**CSE2003-1 기초공학설계**

(프로젝트: Flappy Bird Game)

담당교수: 서강대학교 컴퓨터공학과 장형수

1. **설계 문제 및 목표**

이번 프로젝트는 학생들이 Flappy Bird Game을 구현하는 것이다.

이 프로그램의 제작 과정은 기초공학설계 수업의 기본 과정을 따라야 하고, 공지한 보고서 형식에 맞추어 제출해야 한다.

1. **요구사항**

2.1 합성

**구현해야 할 사항**

1) 화면

- 화면은 score를 표시하는 부분과 게임이 진행되는 부분으로 나뉜다.

- score 표시 부분에서는 새가 장애물을 부딪치지 않고 통과하게 되면 점수가 1점씩 올라간다. 이때, 장애물 통과의 기준은 장애물 기둥보다 새가 오른쪽에 있는가이다.

- 게임이 진행되는 부분은 가로 40칸, 세로 25칸로 구성되어 있다. 게임은 새를 기준으로 진행된다. 새의 가로 위치는 고정되어 있고, (즉, 위아래로만 움직일 수 있다) 장애물이 새를 향해 다가온다.

- 이 게임의 시간단위는 0.2 초이다. 0.2초를 1시간단위라고 하자.

2) 장애물

- 장애물은 green 색이고, 가로 10칸, 세로 25칸의 크기를 유지한다.

- 장애물 기둥 사이에는 10칸만큼의 새가 통과할 수 있는 공간이 주어진다(앞으로 이 공간을 장애물 속 빈 공간으로 표현하자). 그 공간은 랜덤하게 만들어져 게임 플레이어가 매번 다른, 다양한 높이의 공간을 통과하게 해야 한다.

3) 새

- 새는 cyan 색이고, 새의 초기 위치는 전체 화면의 가로/4, 세로/2 위치이다.

- 앞에서 말했듯 새의 가로 위치는 변하지 않고, 세로 위치만 변화한다. 새의 세로 위치는 다음 세 가지 변화만을 갖는다.

(1) 게임 플레이어가 점프 키(‘w’, ‘a’, ‘s’, ‘d’) 중 하나를 누른 후 1 시간단위 경과 : 새는 위로 2칸 올라간다.

(2) 게임 플레이어가 점프 키를 누른 2시간단위 경과 : 새는 위로 1칸 올라간다.

(3) 이외의 경우 : 새는 1 시간단위 당 아래로 한 칸 내려간다.

위의 세 가지 경우는 번호 순으로 우선순위를 갖는다. 즉, 점프키를 누른 후 2시간단위 내에 다시 점프키를 누른다고 생각해보자. 1번 항목이 우선순위가 높으므로 2칸 점프 후 1칸 점프가 아닌 2칸 점프 후 다시 2칸 점프, 그 후 1칸 점프하게 된다.

- 새가 장애물에 부딪히면, 즉 새와 장애물이 동일한 위치에 있게 되면 게임이 끝나고, “Game over!” 문구가 출력된다.

**제작에 필요한 문법적 지식**

본 게임 코드 속 user code 칸을 채우기 위해서는 printf() 함수, if - else 구문, for 구문, 전역변수와 지역변수, rand() 함수와 srand() 함수, text 색 지정 등에 대한 지식이 필요하다. 이는 모두 정규 강의 시간에 학습한 내용이므로 간단하게 설명하겠다.

printf() 함수는 화면으로 주어진 문자 또는 숫자를 출력하는 역할을 하는 함수로, 큰 따옴표(“”) 안에 출력할 내용을 적는다. 변수를 출력하고자 한다면 큰 따옴표 안에 해당 변수의 형식문자(%d, %c 등)를 입력하고, 따옴표 뒤에 해당 변수의 이름을 적는다.

if-else 문은 조건문 중 하나로, 괄호 안에 들어가는 조건식의 값이 참일 때와 거짓일 때 다른 구문을 실행한다. If-else문은 반복문이 아니므로 한 번의 실행으로 끝난다.

for 문은 반복문 중 하나로, 먼저 초기식을 실행한 후 조건식이 참일 때 계속해서 for문 속 구문을 실행한다. 이때, 한 번 실행될 때마다 변화식을 함께 실행한다.

전역 변수는 코드 전체에서 사용되는 변수이고, 지역변수는 함수 내에서만 사용되는 변수이다. 따라서 여러 함수에서 사용되는 변수라면 함수가 입력 값을 갖게 하는 방법 외에 전역변수를 이용하는 방법도 있다.

rand() 함수는 무작위한 숫자를 불러오는 역할이고, srand() 함수는 rand() 함수가 불러오는 무작위 값의 최대값을 결정하는 함수이다. 이 함수를 사용하기 위해서는 stdlib.h 헤더파일을 사용해야 한다. srand() 함수의 가장 대표적인 사용 형태는 srand(time(NULL)) 인데, 시간을 이용해 무작위 값을 얻는 것이다. time() 함수는 시간을 불러오는 함수이고, 이를 사용하는 데에는 time.h의 헤더파일이 필요하다.

printf() 함수를 이용할 때 출력되는 내용의 색을 변경할 수 있는데, 다음과 같은 방법을 이용하면 된다.

printf(“\033[색코드mtext\033[0m”)

여기서 색코드 부분에는 원하는 색의 코드를 입력하고, text라고 적힌 부분에는 그 색으로 출력하고자 하는 내용을 입력하면 된다.

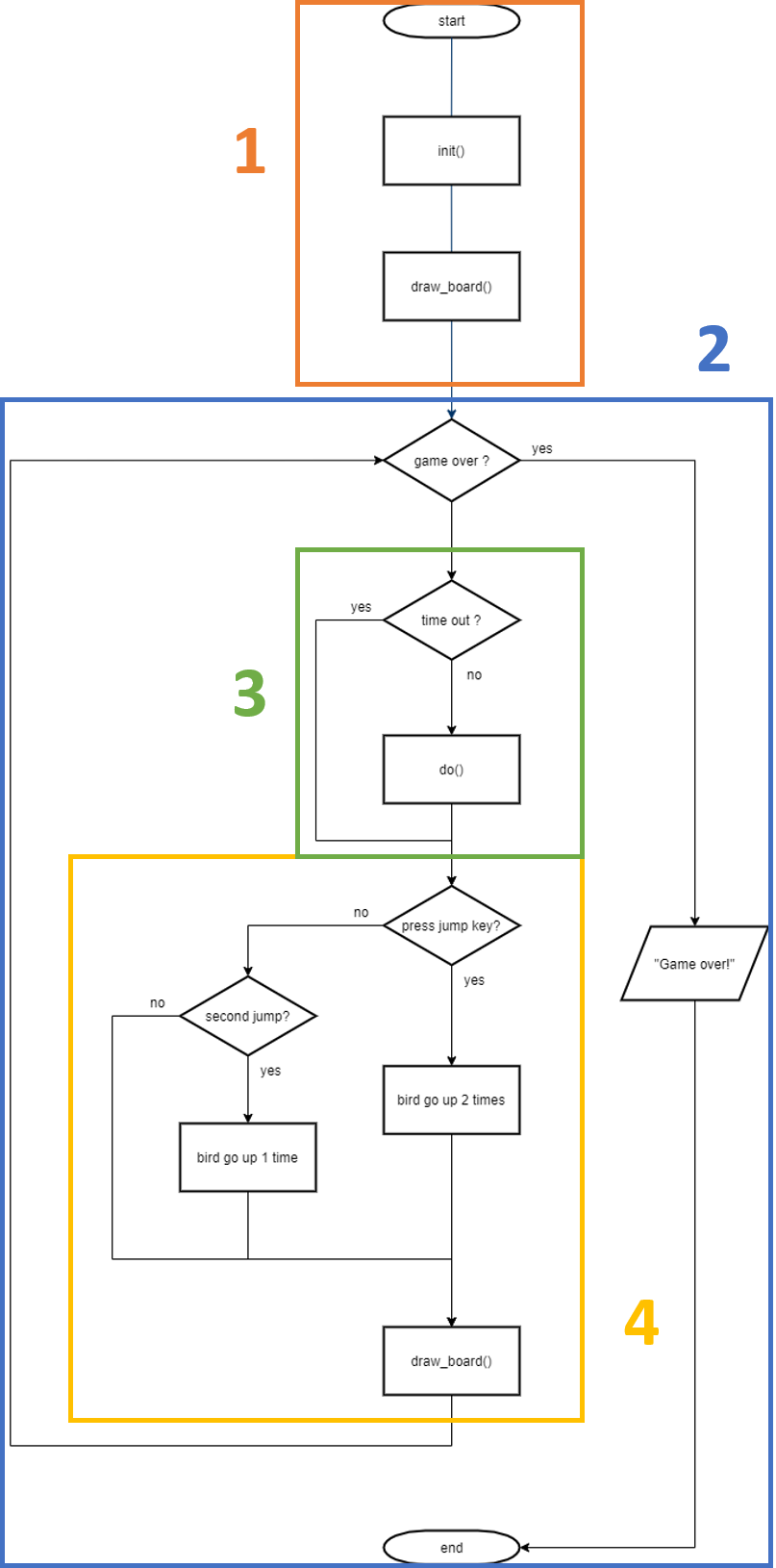
**환경과 기본 요구사항. 그리고 이것이 프로그램의 제작에 미치는 영향**

본 게임은 linux 환경에서 제작된다. (자세한 설명은 3.1 환경 구성 참고) linux는 unix를 기반으로 만들어진 운영체제인데, unix처럼 다중 사용자, 멀티태스킹을 지원하는 네트워크 운영체제이다. Linux의 원형인 unix가 통신 네트워크를 지향해서 만들어졌기에, linux 또한 서버에 최적화 되어있다. 입출력이나 네트워크 처리 등이 운영체제에서 일어나는 모든 일이 단일 커널을 통해 처리되기 때문에 빠른 속도로 구현되는 것이 특징이다. 따라서 화면을 계속해서 업데이트해야 하는 게임의 경우, 이런 점에 있어 유리할 수 있다. 또한 내용이 어려워 자세하게는 모르겠으나, 본 게임 제작에 사용한 termios.h 헤더파일을 사용하는데 이는 linux(unix)만의 독특한, 윈도우 등에서는 사용할 수 없는 헤더파일인 것으로 보인다.

2.2 분석

**순서도 작성**

아래의 그림은 main 함수 속의 코드를 바탕으로 만든 개관적인 flow chart 전체이다.



이제 flow chart 각 부분을 자세하게 살펴보자.

먼저 1구역을 부분을 살펴보자. 1구역에서는 게임이 시작하자 마자 init() 함수와 draw\_board() 함수를 실행함을 보여준다. init() 함수는 새의 초기 위치를 설정하는 함수이고, draw\_board() 함수는 화면을 그려주는 함수이기 때문에 두 함수를 실행하면 초기 화면을 구성하게 되는 것이다.

초기 화면이 구성되고 나면 본격적으로 게임을 진행하게 된다. 2구역을 간단하게 살펴보자면, game over이 되기 전까지 게임이 실행되고, game over이 되면 “Game over!”를 출력하고 게임을 끝내게 된다. 게임 상에서는 game over이라는 변수를 통해 게임이 끝났는지 판단하게 된다.

3구역을 살펴보면, 시간이 초과되면, 즉 time out이 되면 do() 함수가 실행되지 않고 다음으로 넘어간다. 즉, do() 함수가 1 시간단위 내에 실행되어야 하는 것이다. do() 함수는 새의 위치와 장애물의 위치를 게임의 흐름에 맞게 설정하는 함수이다.

3구역이 실행되고 나면, 4구역으로 넘어가는데, 4구역에서는 점프 키를 눌렀을 때의 순서도를 나타낸 것이다. 점프키를 눌렀을 때, 1시간단위에는 2칸을 점프하고, 2시간단위에는 한 칸을 추가로 올라가야 한다. 따라서 이를 확인하기 위해 점프키를 눌렀는지, 점프키를 누른 뒤 1시간단위가 지난 후인지, 점프키를 누르지 않았는지를 구분하여 각각에 알맞은 코드가 구현되어야 한다.

점프까지 모두 수행하고 나면 draw\_board() 함수를 통해 새나 장애물의 최종 위치를 모두 업데이트한다. 여기까지 2구역의 모든 함수를 다 거치게 되면 다시 앞으로 돌아가 game over이 되었는지를 1시간단위마다 계속해서 확인한다. 도중에 새와 장애물의 위치가 겹치거나(즉, 새가 장애물에 부딪히거나), 새가 화면 밖으로 나간다면game over이 된다.

**임의의 장애물 생성 방법**

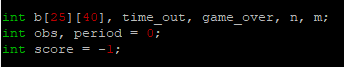
임의의 장애물의 경우 세로 25칸 중 위쪽 3칸, 아래쪽 3칸은 장애물로 고정하고, 남은 19칸 내에서 빈 공간을 만들었다. 빈 공간의 경우 10칸이 모두 연결되어 있으므로 제일 윗 칸만 난수로 받아 받은 난수의 아래 10칸을 빈 공간으로 만드는 방식이다. 따라서 19칸 중 10칸은 비어 있어야 하므로, 빈 공간 중 제일 위 칸을 결정하기 위해서 난수는 0부터 8까지의 수 내에서 결정되어야 한다.

모든 블록을 기준으로 하지 않고 위 블록 일부와 아래 블록 일부를 장애물로 고정한 이유는 그것이 게임의 원 설정과 더 잘 부합한다고 생각했기 때문이다. 이 게임의 모토였던 flappy bird에서의 장애물은 위아래에서 나오는 굴뚝이다. 화면 위쪽에서 뻗어 나오는 굴뚝과 아래에서 뻗어 나오는 굴뚝 사이의 공간으로 새가 날아가야 하는 것이다. 그렇기에 위쪽과 아래쪽 모두 굴뚝이 낮게나마 있어야한다고 생각했고 이를 임의의 장애물 구현에 반영했다.

2.3 제작

각 함수가 어떤 기능을 수행하는지 설명하고, 직접 코드를 작성한 부분은 어떻게 그런 기능을 수행하게 되었는지 작성한 코드와 함께 설명하겠다. 설명을 이해하기 쉽도록 코드를 구성한 순서와 최대한 맞추어 설명할 것이다.

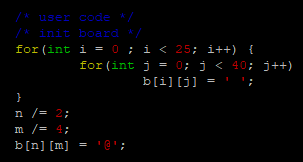
**1) 전역 변수**

게임의 구현을 위해 선언된 전역변수이다. 여기서 배열 b는 게임 속에서 등장할 화면을 나타내는 것이고, time\_out은 시간 초과를 확인하는 변수, game\_over은 게임이 끝났는지를 확인하는 변수이다. (시간이 초과되면 time\_out 변수의 값이 1이 되고, 게임이 끝난다면 game\_over 변수의 값이 1이 된다.) 그리고 n과 m은 새의 세로위치와 가로위치이다.

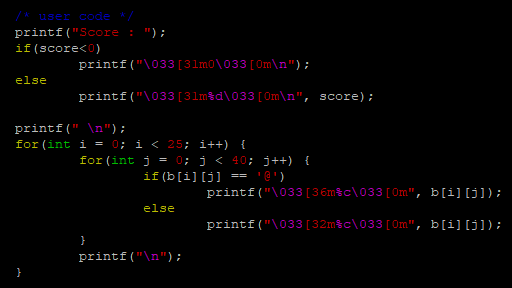
이 외에도 추가적으로 선언한 전역변수로는 obs, period, score이 있다. obs는 장애물 속 빈 공간의 위치, period는 장애물이 표시되는 주기를 나타내는 변수이고, score은 점수를 나타내는 변수이다.

각 변수에 대한 자세한 내용은 변수가 주로 사용되는 함수에서 설명할 것이다.

**2) init()**

새의 초기 위치를 설정해준다. 이때 새의 위치는 계속해서 변화해야 하고, 함수들 사이에서도 계속 전달되어야 하기 때문에 전역변수인 n, m에 각각 새의 세로위치와 가로위치 값을 갖도록 설정했다. 이후의 모든 함수에서도 n과 m을 변화시켜 새의 위치를 바꾼다. init() 함수의 앞부분에서 n과 m을 각각 세로길이와 가로길이로 설정했고, 새는 초기에 가로/4, 세로/2의 위치에 있어야 한다. 따라서 새의 초기위치를 사진 속 코드처럼 변경해준 후 변경된 자리에 새, 즉 ‘@’를 대입했다.

새의 위치 설정 전 등장하는 for 문은 새 이외에는 공백만 있어야하는 초기화면의 모습을 설정해주는 것이다. 이차원 배열 b 속에 모든 칸을 공백, 즉 ‘ ‘로 채워준 뒤 새의 위치에는 ‘@’를 대입한다.

**3) draw\_board()**

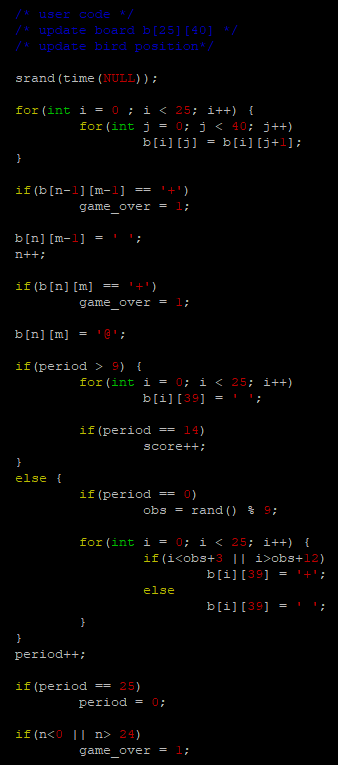
이 함수는 이름에 걸맞게 화면을 구성하는 함수이다. 화면은 점수 부분과 게임이 실행되는 부분으로 나눌 수 있는데, 먼저 점수는 score 변수를 이용한다. do() 함수에서 새가 장애물을 무사히 통과했는지를 파악하여 점수를 매겨 변수 score에 기록한다. 그리고 이 score 변수를 전역변수로 선언하였기에 이 변수를 draw\_board() 함수도 사용할 수 있다. score 변수를 출력하여 점수를 표시하였고 이 값을 예시처럼 빨간색으로 출력하게끔 했다. 여기서 한 가지 주목해야할 것은 score의 초기값이 0이 아니라 -1이라는 점이다. 이 부분에 대해서는 do() 함수에서 자세히 설명하도록 한다.

이 함수를 처음 실행했을 때는 문제가 있었다. 새의 위치를 문제 조건에 맞게 설정했지만 새가 원하는 위치에 있지 않았던 것이다. 그 원인을 살펴보니 score 부분을 프린트한 후 ‘\n’(줄바꿈)이 실행되어야 하지만 커서가 아랫줄 제일 왼쪽이 아닌 출력된 score 바로 아래로 이동하는 것이었다. 따라서 ‘ ‘(공백)을 출력한 후 ‘\n’을 출력하여 화면 제일 왼쪽 줄에 커서를 놓게 하였다.

다음으로 출력해야 할 곳은 게임이 실제 실행되는 부분이다. 게임에는 세 가지 문자가 등장하는데, 새를 나타내는 ‘@’, 장애물을 나타내는 ‘+’, 그리고 공백인 ‘ ‘이다. 그리고 문제 조건에서 새는 cyan색이어야 하고, 장애물은 green색이어야 한다고 명시했기에 출력되는 사물에 따라 색을 변형시켜야 한다. 공백은 색을 입힌 것과 입히지 않은 것이 다르지 않다는 점에서 착안하여 새인 경우 cyan색을, 새가 아닌 경우(즉, 장애물이거나 공백인 경우) green 색을 입히도록 코드를 구성했다.

**4) do()**

do() 함수는 각 시간단위당 계속해서 바뀌는 새와 장애물의 위치를 새롭게 설정해주는 역할을 수행한다. 그렇기에 이 게임 중 가장 중요한 함수라고도 할 수 있겠다.

장애물 속 빈 공간이 무작위한 위치에 출현해야 하기 때문에 rand() 함수와 srand() 함수를 사용해야 한다. 자세한 설명은 앞에서 명시했으니 생략한다.

게임이 진행되려면 새를 제외한 게임 화면이 1 시간단위마다 모두 한 칸씩 왼쪽으로 움직여야 한다. 따라서 이를 구현한 코드가 가장 위쪽에 있는 for 문이다. 이때, 새의 위치 또한 함께 옮겨지기 때문에 옮겨지는 과정에서 장애물과 부딪쳤는지 확인하는 코드가 바로 아래에 있다. 이렇게 왼 쪽으로 한 칸씩 옮기게 되면 배열 b의 제일 마지막 열만 비어있게 된다.

이후 새는 1 시간단위당 한 칸씩 떨어지기 때문에 새의 세로 위치를 이동해준다. 그리고 내려간 위치에 장애물이 있다면 이는 또한 부딪친 것이 되므로 장애물이 있는지 확인해준다. 그리고 이동된 위치에 새를 놓는다.

다음으로 보이는 if 문은 앞에서 언급한, 배열 b의 비어있는 마지막 열을 채우는 역할을 한다. 장애물의 가로 크기가 10칸, 이후 나오는 공백의 가로 크기가 15칸이라는 것에서 아이디어를 얻었다. 이 25칸이 지나갈 동안의 시간을 한 주기로 보고, 주기 내 시간대별로 배열 b의 마지막 열에 알맞은 것을 대입한다. 그리고 이 주기를 판단하는 변수가 period이다. 이 변수는 전역변수인데, 함수가 실행될 때마다 누적되어야 하는 변수이므로 함수가 여러 번 실행되는 과정에서도 변수 값이 초기화되면 안되기 때문이다. period가 0부터 9까지는 장애물이 있는 구간이고, 10부터 24까지는 장애물이 없는 구간이다. period 값이 무엇이냐에 따라 다른 것을 출력한다.

(1) period == 0 : 새로운 장애물의 시작이다. rand() 함수를 이용해 장애물 속 빈 공간이 시작되는 위치를 무작위로 얻는다. rand() 함수에서 만들어내는 무작위 값을 9로 나누어 0부터 8까지의 무작위 값을 얻었다. 이 값을 obs라는 변수에 대입했는데, obs는 빈 공간의 첫 시작을 알려주는 변수이다. (이 값은 함수가 실행되는 내내 바뀌어서는 안되기 때문에 period와 마찬가지로 전역변수로 선언했다.) 이 obs 값은 period가 0일 때, 즉 새 주기가 시작될 때만 만들어져야 하는데, 그 이유는 하나의 장애물이 만들어 질 때는 빈 공간의 위치가 일정해야 하기 때문이다.

(2) 0 <= period < 9(이 구간은 0을 포함한다) : 배열 b의 마지막 열에 장애물이 만들어져야 하는 구간이다. period가 0일 때 얻었던 obs 값을 이용해서 ‘+’가 출력되어야 하는 부분과 공백이 출력되어야 하는 부분을 구별한다. 제일 위의 3칸은 고정했기 때문에 obs+3부터 obs + 12까지 공백이 출력되고, 그 이외의 부분은 ‘+’, 즉 장애물이 출력되게 한다.

(3) 10 <= period < 25 : 장애물이 만들어지면 안되는, 공백이 출력되어야 하는 구간이다. 따라서 period가 이 구간에 위치하면 배열 b의 마지막 열은 공백을 입력한다.

(4) period == 14 : 이때는 새가 하나의 장애물을 넘어간 시간이다. 따라서 score 값을 1 올려준다. (이 시간은 경험적으로 구한 시간이다; 여러 번 게임을 실행해보고 넘어간 직후 점수가 표시되는 period 값이 14였다) 여기서 score의 초기값이 -1이어야 하는 이유가 나오는데, 게임이 실행된 후 처음 period가 14가 되었을 때는 아직 장애물이 새한테까지 도달하지 못하였기 때문이다. 따라서 score의 초기값이 -1이면 새가 처음 장애물을 통과할 때의 score 값이 1이 된다. 하지만 게임 플레이어는 점수가 0부터 시작하는 것처럼 보여야 하기 때문에 draw\_board() 함수에서는 score이 -1이라도 0을 보이게 한다.

(5) period == 25 : 모든 주기를 수행하였다. 변수 period가 0부터 시작하였음에도 주기가 끝났을 시간이 24가 아닌 25인 이유는 출력이 끝난 후 period가 증감연산자를 통해 1이 더해지기 때문이다. period가 25가 되면 period를 다시 0으로 초기화시킨다.

주어진 조건 중에서 새가 화면을 벗어나면 게임이 끝나야 한다는 내용이 있다. 따라서 새의 새로 위치가 0보다 작아지거나 24보다 커진다면game\_over 변수에 1을 부여하여 게임이 종료되게 한다.

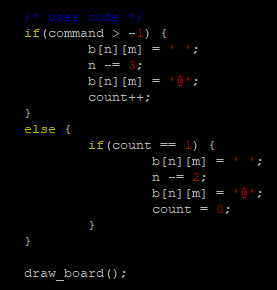
**5) getch()**

사용자가 입력한 키를 반환하는 역할을 수행한다. 이 함수는 아래에 소개될 GetCommand() 함수와 함께 사용되는 함수이다.

**6) GetCommand()**

getch() 함수가 입력한 키를 반환한다면, 이 함수는 getch() 함수를 호출하여 키를 정수로 변환하여 반환한다. 점프 키인 ‘w’(또는 ‘W’), ‘a’(또는 ‘A’), ‘s’(또는 ‘S’), ‘d’(또는 ‘D’)를 입력하면 음이 아닌 정수 값을(각각 0, 1, 2, 3), 그 외의 다른 값을 입력하거나 키가 입력되지 않는다면 -1을 반환한다.

**7) main()**

main() 함수가 가장 중심인 함수인만큼 전체적인 게임 진행을 담당한다. 왼쪽의 코드에는 나와있지 않지만, do() 함수를 반복해서 호출한다. 여기서 살펴보아야 할 것은 키 값을 반환하는 함수인 GetCommand() 함수가 값을 main() 함수에만 반환하고 있다는 점이다. 그렇기에 다른 함수에서는 점프를 구현할 수 없고, main() 함수에서 이를 구현해야 한다는 것을 알 수 있다.

GetCommand() 함수에서 값을 command 라는 변수로 반환하고 있기 때문에 command 값을 통해 점프 키가 입력되었는지 확인할 수 있다. command 값이 -1보다 크다면, 즉 점프키가 입력되었다면 새는 세 칸 점프해야 한다. 두 칸 점프하는 것이 조건이기 때문에 do() 함수에서 새가 한 칸 내려갈 것을 고려한다면 세 칸을 올려야 최종적으로 두 칸 올라간 것이 된다. 그리고 count라는 변수가 있는데, 이 변수는 새가 2칸 점프해야 하는지(점프키를 누른 직후인지), 새가 1 칸 점프해야 하는지(점프키를 누른 뒤 1 시간단위가 지난 후인지)를 확인한다. 새가 두 칸 점프한 후 count 변수를 1 증가시킨다. 이후 1 시간단위가 흘러 다시 위의 if 구문에 도착한다면 else 안의 if문을 통해 count가 1임을 확인하고, 1칸 점프할 수 있는 것이다.

그리고 새나 장애물 등 모든 것의 위치가 결정되고 나면, draw\_board() 함수를 호출하여 배열 b 속에 저장된 위치정보를 게임 화면에 출력한다.

2.4 시험

줄 맞추기는 성공적이었다고 판단된다. 만약 공백을 나열하여 줄을 맞추었다면, 실행하는 화면의 크기가 변할 때마다 첫 줄의 화면이 깨질 것이다(제대로 출력되지 않을 것이다). 하지만 줄바꿈 처리를 진행한 덕분에 게임 화면보다 큰 화면에서 실행되기만 한다면 문제없이 구동된다.

새가 장애물에 부딪혔을 때 또한 문제없이 게임이 종료된다. 새가 위쪽 장애물에 부딪혔을 때, 아래쪽 장애물에 부딪혔을 때, 오른쪽에서 장애물과 부딪혔을 때 모두 부딪히자마자 게임이 종료되게끔 설계되었고 설계된 대로 구동되었다. 또한 새가 화면을 벗어날 때도 벗어나자마자 게임이 종료되는 것을 확인할 수 있다.

장애물도 임의대로 잘 생성되는 것으로 보인다. 여러 번 게임을 플레이해본 결과 규칙적으로 장애물이 생성되는 것 같다는 느낌은 받을 수 없었다.

전체적으로, 주어진 게임 플레이 동영상 예시와 거의 동일하게 진행되는 모습을 보아 바르게 설계되었고 또 바르게 구동되는 것으로 보인다.

2.5 평가

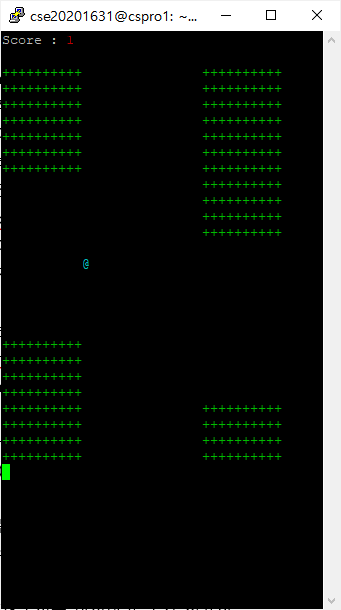
프로그램을 여러 번 플레이해보았다. 또한, 내가 발견하지 못한 오류에 대한 정보를 수집하기 위해 가족들에게 게임 플레이를 부탁해서 의견을 받았다. 평가한 기준은 다음과 같다.

- 새가 키를 누르는 대로 잘 움직이는가?

- 장애물이 무작위 하게 생성되는가?

- 장애물을 통과함에 따라 점수가 맞게 올라가는가?

- 새의 점프는 올바르게 구현되었는가?

위 평가 기준에 모두 대체적으로 좋은 답변을 얻었다. 새가 점프할 경우 두 칸 점프한 후 한 칸을 점프하는 모습을 볼 수 있었고, 장애물 속 빈 공간 또한 여러 번 실행할 때마다 다른 위치에서 생성되었다. 장애물을 통과할 때마다 점수가 올바르게 올라갔다. 점수가 올라가는 화면의 사진을 오른쪽에서 확인할 수 있다.

새는 대체적으로 키를 누르는대로 잘 움직이는 것으로 보인다. 다만, 계속해서 키 씹힘 현상이 일어나는 등의 오류가 발생하는 경우가 많이 보였다. 하지만 이는 많은 학우들이 제시하는 문제인 것으로 보아, 실행 환경의 문제인 것으로 보인다. 다행히도 이 문제가 게임 플레이가 불가능할 수준으로 무분별하게 발생하는 것이 아니라, 플레이어에게 약간의 짜증을 유발하는 정도의 수준으로만 발생하는 데 그치는 것으로 보인다.

1. **기타**

3.1 환경 구성

앞에서도 언급했듯이 본 게임은 linux OS 내에서 개발되고 실행된다. Linux에 대해 보충하여 설명하자면, 리누스 토르발스가 개발한, Unix 커널로부터 파생된linux 커널을 사용하는 운영체제이다. Linux는 오픈소스 소프트웨어이기 때문에 누구든 이 소프트웨어를 사용할 수 있고, 다양한 배포판이 존재한다. PC 외에도 다양한 플랫폼에서 설치가 가능하다.

이 ‘Linux’라는 용어는 linux 커널만을 뜻하지만, 리눅스 커널과 GNU(운영체제 중 하나) 프로젝트의 라이브러리와 도구들이 포함된, 전체 운영 체제를 나타내는 말로 흔히 쓰인다. (본 보고서에서도 이와 같은 의미로 많이 사용되었다.) 본 수업에서는 우분투OS를 사용하고 있는데, 다른 배포판에 비해 설치나 이용이 쉽다는 점에서 초보자를 위한 linux로 알려져 있다.

Linux 환경에서 사용되는 명령어 중 컴파일과 실행에 관련된 명령어들을 간단히 소개하자면, vi 명령어와 gcc 명령어가 있다. vi 명령어는 ‘vi [file]’ 의 형태로 사용되는데, file을 열어주는 역할을 한다. 여기서 file이 c파일이라면, c 언어 코드를 작성할 수 있게 되는 것이다. vi 명령어로 작성한 코드는 gcc 명령어로 컴파일한다. gcc 명령어는 ‘gcc [file]’ 의 형식으로 사용할 수 있고, file을 컴파일하고 실행파일을 생성하는 역할을 수행한다. 그리고 이 실행파일은 (특별한 옵션이 없다면) a.out의 이름을 갖고 있는데, 실행파일은 ‘./[file]’(실행파일의 이름이 a.out의 경우 ./a.out 입력)을 입력하여 실행할 수 있다. 여기서 ‘/’는 경로를 보여주는 문자이고, ‘.’은 자신임을 나타내는 문자라고 한다. 즉, 어떠한 디렉토리 내에도 속하지 않는다는 뜻이라고 추정된다.

따라서 본 게임을 실행하기 위해서는 user\_flappy\_bird.c라는 본 게임의 코드가 담긴 c파일을 가지고, linux 서버에 이 파일을 옮겨야 한다. 그 후, gcc 명령어를 이용해 실행파일을 만들고 파일을 실행해야 한다.

3.2 팀 구성

개인이 1팀을 구성한다.

3.3 수행기간

2020/06/19 (금요일) 자정까지