Ingeniería y Ciencia
ISSN:1794-9165 | ISSN-e: 2256-4314
ing. cienc., vol. 11, no. 22, pp. 95–119, julio-diciembre. 2015.
http://www.eafit.edu.co/ingciencia

Towards Procedural Map and Character Generation for the MOBA Game Genre

Alejandro Cannizzo[1] and Esmitt Ramírez[2]

Received: 15-12-2014 | Accepted: 13-03-2015 | Online: 31-07-2015

MSC:68U05 | PACS:89.20.Ff

doi:10.17230/ingciencia.11.22.5

Abstract

In this paper, we present an approach to create assets using procedural algorithms in maps generation and dynamic adaptation of characters for a MOBA video game, preserving the balancing feature to players. Maps are created based on offering equal chances of winning or losing for both teams. Also, a character adaptation system is developed which allows changing the attributes of players in real-time according to their behaviour, always maintaining the game balanced. Our tests show the effectiveness of the proposed algorithms to establish the adequate values in a MOBA video game.

Key words: procedural content generation; multiplayer on-line battle aren; video game; balanced game

[1] Universidad Central de Venezuela, Caracas, Venezuela, alejandro.cannizzo@gmail.com.
[2] Universidad Central de Venezuela, Caracas, Venezuela, esmitt.ramirez@ciens.ucv.ve.

Generación procedimental de mapas y personajes para un juego del género MOBA

Resumen

En este artículo, presentamos un enfoque empleando algoritmos procedurales en la creación de mapas y adaptación dinámica de personajes en un videojuego MOBA, preservando el aspecto de balance para los jugadores. Los mapas son creados para ofrecer igual oportunidad de ganar o perder para los equipos. También, se desarrolló un sistema de adaptación de personajes que permite cambiar en tiempo real los atributos de los jugadores de acuerdo a su comportamiento, siempre manteniendo el balance. Nuestras pruebas demuestran la eficacia de los algoritmos para establecer los valores adecuados en un videojuego MOBA.

Palabras clave: generación procedimental de contenido; batalla en arena multijugador en línea; videojuego; juego equilibrado

1 Introduction

Nowadays, a computer game demands several materials to compose complex and large scene, carrying high costs in their content development. The Digital Content Generation (DCG) - subset of the Procedural Content Generation (PCG) algorithms - is a keystone in modern video games which is formed by a set of algorithms to produce levels, maps, assets, characters, weapons, and others, offering a reduction on game production costs. The main goal of DCG is to create novel game elements, off-line or on real-time.

Since early days of video game development, developers had issues regarding the memory taken up by the game levels they created. Generally, this introduces a limitation to the size of the virtual worlds created. In order to be able to create bigger worlds, the developers focused on reducing the size of data employed for virtual world creation. Thus, DCG algorithms were created to be used as a tool to reduce the storage space taken up by the levels created on video games.

The DCG is used by developers to solve a big problem in the development cycle of video games: time. Developers use these algorithms to reduce the amount of time taken to create a game or certain features of it. Particularly, there is a video game genre called Real-Time Strategy (RTS) where participants have to control secure areas of a map and eliminate their

opponents assets (*e.g* towers, buildings, shelters, and so on). In this genre, the *Multiplayer on-line Battle Arena* (MOBA) is a sub-genre which allows the presence of two teams to battle against each other. Generally, maps are static and the content available to players is constantly growing to scale the game. When these games are created, game designers have to spend increasingly amounts of time due to design issues caused by the newly added content. Also, terrain of game levels in MOBA are based on height maps, which are conveniently designed to apply procedural algorithms.

A distinguished area in MOBA is concerned with characters, where they are designed to evolve in time during the game (*i.e.* on their power or abilities). In order to reduce the time required and to make fast MOBA prototypes, we present an approach to the study and implementation of a DCG algorithm in a MOBA video game. Mainly, we focus on the generation of terrain maps and the evolution of player's abilities in the game using genetic algorithms.

This paper is structured as follows: Section 2 presents a brief introduction in dynamic content generation as a video game design solution, including a few previous works in that area. Following that, in section 3, we explain details of our proposal to use the DCG algorithms in MOBA video games. Section 4 shows the experiments and results achieved using our approach. Finally, conclusions and future works are presented in section 5.

2 Dynamic content generation

The procedural generation is the process to generate content algorithmically rather than manual. This research area is not new in Computer Science, where several works were developed. Procedural generation originated in 1975 when Benoit Mandelbrot [1] defined the *fractal* object which describes such repeating or self-similar mathematical pattern. This idea is used in several application such as creation of textures [2], cities [3], forests based on L-Systems [4], terrains [5], video games [6], and other areas. Commonly, in video games this process is called Dynamic Content Generation.

The powerful of procedural generation allows to integrate several re-

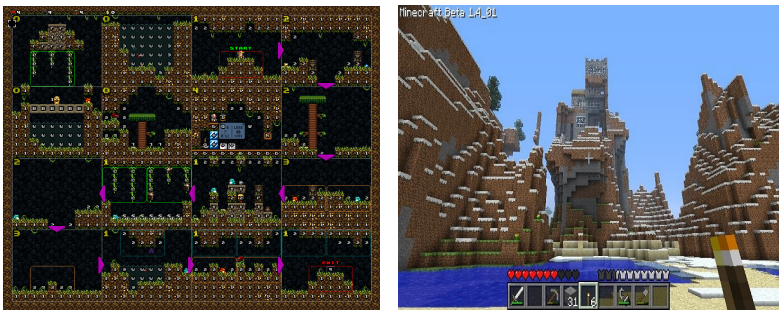
search field to solve problems. For instance, in 2013, Genevaux et al. [7] define a novel way of generating terrain based on hydrology. Their algorithm takes as initial input the contour of the terrain and some rivers given by the user. Then, the algorithm will take the input information and start generating a complete drainage river network, resulting in a complete formation of rivers that go from springs to outlets, with the expected height variation. After the rivers are created they start working on the remaining terrain using different building blocks to and modifying the terrain accordingly to the rivers so that the overall result of the map is topologically correct.

Specifically in the video game development area, Dynamic Content Generation was used to create larger levels with lesser amounts of data to address issues regarding the memory storage at that time. After the memory storage concerns dissipated, the DCG started to be used to generate other kinds of content to increase the effective range of content that was show to the player, removing much of the monotony produced by repetitive content. These algorithms are not restricted to be used during run-time and that they can generate content to be embedded inside a game once it is finished. As an example, a forest may be created procedurally to be later inserted into the game as a static asset that will remain the same throughout the game.

It is possible to classify DCGs according to the moment of the generation of the data (on-line vs off-line), or the prioritization of their dependences in the game (necessary vs optional), or based on the base values to start the generation of dynamic data (simple seed vs parameter vector), or by level of randomness (deterministic vs no-deterministic), or by measuring the evolution of generations (one-time construction vs constant-time construction), etc. The important fact is that these techniques were used as tools by developers to shorten the development time of the project and to increase the amount of content presented to the player. A remarkable example of this was presented in 2011 by Browne [8] developing a tool which involves the deep integration of procedural content generation into game mechanics and aesthetics to many genres, including PCG-based games.

There are several research works in procedural game content to generate coherent visual aspects on a game. In the literature there are several examples in adaptive or personalized procedural content according to a

game genre [9],[10]. For instance, distinguished examples such as the optimization of tracks in car racing games [11], weapons generation for space shooter games [12], rulesets for board and predator-prey games [13, 14], automatic levels based on playing style for platform games [15],[16], music generation according to the genre or mood of games [17], and others. Similarly, it is possible to generate dynamic content based on a set of instructions using some language such as ASP (Answer Set Programming) for PCG on spaces [18]. Particularly, level generation has been applied since the rise of previously popular games such as Rogue®[1], Diablo®[2] and Spelunky®[3] to current games such as Minecraft®. Figure 1 shows a part of Spelunky® where rooms and obstacles were generated procedurally, as well as a screenshot of Minecraft®.



(a) **(b)**

Figure 1: An example of the procedural generation, in (a) Spelunky®, and (b) Minecraft®[4].

2.1 Dynamic Content Generation on MOBA

As mentioned before, MOBA *Multiplayer on-line Battle Arena* is a subgenre of RTS in which two teams exists, each with the objective of destroying the main structure (*e.g.* tower, castle, fort) of the opponent. Typically,

[1] http://science-fiction.fch.ir/rogue/doc/Rogue_1984-The_DOS_Game-The_History-The_Science.html
[2] http://blizzard.com
[3] http://spelunkyworld.com/
[4] https://minecraft.net/

a character belonging to a team has various abilities and advantages that improve over the course of the game (increasing the global performance of its team).

There are a number of MOBA video games that created game worlds procedurally, mostly in the form of some kind of level generator [19]. The generation of maps or virtual spaces (levels) can be carried out in four different ways [20]: designer-created spaces (mostly static), random generation, player-created and procedurally-generated game spaces. Any of these approaches can be generated by implicit or explicit description depending of the game context. In this paper, we focus on mixing the core plateaus of played-created and procedurally-generated approaches.

A more in-depth study on procedural content generation techniques in games, including MOBA, can be found in [6]. In this work, PCG games are classified following a six-layered taxonomy: bits, space, systems, scenarios, design and derived. Additionally, the authors show a majority of methods employed in procedural generation for commercial and prototype games.

In order to be able to include DCG in a MOBA map, it is necessary to outline what content will be changed in order to define the bounds of algorithms. There are many variables involved, which range in complexity from the simplest to the most convoluted. Players interact with many different systems that work together to deliver the expected experience of a MOBA game. Next, we present some relevant aspect to be considered for application in our algorithm.

3 Our methodology

In this section we explain in detail our approach to the map generation and CAS system.

3.1 The problem

MOBA games tend to use a predefined map or level in which players play each match. This map is carefully designed to provide interesting situations to players as they play, and give them tools to make strategic decisions

to win the game. Because of its careful design, these maps cannot be changed without analysing the consequences of these changes and as a result developers tend to not change the features of the map.

The proposal of procedurally generating a map so that it is different each time a new game starts is inherently a complex design issue that can only be addressed by the careful implementation of an algorithm capable of taking into account the strategic context of each feature of the map.

Focus on Character Adaptation Systems (CAS), MOBA games have an inherent problem directly related to the amount of content they provide to their players in the form of new playable characters. These characters are made to be unique and provide new and interesting ways of playing the game, but as the pool of characters rises, the developers incur in frequent balancing problems. These balancing problems mean that, for instance, a new character overshadows an existing one, making it less desirable to be played and forcing the developers to rethink the already existing character in a way that will make it more appealing for players to play with them. This problem is directly proportional to the amount of characters the game has, and there is no real solution to it.

3.2 Solution

Since there are so many different systems that could be created procedurally, we arbitrarily chose two that tamper with the very design of the game. The first one would be the map, where the players play the actual game, and the second one would be the characters that players use through a match.

3.3 Map generation

For map generation, we based our work using the maps provided in the DotA 2®[5], because maps are perfectly aligned with any standard MOBA game. Also, these maps offer all the mechanisms required to be used in order to achieve a well-balanced map that lets both teams of players stand

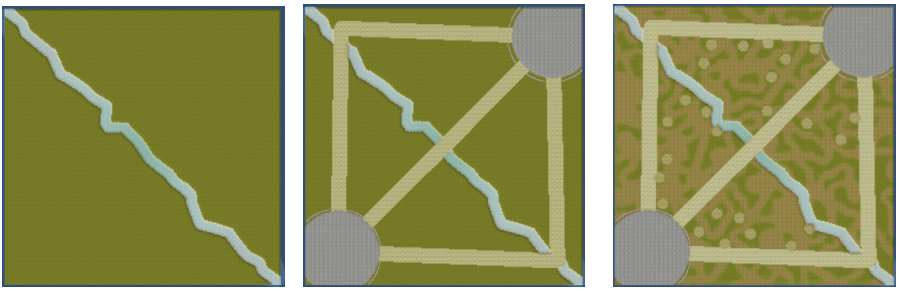
[5]http://dota2.com/

equally with respect to their chances of winning. Then, we created an algorithm which delivers a very similar experience to playing maps in DotA. Then, characters (players) are present, as well as some dynamic characters called creatures, equally distributed to each team.

Before the explanation, it is imperative to know how the original map is composed in order to alter different features without damaging its original purpose. This aspect is extremely important when dealing with this kind of game where players compete and must have equal chances of winning a match.

The map will be broken into the five different features that compose it: the River, Bases, Lanes, Jungles and the Design of the map. The following explains in detail each of them.

The river: The most outstanding element of the map is the river that traverses it, effectively dividing the map in two halves as shown in the Figure 2a. Each half of the map is considered to be one zone of each team. The strategic value of the river is huge because it is the quickest route to traverse the map. Also, as players transit this river frequently, it is a common place for skirmishes between enemy players.



(a) **(b)** **(c)**

Figure 2: Map composition: (a) a river, (b) two bases connected with lanes and, (c) a jungle inside it.

Bases: Players start each match on their corresponding Base. This is the safest place a player can have throughout the match, and it is where players frequently go back to heal and obtain new items that will strengthen their

character's abilities. The 2 bases in the map are placed as far as possible from each other (opposite extremes), since the ultimate objective of the game is to invade the enemy base, and destroy their main building.

Lanes: The map offers 3 lanes that form roads between each base (see Figure 2b). These lanes are guarded by special buildings that will attack enemy players, called Turrets. These Turrets must be destroyed in order to advance through the lane and eventually gain access to the enemy base. Players will spend the first minutes of the game trying to push forward their lanes in order to destroy enemy Turrets. However, creatures that belong to each team are constantly moving through the lanes with the objective of reaching the enemy base while attacking anything in between. Lanes play a major role in strengthening abilities of characters and this is why players spend most of their time in or near a lane.

Jungles: Jungles are placed between each lane and river, forming a total of four across the map, as Figure 2c shows. Each pair of jungles is on one side of the river. Jungles offer quick ways to travel between lanes and from a lane to the river in the form of intricate roads (i.e. ways for enemy players to invade the other zone). These are important for the overall pacing of the game since they allow players to move through the map without stepping on any lane, making it easier for players to move without being seen. They have a very high strategic value for players to coordinate attacks or even retreat. There are also special zones inside Jungles denominated Camps (in Figure 2c, the dots inside jungles), where neutral creatures (meaning they do not belong to any team) appear after certain periods of time. These creatures can be defeated by players in order to help their characters get stronger.

Design: In Lanes, outer Turrets (the turrets that are outside a base) have enough spacing between each other so that players can go through the Lane without being in range of any Turret. This is meant to offer a chance for a team to coordinate an attack on an unsuspecting player that is travelling through a lane. The River offers the fastest way to travel from lane to lane across the map, and because of this, players usually look for opportunities to catch enemy players that are moving through the river alone and attack

them.

The overall design of the map causes players that are in need of defending to have an advantage over their enemies, while players that are attacking are often in dangerous zones where they are the most vulnerable to attacks. Figure 3 shows inside the map, the danger zone coloured in red for the left team. This makes the game a high-risk high-reward situation, where players are forced to put themselves in danger in order to advance through a lane and eventually destroy the main building of enemy team.

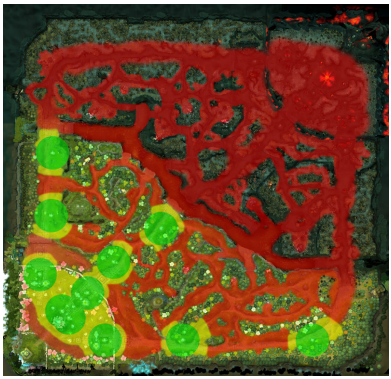


Figure 3: The red zone represents the danger zone for the left team in the map during an attack to the right team.

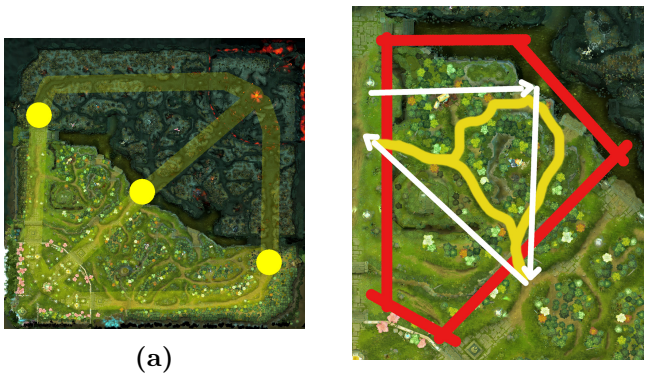
3.3.1 Algorithm Having analysed the map of DotA it was now possible to create an algorithm that was able to generate content following the previously defined guidelines. However, in order to have a proper point of comparison to how viable the generated map is, we need to define what a viable map looks like. In this case, the DotA map was used as the reference map, since it is already proven to be a viable map because it has undergone several design iterations.

Since we used DotA maps to determine the metrics, our algorithm is focused into the creation of a map, and it would only make sense to use this map as the perfect case scenario to compare all the generated maps. The closer a generated map is to the original one, the more viable it is to be used in a match.

The first step is to create the River in the empty map. Thus, the algo-

rithm uses a set of perturbed points along the diagonal of the map. Next, points are interconnected using a quadratic Bézier curve to create a set of points which represents the River. After its creation, the Bases would be placed in the corners of the opposite diagonal of the map. These Bases start off with a middle point which defines where the main building will appear, and then they are surrounded by walls in a randomly generated radius around the middle point. Note that the radius generated for both bases must be equal or very similar in order to avoid having a team with a much bigger Base than the other, which can prove advantageous/disadvantageous to the team.

After that, the Lanes must be created to interconnect the Bases. Three points are chosen along the river, one near the top-left corner of the map, one around the center of it, and the other near the lower-right corner of the map. These are then connected with the respective entrances of each base, forming roads that traverse the whole map. Figure 4a shows the result of applying our algorithm until now drawn over a background reference image.



(a) **(b)**

Figure 4: Previous results of the algorithm during its construction where (a) points are placed to interconnect lanes, and (b) roads were generated using the Breadth-First Search algorithm.

The last step consists of creating the Jungles. These are particularly more complex to create than the previous elements because of the amount of variables involved. A jungle must guarantee that a player can enter it

at least from 3 different places. Since the Jungles are placed between the river and 2 lanes, it is necessary to ensure that these places can be accessed from at least one entrance of the Jungle. Usually, the entrance to a Jungle from within a Lane lies between Turrets and more precisely, allows players to enter the Lane from the Jungle without being targeted by an enemy Turret.

Since the River is the fastest way to travel across the map, it is mandatory that no only lanes are accessible through it, but also the Jungles. Because of this, there must be an entrance to each Jungle that is directly connected to the River. To increase the dynamism of the generation of roads in Jungles, a ridged multifractal algorithm [2] was used to generate them. The viability of a Jungle would be evaluated by the distance needed to traverse any of its roads from each Lane to the River, and between Lanes. To evaluate this, a BFS (Breadth-First Search) algorithm was used to measure the distance of the available roads in the Jungle. Figure 4b shows that the BFS was executed 3 times (following the white arrows). In the figure, roads are developed to find a path (yellow color) from different point (initial points of white arrows).

After completing the creation of Jungles, the camps are added randomly wherever there is enough space, resulting in an unpredictable amount of camps in each map side. Because of this unpredictability, the sum of the camps on each side of the map is compared, and then the camp-wise viability of the whole Jungle setup is defined by how big is the difference in the amount of camps on each side of the map. The smaller the difference, the more viable the Jungle setup becomes.

To this point, a map is completely created, and the viability of all its parts has already been calculated. To measure the map as a whole, we need to take these viability points on each element of the map, and put them together to get a single value that will determine how viable is the map itself. In order to do that, a Multi-Objective Evolutionary Algorithm (MOEA) [21] is performed, because we need to perform a comparison of not only how viable one map is, but how viable is it compared to the other maps generated with it.

In the MOEA, two fitness functions were created to evaluate a map: one evaluates the length of each path in the Jungle and then averages it;

the other uses the number of camps on each jungle to ensure that a team would not have an advantage in this number on each side of the map. After the new map is created completely, we move to the system for character adaptation.

3.4 Character adaptation system (CAS)

This system is based on the idea of how to change in real-time the character attributes of players and how this can be used to preserve the original design of a MOBA game.

3.4.1 MOBA character design The characters in MOBA games are based in the Role-Playing Game (RPG) model [22]. This model defines a character through a set of values that determine its strengths and weaknesses. They have a Level value that works as a direct metaphor to how the learning process works, a character is able to earn experience points through different tasks. Then, reaching a certain value of these points their character is promoted to a higher Level, effectively making it stronger, which translates to a more experienced individual.

There are also a set of attributes that usually define aspects such as: amount of Hit Points (damage a character can receive before dying), resilience to damage, magical damage, and others. These attributes define how well a character performs in certain situations. In addition to level and attributes, characters are broken up into different classes, which define what set of skills this character will be able to have. These skills are abilities that a character can make use of to aid them in combat and perform different special actions.

In a MOBA game, each playable character follows the RPG character model. Usually the types of characters can be broken down into three different roles: Tank (able to sustain high amounts of damage and usually focused on protecting their team-mates), Damage (focused on dealing high amounts of damage to the enemy), and Support (flexible characters that focus on special abilities and behaviours that help their team). Each character is designed to bring something unique to the game. Besides the roll they can fulfil, they are made to be fun to play in a unique way that is not

like any other character in the game, just so that the game itself has more variety for players to choose what they want to play. Because of this, each design of character is carefully thought to reinforce their gameplay style.

As mentioned before, there is a problem related to the amount of playable character a MOBA game provides, and it is that the addition of a new character forces the developers to revisit older characters in order to make them more appealing for the players. As an attempt to solve this problem, or at least lessen it, the proposed Character Adaptation System will change the way characters are constructed in the game.

Normally, characters are created with a unique set of skills that define their gameplay style, in this case for the CAS, when beginning the game, all characters are equal. Players then have to decide which skills will compose their character. This changes the way new content is added to the game. Instead of adding new characters, developers need to add new skills that players can choose from the skills pool at the beginning of the game. This will make the developers focus their design efforts on skills instead of complete characters, which consists of a more complex set of variable to deal with in matters of design.

The CAS is expected to not solve the balancing problem, but to at least shorten the re-balancing cycle developers are forced to go through each time new content is added.

3.4.2 Changing the RPG paradigm In a normal RPG game, characters strength comes from the values of its attributes, which increase each time the character gains a level. This process is entirely discrete, where the player is not able to experience any increase in their strength until the needed amount of experience points has been gained and the character gains a level.

In our approach, the procedural modification of a character starts by moving the discrete nature of this *leveling-up* system towards a continuous behaviour. This is done to continuously reward the player for their efforts and to give an organic feel to the overall gameplay of the game. This system would be in charge of understanding how the player wants to play their character, and modifying continuously its attributes to correspond to its play style.

3.4.3 Implementation All characters in a MOBA game start a match with two common actions they can perform: walking and basic attack. Walking lets the character move where the player wants to go, as expected. Basic attack can happen whenever a player chooses to attack an enemy without using any of their skills.

An attack inputs to the Character Adaptation System (CAS) whether the attack successfully damaged an enemy, and how frequently is the character using it. By doing this, the CAS algorithm can start interpreting what is the goal of the player and increases its attributes accordingly to encourage its play style. If the player is trying to use the attack as frequently as possible, the CAS may reward him with increased attack speed, reducing the time to wait before performing the next attack.

The CAS is a system that focuses entirely on gathering real-time information from a player's behaviour and actions, transforming them into attributes increased for their character. This feedback loop is presented in Figure 5, which keeps a character constantly changing towards what the player intends to play.

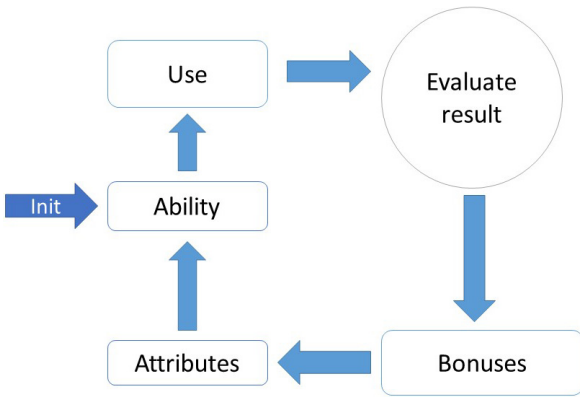


Figure 5: Scheme of the feedback loop to transform behaviours in attributes.

The attributes usually bring to a character benefits such as increased amount of hit-points, which represents how much damage can the character sustain before being defeated. Then, the goal is to increase the values that directly increase the effectiveness of the character by letting it sustain more damage and/or use more abilities.

The increase that an attribute will receive depends entirely on each skill involved. This is due to the fact that each skill has different scenarios for what *effective usage* means. For instance, effective usage for a fireball might be hitting the intended target, which would award a bonus of X points to the Intelligence attribute, but if it doesn't hit the intended target, then the Intelligence bonus would be $X/3$.

To further improve the benefits of the CAS feedback loop, abilities have a scaling factor that depends on certain attribute. For instance, an ability that lets the character throw a magic fireball (*i.e.* a power) to the enemy, delivering a certain damage to the target within a certain radius around the impact point, can have its behaviour modified by the intelligence attribute of the character. As shown in Figure 6, the higher the intelligence of the character, the higher the damage and radius of explosion of the fireball.

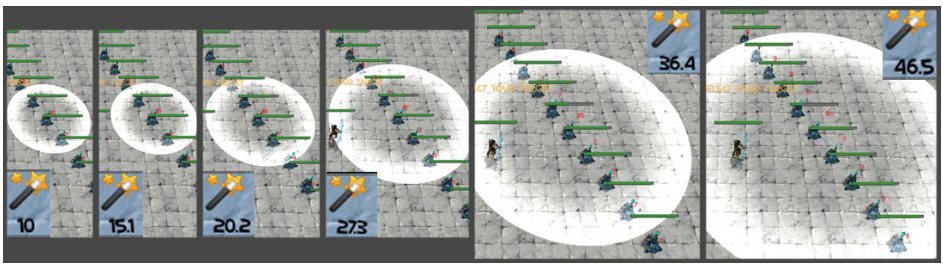


Figure 6: Example of the behaviour using a fireball power in time which shows the increase of attack's range.

Thus, any skill can be implemented to depend on certain attributes of a character. Therefore, it could be better or worse in certain scenarios depending on the focus of the player. The algorithm used by the CAS is composed by a set of systems, where each system is in itself, a skill that a character can use. These systems start working as soon as the player enters in game.

During a combat in the game, a player will input different actions for the character to perform in order to achieve their goal. This is calculated as a sum of all bonuses per attribute. In this way, each attribute will be increased to sum up all bonuses associated to this attribute. The final result is the final value of an attribute, to all attributes. Also, this value

represents the new value after the combat phase ends and will increase immediately the abilities of the character and its effectiveness in combat.

At the beginning of a game using CAS, each character starts the game with identical attributes. Players have to choose which set of skills they want their character to have, and throughout the game they will develop these abilities to adapt to their game-style to ultimately help them in achieving victory against the opposing team.

During a match, players will try to use strategies to attack their enemies in order to have an advantage. In this way, it is possible to deduce that the match is auto-balanced since players constantly react and counter-attack their enemies actions. This auto-balance addresses a concurrent problem in MOBA games where characters are often changed and calibrated because they are too powerful or weak.

4 Results and discussion

The Map Generation System and the Character Adaptation System were implemented together in a prototype of a MOBA game made specifically for this project. In this prototype, up to 4 players could join a game and play together on-line, which is the method used to perform some of the tests described next.

4.1 Testing the map generation

The algorithm used to generate the map was executed 20 times, and each of these results was compared in order to determine the average viability of the generated maps. Figure 7 shows the visual result of the maps constructed.

The obtained results reflect the variable nature of the genetic algorithm used to generate each map. Because of the inherent behaviour of this kind of algorithms, it is not possible to determine if the result will be viable prior to executing the algorithm. Therefore, in cases where the generated map was not a viable map, it was nevertheless chosen to be used in the game.

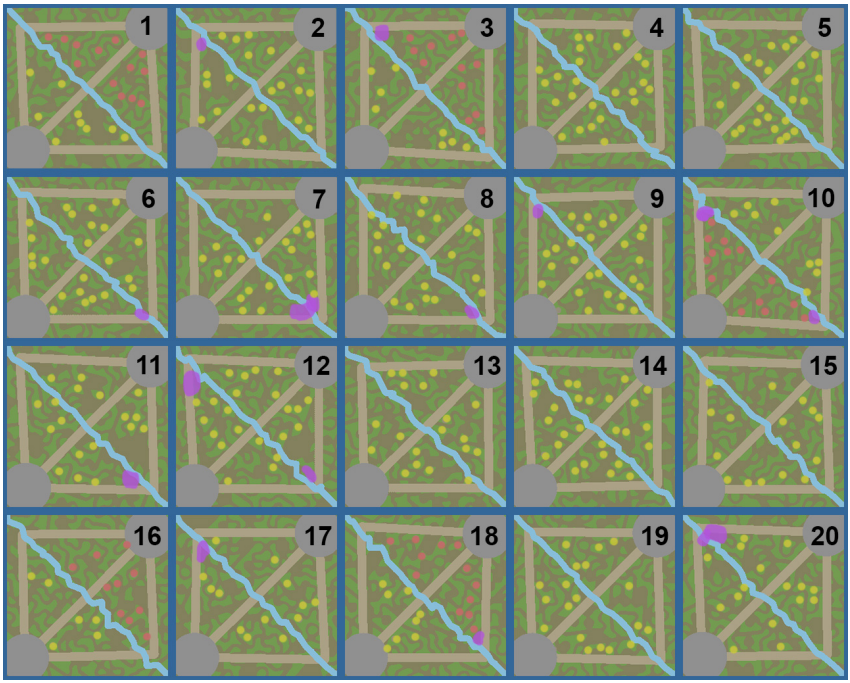


Figure 7: An example of 20 maps generated using our algorithm. Purple regions indicate the bases, beige region the lines, brown areas the jungle's path. Also, green color represents the forbidden area to walk, red color is "excess" camps on that side, yellow indicates the camps, and purple ones road problems between line and river.

In samples 2, 7, 8, 9, 11, 12, 17 and 20 of Figure 7, there are irregularities that could hinder the gameplay in certain zones of the map because some of the jungles were not formed properly. As for samples 3, 10 and 16 the amount of camps and badly formed jungle roads account for a disrupted gameplay that is sure to provide a major advantage to one of the teams. This experiment was repeated 5 times in order to obtain an average, obtaining similar results.

It is important to note that the viability of a map lies on its similarity to the original DotA 2 map, which was taken as the best case scenario. The ideal tests would be letting a group of players play each map for a certain time and determine from the outcome of each game what maps turned out to be more viable, but these tests were far beyond the scope of the project

since the testing environment consisted of a prototype MOBA that was not suited for long periods of testing. Therefore, geographical metrics had to be used to determine the viability of the generated maps.

4.2 Testing the CAS

To determine if the basic expected behaviour of the CAS was working, a test was performed where a group of 4 players played a match of the prototype game on-line. The time window for this test was nearly 10 minutes, and the values analysed were related to a single player throughout the entire match.

Information such as time spent in the different movement states of a character, such as: Idle, Moving, Dead, In-Combat and In-Combat-with-other-players; was gathered to determine in which the player spent the time (see Figure 8). In this case, the time in combat was nearly as much as the time spent, and it is important to also take into account that because of the map size, players are bound to spend long periods of time traversing it without encountering enemies.

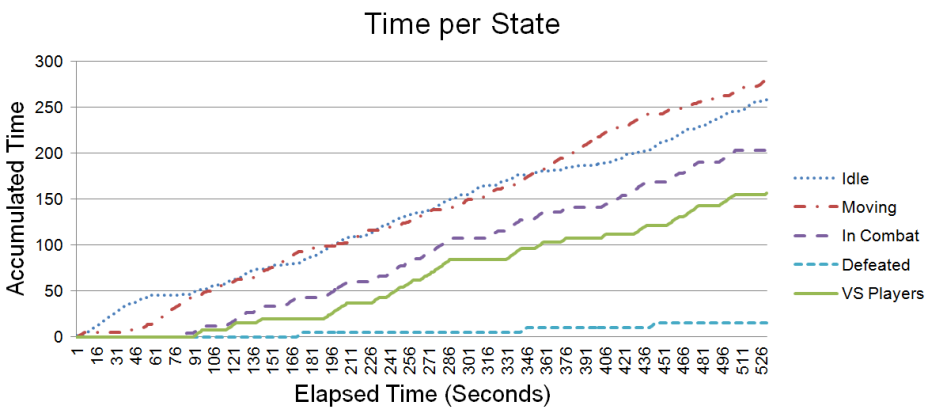


Figure 8: Graph of the accumulated time of a player during an average match.

Also, each time an action not related to movement was performed it was counted to determine the relation between the usages of a character skills and the improvements provided by the CAS, resulting in the curve

shown in Figure 9 that there was a little amount of attributes gains over the testing period. This curve reflects all the periods of time a player got in and out of combat, during this time the amount of skills used is not important as it will all account for only one attribute bonus gain that will sum up all the results of the battle.

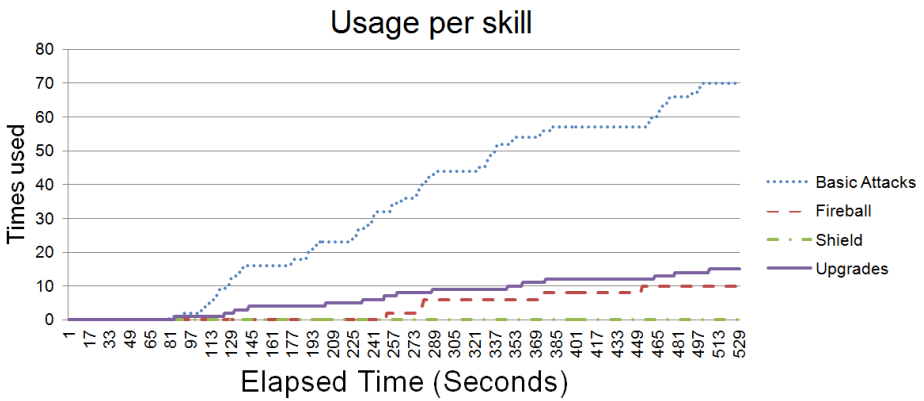


Figure 9: Relation between skills and their usage during an average match.

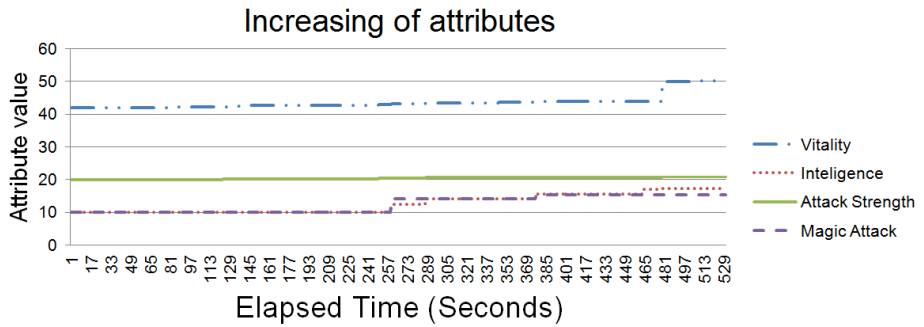


Figure 10: An example of the increasing of attribute values according its usage.

Lastly, information on the specific attribute gains was gathered to determine that each of the corresponding attributes of a character that was related to the skills provided, was increased accordingly to their usages. This was to be expected since this is part of the CAS cycle to enhance

an ability by making use of it, and increasing accordingly the attribute it depends on, as shown in Figure 10. This feature reinforces the problem of this system which consists of implementing skills that have a very complex underlying design to provide the variety and flexibility the system was conceived to have. If all the skills implemented with this system only work based on a single attribute, the CAS would not be able to provide effectively a player with tools to develop their character to play any way they want.

Note that for this particular project it is not possible to confirm that the CAS indeed solves the balancing issue, since this problem only appears when the game is played by an already established community of players. Since this was a prototype, we can only base the results on the theory behind the system created to address the balancing issue. Although the results cannot support this, theoretically, the CAS does not actually solve the balancing issue mentioned, but instead moves its focal point to the addition of new abilities to the game, instead of new characters, turning the original balancing problem into a different one.

The new question that arises from this is: How much could the CAS actually alleviate the balancing problem of a MOBA game if it changes the set of contents it is related to?. If applying re-balancing to an existing character is indeed harder than doing this re-balancing to a single skill, then the CAS would be an alternative to diminishing the balancing problem of MOBA games.

The overall results show that both the map generator and the CAS worked as expected. With regards to how attributes of character are modified throughout the game to match the gameplay style of a player. However, the objective of the CAS was to provide a more flexible structure to add content to the game and possibly avoid having to rebalance existing content, which besides dynamic attributes also includes letting the player construct their character with a set of skills that they can pick from a skills pool defined by the developers. Whether the CAS solves or lessens the balancing problems can't be answered from the data gathered in the small environment the tests were performed. To be able to confirm practically how the CAS affects the balancing issues, it would have to be implemented in a commercial MOBA game with an already defined community of players.

5 Conclusions

In order to bring variability to the content provided by developers in a MOBA game, the two systems mentioned previously were implemented to be able to generate a map from scratch in which players could play the game. Also, to be able to adapt in real time the attributes of the character played by a player to correspond its strengths with the play-style of the player. These systems still have a long way to go in matters of optimization and design, but they prove that it is indeed possible to introduce DCG to an inherently static game such as a MOBA and still preserve its core design and gameplay.

Future improvements for the map generation system could include features such as a variable number of lanes, including the possibility of producing a map that has no lanes at all and that, besides the bases, it would consist entirely of jungles. Moreover, optimizations could be implemented in order to be able to perform more iterations of the MOEA algorithm to have better chances of generating a viable map.

Also, the character adaptation system presents a design challenge in itself. It was created in order to eliminate the constant rebalancing developers have to constantly do when introducing new content to the game, but this problem is not entirely eliminated by this systems but instead it is forwarded exclusively towards the pool of skills players have access to. Besides the problem related to balancing the skills, the level indicator is eliminated due to the pseudo-continuous nature of the progress of a character throughout a game. In order to address this issue, an implementation consisting of representing each skill a character possesses as a piece of equipment that stands out easily. This piece would represent a specific skill this character has and also, its visual features would indicate how strong this skill is for this specific character. The reason behind this system lies in the fact that by eliminating the character level, the explicit strength indicator is lost and therefore there must be another way to let a player know instantly how strong an enemy is.

Acknowledgements

The authors would like to thank the reviewers for their comments that help improve the manuscript. Also, authors gratefully acknowledge use of the services and facilities of the Computer Graphics Center at the Central University of Venezuela, as part of its research fields.

References

[1] B. Mandelbrot, *The Fractal Geometry of Nature*, 1st ed. W. H. Freeman and Company, 1982. [Online]. Available: http://adsabs.harvard.edu/abs/1983whf..book.....M 97

[2] D. Ebert, F. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing and Modeling: A Procedural Approach*, 3rd ed. Morgan Kaufmann, 2002. 97, 106

[3] Y. Parish and P. Müller, "Procedural Modeling of Cities," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. ACM, 2001, pp. 301–308. [Online]. Available: http://dx.doi.org/10.1145/383259.383292 97

[4] J. Kenwood, J. Gain, and P. Marais, "Efficient Procedural Generation of Forests," *Journal of WSCG*, vol. 22, no. 1, pp. 31–38, 2014. [Online]. Available: https://otik.uk.zcu.cz/handle/11025/11896 97

[5] R. Geiss, "Generating Complex Procedural Terrains Using the GPU," *GPU Gems 3*, vol. 1, pp. 7–37, 2007. 97

[6] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural Content Generation for Games: A Survey," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 9, no. 1, pp. 1:1—-1:22, 2013. [Online]. Available: http://dx.doi.org/10.1145/2422956.2422957 97, 100

[7] J.-D. Génevaux, E. Galin, E. Guérin, A. Peytavie, and B. Beneš, "Terrain Generation Using Procedural Models Based on Hydrology," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 143:1—-143:13, 2013. [Online]. Available: http://dx.doi.org/10.1145/2461912.2461996 98

[8] G. Smith, "Expressive Design Tools: Procedural Content Generation For Game Designers," Ph.D. dissertation, University of California, 2012. [Online]. Available: http://escholarship.org/uc/item/0fn558gq#page-4 98

[9] C. Fencot, J. Clay, M. Lockyer, and P. Massey, *Game Invaders: The Theory and Understanding of Computer Games*, 1st ed. Wiley-IEEE Computer Society, 2012. 99

[10] W. Muehl and J. Novak, *Game Development Essentials: Game Simulation Development*, 1st ed. Cengage Learning, 2007. 99

[11] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *IEEE Symposium on Computational Intelligence and Games*, 2007, pp. 252–259. [Online]. Available: http://dx.doi.org/10.1109/CIG.2007.368106 99

[12] E. Hasting, R. Guha, and K. Stanley, "Automatic Content Generation in the Galactic Arms Race Video Game," *IEEE Transactions on Computational Intelligence and AI Games*, vol. 1, no. 4, pp. 245–263, 2009. [Online]. Available: http://dx.doi.org/10.1109/TCIAIG.2009.2038365 99

[13] C. Browne, "Automatic Generation and Evaluation of Recombination Games," Ph.D., Queensland University of Technology, 2008. [Online]. Available: http://eprints.qut.edu.au/17025/ 99

[14] J. Togelius and J. Schmidhuber, "An Experiment in Automatic Game Design," in *Proceedings of the IEEE Symposium on Computational Inteligence and Games*, 2008, pp. 111 – 118. [Online]. Available: http://dx.doi.org/10.1109/CIG.2008.5035629 99

[15] N. Shaker, G. Yannakakis, and J. Togelius, "Towards Automatic Personalized Content Generation for Platform Games," in *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2010, pp. 63–68. [Online]. Available: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE10/paper/viewFile/2135/2546 99

[16] K. Compton and M. Mateas, "Procedural level design for platform games," in *Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 2006, pp. 109–111. [Online]. Available: http://aaaipress.org/Papers/AIIDE/2006/AIIDE06-022.pdf 99

[17] A. Baskar, G. Bradway, J. Kennington, A. Hathaway, and N. Cumming, "Dynamic Music Generation," in *Proceedings of the Conference on Machine Learning*, 2013. 99

[18] A. Smith and M. Mateas, "Answer Set Programming for Procedural Content Generation: A Design Space Approach," *IEEE Transactions on Computational Intelligence and AI Games*, vol. 3, no. 3, pp. 187–200, 2011. [Online]. Available: http://dx.doi.org/10.1109/TCIAIG.2011.2158545 99

[19] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin, "Polymorph: Dynamic Difficulty Adjustment Through Level Generation," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ser. PCGames '10. ACM, 2010, pp. 11:1—-11:4. [Online]. Available: http://doi.acm.org/10.1145/1814256.1814267 100

[20] M. Nitsche, C. Ashmore, W. Hankinson, R. Fitzpatrick, J. Kelly, and K. Margenau, "Designing Procedural Game Spaces: A Case Study," in *Proceedings of the FuturePlay*, 2006. 100

[21] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. Nagaratnam, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32–49, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.swevo.2011.03.001 106

[22] A. Tychsen, "Role Playing Games: Comparative Analysis Across Two Media Platforms," in *Proceedings of the 3rd Australasian Conference on Interactive Entertainment.* Murdoch University, 2006, pp. 75–82. [Online]. Available: http://dl.acm.org/citation.cfm?id=1231906 107