www.ijemr.net ISSN (ONLINE): 2250-0758, ISSN (PRINT): 2394-6962

Vandana Publications
IJEMR

Volume-5, Issue-2, April-2015
International Journal of Engineering and Management Research
Page Number: 541-543

Randomized Generation of Game Levels through Programming using Binary Space Partitioning

Shreevallabh Sunil Kulkarni
Sinhgad College of Engineering, Pune, INDIA

ABSTRACT
Video Games are fruitful applications of various branches of Science and Technology. Artificial Intelligence is one of most important branch of this category. Every year game developers are coming up with numerous new games. Day by day involvement of Artificial Intelligence in games is increasing to make game playing experience better. Artificial Intelligence can be considered as bridge between game world and programming, as various aspects of games interact with game world objects with use of Artificial Intelligence only. Building a game level is a part of Game Design process. Game Level Design is usually done with complete human involvement to design immersive game world. But inducing Artificial Intelligence in Game Level Design process can drastically decrease the time require to work on Game Design pipeline. This plummet will reduce the total application development time and will result in cost reduction of project. This paper will show process to use Binary Space Partitioning (BSP) to generate game levels which will require little to no attention of human. This process can also be used to generate game levels on the fly at run time. Either the intended algorithm can be induced in Game Logic to generate game world at run time or it can be used as hardcoded mechanism to produce levels while developing the game itself.

***Keywords*—** Binary Space Partitioning; Game Level Design; Game Optimization; Procedural Level Generation

I. INTRODUCTION

Game Development is undoubtedly one of most interesting branch of Software Development. Despite this fact, game development goes through many different phases. Game Development process can be considered as a glorious statue standing on three pillars viz. Game Art, Game Design, and Game Programming. Unless all these three units work hand in hand we cannot see a game becoming famous within intended audiences. Out of them Game Design brings remaining 2 phases together and creates a world where a player can interact. Being a salient process, it's important to give more emphasis on designing aspect.

II. BACKGROUND AND NECESSITY

Game Development team takes major efforts on designing game world. But having an algorithm to do this work can effectively reduce downtime for other work like programming and will give developers more time to spend on other aspects of game as well. So the proposed algorithm can dynamically generate game levels and can place in-game entities by itself and create a playable level in a matter of seconds. This process is also called as Procedural Level Generation [1]. Games developers have already tried to implement this in their games and were able to achieve good results too. Some games used this technique to place textures on brushes at run time while some games implemented it in a way that can increase the playable area if player tries to explore more than he/she intended to. A game named "RoboBlitz" used procedurally generated textures in order to reduce the file size of the game [2]. If used, such Procedural algorithms can change the way of developing games especially for specific genres, e.g. Endless Runners for Mobile Platform, Roguelike games.

Drunkard Walk Algorithm is considered as one of the simplest dungeon generation algorithm so far which works on grids [3]. This algorithm is randomization based and gives fairly good results without sacrificing much of system resources.

Drunkard Walk Algorithm:

```
1. X= Pick(Random_Point).
2. Mark x as Floor.
3. If(!Floor)then move(Random_direction(North,East,West,South)).
4. Repeat (2), (3) until room explored.
```

III. LEVEL GENERATION USING BSP (BINARY SPACE PARTITIONING)

A. What is Binary Space Partitioning?

Binary Space Partitioning is a method of dividing an area into smaller pieces. Basically you take an area called “leaf” and split it either vertically or horizontally. This process repeats on smaller areas over and over again until each area is having a desired size.

B. Significance of BSP in Level Generation?

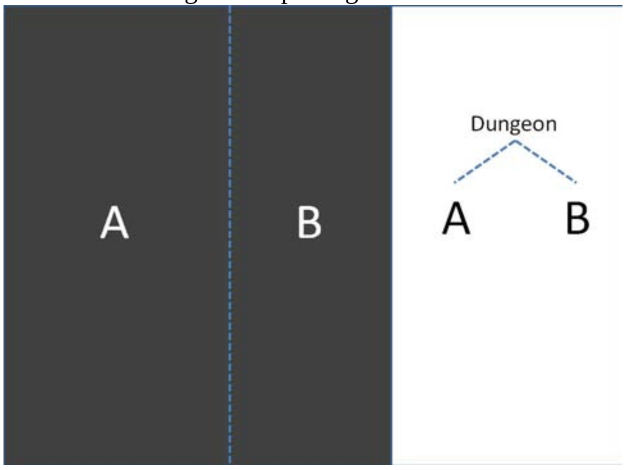
It is possible to write a simple logic to create randomly sized rectangles at random positions but this could result a map which is full of overlapping, clumped or strangle placed rooms. It also makes it a little more difficult to connect the rooms together and make sure there are no orphaned rooms. But BSP can guarantee more evenly spaced rooms while making sure the rooms can be connected to each other flawlessly [4].

IV. ALGORITHM

The algorithm starts with a rectangular dungeon with filled wall cells. The algorithm then split this dungeon recursively until each sub-dungeon has approximately the size of a room. The dungeon splitting goes through following operations [5]:

1. Choose Random Direction: Vertical or Horizontal.
2. Choose Random Position: X for Vertical, Y for Horizontal.
3. Split the dungeon into 2 sub-dungeons.

Figure 1: Splitting First Time.



Now as we have received 2 sub-dungeons, we can apply the algorithm again on each sub-dungeon individually. This will give us a logical structure of rooms in form of a tree.

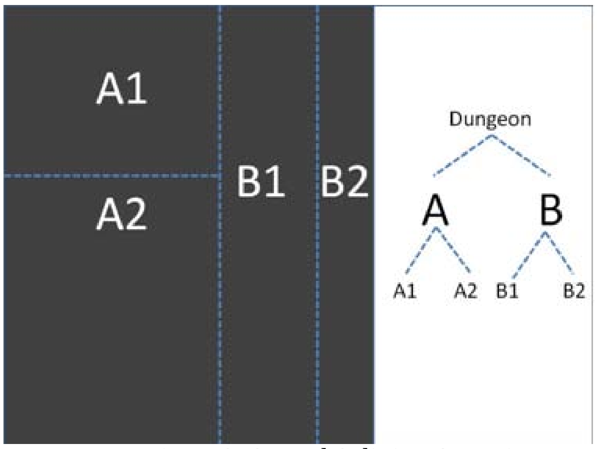


Figure 2: Second Splitting Operation

These iterations will result in various dungeons of varying size. The only work left now is to connect these dungeons together. This can be done by running a new algorithm to find proper direction for each dungeon to create doorways. In the end, we will receive a complete dungeon comprise of various sub-dungeons connected to each other.

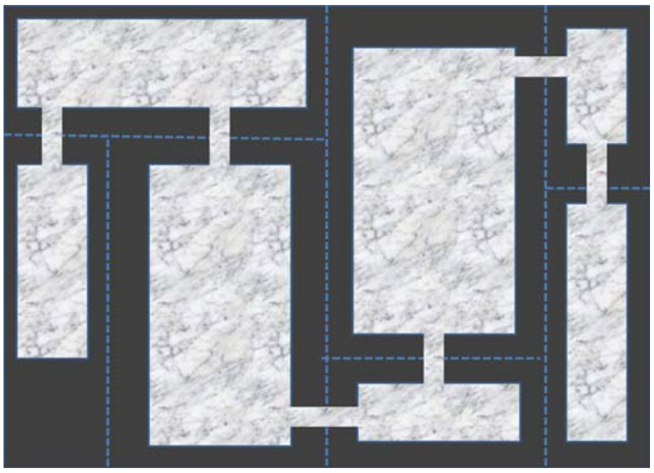


Figure 3: End Result, After Connecting All Sub-Dungeons Together

While connecting sub-dungeons we start from leaves, and then if design demands we connect non-leaf nodes too. This will add complexity in map design.

To make maximum use of this algorithm we can randomly place treasures, hidden rooms and in-game items into dungeon. A player start and Dungeon Exit entities can also be placed this way. A series of ambient light scattered all over dungeon can complete base setup need to test dungeon. If the application is strong enough, it can produce ready to play level or this procedure can be used to make a prototype of very large level.

To find neighborhood entities, simple coordinate level Neighborhood automata can be used. For two-dimensional automata, the two most common types of neighborhoods are Moore neighborhoods and Von Neumann neighborhoods [6].

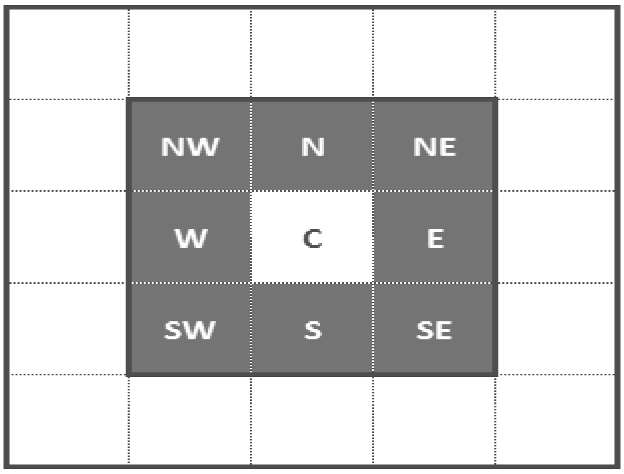


Figure 4: Moore Neighborhood

A Moore neighborhood is a square of size 1(unit size) consists of the eight cells surrounding "c", including those surrounding it diagonally. The surrounding 8 cells indicate names of directions.

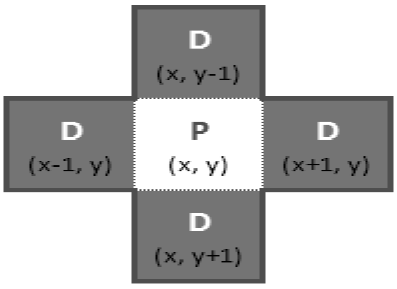


Figure 5: Von Neumann Neighborhood

Von Neumann Neighborhood is like cross centered on P: Above, Below, Left, Right. Von Neumann Neighborhood can primarily use coordinates to find nearby entities or brushes.

V. Future Scope

This system currently works well with regular shaped dungeons/rooms. However using irregular shaped dungeons/rooms will add more versatility and decrease monotone feel. This unevenness will always give user to something new to explore in the world.

VI. Concerns

It's not easy to make sure that world is always playable in terms of experience. Sometimes player may be unlucky to get an interesting game level. Poorly done procedural generation can make the world feel monotonous and boring. After player have seen dozen randomly generated mazes, they all starts to looks same, even they have difference in details [7].

VII. Conclusion

This algorithm will need different implementation strategies of various games. Depending on game genre and type of world the game demands this algorithm need to be enhanced. But if implemented, starting from Roguelike games to high end games, this technique can make dramatic changes in development phases of game design and can even change the way game loads behind the loading screen where player keeps starring and dreaming about the upcoming world he is going to live in [8].

References

[1] http://en.wikipedia.org/wiki/Procedural_generation
[2] http://en.wikipedia.org/wiki/RoboBlitz#Development
[3] http://pcg.wikidot.com/pcg-algorithm:drunkard-walk
[4] http://gamedevelopment.tutsplus.com/tutorials/how-to-use-bsp-trees-to-generate-game-maps--gamedev-12268
[5] http://www.roguebasin.com/index.php?title=Basic_BSP_Dungeon_generation
[6] http://www.raywenderlich.com/66062/procedural-level-generation-games-using-cellular-automaton-part-1
[7] http://gamedev.stackexchange.com/a/58326
[8] http://www.roguebasin.com/index.php?title=Main_Page