International Journal of Engineering and Technical Research (IJETR)
ISSN: 2321-0869, Volume-3, Issue-5, May 2015

Endless Runner using Procedural Content Generation & Real-Time Difficulty Curve Generation

Ninad Kulkarni, Paritosh Desai, Suraj Jaiswal, Sachin Chauthe, Yogini Bazaz

***Abstract*— Procedural Content Generation (PCG) is the branch of AI that deals with generating content algorithmically. It is used to reduce the cost of content creation while creating new types of content at a much greater speed with reduced effort. We aim to implement PCG by using Vasconcelos Genetic Algorithm (VGA) and the concept of Difficulty Curves. The obstacles patterns in the game will be generated procedurally at run time. Here, we provide the detailed characteristics of a difficulty curve and explain the specifics that go into making a balanced curve that keeps the game interesting and maintains a proper level of challenge for the player. Since the game focuses on endless content generation, the random or repetitive obstacle patterns would reduce the 'fun' factor of the game because user can get used to it and can also predict the content generated in such games. The game's content will be generated on the basis of a difficulty curve which will be adjusted depending on the progress of the user.**

***Index Terms*— procedural content generation; games; genetic algorithms; difficulty scaling; game design.**

I. INTRODUCTION

Gaming, as an industry is growing at an amazing rate and the expectations that players have from game content are rising with each released game title. To meet these demands game development studios incur rising costs in terms of payment to artists and programmers that supply that content along with the time required to develop such content. This gives rise to a unique application of AI algorithms, focusing more on the creative and artistic side of the game content rather than the strategic and tactical aspect of it; thus saving significant expense by producing desirable content algorithmically.

Procedural content generation (PCG) refers to creating game content automatically, through algorithmic means. In this paper, the term game content refers to all aspects of the game that affect gameplay other than non-player character (NPC) behaviour and the game engine itself [1].

Even established game companies can benefit from PCG, using it to generate 3D worlds, missions and other types of content. However, the first problem facing a game designer wanting to incorporate PCG techniques is the loss of control over the generated content. One of the main arguments against procedural content generation by the representatives of the gaming industry, at least when discussing online content generation, difficulty scaling and artificial intelligence adaptation, is the lack of reliability. Due to the manner in which most commercial games are designed, presenting content that is unplayable to the player is simply unacceptable.

Ninad Kulkarni, Paritosh Desai, Suraj Jaiswal, Sachin Chauthe, B.E. Student, Department of Information Technology, Atharva College Of Engineering, Mumbai.

Yogini Bazaz, Assistant Professor, Department of Information Technology, Atharva College Of Engineering, Mumbai.

II. PREVIOUS WORK

Continuing on our previous work [3], we have attempted to lay down the specific requirements of the curve generation process and the characteristics that define it. In this section author should discuss about related research has been done in the same domain or related domains with the name of the researcher and should be mentioned in the references. While in our previous paper we only outlined the general idea of scaling difficulty in an endless runner based on a difficulty curve that is generated in real time, in this paper we lay down the specifics and develop an algorithm to create a difficulty curve on the fly.

III. PROPOSED METHODOLOGY

A. Difficulty Curves & Procedural Content Generation

Though PCG overcomes challenges that occur in general game development, it has certain problems of its own; dynamic scaling of the difficulty level so as to match to the skill of the player. Also, the content must be correct and playable. Testing content for correctness and playability adds to the overhead of PCG content generation.

By using difficulty curves, some of these issues can be addressed. The curves allow the difficulty of a level to be tuned with gradual increments or decrements as per the wish of the designer.

We aim to extend the system of difficulty curves used by Diaz-Furlong [2] so that the curve generation process is automatic and continuous.

IV. LEVEL GENERATION PROCESS

In this section author need to describe experimental / simulation results with graphs and appropriate tables.

The difficulty curve is designed keeping the theory of flow in mind. The curve should find a balance between the player's skill level and a sense of challenge that's needed to engage the player. The curve is defined by specifying certain points on it. The curve needs to be virtually limitless since there is no concept of a fixed level length in the game. The min and max levels of difficulty on the curve are gradually increased on the curve as the player makes progress in the game.

In order to keep the level of curve based around the progress of the player, a "Base Point" is established as a central point to generate a curve for each patch. This Base Point is itself dependent on the number of patches that have

been produced already i.e. the "Patch Number" (an indicator of the player's progress).

Every curve is 2000 units long and corresponds to the length of a single patch. It is generated by forming a spline curve based on 21 Control Points (Generated by our algorithm) which occur at equidistant points with a gap of 100 each. The only exception being the very first point of the curve which will always be equal to the value of the Base Point.

Each curve has the following characteristics:-

a) Base Point: Patch Number*5 + Minimum Difficulty
b) Range of the normal points on the curve: [Base Point – 5, Base Point + 5]
c) Peak Points (Higher Difficulty): 1, 2 or 3. Selected at random from the array [1, 2, 2, 2, 3] for proper probability distribution.
d) Break Points (Lower Difficulty): 1, 2 or 3. Selected at random from the array [1, 2, 2, 3] for proper probability distribution.
e) High Peaks: Conversion of a Peak Point to a Higher Peak is based on probability (20% chance).
f) Low Breaks: Conversion of a Break Pont to Lower Break is based on probability (30% chance).
g) Ascending Curve: Series of points decreasing in difficulty. Either 3 points or 5, chosen at random. They are sequential.

Name	Lower Limit	Upper Limit	Probability
Normal Points	**-5**	**+5**	**-**
Peak Points	**+5**	**+15**	**1 (20%) 2 (60%) 3 (20%)**
Break Points	**-5**	**-15**	**1 (25%) 2 (50%) 2 (50%) 3 (25%)**
High Peaks	**+10**	**+30**	**20%**
Low Breaks	**-10**	**-30**	**30%**

Table 4.1 Probability Distribution of Characteristics

The Curve Generation Algorithm is as follows:

1. Declare array variables xCP, yP, curve, points;
2. Initialize value of the variable patchNum;
3. Declare constants minimumDifficulty ← 20, splineSize ← 2000, patchMultiplier ← 5, peakLower ← 5, peakupper ← 15, breakLower ← -15, breakUpper ← -5, highPeakProbablity ← 20, lowBreakProbability ← 30, generalLower ← -5, generalUpper ← 5;
4. Declare and initialize array variables peakPoints ← {1,2,2,2,3}, breakPoints ← {1,2,2,3}, ascCurve ← {3,5};
5. basePoint ←(patchNum * patchMultiplier) + minimumDifficulty;
6. numPoints ← randElement (peakPoints);
7. numBreakPoints ←randElement (breakpoints);
8. ascCurveLength ←randElement (ascCurve);
9. descCurveLength ← randElement (descCurve);
10. for i ←0 to yCP.Length:
    10.1 points[i] ← 0;
    10.2 yCP[i] ← UnityEngine.Random.Range(basePoint - generalLower, basePoint + generalUpper + 1);
11. yCP[0] ← basePoint;
12. while (true)
    12.1 p ← UnityEngine.Random.Range (1, points.Length);
    12.2 If (p+ascCurveLength)<(yCP.Length)
        12.2.1 low ← basePoint – generalLower;
        12.2.2 incr ← (generalUpper-generalLower)/ ascCurveLength;
        12.2.3 for i ← p to (p + ascCurveLength)
            12.2.3.1 yCP[i] ← UnityEngine.Random.Range(low,low+incr);
            12.2.3.2 points[i] ← 1;
            12.2.3.3 low ← low + incr;
13. for i ← 0 to numPeakPoints
    13.1 while (true)
        13.1.1 p ← UnityEngine.Random.Range (1, points.Length-1);
        13.1.2 if points[p] = 1
            13.1.2.1 continue;
        13.1.3 if chance(highPeakProbability) = true
            13.1.3.1 yCP[p] ←UnityEngine.Random.Range (basePoint + 2*peakLower, basePoint + 2*peakUpper)
        13.1.4 else
            13.1.4.1 yCP[p] ← UnityEngine.Random.Range (basePoint + peakLower, basePoint + breakUpper);
            13.1.4.2 break;
        13.1.5 points[p] ← 0;
14. for i ← 0 to numBreakPoints
    14.1 while (true)
        14.1.1 p ← UnityEngine.Random.Range (1, points.Length-1);
        14.1.2 if points[p] = 1
            14.1.2.1 continue;
        14.1.3 if chance(lowBreakProbability) = true

```
                14.1.3.1 yCP[p] ← UnityEngine.Random.Range (basePoint + 2*breakLower, basePoint + 2*breakUpper)
            14.1.4 else
                14.1.4.1 yCP[p] ← UnityEngine.Random.Range (basePoint + breakLower, basePoint + peakUpper);
                14.1.4.2 break;
            14.1.5 points[p] ← 0;
    15. for i ← 0 to xCP.Length
        15.1 xCP[i] ← (i)*splineSize/(xCP.Length-1);
    16. curve ←splineInterpolator.CubicSpline (xCP, yCP, splineSize);
    17. curvePlotter.setCurve (curve);
```

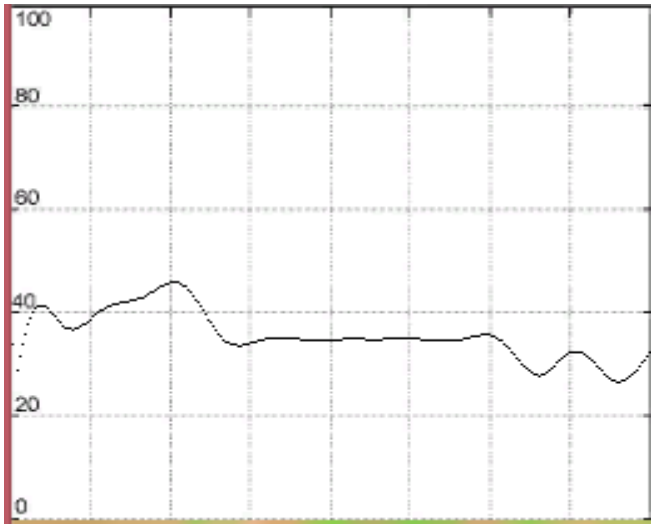

Fig 4.1 Sample Curve for Patch #1

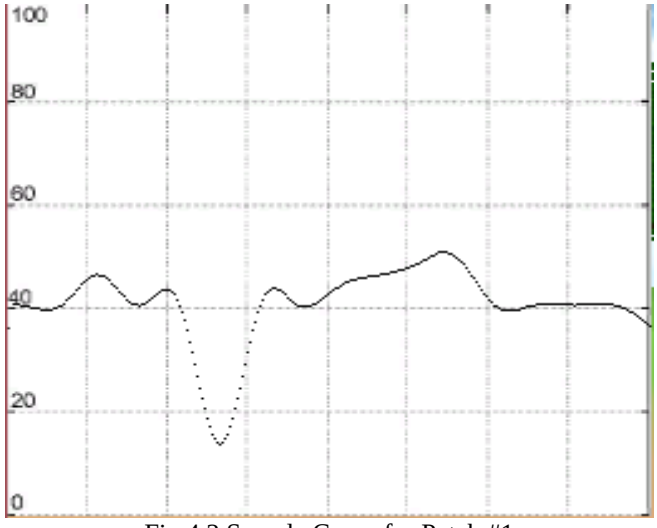

Fig 4.2 Sample Curve for Patch #1

V. CONCLUSION

One of the biggest obstacles in PCG, real-time efficiency can be managed by tweaking a combination of factors such as the level generation restrictions, efficiency of the algorithm that is used to generate the content and validate it and the level of abstraction at which the level is created. We believe that by optimizing these factors, it is possible to use online-PCG in various game scenarios including the one presented by us.

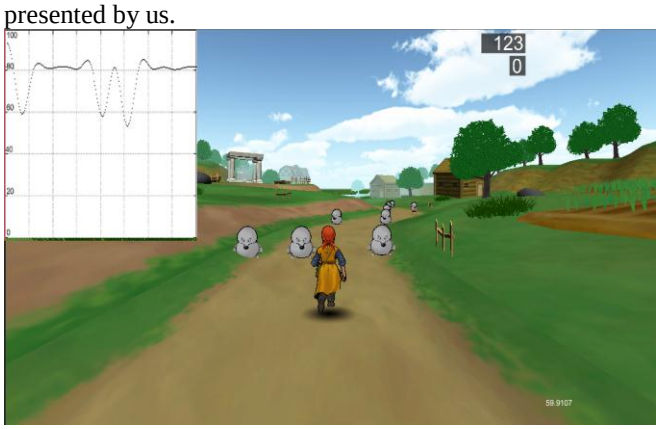

Fig 5.1 Game Screenshot - 1

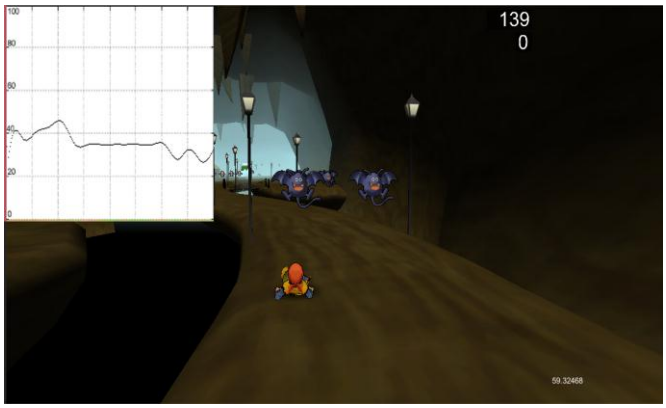

Fig 5.2 Sample Curve for Patch #1

VI. FUTURE SCOPE

In this case PCG (Procedural content generation) which is heart of our project plays a very vital role. By using this simple method can not only make game development faster and meaningful, but will also promote the individual developers, who can now be more focused on game's artwork, story etc.,and also make the more challenging and self-defining.

Most of the games available in the market use very generic technique for game implementation, this not only makes users lose interest in the game but also might feel like as if it's just re-skinned/rebooted from some older prequels of the corresponding game or other similar game. Our implementation which is basically used in an endless runner game is not just limited to endless runners only; PCG is widely used in many other genre of games like RPG, Rogue-like, Racing, Plat former, Dungeon Crawler, Shooters etc. The AI makes games environment more dynamic and also assure the users/player that situations now won't be same later on.

REFERENCES

[1] J. Togelius, G. N. Yannakakis, K. O. Stanley and C. Browne, "Search-based procedural generation: a taxonomy and survey" *IEEE Transactions on Computational Intelligence and AI in Games*, September 2011.

[2] Diaz-Furlong Hector Adrian, An Approach to Level Design Using Procedural Content Generation and Difficulty Curves, *Computational Intelligence and AI in Games, IEEE Transactions*.

[3] Desai et al., Review Paper on PCG and Difficulty Curves, *International Journal Of Computer Science and Information Technologies*, Volume - 6, Issue - 2: April- 2015.

[4] J. Togelius, G. N. Yannakakis, K. O. Stanley and C. Browne, "Search-based procedural generation: a taxonomy and survey" *IEEE Transactions on Computational Intelligence and AI in Games*, September 2011.

[5] M . Kerssemakers, J. Tuxen, J. Togelius and G. N. Yannakakis, "A procedural level generator», in *IEEE Conference on Computational Intelligence and Games*, 2012.

[6] J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modelling and track evolution," in *Proceedings of the SAB Workshop on Adaptive Approaches to Optimizing Player Satisfaction*, 2006.