UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Raigo Kõvask

Procedural generation of skill-based systems

Bachelor Thesis(9 EAP)

Supervisor: Margus Luik

Tartu 2019

Protseduuriliselt genereeritud oskussüsteem

Lühikokkuvõte:

Käesoleva töö eesmärk on kirjeldada protseduuriliste süsteemide ülesehitust. Protseduurilise süsteemide all mõeldakse antud juhul süsteeme, mis kasutavad objektide loomiseks protseduurlisi meetodeid. Töö praktiline osa on luua süsteem, mis genereerib oskusi oskustel põhinevatele mängudele. Süsteemi eesmärk on genereerida ja visualiseerida oskusi, ning antud oskusega mõjutada virtuaalses maailmas olevaid karaktereid.

Võtmesõnad: Protseduuriline genereerimine, mängud, oskus-süsteemid, Unity

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Procedural generation of skill-based systems

Abstract:

The purpose of this work is to describe the structure of procedural systems. Under the term, procedural systems, the author means the system, that uses procedural methods to generate objects. The practical part of the work is to create a system, that generates skills for the games, that use a skill-based approach inside of them. The goal of the system is to generate, visualize and affect characters, that are inside of the virtual world.

Keywords: procedural generation, games, skill-based systems, Unity

CERCS: P170 Computer science, numerical analysis, systems, control

Table of Contents

1 Introduction

Skill-based systems are an inseparable part in fantasy games such as multiplayer online battle arena (MOBA), where the player controls certain hero, who has different skills that he can use to defeat enemies[1]. Most popular of the MOBA type of games are Dota 2 and League Of Legends, where at August 2018 League Of Legends was played by 100 million active players[1] and Dota 2 was played by 0.83 million active players[2]. These type of games were a variant of RTS genre, that became popular at the beginning of the 21st century [1].

Skill-based systems are also used in games, that do not have a certain character as the main character, but rather a set of objects that use skills. For example in the game HeartStone, which also had 23.9 million active players in August 2017[3], where the player has a set of cards, that also has a type of card called spell card. Purpose of this card is same as for the MOBA type spell, where you need to cast a spell against enemy[4]. In the given paper, the objects for this are called characters, because the paper focuses more on the MOBA type of games.

The reason, why the current problem is being researched, is that the author couldn't find any systems, that uses procedural approach for generating the skills. The main research method is to look through the different types of games, that use predefined spells and try to find similarities from those skills to the procedural methods.

The paper consists of four different parts:

- the first chapter gives an overview of used software for the given system;
- the second chapter focuses on defining procedural generation and methods of using it. Under the given chapter, there is also pros and cons of the procedural approach;
- the third chapter describes methodologies, that were used to investigate skills and also what games were investigated:
- the fourth, fifth and sixth chapter defines the character and skills, that are defined so that they can be directly connected to the procedural method and seventh describes the implementation of the system:

1 Rachel Samples, How Many People Play League of Legends, https://www.dbltap.com/posts/6249990-how-many-people-play-league-of-legends (08.05.2019)
2 Dota 2 players charts, https://steamcharts.com/app/570#1y (29.04.2019)
3
4 Heartstone card https://hearthstone.gamepedia.com/Card (29.04.2019)

- the fifth chapter describes the program, that was made.

1.1 Technologies used

1.1.1 Unity

Unity is a program, that is used to create two- and three-dimensional video games and different simulations. It supports building games for 27 different platforms, where exporting the games for different platforms is done the same way for every platform. This saves the time for configuring the system for different platform. It consists of a graphics engine and a full-featured editor. It supports importing art and assets such as different models from other software and assembles those assets into scenes (virtual room, that contains objects of the game[5]). From the development side, Unity supports adding lighting, audio, special effects, physics, animation onto the scene. Unity has interactivity inside of the scene, where the user can transform (transform consists of position, scale, rotation) different objects. and allows developers to write game logic, that can be written in C#. The reason, why this game engine is used for developing is that its free to use and it had all the necessary components for creating a procedural skill system [6]. Following components are necessary for the given system:

- reading input;
- rendering;
- particle system;
- collision detection between the objects;
- objects transformations:

Figure 1. Logo of Unity software. Image is from Unity homepage[7].

5 Unity Scene, https://docs.unity3d.com/560/Documentation/Manual/CreatingScenes.html
6 GameObject, https://docs.unity3d.com/ScriptReference/GameObject.html (04.04.2019)
7 Unity home page https://unity.com/ (04.04.2019)

1.1.2 Gimp

Gimp (GNU Image Manipulation Program) is a cross-platform image editor, which is a freeware program used to make or edit images. Gimp is used since it is free to use and it allows the user to make images, that have an alpha channel (the alpha channel controls the transparency or opacity of the color). This functionality is needed to draw textures for particle system effects[8].

Figure 2. Logo of Gimp software. The image is from Gimp homepage[3].

8 Gimp home page https://www.gimp.org/ (04.04.2019)

2 Procedural Content generation

People have used procedural content generation (PCG) for a different type of uses as in games, video processing, image manipulation. Procedural content generation (PCG) has been around in video games from the 1970s. Methods got more popular, when limited capabilities of home computers back in the 1980s could not run more advanced games, due to reason, that advanced games required more memory than the computers had. The earliest example of a game using procedural methods is a game called *Elite* what was developed in the early 1980s where the game had to generate eight galaxies and each containing 256 planets. Since it used too much memory for computers at the time, developers needed to find other ways such as PCG[2].

At the beginning of the 21st century, the memory limit problem has been pushed into the background, by the development of newer and larger volumes of memory. The PCG is still around in games. The main reason for this is that PCG can also be used to create extra content into games. Example of this is the sandbox construction game Minecraft, where the game world is generated procedurally[9] or No Man's Sky, where planets and the planets and part of the surface is generated using PCG methods[10].

Before going further with PCG methods, we need to define exactly what PCG means. The definition of the method from the book „*Procedural Content Generation in Games: A Textbook and an Overview of Current Research*" is „PCG is the algorithmic creation of game content with limited or indirect user input". This means that the computer is capable of creating content by itself or together with people [2]. There are different ways how to achieve PCG in games. In 2017 at Game Developers Conference the Tracery developer Kate Compton gave a reasonable way of defining PCG methods[11]. According to Kate Compton, there are two different methods, that describe PCG. Those two are additive and subtractive methods. Additive methods are used to generate data endlessly. Subtractive methods are used to find data from the pool of generated objects the ones that suit for developers needs. Next methods are the ones, that are used in the procedural generation of skill-based systems:

Additive methods:

9 Minecraft procedural methods http://pcg.wikidot.com/pcg-games:minecraft (02.04)
10 A Look At How No Man's Sky's Procedural Generation Works https://kotaku.com/a-look-at-how-no-mans-skys-procedural-generation-works-1787928446 (02.04)
11 Practical Procedural Generation for Everyone https://www.gdcvault.com/play/1024213/Practical-Procedural-Generation-for (02.04)

- Parametric is a system, that makes up of a number of different parameters. There is an array of floating-point numbers representing different settings of a system.
- An interpretive way where the program starts with input and runs an algorithm to process data into some other data.
  - Noise (Perlin/simplex)

Subtractive methods:

- Saving seeds that generates the same skills again.
- Generate and test
  - Throwaway or ranking (Finding the best seeds from the set)
  - brokenness and connectivity[12]

2.1 Pros and Cons of Procedural Systems

Procedural systems are used in games, where the game needs a large number of objects, that are slightly different. For example games such as Galcon, where all the maps are generated using procedural generation[13]. But even if the method helps developers to generate a large amount of content it also has cons. In order to understand what are the pros and cons of different procedural systems, different discussions about the pros and cons were looked through. Comparison is made between different articles and it is divided into three categories (Pros, Cons, and points that vary between good and bad) [14] [15] [3]:

- Pros
  - Monotony free – if an artist needs to make a number of different skills (e.g, 1000), it takes time and money. Since it is repetitive work, then it is cheaper and faster to create a generator, that does work by itself.
  - Scalability – The procedural system helps the developer create a large number of different objects that vary by some amount. As the computer doesn't need much time to generate a different set of numbers, it takes only seconds to generate a new certain type of object with randomized parameters. Whereby doing it manually could take minutes or hours, depending on the size of the object.

12 Practical Procedural Generation for Everyone https://www.gdcvault.com/play/1024213/Practical-Procedural-Generation-for (02.04)

13 Ra´ul Lara-Cabrera, Carlos Cotta and Antonio J. Fern´andez-Leiva, Procedural Map Generation for RTS Game, http://www.lcc.uma.es/~ccottap/papers/lara12procedural.pdf (08.05.2019)

14 Procedural Generation: Pros and Cons https://www.gamedev.net/articles/programming/graphics/procedural-generation-pros-and-cons-r4362/ (02.04)

15 The pros and cons of procedural generation https://www.oreilly.com/library/view/procedural-content-generation/9781785886713/ch11s02.html (02.04)

  - Compression – PCG can reduce the amount of required memory by generating objects during runtime. An example from the article „Procedural Generation: Pros and Cons” at forum Gamedev, where the loaded content took over 300 Mb, but the file itself took 96Kb of memory[5].
- Cons
  - Control – Using this system reduces the number of details that you would use otherwise on the objects generated (specific textures or effects on skills, different types of constructs, etc.).
  - Unusable content – procedurally generated content can be unbeatable or useless. For example, really hard or impassable levels or skills that are too powerful so that the game will be unbalanced and easy to defeat.
- Varies
  - Efficiency – it depends on what the system should be doing. If the programmer's goal is to make a single object then the procedural way takes more time to implement, due to the time needed to develop a procedural system. If the task is to make a larger amount of objects, then the procedural way is a lot more efficient from a time perspective.

2.2 Procedural approach in skills

RPG and MOBA games have different predefined types of skills, where every single one of them needs different defining and extra memory. The reason for this is that usually, the skills have really accurate visualization (for example a skill where ship charges towards characters35). The system, that is built during the research of procedural systems won't be that accurate to create really sophisticated skills, but it will be able to create easier skills such as fireballs, ice blast, etc.. Since procedural methods were investigated first and some games, that have similar solutions, the definition of skills were done so that it would fit into the procedural system without problems. The main problem with the procedural approach is to find out what parts to we need to modify, how we can modify and how much can we modify them.

The result of observing different skill-based systems and skills inside of them leads to the conclusion that they have similarities inside. For example two skills from game Path Of Exile. The first skill that will be inspected is a ground slam. The outcome from inspecting the skill showed, that what it really does is that it deals 100-122.8 damage, takes 6 mana, has cast-time, etc. [16].

16 Ground slam, https://pathofexile.gamepedia.com/Ground_Slam (29.04.2019)

Another skill that was observed is a heavy strike. It deals between 150% – 193.7% base damage to the character, takes 5 mana, has cast-time 1 sec, etc. [17]. This leads to the conclusion, that they are quite close to being the same, just they are different from the parameters. These two were only examples from the pool of skills that were observed, and most of them worked the same way. Skills work the same way, but since the procedural system needs a certain number of parts for the skill then we need to find out how many stats the skills required to get it to function correctly. It leads to the conclusion, that the procedural approach, that will be used is parametric, where every value needs to be set to a different range.

2.3 Similar systems

On the next two chapters, there are different systems, that are using PCG and are close to the current solution from different sides and also a structure of PCG, that was used inside of the program.

2.3.1 Borderland 2 weapons

Borderlands 2 is an open-world action role-playing first-person shooter that was made by 2K games. The reason, why this is brought up here is that it uses a parametric approach as well as the procedural skill-system. It managed to stand out with the high variability of different weapons that you could get in-game using the procedural approach. How they achieved this is that they defined every single part that gun has or will use and found out the limits of given parts. Every gun had 5 basic stats in these types of games (damage, accuracy, fire rate, reload speed and magazine size). These parts were inside every gun type, that was in-game. Altogether there are six types of different weapons: pistols, sub-machine guns, shotguns, assault rifles, sniper rifles, and rocket launchers. In those, they defined peculiarities of them (for example shotguns fire multiple projectiles at the same time and Rocket Launchers bullets explode, etc.). Also to increase the variety of weapons they have different elemental effects (weapons have a chance to get the extra ability to weapons). There are 5 types of effects: Corrosive, Incendiary, Explosive, Shock and Slag. To generate these types of weapons they have weapon manufacturers added into games. There are 8 different types of manufacturers, and they tweak different parameters of weapons. The visual part of the weapons (Textures, effects, etc.) was also created by the manufacturer. Weapon body or barrel was specific to every manufacturer, but the rest of the weapon parts (weapon tray, aim sight, etc.) could be combined together with all of the weapons bodies[18]. As for the skills we can see similarities with a different type of skills such as area targeting or a single target.

17 Heavy strike, https://pathofexile.gamepedia.com/Heavy_Strike (29.04.2019)
18 Borderland 2 weapons, https://borderlands.fandom.com/wiki/Borderlands_2_Weapons (01.05.2019)

2.3.2 .kkrieger memory

.kkrieger is a game, that was developed back in 2004. .kkrieger is a 3D first-person shooter without any specific story. It was specifically developed for the Breakpoint Demoscene Party's 96k Game Competition on April 2004, where the goal of the event was to develop games, that are similar to games Quake and Unreal with comprehended memory[19]. .kkrieger takes only 97289 bytes of data. According to developers, if it were to be stored in a conventional way, it would have taken around 200-300mb[20].

19 .kkrieger description, https://www.mobygames.com/game/kkrieger-chapter-1 (27.04.2019)
20 .kkrieger repository, https://github.com/farbrausch/fr_public (27.04.2019)

3 Research methods

The research method, that is used is observing different types of spells inside of the games and try to find minimal parameters required to generate the skill. There are three games, that will be observed through. The first game is League of legends, the second is Dota 2 and the last one is Path of Exile. Path of Exile is a role-playing game, where the player picks a character, that can add skill gems to its inventory, that give a character power to cast a spell that is defined in the gem[21]. The data of the skills are located on different sites. The objective of the research is to find out a minimal number of parameters required to make a skill.

The first game that will be observed is Dota 2. The distribution of the skill in the wiki is focused on how the spell is cast. There are seven different types of casting variants and altogether 734 skills[22]:

- target unit (184 different skills);
- target point or unit (17 different skills);
- target point (79 different skills);
- target area (69 different skills);
- no target (200 different skills);
- toggle (15 different skills);
- passive (170 different skills);

The second game that is observed is League of legends. There are 794 different skills. There isn't any classification of how the different skills are divided as for Dota 2. All the information about the skills is found on the mobafire home page[23].

The third game is the observed Path of Exile. The skills are divided into three different categories. There are 53 strength, 93 intelligence, and 79 dexterity based skills. In this game character can change the skills by changing the skill gem, that is currently being held inside of inventory. This is different from the MOBA type of games, where each character could use only limited skills, that are applied to certain heroes.

21 Path of exile, https://www.pathofexile.com/game (08.05.2019)
22 Dota 2 abilities, https://dota2.gamepedia.com/Abilities/Abilities_by_type (09.05.2019)
23 League of Legends abilities https://www.mobafire.com/league-of-legends/abilities (09.05.2019)

3.1 Analysis

The analysis was done on 80 skills from each game. From Dota 2 the skills were evenly picked from different casting types. Since there wasn't any clarification on the skills in the League of Legends, then 80 skills were randomly picked from the list of skills. From the Path of exile, the skills evenly picked from a different category.

The first characteristic that is noted on different types of skills is that they could be cast differently. Since the Dota 2 has skills divided into the category on the way how skills are cast, the analysis for the other games on casting worked the same way. The casting type is used in the given program and it is defined in the skill chapter under the targeting types.

There was a significant positive correlation between 80% of the skills, where skills could be visualized using three different objects. The first object handles the creation effects and area explosions around the player or start point. Next handles the visualization between the start and the endpoint. The last object handles effects in the endpoint such as explosions. The spells, that couldn't fit into this category were skills, that modified characters regular attacks.

The parameters of skills were mostly the same which means, that we can use every single parameter for creating skills. These skills modify character different parameters. The more precise description of skill manipulation and different parameters are described in the skill chapter, where the picked design pattern is also picked. design choices that were chosen and further results of observing is described under the skill chapter.

4 Characters

Characters are part of every spell-system since they are the casters and the holders of the skills. Beside of the skill system that players have, they will have also basic fighting abilities[12]. They are used in the games for the player to manipulate with other characters in the game. The gameplay is built around using the character in investigated games.

Illustration 1. Different characters from the game Dota 2.

Each game has its own types of attributes, that describe the character. For example, tabletop RPG Dungeons and Dragons have six different attributes[24]:

- strength (physical power);
- constitution (health, stamina, vital force)
- dexterity (hand-eye coordination)
- intelligence (ability to learn and reason)
- wisdom (common sense, perception)

24 Creating a character in Dungeons and Dragons, http://dnd.wizards.com/dungeons-and-dragons/what-is-dd/classes (10.05.2019)

- charisma (leadership)

4.1 Design

This chapter focuses on design choices done on the character. Character, that is being used in the program is added from the Unity asset store. It is a shop for different assets, that can be imported directly to unity. The image below shows the model in the unity scene[25]:

Figure 1. Model of the character, that is used in-game. The character was uploaded by Maksim Bugrimov at 30 March. 2018. Photo is made inside the Unity editor's scene view.

As for the design choices on parameters, the character has eight different parameters (health, mana, strength, speed, intelligence, damage, defense and attack speed), that describe the character, and helps them manipulate with the rest of the game world.

Firstly the main attribute that characters from three invested games had in-game is health. Health can be reduced by skills, that reduce the health according to the amount of damage the skill can do. The amount of health that player has is determined by the base health player has plus the amount of the strength the character has (usually it is multiplied by some value and added up to health)[12].

The second attribute that every character from the investigated games had is mana. Mana is directly connected with skills that they have since skills are the only objects that use mana. Mana can be added or removed by the skill. The amount of mana player can have is almost the same as for the

25 Maksim Bugrimov, Elf character, https://assetstore.unity.com/packages/3d/characters/humanoids/character-elf-114445 (05.05.2019)

health. There is some amount of mana at the beginning and it can be increased by the amount of intelligence player has[12].

These two are the basic parameters of every RPG character. The following parameters vary, and a skill-based system, that is being made as a practical side of this thesis can be easily changed to add more different parameters to the character. The next three stats define all the different values, that character can have (speed, hit points, mana, etc.)[12].

Strength is connected with two character stats health that was brought up before and a defense that is also added by multiplying with a certain constant. Another stat is speed. Speed influences the movement speed and attacking speed of a character where attacking speed is also a stat by itself. The third stat that also influences other parameters is intelligence. Intelligence is mostly connected with players mana regeneration and the size of the mana pool[26].

The Last parameters are connected with the character attacking abilities where the player can attack another player by pressing right-click on the character. There are three different parameters (attack speed, defense, damage) related to the character fighting system. Damage shows, that how much health will be reduced after each attack. Attack speed as it says by itself shows the speed over what time the player can attack other characters again. Defense shows, that how much damage will be reduced after another character has attacked it[27].

26 How to make RPG stats, http://howtomakeanrpg.com/a/how-to-make-an-rpg-stats.html (01.05.2019)
27 How to make RPG stats, http://howtomakeanrpg.com/a/how-to-make-an-rpg-stats.html (01.05.2019)

5 Particle system

Since the 1980s, the particle systems have been used in different areas, such as video games, animations and for different art pieces. A particle system is a system that emits particles, that have different parameters. As William Reeves noted in his article [28]: "*A particle system is a collection of many many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system.*" [4].

Emitter task in the particle systems is to create new particles. The reason why lots of developers use the particle system is that it helps them create objects that are irregular and complex such as clouds, smoke, fire, water, etc[4]. For example creating different explosions, where first the starting particles are small and have an orange color, and the end ones are large and gray (Illustration 2).

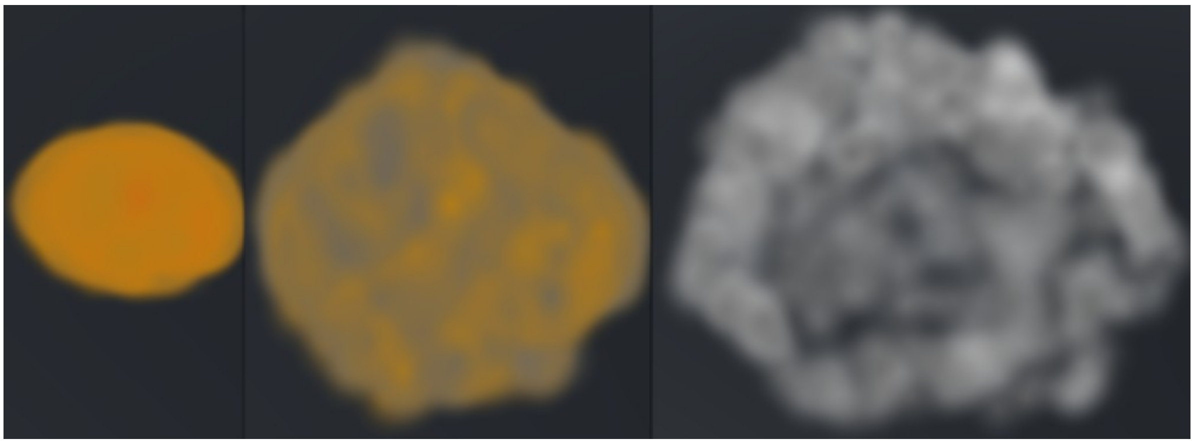


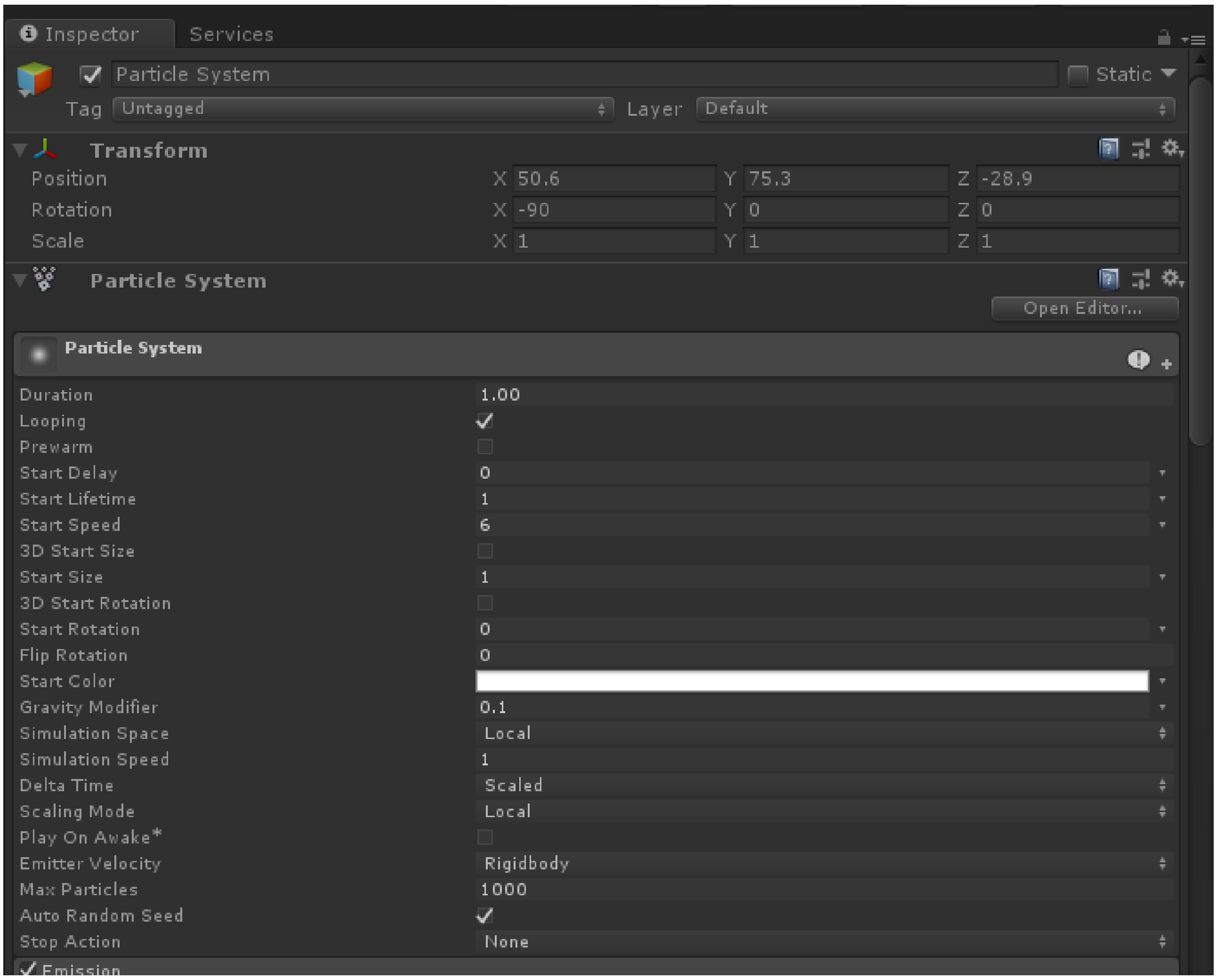
Illustration 2. Explosion simulated in Unity with the particle system.

Property that each particle can be controlled differently in the simulated world makes it a good system to use in different types of games, that use procedural generation. The reason for this is, that the purpose of the procedural system is to find a maximum set of different objects, that can be generated. The particle systems increase the number of skills since there is a large set of parameters, described in the next chapter, that change the skill visual part and so the number of skills increases as well. Why the particle system is the best choice and why they are used in skill-systems is explained under the skill visualizing section.

28 William Reeves, *Particle Systems—A Technique for Modeling a Class of Fuzzy Objects*, https://www.lri.fr/~mbl/ENS/IG2/devoir2/files/docs/fuzzyParticles.pdf (03.03.2019)

5.1 Structure

The particle system is a common feature inside of game engines such as Unreal, Unity, Game Maker, etc. Since the goal of this paper is to build a procedural skill system on the Unity game engine, then this paper focuses on Unity built-in particle systems. Particle system consists of the main module, which consists of general parameters such as duration, start size, etc. and a set of different properties that are organized into other modules. These properties can, for example, modify size over a lifetime, color over lifetime, etc. [4][29]. In Unity, all of the object parameters that are inside of a scene are in the inspector window, which shows the public parameters of an object (Illustration 3) when it is selected.



Particle System ✓ | Static
Tag: Untagged | Layer: Default

Transform

Position	X 50.6	Y 75.3	Z -28.9
Rotation	X -90	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Particle System

Duration	1.00
Looping	✓
Prewarm	
Start Delay	0
Start Lifetime	1
Start Speed	6
3D Start Size	
Start Size	1
3D Start Rotation	
Start Rotation	0
Flip Rotation	0
Start Color	⬜
Gravity Modifier	0.1
Simulation Space	Local
Simulation Speed	1
Delta Time	Scaled
Scaling Mode	Local
Play On Awake*	
Emitter Velocity	Rigidbody
Max Particles	1000
Auto Random Seed	✓
Stop Action	None

Illustration 3. Particle system parameters inside of the inspector.

Transform, that shows objects position, rotation, and scale on the scene, is automatically applied to every object that has been added to the scene and it can't be removed. In unity, developers can't add

29 Particle system, https://docs.unity3d.com/ScriptReference/ParticleSystem.html (09.02.2019)

new game objects into the scene without the transform, since it gives the object the properties, that describe them, where it is, how large it is and how it is rotated.

5.2 Particle system modules

Particle system modules are part of the unity particle system where a rather complex set of properties are organized into ease of use modules. Unity has altogether 19 different modules, including the main module. Since the skill-based system will not use some of them, thus on the next list there are modules that are only needed to build the system [30]:

- emission module (emission module is a necessary module since it contains information of how many particles will be emitted over time and how many are emitted over distance [31])
- shape module (shape module is used, to know, at which direction will the particles start moving and at which radius or the volume of the shape that will emit particles [32])
- force over lifetime module (force over lifetime module is used to add particle force to a certain direction and change its movement according to it [33]).
- size over lifetime module (size over lifetime module adds extra effects to skill visual side and it changes is particles size over lifetime [34])
- color over lifetime module (same as for the size, where the only changing parameter of the particle is color, and it is taken into account over particle lifetime [35])
- noise module (noise module is used to add random movement to particles since it increases the variety of skills [36])
- texture sheet animation module (this module is only used to change the texture of a particle (all particle textures are saved on a single image (also called as sprite-sheet) and given module helps to pick certain particle from image) [37])
- renderer module (renderer module settings show, how particles image is transformed, shaded and overdrawn by other particles [38])

30 Particle System modules, https://docs.unity3d.com/Manual/ParticleSystemModules.html (03.04.2019)
31 Emission module, https://docs.unity3d.com/Manual/PartSysEmissionModule.html (03.04.2019)
32 Shape Module https://docs.unity3d.com/Manual/PartSysShapeModule.html (03.04.2019)
33 Force Over Lifetime module https://docs.unity3d.com/Manual/PartSysForceOverLifeModule.html (03.04.2019)
34 Size Over Lifetime module https://docs.unity3d.com/Manual/PartSysSizeOverLifeModule.html (03.04.2019)
35 Color Over Lifetime module https://docs.unity3d.com/Manual/PartSysColorOverLifeModule.html (03.04.2019)
36 Noise module https://docs.unity3d.com/Manual/PartSysNoiseModule.html (03.04.2019)
37 Texture Sheet Animation module https://docs.unity3d.com/Manual/PartSysTexSheetAnimModule.html (03.04.2019)
38 Renderer module https://docs.unity3d.com/Manual/PartSysRendererModule.html (03.04.2019)

5.3 Particle textures

In addition to different modules, each particle also can have a different texture. That is the reason why the texture sheet animation module is needed. Altogether there will be 36 different textures, that symbolize different types of ways to give skills different look. Sprite-map below contains different types of particle textures, that are added to the skill generation system. The ideas for the textures were taken from opengameart [39] and from unity asset store [40].

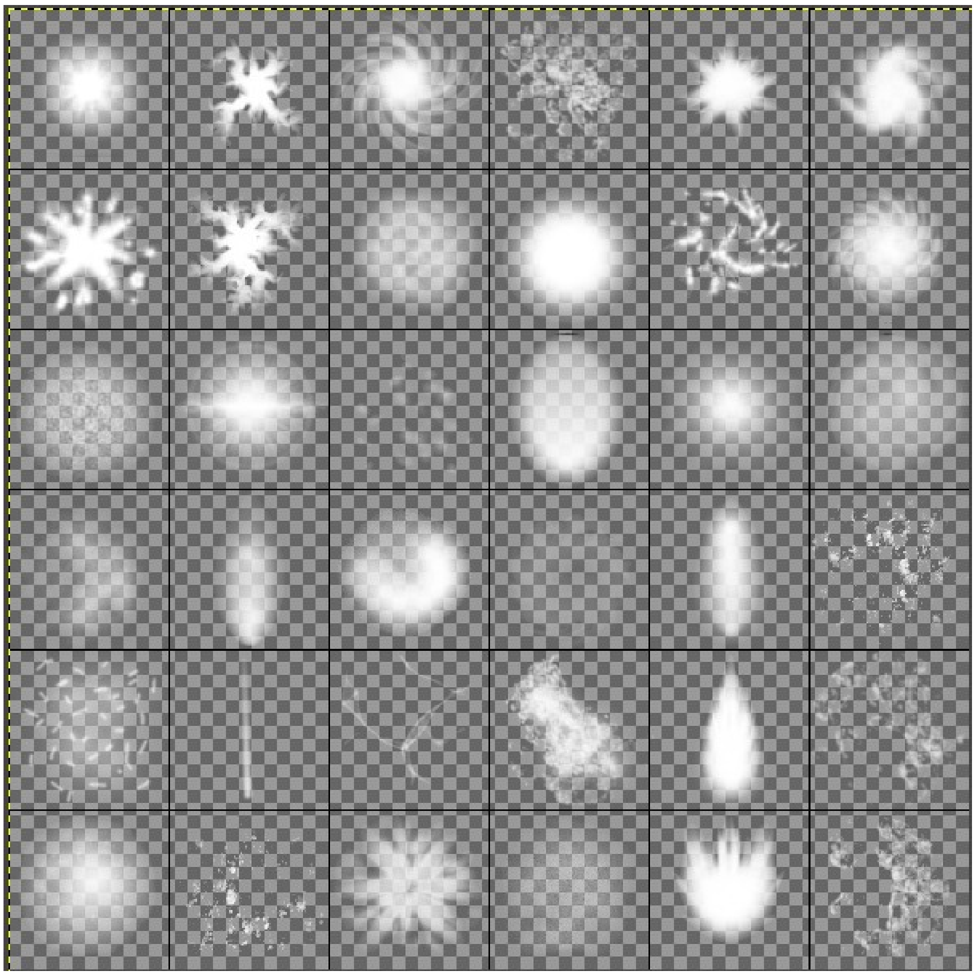


Illustration 4. particle textures

39 Spritemap, https://opengameart.org/content/particle-pack-80-sprites (03.04.2019)
40 Asset store, https://assetstore.unity.com/packages/2d/textures-materials/particle-textures-deluxe-8918 (03.04.2019)

6 Skills

Skills are an essential part of magic type systems since the game-play inside of those games is usually focused on developing and upgrading different skills. Skills are used inside of the games for different actions, such as attacking enemies, collecting gold, destroying doors, etc.. Under the skill we can think of different characters abilities to affect other characters throughout different effects, such as projectiles (fireballs, ice blasts, etc.), different passive effects (burning floor, acid, etc.) or even simple explosion around the character[41]. From the developer perspective of the skill, it consists of the collider, that affects all of the targets it meets with skill, and particle system, which generates visual effects for the skill (Illustration 5).

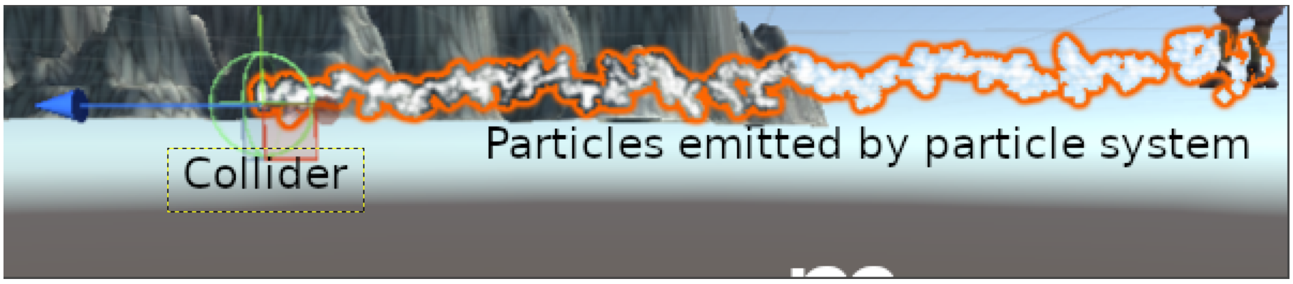


Illustration 5. Skill main structure

Results from observing different skills showed, that most of the spells can be visualized with three same types of skill parts such as showed on Illustration 5. The first part is used to cast spells, that make surrounding damage or different starting effects. This is used to generate skills that are area targeting type (explained on targeting types chapter). The second part is a trail, that leads from the start point of the skill to the endpoint of the skill. The last part is most of the time an explosion that affects every character inside of the endpoint collider.

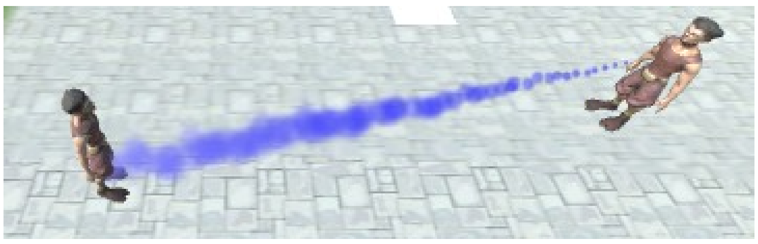


Illustration 6. Skill cast from the character on the left to the character on the right

41 Dota2 abilities https://dota2.gamepedia.com/Abilities (05.04.2019)

6.1 Structure

The skill consisted of three different parts starting point, middle trail and ending point, so the skill uses a box-based approach, where skill is made up of different definable sub-items. The main difference about them is that the starting point gets only starting point inside, the trail gets an array of points and the ending point gets the end of the point. The difference comes in the visual side of the skill. The particle system module called emission module allows us to set how many particles over a lifetime or over distance the particle system will emit. Start and end point visualizers will emit particles over time and the trail in the middle emits particles over distance. The structure described before allows the user to define each block of the skill separately and not at all if the player decides, that it is not necessary.

6.2 Parameters

An initial objective of defining the character was to identify all of the modifiable parts of the character. Observation showed, that skills also affected friendly targets inside of the game. So the skill can affect two types of characters inside of the game:

a) enemy characters;

b) friendly characters;

Because of that, every character stat needs to have a positive effect on the friendly units and a negative effect on the hostile units to make the generation more pseudo-random. Next parameters from the character are the parameters or modifiable stats, that skill will change. Findings on the character chapter show, that there is all-together seven modifiable values:

a) health;
b) mana;
c) strength;
d) speed;
e) intelligence;
f) damage;
g) shield

These findings may be somewhat limited since characters defined by the developer itself could have extra stats for the character so the size of the modifiable stats could be larger. Another important finding was when observing skills from different games like Dota 2, Path of Exile and League of

Legends was that skills had most of the time more than 1 modifiable value. For example, a skill in Dota 2 called "Poison Touch" reduces enemy character speed by 14% and does 16 damage per second[42]. This leads to the conclusion, that we need to save them as a list in skill object because we can have multiple parameters that skill can change.
Another result about observing these values is that for the characters only health and mana are regenerating by themselves. Other values are final and can be changed by leveling up, acquire a certain type of item, by skills, etc.. This means, that these other values need to be returned to the player after skill duration has ended since otherwise, the game would be unbalanced. As an example, characters would run out of strength or speed.
Another result in observing different skills showed, that there are also parts, that are not connected with the character but are essential for each spell. There are four parameters, that are that type:

- duration, that shows how long the skill effect will last.
- iterations, that how many iterations each spell has (for example 16 damage, six iterations, and 10-second duration means that it does 16*6 = 96 damage over the time of 10 / 6 seconds with 16 damage each iteration);
- mana, that shows how much mana the spell needs to be used and how much will be reduced from the player mana pool. This is used to balance the game between different spells;
- cool-down, what shows that how much time does it take the player to reuse the spell again. Also used to balance the game;

6.3 Targeting types

Another necessary part for the skill is how would a system affect different characters in-game. Under the targeting types, we are thinking of the way spells will be delivered to character objects in-game. In observed games, it is done by so that the player first clicks on the spell and after that clicks on the enemy. The result of it will do a direct hit to that certain enemy.

That is only one way of doing it, but there is a big variety of ways we can approach this. For example, instead of one enemy, the player can target two. A way to track down as much targeting types as possible is to look through different types of games that are using spell-system. One of the way, that will be used to determine targeting types, would be looking through most popular MOBA-s since they all have some type of spell-system in place. Also during defining the types, skills will get extra parameters, since it is necessary for some type of skills to exist.

42 Dazzle, https://dota2.gamepedia.com/Dazzle#Poison_Touch (05.04.2019)

At the beginning of targeting types chapters, there will be a short summary of added variables and overall structure of types, so its easier to understand different types. After the structure of the targeting type is defined, there will be a definition of different targeting types by showing uses on different games.

6.3.1 Single-target

With a single-target type, the spell only affects the enemy to who the spell was cast. The spells will follow the enemy once the spell has been cast. We only need to know the enemy target, which is found when the player clicks with active spell-cast. Example of this is a skill from the game Dota 2 called „Magic Missile" that can be cast only to one target[43]. The image below illustrates the use of single-target skills:

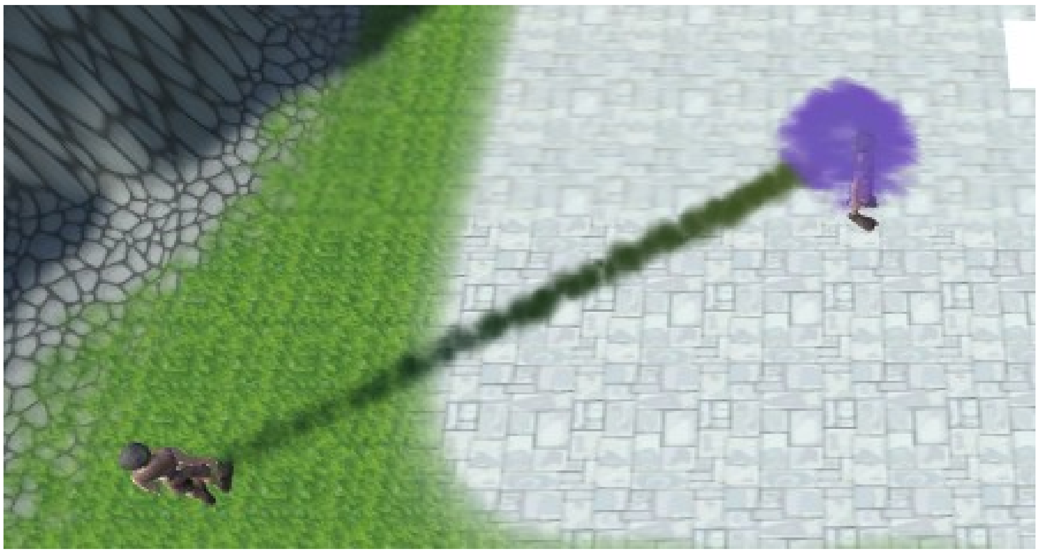


Illustration 7. A projectile that explodes when the character has been hit

6.3.2 Area targeting

Area target is a type of targeting type, where all the targets around the character that was inside of the spell effect range will be affected by the spell. On activation, the area target spell can be cast to a certain point or another way would be to immediately cast it around the player. For this type of spells, we need to define the area of the spell [44]. The image below illustrates the use of area target skills, where it is cast around the character:

43 https://dota2.gamepedia.com/Vengeful_Spirit#Magic_Missile (02.02.2019)
44 https://dota2.gamepedia.com/Abilities (02.02.2019)

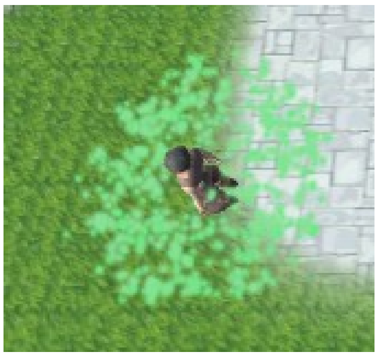


Illustration 8. Character activating area skill

6.3.3 Vector targeting

Under the vector targeting, we are looking at the skills, where ability has start and direction, and all the characters between the „line" will be affected with the skill. To activate the current skill, the player needs to first cast the spell. Then it needs to click on the starting point to initialize the first point. After that, the player needs to hold the key and drag to the wanted direction. After that, the skill will be released. Under this category we can also add spells, that can use the player position as a starting point. All that player needs to do is that cast the spell and click on the wanted position. To sum up vector targeting we need to know start point, direction (We can calculate it by subtracting endpoint from start point), and the width of the spell. An example of this type is a skill called „Wall of replica", that creates a wall from starting point to the given direction and affects all the characters that pass it with skill.[45] On the image below, the character is activating the skill from one point to another, inside of the casting range.

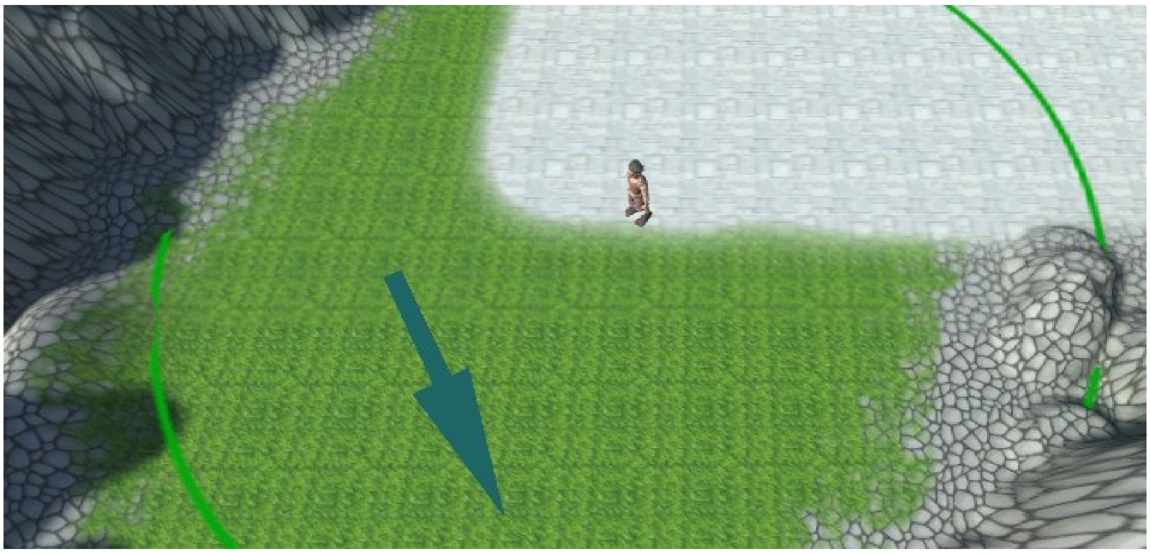


Illustration 9. User pulling a vector from the start point to the endpoint

45 https://dota2.gamepedia.com/Dark_Seer#Wall_of_Replica (02.02.2019)

6.3.4 Conclusion of targeting types

As we found out, there are three types of targeting, that can be considered as type. Part, that comes out from this is that we can now find out, what input are we waiting from the player. For a single target, we only need to know the endpoint or the target, that was selected. For area target, we only need to know if the player has activated the spell. For vector targeting, we need to know start and endpoint. So we can say, that we need to know 3 positions from the player:

- start point;
- endpoint;
- target (null, if the target was ground);

The program needs to find out the rest of the parameters (area, width, etc.) out by itself procedurally.

6.4 Trails

The trail is a part of the skill, that shows where the skill needs to pass through. Skill path tracker is a part, that handles the detection of characters in-game and sets the skill whether it should be affected by skill or not. The trail is necessary since this system is using a particle system as a visualization system and the particle system is emitting particles on the passed trail. There are different types of traces that can be created, but the goal of this is to find out the way to make them procedural as well. Since there are a large number of different games focusing on skill-based system, one of the easiest way would be to look through some of the skills and highlight most common trails inside of games. We are looking at the spells from Dota 2 since it has an overview of all of the skills in wiki[46]. As going through all the spells, most of the spell trails are linear from start point to endpoint. There are also skills, that move towards the player, as it would move on the sinusoidal wave. The skill trail, that was not so typical was type, where the skill makes circles around the cast point. In this paper, we will only add these three to the program. But there are more different paths that can be implemented (for example skill, that swirls around the character).

46 https://dota2.gamepedia.com/Abilities/Abilities_by_type (06.02.2019)

6.4.1 Linear trail

The linear trail was the most common path of skills to move on a projectile from player to enemy. Usually, as for the linear trail, the particles of which the skill consists of, have randomized movement, so the skill looks like it is moving non-linearly, but it follows the points on the linear path. The image below illustrates the points, that linear trail will pass:

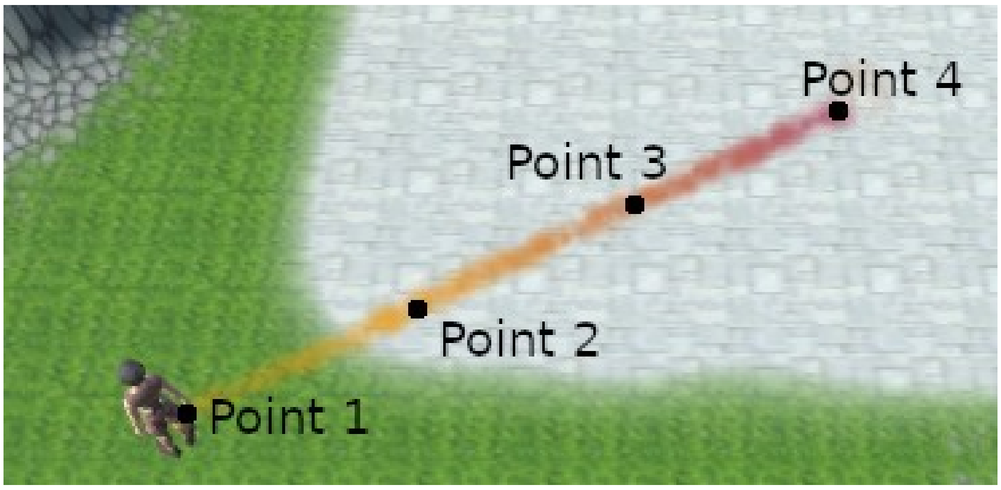


Illustration 10. Linear trail points

6.4.2 Sinusoidal trail

Under the sinusoidal path, all the points are calculated as moving on the sinusoidal wave. Sinusoidal needs to have minimal 4 different trail points. More than 4 will make movement more uniform. The trail from the start point to the endpoint will have a wave, which is in a range from 0° to 360°. There are sine and cosine functions for creating a sinusoidal wave. On the calculation, the sinusoidal wave has two extra points between regular points on the sinusoidal wave. Illustration below shows, how the points are distributed on the trail. The skill itself had 5 points defined (P1, P4, P7, P10, P13). Other points were added to create wave:

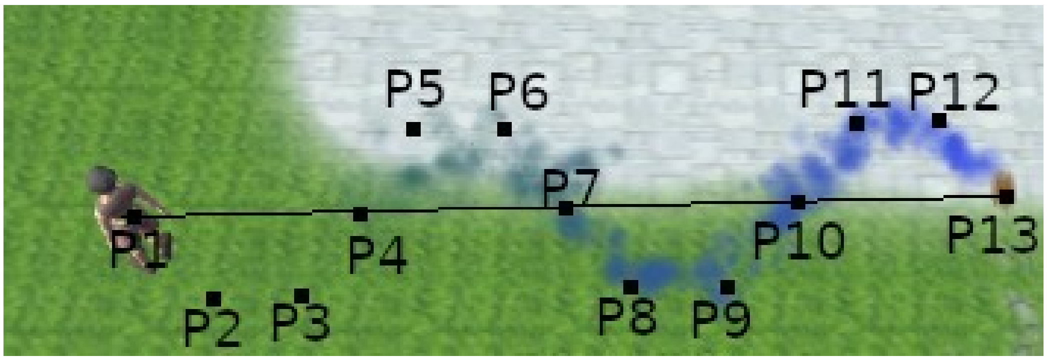


Illustration 11. Sinusoidal path points.

6.4.3 Circular trail

Circular trail creates a path containing points, that will be cast around the character. When the current trail type is picked, first the spell will move away from the player and moves toward the distance of cast point. After the skill has arrived at the distance of the cast point, it will make two extra rounds before ending the endpoint, where the third part of the skill will be activated. For the circular trail, the script calculates 30 different points, that it has to pass. The reason for this is that it will make three circles around the character and to make it look like it is a circular shape, it needs to have enough points to make it look so. The first 10 points are used to move away from the character. The rest of the 20 points will make two circles around the player. Image below shows, how circular movement points are calculated:

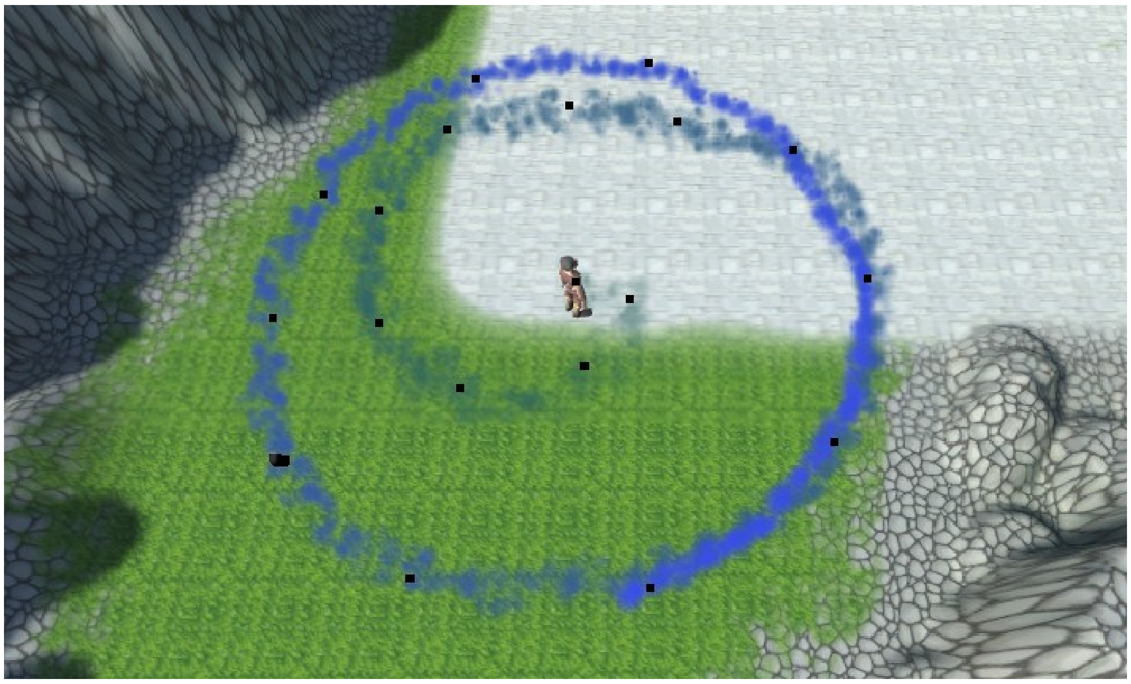


Illustration 12. Circular trail points.

6.5 Scope

Scope of the skill shows, how many and which targets can spell effect. Most of the investigated skills are single-target spells, that gives these spells a scope value of one. There are some more effects that can be brought up. For example the skill „Kinetic blast" from Path of Exile, that affects every enemy in the area[47]. The decision comes down to a range that will be picked, how many ways and how the procedural generation program will set the parameters. In this paper there will be 6 different types defined:

47 https://pathofexile.gamepedia.com/Active_skill_gem (06.02.2019)

- enemies by count (1, 2, 3, 4.. etc);
- all enemies inside of the area;
- single enemy;
- allies by count (1, 2, 3, 4.. etc);
- all allies inside of the area;
- single ally;

6.6 Visualization

Visualization is part of the skill-system that really builds up the feel of the skill system. There are different ways of making visual effects, like example line renderers, trail systems, particle systems, animators, etc.. For current task the particle system was picked, because with line renderers, there is a problem with skills, that are meant to be as projectiles. The reason for this is that the line renderers render the image from point to point. That means, that lots of points are needed to make a line to look like a projectile.

Trail renderer is something that this kind of system needs since they leave a path behind the trail they have passed. The reason, why the particle system is used instead of the trail system is that there are not many variables to modify inside of the unity systems for the trail systems and it is possible to create the same effect with the Unity particle system. The trail renderer does not have an option to visualize explosion or stationary animation when skill is for example with type area targeting. A bonus attribute about the particle system is that we can modify each particle, that it has emitted.

There are more different variables, that are implemented in the Unity particle system like size, color, and rotation over a lifetime. Another positive side of the particle system is the noise system, that changes the movement, rotation, and scale of each particle, which helps to remove the monotony of a particle movement. Particle system will not be a part of skill affecting or character detecting, even though Unity has methods to detect the collision for each particle. Reason for this is that particles have different size and position over a lifetime. When the projectile moves towards the player it is hard to calculate the area of moving projectile. Instead of this, there will be another collision detection box that will be added to the same object as the particle system. To come back to visualization, it will be done with a particle system by emitting particles over distance passed.

7 Program

7.1 Structure

The result of the investigation skills and procedural system leads to a conclusion, that the structure of the procedural approach consists of five different parts. The first part handles the input of player and checks, what type of skill to cast. Before using, the predefined datatype, that has information about the first part need to define three different parameters:

- cast type;
- split count;
- range;

All the given parameters are defined in the skill section. The next part of the skill is needed to build up the next three parts of the skill. As noted in the skill chapter, the skill could be visualized with three different objects. The second part makes sure, that these objects get correct coordinates to follow, sets them active. It also has some parameters, to increase the variety of skills:

- cast height;
- trail;
- skill point path rate;
- points on the path;
- skill parameters;

The last three parts are practically the same with only change of coordinates they will take as a movement path. As for the public parameters, that player needs to set inside of the data-set are the following:

- area;
- depth;
- recast on hit;
- move duration;
- particles lifetime;

These are the parameters, that will affect the back-end of the program. Front-end is focused more on manipulating the particle system parameters. All the parameters, that will be changed are all focused on changing modules, that were explained in the particle system modules. There is a total of 26 parameters for the particle system. All the parameters are defined by their maximum and minimum values. As for the float and integer types, the calculation of value is done with the following formula:

```
    value = minimum + (maximum-minimum) * seed.
```

Boolean values will be calculated by checking if the seed has been calculated less than 0.5 (seed is in a range from 0 to 1). As for the enum values such as trail type, it will be divided into equal intervals, where every interval has information about a single value. The formula for this will be:

```
    value = (enum) enum.length * seed.
```

Next illustration shows the illustration of the skill system inside of the program that was created:

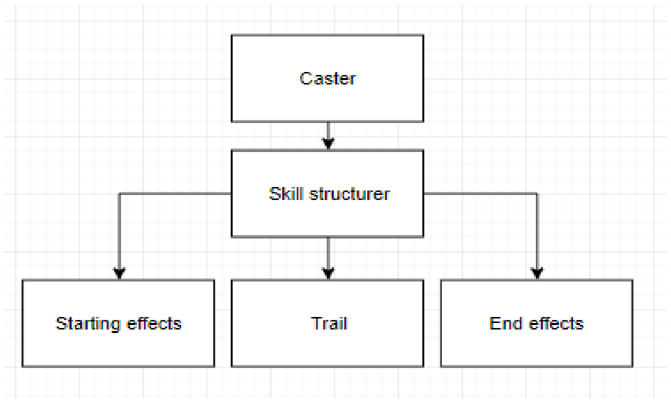


Illustration 13: The structure used in the program.

The seeds for the values are generated with a pseudo-random generator based on iterating through a sequence defined by some seed. Number generator, that is used to get random seeds, wherefrom infinity set of numbers a mathematical function calculates values, that seems as they were random. When generating skills, there is only one single seed, that will set seed for a random number generator. The seed for each parameter is taken from the set of numbers, that are generated by the random number generator. The color of the skill is also calculated the same way as the other parameters, where red, green and blue are in range 0 to 1, where 1 means 255 in color code. The system also uses object pooling. This means, that all the skills, that are generated are saved as inactive objects. This helps to save processor speed from creating and deleting different skill parts[48].

48 Object pooling https://www.raywenderlich.com/847-object-pooling-in-unity (05.05.2019)

7.2 GUI

The program, that was made during this paper is a test generator, to show, how the skills are generated. When starting a program, there are six different buttons in the top right corner as seen on the image below:

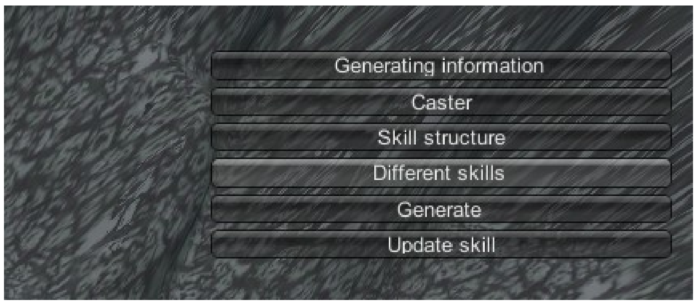


Illustration 13. Buttons of the program.

These are the main settings of changing the skill parameters. When a user clicks on the generating info button, then under these buttons another set of buttons appears. There are two fields that the user can modify. First is the seed from where skill generates fields for itself. Second is a Boolean value, which shows if the seed is changed when the new skill is generated. Generating is necessary to set to false when the user wants to generate skill with a certain seed. Buttons are shown in Illustration 14.

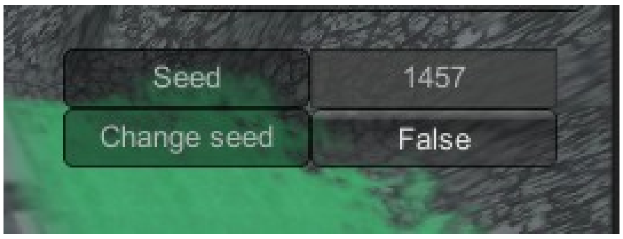
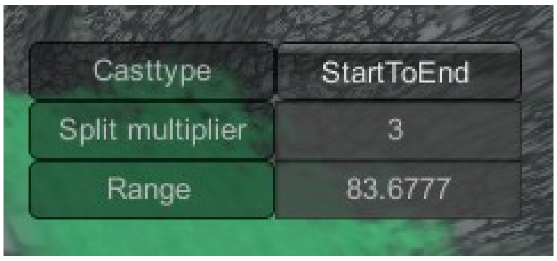

Seed 1457
Change seed False

Illustration 14. Generating info parameters.

The next button is the caster parameters. When clicked on that, the same type of buttons and text fields appear as for the generating information button. Caster view has three parameters, that user can change:

- cast type;
- split multiplier;
- range;

Every time, when there is a change, the user needs to click on the update skill button. The reason for this is that since the system uses object pooling, then old objects have old values, and this button removes these objects. The image below shows the caster view.



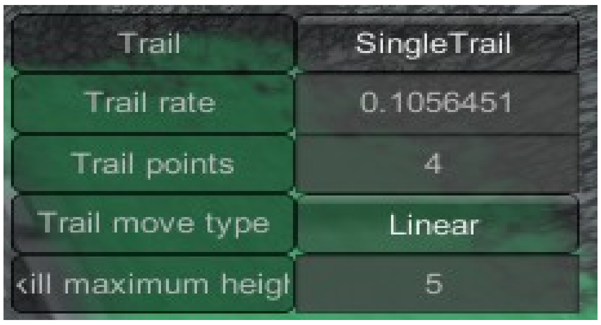
Casttype	StartToEnd
Split multiplier	3
Range	83.6777

Illustration 15. Casting buttons

The structure is the same as for the casting view is a view, that manipulates with skill parameters. There are a total of 5 different parameters, that user can modify

- trail;
- trail rate (works on points up and points down);
- trail points;
- trail move path;
- skill maximum height;

The view of skill structure is shown on Illustration 16.



Trail	SingleTrail
Trail rate	0.1056451
Trail points	4
Trail move type	Linear
kill maximum heig	5

Illustration 16. Skill structure view

Different skills button has a list of predefined skill. These lists are a subset of all of the skills. The next chapter explains, how the data is defined to be on the range. There are currently three defined skills:

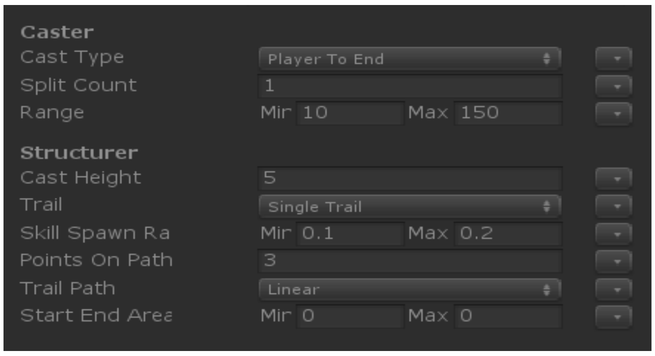
- random is the largest set, that there is. It has the largest interval for each parameter;

- the projectile is the subset of random and it contains all the different types of projectiles, that there are possible;
- area effect is also a subset of random;

´The last two are used to generate and update skill information. Generate button changes the seed and generates new skill according to the seed. This button can also be used to create a skill from a certain seed. User needs to disable changing seed at generating information to do so. Update button is used to apply all the changes, that user made inside skill structure view or caster view.

7.3 Data types

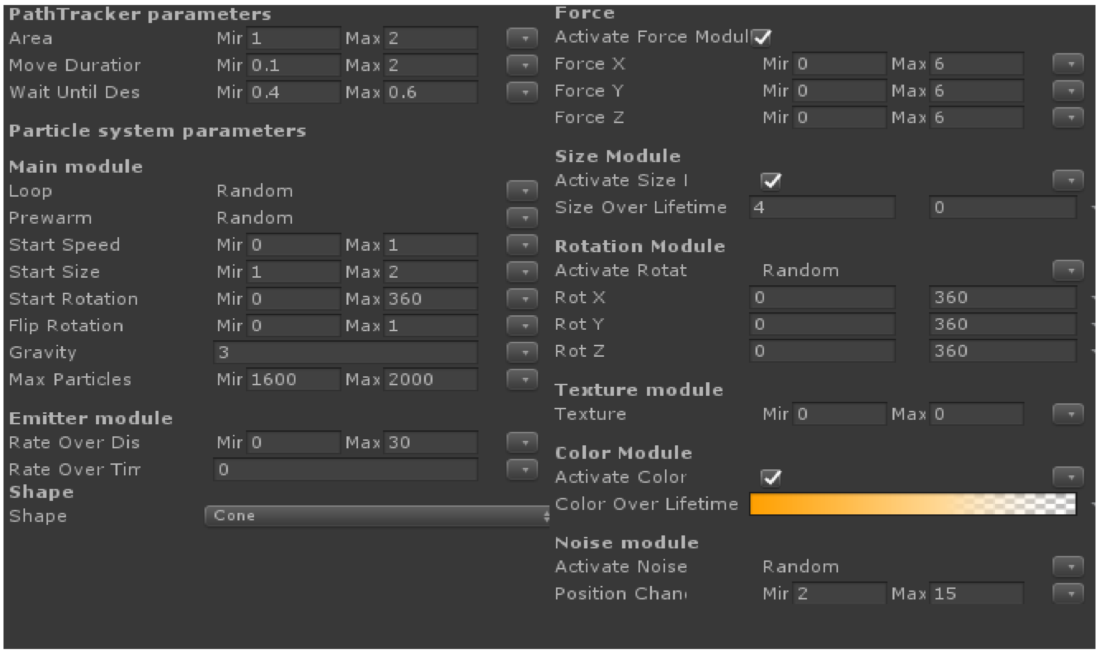
There are three different data types. On the images below, it shows all of the intervals, that are predefined for the projectile type of skills. The first data set contains information about the caster and the skill structurer. Illustration 17 shows all of the parameters, that user needs to define:



Caster		
Cast Type	Player To End	
Split Count	1	
Range	Mir 10	Max 150
Structurer		
Cast Height	5	
Trail	Single Trail	
Skill Spawn Ra	Mir 0.1	Max 0.2
Points On Path	3	
Trail Path	Linear	
Start End Area	Mir 0	Max 0

Illustration 17. The data type for caster and skill structurer.

The next data type defines the parameters for the path trackers and particle system. Illustration 18 shows all the parameters that need to be defined.



PathTracker parameters

Area	Mir 1	Max 2
Move Duratior	Mir 0.1	Max 2
Wait Until Des	Mir 0.4	Max 0.6

Particle system parameters

Main module

Loop	Random	
Prewarm	Random	
Start Speed	Mir 0	Max 1
Start Size	Mir 1	Max 2
Start Rotation	Mir 0	Max 360
Flip Rotation	Mir 0	Max 1
Gravity	3	
Max Particles	Mir 1600	Max 2000

Emitter module

Rate Over Dis	Mir 0	Max 30
Rate Over Tir	0	

Shape

Shape	Cone

Force

Activate Force Modul	✓	
Force X	Mir 0	Max 6
Force Y	Mir 0	Max 6
Force Z	Mir 0	Max 6

Size Module

Activate Size I	✓	
Size Over Lifetime	4	0

Rotation Module

Activate Rotat	Random	
Rot X	0	360
Rot Y	0	360
Rot Z	0	360

Texture module

Texture	Mir 0	Max 0

Color Module

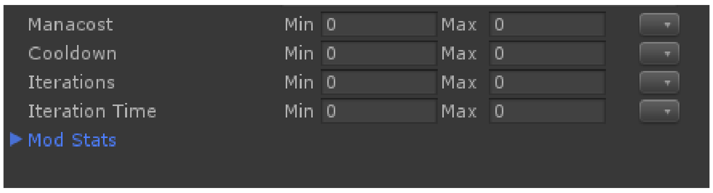
Activate Color	✓
Color Over Lifetime	🟧

Noise module

Activate Noise	Random	
Position Chan	Mir 2	Max 15

Illustration 18. Parameters of the path tracker and particle systems.

The last data type describes the skill parameters, that will affect the characters. The parameters are defined below:



Manacost	Min 0	Max 0
Cooldown	Min 0	Max 0
Iterations	Min 0	Max 0
Iteration Time	Min 0	Max 0

▶ Mod Stats

Illustration 19. Parameters of the skill.

8 Conclusion

The goal of this paper was to describe different procedural approaches, define skills and connect these two values. For this, the author looked through the most popular MOBA games: League of Legends, Dota 2 and Path of Exile. The main research method was to observe different types of skills and try to find similarities between them to connect them with procedural methods. The procedural approach, that was picked was a parametric approach, where the whole system is made up of parameters, that have a maximum and minimum value.

The result of this paper was a system, that generates different skills according to the predefined data types. There are three data types, that defined the whole system. First defines all the parameters, that are connected with casting the skill, second focuses on visualizing different skill parts and the last one defines the parameters of skill, that will be affected with the character. Visualization of the skill has a box-based approach, where skill is divided into three different parts (starting point, middle trail, and endpoint). Users can define each part differently and the rest of the skill stays the same. The skill generator provides an easy way to generate and test different skills.

The main goal, that was set to define and make a system, that generates skill was fulfilled. The program that was created has also a character with different parameters, to show how the skill system would look like with the characters as the users of the skills. For future improvements, there could be added different sound effects and also a skill name or description procedurally. Since the current task of the paper was to find out how the skills work and how to define it, then in future improvements, there could be also testers, who would make a game with a given system and give feedback on how this system helps them.

9 References

[1] A. Drachen, M. Yancey, J. Maguire, D. Chu, I. Y. Wang, T. Mahlmann, M. Schubert, D. Klabajan (2014), "Skill-based differences in spatiotemporal team behavior in defense of the ancients 2 (dota 2)", *Games Media Entertainment (GEM),* Montreal, QC, Canada, *https://ieeexplore.ieee.org/document/7048109* (07.05.2019)

[2] Noor Shaker, Julian Togelius, and Mark J. Nelson (2016), "*Procedural Content Generation in Games: A Textbook and an Overview of Current Research*". Springer.

[3] Julian Togelius, Georgios N. Yannakis, Kenneth O. Stanley, Cameron Browne (2011), "Search-Based Procedural Content Generation: A Taxonomy and Survey", IEEE, https://ieeexplore.ieee.org/document/5756645 (07.05.2019)

[4] Bingqing Zhang, Wenfeng hu (2017), "Game special effect simulation based on particle system of unity3D", IEEE, Wuhan, China, https://ieeexplore.ieee.org/document/7960062 (07.05.2019)

I. License