Journal of Korea Game Society 2014 Dec; 14(6): 19-28
http://dx.doi.org/10.7583/JKGS.2014.14.6.19

게임 프로그래밍

게임 스크립트를 위한 시행착오 회피 요소

정찬용
장안대학교 게임컨텐츠과
tyonarts@empas.com, tyonarts@jangan.ac.kr

Avoiding Trial-and-Error based Elements for Game Scripting

ChanYong Jung
Dept. of Game Content, Jangan University

요 약

게임 스크립트는 프로그래밍 비숙련자가 게임을 구현하는 경우에 자주 활용된다. 이 논문에서는 게임 프로그래밍 구현 과정 중에 발생하는 시행착오 항목들을 조사하여 게임 스크립트를 위한 유용한 요소로 도출 한다. 도출된 시행착오 요인을 회피할 수 있도록 제안되어 Unity3D 게임엔진 환경의 C# 언어로 구현된 보완 명령문들에 대하여 논의 하고, 이 명령문들을 적용한 구현 방법과 기존의 방법을 비교한다.

ABSTRACT

Scripting is often used in games for novices at programming to enable implementation of the game. In this paper we examine the trial-and-error elements taken to implement game programming and identify the useful features for game scripting.

This paper presents complemental elements based on avoiding trial-and-error, and discusses complementary statements implementation of this proposal, where the complementary statements were written in C# on Unity3D engine. The approach is compared and contrasted to the existing approach.

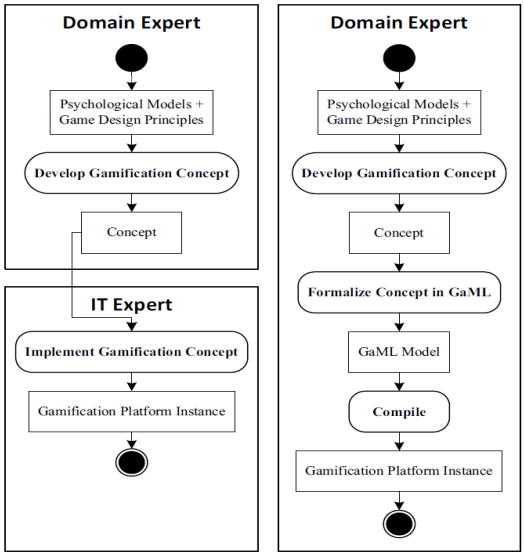
Keywords : Gamification(게이미피케이션), Game Scripting(게임 스크립트), Game Programming(게임 프로그래밍)

Received: Nov. 14, 2014 Revised: Dec. 11, 2014
Accepted: Dec. 12, 2014
Corresponding Author: ChanYong Jung(Jangan University)
E-mail: tyonarts@jangan.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

프로그래밍의 핵심은 어떤 문제를 해결하기 위한 논리적인 절차의 확립과 그 구현이라고 할 수 있다. 그러나 구축된 논리적 절차를 특정 프로그래밍 언어로 표현하여 구현하는 것은 논리적 절차 이해와는 다른 별개의 문제이다. 동일한 알고리즘이라 하더라도 프로그래밍 언어의 문법 특성에 더 종속적인 기법과 표현들이 사용되는 경우가 일반적이다. 더불어 게임 프로그래밍에서는 게임화 콘셉트의 실체화를 위한 모듈들을 구성하여야 하는데 이러한 여러 요인들이 오류나 시행착오를 증가시켜 구현 효율을 떨어뜨리는 중요한 원인으로 작용한다.



[Fig. 1] Traditional(left) vs. trendy(right) approach using game script

여러 분야에서 비숙련자들에 의한 프로그래밍 활용이 늘고 있으며 스크립트 언어가 자주 사용된다. 게임 관련 학과의 교육 과정을 비롯하여 타 영역 전문가가 IT 전문가와의 협업 없이 스크립트로 구현하는 게이미피케이션도 이런 경향을 보여주는 사례이다([Fig. 1]). 산업 현장에서도 게임기획자나 레벨 디자이너 등 비 프로그래머 직군이 게임의 진행 방식 등을 직접 바꾸면서 테스트 할 수 있는 게임 스크립트의 활용이 일반화하고 있다[11]. 최근 3년간 게임 엔진 시장의 큰 변화를 주도한 Unity3D의 스크립트 체계[1]나 2014년에 발표된 Unreal 4의 블루프린트 스크립트를 들 수 있다. 블루프린트는 기능이 정의된 논리 블록들을 제어 라인으로 연결하는 직관적이고 시각화된 프로그래밍 방법을 제시함으로써 비 프로그래머 직군의 프로그래밍 협업을 효율화하였다[4]. 범용 언어로 저수준의 모듈을 직접 코딩하던 게임 개발 패러다임이 상용 게임엔진 환경에서도 서서히 변화되고 있는 것이다. 프로그래밍 비숙련자들을 위하여 효율적이고 쉬운 개념의 스크립트를 활용하는 것은 교육, 연구, 산업 등 여러 분야에서 유용하다[2,5]. 게임 스크립트는 초기에 주로 게임 시나리오나 초기화 데이터의 변경을 위한 간이 기술언어로 사용되었지만 점차 폭넓게 응용되면서 다양한 기능과 복잡성을 갖는 형태도 연구되고 있다[3].

이 논문에서는 프로그래밍 비숙련자들이 기존의 게임 개발 시스템과 언어체계를 이용하여 게임을 구현하는 과정을 분석하여 오류와 시행착오가 많이 발생되는 요인을 도출하고 이를 보완할 수 있는 대체 방법과 설계 방향을 제안하였다.

2. 연구 개요

2.1 관련 연구

프로그래밍 비숙련자를 위하여 직관화, 시각화, 오류 가능성 최소화 등에 초점을 맞춘 프로그래밍에 대한 연구가 활발히 진행되어 왔다[6]. Moons와 De Backer가 연구한 EVizor는 Netbeans IDE에 플러그인 되어 활용할 수 있는 비주얼 자바 프로그래밍 환경으로 정규교과에 활용되어 효율성이 검증되었다[7]. Kirby는 VIP 툴에서 도입한 C++ 인터프리터와 Jeliot 3 UI를 결합하여 C 프로그래밍을 돕는 Jeliot 시각화 변형 버전을 강의에 적용하였다[8]. Wang의 연구는 실행할 때 변화되는 자

료구조의 상황을 애니메이션으로 시각화하여 이해도를 높이고 오류를 줄여주는 프로그래밍 지원 시스템으로 강의에 적용된 피드백 데이터가 분석되었다[9]. Guo의 온라인 Python Tutor는 시각화된 small Python 프로그램 실습을 지원하는 웹 기반 시스템으로 연구되었다[10].

2.2 연구 방법과 목표

이 논문의 연구 방법과 목표는 다음과 같다. 피실험자인 프로그래밍 비숙련자들이 Unity3D와 C#을 이용하는 기존 개발방법으로 게임 8종을 구현하는 과정을 조사하여 오류와 시행착오 항목을 도출한다. 도출된 항목을 분석하여 오류를 감소시킬 수 있는 보완 방안이 있다면 이를 제안한다. 실현 가능한 제안들은 실행될 수 있도록 완성한 코드를 제공하여 보완된 기능을 적용할 수 있게 한 후 먼저 실험되었던 게임들과 유사한 유형의 다른 8종의 게임을 구현하도록 한다. 보완 효과를 적용한 경우와 적용하지 않은 경우의 오류 빈도와 시행착오 시간에 대한 피실험자 전체 평균을 비교하여 연구의 실효성을 평가한다.

오류와 시행착오에 대한 보완 요인은 명령문, IDE, 언어의 문법체계에 이르기까지 대상 범위가 넓다. 즉, 명령문 단위의 기능 개선 수준에서부터 새로운 명령문을 제안하거나 시스템 또는 언어의 설계 방향을 제시할 수도 있다. 게임 구현에 유용한 라이브러리에서 게임 스크립트 설계에 이르기까지 보완의 개념은 광범위하지만 이 논문에서는 기존 명령문의 기능을 보완할 수 있도록 제안된 가상 명령문을 C# 메서드로 구현하여 제공함으로써 효과성을 실험하였다. 소규모 라이브러리 수준의 보완이지만 기존 게임 구현에 이와 같은 요소들이 누적됨에 따라 프로그래밍 비숙련자들에 의한 게임 구현의 효율성을 향상시켜줄 수 있을 것이다.

연구문제 1. *알고리즘이 제공됨에도 불구하고 기존 개발환경과 언어를 사용한 구현 과정에서 어떤 오류들이 평균적으로 자주 발생하는가?*

연구문제 2. *발생된 오류들이 명령어의 기능 개선으로 보완될 수 있다면, 보완된 가상 명령어를 대신하는 메서드를 사용하여 비슷한 유형의 게임을 구현할 경우 피실험자들의 평균적인 구현 효율성이 높아지는가?*

3. 오류 및 시행착오가 발생하는 프로그래밍 작업 단계의 조사

3.1 실험 인원 및 환경

오류와 시행착오가 발생하는 프로그래밍 작업 단계를 조사하기 위한 실험은 장안대학교 게임과 2학년 학부생 71명을 대상으로 매주 2회 3시간씩 총 15주 동안 게임 프로그래밍 실습교과 시간에 실시하였다. 측정 기간 도중에 불참한 3명을 제외한 68명을 유효 실험대상으로 확정하였으며 통계적 특성은 [Table 1]과 같다.

[Table 1] General features of test subject group

Features		N		(%)
		P	Q	
gender	male	30	27	(83.9)
	female	5	6	(16.1)
major	programming	25	20	(66.2)
	graphic	10	13	(33.8)
grade of CS1	A or A+	11	10	(30.9)
	B or B+	14	14	(41.2)
	below C+	10	9	(27.9)
coding experience	longer than 1 year	3	1	(5.9)
	1 year	32	32	(94.1)
total		35	33	(100.0)

피실험자들은 입학 후 2학기 동안 CS1, 3학기부터는 Unity3D 기반 C#을 이용한 게임 구현과목을 이수하여야 하며 P와 Q 두 클래스로 분할 운영되고 있다. 학업성취 이력과 관련한 효과 분석을 위

하여 CS1 성적을 기록하였고 개인별 상담 자료에 의거, 대학 입학 이전의 프로그래밍 경험을 조사하여 유효 인원의 5.9%에 해당하는 4명이 1년 이상의 프로그래밍 경험을 가지고 있는 것으로 파악하였다.

3.2 실험용 게임과 실험 방법

[Table 2]와 같이 네 장르로 분류된 여덟 유형의 게임 16종을 구현하였다. 패턴 학습을 위하여 구현 알고리즘과 테크닉의 유사성이 높은 두 게임을 각각 A, B세트로 하여 하나의 유형을 이루도록 구성 되어 있다.

[Table 2] Work steps of test games

Genre	Type	Set	Game title	Work steps
Act.	1	A	Bubble Bobble	34
		B	Puzzle Bobble	37
	2	A	Space Invader	21
		B	Space Invader 2	25
Sim.	3	A	Tiny SimCity	76
		B	Tiny SimTower	78
	4	A	Mini AngryBirds	45
		B	Mini Gears	49
RPG	5	A	Dungeon Alien	57
		B	Dungeon X	58
	6	A	Simple YS 1	62
		B	Simple YS 2	63
Puz.	7	A	Columns	13
		B	Simple Ani Pang	16
	8	A	Memory Test	11
		B	Door Door	11

실습 게임들은 강의 내용과 일정에 맞춰 작업 단계들이 설정되어 있다. 유형별은 물론 동일 유형 내의 두 게임도 작업 단계의 수와 내용에 차이가 있다. [Table 3]은 21개의 작업 단계로 설정된 Space Invader의 구체적 작업 항목으로 Ebner[11]의 비디오 게임 기술언어 연구 과정에서 제시된 모듈 설계를 바탕으로 유니티 환경에 맞도록 재설정한 것이다. 실험에 사용된 게임들의 작업 단계는 모두 구현을 위한 근거 자료를 기반으로 설정되어 있다[12]. Memory Test와 같은 간단한 퍼즐 게임은 11단계의 작업 과정을 거쳐 완성되지만 Tiny SimCity는 도로와 건물을 배치하는 등 구현할 클래스가 많아 76단계의 작업 과정이 필요하다.

[Table 3] Work steps of Space Invader

Step	Things to do
1	declare variables
2	game manager class
3	event manager class
4	FSM class
5	game world class based on 2D array
6	message manager class
7	process of fixed object's resource
8	fixed object class: spawn points
9	fixed object class: EoS
10	fixed object class: base tile
11	process of moving object's resource
12	avatar class
13	missile class : avatar
14	missile class : enemy
15	saucer class
16	enemy class : initial formation
17	enemy class : translation
18	enemy class : special behavior
19	test of co-routine
20	game manager class : update stage
21	final test

게임을 구현하기 위한 작업 단계는 가변성이 있어 정규화 및 표준화가 어렵지만 개발 환경과 언어체계가 특정되는 조건이라면 가변 폭은 크지 않다고 판단하였다. 또한 작업 단계는 시간 지연이 발생하였는지 여부를 판단하기 위한 구분 단위로 사용될 뿐 보완 요소를 추출하기 위하여 실제 누적하는 항목은 발생된 지연 시간과 오류 종류이다.

3.3 실험 방법

언어에 의한 구현 단계만을 조사 대상으로 하기 위하여 피실험자들에게 게임 규칙과 적용 기법 등 핵심 내용을 설명하고 정확한 알고리즘 및 클래스

설계 문서를 제공하였다. 작업 단계별로 구현이 될 때마다 어느 정도의 시간이 소요되었는지 개인별로 기록한 후 다음 단계로 진행하도록 하였다. 각 단계별로 지연 시간의 상한을 정하여 그 이상 지연되면 작업을 중지시킨 후 최대 지연 시간을 부여하였다. 지연이 일어난 오류의 종류를 기록한 후 다음 단계의 구현을 계속할 수 있도록 실패한 단계에 대한 완성 코드를 제공하였다.

[Table 4] Experimental process

State	Description
1	class: P, Q (game: type N, set A)
2	explain game rules and techniques
3	supply algorithmic & class documents
4	start implemental step
5	if (done) record run-time&go to State 7
6	if (failed & overdue) record maximum run-time, error-code supply prepared codes for current step test running

7	if (not final step)
	prepare next step & go to State 4
	if (final step & set B) go to State 10

8	analysis of the results (set A) statements or methods or system redefine & propose solution if (possible) build C# methods for comparison
9	game : same type, set B supply methods of State 8 to class P or Q (by turns weekly) go to State 4
10	analysis of the results (set B)
11	if (not final type) prepare next type & go to State 2

이와 같은 방법으로 모든 A세트 게임들에 대한 피실험자들의 오류 항목을 누적하였다. 산출된 오류들 중 명령문의 보완으로 개선이 가능하다고 분석된 경우, 보완된 명령문의 기능을 수행할 수 있도록 C#으로 코딩된 메서드 소스를 제공하여 B세트 게임들을 구현하도록 하였다. 비슷한 유형의 게임을 구현하는 상황에서 기인하는 패턴 학습효과에 의한 결과와 비교하기 위하여 메서드를 제공받는 피실험 클래스는 주 단위로 교체하였다. 즉, 홀수 주에는 클래스 P, 짝수 주에는 클래스 Q에만 대체 메서드를 제공하여 기존 방법대로만 구현한 상대 클래스와 결과를 비교하였다.

3.4 실험 결과

실험 결과는 [Table 5]와 같다. 작업 단계별 평균 수행 시간 기준은 A세트 게임(A), 메서드가 제공된 B세트 게임(B), 메서드가 제공되지 않고 기존 방법으로 구현한 B세트 게임(B*)으로 구분되어 있다.

[Table 5] The cause of a failure

The cause of a failure	Average run-time (min.)		
	A	B	B*
FSM class	144.3	92.3	119.2
game world class	141.9	n/a	n/a
messenger of objects	140.8	100.2	112.7
parallel processing	139.5	n/a	n/a
animation FSM class	137.6	112.3	113.2
resource linking	66.2	36.2	44.1
readability	55.1	33.1	49.4
local index variables	54.7	n/a	n/a
global variables	43.9	n/a	n/a
entities of game objects	32.3	22.6	22.7

게임 월드 모델화, 게임 오브젝트 병행처리, 변수 관리 등에 대한 오류들은 언어체계를 재설계하거나 시스템 내부와 연동하여야 하는 등 구현에 한계가 있어 제외하였다.

패턴 학습효과 때문에 B*의 수행 시간 측정값도 A에 비하여 낮았지만 FSM 매니저 등 대부분의 항목에서, 제안된 보완 메서드를 사용한 B의 평균 수행시간이 가장 낮은 값으로 측정되었다. 즉, 보완 방안을 사용할 경우 효율성은 향상될 수 있다.

4. 기존 개발체계에서의 보완 요소

4.1 변수 규칙과 관리

게임을 구현하는 과정에서 비숙련자일수록 변수 관련의 기본적인 시행착오가 의외로 빈번하다는 실험결과를 얻었다. 순환구조 내부에 제어용으로 사용되는 변수들의 연산은 오류를 일으키기 쉬우며 작은 수식 변환에도 큰 동작의 차이가 발생하여 디버깅에 적지 않은 시간이 소모되었다.

```
for(j=0; j<5; j++)
 for(i=0; i<11; i++) {
  switch(j) {
   case 0:
    Enemy[j][i].id = IDEN0;
    Enemy[j][i].frameoffset = 6;
    break;
   case 1: case 2:
    Enemy[j][i].id = IDEN1;
    Enemy[j][i].frameoffset = 8;
    break;
   case 3: case 4:
    Enemy[j][i].id = IDEN2;
    Enemy[j][i].frameoffset = 10;
    break;
  }//switch j
  Enemy[j][i].state = STON;
  Enemy[j][i].frame = 0;
  Enemy[j][i].framemax = 2;
  Enemy[j][i].framecols = 6;
  Enemy[j][i].sizew = 64;
  Enemy[j][i].sizeh = 47;
  Enemy[j][i].animode= ANILOOP;
  Enemy[j][i].anicount=3*j;
  Enemy[j][i].anispeed=3*(4+1);
Enemy[j][i].x=20+i*(10+Enemy[j][i].sizew);
Enemy[j][i].y=20+j*(22+Enemy[j][i].sizeh);
Enemy[j][i].xv = Enemy[j][i].sizew/4;
Enemy[j][i].yv = Enemy[j][i].sizeh/2;
}//for i
//for j
```

[Fig. 2] C compatible code for Invaders

실험에서 사용된 Space Invader의 캐릭터 동작 순환부분을 C 호환 코드로 구현한 [Fig. 2]에서 세 종류의 인베이더 객체들은 독립적으로 움직이지 않고 정해진 순서에 따라 한 객체 씩 프레임마다 한 단계 애니메이션 단위동작을 순차 반복하는 것이 특징이다. 이러한 독특한 움직임을 구현하는 핵심 코드는 switch 블록 내부에서 변수 j가 세 종류의 인베이더 객체 그룹을 분할하여 프레임 offset 값으로 각각 6, 8, 10을 지정하는 부분과 지정된 값의 3배수로 애니메이션 패턴을 할당하는 부분이다.

```
Enemy[j][i].anicount= 3*j;
```

이 동작을 사이클마다 하나의 객체 동작으로 변경하려면 명령문의 수식이 바뀌어야 한다.

```
Enemy[j][i].anicount= (j+1)*11 - (i+1);
```

'모든 객체가 한 동작 씩 순차적으로 움직인다'는 논리적 이해와 달리 구현을 위한 실제 코드는 산술식으로 표현되기 때문에 비숙련자에게는 직관적이지 않아 변수 j의 변경 값을 차례로 추적해야 객체의 동작을 이해할 수 있다. 게임 알고리즘과 직관성을 강조한 가상 게임 스크립트로 이 움직임을 표현하면 [Fig. 3], [Fig. 4]와 같은 형식이 가능하다.

```
55 invader objects : column 11 x row 5
do {
    shift one object per frame
    change animation patterns by one clip
    if(object.position == bound)
      { translate to down-position }
    } while (not game over)
```

[Fig. 3] Pseudo algorithm for Invaders

가상 스크립트는 foreach 선두에 조건 블록을 넣어 객체들을 순차 처리할 것을 명시하며 2차원 인덱스를 * 표현으로 직관화 하고 있다.

```
GameObject Enemy {11*5};
foreach {Enemy=11*5;unit=1;seq;} {
  GameObject.Id.Enenmy {line=6,8,8,10,10};
  GameObject.Animation.Enenmy;
  GameObject.MoveRightLeftToggle.Enenmy
    {step=size}
  GameObject.SetAttribute.Enemy
    {
     state = STON;
     frame = 0;
     framemax = 2;
     framecols = 6;
     sizew = 64;
     sizeh = 47;
     animode= ANILOOP;
     anispeed=3*(4+1);
     }
} //loop
```

[Fig. 4] Pseudo game script for Invaders

선두에 위치한 조건 블록은 Tcl 언어의 문법에서 확장한 형태로, 순환 패턴을 정의하거나 논리 조건을 지정하여 순환용 지역변수를 은닉하는 형식을 제안한다. 또한 게임 월드 내의 오브젝트 상황 판단용으로 자주 활용되는 2차원 배열은 순환 과정이나 초기 값, 순환 중지 조건 등이 다양한 오류를 발생시켰다. 따라서 순환 규칙을 블록 내부에 정의하고 전체 원소의 수만 정의함으로써 인덱스 지역변수를 은닉화 하여 2차원 배열 순환의 시행착오 비율을 낮추는 형식을 제안한다. 또한 게임객체에서 사용빈도가 높은 속성은 기본 멤버 변수로 클래스에 포함시켜 접근성을 높이는 방안도 고려할 수 있다. 변수에 대한 제안들은 언어체계의 재 정의와 내부 구조를 변경하는 것이므로 구현할 수 없었지만 비숙련자 중심의 게임 스크립트를 설계할 경우 원칙으로 제시할 수 있다.

4.2 IDE 속성과 스크립트의 교차 반영

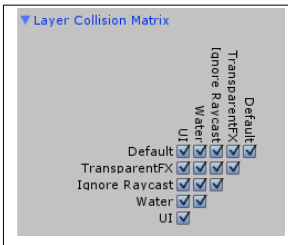
Unity3D IDE에서 프로젝트는 스크립트 파일과 에디터의 설정 값을 분리하여 관리한다. 실용적인 면에서는 문제가 없지만 스크립트 이외의 부분에서 게임의 중요한 속성이 별도로 기술되는 것이 오류를 일으키는 요인이 되었다. 따라서 에디터에서 설정되는 모든 속성들이 스크립트로 표현되고 내용도 교차 반영되어 일관성을 가질 수 있도록 확장되어야 한다. 실험에서도 에디터 상에서 잘못 설정한 값이 오류를 일으키고 그 원인을 스크립트에서 찾느라 시간을 소모한 경우가 있었다.

에디터에서 편집한 속성들을 스크립트에서 확인하거나 변경도 가능하도록 이 값들을 빌드 타임에 별도 스크립트로 생성되게 함으로써 시행착오를 감소시킬 수 있도록 제안한다. 스크립트에서 내용을 변경하면 역으로 IDE의 설정 값에도 교차 반영되도록 시스템의 기능이 개선되면 구현과 디버깅에 모두 효율적일 것이다. Inspector 뷰에서 지정하는 게임 객체의 Tag 속성 예를 들면, 빌드 후 별도 스크립트 파일에 [Fig. 5]와 같은 형식으로 반영되도록 하는 방법이다.

```
[Tag field]
//12 system reserved
//3  user defined
TAG0 = ENEMY
TAG1 = BULLET
TAG2 = WALL
```

[Fig. 5] Pseudo statements for tag field in Unity3D IDE

또한 레이어 충돌 매트릭스를 선택 지정할 수 있는 기능을 IDE에서 제공하지만 코드에 반영되지 않아 이에 대한 상황 파악이 늦는 점이 오류의 원인인 경우가 있었다. 에디터의 매트릭스 설정은 직교 레이아웃으로 디자인되어 있으므로 [Fig. 6]과 같은 개념의 문법형식을 적용한다면 텍스트이지만 시각적인 직관성을 가질 수 있다.




```
Type_of_LayerCollisionMatrix m;

define
  DT="Default"; FX="TransparentFX";
  IR="IgnoreRaycast"; WT="Water"; UI="UI";

m[ 0]={ UI , WT , IR , FX , DT };
m[DT]={  1,   1,   1,   1,   1 };
m[FX]={  1,   1,   1,   1      };
m[IR]={  1,   1,   1           };
m[WT]={  1,   1                };
m[UI]={  1                     };
```

[Fig. 6] An edit panel(top) and pseudo statements(bottom) for layer matrix in Unity3D IDE

하지만 통합에디터들이 표준화 되어 있지 않은 상황에서 특정 IDE의 속성만을 반영하는 것은 호환성 문제를 발생시킬 가능성이 있으므로 기존 IDE들의 표준화 연구가 선행될 필요가 있다.

4.3 게임 오브젝트 간 메시지 교환

게임 객체, 아이템 취득이나 경계 지점 도달, 특

정 환경 변수에의 반응, 플레이 중의 특정 상황에 대응한 리액션 처리 등에 메시지 전달이 요구된다. 실험결과에 따르면 특정 게임 객체에 연결된 스크립트 컴퍼넌트의 메서드를 호출하는 객체는 다른 게임 객체인 경우가 많아 혼돈이 있었다. 실험에 사용된 Unity3D C#에서는 다른 게임 객체의 메서드를 호출하는 것이 한 줄의 명령으로 구현되므로 효율성에서 불편한 점이 없으나 비숙련자를 위한 게임 스크립트라면 직관적이지 못한 표현일 수 있다.

```
GameObject.Find("object_name").SendMessage("method_name", transform.position);
```

예를 들어, 어느 게임 씬에 A, B, C, D 네 개의 게임 오브젝트가 생성되어 있는데 B와 C가 접촉하면 A가 e 아이템을 얻고, C와 D가 충돌하면 D객체가 없어지고 f 아이템을 잃는 상황을 가정한다.

```
// A script for object B:
   if (C on trigger) {
       Find(A); A.GetItem(e);
   }

// A script for object C
   if (D on collision) {
       Find(A); A.LoseItem(f);
       Find(D); D.Destroy(D);
   }
```

[Fig. 7] Pseudo statements written in OOPL including C# on Unity3D

C#을 비롯한 일반적인 객체지향 언어에서는 객체별 스크립트로 [Fig. 7]과 같이 표현하는 것이 일반적이다. 하지만 게임 동작의 직관적 표현을 우선하는 게임 스크립트라면 [Fig. 8]과 같이 기술되는 문법으로 구성하여 혼돈을 줄여줄 수 있다. 스크립트가 특정 객체별로 연결되거나 외부 객체를 검색하는 명령 개념이 직접적 표현 원칙을 방해하므로 해당 메서드의 대상이 되는 객체를 직접 지정하는 표현이 가능하도록 하여야 한다.

실행코드는 내부적으로 [Fig. 7]과 같이 빌드 되겠지만 게임 스크립트 문법의 직관적 표현으로 서술되는 [Fig. 8]의 코드를 파싱할 수 있다면 효과적일 것이다.

```
if (B,C on trigger)
    A.GetItem(e);
if (C,D on collision) {
    A.GetItem(f);
    D.Destroy(D);
}
```

[Fig. 8] Pseudo statements written in the proposed script

4.4 게임 유한상태기계의 기본 클래스

게임 진행 상황이나 캐릭터는 유한상태를 가지며 주변 환경과 플레이어의 판단에 따른 결과로 영향을 받으면서 다른 상태로 변화하게 된다. 초기의 게임 개발부터 FSM이 넓게 활용되었던 것을 감안한다면 FSM을 시스템 기본 클래스로 제공하는 것이 유용하다. 게임마다 별도의 자체 FSM이나 hFSM, 의사결정트리를 구현하는 작업 자체가 비숙련자들에게는 시행착오를 증가시키는 중요한 원인이었다. 기본 FSM 모듈을 C#으로 구성한 제어 메서드들은 피실험자들의 평균 구현 효율을 높이는 데 가장 효과적이었다.

4.5 동작 애니메이션의 매칭 매니저

현실적으로 게임 표현은 대부분 애니메이션의 형태로 이루어지므로 애니메이션 조작에 대한 직관적 문법 체계는 게임 스크립트에서 중요한 비중을 차지한다. Unity3D 뿐만 아니라 Unreal 4 등 최근의 IDE에서는 Animator Component 개념을 도입하여 애니메이션 동작들을 일종의 유한상태로 정의하고 FSM 매니저에서 편집할 수 있게 하고 있다. 그러나 애니메이션 동작 테스트나 브랜딩에 필요한 시행착오는 논리적 시행착오와는 성격이 다르다. 애니메이션은 감성적인 관점에서 적합한 동작표현을 찾아야 하기 때문에 IDE 인터페이스에서 시행착오를 거쳐 작업하는 것이 자연스러우며 그 결과만 스크립트로 반영시켜 가독성을 높이고 가벼운

변경작업은 코드 수준에서 가능하도록 하는 것이 효율적이다. 4.4에서 논의한 게임 시스템의 FSM 기본 클래스를 바탕으로 스크립트 지원이 어려운 브랜딩 등의 애니메이션 메서드들을 추가하고 IDE에서 편집된 애니메이션 유한상태와 전이조건을 교차 반영할 수 있는 확장 형식을 제안한다. 교차 반영은 유니티 시스템 내부와 연동되어야 하기 때문에 구현하지 못하였으나 브랜딩 등의 메서드는 피실험자에게 제공한 결과 큰 차이는 아니지만 작업 시간이 단축되는 효과가 있었다.

```
public AniClip walk;
public AniClip run;
p.Ani.Blending(walk,0.4f,run,0.6f);
```

[Fig. 9] Pseudo statements for blending animations in the proposed script

4.6 게임 오브젝트 병행처리 매니저

여러 캐릭터가 동시에 자율적으로 동작하거나 단일 캐릭터 내부에서 다중 동작 처리가 필요한 경우 이러한 특성을 반영하는 병행처리 명령체계가 필요하지만 그 스크립트 형식은 직관적이고 오류 가능성이 적은 방향으로 보완되어야 한다. 유니티 환경 C#의 경우, 코루틴과 yield return 사용이 일반화되어 있지만 복귀 시간을 여러 간격으로 변경하면서 적절한 타이밍을 찾아야 하는, 병행처리에 관련된 시행착오가 있었다. 스크립트 내에 기술된 특정 레이블을 기준으로 yield return을 동기화 하여 복귀 시점이 자동적으로 조절되는 기능을 가진 명령문으로의 확장을 제안 한다.

예를 들어, 프로세스 A와 B를 코루틴으로 연속 호출하는 상황을 가정해본다. A 처리 중에 <p1> 시점에서 yield return 되어 B를 처리하고 <p2> 시점에서 다시 A 코루틴을 재처리하는 상황을 가정하면 시간을 맞추기 위하여 <p1>의 시간 인수를 변경하면서 시행착오를 반복하여야 한다. <p2>와 같은 개념의 명령문이 제공되면 시행착오 없이 <p1>으로 복귀할 수 있다. 이 제안은 유니티 시스템 내부와 언어체계에 관련한 것으로 구현되지는 않았다.

```
Update() {
  StartCoroutine("Process_A");
  StartCoroutine("Process_B");
}

IEnumerator Process_A() {
 yield return new WaitSeconds(1.5f);//<p1>
}

IEnumerator Process_B() {
 yield return autoBack("Process_A");//<p2>
 yield return;
}
```

[Fig. 10] Pseudo yield return written in the proposed script on Unity3D IDE

5. 결 론

프로그래밍 비숙련자에 의한 게임 구현을 통하여 집계한 오류 요인들을 중심으로 기존 방법에 대한 보완 요소들을 도출하고, 보완 기능을 수행하는 메서드들을 제공함으로써 오류발생 감소를 통한 평균적 구현 효율성이 향상될 수 있다는 결과를 얻은 이 연구에는 물론 한계점도 존재한다. 실험 대상의 개인별 편차 분석이나 제공된 메서드에서 기인한 부작용 분석, 실험에 사용한 게임의 대표성 검증 등이 필요할 것으로 판단된다. 실험 인원과 게임 종류를 확대하고, 오류 요인을 더 세분화한 분석과 함께 다양한 게임 통합개발편집기의 기능 비교를 통한 표준화 연구도 선행되어야 한다.

그렇지만 전술된 바와 같이 다양한 분야에서 활용도가 커지고 있는 '쉬운 게임 프로그래밍 대안'과 '절차는 이해하였지만 구현 과정에서 발생하는 오류를 효과적으로 회피할 수 있는 방법' 문제들은 게임 관련 학과의 교육 과정에서도 항상 고민되고 개선되어야 하는 현실적 이슈이기도 하다. 이에 대한 국내의 기반 연구가 부족한 상황에서, 이 논문에서 제안된 보완 요소는 기존 개발체계의 개선이나 새로운 게임 스크립트의 설계에 유의미한 역할을 할 수 있을 것으로 기대한다.

ACKNOWLEDGMENTS

This research is supported by Jangan University Research Fund in 2014.

REFERENCES

[1] Lim, C. J. and et al., “Educational Game Making-Tool Development using Unity3D Engine: Birth of Game”, Journal of Korea Game Society 2014 Feb; 14(1): pp.29-38, 2014.

[2] Kim, Y. J. and Seo, Y. H., “Authoring Tool for Robot Simulation Creation using Functional Concise Script”, Journal of Internet, Broadcasting and Communications, p.166, 2013.

[3] Anderson, E. F., “A Classification of Scripting Systems for Entertainment and Serious Computer Games”, Proceedings of VS-GAMES 2011: 3rd International Conference on Games and Virtual Worlds for Serious Applications, pp.47-54, 2011.

[4] Husmann, H., “Realtime Interactive Architectural Visualization using Unreal Engine”, Ludwig-Maximilians University, Munich, 2013.

[5] Sweigart,A.,“Making Games with Python & Pygame”,http://inventwithpython.com/pygame, 2012.

[6] Sorva, J. and et al.,“A Review of Generic Program Visualization Systems for Introductory Programming Education”, ACM Transactions on Computing Education, Vol.13, No.4, Article 15, 2013.

[7] Moons, J. and De Backer, C.,“The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism”, Comp. Educ. 60, 1, pp.368-384, 2013.

[8] Kirby, S. and et al., “Program visualization tool for teaching programming in C”, In Proceedings of the International Conference on Education, Training and Informatics (ICETI'10), 2010.

[9] Wang, P. and et al., “During automatic program animation, explanations after animations have greater impact than before animations”, In Proceedings of the 12th Koli Calling International Conference on Computing Education Research (KoliCalling'12), pp.100-108, 2012.

[10] Guo, P. J., “Online Python tutor: Embeddable web-based program visualization for CS education”, In Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE'13), 2013.

[11] Ebner, M. and et al., “Towards a Video Game Description Language”, Artificial and Computational Intelligence in Games, 2013.

[12] Schaul, T., “A video game description language for model-based or interactive learning”, Computational Intelligence in Games (CIG), 2013 IEEE Conference, 2013.

정 찬 용 (Jung, Chan Yong)

(현)장안대학교 IT학부 게임컨텐츠과 조교수
(현)한국게임학회 프로그래밍 분과위원장
(전)삼성 SDS 정보기술연구소

관심분야 : Gamification, 기능성게임, 게임언어