Journal of Korea Game Society 2015 Aug; 15(4): 169-178
http://dx.doi.org/10.7583/JKGS.2015.15.4.169

게임 프로그래밍

자동차 주행 게임에서의 난이도 설정을 위한 도전 배치 방법

김상철, 박도생
한국외국어대학교 컴퓨터 및 전자시스템 공학부
kimsa@hufs.ac.kr, stimulant@nate.com

A Method for Challenge Placement to Set the Level of Difficulty in a Car Driving Game

Sangchul Kim, Dosaeng Park
Div. of Computer Science and Electronic System Engineering,
Hankuk University of Foreign Studies

요 약

다양한 수준의 게임 난이도를 사용자에게 제공하는 것은 게임 개발 시 주요 고려 사항 중 하나이다. 본 논문에서는 1인용 자동차 주행 게임에서 주어진 난이도를 갖도록 주행 트랙에 도전들을 배치하는 방법을 제안한다. 여기서 도전은 자동차 주행을 방해하는 장애물을 말하고, 게임 난이도는 트랙 한 바퀴를 도는데 필요한 예상 주행 시간으로 나타낼 수 있다. 제안된 방법에서는 도전 배치 문제를 IP(Integer Programming) 문제로 모델링한 후, LP 완화 및 시뮬레이티드 어닐링 방법으로 해를 구한다. 실험 결과, 주어진 목표 시간에 맞는 주행 시간을 갖는 도전 배치를 구할 수 있었다. 이들 도전 배치를 트랙에 적용한 후 시험 주행해 봄으로써, 실제 주행 시간은 평균적으로 해당 도전 배치의 목표 시간과 일치함을 보였다. 제안된 방법은 사용자에게 다양한 난이도의 게임 플레이를 제공함으로써, 게임의 흥미와 몰입감을 높일 것이다.

ABSTRACT

Providing various levels of difficulty of game play is one of important considerations in game development. In this paper, we propose a method for obtaining the challenges that will be placed on the track of an one-player car driving game. Herein challenges denote obstacles on the track, and the level of difficulty is represented by an estimated time needed for driving one lap of the track. In the proposed method, the problem for finding challenge placement is modeled as an IP(Integer Programming) one, and then LP relaxation and Simultaneous Annealing are employed to find a solution. To the experiment with the proposed method, we can obtain challenge placements to approximately meet given target driving times. Also, after practically driving on the track where those obtained challenges are being placed, it is seen that the average driving times approximate the target driving times of those challenge placements. Our method can allow game play with various levels of difficulty so that the users' interest and the level of immerse are expected to be raised.

Keywords : Level of Difficulty(난이도), Car Driving Game(자동차 주행 게임), IP, SA

Received: 7. 10, 2015 Accepted: 8. 17. 2015.
Corresponding Author: Sangchul Kim(HUFS)
E-mail: kimsa@hufs.ac.kr
ISSN: 1598-4540/eISSN: 2287-8211
This work was supported by Hankuk University of Foreign Studies Research Fund 0f 2014.

1. 서 론

사용자의 게임 플레이 수준에 맞게 난이도를 설정하는 것은 사용자의 즐거움에 큰 영향을 미친다. 사용자는 자신의 능력에 비해 게임 난이도가 너무 낮으면 지루한 감을, 너무 높으면 좌절감을 느껴 게임에 대한 흥미를 점차 잃게 될 것이다. 따라서 한 게임에서 여러 난이도(Level of Difficulty)를 가지는 레벨을 제공해 사용자가 선택적으로 플레이하게 하거나 또는 사용자의 수준을 자동으로 판단하여 게임 난이도를 동적으로 적절하게 조절하는 것이 필요하다[1,2,3,4,5,6].

자동 난이도 조절에 관한 기존 연구는 대부분 사용자 수준에 맞게 점진적으로 게임 난이도를 조정하는 방법에 대한 것이다. 즉, 사용자의 게임 플레이 결과에 따라, 적 캐릭터의 능력치를 조절하거나 또는 좀 더 쉽거나 어려운 도전(challenge)을 제공하는 것이다. 하지만, 주어진 난이도가 있을 때, 이를 제공하도록 게임을 어떻게 세팅할 지에 대한 연구는 상대적으로 빈약하다.

본 논문은 자동차 주행 게임에서 주어진 게임 난이도를 제공하는 도전 배치를 구하는 방법을 제안한다. 도전 배치란 트랙 상에 배치할 도전의 타입, 수 및 위치를 정하는 것이다. 게임 난이도는 트랙 한 바퀴를 도는데 걸리는 주행 시간으로 표현된다. 도전은 트랙 상에 배치할 장애물로서, 이들의 모양, 크기, 역할에 따라 어려운 정도가 다를 것이다. 각 도전은 그 어려운 정도는 자동차가 해당 도전을 통과하는데 걸리는 시간으로 정하면 될 것이다. 자동차 주행 게임을 대상으로 한 이유는 성별 및 나이에 상관없이 많은 사람이 즐기는 게임 장르이기 때문이다.

본 논문에서 제안된 방법에서는 주어진 게임 난이도를 만족하는 도전 배치를 구하는 업무를 IP 문제로 모델링하고, 그 문제의 해를 LP 완화(Linear Programming Relaxation) 및 SA(Simulated Annealing) 알고리즘으로 구한다. SA는 GA(Genetic Algorithm)와 함께 NP(Non Polynomial) 문제 등의 폴리노미얼 타임(Polynomial Time) 에 해를 구하지 못하는 문제 풀이에 적용되고 있다. GA와 SA는 성능 면에서 비슷하다고 보고되는데, 우리는 알고리즘 측면에서 단순한 SA를 선택했다.

제안된 방법을 실험한 결과, 목표 주행시간으로 표현된 난이도에 근접한 주행 시간을 갖는 도전 배치를 구할 수 있었다. 이들 도전 배치를 트랙에 실제 적용한 후 시험 주행해 봄으로써, 자동으로 설정된 게임 난이도와 실험자가 보여주는 실제 주행 결과는 일치하는 경향을 보였다. 제안된 난이도 자동 설정 방법은 난이도를 수동으로 결정하는 수작업의 번거로움을 크게 줄일 것이다. 또한, 게임 플레이 중에 사용자의 주행 실력에 맞추어 게임 난이도를 동적으로 자동 조절하는데 활용될 수 있을 것이다. 본 논문은 우리의 사전 연구 [7]을 확장한 것이다.

최근 게임 난이도를 자동으로 설정하는 연구가 활발히 진행되어 왔다. 초기의 연구는 주로 게임내 AI 에이전트의 행위 능력을 조절하는데 초점이 맞추어져 있었다[2,3]. [3]에서는 사용자의 게임 플레이 결과에 따라 AI 에이전트의 행위 룰을 적절한 수준으로 선택하는 동적 스크립팅(Dynamic Scripting) 방법을 제안했다. 이 방법은 자동차 주행 게임같이 장애물이 정해진 지역 내에서 고정되거나 또는 정해진 패턴으로 움직이는 경우에는 맞지 않다.

FPS 나 RPG 게임은 속성상 오랜 시간 플레이가 이어진다. 이런 게임을 대상으로 사용자의 플레이 수준에 맞게 동적으로 게임 난이도를 조절해서 사용자가 게임에 흥미를 유지하도록 하는 방법들이 연구되어 왔다[4,5]. 이들 연구에서는 사용자 게임 플레이를 계속 모니터링하면서, 사용자의 현 수준에 맞추어 게임 세팅 (예를 들면, 적의 수나 능력, 출현 시기 등)을 적절히 조절해 나간다.

주어진 게임 난이도를 만족하는 게임 세팅을 정하는 연구는 플랫폼 게임(platform game)을 대상으로 활발히 진행되었다[6]. [7]에서는 GA를 이

용해 원하는 난이도를 갖도록 플랫폼 간의 간격, 방해물 출현 유무 등을 결정한다. [2]에서는 도전의 수준을 해당 도전을 통과하는데 필요한 목숨의 수로 간주하고, GA을 이용해 주어진 난이도에 맞는 게임 세팅을 구한다.

본 논문과 관련된 분야로 PCG(Procedural Content Generation)와 사용자 숙련도를 측정하는 모델에 관한 연구가 있다. 기존 PCG 연구에서는, 게임 플레이에서 예상되는 사용자의 입력 패턴을 정의하거나[8] 또는 미리 정해진 게임 요소들을 조립함으로써[9] 다양한 레벨들을 생성하는 방법들이 발표되었다. 하지만 이들 방법들에는 지정된 게임 난이도에 맞도록 레벨을 구성하는 것에 대한 고려가 충분하지 않다. 또한 게임 월드의 구성과 사용자의 게임 플레이 경험들을 토대로 퍼셉트론(perception)[10] 또는 지도 학습(supervised learning) [11]을 이용해서 사용자의 플레이 수준을 파악하는 연구도 발표되었다.

2. 문제 정의

우리는 정해진 트랙 위를 달리는 게임을 위한 도전 배치 방법을 고려한다. 물론 본 논문에서 제안된 방법은 한정된 트랙이 아닌 개방된 필드를 달리는 주행 게임에도 응용될 수 있을 것이다. 게임에서의 도전과 난이도는 게임의 종류별로 적절히 정의될 것이다.

1인용 자동차 주행 게임에서 도전은 트랙 상에 배치되는 장애물로 정의한다. 차량이 장애물에 부딪치거나 피하다 트랙을 벗어나면, 장애물과 만나기 직전 지점으로 돌아 간 후, 차량을 주행을 계속한다. 각 도전별 난이도는 해당 장애물을 통과하는데 걸리는 시간으로 정의한다.

전체 트랙은 Z개의 존으로 구성되고 이들 존들의 길이는 동일하다고 가정한다. 존별 특성에 따라 배치하는 장애물 종류를 달리 할 수 있을 것이다. 또한 같은 역할의 장애물이라도 배치하는 존의 특성에 따라 통과 시간이 다를 수 있기 때문이다, 예를 들면, 직선 존과 곡선 존으로 구분하고, 존의 성격에 따라 배치하는 장애물 종류를 달리함으로써 게임의 흥미를 높일 수 있을 것이다.

N개의 도전 타입들이 있는데, 이들은 $c_1, c_2, ..., c_N$으로 부른다. 트랙에는 동일한 도전 타입의 도전 인스탄스(줄여서 도전) 여러 개가 배치될 수 있다. 성질이 비슷한 도전 타입들을 같은 범주로 묶을 수 있는데, 총 G개의 범주가 존재한다. 예를 들면, '갑자기 트랙을 횡단하는 사람 한명'이라는 도전 타입 과 '갑자기 트랙을 횡단하는 강아지 한 마리'라는 도전 타입은 같은 범주에 속할 것이다. 각 도전 타입별 어려운 정도는 해당 도전 타입을 통과하는데 걸리는 평균 시간으로 표현한다.

문제 정의: Z개 존으로 구성된 트랙에 대해서, 목표 주행 시간 T가 주어지면, 제약 조건 *C0-C4* 들을 만족하도록 이들 존들에 배치할 도전들의 타입, 개수 및 위치를 구한다.

C0) 트랙 한 바퀴(즉, Z개 존)를 주행하는데 T 시간이 걸린다.

C1) 각 도전 타입 c_i $(1 \le i \le N)$에 대해, c_i 타입에 속하는 도전의 수는 하한값 MIN_i과 상한값 MAX_i 사이에 있다.

C2) 각 z $(1 \le z \le Z)$ 번째 존에 대해, 해당 존에 배치되는 도전의 수는 하한값 MIN_z^{zone}과 상한값 MAX_z^{zone} 사이에 있다.

C3) 각 범주 g $(1 \le g \le G)$ 에 대해, g 범주에 속하는 도전의 개수는 하한값 MIN_g^{cat}과 상한값 MAX_g^{cat} 사이에 있다.

C4) 도전 타입의 성격상, 특정 존에는 배치할 수 없는 경우도 있다

[Fig. 1] Problem Definition

우리는 자동차 주행 게임의 난이도 설정을 위한 도전 배치 문제를 [Fig. 1]처럼 정의한다. 조건 *C0*에서 목표 주행 시간 T는 게임 난이도를 결정한

다. 예를 들면, T를 감소시키면 그만큼 사용자는 T 시간 내 트랙 한 바퀴를 주행할 가능성이 높아지므로, 게임의 난이도가 낮아진다고 할 수 있다. 조건 *C1, C2* 및 *C3*의 주된 역할은 도전들이 타입별, 존별 및 범주별로 골고루 배치되도록 하는 것이다. 조건 *C4*는 존의 성격에 맞추어 설계된 도전 타입이 있다는 점을 반영한 것이다. 예를 들면, 곡선 구간을 위한 장애물을 해당되는 존에만 배치하기 위함이다.

Procedure *Construct_Level_Of_Difficulty*
Input: T, Z, Z'
Output: R = $<x_{1,1}, \ldots x_{1,N}, \ldots x_{Z+Z',1}, \ldots x_{Z+Z',N}>$

Step 1) Initialize $x_{z,i}$ = 0, $1 \le z \le Z+Z'$ and $1 \le i \le N$.

Step 2)
Find R_1 = $<x_{1,1}, \ldots x_{1,N}, \ldots x_{Z,1}, \ldots x_{Z,N}>$ by calling *Solve_IP_Problem(T, 1, Z, 1);*

Step 3)
$T' = T/Z$;

Step 4)
for($z = Z+1$; $z \le Z+Z'$; z++) {
Find R_2^z = $<x_{z,1}, \ldots x_{z,N},>$ by calling *Solve_IP_Problem(T', z, z, Z);*
}

Step 5) Construct R by combining R_1 and R_2^z's.

[Fig. 2] A Procedure for Finding the Challenge Placement

3. 난이도 설정을 위한 도전 배치 방법

본 장에서는 [Fig. 1]에서 정의한 문제의 해를 구하는 방법을 기술한다.

3.1 도전 배치 알고리즘의 전반

도전 배치를 구하는 과정은 [Fig. 2]와 [Fig. 3]에 기술되어 있다. xz,i는 z번째 존에 배치된 ci 타입의 도전들의 수를 나타난다. T(ci)는 게임 플레이하는 동안에 ci 도전 타입을 통과하는데 걸리는 시간이다. S(g)는 트랙에 배치된 도전 타입들 중 범주 g에 속하는 것들의 집합이다. 도전 c에 대해 I(c)는 c의 타입을 나타낸다.

[Fig. 2]의 Step 2는 트랙 한 바퀴를 구성하는 Z개의 존에 대해 도전 배치를 서브루틴 Solve_IP_Problem을 이용해서 구하는 단계이다. T에 최대한 근접한 전체 주행 시간을 가지도록 첫 번째 존부터 Z번째 존까지의 도전 배치를 구해, 그 결과를 $x_{z,i}$들 ($1 \le z \le Z$, $1 \le i \le N$)에 저장한다.

Step 2에서 구한 도전 배치가 T 시간동안 평균적으로 Z개의 존을 주행할 수 있도록 설정되었더라고, 실제 게임 플레이에서 사용자는 더 멀리 주행할 수 있다. 따라서 뒤따른 Z'개의 존에 대해서도 도전 배치를 구할 필요가 있다. Z'는 게임 설계자의 의도에 따라 적절한 값으로 결정하면 될 것이다. 만약 실제 게임에서 사용자가 ($Z + Z'$)개 존도 초과해 주행하면, 이들 초과 존들에 대해서는 Z'개 존의 도전 배치를 반복해 사용하면 된다.

[Fig. 2]의 Step 3과 4에서는 Z+1 번째 존부터 Z+Z' 번째 존까지에 대한 도전 배치를 구한다. 각 존에 대해, 목표 주행 시간 T'를 T/Z로 설정한 후 해당 존에 배치할 도전들을 서브루틴 Solve_IP_Problem을 이용해서 구한다.

서브루틴 Solve_IP_Problem는 [Fig. 3]에 기술되는데, S번째 존에서 E번째 존까지의 도전 배치를 구하는 기능을 수행한다. [Fig. 2]의 Step 2에서 호출된 Solve_IP_Problem의 실행은 다음과 같이 진행된다. $m_i = MIN_i$, $M_i = MAX_i$, $m_g^{cat} = MIN_g^{cat}$, $M_g^{cat} = MAX_g^{cat}$, $m_z^{zone} = MAX_g^{cat}$, $M_z^{zone} = MAX_z^{zone}$로 정해진다. Step 2에서는 [Fig. 1]에서 정의된 도전 배치 문제를 IP(Integer Programming) 문제로 표현하고, 그 해를 구한다. 그리고 [Fig. 2]의 Step 4에서 호출된 Solve_IP_Problem의 실행에서는 $m_i = MIN_i/W$,

$M_i = MAX_i/W$, $m_g^{cat} = MIN_g^{cat}/W$, $M_g^{cat} = MAX_g^{cat}/W$, $m_z^{zone} = \sum_{z=1}^{Z} MIN_z^{zone}/W$, $M_z^{zone} = \sum_{z=1}^{Z} MAX_z^{zone}/W$로 정해진다.

Subroutine *Solve_IP_Problem($\mathcal{L}$, S, E, W)*
Output: $<x_{S,1}, \ldots x_{S,N}, \ldots x_{E,1}, \ldots x_{E,N}>$
Step 1)
Set m_i, M_i, m_g^{cat}, M_g^{cat}, m_z^{zone}, and M_z^{zone} to proper values.

Step 2) Find s = $<x_{S,1}, \ldots x_{S,N}, \ldots x_{E,1}, \ldots x_{E,N}>$ by solving the following IP problem:
Minimize $F(s) = |\mathcal{L} - (\sum_{z=S}^{E}\sum_{i=1}^{N} x_{z,i}T(c_i) + T_{track})|$
subject to

(*CO0*) For each $x_{z,i}$, $S \le z \le E$ and $1 \le i \le N$,
$x_{z,i} \ge 0$, $x_{z,i}$ is an integer

(*CO1*) For each challenge type c_i, $1 \le i \le N$,
$$m_i \le \sum_{z=S}^{E} x_{z,i} \le M_i$$
(*CO2*) For each zone z, $S \le z \le E$
$$m_z^{zone} \le \sum_{i=1}^{N} x_{z,i} \le M_z^{zone}$$
(*CO3*) For each group g, $1 \le g \le G$,
$$m_g^{cat} \le \sum_{z=S}^{E}\sum_{c \in S(g)} x_{z,I(c)} \le M_g^{cat}$$
(*CO4*) For each $x_{z,i}$, $S \le z \le E$ and $1 \le i \le N$
$x_{z,i}$=0 if the properties of zone z don't match challenge type i
Step 3)
return $<x_{S,1}, \ldots x_{S,N}, \ldots x_{E,1}, \ldots x_{E,N}>$;

[Fig. 3] Subroutine Solve_IP_Problem

자동차 주행 게임에서 이론상 게임 난이도는 배치된 도전들의 난이도 합으로 결정될 것이다. S=1이고 E=Z일 때, 아래 f(s)는 도전 배치 s에 따라 장애물들을 배치한 트랙을 한 바퀴 도는데 필요한 예상 시간이 된다.

$$f(s) = (\sum_{z=S}^{E}\sum_{i=1}^{N} x_{z,i}T(c_i) + T_{track})$$

T(ci)는 도전 타입 ci가 설치된 영역을 통과하는 데 걸리는 시간을 나타내는데, 해당 지역에 아무런 도전을 배치하지 않고 통과할 때 걸리는 시간은 차감한다. T_{track}은 트랙 상에 특별한 장애물을 배치하지 않았을 때의 총 주행시간을 나타낸다. 따라서 $F(s)$는 목표 주행 시간 $\mathcal{L}$과 도전 배치 s하에서 트랙 한 바퀴를 도는데 필요한 시간간의 차이를 나타낸다. Step 2에서 IP 문제의 목적 함수는 $F(s)$를 최소화 하는 것이다. CO1, CO2 및 CO3은 [Fig. 1]의 C1, C2 및 C3을 표현한다.

3.2 도전 배치 IP 문제의 해를 구하는 방법

IP 문제는 시간 복잡도면에서 NP-hard한 문제이기에, 앞에서 언급했듯이 해를 구하는 폴리노미얼 타임 알고리즘이 존재하지 않는다. 우리는 Fig 3의 Step 2에 기술된 IP 문제의 해를 구하기 위해 SA(Simulated Annealing)[12] 알고리즘을 사용한다. 본 연구에서 사용한 SA 알고리즘은 Fig 4에 기술된다.

Fig 4는 SA 알고리즘의 전형적인 형태를 가지고 있다[12]. 초기 해인 s_0는 IP 문제를 LP 문제로 완화시켜 그 해를 구한 후, 각 $x_{z,i}$ 값을 정수화한 것이다. 즉, LP 문제로 완화하기 위해서는 조건 CO0에서 「$x_{z,i}$ is an integer」를 제거하면 된다. Step 2-1에서는 최신 해인 s의 이웃 해들 중 한 개를 무작위로 선택해 s_{new}를 생성한다. s는 벡터 형태로 표현되는데, 원소 두 개를 임의로 선택해 한 원소는 1만큼 증가시키고 다른 원소는 1만큼 감소시킨다. 예를 들어, s=<5, 4, 6, 3>이고 무작위로 세번째 및 네번째 원소를 선택하게 되면, 이들을 각각 7과 2로 바꾼 <5, 4, 7, 2>이 s_{new}가 된다.

[Fig 4]에 나타나는 에너지 함수 $F(\cdot)$는 [Fig. 2]에 기술된 IP 문제의 $F(\cdot)$와 같다. Step 2-3-2에서는 [13]에서 제안한 방법을 따라, s_{new}가 기존해 s보다 나쁘더라도 확률적으로 새로운 해로 수용한다. 초기 온도 $Temp_0$와 온도 쿨링 계수(cooling coefficient)인 α는 s_{new}가 나쁘더라도 수용하는 확률에 영향을 미치는 주요한 파라미터로서, 실험적으로 정해지게 된다.

Step 1) $s = s_0$, $Temp = Temp_0$;
Step 2) For $h = 0$ through $h_{\max}$ {
2-1) Pick a random neighbour of s and create s_{new} ;
2-2) $\Delta = F(s_{\text{new}}) - F(s)$;
2-3) if ($\Delta < 0$)
2-3-1) $s = s_{\text{new}}$;
else
2-3-2) $s = s_{\text{new}}$ with probability of $e^{-\Delta/Temp}$;
2-4) $Temp = \alpha * Temp$;
}
Step 3) Return s;

[Fig. 4] An SA-based algorithm for solving the IP problem depicted in [Fig. 3]

3.3 실제 적용에서의 고려사항

게임 개발자의 의도에 따라, [Fig. 2]의 문제 정의에 추가 제약 조건들을 설정할 수 있을 것이다. 이 경우에 해당 조건들이 IP 문제의 조건으로 표현될 것이다. 대표적인 조건 예들이 [Fig. 5]에 나타나 있다.

만약 도전 타입 ci를 존 z에 반드시 배치하고자 한다면, ACO1 조건을 추가하면 될 것이다. 반대로 ci를 존 z에 배치하지 않으려면 ACO2 조건을 추가면 될 것이다. 또한 존 z에 b개의 도전을 배치하고자 한다면, 조건 ACO3으로 설정하면 될 것이다. 또한 ci와 cj가 동시에 존 z에 배치되는 것을 피하고자 하면, 조건 ACO4를 이용하면 되는데, M은 임의의 큰 상수, wx와 wy는 이진 변수이다. 또 다른 방법으로서, 변수 $x_{z,i}$와 $x_{z,j}$을 제거하고 새로운 변수 $x_{z,w}$을 도입해 해를 구한 후, $x_{z,w}$의 값을 $x_{z,i}$ 또는 $x_{z,j}$의 값으로 해석하면 된다.

(ACO1) $x_{z,i} >= 1$
(ACO2) $x_{z,i} = 0$
(ACO3) $MIN_z^{zone} = MAX_z^{zone} = b$
(ACO4) $x_{z,i} \le Mw_x$
$x_{z,i} \ge 1 - M(1 - w_x)$
$x_{z,j} \le Mw_y$
$x_{z,j} \ge 1 - M(1 - w_y)$
$w_x + w_y \le 1$
$w_x, w_y \in \{0, 1\}$

[Fig. 5] Additional constraints of the IP problem depicted in [Fig. 3]

$T(c_i)$는 실험을 통해서 구해 질 수 있다. 사용자 일반을 대상으로 게임 난이도를 설정하고자 한다면, 미리 여러 번의 실험을 통해 c_i를 통과 하는 평균 시간을 구하고, 이를 이용해 $T(c_i)$를 계산하면 될 것이다.

만약 개별 사용자별로 현 플레이 수준에 맞추어 동적으로 게임 난이도를 설정하고자 한다면, 게임 세션마다 사용자가 c_i를 통과하는데 걸리는 시간을 모니터링하는 작업이 필요하다. 모니터링 결과를 토대로 $T(c_i)$를 지속적으로 수정해가면 될 것이다. 우리의 경험에 의하면 자동차 게임에서의 주행 실력은 짧은 기간에 바뀌지 않는다. 따라서 일정한 기간 (예를 들면, 최근 1주 또는 2주) 내에 축적된 사용자의 주행 데이터를 기반으로 각 도전 타입별 평균 통과 시간을 측정한 후, 이를 기반으로 $T(c_i)$를 구해 주기적으로 업데이트하면 된다.

4. 실 험

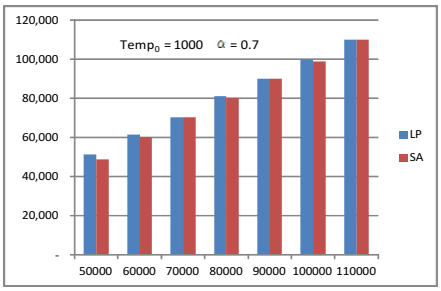
우리는 도전 배치 알고리즘의 유용성을 실험을 통해 분석했다. 제안된 방법은 C 코드로, 자동차 게임 프로토타입은 ActionScript로 구현했다. 실험에 사용된 컴퓨터의 사양은 i5 및 3.4Ghz CPU 및 8G 메모리이다.

실험에 사용한 트랙은 처음 절반은 직선 도로이고, 나머지 절반은 우회전이 필요하도록 오른쪽으로 휘는 곡선 도로 형태이다. 단순한 트랙의 형태를 사용한 이유는 특정 영역을 통과하는 시간이 트랙의 모습보다는 도전 배치에 영향을 더 받도록 하기 위함이다. 트랙은 총 4개의 존으로 구성되고, 총 8개의 도전 타입들이 있고 이들은 3개 범주로 분류된다. $T(c_i)$는 개인별로 다를 수 있기에, 성인 한명을 대상으로 총 20회 시험 주행을 한 후, 각 도전 타입별 $T(c_i)$를 구했다. $T(c_i)$들의 평균값은 1,828이다. 참고로, 본 장에서 언급되는 주행 시간 값은 msec 단위로 표현된다.

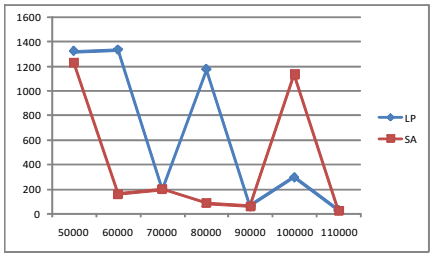
먼저 다양한 목표 주행 시간 T에 대해 [Fig. 2]에 기술된 도전 배치 알고리즘을 적용해 보았다. 이때, 각 MIN_i는 0, MIN_z^{zone}는 0, MIN_g^{cat}는 0, MAX_i는 8, MAX_z^{zone}는 16, MAX_g^{cat}는 25로 설정했다, [Fig. 6]은 Step 2의 결과를 보여주는데, 구해진 도전 배치 s하에서 트랙 한 바퀴 도는 주행 시간을 $f(s)$로 계산한 것이다. [Fig. 6]에서 "LP"는 IP 문제를 LP 문제로 완화시킨 후 구한 해의 주행 시간이고, "SA"는 [Fig. 4]의 알고리즘으로 구한 해의 주행 시간이다.

[Fig. 6]에서 보다시피, 도전 배치 알고리즘은 목표 시간 T에 접근하는 주행 시간을 제공하는 도전 배치를 구함을 알 수 있다. 본 실험의 경우, 목표 시간 T와 도전 배치들 간의 주행 시간 차이는 크지 않다. 이 차이는 도전 타입의 종류, 주행시간, 도전 배치 조건들에 따라 달라질 수 있을 것이다.

[Fig. 7]은 다양한 목표 주행 시간 T별로, T와 도전 배치의 주행 시간간의 차이를 보여준다. 그림에서와 같이, [Fig 4]의 SA 알고리즘은 LP로 구한 해를 개선시키는 효과가 있다. T=100,00인 경우에는 SA의 해가 LP의 해보다 못하다. 이것은 SA 알고리즘은 해를 점진적으로 개선하는 과정에서 [Fig. 4]의 Step 2-3-2에서와 같이 이전 해보다 나쁜 해도 확률적으로 수용할 수 있기 때문이다. 실제 응용에서 SA의 해가 LP의 해보다 못하면, SA의 해를 무시하면 될 것이다. [Fig. 7]에서는 T가 증가하면, 시간 차이가 줄어드는 경향을 보인다. 이것은 목표 주행 시간이 늘어나면, 고려할 수 있는 도전들의 가 증가하게 된다. 따라서 많은 도전 조합들 중에서 목표 주행 시간에 보다 근접하는 해를 구할 가능성이 높아지기 때문으로 해석된다.



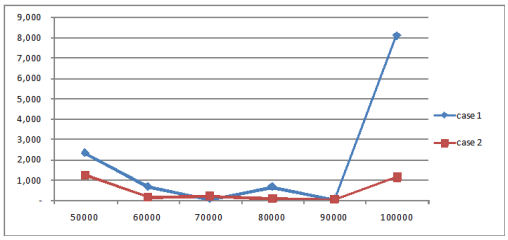
[Fig. 6] The result of experiments with the procedure in [Fig. 4]



[Fig. 7] The effect of SA on the result

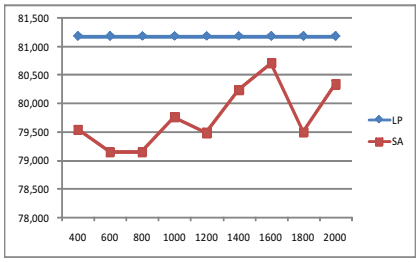
도전 배치 알고리즘에서 제약 조건이 해에 미치는 영향을 알아보기 위한 실험도 수행했다. 본 실험에서는 MAX_z^{zone}=8한 것 외에는 앞에서 언급한 실험과 조건을 동일하게 함으로써, 제약 조건을 다소 강화했다. [Fig. 2]의 Step 2의 결과를 정리한 것이 [Fig. 8]이다. *"case 1"*은 본 실험에서의

결과를, *"case 2"*는 앞선 실험의 결과를 보여준다. 그림에서 보다시피, 제약 조건이 강해지면, 구하는 해와 목표 시간 T와의 차이가 커짐을 알 수 있다. 이것은 제약 조건이 강해지면 고려할 수 있는 도전 배치의 경우 수가 줄어들어 좋은 해를 구할 가능성이 낮아지기 때문일 것이다. MAX_z^{zone}=8이란 제약 조건 때문에, *"case 1"*에서는 T=100,000처럼 목표 시간에 근접한 도전 배치를 찾지 못하는 경우도 발생했다.



[Fig. 8] The comparison between different sets of constraints for the IP problem

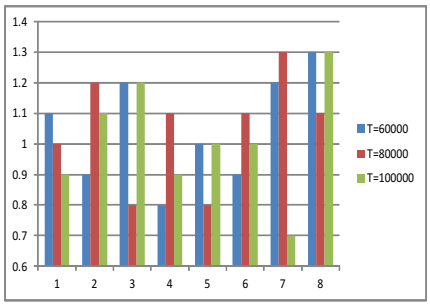
Fig 4에 기술된 SA 알고리즘의 주요 파라미터 중 하나는 초기 온도 $Temp_0$이다. 앞에서 언급했듯이 온도는 새로운 해가 기존 해보다 열등하더라도 그것을 수용하는 확률을 결정한다, [Fig. 9]는 초기 온도 $Temp_0$이 해에 미치는 영향을 실험한 결과를 보여준다. 여기서, T=8,000, α=0.7에 설정되었다.



[Fig. 9] The effect of Temp0 in the procedure in [Fig. 4] (α = 0.7)

그림에서는 초기 온도가 높아지면 SA 알고리즘이 보다 나은 해를 구하는 경향을 보인다. 이것은 SA의 점진적 개선 과정에서, 높은 온도는 열등한 해를 수용할 확률을 높인다. 열등한 해를 확률적으로 수용하는 이유는 로컬 맥시마 (local maxima)에서 빠져나오도록 하기 위함이다. 하지만, 초기 온도를 높이더라도 항상 해가 개선되는 것이 아니기 때문에, 주어진 도전 배치 문제의 특성에 맞는 초기 온도는 실험적으로 구해야 할 것이다.

온도 쿨링 계수 α도 열등한 해를 수용하는 확률과 관련되어 있어, 해에 영향을 미치는 파라미터이다. 다양한 쿨링 계수들에 대해 해를 구하는 실험을 한 결과, 높은 쿨링 계수들이 낮은 쿨링 계수보다 나은 해를 구하는 경향을 보였다. 하지만 초기 온도와 같이, 주어진 도전 배치 문제의 특성에 적합한 쿨링 계수는 실험적으로 구해야 할 것이다.

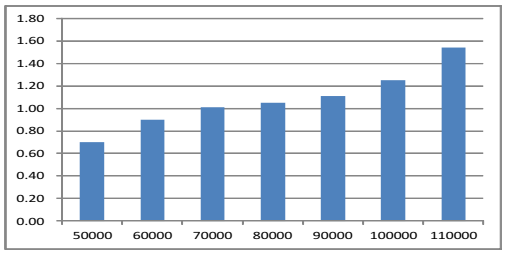


[Fig. 10] The result of test drives on the tracks with different obstacle placements

먼저 우리의 자동차 주행 게임 프로토타입에서 도전 타입들을 무작위로 트랙 상에 배치한 후, 실험자가 수차례 시험 주행을 했다. 각 도전 타입별로 평균 통과 시간을 구해 이를 $T(c_i)$들로 설정했다. 다음에는, 도전 배치 알고리즘을 통해 배치할 도전들을 구했다. 이때 난이도인 목표 주행 시간을 T=50,000, 80,000, 100,000로 각각 설정했다. 사람마다 주행 실력이 차이가 나기 때문에, 동일한 실험자가 구해진 도전 배치하의 트랙을 8번씩 주행해 보았고, 그 결과는 [Fig. 10]에 나타난다. 그림의 가로축은 주행 회차, 세로축은 목표 주행 시간까지 실제 주행한 트랙 바퀴 수를 나타낸다. 각 목표 주행 시간별로 0.9 ~1.3 바퀴를 실제 주행했고, 그 평균은 1바퀴 정도이다. 여러 명의 타 실험자들

을 대상으로 동일한 실험을 했을 때도 비슷한 결과를 보였다.

[Fig. 11]은 T=80,000에 맞게 장애물이 배치된 트랙에 대해, 다양한 제한 시간까지 시험 주행한 결과이다. 그림의 가로축은 제한 시간이고, 세로축은 주행한 트랙 바퀴 수이다. 그림에서 보다시피, 제한 시간이 80,000보다 커지면 주행 거리도 그만큼 늘어나는 현상을 보여준다. 당연하게 제한 시간이 80,000보다 작아지면 주행 거리도 그만큼 줄어듦을 보여준다.



[Fig. 11] The result of test drives on the track with a obstacle placement for T = 80,000 under different time limits

5. 결 론

사용자의 플레이 수준에 맞는 게임 난이도를 제공하거나 또는 여러 난이도를 가진 레벨들을 제공하는 것은 게임 몰입도를 높이기 위한 중요 요소들 하나이다. 게임 난이도를 수작업으로 조절하는 것은 지루하면서도 힘든 일이다. 본 논문에서는 1인칭 자동차 주행게임에서 게임 난이도의 자동 설정을 위한 IP 기반의 도전 배치 방법을 제안했다. 여기서 도전이란 주행 트랙 상에 배치하는 장애물을 말한다.

제안된 방법에서는 트랙 한 바퀴를 도는데 소요될 목표 주행 시간 형태로 게임 난이도가 주어지면, 트랙 상에 배치할 도전 타입, 도전 수 및 위치를 구하는 업무를 IP 문제로 모델링했다. IP 문제의 해는 LP 완화와 SA 알고리즘을 통해 구했다.

실험을 통해서, 본 논문에서 제안된 방법이 목표 주행 시간과 큰 차이가 나지 않는 주행 시간을 제공하는 도전 배치를 구함을 알 수 있었다. 또한, 구한 도전배치를 적용한 트랙을 시험 주행한 결과, 시험 주행 시간은 해당 도전 배치의 난이도에 근접하는 주행 시간을 보였다.

자동차 주행 게임에서 트랙 한 바퀴를 도는데 걸리는 시간은 게임 플레이의 수준을 나타내는 측도이다. 본 논문의 도전 배치 방법으로 다양한 주행 시간을 필요로 하는 트랙을 구성함으로써, 자동차 주행게임[14, 15]에서 다양한 난이도의 게임플레이가 가능한 환경을 사용자에게 제공할 수 있을 것이다.

REFERENCES

[1] Fausto Mourato, Manuel Próspero dos Santos, Fernando Birra, "Automatic Level Generation for Platform Videogames Using Genetic Algorithms", Proceeding of the 8th International Conference on Advances in Computer Entertainment Technology, 2011, pp.1-8.

[2] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. "Difficulty Scaling of Game AI", In Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004), 2004, pp.33 - 37.

[3] R. Hunicke and V. Chapman. "AI for Dynamic Difficulty Adjustment in Games", In Challenges in Game Artificial Intelligence AAAI Workshop, 2004, pp.91-96.

[4] Robin Hunicke, "The Case for Dynamic Difficulty Adjustment in Games". Proceedings of the ACM SIGCHI International Conference on Advances in computer entertainment technology, 2005, pp.429-433.

[5] K. Compton, M. Mateas, "Procedural Level Design for Platform Games", In Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE), 2006, pp.109-111.

[6] Nirach Watcharasatharpornpong, Vishnu Kotrajaras, "Automatic Level Difficulty Adjustment in Platform Games Using Genetic Algorithm Based Methodology", http://www.researchgate.net/[DOI:10.5176/978-981-08-3190-5_482], 2010.
[7] S. Kim and D. Park, "Integer Programming-based Generation of Difficulty Level for a Racing Game", Proceeding of ICEC, 2015 (to be published).
[8] G. Smith, M. Mateas, J. Whitehead, M. Treanor, "Rhythm-based Level Generation for 2D Platformers", In Proceedings of the 4th International Conference on Foundations of Digital Game, 2009, pp.175-182.
[9] P. Mawhorter, M. Mateas, "Procedural Level Generation Using Occupancy-regulated Extension", IEEE Conference on Computational Intelligence and Games, 2010, pp.351-358.
[10] N. Shaker, G. Yannakakis, and J. Togelius. Towards, "Towards Automatic Personalized Content Generation for Platform Games", In Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE'10), 2010, pp.63-68.
[11] Olana Missura and Thomas Gärtner, "Player Modeling for Intelligent Difficulty Adjustment," Proceedings of the 12th International Conference on Discovery Science, 2009, pp.197-211.
[12] Simulated Annealing, Https://en.wikipedia.org/wiki/Simulated_annealing
[13] S. Kirkpatrick, et. al, "Optimization by Simulated Annealing", Science. 220, 1983, pp.671-679.
[14] S. Kim, H. Park, "A Single-Player Car Driving Game-based English Vocabulary Learning System", Journal of Korea Game Society. 12(5), 2015, pp.95-104.
[15] Ock-Tae Kim, "Effectiveness of G-Learning Math Class in Increase of Math Achievement of K-5 Students in USA", Journal of Korea Game Society. 11(6), 2011, pp.201-212.

김 상 철(Kim, Sangchul)

약력 : 1994 미시간주립대학교 컴퓨터공학과 박사
1983-1994 ETRI 연구원
1994-현재 한국외국대학교 컴퓨터공학과 교수

관심분야 : 기능성게임, 게임 AI, 멀티미디어시스템

박 도 생(Park, Dosaeng)

약력 : 2011 한양사이버대학교 정보통신공학 학사
2013-현재 한국외국대학교 컴퓨터공학전공 석박사통합 과정 학생
2013-현재 KAIAIST 대표
2011-2012 (주)엔터유 대표
1993-1996 워드프로세서 및 프로그래밍 도서 10여권 집필

관심분야 : 멀티미디어시스템, 기능성 게임, 임베디드 시스템