Player Profiling Component

TUGraz – T3.4E – Apache 2.0 – Client side C#

About this component

This component will be implemented as a client-side component in C#. It is concerned with evaluating the player in an initial questionnaire. Resulting information is supplied to the game in terms of a set of values. The component contains three predefined questionnaires and an authoring component that allows the modification of these existing questionnaires, as well as the creation of new questionnaires. The component translates the answer patterns into a small set of values, which can be used by the game to pre-configure itself. The component architecture will prevent multiple component creation; only one client-side component of this component per game is needed.

Component mechanics

An XML-file containing all necessary information about the questionnaire and the evaluation of the questionnaire is loaded from a local storage. A supplied authoring tool, accessible via the Integrated Authoring (Tool T3.7), creates this XML. Out of this information a data structure is created by the C# component, which represent the questionnaire. This questionnaire needs to be presented to the player by the game. After filling out the questionnaire the resulting data is sent back to the component, which will evaluate the answers and supply the results to the game.

Component interfaces

For using this component proceed like the following:

- The xml questionnaire can be accessed directly:

  String getQuestionnaireXML()

  Using a static method this xml String can be transformed to a data-type object containing the questionnaire:

  QuestionnaireData QuestionnaireData.getQuestionnaireData(xml)

- The answers of the questionnaire can be returned for the score calculation by using the following method:

  Void setAnswers(Dictionary<String,Integer> answers)

The resulting score can be accessed (for both options) by the following code snipped:

- The result of the questionnaire will be available as text-value combinations. The value can be requested from the component. If null is returned no questionnaire data is available.

  Dictionary<String,Double> getResults()

Component dependencies/requirements

- GameStorage component – used to store questionnaire results

Milestones

Milestone 1

- t1.1: Creating the first version of the design document, defining the API and creating a dummy component with the API implemented

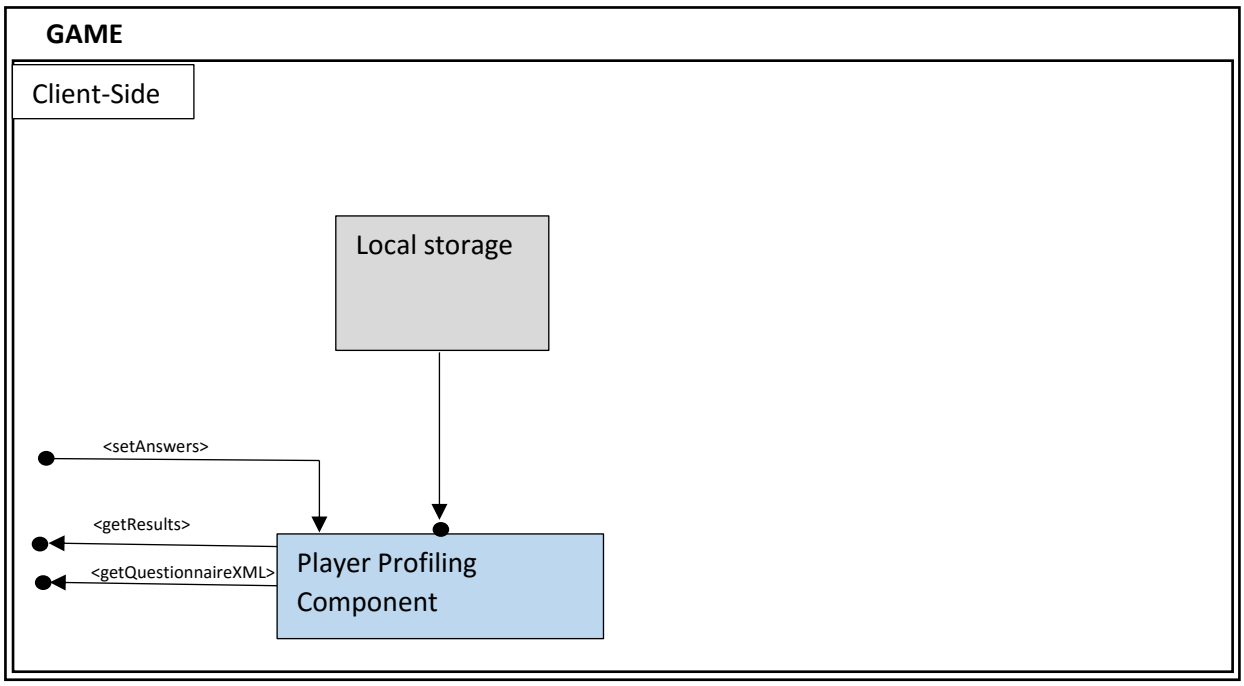
Milestone 2

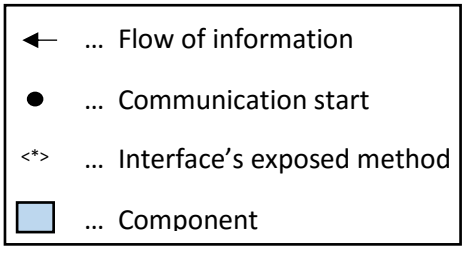
- t2.1: Elaborate XML-Structure containing questionnaire-data.
- t2.2: Create Software component in line with Component-Manager infrastructure.
- t2.3: Elaborate Settings-structure within the Component-Manager infrastructure.
- t2.4: Integrate Game Storage functionality into the component-functionality.
- T2.5: Create Authoring tool for XML-Structure generation

Milestone 3

- t3.1: testing the component with a game
- t3.2: instructions and scripts for building and deploying

Graphical representation





Set up the Component

For setting up this component there is mainly one things to do (additionally to create the component), which is:

- Supply the Questionnaire data via a XML-file to the component. Therefore, a JS-HTML tool will be used (more information where to find and how to use this tool will be supplied at a later stage).

The following code can be used to specify the XML file id:

```
PlayerProfilingAsset ppa = PlayerProfilingAsset.Instance;

PlayerProfilingAssetSettings ppas = new PlayerProfilingAssetSettings ();

ppas.QuestionnaireDataXMLFileId = ”file.xml”;

ppa.Settings = ppas;
```

Use the component

The questionnaire data can be requested from the component using the following code:

```
PlayerProfilingAsset ppa = new PlayerProfilingAsset.Instance;

String xml = ppa.getQuestionnaireXML();

QuestionnaireData qd = QuestionnaireData.getQuestionnaireData(xml);
```

Deployment

For the source code the following GitHub-link can be used https://github.com/RAGE-TUGraz/PlayerProfilingAsset - it contains the Visual Studio solution of the component's C# implementation. Furthermore, the broken links to external component DLLs need to be fixed for each project and the Bridge code need to be adopted to the new environment, e.g. changing the IDataStorage path.

For integration into Unity, the resulting DLLs need to put into a folder in the Unity working-directory.

Unit test

For executing unit tests, the source code need to be open in visual studio and all links need to be fixed. In the test-explorer all tests can be executed.