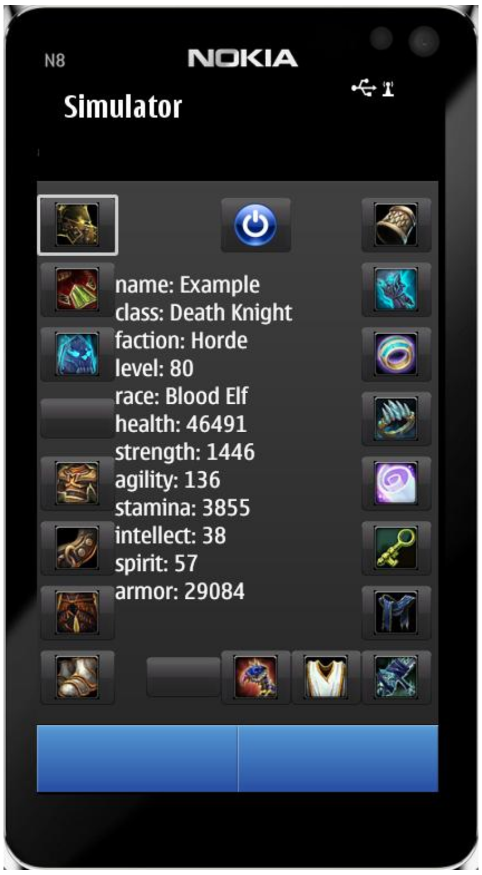
Antti Hannuksela (antti.hannuksela@lut.fi)

Otto Luttinen (otto.luttinen@student.saimia.fi)

Riku Pulliainen (riku.pulliainen@gmail.com)

WoWMobileArmory



name: Example
class: Death Knight
faction: Horde
level: 80
race: Blood Elf
health: 46491
strength: 1446
agility: 136
stamina: 3855
intellect: 38
spirit: 57
armor: 29084

The Idea

World of Warcraft Armory is character and item database of the game. Its main goal is to provide browser based user interface for in game data, like character/item stats, talents, in-game calendar, auction house and much more. Our goal was to make a simple front-end for QT-capable mobile phones, allowing users to browse character's basic stats and item stats worn by characters.

Program usage is fairly simple. First you enter character name and realm, and then program fetches character and item data from Blizzard's servers and displays it similarly as browser-based armory.

Because idea behind our application was already invented, our problems were only technical and making decision about what to implement, what kind of graphical user interface would be good and so on.

In character stats user interface we ended up using armory-like view, displaying items in three bars: left, right and bottom. Normally, in middle of the user interface would be 3d model of character, but as we haven't implemented this, we put character stats in middle. Item stat display is opened as popup by pressing icon of desired item.

Character selection interface is very straight-forwarded. It contains two text fields, one for character name and other for realm. For technical reasons, these fields are not on same page and user must press forward button between them.

Technically, program components consist of user interface components, downloader, download manager and XML parser.

Technical Specifications

There are actually two different layouts in the program. First layout consists of Begin button and two different QInputDialogs asking character name and realm. All those are just aligned to the top of the empty start screen.

Second layout is the main screen where character's stats are shown in textLabel which is positioned in the middle of the screen. Item slots are divided to three bars positioned in left side, right side and bottom of the screen. The quit button is located in the top of the screen. All item slots and quit button are QPushButtons. The alignment is done using QGridLayout. Item stats are shown when the icon of the item is clicked. QMessageBox is opened to the front of the current screen to show item stats.

Download manager is practically just map containing multiple downloaders and slots where downloader signals are connected. Manager's task is to ensure that all download processes have finished before user interface components are drawn and that downloader objects are not killed before they have finished. Once everything required is downloaded, manager signals main program that user interface components are ready and exits.

Downloader is simple component, it gets address of page or file to be downloaded, sends HTTP request and saves reply(=requested file) to disk in cache folder.

XML Parser is the component responsible of reading data from XML-files and storing it to QVector for other components to use. Parser is hard-coded to detect what data it is parsing by XML file name. Thus, parser knows what tags or attributes it must search and knows in which vector parsed data goes. Parser uses SAX2 QT Api and QT XML libraries.

Program components aren't designed to handle errors but at least false input doesn't cause any problems. Broken XML-files and failed file transfers have not been tested. Also, application is designed to work when phone is rotated "upwards", using wide screen will probably result in funny things in GUI.

Problems during development

Most of our problems during application development concerned GUI and object lifecycles. GUI was tricky to develop as it was completely new to our team and it had to be real-time editable because of button images. This one took most time of all technical problems we had. Another really tricky thing was lifecycles. The problem was, especially with downloader that caller objects finished before the downloader finished, resulting killing downloader too early. Sometimes we fought similar problem with GUI, when object called GUI creation and then excited, killing GUI. Some of these were fixed with object maps, some with better design and using signals/slots. We found out that Blizzard sends us different XML-files depending on what browser we use and it took us some time to solve the problem and introduce our downloader as Mozilla Firefox, which gave us correct XML-file. Also Blizzards XML data was sometimes tricky to read, but these were quite easily overcome with if/else structures.

Because of lack of time the item stats component is still half-ready, meaning that it fetches stats of all items worn by character but fails to display them correctly. This issue is probably related to vector handling, as it seems that vector containing items stats is not correctly emptied after each stat display call.