**Final Project: Sorting Game**

전공: 수학 학년: 4 학번: 20161255 이름: 장원태

1. **프로젝트 목표**

오픈프레임웍스를 이용하여, 폰트를 화면에 출력해보고, 무작위(random) 알고리즘을 구현하며, 제한 시간이 있는 간단한 게임을 구현한다. 이번 프로젝트에서는 블록들을 같은 색이 있는 타일로 보내 분류하는 게임을 구현하였다.

1. **프로젝트 수행 환경**

Visual Studio 2019 – Debug, Win32

1. **각 변수에 대한 설명**

다음은 ofApp.h 파일에 선언되어 있는 변수들이다.

ofTrueTypeFont font : 폰트를 불러와 저장한다.

Float height: 윈도우의 높이를 저장한다.

Float width: 윈도우의 너비를 저장한다.

(이하 centerx와 centery는 후술할 타일 배치를 용이하게 하기 위해 따로 선언했다.)

Float centerx: 윈도우의 정 가운데의 x좌표를 저장한다.

Float centery: 윈도우의 정 가운데의 y좌표를 저장한다.

Int life: 게임 내에서 주어지는 life의 개수를 저장한다. 라이프는 시작할 때 3개가 주어지며, 0개가 되면 게임 오버가 된다.

Int record: 이번 게임에서 분류에 성공한 블록의 개수를 저장한다.

Int Bestrecord: 저장되어 있는 최고 기록을 BestRecord.txt에서 불러와 저장한다.

Enum colors tile[3]: 순서대로 왼쪽, 위쪽, 오른쪽에 있는 세 개의 타일의 색깔을 저장한다.

Enum colors block[4]: 위에서부터 4개의 블록의 색깔을 저장한다.

Bool gamestart: 게임을 시작했는지 여부를 저장한다. 이는 화면 전환 및 게임 내 기능 활성화를 위한 변수이다.

Bool gameover: 게임오버가 되었는지 여부를 저장한다. 이는 화면 전환 및 게임 내 기능 비활성화를 위한 변수이다.

Bool gamebegun: 게임이 시작되었는지 여부를 저장한다. 이는 프로그램의 elapsed time을 게임 시작 시로 초기화해주기 위한 변수이다.

Int correct: 특정 방향키를 입력했을 때, 입력한 방향에 있는 타일의 색과 현재 블록의 색이 일치하는지를 판별하는 변수이다. 초기 상태 및 판단 대기 시에는 -1로 설정되어 있으며, 0으로 설정되면 타일과 블록의 색이 불일치함을, 1로 설정되면 타일과 블록의 색이 일치함을 의미한다.

1. **자료구조 및 알고리즘 상세히 설명**

게임의 전체적인 흐름은 tile[3] 배열과 block[4] 배열에 의해 흘러가며, gamestart, gameover, gamebegun 변수에 의해 조정된다. 가독성을 높이기 위해 enumeration type으로 colors 타입을 선언해주고, Red, Green, Blue를 각각 0, 1, 2 값으로 배정해주었다.

우선, gamestart가 false인 경우는 게임의 초기 화면을 보여준다. 초기 화면에는 게임의 제목과, 조작법은 콘솔을 확인하라는 확인 메시지, 시작하려면 ‘s’ 키를 누르라는 안내 메시지, 그리고 제작자 이름이 적혀 있다. 이 일련의 글씨들은 ofTrueTypeFont font를 이용하였으며, maze 실습 때 사용한 verdana.ttf 폰트 파일을 사용하였다. 그리고, 콘솔에 조작법을 출력해주고, 동시에 최고 기록도 출력하기 위해 BestRecord.txt 파일에서 최고 기록을 읽는다. 그 후, 그것을 Bestrecord 변수에 저장한 뒤 콘솔에 출력해준다.

이제 ‘s’ 키를 누르면 gamestart가 true가 되며 게임 화면이 뜬다. 여기서 우선적으로 필요한 것은 elapsed time reset이었다. 이는 게임 내의 제한 시간을 구현하기 위해 ofGetElapsedTimef() 함수를 이용해야 하는데, 이미 게임 시작 화면에서 시간은 흘러갔다. 이 때, default값이 false였던 gamebegun도 동시에 true로 바꾸어준다. 이에 따라 update() 함수에서 gamebegun이 true일 때의 과정을 수행하게 되는데, 이 때 ofResetElapsedTimeCounter() 함수를 통해 elapsed time을 초기화해준다. 이 때, 화면을 유지하기 위해 gamestart는 true가 되어야 하지만, elapsed time 초기화는 단 한 번만 이루어져야 하므로 gamebegun을 false로 바로 바꿔줌을 통해 중복 초기화를 방지한다.

이제 시간은 설정이 되었고, 화면에 여러 정보를 그려주어야 한다. Draw 함수에서 gamestart가 true가 되었고 아직 gameover는 false인 상태이므로, else if 구문에 들어가 타일과 라이프, 시간을 그려준다. 블록도 그려주어야 하는데, 앞서 3초를 카운트 한 뒤 게임을 시작할 때 블록이 나타나게 구현하기 위해 elapsed time이 3초 이상일 때만 블록을 그려주게 하였다.

이 때 타일과 블록은 각 색깔이 무작위로 배정되어야 하므로, rand() % 3을 통해 그 값을 정한다. 그 값이 0이면 빨강색으로, 1이면 초록색으로, 2이면 파란색으로 결정한다. 타일의 경우는 초기에 설정되고 더 이상 바뀌지 않으므로, setup 함수에서 setTileColor 함수를 호출하여 타일 색을 단 한 번만 설정해준다. 그리고 draw 함수에서 매 프레임마다 해당 타일들을 그려준다. 블록의 경우는 초기 뿐만 아니라 블록 분류에 성공할 때마다 색깔을 update해주어야 한다. 초기의 블록은 타일과 마찬가지로 setup 함수에서 InitBlockColor 함수를 호출하여 그 색깔들을 결정한다. 이후에는 색이 update되어야 한다. 4개의 블록 중 1번째 블록 분류에 성공하게 되면, 2~4번째 블록들은 각각 1~3번째 블록으로 한 칸씩 이동해야 한다. 그 후, 비어있을 4번째 블록에 대해서는 또 다시 rand() % 3 값을 통해 색깔을 정해준다. 이는 앞서 언급한 블록 그리는 함수(DrawBlock)가 draw 함수에서 매 프레임마다 호출되고 있으므로, block[4] 배열만 이와 같이 update해주면 화면 상에서는 자연스럽게 그 결과가 그려진다.

다음으로 라이프를 보면, 라이프는 초기에 3개로 설정되어 있다가 분류에 실패할 때마다 1개씩 깎인다. 이를 DrawLife 함수에서 구현하여, 라이프를 가리키는 3개의 사각형을 그리되 라이프가 깎이면 오른쪽 사각형부터 사라지게 되므로, life 값에 따라 3개의 사각형 각각을 그릴지 말지 결정한다. 이에 따라 맨 오른쪽 사각형은 life가 3일 때에만, 가운데 사각형은 life가 2 이상일 때, 왼쪽 사각형은 life가 1 이상일 때(즉 game이 진행되는 동안) 그려진다. 이 함수 역시 draw 함수에서 호출되며, life 값을 update 함수를 통해 -1 해주면 그 결과는 화면에 자연스럽게 그려지게 된다.

이어서 시간을 보면, 시간을 가리키는 게이지는 총 6칸으로 구성되어 있다. 각 칸은 5초씩을 가리키게 된다. 라이프와 마찬가지로, 시간을 도식화하여 그리는 함수(DrawTime)도 draw 함수에서 호출되며, 별도의 update 함수 없이 ofGetElapsedTimef() 함수 값이 자체적으로 갱신되므로 자동으로 화면에 그려질 수 있다. 단, 게임이 시작되기 전 3초의 카운트다운이 있으므로, 오른쪽에서부터 3~8초, 3~13초, 3~18초, 3~23초, 3~28초, 3~33초동안 화면에 표시된다. 이 때, 게임 시작부터 28초가 지난 시점부터는 마지막 카운트다운을 시작하며, 이 때는 1초마다 그리는 숫자를 갱신하여 “5”, “4”, “3”, “2”, “1”을 화면에 그려준다.

이제 시간이 33초가 모두 지나거나 life가 모두 소진되면 gameover 값이 true가 되며, draw 함수에서 gamestart인 상태에서 gameover 역시 true가 되므로 else 구문을 통해 게임 오버 화면을 출력한다. 이 상태에서 다시 s를 눌러도, gamestart는 이미 true인 상태이기 때문에 아무 일도 일어나지 않는다.

타일과 블록의 색을 match시키는 과정은 tile[3]과 block[4] 배열을 이용한다. 타일의 색은 tile[3]에 저장되어 있으며, 인덱스 순서대로 왼쪽, 위쪽, 오른쪽 타일을 가리킨다. 이 때, 현재 블록은 block[0]에 저장되어 있다. 이 상태에서 각각 왼쪽, 위쪽, 오른쪽 방향키를 누르면 그 방향에 있는 타일과 block[0]의 색깔을 비교한다. 예를 들어 위쪽 방향키를 누르면 tile[1]과 block[0]의 색을 비교하는 것이다. 이 두 값은 모두 enum colors 타입의 값이므로, 별도의 전환 과정은 필요하지 않다. 이 두 값이 같다면, correct 변수를 1로 설정하고, 다르다면 correct 변수를 0으로 설정한다. 이 correct 변수는 update 함수에서 판정에 이용한다. Correct가 1이고, 게임이 한창 진행중인 상태라면, 블록의 색을 재배열하고 record에 1씩 더하여 타일 분류 성공을 알린다. 게임이 한창 진행중이라는 것은 gamestart가 true이고, gameover은 false이며, 동시에 elapsed time이 카운트다운 이후인 3초 이상인 경우를 의미한다. 그 후부터 매 프레임마다 record가 1씩 더해지는 것을 방지하기 위해, 판정이 끝났다는 것을 correct = -1을 통해 알린다. Correct가 -1이고, 게임이 한창 진행중인 상태라면, 블록은 재배열하지 않고 life만 1 소진시킨 뒤 correct = -1로 설정한다. 이 과정을 게임이 끝날 때까지 반복한다.

어느 때든 게임을 종료하려면 ‘x’ 키를 누르면 된다. 이 때, 콘솔에 record 값, 즉 이번 게임에서 분류에 성공한 타일의 개수를 출력해주며, 만약 저장된 최고 기록보다 그 값이 더 크다면, BestRecord.txt 파일에 그 기록을 갱신해주고 최고 기록 달성을 축하한다는 메시지를 콘솔에 출력해준다. 그 후, 프로그램을 종료한다.

1. **각 함수에 대한 설명**

추가적으로 구현한 함수는 다음과 같다.

Void ReadBestRecord() : BestRecord.txt로부터 최고 기록을 읽어온다.

Void WriteBestRecord(): 최고기록을 넘어선 기록이 나온 경우, BestRecord.txt에 해당 기록을 적는다.

Void printHowToPlay(): 콘솔에 조작법과 최고기록을 출력한다.

Void setTileColor(): 게임 화면에서 왼쪽, 위쪽, 오른쪽에 그릴 타일의 색을 정한다.

Void DrawTile(): 정해진 타일의 색을 바탕으로 그 타일들을 화면에 그린다.

Void DrawLife(): 라이프를 게임 화면 우측 상단에 그린다. 라이프의 개수에 따라 갱신된다.

Void DrawTime(): 게임 중에 남은 제한 시간을 게임 화면 좌측 상단에 그린다. 흘러간 시간에 따라 갱신된다.

Void InitBlockColor(): 네 블록의 초기 색깔을 정한다.

Void ChangeBlockColor(): 블록 분류에 성공한 경우, 블록의 색깔을 갱신한다.

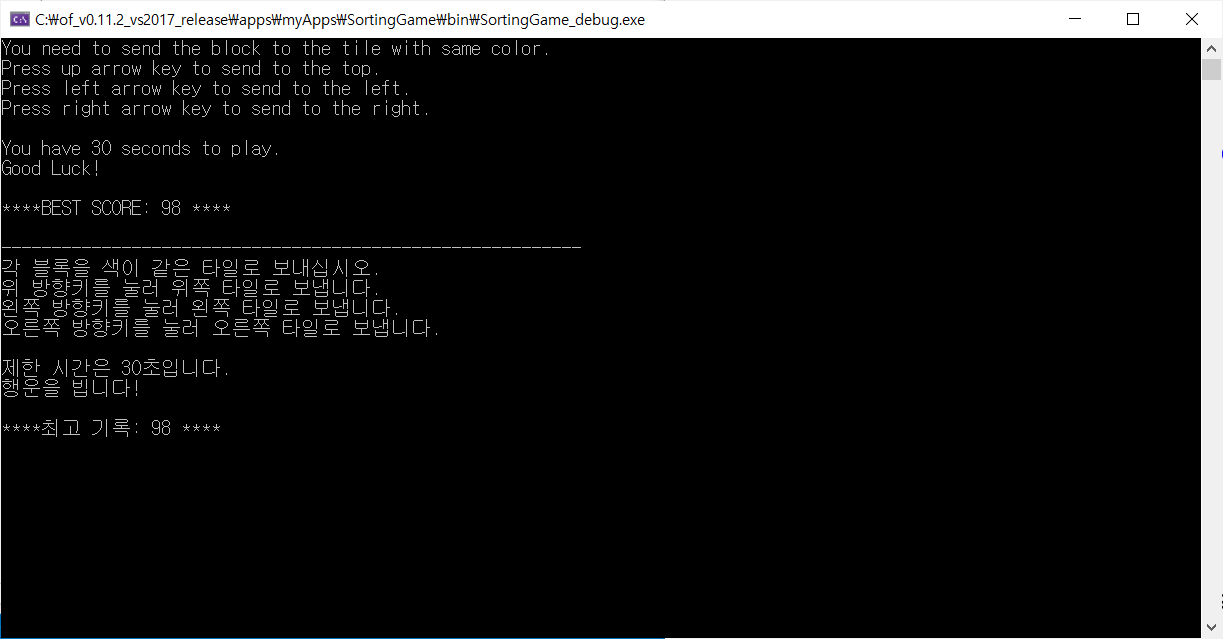
Void DrawBlock(): 정해진 블록의 색을 바탕으로 그 블록들을 화면에 그린다.

1. **본인이 어떻게 창의적 구현을 하였는지에 대한 설명**

게임 화면을 게임 시작 전, 게임 중, 게임 오버 상태로 총 3가지로 분류를 하였으며, 이는 Boolean 변수인 gamestart, gameover를 이용하였다. 그리고, 라이프와 시간을 가리키는 게이지가 각각 life, elapsed time에 따라 화면에 그려지는 바가 다르게 설정했다. 특히 시간을 이용할 때는 실습 시간에 다루지 않았던 ofGetElapsedTimef() 함수를 사용하였다. 게임 시작 시 reset된 elapsed time을 기준으로, 특정 시간들을 기준으로 그려지는 바가 다르게 하였으며, 매 초마다 그리는 글씨를 다르게 하여 카운트다운을 구현했다. 마지막으로, 타일과 블록에 모두 무작위성을 부여하여 매 게임마다 색이 달라지게 하였으며, 단순히 특정 방향에 특정 색을 match시키는 것이 아니라 특정 방향에 대한 색과 블록의 색을 match시키는 방법으로 매 게임마다 새로운 광경이 나오게끔 설정했다.

1. **프로젝트 실행 결과**

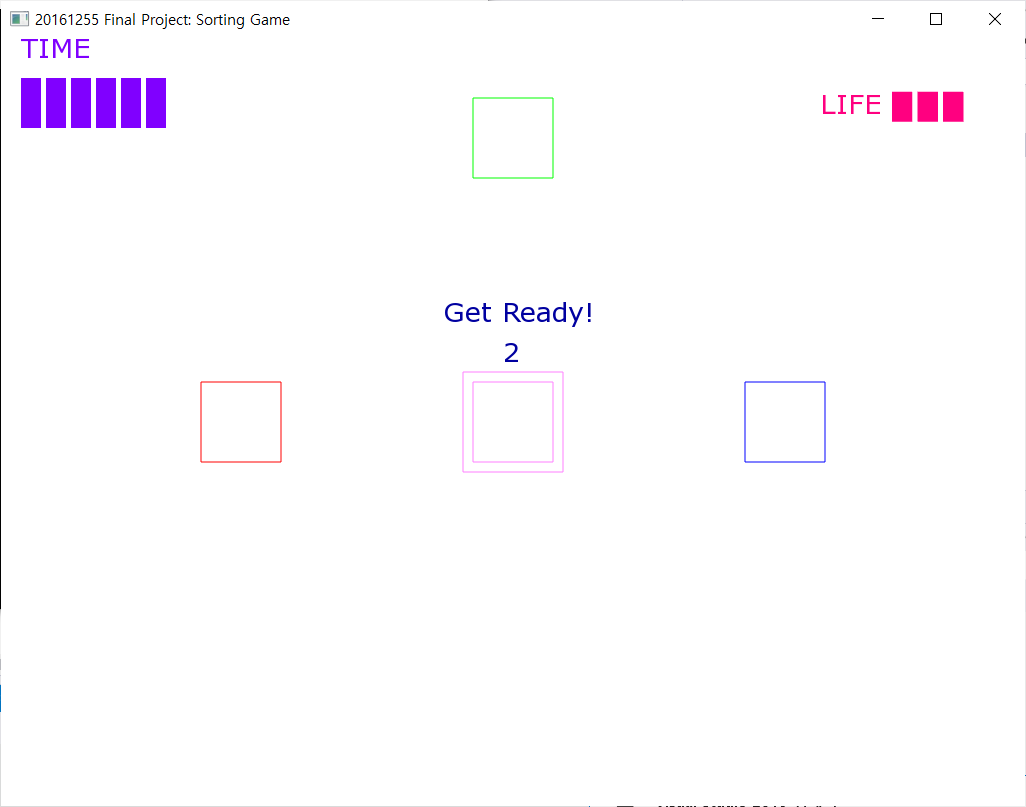
게임 시작 시 콘솔에 출력되는 바는 다음과 같다.



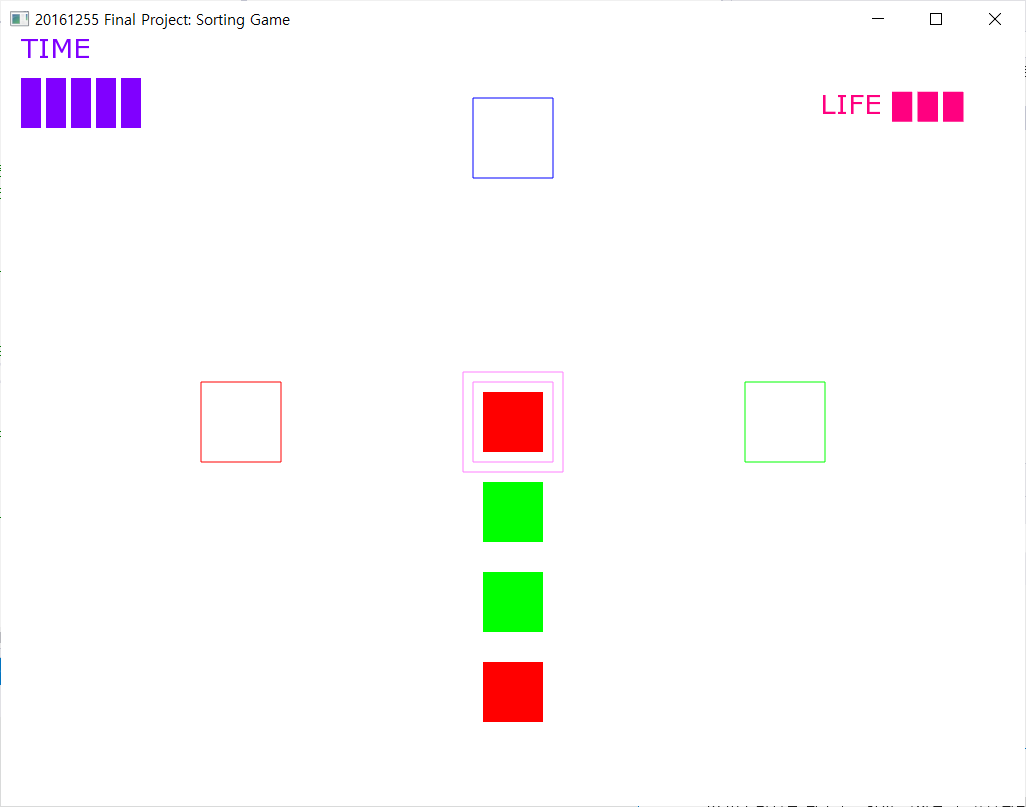
게임 화면에 출력되는 바는 다음과 같다.



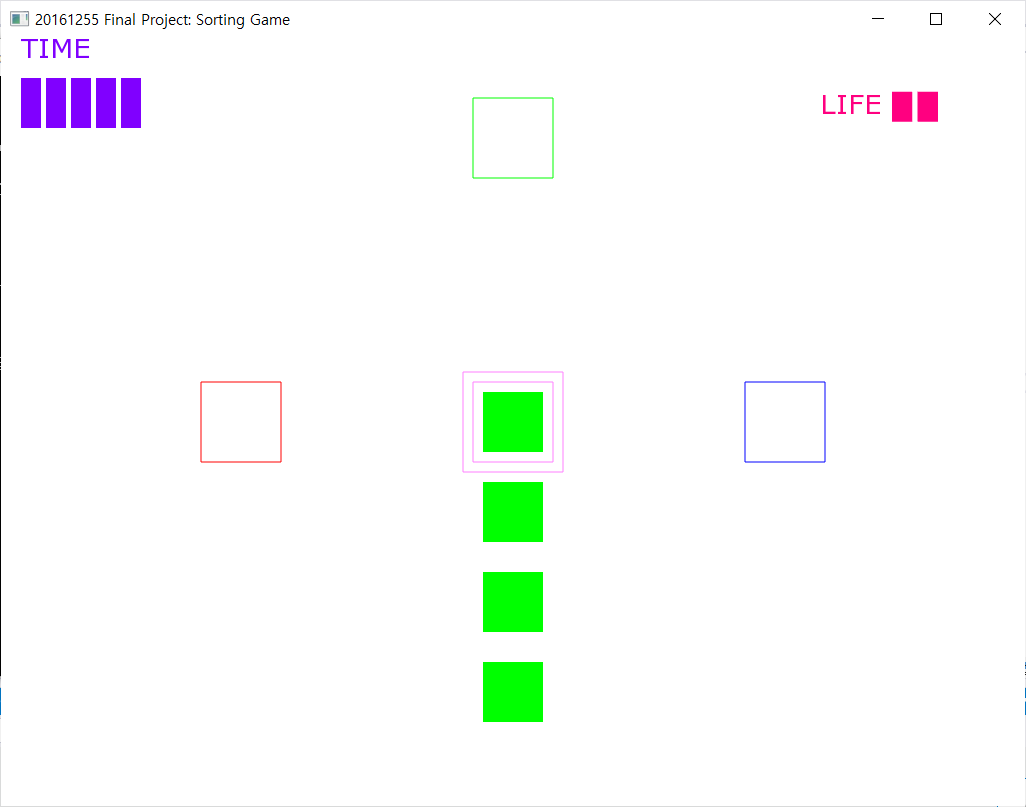
초기 카운트다운을 시작할 때의 화면은 다음과 같다.



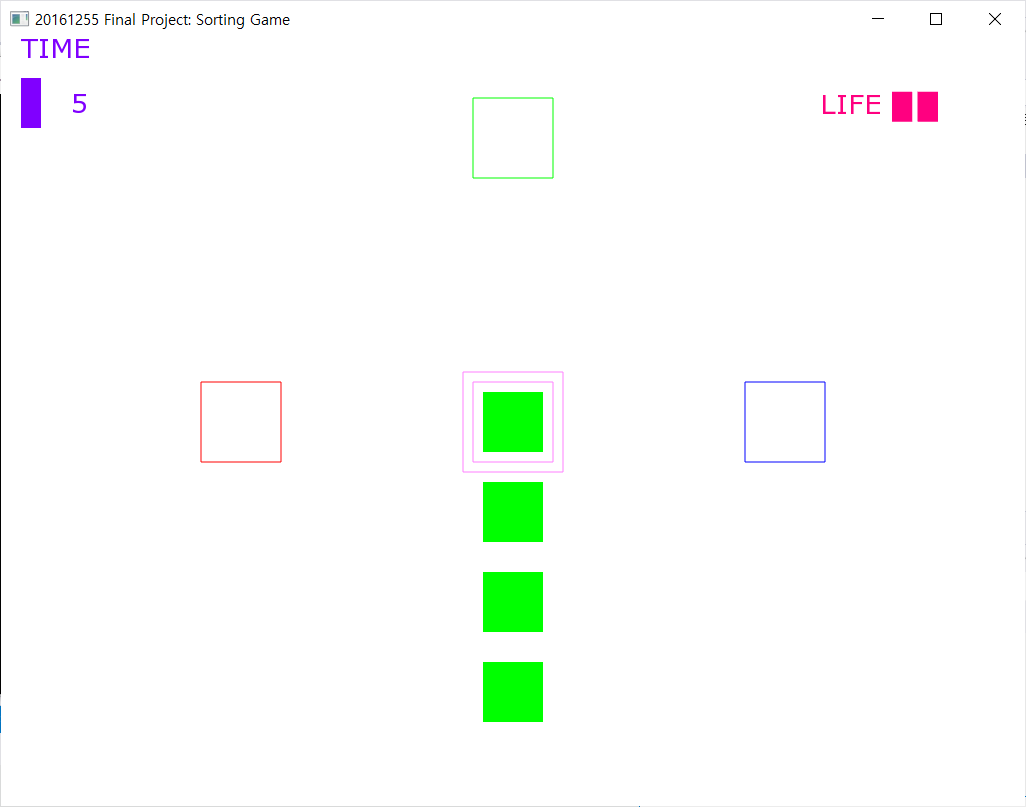
게임이 시작되고 5초 정도 지났을 때의 화면은 다음과 같다.



분류를 한 번 실수하여 라이프가 1 감소하였을 때의 화면은 다음과 같다.



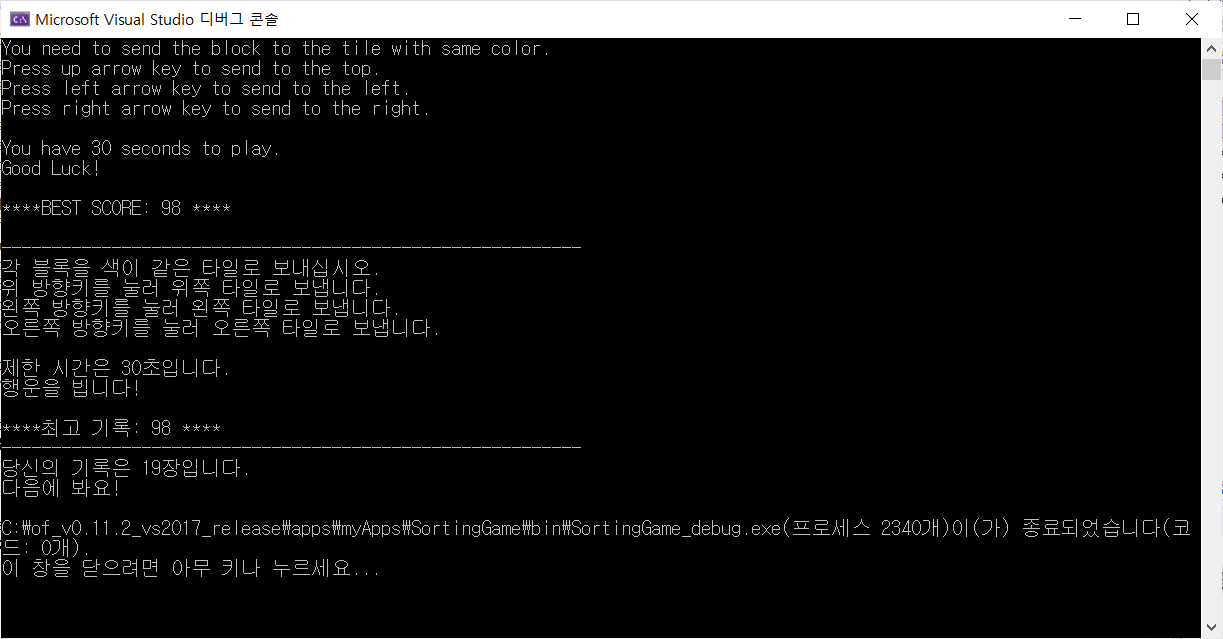
마지막 5초에 대한 카운트다운 화면은 다음과 같다.



게임 오버 화면은 다음과 같다.



‘x’ 키를 눌렀을 때 콘솔에 기록을 출력하고 프로그램을 종료하는 것은 다음과 같다.



1. **느낀 점 및 개선 사항**

오픈프레임웍스는 이번 컴퓨터공학설계및실습I 과목을 수강하면서 처음 사용해본 툴이었다. 이 툴을 이용하여 waterfall과 maze를 구현해보았지만, 이 툴을 이용하여 보다 더 즉각적으로 화면에 결과가 나타나는 간단한 게임을 만들고 싶었다. 이번 프로젝트를 통해 키보드로 입력한 바가 화면에 그대로 나타나는 것을 조금 더 실감나고 많이 확인할 수 있었다. 그리고, 최고 기록을 알 수 있게 만들면서, 기록을 세우겠다는 욕심을 낼 수 있고 적당히 긴장감도 있는 게임을 내가 스스로 만들었다는 점에 감탄했다. 다만 기존에 짰던 C코드와는 같은 듯 다른 느낌의 툴을 이용하여, 이 프로젝트를 계획했을 때 예상한 바보다는 구현이 조금 어려웠다. 그러나 기능을 하나하나 구현하다 보니 오픈프레임웍스 툴과 C++ 언어 자체와, setup, draw, update, keyPressed 함수 이용에 더 익숙해진 느낌을 받았다. 다만, 게임의 시각적 효과가 비교적 밋밋한 편에 속해, 모양을 다양화하거나, 특정 옵션을 추가하거나, 블록 분류에 성공 및 실패했을 때마다 나오는 극적인 효과를 부여하는 것을 생각하며 개선할 필요가 있어 보인다.