenever the name of a graph is displayed, it should be hyperlinked to its image in the database: By clicking on the graph name, the image is displayed.
5. In Assignment 4, we will all use a common database instance. Have a look at the current version of the schema. Since we are now using Microsoft SQL

Server, some minor changes had to be made. In particular, see how timestamps, blobs, and the key of the table Challenges is now implemented. Two views are provided that you can use. Please do not attempt any changes to the schema. In particular, you cannot add your own views.

6. Make sure that your game server handles accesses to the database that fail because of concurrent transactions appropriately. For example, if a user tries to accept a challenge but another user was quicker at that, your game server should not crash but give a descriptive error message indicating that someone else has accepted the challenge, and should provide a link back to the main page of the game server.
7. Make sure that your code cannot cause the database to become inconsistent if updates may occur concurrently. For this, you need to decide for yourself which kinds of concurrent update transactions could conflict. Please test your code locally before you deploy it on dbdevel-dmz.
8. Make sure that your Game server does not allow for SQL injection. You may have to revisit the code you developed for the previous assignments.
9. In case we discover some problems with your implementation during the first demo session, fix these.
10. Play games against the other students.

That's all: of course you are invited to add further improvements but do not have to.

You have received details on connecting to dbdevel-dmz by email. This includes an administrator password for the application server with which you can deploy your war files. Abuse of these privileges is strictly prohibited. You may deploy and undeploy your own war files, but please make sure that you do not modify anything else. If you make a mistake or have problems with the application server on dbdevel-dmz, please contact Fan.

Note that since you will deploy your war file remotely, you cannot just copy it into an autodeploy directory. You can either use the Eclipse plugins for deploying or use the Web interface at

<http://dbdevel-dmz.cs.cornell.edu:4848/>

that I have demonstrated in class. (Note that the J2EE installations in the CSUGlab also provide such a administration interface at port 4848.)

The Web applications on dbdevel-dmz are again served at port 8080.

IMPORTANT: your Web application cannot be just called “a4.war” or so, it must contain your netid in the file name, otherwise there will be naming conflicts! Also, when testing and playing, you should only use user names that have your netid as prefix, otherwise you might confuse others who are testing their systems. (But you will need more than one user name for testing, so I suggest user names such as XYZa and XYZb if your netid is XYZ).

Note that I have installed a servlet for displaying the content of the database at

<http://dbdevel-dmz.cs.cornell.edu:8080/tables/servlet/>

The source code for this servlet is provided on the course homepage as tableservlet.java. Please have a look at the updated schema definitions in efgs_schema_a4.txt.

3.1 Screen shots.

Screen shot: Main page of the game server.

cek37's EF Game Server

You are logged in as cek37 ([log out](#)).

State as of 3:05:21pm ([refresh page](#))

Players logged in:

Player	wins:losses total	wins:losses vs. you	
you	7:5		
tst5	2:4	1:1	challenge
xyz17	11:0	2:0	challenge

Current challenges ([pose new challenge](#)):

by	as	to	graphs	#rounds max	Message	
you	S	*(open)	g1, g2	3	Resistance is futile	withdraw
abc4	D	*(open)	g1, g3	3	Would you like to play?	accept reverse
tst5	D	you	g3, g4	4	Accept if you dare	accept reverse refuse

Games: ([view all](#) | [view own](#) | [view finished](#) | [view ongoing](#))

Spoiler	Duplicator	graphs	#rounds max	current move	
tst5	you	g1, g2	3	S2	go to game
you	bla1	g3, g4	3	D3	go to game
you	xyz17	g3, g4	3	over: you have lost	go to game request to play again

Done

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

Mozilla Firefox

File Edit View Go Bookmarks Tools Help

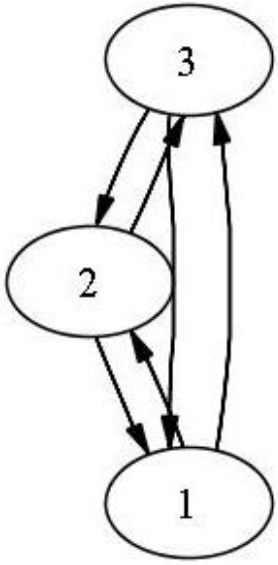
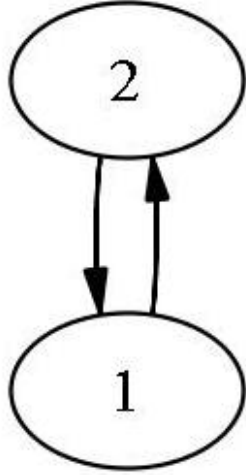
cek37 Go

Getting Started Latest Headlines

file:///C:/Documents%...esktop/tst/login.htm file:///C:/Docume...top/tst/game.htm

EF Game ([return to main page](#))

Spoiler: you (cek37). Duplicator: xyz17.

Graph A	Graph B
	

Previous moves: S1:a1, D1:b2, S2:b1, D2:a3

Your turn (S3): [a1](#) | [a2](#) | [a3](#) | [b1](#) | [b2](#)

[give up](#) | [return to main page \(you may resume the game later\)](#)

Done

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

Mozilla Firefox

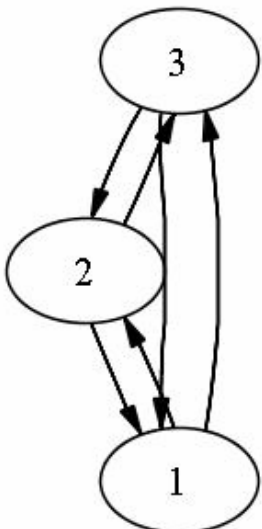
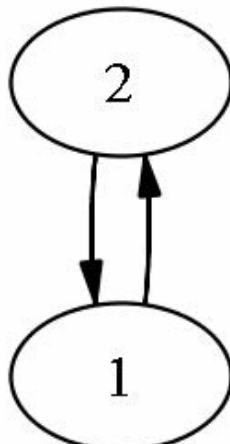
File Edit View Go Bookmarks Tools Help

cek37

Getting Started Latest Headlines

EF Game ([return to main page](#))

Spoiler: cek37. Duplicator: you (xyz17).

Graph A	Graph B
	

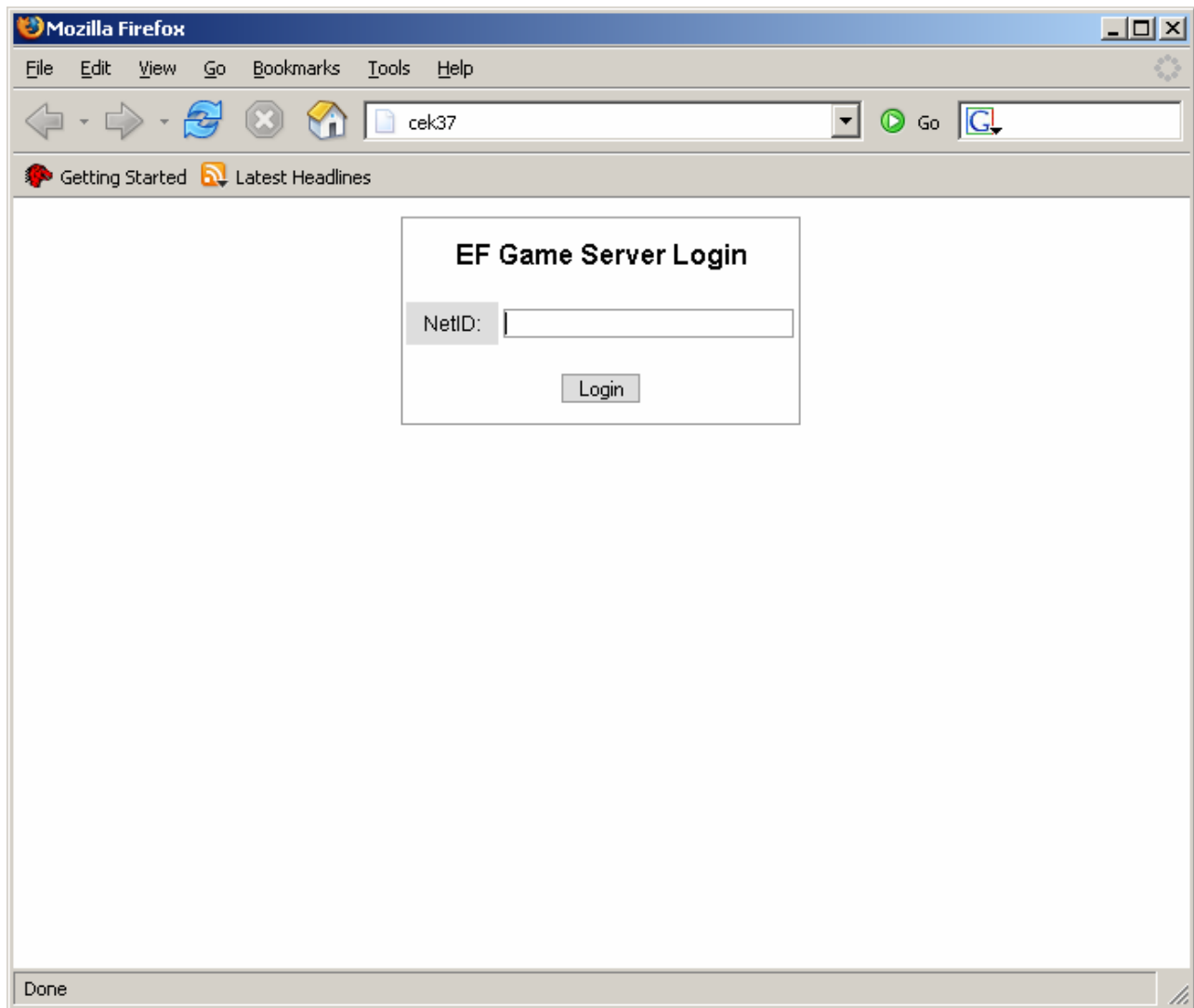
Previous moves: S1:a1, D1:b2, S2:b1, D2:a3

It is your opponent's turn. ([refresh page](#))

[give up](#) | [return to main page \(you may resume the game later\)](#)

Done

Screen shot: Login screen for Assignments 2 and 3 (no passwords).



Screen shot: Creating a new challenge.

