최종프로젝트 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20231604 이름: 정유연

1. 프로젝트 목표

크롬에서 인터넷 연결이 안됐을 때 기본으로 열리는 **공룡 게임**을 구현한다.

윈도우os 아래 오픈 프레임 워크에서 기본 제공되는 함수들을 이용했다.

ofApp.h, ofApp.cpp : 공룡과 장애물 객체가 이용된 게임 플레이 기능이 구현되어 있다.

Dinosaur.h, Dinosaur.cpp : 공룡의 모양, 위치, 점프 정보를 다룬다.

Obstacle.h, Obstacle.cpp : 장애물의 모양, 위치, 그림 수정, 장애물이 화면 밖을 나갔는지 등을 다룬다.

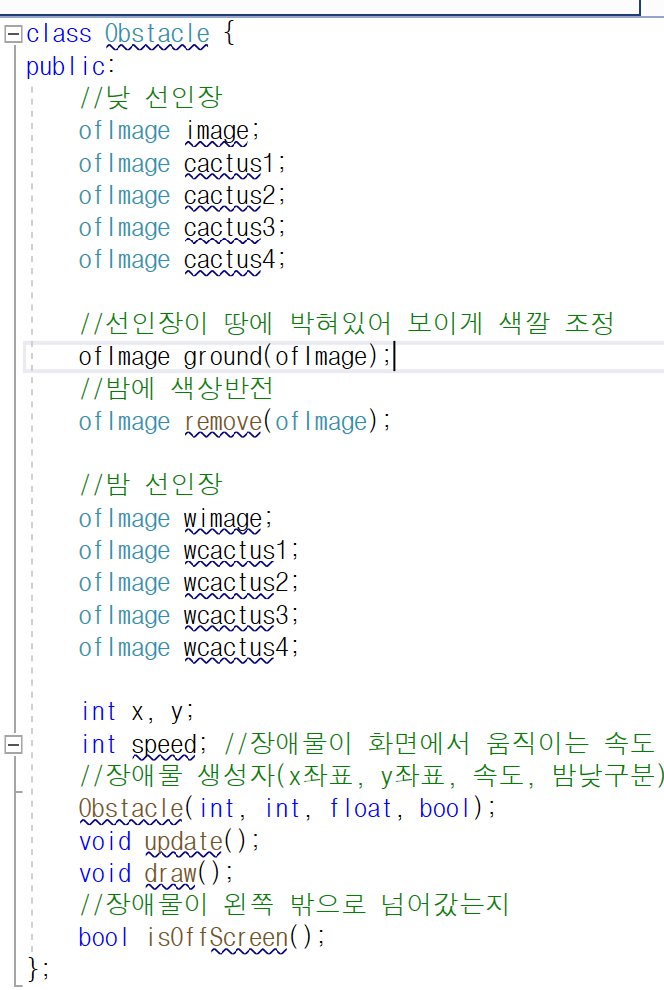
1. 각 변수에 대한 설명
2. Dinosaur

leftdino, rightdino가 번갈아 화면에 나오며 공룡의 발이 바뀌는 것처럼 보인다.

footSwitch가 lastSwitchTime을 바탕으로 0.1초에 한번씩 갱신되며 좌우 공룡을 번갈아 띄운다.

isJump, jumDuration, jumHeight, jumpStartTime에서 공룡이 점프 환경을 설정한다.

1. Obstacle



speed에 따라 선인장의 x위치가 바뀌며

왼쪽으로 움직이는 것 같은 효과를 준다.

isOffScreen검사로 선인장이 바깥으로 나갔는지 검사한다.

Dinosaur에서와 같이 밤에 나타나는 흰색 이미지를 remove로 수정한다.

ground에서 선인장이 땅에 박혀 있는 듯한 효과를 주기위해 선인장의 바닥부분 주변을 땅색으로 바꾼다.

1. ofApp

dino 공룡 객체를 선언한다.

obstacles 장애물 벡터를 선언한다.

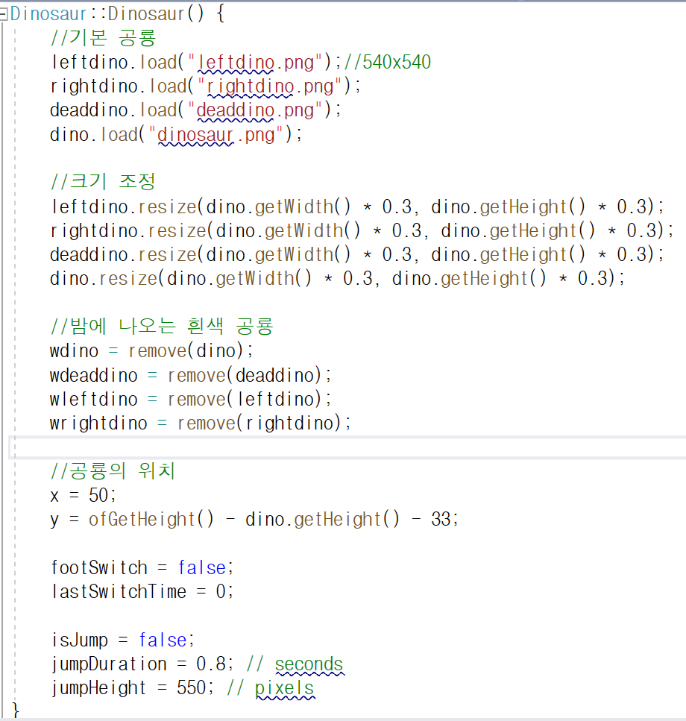
score와 speed에 현재 점수와 속도를 저장한다.

bw에 현재 밤인지 낮인지 저장한다.

update에서 현재 공룡, 장애물, 점수, 속도 정보를 갱신하고 밤낮을 구분한다.

draw에서 배경, 땅, 공룡, 장애물을 연속해서 그린다. 현재 점수, 속도, 최고점수를 화면 오른쪽 위에 띄운다.

1. 각 함수에 대한 설명
2. Dinosaur
   1. 생성자

4가지 공룡의 이미지를 업로드하여 크기를 화면에 맞게 조정한다.

밤에는 기존의 검은색상을 반전시킨 4가지의 흰색 공룡 이미지를 따로 로드한다.

공룡의 기본 위치를 지정한다.

* 1. update()

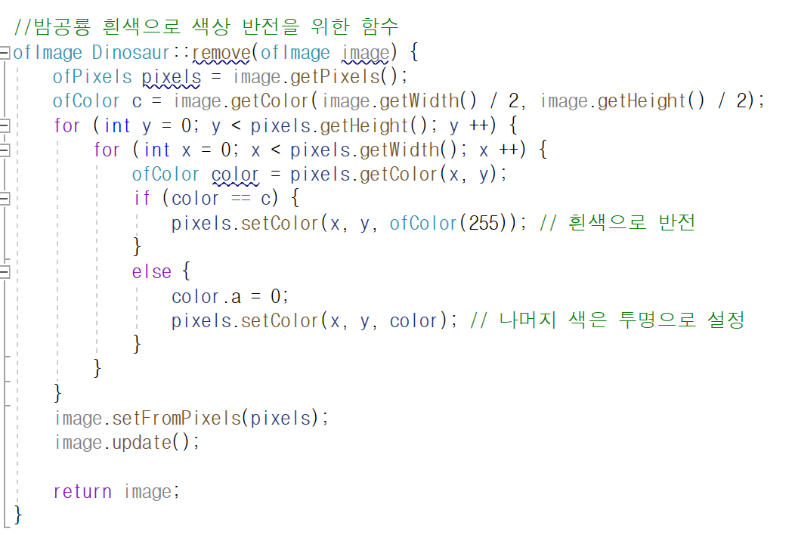
0.1초에 한번씩 footSwitch와 lastSwitchTime을 갱신한다.

draw함수에서 footSwitch 변수가 이용된다.

isJump에 지금 점프 중인지를 저장한다.

생성자에서 정한 jumpDuration과 jumpHeight를 바탕으로 sin함수를 계산해 곡선으로 점프하는 공룡의 y좌표를 업데이트한다.

* 1. remove()



이중 반복문을 통해 이미지를 전부 검사한다. 조건은 공룡색(검은색)인지이고, 만족한다면 해당 색을 흰색으로 바꾼다. 이외의 색은 투명색으로 설정한다.

* 1. draw()

bw(밤)일 때 w로 시작하는 공룡들을 그린다.

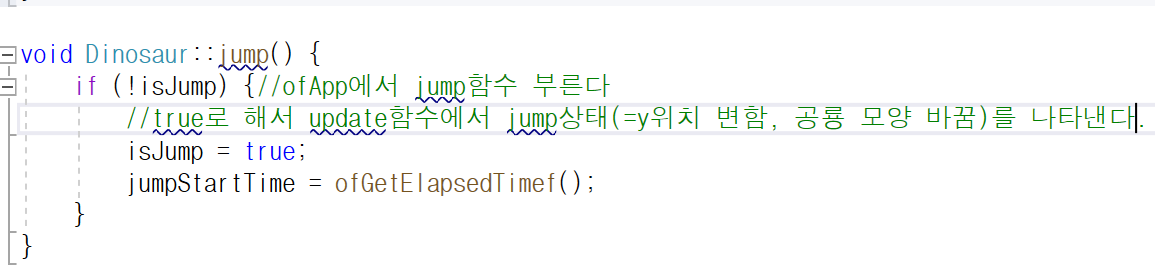
점프 중일때는 발을 바꾸지 않고 기본 공룡만을 내보낸다.

gameover일때는 죽은 공룡을 내보낸다.

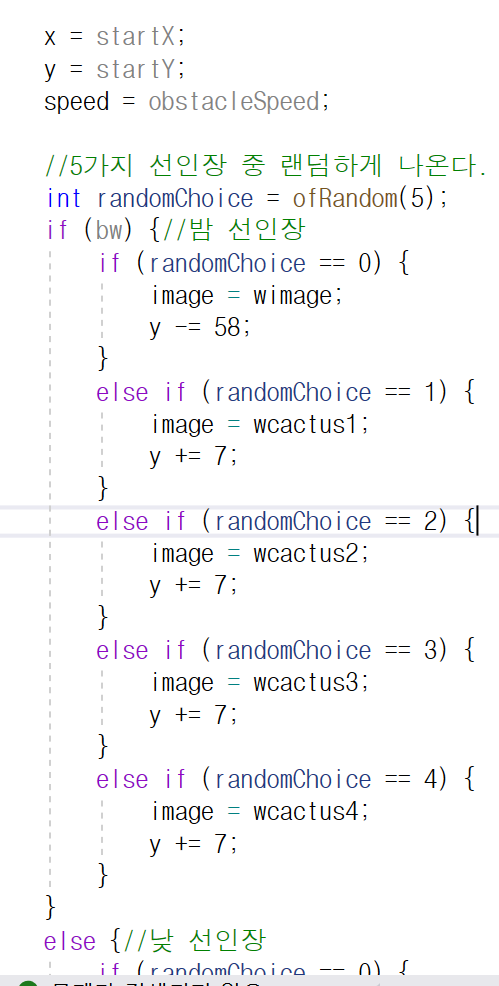
그 외 상황에서는 왼쪽 공룡 오른쪽 공룡을 번갈아 내보내어 공룡을 계속 걷게 한다.

bw가 아닐 때(낮)는 w가 아닌 낮 공룡을 대상으로 똑같은 코드를 사용한다.

* 1. jump()



1. Obstacle
   1. 생성자

Dinosaur때와 비슷하게 기본 이미지를 업로드해서 크기를 조정하고 밤 버전인 흰색 장애물도 따로 저장한다.

이때 선인장은 땅에 박힌 것처럼 보이도록 ground함수를 한번 더 이용한다.

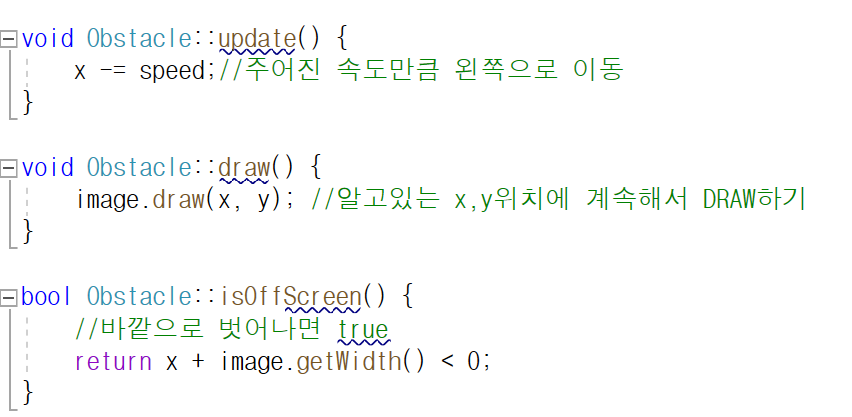
필드에 5가지 선인장 중 하나가 랜덤으로 나오도록 ofRandom함수를 이용해 5개중 하나를 고른다. bw(밤)일때는 w로 시작하는 것을 내보내고, 그렇지 않을 때는 기본 선인장들을 내보낸다.

* 1. remove()

dinosaur의 remove와 거의 같은 함수다.

선인장은 회색의 땅 색을 제외한 선인장 색을 흰색으로 반전시킨다. 그 외의 색은 밤의 배경색인 검은색으로 바꾼다.

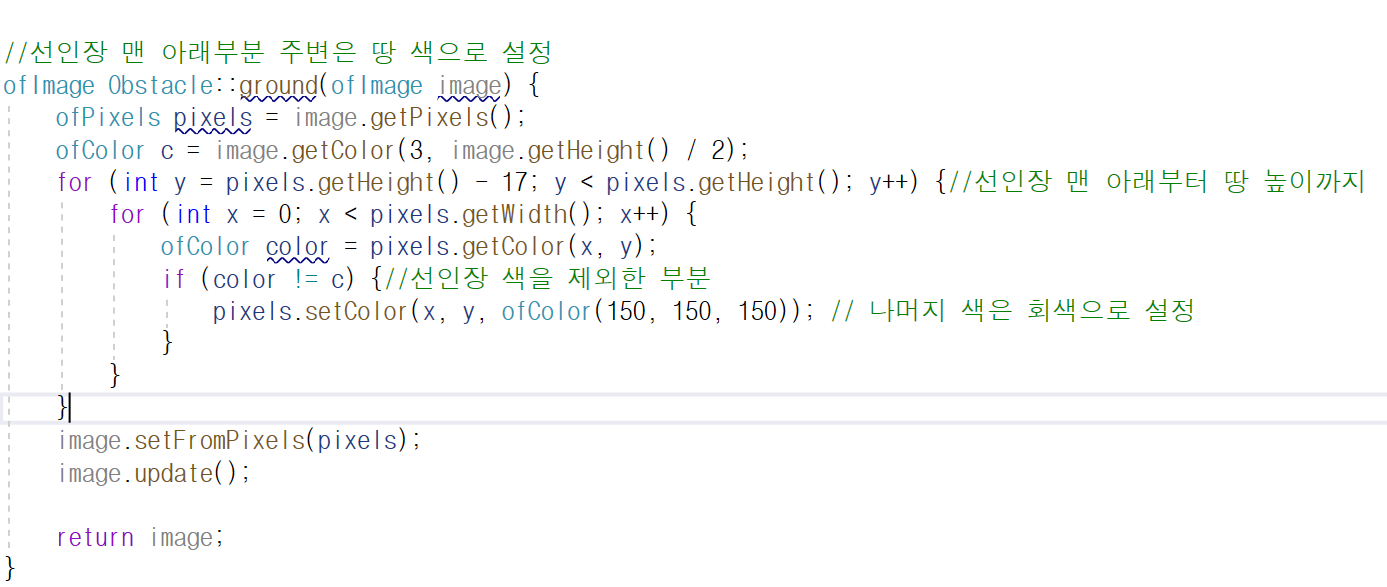
* 1. update(), draw(), isOffScreen()

update에서 속도만큼 x좌표를 왼쪽으로 이동시킨다.

draw에서 주어진 x,y위치에 선인장을 그린다.

isOffscreen에서 선인장이 왼쪽 화면바깥으로 나갔는지 검사한다.

* 1. ground()

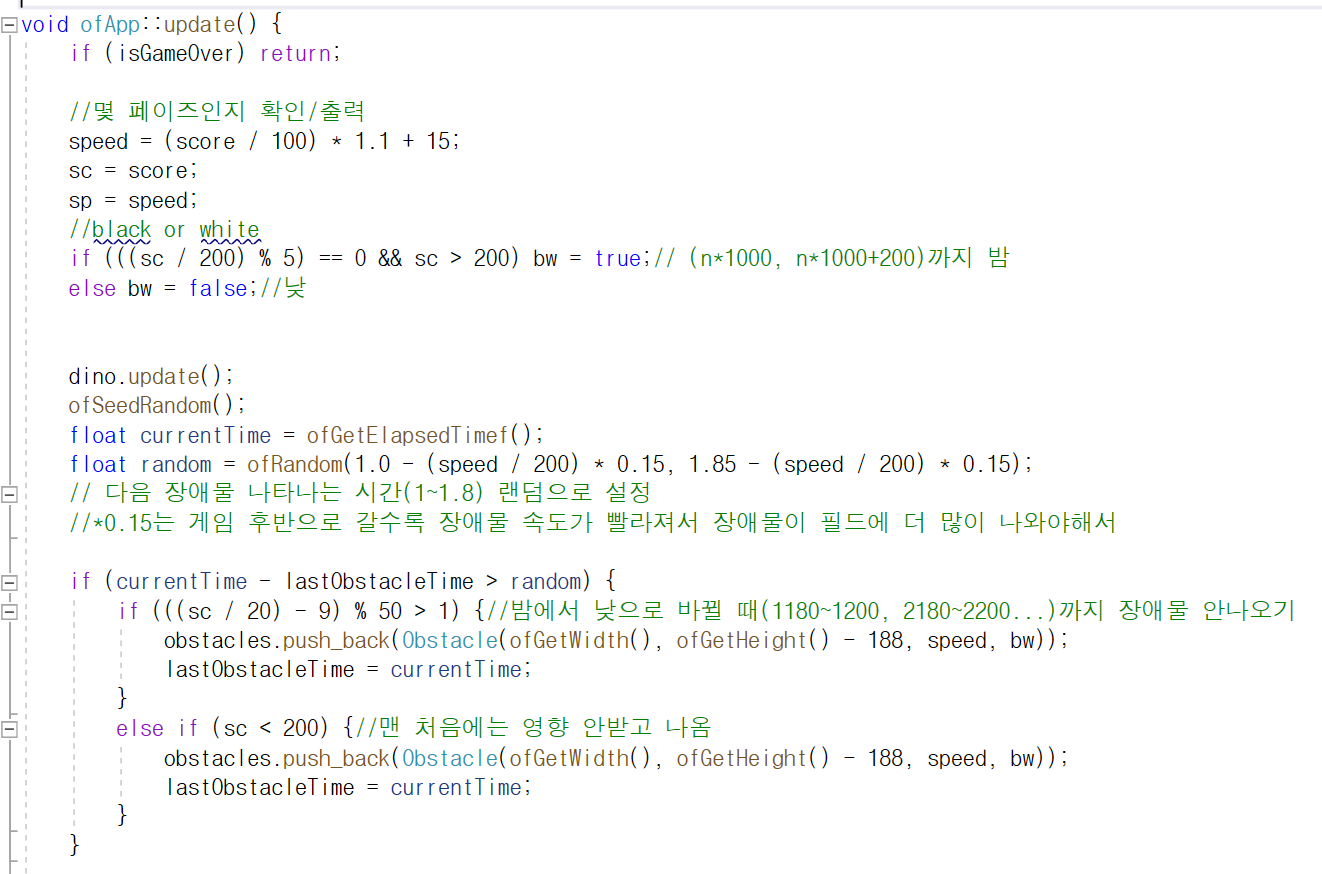


remove와 거의 같다. 지정한 높이, 선인장의 맨 아래부터 땅 높이까지만 선인장 색을 제외한 부분을 땅 색(회색)으로 바꾼다.

1. ofApp
   1. setup()



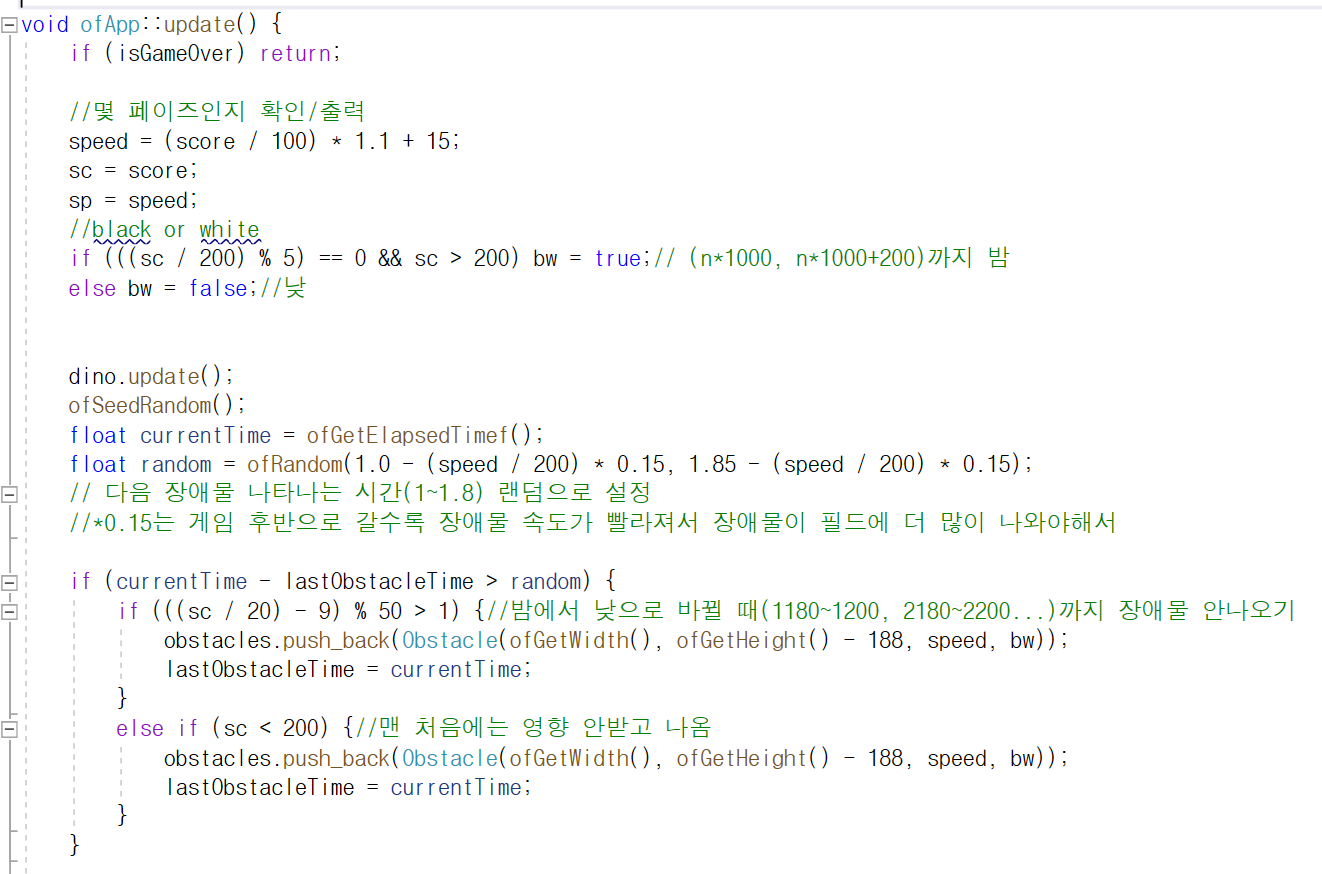
* 1. update()



갱신되는 score를 바탕으로 화면전환 속도인 speed를 계산한다.

점수가 n\*1000~n\*1000+200점 사이일 때 밤으로 바뀐다. bw가 true일 때 밤이고, 배경은 검은색, 공룡과 장애물은 흰색으로 낮의 색깔과 색상반전된다.

bw가 false일때는 낮으로, 배경이 흰색이고 공룡과 장애물은 기본 이미지이다.



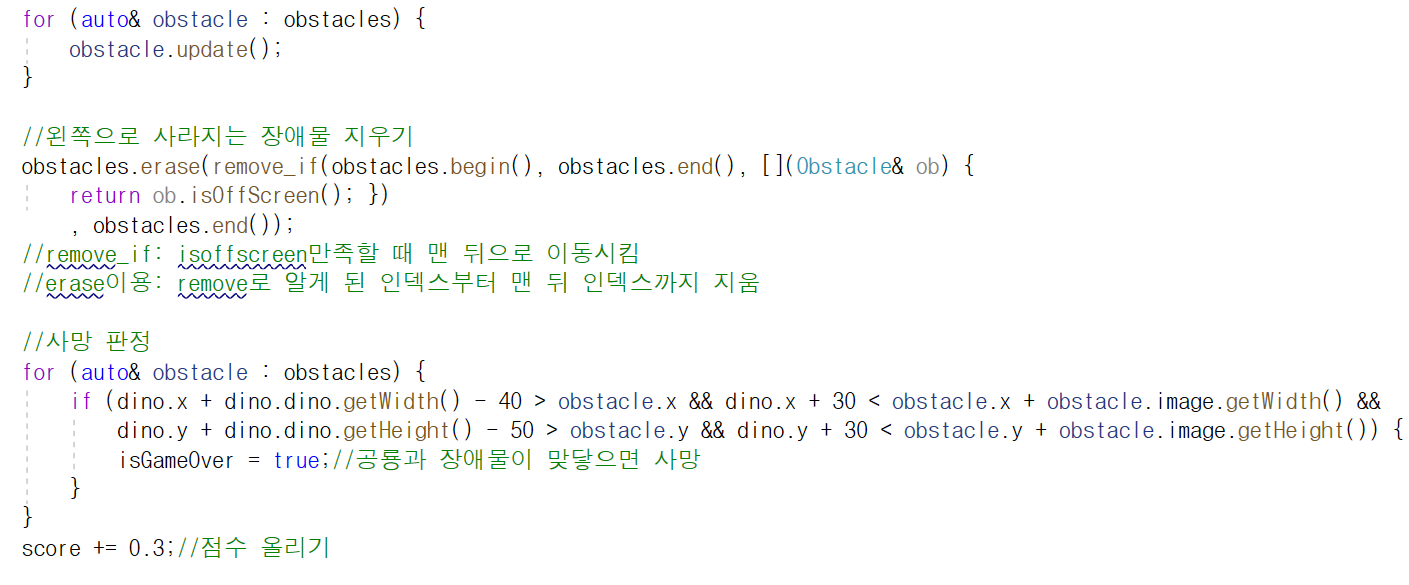
ofRandom을 이용해 다음 장애물이 나타나는 시간을 랜덤으로 정한다.

게임 후반으로 갈수록 속도가 빨라져서 장애물이 더 자주 등장해야하므로

- (speed / 200) \* 0.15 계산을 추가한다.

현재시간과 마지막으로 장애물이 나타난 시간이 random에 부합할 때 새로운 장애물을 생성한다. 즉 obstacles벡터에 push\_back한다.

이때 밤에서 낮으로 넘어가는 n\*1000+180~n\*1000+200점 사이에는 장애물이 나타나지 않도록 한다.



obstacles벡터에 있는 원소들을 계속해서 업데이트해서 그 위치를 움직인다.

remove\_if와 erase 문법을 이용해서 왼쪽으로 사라지는 장애물을 벡터에서 지운다. 미리 만든 isOffScreen을 통해 왼쪽으로 사라졌는지 검사한다.

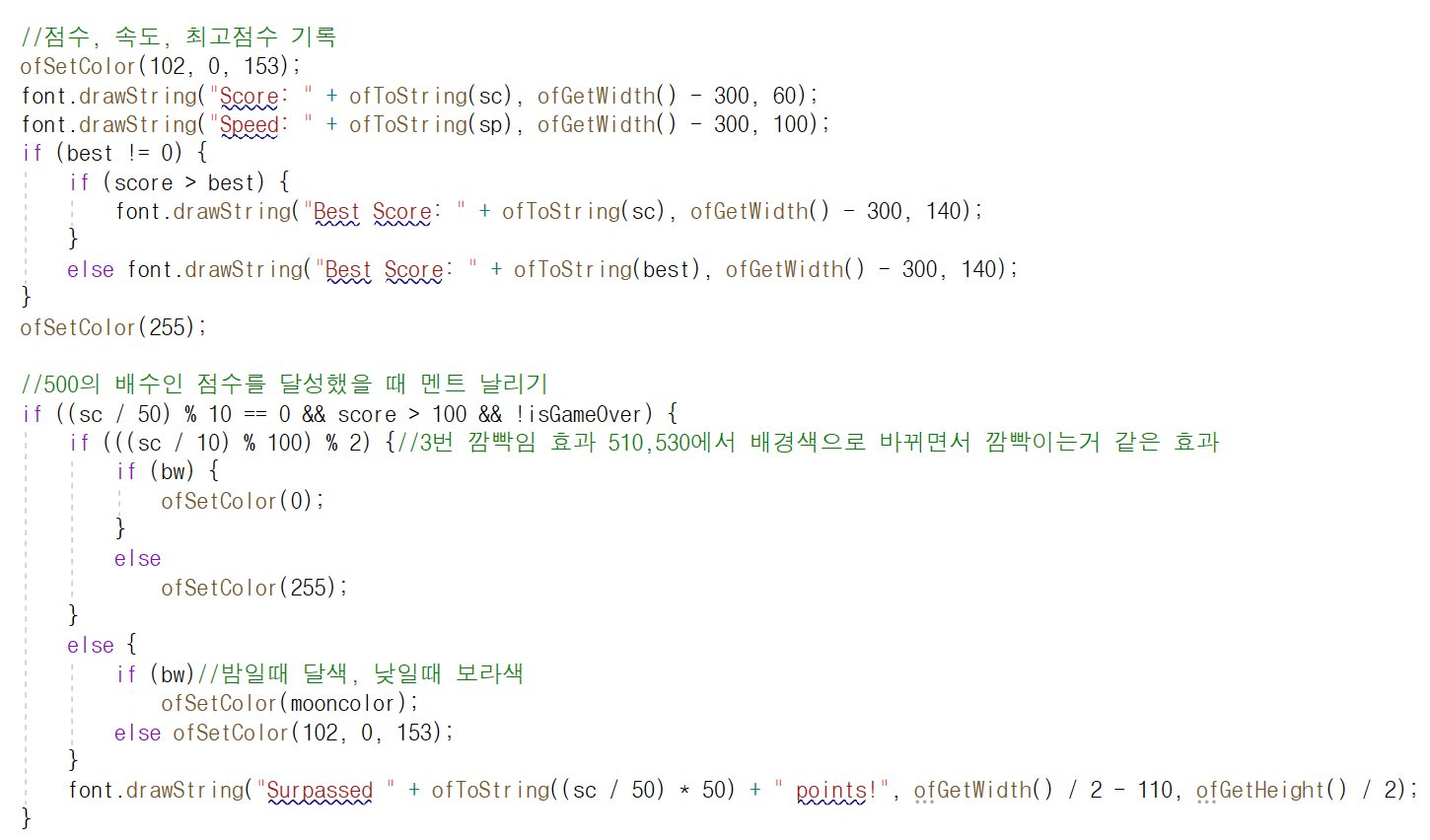
공룡과 장애물이 맞닿으면 게임이 종료되고 공룡이 죽는다.

매 업데이트마다 점수를 0.3점씩 올린다.

* 1. draw()

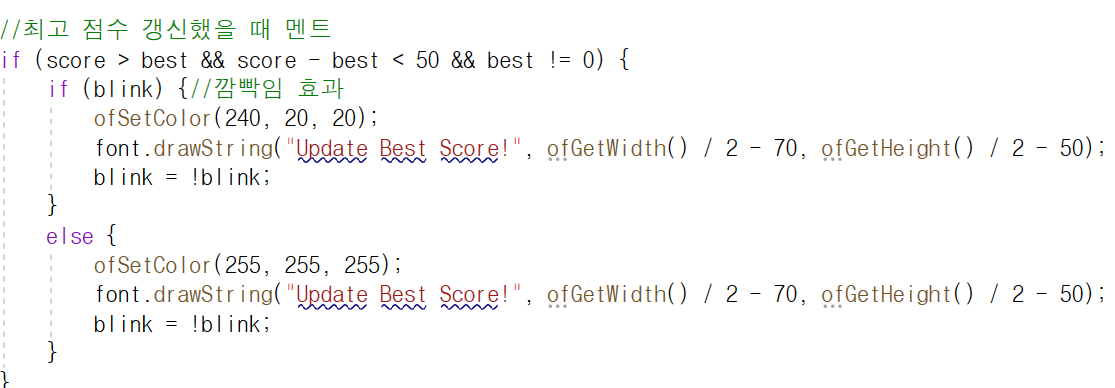
밤과 낮일 때 각각 배경색상을 설정하고 땅, 공룡, 장애물을 순서대로 그린다.

밤에는 화면 왼쪽 상단에 점수에 비례하여 커지는 달을 그린다.

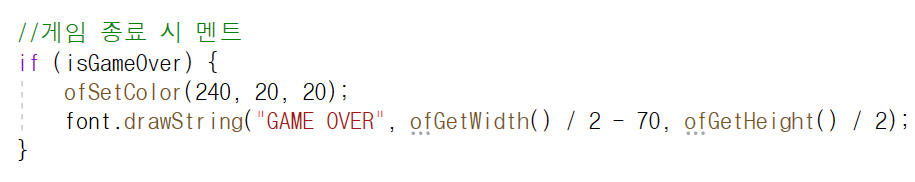


화면 오른쪽 상단에 현재점수, 속도, 최고점수가 출력된다.

점수가 500의 배수를 기록했을 때 화면 정가운데 surpassed $score points!를 3번 깜빡이며 출력한다.

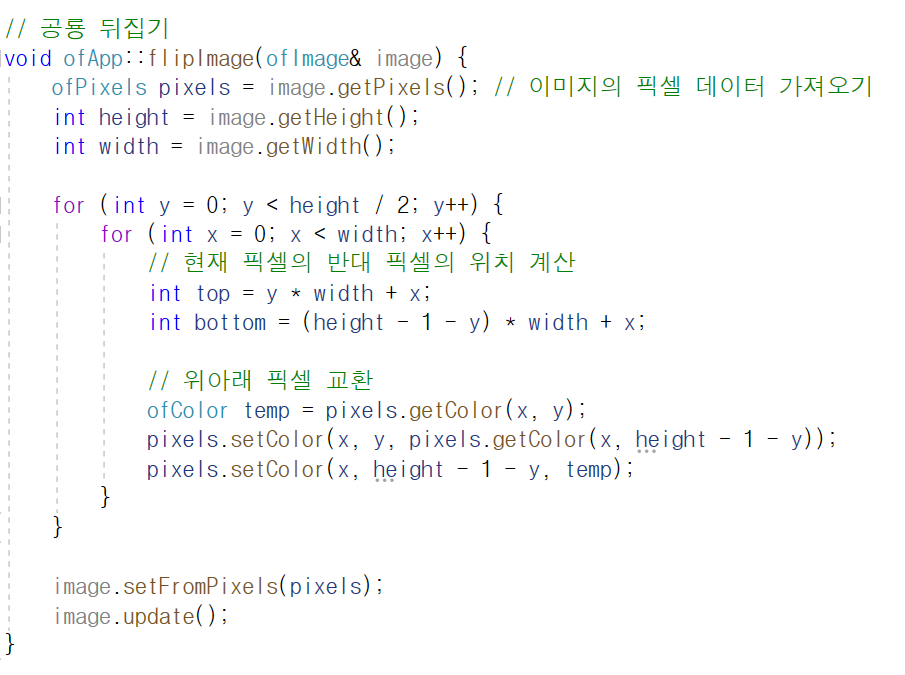


점수가 최고기록을 갱신했을 때 화면 정가운데 update best score! 반복해서 깜빡이며 출력한다.

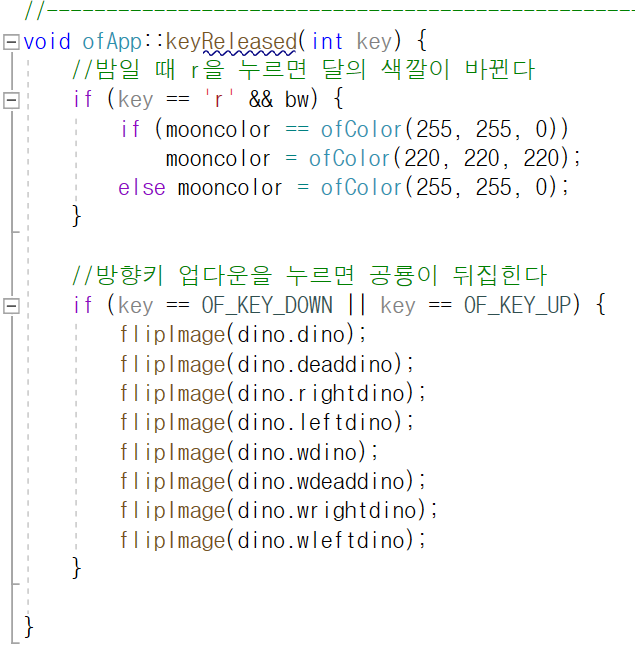


공룡과 장애물이 맞닿았을 때 화면 정가운데 game over뜨며 게임 종료된다.

* 1. flipImage()

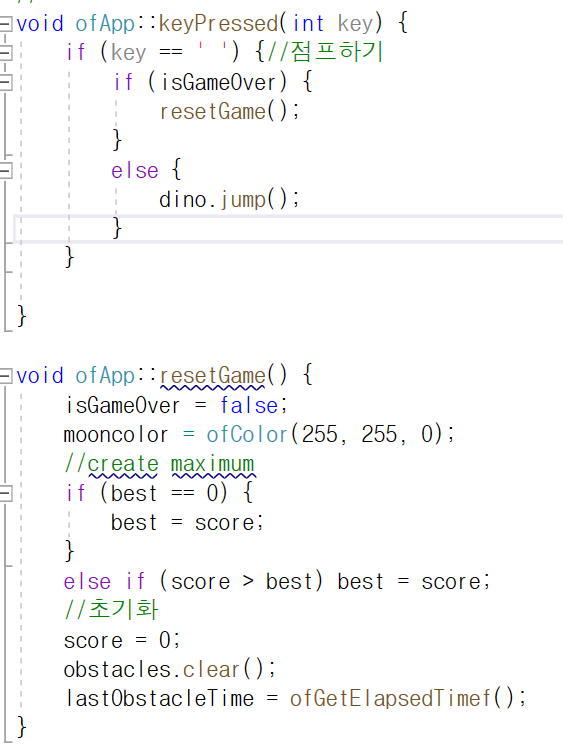
주어진 이미지의 x, y픽셀 정보를 받아들여 y좌표를 반전시킴으로써 공룡의 위아래를 뒤바꾼다.

* 1. keyReleased()

업다운 키: 공룡 상하로 뒤집는다.

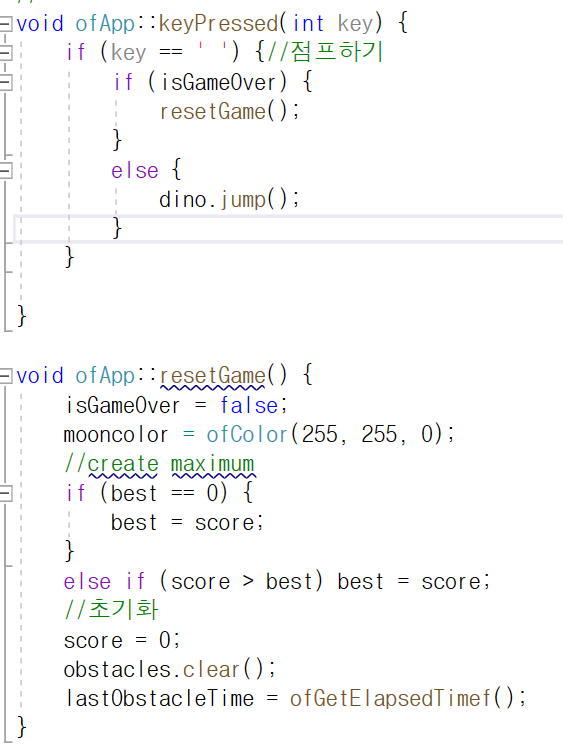
r키: 밤에 누르면 달의 색깔이 바뀐다.

* 1. keyPressed()



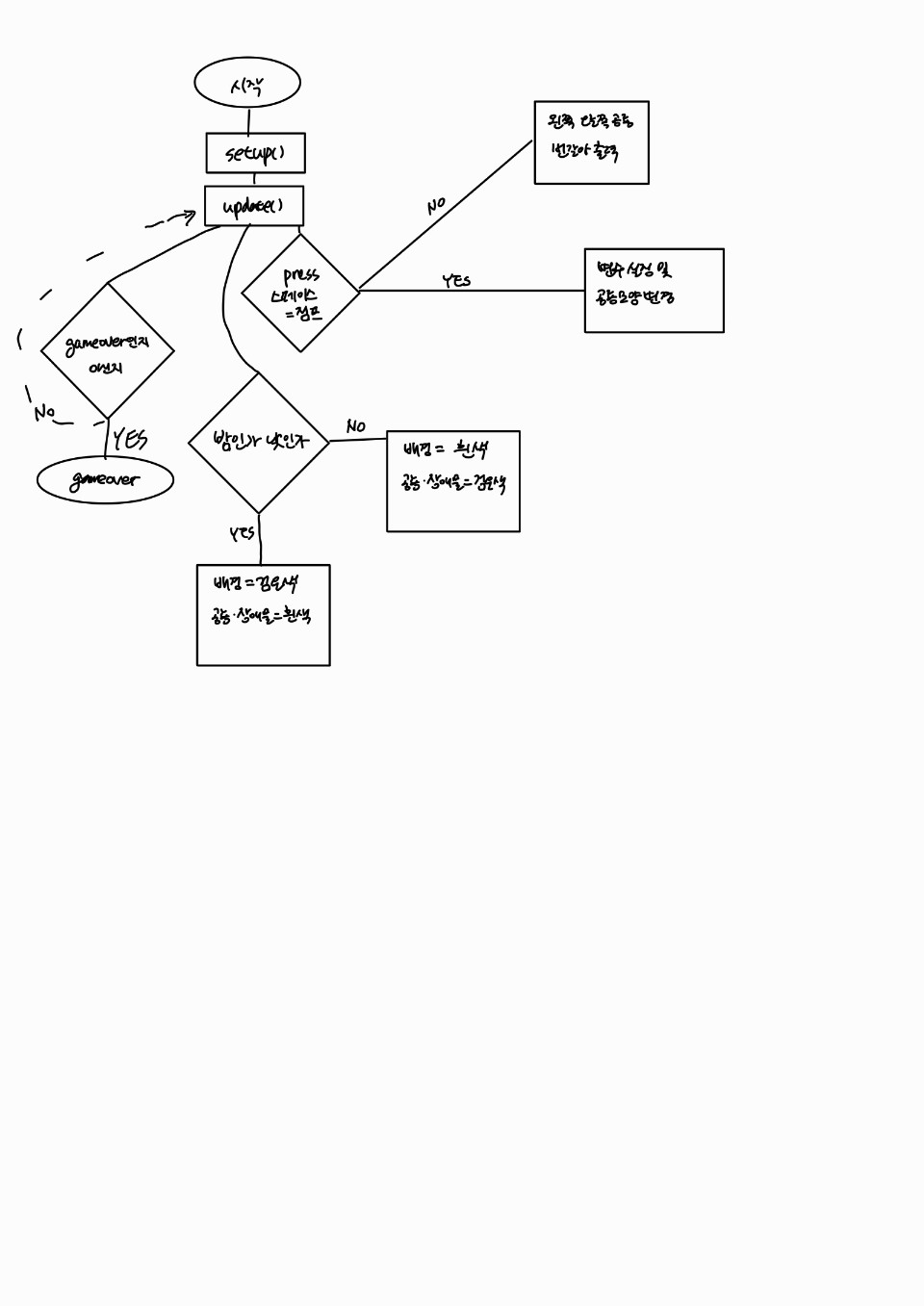
스페이스를 누를 때 점프한다. 죽은 상태라면 스페이스를 누르면 재시작한다.

* 1. resetGame()



bestscore인지 아닌지 검사한다. bestscore보다 높다면 현재점수가 bestscore가 된다.

1. 프로젝트 전체 플로우 차트



1. 자료구조 및 알고리즘, 시간/공간 복잡도

여러 개의 장애물을 저장하기 위해 vector를 이용했다. 각각의 장애물과 공룡을 class객체로 받아들여 ofApp함수에서 이용했다.

ofRandom 알고리즘이 사용됐다.

공룡의 점프 높이를 계산하기 위해 사인함수 계산법을 이용했다.

흰색공룡, 선인장으로 반전시킬 때 이미지의 width\*height만큼 검사를 진행할 때 시간이 소요된다. 그러나 검은색 공룡과 선인장, 흰색 공룡과 선인장을 다른 객체에 저장했기 때문에 메모리는 더 차지하지만 함수 실행 중에 흰색 공룡과 검은색 공룡 사진 수정은 한 번만 이뤄지기에 시간복잡도는 Θ(1)이다.

마찬가지로 선인장이 박혀 있는 효과를 주기 위해 호출한 ground함수도 width\*height의 검사를 진행하지만 프로그램 플레이 중 맨 처음 한 번 호출되기에 Θ(1)이다.

그러나 공룡을 위아래로 뒤집는 기능은 4가지 버전의 흰색, 검은색 공룡 객체를 직접적으로 변환하기 때문에 검사가 매번 이루어진다. 만약 상하키가 끊임없이 눌린다면 Θ(height\*width)의 시간복잡도가 계속해서 발생하여 프로그램의 속도가 느려질 것이다.

이외의 반복문은 화면 안의 장애물을 모두 update하는 코드가 있다. 한 화면에 최대 5개의 장애물이 들어가기 때문에 시간복잡도는 O(5)로 크지 않다.

또 화면의 장애물의 위치와 공룡의 위치를 검사해 공룡의 사망판정이 이뤄지는 반복문이 있다. 이때는 조건문도 포함되므로 O(5\*4)의 시간복잡도를 갖는다. 둘 다 빅오 표기법에 의해 O(1)이지만 전체 프로그램의 시간복잡도에 이 코드는 조금 더 영향을 미칠 것이다.

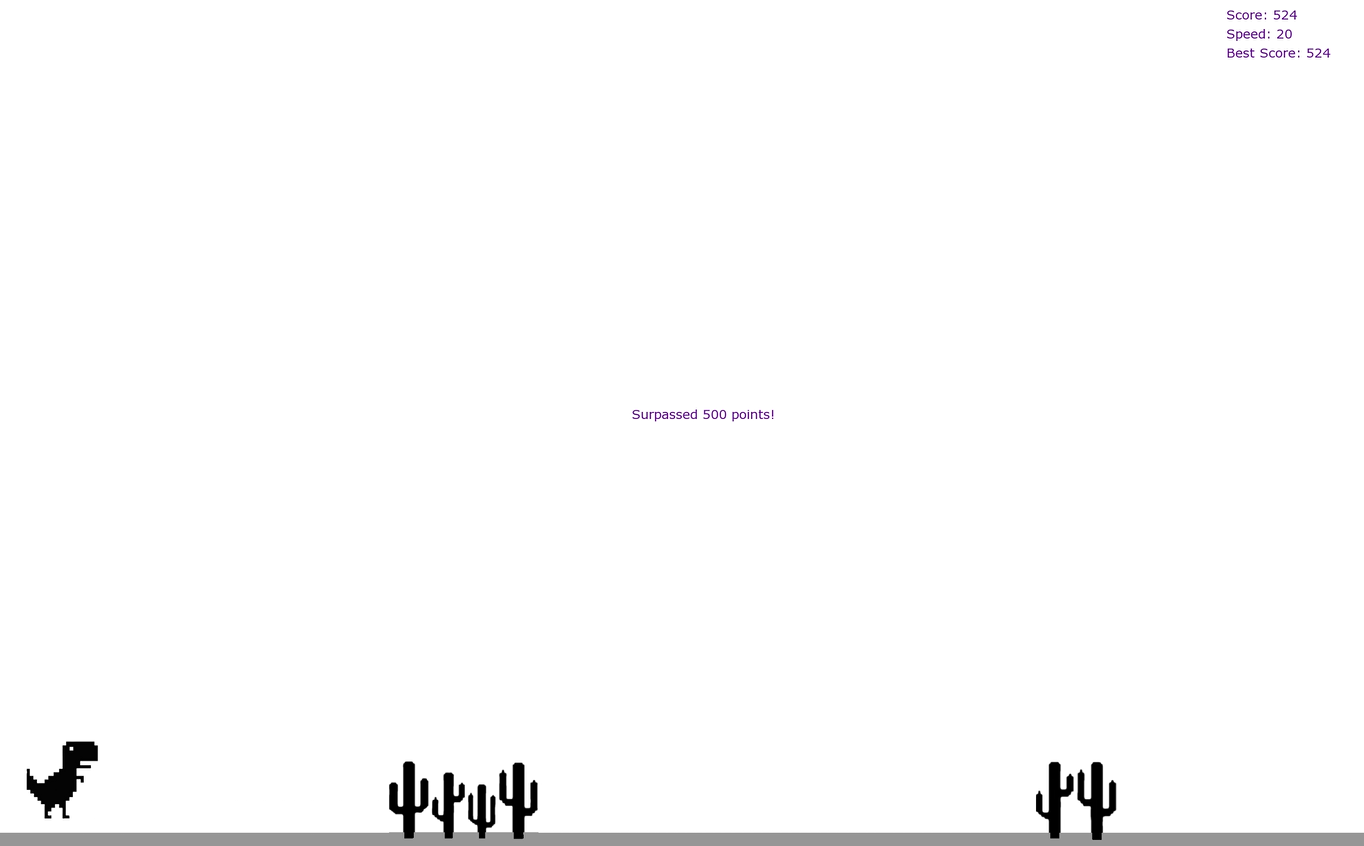
이외에 반복문은 없지만 openframework상 update와 draw함수가 끊임없이 호출되므로 전체 코드에 n개의 조건이 있을 때 함수가 m번 호출된다면 O(2\*n\*m)시간복잡도를 가질 것이다.

1. 창의적 구현항목

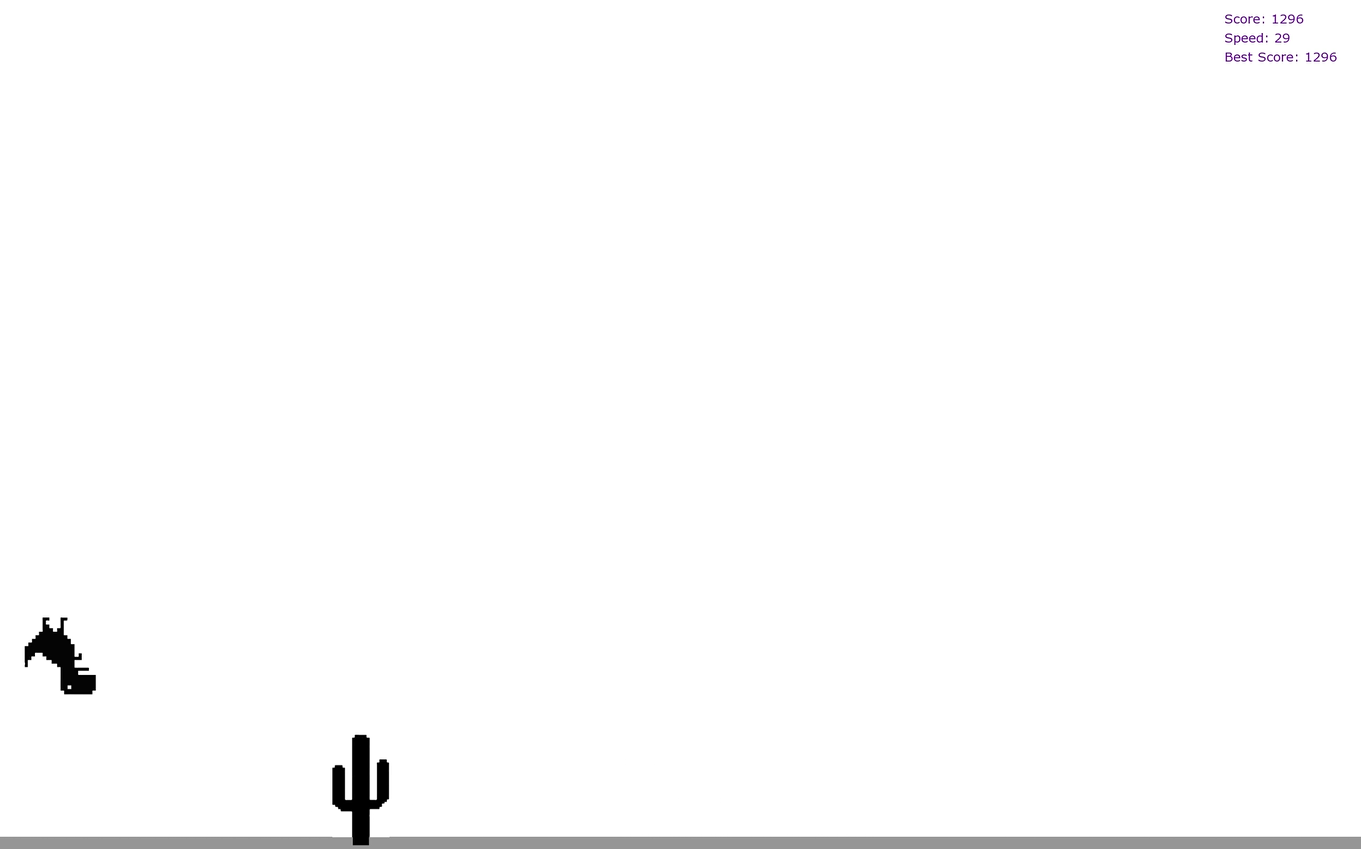
* 점수가 n\*1000~n\*1000+200점 사이일 때 낮에서 밤으로 바뀐다. 이때 밤에서 낮으로 넘어가는 n\*1000+180~n\*1000+200점 사이에는 장애물이 나타나지 않는다.
* 밤에는 왼쪽 위에 달이 뜬다. 달의 색상은 r키를 눌러 변경할 수 있다. 달의 크기는 점수에 비례하여 점점 커진다.
* 점수가 최고기록을 갱신했을 때 화면 정가운데 update best score! 반복해서 깜빡이며 출력한다.
* 오른쪽 위에 현재 점수, 속도, 최고점수가 출력됨
* 점수가 500의 배수를 기록했을 때 화면 정가운데 surpassed $score points! 3번 깜빡이며 출력된다.
* 점수가 최고기록을 갱신했을 때 화면 정가운데 update best score! 반복해서 깜빡이며 출력된다.
* 업다운 키를 눌렀을 때 공룡이 상하로 뒤집힌다.

1. 프로젝트 실행 결과

500점 돌파



뒤집힌 공룡



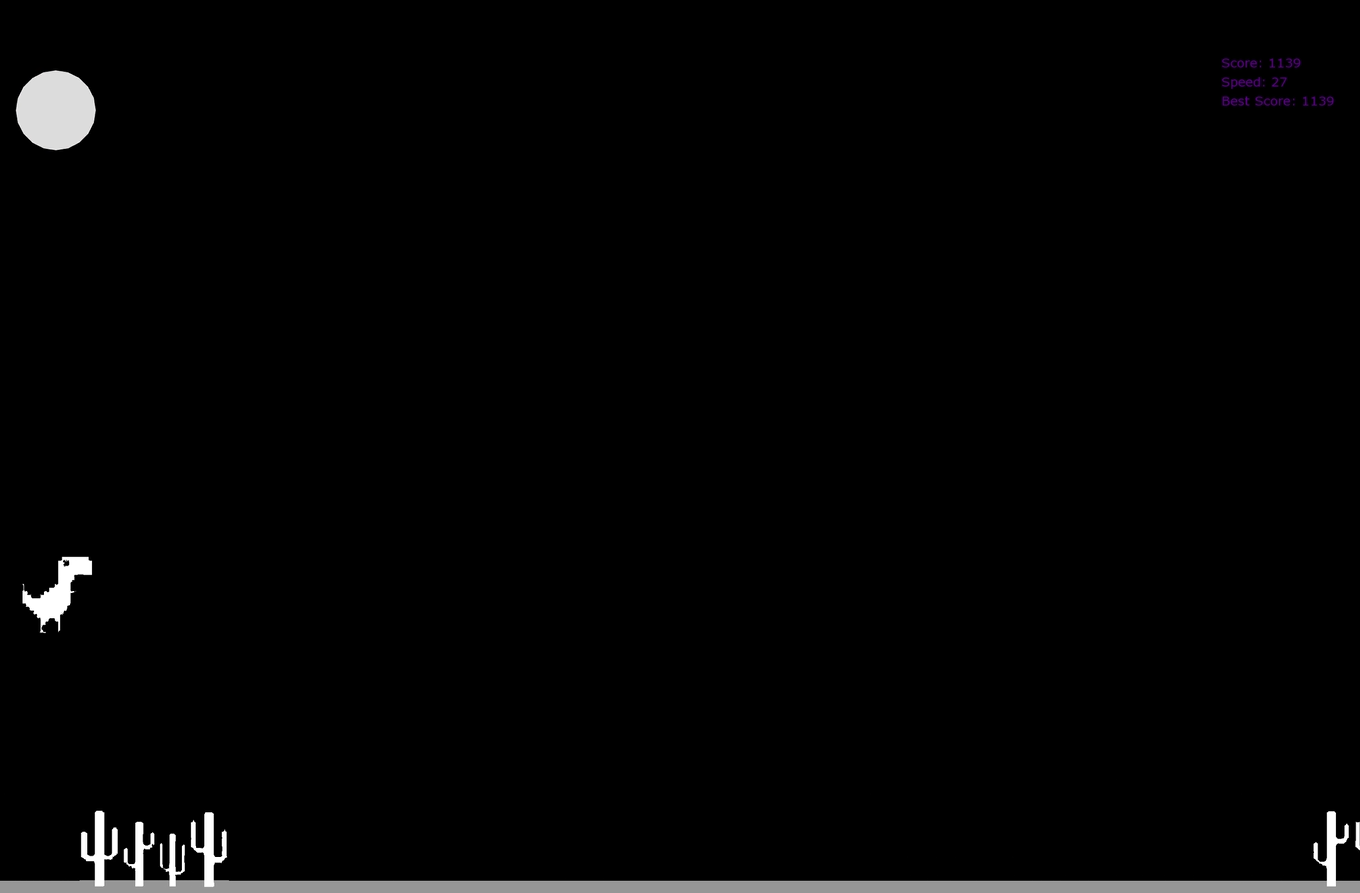
밤에 색상 반전 버전



점수 증가에 따른 달 크기 증가

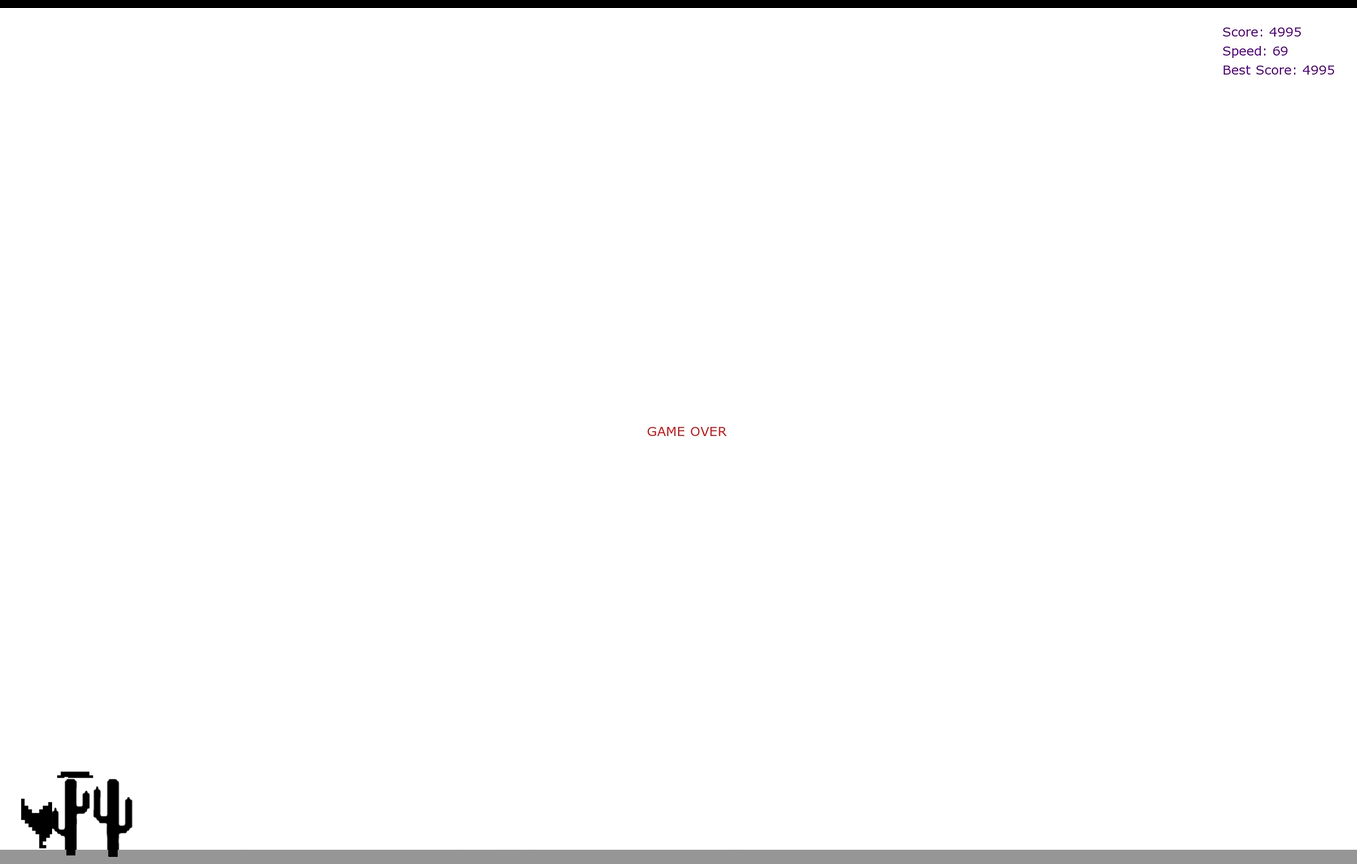


바뀐 달 색



최고기록갱신

게임 오버



모든 화면의 오른쪽 위에 현재 점수와 속도, 최고 점수가 나온다. 최고점수가 0일 때는 화면에 표시되지 않는다.

1. 느낀 점 및 개선 사항

기존 공룡게임에서 제공하는 숙이기 기능, 까마귀, 하늘 위 구름 등 구현하지 못한 기능들이 많다. 이 게임을 더 발전시킨다면 더 재밌어질 것이다.

공룡이 죽는 판정을 어떻게 할까 고민을 많이 하였다. 결국 공룡과 장애물의 너비, 높이를 바탕으로 겹쳤을 때 죽는 것으로 했지만 공룡 자체의 모양과 장애물 각각의 모양 정보를 안다면 그 필드 정보를 바탕으로 더 좋은 판정을 이룰 수 있을 것 같다.

공룡과 선인장이 사각형이 모양으로 필드에 나타나 마지막 사진처럼 두 사진이 겹쳤을 때 사각형 모양이 그대로 드러난다. 오로지 검은색인 객체 모양만 나타나면 더 좋을 것이다.

직접 좋아하는 게임을 구현하고 실행해보는 좋은 경험이었다. openframeworks의 사용법과 원리를 아주 조금은 알 수 있었다.