Search-based Procedural Content Generation for Race Tracks in Video Games

Hafizh Adi Prasetya and Nur Ulfa Maulidevi

School of Electrical Engineering and Informatics, Institut Teknologi Bandung
hafizh.adip@gmail.com

Abstract: in recent years, Procedural Content Generation (PCG) has become a popular alternative for creating video games content. In this paper, we attempt to create a PCG for valid video game racing tracks while optimizing the fun they give to players. To achieve this, we represent a racing track as a sequence of parameterized segments, evaluate it using a multi objective fitness function, and then search for the optimum track by using search algorithms. Two search algorithms are used as alternatives to compare the effect it has on the racing tracks generated. The result we report show that the tracks generated can successfully match the validity criteria of being closed, continuous, and non-intersecting. More than that, they can also successfully create a certain degree of fun for players, according to data acquired from a questionnaire given to testers.

Keywords: Procedural content generation, racing games, race tracks.

1. Introduction

Procedural Content Generation (PCG) has become an increasingly popular alternative for content generation in recent years, especially in video games. As a method for generating content, the main focus for developing PCG has evolved from simple data compression to the more complex content optimization. Spurred also by the successes of games that makes use of PCG such as Mojang's Minecraft[1] and Gearbox's Borderlands[2], researches on PCG are also increasing in numbers and varieties. Examples of such researches are PCG for dungeons generation [1] and PCG for maze-like game levels [2], among many other.

One particular application for PCG is to generate race tracks. Race tracks are one of the more challenging and interesting content to generate and optimize due to its various limitations. Judging from the number of racing video games released (or about to be) in 2015 (Need for Speed[3], Project CARS[4], and many more), it's also still a highly relevant and useful content to generate. Several researches regarding race track creation has been done in the past [3] [4] [5], each with its own strengths and weaknesses. In this research, we try a slightly different approach in procedural race tracks generation in hope of generating race tracks that will produce an optimized fun experience when played inside a video game.

2. Search-Based Procedural Content Generation

In the study of PCG, there exists a prominent field of research named search-based PCG [6]. A search- based PCG is a PCG that typically (but not always) employs the methods of evolutionary computation to generate optimized content. We can loosely say that a search-based PCG is a PCG that searches for the best content among every other possible content. Currently, there have been quite a few researches under the area of search-based PCG that focuses on creating optimized contents [7] [8].

---

[1] http://minecraft.net (Mojang, 2009)

[2] http://borderlandsthegame.com (Gearbox Software, 2009)

[3] https://en.wikipedia.org/wiki/Need_for_Speed_(2015_video_game) (Electronic Arts, 2015)

[4] http://www.projectcarsgame.com (Slightly Mad Studios, 2015)

Received: September 22th, 2016. Accepted: Desember 4th, 2016

DOI: 10.15676/ijeei.2016.8.4.6

The term 'search-based PCG' is coined by Togelius in his paper about the taxonomy of PCG [6]. A search-based PCG refers to a special case of generate-and-test PCG. A generate-and-test PCG is a PCG that doesn't directly dishes out content it generates, but instead tests the content first using a test function. Depending on the test result, the PCG can either accept the content or reject it and create new contents. A search-based PCG is a test-and –generate PCG that satisfies two criteria.

Instead of simply accepting or rejecting content, the test function of a search-based PCG assigns a real value that measures the acceptability of the content. This value is often called fitness, and the function that produces it is called fitness function.

In the creation of new, better content, a search-based PCG uses the previous rejected content as the creation base. For example, the new content is a slightly modified old content.

The PCG developed in this paper is a test-and generate PCG that optimizes the fun experience of produced race tracks. The PCG is also in line with two criteria mentioned above and as such is included in the search-based PCG family.

Typically, a search-based PCG is solved by using a search algorithm. In order to use a search algorithm, the problem of content generation must first be modeled as a problem familiar for a search algorithm. One example of such model is CSP, a problem model that represent a search state as a configuration of a set of variable's values [9]. We will see how the approach proposed in this paper closely follows the convention of a CSP.

3. Generating Procedural Race Tracks

There are several different approaches that can be used to generate a race track procedurally. Before we explain our proposed method, first we take a look at the kinds of race track that we would want to generate. Also, we will take a look at approaches used in previous researches to see their merits and demerits.

In popular video games, there exist various types of race tracks depending on the characteristics of the video game. However, we will limit the kind of track that we can generate in this research. We are in particular interested in creating circuits, which are tracks which start and end at the same point. This allows the racers to race continuously for more than 1 time on the race track. Also, we would like our tracks to not intersect in the middle, since an intersection would allow the racers at the front to crash with the ones on the back.

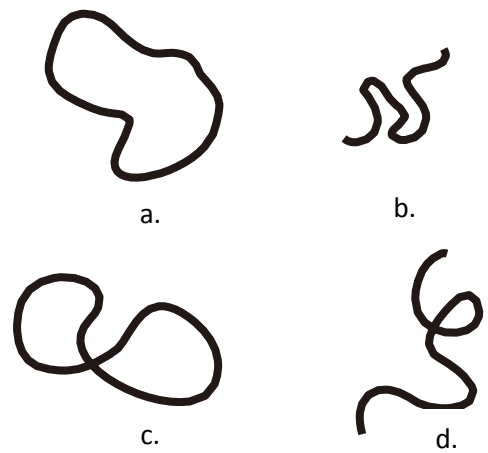

Figure 1. Examples of race tracks

Figure 1 shows different shapes of a track according to our criteria. The shape of the track that we're after is (A), a closed one that's free of intersections. We don't want (B) since it's not closed. We also don't want (C) since it has an intersection. Finally, (D) is an example of a track violating both of our criteria. After defining the tracks that we want, we will now take a look at the approach used to generate these tracks. We first take a look at past approaches and then describe the one we propose.

A. Previous Approaches

Previous researches for procedural track generation procedurally fell into two distinct categories in terms of the representation of the track. The first category is researches that represent a track as a set of control points, and the second one is researches that represent a track as a sequence of segments. We will briefly talk about both categories.

To the best of our knowledge, there are two researches that fall into the first category. One is the work of Togelius et al. [3] which in turn served as an inspiration for the second paper by Loiacono et al. [4]. These two previous works developed a race track generator that represents a race track as a set of control points. Then, from those control points, the shape of the race track can be produced by creating a polygon which enclosed every single control point. As a consequence, race tracks produced by this representation will always fulfill our criteria of enclosure and intersection.

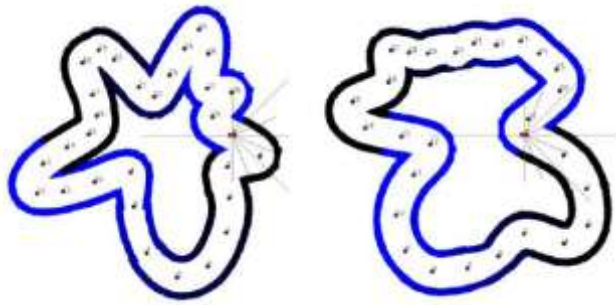

Figure 2. Examples of tracks generated by Togelius et al. [3]

The research by Togelius et al. [3] focuses on producing tracks that would fit a specific player profile, mainly in terms of difficulty. To achieve this, they first created a player profile by recording a race done by a human player. This player profile is then used to create a controller that could somewhat mimic the actions of the player. After that, the controller is finally used to evolve tracks that fit its profile. Figure 2 shows two examples of tracks produced by this research.

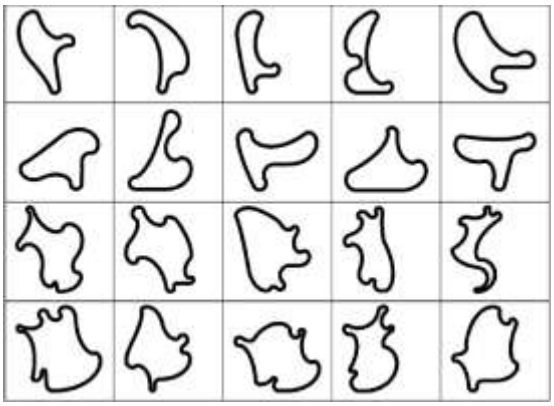

Figure 3. Examples of tracks produced by Loiacono et al. [4]

Meanwhile, the research by Loiacono et al. [4] focuses more on optimizing the fun value a race track could produce. This is done by maximizing the diversity of the tracks produced, under the idea that a track's diversity contributes greatly towards the fun it provides to the player. Loiacono extends the control point representation by including two additional parameters to each control point. The diversity of a track is then measured by taking into account the curvature and speed profile of the track. Figure 3 shows examples of tracks produced by Loiacono et al. The first two rows are tracks generated from 5 control points and the latter two rows are generated from 10 control points. As we can see, tracks generated from the same number of control points relatively possess the same number of curve.

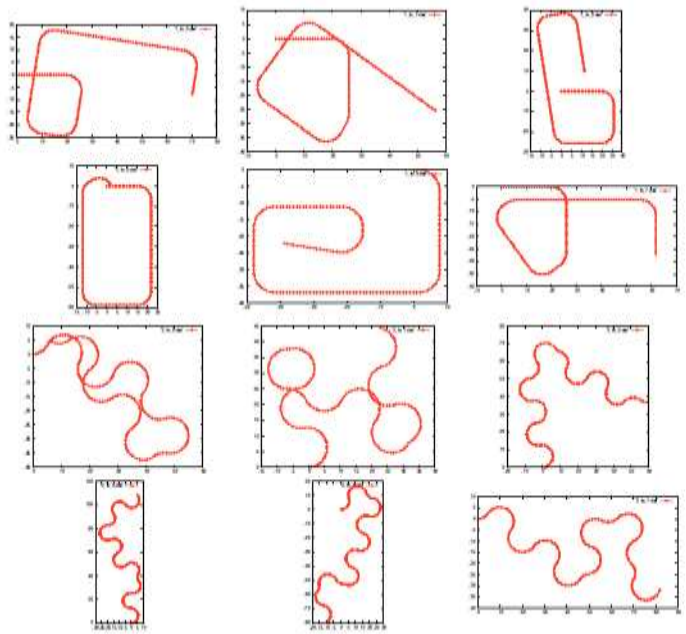

Figure 4. Examples of tracks generated by Wang and Missura [5]

The second category of representation has only been used in one research. Wang and Missura [5] writes a paper about the improvisation of race tracks using methods commonly found in the improvisation of music. This paper focuses more on creating new race tracks based on currently existing ones, and represent race tracks as a sequence of fixed segments. They define three fixed type segment that can exist in a race track each with the same length: a straight, a left turn of 9 degrees, and a right turn of 9 degrees. Then they model the improvisation of a race track as a discrete sequence prediction problem that is solved by combining two algorithms: a lossless compression method named LZ-MS and a string matching algorithm called factor oracle. However, as seen on figure 4, the tracks produced by the improvisation are not guaranteed to fulfill our criteria of enclosure and intersection.

B. Our Approach

We propose an approach inspired by each of the researches mentioned above. First of all, we take the focus of the research done by Loiacono et al. and aim to produce a PCG that can optimize the fun provided by the tracks produced. We also adapt the idea that diversity plays a critical role in determining a track's fun and will also try to measure a track's diversity from its curvature and speed profile. The first difference is the simplification of the fun evaluation function. This simplification is mainly done because of the lack of resources available to create profiles of existing tracks as a base for diversity evaluation. However, given the clear

connection already shown by Loiacono et al. between track diversity and fun, we believe that a simplified evaluation would suffice.

The second difference lies on our track representation. To further improve the diversity of the tracks produced, we propose a sequence-of-segments track representation. However, we add several parameters to each segment, enabling variety between segments of the same type. This variety would increase the number of possible tracks to generate. This representation is also expected to better optimize the fun value of the track since it does not limit the number of curve allowed in the track.

There are three major components in the our search-based PCG: the track representation that defines the information of a track and how to store it, the fitness function to assess the quality of a produced track, and a search algorithm to optimize the track generated in accordance to its fitness function. Each of the three components will be explained on the following sections.

4. Track Representation

We represent a race track as a sequence of segments. This is somewhat similar with previous research by Wang [5], but we improve the representation by adding parameters into the segments. By doing this, we allow segments of the same type to vary greatly and increase the diversity of the tracks produced. A segment of a track can either be a straight segment or a curved segment. Each type of segment requires different information and will be treated differently in the generation of race tracks. Figure 5 shows an example of a track constructed by 8 segments: S1 – S8. The red segments are curves and the black segments are straights.

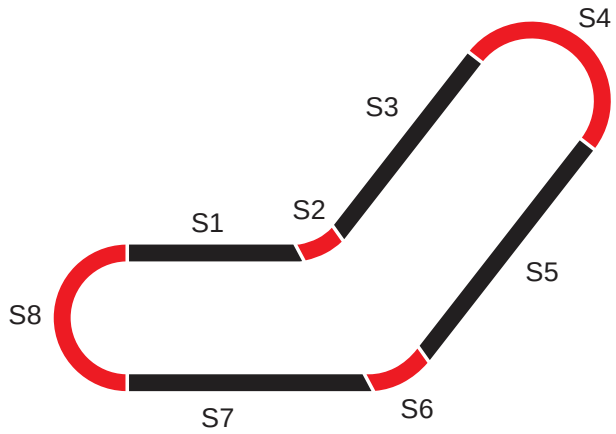

Figure 5. An example of racing track with 8 segments.

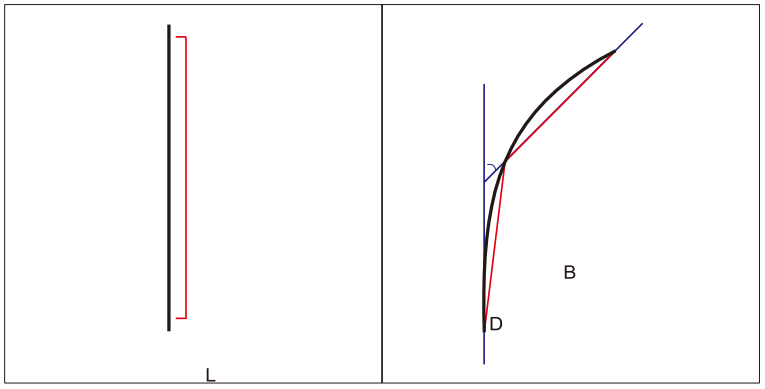

Figure 6. A straight segment, a curve segment, and their parameters

Figure 6 illustrates the difference between a straight segment and a curved segment. A straight segment only contains information of its type (T) and length (L), while a curved segment contains information on its type, the curve's arc (D), and its radiuses (A, B). We define a track Tn, as an array of segments (S1… Sn). Each segment is a 5-tuple containing the information of the segment. Formally:

$$Tn = (S1, S2 \ldots Sn - 1, Sn),$$
$$Sx = (T, L, D, A, B),$$
$$T \in \{0, 1\},$$
$$L \in \{20 \ldots 100\},$$
$$D \in \{-270 \ldots 270\},$$
$$A, B \in \{15 \ldots 50\}$$

Note that each value of the 5-tuple has a different domain. Also note that the segment representation fits in term of modeling the problem into a CSP. The variables (X) of the CSP are the 5 parameters of every segment in the track. The domain (D) of the CSP are the domain described above, and the constraint
(C) is defined implicitly by the fitness function used to evaluate the track.

5. Fitness Function

The fitness function of our search-based PCG returns the value of two major aspects of a track. First is the validity of the track; how close the track is to our initial requirement of enclosure and intersection. The second is the fun value of the track; the estimation of how much fun the track will produce when played by players.

A. Determining The Validity of Race Tracks

The first aim of this study is to create a PCG that's able to generate a valid track. We have already established intuitively that a valid track that we want is a circuit that doesn't intersect with itself. From that, we can define four criteria of validity:

1. The track is continuous from the start to the end point.
2. The start and end point of the track lies on the same coordinate.
3. The vehicle's direction on the start and the end point is the same.
4. The track doesn't intersect itself at any point.

A valid track is a track that fulfills each of these 4 criteria. To determine the validity of the track, the fitness function should have a component that can check the fulfillment of each criterion. The first criteria is guaranteed to be fulfilled given our track representation that doesn't allow any gaps between segments, but each of the other three criteria needs to be checked individually using different methods.

The end point of a track can be obtained by calculating the total displacement caused by the track's segments. Since there are only two types of segment, and a curved segment is represented by two straight lines, we can count the displacement caused by a segment by using the Pythagorean Theorem. Given the direction and coordinate at the start of the segment, we can calculate the displacement caused by the segment. Calculate this for every segments of the track and we can get the total displacement of the track. If we add the total displacement to the coordinate of the start point, we will get the coordinate of the end point. We assume that the track begins at coordinate (0, 0), so a valid track would also end at (0, 0).
The second criteria regarding the vehicle's direction is easier to check. Assume that the initial direction of a vehicle in the track is 0^{O}. Since we store the information of the arc of curved segments in the track, we can simply add the arc of every curved segment to get the final direction of the vehicle. If the sum is a multiple of 360, it means that the vehicle is facing the same direction at the end and the start of the track and the track is valid.

As for the last criteria, since we represent a track as a sequence of segments, we can check whether or not a track intersects itself by checking the intersection of every possible segment pair on the track. To check the intersection of two segments we make use of the concept of point orientation.

B. Evaluating the Fun Value of Race Tracks

There are a lot of factors that could contribute towards a fun experience in racing through a race track. This paper adapts the approach previously used in evaluating the fun of a track. Loiacono et al. [4] created a PCG for race tracks in which the fun of a track is largely determined its diversity. In turn, the diversity of a track itself is determined by two things: its curvature profile and its speed profile. In creating our PCG, we try measure fun using the same aspects, although with a simpler approach and formulation in producing the diversity value.

The first aspect is the curvature profile of the track. The curvature profile of a track refers to the amount, distribution, and diversity of curves in it. The more diverse the profile, it is assumed that the more fun it will gives to player. We define C(Tn), the evaluation function for the curvature of track Tn as follows:

$$C(Tn) = \sigma(TT(Tn)) + \sigma(TA(Tn)) + D(Tn) \quad (1)$$

The first part of the formula measures the proportion between the amount of curved segment on the track and the amount of straight segment. TT(Tn) is an array containing only the segment type of each segment in track Tn. The second part of the formula measures the diversity of the curve. TA(Tn) is an array containing only the arc of every curved segments in track Tn. Both are diversity values measured by using standard deviation of the respective arrays. The final part measures the distribution of the curved segment. The function D is the average of the difference between two segments that are next to each other.

The second aspect we evaluate is the speed profile of the track. The speed profile of a track is the diversity of speeds that can be achieved in a track. We acquire the speed data by calculating the speed of a vehicle at the end of each segment in the track, assuming optimal driving. The result is an array containing the velocity a car reach at the end of every segments in the track, assuming optimal driving, Ts. Therefore, the evaluation function for the speed profile of track Tn, S(Tn), is as follows:

$$S(Tn) = \sigma(TS(Tn)) \quad (2)$$

Where TS(Tn) is the aforementioned calculated array of speed. Note that these calculations would need the data of the vehicle such as the acceleration, grip, and top speed. Therefore, the required information is determined at the beginning of track generation and the track's speed profile will be optimized accordingly.

Finally, as we have more than one factor to consider, we combine both evaluation function into a single value by scalarizing them into a single evaluation function, F(Tn):

$$F(Tn) = \alpha \cdot C(Tn) + \beta \cdot S(Tn) \quad (3)$$

Where α and β are adjustable parameters that can be changed to emphasize the importance of one factor compared to the other. On the experiments that follow, both are set to 1 so that both factors are considered equally important by the algorithm.

6. Search Algorithm

The final component of our search-based PCG is the search algorithm used to optimize the track generated according to our fitness function. Since we model the problem of track generation as CSP, it is given that we solve the problem using search algorithms common to CSPs. Two classes of search algorithm is commonly used to solve CSPs, backtrack search

and local search. Here we choose two search algorithms from the local search class. The reason of choosing local search over backtrack is that backtrack optimizes its search by using partially assigned information on variables. In our problem, the fitness function cannot operate on partially assigned variables so using backtrack is not possible.

Two popular local search algorithms are selected as alternatives: tabu search [10] and genetic algorithm [11]. In this research we do not implement the search algorithm, instead we use search library since it is not the focus in this research. The tabu search are implemented using OpenTS[5] library, and the genetic algorithm utilized the Watchmaker Framework[6]. We provide two alternatives of search algorithms in order to observe the effect of search algorithm on the tracks generated by the search-based PCG.

The tabu search mechanism for one iteration is illustrated in Figure 7. Every Move consists of 2 components. First component is 2-tuple of segment ordering, and the second componen is 5-tuple of movement value for each segment.

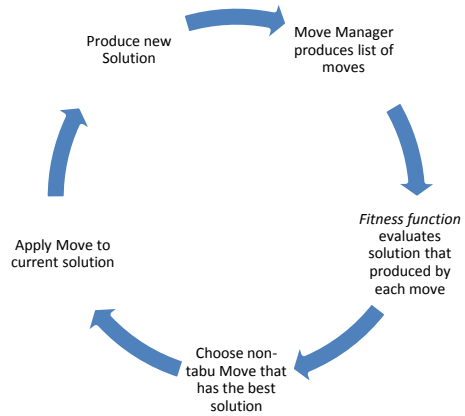

Figure 7. Tabu Search Cycle

Below is the example of 4-segment race track generation using tabu search for one iteration. Start with random initialisation, for example the current solution is: Tn = ((0, 100, 15, 10, 10), (1, 10, 180, 20, 20), (0, 100, 15, 10, 10), (1, 10, 170, 20, 20)). For the sake of simplicity, we only have to change the value of S in Ln[4] in the example into 180 to yield a valid solution.
Using the current solution, the algorithm produces every possible move, and calculate the fitness value when each move is implemented to find new solution. Algorithm will choose the best move, in this example is (4, (0, 0, 10, 0 0)), which adds 10 degree to 4th segment angle in order to have valid track.

Apply the chosen move, and current solutio is: Tn = ((0, 100, 15, 10, 10), (1, 10, 180, 20, 20), (0, 100, 15, 10, 10), (1, 10, 180, 20, 20)). Move that will undo Ln to previous state, which is (4, (0, 0, -10, 0 0)), is stored in the tabu list.
At this state, the algorithm is stop since the fitness value is satisfied the stop condition (valid track). When the stop condition is not met, repeat the second step until the stop condition is fulfilled.

For genetic algorithm, the mechanism after creating initial random population is illustrated in Figure 8. Mutation and crossover are the evolution operators that implemented in this paper. Mutation is changing value of one parameter in one track segment, while crossover is combining two solution by cutting each solution at certain location, and mix the cut from both solution.

5 http://www.coin-or.org/Ots/

6 http://watchmaker.uncommons.org/ (Dyer, Daniel W., 2006)

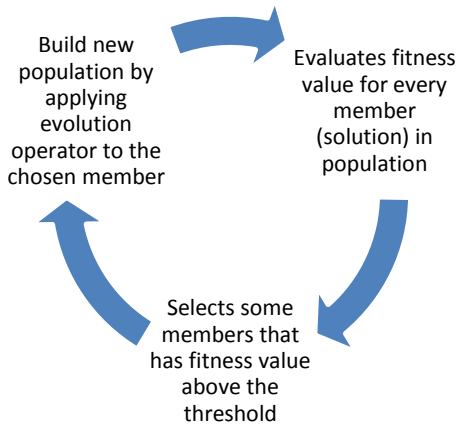

Figure 8. Genetic Algorithm Cycle

Below is the example of 4-segment race track generation using genetic algorithm for one iteration.

1. Initial population is generated randomly, for example we have 4 members (solutions) in initial population, as follows:
   a. T1 = ((0, 100, 15, 10, 10), (1, 10, 180, 20, 20), (0, 100, 15, 10, 10), (1, 10, 170, 20, 20)),
   b. T2 = ((0, 100, 15, 10, 10), (1, 10, 180, 20, 20), (0, 100, 15, 10, 10), (1, 10, 180, 20, 20)),
   c. T3 = ((0, 100, 15, 10, 10), (1, 10, 170, 20, 20), (0, 100, 15, 10, 10), (1, 10, 170, 20, 20)),
   d. T4 = ((0, 100, 15, 10, 10), (1, 10, 170, 20, 20), (0, 100, 15, 10, 10), (1, 10, 180, 20, 20)),
2. Fitness value for each member is calculated, and when there is a member that has fitness value conforms to the stop condition, the iteration is stop. In this example we assume the fitness value of 4 members do not conform to stop condition.
3. In this example the order of fitness value from highest to the lowest is T2, T3, T4, and T1. For example we choose T2 and T3.
4. New member (solution) is generated by applying the evolution operator to T2 and T3.
   a. Mutation: we have T5 and T6 as the result of T2 and T3 mutation
      T5 = ((0, 100, 15, 10, 10), (1, 10, 190, 20, 20), (0, 100, 15, 10, 10), (1, 10, 180, 20, 20))
      T6 = ((0, 100, 15, 10, 10), (1, 10, 170, 20, 20), (0, 100, 15, 10, 10), (1, 10, 160, 20, 20))
   b. Crossover: cut L2 and L3 at randomly location (for instance in position 1), and combine the result. We have L7 and L8 as the new members.
      T7 = ((0, 100, 15, 10, 10), (1, 10, 180, 20, 20), (0, 100, 15, 10, 10), (1, 10, 180, 20, 20))
      T8 = ((0, 100, 15, 10, 10), (1, 10, 170, 20, 20), (0, 100, 15, 10, 10), (1, 10, 170, 20, 20))
5. The new population now consists of T5, T6, T7, T8. The process is repeated until the stop condition is satisfied.

However, to prevent searches getting stuck on a particular state, we employ another strategy outside of the search algorithm. The strategy we employed is restart diversification, a common diversification strategy for searches [11]. Basically, when the search has reached a certain amount of iteration while the solution hasn't yet been found, this strategy restarts the search from the random initial point so that the algorithm wouldn't be stuck on particular conditions.

7. Result

Several tests are done in order to validate the solution proposed. First, the solution above is implemented as a java program that can generate the information of a track and also display the shape of the track simply. We need to guarantee that the implementation will produces a valid track 100% of the time. To make sure of this, we run the program 100 times and see the value of the validation function each run. The result of this test is that the program achieved a

mean validation value of 0, which means that that every track generated by this search-based PCG is enclosed and have no intersection. Figure 9 shows the examples of the tracks produced by the search-based PCG using 20 track segments.

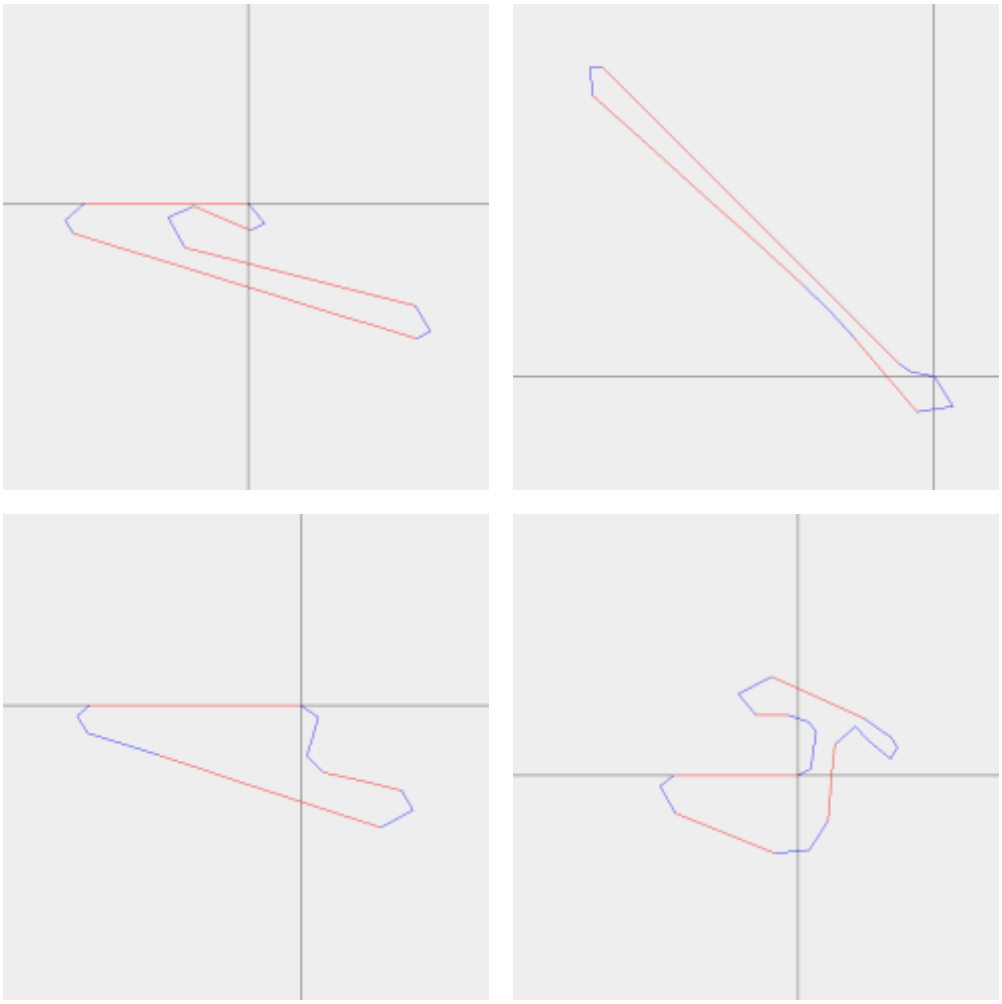

Figure 9. Examples of generated tracks

The second test aims to compare the performance of tabu search and genetic algorithm. After seeking the best configuration for both algorithms by using a one-factor-at-a-time experiment design, we arrive at the following configuration for the algorithms:

Genetic algorithm:

- Population = 15
- Elitism = 5%
- Number of Crossover Point = 1
- Mutation Probability = 0.5
- Maximum iteration before restarting search = 1200

Tabu search:

- Tabu list size = 10
- Maximum iteration before restarting search = 1200

Using the aforementioned configuration, we run both algorithms 100 times again for a 20-segment track and compare the result. The result for this experiment is can be seen on table 1.

Table 1. Comparison between Genetic Algorithm and Tabu Search

Search Algorithm	Average Generation Time	Evaluated Fun Value
Genetic Algorithm	7.863	103.63
Tabu Search	11.549	102.82

The evaluated fun value is computed using Formula (3), which consist of value of curvatory track and speed profile for the track. The detail of the formula is discussed in section V.B. For example, in curvatory track we have to find the TT(Tn) and TA(Tn), and we have Tn that consists of 5 segments as follows: T5=((0, 100, 15, 10, 10), (1, 10, 190, 20, 20), (0, 100, 15, 10, 10), (1, 10, 180, 20, 20), (0, 50, 22, 5, 5)). Then TT(T5) = (0, 1, 0, 1, 0) dan TA(T5) = (190, 180). The deviaton standard for TT is in the range 0 (track consists only one segment) until 0.5 (the number of straight segment and curve segment is equal). The range of TA is from 0 (every curved segment has the same angle) until 270 (the angle degree all of the curved segments is either -270 or 270, and the total number of segments is the same).

There are differences on the result using genetic algorithm and tabu search, most prominently on the average generation time. Genetic algorithm is faster than tabu search. The fun evaluation value also differs although it's not big enough to indicate a clear advantage in choosing a particular search algorithm. Another thing to consider is that the termination condition of both search algorithms can be set based on the fun value, so both algorithms are actually capable to guarantee a production with a predetermined fun value. In the experiment, the highest fun value achieved is around 130 for both algorithm. Therefore, the difference in generation time might be a more significant factor to consider in choosing the search algorithm.

Next, we need to determine if a score of around 100 in the fun value evaluation actually translates to actual fun when the track is played, so there needs to be another test. The last test involves testing the tracks produced by the search-based PCG for any fun value in playing it. To test this, we use TORCS, a widely used-and-accepted racing simulator that has been used in previous research [4]. We first create a track in TORCS with the exact same curves as the one generated by our search-based PCG. Next, we ask respondents to try out the tracks and rate the track in terms of fun felt while trying it.

Figure 10. Track a (left), b (center), and c (right)

We use three tracks for the last track as shown in figure 10. Track (a) is a manmade track available by default on TORCS to be used as a benchmark, and rest (track b and track c) are tracks generated by the search-based PCG with almost identical fun value, only differing in shape. This, of course, is outside the knowledge of the respondents. The scenario for the test is as follows: the respondent first plays two laps on one of the tracks available on TORCS to familiarize him/her with the control. Afterwards, he/she proceeds to play two laps on each test tracks. After he/she finished, the respondent then fills a questionnaire regarding the track. There were 33 respondents that participated in this questionnaire.

The data acquired from the questionnaires show a favorable result. Regarding the issue of the track's fun, on the scale of 0 – 10, 0 being completely not fun and 10 being completely fun, track c managed to get a score of 7.52, which indicates that that track has successfully creates a fun experience for the player. Meanwhile, track a and b both scores 6.52. This means both of our tracks can successfully measures up to manmade tracks in terms of fun. On the issue of the attractiveness of the track's shape, 72.8% of the respondents are willing to try out both track b and c based on looking at the shape alone. These percentages are lower than the control track which scores 78.7%, but nonetheless still leans towards the positive side.

8. Conclusion

In this paper we have explained an approach for generating race tracks procedurally and optimizing the fun value of that track. In our approach, we represent a race track as a sequence of parameterized segments. In the optimization process, we evaluate the fun of the track based on the track's curve and speed profile. We also made sure that the tracks adhere to our criteria of enclosure and intersection by employing several methods.

The results of the evaluation show that the search-based PCG developed is successful in creating a track that's enclosed and have no intersection 100% of the time. From the result of the questionnaire, we also see that the search-based PCG is also successful in creating tracks that provides a degree of fun for the people playing it, even compared to a manmade benchmark track. However, comparisons with other researches in the past are yet to be done.

It must be noted that the tracks generated by the search-based PCG still needs a lot of processing before they are ready to be played in actual racing games. The search-based PCG developed in this research is still limited on producing only the overall shape of the track. Further improvement and research on this area should focus on adding more features to the track such as track width, track elevation so that tracks that are ready to be used in video games can be automatically generated.

9. References

[1]. Van der Linden, Ruud, Roseli Lopes, and Rafael Bidarra. "Procedural generation of dungeons." *Computational Intelligence and AI in Games, IEEE Transactions* on 6.1 (2014): 78-89.

[2]. Ashlock, Daniel, Colin Lee, and Cameron McGuinness. "Search-based procedural generation of maze-like levels", *Computational Intelligence and AI in Games, IEEE Transactions* on 3.3 (2011): 260-273.

[3]. Togelius, Julian, Renzo De Nardi, and Simon M. Lucas. "Towards automatic personalised content creation for racing games", Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on IEEE, 2007.

[4]. Cardamone, Luigi, Daniele Loiacono, and Pier Luca Lanzi. "Interactive evolution for the procedural generation of tracks in a high-end racing game", *Proceedings of the 13th annual conference on Genetic and evolutionary computation. ACM*, 2011.

[5]. Wang, Jiao Jian, and Olana Missura. "Racing tracks improvisation", *Computational Intelligence and Games (CIG), 2014 IEEE Conference on IEEE*, 2014.

[6]. Togelius, Julian, et al. "Search-based procedural content generation: A taxonomy and survey", *Computational Intelligence and AI in Games, IEEE Transactions* on 3.3 (2011): 172-186.
[7]. Frade, Miguel, Francisco Fernandéz de Vega, and Carlos Cotta. "Breeding terrains with genetic terrain programming: the evolution of terrain generators", *International Journal of Computer Games Technology* 2009 (2009).
[8]. Togelius, Julian, et al. "Multiobjective exploration of the starcraft map space." *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on. IEEE*, 2010.
[9]. Russell, Stuart, and Peter Norvig. "*Artificial intelligence: a modern approach*", (1995).
[10]. Glover, Fred. "Tabu search-part I." *ORSA Journal on computing* 1.3 (1989): 190-206.
[11]. Mitchell, Melanie. An introduction to genetic algorithms. MIT press, 1998.

Hafizh Adi Prasetya was born in Okinawa, Japan, in 1994 and returned to Indonesia 1 year after. For 17 years he was raised in Bogor, West Java, finishing his primary and secondary education before entering Institut Teknologi Bandung (ITB) in 2011. He graduated with honor in 2015, earning his BS degree in computer science with a procedural content generation and video game-themed final project under the supervision of Dr. Nur Ulfa Maulidevi. After graduation, he enriched himself with various professional experiences ranging from developing Web APIs in a medium level multinational enterprise to freelancing. Currently, he is an awardee of the LPDP scholarship from Indonesia's Ministry of Finance, looking to pursue an MS degree in Artificial Intelligence sometime in 2017 or 2018.

Nur Ulfa Maulidevi graduated from Institut Teknologi Bandung (ITB), Bandung, Indonesia, in 1998, and received the BS degree in computer science. She joined Department of Informatics at ITB in 1998, immediately after graduation. In 2000, she was granted scholarship from Netherlands Education Center to pursue MS degree in University of Twente The Netherlands. After receiving the MS Desgree in 2002, she continued working at Department of Informatics. Two years later, she pursued PhD degree in ITB, and received grand from Faculty for The Future, a Schlumberger Foundation that encourage women to pursue higher education in science and engineering. In 2008 she received PhD degree from ITB. Dr. Maulidevi is currently an assistant professor in the School of Electrical Engineering & Informatics at ITB. Her research interests include machine learning, expert system, AI based game development, and procedural content generation.