Difficult OOP exercises

(1) Random Text Generator

The task is to write an OO program that generates text automatically. This program should implement a simplified version of an early artificial intelligence model for ”machines writing text”, described below. The generated text is not meant to make complete sense, but rather to *look like* text that has been written by humans. We will use the attached 3-gram file. Each line in the 3-gram file consists of the following 4 columns: *frequency word1 word2 word3*

- *word1 word 2 word3* are an ordered sequence of three words (3-gram)
- *frequency* is how often the above 3-gram occurs in a large collection of English text

For example: *37 a robot to* means that the word sequence *a robot to* occurs 37 times in the text collection.

The program should do the following:

1. Ask the user for a single word (*seed word*). Let’s say the user types *robot*.
2. Find all matches of *robot* in the 3-gram file. The seed word can match any of the three words in a 3-gram. Use exact matching, choose the matching 3-gram(s) and proceed to step 3.
   a. If there is no exact match, use partial matching. The seed word should match at minimum 50% of the letters of a word in the 3-gram file. This match has to be ordered. E.g. *robot* would match *robotized* but not *bored*. Choose the best matching 3-gram(s) and proceed to step 3.
   b. If there is no match between the seed word and any 3-gram, inform the user that you need a bit more inspiration and ask him for another seed word. Repeat the above process until there is at least one match. Choose it and proceed to step 3.
3. Sort the 3-grams that match the seed word by their frequency and split them into 2 equal-sized groups: group 1 contains the most frequent matching 3-grams, and group 2 contains the least frequent matching n-grams. Generate a random number between 1-2. If it is 1, choose group 1; if it is 2, choose group 2. From the chosen group, select the single most frequent 3-gram. If there is an odd number of matches, group 1 should have the extra 3-gram.
4. Start a sentence with three dots (…) followed by the selected 3-gram. For example, if the chosen 3-gram is *a robot to*, output: *... a robot to*
5. Take word2 and word3 of the above 3-gram (*robot to*) and find all 3-grams in the file that contain them as word1 and word2 respectively. I.e., try to find 3-grams that begin with *robot to*. Find all such 3-grams using ***only*** exact match, rank them by their frequency and repeat the selection described in step 3. Append the single last word of the new matching 3-gram to the sentence you are building.
   a. If you cannot find a 3-gram that begins with word2 and word3 of the original 3-gram, try to find a 3-gram that begins with word3 of the original 3-gram. I.e. if you cannot find a 3-gram that begins with *robot to*, try to find a 3-gram that begins with *to.* Repeat the selection process described in step 3 and append the last two words of the selected 3-gram to your sentence. Let’s say your next 3-gram is *to gain access*. The sentence you are building is now: *... a robot to gain access*.
6. Repeat the above process to append more words to your sentence. The above example could, for instance, hypothetically produce something like: *... a robot to gain access to transportation* and then there might be no more matches.

7. When a match cannot be found, end the sentence with a fullstop. Start a new sentence with a random 3-gram that begins with *a* or *the* or *this*. Convert the first letter of your new sentence to capital. Repeat the above process until a match cannot be found or until your sentence is 30 words long. If the sentence you are building is longer than 30 words, stop appending words to it, add three dots, and start a new sentence as described in step 6. This could, for example, produce something like *… a robot to gain access to transportation. This poor little girl has to raise two crew members of your dreams and hopes for peace of westphalia.*
8. Output the first five sentences you generate and ask the user if he would like to try another word. Ask also the user if he would like to have the output printed as text on the terminal or played as audio. If he chooses audio, feed the output to .NET's speech synthesiser. See here: https://msdn.microsoft.com/en-us/library/system.speech.synthesis.speechsynthesizer(v=vs.110).aspx?cs-save-lang=1&cs-lang=fsharp#code-snippet-1

(2) Basketball simulator

Write an OO program that simulates basketball games between all the national teams participating in the Eurobasket 2015 championship and finds the champion. The program initially fetches player statistics for all players of all national teams from the Eurobasket website, and then starts playing games between pairs of randomly chosen teams. At the end of each game, the losing team is thrown out of the championship and the winning team proceeds to play against another randomly chosen winning team. This process is repeated until a single winner is left; that team is the champion.

How the simulation works:

We collect for each player the following statistics:

- the average number of minutes he has played per game (Minutes)
- the average number of points he has scored per game (Points)
- the average number of fouls he has committed per game (Fouls).

We choose randomly two teams to play against each other. The game lasts 60 minutes, and there are 5 players in each team. A team can substitute any number of players freely.

We start the game by choosing 5 players from each team: we choose the 3 players with the highest Points values, and we further choose randomly two more players from that team.

Each player will stay in the game for the exact duration of his own Minutes, without interruptions. I.e. he will not pause playing for a bit and then resume playing. During that time, each player will score his Points as follows:

- The value of his Points is multiplied by a factor that indicates how good or bad of a day he is having. The factor can be either 1.1 (good day) or 0.9 (bad day); one of these two values is chosen randomly.
- The value of (Points * factor) is distributed equally among the Minutes the player is in the game. E.g., if a player has 7.6 Points, 10 Minutes, and a good day, he will be scoring 7.6 * 1.1 / 10 = 0.836 points per minute during the 10 minutes that he will be in the game.

The value of Fouls of the player is also distributed among the Minutes he is in the game, as follows: if the player has x Fouls, and he is in the game for y Minutes, we divide the y Minutes into x equal intervals, and each foul is committed in the middle of each interval. E.g., if a player has 2 Fouls and plays for 10 Minutes, the first foul is committed at 2.5 minutes, and the second at 7.5 minutes.

When the player's time is up (i.e. he has completed playing for the value of his Minutes), he leaves the game and the player with the highest Points value on the bench of his team (i.e. not in the game) replaces him.
Let's assume that team A is playing against team B. If the collective fouls of all the players in team A at any given time are more than 14, one of the following two things can happen (choose randomly):

- The player of team B with the highest Points currently in the game is taken out of the game with an injury and is replaced by the next best player on the bench of his team. Team B scores 3 more points against team A.
- 10 minutes of play are removed from the best player of team A. If that player has less than 10 minutes of play left to him, the remaining minutes are removed from the next best player in that team. If there are no more players left on the bench of that team, the team will have to play with fewer players for the rest of the game.

In the last 15 minutes of the game, the following happens to the team who is losing:
The points remaining to be scored by the players of the losing team are multiplied by:

- 1.1 if the difference in score is 1-4 points
- 0.9 if the difference in score is 5-10 points
- 0.8 if the difference in score is 11-15 points
- 0.7 if the difference in score is 16 points or more

The game finishes after 60 minutes of play, even if there is still time left in the Minutes of some players. We assume no extra time.

The statistics for each player can be collected from the player listings here:
http://www.eurobasket2015.org/en/pageID_3PuY9mb4JskhPFFaT3KY72.compID_qMRZdYCZI6EoANOrUf9le2.season_2015.roundID_9322.html
Several players return a broken link – you can ignore them.

There are different ways to get the statistics from the webpage of each player. For example, I used:

```
open System
open System.IO
open System.Net

let http (url: string) =
    let req = WebRequest.Create(url)
    use resp = req.GetResponse()
    use stream = resp.GetResponseStream()
    let reader = new StreamReader(stream)
    reader.ReadToEnd()

let player = http
"http://www.eurobasket2015.org/en/compID_qMRZdYCZI6EoANOrUf9le2.season_2015.roundID_9322.teamID_381.playerID_62895.html"
Console.Write(player)
```

This outputs the html version of the player's webpage to the terminal. I then pipe it to html2text, which converts it to simple text format, and then I grep:
- the Minutes of game played: 2 lines above **** PHOTO GALLERY ****
- the Fouls: 4 lines above **** PHOTO GALLERY ****
- the Points: 1 line above **** PHOTO GALLERY ****