Agent-based terrain generation: comparing sequential vs concurrent approaches

Inês Oliveira
School of Technology
*IPCA**
Barcelos, Portugal
a17978@alunos.ipca.pt

Madalena Barros
School of Technology
*IPCA**
Barcelos, Portugal
a19180@alunos.ipca.pt

Alberto Simões
2Ai, School of Technology
*IPCA**
Barcelos, Portugal
asimoes@ipca.pt

Duarte Duque
2Ai, School of Technology
*IPCA**
Barcelos, Portugal
dduque@ipca.pt

***Abstract*—This paper explores the benefits of a concurrent implementation of an agent-based terrain generation tool, inspired by the work of Doran and Parberry [1]. In the present work, a Unity plug-in was developed for both the concurrent and sequential approaches. In the sequential approach, each agent is executed in turn, while in the concurrent version, a thread is created for each agent. The tests showed a significant performance increase in the concurrent approach.**

***Index Terms*—Concurrent, Terrain, Agents, Generation**

I. Introduction

For a better understanding of the structure of terraforming using agents, it was important to analyse two articles based on the same project. The first was written by Noor Shaker, Julian Togelius and Mark J. Nelson which was based on the project developed by Jonathon Doran and Ian Parberry, and it allowed for an understanding of the purpose of the agents themselves and the benefits of using them. The other, written by Jonathon Doran and Ian Parberry, showed and explained the development of the six types of agents that are needed for the generation of an environment [1, 2].

II. Terrain

This Unity plug-in generates a mesh from a heightmap that is created through a sequence of agent-based generation steps, using a list of at most 4 billion points. With the intent of giving more control to the designer on how the heightmap is created, the complete generation process is based on random values generated in relation to a given seed that is set to each agent. This makes it easier for the designer to keep a good result, while changing some parameters. During a sequential run, each agent runs on its own turn, while in parallel, a thread is created for each agent. Note that each generation step is still ran sequentially.

A. Coastline Agents

The only objective of the coastline agent is to create the silhouette of the island, which makes it essential, since the next agent depends on the points from the coastline to generate the landmass where all the other agents will work. To make the silhouette more organic, a algorithm for randomness was added, which is the main reason why these agents use the A* algorithm. After all of the agents finish their execution, the coastline points are reordered and stored to a temporary structure to be used by the next agents. In the concurrent implementation, a thread is created for each agent and a set of parameters is passed to them that includes the random seed and the maximum and minimum height to which it must move the points.

B. Flood Agents

These agents use the flood fill algorithm in order to fill the space on the grid. Afterwards, they are added to a queue as they move to the next point, always raising to a specific height and adding them to the landmass points that will be saved to the heightmap grid structure. Then, the agents are placed inside their respective area. The agent will raise the points to the height set by the designer and add those modified points to a list.

C. Mountain Agents

Mountain Agents are randomly spawned inside the landmass. Each agent moves to a random neighbour that belongs to the landmass and raises that point and all the neighbours that respect the same conditions. These points will be added to the mountains lists, so that it is possible to apply smooth and noise in an independent way. For the threads, at each step the agent does, it checks all the neighbours inside a loop. At each neighbour with an odd index in the loop it raises it to the maximum defined by the designer while the even ones are kept at the minimum. This technique allows a more irregular result to be obtained.

D. Hill Agents

Hill Agents are structured very similarly to the Mountain Agents, but their goal is to create hills. The designer will define the number of agents that he wants to use, where each one of them will be in charge of creating one hill, not necessarily meaning the end result will present the exact same number of hills as agents since they can end up merged, giving a more organic look. The base of preparing the threads is exactly the same as the Mountain Agents.

*IPCA - Polytechnic Institute of Cavado and Ave

This project was funded by Portuguese national funds (PIDDAC), through the FCT – Fundação para a Ciência e Tecnologia and FCT/MCTES under the scope of the project UIDB/05549/2020.

E. Beach Agents

The basic behavior of the Beach Agents is similar to the Hill and Mountain Agents. They depend on tokens, and they will move through the coastline, creating beaches along the way. The amount of height change decreases as new points are visited. This process guarantees that no points from the mountains are visited. Once they run out of tokens for the landmass walk, they return to the very next point on the coastline and restart the process until they run out of tokens again.

F. River Agents

These agents use the A* algorithm to find a path from a random point on the bottom of the mountains and a point belonging to the coastline. It stops once it finds the target coastline point or if it happens to find another river point on the way. At each step it will randomly decide if it moves to a random possible neighbour or to the one that ensures the shortest path to the target position. The A* pathfinding algorithm is used to avoid going through the mountains on its way to the coastline. The agent starts by choosing a random point that belongs to the bottom of the mountains as the initial position and an accessible coastline point as the final position.

G. Lake Agents

The number of Lake Agents picked by the designer determines the amount of lakes to be created. These agents start after the rivers are created and are randomly spawned on the landmass, avoiding proximity to rivers or mountains. After that, they start a random walk around that starting point, lowering the points, adding them to their own lake list, and then reducing walk token count. Once they run out of walk tokens, they return to the starting point, remove a restart token and restart the process until the amount of restart tokens reaches zero.

H. Smooth Agents

The Smooth Agents are the last ones running and are the ones that make all the modified points merge in a more organic way. They visit a list of points a certain amount of times, respecting their order. At each point, they calculate the height average of the current point and its neighbours, always giving more strength to the current one so that the smoothness does not completely change the terrain. When working sequentially, this agent is called right after each agent terminates its job. However, when the concurrent mode is activated, it is called after all the other agents from all the other types are terminated.

III. Tests

In order to facilitate performance testing of our agents, it was decided to use a checkbox, at the top of our plug-in, where it is possible to choose whether to test simultaneous or sequential mode. For this, a computer with an Intel® Core™ Processor i7-6700HQ CPU @ 2.60GHz and 16 GB of RAM was used. The evaluation was divided into two categories: the Sequential Mode and the Concurrent Mode, where it was written down the processing time of the execution of each agent individually and when all agents were working at the same time (Table I).

TABLE I
Agents Performance Tests

Agents	Sequential Time	Concurrent Time
Coastline (with smooth)	15 sec.	1 sec.
Flood (with coastline and smooth)	240 sec.	36 sec.
Mountain (with coastline, flood and smooth)	240 sec.	34 sec.
Hill (with coastline, flood and smooth)	240 sec.	44 sec.
Beach (with coastline, flood and smooth)	240 sec.	42 sec.
River (with coastline, flood, mountain and smooth)	240 sec.	60 sec.
Lake (with coastline, flood and smooth)	240 sec.	40 sec.
With every Agent	240 sec.	120 sec.

IV. Conclusions

Despite this project being based on the work of Doran and Parberry [2], we were able to achieve our goals, creating a tool that allows the designers to generate random terrain and still have fairly good control over the result. Coastline, Flood, Hills, Mountains, Beach, River, Lake and Smooth agents are the ones that we consider to be the core of a good terrain building tool. Agents-based terrain generation has definitely its perks and, from a user perspective, the most noticeable is making it possible to have a really good control of the results. We developed an option for concurrent mode prepared to be as fast as possible, making it very close to the speeds obtained by noise generation tools. Comparing the sequential and concurrent results, the generation achieved 4 minutes and 2 minutes, respectively, in a 512×512 heightmap. There is a lot of room to improve in terms of generation speed, namely by polishing the code and also searching for better parallel algorithms than the ones currently in use.

The project is open source and can be visited in this Dropbox link[1].

References

[1] Doran, J., & Parberry, I. (2010). Controlled procedural terrain generation using software agents. *IEEE Transactions on Computational Intelligence and AI in Games*, *2*(2), 111–119. https://doi.org/10.1109/tciaig.2010.2049020

[2] Shaker, N., Togelius, J., & Nelson, M. J. (2016). Fractals, noise and agents with applications to landscapes. *Procedural Content Generation in Games Computational Synthesis and Creative Systems*, 57–72. https://doi.org/10.1007/978-3-319-42716-4_4

[1]The source code can be downloaded from https://www.dropbox.com/s/vrrus17z6tvbvw6/AgentBasedTerraforming.unitypackage?dl=0