# 초급 프로그래머를 위한 게임 프로젝트 환경

이주호, 이인환
한양대학교 컴퓨터공학부
e-mail: racin@hanyang.ac.kr

# Game Project Environment for Entry-Level Programmers

Juho Lee, Inhwan Lee
Dept. of Computer Science & Engineering, Hanyang University

**요 약**

본 논문에서는 초급 프로그래머를 위한 게임 프로젝트 환경을 개발한다. 이 환경은 초급 프로그래머의 구현 능력 향상을 목표로 이들의 현재 실력에 적합한 2D 좌표 연산을 핵심 구현 과제로 제시한다. 이 환경을 실제 수업에서 운용해 본 결과, 학생들이 프로젝트 진행 과정에서 스스로 다양한 시도를 통해 게임을 개량하려 하는 것을 관찰할 수 있었으며, 그에 따른 결과물의 우수성 및 학생들의 만족도 또한 높게 나타났다.

## 1. 서론

프로그래밍은 사고를 구현하여 문제를 해결하는 것이다. 사고는 주어진 문제에 대해 목표에 가장 적합한 안을 선택하는 것이며, 구현은 사고의 결과를 코드로 표현하는 것이다. 좋은 프로그래머가 되려면 사고에 필요한 논리력과 기반 지식, 구현에 필요한 코드 제어 능력을 모두 갖추어야 한다.

현재 대학에서는 학부 1, 2학년에 해당하는 초급 프로그래머들에게 다양한 수업을 통해 기반 지식을 제공하고 있다. 그러나 이에 비해 구현 능력에 대한 훈련은 상대적으로 충분히 이루어지지 않고 있다. 적지 않은 고학년 학생들이 '이해는 되지만 어떻게 코드로 작성해야 할 지 모르겠다'는 고충을 제기하는 것에서 이러한 현상을 확인할 수 있다.

초급 프로그래머에게 게임 프로젝트는 이러한 구현 능력 부족 문제에 대한 좋은 대안이다. 프로젝트는 목표 수립, 문제 가시화, 최적안 선택을 스스로 수행한 다음 구현에 임한다는 측면에서, 단순히 정해진 문제에 대한 답안 코드를 작성하는 일반적인 과제에 비해 더 포괄적인 프로그래밍 경험을 제공한다. 그리고 프로젝트 주제를 게임으로 설정하는 것은 '재미'를 주요 목표로 둠으로써 성능, 유연성 등을 추구할 때에 비해 프로그래밍 기반 지식의 필요성을 낮출 수 있으므로, 특히 초급 프로그래머들이 보다 능동적으로 프로젝트에 참여할 수 있게 해 준다. 물론, 초급 프로그래머가 게임을 구성하는 모든 요소를 직접 구현할 수는 없다. 따라서 이들이 더 큰 재미를 추구할 수 있으면서 동시에 구현 능력을 기를 수 있게 하려면, 이들의 현재 실력에 맞춘 적당한 수준의 게임 프로젝트 환경이 필요하다.

본 논문은 초급 프로그래머가 자신의 실력에 적합한 프로그래밍 문제에 몰두할 수 있게 하는 게임 프로젝트 환경을 설계한다. 먼저, 기존 환경들을 분석하여 이들이 어떤 사용자 계층을 위해 어떤 구성을 선택했는지 살펴보며, 이를 토대로 새로운 환경이 갖추어야 할 기능적 특성을 도출한다. 그 다음, 초급 프로그래머의 관점에서 게임 제작 도중 경험하는 여러 문제들을 그 난이도에 따라 분류하기 위한 기준을 제시하고, 각 영역별 문제들에 대해 미리 고려해야 할 사항들을 확인한다. 논문의 후반부에서는 이러한 연구 결과를 토대로 개발한 코어 라이브러리와 이를 실제 수업에 적용한 사례를 소개한다.

## 2. 기존 환경 분석

초급 프로그래머를 위한 게임 프로젝트 환경이 갖추어야 할 기능적 특성을 도출하기 위해 게임 개발에 주로 사용되는 DirectX[1], Scratch[2], Unity[3]를 비교 및 분석하였다. 어떤 환경의 특징을 파악하는 좋은 방법은 이들이 입문자에게 제시하는 첫 번째 튜토리얼을 살펴 보는 것이다. 표 1은 위의 세 환경에서 2D 게임을 만들기 위한 첫 튜토리얼의 제목과 앞 세 단계를 보여준다.

<표 1> 기존 환경들의 첫 튜토리얼 비교

| 환경 | 내용 |
|---|---|
| DirectX | Drawing a Simple Rectangle<br>1: Include Direct2D Header<br>2: Create an ID2D1Factory<br>3: Create an ID2D1HwndRenderTarget |
| Scratch | Getting Started with Scratch<br>1: Start Moving<br>2: Add a Sound<br>3: Start a Dance |
| Unity | 2D Game Development Walkthrough<br>1: Import background layers from Photoshop<br>2: Set foreground collider area<br>3: Arrange character animations |

DirectX는 처음부터 API를 사용하기 위한 구체적 코드 작성 절차를 설명하고 있다. DirectX는 사용자가 C/C++에 대한 사전 지식을 보유하고 있음을 가정한다. 또한, DirectX로 게임을 만들기 위해서는 간단한 사각형 그리는 방법을 토대로 하나의 게임을 구성할 수 있는 수준의 구현 능력이 요구된다.

이에 반해 Scratch는 게임 제작에 필요한 대부분의 요소가 미리 구성되어 있다. 처음 환경을 구동하면 화면 중앙에 캐릭터 하나가 생성되어 있어 즉시 튜토리얼을 시작할 수 있다. Scratch의 튜토리얼은 이 캐릭터의 행동을 구성하는 방법, 지시하는 방법 등을 상세히 설명하고 있다. 이 과정에서 사용하는 모든 요소들은 'move X steps', 'play drum A for X beats'와 같은 블록으로 제공되므로 프로그래밍 경험이 없는 사용자도 간단한 마우스, 키보드 조작을 통해 게임을 만들 수 있다.

Unity는 위의 두 환경과 달리 제목에서부터 '게임 개발'을 언급하고 있다. 튜토리얼은 실질적인 게임 제작 순서에 따라 진행된다. 각 단계에서는 특정 작업을 손쉽게 수행하기 위해 Unity가 제공하는 여러 도구들을 소개하며, 이들을 다른 프로그램과 함께 사용하는 방법을 제시한다. Unity는 사용자가 기본적 코드 작성 능력과 함께 게임 내 필수 요소인 배경 레이어, 충돌 영역, 캐릭터 애니메이션 등의 개념을 이미 이해하고 있으며, 자신이 만들 게임의 전반적 구조를 설계할 수 있다는 것을 전제하고 있다.

각 환경이 가진 특징을 종합해 보았을 때, 초급 프로그래머를 위한 게임 프로젝트 환경이 갖추어야 할 기능적 특성은 다음과 같다. 먼저, 초급 프로그래머에게 적절한 난이도의 요소는 DirectX와 같이 실제 코드를 통해 구현하도록 해야 한다. API 사용 방법 및 주요 개념을 구현하기 위한 접근 방법은 Scratch와 같이 상세한 소개를 제공해 줌으로써, 초급 프로그래머들이 게임 제작에 활용할 기초 지식을 마련해 주어야 한다. 이 때 구현 목표와 관련이 적은 기능(이미지, 오디오 등)은 환경에 포함하지 않고 Unity와 같이 외부 프로그램을 활용하는 것을 고려해 볼 수 있다.

## 3. 게임 프로젝트 내 프로그래밍 문제 분석

초급 프로그래머가 스스로 해결할 수 있는 문제를 특정하려면, 먼저 이들이 프로젝트 진행 과정에서 어떤 문제에 부딪히는지를 알아야 한다. 지난 4년간 초급 프로그래머들만으로 구성된 40여 팀의 게임 프로젝트를 지도하면서, 우리는 이들이 각각 다른 게임을 만들면서도 서로 비슷한 문제를 제기하는 것을 관찰했다. 그리고 이러한 문제들을 자의적 해결 가능성, 필요한 도움 요소 등에 따라 분류하면, 학생들의 도움 요청에 대한 지원이 용이하고, 나아가서는 이들이 진행하면 겪을 문제에 미리 대비하거나 이들이 특정 문제에 많은 시간을 고민하도록 환경을 구성할 수 있다는 점을 확인했다. 표 2는 이러한 시도의 결과로서 초급 프로그래머가 게임 프로젝트를 수행하며 제기하는 문제들을 그 원인 요소에 따라 분류한 것이다.

<표 2> 문제의 원인 요소에 따른 분류 기준

| 영역 | 내용 |
|---|---|
| 창작 | 어떤 게임인지를 특정하는 핵심 개념 |
| 구성 | 게임의 논리적 골격 |
| 활용 | 게임이 진행되기 위한 기반 요소 |
| 기계 | 적어도 이번에는 고려하지 않아야 할 것들 |

창작의 영역은 만들려는 게임의 특징을 자신 및 타인에게 설명하며 프로젝트의 목표를 설정하는 영역이다. 구성의 영역은 이러한 특징을 실제로 반영하기 위해 반드시 직접 구현해야 하는 요소들이 포함된다. 활용의 영역은 직접 구현하기 어려워서 미리 환경에 포함하여 제공하는 요소들을 간단한 설정 조작을 통해 그대로 가져와 사용하는 영역이다. 마지막으로 기계의 영역은 실제 하드웨어를 포함하여 초급 프로그래머가 현 수준에서 거의 이해할 수 없는 요소들이 포함된다.

이러한 네 영역들은 수직적 계층 관계를 형성하며 이들 중 실질적으로 구현 능력을 훈련하는 곳은 구성의 영역이다. 게임 진행에 필요한 데이터를 기록하고 활용하는 방법, 특정 상황 발생 유무를 검사하여 게임의 흐름을 제어하는 방법 등을 직접 고민하고 시도해 보는 것은 매우 유익하다.

초급 프로그래머는 아직 자료구조론, 알고리즘 관련 지식이 부족하므로 구성의 영역에서 세부 구현 방법을 착안하고 결정하는 능력이 떨어진다. 따라서 상위 및 하위 계층인 창작, 활용의 영역을 넓힘으로써 게임 구성의 복잡도를 줄일 필요가 있다. 구체적으로, 구성의 근거가 되는 창작의 영역에서는 '장애물 피하기'와 같은 단순한 목표를 '점점 빠른 속도로 달리는 캐릭터를 점프시켜 무작위로 나타나는 풍선을 터뜨리지 않고 최대한 멀리 달리는 게임'과 같이 상세하게 설정해 두어야 한다. 구성의 기반이 되는 활용의 영역에서는 환경이 제공하는 라이브러리의 각 기능 및 설정들을 미리 확인하고 조작해 보며, 자신이 만들 게임이 어떻게 동작하게 될 것인지, 특정 구성 문제를 해결하기 위해 어떤 기능을 사용할 수 있는지 등을 파악해 두어야 한다. 위에서 살펴 본 각 영역별 핵심 문제를 요약하면 표 3과 같다.

<표 3> 영역별 핵심 문제

| 영역 | 핵심 문제 |
|---|---|
| 창작 | 어떤 게임을 만들 것인가<br>구체적으로, 무엇을 어떻게 하는 게임을 만들 것인가 |
| 구성 | 어떤 정보를 어떻게 담아 두고 어떻게 활용할 것인가<br>어떤 상황을 어떻게 감지하고 어떻게 처리할 것인가 |
| 활용 | 내 게임에 가장 적합한 라이브러리 설정은 무엇인가<br>내 게임이 라이브러리의 어떤 기능을 사용해야 하는가 |
| 기계 | (고려하지 않음) |

창작의 영역에 해당하는 문제들은 대부분 프로젝트 초기에 미리 제기되나 이 때의 결정은 프로젝트의 전체적 목표 및 전반적 흐름을 결정짓는 중요한 요인이 된다. 따라서 미리 충분한 검토를 거쳐 이들이 '완성

할 수 있는 프로젝트'를 진행하도록 할 필요가 있다.

구성의 영역에서, 대부분의 초급 프로그래머는 한 번에 이해할 수 있는 코드의 길이가 짧다. 이를 감안하여 이들이 가급적 단순 구문만으로 게임 논리를 구현할 수 있도록 복잡한 API는 피하고 성능상 선택이 필요한 요소들은 가급적 활용의 영역으로 낮추는 편이 좋다. 적지 않은 학생들이 스스로 구성한 게임 구조에 한계를 느끼게 되며, 이 때는 강사가 직접 해당 요인을 검토하여 부분적 또는 전체적 재설계를 지도해 주어야 한다.

어떤 요소가 활용의 영역과 기계의 영역 중 어디에 속할지를 결정하는 큰 요인 중 하나는 이 요소의 설정 변화가 전체 프로젝트에 미치는 영향력이다. 그만큼 활용의 영역에 해당하는 요소들은 그 설정 변화가 코드 또는 실제 게임에 명백한 변화를 줄 수 있어야 하며, 이러한 변화를 직접 테스트하며 각 설정 및 기능의 의미를 인지할 수 있도록 충분한 양의 샘플 코드를 제공해 주어야 한다.

기계의 영역은 초급 프로그래머가 현 시점에서 이해할 수 없는 부분이나, 일부 학생들은 이러한 인프라 요소에 흥미를 가지며 추가적인 학습을 희망한다. 따라서 하위 플랫폼이 허용하는 한도 내에서 이 영역에 해당하는 코드 및 그에 대한 자세한 설명을 덧붙여 제공하는 것을 고려해 볼 수 있다. 이러한 영역별 주요 고려사항을 정리하면 표 4와 같이 나타낼 수 있다.

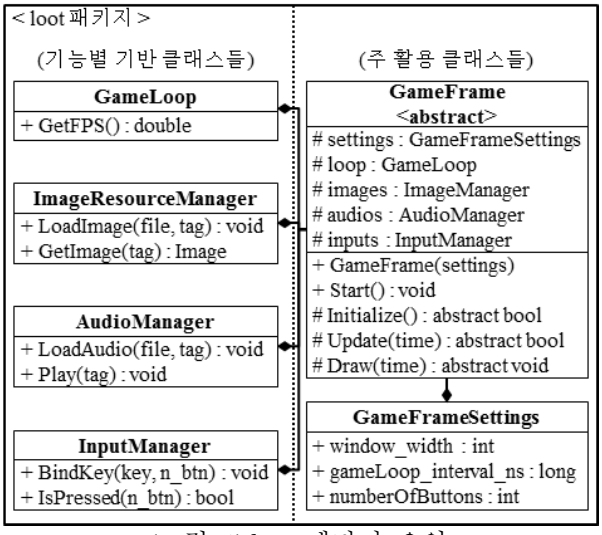
<표 4> 영역별 주요 고려사항

| 영역 | 주요 고려사항 |
|---|---|
| 창작 | 프로젝트 초기에 사전 검토 및 조율을 수행하여<br>프로젝트의 완성 가능성 확보 |
| 구성 | 단순 구문 위주의 실제 코드 사용,<br>필요한 경우 전체 프로그램 구조 검토 및 재설계 지도 |
| 활용 | 직관적이고 즉각적인 설정 인터페이스 제공,<br>샘플을 통해 각 설정 및 기능의 의미 설명 |
| 기계 | 가급적 쉬운 알고리즘을 적용하여<br>원하는 경우 코드를 읽으며 이해를 시도할 수 있도록 |

## 4. 코어 라이브러리 설계 및 구현

위의 사전 분석을 바탕으로 우리는 학부 2학년 1학기 객체지향프로그래밍 수업에서 팀 단위 게임 프로젝트를 운용하기 위한 LOOT(Library Of OOP Term-Project)를 설계하였다. LOOT는 학생들의 편의를 위해 수업에서 사용하는 플랫폼인 JavaSE 1.7에서 Eclipse를 통해 프로젝트를 진행하도록 설계되었다.

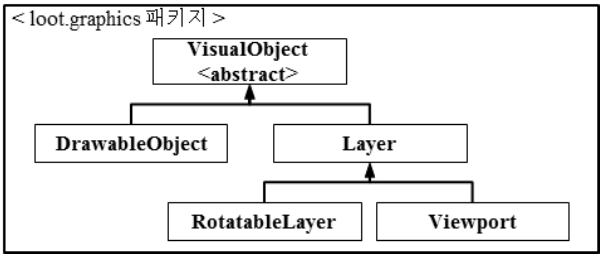
LOOT에서 구성의 영역에 해당하는, 학생들이 직접 코드로 구현해야 하는 부분은 2D 공간 내 좌표 연산이다. 구체적으로, 시간 흐름에 따른 요소의 위치 및 크기 제어, 요소간 거리 비교를 통한 조건 검사를 핵심 구현 목표로 제시한다. 이러한 접근이 갖는 장점으로는, 창작의 영역에서 물리 상식에 기반한 더 세세한 구성 계획을 세울 수 있는 점, 해당 계획에 따라 특정 물리 현상을 구현하기 위한 다양한 수식을 도출하고 적용 및 비교해 볼 수 있는 점, 구현의 결과가 게임 화면에 직관적으로 반영되므로 코드 변화에 따른 피드백을 빠르게 얻을 수 있는 점을 들 수 있다.



(그림 1) loot 패키지 요약

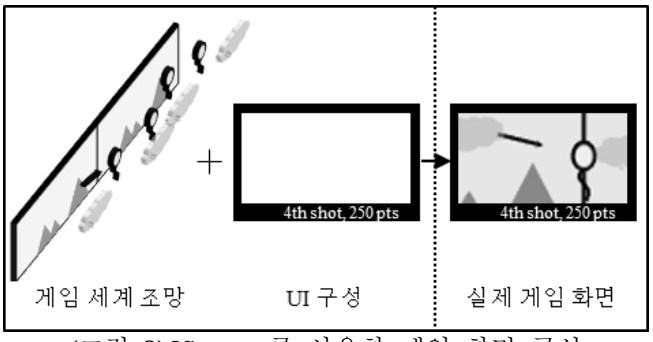
LOOT는 게임 기반 요소들을 미리 구현해 둔 loot 패키지와 2D 기반 시각적 요소들이 계층 구조로 정의되어 있는 loot.graphics 패키지로 구성되어 있다. 그림 1은 loot 패키지에서 활용의 영역에 해당하는 중요 필드와 메서드를 클래스별로 요약하여 보여준다. 그림의 좌측에는 각각 실행 흐름, 이미지 파일, 오디오 파일, 입력 장치를 제어 및 관리하기 위한 클래스들이 나열되어 있다. 우측의 GameFrame 추상 클래스에는 이러한 기반 요소들을 포함하여 게임 실행에 필요한 모든 내부 논리들이 미리 구현되어 있다. 마지막으로, GameFrame 클래스의 생성자 인수로 사용되는 GameFrameSettings 클래스는 여러 내부 요소들에 대한 설정 필드들을 가지고 있다.

loot.graphics 패키지에는 '화면에 그려질 수 있는' 요소들이 그림 2와 같은 계층 구조로 정의되어 있다. 최상위 클래스인 VisualObject 추상 클래스는 해당 요소의 x, y, z좌표 및 너비/높이를 나타내기 위한 double 형식 필드들을 정의한다. (여기서 z좌표값은 일반적으로 원근감 표현을 위해서만 사용되며 실제 게임 논리에서는 거의 쓰이지 않는다.) 또한, 각 하위 클래스에 따른 실제 렌더링 작업을 정의하기 위한 추상 메서드를 선언한다. 이를 상속받는 DrawableObject 클래스에는 실제로 그릴 이미지를 담기 위한 필드가 추가되며, DrawableObject 클래스는 학생들이 창작의 영역에서 생각해 둔 캐릭터, 장애물과 같은 게임 내 요소를 실체화하기 위한 기본 클래스가 된다. Layer 클래스 및 그 하위 클래스들은 자식 요소 목록과 함께 하나 이상의 '내부 좌표계'를 가지고 있으며, 자신이 속한 외부 좌표계와 자식 요소들을 다루는 내부 좌표계 사이의 선형 변환기능을 제공한다. 이들 중 가장 간단한 구조를 가진 Layer 클래스는 기본적인 평행 이동 및 확대, 축소 변환만을 제공하며, 전체 게임 화면 구성과 내부 좌표 연산을 통한 게임 논리 구성을 따로



(그림 2) loot.graphics 패키지 내 클래스 계층 구조

진행할 수 있게 해 준다. RotatableLayer 클래스는 여기에 회전 변환을 추가로 제공하여 날아가는 화살, 바람개비와 같은 몇몇 요소들을 보다 다루기 쉽게 해 준다. 가장 복잡한 Viewport 클래스는 전체 게임 화면 구성에 사용된다. 이 클래스는 항상 화면 가장 위에 표시되는 UI 등을 표현하기 위한 2D 내부 좌표계와 전체 게임 세계를 조망하기 위한 3D 내부 좌표계를 함께 사용한다. Viewport 클래스가 다루는 3D 공간은 단순히 Z축에 수직인 평면들의 집합이며, Viewport 클래스는 이러한 공간 내에서 원하는 위치를 실제 게임 화면에 투사하기 위해 간단한 시점 조작 기능을 제공한다. 학생들은 실제 게임이 진행되는 2D 공간을 하나의 Layer 인스턴스로 다루면서 구름, 산과 같은 배경 요소들을 앞 또는 뒤에 배치함으로써, 입체적인 장면을 보다 손쉽게 구성할 수 있다. 그림 3은 이러한 Viewport 클래스의 기능을 보여준다.



(그림 3) Viewport를 사용한 게임 화면 구성

학생들은 새 게임을 만들기 위해 먼저 GameFrame 클래스를 상속받는 새로운 클래스를 정의하게 된다. main()은 새 클래스의 인스턴스를 적절한 설정값을 통해 생성하고 Start()를 호출한다. 실제 게임을 구성하는 논리는 Initialize(), Update(), Draw()의 세 가지 추상 메서드를 새 클래스에서 오버라이드함으로써 구현하게 된다. 학생들은 DrawableObject를 상속받는 새 클래스 정의, Update() 구성에 대부분의 시간을 할애하게 되며 이 과정에서 2D 좌표 연산을 중심으로 한 단순 구문 위주의 코드를 재차 작성하며 코드 작성 능력을 훈련하게 된다.

## 5. 적용 및 결과

실제 수업에서는 창작의 영역에서 전체 프로젝트의 완성 가능성을 확보하기 위해 미리 팀별 면담을 통해 각 팀의 게임 구상을 검토하였다. 이 과정에서 모든 팀이 비슷한 난이도의 완성 가능한 목표를 가질 수 있도록 조율하고 이를 모든 학생들에게 공지하였다. 또한, 여러 팀이 공통적으로 사용하는 개념에 대한 예제 코드를 추가로 작성하여 제공하였다.

프로젝트 진행 단계에서 가장 먼저 지시한 것은 '외견만 완성한 프로토타입' 제작이었다. 대부분의 팀은 이를 제대로 완성하지 못했으나, 이 과정을 토대로 초기 목표들 중 난점이 무엇인지 짚고, 이에 대한 구체적 대책을 찾아 적용해 가면서 전체 게임을 다시 구현하기 시작했다.

'성공이 약속된, 여러 번 구현하는' 이러한 분위기 아래에서, 학생들은 게임이 완성된 이후에도 좀 더 마음에 드는 게임을 만들기 위해 계속 새로운 시도를 수행하며 개량해 나갔다. 그 결과는 고스란히 최종 발표로 이어져, LOOT를 사용하지 않았을 때에 비해 더 뛰어난 게임들을 소개하고 관람할 수 있었고, 이를 직접 제작한 학생들의 성취도 또한 그만큼 높게 관찰되었다.

이러한 긍정적 효과에는, LOOT가 학생들에게 요구하는 코드 작성 난이도가 낮아 각 코드 부분의 의미를 파악하기 쉬웠던 점, 2D 좌표 연산을 주 구현 목표로 설정함으로써 게임 화면을 통해 코드 오류를 거의 즉각적으로 감지하고 수정할 수 있었던 점, 마지막으로, 전체 프로젝트 수행에 대한 사전 고찰을 토대로 학생들이 진행 과정에서 접하는 다양한 상황들을 미리 예측하여 대비하고 대응할 수 있었던 점이 큰 영향을 주었다고 판단된다.

## 6. 결론

본 논문에서는 초급 프로그래머의 구현 능력 향상을 목적으로 하는 게임 프로젝트 환경을 만들기 위한 연구를 수행하고, 이를 토대로 학부 2학년 학생들의 현재 실력에 적합한 2D 좌표 연산을 핵심 구현 과제로 제시하는 LOOT를 개발하였으며. 이를 실제 수업에 적용하여 그 효과를 확인하였다. 창작의 영역에 속하는 게임 구상은 프로그래밍 기반 지식을 거의 요구하지 않는다. 따라서 본 논문의 접근 방법을 응용하여 비전공 프로그래머 양성 과정, 교양 교과 과정 등을 위한 게임 프로젝트 환경을 개발, 운용하는 것 또한 고려해 볼 수 있을 것이다. 이를 위해 다양한 계층의 학생들을 직접 지도하며 이들이 직접 구현해 보기 적합한 프로그래밍 문제들을 분석, 정리할 계획이다.

## 참고문헌

[1] Microsoft, "Getting Started with DirectX Graphics", http://msdn.microsoft.com/en-us/library/windows/desktop/ hh309467.aspx
[2] Lifelong Kindergarten Group, "About Scratch", http://scratch.mit.edu/about/
[3] Unity Technologies, "Unity Overview", http://docs.unity3d.com/Manual/UnityOverview.html