Teaching Reinforcement Learning Procedural Content Generation (RL-PCG) via Educational Game

1 INTRODUCTION

At the time of this writing, there is little (if any) methodology for teaching game developers Reinforcement Learning Procedural Content Generation (RL-PCG). Prior to the development of modern game engine A.I. technology, such as Unity's ML-Agents toolkit, the barriers of implementation prevented the majority of game developers from pursuing this track of content generation. With the introduction of Tensorflow and Unity's ML-Agents toolkit however, a whole new category of game developers have access to Machine Learning techniques.

In the game industry, where many games rely on traditional non-AI PCG, this process is typically performed by hand-crafted heuristic algorithms (Shaker, Togelius, and Nelson 2016). An example of RL-PCG on the other hand was demonstrated by Khalifa et al. via level designing agents; they demonstrated that you can frame the level design problem as a sequential task and utilize a Markov Decision Chain to effectively generate levels (Khalifa, A., Bontrager, P., Earle, S., & Togelius, J., 2020). Why then, are more developers not taking advantage of this research? According to Khalifa et al., to the best of their knowledge, this is the first time that reinforcement learning has been brought to bear on this problem (Khalifa, A., Bontrager, P., Earle, S., & Togelius, J., 2020). With this in mind, a method of teaching this strategy to game developers and bridging the gap between research and implementation could benefit the overall field of AI research and game development.

Finally, reinforcement learning itself has been adopted by major game engines, one of which is called Unity3D. Unity3D is a game engine that's been a part of thousands of games and is extremely open as a platform; to use the engine personally is free of cost. Major game developers also utilize the platform, one example being Blizzard Entertainment's Hearthstone ™ game. On September 17, 2017 the first version of Unity3D's ML-Agents toolkit was released, an effort that has since dramatically reduced the average developers barriers to utilization; as of May 5th, 2020 the toolkit has reached maturity and version 1.0 (Mattar M., Shih J., Berges V., Elion C., Goy C., 2020). This toolkit (and others like it) have provided game developers the world over with the tools to implement machine learning.

This has resulted in a need for teaching RL-PCG that is tailored to game developers. Considering Unity's popularity as an entry level game engine, it's support for several machine learning techniques, and it's wide-spread educational material, it was chosen as the infrastructure for which a game could be built that would teach game developers via the very medium they are using for construction. An educational game is uniquely tailored to the needs of a game developer, because in this case the game developer can 'graduate' from the content of

the game to analyzing the game's source code and construction to learn a real implementation of RL-PCG. The approach to designing a game as an introduction to RL-PCG was inspired by results that showed increased retention, enthusiasm for the material, and participation that did not require sacrifice of technical depth (Leutenegger, S., & Edgington, J., 2007).

2 RELATED WORK

The majority of related RL-PCG work originates in whitepapers. Section 2.1 describes the work of Khalifa et al., which was directly referenced in the construction of this project. Section 2.2 describes a published educational game that teaches machine learning principles by using a series of 'jobs' for the gamer to complete, where the solutions built involve the use of a visual coding graphical user interface. Finally, section 2.3 describes an educational game that teaches machine learning that has yet to be published and uses a system closer to games like Factorio. The games in sections 2.2-2.3 are used to demonstrate the marketability and success of games that teach machine learning and coding respectively, and provide a base of experience to draw from (player feedback, developer publications, expert reviews) on the topic. In section 2.4, the game Dwarf Fortress is highlighted; Dwarf Fortress is an almost universally known game that demonstrates PCG to include world generation, similar to what this project is trying to accomplish.

2.1 Procedural Content Generation via Reinforcement Learning

In this whitepaper, Khalifa et al. describes how reinforcement learning can be used to train level-designing agents; level design itself is framed as a game, and the content generator is learned (Khalifa et al., 2020). The following list of key details from Khalifa et al. are specifically relevant for this project proposal:

1. Conceptually, the strategy outlined in this whitepaper explores content space instead of generator space; the whitepaper suggests that this strategy is more scalable than traditional PCG due to its automatic learning.
2. Random layouts are generated first, before tasking the RL Agents to start making changes.
3. The number of tiles (ranging from 0% to 100%) an RL Agent can modify during inference is varied for experimentation.
4. Different styles are used for the RL Agents: Narrow, Turtle, Wide.
5. According to the whitepaper, "If one has a PCG problem definition and reward function, one can readily adopt our narrow, turtle, or wide agent environment interfaces, and run the PCGRL algorithm on any content generation problem."

6. According to the whitepaper, the strategy used both in the paper and being proposed in this project fit very well in a mixed initiative, turn-taking paradigm; in other words, multiple agents can be trained to work together or with a human designer.
7. According to the whitepaper, PCGRL offers the possibility of moving most of the time consumption from inference to training, or, in game development terms, from runtime to development; this might make PCGRL a viable choice for runtime content generation in games.

2.2 while True: learn()

> "You're a machine learning specialist who makes neural networks but your cat seems to be better at it. Now you must solve puzzles to build a cat-to-human translation system (who knows what else this cat is capable of!). Earn a fortune, buy kickass cat outfits and learn how machine learning really works!" (While True: Learn() on Steam, 2019)

The first educational game that also teaches machine learning concepts is called while True: learn(). In order to demonstrate that a game of this genre has a market, the following statistics were pulled from both Steam and a site called Steamspy (While True: Learn() on Steam, 2019):

1. Rated Very Positive, the highest positive rating on Steam
2. 3,922 reviews
3. 200,000 .. 500,000 owners (estimated)

Other educational games from this same publisher can be found on luden.io.

2.3 Human Resource Machine

> "Human Resource Machine is a puzzle game for nerds. In each level, your boss gives you a job. Automate it by programming your little office worker. If you succeed, you'll be promoted up to the next level for another year of work in the vast office building. Congratulations! Don't worry if you've never programmed before - programming is just puzzle solving. If you strip away all the 1's and 0's and scary squiggly brackets, programming is actually really simple, logical, beautiful, and something that anyone can understand and have fun with! Are you already an expert? There will be extra challenges for you." (Human Resource Machine on Steam, 2015)

Human Resource Machine exemplifies the idea of taking a subject like programming, abstracting it, and making it palatable to an audience that is interested in the puzzle of programming with lowered barriers to entry. Additionally, it manages to still be challenging to programming experts. In order to demonstrate that a game of this genre has a market, the following statistics were pulled from both Steam and a site called Steamspy (Human Resource Machine on Steam, 2015):

1. Rated Very Positive, the highest positive rating on Steam
2. 1,629
3. 500,000 .. 1,000,000 owners (estimated)

2.4 Dwarf Fortress

Dwarf Fortress is a game built on simulation, generating unique game worlds for the player to experience through PCG (Adams, T., 2015). The lessons learned from this game will be extremely useful as a framework to support this project effort; according to Adams, four guiding principles kept the game simulation robust and easy to work with (Adams, T., 2015). But before those principles are listed, it's first important to describe why one should care about Dwarf Fortress in the context of PCG; Adams goes into detail in a 2019 interview (Harris, J., 2019):

1. Dwarf Fortress procedurally generated the entire world from scratch (in other words, it entirely relies on PCG).
2. Dwarf Fortress has been in development for over 17 years and has an enormous playerbase (an exact count is difficult, but).
3. Dwarf Fortress uses a grid-based tilemap to depict its world, a simple, efficient and effective means of representation.

3 THE SOLUTION

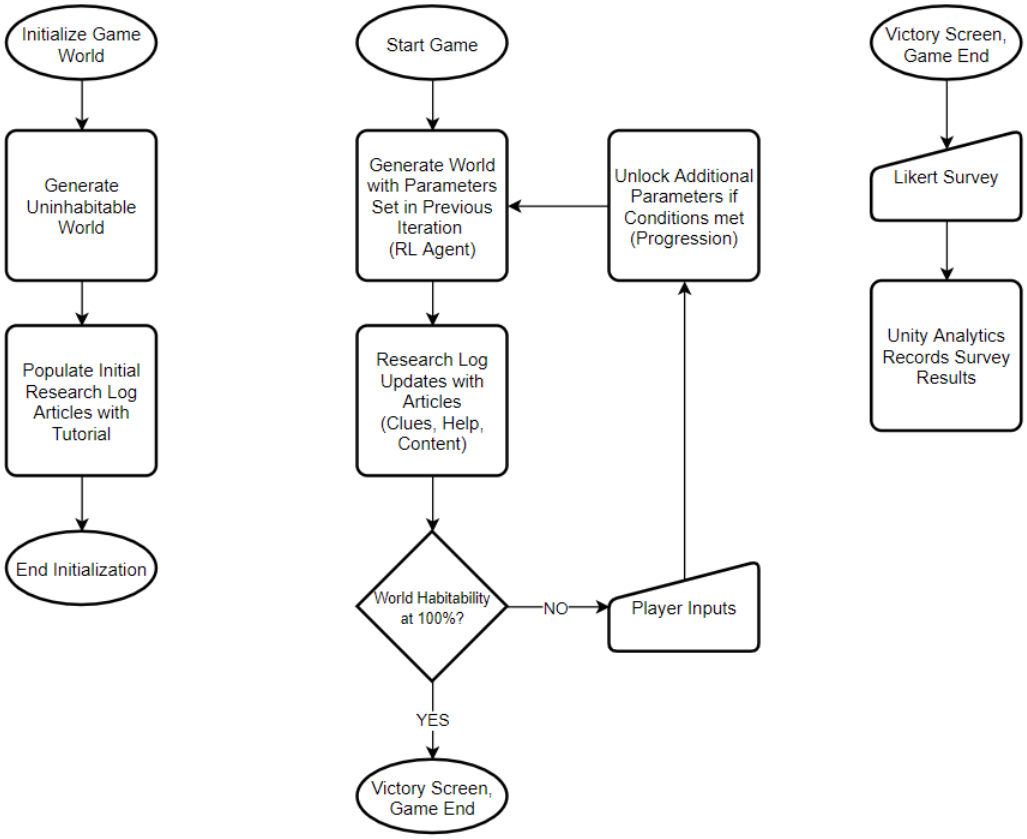


Figure 1—Core game loop, including game initialization and the collection of survey results once the game ends. Originally the research log was included as part of the design, however it was replaced with an introduction section that takes place before the game starts to reduce the visual clutter.

To teach RL-PCG, I implemented three major components: a short educational game that utilizes the Unity ML-Agents toolkit, a library of links to other RL-PCG content to lower the barriers to further learning, and the source code of the game itself which can serve as a learning tool for game developers. The game is divided into two sections, the first of which is a series of educational slides that seeks to introduce the basic components of Reinforcement Learning. The second section is the actual implementation of RL-PCG in the form of two neural networks that work as a team to generate the world; the game loop is shown in Figure 1 above. The reasoning behind this was to introduce as many strategies as possible, with the hope that one of them would inspire participants to continue with RL-PCG learning overall. Finally, none of this content was created using tools that incur a cost in order to eliminate yet another barrier to entry. The models for the game were all designed in Blender, a free open source 3D Modeling software that is compatible with Unity. Unity itself provides a free personal edition of the game engine and the packages used in the development of this project were also freely available.

3.1 The Introduction / Research Log

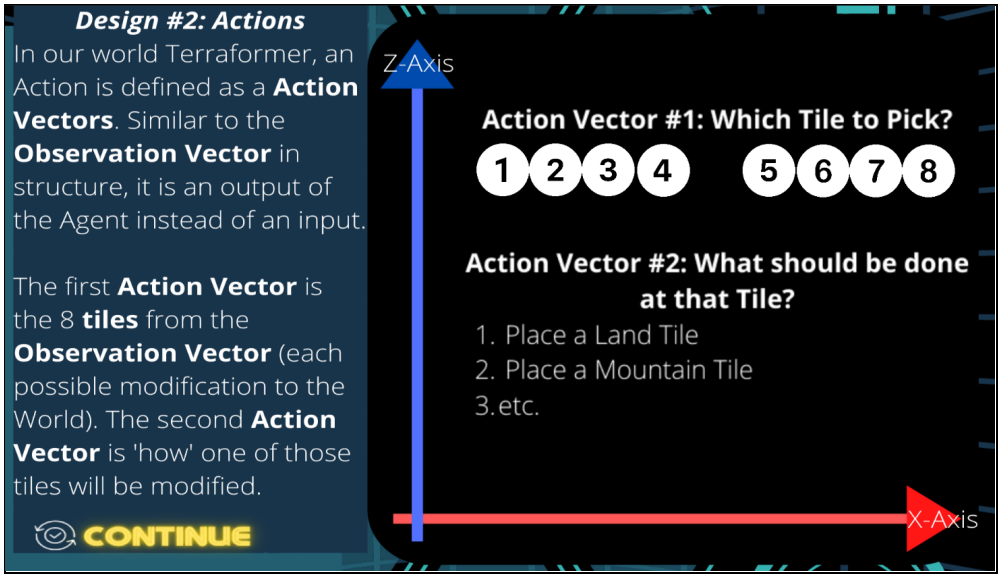


Figure 2—The introduction provides players with a quick review of the topic of RL-PCG as it is implemented in the game. It includes many slides, some of which are careful to specify that this is only one of many implementations.

The introduction was implemented to support novice users in overcoming the first few barriers to understanding, such as defining Reinforcement Learning and Procedural Content Generation. From the main menu, more advanced users can skip this section and go directly to the Game. The introduction includes basic RL-PCG concepts, such as the observation vector, the action vector, and the reward function and an example is given above in Figure 2.

3.2 The Game

The Game

Figure 3—Going clockwise starting from the top left corner, Controls shows the player how they can manipulate the neural networks. Current Observation is a graphical element that visually communicates the Continent Generators observation vector. The Habitability graphic increases from red to yellow to green as the player improves the world. The reset buttons 'reset' the world to allow a fresh start. The two Generate buttons allow the player to switch between Continent & Biome generation. Take Survey leads the player to a survey of the game. Quit exits the application.

The second section, world generation, was implemented primarily to instill enthusiasm in the technology and to demonstrate RL-PCG in action in a way that users could interact with. The context of the game involves the player taking on the role of a scientist using an Artificially Intelligent machine to terraform a new world for humanity. The concept of terraforming was utilized to assist players in making the connections between RL-PCG and the generation of content by relying on their previous experience with the term terraforming, and is a major theme in the game.

In Figure 4 and Figure 5 below, the two neural network approaches are shown during execution. The biggest differences between them are the structure of the Observation Vector, and the Reward Function. Both generators 'move' across the X by Z grid of tiles one tile at a time, where the current position is considered the 'Agent Cursor'. The Continent Generator takes the eight tiles surrounding the Agent Cursor as an observation; this design was introduced to further generalize the network by making it compatible with any X by Z grid size and standardizing the input. The Biome Generator on the other hand, uses temperature and

humidity as observations to decide what biome to change a tile to. Temperature is a calculation of distance from the center (equator) of the grid. Humidity is a count of the number of ocean tiles adjacent to the Agent Cursor. This design also makes this neural network compatible with any X by Z grid size. The player can see the actual observation vector in the 'Current Observation' GUI element, with the Agent Cursor at the center of the right hand side and the number of tiles placed on the left hand side.

3.2.1 *The Continent Generator Neural Network*

The Continent Generator Neural Network

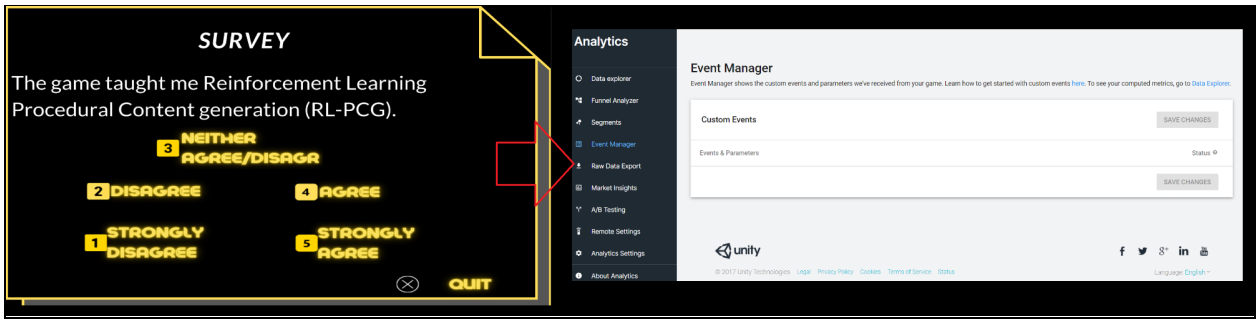
Figure 4—The first neural network the player uses in the game, to 'terraform' continents on the world grid. This will serve as the foundation for the second neural network, the Biome Generator. It includes an observation component, an action component, and a reward function.

3.2.2 *The Biome Generator Neural Network*

The Biome Generator Neural Network

Figure 5—The second neural network the player uses in the game, to 'terraform' continents on the world grid. It utilizes a different strategy from the first, using temperature (north/south poles are colder, equator is hotter). In this image, it is shown part-way through the terraforming process; the black tiles to the right are barren and contribute no habitability until changed by this process.

4 METHODOLOGY



Integrated Feedback Mechanism

Figure 6—When a user clicks a response on the survey, it pings the Unity Analytics server with the response (if the user has internet access). Because the survey is embedded in the game, it eliminates the barrier of having to leave the game and visit a site to complete the survey and makes it easy to prompt the user for feedback. The

survey itself is delivered to the user upon completion of the game, but the user can initiate a survey at any time.

At this moment, the method of obtained results has been created but not yet fully implemented; feedback was obtained on designs, but a key next step is finding more players for the game. Unity3D as a platform provides extensive tools to evaluate the usage of a project. Everything from how many daily users, monthly users, geography, and how long users are playing the game to name a few. In addition, Unity3D's Analytics can be customized towards specific goals, such as keeping an eye on whether players are unable to access certain areas of the game and discovering where players are getting stuck/bored. In addition, it can be utilized to implement a survey system within the game itself. The goal is to reduce the barriers (such as exiting the game) that occur when eliciting player feedback. For this project, a 5-point Likert scale that is oriented towards game development was used (Howard, M. G., Collins, H. L., & DiCarlo, S. E. 2002; Rankin, Y., Gooch, A., & Gooch, B. 2008; Doane, N., 2015).

4.1 Likert Survey Questions

1. I was able to navigate the menus in the game.
2. The game was not difficult to play.
3. I did not encounter bugs in the game.
4. The game taught me Reinforcement Learning Procedural Content generation (RL-PCG).
5. The material presented in the game was easy to understand.
6. The game made learning RL-PCG fun.
7. The game motivated me to learn more about reinforcement learning.
8. The game motivated me to learn more about procedural content generation.
9. I would enjoy games that implemented this style of procedural content generation.
10. I want to implement procedural content generation in my own game.

5 THE RESULTS

Early feedback on the concept and design of the game so far has been highly positive. In the course of development recorded videos of the game being played and images taken were shared with several game development communities for feedback, as well as with students of the OMSCS program. Out of 1-5 scale rating for interest, 83% responded with 4 (interested in playing the game) and 17% responded with 5 (very interested in playing the game). The next step is to share the completed game with the community and obtain data from the integrated survey.

Some quotes from students of the OMSCS program that provided feedback during development:

1. “This is the most remarkable project I have ever seen in the course. I bet after the game will have a lot of users after completion! I have to say, you have done a super great job!”
2. “Great job! Very detailed and well explained. Love your project! It could perhaps be slightly shorter, but I believe you needed all the time to explain your thoughts and plan in detail. As a novice in game development, I would use your product myself.”
3. “Wow your progress looks amazing and the project is a really neat idea. I'm not sure how familiar you are with unity but there are declarators you can put in the code to isolate items in the inspector to inform the user what they should or should not mess with. If you are sharing the code base when your project is done I would love to dive into it.”
4. “This is an excellent topic to teach through gaming, as RL itself is a complex subject. I watched your video twice before coming to the comment section, as you have provided a great detail of the work you've already done for this project. I can sense that this game would serve the purpose of fun and learning both. However, I feel like there would be a steep learning curve for people to jump from novice(game players for fun) to people who are there for learning, maybe the gap can be bridged by inserting a few more stages like, Novice -> Intermediate -> Expert. Intermediate players can tweak some of the hyperparameters to see rather fabulous results than being a novice user.”

6 LIMITATIONS

There are several limitations to the approach of using a game to introduce the concept of RL-PCG. First, this game heavily relies on a specific game engine, Unity3D; limitations specific to this tool can be found here. Game developers that are not experienced with this game engine may find less value in it, assume it won’t be useful to them, or will be unfamiliar with the engine's interface when viewing the source code. Second, the technology was novel and learned during development; in other words, a novice user created the game. Having an expert user review the game and provide feedback and improvements would greatly enhance the experience. Third, it was found that the models used in the game consume processing power and so training may be faster if 2D sprites or other low cost representations of world tiles were used. Fourth, due to time constraints the game has not been as rigorously tested for bugs and issues when compared to mainstream games and should be considered more of a prototype to help inspire others to develop similar models of teaching RL-PCG. Finally, the hardware I trained the neural networks on was very limited compared to most professional machine learning workstations, limiting me to just a couple parameters. Increasing the complexity of the world generation would add a lot of value.

7 CONCLUSION

RL-PCG is a method of applying Reinforcement Learning to Procedural Content Generation, a new field that has already demonstrated promising results in game development. Despite this, little educational material exists specifically for this purpose. This project has sought to demonstrate one method of teaching RL-PCG via educational game, both because game developers have the skillset to open game source code and learn from its implementation and because a demonstration of the technology in a game can inspire enthusiasm for the subject material (Leutenegger, S., & Edgington, J., 2007). Preliminary review of the subject material has been met with enthusiasm and interest from game developers interested in the subject, despite the limitations introduced by the novice skill level of the project developer as well as the need for a specific game engine, the complexity of the topic, and the unpolished nature of the prototype. In the end, the goal of this project is to inspire game developers to explore this little-known intersection of game development and machine learning in order to advance both fields.

8 FUTURE WORK

The potential for RL-PCG is nearly limitless, and just depends on how many different creative ways game developers find to apply Reward Functions to Observations and Actions. In this project I produced two entirely different neural networks to help tackle a single problem, which opens up the possibility for developers to use neural teams to tackle highly complex generation tasks. This could take the form of neural networks generating things in sequence, using each other's output as an input (such as generating a continent and then generating biomes from the resulting tile relationships), or running them in parallel to massively increase the speed of generation for little increase in processing requirements. Further testing is required to verify, but one preliminary takeaway of this project was that replacing costly traditional PCG algorithms with well trained neural networks might be a way to dramatically increase performance of various PCG implementations. An example of this is Perlin Noise, which has an $O(n^2)$ complexity rating in general, and multiple iterations of Perlin Noise are often required to achieve desired effects.

The next step (and an ongoing process) is to elicit player feedback on the game itself, via distribution and testing. One of the primary benefits of a prototype like this is in inspiring ideas directly from users on how to teach the subject. Two neural networks were displayed here, and increasing the number of networks and strategies for their training could increase the value of this project. The knowledge of the subject of RL-PCG implementation in games is currently scattered, and building that knowledge into a project that game developers can both play and take apart could save many developers the time and costly mistakes that are so common in neural network development.

9 REFERENCES

1. Khalifa, A., Bontrager, P., Earle, S., & Togelius, J. (2020). Pcgrl: Procedural content generation via reinforcement learning. arXiv preprint arXiv:2001.09212.
2. Shaker, N., Togelius, J., & Nelson, M. J. (2016). Procedural content generation in games. Switzerland: Springer International Publishing.
3. Mattar M., Shih J., Berges V., Elion C., Goy C., (2020, May 12). Announcing ML-Agents Unity Package v1.0!. *Unity Blog*.
4. Technologies, U. (n.d.). Wondering what Unity is? Find out who we are, where we've been and where we're going. Retrieved October 01, 2020, from https://unity.com/our-company
5. While True: Learn() on Steam. (2019, January 17). Retrieved October 03, 2020, from https://store.steampowered.com/app/619150/while_True_learn/
6. Human Resource Machine on Steam. (2015, October 15). Retrieved October 03, 2020, from https://store.steampowered.com/app/375820/Human_Resource_Machine/
7. Adams, T. (2015). Simulation principles from Dwarf Fortress. Game AI Pro, 2, 519-521.
8. Harris, J. (2019, June 3). Q&A: Dissecting the development of Dwarf Fortress with creator Tarn Adams. Retrieved October 03, 2020, from https://www.gamasutra.com/view/news/343859/QA_Dissecting_the_development_of_Dwarf_Fortress_with_creator_Tarn_Adams.php
9. Fiebrink, R. (2019). Machine learning education for artists, musicians, and other creative practitioners. ACM Transactions on Computing Education (TOCE), 19(4), 1-32.
10. Juliani, A., Berges, V. P., Vckay, E., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2018). Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627.
11. Doane, N. (2015, March 16). How to Design a Survey for User Feedback. Retrieved October 04, 2020
12. Rankin, Y., Gooch, A., & Gooch, B. (2008, February). The impact of game design on students' interest in CS. In Proceedings of the 3rd international conference on Game development in computer science education (pp. 31-35).
13. Howard, M. G., Collins, H. L., & DiCarlo, S. E. (2002). "Survivor" torches "Who Wants to Be a Physician?" in the educational games ratings war. Advances in physiology education, 26(1), 30-36.
14. Leutenegger, S., & Edgington, J. (2007, March). A games first approach to teaching introductory programming. In Proceedings of the 38th SIGCSE technical symposium on Computer science education (pp. 115-118).