Implementation and Playtesting for a world adventure game's Procedural Content Generation System

João Paulo Sousa[1[0000-0002-9005-084X]] Gonçalo Oliveira[2[0000-0002-0129-3613]] Inês Barbedo[1[0000-0001-6350-9697]] Bárbara Barroso[[0000-0001-6350-9697]] Rogério Tavares[1[0000-0001-9366-2617]] Rui Pedro Lopes[1[0000-0002-9170-5078]]

[1] Research Centre in Digitalization and Intelligent Robotics (CeDRI); Instituto Politécnico de Bragança, Bragança, Portugal
[2] Instituto Politécnico de Bragança, Bragança, Portugal
jpaulo@ipb.pt, a42070@alunos.ipb.pt, inesb@ipb.pt, bbarroso@ipb.pt, rogerio.tavares@ipb.pt, rlopes@ipb.pt

***Abstract*—**This paper presents an experimental methodology of procedural content generation (PCG) for natural environments with a focus on game design and visualization. This approach allows us to define a set of instructions to shape the content generation to meet the requirements of game design and level design teams. For the validation process, we have discussed and analyzed a playtesting session.

Keywords: PCG, Procedural content generation, Game Design, Open-world Adventure Game, Computer Game.

1 Introduction

Procedural Content Generation (PCG) refers to the practice of generating game content through an AI-processed procedural generation method. PCG has been used in the game and cinema industry since the 80's for several different purposes. Some of the popular motivations are replayability, adaptability, and economics. Due to the amount and diversity of automatically generated content, it is possible to offer more refreshing gameplay each time the player starts the game, they will have a different experience. In a more advanced application, the PCG AI can be sensitive to the player skills, and the expected level of difficulty, to adapt the gameplay to their skill level. On the other hand, it is still possible to replace a somewhat time-consuming process in game development by automating the processes typically addressed by a Human that is easily performed by AI. This method is, for the reasons mentioned above, very often approached by independent studios that often do not contain the human, technological and sufficient funds to invest in manpower for these areas of video game design. Thus, with this method it is possible to use algorithms that are easily adaptable to various situations depending on the circumstances presented during content generation [1, 2].

Based on some of best practices identified in the community [3], this paper presents the development and playtesting of a third-person adventure game, using a procedural

content generation system using the Unity engine. With its debut in the early 1980s, in Richard Garriot's Akalabeth and River Raid [4], these techniques have evolved greatly, and can currently be seen in AAA games such as Left4Dead [5], Minecraft [6], Skyrim [7], Borderlands [8], Payday 2 [9], No Man's Sky [10], Don't Starve [11], and in several infinite runners, popular on tablets and cell phones. In addition to map generation, as in this game, PCG techniques can be used in the generation of narratives and visual assets, such as some of the missions in Skyrim, some of the weapons and rewards in Borderlands 2, or creating randomness in scenarios as in Payday 2.

In addition to the current introduction, this paper is structured in three additional sections, dealing with the implementation of the model, and the playtesting of the game, with several examples ranging from screenshots to examples of the generated visuals, and a conclusion, which compiles the concepts used as well as the results obtained.

2 Implementation

The term "open-world" has a specific connotation attached to it: composed of vast, free-to-explore open environments, where spatial orientation and navigation are key to the interpretation of gameplay goals. As such, there's an emphasis on "a large number of different verbal and nonverbal clues, which together form the guidance systems" of the game to induce the player's awareness to use a specific path toward a goal [12]. Furthering a sense of immersion, the sense of gameplay progress and even "in the plot of a digital game is equated with progress in spatial environments" [13]. So "features such as high points for a better overview, spectacular landscapes underscoring dramatic storylines, prominent locations serving as obvious landmarks, regularly distributed settlements justifying the next resources to procure or missions to execute, and labyrinth-like forests or towering high-rises, (...) [13] in a situation where the environment containing e.g. train lines, with various entrances and exits, allows for variations that make the world less flat, empty or repetitive. The landscape stages multiple calls to action.

In this context, the development of the procedural content generation system, a 3rd Person Open-World Adventure Game, was made using the Unity game engine (**Error! Reference source not found.**) [14].

Table 1. Executive summary

Genre:	3rd Person Open-World Adventure Game
Category:	Single-player
Plataform:	PC
Average duration:	10 hours
Target audience:	+12 Pegi [a] – Casual
Reference list:	*The Legend of Zelda: Breath of the Wild*, *No Man's Sky*, *Astroneer*, *Journey*, *Hob*, *Valheim*, *Genshin Impact*, *Dark Souls*.
Key features:	A large, procedurally generated game map; A biome with specific visual characteristics to it.
Mechanics:	The player will explore a world where he can find various regions, enemies and obstacles to overcome. The player will be able to find various items (e.g. swords, shields, magic potions) scattered around the world to help him on this journey. The player has several combat mechanics such as roll, dodge, heavy attacks, light attacks, jump, run, counterattack and even block attacks.
Technology:	The game will be made with the aid of the Unity game engine. For the assets: Photoshop, Blender [15] and others.

[a] Pegi 12 – "Video games that show violence of a slightly more graphic nature towards fantasy characters or non-realistic violence towards human-like characters would fall in this age category. Sexual innuendo or sexual posturing can be present, while any bad language in this category must be mild" [16]

The PCG system uses the built-in Terrain tool to create a terrain object that will be automatically shaped by the main algorithm. For this, the system transcribes the pixel values of a texture from a Perlin noise map [17], created by Unity, to concrete values of elevation in a heightmap. Perlin noise is a pseudo-random pattern of fluctuating values normally generated in a 2D plane, but which can be generalized to three or more dimensions. Perlin noise is characterized by not generating a completely random value at each point. Instead, it generates noise in the form of "waves" whose values gradually increase and decrease along the pattern. This type of noise is used in various situations, from base to texture effects, but also in animation, generating terrain height maps and many other things. Unity has a PerlinNoise function that allows Generate 2D Perlin noise [18]. A new noise map is also created later for positioning objects and buildings on the terrain. The system then uses this noise map and a raycast projected to the ground in order to detect the terrain and, consequently, the correct locations to instantiate these objects according to the defined rules (e.g. noise map, height, building zones).

Taking into account this relevance of exploring, cartographing and traversing an expansive world implied in the open-world genre in combination with adventure, where the player must carry out different missions of a strategic type to evolve and succeed, the enunciation of narrative through environmental storytelling, generating emergent narratives among players, rather than just embedding plot-driven events, further evidences the importance of landscape variance that can be proposed through different biomes. A biome is a large area characterized by its vegetation, soil, climate, and wildlife. There are five major types of biomes: aquatic, grassland, forest, desert, and tundra [19].

To generate the biomes, first the terrain is divided into tiles and then the various biome seeds are released, which will germinate and create the biomes along these tiles.

For the expansion of each of the biomes, a value of probability of spawning the tiles of this biome is assigned to each tile, which degrades according to the distance from the starting point and according to a value of "Decay". In this way, it is less and less likely to expand to neighboring tiles as the distance increases, until reaching a point where it is no longer possible to expand. Thus, it is possible to create an environment with several more natural biomes and with smoother transitions between them that provide a much more realistic environment to the game world (Fig 1).

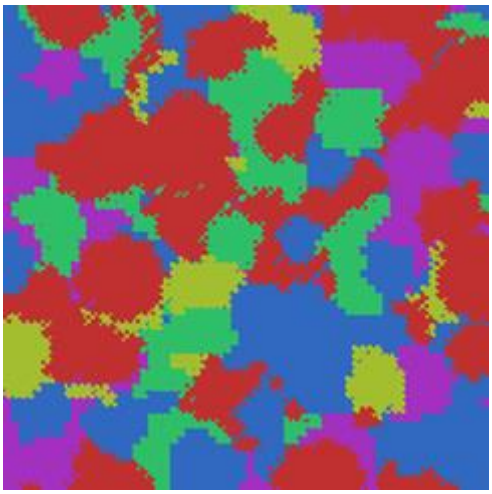

Fig. 1. Generated biome map

Finally, a slope map is generated (Fig 2) that will be used by the pathfinding AI, to generate the navmesh, so that it is not possible for the player to navigate on very steep slopes.

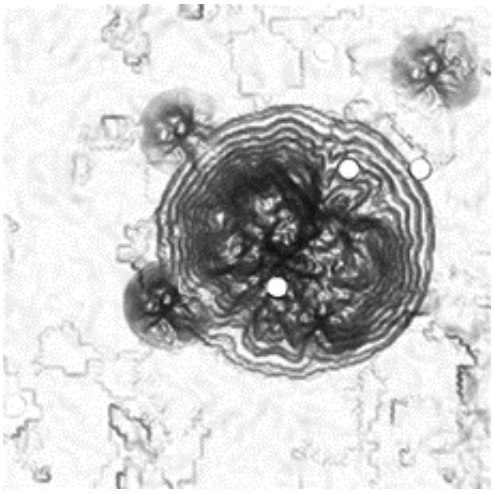

Fig. 2. Slope map generated by the PCG system

All rules defined by the level designer and programmer are passed to the algorithm through Scriptable Objects. (Fig 3) shows an example of a bush that is spread over the ground. In this case, the intensity varies from 50 to 100% of the possible spreading level in some regions of the map.

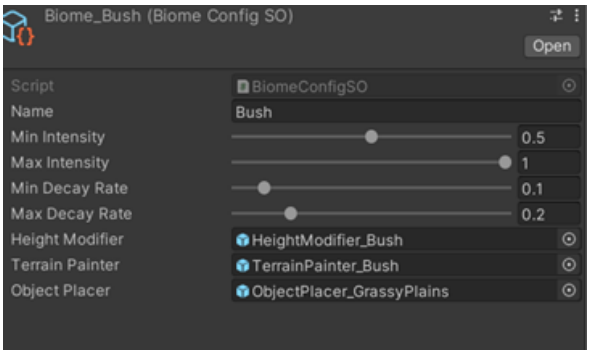

Fig. 3. Setting a specific biome

These can be configured for each biome and for the terrain of the world in general, being able to define rules such as the “weight” of each biome, the value of “decay” of the respective biome, the number of biomes, among others (Fig 4).

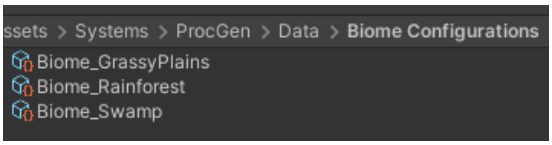

Fig. 4. Different game biomes

The rules to apply to the terrain relief of each biome, the terrain in general, the positioning of assets, buildings, and textures, are passed to the algorithm through prefabs. Fig 5 shows some examples of HeightModifiers prefabs used to create a specific scenario, in this case, bushes, grassy plaints, rain forest, scrub, and swamp. These are HeightModifiers run in a specific order to apply your changes to the terrain. The HeightModifiers_Initial, as the name implies, is responsible for applying initial changes to the terrain in general. For example, when a specific biome has a higher starting height value than normal. We must apply this value in the HeightModifier_Initial so that this modification is applied first, and only then the others are applied. As for the HeightModifiers_PostProcessing, it is responsible for applying minimal changes on top of other changes already made. For example, all terrains have a certain irregularity, this irregularity is only applied after all other changes to make the terrain slightly more natural. The same can be done if we want to apply a "smooth" effect in some situations. The post-processing will take these values to give the "finishing touches" to the terrain making slight changes in this sense.

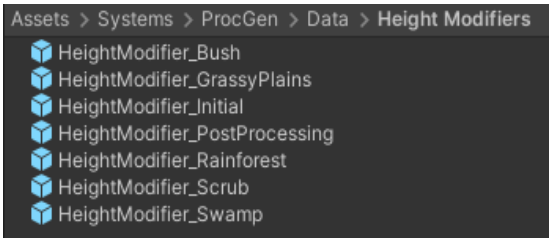

Fig. 5. Height modifier prefabs

Examples of some rules: height range that certain objects can be spawned, the maximum number of objects, whether a respective object can be spawned underwater or just above water, height range of biomes, from which height certain textures appear and disappear, among others. These prefabs are then applied as modifiers to scriptable objects. Then all these rules are passed to the main generation algorithm to shape the terrain and spawn the objects in the right place. In Figure 6 we have a similar interface to Figure 3, however, instead of using the bush biome it shows the swamp biome.

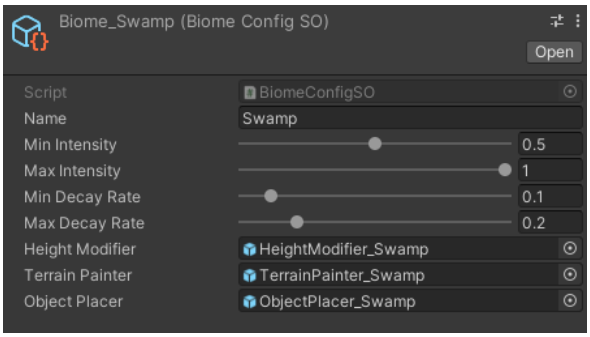

Fig. 6. General PCG configuration

This content generation process allows the Level Designer and the Programmer finer control in the creation of the worlds according to the design rules defined for the game. Thus, all these rules are passed to the procedural generator through these configurations so that it can create a more coherent world in relation to the ideas of our game and its universe, being possible to maintain good control over the procedural generation and effectively avoiding, let this one be quite random and chaotic.

Fig. 7 shows an example of a world generated by the PCG system in edition mode and runtime.

Fig. 7. Terrain with different biomes and assets

Fig. 8 shows an example in-game generated terrain by the PCG system in runtime with the player character ready to explore the world.

The PCG success is mainly assessed in the playtesting phase, providing game designers and developers important information regarding game options.

Fig. 8. In-game generated terrain

3 Playtesting

Playtesting has been increasingly taken up as a standard step of the game design and development cycle. It is a commonly used approach in Games User Research (GUR) to improve both game usability and game user experience, essentially by gaining relevant player's feedback along the design and development cycle to inform the revision of the prototype being worked on. From the perspective of GUR, the aim is to provide game designers and developers new insights through a formative evaluation with potential users that allows the development team to better achieve their design intentions, checking whether the game is meeting their goals, and discerning on how the game design can be improved [20] [21].

Two types of playtesting were performed. The first playtesting took place internally, in which the game was carried out by the members of the game's creation group. This playtest aimed to see if the game was functional and presentable before moving on to the second playtest. The second playtesting was a blind playtesting type, in which players were invited to play independently and without assistance from the creative team. In the end, they would have to answer a survey that evaluated their experience in the game.

This second playtest session took place on the 8th and 14th of June of 2022 in an online and autonomous way and was attended by 12 players, students, and teachers from a game design course and others. The playtesting feedback form was divided into several sections that asked the playtesters about: the game's tutorial, the game mechanics, the game's art and visual effects (VFX), the game's user interface (UI), and procedurally generated scenarios. The form has two types of questions: multiple-choice questions and open questions/feedback. Fig. 9 shows the results of the playtest session, about the procedurally generated worlds to the multiple-choice questions.

The free feedback sentences/questions about the generation of worlds were as follows:

- Write a feedback about the scenario and give suggestions on changes or things we should add.
- If you could change the map, how would you do it?

From the answers, it was evident that the algorithm at the operational level worked without problems. However, tended to create scenarios that were too dense or too empty, and the relief exaggeratedly uneven. There was no coherence between objects, textures and colors, or styles. Some objects such as bushes and vegetation appeared in a realistic style and trees and other objects in the scenery in a cartoon style. In this way, it was difficult to provide the aesthetics of the game which did not help in the player's emergence. It was not clear the way forward or the objective and was necessary to review collisions and the mechanics of interaction with chests. In this way and following the suggestions obtained regarding the map, greater coherence between styles, textures, and colors was created. The relief was revised, and a more balanced scenario was created in which the character and the player emerged. The light and position of the camera were also revised to guide the player on his journey.

These answers were helpful in fine-tuning the PCG system rules for world generation. The full results of the playtest form can be consulted in [22].

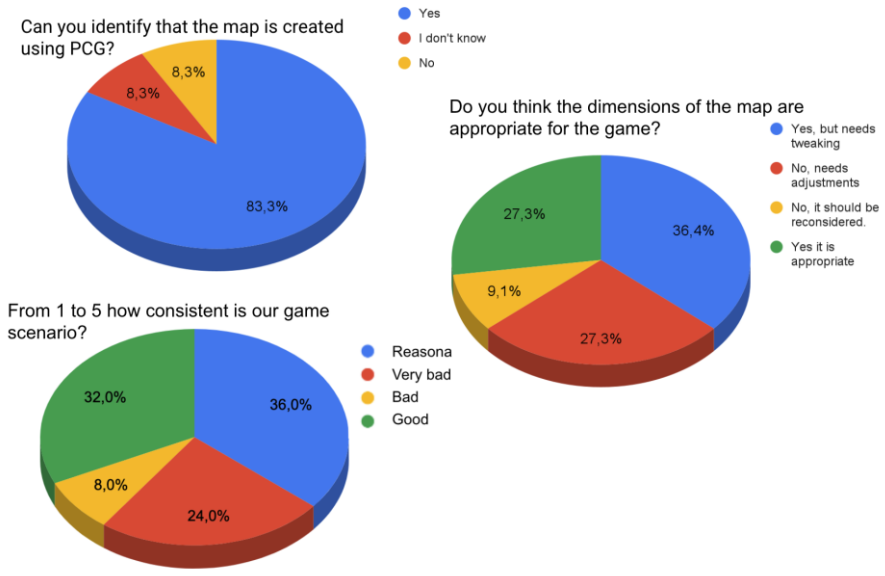

Fig. 1. Playtest session results

4 Conclusions

Procedural content generation can significantly reduce costs and development time of games. However, the systems are often not flexible enough to the requirements defined

by the level designer. This paper showed how we approached the design and implementation of a ruled biome-based procedural content generation system, with attention to the requirements of the level design. The proposed system was designed to be used in a 3rd person open-world adventure game and was developed using the Unity game engine.

During the development of the game, a playtesting session was held, from which it was possible to obtain positive feedback about the game, the procedural generation of the content, and the gameplay that it provides to the players. The responses among the various participants were quite consistent, pointing especially to the lack of coherence in some regions of the map. This feedback allowed us to adjust certain parameters of the algorithm with the aim of improving mainly the gameplay, the game experience, and its difficulty balance as the player progresses in the game. A point of improvement for the playtest could be a session with the development team in which it is possible to observe the players' reactions, which is not always possible to observe through the surveys.

In future work, we intend to carry out a new playtesting session to evaluate the changes made after the first playtesting. We also intend to apply L-systems [21] [25] to the modeling of plants, and a terrain painter based on a Fuzzy Logic controller that can be influenced by parameters [24] [25] or self-overlapping stencil [28].

Acknowledgment

The authors are grateful to the Foundation for Science and Technology (FCT, Portugal) for financial support through national funds FCT/MCTES (PIDDAC) to CeDRI (UIDB/05757/2020 and UIDP/05757/2020) and SusTEC (LA/P/0007/2021).

References

1. Koltai, B.G., Husted, J.E., Vangsted, R., Mikkelsen, T.N., Kraus, M.: Procedurally generated self overlapping mazes in virtual reality. Lect. Notes Inst. Comput. Sci. Soc.-Inform. Telecommun. Eng. LNICST. 328 LNICST, 229–243 (2020). https://doi.org/10.1007/978-3-030-53294-9_16.
2. Wulff-Jensen, A., Rant, N.N., Møller, T.N., Billeskov, J.A.: Deep convolutional generative adversarial network for procedural 3D landscape generation based on DEM. Lect. Notes Inst. Comput. Sci. Soc.-Inform. Telecommun. Eng. LNICST. 229, 85–94 (2018). https://doi.org/10.1007/978-3-319-76908-0_9.
3. Oliveira, G., Almeida, L., Sousa, J.P., Barroso, B., Barbedo, I.: A rule based procedural content generation system. In: Proceeding of OL2A: International Conference on Optimization, Learning Algorithms and Applications 2022. Springer International Publishing, Póvoa do Varzim (2022).
4. Melotti, A.S.: Geração de Conteúdo Procedural por Busca Inovativa em Nichos. (2016).
5. Thibodeaux, C.R.: Dynamically Adaptive Procedural Generation of Dungeons. 14 (2014).

6. Salge, C., Green, M.C., Canaan, R., Skwarski, F., Fritsch, R., Brightmoore, A., Ye, S., Cao, C., Togelius, J.: The AI Settlement Generation Challenge in Minecraft. KI - Künstl. Intell. 34, 19–31 (2020). https://doi.org/10.1007/s13218-020-00635-0.
7. Pereira de Araujo, R., Souto, V.T.: Game Worlds and Creativity: The Challenges of Procedural Content Generation. In: Marcus, A. and Wang, W. (eds.) Design, User Experience, and Usability: Designing Pleasurable Experiences. pp. 443–455. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-58637-3_35.
8. Pace, A., Thompson, T.: Procedural Content Generation and Evolution within the EvoTanks Domain. 2.
9. Plante, C.: Payday 2 review: the getaway, https://www.polygon.com/2013/8/16/4628602/payday-2-review, last accessed 2022/10/01.
10. Catherine, F., Meghan, D.L., Andrew, R.: Exploring simulated game worlds. ORBIT J. 1, 1–13 (2017). https://doi.org/10.29297/orbit.v1i2.46.
11. Alves, T., Coelho, J.: A Framework for Massively Multiplayer Online Game Content Generation. In: 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA). pp. 834–841 (2016). https://doi.org/10.1109/AINA.2016.22.
12. Suter, B., Kocher, M., Bauer, R. eds: Games and Rules. Game Mechanics for the "Magic Circle." Transcript, pp. 169-189., Bielefeld (2018).
13. Suter, B., Bauer, R., Kocher, M.: Narrative Mechanics: Strategies and Meanings in Games and Real Life. transcript Verlag. pp. 161-176 (2021). https://doi.org/10.14361/9783839453452.
14. Unity: Unity Real-Time Development Platform | 3D, 2D VR & AR Engine, https://unity.com/, last accessed 2022/07/11.
15. Blender: blender.org - Home of the Blender project - Free and Open 3D Creation Software, https://www.blender.org/, last accessed 2022/07/11.
16. What do the labels mean?, https://pegi.info/what-do-the-labels-mean, last accessed 2022/09/13.
17. Perlin, K.: An image synthesizer. ACM SIGGRAPH Comput. Graph. 19, 287–296 (1985). https://doi.org/10.1145/325165.325247.
18. Technologies, U.: Unity - Scripting API: Mathf.PerlinNoise, https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html, last accessed 2022/10/30.
19. National Geographic: The Five Major Types of Biomes, https://education.nationalgeographic.org/resource/five-major-types-biomes, last accessed 2022/10/01.
20. Choi, J.O., Forlizzi, J., Christel, M., Moeller, R., Bates, M., Hammer, J.: Playtesting with a Purpose. In: Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play. pp. 254–265. ACM, Austin Texas USA (2016). https://doi.org/10.1145/2967934.2968103.
21. Fullerton, T.: Game design workshop: a playcentric approach to creating innovative games. CRC press (2014).
22. Project E - Playtest session results, https://docs.google.com/spreadsheets/d/14VxBotKdlwKcJWEddEIKepSsdXWrNB-

kDRISS67HELQ/edit?resourcekey&usp=forms_web_b&usp=embed_facebook, last accessed 2022/07/13.
23. Prusinkiewicz, P.: Graphical applications of L-systems. In: Proceedings on Graphics Interface ’86/Vision Interface ’86. pp. 247–253. Canadian Information Processing Society, CAN (1986).
24. Prusinkiewiczy, P., Hammely, M., Hananz, J.: L-SYSTEMS: FROM THE THEORY TO VISUAL MODELS OF PLANTS. 32 (2002).
25. Kim, L.C., Talib, A.Z.: Improving L-system music rendering using a hybrid of stochastic and context-sensitive grammars in a visual language framework. Lect. Notes Inst. Comput. Sci. Soc.-Inform. Telecommun. Eng. 101 LNICST, 46–53 (2012). https://doi.org/10.1007/978-3-642-33329-3_6.
26. Beebe-Wang, J., Tang, J.: Injection Painting Optimization with Fuzzy Logic Expert System. 3 (2001).
27. Mohamad, W.M.W., Ahmad, T., Ashaari, A.: Intelligent fuzzy logic controller for painting system of vessel. AIP Conf. Proc. 2423, 020029 (2021). https://doi.org/10.1063/5.0075684.
28. Neerdal, J.A.I.H., Hansen, T.B., Hansen, N.B., Bonita, K.L.F., Kraus, M.: Navigating procedurally generated overt self-overlapping environments in VR. Lect. Notes Inst. Comput. Sci. Soc.-Inform. Telecommun. Eng. LNICST. 328 LNICST, 244–260 (2020). https://doi.org/10.1007/978-3-030-53294-9_17.