Generate 2D/3D Citymaps for Tabletop RPGs

Teresa Hong, Hasham Mukhtar
yhong3@ncsu.edu, hmukhta@ncsu.edu

Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27695

Abstract

In this project, we created a generator for 2D grid based maps with 3D visualization, which allow user to define the inputs based on their needs and generate a valid city that can be explored and navigated intuitively in tabletop RPGs.

Keywords: generative-contents, map-generation, city-generation

Introduction

For our project, we wanted to create a city map generator that will accept user input parameters to determine the characteristic of a town/city, and output a 2D grid with different colored blocks representing different aspects of a town/city like paths, buildings, water, etc. follows with a explorable 3D city viewer. The generator will procedurally generate the map while meeting the requirements of being a valid city map to explore and using parameters from a user.

Background and related work

When playing tabletop role-playing games such as Dungeon and Dragon or Call of Cthulhu, we usually face the situation that we will need a town/city with specific characteristics to match with the game session. Though generating text description for a town is easy, having a map completely hand-drawn for it is hard, and looping through randomly generated maps to match the description is even more tedious. It will be great if we can have a generator that can accept a simple descriptive setting for a town and generate a map for it.

Though there have been many procedural city map generator like Medieval Fantasy City Generator (watabou 2018) as well as some textual city generation like Town Generator for D&D (kassoon 2018), each generator was tuned to their specific use. The Medieval Fantasy City Generator was the most aesthetically pleasing one with the most degrees of freedom, while the author described The generation method is rather arbitrary, the goal is to produce a nice looking map, not an accurate model of a city, so we have no proof that the generated city is valid. Town Generator for D&D has most of the ideal input parameter we want to accept from the user, but it will only generate text and numeric results, which requires an extra amount of effort to create the visualization. Things generated from Wizardawn Fantasy Settlement Generator (WizardawnEntertainment 2006 2018) is promising, but it does not give the user much freedom of determining the characteristic of the city.

Our project is an in-between step of generating a city map for a tabletop RPGs. Since there are already some story and text generation for different aspects of a town, as well as detailed map generators of town. Our project was able to make a simple, grid-like representation of those generated details, and the city is valid enough to be explored for players in sessions. Furthermore, it provide a interative 3D viewer for exploring the generated city.

Besides the validty of the city, the other one of the main problems we want to solve is to generate a map that can be explored intuitively – that is, the player know what to expect to be in a city, so they know which way they should go and what they should expect from their behavior. While at the same time, we also expect the story creator will add their creative part into the city, where the player will be able to sense if there is something wrong with the city.

Since the initial idea came from tabletop RPGs, the game master who plans to run the session can use it as part of their set up and distribute it to the players. Even more, this project would actually be a handy tool for whoever wants to start working on their story with the background set to the city, as it provides a straightforward, simple visualization. Our goal is to generate the city base for the game masters or story creators, so that they can use the generated result as a base, and add their own "suspicious" or "fun" part – a commercial refrigerated truck frequently showing up in the residency area, a religious statue in the middle of business division, even a butcher house in a bar street.

Approach

We wanted our map to be generated based on simple input parameters that people could obtain from online tabletop RPG generators. The inputs we focused on were size and water source. Currently the tool accepts three different sizes: small, medium, and large, while small would be considered a village, medium would be a town, and large would be a city. For the water source, we worked on adding a river

to the map. Early on we had thought about adding walls to the map, but we decided to not move forward with it given the fact that our map is a grid and every square would be a path, building, or water. If we did have a wall, it would just be the outer layer of squares which we felt would not be interesting.

Front-end Web Interface

To allow the user input their parameter without too much background knowledge, a frontend web interface is created in Javascript, with Boostrap and jQuery. Each parameter is implemented as dropdown widget to minimize the possibility of invalid input.

Path Generation

Our path generation used the algorithm called Random Walk (Handelman 1987). The Random Walk algorithm creates a series of random steps of varying length in a defined space. For our project, we needed to set up some initial variables. They were the number of turns, or steps, we wanted our path to have, the max length that each path could be, and a randomly chosen initial square on the grid to start the path.

We created a loop that kept going as long as there were still turns to be made for the path. In that loop, a random length for the path segment is chosen and a random direction is chosen for the path segment to go. Then the path segment was created one square at a time in the random direction and coloring the squares gray to show the path. When the path segment reached the desired length or if it reached the edge of the grid, it stops. The max turns was decreased by 1 and the chosen random direction was stored as a reference. We did this because we wanted the new direction for the next path segment to be perpendicular to the last one. This prevented the path to double back on itself. Then the loop repeats itself until all turns are made.

River Generation

The way a river was created is similar to the path, but with more restrictions. First, we started by randomly picking one of the sides of the grid and then randomly picking a square on that side. Then the direction of the river was chosen based on the starting side. If the top of the grid was chosen, then the main river direction was down. If the right side was chosen, then the main river direction was left and so on. Then we create a loop that is true until the river reached the edge of the grid. We gave it a more riverlike look by having the river change direction each iteration while still going in the main overall direction. We limited the directions by having it only go perpendicular to the last direction and by not being able to go towards the starting side. So if the river started in the top section of the grid, then the direction for the river could never go up, only down, right, or left. The loop colors the squares on the grid blue to represent water.

We also added bridges to the river so that you would be able to access buildings on each side. This was done by looping through the river squares and then checking if both squares above and below, or left or right of that water square is a path square. If it is, then we color that square brown to represent a bridge.

Possible Buildings in the city

We manage all possible building types in a global dictionary to each of the corresponding type and the characteristics.

- `building_type`: String, the building type e.g. tavern, shop, blacksmith, etc.
- `max_width`: Integer, this is used in both 2D and 3D, which determines how wide the building will cross at the side facing the road in front of it.
- `max_length`: Integer, this is used in both 2D and 3D, which determines how deep the building may grow toward the center of the buildings
- `max_height`: Integer, this is only used in 3D version, which determines the maximum height of the current building, the whole building will have the same height.
- `center_possibility`: Double from 0 to 1, this is used to control how often the shop may show up in the center range of the neighborhood. As some of the building may have less demand in the city, thus they are proportionally showing up less in the city. All `center_possibility` in the dictionary should add to 1.
- `feathering_possibility`: Double from 0 to 1, this is used to control how often the shop may show up in the feathering range of the neighborhood. All the `feathering_possibility` in the dictionary should add to 1.
- `outside_possibility`: Double from 0 to 1, this is used to control how often the shop may show up outside the feathering range of the neighborhood. All the `outside_possibility` in the dictionary should add to 1.
- `base_color`: String, this will be the HTML hex color code in format, like `#FFFFFF`, for visualizing the different instances of the buildings in same type. The color will be randomly interpolated between white to the given color, so that the building in the same type will have different brightness value in the same color (e.g. light grey and dark grey)

`center_possibility`, `feathering_possibility`, `outside_possibility` will determine the possibility for the current building type to be generated based on the region it is current located. `center_possibility` is no longer used in the current implementation since we introduced the idea of multiple urban area (neighborhoods) instead of using fixed possibility. Each of the different city size will now have different possible center of the neighborhood based on the type of development the city have went through. Currently small map will have 1, medium has 2, and large city has 3 different neighborhoods. The current implementation will only detect the current neighborhood center and create different instances of current building type. For `feathering_possibility`, we use this variable to determine the building distribution in the suburban area. Finally, we use `outside_possibility`

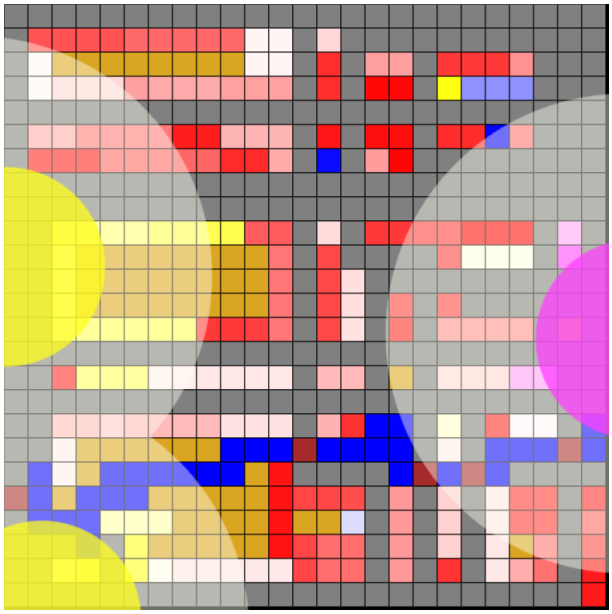

Figure 1: Diagram for the neighborhoods, solid color shows the different neighborhood center, white translucent area are the feathering area, everything else is outside of the neighborhoods.

to determine the building distribution in rural area, which does not belongs to any of the neighborhoods. A diagram that visualize the idea can be found in figure 1.

Building Generation

First of all, determine the building type of the neighborhood. Our idea is to mimic the process of how a city developed from downtown as the core of the city, and gradually scatter out to the urban area when downtown is too crowded. The process will divide the city into urban, suburban, and rural area, which we use the neighborhood center for urban, feathering radius for suburban, and out of neighborhood for rural, to represent such characteristics.

After deciding the type of the neighborhood, generate the location coordinate of the neighborhood center. At the same time, determine how large the main neighborhood area will be in radius, as well as decide the feathering area radius. Once everything about the neighborhoods is ready, start looping through each of the cells in the grid, by always loop the grid from up to down, from left to right. A diagram explains the generation process can be found in figure 2.

1. Calculate the distance between the current cell and each neighborhood center. Determine if the current cell is in any of the neighborhoods, or in any of the feathering area of the neighborhoods.
2. Use the possibility from the possible building dictionary to select the type of the building
3. Detect what is the status of the next few cell to the right in the same row to know how far the current building can extend, call the detected amount as `empty_right_cells`

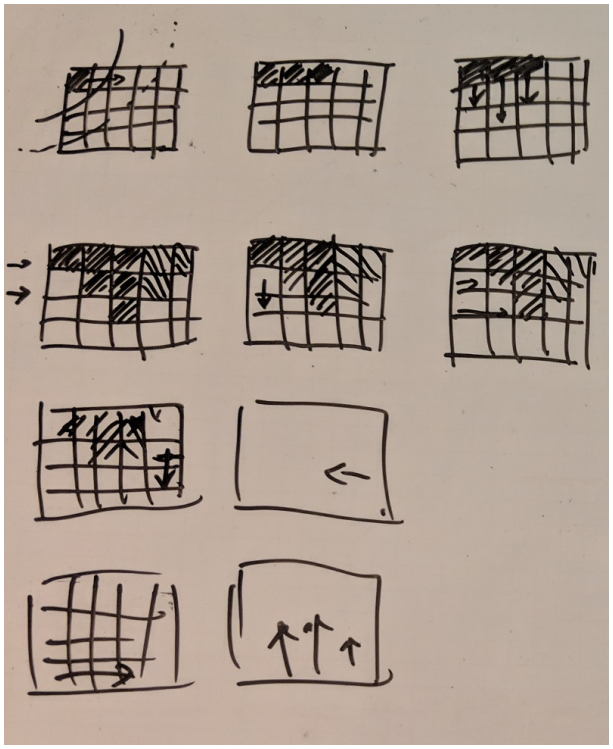

Figure 2: Building generation steps

4. Determine width of the building between min(`max_width`, `empty_right_cells`)
5. Grow the building horizontally
6. At each column of the building, detect what is the status of the next few cell inward to know how far the current building can extend, call the detected amount as `empty_down_cells`
7. Determine width of the building between min(`max_length`, `empty_down_cells`)
8. grow the building vertically
9. repeat the above steps for each direction

If the cell is in the conjunction of multiple neighborhoods, the current design just assign the current cell to the first neighborhood it belongs to, but this can again be a future improvement goal that make the choice based on which neighborhood is more powerful or more important.

Note that while this cannot guarantee there will be enough building space in the same neighborhood, this is still one of the design decision we made so that the neighborhood size feels balanced. This can also be modified to emphasize the unbalanced force distribution in the city (e.g. A specific powerful noble family) for more flexible result.

3D Visualization

To isolate the structure of 2D and 3D generation, we created the 3D viewer as a independent webpage, then used iframe to load the 3D viewer and pass the generated result from 2D

to 3D for visualization. To ensure compatibility on multiple browsers, We used the javascript 3D library for webGL named three.js.

The 3D viewer will update the scene every time when a new map is generated. To allow intuitive interaction, we chose the control similar to most of the PC games, using WASD for moving and space key for jumping. We also allow capturing mouse movement to simulate head movement for view angle change. To allow user go back and forth between the webpage and the 3D viewer, the 3D viewer is by default locked and will only be unlocked when user click inside the 3D viewer. The user can also exit the 3D viewer anytime by pressing Escape key, which will lock the 3D viewer again.

Results

Since our intention was to generate an intuitive city map, most of the evaluation are visual and subjective. In this section we will briefly discuss on what is the result we find and how did we try to solve the found problem.

2D v.s. 3D

As the project originally was to just generate 2D version of the map, we soon find out that in 2D, it is hard to tell each different instances of the same type of building (e.g. different houses in same living area) even with color coding. Also, It is hard to keep track on what will be visible to the user in their current position, which make the map less usable for a real tabletop RPG session. To solve the above problem, we implemented the 3D viewer and use the 2D version majorly for overview, but rely on the 3D version more for fine inspection.

City validity

City size One thing we noticed in the middle of the development is the difficulty to tune the best result for all 3 different size. We originally thought that coming up one generic algorithm will be enough for different sizes, but even with a glance, it is easily told that the same algorithm was not giving the same level of results. This is one of the major problem that has significantly make the fine-tuning work time-consumable. It would be much better if we targeted for only one size and make it better. The problem still persist at the current version of the project, which large version looks more natural, while the small version sometimes generate the result that lack of some of the buildings. To sum up, we found out that using the same parameter in different sizes may lower the validity of the generated city.

Building Generation At very first, we treat each connected-components as one randomly shaped building. It was implemented by converting the result into binary inputs by treating ground as foreground pixel (1) and roads as background pixel (0) to apply connected-component labeling problem (He et al. 2017) to label the connected components as same region to use as building space after the random walk have decided which spot is ground and which is the road. However, after quickly visualizing the output like figure 3, it is recognizable that we have no control on how

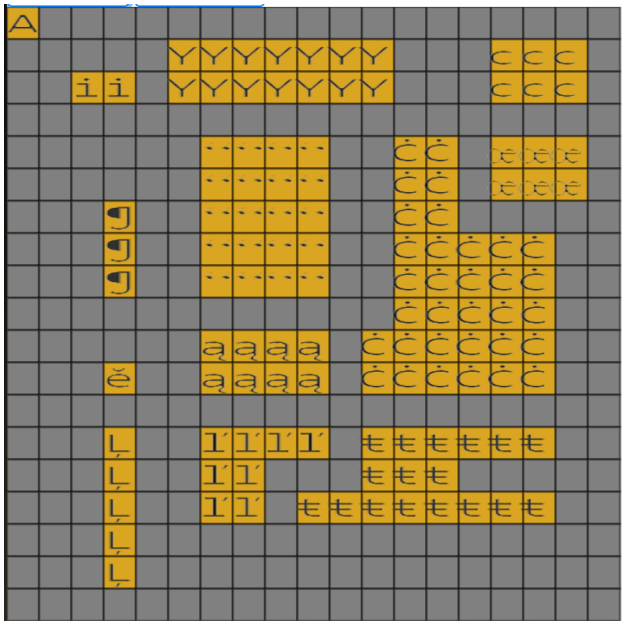

Figure 3: Connected Components Labeling output, which each different character represent a block belongs to the region

large each connected components are, and each of the connected components are scattered out based on the random walk's path. The output result is not even close to realistic.

To solve the problem, we introduce the idea of neighborhood from inspiriation of Mouse Reeve's talk about Mapping Imaginary Cities (StrangeLoop 2018) that create voronoi digram that use a point and specific radius to determine what is in the same neighborhood. At very first we wanted to follow the idea of downtown as a center of business district. Therefore we determined the neighborhood center to one of the non-residency buildings by adding a Boolean variable `can_be_center` to indicate if the current building type can be considered as center of the neighborhood randomly choose from the predefined buildings that has `can_be_center` being `true`. However, after reading a variety of different scripts online, we found out that it is still worth considering for some of the specific stories that want to be more centralized in reactions between players and non player characters (NPCs), In this case, allowing a city that is centralized in residency may be acceptable. Besides, it will actually reduce the complexity of the implementation.

After introduce the neighborhood concept, we still found the result far from perfect. The current setting only have one preset of building types, so it only consider each neighborhood area have the same building type in the center, as well as using the same ratio for building generation for all of the suburban area every time. However, in the real city, the building type in the urban and suburban area should varies for each type of the neighborhood. Therefore, this project only demonstrated one combination of the possible results, which limits its adaptability.

Limitation

Since the result is randomly generated every time, it is not practically usable for a multi-player tabletop RPGs session that requires all players opening the same generated map. Plus, it is currently impossible to save and retrieve a specific generated result. Also, since the current session only allow single player, it is not trivial for the user to manage who are in the same sight.

Discussion

Evaluation design

Like most of other generative contents models, this project cannot benefit much from computerized and quantified evaluation. Since it is meant to be used as part of the base creation tool in tabletop RPG sessions, and tabletop RPG is especially famous for providing out-of-ordinary experiences for each session due to its flexible and open-ended gameplay. Even with exactly the same script, the different group of players may have completely different playing styles. These factors make evaluation being, even more, harder and subjective and require many test plays with the tool to find out a less biased result.

To design a valid evaluation, we want to emphasize on the gap between the player and game master on the understanding of what to expect in a city, which make the generated city non-intuitive to navigate through.

Attendent Playing Style Since the tool is designed for tabletop RPG players, it is important to make sure the evaluation design is adaptable to various playing style, Therefore, it is better to design the evaluation start from taking some of the most generic playing styles into consideration. Though not officially defined, one of the well-known categorizations are "Real Man, Real Roleplayer, Loonie, and Munchki" by Jeff Okamoto, Sandy Petersen, and other people (Kovalic and Lowder 2016), which mentioned: " The Real Man ... the tough macho type who walks up to the attacking dragon and orders it to leave before he gets hurt. The Real Roleplayer ... the intelligent cunning guy who tricks the constable into letting you all out of prison. The Loonie ... The guy who will do anything for a cheap laugh, including casting a fireball at ground zero. The Munchkin ... Need we say more? "

Ensuring a complete coverage of these general playing styles will increase the usability of the tools. Even though it is possible for players to create different characters and role-play in a different style, it still worth noting that sometimes personality may still overweight role-playing skills, so it is still important to find different players for each playing styles.

Player's Characters Since tabletop RPG normally allow people to either (1) Create a completely new character for current session, or (2) Use a pre-existed character who survived from previous sessions and gain specific skills or artifacts that can only be gained by surviving sessions, it is also vital to make sure mix and match on the whole evaluation. For example, characters in Call of Cthulhu will not be able to take skill points on the skill Cthulhu Mythos by creation. This specific skills will only grow when a character encounters mythos entities and the encounter caused insanity, so the character will need to survive such session and continue to be used in future sessions. Also, characters may obtain artifacts or spells, which may allow characters to have superhuman strength that has out-of-scope effect (e.g. teleporting) during the exploration.

Even for a newly created character, it is possible for the character to be created without holding any of the recommended skills for the specific session, The evaluation should also consider what will happen for all different possible occurrences including characters who come with recommended skills, those who have some but not much, and those who do not have recommended skills at all (mostly loonies).

Script Since there are all kind of different scripts in Dungeon of Dragon and Call of Cthulhu, It will be better to scatter out the variation of scripts used in the evaluation. This project will benefit more from scripts that are exploration-based, puzzle-based and escape-based.

Tasks During Exploration During the session, ask the player to think aloud. That is, ask the players to talk out whatever they think. This will help on collecting the user's thought on how they are making the decisions and how they receive their current situation during the exploration.

One thing to note is that sometimes think aloud may not apply well in the session, especially for people who role-play a lot. In that case, it is necessary to let the users know in advance and remind them during the session. It may also help if the session is happened online or through text-chat tool, so the user can think aloud without exposing their thought to other players.

Post-Session Evaluation Instead of quantified result, this tool will benefit more from open-ended evaluation that collect user's reaction and categorize based on the difference between user's expectation and game master's assumption. If we treat the game master's assumption as hypothesis and user's expectation as the result, we can use the hypothesis testing to categorize the outcomes being true negative, false negative, true positive, and false positive in table 1.

Even though each of the results worth considering, note that results that fall into for Type I and II errors (false positive and false negative) may be considered having more weight than other results, as resolving these issues can lead to smoother gameplay as it can decrease the difference between the expectation between player and the game master

Future Work

As each part will have a few part to be improved, we will use the corresponding section title mentioned in approach section

Front-end Web Interface As mentioned in limitation, for the tool to be practically usable in multiplayer session, a function to save generated result as a specific seed for sharing is essential, so that all player can use the 3D viewer to explore in the city for a more realistic experience. Furthermore, to solve the problem that user cannot know other user's location, implementing networking feature to allow

	Predicted +	Predicted -
Result +	True positive What player expected is the same as storyteller intended	False negative What storyteller intended is not expected by the player
Result -	False postivie What player expected is not intended by storyteller	True Negative What player did not expect is the same as storyteller intended

Table 1: Possible hypothesis testing errors

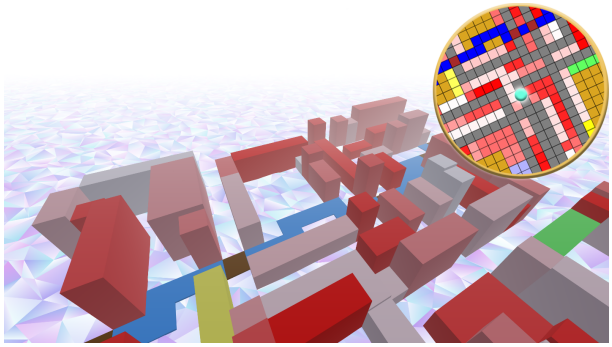

Figure 4: Prototype for minimap in 3D viewer

multiple player in the same map will be helpful to improve the playing experience.

Also, if we want to make the background building setting editable, we should either provide upload feature or edit tool for the internal dictionary that keeps the building characteristics.

Possible Buildings in the city Ideally, the possibility should also vary for different type of neighborhood (e.g. blacksmith may show up more in trading neighborhood, but may not show up much at a residential neighborhood).

To further extend flexibility, the global dictionary used for managing building style and parameters can be converted to json style and become available to the user to modify, with the feature mentioned in the frontend web interface part.

Building Generation The neighborhood center type should be one of the user parameter, and should also be available for the user in the front-end, so it will match the original attempt of allowing user to define the characteristic of the city.

Background information To increase the trustability of the result, we can combine multiple things with the city generation, such as terrain generation, historical background, etc. Dwarf Fortress is a perfect example the history is generated and the current state of the world are all based on the influence of history.

3D Visualization As 3D is the major visualization in this project, there are a lot of possible thing can be done in the 3D viewer. First, we can use the 2D map as mini-map in the 3D viewer, which will act like a radar and show the user where they located in the whole city, a prototype can be found in figure 4. Second, we can improve the recognizability of the different building by adding name tag, sign, or even use generative 3D model for buildings.

The control can also be improved by adding flying mode or quick viewpoint short cut (e.g. bird view, 45 degree view, etc.)

If we want to introduce the idea of time management, it can also be achieved by adding the light source as the sun light and change the shading during time of the day, as well as adding a timer into the viewer.

Summation

This project demonstrated a possible implementation for creating the city map generator tool by allow user input parameter and create a valid city map that can be explored intuitively. The project implementation can be found in https://github.com/yhong3/trpg-citymap

References

Handelman, D. E. 1987. A random walk problem. *Lecture Notes in Mathematics Positive Polynomials, Convex Integral Polytopes, and a Random Walk Problem* 1427.

He, L.; Ren, X.; Gao, Q.; Zhao, X.; Yao, B.; and Chao, Y. 2017. The connected-component labeling problem: A review of state-of-the-art algorithms. *Pattern Recognition* 70:2543.

kassoon. 2018. Town generator.

Kovalic, J., and Lowder, J. 2016. *The Munchkin book read the essays, (ab)use the rules, win the game*. Smart Pop.

StrangeLoop. 2018. Mapping imaginary cities.

watabou. 2018. Medieval fantasy city generator by watabou.

WizardawnEntertainment. 2006-2018. Fantasy settlements.