Balanced Civilization Map Generation based on Open Data

Gabriella A. B. Barros
Center for Computer Games Research
IT University of Copenhagen
Copenhagen, Denmark
gbar@itu.dk

Julian Togelius
Department of Computer Science and Engineering
New York University
New York, USA
julian@togelius.com

***Abstract*—This work investigates how to incorporate real-world data into game content so that the content is playable and enjoyable while not misrepresenting the data. We propose a method for generating balanced Civilization maps based on Open Data, describing how to acquire, transform and integrate information from different sources into a single content. Furthermore, we evolve players' initial positions in order to obtain balanced maps, while trying to minimize information accuracy loss. In addition, this paper describes a tool to assist users in this process. Maps generated using these method and tool are playable and balanced yet faithful to the original sources.**

***Keywords*—Data games, map generation, procedural content generation, Civilization.**

I. Introduction

Why do we need to design video game maps and levels? Why can't we just use the real world instead? Many video games feature topological content in a functional role. Players are often tasked with interacting with some sort of map: moving their characters across it, direct units around it, or building things on it. In game development, large resources go into making rather small maps; even AAA games (i.e. games with high budgets for development and marketing) renowned for their world building (such as *Skyrim* or *Grand Theft Auto*) commonly feature maps that are just a few kilometers across. Real-time strategy games and first-person shooters commonly launch with just a few good multiplayer maps.

The lack of expert-designed content drive players to generate their own content in many games. *Steam Workshop* (a community for player-generated content) features tens of thousands of player-created maps for various games. A simple search for "earth map" in the Civilization Fanatics Center forum[1] returns 300 different entries. However, player-generated content is not likely to solve this content shortage on its own, as not all games (or players) are as easy or engaging to generate content for, and that such content is not generated according to the constraints or wishes of either game developers or players. There is clearly a content shortage.

Procedural content generation (PCG) techniques have been proposed to deal with the content shortage problem, and are commonly used in some games; this includes some strategy games such as Civilization, where each game starts with a freshly generated maps [1]. However, current techniques only allow fully automatic generation of maps for some types of game designs. For many games, generated maps may come across as unbalanced or uninteresting, and so are not used.

So why don't we just use the real world for our game maps? The real world is huge, meaning there will certainly be enough content for almost any type of game, and undoubtedly interesting by virtue of being lived in and shaped by a multitude of people and their histories. Everyone has a special connection to one or several places in the world; imagine playing a strategy game in the province of your hometown or an adventure game in your workplace. While a map based on the real world *could* be as uninteresting as a poorly human-designed map, if

978-1-4799-7492-4/15/$31.00

[1]Civilization Fanatics Center : http://forums.civfanatics.com/

the same amount of polishing is performed on a map based on the real world as on one that is not based on the real world, the former have greater odds of being more interesting. Basing game maps on the real world could also decrease development time and potentially increase diversity and productivity. The increase of open data use has resulted is a large amount of information available about multiple facets of our world, and this amount is growing all the time. With (semantic) web technologies, we should be able to simply pour this data into our games. The process, for example, of transforming the map of Europe into a Civilization V or Sim City 4 map has no longer to be the arduous work of users who wish to play in it, but can be created for the user in an automatic or semi-automatic manner.

The significance of this goes beyond making games more fun. If we base our games on real data, we might also learn about the world as we play. A game could showcase particular parts or facets of the world's geography, simply tweaking data selection. Interacting with the game could mean interacting with a faithful representation of the real world.

So why are current game maps not generated from real-world maps? Because the world was not designed to be played on. At least not without a number of adjustments. Game designs put various constraints on maps and levels, and these constraints are often not met by the real world. A real-time strategy game map might need a balanced distribution of resources, a first-person shooter might need cover points and weapons caches, an adventure game will need world elements that support its storyline and so on. It is likely that someone has forgotten to equip your hometown with these features.

One way of resolving this would be to start with parts of the real world and use PCG methods to change it into a playable map or level. Move things around, and remove or add parts until the content works with the game design. This carries the downside that the in-game world is subverted so that it no longer matches the outside world. This can possibly suspend disbelief, reduce enjoyment and/or certainly negate any learning effects by essentially lying about the world.

Instead, this paper tackles the harder challenge of being true to both the game and the world: taking open data about the world and transforming it into playable game content while minimizing the introduction of inaccuracies. There are essentially two ways of doing this: by choosing which aspects of the world to model, and by procedurally generating those parts of the game which are peculiar to the game design and which do not have direct counterparts in the real world. Concretely, we will create maps for a version of Civilization that reproduce parts of the real world – any part the player chooses, at any scale – and find combinations of resources and starting positions that allow for entertaining and balancing gameplay. Our solution involves merging of data from multiple sources, and a balancing mechanism based on evolutionary computation.

II. Map Generation

Many methods for map generation have been proposed before. One way to get interesting results and fast runtime is the use of fractals: using diamong-square algorithm to iteratively divide the space, changing the midpoint slightly by a random value [2], [3]. This, however, does not allow for much control. Dungeon layouts can also be produced by placing various sized rooms and hallways on a two-dimensional area, using as fitness function the length from start to finish [4]. Togelius et al [5] use search based PCG through multiobjective evolution to create maps for StarCraft. StarCraftmap generation was also attempted by Uriarte and Ontan, using Voronoi diagrams to define the initial terrain layout and populating it using metrics [6]. Cardamone et al [7] evolved FPS maps, using as fitness function the time that a player spent alive and fighting, and the free space of a map. Mahlmann et al. [8] proposed a search-based algorithm to generate playable maps for RTS Dune 2. In [9], another search-based algorithm creates maps for *Planet Wars*.

III. Data Games

Data games are games that use real-world open data to create content that appears in-game. Therefore, these data can be explored while playing, allowing different forms of visualisation and learning to emerge from it [10], [11]. However, the data in itself can be hard to obtain and is usually not in a

form that allows it be straightforwardly incorporated into the game, thus creating the dual challenges of data acquisition and data transformation.

An early example of a data game is Open Data Monopoly [12]. Here, Friberger and Togelius created a Monopoly board generator based on real-world demographic information. Similarly, Urbanopoly [13] uses open data to generate storyboards.*Bar Chart Ball* [14] is a data game where the player moves a ball that sits on top of a bar chart by choosing different demographic indicators, which change the shape of the chart in question. Another examples is Open Trumps [11], a card generator for the simple card game Top Trumps. Sets of cards are automatically created and balanced based on countries using evolutionary algorithms. Several examples of further data game prototypes can be found in a recent overview paper [10].

IV. Civilization and FreeCiv

Civilization is an epic turn-based strategy game designed by Sid Meier released in 1991, which has been very influential and received multiple sequels. In this game, the player takes on the role of leading a civilization through 6000 years of history. While the game features military conflict and can be won through conquest, large parts of the game focus on exploring the world, founding and growing cities, planning production, balancing a budget, and conducting scientific research. A game of Civilization is to a large extent defined by the map topology and which other civilizations and resources (coal, gold, etc) are available in different places. The game design, which rewards exploration and only reveals certain resources once a particular level of technology has been reached, means that interesting conflicts and complex gameplay can arise out of resource competition. At the beginning of a new game, a complete new map is generated in order to provide novelty. The use of maps that simulate the real world make Civilization a useful testbed.

FreeCiv is an open source turn-based strategy game, inspired by the Civilization series, and comparable to Civilization I and II[2]It is written by a team of enthusiasts in C, and scriptable via Lua. We use FreeCiv in this study because it is open source.

V. Data Acquisition

Two different categories of data were necessary for this generation: terrain information and resource locations. The former is acquired just before the actual map generation, rendering the map with OpenStreetMap (OSM), a community based world-mapping project [15][3], and JMapViewer, a Java component that can incorporate an OSM map into a Java application[4]. An interactive tool was developed to assist the importing process and manage resource map images from the user's computer, and to allow for the selection areas for generating the map using JMapViewer. We could not find a single source of information for all (or most of) resources, and map images differ greatly in design so a single pattern could not be used for recognition. Thus, user input is required during the importing process.

A. Resource locations

Information about resource locations was obtained by processing images in three steps. In the first step, several images were collected using searches in the Google search engine, with sentences like "oil deposit + maps". These images were saved and fed to the developed tool, as shown in Fig. 1.

In step 2, the tool displays this image on screen, and allows selection of resource type (coal, oil, gems, gold or iron), as well as colors from the image (Fig. 1). For each color selected, it is necessary to define its type (i.e. resource, background, terrain or water). Selected shades are used to infer non-selected ones, using a color similarity algorithm. This works as follows: a new blank image is created with the same dimensions as the original one. For each color c_i in the original image, a distance is calculated between c_i and all selected colors. The shade with the smallest distance is then applied to the new image. Distance is calculated using DeltaE 1994, or ΔE 1994, a distance metric between two colors in the CIE Lab color space. CIE Lab is a color

[2]FreeCiv: https://play.freeciv.org/

[3]http://www.openstreetmap.org/

[4]http://wiki.openstreetmap.org/wiki/JMapViewer

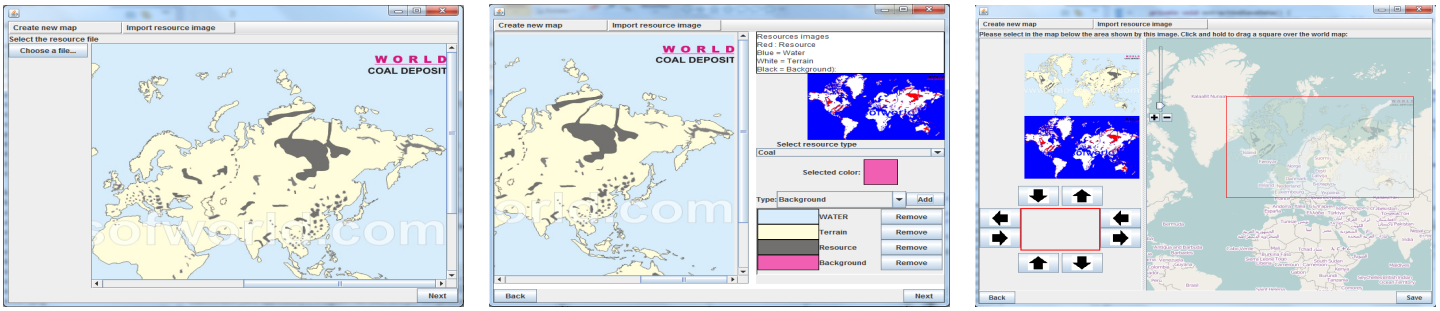

Fig. 1: **Left:** Importing resource image screen. Users can select a file from their computer in this screen. **Middle:** Another screen from the importing tool. Users can select colors from the left side of the screen. On the upper right corner, a preview of the resource version is shown. On the bottom right, users can see each color and its resource type. **Right:** Third and final resource importing feature screen. User can select a rectangle from the map and save the file.

representation based on a vertical L* axis ("Lightness"), that ranges from 0-100, and two horizontal axes a* and b* (green and red, respectively) [16]. This step allows for inferring the type represented by all colors by selecting only a few shades. Watermarks and text in images can be handle by either selecting the color as background (thus essentially excluding it) or, if the whole mark is within terrain or water, selecting the appropriate type.

In the final step the user can select a rectangle in an actual world view rendered with JMapViewer, indicating were the image would fit inside the world, as seen in the right-most image in Fig. 1.

B. *Terrain information*

Terrain information is obtained just prior to the actual generation of maps. Using the interface shown in Fig. 2, the user can zoom in and out and select a portion of the map, as well as the resolution and name of the final FreeCiv map. The selected portion is then saved as an image and processed later, during data transformation.

VI. Data Transformation

The map generation process consists of four steps: map selection and terrain transformation, resources intersection and creation, players' initial positions, and finally post-processing.

A. *Map selection and terrain transformation*

In the map selection, a world image is extracted as explained in V-B, and recorded as an image. Longitude and latitude coordinates from top left and bottom right corners are gathered for latter use. In addition, the user can select the final map's name and dimensions. This image is then rescaled to the desired dimensions. An integer matrix *mapTerrain* with the same size is also created and initialized with zero values. For each pixel in the original image, a value is attributed to the matrix's relative position, using the color of that pixel as terrain type (green pixels represent forest, blue pixels ocean or lake, etc). If the original image is greatly larger than the output, parts of it will be comprised into chunks.

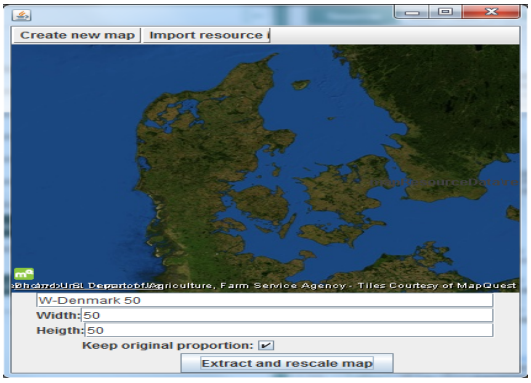

Fig. 2: World place selection for map generation.

Algorithm 1 Resource creation

```
for all Images img ∈ resourceImgs do
  for j = 0 to mapResources.height do
    for i = 0 to mapResources.width do
      rType ← resource type of img
      rChar ← character representation of rType
      if img[j][i] = WHITE and randomdouble < probRes and mapTerrain[j][i]
      is compatible with rType then
        mapResources[j][i] = rChar
      end if
    end for
  end for
end for
for all resourceType ∉ resourceImgs do
  rType ← resource type ∉ resourceImgs
  rChar ← character representation of rType
  for j = 0 to mapResources.height do
    for i = 0 to mapResources.width do
      if mapResource[j][i] is EMPTY and random double < probRes and
      mapTerrain[j][i] is compatible with rType then
        mapResources[j][i] = rChar
      end if
    end for
  end for
end for
```

Fig. 3: Pseudo-code for resource position selection.

B. Resource intersection and creation

Subsequently, these coordinates or the image corners are used to select intersecting resources' information. An array of images, resourcesImgs, represents the resources. At first, all images are initialized with black pixels and the same size as the map. Each image represents a resource type (e.g. coal or oil). For each resource imported, the intersecting rectangle is chopped off it and merged in the relative position of the image. Resources are painted white, and all others are ignored.

Afterwards, a character matrix, mapResources, is created with similar dimensions as mapTerrain, and initialized with blank spaces. The resource insertion algorithm is shown in Fig. 3. For each image in resourceImgs, and for each tile in mapResources, the resource type and its character representation are extracted. If a pixel in resourceImgs indicates that there can be a resource there, it has $\%0.35$ probability of being assigned. Resources not imported in the resource deposits' location process (V-A) are randomized in empty, terrain compatible spaces.

C. Players' initial positions and post-processing

The third step evolves initial positions on map, using a evolutionary strategy algorithm. The algorithm and fitness functions are discussed in detail in VII. Finally, in last step auxiliary choices (e.g. player civilization) are made at random. All data is saved in FreeCiv's save file format.

VII. Evolutionary Balancing

Our strategy for balancing maps focuses on base position. Given a terrain and resource map, the positions for n given players is evolved using a $12 + 88$ evolution strategy with one-point crossover and random initialization. This algorithm was chosen because of its simplicity and robustness. The individuals are represented as vectors of length $2n$, where n is equal to the amount of players. This vector contains x and y-values of each player's base.

Our implementation differs from standard evolution strategies through the use of cascading elitism [17] in selection. Each generation, selection happens as follows: Initially, fitness fDB_i is used to sort the population, and the lower rated half of the population is eliminated. The remaining population is resorted using fOR_i, and again half of it is eliminated. Again, remaining individuals are sorted and half eliminated, now according to $fBPR_i$. The remaining eighth of the population is cloned repeatedly, until it returns to its original size. Each new individual has 20% chance of being mutated. If it is, each base of that individual also has 20% of being replaced with a new position. Then, the mutated population is passed to the next iteration. The order of fitness (fDB_i, fOR_i and $fBPR_i$) was chosen after testing with all possible combinations.

Fitness calculation takes into account **exploration** and **fairness**. By exploration, we define the necessity of searching new places and how far can one player go before encountering another one. It is measured by a value fDB_i, short for fitness of distance between bases, as shown in Eq. 1.

$$fDB_i = \begin{cases} 1 - \left(\sqrt{\frac{\sum_{i=0}^{n} (mean_{dist} - dist_{ij})^2}{n}} \right), \\ \text{if } dist_{min} > threshold \\ 0, \text{ else} \end{cases}$$
$$mean_{dist} = \left(\sum \frac{dist_{ij}}{dist_{max}} \right) \div n \quad (1)$$

where n is the amount of players, $dist_{ij}$ is the distance between two bases i and j, $dist_{max}$ is the maximum distance in map (i.e. its diagonal). $mean_{dist}$ represents the average normalised distance between all bases. Thus, fDB is 1 minus the standard deviation of distances between bases. We do this inversion to maximize the value, so we can search for a maximized fitness. In truth, the lower the standard deviation, the better, since we would obtain a more equal distance among all bases.

Fairness, on the contrary, implies giving similar opportunities to every player of obtaining resources. It is measured by $fBPR_i$ and fOR_i (short for fitness of bases per resources and fitness of owned resources), shown in Eq. 2 and 3, respectively.

$$fBPR_i = 1 - \left(\sqrt{\frac{\sum_{i=0}^{n} (bigMean - mean_i)^2}{n}} \right)$$
$$bigMean = \sum_{i=0}^{n} mean_i$$
$$mean_i = \left(\sum_{j=0}^{qR} \frac{dist_{i \to j}}{dist_{max}} \right) \quad (2)$$

where n is amount of players, $dist_{i \leftarrow j}$ is distance between bases i and resource j, $dist_{max}$ is maximum distance in map, qR is total amount of resources. It represents standard deviation of average of the average distances between resources and bases.

$$fOR_i = \begin{cases} 1 - \sqrt{\frac{\sum_{i=0}^{n} \left(\left(\frac{\sum_{i=0}^{n} s_i}{qOR} \right) - s_i \right)^2}{n}} & \text{if } qOR \neq 0 \\ 0 & \text{if } qOR = 0 \end{cases} \quad (3)$$

where qOR is total quantity of owned resources in map, n is amount of players, and s_i is amount of safe resources owned by base i. A safe resource is one that is closest to i than to all other bases, and that is guarded by base i itself, in the sense that any other player who tries to get to it has to pass by i first. If there is no owned resource, it returns 0.

VIII. Results and Discussion

A. Balancing Results

Three different series of experiments, of 10 runs each, were performed using random maps, populated with water, grassland and resources. The first batch had maps of 100x100 dimension and 10 players, the second had the same dimensions, but 10 bases; and the last was 250x250 with 5 bases. Initial positions were evolved in 200 iterations using the method described in Section VII. Fig. 5 shows, for each fitness, the average between the best individuals of that iteration (in said feature) in all runs. All three features show a increase over time, but the outcome is higher when using a larger map, especially for fOR_i. $fBPR_i$ show a smaller difference between tests, probably due to being the last feature used in the cascading process.

Another experiment was done using a NSGA-II, a multi-objective optimization algorithm [18]. It attempts at minimizing the inverse of each fitness (fDB_i, fOR_i and $fBPR_i$) without them damaging each other. It used 10 experiments with maps of 100x100 dimension, 5 bases and 250 iterations, and results are shown in Fig. 4. Fitness fOR_i seems to optimize much faster than the others. Further investigation using multi-objective algorithms is planned.

B. Map creation

Some of the maps generated are shown below. Fig. 7 shows a map of Denmark, in-game. Fig. 6 show maps from Europe, Africa, and North and

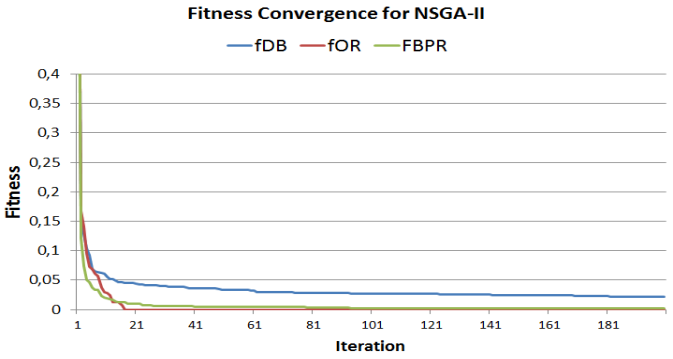

Fig. 4: Graph that shows the fitness convergence of the NSGA-II algorithm.

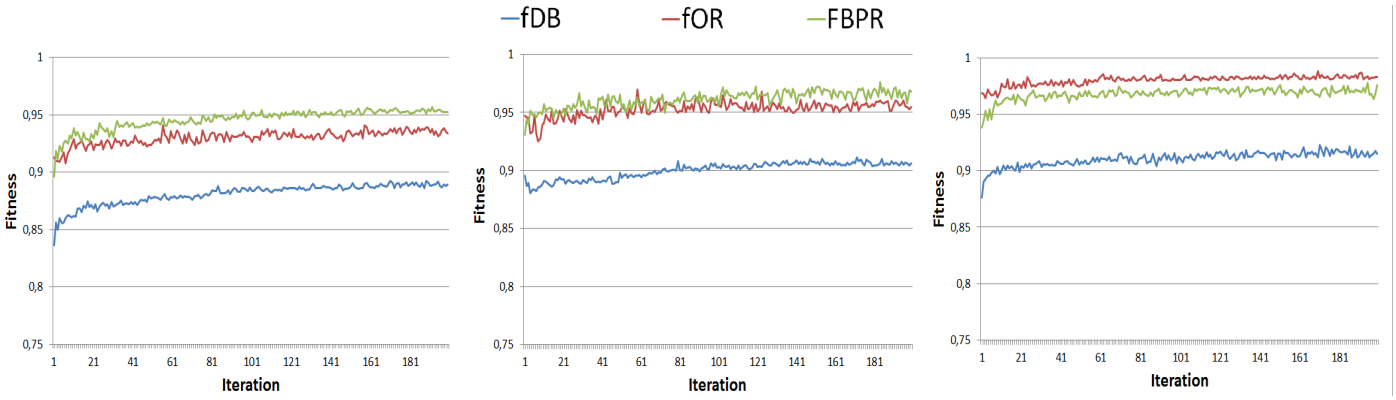

Fig. 5: Average convergence of the algorithm with different map sizes and quantity of bases. **Left:** Using 100x100 map and 10 bases. **Middle:** 100x100 map and 5 bases. **Right:** 250x250 map and 5 bases.

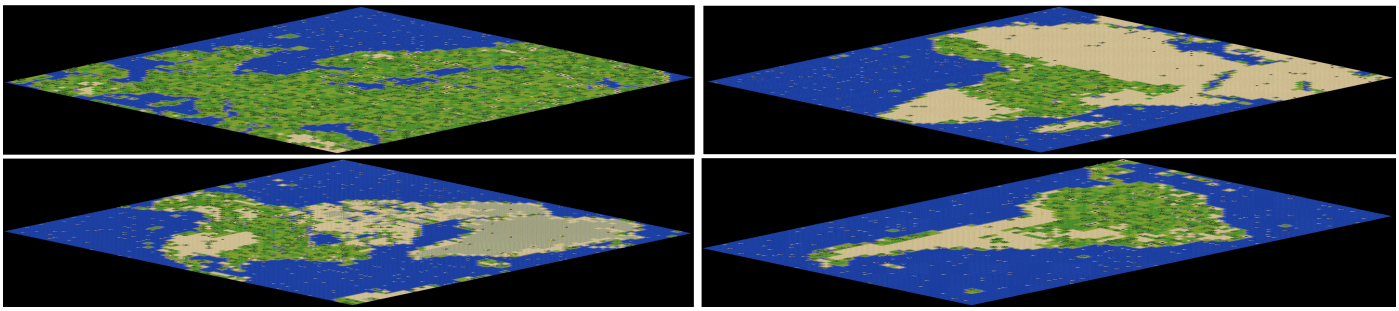

Fig. 6: **Top left:** Europe. **Top right:** Africa. **Bottom left:** North America. **Bottom Right:** South America.

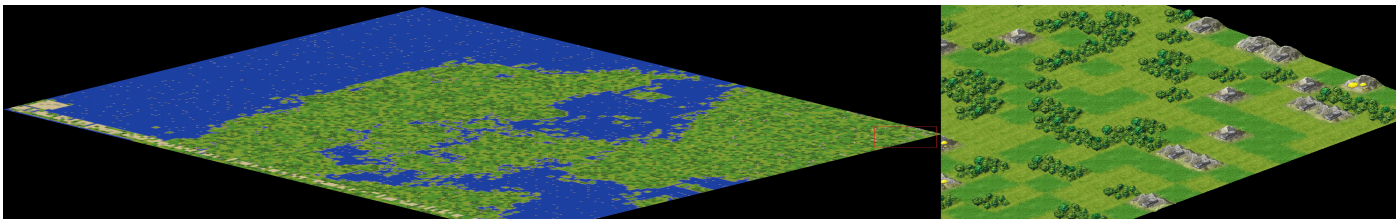

Fig. 7: Map of Denmark, in-game, on the left. On the right, the right upper are (highlighted in red) of map is zoomed in. Note that it has isometric topology, thus is ”inclined” to the right in comparison to the original map image.

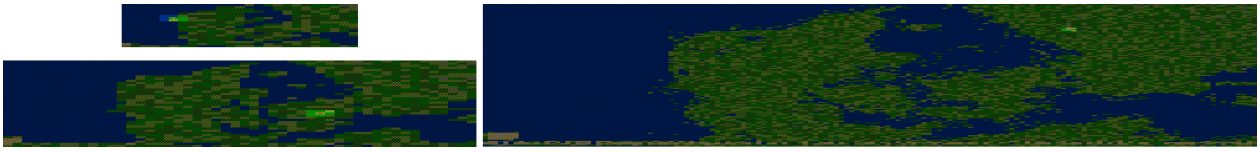

Fig. 8: Maps generated using Denmark as input, with 25x30 (top left), 50x40 (bottom left) and 150x121 (right) dimensions.

South America. As can be seen, the generated maps retain essential geographical information. Fig. 8 compares loss of quality in a level of 25x20, 50x40 and 150x141 dimension, showing a small amount of difference in comparison with the original, which is the exact same as the one shown in Fig. 2.

One problem encountered was that some resource maps gathered from the internet had the wrong proportions or required tilting and/or warping to properly fit the OSM view. This lead to some erroneous placing (e.g. placing coal mines where they do not exist in reality).

The current generator and tool is fully functional, but could be improved in several ways. For example, the current evaluation function could be more accurate by using simulations, using AI agents to play on the maps in order to evaluate them. It would also be interesting to make a complete user study of the system, including its usability and the perceived fairness, accuracy and playability of generated maps.

IX. Conclusion

This paper describes a method for creating complete, playable maps based on open data about the real world for a clone of the popular strategy game Civilization. The method creates maps that are true to the real world, by preserving the topology of the map, as well as the placement of various resources. The method is incorporated into a framework which lets the user select any part, of any size, of a world map and create a playable civilization map out of this area. This tool can serve the dual purposes of enabling more interesting content generation and providing a way of exploring the real world by playing on it. We believe that with slight modifications, this method could apply to other games that feature maps or map-like levels.

Acknowledgment

Ms. Barros thanks CAPES and Science Without Borders for financial support. Bex 1372713-3

References

[1] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2015.

[2] G. S. Miller, "The definition and rendering of terrain maps," in *ACM SIGGRAPH Computer Graphics*, vol. 20, no. 4. ACM, 1986, pp. 39–48.

[3] J. Olsen, "Realtime procedural terrain generation," 2004.

[4] N. Sorenson and P. Pasquier, "Towards a generic framework for automated video game level creation," in *Applications of Evolutionary Computation*. Springer, 2010, pp. 131–140.

[5] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelback, and G. N. Yannakakis, "Multiobjective exploration of the starcraft map space," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 265–272.

[6] A. Uriarte and S. Ontanón, "Psmage: Balanced map generation for starcraft," in *IEEE Conference on Computational Intelligence in Games (CIG)*. IEEE, 2013, pp. 1–8.

[7] L. Cardamone, G. N. Yannakakis, J. Togelius, and P. L. Lanzi, "Evolving interesting maps for a first person shooter," in *Applications of Evolutionary Computation*. Springer, 2011, pp. 63–72.

[8] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Spicing up map generation," in *Applications of evolutionary computation*. Springer, 2012, pp. 224–233.

[9] R. Lara-Cabrera, C. Cotta, and A. J. Fernandez-Leiva, "Procedural map generation for a rts game," in *13th International GAME-ON Conference on Intelligent Games and Simulation*, 2012, pp. 53–58.

[10] M. G. Friberger, J. Togelius, A. B. Cardona, M. Ermacora, A. Mousten, M. Møller Jensen, V.-A. Tanase, and U. Brøndsted, "Data games," in *Foundations of Digital Games*, 2013.

[11] A. B. Cardona, A. W. Hansen, J. Togelius, and M. Gustafsson, "Open trumps, a data game," in *Foundations of Digital Games*, 2014.

[12] M. G. Friberger and J. Togelius, "Generating interesting monopoly boards from open data," in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2012, pp. 288–295.

[13] I. Celino, D. Cerizza, S. Contessa, M. Corubolo, D. Dell'Aglio, E. D. Valle, and S. Fumeo, "Urbanopoly–a social and location-based game with a purpose to crowdsource your urban data," in *International Confernece on Social Computing (SocialCom)*. IEEE, 2012, pp. 910–913.

[14] J. Togelius and M. G. Friberger, "Bar chart ball, a data game," in *Foundations of Digital Games*. Society for the Advancement of the Science of Digital Games (SASDG), 2013.

[15] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *Pervasive Computing, IEEE*, vol. 7, no. 4, pp. 12–18, 2008.

[16] P. J. Baldevbhai and R. Anand, "Color image segmentation for medical images using l* a* b* color space," *Journal of Electronics and Communication Engineering*, vol. 1, no. 2, 2012.

[17] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *IEEE Symposium on Computational Intelligence and Games (CIG)*. IEEE, 2007, pp. 252–259.

[18] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.