Procedural generation of populations for storytelling

Bas in het Veld
Utrecht University
basinhetveld@gmail.com

Ben Kybartas
Delft University of Technology
B.A.Kybartas@tudelft.nl

Rafael Bidarra
Delft University of Technology
R.Bidarra@tudelft.nl

John-Jules Ch. Meyer
Utrecht University
J.J.C.Meyer@uu.nl

ABSTRACT

Procedural world generation is often limited to creating worlds devoid of people and any background. Because of this, creating a vibrant, living world is still a problem that requires a skilled designer. In this paper, we present a method that generates a socially connected population in any virtual terrain, using a mixed-initiative simulation of settlements that adapt to the world and to a designer's input. Using this simulation, we develop a number of sample worlds that convey the expressive potential of the approach. We further evaluate ease of use with a user study. As a proof-of-concept, we implement the system to bridge the output of a terrain generation tool to the input of a narrative generation tool.

Keywords

computational storytelling, population generation, procedural content generation

1. INTRODUCTION

Procedural generation techniques are often necessary for creating the diverse, rich worlds found in many exploration and role-playing games. Games like SKYRIM [2] make use of procedural landscapes and, uniquely, procedural story generation (in the form of the Radiant Quest system). However, the landscapes by themselves are boring and lifeless, and fail to provide the most important elements for a story, the people. We argue that to better integrate story generation into existing procedural generation techniques, we need to explore methods for creating populations that contain the necessary data to be used for story generation. DWARF FORTRESS [1] notably creates worlds filled with an overwhelmingly diverse population, by simulating the *history* of the world, creating towns, populations and social relations out of the evolution of this world over time. This leads to very diverse results, however the methods of DWARF FORTRESS do not allow for designer intent and are therefore inescapably tied to one context.

This paper describes a method that takes a landscape as input, and uses a designer-driven historical simulation to generate an entire population, complete with settlements, individual people and social relations. The designers can customize their worlds in a number of ways, and work at the level of population building rather than individual character design. Using the definitions provided by the designer, a world is simulated, using an optimization algorithm based upon evolutionary algorithms to accurately determine whether populations migrate, collapse, or develop new relations with each other. In the case of *offline* generation, the designer is granted further interactions during simulation, to be able to fine-tune the positions and layouts of their populations as seen fit. A massive set of characters are created as the result of this simulation, complete with relations and properties relating to their corresponding settlement. Since the population is created after the simulation, the user has full control over its size. For experimentation, the algorithm has been tested with a variety of input landscapes and population designs, and an evaluation was performed to determine the ease of use and openness of the algorithm to iterative design. Furthermore, as a proof-of-concept, the algorithm has been successfully used to connect a sketch-based landscape generation tool [13] to a narrative generation tool specifically geared towards creating quests for RPG games [7].

2. RELATED WORK

Storytelling systems which work with large populations are quite rare. In interactive fiction, the worlds are typically populated with a small number of well-defined, complex, hand-authored characters. Even larger scale emergent storytelling games such as McCoy et al.'s PROM WEEK [11] contain relatively few characters.

Typical approaches to combining procedural content with game worlds have focused on tying the story directly into the world generation, by creating dungeons [4], maps [14] or even entire game worlds [6]. In the latter case characters are generated, but only to serve particular events in a given plot. Our approach, instead, targets *emergent* game environments, in that we do not care about the representation of a single story, but in creating what Mateas [9] describes as a *narratively pregnant* world, one rich with potential for many stories.

Lebowitz [8] explored methods for creating characters to be used for storytelling in his UNIVERSE system. His focus was on creating characters with a personality that was consistent and coherent with an existing world. However the goal of that study was to create complex new characters that integrate properly into a hand-authored set of characters. Our method does not focus on creating complex, completed characters, but instead on large-scale populations with plausible and consistent interrelations.

The history generation of DWARF FORTRESS [1] served as inspiration for our approach to population generation. While the details of the generation are unknown, the game creates relatively templated characters, but situates them in a world in which several hundred years of history are simulated. Likewise, the large scale approach to having relations between settlements is inspired by the complex social relations present in CRUSADER KINGS II [12], where European countries develop alliances and conflicts based on events during game play.

Even though population generation is never mentioned, Emilien et al. [5] use an algorithm based on lichen growth [3] to determine the placement of villages in an arbitrary terrain. Inspired by the results of this method, we expand upon the original method to integrate social interactions and support population generation.

3. OVERVIEW

Our method can produce a world populated by virtual characters that can be used for storytelling, using an empty landscape as input. The resulting population is not just a large set of randomly generated characters - each character has a little background and relations to other characters and places in the world. Characters generated by our method are simple, but still contain enough information to allow them to become a believable and interesting part of a story, while all of this information is still consistent with other characters from the population.

To create our population, we simulate a number of settlements in the landscape, much like the method described by Emilien et al. [5]. These settlements form the basis of the creation of the population, as each settlement represents a sort of blueprint for the characters that are generated from them.

We use an algorithm that optimizes the fitness score of each settlement, which is determined by the landscape and the designer-defined way the settlement interacts with this landscape and other settlements. Our method lets these settlements optimize, and also allows relations between them. We allow a designer to create populations by looking at the landscape and specifying how characters would live there. They can design the types of settlements they'd like to see as well as the relations that may exist, and simulate the world until the results are satisfactory.

4. METHOD DESCRIPTION

In this section, we take an in-depth look at the proposed method, how to design a population, and how the simulation and character generation take place.

4.1 Definitions

The input of our method, **the landscape**, is defined as

$$\langle H_m, T_m, (F) \rangle$$

where H_m is a height map, T_m is a terrain type map, and F is an optional set of terrain features such as forests and rivers. The terrain type map simply indicates the terrain type for each pixel on the terrain. Examples of these terrain types are grasslands, mountains, hills, etc. Both the terrain types and terrain features can be defined by the designer and are used later to determine where settlements, and their inhabitants, prefer to be.

The output of our method is a population composed of virtual characters. A **character** is defined as

$$\langle S, D, R \rangle$$

where S is the character's settlement, D is the character's district, and R is the set of relations this character has to other characters and settlements. The characters generated by our method are relatively simple: they do not have a strong personality, life goals or a specific history. What they do have, however, is a general but consistent background and social network, which should make it fairly easy for a storyteller to adapt them to be more specific. Essentially, our characters are nodes in a population's social network.

A **settlement** is defined as

$$\langle P, D_l, R, P_r \rangle$$

where P is a position in the virtual world, D_l is a list of one or more districts, R is the set of relations this settlement has with other settlements and P_r is this settlement's prototype. Settlements are used as the main tool for creation of the population, as they serve as a blueprint for characters generated from them.

A **district** is defined as

$$\langle N, P_d, R_a \rangle$$

where N is a list of needs, P_d is a list of products and R_a is a list of relations this district allows its parent settlement to use. A settlement adopts needs, products and allowed relations from its districts, making districts an important building block of settlements.

A **product** is a resource function, much like those used by [5] and [3]. These functions can be seen in figure 1. Examples for use of these functions are distance from water (Close distance function), terrain slope (Balance) and terrain types that provide resources - woods, mountains, water (for fishing) and fertile land (Open distance function). These functions are especially well suited for our optimizing algorithm, as their gain diminishes as they approach the optimum.

A **relation** is defined as a

$$\langle T, S_s, D_s, R_e, (A) \rangle$$

where T is the relation's type, S_s is the set of settlements the relation applies to, D_s is the set of distances relevant for this relation (a preferred distance and a maximum distance), R_e is the set of resources that can be exchanged for this relation, and A is the relation's optional attitude, which

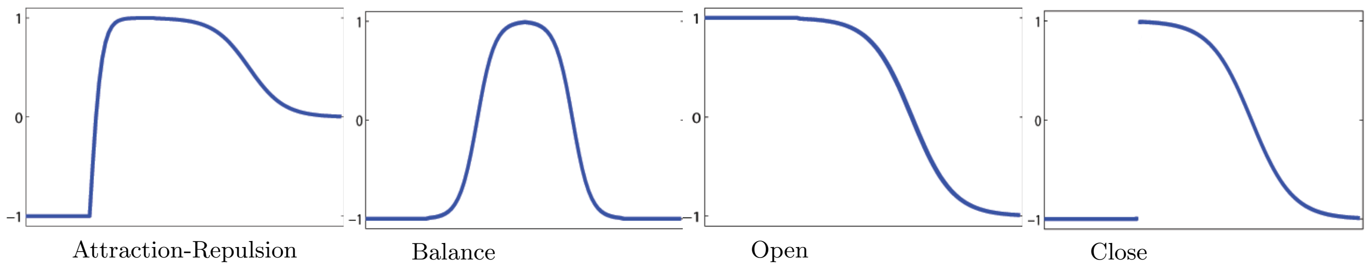

Figure 1: The resource functions[5] used by a district to determine its resource production efficiency. The x-axis is the relative distance from the terrain feature that allows resource income, the y-axis is the resource fitness - the amount of resource produced.

might restrict other relations from forming once an attitude is established. Relations form an important way for settlements to exchange resources, and also a strong basis for the relationships between the characters in the population.

4.2 Designing the virtual world

To create a world, a designer needs to define three of the method's main ingredients: Districts, Relations and Prototypes. Also, designers must tell the system how many settlements they want of each prototype, and from there they can just watch the world unfold, interfering if desired.

The relations and districts are strongly tied to the population that is later generated: Any relation that is between two settlements will cause such relations to exist between members of their respective populations, while districts can say a lot about a character's background. **Prototypes** have an indirect effect, as they force settlements to adopt a certain stereotype. The effect is that rather than adapting to the world, the settlement should instead find its optimal place in the world. To illustrate the strong effect of prototypes on designing a world, consider Figure 2. In the first image, we did not design any prototypes, and designed fishers to be slightly more efficient at producing food when close to the sea. As a result, most settlements decide to be fishing settlements. In the second image, we introduce two prototypes: Fishers, which demands the settlement has a fishing district and no agricultural district, and Farmers, which works the other way around. Both examples were allowed 15 iterations of optimization.

Designing a district is very simple. Throughout the paper, please refer to Table 1 for some definitions. In these examples, the resource functions should be read as:

$$\delta(R, R_m, T, D_{min}, D_{max})$$

where δ is the type of distance function, for example distance_open, R is the resource that is being produced, R_m is the multiplier, signifying how much of the resource is being generated, T is the type of terrain feature, D_{min} is the minimum distance to this terrain feature and D_{max} is the maximum distance to this terrain feature.

By adding the option of a *resource multiplier*, one can make certain districts more efficient at producing them. In effect, a multiplier of 1.0 means that under optimal conditions (for example, a fishing village at its minimum distance to water), will produce exactly enough food to sustain itself.

Table 1: Example district definitions

Fishing
needs: food produces: distance_open(food, 2.0, water, 30, 150) relations: Trade
Agricultural
needs: food produces: distance_open(food, 3.0, fertility, 0, 5) relations: Trade
Military
needs: food, metals produces: distance_open(manpower, 4.0, domination, 0, 0.3) constant(1.0) relations: Raid
Mining
needs: food produces: distance_open(metals, 1.0, mountains, 0, 200) relations: Trade

Using only the definitions from Table 1, we can create military settlements raiding fishing settlements, while other settlements are optimized to have agricultural, mining and military districts and to be self-sufficient. In this example, we are using only three resource types and two types of relations. Recall that the open distance function has optimal results for close distances, but declines as it gets closer to the maximum. In the Fishing district, the production of food is an open distance function using water as terrain feature, a minimum distance of 30 and a maximum distance of 150. This means that such a settlement's fitness will be optimized by moving closer to water, until its distance is 30 or less. The Military district, described in Section 5, behaves similarly, but our domination map used values in a range [0.0 1.0], hence the smaller range. This domination map was generated from the height map, and is useful to determine relative height for each terrain pixel. The military district produces its resource manpower at a constant rate, no matter the terrain domination. All resources and types of districts are simply examples we came up with, so a designer can define any named resource by defining it as need or product for a district.

Designing relations is quite easy, as they are modeled as restrictions on a resource exchange. In Table 2 we can see

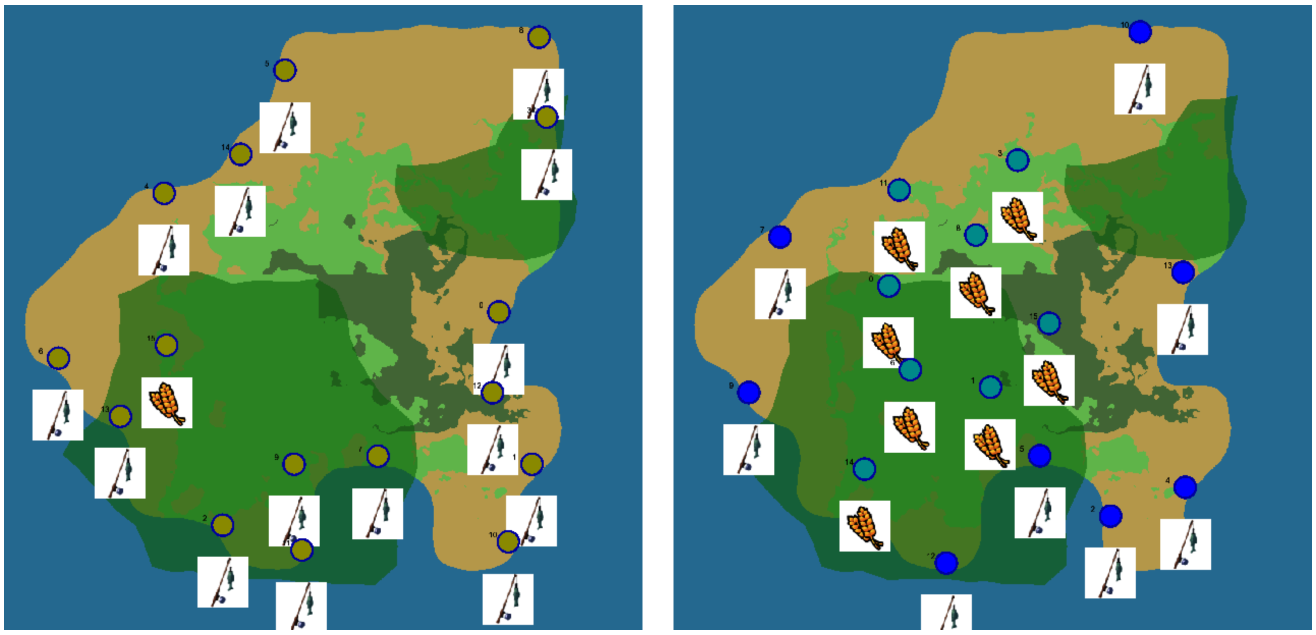
Figure 2: Left: A simple island with only fishers and farmers, no relations and no defined prototypes. Right: A simple island with only fishers, farmers, no relations, but with two prototypes that cause a designer-defined amount of fishers (8) and farmers (8) to exist.

Table 2: Example relation definitions

Trade
in(*)
out(*)
distance_open(550, 650)
attitude_required(not Negative)

Raid
in(*)
out(manpower)
distance_open(250, 400)
attitude_effect(Negative)

two basic relations we used during testing. In this example, the Trade relation has no restrictions on its import or its export resources, it has a preferred distance of 550, a maximum distance of 650 and it requires the existing attitude not to be Negative. The attitude of a relation prevents another relation that has a different attitude to also establish between two settlements. For example, if settlement A raids settlement B, we would not expect B to establish a trading relation with A. To do this, we allow relations to have an attitude *effect*, and an attitude *requirement*, which can take the value 'Positive', 'Negative' or 'Neutral'. As soon as two settlements have a relation between them that has such an effect, the attitude between those takes on this value. For example, a raid relation's effect is 'Negative', while a trade relation's requirement is 'not Negative'.

Designing prototypes involves determining what the restrictions are for a settlement. The main restrictions the designer can apply are:

- Which district and relation types are (not) allowed?
- How many districts/relations are allowed?
- How does the settlement respond to proximity of other settlements?

With these simple properties, prototypes become a powerful tool for designers to steer their settlements in their desired direction. For example, by limiting the maximum number of relations and districts, we can force settlements to specialize. We can also create a simple raiding prototype by using Military as the only allowed district, and having Raid as preferred relation.

Finally, we allow our designer to determine a prototype's *social* type, which helps determine how the settlement deals with other nearby settlements. The social type can have two values, or it can be undefined: Master or Slave. A master settlement has a territory and benefits from having slave settlements in that territory, and is penalized when its territory overlaps other masters' territory. A slave settlement benefits from being in any master's territory, and only has slight penalties for being close to other slave settlements. If no social type is defined, a settlement simply benefits from keeping its own preferred distance from other settlements. A master's territory and regular settlements' preferred distances can be defined for each prototype, making it very easy to introduce a degree of size to each prototype's settlement. We experimented with around 5 master settlements and around 20 slaves, and simulating the world quickly caused the masters to take their own space in the world, while the (smaller) slave settlements quickly distributed in these territories. In the resulting population, there was a clear social network between masters and their slaves, while settlements without a social type tended to be

Table 3: Example prototype definitions

Large_Military	districts: Military social_type: Master master_territory_size: 350
Small	max_districts: 2 max_relations: 3
Small_Food : Small	districts: not Military max_districts: 1 social_type: Slave
Extra_Large	social_type: Master master_territory_size: 450
Small_Raider : Small	districts:Military relations: Raid districts: not Fishing not Agricultural
Small_Mining	social_type: Slave districts: Mining max_districts:1

solitary. Table 3 shows a few prototype definitions. Note how definitions can be used as 'parent' definition, copying all properties from it; in this way, for example, Small_Food has all properties from the Small prototype.

4.3 Simulating the virtual world

In this section, we discuss the simulation of the virtual world, and the mixed-initiative interaction between the simulation and designer. It is possible to simply simulate a number of generations, then generate a population with the press of one button. However, if designers have a certain world in mind, we wish to enable them to create it just as they want it. For this reason, we designed the method to be mixed-initiative, allowing the designer to proceed to the next generation of settlements, make customizations, and continue. We designed our interface to allow designers to simply drag and drop settlements, and change a settlement's properties, immediately seeing the effect that it has on their fitness. At any time, designers can allow the system to simulate the next generation. Once the designers are done, they can generate the population from the current generation, and still keep going after that too.
Before the initial generation can be created, the designers specify the types of settlements they would like to see in his world, by supplying a list of prototypes. Each prototype is assigned to a settlement and placed in the world by 'polling' a number of locations (we polled 15 times) in the landscape, and determining the fitness of the settlement, should it be placed there. For each settlement, the best scoring poll is selected for the first generation. The advantage of using a polling method is that the resulting locations are nondeterministic and cause the settlements to be in local optima rather than global optima after simulating. After all, we are not looking for the optimal situation of the simulated world - rather, we want to generate a possible way a population may exist in that world. The fitness of a location for a settlement is determined by the following factors:

Landscape : is the slope suitable, is the spot legal (e.g. no water)?
Resource : based on the settlement's districts and relations, how well will it do resource-wise?
Spacing : are we too close or too far from other settlements. This depends on the social type of the settlement.
Prototype : is the settlement true to its prototype?

Because we are working with an optimization algorithm, we scored all these factors on a scale from 1.0 to potentially negative infinity (but usually -1.0). For example, if the landscape is perfect - no slope and the position is legal - the score for that will be 1.0. For resources, this was trickier, since more should always be better. For this reason, meeting all needs gives a score of 0.5, and as the amount of resources approaches infinity, the score approaches 1.0. We use similar strategies for the other fitness factors and compute the final fitness by taking their equally weighted sum, causing settlements to optimize on all fronts.

The spacing fitness is determined by distance to all nearby settlements. It depends on the social type of the settlement, but each settlement always has a preferred distance to each other settlement. A settlement with no other settlements in its preferred distance radius has a spacing fitness of 1.0, while each nearby settlement diminishes this value. Master and slave settlements are an exception: Master settlements do not lose fitness when slaves are in their territory, while slaves lose fitness if they are not in a master's territory, proportional to the distance to the nearest master (causing them to move toward that master).

When a new generation is created, each settlement makes copies of itself and mutates them slightly. Mutation involves a change in one or more properties including position, relations and districts. The 'child' that has the highest scoring fitness is chosen and used in the next generation. To prevent a settlement from becoming less optimal by mutation, we always include the original settlement in the selection process.

4.4 Population generation

When the designer is satisfied with the world, the population generation can take place. For each settlement (prototype), the designers can determine how many characters they want to have generated by the method. In this phase, the settlements act as character generators: each settlement now has districts, which help determine the background for a character. For example, a settlement with a Military district might produce soldiers or raiders (depending on relations), while fishing villages should produce mainly fishermen. Furthermore, settlements have relations to other settlements, and often they already have attitudes to other settlements too. So, if we have settlement A raiding settlement B, the attitude between these two settlements is Negative (see Table 2). If we now generate a character for settlement B, it can automatically inherit this from its settlement: it has a negative attitude towards A, and anyone from A. To give the character more flavour, we can even give this attitude flavour: the attitude is negative because A raids B.

However, if we generate all characters like this we end up with a whole group of characters from settlement B that hates every single person from settlement A. For a number of reasons, this is undesirable. For example, a story such as Romeo and Juliet would not be possible. It also makes sense that many characters from settlement B do not even know anyone from settlement A. This is why we allow the designer to set a few simple values for character generation:

inter_settlement_relation_chance: the chance a character

knows another character from a different settlement.
inter_district_relation_chance: the chance a character knows another character from a different district (in the same settlement).
intra_district_relation_chance: the chance a character knows another character from the same district.
attitude_change_chance: the chance a random attitude change occurs between any two characters.

These values can be used to add some variation to the initially generated characters. We found that if these values are not used, an enormous amount of relations is being generated: every character will have a relation to every other character in the settlements their own settlement has a relation to. However, since we expect the output of our method to be used by a procedural story generator, it can also be considered the responsibility of that system to tune these possibilities.

5. APPLICATION

To test our method, we have built a number of different input landscapes, and designed varying sets of districts, relations and prototypes. In this section, we present a few designs we made, and show how our method deals with these inputs. We show a simple world in detail, and consider some others more broadly.

Our first world is a medieval-themed design, and has also been the running example throughout the paper. We used exactly the districts as presented in Table 1, the relations in Table 2 and the prototypes in Table 3. The initial generation of the resulting world can be seen in Figure 3 (left). The images under a circle signify which districts a settlement has. Even though no optimization has yet taken place, the settlements often have locations that are quite suitable for their prototype. However, without simulation the social graph is quite limited. Figure 3 (right) shows the same world after 10 generations without user intervention. It is clear that most settlements remain true to their stereotype, but without user intervention, some others have not been able to escape their local maxima (yet). For example, the settlement in the far left bottom should be large and have a military district, but since there are absolutely no other settlements around to have relations with, it became a fishing settlement instead. Of course, this problem was already there in generation 0, and could have been fixed by the designer, e.g. by simply dragging it to a new position.

Expanding this first example to be more specific is quite simple. We added a hunter-gatherer district that generates food from forests, and to give our world a fantasy game-like feeling, we introduced a few prototypes: *Elven* : Using the new hunter-gatherer district only, making them drawn to forests. *Dwarven* : Using the mining district, making them drawn to mountains. *Orcs* : Only allowed to use Military districts, and Raid relations. By making only these simple changes, the designer can expect elven settlements (usually) in forests, dwarven settlements in the mountains, and orcs raiding everyone, everywhere. Although interestingly, sometimes an orc settlement would be too far away to raid anyone and would instead become a self-sustaining fishing settlement.

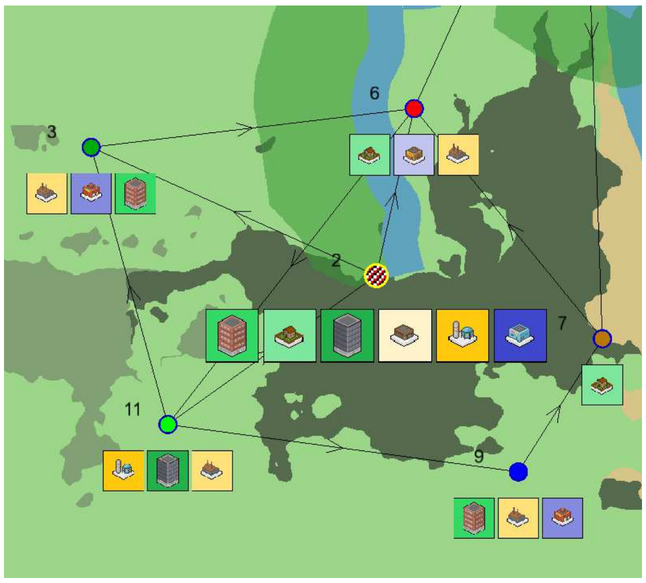

Figure 4: Simulation of a SimCity [10] like world. Commercial, industrial and residential districts of low, medium and high wealth form cities that exchange resources like workers, materials, products and even influence and money.

Beyond this quite straightforward example, we developed some very different approaches. Figure 4 shows a region of the same landscape for our fantasy-themed example, but modern settlements are being simulated instead. Aiming to mimic the basic mechanics of SimCity [10], we defined residential, commercial and industrial districts of different social classes: low wealth, medium wealth and high wealth. We model unique relations between cities, allowing trade and freight shipments, but also allow cities the option to use influence and bribery to get resources from other cities. The result is a much darker, more corrupt vision of modern society that may be more befitting of storytelling.

Another popular setting for storytelling and games is the wild west, so we simulated a world where the collection of gold is the goal for (most of) the population. We set up our landscape to contain just a few places where gold can be collected, and designed a specialized prototype that can collect gold while the others cannot. Other settlements are allowed to be towns that provide workers and earn gold by having them work in the mines, bandits that can steal gold, and 'lawbringers' that can arrest bandits. In Figure 5, we can see how all settlements converge toward each other, even though we did not define any master or slave prototypes in this scenario.

Exploring alternative forms of resources, our final population example aimed to mimic zombie apocalypse worlds. This setting uses people and zombies as the main resources, allowing zombies to infect human settlements and humans to cure zombie settlements. Relations are mapped to scavenging, zombie hunting, resource gathering, and acquiring medical supplies. To demonstrate the versatility of the system, we generated four different populations in very different landscapes, as shown in Figure 6. Likewise, we split the prototypes into one large city which contains the medical center

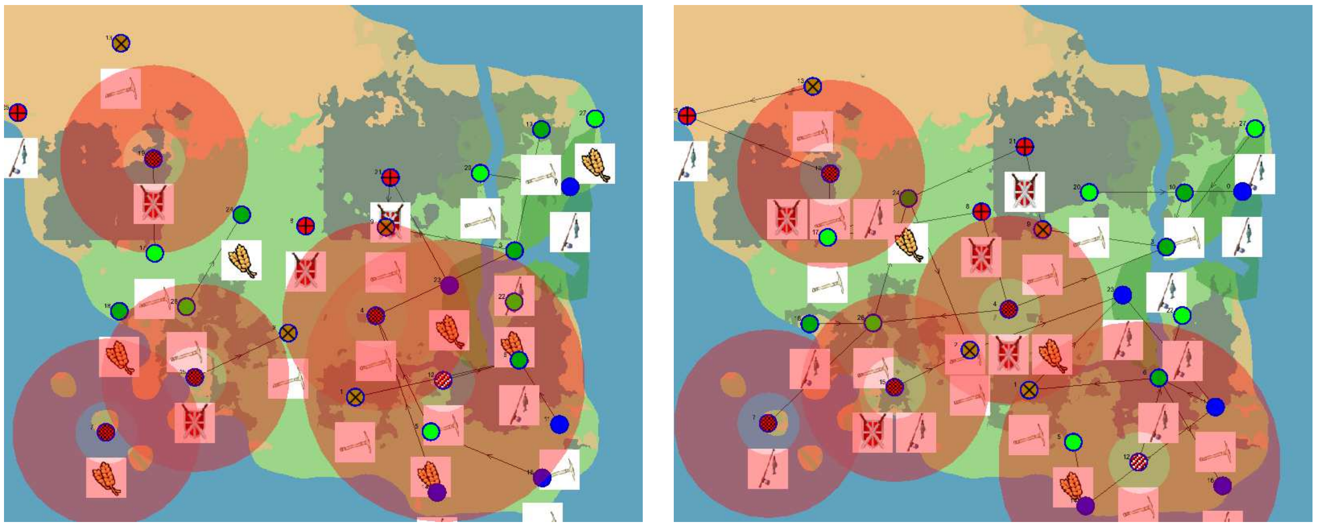

Figure 3: The initial generation of a medieval-themed world, defined with the values presented in the example tables. A plus sign is a Raider prototype, a cross is a Mining prototype, settlements with a territory are the Large (Military) prototypes, and all the others Small. Left: Initial generation. Right: world after 10 generations.

and laboratory needed to create zombie cure, several cities largely infected with zombies, and then a number of small scavenger or survivor groups.

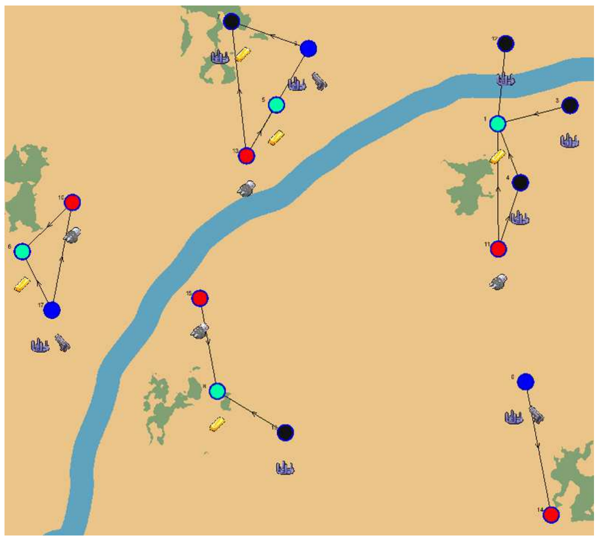

Figure 5: A wild west scenario, with only gold as terrain-based resource. Because of this, all settlements tend to converge to these locations.

6. EVALUATION

With our method, designers can populate a virtual world by designing districts, relations and settlement prototypes. Since this approach is rather involved, we wanted to test how easily people unfamiliar with the system could use it to create a population.

For the experiment, we asked 9 persons to do a number of tasks with our method. Our participants were all between 20 and 26, 5 were female, 4 were male and none of them has a background in computer science, although two have experience playing computer games. We briefly explained what the method does, how the designer can interact with the system and introduced the medieval population example shown in Figure 3. Furthermore, the participants were given a 'cheat sheet' that displayed the main options and available terrain features for the landscape they worked in. After this introduction, the participants were given definitions of the Fishing district as shown in Table 1, the Trade relation as shown in Table 2 and the Small prototype as shown in Table 3 to serve as a start for their design. Using this, they were asked to do the following tasks to incrementally improve their world.

1. Add an agricultural district that produces a resource 'food' from the terrain feature 'fertile'.
2. Create a world that has around 8 settlements that have the Agricultural district only, and around 8 that have the Fishing district only.
3. Create a large settlement that has surrounding villages.
4. Create small settlements that cannot produce food, but rather have to steal it from other settlements (raiders).
5. Make the raider settlement rely on 'metals' as well, and create a district that can generate this resource from mountains.

We were primarily interested in seeing how many times participants had to make a change to their design after getting results that differed from their expectations. After working with the system ourselves, we already noticed that trial-and-error is a common way of fine-tuning your world, and this is visible in the retries of participants for each trial shown above:

Task 2 proved especially challenging, as our landscape had a great amount of fertile land. Because of this, the chance of

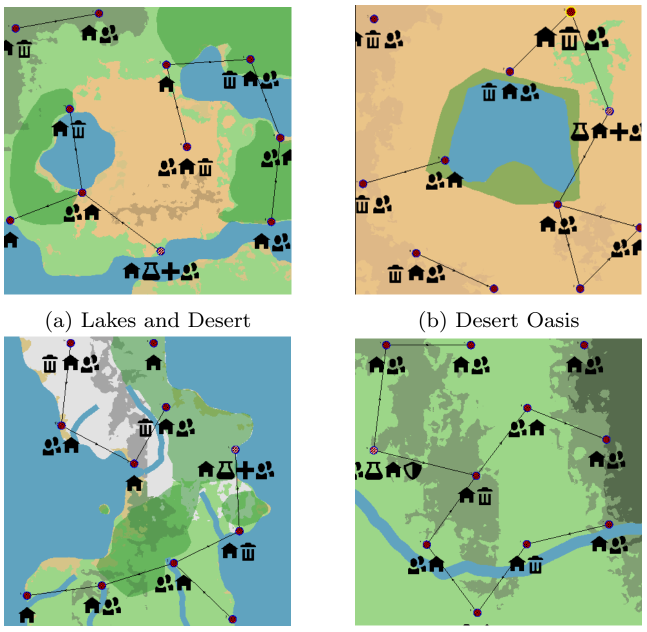
(a) Lakes and Desert (b) Desert Oasis (c) Frozen Island (d) Natural Valley

Figure 6: Four different populations of the zombie apocalypse scenario. Each contains radically different resources available, yet the system is still able to create logical populations for each landscape.

Task	1	2	3	4	5
Average retries	1.6	5.7	2.3	4.0	2.9

a settlement becoming agricultural was greater, and participants needed to utilize prototypes to make fishers appear more often. We required the participants to have 7 to 9 of each settlement type in the first generation, without user intervention, in order to complete this task. Similarly, task 4 required participants to define a new relation, and this was especially interesting because that boils down to using the more abstract resource 'manpower'. Four of our participants used similarly abstract resources ('soldiers', 'barbarians' and 'manpower'), while the rest used resources like 'weapons'.

After the tasks, we allowed our participants to freely try adding districts, relations and prototypes to the world, and asked them if they felt how well the method could help them to create a population as they wanted it. The responses were rather similar: Most participants found the method to be quite complex, but also rather intuitive once they understood the basics. In particular, 8 out of 9 participants mentioned 'trial-and-error', meeting our expectation, especially considering none of the participants had a background in computer science.

7. DISCUSSION

We implemented our method in C++, using Simple and Fast Multimedia Library (SFML) for simple 2D rendering. Most tests were done on a computer with a Intel Core i7 @2.20 GHz. Our implementation is capable of automatically iterating through 50 generations of settlements in under 2 s on average, then generate around 1450 characters in under 1 s on average.

To create our landscape model, we used the procedural terrain sketching tool SketchaWorld [13]. SketchaWorld allowed us to make landscapes in minutes, which made the whole progress much more streamlined.

Our main goal is to make characters fit for storytelling, and to test this goal, we succesfully integrated our system with ReGen [7], a graph-rewriting tool that can be used for narrative generation. We were successfully able to generate narratives within our populations, and further have the population be updated by changes made within each narrative.

Our method has shown to be an effective tool for designing a population for a world. It is true that one cannot fully control or predict how exactly certain ideas play out, since many factors have influence on settlements in the world. For example, when a designer wants a world with lots of farms and fisher villages, but fails to define prototypes, they might find all of them become fishers (if those are defined to be more efficient). However, even if the design is nontrivial, the mixed-initiative designer interaction during the simulation makes up for that.

8. CONCLUSION

In this paper, we present a method for the generation of large populations of virtual characters, with basic but intuitive relations between them. A designer is required to do the creative work, while the method itself simulates a believable world based on the designer's ideas. Even when the designers do not get exactly what they want, they can still strongly influence the outcome of the program. The method is quite fast, making it easy for the designer to experiment and try different approaches. After the designers are satisfied with their world, they can proceed to the generation of meaningful characters who have a real place in the world they were created in, and personality traits that are derived from their origin.

We applied our method to fundamentally different scenarios, proving that it can deal with a wide variety of worlds and populations, from medieval/fantasy themed worlds to wild west and apocalyptic settings.

By letting a small number of people without background in computer science test our method, we have seen that technical knowledge is no prerequisite for using our method. However, the method is not trivial either, and requires designers to understand a few basic concepts before they can start. The experience of our participants has shown that effectively creating a world requires some degree of trial-and-error, since the interaction between settlements, the world and the designer can become quite complex.

For future work, we will improve the interaction the designer has with the method. Right now, designers can edit their designs, then reload the program to see their changes take effect. Instead, being able to make changes to districts, relations and prototypes while running the program would make the interaction even more fluent. Our method is particularly well suited for this kind editing, because it will simply keep optimizing based on its new inputs.

9. REFERENCES

[1] T. Adams. Slaves to Armok: God of Blood Chapter II: Dwarf Fortress. Bay 12 Games, August 2006.

[2] Bethesda Game Studios. The Elder Scrolls V: Skyrim. Bethesda Softworks, 2013.

[3] B. Desbenoit, E. Galin, and S. Akkouche. Simulating and modeling lichen growth. *Computer Graphics Forum*, 23(3):341–350, 2004.

[4] J. Dormans and S. Bakkes. Generating missions and spaces for adaptable play experiences. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):216–228, Sept 2011.

[5] A. Emilien, A. Bernhardt, A. Peytavie, M.-P. Cani, and E. Galin. Procedural generation of villages on arbitrary terrains. *Visual Computer*, 28(6-8):809–818, June 2012.

[6] K. Hartsook, A. Zook, S. Das, and M. O. Riedl. Toward supporting stories with procedurally generated game worlds. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 297–304. IEEE, 2011.

[7] B. Kybartas and C. Verbrugge. Analysis of ReGEN as a graph-rewriting system for quest generation. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(2):228–242, 2014.

[8] M. Lebowitz. Creating characters in a story-telling universe. *Poetics*, 13(3):171–194, 1984.

[9] M. Mateas. *Interactive Drama, Art, and Artificial Intelligence*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, December 2002.

[10] Maxis. SimCity. Electronic Arts, March 2013.

[11] J. McCoy, M. Treanor, B. Samuel, A. Reed, M. Mateas, and N. Wardrip-Fruin. Social story worlds with Comme il Faut. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(2):97–112, June 2014.

[12] Paradox Development Studio. Crusader Kings II. Paradox Interactive, February 2012.

[13] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. Interactive creation of virtual worlds using procedural sketching. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pages 1–8. ACM, June 2010.

[14] J. Valls-Vargas, S. Ontanon, and J. Zhu. Towards story-based content generation: From plot-points to maps. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8, Aug 2013.