Realising and Applied Gaming Ecosystem

Research and Innovation Action

Grant agreement no.: 644187

D3.4 – Player-centric rule-and-pattern-based adaptation asset - CONFIGURATION TUTORIAL

RAGE – WP3 – D3.4

DRAFT

Project Number	H2020-ICT-2014-1
Due Date	29 February 2016
Actual Date	January 2016
Document Author/s	Dessislava Vassileva
Version	0.4
Dissemination level	PU
Status	Draft
Document approved by	

This project has received funding from the European Union's Horizon 2020

research and innovation programme under grant agreement No 644187

About the player-centric rule-and-pattern-based adaptation asset

The Player-centric rule-and-pattern-based adaptation asset uses metrics of player's performance, emotional status and/or playing style for realization of dynamical adaptation of various game features such as adaptation of player-driven game tasks and/or game assistance, dynamic adjustment of task difficulty, and/or adjustment of properties of audiovisual content and effects. The asset receives as input registration requests of player-centric metrics together with simple formal definitions of rules and patterns of variation of these metrics during the play time or their features such as mean, deviation and moving average within a desired time window. Next, it receives values of registered metrics and checks each incoming metric value for occurrence of a rule or a pattern defined for the metric or its feature. In case of finding such an occurrence, the asset fires a triggering event about this rule or pattern and executes its event handler, which is to be defined by the game developer depending on his/her goal to adapt specific game feature(s).

Document scope

The present document provides a concise functional description of the player-centric rule-and-pattern-based adaptation asset and, next, presents what settings are required for putting it into action.

The document should be read together with the other tutorials bout this asset, namely the Installation tutorial and the Game integration tutorial.

Description of asset functionality

Due to the very complex and volatile nature of the player's character, player-centric adaptation remains being a crucial issue for both entertainment and applied digital games. To be effective, player-centric adaptation should follow deterministic player models for tracking, measuring and analyzing player's behavior.

The *Player-centric rule-and-pattern-based adaptation asset* uses metrics of player's performance, emotions and/or playing style for realization of dynamical adaptation of various game features such as generated content at micro and/or macro level [1], or task difficulty. More precisely, the player-centric metric may indicate three important issues:

1. player performance, which embraces knowledge and intellectual abilities of the player's including synthetic, analytical and practical skills [2], e.g. abilities for comprehension, memorization, evaluation, reasoning, decision making, and so on;
2. player affective (emotional) status, which represents emotional experiences, feelings and motivation of the player [3]. The affective status may be inferred by measuring facial, gestural, vocal, neural and/or psychophysiological bodily reactions [4];
3. playing style, which depends on player's personality and ways of thinking and reacting to game challenges [5].

The asset receives as input registration requests of player-centric metrics about individual performance, emotional status and/or playing style, together with simple formal definitions (using a simple but yet powerful syntax) of rules and patterns of variation of these metrics during the play time or their features such as mean, deviation and moving average within a desired time window. The idea of using patterns and rules for adaptation purposes is adopted from the game adaptation control framework using implicit derivation of the player character during the game play developed within the scope of the ADAPTIMES research project [12]. Next, it receives values of registered metrics and checks each incoming metric value for occurrence of a rule or a pattern defined for the metric or its feature. In case of finding such an occurrence, the asset fires a triggering event about this rule or pattern and executes its event handler, which is to be defined by the game developer depending on his/her goal to adapt specific game feature(s). Thus, the asset does not depend on any concrete digital game and provides the game developer with freedom to program any control over game adaptation. Various game features can be dynamically adapted, which fall into three main groups regarding the three main component of the Mechanics-Dynamics-Aesthetics (MDA) framework [13], namely game mechanics, dynamics and aesthetics as follows:

- adaptation of player-driven game tasks and/or game assistance such as helping instructions [6]. As well, the managed appearance of tasks and assistance in their performing during the game flow can be adjusted [7];
- dynamic adjustment of task difficulty – like in [8] where the adaptation control is based on the player's anxiety, or in [4] where adaptation is controlled according the skill level of the player;
- adjustment of properties of audiovisual content and effects, such as ambient light in rooms in a video game [9].

The asset works only on the client side. It can be used without any other asset provided the game developer is able to provide the player-centric metric(s) used as a basis for game adaptation. Otherwise, the developer should use another asset(s) for provisioning of the metric(s) values, such as the Real-time Emotion Detection Asset or the RT Arousal Detection Using Galvanic Skin Response Asset.

For understanding the installation process of the asset, refer to the Installation tutorial. For realizing how the asset is to be used for game adaptation together with a video game based on the Unity 3D game engine [14], see the Game integration tutorial.

Asset configuration stages

The asset configuration process includes four stages similarly to the adaptation methods described in [12]:

Registering player-centric metrics

Registering player-centric metrics for monitoring whose variation is to be observed by the asset – metrics regarding player's performance (knowledge and intellectual abilities), effective status (emotional experiences, feelings and motivation) or about the playing style are to be registered by names chosen by the game developer by calling the `public bool RegisterMetric(String metricName)` method selecting the metric features the developer in interested in. Features initially supported are average, standard deviation and moving average for a predefined time window (can be changed by the game developer).

Example:

```
 PatternBasedAdaptationAsset testMetricPattern = new PatternBasedAdaptationAsset();

 testMetricPattern.RegisterMetric("Number of hits");
```

Setting adaptation triggering rules and patterns

At this stage, one or more rules or patterns of change of an already defined metric or its feature are to be defined by using a simple but yet powerful syntax, by calling `public bool RegisterPattern (String patternName, String metricName, String featureName, String timeInterval, String valuesRule)` method for setting rules/patterns. Every rule/pattern, when fired, triggers an adaptation event, which causes execution of the event handler specific for rule/pattern (one handler may be used for several riles/patterns). For explanation of the types of patterns and rules, see the next section of this tutorial.

Example:

```
PatternBasedAdaptationAsset testMetricPattern = new PatternBasedAdaptationAsset();

testMetricPattern.RegisterPattern("Change color in blue", "Number of hits", "none",
"GT(1)", "2");

testMetricPattern.RegisterPattern("Change size", "Number of hits", "none", "GT(1)",
"GT(0)");
```

Definition of adaptation event handlers

For each adaptation event triggered upon specific rule or pattern, a specific custom event handler will be executed upon the rule/pattern name, acting as adaptation method changing some game features about game mechanics, dynamics and aesthetics [10]. The custom event handler will implement the ***PatternEventHandler*** method of the abstract class ***PlayCentricStatisticExtractor***.

Example:

```
class PatternBasedAdaptationAsset : PlayCentricStatisticExtractor

{

        public override Object PatternEventHandler(Object patternInput, Object
gameObject) {

            PatternAction((System.Collections.Generic.List<string>)patternInput,
(UnityEngine.GameObject) gameObject);

            return null;

        }

      public void PatternAction(System.Collections.Generic.List<string> patterns,
UnityEngine.GameObject gameObject)

        {

            if (patterns.Contains("Change color in blue"))

            {

                gameObject.GetComponent<UnityEngine.Renderer>().material.color =
UnityEngine.Color.blue;

            }

            else if (patterns.Contains("Change color in green"))

           {

                gameObject.GetComponent<UnityEngine.Renderer>().material.color =
UnityEngine.Color.green;

            }

            ……………………………

            if(patterns.Contains("Change size"))

           {
                UnityEngine.Vector3 currentScale = gameObject.transform.localScale;

                gameObject.transform.localScale = currentScale - currentScale * 0.15f;

            }

        }

}
```

Setting the global time and the moving average time window for the asset

The game developer can set/reset the asset timer at any time (the time is given in milliseconds) in order to synchronize it to the game engine by using the `public void SetGlobalTime(int synchronizationTime)` method, as far as setting the moving average time window by using the `public void SetTimeWindow(int milliseconds)` method (as explained below).

Example:

```
PlayCentricStatisticExtractor test = new PlayCentricStatisticExtractor();

//the global time will be 2 minutes

test.SetGlobalTime (120000);

//the time window will be equal to 40 seconds

test.SetTimeWindow (40000);
```

Next, game metrics values can be send to the asset by calling its method for receiving a metric value for given metric (specified by its name). If an occurrence of specific pattern/rule will be found, the asset automatically triggers the event and executes its event handler producing adaptation changes in the game desired for this pattern/rule. Therefore, no explicit output is produced by the asset.

Configuration of asset patterns and rules

For any player-centric metric registered at the asset (and/or one of its features), the game developer can set various patterns or rules of changing the values of the defined metric or one of its features. Patterns can be easily defined by the game developer for absolute or relative values of player-centric metrics such as number of shots, number of hit objects, values of player's arousal, etc. Game developer can use the metric value or, as well, values of any feature of this metric. For the current version of the asset, several metric features are supported, as follows:

- ***none*** – no features are applied in pattern definition for the particular metric

- *`average`* – the average value for the metric since staring registration of values of the chosen metric at the asset
- *`standard deviation`* – for the average of the metric
- *`moving average`* – the moving average for the metric values registered at the asset within a time window (the time window duration is to be set in advance before starting registration of values for all the metrics)

A pattern or rule may refer to absolute or relative moments of time. Using relative time vector means a cyclic loop starting at the first vector element. Example of sample definitions of patterns and rules can be the following lines below.

- Pattern example with absolute feature value using relative time moments after time t, namely t+3000, t+6000 and t+9000 milliseconds, i.e. **`{ name=″GSR mean pattern″, metric=″GSR″, feature=″average″, time=″t t+3000 t+6000 t+9000″, values=″16 20 24 20″ }`**:

```
testMetricPattern.RegisterPattern("GSR mean pattern", "GSR", "average", "t t+3000
t+6000 t+9000", "16 20 24 20");
```

- Pattern example with relative value x of the moving average using absolute time (at the second, fifth and the tenth minute), i.e.
`{ name=″Happy pattern″, metric=″happiness″, feature=″moving average″, time=″120000 300000 600000″, values=″x  x+10  x-20″ }`

```
testMetricPattern.RegisterPattern("Happy pattern", "happiness", "″moving average",
"120000 300000 600000", "x x+10 x-20");
```

- Rule example for checking if the metric about quiz result is between 20 and 30 points (GT stands for Greater Than, LT stands for Less Than) between the third minute (after 180000 ms) and before the eighth minute (before 480000 ms):
`{ name=″A quiz points rule″, metric=″Quiz result″, feature=″none″, time=″GT(180000) LT(480000)″, values=″GT(20) AND LT(30)″ }`

```
testMetricPattern.RegisterPattern("A quiz points rule", "Quiz result", "none",
"GT(180000) LT(480000)", "GT(20) AND LT(30)");
```

In fig. 1, the ”GSR mean pattern” is shown to be found at times t1=15s (points in red color) and t2=20s (points with yellow color) after starting the game, i.e. for the times sequences “6 9 12 15” and “11 14 17 20” seconds.

Fig. 2 presents two possible occurrences of the “Happy pattern” for x=42 and x=60, where pattern points are given in yellow.

Note that using relative time moments after given moment of time t means checking for an occurrence of the specified metric/feature vector at every moment after t, when a new value of the metric is sent to the asset, i.e. the check is repeated always after t until the metric is deregistered for monitoring. If the game developer needs to specify a cycling check of a rule for given time period, he/she can do it by using a cycle construction as follows:

Rule example for starting to check each one minute 10 min after the start of the game within the next 5 min for the condition if the number of shots metric is greater than 100:

```
{ name=”A shot rule”, metric=”No of shots”, feature=”none”,
  cycletime=”GT(600000)  6000  LT(900000)”, values=”GT(100)” }
```

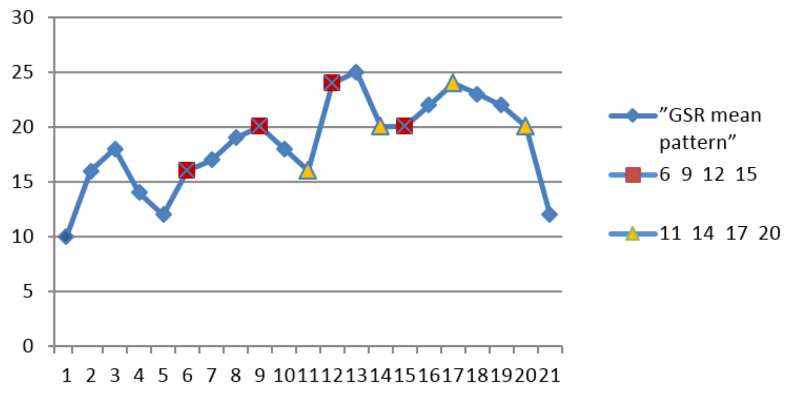

Fig. 1: Two occurrences of a ”GSR mean pattern” found at times t1=6s (points in red color) and t2=11s (points with yellow color)

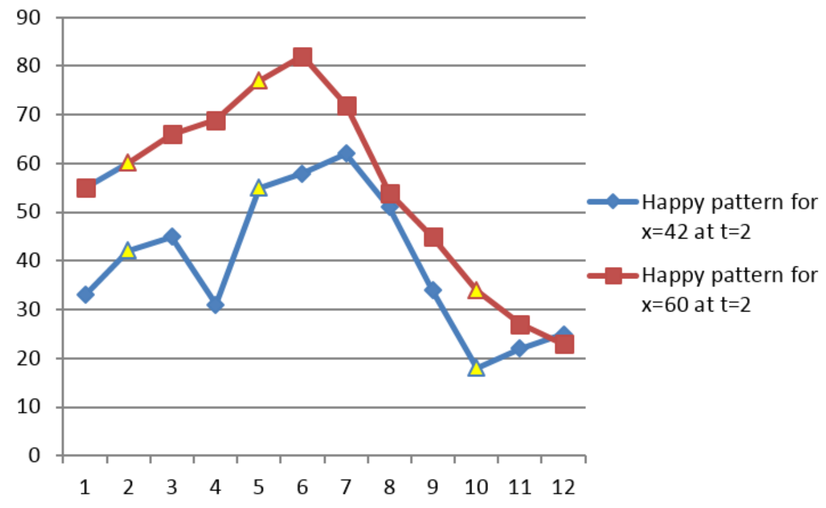

Fig. 2: Two possible occurrences of the “Happy pattern” for t=[2 5 10] minutes (pattern points are given in yellow)

During playing time, monitoring of registered metric can be canceled by calling the ***`public bool RegisterMetric(String metric)`*** method with the metric name, which will incur stopping the monitoring of this metric and evaluation of its features. For a monitored metric, any of its patterns or rules set for observing can be stopped by calling the ***`public bool UnregisterPattern(String patternName)`*** method.

Examples of use in games

The player-centric adaptation technology continues being very promising and attractive for both entertainment and applied video games. State current examples if it is already in use. Various event handlers can be developed for game adaptation based on changes in player's character, namely:

- Player's performance – including knowledge, motivation, skills and abilities
- Player's affective state - emotions and arousal;
- Player's style – such as killer/achiever/explorer/socializer of Bartle [11] or conqueror/manager/wanderer/participant styles of Bateman and Boon [5]

The player character is to be implicitly derived during the play process, instead of explicit ways like using self-reports. The asset can be used for adapting various features regarding game mechanics, scenarios, dynamics, or aesthetics, as programmed in the event handlers by the game developer.

Value

It is not possible to predict all possible types of features to be adapted even for a single complex game. Therefore, for not losing generality and for preserving the blackbox communication (the asset has no information about the game logic, scenarios, dynamic variables, etc.), the game developer is free to overwrite the default event handler as he likes. Thus, the asset offers:

- Independence from game logic, dynamics, and scenarios – includes independence from the adaptivity scale (micro/macro adaptivity [1]);
- Independence from game engine used;
- Easy use and integration – no specific domain expertise is required to use the asset, however, the game developer should be able to define reasonable and consistent patterns and rules of metric variation. Thus, the asset can be used by trainers, pedagogues and psychologist for creating various adaptive applied games;

- Flexibility - patterns and rules of metric variation can be set and cancel for monitoring during run time;
- Monitoring of heterogeneous player-centric metric – asset user is able to specify various metric about player’s performance, emotional status and playing style. For example, for the playing style can be observed at given together several metric and specific playing style can be inferred based on their change.

Dependencies

The asset can be used in isolations provided the player-centric metrics are known by the developer at run time. Otherwise, the game developers may use it together the Real-time Emotion Detection Asset or the RT Arousal Detection Using Galvanic Skin Response Asset, in order to provide emotion or arousal metric, respectively.

Technical details

The asset has only a client component (in form of dll - RuleBasedAdaptation.dll). Its run-time requirements are as follows:

- Windows Vista or higher
- Microsoft .NET Framework 3.5

Source project requirements:

- Microsoft Visual Studio Professional to import the solution (project).
- Support for C#.

Licensing

No specific licensing is required.

References

1. Kickmeier-Rust, M. D., Albert, D. Educationally adaptive: Balancing serious games. Int. Journal of Computer Science in Sport, 2012, 11(1), pp.1-10.
2. Tremblay, J., Bouchard, B., Bouzouane, A. Adaptive Game Mechanics for Learning Purposes-Making Serious Games Playable and Fun, Proc. of CSEDU (2), 2010.
3. Tijs, T., Brokken, D., IJsselsteijn, W. Creating an emotionally adaptive game. ICEC 2008, LNCS 5309, Springer Berlin Heidelberg. pp.122–133.
4. Fairclough, S., Gilleade, K. Construction of the biocybernetic loop: a case study. Proc. of the 14th ACM Int. Conf. on Multimodal interaction, ACM, 2012, October, pp. 571-578.
5. Bateman, C., Boon, R. 21st Century Game Design, vol. 1. Charles River Media, Londan, 2005.
6. Murphy, C., Chertoff, D., Guerrero, M., Moffitt, K. Design Better Games: Flow, Motivation, and Fun. Design and Development of Training Games: Practical Guidelines from a Multidisciplinary Perspective, 2014, p.1773.
7. Sweetser, P., Johnson, D. M., Wyeth, P. Revisiting the GameFlow model with detailed heuristics. Journal of Creative Technologies, 2012 (3).
8. Rani P., Sarkar N., Liu C. Maintaining optimal challenge in computer games through real-time physiological feedback. Proc. of the 11th Int. Conf. on Human Computer Interaction, 2005, pp.184–192.
9. Grigore, O., Gavat, I., Cotescu, M., Grigore, C. Stochastic algorithms for adaptive lighting control using psycho-physiological features. Int. J. of Biology and Biomedical Engineering 2, 2008, pp.9–18.
10. Parnandi, A., Gutierrez-Osuna, R. A comparative study of game mechanics and control laws for an adaptive physiological game. J. on Multimodal User Interfaces, 2014, pp.1-12.
11. Bartle, R. Hearts, Clubs, Diamonds, Spades: Players Who suit MUDs, 1996, http://mud.co.uk/richard/hcds.htm
12. Bontchev, B. Methods of adaptation control by implicit derivation of the player character during the game play, Deliverable D3, Version 1.0, ADAPTIMES – WP3 – D3
13. Hunicke, R., LeBlanc, M., Zubek, R. MDA: A Formal Approach to Game Design and Game Research, 2005, CiteSeerX: 10.1.1.79.4561
14. Smith, M., Queiroz, C. Unity 4.x Cookbook, Packt Publishing, 2013