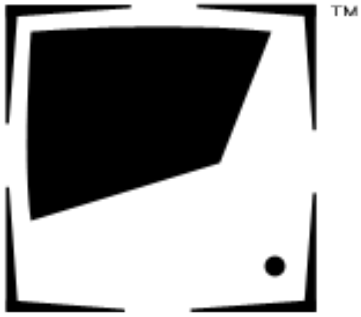


jokebox.

v4.0

튜토리얼 (Level A 기본)

1. 게임 초기화하기 (initialize)
2. 무언가 그려보기 (draw)
3. 캐릭터 이동시키기 (update)
4. 간단한 애니메이션 만들기 (update)

Powered by GPOS The World Creatives

게임 초기화하기

어떤 프로그램이든 먼저 초기화를 하고 시작해야겠지요. 게임도 마찬가지로, 캐릭터 초기 위치나 스코어 등등을 게임에서 바로 사용할 수 있도록 초기화해주어야 합니다. Jokebox에서는 **초기화** 작업을 주로 **initialize** 함수에서 진행하는데요, 깨끗한 Jokebox 프로젝트에서 **game.c** 소스를 켜고, `#include` 바로 밑에 다음 변수들을 선언해봅시다.[1]

```
1  #define JOKEBOX_LEVEL_A
2  //#define JOKEBOX_LEVEL_B
3  #include "jokebox.h"
4  #include "jokeboxhelper.h"
5  #include "game.h"
6
7  VECTOR2 pos;
8  char ch;
9
10 void initialize()
   {
```

[이 소스가 하는 일]

- **Level A 함수를 사용하도록 선언하였습니다.**

Jokebox는 복잡도나 직관성에 따라 함수가 두 분류(Level A/Level B)로 나뉘어져 있는데, Level A에는 주로 유사-CUI 환경을 구현하는 데 적합한 직관적인 함수들이 있으며, Level B에는 2D 게임을 구현할 수 있을 정도의 함수들이 있습니다. 쉽게 표현하면 Level A에는 쉬운 함수들이, Level B에는 고급 기능을 가진 함수들이 있다고 할 수도 있겠습니다.

이번 튜토리얼에서는 쉬운 접근을 위해 직관적인 Level A를 사용하도록 하겠습니다. Level A를 사용하기 위해 **1번 줄**의 `#define JOKEBOX_LEVEL_A` 를 주석 해제하였고, **2번 줄** `//#define JOKEBOX_LEVEL_B`는 주석처리 하였습니다.

- **변수를 몇 개 선언했습니다.**

게임에서 쓰기 위해 몇 가지 변수를 선언했습니다. **pos**라는 변수는 캐릭터의 위치를 저장하기 위해, **ch**라는 변수는 캐릭터의 모양(글자)를 저장하기 위해 선언한 것입니다. 특히 **pos**라는 변수는 위치를 쌍으로 묶어 관리하기 위해 **VECTOR2**라는 구조체를 사용하였습니다. **VECTOR2**는 `#include "jokeboxhelper.h"`으로 포함된 구조체로, (x, y) 쌍을 관리할 때 편리하게 사용할 수 있습니다.

Jokebox에 대한 첫 설명이니만큼 Jokebox 자체에 대한 원론적인 설명을 덧붙였는데, 잘 이해가지 않는 부분도 있을 수 있습니다. 겁먹지 말고 그냥 둥그스름하게 이해한 다음, 다음 단계로 일단

[1] 회색 글자는 새로 적을 코드(색깔이 있는 글자) 전후에 위치할 코드들이므로 따로 적지 않아도 됩니다.

진행해 봅시다. 이제 이 변수들을 초기화할 차례인데요, initialize 함수에 다음과 같이 작성해봅시다.

```
1  void initialize()
2  {
3          game_set_window_title("My First Game");
4
5          pos.x = game_get_cell_x_count() / 2;
6          pos.y = game_get_cell_y_count() / 2;
7
8          ch = '0';
9  }
```

[이 소스가 하는 일]

- **먼저 프로그램의 제목부터 바꿨습니다.**

3번째 줄에서 **game_set_window_title** 함수를 볼 수 있습니다. 이 함수는 인수로 전달되는 문자열을 프로그램의 제목으로 지정하는 함수인데요, 프로그램 제목은 제목 표시줄에 그대로 나타납니다.

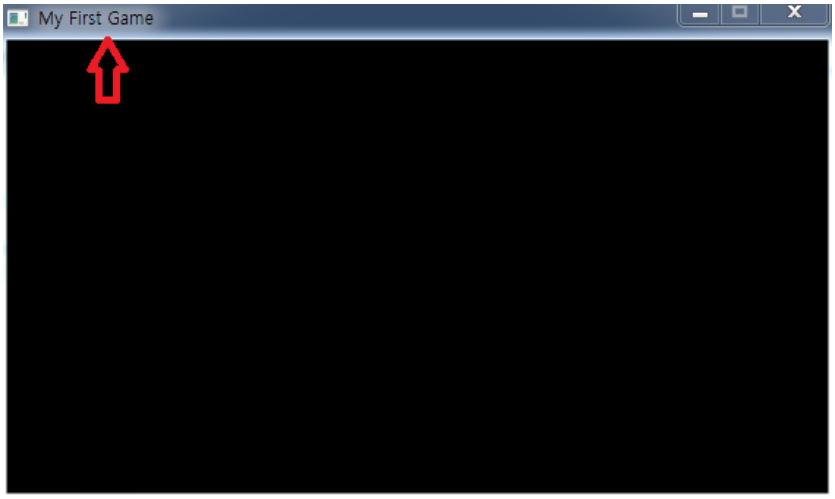
- **캐릭터의 위치를 초기화했습니다.**

이전 단계에서 **pos** 변수에 캐릭터의 위치를 저장하기로 했던 것 기억하시는지요? 5~6번째 줄에서는 캐릭터의 위치를 화면의 중앙으로 초기화하였습니다. **game_get_cell_x_count, game_get_cell_y_count** 함수는 각각 화면의 가로, 세로 폭을 리턴하는 함수입니다.

- **캐릭터 모양은 '0'으로 정했습니다.**

변수 **ch**는 캐릭터 모양을 저장하는 변수인데요, 이것을 문자 '0'으로 지정해 주었습니다.

코드를 작성하고 빌드를 눌러 실행시켜 보면 제목 표시줄에 'My First Game'이 표시되고 있는 것을 볼 수 있습니다.



이번 장에서는 간단히 게임과 관련된 변수들을 초기화 하였습니다. 쓰인 함수 가운데에는 유독

game_으로 시작하는 함수가 많았는데요, **game_**으로 시작되는 함수들은 게임 시스템과 관련된 함수들입니다. Jokebox에서 게임 시스템과 관련하여 지원하는 함수를 간략히 훑어보면 다음과 같습니다.

game_exit()	게임을 즉시 종료
game_playing()	게임 프로그램이 실행 중인지 여부
game_set_window_title()	프로그램 제목을 지정
game_get_cell_x_count()	화면상 칸의 가로 개수를 리턴
game_get_cell_y_count()	화면상 칸의 세로 개수를 리턴
game_set_cell_size()	칸(cell)의 사이즈를 설정
game_set_cell_count()	화면상의 칸의 가로/세로 개수를 설정

이제 초기화도 마쳤으니 화면에 뭔가를 그려보도록 합시다.

무언가 그려보기

Jokebox에서는 그리기와 관련된 모든 코드를 **draw** 함수 안에서 작성합니다. 이제 위에서 초기화한 위치와 캐릭터의 모양을 가지고 뭔가를 그려볼 텐데요, 그 전에, 먼저 Jokebox가 어떻게 그림을 그리는지 이해해봅시다. Jokebox는 움직이는 캐릭터를 어떻게 그리는 걸까요? 우리가 생각할 수 있는 방안은 두 가지 정도가 있습니다.

A안. 화면에서 캐릭터 그림'만' 지우고, 움직인 캐릭터 위치에 캐릭터'만' 다시 그린다.
B안. 전부 다 지우고 전부 다 다시 그린다.

어떤 쪽이 더 편리하고 확실한 방법일까요? 우리가 **'컴퓨터는 무지하게 빠르다'**라는 것을 받아들이고 나면, **B안**이 쉽고 확실하다는 것을 바로 알 수 있습니다. 컴퓨터(엄밀히는 CPU와 GPU)는 우리가 생각하는 것 이상으로 빠르기 때문에, 밀리세컨드(ms)단위로 **지웠다 그렸다 지웠다 그렸다 지웠다 그렸다** 하는 것을 반복 해도 무리가 없습니다. 따라서 근래의 그래픽 표현은 모두 **B안**의 방법을 채택하고 있고, Jokebox 역시 **draw** 함수를 실시간으로 계-속 호출함으로써 B안을 구현하고 있습니다.

이제 그리기가 어떻게 이루어지는지 알았으니, 실제로 그리기 코드를 작성해봅시다.

```
1  void draw()
2  {
3          draw_begin();
4          draw_clear(BLACK);
```

```
5
6           draw_char(ch, pos.x, pos.y);
7
8           draw_end();
9   }
```

[이 소스가 하는 일]

- **draw함수를 쓰기 전에 draw할 준비를 시킵니다.**

draw 함수는 **initialize** 함수와는 다르게, 무언가 그리게 하기 위해서는 사전에 엔진을 준비시켜야만 합니다. 비유를 하자면, 운전을 하려면 먼저 기어부터 넣어야겠지요. 어쨌든 엔진에게 '이제 draw 할꺼야' 라는 걸 알려주는 의미에서 **draw_begin()**이라는 함수를 호출합니다. 이 함수가 호출되면 엔진은 창에 뭔가를 그릴 준비를 합니다.

- **이전에 그린 것들은 전부 지웁시다**

전부 다 다시 그리기 전에, 일단 먼저 전부 지워야 할 것입니다. 6번째 줄의 **draw_clear()**함수가 그 역할을 하고 있습니다. 인수로 전달된 **BLACK**은 열거형 **COLOR**의 값 중 하나인데요, **COLOR**에는 검은색(BLACK) 말고도 여러 가지 색을 추가로 더 가지고 있습니다. 다른 색에 대해서는 **레퍼런스** 문서를 참조해 주세요!

- **드디어 뭔가를 그려볼 것입니다.**

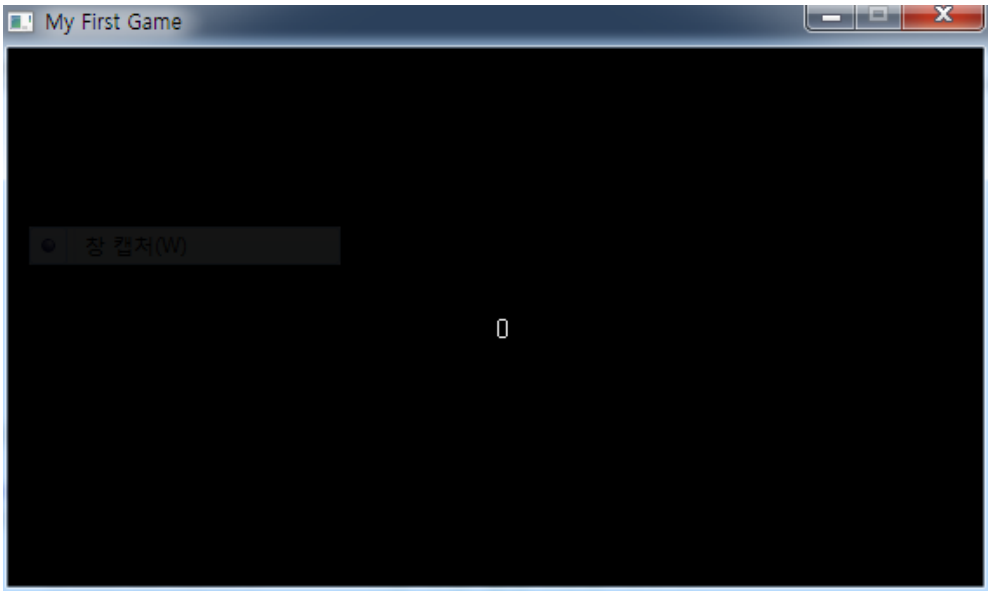
준비시키고 지우는 작업을 했으니, 이제 실제로 뭔가를 그려볼 차례입니다. 6번째 줄에선 **draw_char()** 함수를 이용해서 드디어 화면에 문자 ch를 그리라는 명령을 내렸습니다. **draw_char** 함수에는 총 3개의 인수가 들어가는데요, 첫째 인수로 **화면에 그릴 문자**, 둘째와 셋째 인수로 **문자를 그릴 x, y 좌표**를 넣습니다.

- **다 끝났다는 걸 엔진에게 알립니다.**

자동차도 운전 후에 엔진을 꺼야 하는 것처럼, draw가 끝난 후에 draw 마치기를 선언해야 합니다. 8번째 줄에서 **draw_end()** 함수를 호출시켜 draw가 끝났음을 엔진에게 알리고 있습니다.

결론적으로, 이 코드는 문자 '0'을 화면 중앙에 그리라는 명령을 내리고 있는 것입니다. 여기서 draw_char() 함수에 좌표 (x, y)를 넣어줄 때, 넣어준 값이 **'칸 단위'**라는 것을 숙지하도록 합시다. 콘솔(명령 프롬프트)에서 글자들이 칸 단위로 출력되듯이, Jokebox 역시 **칸(cell)을 기준으로 위치나 길이를 표현**합니다. 여타 라이브러리에서 자주 사용하는 픽셀 단위와 헷갈려서는 안되겠습니다.

이제 프로젝트를 빌드시키고 실행시켜보면, 화면 중앙에 '0'이 그려진 것을 볼 수 있을 것입니다.



우리가 넣어준 문자 0이 화면 정중앙에 위풍당당하게 서 있는 것을 볼 수 있습니다. 그런데 **draw_char()** 함수를 이용해서는 검은 바탕에 흰 글자를, 그것도 한 번에 하나밖에 그릴 수 없습니다. 비록 글자를 그리는 것에 불과할 지라도, Jokebox에서는 좀 더 편리하고 아름답게(?) 그릴 수 있는 방법으로 여러 가지 draw 함수를 제공합니다. 아래는 현재 Jokebox에서 쓸 수 있는 draw 함수를 열거한 것입니다.

draw_char()	검은 바탕에 흰색으로 글자 한 개 그리기
draw_charc()	지정한 바탕색과 글씨색으로 글자 한 개 그리기
draw_string()	검은 바탕에 흰색으로 문자열 그리기
draw_stringc()	지정한 바탕색과 글씨색으로 문자열 그리기
draw_area()	검은 바탕에 흰색으로 사각형 영역 그리기
draw_areac()	지정한 바탕색과 글씨색으로 사각형 영역 그리기

이들 함수의 자세한 사용법은 **레퍼런스 문서**를 참조하도록 합시다.

지금까지 뭔가 그려보긴 했지만, '0'만 덩그라니 세워놓고서는 게임이라고 할 수 없습니다. 이제 게임을 만들기 위한 기초 단계로, 캐릭터를 움직이는 코드를 작성해 봅시다.

캐릭터 이동시키기

지금까지 **initialize** 함수와 **draw** 함수를 채워보았습니다. **initialize** 함수는 게임에 이용할 각종 변수를 초기화하고, 제목을 붙이는 등의 초기 작업을 하는 곳이고, **draw** 함수는 말 그대로 이것 저것을 'draw'하는 곳입니다.

그렇다면 남은 함수인 **update**에서는 무엇을 해야 할까요? 쉽게 풀이하자면, **update** 함수에서는 위에 두 가지를 제외한 모든 작업을 수행합니다. 키보드 입력을 확인하고, 타이밍에 맞춰 장애물 좌표를 조정하고, 게임오버를 체크하는 등.. 게임 작동(게임 로직)과 관련된 일련의 코드들이 바로 **update**에 들어가야 할 코드입니다.

이러한 작업들 역시 draw와 마찬가지로 실시간으로 이루어져야만 하는 작업들인데요, 그래서 Jokebox는 **draw** 함수와 더불어 **update** 함수도 실시간으로 호출합니다. 굳이 정확하게 이야기 한다면, 매 순간마다 **update – draw – update – draw – update – draw - …** 순서로 끝없이 번갈아 호출할 것입니다.

한편, 우리가 이번에 구현할 캐릭터 움직이기도 **update** 함수에서 이루어져야 합니다. 이전 장에서 **draw_char()**를 하면서, 그릴 좌표로 **pos.x, pos.y**를 넣어줬던 것 기억나시는지요? **draw** 함수에서는 이제 지속적으로 ch에 저장된 문자를 **pos.x, pos.y**에 그려줄 것입니다. **update** 함수에서 바로 이 변수들의 값을 조정한다면, 캐릭터를 시시각각으로 다른 위치에 그려줄 수 있을 것입니다.

이렇게 전략을 짜고, 이제 **update** 함수에 아래와 같이 코드를 작성해 봅시다.

```
1  void update(GAMETIME gametime)
2  {
3          if (keybd_is_key_pressed(KEY_LEFT))
4                  pos.x--;
5          else if (keybd_is_key_pressed(KEY_RIGHT))
6                  pos.x++;
7          else if (keybd_is_key_pressed(KEY_UP))
8                  pos.y--;
9          else if (keybd_is_key_pressed(KEY_DOWN))
10                 pos.y++;
11 }
```

[이 소스가 하는 일]

- **먼저 무슨 키가 눌렸는지 확인합니다.**

Jokebox는 우리가 키보드를 확인하든 확인하지 않든, 항상 어떤 키보드가 눌렸는지 감시하고 있습니다. 우리가 할 일은 Jokebox에게 '이런 키가 눌렸었니?'라고 물어보는 것입니다.

keybd_is_key_pressed() 함수는 인수로 전달되는 키가 방금 눌렸는지 Jokebox에게 물어보는 함수입니다. 만약 그 키가 눌렸다면 1, 즉 true를 리턴할 것이고, 아니면 0(false)를 반환할 것입니다.

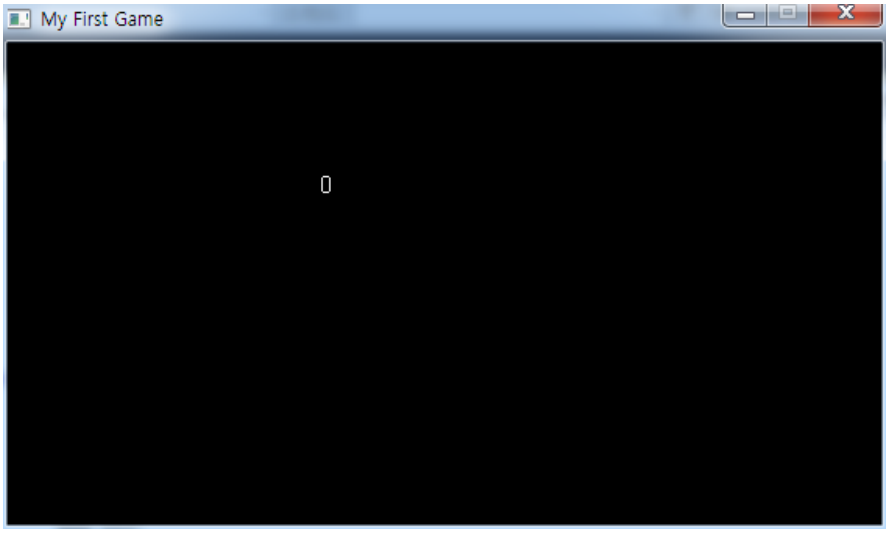
인수로 전달되는 **KEY_LEFT, KEY_RIGHT**..와 같은 값은 Jokebox에서 정의하는 열거형 **KEYS**의 값 중 하나입니다. 열거형 **KEYS**에는 방향키 외에도 다양한 키가 정의되어 있는데요, 이들 키에 관해서는 **레퍼런스 문서**를 참조해주시기 바랍니다.

- **방향키가 눌렸다면, 캐릭터 좌표를 이동시킵니다.**

어떤 방향키가 눌렸는지 확인한 후에, 해당 방향키가 눌렸다면 그 방향으로 좌표를

이동시켜줍니다. 여기서 주의해야 할 점은, Jokebox에서 **원점** (0, 0)은 **화면의 왼쪽 위**이며, **+y 방향** 역시 화면 위에서 **아래로 향하는 방향**이라는 점입니다.

이제 코드를 작성하고 게임을 실행시켜 봅시다. 방향키를 이리저리 눌러, 캐릭터가 잘 움직이고 있는지 확인해 봅시다. 혹시 '↑' 버튼을 눌렀는데 캐릭터가 아래로 내려간다면, Jokebox의 좌표 시스템에 유의하여 코드를 수정해봅시다.



참고로, 이번 코드에서 사용된 **keybd_is_key_pressed()**는 '방금' 키가 눌렸는지 여부를 반환하는 함수입니다. 만약 키가 '방금' 떼어졌는지 여부를 알고 싶다거나, 계속 눌려 있는지 여부를 알고 싶다면 어떻게 해야 할까요?

이런 경우를 위해 Jokebox에서는 키보드 입력을 확인할 수 있다는 함수를 몇 가지 더 제공합니다. 아래는 그 함수들을 간략하게 소개한 것입니다.

keybd_is_key_pressed()	키가 방금 눌려졌는지 여부를 리턴
keybd_is_key_released()	키가 방금 떼어졌는지 여부를 리턴
keybd_is_key_down()	키가 눌려있는지 여부를 리턴
keybd_is_key_up()	키가 떼어진 상태인지 여부를 리턴

Jokebox에서는 키보드뿐 아니라 마우스 입력을 확인할 수 있는 함수도 제공합니다. 아래는 마우스 입력 관련 함수들을 요약한 것입니다.

mouse_is_btn_pressed()	버튼이 방금 눌려졌는지 여부를 리턴
mouse_is_btn_released()	버튼이 방금 떼어졌는지 여부를 리턴
mouse_is_btn_down()	버튼이 눌려있는지 여부를 리턴
mouse_is_btn_up()	버튼이 떼어진 상태인지 여부를 리턴

그런데 여기서 한 가지 의문이 들 수 있습니다. 캐릭터 이동 속도를 조절하거나, 애니메이션 속

도를 조절하기 위해선, 현재 시간이 어떻게 되는지 꼭 알아야 할 것입니다. Jokebox에서는 어떤 식으로 현재 시간을 알 수 있을까요? 아-주 간단한 애니메이션을 만들어보는 다음 장에서 이를 살펴보도록 합시다.

간단한 애니메이션 만들기

앞 장에서 언급한 바와 같이, 어찌됐든 현재 시간을 알아야 애니메이션을 만들든, 이동 속도를 조절하든 하는 것이 가능할 것입니다. 이제 캐릭터를 꿈틀거리게 해봄으로써, Jokebox에서 시간을 이용하는 방법을 알아봅시다. 다음과 같은 코드를 **update** 함수에 추가해 봅시다.

```
1         else if (keybd_is_key_pressed(KEY_DOWN))
2                 pos.y++;
3
4         if (gametime.total % 500 > 250)
5                 ch = '0';
6         else
7                 ch = 'o';
8 }
```

[이 소스가 하는 일]

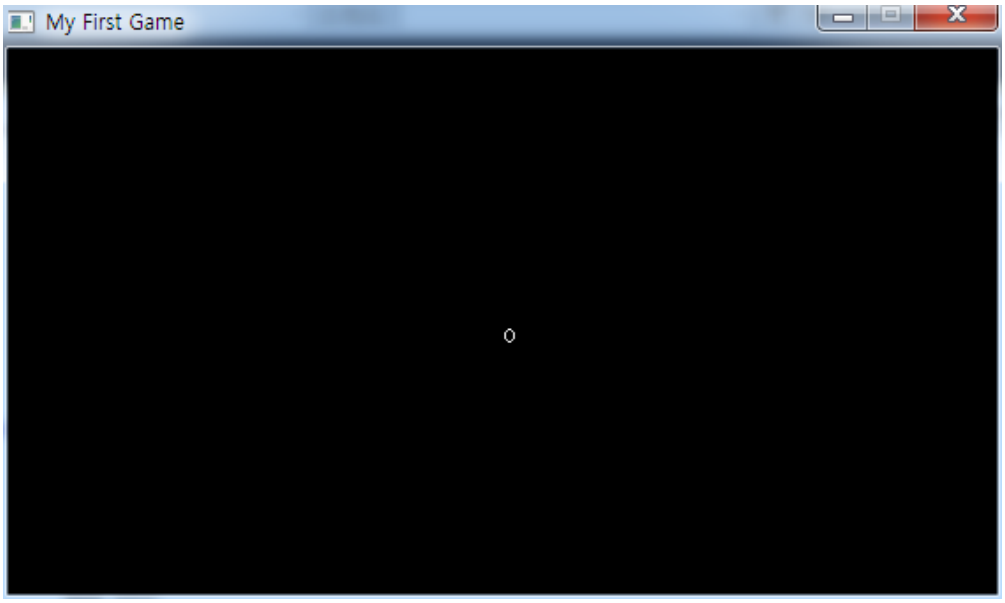
- **gametime을 이용하여 게임에서 흐른 총 시간을 가져옵니다.**

지금 바로 **update**와 **draw** 함수의 인수로 **gametime**이라는 것이 전달되고 있음을 확인해 봅시다. **gametime**은 update, draw 함수가 호출되는 시점에 게임 시간에 관한 정보를 담고 있습니다. 이 인수는 Jokebox에 정의되어 있는 구조체 **GAMETIME** 타입의 인수인데요, 구조체 **GAMETIME**은 멤버 변수로 **total, elapsed**를 갖고 있습니다. **total**은 게임 프로그램이 시작된 후부터 update, draw 함수가 호출되기 직전까지 흐른 시간을 담고 있으며, **elapsed**는 바로 이전에 함수가 호출된 후부터 이번에 호출될 때까지의 시간을 담고 있습니다. 두 시간 모두 밀리세컨드(ms, 1 second = 1000 ms) 단위로 시간을 저장합니다.

- **총 시간에 따라 다른 캐릭터 모양을 지정합니다**

4번째 줄에서는 밀리세컨드로 저장된 게임 총 시간(total)을 500으로 나눈 나머지를 구하는데요, 이 값이 250, 즉 절반보다 크면 캐릭터를 '0'으로 표시하고, 작으면 'o'(알파벳 O의 소문자)로 표시하도록 합니다.

코드를 작성하고 프로그램을 실행시키면 꿈틀대는 캐릭터 모습을 볼 수 있습니다.



이것으로 숫자 0을 움직이는 가장 간단한 게임(?)을 만들어 보았습니다. 동시에 Jokebox의 기본 작동 방법과, 사용할 수 있는 함수들, 구조체들에 대해서도 살펴보았는데요, 이제 이들을 이용하여 진정한 의미의 '게임'을 한번 만들어봅시다 :)