GUI가 존재하지 않는 세상에 게임 올리기

석환

GUI가 없다?

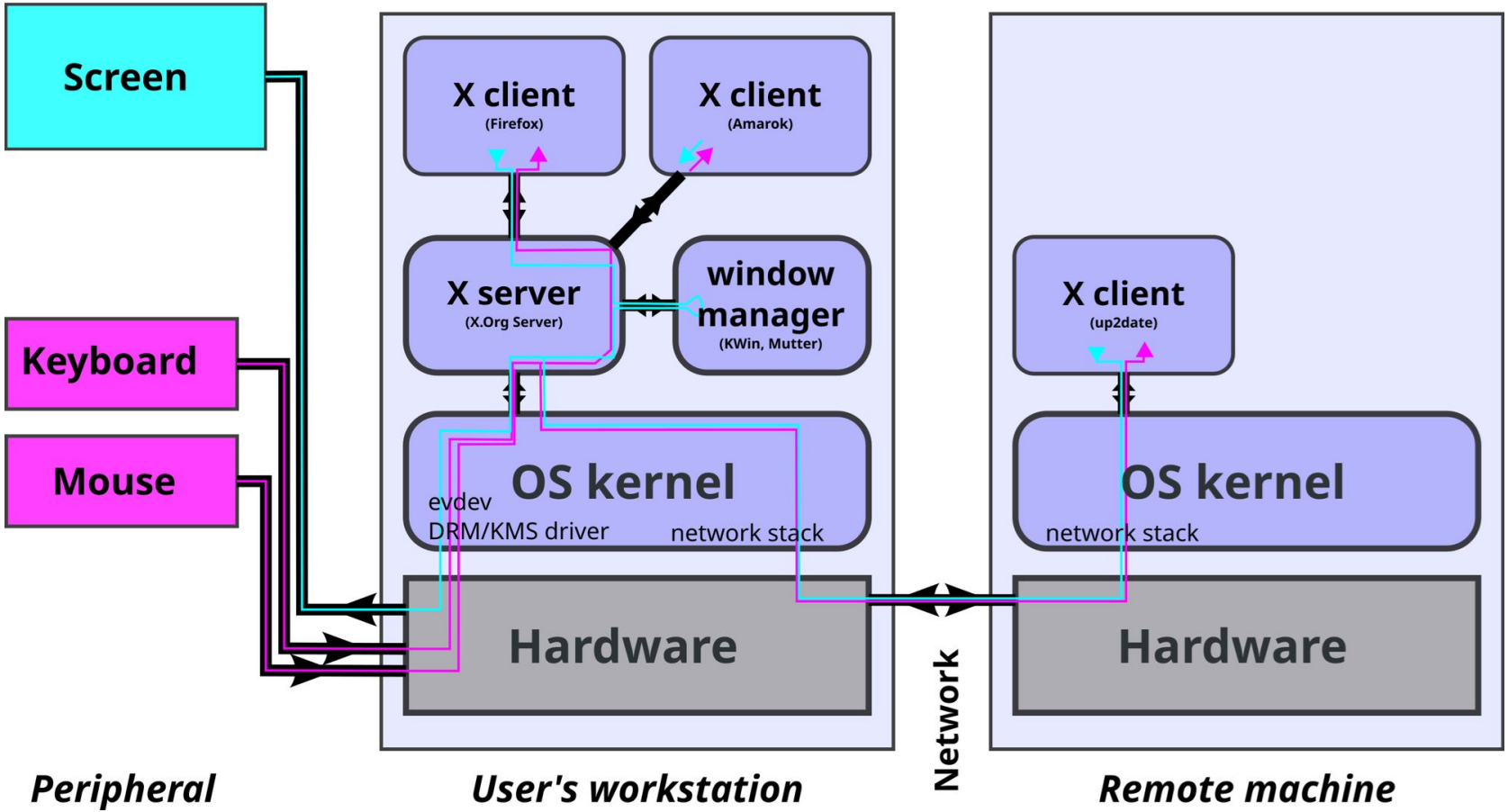
텍스트로 하나?

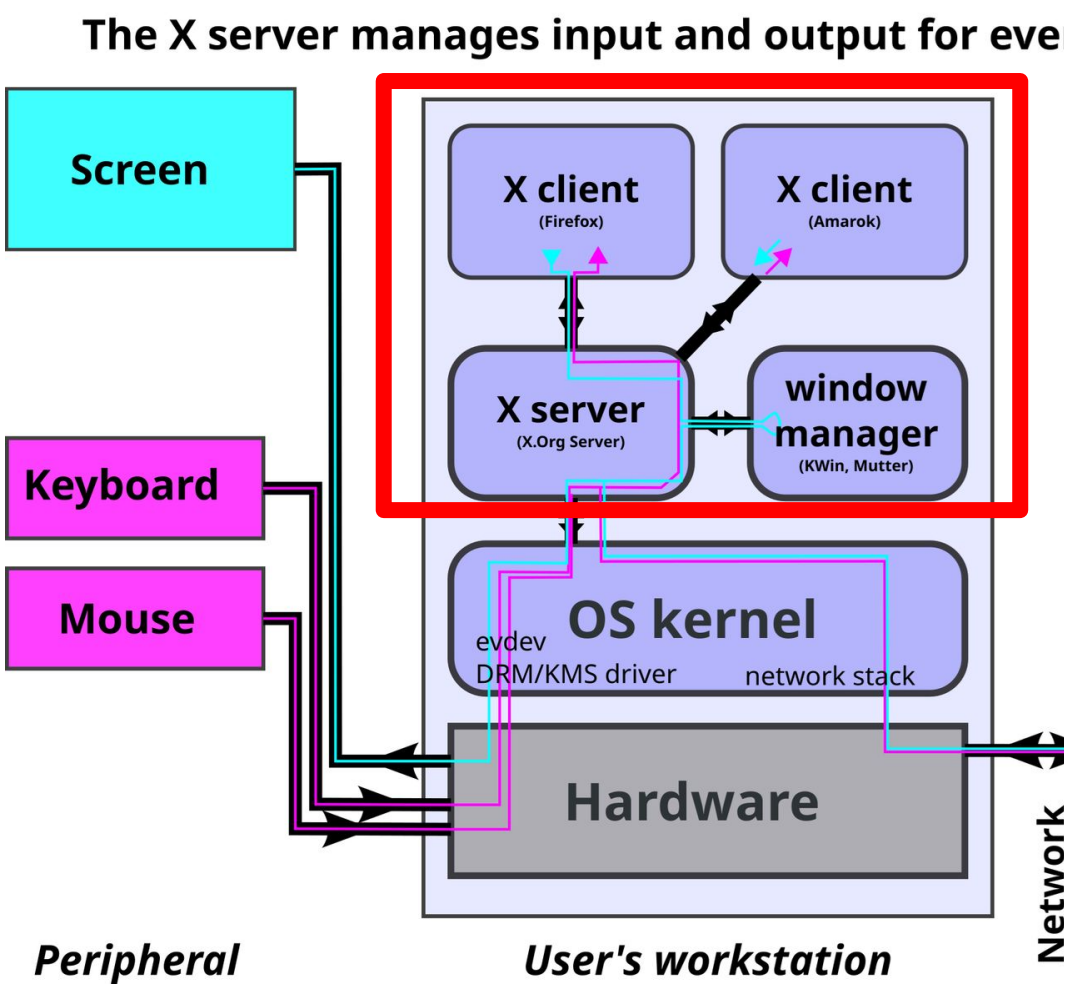
화면 자체가 없나?

윈도우 시스템이 없는세상

윈도우 시스템?

The X server manages input and output for even remote clients:

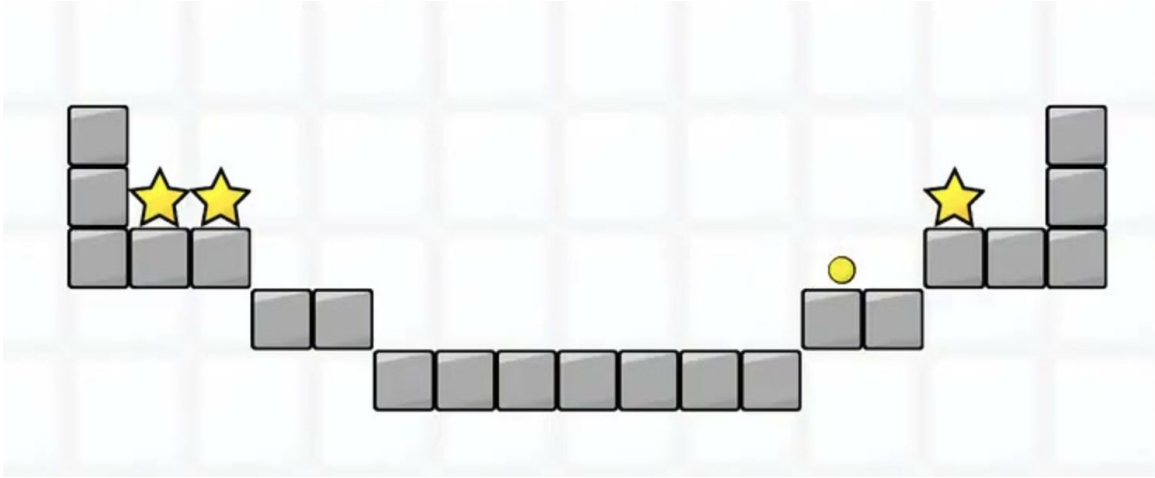




이 아늑한 환경이 없는 세상!

없으면 만들어 나간다.

무엇을 만들어볼까



실행 환경

죽지못해 살아있는 2015년산 산딸기

Broadcom BCM2837B0 SoC

RAM 1GB

프레임 버퍼 사용

```cpp
    int fb_fd = open("/dev/fb0", O_RDWR);
    if (fb_fd == -1) {
        std::cerr << "Error: cannot open framebuffer device." << std::endl;
        return 1;
    }

    // 정보 얻어오기... 중략...

    // 화면 크기 계산
    long screensize = vinfo.yres_virtual * finfo.line_length;

    // 메모리 매핑
    uint8_t* fb_ptr = (uint8_t*)mmap(0, screensize, PROT_READ | PROT_WRITE, MAP_SHARED, fb_fd,
0); if ((intptr_t)fb_ptr == -1) {
        std::cerr << "Error: failed to map framebuffer device to memory." << std::endl;
        close(fb_fd);
        return 1;
    }

    // 픽셀 그리기 (예: 빨간색 사각형 그리기)
    int x, y;
    for (y = 100; y < 200; ++y) {
        for (x = 100; x < 200; ++x) {
            long location = (x + vinfo.xoffset) * (vinfo.bits_per_pixel / 8) +
                            (y + vinfo.yoffset) * finfo.line_length;
            if (vinfo.bits_per_pixel == 32) {
                *(fb_ptr + location) = 255;        // Blue
                *(fb_ptr + location + 1) = 0;      // Green
                *(fb_ptr + location + 2) = 0;      // Red
                *(fb_ptr + location + 3) = 0;      // Transparency
            } else { // Assuming 16bpp
                int b = 10;
                int g = (x - 100) / 6;     // A little green
                int r = 31 - (y - 100) / 16;    // A lot of red
                unsigned short int t = r << 11 | g << 5 | b;
                *((unsigned short int*)(fb_ptr + location)) = t;
            }
        }
    }
```

장치를 열고

장치의 메모리와 포인터를 연결

그리고 메모리 카피

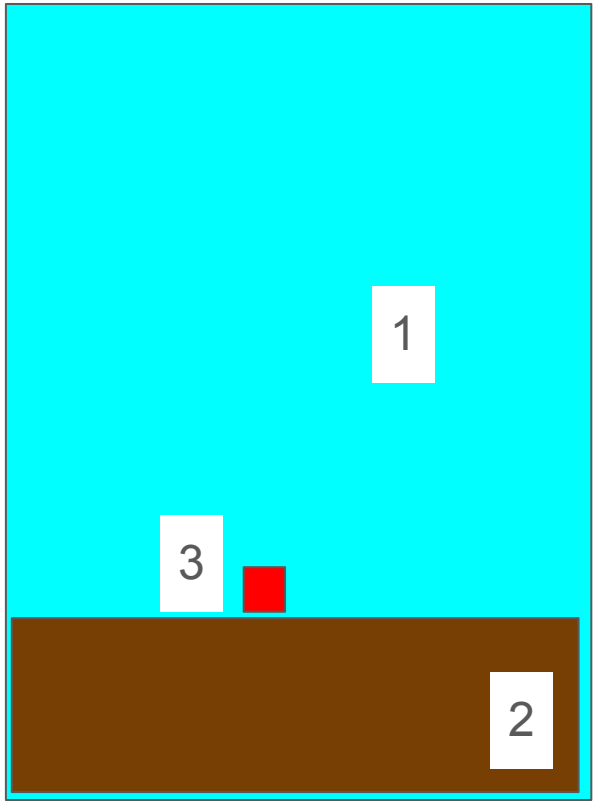
실제 원하는 내용 그리기

```
    uint8_t * buffer_ptr = (uint8_t*)malloc(screensize);

    // 배경 색상 채우기
    fillBackground(buffer_ptr, vinfo, finfo, SKY_BLUE);

    // 땅 색상 채우기
    fillGround(buffer_ptr, vinfo, finfo, BROWN);

    updateScreen(fb_ptr, buffer_ptr, vinfo, finfo);

    // 플레이어 초기화
    Player player(100, HEIGHT - 60);

    // 간단한 이동 예제
    bool movingRight = true;
    for (int i = 0; i < 200; ++i) {
        // 배경 다시 그리기
        fillBackground(buffer_ptr, vinfo, finfo, SKY_BLUE);
        fillGround(buffer_ptr, vinfo, finfo, BROWN);
        updateScreen(fb_ptr, buffer_ptr, vinfo, finfo);

        // 플레이어 이동
        if (movingRight) {
            player.move(2);
            if (player.getX() > WIDTH - 10) movingRight =
false;  } else {
            player.move(-2);
            if (player.getX() < 0) movingRight = true;
        }

        // 플레이어 그리기
        player.draw(fb_ptr, vinfo, finfo);

        // 간단한 지연
        usleep(5000);
    }
```

버퍼링용 추가 버퍼

배경을 그리고

플레이어 그리기



키보드 처리

```
    // 입력 장치 파일 열기
    int keyboard_fd = -1;

    // 키보드 파일 찾기
    for (int eventid = 0; eventid < 32; ++eventid) {
        std::string device = "/dev/input/event" +
          std::to_string(eventid);
        int fd = openInputDevice(device);
        if (fd != -1) {
            keyboard_fd = fd;
            int flags = fcntl(keyboard_fd, F_GETFL,
0);          fcntl(
              keyboard_fd, F_SETFL,
              flags | O_NONBLOCK
            );
            break;
        }
    }

    if (
        keyboard_fd == -1
    ) {
        munmap(fb_ptr, screensize);
        close(fb_fd);
        return 1;
    }

    // 중략...
```

키보드 이름이 무엇으로 나올지
몰라 장치를 하나씩 다 열어보기

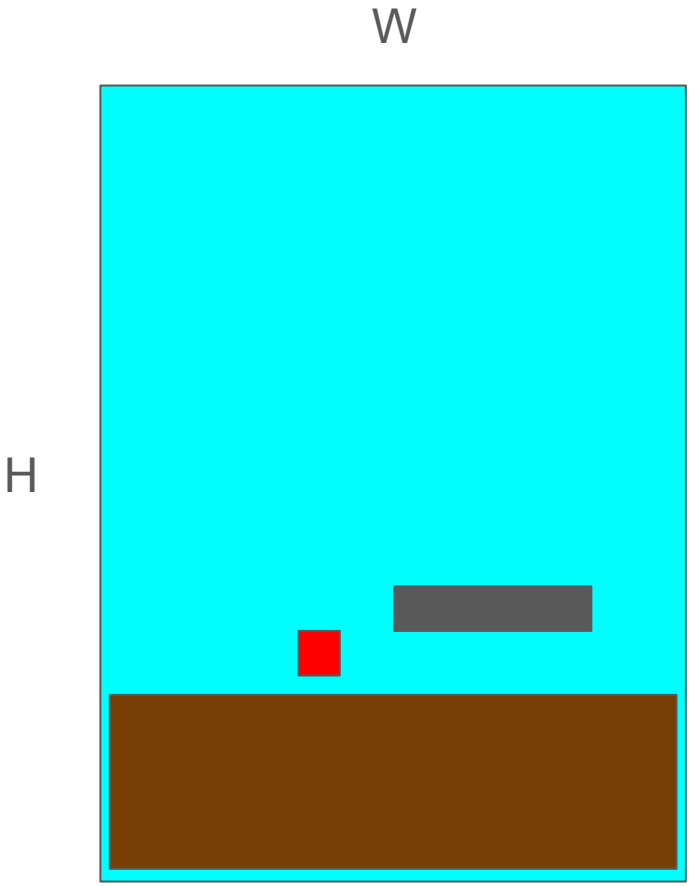
```
    // 중략...

    // pollfd 구조체 설정
    struct pollfd fds;
    fds.fd = keyboard_fd;
    fds.events = POLLIN;

    // 이벤트 루프
    bool running = true;
    while (running) {
        struct input_event ev;

        // poll 함수를 사용하여 키보드 이벤트 폴링
        int ret = poll(&fds, 1, 16); // 16ms 타임아웃 (약 60 FPS)
        if (ret > 0) {
            if (fds.revents & POLLIN) {
                if (read(keyboard_fd, &ev, sizeof(ev)) > 0) {
                    if (ev.type == EV_KEY) {
                        if (ev.value == 1) { // 키가 눌림
                            switch (ev.code) {
                                case KEY_LEFT:
                                    key_left_pressed = true;
                                    break;
                                case KEY_RIGHT:
                                    key_right_pressed = true;
                                    break;
                                case KEY_ESC:
                                    running = false;
                                    break;
                            }
                        } else if (ev.value == 0) { // 키가 떼어
짐                          switch (ev.code) {
                                case KEY_LEFT:
                                    key_left_pressed = false;
                                    break;
                                case KEY_RIGHT:
                                    key_right_pressed = false;
                                    break;
                            }
                        }
                    }
                }
            }
        }
      // 플레이어 이동 로직
```

사용자 입력이 없으면 넘어가야 하기 때문에
poll 로 입력이 있을때만 처리

화면에 무언가를 그리고, 사용자와 상호 작용가능!

최적화

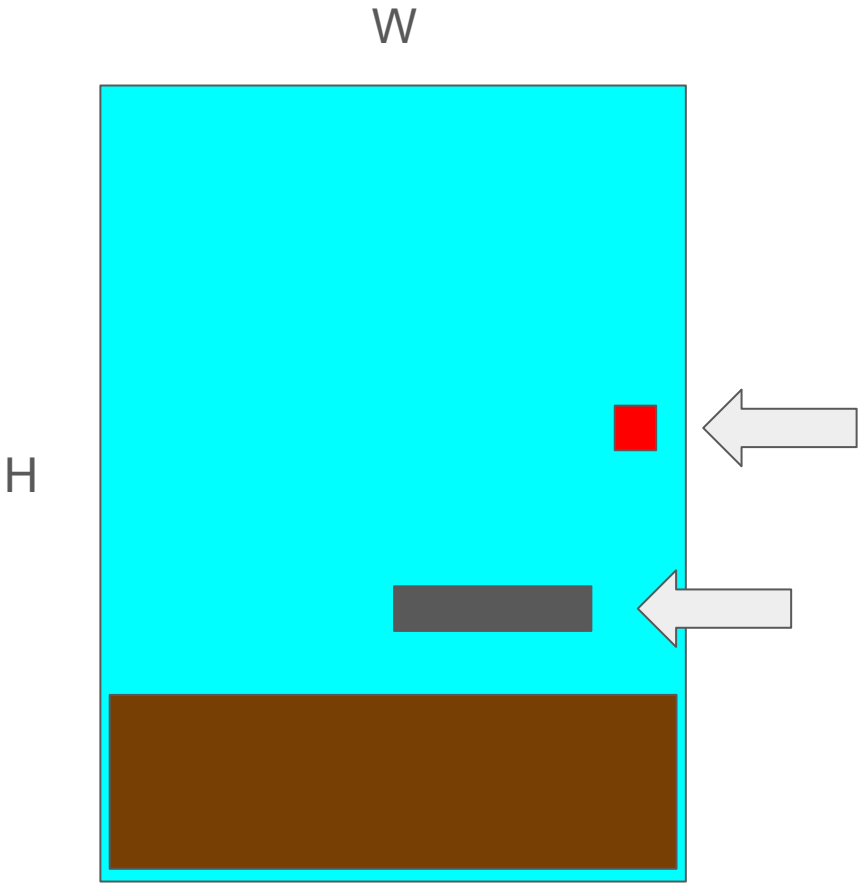
편하게 넘어가지 않는다...



1tick 당 O(W*H*2) 번 연산중

해상도 * (버퍼에 그리기 + 프레임 버퍼에 업데이트)

해상도가 커질수록 속도가 급격히 떨어짐



장애물과 플레이어
둘정도만 그려줘도 이상없음

이미지 처리

공을 표현하기 위한 20*20 짜리 하찮은 BMP파일

```
    Image(const char * imagePath) {
        FILE * bmp24 = fopen(imagePath, "rb");
        if (bmp24 == nullptr) {
            std::cerr << "Error: cannot open image file " << imagePath << "." <<
std::endl;  return;
        }

        uint8_t header[54];
        fread(header, sizeof(uint8_t), 54, bmp24);

        width = *(int*)&header[18];
        height = *(int*)&header[22];
        int size = width * height * 3;

        uint8_t * bmpdata = new uint8_t[size];
        fread(bmpdata, sizeof(uint8_t), size, bmp24);

        // bmp888 to FIXEL_FORMAT
        data = new FIXEL_FORMAT[width * height];
        for (int i = 0; i < width * height; i++) {
            Color color;
            memset(&color, 0, sizeof(Color));
            color.b = bmpdata[i * 3];
            color.g = bmpdata[i * 3 + 1];
            color.r = bmpdata[i * 3 + 2];
            data[i] = convertTo(color);
        }

        delete[] bmpdata;

        fclose(bmp24);
    }
```

Basic BMP File Format

Name	Size	Description
Header	14 bytes	Windows Structure: BITMAPFILEHEADER
Signature	2 bytes	'BM'
FileSize	4 bytes	File size in bytes
reserved	4 bytes	unused (=0)
DataOffset	4 bytes	File offset to Raster Data
InfoHeader	40 bytes	Windows Structure: BITMAPINFOHEADER
Size	4 bytes	Size of InfoHeader =40
Width	4 bytes	Bitmap Width
Height	4 bytes	Bitmap Height
Planes	2 bytes	Number of Planes (=1)
BitCount	2 bytes	Bits per Pixel 1 = monochrome palette. NumColors = 1 4 = 4bit palletized. NumColors = 16 8 = 8bit palletized. NumColors = 256 16 = 16bit RGB. NumColors = 65536 (?) 24 = 24bit RGB. NumColors = 16M
Compression	4 bytes	Type of Compression 0 = BI_RGB no compression 1 = BI_RLE8 8bit RLE encoding 2 = BI_RLE4 4bit RLE encoding
ImageSize	4 bytes	(compressed) Size of Image It is valid to set this =0 if Compression = 0
XpixelsPerM	4 bytes	horizontal resolution: Pixels/meter
YpixelsPerM	4 bytes	vertical resolution: Pixels/meter
ColorsUsed	4 bytes	Number of actually used colors
ColorsImportant	4 bytes	Number of important colors 0 = all
ColorTable	4 * NumColors bytes	present only if Info.BitsPerPixel <= 8 colors should be ordered by importance
Red	1 byte	Red intensity
Green	1 byte	Green intensity
Blue	1 byte	Blue intensity
reserved	1 byte	unused (=0)
repeated NumColors times		
Raster Data	Info.ImageSize bytes	The pixel data

마침 및 주제 선정 이유

사용자가 보는 화면이 만들어지기까지 정말 수많은 조치가 있음을 다들 인지하였으면 하는 바램

오늘의 예시는 단순했지만, 아래 소프트웨어들처럼 거대한 프로젝트들은 무슨일이 벌어질지 생각해본다면 재밌을 것

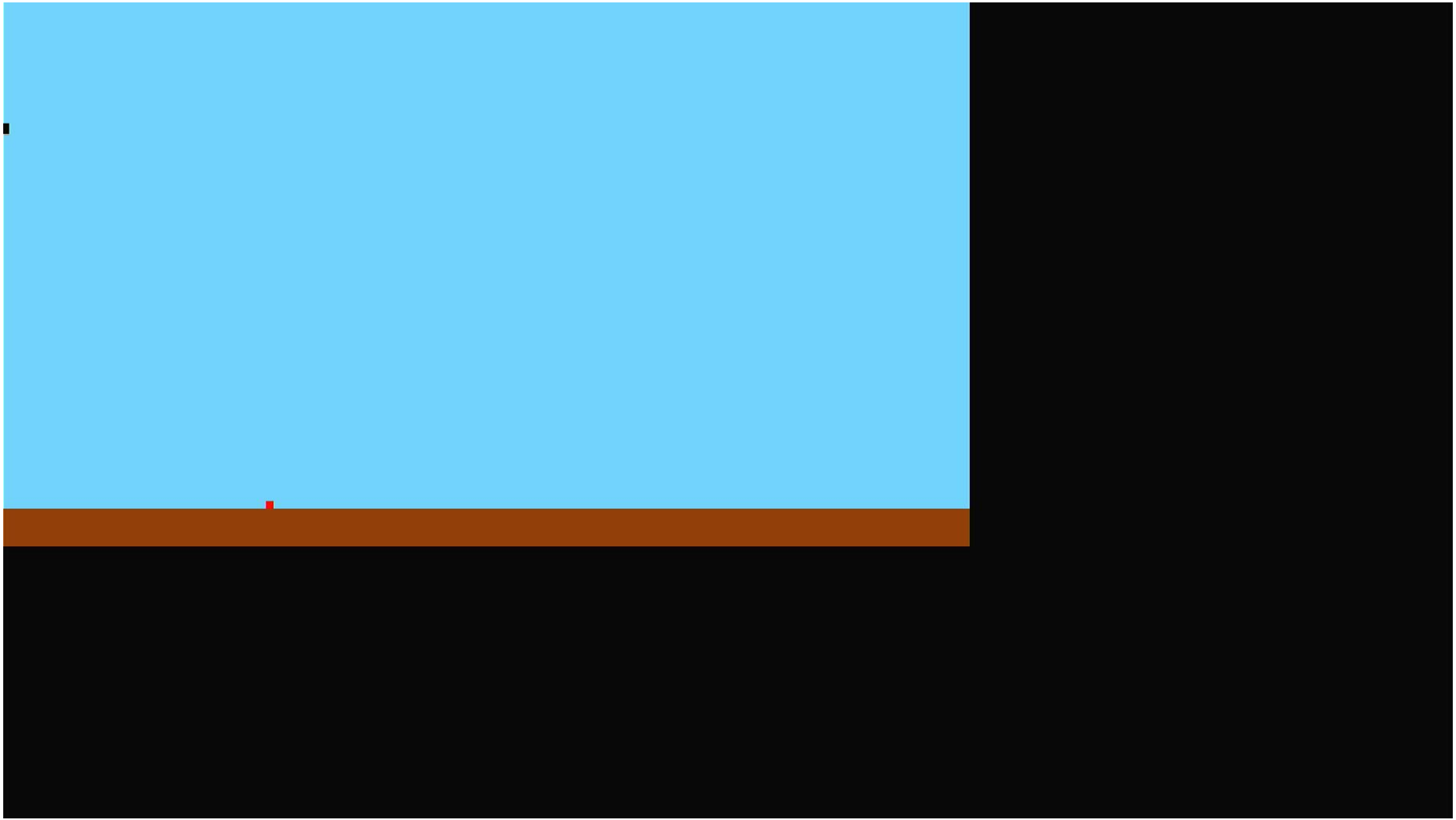
감사합니다.

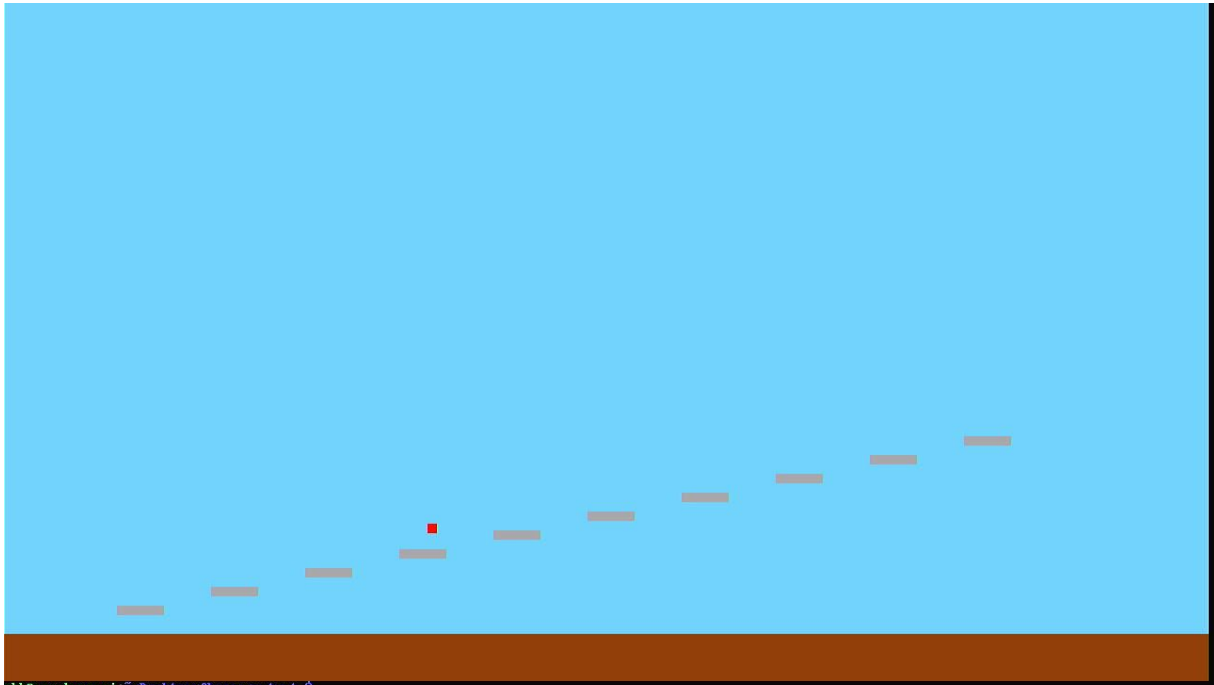
소스코드

https://github.com/bedrock17/fbgame

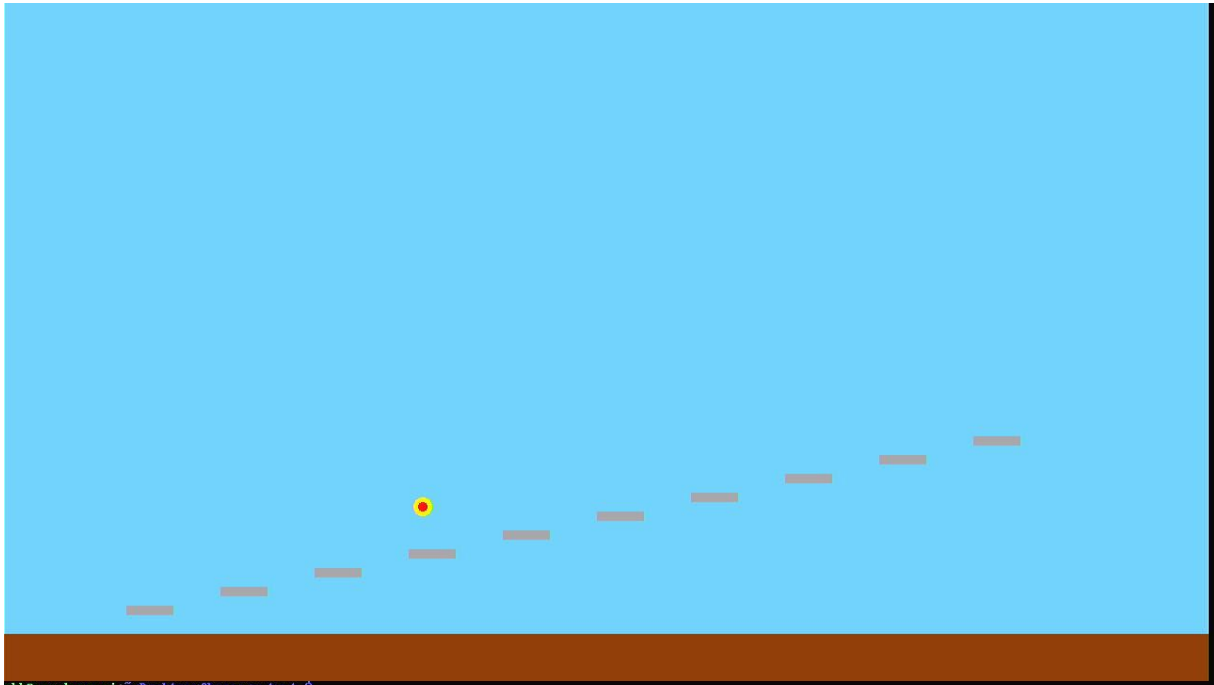
시연 불발 대비용 사진

```
shk@raspberrypi:~/Desktop/fbgame/output $ ./0_draw_rect
shk@raspberrypi:~/Desktop/fbgame/output $
```





```
shk@raspberrypi:~/Desktop/fbgame/output $
shk@raspberrypi:~/Desktop/fbgame/output $
shk@raspberrypi:~/Desktop/fbgame/output $
shk@raspberrypi:~/Desktop/fbgame/output $
shk@raspberrypi:~/Desktop/fbgame/output $
shk@raspberrypi:~/Desktop/fbgame/output $ ./4_performance
disableInputEcho
width = 1920, height = 1080, xres_virtual = 1920, yres_virtual = 1080
screensize = 8294400
bits_per_pixel = 16
openInputDevice: /dev/input/event0
```



```
shk@raspberrypi:~/Desktop/fbgame/output $
shk@raspberrypi:~/Desktop/fbgame/output $
shk@raspberrypi:~/Desktop/fbgame/output $
shk@raspberrypi:~/Desktop/fbgame/output $
shk@raspberrypi:~/Desktop/fbgame/output $
shk@raspberrypi:~/Desktop/fbgame/output $ ./4_performance
disableInputEcho
width = 1920, height = 1080, xres_virtual = 1920, yres_virtual = 1080
screensize = 8294400
bits_per_pixel = 16
openInputDevice: /dev/input/event0

shk@raspberrypi:~/Desktop/fbgame/output $ ./main
width = 1920, height = 1080, xres_virtual = 1920, yres_virtual = 1080
screensize = 8294400
bits_per_pixel = 16
openInputDevice: /dev/input/event0
```