Realising and Applied Gaming Ecosystem

Research and Innovation Action

Grant agreement no.: 644187

D3.4 – Adaptive Roll-a-ball: A game demo of player centric rule and pattern based adaptation asset

RAGE – WP3 – D3.4

FINAL

Project Number	H2020-ICT-2014-1
Due Date	18 January 2017
Actual Date	January 2017
Document Author/s	Dessislava Vassileva
Version	0.3
Status	Final

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644187

About the game demo of player-centric rule-and-pattern-based adaptation asset

The simple game was developed following the Unity 3D Roll-a-Ball tutorial (https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial) in order to be demonstrated how the RAGE player-centric rule-and-pattern-based adaptation asset can be configured, integrated and used in a Unity 3D game as an asset. The adaptation added to the Unity 3D Roll-a-Ball game is based on the asset functionality and depending on the player performance. The demo game can be exported to and run in all platforms that the Unity 3D IDE supports (Windows, Linux, Mac, iOS, Android, etc.).

Document scope

The present document provides a functional description of the Adaptive Roll-a-ball game using the RAGE player-centric rule-and-pattern-based adaptation asset and, presents how to integrate and use the asset in a Unity or C# based game the asset.

The document should be read together with the asset design document and the other tutorials about this asset, namely the Installation tutorial and the configuration 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.

Description of the Adaptive Roll-a-ball demo game

The Adaptive Roll-a-ball game is based on Unity 3D Roll-a-Ball tutorial (https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial) where the player controls a ball rolling around a game board. The ball moves upon player's input (arrow keys) and has to pick-up target objectes (presented as cubes). Each hit cube (with the ball) increments the result. The goal is each cubes to be hit as faster as possible.

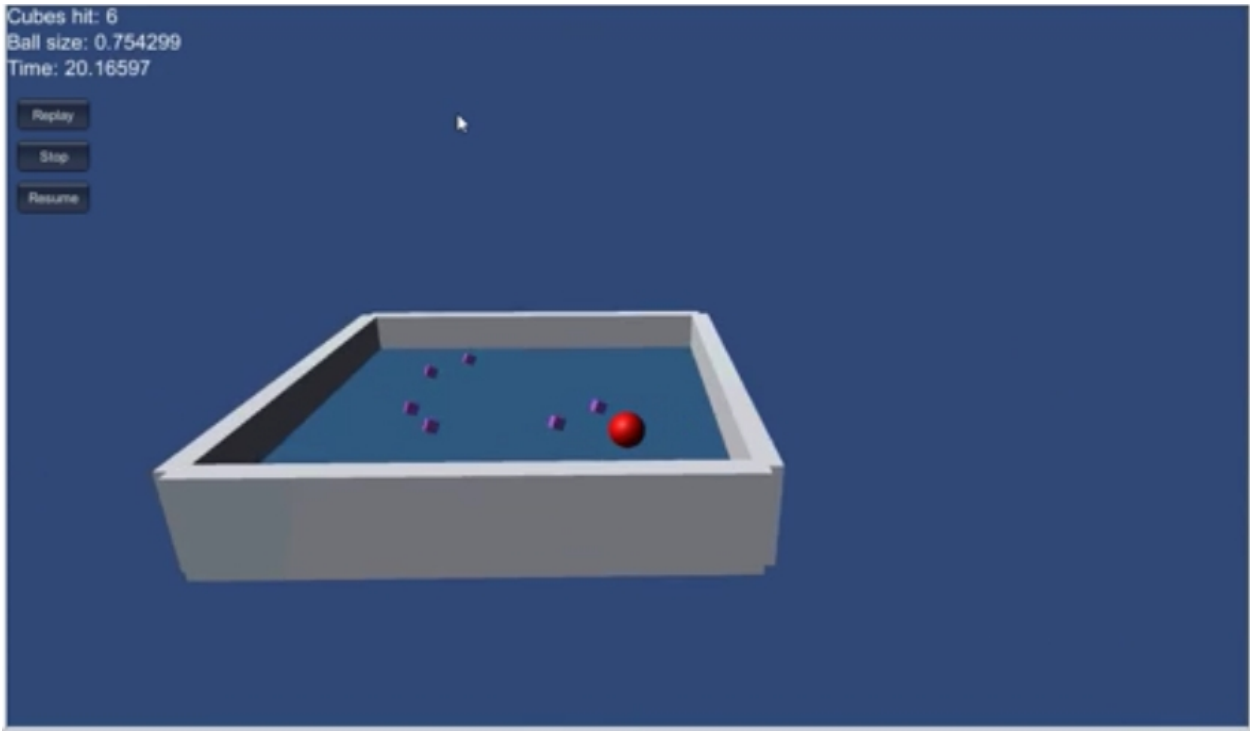

Fig. 1: A screenshot of the Adaptive Roll-a-ball game.

The game is adaptive upon player performance – with each collected cube, the size the ball decrease making next hit to be more difficult (fig. 1). The ball color depending on the number of collected balls, as well. Thus the number of hitting cubes serve as a metric for player-centric adaptivity and used by the RAGE player-centric rule-and-pattern-based adaptation asset.

Asset integration steps

The asset integration process includes following steps:

1. Download the zip file “PlayerCentricRuleBasedAdaptationDlls.zip” with dll files implementing the asset functionality;
2. Copy the asset dll archive file in a folder in your Unity project and uncompressed it there;
3. Import dll files to the Unity project;
4. Add dll files to the references of your project in Visual Studio or other IDE that you use for your C# code;
5. Add to your game project in Visual Studio a class extending the abstract class *Assets.Rage.PlayerCentricRulePatternBasedAdaptationAsset.PlayCentricStatisticExtractor*;
6. Implement the method *public abstract Object PatternEventHandler(Object patternInput, Object gameObject).* This method has two parameters. The first of them patterInput contains the list of founded patterns, and the second parameter – gameObject – presents the game object, where the asset is used. The method should defined what action will trigger on the game object depending on founded patterns. An example of an implementation is given below.

```
using System;

namespace Assets.Rage.PlayerCentricRulePatternBasedAdaptationAsset

{

    class PatternBasedAdaptationAsset :
Assets.Rage.PlayerCentricRulePatternBasedAdaptationAsset.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;

            }

            else if (patterns.Contains("Change color in green2"))

            {

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

            }

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

            {

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

            }

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

            {

                gameObject.GetComponent<UnityEngine.Renderer>().material.color =
```

```
UnityEngine.Color.white;

            }

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

            {

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

            }

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

            {

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

            }

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

            {

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

            }

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

            {

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

            }

            else if (patterns.Contains("Change color in blue2"))

            {

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

            }
```

```
            if(patterns.Contains("Change size"))
            {
                UnityEngine.Vector3 currentScale = gameObject.transform.localScale;
                gameObject.transform.localScale = currentScale - currentScale * 0.15f;
            }
        }
    }
}
```

7. It should be create an instance of the asset in the class related to the game object that will use the asset. In the method `void Start()` has to be defined and registered all metrics (with the method `public bool RegisterMetric(String metricName)`) and patterns (with the method `public bool RegisterPattern(String patternName, String metricName, String featureName, String timeInterval, String values)`). When a metric value is changed, the new value has to be set for the correspond metric with the method `public List<String> SetMetricValue(String metricName, int value)`. This method returns the list with all patterns founded depending on the new metric value. At the end the event handler (`public void PatternAction (List<string> patterns, UnityEngine.GameObject gameObject)`) will be executed upon the list of rule/pattern name, acting as adaptation method changing some game features about game mechanics, dynamics and aesthetics [10].

Technical details

The demo game was developed under Windows 8.1 and with Unity 3D IDE version 5.3.0f4. Correspondingly it can be run in all building format supported to the Unity 3D IDE version 5.3.0f4 that can be run on Windows, Linux, Mac, iOS, Android, etc. (fig. 2).

The code source of the Adaptive Roll-a-ball game can be found and downloaded at: https://github.com/ddessy/PlayerCentricRuleBasedAdaptation/tree/master/UnityDemo_Code. An execution version of the demo game (it can be run on the browser firefox version 50.1.0 or higher) is available at: https://github.com/ddessy/PlayerCentricRuleBasedAdaptation/tree/master/UnityDemo_Bin

Licensing

References

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.