논문 투고일:2018.01.16, 논문 심사일:2018.01.16-2018.02.21, 게재 확정일:2018.02.21

그래픽과 멀티미디어 사고를 접목한 어린이 코딩교육 융합 커리큘럼 개발 연구

Developing Integrated Curriculum for Coding Education with Graphic Thinking and Multimedia Designing

주저자
이현진 (Lee, Hyun Jean)
연세대학교 커뮤니케이션대학원
hyunjean@yonsei.ac.kr

이 논문은 2016학년도 연세대학교 연세미래선도연구사업 지원에 의하여 작성된 것임(2016-22-0098).

목차

(요약)

오늘날 코딩교육은 컴퓨팅 사고력 함양이란 목적 하에 컴퓨터 과학의 세부 교육으로 집중되고 있다. 하지만 어린이 코딩교육은 문제해결력, 추상화능력, 추론과 논리적 사고력 등을 통해 어린 학습자들에게 더 큰 학습능력을 갖추기 위한 교육적 목표를 갖는다. 본 연구는 융합교육과 미디어 리터러시에 대한 관심에서 출발하여, 스크래치를 포함한 많은 코딩교육 초기학습환경이 시각적 프로그래밍 언어에 기반하고, 게임, 애니메이션 등 멀티미디어 프로젝트 제작 중심의 커리큘럼을 통해 구성되는 측면에 주목한다. 이는 코딩교육이 그래픽사고와 함께 통합적으로 진행될 수 있는 가능성을 내포한다. 본 연구는 여러 선행연구를 통해 어린 학습자 인지발달 과정과 연계, 개발된 코딩 커리큘럼을 검토하며, 이러한 학습과정 안에 그래픽사고가 접목되어 프로젝트 완결을 위한 문제해결력과 종합적 사고를 위하여 학습될 수 있는 내용을 논한다. 투명도, 레이어, 프레임 개념, 화면 구성과 그래픽 파일에 대한 이해 등은 코딩교육의 융합적 접근을 위한 구체적 교육 내용으로 제안된다.

(Abstract)

Coding education often tends to focus on developing computer science concept. However computational thinking can be taught widely to develop the basic thinking skill for young learners, such as problem solving, abstraction, reasoning and logical thinking. With an interest in convergence education and media literacy, this study focuses on how to develop integrative curriculum for early coding education which visual thinking can co-evolve with a programming language. Initial Learning Environments, such as Scratch, is based on the visual programming interface and language and takes project-based approaches such as games and animations making. Thus I argue that the educational effects can be greater when such approaches are integrated with graphic thinking skills. After investigating previous studies about the coding curriculum based on the young learners' cognitive process, how the integrated curriculum can incorporate graphic thinking is discussed. In detail, how visual thinking such as transparency and layer, frame, and screen composition can be taught to enhance the problem solving and general thinking skills is discussed. Through this, the study suggests a convergent curriculum that can support this.

(Keyword)

Coding education, Graphic thinking, Computational thinking, Convergence

1. 서론

1-1. 연구의 배경

오늘날 어린이 코딩교육에 대한 관심이 상당하다. 세계 여러 나라들은 정부와 기업, 민간의 노력을 통해 다양한 정규 교육과정 및 클럽 등 사회적 커뮤니티 활동을 통해 다양한 코딩교육 기회를 마련하고 있다. 미국의 Code.org, Code Club, CoderDojo, Black Girls Code, Codecademy, Code Avengers, CodeHS and MotherCoders 등과 영국의 Micro:bit, Code Club[1] 등이 그러한 예이다. 우리나라도 정부가 발표한 '2015 개정 교육과정'에서는 문, 이과 통합 및 소프트웨어(SW) 교육을 중학교 교육과정에서의 필수과목으로 지정하였으며, 2019년부터는 초등학교 5~6학년 과정에도 코딩교육과정이 들어가게 된다.[2] 또한 기업체인 Naver, 비영리 교육기관인 엔트리(Entry, https://playentry.org/#!/), 커뮤니티 그룹인 Coding Club(http://codingclubs.org/) 등 여러 기업과 사회 기관 및 커뮤니티에서 코딩교육을 위한 다양한 노력을 펼쳐나가고 있으며, 최근 들어 사교육 시장도 연계하여 번성하고 있다.

이처럼 코딩교육의 중요성이 부각되고 교육적 접근이 다양화 되면서, 코딩교육이 중요하게 인식되는 본질적 의미와 가치를 재점검할 필요성이 대두된다. 흔히 코딩교육은 컴퓨팅 사고(computational thinking, 이하 CT)를 함양시키기 위한 접근으로, 컴퓨터 과학(computer science, 이하 CS)의 세부 교육 과정으로 인식되고 있다. CT의 많은 개념이 프로그래밍 교육을 통해 학습되고 향상될 수 있다는 것은 분명한 사실이며, 실제 많은 초기학습환경들(Initial Learning Environments, ILEs)이 CT를 장려하도록 개발되고 있다. 그러나 본 연구는 CT 교육이 비단 컴퓨터 과학적 사고신장과 프로그래밍 자체에 대한 교육만으로 강조되기 보다는, 이를 통해 추구되는 문제해결력, 추상화능력, 추론적 사고력 혹은 논리적 사고력 등이 좀 더 폭넓고 종합적이며 또한 융합적으로 강조되어야 함을 주장하고자 한다. 이는 CS 혹은 CT 교육만을 중심으로 한 접근이 최근 비판적으로 논의되고 있으며 재검토되어야 한다고 주장하는 논의들과 맥을 같이 한다.[3] 이러한 논의들은 어린 학습자들에게 기본적인 컴퓨팅 기술능력으로써 코딩을 가르치는 데서 출발하여 컴퓨팅적 리터러시(computational literacy) 함양, 그리고 코딩을 통하여 배움 그 자체의 과정에 다가가게 하고자 하는 흐름을 강조한다. 이는 코딩교육, 특히 어린 학습자를 대상으로 하는 코딩교육이 가지는 보다 근본적인 이유를 되새기고 그 방향성을 분명히 하고자 하는 노력에 다름 아니며, 코딩교육을 통해 학제간 융합적 커리큘럼을 고찰하고 있는 다양한 연구자들에게도 주목될만한 논의이다.

1-2. 연구와 범위 및 방법

본 연구자는 융합 교육과 미디어 리터러시 교육에 대한 관심에서 출발하여 어린이 코딩교육 현장에서 학습자 관찰과 인터뷰를 진행하면서 어린이 코딩교육의 현재 방향성이 보다 확장될 필요성을 공감해왔다. 또한 이러한 관심의 연장선상에서 코딩교육을 위한 여러 초기학습환경들, 즉 코딩에 입문

---

1) 영국의 경우 초등학생 때부터 코딩을 가르치는 시도가 민간에서도 활발하며, 현재 1300여 초등학교에 방과후 교육과정을 무료로 운영 중인 '코드클럽'이 대표적이다. 코드클럽은 웹 디자이너 클레어 서트클리프(Clare Sutcliffe)와 웹 프로그래머 린다 샌드빅(Linda Sandvik)이 지난 2012년 4월 만든 비영리단체다. 방과후 코딩교육을 원하는 학교에 무료로 클럽을 개설해 주고 자원봉사자들이 교사로 뛴다. 서트클리프는 학교에서 코딩을 가르치지 않는 것을 걱정했고 9~11살 아이들에게 코딩을 배울 수 있는 기회를 주자는 단순한 아이디어에서 시작했다"고 말했다. http://www.zdnet.co.kr/news/news_view.asp?artice_id=20131209220219

2) 초등학교 5~6학년 실과 교과서에 실리게 되는 '소프트웨어 기초소양'의 세부 내용으로 '소프트웨어의 이해' '절차적 문제 해결' '프로그래밍 요소와 구조' 등이 포함된다. 초등학교에서 소프트웨어 기초소양을 다뤘다면, 중학교 정보 교과는 알고리즘과 프로그래밍에 관한 내용으로 구성된다.

3) Hsiu-Ying Wang, Iwen Huang, Gwo-Jen Hwang(2014)은 컴퓨터 프로그래밍 교육이 대부분 프로그래밍 언어의 구문과 프로그래밍 스킬을 가르지는 것에 집중하는 반면, 문제해결력은 종종 무시된다고 말하며, 프로그래밍 교육의 핵심은 구문을 암기하고 프로그래밍 툴을 조작법을 익히는 것이 아니라 문제해결력을 함양시켜야 함에 목적이 있다고 주장한다. 또한 Caitlin Duncan, Tim Bell & Steven L. Tanimoto(2014) 역시 어린학습자를 대상으로 한 코딩교육의 의미를 비판적으로 검토하며, 코딩교육의 본질에 대하여 짚고 있다.

하는 학습자들을 위한 교육용 툴과 언어들이 시각적 프로그래밍 언어(visual programming language)에 기초하고 있으며, 많은 경우 게임과 애니메이션 등 멀티미디어 및 그래픽 프로젝트 제작을 중심으로 하여 진행되는 점에 주목한다. 이는 코딩교육을 통해 추구될 수 있는 전반적인 사고에 대한 접근이 그래픽 사고 교육과 함께 통합적으로 진행될 수 있는 가능성을 발견하게 한다. 그리고 코딩교육의 학습효과 향상과 함께, 보다 융합적이고 통합적인 교육적 방법론에 대한 가능성과 기대, 그리고 이를 위한 새로운 커리큘럼 개발의 필요성을 발견한다.[4] 본 연구에서는 이러한 인식을 바탕으로, 스크래치와 같은 어린이 코딩교육 커리큘럼 안에 시각적이고 멀티미디어적 사고가 통합된 커리큘럼 개발을 위한 구성 내용으로 어떤 것들이 다루어질 수 있을지 논의하고자 한다. 이를 위해 먼저 코딩교육 초기학습환경으로서의 스크래치에 대하여 간단히 분석한 후, 어린 학습자 인지과정을 바탕으로 코딩교육 초기학습환경 커리큘럼을 고찰 검토하는 선행연구들을 검토할 것이다. 또한 이 중 스크래치에서의 코딩교육과 연계되어 학습자 인지과정에 기반 한 시각적 사고와 디지털 그래픽 사고 및 멀티미디어적 사고가 융합된 커리큘럼이 어떠한 내용을 담으면 좋을지 구체적 논의를 진행하고자 한다. 이는 향후 실제적인 커리큘럼을 구성하기 위한 기초자료로서 활용될 수 있을 것이다.

2. 초기학습환경으로서의 스크래치와 관련 커리큘럼 선행연구 검토

2-1. 코딩교육을 위한 초기학습환경으로서의 스크래치

어린이 코딩교육에 대한 관심이 증가하면서 이를 교육시키기 위한 다양한 툴, 온라인 플랫폼, 혹은 코딩교육용 초기학습환경(Initial Learning Environment, ILE)이 함께 개발되며 접근되는 추세다. 앞서 말한 BBC micro:bit[5], Code.org 싸이트 등이 그러한 예이며, Tynker[6]와 우리나라에서 개발된 Entry 등도 그러한 예이다. 이들은 학습자가 쉽게 코딩에 대해 이해할 수 있도록 돕는 한편, 코딩에 즐겁게 입문하도록 이끄는데 중점을 두고 있다. 과거에도 코딩교육을 위한 학습환경으로써 Logo[7], Alice[8] 등이 개발된 바 있다. 이들은 스토리텔링이나 게임 애니메이션들을 위한 3D 모델들을 사용하여 코딩교육을 이끄는 특징을 지녔다. 오늘날 코딩교육을 위하여 개발된 초기학습환경들도 이러한 멀티미디어 프로젝트 제작과 함께 코딩 교육을 유도하는 특징을 유지하는 경우가 많으며, 그 중 오늘날 가장 널리 이용되며, 본 논의에서 중점적으로 다루고자 하는 스크래치도 그러한 예이다.

스크래치(http://scratch.mit.edu/)는 MIT Media Lab의 미첼 레즈닉(Mitchel Resnick) 교수팀에 의해 개발되었다. 현재 웹상에서 무료로 접근되는 오픈 소프트웨어(open software) 형식으로 운영되며 코딩교육용 초기학습환경으로 40개국의 다양한 언어로 번역되어 널리 활용되고 있다. 보다 많은 사람들에게 코딩을 익히도록 이끄는 목표 아래, 주 교육대상층을 초등학생이나 코딩을 접해 본 적이

4) 본 연구자는 스크래치를 통한 코딩교육에서의 그래픽 사고를 융합적 접근의 필요성을 논하기 위하여 기초연구로서 두 명의 스크래치 학습자를 인터뷰하며 그들의 학습경험을 조사한 바 있다(이현진, 2017).

5) http://microbit.org/. Micro:bit은 작은 프로그래밍이 가능한 컴퓨터이다. 이는 배움과 가르침이 쉽고 재미있도록 만들기 위해 디자인되었다. 영국 BBC 방송국과 microsoft global이 함께 개발하고 교육 커리큘럼을 운영한다.

6) Tynker(https://www.tynker.com/)는 웹기반 학습 플랫폼이자 시각 프로그래밍 언어

7) Logo는 교육용 프로그래밍 언어이다. 1967년 Wally Feurzeig, Seymour Papert과 Cynthia Solomon에 의해 개발 디자인되었다. "Logo"는 그리스어로 단어(word) 혹은 "생각(thought)"을 뜻한다. 이는 그 언어 자체가 숫자를 주로 사용하는 다른 프로그래밍 언어와 다르다는 것을 나타내며, 그래픽스와 논리에 주로 관계함을 드러낸다. Logo는 특히 Turtle과 그래픽스graphics로 유명하다. 터틀은 로봇이며 사용자가 컴퓨터에서 작은 펜을 통하거나 로봇의 몸체에 붙은 드로잉 기능을 통해 이들 움직이고 조종하며, 2차원 평면상에 터틀(turtle)이라 부르는 방법을 통해 그림을 그린다. 이는 Seymour Papert에 의해 1960년대 후반 추가되었다.

8) https://www.alice.org/. Alice는 교육용 프로그래밍 언어이다. Alice는 drag and drop 환경에서 3D 모델을 통한 컴퓨터 애니메이션을 제작하도록 하는 통합환경(integrated development environment (IDE))으로 구성되어 있다. 이 소프트웨어는 처음에 University of Virginia에서 1994년 제작되었으며, 후에 1997년부터 Carnegie Mellon 대학의 Randy Pausch 교수가 이끄는 연구그룹을 통해 개발되었다.

없는 초심자들로 삼는다. 이러한 초기학습환경(ILE)은 학습자의 순조로운 입문을 유도키 위하여 최소한의 장벽을 제공하는 낮은 마루(a low floor)와, 그들이 코딩을 통해 최대한 언어적이고 구문적인 제약 없이 프로젝트를 시도할 수 있도록 돕기 위한 높은 천정(a high ceiling), 그리고 여러 다양한 프로젝트들을 만드는데 융통성을 가질 수 있는 넓은 벽(wide walls)을 제공하여야 한다는 철학적 바탕에서 개발되었다.[9] 특히 이러한 철학은 스크래치와 code.org 등의 아동용 코딩교육용 학습환경에서 '비주얼 프로그래밍' 혹은 그래픽 유저 인터페이스(Graphical User Interface, GUI) 방식으로 구현되고 있다. 이러한 비주얼 프로그래밍 언어들은 과거 개발자들이 사용하는 프로그래밍 언어들처럼 중괄호(' {', curly brackets), 세미콜론(';', semicolons) 등이 없는 알고리즘 빌딩 블록을 사용한다. 또한 학습자들은 그 기능과 성격, 특징들이 서로 다른 색으로 구별되어 안내되는 코딩 블록들을 조합하여 쉽고 재미있게 코딩 개념을 학습한다. 또한 이러한 블록들은 외관 상 애초에 잘못 연결될 수 없도록 구조화시켜, 학습자들이 블록의 순서를 서로 바꾸어 연결하거나, 문맥에 맞지 않는 블록들을 끌어와 사용하더라도, 구문 에러(syntax error)를 만들지 않는다. 이러한 방식을 통해 비주얼 프로그래밍 언어들은 과거 프로그래밍 언어에 대하여 학습자들이 느끼던 어려움, 즉 프로그래밍 구문(syntax) 쓰기에서의 에러-구문 안에서 단 한글자라도 오탈자가 있거나, 하나의 단어에서도 대문자, 소문자를 잘못 사용할 경우 마주하는 에러-를 극복하도록 이끈다.

이러한 초기학습환경(ILE)은 프로그래밍 학습자들이 정확하고 오류 없는 구문을 쓰는 것보다, '의사코드(pseudo-code)'[10] 작성과 같이 어떻게 본인이 만들고자 하는 프로그램을 전체적으로 어떻게 구상하고 이를 구현하기 위하여 어떤 구체적인 접근을 계획할지 고민하도록 유도하는데 초점을 맞춘다. 즉 학습자가 개발하고자 하는 전체의 프로젝트를 하나의 큰 문제라고 인식한다면, 이 문제를 해결해 나가고자 어떻게 단계별 과정별로 접근할 수 있는 작은 문제들로 쪼갤 수 있는지 계획을 세우도록 한다. 이는 코딩교육에서 어린 학습자들에게 문제해결력과 논리적 사고력을 함양시키기 위한 목표와 연결된다. 또한 프로그래밍 언어들이 각각 그 구문(syntax)은 다를 수 있지만 그 기본 개념들은 서로 호환가능하며, 인식론적으로 공통적이라는 생각에 근거한다. 한편 이는 어떻게 컴퓨팅 문제(computing problem)를 해결할 것인가, 즉 기계를 위한 업무와 구조를 어떻게 정리하고 명령내릴 것인가(기계가 사고하는 방식, 기계가 일을 처리하는 방식으로 사고하고 이를 제어하고 조종할 것인가), 이를 위한 프로그램적 수행은 어떻게 접근되고 진행되는가 등, 컴퓨팅적 사고와 관련한 '메타(meta)적 사고'와 '컴퓨팅적 리터러시(computational literacy)' 함양이라는 학습 목표에 다가가도록 접근한다.[11]

---

9) 이는 "스크래치: 모두를 위한 프로그래밍"(2009)를 쓴 레즈닉 교수와 동료 연구자들이 주장한 바이기도 하다. Resnick, M. Maloney, J. MonroyHernández, A. Rusk, N. Eastmond, E. Brennan, K. Millner, A. Rosenbaum, E. Silver, J. Silverman, B. & Kafai, Y.. (2009). Scratch: Programming for all. Communications of the ACM, 52(11), 60-67. 한편, Duncan, Bell & Tanimoto(2014)는 스크래치는 자유도가 높은 프로그래밍 환경을 제공하므로, 학습자들은 여러 가지 블록들을 가지고 실험하며, 그들이 구현하고 싶은 개념들을 구현할 수 있다고 말한다. 또한 스크래치에도 튜토리얼과 도전 과제들이 존재하지만 그들이 학습자들에게 반드시 따르도록 요청되는 것은 아니라고 말한다. 한편, 동시에 이러한 학습환경에서 학습자들이 프로그래밍이나 CS 혹은 컴퓨팅 사고력(CT) 개념에 대하여 어떠한 얻는 바도 없을 수 있음을, 혹은 프로그래밍이 무엇인지에 대한 잘못된 모델을 형성, 발전시킬 수도 있음을 단점으로 지적하기도 한다. 만약 학생이 단순한 애니메이션이나 거북이를 움직이는 함수 등 (저자들은 이를 'Spritefest'라고 말한다고 하는데) 단순 함수 구현에만 초점을 맞출 경우, 조건문 등 보다 추상적인 개념으로 사고를 확장, 발전시키지 못할 수 있기 때문이다. 따라서 저자들은 이 논문에서 제대로 된 선생님의 가르침과 안내가 중요하며, 이러한 제대로 된 뒷받침이 있는 경우, 위에서 말한 학습환경은 프로그래밍 스킬과 추상적인 CS와 CT 개념을 가르칠 수 있는 좋은 환경이 될 수 있다고 주장한다.

10) '의사코드'에서의 '의사'는 '가짜의(Pseudo)'라는 뜻으로 의사코드를 작성하는 것은 가짜 코드를 작성하는 것을 뜻한다. 여기서 가짜 코드는 컴퓨터 프로그램이 알고리즘적으로 수행할 내용을 우리가 사용하는 자연어(한국어 또는 영어 등의 언어)로 간략히 서술해 놓은 방식으로 작성된다. 의사코드는 보통 코딩 입력 전에 미리 프로그래밍 접근 방향성을 명확히 설계하고 정립하기 위해 사용된다.

11) 따라서 이러한 초기학습환경은 학습자가 어떻게 코드를 짤 것인가(how to write a code) 걱정하지 않도록 유도한다. 그리고 차차 학습내용이 발전하고 코딩 중심의 논리적 사고가 익숙해지면, 서서히 이를 PHP 혹은 Python 등의 프로그래밍 언어를 통해 코딩구문작성을 위한 사고(구문적 사고까지 확장된 사고)로 바꾸어 전문적인 내용을 학습하도록 한다.

한편, 스크래치는 학습자들에게 게임과 애니메이션 등의 프로젝트를 제작하는 가운데 프로그래밍 구조를 익히고 문제해결을 시도하도록 유도한다.[12] 이는 뒤에서 다시 설명하겠지만 '프로젝트 기반의 학습(project-based learning)'을 근간으로 하는 것이다. 스크래치에서는 특히 이러한 프로젝트 기반의 접근이 '스튜디오'라는 웹상의 프로젝트 공유 환경을 통해 지원된다. 학습자는 자신이 만든 프로젝트를 다른 이들과 공유하거나, 또한 역으로 다른 이들의 프로젝트를 플레이해보거나 '리믹스(remix)'해볼 수 있도록 지원된 기능을 통해 실행 코드 내부를 들여다보고 이들을 자유롭게 참조, 모방할 수 있다. 이처럼 코딩 교육을 위한 초기학습환경은 Resnick et al.(2009)도 주지하듯, 어린이 코딩교육의 기본적인 목표가 새로운 세대의 학습자들을 "전문적인 프로그래머로서 준비시키는 것이 아니라, 그들의 아이디어를 프로그래밍을 사용하여 편하게 표현할 수 있는, 창의적이고 시스템적인 사상가, 즉 생각하는 사람으로 키우는 것"있다는 것을 강조한다.

본 연구자는 이러한 코딩교육에 근간이 되는 기본적 교육 철학적 배경과 '프로젝트 제작 중심의 접근'이라고 하는 교육적 방법론에 주목하여, 코딩 학습의 그 기본 목표로 유지하되, 동시에 융합교육이 함께 접목될 수 있는 가능성을 탐색하고자 한다. 또한 융합적인 접근을 통해 보다 확장된 방식으로 공부하는 법에 대하여 학습하도록 유도하는 동시에, 생각하는 사람으로 이끄는 컴퓨팅 리터리시 교육이 어떻게 접근될 수 있을지 생각해보고자 한다. 이를 위해 본 연구는 스크래치 프로젝트 기반의 학습법 가운데, 그래픽 사고와 코딩 사고를 접목시키는 방안으로서의 구체적 커리큘럼 구성 내용을 논해보고자 한다. 따라서 다음 절에서는 기존에 제시되고 활용되는 어린이 코딩교육이 어떻게 학습자 인지에 기반하는지 커리큘럼들의 분석을 시도하는 선행 연구들을 검토하고자 한다. 이는 어린 학습자 대상 프로그래밍 교육 커리큘럼에서 학습자의 연령에 따른 인지력과 이해도 그리고 이에 따른 진도를 고려하여, 기존 코딩과 함께 하되, 그와 더불어 그래픽 및 멀티미디어 교육과 관련된 인지적 사고를 추가적으로 더하고, 융합하려는 본 연구에서의 융복합 커리큘럼을 체계적으로 개발하기 위한 유용한 근간이 된다.

2.2 어린 학습자 프로그래밍 융합교육 커리큘럼 구성 위한 선행연구 검토

2-2-1. 연령과 단계별로 학습자의 코딩교육 진도에 대하여 접근한 선행연구

학습자의 연령과 발달 단계별로 코딩 학습 진도에 접근하는 연구들은 크게 발달심리학(developmental psychology)을 그 기초로 한다. 발달심리학은 피아제의 인지발달론(Piaget's theory of cognitive development), 즉 연령에 따라 인간의 인지발달 단계를 4단계로 나누어 접근하는 이론을 중심으로 한다. 이는 감각운동기(0-2세), 전조작기(2-7세), 구체적 조작기(7-11세), 형식적 조작기(11세 이후)로 구분된다. 그런데 이 구분은 네오-피아제 이론(neo-Piagetian theories)[13]을 통해 재고된다. 인지 발달 단계에 따라 각 행동의 특성에 특정 나이를 부여하는 것이 그렇게 간단하게 정리될 수 없는 "잘못된 카테고리(pigeon holed)"라는 것이다. 대신 이들은 인간의 인지발달이 각자 배움의 영역에 따른 개인차에 상당히 의존할 수밖에 없으며, 개인이 속한 문화, 사회의 발달 정도에 따른 차이 역시 더 심도 있게 고려해야 한다고 주장한다. 이처럼 피아제의 이론과 그 수정 이론은 어린이들의 인지발달 정도와 단계를 고찰할 때 흔히 발견되는 변화들, 그들을 둘러싼 세상과의 관계에서의 네오-피아제 이론은 인지가 어떻게 획득되어 가는가, 그리하여 추상적이고 논리적인 사고를 어떻게 발전시키게 되는가 관찰하는데 참고할 수 있는 지침을 마련해 준다.

---

12) 스크래치와 더불어, 미국 등지에서 역시 광범위하게 코딩교육 현장에서 사용되는 툴이자 커리큘럼인 Code.org 등도 이러한 비주얼 프로그래밍 방식의 인터페이스를 통해 구체화되고 단계적인 미션 형식의 과제를 차례대로 수행해 나가며 커리큘럼을 학습하도록 디자인되어 있다. 그런데 Code.org가 미션 수행 방식으로 진행되는 커리큘럼적 학습을 통해 코딩의 이해를 도모하는 반면, 스크래치는 학습자들에게 게임과 애니메이션 등의 프로젝트를 제작하여 가며 프로그래밍 구조를 익히도록 유도한다.

13) Morra, S. Gobbo, C. Marini, Z. & Sheese. R. Cognitive development: neo-Piagetian perspectives. Psychology Press, 2007.

한편, 발달심리학 이론들은 우리에게 어떤 종류의 CS와 CT 개념들이 코딩을 학습하는 다양한 연령대 학생들에게 부합될 수 있을지 구체적인 커리큘럼을 모색할 수 있는 대강의 가이드라인을 제시하여 준다. 피아제의 모델에서 구체적 조작기(concrete operational stage)에 해당하는 7-11세 아이들은 신체적으로 손 등의 움직임이 보다 정교해지고 유연해지며, 자신을 둘러싼 세계에 대하여 이해하는 정신적 능력도 훨씬 더 발전한다. 또한 논리적으로 생각할 수 있는 시기가 된다. 2-7세에 해당하는 전조작기(operational stage) 역시 어느 정도의 논리적인 추론과 상징을 사용한 활동을 포함한다. 이에 근거하면 10-12세 초등학교 고학년 학생들에게 프로그래밍 교육이 효과적일 수 있음은 비교적 분명히 확인할 수 있으며, 5-7세 정도의 어린 학습자에게도 언어의 수준과 이해 정도가 프로그래밍 개념을 전달시켜 줄 수 있는 정도로만 확보될 수 있다면, 프로그래밍 교육을 시도할 수 있음을 추론할 수 있다. 다시 말해, 전조작기와 조작기에 컴퓨터 프로그래밍의 초기 학습이 원활하게 접목될 수 있을 것이라 유추해 볼 수 있다.

이러한 발달심리학에 근거한 코딩교육 관련 커리큘럼을 연구한 선행연구들은 어린 학습자의 인지발달 수준에 맞추어 코딩교육을 2-3개의 학습 단계로 구분하여 접근하고 있는 것을 살펴볼 수 있다. 미국의 2011 Computer Science Teachers Association(CSTA, http://csta.acm.org/)에서는 미국 학제 형식인 K-12 커리큘럼[14]에서 프로그래밍 교육을 위한 세 가지 단계별 접근을 제안한다. Level 1(K-6)은 시퀀싱(sequencing)에 초점을 맞추고, Level 2(grades 6-9)에서는 협력적인 프로그래밍(collaborative programming)이 제안된다. Level 3(grades 9-12)에서는 알고리즘을 통한 문제해결력(algorithmic problem solving)이 명시적으로 부각된다. 또한 프로그래밍적 문제해결력을 위해 조건문(conditions), 변수(variables), 조작을 위한 연산자(operators) 활용, 그리고 다른 일반적인 구성(construct) 방법을 함께 학습하도록 제안한다.[15] 영국에서 2014-2015년 초등교육과정에 적용한 프로그래밍의 기본적 요소를 가르치는 새로운 커리큘럼[16]을 살펴보면, 1단계(만 5-7세)에서는 “간단한 프로그램을 만들고 디버깅하기(create and debug simple programs)"가 중점 학습목표로 제시된다. 이 시기에는 학습 보조 재료로써 활용되는 비봇(Beebot)[17] 등을 통해 시퀀스를 이해하도록 가르친다. 2단계(만 7-11세)에서는 “시퀀스, 선택, 그리고 반복(sequence, selection and repetition)"이라는 개념이 등장하며, 교육도구로써 스크래치(Scratch)와 같은 언어들이 제안된다. 호주의 경우[18], 유치원에서 초등 2학년(F-2, 만 8세까지)은 시퀀스, 3-4학년(만 8-10세)는 가지치기(branching)와 유저 인풋(user input)의 사용, 5-6학년(만 10-12세)에는 반복(iteration)의 개념이 추가되며, 스크래치와 같은 단순한 시각적 언어를 사용하는 프로그래밍이 접근되도록 제안된다. 이처럼 많은 국가들의 교육 과정에서 살펴지듯, 프로그래밍 교육에서 학습될 수 있는 전반적 범위가 초등교육이 끝날 때쯤 모두 끝나도록 기대되고 있다.[19]

한편, Duncan, Bell, & Tanimoto(2014)는 위와 같은 선행연구를 바탕으로, 어린이 코딩교육을 위한 여러 초기학습환경들을 조사하여 레벨 0에서 레벨 4까지로 구분하고 있다(〈표 1〉 참조)[20]. 이들은 어린 학습자들 나이에 상관하는 능력치, 그리고 학습 결과에 대한 정도들을 체험적이고 발견적인 세트(a set of heuristics)를 통해 구분, 정리했다고 밝힌다.

14) 미국 학제간 분류: K는 kindergarten으로 유치원 교육과정이며, 12는 1학년부터 12학년까지 우리나라로 하면 초등학교에서 고등학교에 이르는 교육과정을 학년별(grade)로 지칭하는 것이다.

15) Duncan, Bell, & Tanimoto (2014) p.5에서 재인용

16) http://www.computingatschool.org.uk/data/uploads/primary_national_curriculum_-_computing.pdf

17) Beebot은 프로그래밍 가능한 로봇으로 어린 아이들이 코딩을 학습할 수 있도록 디자인된 상품이다. https://www.bee-bot.us/ 참조.

18) http://www.australiancurriculum.edu.au/technologies/digital-technologies/Curriculum/F-10

19) Duncan et al. (2014) p.5에서 재인용

20) 〈표 1〉은 Duncan et al. (2014) p.6에서 기술된 내용을 바탕으로 본 연구자가 만들과 정리한 표이다.

〈표 1〉 A set of heuristics used to classify an ILE

Level 0	만 2-7세 (Age range 2-7 years)	Drag-and-Drop or simpler. Teaches planning (sequence) only. Requires no abstraction. Contains no significant use of: functions, variables, iteration, indexed data structures, conditional execution.
Level 1	만 5-10세 (Age range 5-10 years)	Drag-and-Drop. Requires no abstraction (or small amounts). Contains none or few of: functions, variables, iteration, indexed data structures, conditional execution.
Level 2	만 8-14세 (Age range 8-14 years)	Drag-and-Drop or text-based. Includes some abstraction. Contains some or most of: functions, variables, iteration, indexed data structures, conditional execution.
Level 3	만 12세 이상 (Ages 12 years and up)	Drag-and-Drop or text-based. Includes abstraction. Contains all of: functions, variables, iteration, indexed data structures, conditional execution.
Level 4	만 14세 이상 (Ages 14 years and up)	Teaches an industry-level Turing-complete programming language. Advanced, with extensions available. Contains all of: functions, variables, iteration, indexed data structures, conditional execution.

이에 따르면 레벨 0은 매우 어린 학습자들을 대상으로 한다. 대부분 '드래그와 드롭(drag and drop)' 블록들을 선택, 배치하며 수행의 순서(sequences of operations)를 학습시키는데 집중한다. 이 부분은 스크래치가 개발된 이후 좀 더 어린학습자를 대상으로 추가적으로 개발된, '스크래치 주니어(ScratchJr)'에서 주요하게 접근하도록 특화된 내용이기도 하다. 이들은 학습자들에게 프로그래밍을 구상하기 위한 기초적인 개념으로서 학습자들이 선택한 액션들이 순서에 따라 수행되도록 시퀀스를 구성하는 개념을 가르친다. 예를 들어 비봇(Beebot)이 앞뒤로 갈지, 좌우로 회전 할지 등을 조정하도록 학습시킨다. 이를 통해 학습자들에게 그들이 행한 액션들이 프로그래밍을 통해 제어하고자 하는 오브젝트들에 분명하고 직접적인 영향을 주고 있음을 확실히 이해시킨다. 레벨 1 역시 오브젝트들을 조작하는 드래그와 드롭(drag-and-drop) 환경에 머문다. 그러나 레벨 0보다는 많은 기능들을 가진 함수(functions) 혹은 조건문(conditionals) 등의 블록들을 추가한다. 레벨 2에서는 변수(variables)과 함수(functions) 등 보다 다양한 기능이 추가되어 추상화되며 복잡해진다. 이 단계에서는 텍스트에 기반 한 산업용 프로그래밍 언어들도 약간 포함된다. 레벨 3은 기존의 언어기반 프로그래밍 언어들, 즉 Turing이 완성한 언어들이나 이와 유사한 언어들이 가진 영역들을 모두 다룬다. 이들은 소위, '넓은 벽'을 가지고 있어 응용 가능성이 높도록 지원되므로, 학습자들은 그들의 아이디어를 보다 쉽게 구현할 수 있다. 여기서는 학습자들이 창의성을 발현시키도록 돕는 구성주의자의 방법론(a constructivist manner)을 배울 수 있도록 유도된다. 레벨 4은 프로그래밍에 관심을 더 크게 가지며 초급단계의 프로그래밍 학습환경이 모두 학습되어 더 이상 초급단계가 시시하고 지겨워진, 상위레벨의 학생을 대상으로 한다. 저자들은 이러한 상위 레벨은 전형적인 정규 교육 커리큘럼에 필요하지 않을 수도 있으나, 만약 학생과 교사가 도전적인 수준을 원한다면 고등학교 수준에서 여전히 이용될 수 있다고 말한다. 또한 이런 레벨의 프로그램들이 존재한다는 것 역시 매우 중요한데, 수준 높은 학생들이 좀 더 관심을 가지고 그들의 능력을 신장시킬 수 있는 기회를 확대하여 제공하기 때문이다. 그러나 자칫 이 레벨의 프로그램들에서는 접근 가능한 학습 콘텐츠들이 고갈되어 학습자를 지치게 할 수 있다는 위험도 있다고 말한다. 즉 학습자가 원하는 방향의 확장이 불가해졌을 때 지루하게 만들 수 있다는 위험성을 말하는 것이다. 결론적으로 저자들은 이러한 그들의 분석을 기초로 초등학생 학습자들에게는 레벨 1~3이 이상적이라 제안하고 있다. 한편, 보다 구체적인 연령대의 어린 학습자를 조사한 연구들(Bers, 2007; Bers, 2012)은 만 4세의 어린이가 간단한 컴퓨터 프로그래밍 개념을 배울 수 있다는 것을 보여준다.[21] Kazakoff & Bers(2012) 역시 유치원 연령의 학습자들

21) 이 연구 역시 어린 학습자들에게 제안되는 코딩 수업에서는 코딩의 제일 기초적 접근인 '시퀀싱(sequencing)'에 대하여 다룬다고 분석한다.

에게 로보틱스 프로그래밍을 통해 시퀀싱을 가르치는데 성공하였다고 밝힌다. 그리고 슬로바키아의 초등학생들을 대상으로 실험한 바에 의하면 만 8-10세의 학생들은 2D공간에서의 목표물에게로 오브젝트를 명령을 통해 제어하여 움직일 수 있도록 디자인된 환경에서 성공적으로 작업을 수행하였다고 한다(Duncan et al., 2014, p.4. 재인용).

한편, Seiter & Foreman(2013)은 초등학생들을 대상으로 한 커리큘럼 디자인을 위하여 "초기 컴퓨팅 사고의 발달 모델(Progression of Early Computational Thinking Model)" 연구를 진행했다. 이 모델은 스크래치 수업현장에서 학생들이 프로그래밍을 하는 패턴을 관찰하고 이를 기반으로 제안되었다는 점에서 의미 있다. 이 연구의 세부 논의들은 초등학생의 대상으로 연령대를 고려한 코딩 커리큘럼을 개발하고자 하는 본 연구에도 상당한 도움을 준다. 위의 연구는 초등 1학년생에게서는 'Animate Looks(스프라이트의 외관을 바꾸는 것, changing the look of a sprite)'과 'Conversate(스프라이트들 사이의 대화, conversations between sprites)', 'Animate Motion(동작을 애니메이션으로 만드는 것)'과 관계된 것들을 이용한 작업 패턴을 발견할 수 있었다고 한다. 그러나 '선택(selection)' 혹은 '제어된 반복(controlled repetition)' 혹은 '충돌(collide)', '사용자 상호작용(user interaction)', 그리고 '점수 유지(maintain score)' 개념 등은 1학년 학생들은 대부분 사용하지 않는 것으로 나타났다고 밝힌다. 점수 유지 및 점수 제어는 변수(variables)와 불린(boolean)을 사용한 개념과 표현을 잘 이해하여야 하는데, 이는 일반적으로 4학년이나 혹은 그 이상의 나이에서나 이해할 수 있는 영역이기 때문이다. Seiter & Foreman은 이러한 발달 모델이 학년마다 어떤 특별한 패턴의 학습이 적합할 수 있다는 것을 보여준다고 결론짓는다. 그리고 앞서 언급한 Duncan et al.(2014)은 다양한 연령대의 학생들을 가르쳐본 경험에 의거해, 만10에서 12세 학습자가 특별히 컴퓨터 과학(CS)의 새로운 아이디어를 흡수하기에 준비가 된 나이라고 판단한다. 추가적으로 유아 등 어린 학습자들의 경우 새로 접하는 학습영역에 매우 열정적이며 학습에 열린 태도를 가질 수 있는 반면, 이들은 그들보다 높은 연령대의 학습자들에 비해 학문적 지식과 사회적 이해가 적기 때문에, 발전 속도와 학습 속도가 다소 느릴 수 있음도 언급한다. 이에 비교하여 만 12세 이상의 학습자들은 학습과 발전 속도는 빠르지만, 새로운 아이디어나 개념에 덜 오픈되어 있다고도 한다(p.5). Smith, Sutclie, & Sandvik(2014)은 영국 Code Club에서 어린 학생들을 가르친 경험과 그들이 학습자들에게 설문조사를 통해 확인한 바를 바탕으로, 어린 학습자들은 일반적으로 초반에 지도하고 안내해 준 내용(guided instructions)은 잘 따르지만, 안내되지 않은 도전적인 영역(unguided challenges)에 대하여는 어떠한 새로운 지식도 적용시키지 않으려 한다고 밝힌다. 또한 학생들이 제일 어려워하는 개념이자 기술은 디버깅(debugging)과 디자이닝(designing)이라고도 하였다.[22)]

2-2-2. Project-based learning 관련 선행연구

한편, 컴퓨터 프로그래밍 교육이 대부분 프로그래밍 언어 구문과 프로그래밍 스킬을 가르치는데 집중하는 반면, 문제해결력은 종종 무시된다고 말하며, 프로그래밍 교육의 핵심은 문제해결력을 함양시키는데 있다고 주장하는 Wang, Huang, & Hwang(2014)의 연구도 주목해서 바라볼 필요가 있다. 이들은 스크래치를 통해 프로그래밍 교육에서의 문제해결(problem-solving) 능력이 접목된 시나리오를 프로젝트 중심의 학습법(project-based learning)과 통합시키는 실험을 진행한다. 평균 수준의 학생들과 수학 영재 학생들을 대상으로 프로젝트 중심 학습활동을 수행하는 실험을 진행하였는데, 두 그룹의 학생들에게 문제해결력 중심의 프로젝트 중심 교육이 유효함을 입증하였고, 그 결과 공통적으로 문제해결력이 향상됨을 확인하였다고 밝힌다.[23)] 또한 프로젝트 중심

22) N. Smith, C. Sutclie, and L. Sandvik. Code club: Bringing programming to uk primary schools through scratch. In Proc. 45th ACM Technical Symposium on Computer Science Education, SIGCSE '14, pages 517-522, New York, NY, USA, 2014. ACM. Duncan et al., 2014, pp. 3-4, 재인용

23) 학습 동기와 태도, 그리고 문제해결력에서 수학 영재들이 일반평균 학생들보다 훨씬 더 좋은 수행능력을 보였으나, 두 그룹 모두 공통적으로 문제해결 능력들은 향상되었다고 밝힌다.

의 접근이 학습동기를 향상시킬 뿐만 아니라, 학습성취도 또한 향상시키고 있음을 통계적으로 확인한다. 코딩교육에서 프로젝트 중심의 접근법을 연구한 또 다른 여러 연구들(Markham, Mergendoller, Larmer, & Ravitz, 2003; Taylor, Harlow & Forret, 2010; Schaffer, Chen, Zhu & Oakes, 2012; Hung, Hwang, & Huang, 2012)에서는 이러한 교육 방식이 학생들에게 주어진 임무를 동료들과 함께 협력하여 수행할 수 있게 이끌며, 그 과정을 통하여 학생들이 학습기간 내 배운 지식과 과거 경험을 바탕으로, 새로운 학습 자원을 탐색하고 이를 문제해결에 적용시킨다고 주장한다. 이러한 연구들은 프로젝트 중심 학습 방식을 통해 코딩교육에서의 문제해결력이 향상되는 효과를 공통적으로 주장하는 것이다. 본 연구도 문제해결력 및 학습성취도, 그리고 더 큰 학습능력을 갖추기 위한 기본적 사고력을 학습시키는 방안으로서 프로젝트 중심으로 접근하는 융합커리큘럼을 적극적으로 모색하고자 한다.

3. 코딩사고와 그래픽 및 멀티미디어 사고가 융합된 커리큘럼

3-1. 그래픽 사고와 멀티미디어 사고

본 연구는 코딩 사고에 그래픽 사고와 멀티미디어 사고가 서로 어떻게 연결될 수 있을지 논하려 한다. 이는 코딩 사고가 그래픽 사고와 단순히 결합되는 것 이상으로, 컴퓨팅 사고로서 논해지는 코딩 사고가 그래픽 사고에도 어느 정도 공통적 관점에서 접근될 수 있다고 보기 때문이다. 그러한 의미에서 '그래픽 사고(graphic thinking)' 및 '멀티미디어 사고'에 대하여 본 연구에서 의미하는 바를 좀 더 명확하게 짚을 필요가 있다. 본 연구에서 접근하고자 하는 그래픽 사고는 '컴퓨터 소프트웨어 등에서 디지털 그래픽 이미지와 동영상 등을 편집하고 조작하며 개발하는 과정에 필요한 사고'를 지칭한다. 폴 라소(Paul Laseau, 1989)에게 그래픽 사고(graphic thinking)란 좀 더 폭넓은 '시지각적 사고의 흐름'을 말하는 것이기도 하다. 라소는 건축가나 디자이너들, 소위 시각적 대상을 다루는 이들이 어떻게 그래픽 사고를 통해 시각적 아이디어를 개설, 정리, 전달하는지 논한다. 이 때 라소는 디자이너와 건축가들의 손으로 그린 스케치 등 비디지털 영역들을 포함한 전반적인 시각적 사고, 그래픽 사고에 관해 논한다.[24] 본 연구에서 논하는 그래픽 사고는 이러한 광범위한 시각적 사고 및 비디지털 영역들을 포함한 라소의 개념과는 조금 다른 면이 있지만, 라소가 언급하는 것처럼 그래픽 사고가 추상화 능력, 과정적으로 사고하는 능력과 문제해결능력 등과 연결되는 지점들을 다룬다는 부분에서는 동일한 생각을 갖는다. 또한 앞서 컴퓨팅 사고력을 통해 개발하고 발전시키고자 하는 능력들과 이러한 그래픽 사고와 공통 지점들을 언급하였는데, 비록 각 영역(프로그래밍과 디자인 혹은 건축 등)에서 다루는 추상화 과정, 순차적 사고 과정, 문제와 해결에 대한 인식과 방법론 등은 구체적으로 다를 수 있으나 양쪽 영역에서 작업자들이 자신의 능력을 신장시키고 또한 학습 개발된 능력을 동원하여 매 단계 자신이 해결하여야 할 작업을 수행하여 나간다는 점 역시 동일하게 접근한다.

한편 시지각적 사고를 통해 발전되는 이런 능력들은 오래 전부터 논의되어왔다. 예술심리학자 루돌프 아른하임(Rudolf Arnheim)가 『시각적 사고(Visual Thinking)』(1969)에서 탐구한 미술의 형태와 기능에 관한 심리학적 논의에 따르면, 미술 활동은 인지활동(認知活動)으로서 감각경험을 포함하며, 지각하기(perceiving)와 생각하기(thinking)가 구분될 수 없을 정도로 얽혀있는 일종의 추리(推理, reasoning)라 말한다.[25] 따라서 아른하임의 시각적 사고는 "시각의 기본 과정들조차 추리의 전형적 기제들을 포함하고 있으며, 과학의 사고 모형에 따르는 심상 과정을 기술할 뿐 아니라 미술에 있어서도 혁신적인 문제해결의 과정을 제시하고 있다"고도 논해진다.[26] 이처럼 시각적 사고는 "미술

24) Paul Laseau, Graphic Thinking for Architects and Designers, Van Nostrand Reinhold, 1989, p.vii

25) 루돌프 아른하임 저, 김정오 역, 시각적 사고(Visual Thinking), 이화여자대학교 출판부, 2004. p.7

26) 아른하임의 『시각적 사고(Visual Thinking)』에 대한 출판사 온라인 책 소개란 참고.

과 심리학은 물론 창의적 사고와 문제해결 분야 모두에 관계"하며,[27] 이러한 이유에서도 그래픽 사고를 포함하여 코딩사고에서 함양코자 하는 능력과 유사한 지점을 공유한다고 볼 수 있다.

그래픽 사고가 위와 같이 코딩사고 혹은 컴퓨팅 사고와 관계한다면, 컴퓨터를 사용한 그래픽 사고, 즉 디지털 그래픽 사고와 코딩사고는 좀 더 긴밀하다고 볼 수 있을까? 이 질문은 미국 워싱턴 대학의 컴퓨터 공학과 엔지니어링 분야 교수인 스티브 타니모토(Steve Tanimoto)의 『이미지 프로세싱에 대한 간학제적 소개: 픽셀들, 넘버들, 프로그램들(An Interdisciplinary Introduction to Image Processing: Pixels, Numbers, and Programs)』(2012)에서의 논의에서 도움이 되는 답을 발견해볼 수 있다.[28] 타니모토는 이 책에서 이미지 프로세싱을 통해 컴퓨터 공학도들에게 코딩의 기초개념을 학습시키고자 접근하는데, 그 배경은 이러하다. 오늘날의 이미지는 일반적으로 디지털 사진으로 생산되고 또한 디지털 방식으로 재현되기에, 이미지 프로세싱은 이러한 이미지를 시그널과 시그널 프로세싱으로 접근, 구현하는 것이 된다. 또한 이미지 프로세싱은 전자공학부터 관계와 분산을 다루는 통계학, 알고리즘적 계산을 다루는 수학, 색과 감각인지(perception)와 연결되는 컴퓨터 그래픽 아트와 디지털 아트 등의 예술 분과 영역 등 다양한 학문 분야에서의 종합된 사고를 요하는 영역이라 할 수 있다는 것이다. 또한 이미지 프로세싱에는 일루전 등 뇌와 정신적 과정이 동원되는 시각적 경험(visual experience)에 대한 이해부터 색깔 및 색깔 팔레트에 대한 이해, 밝기/대조(brightness/contrast) 등에 대한 이해, JPEG 혹은 GIF 등 그래픽 이미지 포맷, 패턴 인지 등 종합적이며 다분과적 이해가 필요하기도 하다(pp. xvii-20). 결국 이러한 타니모토의 관점과 접근은 이미지 프로세싱에 대한 접근을 통해 그래픽 사고와 코딩사고를 연결시키는 동시에, 그래픽 사고를 통해 궁극적으로 코딩사고에 이르도록 유도하고 있다. 이는 코딩사고와 그래픽 사고를 좀 더 긴밀하게 관계시킨다는 점에서 본 연구에도 유용한 시사점을 제공한다.[29]

이처럼 본 연구에서 논의하고자 하는 그래픽 사고는 그것이 그래픽을 통한 문제 해결이던, 혹은 디자인적이며 시지각적 사고이던 혹은 타니모토가 접근하는 이미지 프로세싱을 위한 종합적 방법론과 인식이던 간에, 하나의 문제를 해결하기 위해 효율적이며 과정적으로 사고하고, 때에 따라 추상적으로 혹은 논리적이며 분석적으로 사고할 수 있는 능력이라 종합해 볼 수 있다. 여기서 다시 스크래치 교육으로 돌아가 그 기저에 깔린 접근을 상기해보자. 스크래치 프로그래밍 언어를 개발한 레즈닉(Mitchel Resnick)은 스크래치 프로그램을 게임과 애니메이션을 만들고 이를 통해 멀티미디어를 활용하고 내용적으로 스토리텔링 등을 담아낼 수 있는 툴로 접근하고 있다.

3-2. 스크래치에서의 그래픽과 멀티미디어 사고 함양을 위한 커리큘럼 구성내용

스크래치를 통한 프로젝트 중심의 교육접근법을 적용하는 동시에 그래픽 및 멀티미디어적 사고력을 코딩교육과 접목하기 위한 커리큘럼은 어떤 내용을 다룰 수 있을까? 먼저 융합교육의 목적을 지닌다 하여도 일차적으로는 코딩학습이 그 중심이 되어야 하므로 본 연구는 앞에서 선행연구들을 통해 인지발달을 고려한 여러 코딩교육 커리큘럼 내용을 고찰해 보았다. 이제 학습자의 인지 발달 단계에 따른 단계별 코딩 학습과정과 더불어 그래픽 혹은 멀티미디어 사고를 함께 접목시키는 방향을 접근해 보려 한다. 또한 스크래치의 특성상 게임과 애니메이션 프로젝트 제작이 유도되므로, 코딩학습에

27) 아른하임, op, cit. p.6. 이 책 개정번역판의 역자 글에서 김정오는 이른하임의 시각적 사고가 창의적 사고와 문제해결 분야 모두에 관계한다고 정리한다.

28) Steven, L. Tanimoto, An Interdisciplinary Introduction to Image Processing: Pixels, Numbers, and Programs, MIT press. 2012.

29) 본 연구자는 타니모토 교수의 초청으로 2015년 9월부터 2016년 8월까지 일년간 University of Washington, Computer Science and Engineering과에서 함께 그래픽 접근이 융합된 코딩 교육이라는 주제의 연구를 진행하였다. 타니모토 교수가 이끌고 코딩교육 분야를 전문적으로 연구하는 석박사생들과 함께 진행하는 세미나에 두 학기동안 참여하며, 본 논문에서 분석하는 선행연구들을 포함, 여러 전문적이며 현재 진행형의 논의들을 다양하게 접하고 공부할 수 있었다. 타니모토 교수에게 이 기회를 빌려 다시 한 번 감사드린다.

그래픽과 멀티미디어가 접목되는 방향은, 각 학습자 스스로 제작하고자 희망하는 프로젝트를 기획하도록 유도하며 이를 위한 프로젝트 구상과 디자인을 구체화하도록 가이드 하는 방향이 될 것이다. 그리고 프로젝트 제작과 구현 과정에서 시각적 지식과 수학과 물리 등 코딩 혹은 프로젝트 제작에 관계하는 학제간 연결을 융합적으로 도모하는 방법을 모색하고자 한다. 이는 하나의 프로젝트 개발에 많은 개념과 지식들이 서로 다른 교과영역으로부터 파생, 연계되어 구성된다는 데 착안한다. 이러한 관점에서 그래픽 및 멀티미디어적 사고 역시 다른 교과목과의 연계 상에서 추구될 것이다.

커리큘럼 구성에서 가장 중요시될 내용은 학습자가 스스로 프로젝트를 기획하고 이를 완결할 수 있도록 필요한 제반 지식을 제공하도록 돕는 것이다. 또한, 스스로 문제해결을 완료할 때 성취감을 느끼도록 각 단계에서의 해결할 문제들을 인지시키고, 이를 스스로 해결하도록 유도하여야 한다는 것이다. 이를 위해 시지각적으로 도움이 되는 지식과 접근방법론을 코딩교육과 함께 가이드 해주어야 한다. 위에서 Smith et al.(2014)이 말한 것처럼 어린 학습자들은 일반적으로 초반에 지도하고 안내한 내용은 잘 따르지만, 안내되지 않은 영역에 대하여는 새로운 지식 적용을 어려워한다. 따라서 많은 프로젝트에 일반적으로 적용될 수 있으면서도 응용 가능한, 최대한 기본적이면서도 유용한 정보와 지식, 개념 이해를 제공하는 전체 커리큘럼을 구성하는 가운데, 네오-피아제의 이론을 추가적으로 적용하여 개별학습자간 차이를 인정하는 개별화된 접근도 함께 이끌어야 할 것이다.

이러한 목표 하에 구체적으로 수업 초반에는 스크래치 학습환경, 특히 그 인터페이스에 대하여 보다 자세하게 소개하는 과정이 중요하다. 특히 게임과 애니메이션, 인터랙티브 스토리텔링 프로젝트 등의 제작을 위한 스크래치의 여러 인터페이스에 대해 상세히 소개하는 과정이 필요하다. 스크래치 교육에서 현재 이러한 과정은 생략되거나 매우 간단히 소개된 채 넘어가는 경우가 상당히 많다. 코딩교육이란 목적 하에 바로 코딩개념 및 블록을 구성하는 스크립트 창에 대한 소개로 넘어가는 경우가 대부분이기 때문이다. 이는 학습자가 향후 당면할 여러 프로젝트 구성과 개발 관련 문제를 해결하는데 어려움을 느끼게 될 경우, 흥미를 잃거나, 다시 튜토리얼 등의 자료를 찾아보면서 따로 그리고 스스로 학습해야 하는 번거로움을 줄 수 있다.

① 스크래치 학습환경 사용자 인터페이스에 대한 소개

스크래치 학습환경 인터페이스에는 코딩블록뿐만 아니라, 게임 오브제나 애니메이션 캐릭터 등의 객체 및 배경을 이룰 이미지 등이 구성되는 '스프라이트(sprite)' 창, 이러한 스프라이트들이 등장인물과 같이 앉혀지고 움직일 '무대(stage)', 그리고 이들을 코딩 블록들을 통해 제어하고 조작할 '작업영역'이 있다. 작업 영역은 '스크립트(scripts)'와 스프라이트의 모양과 효과들을 만들어내는 '코스츔(costumes, 스크래치 한국어 버전에는 코스츔이 '모양'이라고 번역되어진다)', 그리고 소리(sound)를 제어할 수 있는 영역으로 나눠진다. 그런데 '코스츔' 영역에는 스프라이트 및 무대 배경이미지를 그리거나 이미 제작된 이미지를 바탕으로 편집 가공할 도구 툴이나 색상 팔레트 툴 등의 '그래픽 팔레트(graphic pallette) 툴'이 있다. 또한 사용자(학습자)가 캐릭터 혹은 배경 이미지들을 이미 프로그램에 내장된 이미지들에서 골라 사용하고자 한다면, 이러한 프리셋(pre-set) 가운데 배경이미지를 선택하고 이를 프로젝트에 가져올 수 있는 수행 버튼 등에 대한 소개도 동반되어야 한다. 이러한 구성요소들을 이해하고 이들의 관계를 파악하는 것은 향후 학습자가 프로젝트를 전체를 구성하고 제작하기 위한 매우 중요한 과정이다. 전체 작업을 수행하는 과정에 대한 파악과 이해는 개별 학습자가 만들고자 하는 프로젝트를 하나의 큰 문제라고 간주할 때, 이를 제작하기 위한 세부과정들을 나누어 접근하는 방법, 즉 큰 문제를 세부 문제들로 나누고, 또 그 작은 세부 문제들을 더욱 작은 세부 문제들로 나눠 접근하는 과정과 유용하게 연결된다. 스크래치 학습환경에 대한 소개 후 기본적인 코딩 개념에 대한 학습과 더불어 그래픽 사고를 위하여 기본적인 개념을 소개하는 과정이 필요하다. 다음은 애니메이션과 게임 등 프로젝트 구성을 위해 기본적으로 인지하면 좋을 몇 가지 내용들을 차례로 안내하고자 한다.

② 무대(stage)에 대한 이해: 멀티미디어 프로젝트의 공간적 구성

Flash, Processing, Unity 등 전문가용 멀티미디어 프로젝트 제작환경들이 사용자에게 프로젝트를 구성할 수 있는 무대(화면)의 크기와 방향(horizontal, vertical)을 자유로이 정하도록 허락하는데 비해, 스크래치는 무대 크기가 고정되어 있다. 이처럼 무대 크기가 고정되면 애니메이션이던 게임이던 그 디자인 구성이 더욱 중요해지기 마련이다. 무대는 제작된 게임이 플레이되고, 캐릭터가 움직이는 애니메이션이 재현되는 공간이다. 간단한 애니메이션은 학습자 스스로 무대 위에 혹은 그림판과 같은 모양영역에 바로 그림을 그리던지, 아니면 프리셋 저장소나 로컬 컴퓨터, 혹은 웹상에서 이미지 파일들을 선택하여 불러오는 방식으로 무대에 올려 사용하게 될 것이다. 만약 애니메이션이 코딩 스크립트를 통해 구현되거나, 인터랙티브한 게임 플레이가 진행되도록 하려면 이들을 제어하는 프로그래밍이 필요하다. 따라서 무대는 시각적이며 수동적으로 그려져 구성될 수 있는 반면, 경우에 따라서 프로그래밍을 통해 무대구성 요수들이 제어될 수 있음이 적절히 안내되어야 한다. 또한 학습자의 이해가 높아지면서 점차 이런 프로그래밍적 제어가 어떠한 접근을 통해 해결하여갈 수 있을지 소개되는 것이 좋다.

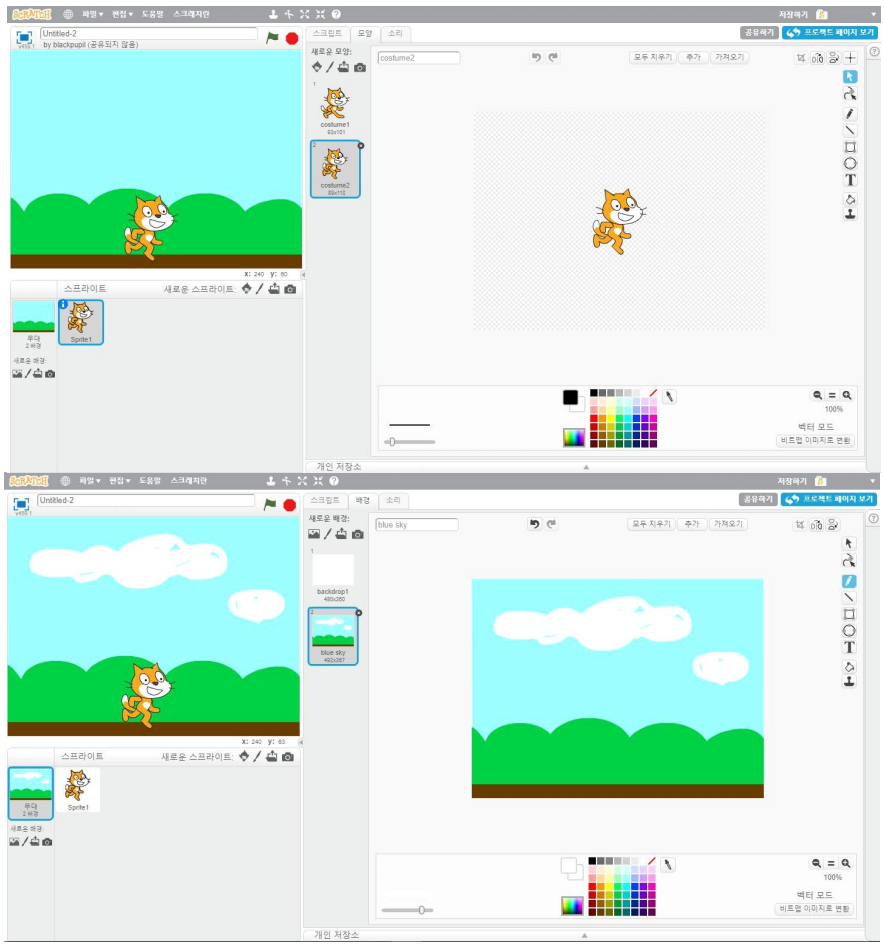
한편, 애니메이션과 게임 오브젝트들의 코딩적 제어를 위해서는, 무대를 2D 화면이라 볼 때 X, Y 좌표계(coordinates system)에 대한 이해가 필요하다. 무대에서 객체 스프라이트가 공간적으로 이동할 때 좌표계에서의 중심(x=0, y=0)이 어디인가에 따라, 공간과 방향의 이동에 대한 매핑은 달라진다. 화면 위로의 이동은 x축의 증가, 화면 아래로의 이동은 x축의 감소라는 식의 이해가 필요하다. 다시 말해, x축 +이동과 −이동, y축 +이동, −이동 등의 이해가 요청되는 것인데, 이는 음수 개념(minus number)에 대한 이해를 동반한다. 그런데 음수개념은 6-7세 정도나 초등학교 저학년 학습자들의 정규 수학 교육과정에서 소개되지 않는다. 따라서 현재 어린 학습자를 상대로 한 많은 코딩 초기학습환경들은 이런 음수개념에 대한 이해를 피하고자 +/- 이동 대신 위/아래(up/down) 혹은 오른쪽/왼쪽(right/left) 이동 등의 방식과 이러한 방향성에 기반 한 자연어 스크립팅을 통해 접근하고 있다. 그러나 오브제 자체의 시각적 모습이 특정 방향성을 가질 경우, 그리고 이들이 무대 안에서 회전 이동을 포함하는 등 그 움직임의 재현이 복잡해질수록, 오히려 오른쪽, 위쪽 등의 접근이 X, Y 좌표계에서의 이동보다 더 혼란스러워 질 수 있다, 본 연구자가 관찰을 통해 발견한 바로는 8-9세의 학습자들에게 이러한 음수 개념은 수 개념으로 접근되기 보다는 물체의 움직임을 통해 직관적으로 이해되도록 유도하는 방법으로 시도되면 충분히 이해될 수 있다.

한편, 무대상에서의 이동 개념과 더불어 무대 안과 밖의 개념도 함께 이해될 수 있다. 이는 무대를 하나의 사각 화면영역, 사각 프레임으로 보는 접근을 요구한다. 이는 그래픽 영상에서 전문적 용어로 미장센에 대한 이해로 설명될 수 있는데, 스크래치에서는 이와 관련하여 무대를 구성하는 요소로서 배경(backdrop), 그리고 배경 위의 캐릭터 등 오브제와 배경의 상호 관계를 적절히 이해하는 것으로 접근될 수 있다. 이는 다음에서 보다 자세히 논의하겠다.

③ 스프라이트 오브제(sprite)와 배경(backdrop)의 관계 이해

무대에 '오브제'라 불리는 객체 스프라이트를 배치하는 것은 어쩌면 스크래치 프로젝트 제작에서 코딩블록을 올려놓는 것보다도 학습자가 어쩌면 더 먼저 접하게 되는 최초의 스크래치 경험이 될 수 있다. 처음으로 스크래치를 열면, 학습자는 디폴트(default)로 무대 위의 서 있는 고양이를 만나게 된다. 이 때 스크래치는 무대 위의 고양이가 동일하게 스프라이트 창에서도 보이게끔 배치하여, 이를 통해 무대와 스프라이트 사이의 관계를 직관적으로 파악하도록 유도한다. 그리고 이 스프라이트 영역 안에 고양이와 더불어 몇 개의 '코스츔(costumes)'이라 불리는 스프라이트 시각적 외관을 보여주는 일련의 이미지들을 쭉 늘어놓고 있어, 이러한 이미지들을 통해 고양이가 움직이는 것 같은 애니메이션 연출과 구성을 이해할 수 있도록 한다. 이는 학습자들에게 무대 위에 놓이는 대상이 스프

라이트를 통해 움직임을 가지는, 즉 애니메이션이 만들어지는 객체들로 만들어질 수 있다는 것을 암시한다. 그러나 어린 학습자들에게 이러한 관계를 직관적으로 이해할 것을 기대하는 것은 쉽지 않다. 또한 그들이 스크래치에서 디폴트로 보여주는 고양이 외에 새로운 종류의 스프라이트 오브제(sprite)를 다루고자 희망할 경우, 이러한 스프라이트 구성에 대한 이해와 무대상의 움직임에 대한 관계가 자세히 설명되지 않는다면, 학습자들은 바로 인지적 혼란을 경험할 수 있다. 또한 스스로가 그림을 그려 개성적인 오브제 스프라이트를 만들고자 할 경우, 코스츔 영역에서 그림을 그리기 위해 스크래치에서 제공하는 그래픽 팔레트 등을 이용하기 마련인데, 그래픽 팔레트에 대한 이용법 등이 그들에게 이해되지 않으면, 학습자들은 또 다른 인지적인 혹은 경험적인 혼란을 경험한다. 반대로 이 일련의 과정들을 수월하게 진행하면 학습자들은 흥미를 잘 유지할 수 있을 것이다. 그러나 연구자는 이 과정 가운데 학습자들이 원하는 단계로 접근하지 못하거나 원하는 시각적 결과를 얻지 못해 다소 의기소침해지거나 흥미를 잃게 되는 경우를 보게 되었다.[30]



〈그림 1〉 스크래치 인터페이스, 위)고양이 스프라이트의 '코스츔(모양)'을 보여주는 창, 아래) '배경'을 보여주고 꾸밀 수 있는 창

한편, 스프라이트 창에는 한쪽 옆에 '무대'라고 안내되는 탭이 함께 있어, 이를 누르면 스프라이트일 때 '코스츔(costumes)'로 활성화된 창이 '배경(backdrop)'이란 창으로 바뀐다. 이를 통해 배경 역시 하나의 스프라이트처럼 객체로서 제어될 수 있음이 구조적으로 드러난다. 그러나 이 역시 복잡한 멀티미디어적 인터페이스에 대한 원활한 이해가 필요한 부분이라 생각된다. 배경 이미지의 선택 역시 스크래치의 저장소나 학습자 컴퓨터의 로컬 파일을 활용할 수 있으나, 학습자가 자신만의 그림을 그

30) 본 연구자는 선행연구에서 개별학습자 인터뷰를 통해 이 단계에서 원하는 시각적 결과를 얻지 못하여 흥미를 잃는 학습자를 관찰하였다.(이현진, 2017, pp. 493-499)

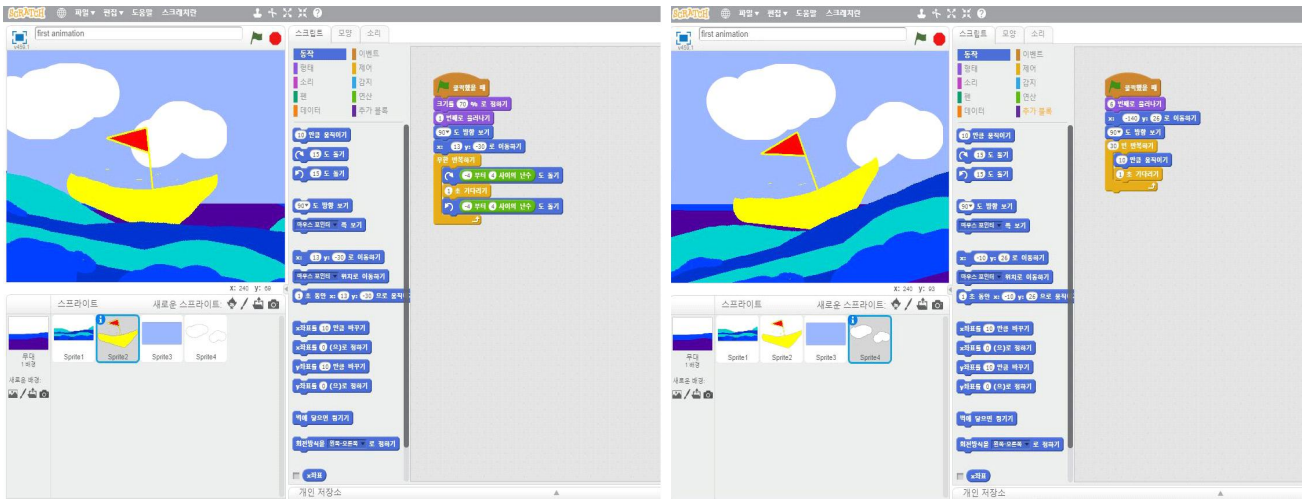
려 배경을 만들고자 할 경우, 그래픽툴을 통해 그릴 수 있으며, 이 배경이미지도 스프라이트와 같이 스크립트 제어를 통해 코딩으로 배경 바꾸기를 시도할 수 있음도 전달되어야 한다.

한편, 이처럼 스프라이트와 배경 이미지는 인터페이스가 유사하게 구조화되어 동일하게 접근될 수 있다. 그러나 그 접근이 동일하다고 하여도, 스크립트상 이들을 제어하는 각각의 코딩블록이 다르듯이, 그 개념도 디자인적 사고에서 분리되어 학습자들에게 소개될 필요가 있다. 스프라이트 오브제와 배경의 관계는 그래픽적 구성과 운용 측면에서 전경(foreground)와 후경(background)의 관계로 나누어 접근될 수도 있을 것이다. 애니메이션과 게임 플레이를 구현하기 위해서는 무대 위의 배경, 그리고 배경 위의 오브제를 이동시키는 방법이외에도, 오브제는 정지시켜놓고 배경 자체를 이동시키는 방법이 있다. 가령, '자동차 경주' 게임에서 경주용 차들을 화면위에 세로 배치할 것인가, 가로 배치할 것인가, 그리고 차를 이동시켜 움직임을 표현하고자 하는가, 아니면 배경을 움직여 차가 움직이는 것과 같은 효과를 만들고자 하는가에 대한 결정과 판단들은 달라질 수 있다. 이는 얼핏 시각적인 판단과 선택일 듯 여겨지지만, 이들은 각각의 경우에 따라 향후 학습자들이 코딩 과정에서 각각 어떠한 수준의 도전적인 문제들을 마주하게 될 것인가를 좌우하게 되는 등이 매우 크게 달라질 수 있는 선택이 된다는 것이다. 즉 시각적 구성에 따라 코딩 과정에서 당면하는 문제들이 매우 다르며, 이에 따라 문제해결 역시 매우 다르게 접근되는 결과를 낳게 된다. 다만, 이러한 디자인적 결정과 판단, 그리고 코딩적 접근 사이의 관계는 어린 학습자들에게 미리 인식하거나 이해되기는 결코 쉽지 않다. 이는 많은 멀티미디어 프로젝트 디자인과 제작 경험을 통해 얻어지는 이해이기 때문이다. 본 연구자는 실제 많은 학습자들이 이러한 관계를 전혀 고려하지 않으며, 깊이 있는 디자인적 판단을 하지 않는 채, 바로 화면 위에 대상 배치를 하며 프로젝트를 시작하는 것을 관찰하였다. 대부분은 옆의 친구나 그들이 본 샘플 프로젝트 혹은 그들이 경험한 게임의 방식을 그대로 답습하여 큰 고민 없이 디자인적 배치를 시작하곤 하는 것이다. 만약 이러한 시지각적 관계에 대한 인식이 문제해결적 접근을 목표로 한 코딩적 접근과 연결되어, 프로젝트 제작 중심의 코딩학습 커리큘럼을 통해 단계적이고 체계적으로 접근될 수 있다면, 이는 보다 종합적으로 사고하는 학습태도를 긍정적이고 효과적으로 변화시킬 수 있을 것이다. 즉 오브제와 배경 등의 디자인적 배치를 중심으로 시각적 사고와 코딩사고를 결합할 수 있으며, 그로 인한 다양한 구성과 그에 따른 결과가 연결되어 접근된다면, 그 자체로 문제해결력에 대한 이해와 접근 을 경험시킬 수 있는 교육적인 커리큘럼이 개발될 수 있다. 이는 학습자의 인지발달 단계와 연령, 프로젝트의 방향성에 따른 상위 레벨의 학습단계에서 접근될 필요가 있겠다. 하지만, 스프라이트 오브제와 배경의 관계는 어린 학습자에게도 소개된다면 그들의 기초적인 수준의 프로젝트 개발에서도 유용할 수 있는 내용임은 분명하다.

④ 애니메이션을 위한 그래픽 개념

애니메이션 작업을 시도하는 많은 학습자들의 경우, 오브제를 무대 위 혹은 '모양(costumes)'창 위에 올려놓고 이를 조금씩 이동시키거나 변형시키는 방법을 통해 프로젝트를 시작한다. 이러한 하나하나의 이미지들이 각각 서로 다른 costume 이미지로 저장되면, 후에 플레이 버튼이 눌러질 때 자동적으로 연결되어 재생된다. 이는 스프라이트 외관 변화 통해 움직임을 구현하는 것으로, 이러한 과정에서 '프레임'의 개념이 소개가 될 수 있겠다. 클레이아트 애니메이션(clay art animation) 혹은 셀 애니메이션(cell animation)과 같은 한 프레임 한 프레임을 이어가면서(frame by frame 개념으로) 오브제나 배경의 조작을 통해 스토리와 구성을 이어가는 방식들을 소개할 수 있겠다. 이러한 소개는 학습자들이 애니메이션 구성 원리에 대해 더욱 깊게 이해하도록 돕는다. 한편, 미세한 그래픽 이미지 조정을 통해 그래픽 툴에 대한 학습을 자연스럽게 유도할 수 있다. 지우기와 그리기, 색을 바꾸기, 크기 변형하기 등의 간단한 조작이 스크래치 상에 주어진 그래픽 편집툴을 통해 가능하다. 한편, 앞서도 간단히 설명하였지만, 학습자들에게 애니메이션을 만들기 위한 또 다른 방법으로 위에서 만든 오브제들을 간단한 코딩을 통해 제어하는 방법 등을 이어 소개할 수 있다. 이는 코딩에서

어린 학습자들이 쉽게 이해할 수 있는 시퀀싱 개념이 접목되는 것으로 컴퓨팅 사고를 통해 이동하는 애니메이션이나 이미 만들어진 이미지를 무대 안에서 재구성할 때 오브제의 크기나 위치의 변경 등을 스크립트를 통해 제어하도록 하는 것이다(〈그림 2〉 참조). 이 때, 애니메이션의 제작을 위하여 프레임(frame)에 대한 개념을 확장하여 프레임 안과 밖의 개념(앞서도 설명한 미장센에 대한 개념)도 연장하여 소개하면 좋겠다. 즉 프레임 안에 등장한 오브제는 씬(scene)에 등장하는 것(게임 씬이나 애니메이션 씬 안에 등장하는 것), 그렇지 않고 프레임 밖에 존재하거나 프레임 밖으로 빠져나가면 이는 아직 등장하지 않았거나 잠재적으로 등장할 수 있는 가능성이 있거나 혹은 사라진 오브젝트가 될 수 있음을 인지시키는 것이다. 이러한 인지는 위에서 설명한 무대에 대한 이해와 이동에 대한 개념들이 더불어 안내될 수 있으며 이에 따라 다양한 시각적 연출을 시도하도록 유도할 수 있다. 이 단계보다 좀 더 복잡한 인지와 사고가 가능한 학생의 경우, 오브제와 오브제가 서로 더욱 복잡한 관계를 맺으며 제어되는 방식으로 학습의 단계를 발전시켜갈 수 있다. 이때는 오브제들이 서로 상호작용하거나, 오브제와 배경과의 관계를 변화시키며 다양한 구성을 시도하도록 유도할 수 있겠다. 이는 뒤의 '⑥ 시각적 대상의 인터랙션 구현'에서 보다 자세히 설명하겠다.



〈그림 2〉 스크립트 코딩을 통해 애니메이션을 구현하는 설명하기 위해 본 연구자가 만들고 활용한 교육자료 예

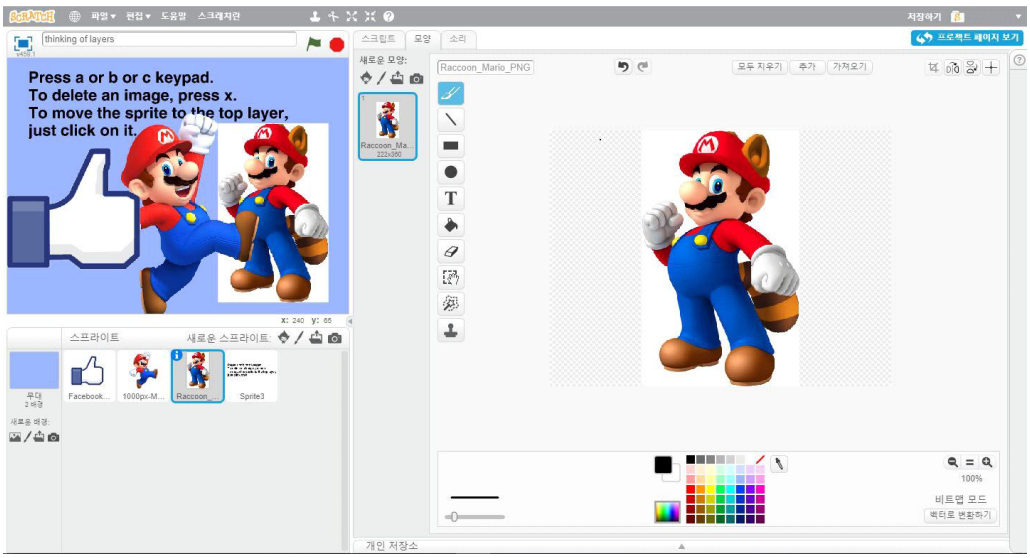
⑤ 레이어 및 투명도

앞서 코딩교육에서 연령 혹은 학년별로 학습자의 인지 발달 수준과 이해정도의 차이를 비교적 자세히 구분하여 논의하는 것과 비교하여, 그동안 시지각 경험과 지식, 그래픽 개념에 대한 이해 정도와 지식수준은 연령 혹은 학년별로 어떤 발달 수준과 이해 정도의 차이를 가지는지 그다지 체계적이고 정교하게 논의되지 않아왔다. 가령 실제 사계에서 물리적으로 앞에 놓인 대상과 뒤에 놓인 대상을 어떻게 인식하여 그림으로 표현하는가, 혹은 그려진 대상들의 순서를 어떻게 인지하는가 등의 논의는 심리학 연구 혹은 인지 발달 연구 등에서 간혹 발견되긴 하지만, 시각 교육에서는 직관적으로 접근될 뿐 그리 체계적으로 논의되지는 않았다.[31]

하지만 오브젝트 대상의 크기와 배열은 시각적 주제적 의도에 따른 게임이나 애니메이션 화면을 구성할 때 중요하게 접근되어야 하는 개념 중 하나이다. 즉 화면 위에 대상이 놓이는 순서가 서로 뒤바뀌고 재배치되며 이야기가 전개되거나 게임이 진행될 수 있기 때문이다. 이는 그래픽 작업 과정에

31) Charles Crook(1984)은 투명도(transparency)에 대한 이해는 학년과 나이에 따라 차이가 난다고 밝힌다. 오브제의 직선적 일렬 배치에 따라 보는 시점을 다르게 적용하는 접근이 나이에 따라 다르게 관찰된다는 것이다. 4-5세 정도의 어린 학생들은 정렬 혹은 수열적 방법(array-specific manner)을 잘 이해하고 적용하지 못하지만 이것이 6-7세 정도에는 어느 정도 습득된다. 또한 가려진 오브젝트에 의한 처리와 레이어에 의한 투명도 처리에 대한 이해 역시 학년과 나이에 따라 다르게 접근된다고 한다.

서는 ‘레이어(layer)’ 개념으로서 접근되고 처리된다. 그런데 레이어 개념은 어린 학습자들에게 익숙하지 않다. 대부분 미술시간에 종이위에서 색종이나 크레파스, 물감을 다뤄 본 경험을 바탕으로 한, 물리적인 레이어를 다뤄 본 정도의 경험을 가지고 있을 것이다. 그러나 이런 아날로그적이고 물리적인 경험과 디지털 컴퓨터 그래픽 세계에서의 경험은 크게 다르다. 물리적인 방식으로 레이어를 쌓으면, 가령 서로 다른 색종이들을 풀로 붙이거나, 크레파스로 색칠된 색깔 층의 순서를 되돌리고 뒤바꾸기는 쉽지 않으나, 디지털 방식으로는 그 순서를 쉽게 뒤바꿀 수 있기 때문이다. 이러한 디지털 그래픽에서의 레이어 개념은 간단한 예시를 통해 어린 학습자에게 효과적으로 이해 전달시킬 수 있으며, 그들 스스로 이들을 애니메이션 표현이나 게임 오브제의 제어(오브제를 화면 안에 등장하고 사라지게 하거나, 보이고 안보이게 제어)에 유용하게 사용하도록 안내 할 수 있다. 이는 여러 객체 스프라이트를 무대화면 위에 배치 구성하고 서로 관계를 가지도록 연출하는데 매우 유용한 것이다. 또한 이와 연장선상에서 투명 레이어의 개념, 즉 투명의 개념과 투명 배경을 함께 가진 형태의 대상을 사용하는 법, 그리고 이러한 투명한 속성을 담고 있는 PNG 파일 등의 그래픽 파일 개념도 함께 소개될 수 있을 것이다. 특히 외부에서 가져온 이미지를 부분적으로 사용하거나 배경과 분리하여 사용하고자 할 때도 배경을 지워버리는 방법, 혹은 투명 배경이 함께 저장되고 기능하는 그래픽 파일 형태를 이용하는 법이 설명될 수 있다(〈그림3〉 참조).



〈그림 3〉 오브제 스프라이트를 통해 레이어의 개념 및 투명 레이어의 개념을 설명하기 위해 연구자가 만들고 활용한 교육 자료 예

⑥ 시각적 대상의 인터랙션 구현

학습자 관찰을 통해 확인한 바로는 주로 취학 전 혹은 초등학교 저학년 학습자들에게는 인터랙티브한 표현이 추가되지 않은 순수한 애니메이션 프로젝트 제작이 편하게 접근되고 완결될 수 있었다. 코딩을 통해 애니메이션을 제어하거나 스프라이트 대상 간의 상호작용을 구현하는 등의 게임 프로젝트는 고학년 학생들에게서 주로 시도되는 것을 볼 수 있었다. 그러나 오늘날 어린 학습자들도 게임 플레이 경험을 많이 가진 경우, 그들의 코딩능력의 유무와 수준에 대한 스스로의 인식과는 무관하게 처음부터 난이도 높은 인터랙티브한 작업을 시도하기도 한다. 이런 학습자들은 직관적이고 본능적으로 자신들의 게임 플레이 경험을 바탕으로 게임과 같은 인터랙티브한 작업을 시도하고자 하는 것이다. 게임에서 플레이어(사용자)의 키보드 인터랙션을 통해 게임이 진행되거나, 스프라이트 내부적 움직임 혹은 애니메이션 동작을 가진 오브제들이 서로 상호관계를 맺으며 그로 인한 결과를 시청각적인 효과로 표현해 내는 방식들은 종종 기획되기 마련이다. 또한 학습자들이 애니메이션을

제어하는 방법으로 코딩을 통해 오브젝트의 움직임을 반복(repetition)할 수 있으며, 캐릭터 오브젝트의 대비와 대조(contrast)를 통한 애니메이션이나 스토리텔링 프로젝트도 기획할 수 있다. 게임 안에서 인터랙션 작용에 따라 반응하거나 물리적 세계의 효과들을 표현하고자 할 경우, 어느 정도의 그래픽 일루전을 통한 표현들이 잘 접목되어야 자연스러운 구현을 이끌어낼 수 있다. 이들의 자연스런 구현을 위해서는 상당한 시각적 표현과, 때에 따라서는 물리적 지식까지도 요구될 때도 있다. 또한 높은 수준의 코딩 지식이 요구되기도 한다. 관객의 상호작용에 반응하는 게임이나 인터랙티브 애니메이션의 경우, 'if then 조건문' 혹은 'for 구문' 등은 많이 사용된다. 또한 오브제들 간의 터치와 충돌 등을 인지(collision detection)하는 시스템도 많이 사용되는 내용 중 하나이다. 이를 수행할 구체적인 코딩 방법론으로는 여러 가지가 있겠으나, 스크래치에서는 색을 감지하여 충돌을 인식하는 코딩블록이 이미 마련되어 있어, 비교적 어린 학습자들도 어렵지 않게 접근될 수 있다. 그러나 이런 충돌 인지를 위해서도 어떤 색을 서로 인식시키는 것이 좋을지, 충돌 감지를 위한 요소를 대상의 어디에 위치시킬 것인가, 즉 스프라이트의 어느 위치에 충돌 감지를 위한 장치를 어떻게 배치하면 좋을지 등의 디자인적 기획이 요구된다. 이 역시 하나의 문제해결을 위해 코딩과 그래픽 사고와 지식이 종합적으로 결합되어 요청되는 경우이다.

⑦ 프로젝트 완결 이끌어내기

앞서도 언급한 것과 같이 학습자의 문제해결력을 증진시켜 그들이 수행하고자 하는 프로젝트를 완결할 수 있도록 이끌어주는 것은 학습자의 성취도를 이끌어서 학습에 대한 흥미를 이끌어 내는 효과적인 접근이다. 이는 단지 코딩지식을 가르치기보다 코딩을 통해 더 큰 학습능력을 갖추는 것과 방향을 같이 한다. 그런데 게임이나 애니메이션 프로젝트에서 학습자들은 진행 상태(status)에 따른 다양화된 씬의 구성이 순조롭게 연결되길 원한다. 게임시작, 플레이 방식소개, 게임레벨 변화, 게임종료 등이 페이지들이 원활하게 전환되고, 또한 이를 위해 그들이 구성한 플레이의 게임 점수 시스템도 의미 있게 제어되길 원한다. 그리고 캐릭터의 에너지 레벨이나 life time도 게임의 맥락 안에서 의미있게 제어되길 희망한다. 그리고 게임이나 애니메이션이 종료되면 다시 반복 플레이되는 구조를 만들고자 한다. 이를 위해서는 게임 전체가 어떻게 구성되고 그 과정이 어떻게 전개될 지 전체적으로 파악하고 기획하는 메타인지가 요구된다. 구체적으로는 시간 변수, 점수 변수, 목숨 변수 등 변수를 효과적으로 사용하고, 전체 구조가 반복(looping)되기 위해 전체의 프로세스를 구조화시키고, 구조 안의 구조로서 네스팅(nesting) 시킬 수 있는 방법론 등을 알고 있어야 한다. 커리큘럼은 학습자들이 스스로 프로젝트 완결을 해나갈 수 있도록, 이러한 기본개념과 쏘스 코드들, 샘플 예제들을 단계별로 제공할 수 있다. 이를 통해 학습자들이 프로젝트의 완결을 위한 기술적이고 방법론적 내용들을 이해하고 응용하여 각자의 프로젝트에 적용시킬 수 있도록 도울 수 있다. 이는 코딩사고력, 전체 디자인적 기획과 아이디어, 구성 능력 등을 종합적으로 발전시킬 수 있는 길이 될 것이다.

4. 결론

본 연구는 코딩교육에 대한 현재의 다양한 논의들에 대한 고찰에서 출발하여, 코딩교육에서 강조하는 코딩사고, 즉 컴퓨팅 사고력이 시각적이고 멀티미디어적 사고력과 함께 개발되도록 유도되는 융합적 커리큘럼과 그 구성내용을 논하였다. 특히 학습자 인지 발달의 단계에 따른 코딩 커리큘럼을 분석적으로 접근 하는 선행연구들을 검토하는 동시에, 연구자 스스로가 코딩학습자 관찰을 통해 경험하고 취득한 내용을 바탕으로, 시각적 사고 및 그래픽과 멀티미디어적 사고가 코딩학습에 함께 결합될 수 있는 프로젝트 제작과 개발 중심의 커리큘럼 내용을 검토하고자 하였다.

프로젝트 중심의 커리큘럼은 기본적으로 학습자가 원하는 프로젝트를 기획하고 제작하도록 이끌어주어야 한다. 그러나 이에도 좀 더 학습자의 나이와 인지능력에 따른 세분화된 접근이 필요하다. 비교

적 어린 학습자들에게는 시퀀스에 대한 이해를 바탕으로 한 간단한 애니메이션 프로젝트를 제작하게 하고, 점차 코딩 학습레벨이 상위레벨로 진행된 경우 혹은 초등학교 고학년 학습자의 경우, 코딩으로 간단한 인터랙션을 제어할 수 있는 게임 제작 혹은 애니메이션 제작이 유도될 수 있다. 또한 더욱 상위레벨로 향하는 학습자에게는 점차 보다 복잡한 기획이 요청되는 게임구성을 유도할 수도 있다. 그리고 교육적 목적과 효과의 측면에서 볼 때, 위의 단계별 과정에서의 교육 내용은 무엇보다 학습자가 시도하고 기획하는 프로젝트가 완결될 수 있도록 이끌어가고, 이를 위한 과정에서 학습자가 어려움을 느끼는 부분을 잘 파악하고 이를 해결하도록 도와주는 것에 집중되어야 한다. 또한 이 과정에서 인지적 사고력 발달과 학습 진도에 따라 각 단계에서 유용하게 적용될 수 있는 그래픽과 코딩 사고가 융합적으로 접목될 수 있는 학습 내용과 지식이 전달될 필요가 있다. 이는 Resnick et al.(2009)과 Taylor et al.(2010) 등 선행 연구자들이 스크래치 프로그래밍 툴을 이용한 코딩교육을 통해 학생들의 창의력과 반영적인 사고력을 함양시키고, 문제해결 과정을 향상시키어 궁극적으로 공부하는 법을 익히게 이끌고자 했다는 점을 다시금 상기하려는 것이다. 이러한 점에서 본 연구는 융합적인 코딩교육뿐만 아니라 이 가운데 문제해결력과 논리력, 그리고 사고하는 법을 체계적으로 교육, 훈련시키어 보다 폭넓은 학습력 신장을 꾀하는 것을 그 중심으로 삼고, 이를 위한 커리큘럼 구성내용들을 탐색하는 연구였다고 볼 수 있다. 향후 연구에서는 본 연구에서 논의한 커리큘럼 구성내용을 바탕으로 여러 학습단계 혹은 발달단계로 나누어 보다 체계적인 커리큘럼으로 개발할 필요가 있다. 또한 학습자들에 성향에 따라 구분해 접근하거나, 특별한 영역에 재능을 가진 학습자들을 따로 구분하여 각기 다른 커리큘럼을 적용하는 교육 방법론도 함께 모색되어야 할 것이다.

Duncan et al.(2014)은 어린 학습자들이 코딩교육에 입문하는 과정에서의 경험이 특히 매우 중요함을 강조한다. 이들에게 친절하고 실력과 자신감을 가진 교사로부터 코딩교육이 이루어질 경우와 그렇지 않을 경우, 학습자가 코딩에 대하여 느끼는 흥미와 성취도, 친밀도의 차가 매우 크게 달라진다는 것이다. 이러한 면에서 본 연구가 제시하는 융합적 접근을 시도하는 코딩교육에서도 그래픽 및 멀티미디어적 작업 경험과 지식, 그리고 코딩 지식과 경험을 동시에 가진 전문적이고 능력 있는 교사의 역할은 매우 중요하다. 본 연구에서 제시한 커리큘럼의 내용들을 구성하고 가르칠 교사들은 학습자가 각자 프로젝트를 통해 어떤 디자인적 선택과 아이디어를 전개해 나가는가에 따라 코딩의 난이도가 매우 달라질 수 있다는 사실을 인지하고 이에 따른 적절한 방향성을 유도해야 한다. 또한 각각의 상황에서 학습자가 느끼고 해결하고자 하는 문제에 대한 시기적절한 도움을 주는 역할을 수행해야 한다. 따라서 실제 교육 현장에서 이러한 역할을 담당할 교사양성 커리큘럼도 함께 지속적으로 연구되어야 한다.

참고문헌

도서

- 루돌프 아른하임 저, 김정오 역, 시각적 사고(Visual Thinking), 이화여자대학교 출판부, 2004
- Marina Umaschi Bers, Designing digital experiences for positive youth development: From playpen to playground. Oxford University Press. 2012
- Marina Umaschi Bers & Mitchel Resnick, The Official ScratchJr Book: Help Your Kids Learn to Code. No Starch Press, 2015
- Paul Laseau. Graphic Thinking for Architects and Designers, Van Nostrand Reinhold, 1989
- Sergio Morra, Camilla Gobbo, Zopito Marini & Ronald Sheese, Cognitive development: neo-Piagetian perspectives. Psychology Press, 2007
- Seymour Papert, Mindstorms: children, computers, and powerful ideas. Basic Books, 1980
- Steven, L. Tanimoto, An Interdisciplinary Introduction to Image Processing: Pixels, Numbers, and Programs, MIT press. 2012

논문

- 이현진, 스크래치 개별학습자 인터뷰를 통해 알아본 코딩교육에서의 융합적 접근 필요성 연구, 기초조형학연구, 18(6), 487-500. 2017
- Marina Umaschi Bers, Project InterActions: A multigenerational robotic learning environment. Journal of Science and Technology Education, 16(6), 537-552. 2007
- Grady Booch, To Code or Not to Code, That Is the Question, IEEE Software, 9-11. 2014
- Hui-Chun Chu & Gwo-Jen Hwang, Development of a project-based cooperative learning environment for computer programming courses. International Journal of Innovation and Learning, 8(3), 256-266, 2010
- Sabahattin Ciftci & Ayse Aysun Baykan, Project based learning in multi-grade class. Educational Research and Reviews, 8(3), 84-92. 2013
- Charles Crook, Factors influencing the use of transparency in children's drawing, British Journal of Developmental Psychology, 2(3), 213-221. 1984
- Caitlin Duncan, Tim Bell & Steven L. Tanimoto, Should your 8-year-old learn coding?, Proc. of the 9th Workshop in Primary and Secondary Computing Education, ACM. 60-69. 2014
- Chun-Ming Hung, Gwo-Jen Hwang & Iwen Huang, A Project-based Digital Storytelling Approach for Improving Students' Learning Motivation, Problem-Solving Competence and Learning Achievement. Educational Technology & Society, 15(4). 368-379. 2012
- Elizabeth Kazakoff & Marina Bers, Programming in a robotics context in the kindergarten classroom: The impact on sequencing skills. Journal of Educational Multimedia and Hypermedia, 21(4), 371-391. 2012
- Thom Markham, John Larmer & Jason Ravitz, Project Based Learning, Handbook: A Guide to Standards-Focused Project Based Learning, 2nd ed., Buck Institute for Education, 2003
- Dylan J. Portelance, Amanda L. Strawhacker, & Marina Umaschi Bers, Constructing the ScratchJr programming language in the early childhood classroom, International Journal of Technology and Design Education, 26(4), 489-504. 2016
- Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman & Yasmin Kafai, Scratch: Programming for all. Communications of the ACM, 52(11), 60-67. 2009
- Scott P. Schaffer, Xiaojun Chen, Xiumei Zhu & William C. Oakes, Self-efficacy for crossdisciplinary learning in project-based teams. Journal of Engineering Education, 101(1), 82-94, 2012
- Linda Seiter & Brendan Foreman, Modeling the learning progressions of computational thinking of primary grade students. Proc. Ninth annual Int'l. ACM Conf. on Computing Educ. Research, 59, ACM, 2013
- Neil Smith, Clare Sutcliffe & Linda Sandvik, Code club: Bringing programming to uk primary schools through scratch. Proc. 45th ACM Technical Symposium on Computer Science Education, SIGCSE '14, 517-522, ACM, 2014
- Merilyn Taylor, Ann Harlow & Michael Forret, Using a computer programming environment and an interactive whiteboard to investigate some mathematical thinking. Proc. of Social and Behavioral Sciences, 8, 561-570. 2010
- Hsiu Ying Wang, Iwen Huang & Gwo Jen Hwang, Effects of an Integrated Scratch and Project-Based Learning Approach on the Learning Achievements of Gifted Students in Computer Courses, IIAI 3rd International Conference on Advanced Applied Informatics, Kitakyushu, 382-387. 2014

인터넷 사이트

- 스크래치 http://scratch.mit.edu/
- Code.org https://code.org/
- Computer Science Teachers Association http://csta.acm.org/