The Chat Room - Documentation

A text-based adventure game, made with .xml-files

The Pitch

The Chat Room is a dialogue-based text adventure in which the player gets connected to one NPC after another and is asked several **questions**. Every question can only be answered with **Yes** or **No**, influencing the NPC's reaction and the dialogue's further process.
From a designer's perspective, each NPC's dialogue can be scripted with **.xml-files**, that are read by the software and automatically put into the game.

For this game's development, I wanted to include an efficient and convenient way to create, manage and change dialogues.
I came to the conclusion that the best solution for developing those dialogues is using .xml-files, due to their clear node-based system.

With that, the creation, access and manipulation of data doesn't directly happen by the player's input, but the game's designer with .xml-files.

Tree Data Structures: The Dialogue System

Due to the design structure of the yes-no reply system, the whole dialogue system for each character is sorted in a binary tree data structure.
Every question (here: a) links to two possible reactions and their following scenes i.e. next two possible questions (b or c).

Each character's whole dialogue is written in **xml**, with its nodes structured like in the following:

```
<scenes nextCharacter=”character a" altNextCharacter="character b">
 <question id="a">
  <lines>
   <line>QUESTION text</line>
  </lines>
  <reaction id="Yes" nextScene="b">
   <lines>
    <line>NPCs REACTION text, if yes</line>
   </lines>
  </reaction>
  <reaction id="No" nextScene="c">
   <lines>
    <line>NPCs REACTION text, if no</line>
   </lines>
  </reaction>
 </question>
 [... more <question> nodes]
</scenes>
```

Each .xml-file is read right before the beginning of every dialogue, immediately stating the first question. Subsequently, it's a steady for-loop of stating the reaction and then the next question, if whether the current NPC nor the player disconnects from the chat.

A **disconnection** by the player happens if he types DISCONNECT into the chat and breaks out of the chat. An NPC can also trigger this function, if the attribute *nextScene* in the called reaction is labeled with DISCONNECT. The latter represents the ending of the current dialogue.

Example, **Amber.xml:**

```
<question id="1">
  <lines>
    <line>hey, this is Amber</line>
    <line>glad, when I'm gonna be finished with it in a few months</line>
    <line>what about you? Do you also hate high school?</line>
  </lines>
  <reaction id="Yes" nextScene="2">
    <lines>
      <line>wow so we suffer from the same fate</line>
      <line>finally someone who can understand my pain</line>
    </lines>
  </reaction>
  <reaction id="No" nextScene="DISCONNECT">
    <lines>
      <line>one of those who like rules, huh?</line>
      <line>well, I'm gonna be glad when I'm gone from that place</line>
      <line>Goodbye, man</line>
    </lines>
  </reaction>
</question>
```

This is the first question of the game's first character, Amber. A disconnection is triggered, when the player types *no* in this situation, due to the *nextScene*-attribute, triggering the output:

```
- hey, this is Amber
- glad, when I'm gonna be finished with it in a few months
- what about you? Do you also hate high school?
yes
- wow so we suffer from the same fate
- finally someone who can understand my pain
```

Subsequently, the game continues with the next questions.

With the use of disconnections, the game creates a **non-regular** and **non-complete binary tree.**

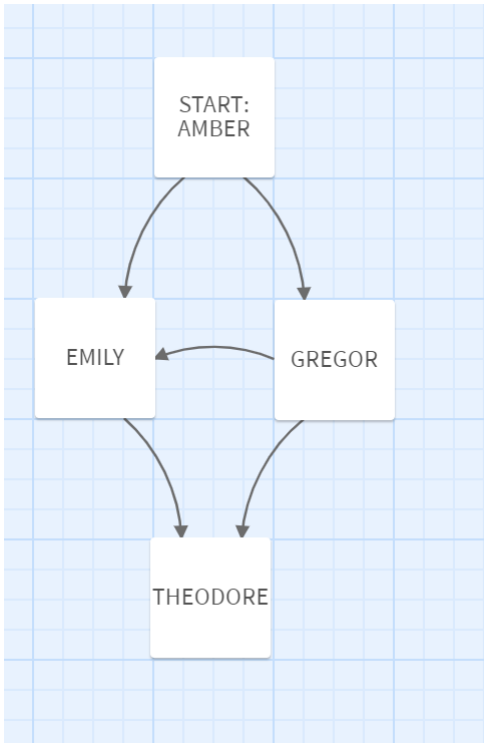
Graph Data Structure: Connecting to NPCs

After every disconnection, the game connects to a new character. This character is not chosen randomly, but set in the **root-node** in the previous character's file with:

```
<scenes nextCharacter="Emily" altNextCharacter="Gregor">
    [..questions]
<scenes>
```

The designer can set two characters. The default character *nextCharacter* is triggered, when the NPC disconnects from the chat.
The alternative character *altNextCharacter* is triggered, when the player chooses to disconnect from the chat.

This enables a graph-like structure, representing how to transition (edge) from one character (vertex) to another:



Linear Data Structures: The End Results

By the end of the game, the player gets to see a list of all the NPCs (*character*) he encountered, including the number of all the questions he got asked.
This function is based on a linear data structure, namely the C++ form of a dictionary's system: **the *std::map*.**

After each question, a counter *int* ***questionCount*** is incremented and at the end of the current *character's* dialogue, *questionCount* is added as a value to the map *rating*, with the key being the *character's* name:

```
rating[character->name] = questionCount;
```

So each element is adressed in map rating<string character->name, int questioncount>.

With a for-loop, the output is created at the end of the game:

```
End of Session

Summary of today's session:
 Amber: 3 Questions
 Emily: 2 Questions
 Theodore: 1 Questions
```

Sources:

C++, general:

http://www.cplusplus.com/

C++, std::map:

https://www.techiedelight.com/print-keys-values-map-cpp/

C++, pointers:

https://www.w3schools.com/cpp/cpp_pointers.asp

RapidXml:

http://rapidxml.sourceforge.net/

https://gist.github.com/JSchaenzle/2726944