INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Adaptative Map Generation For Turn-based Strategic Multiplayer Browser Games

Gonçalo Duarte Garcia Pereira

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Júri

Presidente:	Joaquim Armando Pires Jorge
Orientadores:	Pedro Alexandre Simões dos Santos
	Rui Filipe Fernandes Prada
Vogal:	Daniel Jorge Viegas Gonçalves

October 2009

Acknowledgements

First of all I am very grateful for the invaluable help, feedback and commitment that both my advisor Pedro Santos and co-advisor Rui Prada had towards me and the correct evolution of this work.

I would also like to thank Almansur for the opportunity and continuous support given. I am especially thankful to Marco Quinta for the precious communication we maintained where we exchanged many information and ideas.

Finally I would like to thank Luis Landeiro from PDM&FC for the important supervision, availability and assistance offered to the technical part of this work.

Lisboa, October 2009

Gonçalo Duarte Garcia Pereira

This work would not be possible without the amazing and continued support from my parents throughout not only this thesis but all my studies. Finally to my brother, always committed to to discuss and help on all the subjects related to computer science.

Resumo

Esta tese trata os problemas da Geração de Mapas Aleatórios que sejam estratégica e visualmente interessantes para *Massively Multiplayer Online Games (MMOG)* de estratégia, baseados em turnos. Inicialmente é apresentado o resultado de uma pesquisa sobre o trabalho relacionado com este tema. As metodologias utilizadas actualmente são revistas e as técnicas utilizadas por alguns MMOG analisadas. De seguida descrevemos um método para criar mapas estratégica e visualmente interessantes onde utilizamos um jogo MMOG como caso de estudo. O jogo utilizado é o Almansur que pertence a um subgénero de MMOGs, os *turn-based strategy massively multiplayer browser games*(TSMBG). O método apresentado utiliza mapas dinâmicos que são expandidos de acordo com a quantidade de entradas de jogadores e que se adapta às escolhas dos jogadores. Este elimina limitações do game designer e promove o desenvolvimento de jogos escaláveis e que focam o aspecto estratégico. Permite ainda mitigar problemas de balanceamento existentes nos TSMBG. Por fim apresentamos alguns resultados do nosso método e as conclusões obtidas relativamente ao *gameplay* experienciado pelos jogadores e à escalabilidade do método desenvolvido.

Palavras-chave: mapa aleatório, geração de mapas, balanço do mapa, colocação de jogadores, crescimento de mapa, geração procedimental

Abstract

This thesis addresses the problems of Random Map Generation which are strategically and visual interesting for turn-based strategy Massively Multiplayer Online Games (MMOG). First we present the results of the research regarding works related to this subject. Previously used methodologies are reviewed and the techniques used in some MMOGs analyzed. Next we describe a method to create strategically and visually interesting game map environments with a MMOG as a case study. The game used is Almansur which belongs to a sub-genre of the MMOGs, the turn-based strategy massively multiplayer browser games (TSMBG). Our method uses dynamic maps which expand according to the flow of players' subscriptions and adapts to their choices. It also eliminates current game designer limitations and promotes the development of scalable strategy focused games. Furthermore it mitigates balancing problems in TSMBG. Finally we present some results from our method and the conclusions regarding the gameplay experienced by players and the scalability of the developed method.

Keywords: random map, map generation, map balance, player placement, map growth, procedural generation

Index

List of Figures

List of Tables

Acronyms

CRUD Create, Read, Update and Delete

FOW Fog Of War

FPS First Person Shooter

MMOG Massively Multiplayer Online Game

PCG Procedural Content Generation

PSS Player Specific Size

NTS Neutral Terrain Size

NZS Neutral Zone Size

ORM Object-relational Mapping

PO Places Order

PTC Player Terrain Composition

RTS Real Time Strategy

RPG Role Playing Game

TSMBG Turn-based Strategy Massively Multiplayer Browser Game

1 Introduction

Internet-based gaming has grown tremendously in the last few years and it is possible to find many games from different genres [1]. Among these are the browser games which are defined by only requiring internet users to have a browser installed in their computer in order to play. Some are multiplayer and take advantage of the internet's widespread nature to link many players into a group game experience. Communities in these games are large and can vary from several hundreds to thousands of players gathered for a collective game experience[2].

With a few exceptions multiplayer browser games fall into two main categories, namely Role-Playing Games (RPG) and strategy games[3]. In strategy browser games, the player usually starts with a single "village", "castle", "territory" or "planet", and then must balance economic development with military development, in order to gain dominance over its neighbours. To increase variety, each player can choose some special traits that give him specific advantages and disadvantages. These trait sets can be represented as "races" or "tribes". Strategy Browser Games have been evolving for more than a decade into more complex and graphically appealing games. A way to see this evolution is through the interfaces presented to players to access the game data and provide different game views such as a map. Some started as text-based games with no representative graphics whatsoever (1996 - Earth: 2025[4]), then others appeared with simple maps that are used as a simple referencing environment without any actual relevant influence in the game (2003 - Travian[5]) and in present time there are some which use fully detailed maps that influence the game (2007 - Almansur Battlegrounds[6]).

In strategy games, player location, terrain type and richness in resources influence the strategic balance and player challenge which then affect the player's experience. Beyond the

[1] http://www.gamasutra.com/php-bin/news_index.php?story=23954
[2] http://en.wikipedia.org/wiki/History_of_massively_multiplayer_online_games
[3] http://en.wikipedia.org/wiki/List_of_multiplayer_browser_games
[4] http://games.swirve.com/earth/
[5] http://www.travian.com/
[6] http://www.almansur.net/

map's strategic influencing characteristics there is also the visual effect the map has. Therefore, the map's variety of visual characteristics such as shape and detail also play an important role regarding the player's experience and its desire to continue playing. A game where the map always has the same shape risks causing boredom on players that replay the game. Another factor which creates interest is the options the players have available to explore and create different gameplay experiences. Currently many strategy multiplayer online games are using a model where a map is created to accommodate a given fixed number of players and then they are placed as they subscribe in a specific ordered way which varies with the game. In these maps there is currently a tradeoff between the players' choices/map complexity and the scalability for the multiplayer context. The most detailed and strategically complex maps are hand-made and thus hard to scale for many players and the generated maps are usually very simple and strategically poor. Here the map is used just as a referencing environment for player placement.

In our work we focused on the generation of maps for strategy games in an online context which use complex environments (that deeply influence player experience) and that can be played by hundreds or thousands of players. Based on our goals we then created a map generation model which has both the advantages of the hand-made and procedurally generated maps that properly adapt to the TSMBG(Turn-based Strategy Massively Multiplayer Browser Game). This uses a procedural generation method which has the ability to create interesting maps but depends on the game designer for a specific and detailed parameterization in order to be able to add the map detail. Furthermore, in order to adapt to the TSMBG context we generate maps dynamically as players subscribe, thus enabling the game a greater variety of options for the player to explore since the map terrain is generated to accommodate the player's choices. The maps created in this way evolve according to the players and their (race) choices while unbounded by the uncertainty of when will new players join, what choices will they make or how many will join. Our maps are not limited by either the game complexity or game scalability, but simply enable the best strategic and diverse environment the game designer can provide for a given game.

In the following sections we start by reviewing some of the current techniques of map creation and existing multiplayer strategic online games. Then we present our approach to dynamically generate adptative maps. We first present how we procedurally created the desired complex game map environments statically in an automated way. Then we present how we expanded this method to be able to scale for a massive multiplayer context and dynamically generate the

maps by adding players as they decide to join enabling us to take scale the complex environments while preserving the user's ability to choose their race.

The problem we solved in this work originated from Almansur Lda's need for a process which could provide balance, challenge and flexibility for a massive multiplayer context in complex game map environments. However, this problem is common to several TSMBG and thus we generalize it and its solutions for this sub-genre. Then as a case study for our solution of an adaptative map generation process we used the game Almansur Battlegrounds. It is a "strategy game of politics, economy and war, set in the early middle ages and, in some scenarios, in a fantasy world". Following the concrete implementation for our case study we present some examples of what we achieved with our approach and the practical results obtained. Finally some discussion, future work and conclusions are presented.

1.1 Problem and Work Goals

This thesis addressed the problem of adaptative map generation of strategically and visual interesting game maps in TSMBG. Several problems arose from this context regarding the online and the massively multiplayer nature. The online nature brought the problem of unsynchronized player subscriptions and player tolerance to waiting times. The massively multiplayer added the requisite of scalability to the methods created.

1.1.1 Research related work

In order to address our problem we first researched the related work and tried to answer the following questions:

1. How can we procedurally generate interesting random game maps?
2. How can we include game balance in random generated maps?
3. What solutions exist to address map growth and delayed player entries for TSMBGs while maintaining the game balanced and strategically interesting?

This research allowed a good understanding of the subjects involved in game maps and its generation. It was a knowledge basis for the development of our solution to the problems of

adaptative map generation.

1.1.2 Development of a solution

We developed a generic solution for the problem of adaptative map generation of interesting and balanced maps in several contexts. Then we used a game suitable to be a case study and implemented a concrete solution for it. The solution took into account what was learned from the related work.

1.1.3 Evaluation

We also evaluated the performance of our method regarding its goals. To do this we gathered player feedback and in-game data to evaluate the method developed.

1.2 Contributions

With this work we contributed in several ways to the research area of map generation for games. First we provided an important comparative study of the advantages and disadvantages of the processes used in map generation and an analysis of the current state of the art in MMOGs regarding this subject. Next, we provided an adaptative map generation method that creates interesting maps for complex online game environments. Especially important was the innovative dynamic map generation methodology for the massively multiplayer game context which was presented on October 30 at the International Conference on Advances in Computer Entertainment Technology 2009, in Athens, Greece, with the paper "Self-Adapting Dynamically Generated Maps For Turn-based Strategic Multiplayer Browser Games". Additionally, the solution created has been developed and integrated with Almansur Battlegrounds where it has been publicly released to players. Finally, based on our results analysis and discussion we also contributed to the understanding of the impact a dynamic map generation approach has on players/game designer and what problems it arises as well as possible solutions for them.

2 Related Work

2.1 Map generation

In this chapter we will describe some of the methods used for the generation of maps for videogames. The methods presented will not only focus on 2D exclusively but also 3D as many of the methods which are used on 3D can be adapted to 2D and the information on them is more common in today's community. Many 2D games add the third dimension by creating terrain properties which represent for example the altitude and its influence on the game. Even though this is not a graphical 3D representation the game mechanics are affected through the data which represents the altitude. For instance if some troops are trying to cross a mountain range in a 3D map they walk slower and we see them walking slower. The same happens in a 2D game even though the visual feedback is smaller, the troops walk less terrain cells per unit of time.

2.1.1 Terrain representation

Terrain representation serves two major purposes, the storage of the map and the presentation of the map in the game. A terrain representation is the conceptual model created to hold information regarding a terrain map.

2.1.1.1 Heightmaps

"*A heightmap is a simple 2D picture generated in black, white, and the 254 shades of gray in-between*" (Feil & Scattergood 2005a). In this picture the height of the terrain varies with the shade, where white represents the maximum value and the black represents the minimum value. This image is commonly supplied with another file containing metadata about the maximum and minimum height, enabling to scale the shades to represent height as needed.

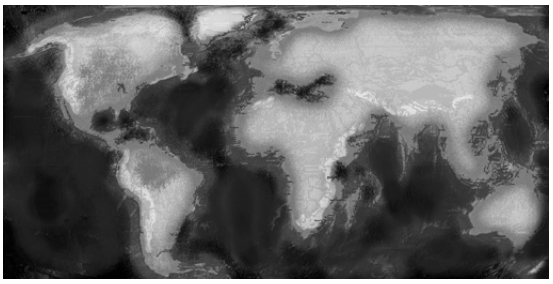

Figure 2.1: Heightmap example

Usually these maps are created through an automated generation process or through real world scanning. For games the most common approach is to first generate and then import them into the game engine. The detailing process of these maps is time consuming but the creation of large changes can be easy as the maps are images which are easily modified with image editing programs such as Adobe Photoshop[1]. In figure 2.1[2] we present an earth heighmap example.

2.1.1.2 Tiled terrain

Tiled terrain is a way of representing terrain based on a set of defined tiles of terrain. From this set a level designer can create any terrain he desires by combining the several types of tiles. Usually these tiles have a well defined type of terrain with fixed characteristics and a clear image to show it. A very simple example of a set of tiles like this would be composed of: plains, forest, mountains and water. A more complex and real use of such a set is seen in Battle for Wesnoth[3]. The base unit of a tiled terrain is a tile and it can be represented with different polygonal forms, next we will describe the most common forms used in games.

Squares It is probably the most common form of tiles used due to its ease of use. It can be used with a cartesian coordinate system where the axes are orthogonal. The square shape also maps without artifacts (imperfections) in the screen, an example is illustrated in figure 2.2 (a). However, when considering the cardinal directions commonly used in object or unit movement in games, they map well but provide higher diagonal movement distances than horizontal or vertical movements.

[1] http://www.adobe.com/products/photoshop/photoshop/
[2] http://www.galciv2.com/mods/gc2modsdoc.htm
[3] http://www.wesnoth.org/

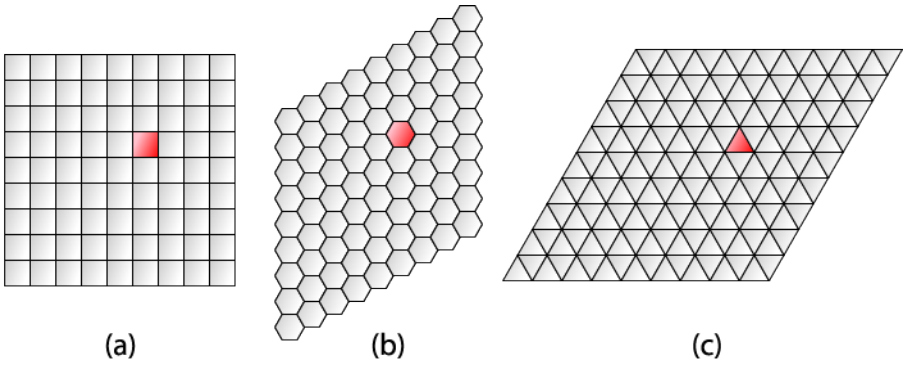

Figure 2.2: Square grid in (a) and vertically aligned hexagonal grid in (b)

Hexagons Also a very popular way of representing tiles, especially in board games like Settlers of Catan[4]. Their biggest advantage is that they have equal center to center distances to all their neighbours contrary to the squares which have greater distances, from center to center, to their diagonal neighbours and thus map well with the cardinal directions of movement. Within hexagonal representations there are two ways of using them, vertically aligned or horizontally aligned. Hexagons have however the disadvantage that their coordinate system is more complex than the square system and might require a bigger overhead in logic processing for the representation[5]. An example of this grid system is illustrated in figure 2.2 (b).

Triangles They are very common in 3D graphics but are rarely used in games as a basic terrain unit. Their main disadvantage is the small area combined with a large perimeter which difficults the placement of game pieces or objects. Furthermore, they only provide three directions of movement which is unfamiliar and do not map well for games which need unit or object movement in at least four or more directions, like in the common cardinal system. An example of this grid system is illustrated in figure 2.2 (c).

2.1.1.3 Comparison

The heightmap representation is a good way to save data about a terrain because it provides an easy way to transport and quickly access the terrain data (since it is an array of pixels containing height values). The tiled terrain is a more visual representation even though it can

[4]http://www.mayfairgames.com/
[5]http://www-cs-students.stanford.edu/ amitp/game-programming/grids/

also be used to store terrain information. There is no best method to represent terrain just different choices where each one has its benefits and disadvantages.

2.1.2 Terrain Height Synthesis

One of the most important steps in map generation is the terrain height synthesis since subsequent steps (like feature creation and resources distribution) build over its information[6]. In the literature this topic is sometimes mixed with others like the creation of terrain features, but our focus here will be the height generation. There are several ways to create the height of a terrain. The first and most straightforward is the manual creation of the map. The second is through procedural content generation (PCG) methods.

Usually in the manual approach it is common that other attributes like resources and features are definable too. This process requires a great deal of work by the map creator as every detail of the map requires his attention(Feil & Scattergood 2005a). It is usually done with an application which allows the creator to build the map by means of a pre-set of tools/functions. These often consist in a set of tiles for the various types of terrain and an object set for the objects that are available in the game. The map also has a size defined by the level designer. The creation of the map is then done in a painter fashion way, where the map is the canvas (initially empty) and the level designer directly places the objects/tiles. This kind of map creation is many times a job done by a person with some artistic skills. Examples of this are the Starcraft[7] Campaign Editor and the Battle for Wesnoth Map Creator[8].

Another way to create a map is to use a PCG methods applied to maps which through a set of pre-defined parameters proceduraly generates them without further inputs as observed in (Brosz, Samavati, & Sousa 2006) or in the stylized map generation (Prachyabrued, Roden, & Benton 2007). These methods usually use a heightmap that represents the heights of every cell of a terrain. Once having this file which describes the terrain, building it is just a matter of data manipulation. The procedural generation of heightmaps can be done using a technique or a combination of several from which we highlight: Perlin Noise, Diamond Square Division and Voronoi Diagrams. We will also describe a different approach which still uses heightmaps as a

[6]http://dwarf.lendemaindeveille.com/index.php/World_generation
[7]http://www.blizzard.com/us/starcraft/
[8]http://www.wesnoth.org/wiki/WesnothMapEditor

form of representing the terrain but has a semi-automatic generation process which uses real world data. Other techniques exist but they either generate poorer results or are similar to one of the highlighted. These have been documented in many works such as Quadrant Average Analysis[9] in (Lecky-Thompson 2000), using genetic algorithms (Ong, Saunders, Keyser, & Leggett 2005) or even fractal based approaches as in (Musgrave, Kolb, & Mace 1989), (Stachniak & Stuerzlinger 2005) and (Laeuchli 2001) or even fractals combined with different approaches such as Fault Formation (Shankel 2000a), Voronoi Diagrams (Shirriff 1993), Midpoint Displacement (Shankel 2000b) or Particle Deposition (Shankel 2000c).

2.1.2.1 Perlin Noise

The Perlin Noise is widely used since it was introduced (Perlin 1985) and then improved (Perlin 2002). It is a pseudo-random number function which has a fractal nature. In our case the Perlin Noise function will generate values based on the two parameters of a 2D heightmap, x and y coordinate. Based on these parameters and given the same Perlin Noise function the results would always be the same hence the same heightmap. The Perlin Noise function for map generation is created by adding several correlated noise functions. So the function can then be parameterized by: noise function, persistence, octaves and interpolation type; to achieve different results in our heightmap. Each parameter affects the Perlin Noise in a different way:

Noise function it is any pseudo-random number generator, the base noise function for all the noise functions that compose the Perlin Noise function;

Persistence the noise functions that compose the Perlin Noise usually have a correlation in terms of amplitude and frequency, and this correlation is commonly done in octaves. This value specifies the amplitude of each frequency with the following formula: $amplitude = persistence^{octave\_number}$.

Octaves in Perlin Noise it is common that the individual noise functions have a relation of $2 * frequency$ and $\frac{amplitude}{2}$. This relation is known as octaves, and the number of octaves specifies the number of functions with this correlation that compose the Perlin Noise function. From the number of octaves we can also determine the set of frequencies we will be adding in our Perlin Noise, where for each frequency we have $frequency = 2^{octave\_number}$

[9] http://www.gamasutra.com/features/19990917/infinite_04.htm

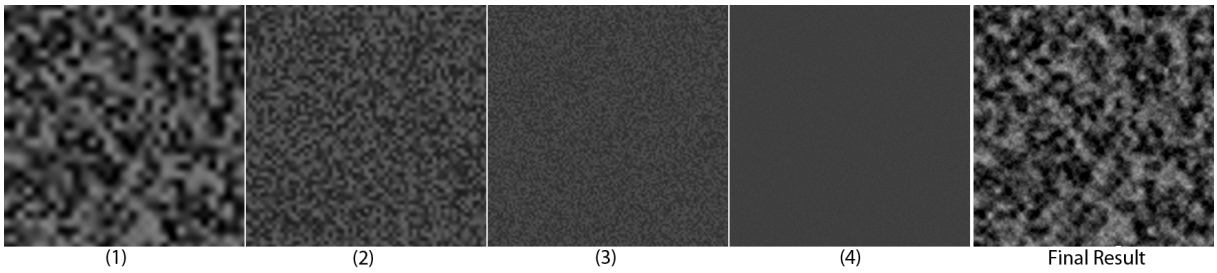

Figure 2.3: Octaves 1 to 4 of the resulting (final result) heightmap.

Interpolation Type it is common that noise functions return integer numbers and we need a floating point number, so the values returned by the noise functions must be interpolated between the neighbour integer values. Two of the most common are linear and cubic.

The example illustrated in 2.3 uses a Perlin Noise function with 4 octaves, a persistence of 0.5 and a cubic interpolation. This figure shows the result from each octave. This figure shows the result from each octave[10] and the final result of this Perlin Noise function.(Perlin 2000)(Tatarinov 2008)[11] [12]

2.1.2.2 Diamond Square Division

This method starts with a flat image/plane that contains the average value of what will be our terrain (which is a good way to define the kind of terrain one needs). The algorithm then iterates over the initial image by choosing and displacing some points. The points to displace are computed each iteration as the midpoints between the already computed points in the map. There is one exception to this step where in the first iteration the points to displace are the corners of the image. Figure 2.4 (a to e) illustrates several iterations on computing the points to displace.

After having the points to displace on a given iteration, each of them is then displaced by a random offset value. Notice that the maximum value of this offset must be inversely proportional to the number of iterations so that the initial values create the biggest displacements (originating for instance mountains or valleys) and the later create the detail of the heightmap. This iterative

[10] http://www.custard.org/ andrew/applets/turbulence/
[11] http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
[12] http://www.thinkinggames.co.uk/content/?p=74

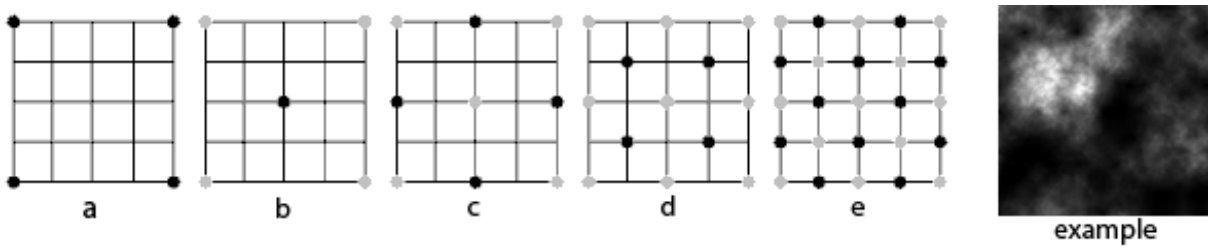

Figure 2.4: From a to e iterations on calculating the points to displace. On the righ an example of a heightmap generated with diamond square division

process stops when the changes are too small to affect the final result. For example in a pixel representation of a heightmap the pixel would be the smallest change possible. An example of a heightmap produced by this method is illustrated in figure 2.4 on the right.

Variations of this method are possible mostly because we can adjust the range of the random offsets. A variation of this method described in (Olsen 2004) and is called smoothed midpoint displacement, where a more eroded terrain look is achieved "*by multiplying the size of the offset range with the height average when calculating new values*" (Olsen 2004). An important property to notice is that the initial image is not necessarily flat, for instance if one wants a mountain range we can easily draw some initial displacements and influence the final result.[13] (Olsen 2004)

2.1.2.3 Voronoi Diagrams

The voronoi diagrams alone can be used to generate terrain but the results are poor in terms of "looking natural". However, in combination with other methods they quickly become more useful due to the arrangement of space they provide. "*A voronoi diagram is a discrete set S of points in Euclidean Space and for almost any point x in it, there is one point of S closest to it. The almost words stands because there may occur situations where there is two or more points equally distant to x.*"[14]. These diagrams can then be used to generate heightmaps (usually with a mountainous nature) by interpolating the height values between the diagram points which creates the differences in height and originate heightmaps like the examples 1-4 from figure 2.5. These heightmaps look unnatural as opposed to the methods discussed before. A way to

[13] http://www.thinkinggames.co.uk/content/?p=74
[14] http://en.wikipedia.org/wiki/Voronoi_diagram

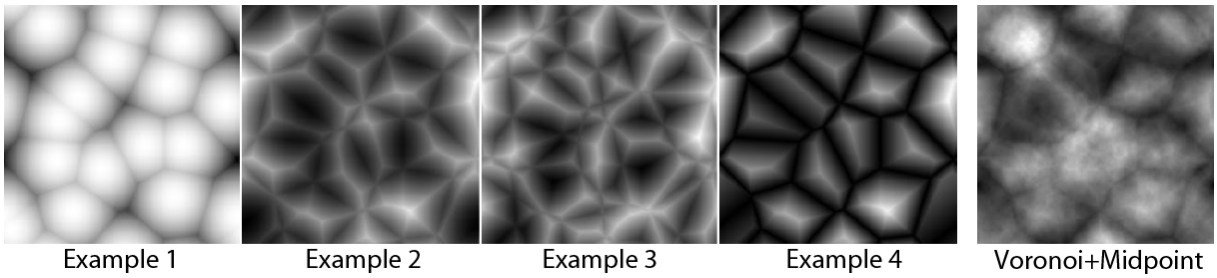

Figure 2.5: Examples of Voronoi heighmaps computed with different interpolation functions (examples 1 to 4) and a combined heightmap with voronoi diagram (1/3) and midpoint-displacement noise (2/3)

achieve better results is by combining these with one of the previous methods, for instance the Diamond Square method. This combination tries to retain the natural look from the previous methods and create a more feature full map. This combination has already been described in (Olsen 2004) where they start by dividing the map in regions and randomly place points in each. From these points (called "feature points") they generate the voronoi diagram. Then they combine the voronoi diagram heighmap with the smoothed midpoint displacement(Diamond Square Division) heightmap, in their case the first has a weight of 1/3 and the second of 2/3. A resulting heightmap of this method can be seen in the figure 2.5.

2.1.2.4 Using real world data

Real world data can also be used for map generation as described in (Chiang, Tu, Huang, Wen-KaiTai, Liu, & Chang 2005). This approach makes use of real world data to achieve natural looking maps. The process has four phases where in the first one the user specifies the kind of terrain he wants to create by a set of simple primitives. For example a mountain would be represented by a cone. For the next step it is important to understand that they have a collection of all the "terrain units"(see figure 2.6) that were manually segmented from real world terrain elevation maps. The next step matches the primitives specified by the user with the terrain units available, achieved by using properties like ridge similarity and contour similarity. The final step is the alignment and stitch where all the best matches from the previous step substitute the primitives through a set of translations, rotations and scaling to align them correctly. A minimal cut approach is used to stitch all the terrain units seamlessly. From this last step the final heighmap file is created.

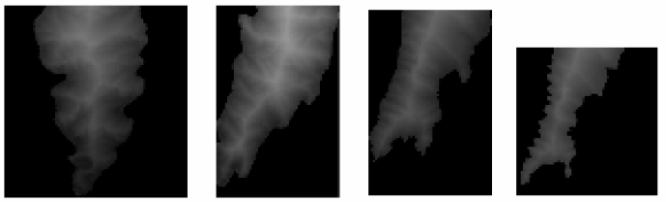

Figure 2.6: Terrain units example (Chiang, Tu, Huang, Wen-KaiTai, Liu, & Chang 2005)

2.1.2.5 Comparison

Each method has its advantages and disadvantages, and we can compare them in some of their characteristics. The parameters for our comparison are the following:

No unnatural shapes(US) if after the generation process there are no recognizable shapes which look unnatural.

Feature definition(FD) if it is possible to guide the creation process in order to define specific features like a mountain range

No parameter tuning(PT) does not require specialized knowledge about the generation process and adjustment of parameters in order to create the objective results.

Player placement(PP) the process creates maps which support player placement without the need of posterior editing.

No Human input(HI) does not need a large amount of input from the map creator.

No data dependency(DD) method is not dependent on the existence of data external to the algorithm which can vary in quantity and quality therefore affecting the final result

[1] the level designer must have a deep knowledge about the map creation process and all the game attributes in order to create a map for a game.

The Note that the green cells represent the desirable value for that characteristic and red cells represent the undesirable. manual creation of a map and the procedural way represent different tradeoffs. The manual approach is very work intensive for the level designer, but allows the creation of very personalized and feature full maps. These can even be historic game maps or specific strategic setups for strategy games. The random generation allows the quick creation of a map but creates some loss of control over its details. The Perlin Noise

Method / Characteristic	US	FD	PT	PP	HI	DD
Hand-made	Yes	Yes	No[1]	Yes	No	Yes
Perlin noise	Yes	No	No	No	Yes	Yes
Diamond Square Division	Yes	Yes	No	No	Yes	Yes
Voronoi Diagrams	No	Yes	No	No	Yes	Yes
Voronoi Diagrams + Diamond Square Division	Yes	Yes	No	No	Yes	Yes
Using real world data	Yes	Yes	Yes	No	Yes	No

Table 2.1: Terrain height synthesis methods comparison

makes too much use for randomization and does not allow control over feature definition. The method of using real world data offers very good results but needs a set of pre-created terrains to generate the actual maps which can be hard to acquire. From the several methods covered the most suitable methods available for our purposes are the midpoint displacement combined with voronoi diagrams which give us the best tradeoff between control and natural look. Even though in the comparison table this method looks very similar to the midpoint displacement alone, it provides more feature definition control because of the shapes inherent to Voronoi Diagrams. With Voronoi diagrams we can create separate areas with different height values and the midpoint displacement with that input can create the randomization needed to achieve the natural look necessary. As for player placement none of the procedural methods can provide it, even though the voronoi diagrams could provide for places specifically to create player starts, the algorithm itself does not incorporate any kind of support for such a detailed generation.

2.1.3 Interesting Maps

This concept is elusive and there is no fixed definition of it. The difficulty in creating such a definition is that each game has its own characteristics which make different map attributes more or less relevant. Therefore, such a definition must be adjusted for its own context. For example in a multiplayer FPS(First Person Shooter) much of the gameplay depends on map flow. This map flow depends on how the map keeps the player competing in a balanced way. Having this notion, such a map should have a circular flow created by having no dead ends with the exception of some tactical places like a sharpshooter position. Other characteristics to take into account are the spawning point locations, choke points, powerup placement, ambush positions and base locations and configurations (Feil & Scattergood 2005b) . A documented example is Battle for Wesnoth where guidelines for what makes an interesting map are given to the map

creators[15]. For strategy games we have two different genres, the real time and the turn based. In both genres there is a common attribute, the terrain types must be used to create challenges but must also be logical in their distribution across the map to create a natural look. An example of non logical distribution is having a terrain densely forested in a mountain with 7000 meters of altitude. In real time strategy(RTS) games the map needs to be strategically interesting. This means it "*should have areas that allow advantages and disadvantages for defenders and attackers*" (Feil & Scattergood 2005c). These advantages can be created by carefully placing terrain features such as "bottlenecks, hills, valleys, unassailable cliffs, open plains and islands" (Feil & Scattergood 2005c). In turn based games there is one major difference which affects the map design, in these players usually have more actions and time to think which of them to choose. Usually these maps tend to represent much bigger terrain portions than RTS games and require the creation of interesting tactical challenges. It is common that such a map is created to demonstrate some advantage or disadvantage that is frequently present in historical games. Much of map creation in turn-based games is done through experimenting with the attributes that affect their challenges. Even though not documented such games should derive some parameters which compose interesting game maps from RTS games such as features like mountains or islands. But the wide range of options provided by these games and more complex internal economy should also add new possibilities to create new challenges from their other attributes, for example more kinds of resources: horses, pigs, etc; or terrain characteristics: swampness, roughness, etc; which could create more challenges like lack of resources, uneven distribution of different resources, game races different moving speeds on more terrain types, etc.

2.1.3.1 Terrain Features

Even though some terrain features were mentioned while describing the processes of terrain generation it is worth exploring them further as they usually affect a map's gameplay. The features on a map can have two major purposes, add to the map realism or create strategy challenges for the game(Feil & Scattergood 2005a). They vary depending on the game and the characteristics influencing it. For instance a game where you need to build boats to cross water, this will influence the players' needs for movement, thus, influencing the game and is a feature

[15]http://catb.org/ esr/wesnoth/campaign-design-howto.html#_making_maps_interesting

to consider. It is common to have features like land form, mountain ranges and rivers. We will focus on the three mentioned because they are typical terrain features amongst many games and there is a scarcity of information on the subject. It is also important to mention we will focus more on the feature's interest than on how they are created.

2.1.3.1.1 Land form The basic types of land forms are continent and archipelagos. A map which has a continent land type means its majority of terrain is interconnected land, so usually we would have a huge land mass. In this case the space is usually not a concern but the player is probably easily accessible for enemies. The archipelagos type as the name suggests is composed of several small landmasses usually separated by a sea which breaks the interconnection between landmasses. In this case the player might be less accessible to enemies but it probably has other challenges like space and resources restrictions. This feature is available in Civilization 2[16] where we can choose the landmass for each game.

2.1.3.1.2 Mountain ranges This feature also plays both the visual and strategic role. For the visual part, a map without mountains or a single mountain range seems unrealistic, unless it is a specifically themed map. On the strategic part the influence is deeper because mountains provide several strategic advantages/disadvantages. Mountains usually influence several factors like battles because of their high ground and movement because higher (and rougher) terrain is usually harder to cross and resources because they are usually one of the rare places where ore resources are abundant originating disputes. An example of both the visual and strategic effect of mountains is Civilization 2, where mountains have the highest movement cost [17].

2.1.3.1.3 Rivers River features are usually created in a map after the terrain height features are defined because they commonly start in some high ground and flow down towards a lake or a sea. Examples of methods to create rivers in terrain have been described in (Prusinkiewicz & Hammel 1993) [18] [19]. Rivers also have both strategic and visual effects. On the visual side having a landscape with rivers helps the map look more natural as they are a common feature

[16] http://www.mobygames.com/game/sid-meiers-civilization-ii

[17] A table with all the terrain effects of civilization 2 can be found in http://www.civfanatics.com/civ2/downloads/poster, "Terrain specifications"

[18] http://www.gamasutra.com/view/feature/3549/interview_the_making_of_dwarf_.php

[19] http://pixelenvy.ca/wa/river.html

in the real world. On the strategy side this feature usually affects the resources production of the terrain on top of which it is placed. It can also serve as another way for the level designer to create restrictions on the ability to cross a terrain region. An example of river usage with resources influence is Civilization 2 and one where it influences the ability to cross terrain is Lord of The Rings Battle for Middle Earth[20].

2.1.3.1.4 Summary A very important factor for map interest are its features and their use in the game. They can be used for aesthetics or game influence. Features can be generated in different ways, for instance the landform, mountain ranges and even some lakes(at the same height as the sea) could be generated with heighmap methods, but the rivers must have a different approach as they cannot be easily represented through height (how to differentiate from a normal terrain depression?). Given our context terrain features like lakes and mountains are the most important due to the strategy influence they can provide in the map.

2.1.3.2 Resources Distribution

Also important in the creation of a game map is the resources distribution. It can be defined as the process of placement of resources throughout a map. The information on this subject is very scarce, even though it is very important because a large part of the player's experience is only achieved through the correct distribution of resources. For instance in a strategy game resources are needed for almost every kind of development and progression. In games where maps are hand-made, the resources are distributed manually. In this case a balanced distribution rests on the level designer, either in terms of amount or distance to players. As for the automated distribution of resources there is a documented example, Dwarf Fortress[21]. In this case all the world is procedurally generated, it starts by generating the heightmap and then uses the altitude to guide the distribution of other map properties like resources such as the vegetation. Here everything is generated by having the factors which influence each resource have some weight in its generation, for instance in the generation of the temperature map the elevation and latitude influence it.[22]

[20] http://www.ea.com/official/lordoftherings/thebattleformiddleearth/us/home.jsp
[21] http://www.bay12games.com/dwarves/
[22] http://dwarf.lendemaindeveille.com/index.php/World_generation

Resources distribution can be done in several ways and even though there was little information on how it is actually done in current games, it is important to understand that they affect the strategy aspect of a map and must be carefully distributed.

2.1.3.3 Objects and Structures

In games it is common that some kinds of structures or objects are randomly dispersed throughout the map, some influence strategy others just to add visual detail(Feil & Scattergood 2005a). Once again one of the ways to do this is by manually placing every object in a hand-made map. A procedural way of doing this has been briefly described in (Smedstad 2000) where it uses an obstacle mask to represent the blockage the objects create. Based on several states in the mask the required map accessibility can be ensured. With this and a small set of rules based on objects' distances a proper object/structure (example towns or mines) distribution can be ensured.

Depending on the game, objects and structures can play an important aesthetic role or even influence game economics. Sometimes they offer different possibilities to players (like shelter from enemies), however, in our context it is important to understand that they add interest and must be considered when the game in question requires it.

2.1.3.4 Player start placement

Any game map needs to have its players start places distributed in some way. In strategy games a player start has three major characteristics which will influence the gameplay, its location, its resources and the terrain suitability. All of these are important for game balance and successful player development and when correctly placed create better gameplay. If a start place is located in a map bottleneck and is poor in resources, the player who starts there is in a very disadvantageous position. It has a high probability of being attacked and probably (unless it was done on purpose) has few resources at his disposition to try and compensate this disadvantage. For player placement many RTS games create pre-configured balanced terrain areas (terrain types and resources) and distribute them as far apart as possible in the map. An example of games which appear to follow such a scheme are Starcraft and Lord of The Rings Battle for Middle Earth. Besides this there was no information we could find about how players

are placed in a game map for strategy games.

2.1.3.5 Terrain analysis

While creating maps several issues arise related to the playability of those. As we have seen in previous chapters random generation methods create maps which retain some randomness and terrain features to make them look natural. However, even if we are able to sometimes generate maps that look very good these might not meet our expectations either in terms of terrain type distribution or player resources/positioning. To improve the results from random generation and help the manual creation of maps we must be able to analyze them. Several games already use probabilistic[23] and other validation methods [24] to decide whether to accept or reject maps for play. However, most of the thoroughly described analysis algorithms are used for game warfare in the form of tactical/strategical planning as seen in (Woodcock & Wyrks 2002),(Tozour 2001a) or sometimes for building assistance (generate building places suggestions). Terrain analysis can also be used for other purposes like the determination of choke points (Higgins 2001). Nevertheless, the area that researches this kind of technologies in games is Artificial intelligence (Buro & Furtak 2004) and the methods we will describe are the influence mapping and pathfinding as are the most useful for our purposes.(Millington 2006)

2.1.3.5.1 Influence Mapping This method helps identify places/patterns of interest in a map to an AI agent. It is generically done by having a grid representation of the map within which we assign a value of influence (positive or negative depending on our goal) to objects or places which are relevant for our analysis. This influence is then propagated through the map and finally the result is evaluated with heuristics that reflect our purpose of analysis. Next we will briefly describe the main steps of influence mapping:

1-Creating the world representation this step consists in creating a mapping between the game and influence map grid representations where we must consider the balance between performance and precision.[25] (Tozour 2001b)

[23] http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural.html in the *Runtime random level generation* topic

[24] http://dwarf.lendemaindeveille.com/index.php/World_generation

[25] http://www.fdaw.unimaas.nl/education/4.5GAI/slides/Influence%20Maps.ppt

2-Initial Influence value The initial influence value of each cell is zero by default, but some cells might have a positive (attractors) or negative (detractors) impact on our decision for which we need to assign a different value.(Olsen 2001)

3-Propagating the influence Propagating the influence creates an analysis suitable influence map by creating the areas of influence around the initial values. Especially important is the "falloff function" and its type (exponential, linear, etc).(Tozour 2001b) (Millington 2006)

4-Analysis The influenced map is analyzed with heuristics which aid us in our decision making.

The method can be used for several kinds of map analysis like look for a player start places for players which meet the initial resources requirements for those players. In In Age of Empires 1[26] this technique was used to solve problems like "*whether to place a gather site by a single resource or a group of the same resources*". In Age of Empires 2[27] these techniques were further developed.(Pottinger 2000)

2.1.3.5.2 Pathfinding There are several algorithms that can be used for pathfinding, but one of the most useful and largely used in the videogame industry is the A* as seen in (Bourg & Seemann 2004), (Forbus 2005) and (Millington 2006). It is a graph search algorithm that finds the least-cost path from a given initial node to one goal node. It can have several implementations and different usages, in our case the most important is the pathfinding in game maps[28]. This algorithm is important when analyzing the map for several parameters like: walkability of a map and reachability of all regions in a map. An example usage of this technology for this purpose is in Heroes of Might and Magic 3[29] where it was applied to ensure that all areas are reachable. (Smedstad 2000)

2.1.3.5.3 Comparison Map analysis allows us to understand map terrain and retrieve specific information. For our purposes of a strategic game map analysis it takes a different form than the most common (example: building placement). It could be used to evaluate the strategic

[26] http://www.ensemblestudios.com/Games/AgeOfEmpires/Default.aspx
[27] http://www.ensemblestudios.com/Games/AgeOfKings/Default.aspx
[28] http://www.policyalmanac.org/games/aStarTutorial.htm
[29] http://www.mobygames.com/game/windows/heroes-of-might-and-magic-iii-the-restoration-of-erathia

advantage created by a map. For this both influence mapping and pathfinding are useful. No single method is more important than the other, simply each method is more or less suitable for a given task. Influence mapping is better suited for several map pattern analysis purposes like resource analysis and terrain defensiveness when parameterized with the correct desirability and heuristic functions for those purposes. Pathfinding is better for the purposes of identification of choke points and rechability, after a map is created by applying a pathfinding algorithm like A* between every player we can determine the choke points of the map where conflicts are more probable to emerge.

2.1.3.6 Map balancing

Balance plays a very important part in a game. This is what keeps the game interesting and fun to play because if it is not balanced then a player can feel stuck and/or frustated with the game outcomes. This makes him feel unmotivated to play because his actions are inconsequent. In literature about game balancing the focus is usually on the game mechanics itself and how a player plays, for example with the usage of negative or positive feedback (Falstein 2005). This does not address map balancing directly, nonetheless, maps end up being tested indirectly as the players test the game with the map. Even though most of game balancing is done with player testing there are other ways being researched like the one described in "Using Coevolution to Understand and Validate Game Balance in Continuous Games". (Leigh, Schonfeld, & Louis 2008)

The idea behind this approach is to have successive evolving generations of strategy formulating agents play against each other. The agents evolve their strategies through genetic algorithms and the evolved agents are then put to test against other agents. The best scoring strategies pass to the next generation of strategies. This creates an evolutionary process of improving strategies. They used a two player game (having a population of agents for each side) as a test case where each population was tested and evolved against each other.

The title of (Leigh, Schonfeld, & Louis 2008) brings the issue of the differences between balancing in continuous and non-continuous games. In discrete games we can analyze the moves of a player with a search tree (more or less complex depending on the game) and try to detect imbalances, but in continuous games the search tree is impractical to build thus leading to the current state of game testing for the continuous games.(Leigh, Schonfeld, & Louis 2008)

However, when approaching the problem of map balancing in strategy games it often consists of primarily three things: a player's proximity to the resources, its position in the map and its accessibility to other players (possible enemies). All of them are set when a player is placed in a map, thus, this is our focus in map balancing. Some information about strategies of game balancing like the "Rock-paper-scissors" and asymmetry in games has been discussed[30] but very little of the discussion has been focused on map balancing. In most RTS games, players are placed in similar pre-configured areas with resources and good for defense helped by being as far from each other as possible, like the corners of a map. This allows all the players to have a good start and conquer weak or uninhabited zones of the map. Assuming equally skilled players the conflicts created are usually between players with the same power given their equal start. It is important to notice that sometimes unbalanced situations are purposefully created to challenge the player to overcome a disadvantageous situation. This disadvantage can come from either map disposition or game constraints like a timelimit to achieve a given task. Examples of this are games which use editors such as Starcraft or the Age of Empires[31] Series. For turn based strategy games most of the map balance goes into creating "interesting tactical challenges" (Feil & Scattergood 2005c) in the map through the wide range of attributes these games and its maps usually have. There is no specific information on the player placement or resources influence on game balance, but like in RTS games these characteristics play an important role on balance due to their importance in the player's defense and development. Some of the games which have editors that enable players or level designers to manipulate the maps in the creation of tactical challenges are: Civilization 2 and Heroes of Might and Magic III. (Feil & Scattergood 2005c)

Map balancing is still an area little documented (especially for strategy games) and some basic ideas of balance like player placement in the corners of the map are used to create balance. In this situation players are equally distant and it is harder (than with other possible distributions) to reach each other which mitigates conflicts. These ideas are too simple to hold the answer as how to create a balanced random generated map, especially when applied with games with more complex internal economies.

[30] http://www.gamasutra.com/view/feature/3291/designers_notebook_a_symmetry_.php
[31] http://www.microsoft.com/games/empires/

2.1.3.7 Dynamic map growth

With the advent of MMOG(Massively Multiplayer Online Game) games map creation has new needs, one of those is being able to accommodate all the players since they join at different game times. In the context of a strategy game map its engine must be able to have a map which dynamically increases size as new players enter as opposed with the classic map generation paradigm. In this a map is created once, all players enter at the same time and finally the game starts. Several online multiplayer games have created game maps in this context and rely on having many players to ensure that a player does not have to wait much time for others in order to start playing[32]. However, in the massively multiplayer context the different player subscription times are unavoidable due to the huge amount of players and the impracticality of synchronizing all of them for a game start. Maps for these games need to have an architecture which allows them to expand their maps as new players enter while still maintaining the consistency of the already existent map terrain for players already playing.

Dynamic map growth stands for a map which has the possibility of growing as needed to accommodate new player entries at any time after the game started. There was no specific information on this subject that we could find. Usually maps are always of a fixed size, even if they are not completely known by a given player they are already generated, but hidden by something like a fog of war. However, there is an article which does not focus on this subject specifically but ends up showing a way to grow a map (without player placement considerations). It describes a dynamic map growth by having the player only see a part of the map at a given moment and generating new parts of the map on-the-fly. These new parts of the map are then stored for future use. In the article's example the player starts in a 20x20 region of map and when he explores beyond the borders of this region a new 20x20 region is automatically generated based on the borders of the older one, if there is no stored data for this new region (determined by the coordinates of the player).[33].

Dynamic map growth is a subject which is not answered by the documented work on the subject. Even though several works regarding infinie landscapes (Schneider, Boldte, & Westermann 2006) or cities (Greuter, Parker, Stewart, & Leach 2003) exist none addresses it regarding a game context's needs. A solution to these needs of MMOGs can be based on the example

[32]http://weewar.com/
[33]http://pixelenvy.ca/wa/river.html

found but many problems remain untouched like delayed player entries, game balance and map visibility for non generated areas.

2.2 Case studies

In this chapter we will focus on analyzing existing games which use terrain generation. A set of game map editors were briefly tested and analyzed regarding their capabilities and its usefulness regarding our goals of map generation and dynamic map growth. Finally a set of Strategy MMOGs are analyzed regarding how they solve the problems we are studying in the MMOG context, map growth, player placement, delayed player entries and player balance.

There are also several other tools used for terrain generating (Saunders 2006) but since they usually are not suitable for game map generation and due to space restrictions we will not analyze them.

2.2.1 Game Map Editors

The evaluation of existent map editors is important to understand the paradigms used in terrain generation for current games and what capabilities do these editors offer. We will briefly analyze and compare two game editors from games which relate to our goals by being turn based games with game influent maps that also have an online context. Civilization II is part of the renowned Civilization series of turn-based strategy games and has many of the characteristics we are focusing while readily providing a game editor for analysis[34]. Battle for Wenoth is a turn-based strategy game which has a special focus on the online context with a complex map environment. These games do not deal with a massively multiplayer context, but still offer the online context and other characteristics we intend to study while providing map editors that enable us to research the current map generation paradigms.

The "Battle for Wesnoth Map Creator" is a typical editor where the user can directly "paint" the map as he wishes over an hexagonal grid. Without the possibility of a random generation or even an auxiliary tool for it, the help only comes in the form of bigger brushes and the fill tool. Nonetheless, inside the game there is the possibility of random map generation.

[34]The latest Civilization IV also provides an editor with similar functionalities but completely integrated in the game which would difficult our analysis (since Civilization 2 provides a standalone editor).

Editor / Characteristic	Random generation	Map editing	Player placement
Battle for Wesnoth	No	Yes	No
Civilization II	Yes	Yes	No

Table 2.2: Map editors from games comparison

Civilization II Map Editor offers the user the choice of hand-made or randomly generated maps. When hand-made the user has a toolset with the terrain types, starts cities, etc, that he can use to create the desired terrain, similar to the Wesnoth's editor. On the other hand the user can issue the random generation of a map based on a set of parameters: landmass, land form, climate, temperature, etc. Note that after generated the maps can still be edited and improved manually.

Each game has its specific needs so each game map editor has different properties. We will compare the editors reviewed in this section regarding some characteristics relevant for our research. The characteristics we will be focusing on are:

Random generation capable of generating maps randomly without the map creator designing the complete map.

Map editing the capability of editing the map regarding terrain details.

Player placement capable of placing players automatically.

Note that the green cells represent the desirable value for that characteristic and red cell represent the undesirable. From this comparison we conclude that the Civilization editor is more complete than the Battle for Wesnoth. Even though it also lacks the player placement (in the editor) the random map generation is already a big advantage. Also note that even when using the random generation the map editor of civilization allows the level designer to edit it.

2.2.2 Games

2.2.2.1 Travian

This is a massively multiplayer online strategy game where a player has to develop villages and evolves through trading, battles, alliances, etc. For Travian there is no map editor so the information was gathered through in-game information and analysis. As to how the world is

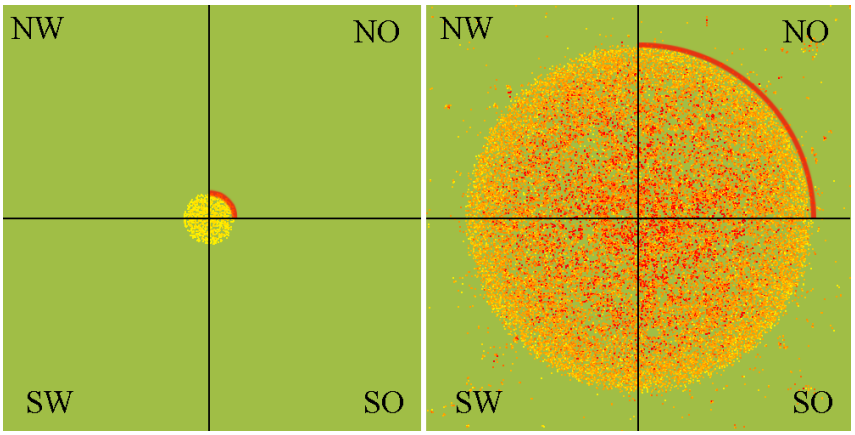

Figure 2.7: Travian player distribution, the red line is the zone players are being placed in the NO quadrant

generated it seems that there is a random generator that distributes the different types of valleys (the game terrain unit) available throughout the map, guided by a probabilistic distribution of tiles for each of the few existing types. Regarding map growth, its map does not grow, it starts out with a map of fixed dimensions (relatively big, so as to accommodate tens of thousands of players) and players are placed until the map is full. Regarding player placement the game initially offers the player the possibility to choose the quadrant he wants to be placed on (northeast, northwest, southeast, southwest) or be randomly placed. This placement seems to evolve in ever growing rings (since the game start) around the (0,0) coordinate as to maintain the game balance as much as possible (see Figure 2.7[35]), This ensures ensures that players which have been playing longer will be closer to players that also have been playing longer and thus tries to diminish the difference in player subscription times and in this way mitigate the imbalance it brings to the game. Another measure applied to further prevent these imbalances is that when a player enters it is under a beginner's protection period. This protection shields the player from older players for a varying maximum of 2 to 14 days, depending on how long the game is running.

2.2.2.2 Ikariam

This is another massively multiplayer online game in which the player manages a village in all its subjects: science, economy, war, etc. The game has five basic resources and gold which are the

[35]http://forum.travian.com/showthread.php?t=78078

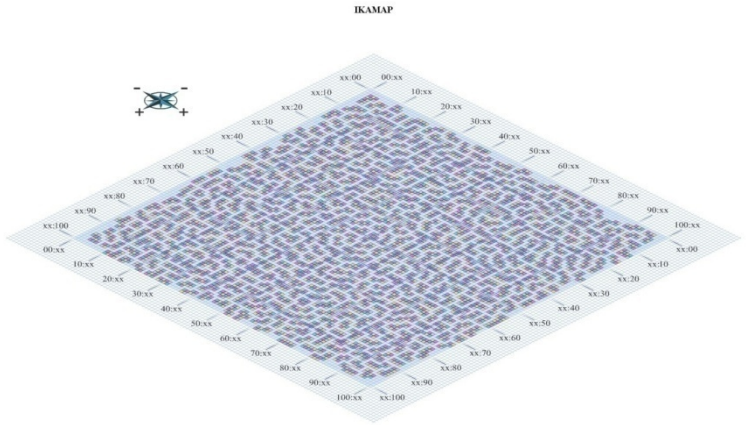

Figure 2.8: A typical Ikariam map

building blocks of everything in the game. The player is placed in an island which has a special resource the player can then use to trade with players from other islands or use for himself. The maps in this game are from the archipelago type contrary to the continental type from Travian. They are of a fixed size therefore not growing as new players enter. By investigating the map, we observed that islands begun at (0,0) and expanded until (100,100). Out of that area it was just sea (see Figure 2.8[36]). Most populated islands were near median values of these coordinates whilst the outer ones were mostly empty, but still had some players. Here players are placed on islands in the middle of the ocean, always in small islands, no continents or large landmasses. The size of all the islands and the number of players each accommodates is always the same.

The distribution of players is basically achieved by filling island places and there does not seem to be a very big worry with balance as a new player can be placed in the same island as a much older player. When a new player enters there are no protections or compensations for it which creates an imbalance between new and old players.[37]

2.2.2.3 Comparison

Both games presented here are from the MMOG context and so we will focus on comparing them about the main problems this context presents. The characteristics we will compare are:

Map growth the map grows as new player enter.

[36] from an unknown source, found in http://homepage.ntlworld.com/papermodels/ikamap.jpg
[37] http://ikariam.com/

Game / Characteristic	Map growth	Player Placement	Balanced entries
Travian	No	Yes	Yes
Ikariam	No	No	No

Table 2.3: MMOG games comparison

Player Placement has a scheme to try to place players in a position which will not cause them to be overwhelmed by older players.

Balanced entries if there is any scheme to compensate or protect players for delayed entries when compared to older players.

Note that the green cells represent that it is the desirable value for that characteristic and red cells represent undesirable.

None of the games has map growth, they have a limited number of player subscriptions for each server, limited by the map size. For placing players Travian adopts a scheme which mitiages the disadvantages of entering later in the game whereas Ikariam seems to have none. Contrary to Ikariam, Travian also offers a beginner's protection period so players can develop for sometime without the fear of being attacked. So Travian has a better system to deal with the balance problems of new player entries, but is still limited by its map not being able to grow.

3 Solution

3.1 Introduction

During our research we obtained answers to part of our research questions. Regarding the procedural generation of maps we found a variety of methods which are suitable for it. However, the question of map interest is elusive and depends on where a specific method is applied. One important problem which persists is the procedural player start placement in games where the map heavily influences gameplay.

Game balance is an area little documented so the information found was scarce. The information on how this is actually done is very basic, like the placement of players as far apart as possible. There are algorithms which can be used to aid balanced terrain generation but are usually applied to different uses (see 2.1.3.5). Nonetheless, it is important to highlight that the player starting places play an especially important role regarding this subject in strategy games.

On the MMOG (or TSMBG) context even less information than regarding game balance was found. Some information on how this problems can be solved was only found based on existing games and its analysis. The map growth was not found on any of these games though.

For our solution we wanted to create complex game map environments which are scalable for a massively multiplayer context. Since none of the current approaches (either hand-made or procedural) to map generation can properly adapt to the combination of these properties we created a solution which balances the work of the game designer and the amount of automatization the generation method provides. Additionally, we also intended that our maps contributed to the game balance, which is commonly done mostly by player testing after map creation. In this subject the player start places disposition played an essential role since we are dealing with strategy games. Since the specification of a map generation method for a game is very dependent on the target game's specific characteristics we needed a concrete game context. Therefore, our solution was developed in collaboration with the game Almansur where the problems we

formalized first emerged.

Throughout this chapter we will describe our solution, first we introduce our game context (section 3.2), second we describe its generic specification (sections 3.3, 3.4 and 3.5) and third we explore its application with our case study Almansur Battlegrounds (section 3.6 and 3.7). The solution and its algorithms will be presented with a detailed description and sometimes images and/or pseudo-code to aid the understanding of the explained concepts.

3.2 Game Context

The approach we will present is focused on turn-based strategy games. Amongst these games there are many differences, namely the grid system used to represent their maps. Throughout our solution description we will use a vertically aligned hexagonal system with the hexagon as our smallest terrain unit due to the advantages of hexagons in tile based representations. These offer visually accurate distances between a given tile and all its neighbours (contrary to squares), moreover they also map directly to the cardinal direction system (contrary to triangles) which is commonly used in geographic orientation and as used in strategy games. Nonetheless, most of the solutions presented can be adapted to other grid systems.

Another difference related to the grid system is the kind of scale our method addresses since in a tile based representation, a tile can can represent different levels of detail which influences the game and also the map generation process. A large scale, which offers much detail means that the map will extend long and thus there will be much more terrain tiles to be generated and for the player to explore. Such games usually have a bigger exploratory component for players. Additionally, this influences the way map balance has to be thought since the distances between players play an important role in strategy games and due to the player start placement distribution. In the description of our solution we will consider games with a small scale, meaning each tile represents a large area, but the schemes provided are parameterizable so that they are also suitable for larger scales as we will see ahead.

Our specific game context is based on Almansur Battlegrounds, it *is a strategy game of politics, economy and war set in the early middle ages or, in some scenarios, fantasy world* [1].

[1] From the game's website

Altitude	The altitude of the terrain.
Relief	The roughness of the terrain.
Swampness	The abundance of swamps.
Fertility	How well living things (including population) reproduce in this terrain.
Arborization	The abundance of vegetation.
Gold	The richness of the terrain in terms of gold.
Iron	The richness of the terrain in terms of iron.
Stone	The richness of the terrain in terms of stone.
Populations (all races)	there are specific populations for each race.

Table 3.1: Most important characteristics of Almansur's terrains

The game is turn-based and the map has a deep influence in the game, ranging from different troop movement speeds for each race to the difficulty and consequent cost and productivity that the same building can have in different terrains. In this environment there are several races for fantasy games: Barbarian, Dwarf, Elf, Human and Orc. Each one has different needs in terms of terrain characteristics, for example Orcs need swamps for better development but Elves need forests. So as a basis for good initial development, game balance and visual coherence each land must have in its initial position terrains created specifically for its race. In Almansur the player starts as a lord from a race of his choice with a small army and territory which he can then develop economically, make alliances or expand by conquering neighbouring territories. Until now this game has relied on manually created maps for a fixed number of players. Each map needed many hours of intensive work to prepare and created restrictions on players' choices since they could only subscribe to play lands from the set previously created for that map. For instance, if every land from a given race was already occupied, a joining player could not play that race and would have to choose from the remainder.

As a realistic game of military strategy, Almansur has a complex internal economy which requires a large set of map characteristics which affect the game. The grid model used is a vertically aligned hexagonal grid which serves as the basic way of representing the map. For every map hexagon there are many characteristics and all are relevant for map balance but a list of the most important and representative for our solution (the complete set and the tile scale used in this game can be consulted in Appendix A) is presented in table 3.1.

Based on the complete problem we sub-divide it into two sub-problems which brings several advantages due to the focus given to each. The two contexts we created are:

- The procedural creation of interesting balanced maps with Almansur's classic multiplayer game context;
- The procedural dynamic growth of a map with Almansur's TSMBG game context, while maintaining interest and balance;

The first context is common in most strategy games which offer the multiplayer game mode. It deals with the problem of interesting balanced maps providing an automated generation solution to many games that have this context. This promotes the continuation and automatization of the classic tournament game model (as present in Almansur or other games like Weewar[2]). In the second context we expand the previous work in order to add scalability to the maps created in the previous one in order to support adaptative online game tournaments. These adapt to the players subscribing by growing the map dynamically when players subscribe as required by a massively multiplayer online context with unpredictable player choices and subscription timing.

3.3 Map Interest

For our solution it is also important to define what an *interesting map* is as it depends on the kind of games we want to explore and it can include many characteristics (see 2.1.3). In our context of online strategy games an *interesting map* is a map which is visually rich and provides several strategic challenges for players. The visual richness depends on the variety of different terrain types present in the map and also on the presence and natural aspect of terrain features 2.1.3.1. The strategic challenges are created with terrain characteristics like choke points or high mountains which are difficult points of passage for armies. An important property these maps must have is no easily recognizable unnatural shapes of terrain.

3.4 Map Growth

At the core of the solutions presented in this chapter is the way the map will grow in terms of the distribution of players (player start placement) and neutral terrains. To create our maps

[2]http://www.weewar.com/

we use an iterative procedural generation which has several input parameters. But first we must understand how our terrain units, each hexagon, is created.

There are several ways to create a terrain, for example randomize all values of a given terrain, realistically simulate all the terrain parameters or create a list of all the possible terrains beforehand. However, each of these approaches has disadvantages regarding our goals, the randomization approach enables the creation of a great variety of terrains but does not enable for fine control over the characteristics of the terrains created, which could originate undesired terrain. Regarding the simulation approach it would additionally require the creation of a complete simulation model as seen in Dwarf Fortress which was not our goal with this work. Finally the creation of all possible terrains is impractical if we want to generate several thousands of different terrains. To solve this we propose a method that balances the work of the game designer and the variety of terrains generated by creating terrain prototypes, which act as blueprints for actual terrains, that are randomized only within preset intervals in order to create many distinct terrains. A terrain prototype contains base values and offsets for each of the terrain's characteristics which the game uses. Some of the most important are presented in table 3.1. Based on these prototypes a concrete game map terrain (represented as an hexagon) can be generated by randomizing all its base values.

The actual randomization process can also be done in several different ways, especially regarding the randomization boundaries created by the randomization offset(s). We could have different upper and lower boundary values for the randomization which would enable to control the balance between positive and negative randomization. However, since the game designer already has to create all the terrain prototypes this level of control adds a considerable amount of work for him which does not bring any great advantage. Therefore, we use an average based variation with a positive or negative single valued offset approach, where the game designer only has to define an offset value described in the following formula:

$$value_{final} = value_{base} - value_{offset} + 2 * random_{0-1} * value_{offset} \tag{3.1}$$

This randomized process enables that very few prototypes originate many different map terrains. The algorithm for the terrain generation is presented in algorithm 3.1.

These prototype terrains are carefully crafted by the game designer, since they are the basis for an adequate and balanced map generation. Besides the base values of each prototype the

Algorithm 3.1 GENERATE($name, prototype$)

Require: The *name* of the terrain owner, if any and the terrain *prototype* to guide the terrain generation.

Ensure: A new terrain is created with the specified owner.

1: $characteristics \leftarrow prototype[\text{“}characteristics\text{”}]$
2: $base\_values \leftarrow prototype[\text{“}values\text{”}]$
3: $offsets \leftarrow prototype[offsets]$
4: $terrain \leftarrow emptyHash$
5: **for** $i = 0$ to LENGTH($characteristics$) **do**
6: $\quad terrain[characteristics[i]] \leftarrow base\_values[i] - offsets[i] + 2 * \text{RANDOM}(0,1) * offsets[i]$
7: **return** *terrain*

game designer also groups prototypes by categories and evaluates the quality of each within their category. The categories will enable the use of the prototype for different purposes, like the the creation of different terrain features or suitable terrains for different races. These prototypes are then grouped based on the use they will have, so we have one category for each race, for each terrain feature type and for neutral terrains. The evaluation of each prototype enables the generator to differentiate prototypes within a single category and use them differently. This process is the basis of richness control in the map, if we had a random usage of prototypes we could not control the richness available and would easily end up having imbalances regarding richness distribution. The evaluation is done based on the game designer's heuristics to classify a given prototype's usefulness regarding the category it belongs to. For example, in the category "race dwarf", prototypes with a higher altitude value are more suitable for dwarf populations and so will be given a higher quality value than the ones with lower values for this characteristic.

Once defined our terrain unit generation we can then explore how to achieve map growth in order to accommodate an unlimited amount of players. There are several methods to grow or expand a map as needed, but since we need to distribute player start places across the map due to its strategic importance in the game types we are addressing (see 2.1.3.6) our method has to take it into account. The most simple way would be to randomly generate the map and randomly place players, however this approach is disadvantageous because we cannot control the map in any way, which easily leads to imbalanced maps due to inadequate player start placement. Even if we used a player start placement based on map evaluation heuristics, this approach would be very intensive since it would require, for each placement, a search for places suitable for a given race. Additionally, in the complex environments we are addressing where a race development depends much on the terrain characteristics many imbalances would originate

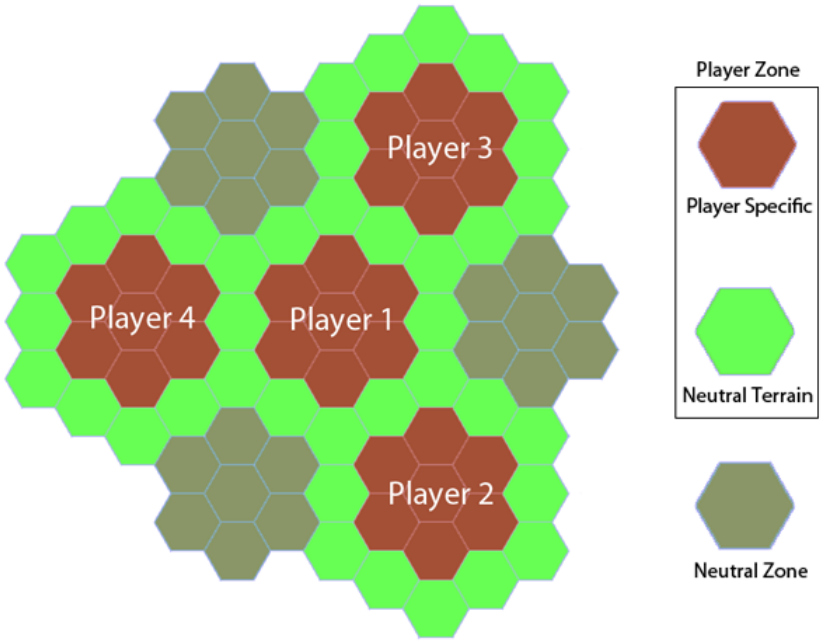

Figure 3.1: Player/Neutral zones schema

due to the complete random nature of map generation and to the influence the map has on player development. We could also place players in the corners of the map as several games do (see 2.1.3.6) but this approach does not enable the map to scale indefinitely for a massively multiplayer context. Therefore, to be able to support an ever growing map the idea is that it should expand by following a pattern of interleaved neutral and player zones. This aims to balance player expansion and military conflict as well as make the growth pattern flexible for different map configurations (for example for a different map scale). A zone is a set of terrain units(single terrains or hex) that are generated with a specific purpose based on one or several categories of terrain prototypes. Based on game tests, with manually created maps previous to this method (Quinta 2008), a player/neutral zone schema was chosen and is illustrated in figure 3.1. The schema shown represents a player which is surrounded by three player zones and three neutral zones. These player zones represent other players which cause direct points of conflict with the player in the center. If we had a player surrounded only by other players the diplomatic aspect of the game would predominate since a player had too many conflict points to manage. So instead of only players we also have three neutral zones act as indirect points of conflict between players restoring the balance we aim to create. Also notice that player zones are composed of two kinds of terrain, the player specific and the neutral territories which are the base of the flexibility of our growth pattern due to the possibility of varying the area of each

one individually.

The player specific terrains are created to be a combination of the most suitable ones for the player's race. The neutral terrains are randomly generated from all the neutral terrain prototypes but a percentage of them are influenced to also be the most suitable (from the "neutral terrains" category) for the neighbouring player's races. This enables to accommodate the players' races well and encourages some initial expansion. The neutral zones have two main purposes. The first is the resources they contain, which are reason for disputes and create indirect points of conflict. The second is the strategic influence given by the terrain type, for example a lake neutral zone forces troops to move around it and might create choke points to the passage of troops. Another example is a mountain range which creates a barrier to the movement of troops since they spend more time to cross them.

In this schema a zone is characterized by a size which specifies how many layers of hexagons around its center point it has. Based on the size s from a zone the number of terrains it contains is calculated with the formula:

$$\text{number of terrains}_{zone}(s) = 1 + 6 * \sum_{n=1}^{s} n - 1 \tag{3.2}$$

3.5 Growth Control

The scheme presented in figure 3.1 is the most tested but several variations can be easily achieved by varying the size of the player zone. This adds flexibility for the game designer regarding the creation of different game experiences (for example in aggressiveness) and the adaptability of our solution to different game contexts (for example with different tile scales). Notice that the player zone size variation also influences the size of the neutral zones and the overall pattern created. Since the player zone is composed of two kinds of terrain we create two parameters which enable the variation of the zone size. The first is the player specific size (PSS), which represents the radius, in terms of terrain units, of the player specific terrain generated for the zone. The second is the neutral terrain size (NTS), which represents the radius of neutral terrain added around the player specific terrain. With these parameters we can easily determine the number of terrains of a player zone with formula 3.2 and $s = PSS + NTS$. However, to determine the number of terrains a neutral zone has in a given schema we need to be able to

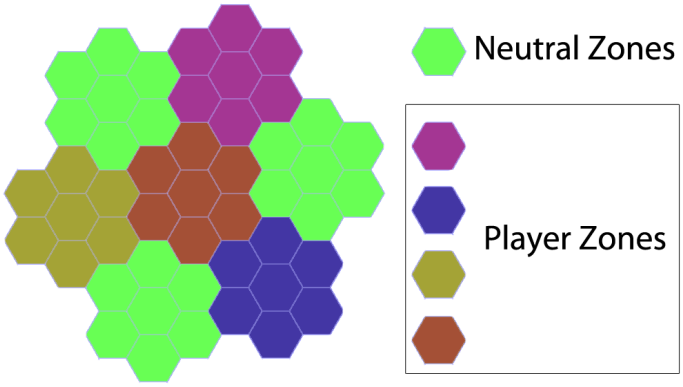

Figure 3.2: A more aggressive growing schema (PSS = 2 and NTS = 0).

calculate the neutral zone size (NZS). It is implicitly defined by the player zone parameters:

$$NZS = \begin{cases} PSS + NTS - 1 & \text{if } NTS > 0 \\ PSS & \text{if } NTS = 0 \end{cases} \tag{3.3}$$

The variation of the PSS and NTS values can attribute different levels of intensity and aggressiveness to a game. For example a game with a player zone composed by PSS=2 and NTS=1 (illustrated in figure 3.1) is less aggressive than another where the player zones is composed by PSS=2 and NTS=0 (illustrated in figure 3.2). This happens because players in the later case are placed closer to each other and have less territories to expand to. Maps actually generated with these different growth parameters configuration can be observed in figure D.1 from appendix D. Since the player specific terrain plays an important role in player development we must detail its creation process. There are several ways it can be generated, but we aim to created equally useful balanced zones even for players from different races. Several approaches can be used, an equal distribution using the same prototype quality, but this would lead to monotonous zones and less strategic challenges for players (since resources distribution among different tiles originates strategic choices). Another possibility is to use a random distribution of prototype qualities, but this would easily lead to imbalances since some players might end up having terrains much richer than others. As such we propose a solution which maintains the variety of terrains created while balancing the player specific terrains amongst the several races. Since the prototype quality specifies the usefullness of it within its prototype category, and if we use the same set of qualities amongst the different races and the game designer correctly evaluated the prototypes we have balanced player specific terrains. To do this we create a configuration called the player

Prototype Quality	Influenced terrains
5	2
3	2
<2	rest

Table 3.2: Player Terrain Configuration parameter example.

terrain composition (PTC) and the same PTC is used for all races. It qualitatively describes the player specific terrains by providing several associations between qualities of prototypes and the number of terrains created with that quality. An example of a PTC configuration is represented in table 3.2. The order of the lines in the table is also important since they dictate the generation precedence. For example if we were generating four terrains then the first two would be from quality five and the second two would be from quality three. In the case of this configuration we would only use all the associations if we were generating more than four terrains.

Notice that the last line is different, it adds flexibility to our PTC definition since without it a change in the PSS and/or NTS would easily require a PTC change too. So the last line of this parameter is always different and the association here is composed of (modifier, quality) - rest. The added modifier enables to create a range of qualities to be used for the influenced terrains, so in this example, the first column and last row can be read as "less than" two. Other possible modifiers are "more than" represented by $>$ and "equal to" $=$. The rest is a special word which means that if the number of terrains already generated is greater or equal to the sum of the previous rows' "Influenced terrains" value then the new one will always be generated with a quality from the range specified in this line.

3.6 Classic Multiplayer Context

This part of the solution aims to procedurally create multiplayer interesting and balanced game maps in a static context. This means that the generation process occurs only once for each map and that all the information needed for the generation must be available beforehand. Examples of maps created in this fashion are the ones used in multiplayer for Weewar or the ones created by the generator programs analyzed in 2.2.1.

From here on all the algorithms' specific details will be based on Almansur's case, for other games some adaptation might be needed. All the algorithms presented in this section were

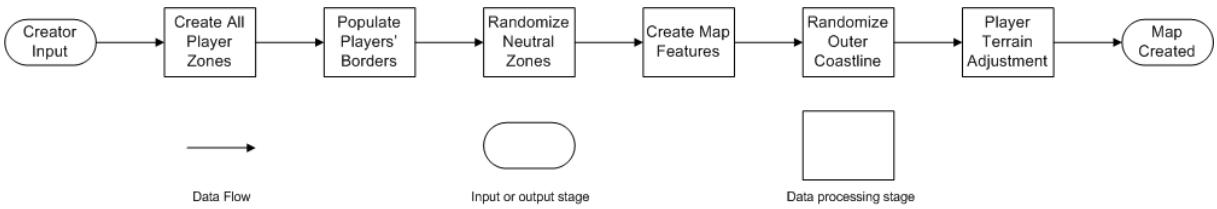

Figure 3.3: Data flow and stages for statically created maps.

completely implemented but due to space restrictions we will explain in more detail the most important stages only.

3.6.1 Generation Process

In order to create the desired maps for the classic multiplayer context a sequential process is used where the map data is processed in several stages. The sequential execution is suitable for a map generation process, since many of the stages depend on the work done by the previous one. The flow of data and the stages of the process are illustrated in figure 3.3.

3.6.1.1 Creator Input

A process of map generation usually needs several inputs, either pre-determined or provided at generation time. In our procedurally created maps this comes in the form of parameterization and player information which, in this context, is required at the start of the generation process. The minimum information needed from each player is:

Name name of the player;

Race the race of the player, examples Human or Orc;

Regarding the parameterization of the procedural algorithm a few configurations are needed. Some of them we will only describe in detail later in the description of the generation process. They include several new concepts which are hard to grasp without the context provided further on as we advance in the description of the process. So the five configurations used are:

Growth Control it is composed of several parameters that modify the way the map is created, it can influence the amount and richness of the terrain a player has or which surrounds

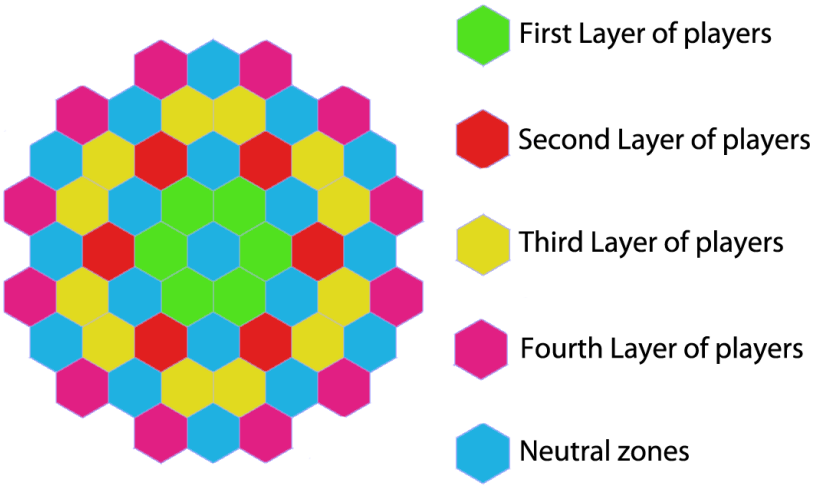

Figure 3.4: Layers of players and the order by which they are generated.

him initially;

Terrain Prototypes each prototype acts as a blueprint for the creation of many actual terrain units in the map. These include prototypes specifically for different purposes, such as races or neutral terrains, and are the basis for an adequate and balanced map generation;

Race Specifics composed of several characteristics different among the races of the game. These characteristics can be limitations, preferences or even terrain variations;

Terrain Features specifies which features should be created and in what amount;

Capitals Only even though the generator may generate several terrains owned by a player, sometimes it is useful that he only starts with his capital. The capital is his most rich terrain;

3.6.1.2 Create All Player Zones

At the beginning of this stage the generator has an empty map and the goal is to add all the players specified in the input stage to the map. We already know what pattern is followed in order to expand the map as much as desired but we must also specify how it actually grows in this static generation context.

Therefore, we need a way to actually expand the map as much as needed. However, such a schema needs to ensure the connectivity between all players and ensure the equality of challenges

presented to players. In order to ensure equal challenges we must ensure the described growth pattern is followed, regarding the connectivity problem there are several solutions, for example a pathfinding algorithm between all players would be suitable to verify it. However, we can create an evolution schema that ensures both these properties without the overhead of associated with pathfinding. To do so, we expand our maps based on hexagonal layers of players where we add them in ever larger layers starting from the center of the map (similar to the way Travian expands in circumferences, see 2.2.2.1). This growing schema is illustrated in figure 3.4, notice that for image simplicity each zone is represented as a single hexagon, but the result is the same. The number of players added by each layer l is then given by:

$$players(l) = \begin{cases} 3 * l & \text{if } l \text{ is even} \\ 3 * (l-1) & \text{if } l \text{ is odd} \end{cases} \tag{3.4}$$

However, to create the desired growth pattern and ensure connectivity we need to be able to describe how we add more players. To do this when more player start places are needed, they are created based on the ones from the previous circumference (layer). For example, considering figure 3.4 if we only needed to add 6 players then we would only use the green layer, but if we needed 7 then we would need both the green and red layers, where the red layer would be determined based on the green one and our growth pattern. To determine the new player start positions we use a localized (on the player start position) method, since it also has to be adaptable to a dynamic context where we have player addition done on a singular player basis. To achieve the specified growth pattern we have several options. The most straightforward would be to generate all the neighbouring positions of a given start position. However, this would originate many repetitions which would have to be checked and discarded, adding overhead to the algorithm. So we use a recursive method which diminishes the repetitions to a minimum and still follows the growth pattern. Based on the ancestor's direction of a player start place and its position we can determine which new player start places are created. Considering the grid system of Almansur there are six possible situations which are illustrated in figure 3.5. Notice that with this system we can also easily determine the neutral zone places of the pattern. In the middle of every two new player zones there is a neutral zone represented in figure 3.5. This information is gathered when new player start places are created since it is needed later on the stage "Randomize Neutral zones". We can then formalize the process of player places and

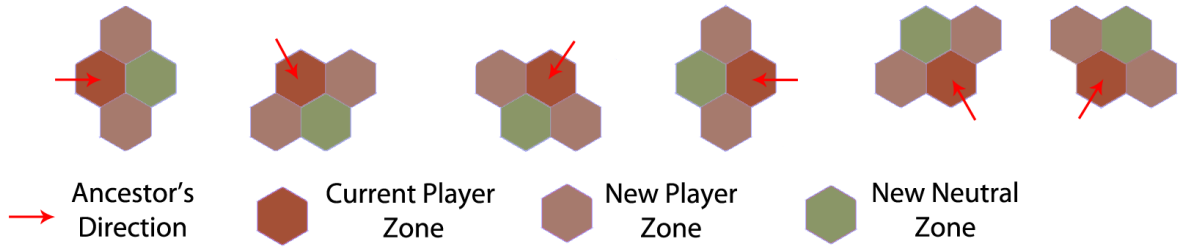

Figure 3.5: All the possible situations for Player and Neutral zone generation based on a player start place and the direction of its ancestor.

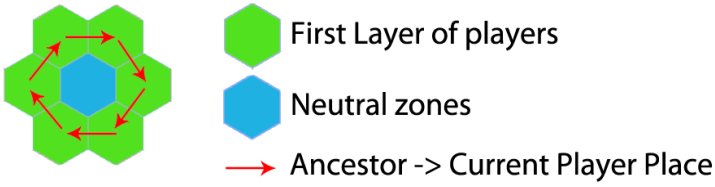

Figure 3.6: The initial layer of player start places and their ancestors. A red arrow points from the ancestor to the player start place it refers to.

neutral zone place generation by the following formula:

$$generate(x_{current}, y_{current}, direction_{ancestor}) = \begin{cases} \{x_{\text{new player 1}}, y_{\text{new player 1}}\} \\ \{x_{\text{new player 2}}, y_{\text{new player 2}}\} \\ \{x_{\text{new neutral}}, y_{\text{new neutral}}\} \end{cases} \tag{3.5}$$

The player start places generated can then be used to place players in our generation process. The available start places are used in a clockwise order starting from the northeast inside each layer of players, but another order could be used. A sequential order is used because later on in the 3.7 we will be using a similar player start placement ordering which benefits from this kind of ordering but other orders could also be used. Since the growing schema is recursive we need some initial player start places. The initial start places are the first layer of players, which is represented in figure 3.6. The process of actually adding the players defined in the input then follows two steps which we will describe next. However, before adding the players, they are usually randomly sorted so there is no repetitiveness in the player placement order in case of repeated input from the game designer. This randomization is not mandatory since the game designer might wish to create a map with a specific player positions. The goal now is to

iteratively create a player zone for each player which is done by the steps 3.6.1.2.1 and 3.6.1.2.2. We describe this process centered in on a player bases since this localization permits the method to later scale properly to the dynamic game context.

3.6.1.2.1 Create player specific terrain For this step the input parameters needed are:

1. Player Specific Size (PSS)
2. Player's Information
3. Terrain Prototypes
4. Player Terrain Composition (PTC)

Algorithm 3.2 CREATEPLAYER($name, race$)

Require: The *name* and *race* of the player to add and the PSS.
Ensure: The player specific terrains for the player is generated.

```
1: race_prototypes ← prototypes[race]
2: for i = 1 to NUMBEROFTERRAINS(PSS) do
3:     {We obtain the number of terrains to generate by formula 3.2}
4:     quality ← PlayerTerrainComposition[i]
5:     prototype ← RANDOMFROM(race_prototypes[quality])
6:     terrain ← GENERATE(name, prototype) {The GENERATE function follows the concrete
       game map terrain generation method described in 3.4 and formula 3.1}
7:     if i = 1 then
8:         SETCAPITAL(name, terrain)
9:     zone[i] ← terrain
10: return zone
```

Based on the player's race we choose the most suitable category of terrain prototypes for the player's race to be used on the generation of the player's terrains. Then based on the PSS and the formula 3.2 we calculate the number of player specific terrains. Finally until the the number of generated terrains reaches the number previously calculated, terrains are generated following the PTC configuration for terrain prototype quality and amount as well as the order of each quality used. The process described is presented in algorithm 3.2.

For example if we had a player zone with PSS=2 then we would have a total of seven terrains to generate. Then if we had a PTC like the one from table 3.2 we would have two terrains from quality five generated first, then two terrains from quality three generated and finally the remainder three terrains would be generated with quality inferior to two.

It is important to notice that while generating the terrains, a random and unused position from the player specific terrain is assigned to each, to avoid repetitiveness in case of repetitive input. For every terrain generated its owner is also assigned the current player's name. Finally the capital of a player is its most important terrain, which by convention is set to be the terrain generated based on the prototype with the highest quality (remember the quality dictates the usefullness of the prototype within its category). To further on be able to provide the game this information the position of the terrain generated with the highest quality is associated to that player's information on the generator.

At the end of this step a set of concrete terrains for all the player's specific terrain for a player zone has been created

3.6.1.2.2 Balance the terrains set In the previous step all the terrains were created by a randomized process and in an individual basis. The lack of evaluation of the player specific terrain as a set could lead to imbalances between players if the same kind of variations occurred in a single terrains set. For example if all variations were negative, then the terrains of that player would be much poorer than its enemies. The goal of this step is to minimize this problem by adjusting the set in case it exceeds the limits of what a typical set from the specified race should have. This permits to globally evaluate the terrains available to players and mitigate imbalances created by map generation which also contributes to game balance even before player testing.

For this step the input parameters needed are:

1. Terrains set from previous step
2. Player's Information
3. Races' Limits (from the Race Specifics input)

The races' limits parameter is part of the Race Specifics input and is selected based on the player's race information. The limitations consist of a map composed by pairs Characteristic - (averageValue, offset). For a given characteristic the range of values created by offsetting (positively and negatively) the averageValue specifies the possible values for that characteristic in a typical terrains set. The core of the process used for terrain balancing is presented in algorithm 3.3.

Algorithm 3.3 BALANCETERRAINS($terrains, race, limits$)

Require: The *terrains* to evaluate and balance, *race* of the limitations we should apply and *limits* all the limitations.
Ensure: The terrains are balanced as a set.

```
1: race_limits ← limits[race]
2: sum ← emptyHash
3: for i = 0 to LENGTH(terrains) do
4:     for all key in terrains[i] do
5:         if ELIGIBLE?(key) then
6:             sum[key] ← sum[key] + terrains[i][key] {a terrain is represented as a hash
               with pairs of (characteristic, value) see 3.1}
7: for all key in sum do
8:     if sum[key] > race_limits[key][max] then
9:         adjustment ← race_limits[key][max]/sum[key]
10:        adjusted ← true
11:    if sum[key] > race_limits[key][min] then
12:        adjustment ← race_limits[key][min]/sum[key]
13:        adjusted ← true
14:    if adjusted then
15:        for i = 0 to LENGTH(terrains) do
16:            terrains[i][key] ← terrains[i][key] ∗ adjustment
17: return terrains
```

For each terrain from the player specific terrains set and for each terrain characteristic eligible for adjustment (for example the terrain owner is not eligible) their values are summed. Next, for each characteristic if its sum value is outside the range of a typical set then it is adjusted so it fits the range. The adjustment is accomplished by scaling the values of that terrain characteristic in the complete terrains set. Notice that the adjustment made places the summed value of the set in the limit value but other adjustments could be easily made, like adjust to the average value.

3.6.1.3 Populate Players' Borders

In this stage all the neutral terrains belonging to player zones are created. The main difference from the normal neutral generation process lies in the fact that a percentage of them are influenced to also be the most suitable for the race from one of its adjacent players. This encourages player expansion and adds to the visual coherence of the map created. Notice that there can be more than one player since this part of the player zones overlaps.

The first step of this stage is to determine all the neutral terrains of the player zones. This

is done based on the occupied player start places and the PSS/NTS parameters. Finally for each terrain, we decide if it is going to be influenced by any neighbouring players based on a fixed probability previously set. If it is influenced then it is generated to best suit one of the neighbouring players otherwise it is created based on a random terrain prototype from the "neutral terrains" prototypes category.

3.6.1.4 Randomize Neutral Zones

In this step all the neutral zone places previously calculated on the "Created All Player Zones" stage are treated. They were determined while calculating the new player start places by the formula 3.5 that also calculated the neutral zone places.

The first step in this stage is to discover the neutral zone places to create. We will not randomize all since the last layer of player start places probably is not completely occupied, so neutral zones with no adjacent player start places occupied will not be created with neutral terrain. This reduces the number of terrains available for expansion to the players in the last layer of expansion and mitigates the advantage these players have because they have less direct points of conflict. Next we use the same normal randomization process for a terrain with a random terrain prototype from the "neutral terrains" category (explained in 3.4).

3.6.1.5 Create Map Features

The creation of map features is very important in strategy games. As such this step can create several kinds of features if the game designer desires it, since it is also possible to create none in which case this step is ignored. The generator currently supports two kinds of features, mountain ranges and other generic features. This separation exists due to the different nature of mountain features and others like swamps, deserts or lakes. Mountain features can span for great extents and form mountain ranges while swamps usually tend to be a localized agglomerate of terrains were that kind of terrain is most predominant. Notice that for each feature desired it is necessary to specify the probability in which it occurs and the terrain prototypes that can originate terrains for the designated feature.

3.6.1.5.1 Mountains When creating mountain ranges we wanted them to add to the map interest, but also be simple to create in order to keep our focus on the map generation and not on realistic mountain range generation. So even though several approaches could be taken in this phase we made use of the different characteristics of some races in Almansur, specifically the races with a mountainous nature. Therefore, to create the mountain ranges we base our method on using mountainous player zones as waypoints for mountain ranges, since they are already mountainous places. By analyzing the races available for fantasy games on Almansur there are two that have a mountainous nature: Dwarfs and Barbarians. Therefore, the eligible player start places for this purpose are chosen from all the start places occupied based on the owner's information regarding his race.

The next step consists in listing the possible mountain links (a connection from a given mountainous player to another, even though other linking system could be easily used). Based on the probability given by the game designer we then eliminate some of the links. Once we have the links we intend to convert them into actual mountain chains. To do this we create a concrete path in the map with an A* pathfinding algorithm (but different pathfinding algorithms could be used) which avoids player specific terrains and impossible paths (NTS can be 0). This path is then randomized to eliminate the distinct shape of direct paths. Finally, the terrains on the paths created are replaced by mountainous ones created with the normal randomization method and a special category of terrain prototypes especially created for mountain features. Notice that the creation of these terrains follows several iterations to simulate the gradient existent in nature where higher mountains are surrounded by lower mountains (higher and lower mountains can be distinguished by the prototype quality). An example of mountain features can be observed in figure D.2 from appendix D.

3.6.1.5.2 Generic Features Generic features are all the other features the game designer might wish to include in the map such as: deserts, swamps, forests and lakes. As in the mountain range generation we took a simplistic approach (even though other more realistic models might exist) since this was not our main focus. Therefore, our method uses the idea that these features have an agglomerated nature where we can use the neutral zones as a base for them. In our algorithm we create feature zones that start at the center of a neutral zone and can expand through the neutral terrains of the player zones. Based on the scale of the tiles in Almansur we then fixed the maximum expansion to be from the center of the neutral zone until half the

length of the side of a player zone. It can be expressed by:

$$maximum\_expansion = NeutralZoneSize + PlayerSize/2 \tag{3.6}$$

In our feature generation algorithm the first step is to find which neutral zones will be changed into feature zones. From the set of all the neutral zone places available we remove the ones already changed to feature zones and we further restrict them with the probability of the feature we are creating. Next, starting from the central place of each zone and expanding in ever larger layers around it we randomize the terrains with the terrain prototypes of the desired feature until we reach the maximum expansion (see equation 3.6). In each layer of terrains we use different qualities of prototypes in order to create a gradient of abundance regarding the feature terrain characteristic as it happens in the nature. An example of these features is presented in figure D.2 from appendix D, where there is a lake generated by this method.

3.6.1.6 Randomize Outer Coastline

The player and neutral zone schema used is very useful to distribute the players in the map in a strategic way but can originate an easily recognizable shape, especially in the borders of the terrain from the map due to the growing pattern based on zones. Since we also aim to reduce the visual impact on the players caused by these unnatural shapes this stage will randomize all the outer terrains which will be the coastline of the map. Later on the generation process the adjacent terrains to the generated map will be filled with water tiles.

The first step is to discover the area of terrains that are part of the map limits. The easier way to understand what zones can be part or not is by looking at an example situation illustrated in figure 3.7. As we can see in this figure the zones that can be part of the limits are the complete next layer of growth (both player and neutral zones), the neutral zones from the current layer and the free player zones from the current layer also. We add all the neutral zones from the current expansion layer because the players placed, in this layer already have the advantage of having less neighbours. With this randomization they will have slightly less terrain to conquer than inner players. After having these zones we can then determine all the terrain to randomize based on the zone sizes. Next the idea is to iterate on a terrain basis, where based on the test if a terrain is a water one and fixed probability (set by the game designer) we either change it

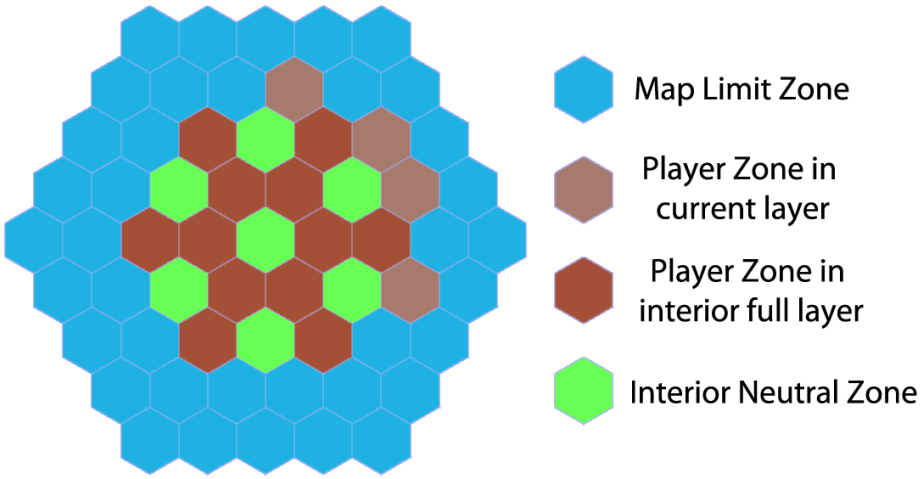

Figure 3.7: The limit zones from a map with two complete and one incomplete player layers of players.

to water or we leave it unchanged.

Finally we have an additional step which is the actual creation of a water area around the map. As a result from the previous generation and randomization process, players can sometimes have access to ungenerated terrains. To avoid this and add a sea effect around the map for players , we create a sea area surrounding the map (which takes into account the players Fog of War(FOW) value). This effect can be observed in the maps presented in appendix D.

3.6.1.7 Player Terrain Adjustment

In this stage the players' owned terrain is changed in case it is needed. The input parameter "Capitals Only" is used to decide if a player only starts with its capital or if it starts with all the terrain generated for him. This option was created because if we make a player only start with his capital he has to expand his territory by himself and we also eliminate recognizable shapes from the terrain generated for the player since he only starts with a terrain. Other approaches could be as easily used, for example instead of starting only with the capitals, start with a random subset of the generated terrains. However, this would add some new problems like the connectivity of the terrains, moreover from this random subset in different players it would originate imbalances due to the different terrains chosen. Our method complies with our goals and does not bring any problems which are not our focus.

The process of reducing the terrains of each player to their capitals only is very simple

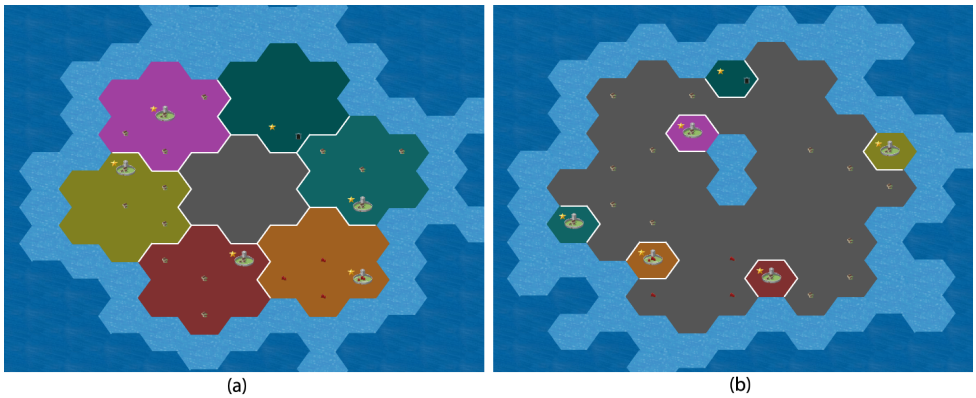

Figure 3.8: The effect on the ownership of terrains by the parameter "Capitals Only".

and straightforward, we remove the ownership of any terrain except if it is the capital of a given player. An example of this process can be observed in figure 3.8 where we present two maps generated with the same inputs except the parameter regarding the capitals. This figure represents the political view of the generated maps, where each color (except blue and gray) represents the terrains owned by different players. We can observe the differences in shapes and territories between a capitals only map (presented in b) and a map where all the terrain generated for a player is assigned to him (presented in a).

3.6.1.8 Map Created

The map and its players' information is created and now all that remains is use it in some context. Since all this process works on a modular basis (totally independent) regarding Almansur all that remains is to integrate it with the game.

3.6.2 Integration in Almansur

The development of a concrete implementation following the generation process described in the previous chapter was accomplished. The solution developed evolved in terms of its integration with Almansur regarding two main aspects: the map creation interface and the map importation procedure.

3.6.2.1 Map Creation Interface

The map creation interface enables the game designer to parameterize the map generator and create the maps. The inputs described throughout the previous chapter were implemented in the following files: players.csv; races_terrains.csv;races_limits_and_variations.csv; map_properties_static.xml. Examples of concrete files used can be found in the appendix B.

File Editing and Command line Initially all the parameterization of the map generator was done by manually editing the input files. After this in order to generate the map it was needed a direct script command to issue the generation.

Visual Interface The manual editing of every option was very cumbersome for untrained game administrators and the script command issue to the creation of the map was even harder. To solve this a visual interface was created to reduce the file editing to a minimum and eliminate the need of knowledge about scripting. The parameterization is now done through an administrator view of the Almansur game.

3.6.2.2 Map Importation Procedure

Once a map is created there must be a way for the generator to communicate and import the map into Almansur's game server. The process of importation for the maps generated in the classic context uses the map importer already implemented in Almansur. This map importer deals with information in a specific convention which we had to comply to, but also brings the benefits that all the game model map representations are created just based on the information transmitted by this convention.

File Based Importation This was the first importation procedure to be used because of its portability and ease of use. The last property is due to an already existent map importer in Almansur and the portability is exists because it requires the creation of the files which comply with the importer's conventions and are easy to transport between different game servers.

Direct Importation However, the files which provide portability in the previous method are many times unnecessary since the generation of the map and the importation place is the same.

Therefore, to eliminate the creation of unnecessary files and speedup the generation/importation process we changed Alamnsur's map importer to directly import the generated maps into the game.

3.6.3 Balance

The game environment is based on a map with many different characteristics which must be carefully balanced in order to provide each player the necessary terrain but never offer any superiority.

A first step towards map balance it the player/neutral zone distribution schema. It creates a map pattern where a player is surrounded by three other players and three neutral zones. This distribution was chosen based on previous (to our method) game tests and tries to balance the amount of direct conflicts a player has to manage. If a player had six other player zones as immediate neighbours, there would be too many potential conflicts points for the player to manage, and the diplomatic aspect of the game would predominate. This would destroy the balance between military and diplomatic strategy in the game. This schema then provides a good balance between direct and indirect points of conflict which promotes all the components of strategy from Almansur.

Based on our schema the responsibility to control this balance rests on the game designer through the correct parameterization of the map generator. First is the specification of the prototypes for the terrains. Since we have the same PTC for all races then the prototypes should be balanced by creating them with a direct correspondence in terms of quantity/usefulness to its race, amongst the several races prototypes categories. Based on this and a carefully defined "neutral terrains" category which contains prototypes suitable for all the races, the generator creates equally useful player zones for all races. This is the basis for player balance but only through player testing the best balance can be achieved.

Even though the previous ideas aim for balanced player zones, the process of terrain creation can still originate imbalances because each terrain is created individually with randomization. This problem is solved in the generator by having a step in the player zone creation which adjusts the set of player specific terrains if they exceed the limits of what a typical set from the specified race should have. This process was described in 3.6.1.2.2.

However, there is still the possibility of map imbalances occurring. If there is a significant difference in the number of players that choose a given race and the others, this means the terrain composing the map will be more suitable for the race most chosen. This is an imbalance hard to mitigate, but a small attempt to do so is the fact that all neutral terrain is randomly generated from a set of terrain prototypes equally suitable for all races. It helps because regardless of the races chosen by players, among all the neutral terrains generated there will always be terrain suitable for all races spread across the map. On the otherhand the map features creation diminishes the effect this prototypes suitability distribution has since, most of the times, it inherently creates terrains more suitable to some races than others.

Even though many of the efforts are towards the game balance, in strategy games the existence of strategic challenges regularly conflicts with this notion. The map features are a common source of imbalance. Aside the strategic challenges they might create for some players the terrain they contain is very specific and may not be suitable for all races. For example a desert feature contains terrains equally unuseful for all races but a swamp feature contains terrains much more useful for Orcs. Again the correct parameterization of the amount of features and its prototypes is essential and the responsibility of the game designer. One way to try and avoid the advantages the feature terrains offer is to create these terrain features poor in terms of resources. However, since Almansur is a game of military strategy such strategic challenges are normal and not a serious problem if no great advantage is offered by the resources in these terrains.

3.7 Dynamic Multiplayer Game Maps

This part of the solution aims to create multiplayer interesting and balanced game maps dynamically. The maps from the previous context are already adptative regarding the player's choice of race, but in this context where the map grows as players subscribe we make our generation process also adapt to the flow of players that are subscribing a game at a given time. This is essential for the scalability of complex environments in massively multiplayer online contexts. It also means that this generation process occurs in several phases where their actual sequence is unknown since it depends on the amount of players entering the game at a given time.

The generation process is similar to the one presented in the previous chapter. However, now we aim to create the terrain for a given player only when he decides to join the game. With this dynamic model we intend to expand the map creation to a TSMBG context where the player can always freely choose his race. Nonetheless, it also brings new technical challenges to which we will describe our proposed solution.

3.7.1 Generation Process

In order to create dynamic multiplayer game maps an asynchronous sequential process is used where the map data is processed in several phases each containing several stages. The asynchronous nature enables the process to accommodate the unpredictability of player subscriptions. Since we cannot predict when players are entering, we cannot know the sequence of player subscriptions and turn processing. Therefore, we have to execute the stage of player creation asynchronously on a "as needed" basis. The sequential execution enables to create all the desired map characteristics progressively, since there are data interdependencies between phases where it is imperative that each executes alone and uninterrupted.

Our generation process has three phases which are illustrated in figure 3.9. The first phase is the configuration where all the necessary parameters needed for the generation process will be loaded into the generator. The second is the player generation, this phase adds a player to the map based on the choices he did. The third phase is the turn processing phase where the map generator updates the map if needed. Even thought the process is asynchronous there are some "rules" to the generation process:

- the first phase only occurs once, at game creation.
- the second phase occurs each time a player joins, so it can occur many times or none during a game turn.
- the third phase occurs once for every turn processed.

The described asynchronous process can be seen in the interaction diagram from appendix C.1.

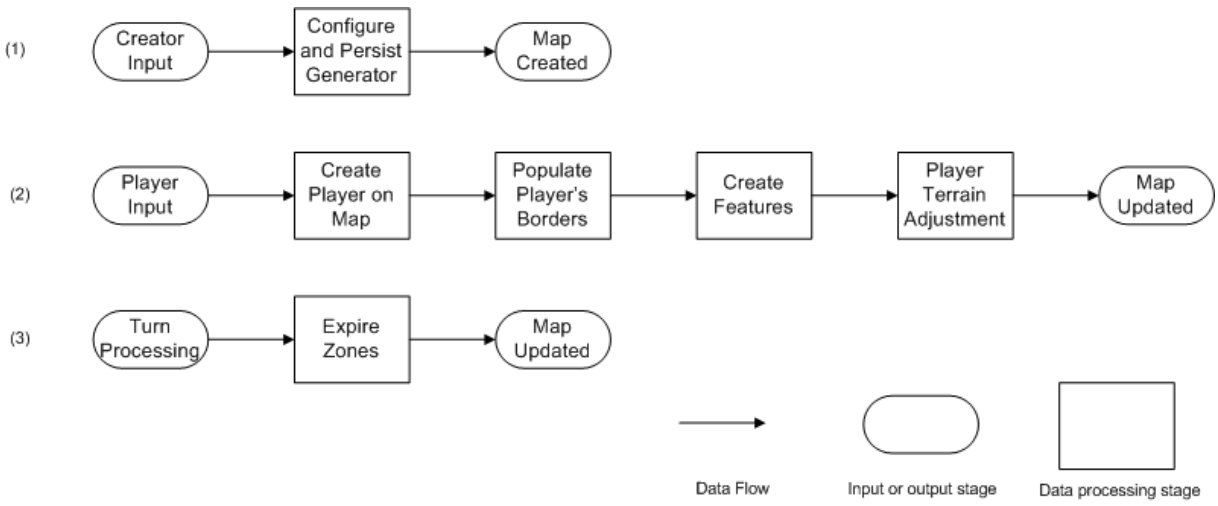

Figure 3.9: Data flow, phases and stages for dynamically created maps.

3.7.1.1 Configuration

This phase of generation is very similar to the "Creator Input" stage in the Classic Multiplayer Context part of the solution. Since our generation process is now asynchronous and we have player subscriptions at different game times, we need to persist some of the information used, like the terrain prototypes (since they are always the same during the complete process). If we did not persist them we would have an overhead on information loading each time a player subscribed. Additionally, the generation process depends on both the already generated map information and the new player information, so we need to persist some map information. Therefore, all the non-player configurations should be loaded and saved (persisted) in this step for later usage on subsequent generation executions contrary to the previous approach of read, use and destroy since the information was needed just during that one execution. There are several processes of persisting data but how this is actually done is not relevant for the description of the generation process. Regarding this subject it is enough to say that the configurations are persisted in this phase.

For the current generation process we will still need all the non-player configurations previously described in 3.6.1.1. Additionally we need one new configuration, the number of turns to expire. As we will see ahead there will be player start places generated that can become unavailable. This parameter specifies how many turns can elapse before an unused player start place becomes unavailable.

3.7.1.2 Player Addition

This phase is also very similar to the sequence of stages 3.6.1.2 to 3.6.1.7 from the previous chapter. However, there are two major differences, first only one player is added each time this generation phase is executed and second the actual map expansion is different.

The map growth idea is still the same as explained in 3.4 but since we now intent to have a map expansion driven by the flow of player subscriptions, the player start place creation and usage will be different. Now we want the map to grow one player at a time, in order to have the desired dynamic map, and the expansion as formalized in 3.5 but only when a player enters and not on a complete layer basis. The determination of new player start places is still done in a localized manner as in the classic context, on a single player subscription, and happens when a new player subscribes.

Even though we now subscribe one player at a time we still need to ensure the connectivity between players, furthermore, since we have players subscribing at different game times we want the differences between player subscriptions to be the smallest possible. The connectivity between players could be ensured in several ways, but most of these would not enable us to have the smallest difference between subscription times. In order to ensure both properties we expand our map in a spiral order for both player and neutral zones. This ensures that a new joining player will be placed in the oldest unused player start place available so the elapsed game time between new players and the ones already playing is globally the smallest possible. It is important to note that even though the neutral zone placement also follows the spiral scheme, the actual placement only happens when a player zone that should be placed immediately after is created. This spiral schema is illustrated in figure 3.10. This figure illustrates a normal evolution, where there is always players entering. Notice again that zones are now represented as just one hexagon for image simplicity, but the result is the same. On the right we present the sequence of evolution by showing the state of the map throughout the player addition iterations (where we present both used and unused but available player/neutral zones) and on the left we show the complete evolution, where each number represents an expansion iteration. Therefore, when the first player joins a neutral zone and a player zone are created, when the second player joins a player zone alone is created, and so on generating the interleaved pattern of players and neutral zones.

However, player entries are not predictable and so we can have long periods of time where

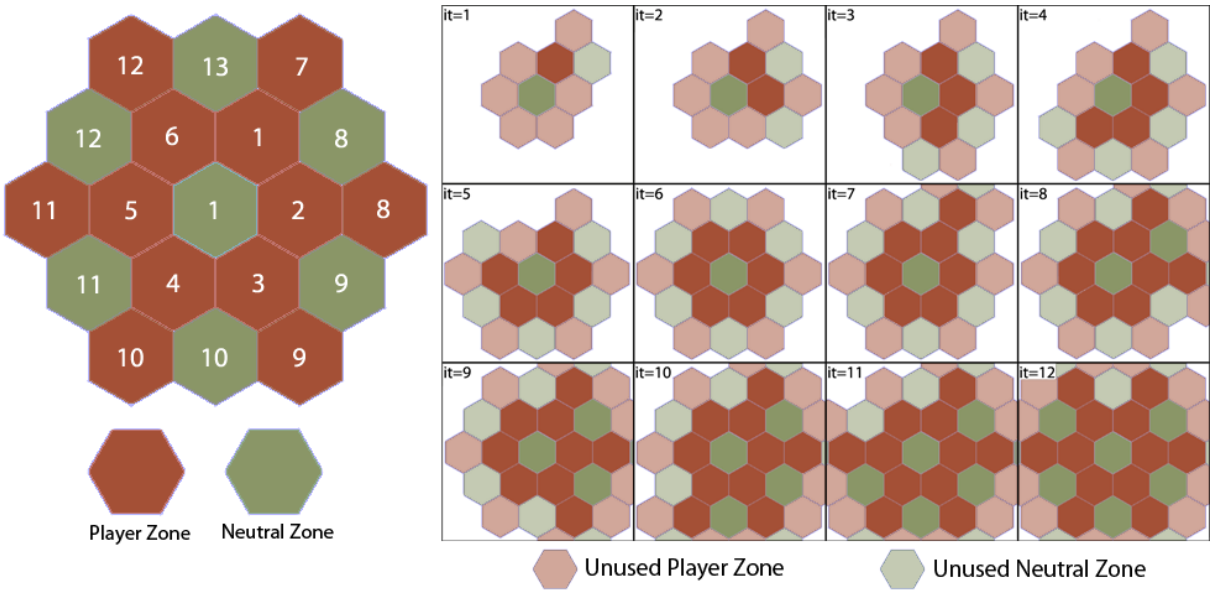

Figure 3.10: Normal spiral expansion. The complete evolution on the left and the sequence of evolution iterations on the right.

no players subscribe. If we followed a strictly spiral order as previously described then the time difference between some player subscriptions would be big. This could then easily originate situation where the new player would be easily overrun by an older one. In order to solve this our map must adapt zones where players do not enter for a long time so that they cannot be assigned to new players but at the same time enable the map to still evolve. Therefore, if there are no player entries for a previously fixed amount of game time (in our case turns) the generator marks as expired the neutral and player zones for which the initially set parameter number of turns to expire has been reached. This procedure limits the head start a player can have regarding any of his neighbours. The expired zones are generated as neutral impassable zones where expansion stops, notice that this is also a way to limit the connectivity of older players to newer ones. If we had generated normal neutral terrains the connectivity would be maintained and older players would still be able to overrun newer ones. To accomplish these kind of zones, we create them as natural barriers, like a lake or a very high mountain impossible for game units to cross. As a consequence the map will not grow in a pure spiral fashion but will only continue to expand in a few places. This system enables the map to expand or contract the places for expansion depending on the flow of players. An example of such a situation is illustrated in figure 3.11. On the right we present the phase execution sequence and its impact on the map and on the left we present the final map where each number represents the players'

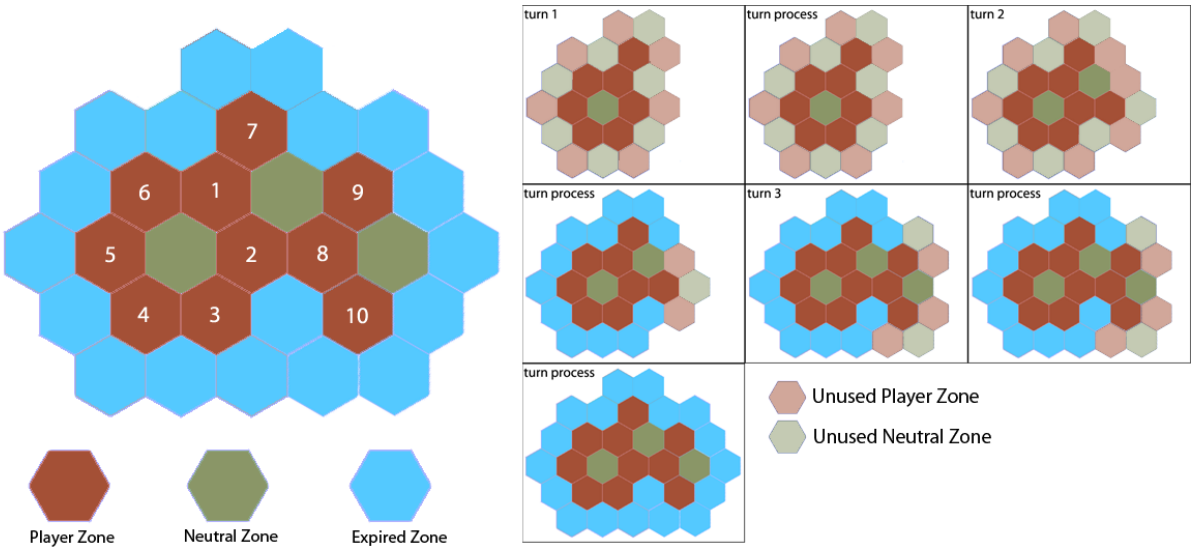

Figure 3.11: Spiral expansion with expirations. The complete evolution on the left and on the right the sequence of phase executions with the corresponding map updates.

order of subscription. In this example we are assuming a "turns to expire" value of two turns. In the first turn seven players subscribe, then a turn is processed, but since the turns to expire is two we still have no expirations and no map updates are needed. In turn two one more player subscribes. Next when the turn is again processed the remainder unused player start places originated from the players which subscribed in the first turn, and the corresponding neutral zones, are expired and the map is updated. On turn three two more players subscribe and on the next turn processing no map updates are needed since there are no unused zones available for two turns. Finally no more players subscribed on turn four and when the turn is processed again the map closes itself since the remainder unused player start places expired.

The result from the previous example shows an important property of these maps. If there are no players entering for a long period of time the map closes itself with natural barriers. When this happens the game continues with the players that are already playing but no more players can join.

After understanding the differences in the map expansion we can now describe the current phase, which is composed by the following stages:

1. Create player on the map;
2. Populate player's borders;

3. Map adjustments

3.7.1.2.1 Create player on the map An important concept in the current generation process is that the spiral order of player and neutral zones generation is kept by the means of a sorted list. Throughout the description of the generation process we will call it "places order" or PO.

To generate the player specific terrain the same process described in the two steps from 3.6.1.2 is followed. Having the set of terrains belonging to the player we need to actually add them to the map. Before adding the terrains to the map the algorithm checks if the map is already closed or not, because if it is then the player cannot join and the algorithm ends. We detect if a map is closed by checking if there is any free player start places.

Next we populate all the neutral zones that precede the player zone to ensure the spiral schema, basically by retrieving all the neutral places of the PO list which appear before the player start place. Next we determine every terrain position belonging to these based on the NZS (formula 3.3) neutral places. Then for each of these terrains the normal process of generation is followed, a terrain prototype is randomly chosen from the terrain prototypes category "neutral terrains" and it is used to generate the terrain.

Next we need to ensure that the growth pattern is followed and that the map expansion continues. Contrary to the classic context were we simply created new layers of players, we now have a localized evolution and therefore the map expansion regarding the addition of new free player start place(s) and the corresponding neutral zone are determined based on the start place we are occupying. We do this by adding them to the PO list, sorted in order to follow the spiral order (1st quadrant, 2nd quadrant, 3rd quadrant and 4th quadrant). To the new start places we are adding we attach an information specifying in which turn they should expire, this value is the sum of the current turn plus the Turns to Expire input parameter value.

Finally we actually add the terrains from the player specific terrain to the map.

3.7.1.2.2 Populate player's borders As in the previous generation process (in 3.6.1.3) we need to populate the neutral terrains from the player zone. However, if we look at the growing schema we see that the neutral terrains from two different player zones overlap. Therefore, when we are populating these terrains we must exclude from the generation process the terrains

already created in previous executions, otherwise older players would see map changes in their views. To determine the ungenerated terrains we use the PSS and NTS, where the NTS dictates the number of hexagon layers to consider around the player specific terrain. Next we exclude the terrains that are already on the map from the layer previously calculated. Finally for the remainder terrains the normal process of generation is followed, a terrain prototype is randomly chosen from the terrain prototypes category "neutral terrains" and it is used to generate the terrain.

3.7.1.2.3 Create Features The creation of terrain features in the dynamic maps follows the same feature control as in the previous chapter (see 3.6.1.5). However, it is difficulted by the fact that we can only create features on the map part that we are adding to the game, otherwise already playing players would see the map changing around them. All the other terrains previously added to the game cannot be changed. The greatest impact on the feature generation regards mountain features, since we will now exclusively focus on single neutral zone places.

For mountain features it is now impossible to follow the same method. As such we create simpler mountain features based on a single neutral zone place. More complex methodologies could be explored, but those are not the focus of our work and this approach still offers the strategic and visual aspect inherent to these features. We follow the same idea in which we link two terrains, now the two highest around the neutral zone, by a direct and randomized path. Next we again replicate the gradient of mountains as previously explained.

For the other features we again use a single neutral zone approach. We generate these zones in a similar way regarding expansion from center and in ever larger layers of terrains around it and regarding the gradient of terrains created. However, we take into account that we can find terrain previously generated, and in these cases we do not replace them to avoid map inconsistencies for players (by seeing terrains change in their map views).

3.7.1.2.4 Player Terrain Adjustment As in the previous section the "Capitals Only" parameter is used to dictate whether the player starts with only his capital territory or if he starts with all the player specific terrain from the player zone. The process is similar to before, we eliminate the owner of all player specific terrain that is not the capital of the player. However,

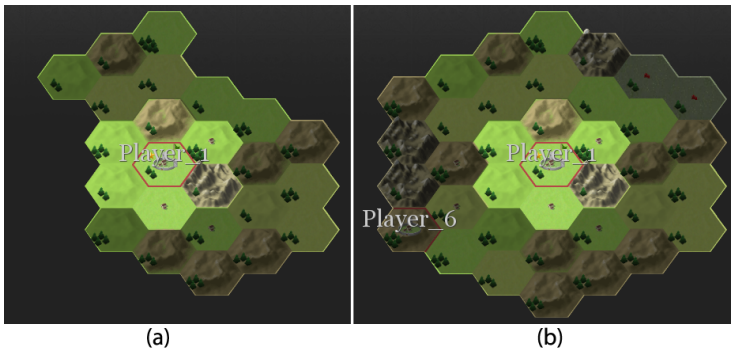

Figure 3.12: An example of a map view update. View at subscription time in (a) and in (b) the view after all the Player_1's neighbour player start places have been occupied.

in this case it has an additional importance, since if players only start with their capitals, this means they will see less terrains in their initial map view. Therefore, this reduces the impact caused by map view updates, since players see less terrain and thus have less terrain updated in their views when new players are added to the map, even though the same amount of terrains is generated. In figure 3.12 we have an example of a map view update for Player_1. In (a) we can observe zones of ungenerated terrain, in the FOW region of the player's view, and in (b) we can observe the same view after all the neighbouring player start places have been occupied where there is no ungenerated terrain.

3.7.1.3 Turn Processing

In order to ensure the evolution with expirations, needed to maintain the map balance in the online context, we need to process expirations and update the game map when needed as a game turn is processed. Additionally, we address the problem where the expiration of zones interferes with the correct growth pattern and players could sometimes (as we will detail ahead) stay in contact with ungenerated zones for long periods of time which is not desirable since these are unnatural regarding a normal strategy game gameplay.

The methodology followed in this phase is presented in algorithm 3.4, where the first step is to advance the number of turns from the generator, so it can properly verify the expired zones. Next we check if the map is already closed, if it is then we report that it is already closed and there is nothing to be done. Next we must check for places that might have expired when we advanced the turn (see algorithm 3.5). Based on the unused player start places and the information of when they should expire we easily mark the start places that should be expired.

Algorithm 3.4 TurnProcess

Ensure: The map is updated with the expired zones.

1: $current\_turn \leftarrow current\_turn + 1$
2: **if** $map\_closed = true$ **then**
3: **return**
4: $expired \leftarrow$ CheckExpiredFreePlayerPlaces()
5: CheckExpiredNeutralPlaces()
6: CheckExpiredPlayerPlaces()
7: ExpireZones($expired$)

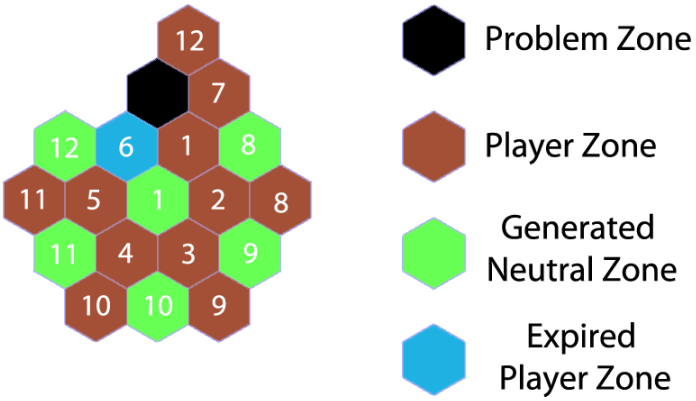

Figure 3.13: This figure illustrated the "neutral ungenerated problem" where a neutral zone can stay ungenerated for a longer time than it is desired to be. The numbers illustrate the order by which the zones were created. If you consider the "Player and Neutral zone generation" method illustrated in 3.5 and the initial generation directions illustrated in figure 3.6 you can see how the problem zone emerges.

Each start place that has a turn to expire smaller or equal to the current turn is marked for expiration.

The next step is to check for neutral zones adjacent to players already placed that are not going to be generated and must be filled so the player does not stay in contact with ungenerated terrain for long (this process is presented in algorithm 3.7). Such a case is shown in figure 3.13. This specific problem could be avoided if the generation of new player and neutral zones were not made by the recursive approach we use. If we used the other solution (as discussed in 3.6.1.2) where all the neighbouring zones adjacent to a player start place were determined when expanding the map we would not have neutral ungenerated zones. However, we would still have this problem with ungenerated player zones, which as we will see next can also occur. Additionally, we would also be adding an overhead in the verification of repetitions of new player start places and neutral zones which would increase subscription times for players which is very

undesirable since players usually have a low tolerance for larger waiting times. It is better to add this overhead in the turn processing where all the game turn information is processed and there is already a much longer waiting time where such an overhead has no impact.

Therefore, in order to do this check once a player is added to the game, he is also added to the players to which we must do this check. Then in this verification, for each player we must take several steps. The first is to determine all the neighbour places and exclude all the generated places and player start places (whether generated, free or marked for expiration). If after the exclusion we have an empty set of places then this player can be removed from this check, since all his neighbour neutral places have been created. In case there are neutral places we must then verify if they still have any unused, unexpired and already generated player start place adjacent to them. If there are then this neutral place can still be generated and nothing is done to this one. Else we must expire the terrains corresponding to this neutral place. If at the end of checking the neutral places we end up filling all the places we were checking then the player can be removed from the players to check on the next turn processing.

Algorithm 3.5 CheckExpiredFreePlayerPlaces

Ensure: It marks all the free player start places that have expired.

```
1: expired ← []
2: for all place in PO do
3:     if PlayerPlace?(place) then
4:         if current_turn >= ExpirationTurn?(place) then
5:             expired ← expired + place
6:         else
7:             return expired
8: return expired
```

To expire these neutral zones we create a natural barrier in order to maintain balance, by limiting the connectivity between old and new players. These natural barriers are created with water terrains, since is the best way (in Almansur's case) to ensure no player will be able to cross it and reach ungenerated terrain and at the same time it does not offer any undesired advantage to the player(s) near this zone. However, if we simply generated all the neutral zone terrain as sea terrain our map would have many unnatural shapes. Expired neutral zones would appear as "big" hexagonal sea zones on the map. Since sooner or later every dynamic map has many expirations, which would be very visually uninteresting and easily noticeable by the players. Therefore, to generate these zones we also have a specific algorithm (3.6). First we generate the

terrains set as sea then we randomize the set based on any coastlines adjacent to it.

Algorithm 3.6 ExpireZone($place, size$)

Require: The *place* of the zone to expire and the *size* of the area to expire in centered on that zone.

Ensure: The map is updated with the expired zone.

```
1: prototype ← RandomFrom(prototypes["coast"])
2: expired ← emptyHash
3: for all position in GeneratePositions(place, size) do
4:     if Eligible?(position) then
5:         expired[position] ← Generate(nil, prototype)
6: RandomizeZoneCoast(expired)
7: for all position in expired do
8:     Map[position] ← expired[position]
```

The set randomization is used to randomize both player and neutral zones and separates the set terrains into two categories, the eligible for randomization and the ones not elegible (for example the NTS layer of a player zone that is shared with possible future players). Next for all the eligible terrains we generate a coast (shallow sea) terrain. The coast randomization process is then applied, starting with the terrains (from the set) which have terrains near them with an altitude greater than 0. Then we iterate all the terrains to randomize and decide if we replace it based on a fixed probability . If we do the replacement is generated as a normal neutral terrain, based on the "neutral terrain" prototypes category. Each terrain that is replaced is removed from the eligible category. Then we update the terrains (elegible) to randomize to the neighbours of all the terrains we replaced in the previous iteration, excluding the ones already replaced or not eligible. Finally before the next iteration we decrease the probability value to further add to coast randomization.

Next we must also check for player zones adjacent to neutral zones already generated that are not going to be generated (as it happened with the neutral zones adjacent to player zones) and must be filled so players do not stay in contact with ungenerated terrain for too long. Such a case is shown in figure 3.14. Once a neutral place is added to the PO list, it is also added to the neutral places to which we must do this check. Then for each neutral place to check we use the same process. If a neutral place has already been expired we remove it from the list to check. Else we obtain all the neighbour places (which are all players as we can see by the growing pattern in 3.1) and exlclude the ones that have either been expired or created. If no place remains to be checked then this neutral place can be removed from the ones to check. Else

Algorithm 3.7 CheckExpiredPlayerPlaces(*place*)

Require: The *place* of the player zone to which we want to verify the neighbour neutral zones for expirations.

Ensure: The player zone is not surrounded by any expired ungenerateerd neutral zones.

```
1: neutrals ← GetNeutralNeighbours(place)
2: neutrals ← neutrals − GetGenerated(neutrals)
3: if Length(neutrals) = 0 then
4:     return
5: else
6:     for all neutral in neutrals do
7:         neighbour_players ← GetPlayerNeighbours(neutral)
8:         neighbour_players ← (neighbour_players ⋂ free_player_places) −
           expired_player_places)
9:         if Length(neighbour_players) = 0 then
10:            ExpireZone(neutral) {for more details on ExpireZonesee algorithm 3.6}
```

if there is in the neighbours to check any place that is still unused and unexpired we skip the checking of this neutral place since these places can still be generated. If not then we expire all the ungenerated player start places neighbour to this neutral zone. The algorithm described is very similar to 3.7 with the difference that we are checking for player start places to expire instead of neutral places.

To expire the player start places marked for expiration we follow the same approach as the one used for the neutral zones, the difference is that we have a different set composed of PTS and NTS, where NTS is inherently shared by players. In player expirations is common that the number of initial elegible terrains is smaller than the number of terrains in the set since players share terrains from their neutral terrains in their player zones.

The next stage in this phase is to actually expire the players marked for expiration. To do this we expire every place (neutral and player) starting from the beginning of the PO list until we reach the last marked player to expire.

The final stage of this phase is to check if there is no more unused player start places, if that is the case, we close the map and report it.

3.7.2 Integration in Almansur

This part of the solution was also implemented in Almansur. The integration process was different from the previous, since the generator's life cycle now accompanies the game's life cycle.

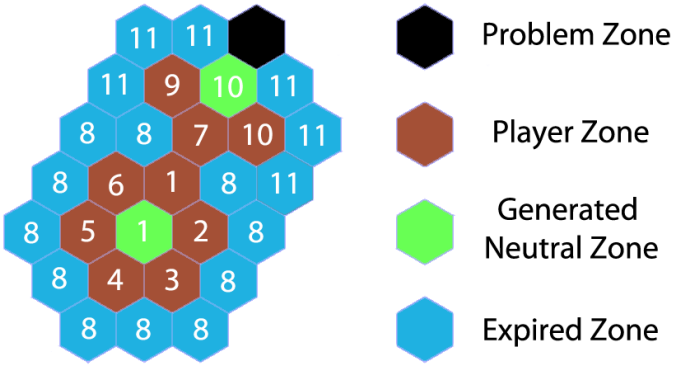

Figure 3.14: This figure illustrated the "player ungenerated problem" where a player zone can stay ungenerated for a longer time than it is desired to be. The numbers illustrate the order by which the zones were created. If you consider the "Player and Neutral zone generation" method illustrated in 3.5 and the initial generation directions illustrated in figure 3.6 you can see how the problem zone emerges.

In a game with a dynamic map when the game is created a generator is also associated with it. Next the game map can grow with new player entries and have several expirations with turn processing. To do this the generator creates all the new information and provides the game all the needed updates. The dynamic games are dependent on the map generator for game map evolution, and the generator only ceases to be necessary when the game map is closed.

The solution developed evolved in terms of its integration with Almansur regarding three main aspects: the map creation interface, the generator's state representation and the map importation procedure.

3.7.2.1 Map Creation Interface

The dynamic map creation started with a visual interface since there is an inherent association between the generator and the game. The initial interface only permitted to configure game options and no generator configurations. The next step was to also allow generator configurations and we created an improved version of the same interface.

After a dynamic game was created and made available for the players, they could start subscribing to this game. Since the old interface for game subscription was only ready to deal with completely generated game maps a new one was developed to be able to give the choices of player name (usually called "land" in Almansur) and race that each player wanted to play. These interfaces are illustrated in figure 3.15 on the left is an example of the old interface and

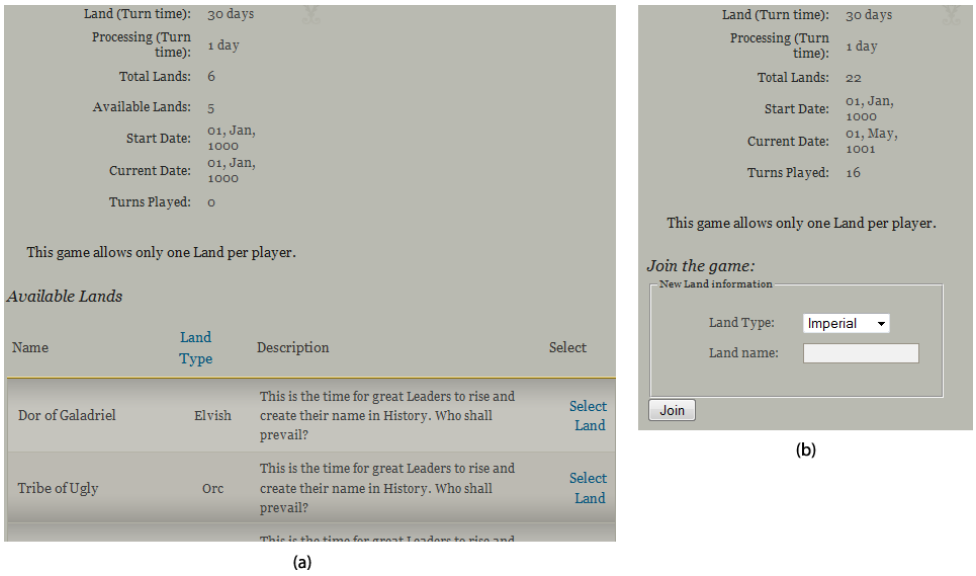

Figure 3.15: The subscription interfaces for both a classic multiplayer game (in a) and a dynamic multiplayer game (in b).

on the right we have the new one.

3.7.2.2 Generator State

In order to use an asynchronous generation model we need to have a way of representing the generator's state and persist its information. This enables us to use the generator, then unload it from memory on the game server, and load it later on with the same state as it was before unloading it. Our integration regarding this subject evolved in terms of the compromises and performance each one offered.

Files Our first approach aimed to maintain the independence that the generation module always had regarding the Almansur game engine. The persistence was based on files, and on a first approach we implemented a visitor pattern(Gamma, Helm, Johnson, & Vlissides 1994) which collected only the essential data that should be persisted. However, as we can see in figure 3.16, in the data represented in blue this method had a very bad performance where we had a rapidly growing time increase between subscriptions since at the 100th player we have a subscription time around 19 seconds which is unacceptable for a user waiting to play. We investigated the lack of efficiency and the problem was related to the fact that we were only gathering specific information where a global marshaling (operation native to the operating

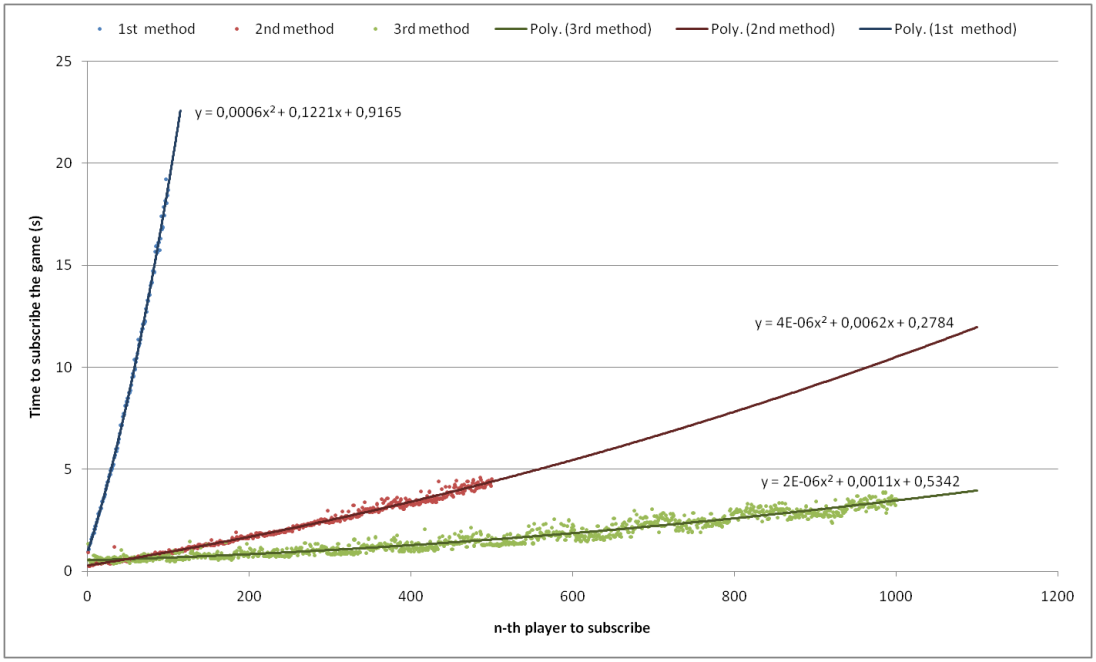

Figure 3.16: Comparison of different performances regarding player subscription, running on a laptop computer (Intel T7700 - 2,4GHz).

system) approach would be more efficient. With this approach we still ended up having a polynomial time increase but with a lower rate of increase for each player addition as we can see in figure 3.16 in the data represented in red, where we could subscribe around 400 players with a subscription time under 5 seconds. Additionally, this approach created files which difficulted the debugging process since they were humanly unreadable.

Database The files approach did not satisfy our scalability requirements where we wanted to be able to subscribe over 1000 players with acceptable subscriptions times. However, in order to increase the scalability of the method we needed a faster way of persisting the generator's state since it still was the phase where the most time was spent. To do this we lost part of the independence from the game engine but also gained an easier debugging process by integrating the generator with Almansur's game model. Almansur as a browser-game also has the need to persist its data, and it does it efficiently through an Object-Relational Mapping(ORM)(Ambler 2003) system which maps a given application domain model into a database seamlessly. Our next step was to convert our generator's model to integrate the ORM pattern and be represented in the database. To do this we had to map all the classes used for dynamic generation in the database schema. For a complete domain model used in the database mapping see appendix

C.2. After that, since we were now in an ORM schema we also had to adapt all the objects in their CRUD(Create, Read, Update and Delete) operations. The performance of this method can be see in figure 3.16 represented in green where we can add more than 1000 players with a waiting time of less than 4 seconds for the user subscribing the game.

However, this approach has the drawback that its performance very slightly decreases with the amount of parallel running dynamic games in the game server. This is due to the game dependency on the database and the increase in amount of data in the tables used by the dynamic games which decreased the speed of information retrieval and map loading.

3.7.2.3 Importing Map parts

In this game context the importation of map parts is essential. Initially we used files but as in the previous section we then evolved to a direct importation. Again some adjustments had to be made to both the importer and generator in order to correctly integrate.

Besides these modifications and since there was no previously created and fully automated way to logically connect partial game information to a game already created we also had to adapt several steps of the game engine player/map creation process. Some of the modifications needed regard: players' initial armies, addition of territory links[3], update game market information, initialize new player's view, update neighbouring players' views, associate the game user to the land created[4], etc.

3.7.3 Balance

As stated in 3.6.3 Almansur has a very rich environment which must be carefully balanced, especially in the parameterization of the generation process by the game designer. More than that the same balancing system for player specific terrain is also applied in the dynamic map player generation. Also the distribution of terrains from the "neutral category" suitable for each race is the same.

[3] Almansur uses an explicit representation for connections between territories. For example if we have a territory A which is adjacent to B, then for a unit to be able to move between these two terrains in the game information there must be an explicit link between the two of them.

[4] In Almansur a user can play several games. To be able to do this there is the concepts of user and land. A user can manage several games by being a player (with a specific chosen land) in each game, so a given game land always has to belong to a user.

The fact that players enter at different game times also creates problems to the game balance. The problem is that different time of entries inherently create imbalances, since players that are already playing had more time to develop. This problem is common to several multiplayer strategy browser games which permit players to enter at different game times. A clear example of a game which faces this problem is Travian. However, this imbalance can be mitigated and our method already uses some schemas that aim to do it.

First the player/neutral zone distribution schema where a player is surrounded by three player start places and three neutral places is a way of diminishing the amount of direct conflicts a player has to manage and player tests have proven it to be successful . If a player had six other player zones as immediate neighbours, there would be too many potential conflict points for the player to manage, and the diplomatic aspect of the game would predominate. This would destroy the balance between military and diplomatic strategy in the game. This scheme provides a good balance between direct and indirect points of conflict which promotes all the components of strategy from Almansur.

Secondly the spiral expanding schema with player start places expiration ensures that a new player never enters near a much older and much more developed player. The spiral evolution makes the map evolve in a way that new players will always be placed in the oldest unused and unexpired player start place. This makes the difference in time between two players in the map globally the smallest possible. Here the expiration of start places eliminates those that have passed the acceptable number of turns a player can have as a head start from another player that enters after the first one. This parameter is obviously dependent on game tests, and is currently being tested.

A less important detail, but still relevant for game balance, from the spiral schema is that it starts centered on a neutral zone rather than on a player. This is a measure to mitigate initial imbalances. If it were centered on a single player, it meant that the first player would be surrounded by three players before any of its neighbours even had two neighbours. This way when a player has all its three neighbours it means that at least some of his neighbours have two neighbours themselves (assuming a normal map evolution without expirations).

4 Results

4.1 Introduction

In this chapter we will analyze the the methods created by evaluating their results. From the two main contexts explored in the solution chapter we will focus our result analysis more on the dynamic context since it combines all the innovative aspects of our work. Nonetheless, we will also briefly address the classic context.

During the evaluation we will be measuring the interest the maps create on players and the effects that different map growth parameters and a dynamic map generation process have on a game. Some of the effects we are measuring are the territories evolution of players, player aggregation, player survival and victory points obtained. Regarding the map interest we expect players to experience interesting challenges in a balanced game and that they do not recognize the basic patterns or any unnatural shapes due to our map creation method. For the dynamic process we want to verify that it does not affect the gameplay for players, and that they can develop and attain the same objectives as if they all started the game at the same time. Finally we present the feedback from the game designer's perspective.

4.2 Classic Game Context

As previously said this game context was the base work for the dynamic context. The generation process behind it enabled the creation of interesting game maps and permitted to test some initial ideas of map growth later needed in the dynamic context.

The practical results of this context are the possibility of creating interesting game maps procedurally based on pre-determined players information and game designer parameterization. This method cannot provide the generation of very specific historical situation, but can create themed maps. The balance between parameterization and procedural generation enables to

easily create maps such as "The Orc War". This map as well as another example of a map generated for the classic game context can be found in appendix D.

4.2.1 Player Feedback

When evaluating a game and the algorithms used in it player tests and especially their feedback are very important (as stated in section 2.1.3.6). In our case we created a player survey with several questions regarding different game aspects some oriented at our method's evaluation and others to game design improvement. Our focus will be on the feedback players gave regarding our goals. Due to our focus on the dynamic game context we had few answers in our survey for this context. The results we will analyze below are based on a survey with a total of 9 respondents. The survey was presented to players after the their game finished and was not mandatory.

4.2.1.1 Map interest

As an integral part of this parameter is the challenge created by the neighbour players. In the most recent survey when we asked the players to rate this challenge (we offered a scale of 5 values) most of the players (60%) considered the challenge neither interesting (20%) or uninteresting (20%). There are several causes for this, reported by the players:

- "We were allied."
- "Situation was strategically interesting but close neighbors were poor players and were quickly destroyed."
- "Because I had two neighbors who were hard to conquer (a dwarf and an orc) so could not expand much without diplomacy and/or early aggressive action."

From these causes we observe that most the uninterest or low interest was caused by factors unrelated to the specific player placement. Nonetheless, the compatibility of different races as neighbours is a fact that might be important to investigate.

Regarding the player disposition 89% of the respondent players claimed to have noticed it. However, in the most recent survey the answers regarding how the player distribution was

made was diversified among the several options. The actual way players are distributed was never chosen which means that the mechanisms of capital random placement inside the player specific terrain and players starting only with capitals works well in hiding the actual map growth pattern.

Further more on the subject of map interest are the presence of unnatural shapes on the map. From the data collected 89% of the players did not find any unnatural shapes. The remainder 11% corresponds to a player that when asked about the unnatural shapes actually referred the fairness of the terrain disposition rather than specifying unnatural shapes.

4.2.1.2 Balance

In game balance the number of players which participate and test the games is a fundamental factor. Nonetheless, we can observe some preliminary results of the player feedback regarding it. From the player's feedback 2/3 reported that they always felt they could change the course of events in the game. From the remainder 1/3 all claimed it was player inexperience except one respondent which said it was both territories richness and political situation. The game seems to be balanced based on these answers but due to the lack of more data we cannot support a strong conclusion though.

4.3 Dynamic Game Context

The analysis of the dynamic game context regarding several parameters like the aggressivity or player survival rate are crucial to understand the efficacy of the method developed. The tests done in this context have several influencing factors which we must take into account when observing the collected data:

Player's experience The players which participated in the games analyzed were mostly experienced players, even though not all of them. Since we needed many more players than it was common in Almansur and it is still in a public beta development phase we asked both beta testers (experienced players) and other less experienced players to participate.

Time to complete a game Since Almansur is a multiplayer online turn-based game, the turn time (realtime between turn processing) for each game cannot be very short. Otherwise the players would not be able to play every turn because many of the persons are occupied most of their day which does not enable them to play very often. As such the turn time used was one day, meaning that each player had one day to plan their "move(s)". Considering that each game usually lasts for around forty turns it means that each game lasted for more than a month and so we did not have time to collect as much data as we desired.

For the analysis done we collected data from two games with different growth pattern configurations so we could also observe its influence in game experience. The configurations of these games were the following:

Dynamic Game II This was the first dynamic game to be (actually) played. Its growth was configured with a PSS=2 and NTS=1. The game was played by 22 players and finished after 46 turns. Due to some difficulties while collecting some of the game statistics, in some cases, we were not able to collect data in the first two turns. Nonetheless, it does not invalidate this game for analysis as we will see ahead.

Dynamic Small-Cap I This was the third dynamic game to be played. Its growth was configured with a PSS=2 and NTS=0, since we wanted to test the effects of growth parameters variation. The game is still active and a total of 32 players subscribed to it, for this analysis we collected data until the 23rd turn. For this game we will only be able to partially evaluate since it has not finished yet.

4.3.1 Territories Evolution

The territories evolution is important to measure the impact, if any, on players' expansion and development, based on their different subscription turns. The expansion and economic development plays an important role in many strategy games, including Almansur. Since we wish to provide equal challenges to all players we intend the impact of turn subscription in territories evolution to be the minimum possible.

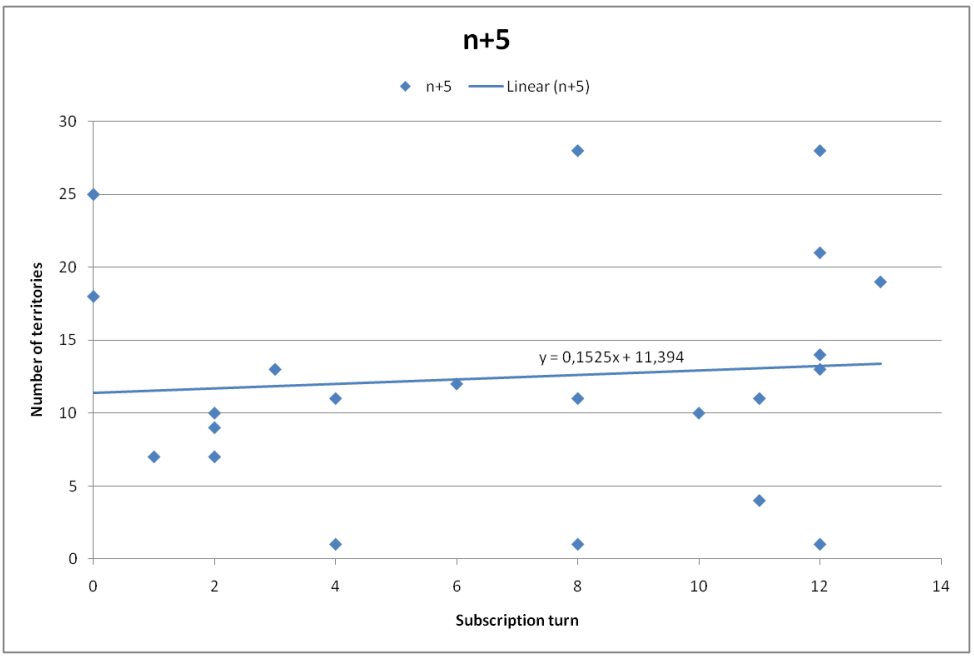

Figure 4.1: Dynamic Game II analysis of the territories evolution five turns after subscription turn.

Dynamic Game II In the graphs from figures 4.1, 4.2 and 4.3 we can examine the influence of the subscription turn in the number of terrains a player manages to obtain after 5, 10 and 15 (respectively) turns after he joined the game. After 5 turns we can observe that the influence of the turn is very small, since we have a linear regression with a very low slope value. After both 10 and 15 turns the influence is still very low, even though when comparing to after 5 turns the sign of the slope changed.

Dynamic Small-Cap I In this case, illustrated in the graphs from figures 4.4 and 4.5, due to the smaller amount of data collected we can only evaluate the influence of the subscription turn on the number of territories 5 and 10 turns ahead. Nonetheless we again confirm the very low relation between the subscription turn and the number of territories, both after 5 and 10 turns. Also notice we again had a change in the slope sign.

Comparison By analyzing the results of the two games we can observe that the several linear regressions are similar in terms of the slope values they present. Additionally the "order" in which a positive or negative slope was obtain in the different games is also different which further adds to the hypothesis that the subscription turn is unrelated to the number of terrains a player

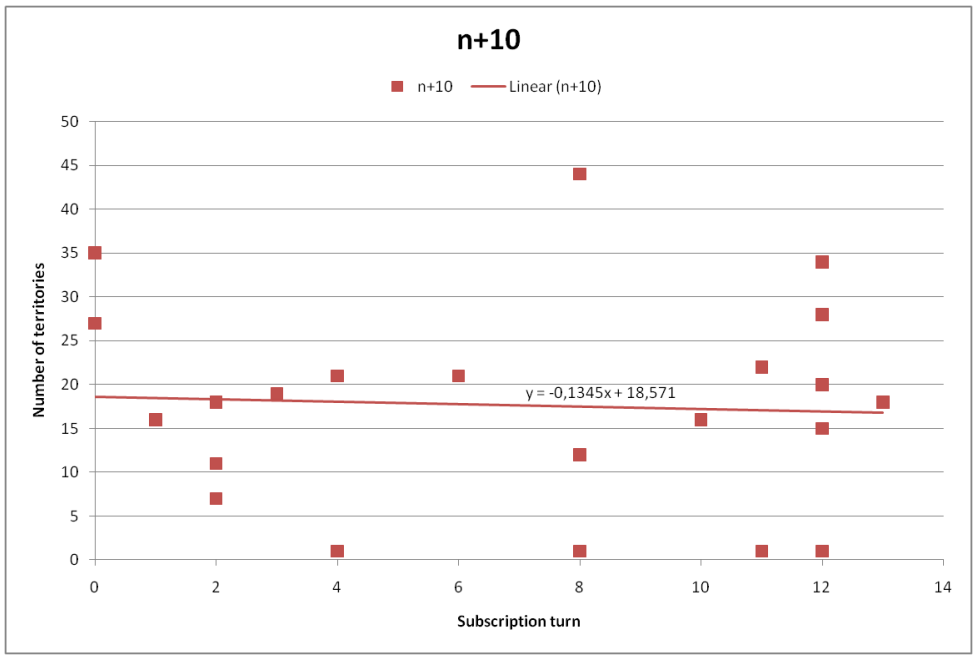

Figure 4.2: Dynamic Game II analysis of the territories evolution ten turns after subscription turn.

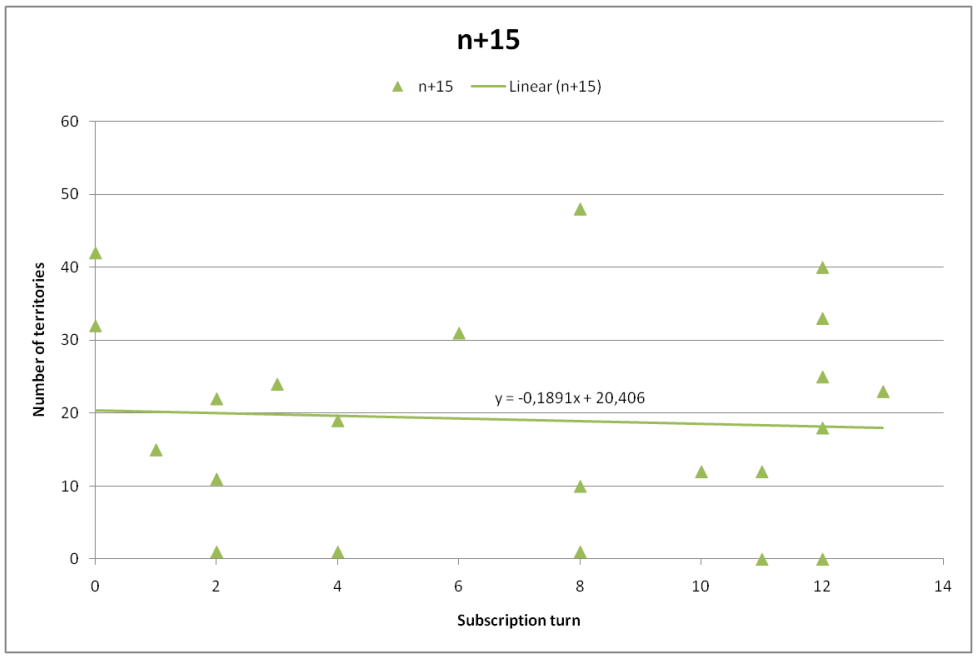

Figure 4.3: Dynamic Game II analysis of the territories evolution fifteen turns after subscription turn.

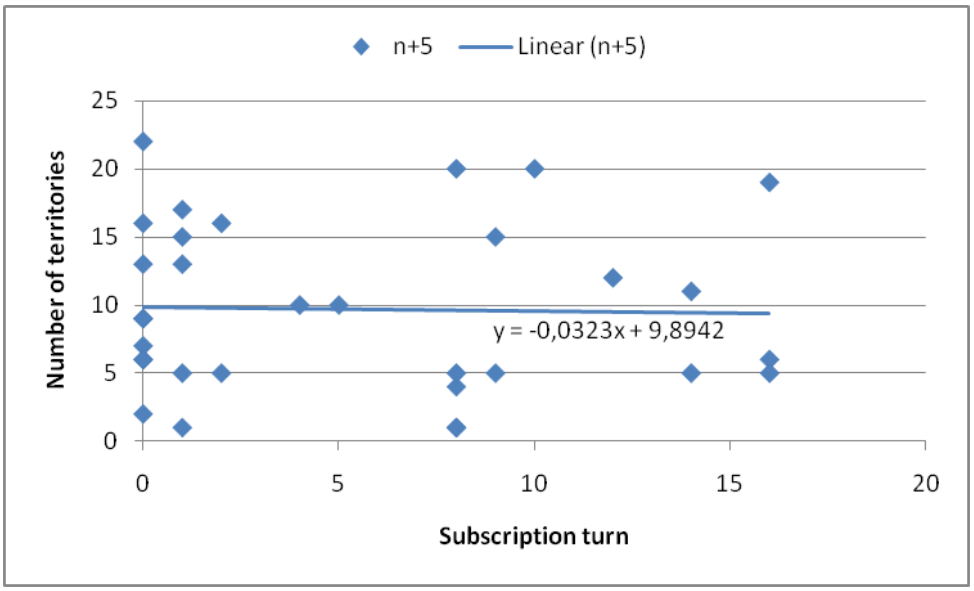

Figure 4.4: Dynamic Small-Cap I analysis of the territories evolution five turns after subscription turn.

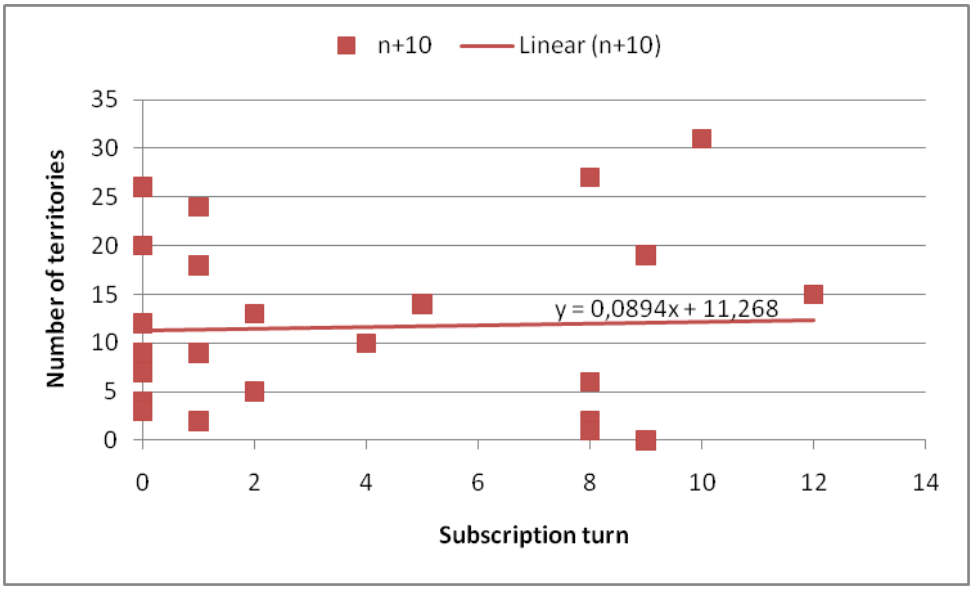

Figure 4.5: Dynamic Small-Cap I analysis of the territories evolution ten turns after subscription turn.

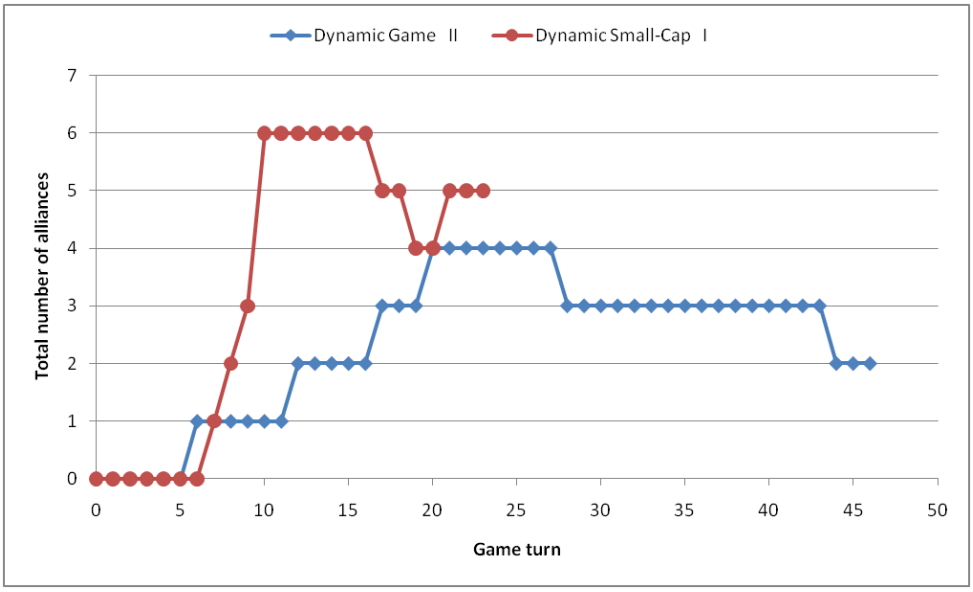

Figure 4.6: The number of alliances in each game each turn.

can own after several turns.

4.3.2 Aggregation

The analysis of the aggregation of players is important to understand the "need of joining" players experienced in the games. This enables us to observe how did the growth parameters influenced the players' need to join so that they could achieve their goals.

Alliance formation In the graph from figure 4.6 we can see a noticeable difference between the number of alliances in each game. In Dynamic Small-Cap I the number of alliances is always higher except in turns 6 (has one less) and at turn 20 (has the same amount). Even though in the Dynamic Game II the first alliance emerges first, in the second game the growth rate of alliances is much faster, going from 0 alliances to 6 in only 4 turns, while the first game takes 15 turns to go from 0 to 5 alliances. The predominance of a higher number of alliances in the second game seems to point to a higher aggregation of players, due to the hypothesized more aggressive nature of it.

Alliance size Complementary to the last graph is the average number of members in each alliance, in each turn, represented in figure 4.7. At first observation we again notice the fast

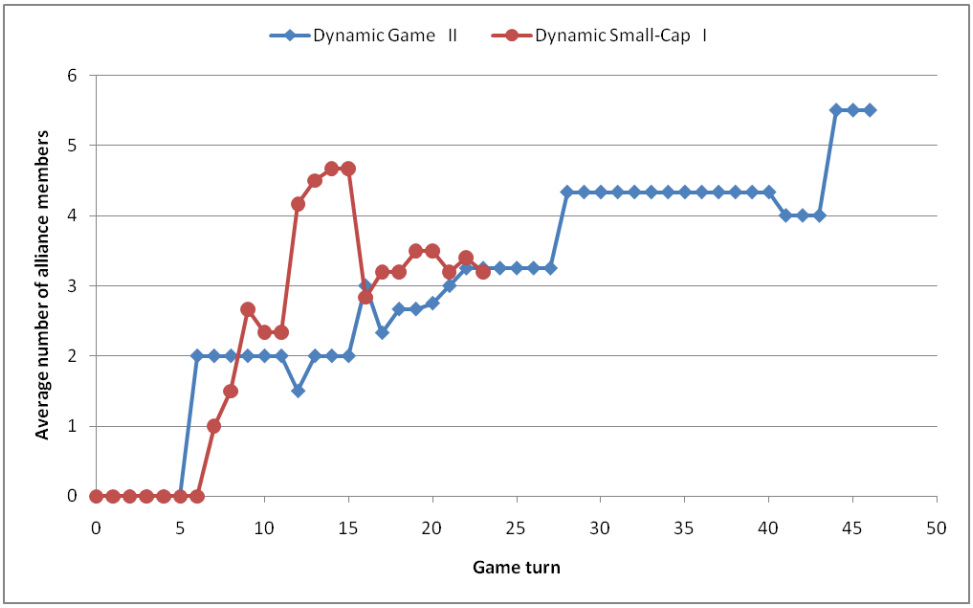

Figure 4.7: The average number of members of alliances in each game each turn.

growth of the amount of members pointing to a bigger need of aggregation from the part of the players. However, even though we cannot derive any strong elation from this graph we can observe the effect the growth parameters have in the dynamics of alliances. A game with NTS=1 has an alliance dynamics much more stable (steady growths with few and small exceptions) than another with NTS=0 (rapid growth and shrinking).

4.3.3 Player Survival

Regarding player survival we observe the distribution of players which survived/died based on their subscription turn and evaluate the independence of the survival or death of players from the turn they subscribed. This is important to our method since the independence of these factors means our method is properly mitigating the imbalances brought by different player subscription times.

Dynamic Game II In this game we can see in graph from figure 4.8 the distribution of player survivals or deaths based on the turn the players entered. Now we want to verify if the survival/death is independent from the subscription turn, but we have very few data collected in order to do a Chi-square test, so we will use the Fisher's exact test which is adequate for these cases. Then from table 4.1 we can observe that there is no evidence, on the significance

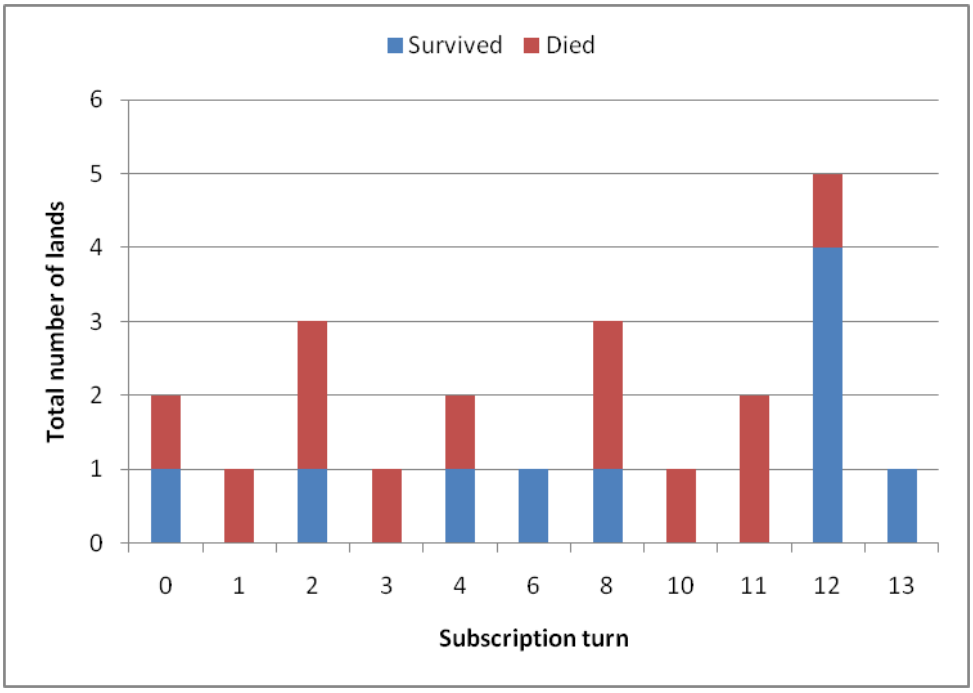

Figure 4.8: The players which survived/died in each based on the turn they subscribed in Dynamic Game II.

	0-4	5-9	10-14	Row Total
Survived	3	2	5	10
Died	6	2	3	11
Columns Total	9	4	8	21
1-Tail P-Value = 0,630				

Table 4.1: Fischer's exact test for the Dynamic Game II.

level of either 0,05 or even 0,1 which supports the rejection of the null hypothesis, meaning that there is independence of the survival/death of players and the turn they subscribe.

Dynamic Small-Cap I In this game we can see in graph from figure 4.9 the distribution of player survivals or deaths based on the turn the players entered. Again we have few data and we use a similar approach as before to test the independence of the survival/death and subscription turn. Then from table 4.2 we can again observe that there is no evidence, on the significance level of either 0,05 or even 0,1 which supports the rejection of the null hypothesis, meaning that there is independence of the survival/death of players and the turn they subscribe. Notice that in this case we have larger data grouping intervals since we have more player subscriptions.

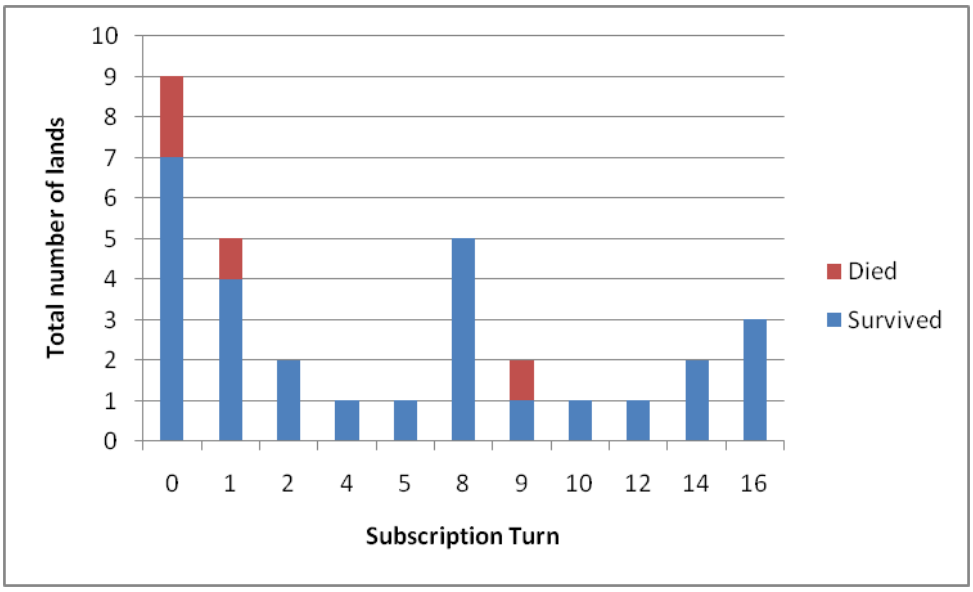

Figure 4.9: The players which survived/died in each based on the turn they subscribed in Dynamic Small-Cap I.

	0-3	4-7	8-11	12-15	16-19	Row Total
Survived	13	2	7	3	3	28
Died	3	0	1	0	4	
Columns Total	16	2	8	3	3	32
1-Tail P-Value = 0,160						

Table 4.2: Fischer's exact test for the Dynamic Small-Cap I.

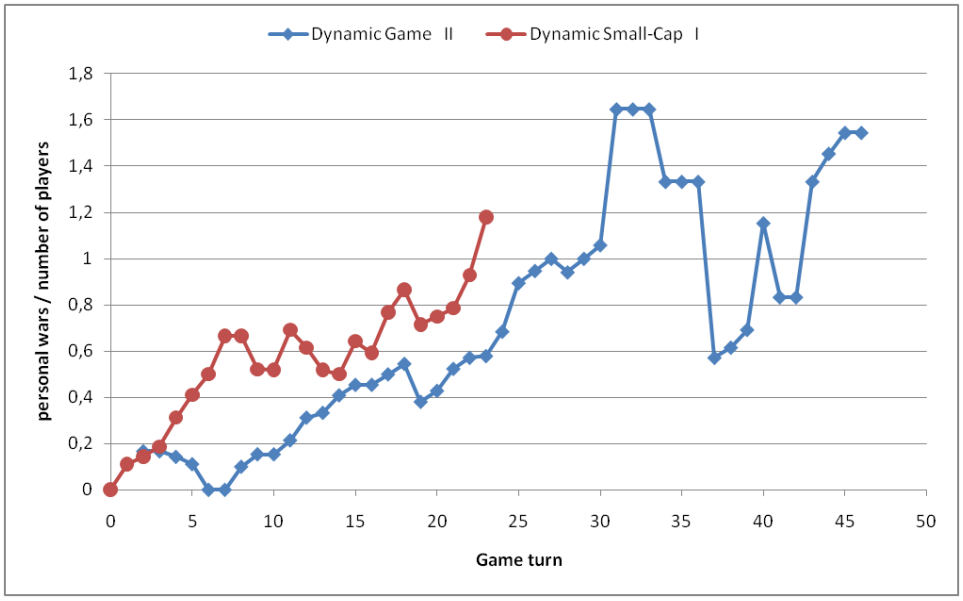

Figure 4.10: Comparison between the number of personal wars in each for the two games analyzed.

4.3.4 Aggressivity

To measure the aggressivity of the games we collected data from two games with different growth parameters where we predicted (in section 3.5) that the configuration of the second game (Dynamic Small-Cap I) would be more aggressive. In order to actually measure this we observed the evolution of two parameters: the number of personal and alliance wars active in a given turn. A personal war is held only by two players while an alliance war is held between a complete alliance (can be only one player or many) and another player or alliance.

Personal wars In this parameter the difference in aggressiveness between the two games is clear and can be observed in the graph from figure 4.10. The Dynamic Small-Cap I game is much more aggressive than the Dynamic Game II. The later always has a smaller ratio between the number o personal wars and the number of players during the period which we collected data. It is then possible to conclude that the difference in NTS (in our case 0 or 1) has a deep influence regarding the game aggressiveness.

Alliance wars Regarding the alliance wars we quickly confirm the results obtained in the personal wars analysis, which can now be observed in figure 4.11. Notice that in the Dynamic Game II the ratio between alliance wars and the number of players only grew beyond 0 nearing

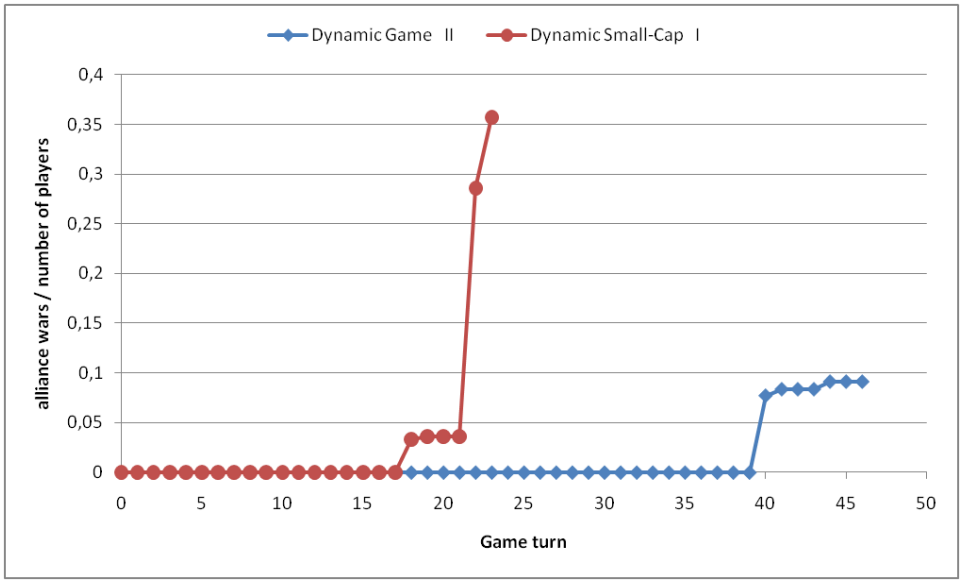

Figure 4.11: Comparison between the number of alliance wars in each for the two games analyzed.

the game ending. Contrary to this the other game, even though it is not even finished and we have around half the turns of data than the first one, demonstrates a much higher ratio of the number of alliance wars and players, rapidly growing around turn 20.

4.3.5 Victory Points

In this subsection we evaluate the distribution of the victory points in the games based on the turn players entered. Victory points are the main value used to rank players and thus have a direct influence in the final position of the players. By analyzing the graph in figure 4.12 we cannot directly relate the victory points earned by players and the turn they joined the game. Both initial and delayed players can attain good scores. However, we see a "saw" pattern in the plots of both games, even though not correlated to specific turns. After a peak of victory points we have a new lower value. A possible explanation for this is that these peaks represent zones of "strong" players and thus others around them always have less victory points. These players might be for example a local alliance which dominates a given map zone. For example if an alliance kills several players then their victory points decrease while the alliance's points increase, creating the observed high and low values.

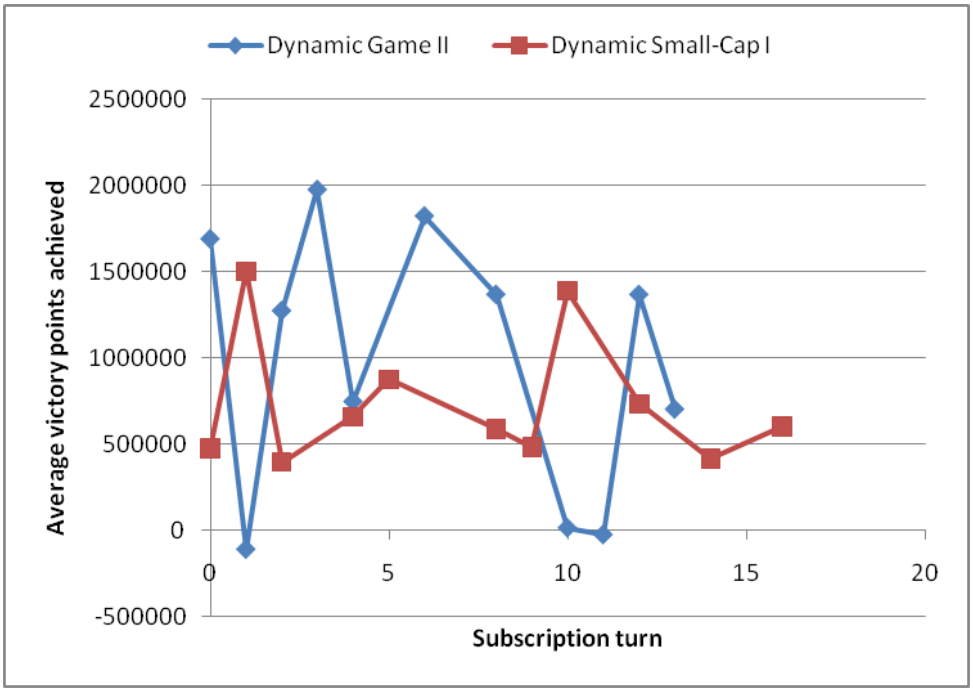

Figure 4.12: Comparison of victory points between the two analyzed games.

4.3.6 Player Feedback

In this section we will again focus on the feedback players gave regarding our goals but for the dynamic context. In this context we were able to have more respondent players, totaling 15. The methodology was the same as in the surveys from the classic context, where we presented them after the game finished. However, since the second dynamic game was still active when we did this results analysis, we presented the survey to the players from this game by considering the last turn from which we collected data as the game ending for the purposes of this analysis.

4.3.6.1 Map interest

The results observed in this parameter were more satisfactory than in the previous context. The challenges experienced in these maps were considered (we offered a scale of 5 values, from uninteresting to interesting) interesting by 53% of the players and mildly interesting by another 47% of them.

Regarding the player placement distribution only 20% of the players did not notice the way players were distributed throughout the map. On top of that now players were actually able to recognize the way players were actually distributed. Even though only 7% of the players

considered this distribution to be "unpleasant" it is interesting to notice the significant difference when comparing to the previous context and further stresses the need for more player tests.

On the subject of unnatural shapes again only 7% of the players found them. This percentage corresponds to a player which reported the following unnatural shape: "A lake really close to the sea.". The reported unnatural shape is arguable considering similar real world situations. Nonetheless, this brings the subject of terrain coherence which was considered but not specially focused.

4.3.6.2 Balance

Based on the player's feedback 73% reported that they always felt they could change the course of events in the game. From the remainder none felt helpless in the beginning of the game. The 27% of other players either felt that way since the middle or end of the game which is normal in a military game where some players suffer defeat. These results are more conclusive than the ones obtained in the previous context and further support our method regarding the maintenance of game balance.

4.3.7 Examples

Throughout this section we analyzed two dynamic games. Here we present the maps (after they closed) corresponding to these games. They are illustrated in figure 4.13.

4.4 Game Designer Feedback

Given the collaboration with Almansur and the time the solution presented takes to develop we were only able to collect the feedback from Almansur's game designer. To get his feedback we questioned him regarding the influence and impact of the solution developed in the way maps are generated in Almansur. The answer is presented in the following paragraphs and confirmed the achievement of our goals regarding the game designer's perspective.

In Almansur version 1, the maps had to be done for a pre-specified number of players. They were hand-made and there were two types:

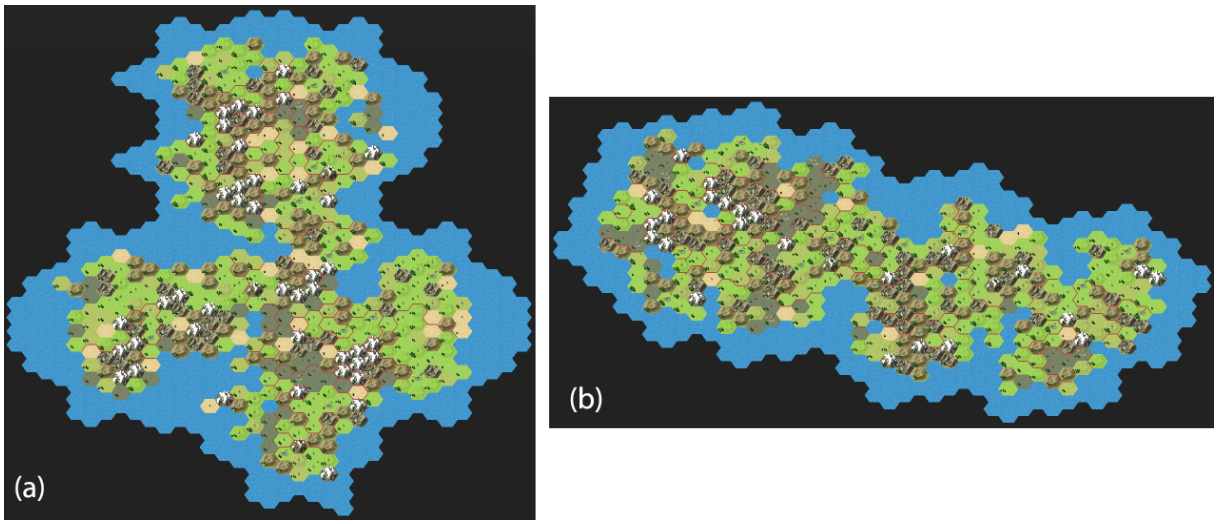

Figure 4.13: In (a) Dynamic Game II map and Dynamic Small-Cap I map in (b).

1. *Asymmetric maps, which depicted an historical situation (ex: Hispania 1135), or a fantasy one, in which the idea was to create an interesting and "credible" situation for the players. These maps were naturally unbalanced (the King of Leon-Castilla is obviously more powerful than the Count of Toledo). Here there were created physical structures like mountain ranges that created choke points, or large planes, where mounted troops had advantage. These maps were very labor-intensive, as each territory (hex) had to have around 20 attributes defined by hand;*

2. *Symmetric maps, which tried to give each player a fairly equal power, and that were done in a semi-automatic way using several huge excel files. These map system used a few terrain prototypes for each race, but:*

   (a) *the maps were repetitive, without any major terrain features;*

   (b) *the system was labor intensive, albeit less than the full hand-made maps;*

   (c) *hard to maintain, as any addition of a new terrain implied changing a lot of files and formulas, and all balancing had to be done by hand.*

In both cases, the games could only start with the map full (which meant that players had to wait sometimes more than a week to start playing the bigger maps), and once they started no more players could join.

The maps which result from applying the new system are symmetric with interesting terrain

features and are auto-balanced. Also it is very easy to add new terrain prototypes, without unbalancing the game. Using the static version option, a map for 100 players that would take before more than one day to make in the semi-automatic version, it is now made in a few seconds with better quality (balance and interesting features).

But, the best feature of the new version is the dynamic generation option. With this option a game can be created on-demand, and players can still join a game after it started, with equivalent chances of success as the players that joined earlier. This is a quantum leap from the earlier version, and improves dramatically the appeal of Almansur to the players.

5 Discussion, Future Work and Conclusions

5.1 Discussion

In this section we will discuss some of the implications and consequences of our approach in both the game designer's part of game creation and the player's perspective while playing.

With our approach the maps created are unbounded by any imposed limit on either the number of players or game design characteristics. With it the game designer has available a procedural way to create interesting maps that are scalable for the massively multiplayer browser games. He can focus only on the map balancing while creating different player experiences without intensive work. Some of the characteristics that can be modified to provide different experiences are the size of the player zones, the balance between PSS and NTS of the player zones, the PTC and terrain feature creation and control. Our method then provides a good balance between the advantages of the manually and procedurally created maps.

As a consequence of the procedural generation our method does not account for historical map creation. This is a disadvantage in a game like Almansur which occasionally intends to create these kind of situations. However, with our method and because of its flexible parameterization we can create "themed" games as briefly explained in section 4.2. Since the method permits a deep degree of parameterization it can be guided to generate specific strategic situations, like the described "The Orc War". While our method is not even directed to historical[1] game map creation this is a possibility offered by it which approximates the most to a procedural generation method to historical map creation.

The terrain prototypes definition plays an important part regarding the balance between manual work and procedural generation. The initial prototype values used by our method were

[1]This kind of game maps require great detail in their creation guided by historical background and information. With such a nature it means this kind of maps are not random inherently. In our method we intend to have a random map method to enable the creation of a great variety of different game maps.

based on previously existent terrain definitions from hand-made maps. The prototypes and the difficult process of their evaluation and categorization can lead to imbalances between the several game races since each one has different distinct characteristics. To try and aid the process of game balancing we started to develop a player specific terrain evaluation tool but did not have time to properly use and validate it. We think that this is a very important tool that can be further explored to improve the process of prototype definition work done by the game designer.

After generating several maps for the classic context we face the problem of map shape repetition, since the schema and growth pattern is always the same. A possible solution to this is the usage of the dynamic method to generate maps for a classic context also. We can easily create random maps with many possible different shapes for a fixed set of players, by employing the dynamic generation method. We could create a random number of expirations between sequential player additions, with a maximum of $turns\_to\_expire - 1$ so the map does not close before we add all the desired players. Nonetheless, this is only a possible solution for a problem which is not our focus, since it regards the classic context while we focused our efforts on the dynamic context to which the classic was a needed basis work.

The problem of delayed player entries is also addressed in a way which brings improvements in both balance and player experience. The method described has schemes that help diminish the imbalances created by the delay. However, since the problem is not completely eliminated we propose be put in place an initial protection period. Such a period with a duration of a few turns would enable new players to develop initially without the risk of being overrun by any older player, even if from just one turn older players. This would provide a safer initial resource development and give the new player time to prepare for the challenge presented by entering the game later than other players. Another solution to this problem would be to increase the strength of the players which enter later. This increased strength might refrain older players from immediately attacking recently added players or at least would enable the new player to have an opportunity in fighting the older player. However, this compensation scheme might create other problems, like older players' insatisfaction with the fact that players which enter later are "benefited". Moreover the balancing of such a system would be very hard and thorough player tests would have to be made. Some further discussion regarding this subject is presented in section 5.2.

The schemes that help mitigate imbalances, specifically created by delayed player entries in

a dynamic context, also have an effect on the way players see these maps. With our method the maps become evolving shapes which react to the flow of players' subscribing the game by growing to properly accommodate the new players. This makes the map shape unpredictable (since player subscriptions are also unpredictable) reducing the map repetition because every map shape is different according to the number of players entering at a given game time. A map where more players enter at a given game time will be different from a map where fewer players are entering at the same game time. This promotes replayability since almost every map will be different from any of the previously created which originates different player experiences and avoids boredom from repetition.

The dynamic type of multiplayer game creates a different game experience for players where they play in an ever growing interesting map. However, for our approach to have the intended effect players must understand the nature of the map they are playing so they are not surprised or frustrated. This could happen when not yet generated map terrain appears in their views. The nature of our maps is that of a growing map and so expanding zones may contain ungenerated terrain for a short time. Therefore, players must understand that in a growing self-adapting map it is natural that for a few turns they can find terrain to where they cannot move or see because they have not been created yet. This can happen in two ways. The first is when a player is placed on the map, and some of the neighbour player zones have not been generated yet since the player enters in the map's frontier of expansion. In this frontier the ungenerated player zones are the expansion points for new players. The terrain is then created in the following turns by either new players entering or by zone expirations, as a consequence players can see a terrain evolution for a few turns near them. A way to make this view updates have less impact on the player is to generate all the terrain for a player, but let him only start with his most important one, making him conquer or explore by himself the rest. The second is when a player explores the map to its borders. Note that in this case he only faces this if the map is still expanding and has not been closed yet. The evolution of this situation is then similar to first case where the difference is that it is caused by the player's fast exploration. Once players understand this nature of a growing map they can fully enjoy the strategically rich multiplayer online browser game.

Regarding the results experimentally collected from the games which used the method developed we were able to observe some relations between the map growth parameterization and

game properties and player feedback. It is important to notice we did not have as many test as we wanted (for a reinforced results analysis) due to the time it took to implement and integrate a stable version of our solution and the great amount of time each test game took to run its course from start to finish. Nonetheless, we were able to collect data which seem to confirm that we achieved our objectives of procedural and balanced map generation of interesting maps in both a classic and massively multiplayer online context. Additionally we were also able to measure more intrinsic effects of the developed method on gameplay, in parameters like the aggressiveness (measured with the amount of wars) of a game and the survival of players. Finally we stress the fact that more player tests are needed and they will be done as the game will be available to the general public.

5.2 Future Work

On top of our work there are still some areas that can be improved or explored.

As previously described, the map expands or contracts the size of the expansion zone and at a given point in time it can actually close itself not allowing more players to join. This can create an interesting player experience and even serve as a map size control method in case the game designers wish it. For example if a maximum number of players is set to the game, then the map will only expand until that number of players is reached and close itself without any direct input from the map creator (this control measure is implemented in Almansur). However, one might intend the map continues to expand indefinitely, even when the number of players joining is too low to keep zones from expiring. One way to enable this would be to create common neutral zones instead of impassable zones until new players joined. However, this approach imbalances the game, since if players do not enter for a very long period then one or several players end up having a large territory to explore and conquer only by themselves and acquire a great resource advantage over the others. How would it be possible to expand even when there are few players entering? Do we create poor terrain? Is it even really desirable? What are its consequences for the game? These are some of the questions which could be further explored in this area of dynamic map generation.

Regarding the map shape, the properties of the dynamic generation process can still be further explored to randomize it more. When the map grows rapidly we have many free player

start places where hardly will all be occupied. Since the growing schema tries to follow a spiral pattern, this commonly leads to some of the places being occupied in an orderly fashion and the others expire, creating a recognizable expansion pattern. In order to avoid this pattern and further randomize the map shape the generation process could be made aware of these situations and randomly expire some of the excessive free player start places.

On the balancing area there is also space for improvement. The player protection period would already be a good help, but could not we also have a development compensation system for players that enter later than others? Such a system could then eliminate disadvantages created by players entering at different game times and having different development levels at a given game time. This could be achieved by creating terrains with different building levels or by assigning different amounts of initial terrain based on the already existent players on the map. But then how would other players feel about this? Possibly they would feel this measure as unjust, since they had had to work to acquire that development, and new players did not. And if one eliminates the disadvantages of entering later, why not wait until later to enter? Of course this is not intended in terms of game design. One wants players to enter as soon as they see and become interested by the game. A possibly more balanced solution would be to create a compensation system that offers less "strength" to the new player than the one achieved by the neighbouring players. This could however be hard to measure and player tests would also be needed to validate it.

Regarding scalability (especially in a dynamic context) our method can also be improved. One of its main obstacles is the amount of data the generator has to persist. The more players subscribe the more data is persisted and slower the load/unload times of the generator are. During our dynamic generation process the generator keeps many information where some of it could be discarded for a better performance. For example it keeps a model of all the generated terrain where inner generated zones could be discarded for a better performance since they will not be used anymore. The decrease of the persistence times would lead to an even slower (than our current method) growth of player subscription time and increase the amount of players we could add.

Another possible future work is the study of how to present ungenerated zones to players. In our method we try to minimize the contact of players with these zones, but since it is not possible to completely eleminate it we propose it is further explored. A simple solution to it

would be the creation of specific terrain tiles for ungenerated terrain, but other solutions might exist.

Finally we could add the parameterization of several fixed values used in our generation algorithms, for example: the probability for coast randomization, the maximum distance a mountain link can have or the number of iterations on feature generation. These would increase the level designer responsibility again, but would provide further control over the generation process. An accurate and correct parameterization of several of these values can only be achieved with user based test approaches.

5.3 Conclusions

Our approach enables us to create interesting maps for turn-based strategy browser games in both a classic and massively multiplayer context. Our method can achieve these results due to its balance between the flexibility of the procedural generation method and the needed game designer input. For the massively context a dynamic map generation model was additionally needed in order to be able to adapt to the unpredictable nature of player subscriptions in online browser games and use a complex game environment in this context. The method developed also contributes to the solution of balancing problems created by delayed player entries and player unpredictability. Nonetheless, these problems are not eliminated and are subject of future work.

Regarding the game designer, he is freed from the limitations of having to create the game around the assumption of a generic or size limited map, especially the massively multiplayer context. By using our method he benefits from the advantages of procedurally generating maps (ex. can quickly create several different maps) and also from some of the hand-made maps (ex. create strategic situations with terrain features). Relatively to the hand-made maps he cannot create historical games but themed games are still possible (ex. “The Orc War”) with a correct parameterization. However, this comes with an increased responsibility regarding the parameterization of the generation process since the intended player experience relies heavily on a good parameterization to ensure game balance. This means that all the parameters, especially the terrain prototypes, must be carefully adjusted for the game before hand, also meaning intensive game testing for a well balanced game.

From the player’s perspective the solution developed for the classic game context does not

bring significant changes. However, on the dynamic context the players can now experience a complex game environments and with interesting maps. Additionally, the risk of players tiring from repetitive play is reduced since map generation depends on the player entry pattern and choices resulting in very different game maps for each game.

Our case study was Almansur Battlegrounds and it provided a real game case for us to apply the methods developed. After implementing the solution created in the game we were finally able to test our approach. From the results obtained we can conclude that we achieved our goals of balanced generation of interesting maps. Beyond this we were able to scale our map generation to the massively multiplayer context while maintaining balance and interest. For the classic game context is now provided with a procedural method that can even be further automated in order to automatically generate games as needed. For the dynamic context Almansur can now host multiplayer games where players can play against many others, do not have to wait for them in order to start playing and can always freely choose their race.

Using the techniques described in our work the TSMBGs can evolve with their complex game environments while providing players with the flexibility of player traits choice and still offer interesting and strategically challenging games.

Bibliography

Ambler, S. (2003, October). *Agile Database Techniques*. John Wiley & Sons.

Bourg, D. M. & G. Seemann (2004). *AI for Game Developers*, Chapter A* Pathfinding, pp. 126–148. O'Reilly Media, Inc.

Brosz, J., F. F. Samavati, & M. C. Sousa (2006). Terrain synthesis by-example. In *Proceedings of the first International Conference on Computer Graphics Theory and Applications*.

Buro, M. & T. M. Furtak (2004). Rts games and real-time ai research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, pp. 51–58.

Chiang, M.-Y., S.-C. Tu, J.-Y. Huang, Wen-KaiTai, C.-D. Liu, & C.-C. Chang (2005). Terrain synthesis: An interactive approach. In *International Workshop on Advanced Image Tech.*

Falstein, N. (2005, October). *Game Developer Magazine - October 2005*, Chapter Negative Feedback, pp. 41. United Business Media LLC.

Feil, J. H. & M. Scattergood (2005a). *Beginning Game Level Design*, Chapter Building Terrain, pp. 39–60. Boston, MA, United States: Course Technology Press.

Feil, J. H. & M. Scattergood (2005b). *Beginning Game Level Design*, Chapter Placing Encounters, pp. 95–118. Boston, MA, United States: Course Technology Press.

Feil, J. H. & M. Scattergood (2005c). *Beginning Game Level Design*, Chapter Specific Genres, pp. 136–149. Boston, MA, United States: Course Technology Press.

Forbus, K. (2005). Terrain analysis in strategy games. Technical report, Northwestern University. Lecture in Artificial Intelligence for Interactive Entertainment.

Gamma, E., R. Helm, R. Johnson, & J. M. Vlissides (1994, October). *Design Patterns: Elements of Reusable Object-Oriented Software*, Chapter Behavioral Patterns, pp. 331–334. Addison-Wesley Professional.

Greuter, S., J. Parker, N. Stewart, & G. Leach (2003). Real-time procedural generation of 'pseudo infinite' cities. In *GRAPHITE '03: Proceedings of the 1st international conference*

on Computer graphics and interactive techniques in Australasia and South East Asia, New York, NY, USA, pp. 87–ff. ACM.

Higgins, D. (2001). *Game Programming Gems 2*, Chapter Terrain Analysis in an RTS - The Hidden Giant, pp. 268–284. Rockland, MA, USA: Charles River Media, Inc.

Laeuchli, J. (2001). *Game Programming Gems 2*, Chapter Programming Fractals, pp. 239–246. Rockland, MA, USA: Charles River Media, Inc.

Lecky-Thompson, G. (2000). *Game Programming Gems*, Chapter Real-Time Realistic Terrain Generation, pp. 484–490. Rockland, MA, USA: Charles River Media, Inc.

Leigh, R., J. Schonfeld, & S. J. Louis (2008). Using coevolution to understand and validate game balance in continuous games. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, New York, NY, USA, pp. 1563–1570. ACM.

Millington, I. (2006). *Artificial Intelligence for Games (The Morgan Kaufmann Series in Interactive 3D Technology)*, Chapter Tactical and Strategic AI. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Musgrave, F. K., C. E. Kolb, & R. S. Mace (1989). The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, pp. 41–50. ACM.

Olsen, J. (2004, October). Realtime procedural terrain generation. Technical report, University of Southen Denmark. Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games.

Olsen, J. M. (2001). *Game Programming Gems 2*, Chapter Attractors and Repulsors, pp. 355–364. Rockland, MA, USA: Charles River Media, Inc.

Ong, T. J., R. Saunders, J. Keyser, & J. J. Leggett (2005). Terrain generation using genetic algorithms. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, New York, NY, USA, pp. 1463–1470. ACM.

Perlin, K. (1985). An image synthesizer. *SIGGRAPH Comput. Graph. 19*(3), 287–296.

Perlin, K. (2000). Making noise. In *Game Developers Conference 1999*.

Perlin, K. (2002). Improving noise. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, pp. 681–682.

ACM.

Pottinger, D. (2000). Terrain analysis in realtime strategy games. In *Game Developers Conference 2000*.

Prachyabrued, M., T. E. Roden, & R. G. Benton (2007). Procedural generation of stylized 2d maps. In *ACE '07: Proceedings of the international conference on Advances in computer entertainment technology*, New York, NY, USA, pp. 147–150. ACM.

Prusinkiewicz, P. & M. Hammel (1993, May). A fractal model of mountains and rivers. In *Graphics Interface '93*, pp. 174–180.

Quinta, M. (2008). Private communication.

Saunders, R. L. (2006, December). Terrainosaurus: realistic terrain synthesis using genetic algorithms. Master's thesis, Texas A&M University.

Schneider, J., T. Boldte, & R. Westermann (2006). Real-time editing, synthesis, and rendering of infinite landscapes on gpus. In *Conference on Vision, Modeling, and Visualization 2006*, pp. 153–160.

Shankel, J. (2000a). *Game Programming Gems*, Chapter Fractal Terrain Generation - Fault Formation, pp. 499–502. Rockland, MA, USA: Charles River Media, Inc.

Shankel, J. (2000b). *Game Programming Gems*, Chapter Fractal Terrain Generation - Midpoint Displacement, pp. 503–507. Rockland, MA, USA: Charles River Media, Inc.

Shankel, J. (2000c). *Game Programming Gems*, Chapter Fractal Terrain Generation - Particle Deposition, pp. 508–511. Rockland, MA, USA: Charles River Media, Inc.

Shirriff, K. (1993). Generating fractals from voronoi diagrams. *Computers and Graphics 17*, 165–167.

Smedstad, G. (2000). The heroes 3 random map generator. In *Game Developers Conference 2000*.

Stachniak, S. & W. Stuerzlinger (2005, May). An algorithm for automated fractal terrain deformation. In *Proceedings of the 7th international conference on Computer Graphics and Artificial Intelligence*, pp. 64–76.

Tatarinov, A. (2008). Perlin noise in real-time computer graphics. In *GraphiCon '2008*, pp. 177–183.

Tozour, P. (2001a). *Game Programming Gems 2*, Chapter Strategic Assessment Techniques, pp. 287–297. Rockland, MA, USA: Charles River Media, Inc.

Tozour, P. (2001b). *Game Programming Gems 2*, Chapter Influence Mapping, pp. 287–297. Rockland, MA, USA: Charles River Media, Inc.

Woodcock, S. & W. Wyrks (2002). *AI Game Programming Wisdom*, Chapter Recognizing Strategic Dispositions: Engaging the Enemy, pp. 221–232. Cengage Learning.

Almansur Battlegrounds Terrain Information

The terrain of Almansur Battlegrounds is composed of hexagons. Depending on the map of a specific game round, each terrain hexagon represents an area of terrain with a radius of 20 kilometers to 100 kilometers. Even though most of the terrain characteristics are self-explanatory here is a complete list of them and their description:

Chracteristic	Description
X	X coordinate of the terrain.
Y	Y coordinate of the terrain.
Terrain Type	Terrains can either be land or water.
Altitude	The altitude of the terrain.
Relief	The roughness of the terrain.
Swampness	The abundance of swamps.
Fertility	How good population reproduces in this terrain.
Arborization	The abundance of vegetation.
Gold	The richness of the terrain in terms of gold.
Iron	The richness of the terrain in terms of iron.
Stone	The richness of the terrain in terms of stone.
Fish	The richness of the terrain in terms of fish.
Wild Horses	The abundance of the terrain in terms of wild horses.
Wild Wargs	The abundance of the terrain in terms of wild wargs.
Wild Game	The abundance of the terrain in terms of wild game.
Human Population	The amount of Human population inhabiting the terrain.
Orc Population	The amount of Orc population inhabiting the terrain.
Uruk Population	The amount of Uruk population inhabiting the terrain.
Dwarf Population	The amount of Dwarf population inhabiting the terrain.
Elf Population	The amount of Elf population inhabiting the terrain.
Barbarian Population	The amount of Barbarian population inhabiting the terrain.
City Level	Represents the level of development in a territory.
Orc Encampment	Level Represents the level of development in a territory (for an Orc player only).
Underground City	Level Represents the level of development in a territory (for a Dwarf player only).
Farm Level	Level of the farming facilities.
Wall Level	Level of the main walls protecting the territory.
Lumber Mill Level	Level of the lumber gathering facilities.
Gold Mine Level	Level of the gold gathering facilities.
Iron Mine Level	Level of the iron gathering facilities.
Fortified Village Level	Represent how well the population (not army related) resists attackers.
Market Level	Level of the trading facilities.
Territory Horses	Amount of horses belonging to the population inhabiting the territory.
Territory Wargs	Amount of wargs belonging to the population inhabiting the territory.
Territory Cereal	Amount of cereal belonging to the population inhabiting the territory.
Salted Fish	Amount of salted fish belonging to the population inhabiting the territory.
Salted Meat	Amount of salted meat belonging to the population inhabiting the territory.
Terrain Owner	The player owner of the terrain.

Table A.1: Almansur Battlegrounds terrain characteristics

B Classic Context Configuration Files

In this appendix we present screenshots of the configuration files previously referenced in the solution chapter.

	A	B	C	D	E	F	G	H	I	J	K
1	name	title	race	type	description						
2	Julius Caesar	Augustus	Human	Imperial	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
3	Galadriel	Ernil	Elf	Elvish	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
4	Ugly	Warchief	Orc	Orc	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
5	Midget	Kuldar	Dwarf	Dwarvish	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
6	Evil	Kriichsman	Barbarian	Barbarian	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
7	Knight of Light	Duke	Human	Feudal	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
8	Bonaparte	Augustus	Human	Imperial	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
9	Nature	Ernil	Elf	Elvish	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
10	One Eye	Warchief	Orc	Orc	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
11	Gimli	Kuldar	Dwarf	Dwarvish	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
12	Destroyer	Kriichsman	Barbarian	Barbarian	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
13	Unknown	Duke	Human	Feudal	This is the time for great Leaders to rise and create their name in History. Who shall prevail?						
14											

Figure B.1: The players's configuration file: **players.csv**.

1	race	type	quality	movclass	altitude	relief	swampness	fertility	trees	gold	iron	stone	fish	horses	wargs	population	human_pa	orc_pa	uruk_hai_pa	dwarf_pa	elf_pa	barbarian_pa	city	encampment	unc
17	Feature_mountain	Feature_mountain	5	L	1500	0,59	0,18	0,25	0,35	0	0	0,45	0	0	0	2800	0,34	0,04	0,08	0,17	0,03	0,34	0	0	
18	Feature_mountain	Feature_mountain	5	L	1000	0,5	0,2	0,3	0,4	0	0	0,4	0	0	1000	2950	0,34	0,04	0,08	0,17	0,03	0,34	0	0	
19																									
20	Feature_swamp	Feature_swamp	7	L	100	0,06	0,85	0,5	0,25	0	0	0	0	0	0	7000	0	0,79	0,07	0	0	0,14	0	0	
21	Feature_swamp	Feature_swamp	6	L	150	0,07	0,75	0,45	0,27	0	0	0	0	0	0	6000	0	0,78	0,06	0	0	0,16	0	0	
22	Feature_swamp	Feature_swamp	5	L	200	0,08	0,7	0,4	0,28	0	0	0	0	0	0	4000	0,02	0,75	0,05	0	0	0,18	0	0	
23	Feature_swamp	Feature_swamp	4	L	250	0,1	0,65	0,35	0,3	0	0	0,1	0	0	0	3000	0,05	0,7	0,05	0	0	0,2	0	0	
24																									
25	Feature_desert	Feature_desert	7	L	100	0,01	0	0,01	0,01	0	0	0	0	0	0	100	0,45	0	0	0	0	0,55	0	0	
26	Feature_desert	Feature_desert	6	L	100	0,02	0	0,05	0,02	0	0	0	0	0	0	300	0,45	0	0	0	0	0,55	0	0	
27	Feature_desert	Feature_desert	5	L	100	0,03	0	0,09	0,03	0	0	0	0	0	0	700	0,35	0	0	0	0,1	0,55	0	0	
28	Feature_desert	Feature_desert	4	L	100	0,05	0	0,1	0,05	0	0	0	0	0	0	1000	0,35	0	0	0	0,15	0,5	0	0	
29																									
30	Feature_forest	Feature_forest	7	L	200	0,2	0,15	0,5	0,75	0	0	0	0	0	0	4000	0,15	0	0	0	0,8	0,05	0	0	
31	Feature_forest	Feature_forest	6	L	350	0,25	0,2	0,4	0,65	0	0	0	0	0	0	3000	0,2	0	0	0	0,7	0,1	0	0	
32	Feature_forest	Feature_forest	5	L	500	0,3	0,2	0,4	0,6	0	0	0	0	0	0	2500	0,25	0	0	0	0,6	0,15	0	0	
33	Feature_forest	Feature_forest	4	L	800	0,5	0,25	0,35	0,55	0	0	0	0	0	0	2000	0,3	0	0	0	0,5	0,2	0	0	
34																									
35	Feature_lake	Feature_lake	7	S	-30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
36																									
37	Coast	Coast	1	S	-30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
38																									
39	Sea	Sea	1	S	-150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
40																									
41	Human	Feudal	7	L	50	0,05	0,05	0,8	0,2	0	0,2	0,2	0	2000	0	45000	0,89	0	0	0	0	0,11	2	0	
42	Human	Feudal	6	L	75	0,05	0,1	0,7	0,1	0	0	0,1	0	6000	0	37500	0,8	0	0	0	0	0,2	1	0	
43	Human	Feudal	5	L	100	0,1	0,1	0,6	0,25	0	0	0,15	0	3500	0	30000	0,83	0	0	0	0	0,17	1	0	
44	Human	Feudal	4	L	200	0,2	0,15	0,4	0,55	0	0,25	0,3	0	2500	0	15500	0,84	0	0	0	0	0,16	1	0	
45	Human	Feudal	3	L	400	0,35	0,05	0,3	0,3	0,2	0	0,45	0	1000	0	10000	0,75	0	0	0	0	0,25	0	0	
46	Human	Feudal	2	L	750	0,5	0,02	0,15	0,25	0	0,6	0,6	0	0	500	4500	0,67	0	0	0	0	0,33	0	0	
47	Human	Feudal	1	L	1000	0,7	0	0,05	0,1	0,55	0	0,8	0	0	0	2500	0,6	0	0	0	0	0,4	0	0	

Figure B.2: The prototypes definition file: **races_terrains.csv**.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	race	type	preferred_terrai	preferred_terrai	limit_tolerance	altitude_limit	altitude+-	relief_limit	relief+-	swampness_limit	swampness+-	fertility_limit	fertility+-	trees_limit	trees+-	gold_limit	gold+-	iron_limit	iron+-	stone_lin
2	Human	Imperial	min_relief	min_trees	0,05	380	100	0,3	0,1	0,07	0,1	0,45	0,1	0,24	0,1	0,5	0,1	1	0,1	2
3	Human	Feudal	min_relief	min_trees	0,05	368	100	0,28	0,1	0,07	0,1	0,43	0,1	0,25	0,1	0,75	0,1	1,05	0,1	
4	Elf	Elvish	max_trees	min_relief	0,05	735	100	0,35	0,1	0,11	0,1	0,44	0,1	0,57	0,1	0,6	0,1	0,5	0,1	2
5	Dwarf	Dwarvish	max_relief	min_swampnes	0,05	3072	100	0,72	0,1	0,05	0,1	0,45	0,1	0,25	0,1	1,05	0,1	1,5	0,1	4
6	Orc	Orc	max_swampne	min_trees	0,05	368	100	0,27	0,1	0,56	0,1	0,43	0,1	0,21	0,1	0,75	0,1	0,9	0,1	2
7	Barbarian	Barbarian	avg_relief	avg_trees	0,05	2143	100	0,58	0,1	0,08	0,1	0,44	0,1	0,44	0,1	1	0,1	0,88	0,1	
8	Sea	Sea	avg_fish	avg_fish	0,05	-150	10	0	0	0	0	0	0	0	0	0	0	0	0	
9	Coast	Coast	avg_fish	avg_fish	0,05	-30	10	0	0	0	0	0	0	0	0	0	0	0	0	
10	Neutral	Neutral	avg_fish	avg_fish	0,05	0	100	0	0,1	0	0,1	0	0,1	0	0,1	0	0,1	0	0,1	
11	Feature_m	Feature_m	avg_fish	avg_fish	0,05	0	100	0	0,1	0	0,1	0	0,1	0	0,1	0	0,1	0	0,1	
12	Feature_de	Feature_de	avg_fish	avg_fish	0,05	0	100	0	0,1	0	0,1	0	0,1	0	0,1	0	0,1	0	0,1	
13	Feature_sv	Feature_sv	avg_fish	avg_fish	0,05	0	100	0	0,1	0	0,1	0	0,1	0	0,1	0	0,1	0	0,1	
14	Feature_fo	Feature_fo	avg_fish	avg_fish	0,05	0	100	0	0,1	0	0,1	0	0,1	0	0,1	0	0,1	0	0,1	
15	Feature_la	Feature_la	avg_fish	avg_fish	0,05	-30	10	0	0	0	0	0	0	0	0	0	0	0	0	

Figure B.3: The file containing the races' limits and its prototypes variations: **races_limits_and_variations.csv**.

```
<properties title="map">
    <!-- specifies the radius of the lands generated for a given player -->
    <player_size>1</player_size>

    <!-- specifies the radius added to the border of the player, generated as neutral terrain -->
    <border_size>1</border_size>

    <!-- specifies if the player only owns it's capital at game start -->
    <capital_only>yes</capital_only>

    <!-- specifies the quality of the terrain that composes the lands generated for a given player -->
    <player_terrain_specification>
        <specification quality="7" amount="1"/>
        <specification quality="6" amount="1"/>
        <specification quality="5" amount="1"/>
        <specification quality="4" amount="1"/>
        <specification quality="3" amount="1"/>
        <specification quality="2" amount="1"/>
        <specification quality="1" amount="1"/>
        <!-- modifier can be 'less', 'equal' or 'more'-->
        <rest modifier="less" quality="5"/>
    </player_terrain_specification>

    <!-- currently possible features: mountain, swamp, desert, forest, lake -->
    <feature_types enabled="yes">
        <type probability="0.2">mountain</type>
        <type probability="0.04">swamp</type>
        <type probability="0.01">desert</type>
        <type probability="0.05">forest</type>
        <type probability="0.25">lake</type>
    </feature_types>
</properties>
```

Figure B.4: The file containing map configurations like the growth parameterization and feature definition: **map_properties_static.xml**.

C Dynamic Context: Interaction Diagram and Domain Model

C.1 Interaction Diagram

The dynamic multiplayer game maps generation process is asynchronous regarding two of its generation phases: player addition and turn processing. The first step of the complete generation only happens once and initializes the game map. This process is illustrated in the figure C.1.

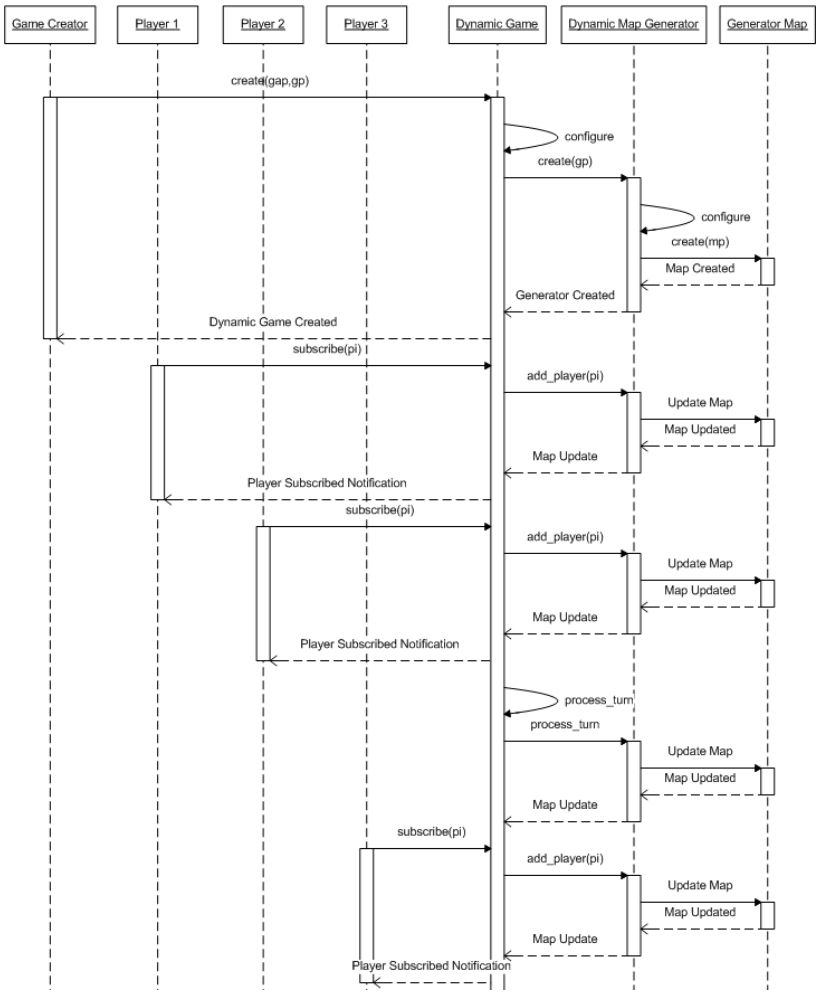

Figure C.1: The interaction process of dynamic maps. The dynamic game map is created, then two players join, a turn is processed and finally another player joins.

C.2 Dynamic Context Domain Model

In a dynamic map context data persistence is a requirement since the generation has several stages executed asynchronously and at different game times. To do this several approaches were tried but due to integration and performance needs we adopted a database persistence model based on Ruby's native ORM. Figure C.2 illustrates the domain model used to map all the classes we needed to persist in the database.

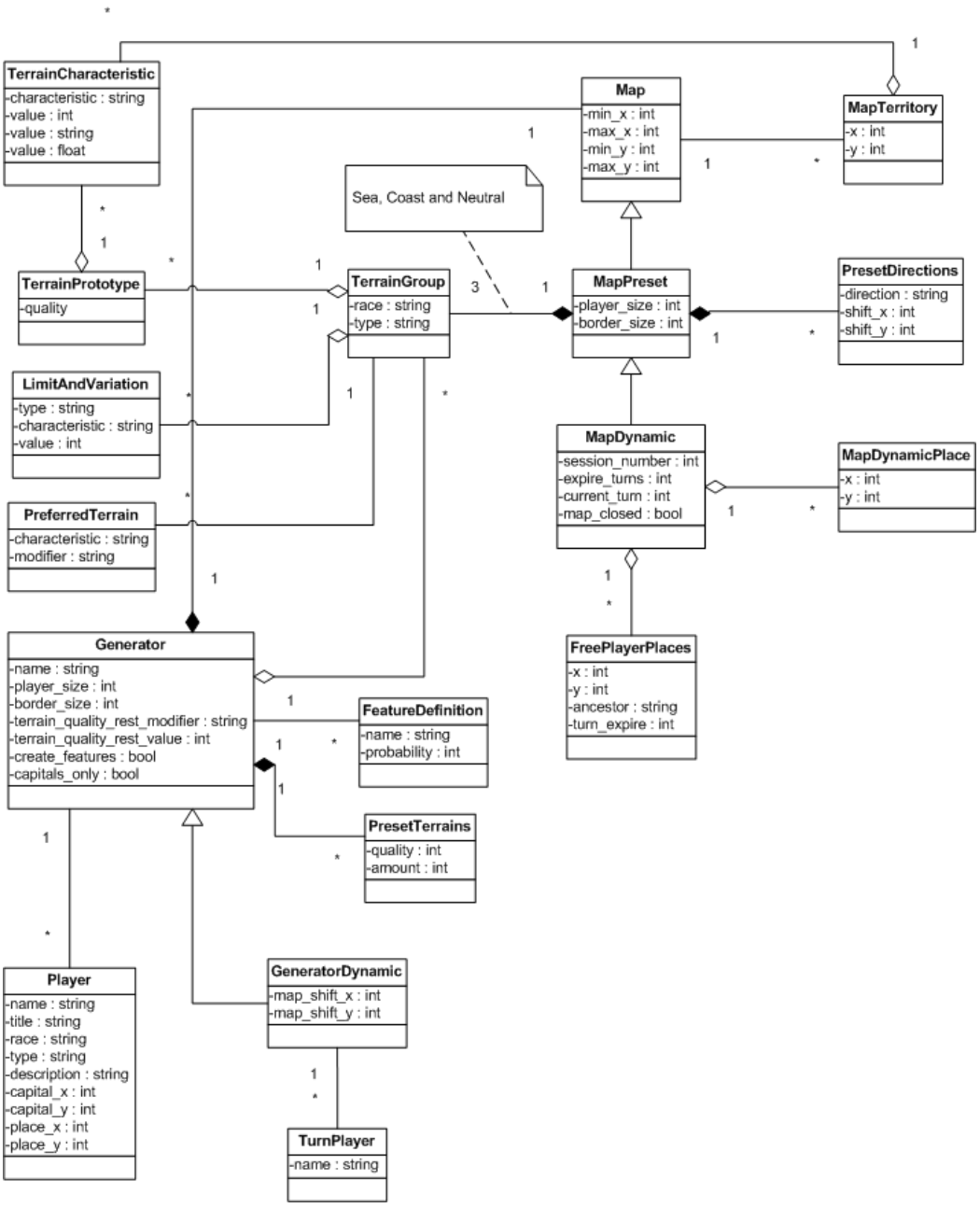

Figure C.2: The domain model for the dynamic generation process used as blueprint for ORM.

D Classic Context Maps

Many different map configurations can be used to generate different maps with our solution in the classic context. Here we present two examples of maps created with this method.

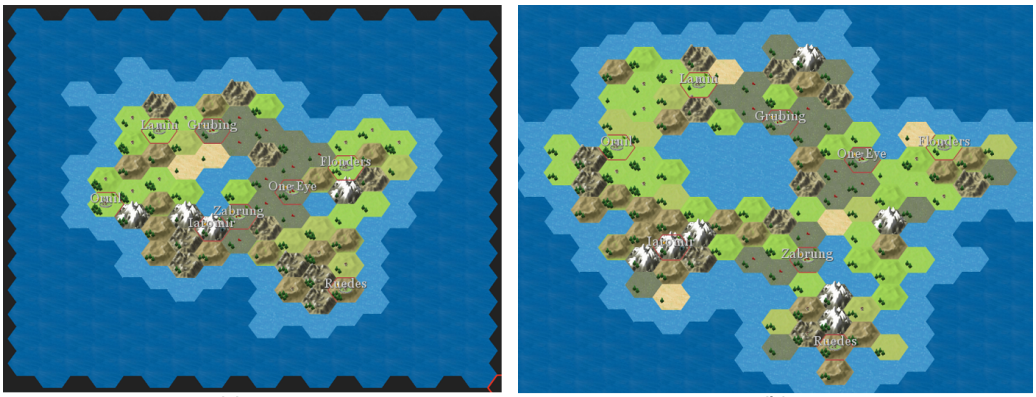

Figure D.1: Two "The Orc War" maps generated for Almansur with different growth parameters but the same player, terrain prototype inputs and PSS value. In (a) we have an NTS=0 and the players closer, and in (b) an NTS=1 and players more separated.

Figure D.2: In this example we have a common classic context map that contains two terrain features: mountains and lakes.