

cub3D



Dosyalar

Header Dosyası

✓ CUB3D

- > .vscode
- ✓ assets
 - > original textures
 - ≡ ea.xpm
 - ≡ no.xpm
 - ≡ so.xpm
 - ≡ we.xpm
 - ✓ checkers
 - C check_characters.c
 - C check_directions.c
 - C check_is_open.c
 - C check_rgb.c
 - C check_wall.c
 - C checkers.c
 - C extention_check.c
 - ✓ free_error
 - C errors.c
 - C free_array.c
 - ✓ init
 - C import_map_file.c
 - C init_direction_vecto...
 - C init_textures.c
 - C setting_ceiling_floo...
 - ✓ lib
 - > gnl
 - > libft
 - > mlx
 - C cub3d.h
 - ✓ loop
 - C camera_orientation.c
 - C key_events.c
 - C press_keys.c
 - C rotate_keys.c
 - ✓ maps
 - ≡ ex.cub
 - ≡ ex2.cub
 - ≡ map.cub
 - ≡ subject_map.cub
 - ≡ test.cub
 - ≡ test1.cub
 - C main.c
 - M Makefile

```
#ifndef CUB3D_H
#define CUB3D_H

#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <math.h>
#include "mlx/mlx.h"
#include "libft/libft.h"
#include "gnl/get_next_line.h"

#define VALID 0
#define EXTENTION_ERROR 1
#define WALLS_MISMATCH 2
#define UNKNOWN_CHARACTER 3
#define DIRECTION_MISMATCH 4
#define EMPTY_MAP 5
#define RGB_MISMATCH 6
#define CANNOT_OPEN 7
#define SURROUND_MISMATCH 8

#ifndef WIDTH
#define WIDTH 1920
#endif
#ifndef HEIGHT
#define HEIGHT 1080
#endif
#ifndef VERTICAL
#define VERTICAL 0
#endif
#ifndef HORIZONTAL
#define HORIZONTAL 1
#endif

typedef struct s_rgb
{
    int r;
    int g;
    int b;
} t_rgb;

typedef struct s_file
{
    int fd_map;
    int fd_east;
    int fd_south;
    int fd_north;
    int fd_west;
    char **map_file;
    char *east;
    char *south;
    char *north;
    char *west;
    char *rgb_f;
    char *rgb_c;
    t_rgb f;
    t_rgb c;
} t_file;

typedef struct s_image
{
    void *image;
    int *address;
    int line;
    int endian;
    int bpp;
} t_image;

typedef struct s_key
{
    int a;
    int s;
    int d;
    int w;
    int left;
    int right;
} t_key;

typedef struct s_cub3d
{
    void *mlx;
    void *mlx_win;
    char **map;
    char start_direction;
    int map_height;
    long rgb_floor;
    long rgb_ceil;
    int t_width;
    int t_height;
    t_ray ray;
    t_key keys;
    t_file *files;
    /* image files */
    t_image screen;
    t_image walls[4]; // respectively: north, south, double
    t_image floor;
    t_image ceiling;
    t_cub3d;
} t_cub3d;

typedef struct s_ray
{
    double pos_x;
    double pos_y;
    double dir_x;
    double dir_y;
    double fov_x;
    double fov_y;
    double camera_x;
    double ray_dir_x;
    double ray_dir_y;
    double delta_dist_x;
    double delta_dist_y;
    double side_dist_x;
    double side_dist_y;
    int step_x;
    int step_y;
    int hit;
    int side;
    int map_x;
    int map_y;
    double perp_wall_dist;
    int line_height;
    int draw_start;
    int draw_end;
    char wall_side;
    char direction;
    double wall_x;
    int tex_x;
    double step;
    double tex_pos;
    double rot_speed;
    double move_speed;
    int tex_y;
    int color;
    t_ray;
} t_ray;

void free_array(char **arr);
void errors(char *str);

/* Checkers */
checkers(t_cub3d *cub);
extension_check(char *map_name);
check_characters(char **str, t_cub3d *cub);
check_directions(t_file *files);
check_rgb(t_file *files);
check_is_open(t_file *files);
check_wall(t_cub3d *cub);

/* Free Functions */
void free_cub(t_cub3d *cub);
void free_files(t_file *files);
void free_array(char **arr);

/* Initializing */
import_map_file(t_cub3d **cub, char *map);
init_textures(t_cub3d *cub, t_file *file, t_image *walls, t_image *);
init_direction_vector(t_cub3d *cub);
setting_ceiling_floor(t_cub3d *cub);
start(t_cub3d *cub, t_file *files);

/* Key */
release_key(int extension_check);
press_key(int extension_check);
key_direction(t_cub3d *cub);
press_w_key(t_cub3d *cub);
press_s_key(t_cub3d *cub);
press_a_key(t_cub3d *cub);
press_d_key(t_cub3d *cub);
press_rot_d_key(t_cub3d *cub);
press_rot_a_key(t_cub3d *cub);
ft_exit(void);

/* Printing Map */
print_map(t_cub3d *cub);
camera_orientation(t_cub3d *cub, t_ray *ray, int x);
set_image(t_cub3d *cub, t_ray *ray);
line_calculator(t_ray *ray);
perform_dda(t_cub3d *cub, t_ray *ray);
side_checker(t_ray *ray);
print_to_screen(t_cub3d *cub, t_ray *ray, int j, int i);
#endif
```

Kısımlar:

1- Uzanti Kontrolü: extension-check

Bu kısımda harita dosyasının uzantisının .cub olup olmadığını kontrol ediyoruz.

Kullanılan fonksiyonlar:

strlen, strcmp

2- Harita dosyasını aktarma:

2. argumanımız olan haritanın dosya yolunu import-map-file fonksiyonuna gönderiyoruz. Ardından satırlarını okuyup struct yapımızdaki mapfile'in içine atıyoruz.

Kullanılan fonksiyonlar: calloc, gnl, strjoin, split

3- Kontroller:

checkers fonksiyonu ile sırayla kontroller yapılıyor.

check-directions: Harita dosyasında yer alan koordinatların dosya yolunu alıp struct'in içine atıyor. Eğer hizini de bulursa başarıyla, eksik bulursa hatayı打印机. 1'den fazla kez yön yazılırsa ikincini doğru kabul ediyor.

Kullanılan fonksiyonlar: strdup, strtrim, strcmp

check_is_open: Yön dosyalarının açılıp açılamadığını kontrol ediyor. Herhangi biri bile açılmazsa hata döndür.

Kullanılan fonksiyonlar: open

check_rgb: Harita dosyasında F ve C ile başlayan satırları arıyor. Bulduğunda da sayılan kontrol ediyor. Ardından kaydedip çıkarıyor. 1'den fazla olduğu durumda ikincini referans alıyor.

Kullanılan fonksiyonlar: strcmp, strtrim, strdup, split, atoi

Check_Characters: Harita dosyası içinde başlayacağımız noktayı arıyor. Eğer harita içinde birden fazla yön işaretçi varsa hata döndür. Eğer O, I, \t dışında bir karakter varsa yine hata döndür.

Kullanılan fonksiyonlar: strtrim, strdup, strcmp, strchr

* Duvar kontrolü öncesi haritayı tekrardan aktarıyoruz.

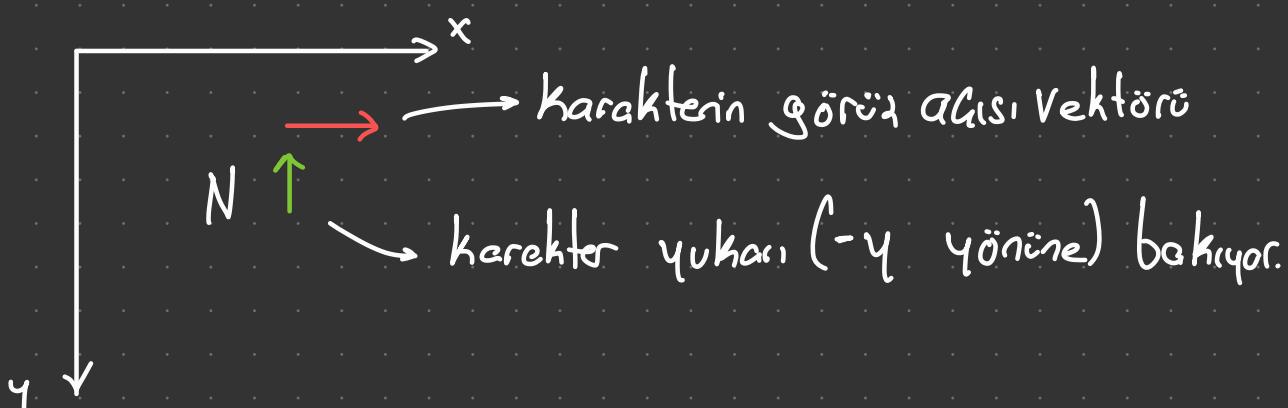
check-wall: Tüm O'ların etrafını kontrol ediyor. Aşik
Varsa hata döndürüyor.

Kullanılan fonksiyonlar: strlen, strchr

Kontrol kısmı bitti. Sıra oyuna başlamada.

Oyun Yükleniyor

Init-direction vector: karakterin yönüne göre karakterinizin 6
vektörünü belirliyor.



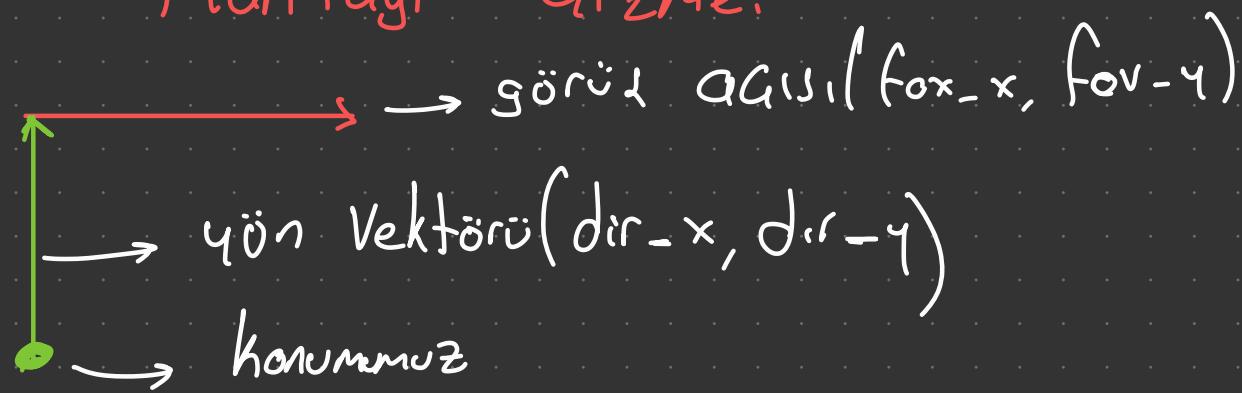
Soru: neden y ekseninin aşağısı + 2 tam tersi olması
gerekniz mi?

Cevap: Buun sebebi haritayı tutan 2 boyutlu char dizisinin
kopası. map[0] haritanın ilk satırını ifade ederken
map[2] ise 2. satırı ifade ediyor. Aşağı indexe ise
sayı artıyor. Diziyi map[y][x] diye düşünün

Soru: Görüş açısının neden sadece yarısı var? Neden bu şekilde değil: \uparrow

Cevap: Hünkhü haritayı çizerek diğer yarısını da eksilisini alarak gizeceğiz.

Haritayı Gizme:

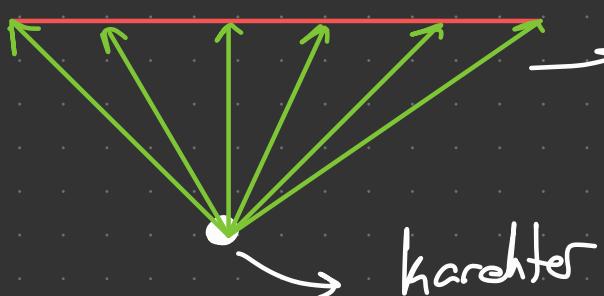


```
void camera_orientation(t_cub3d *cub, t_ray *ray, int x)
{
    ray->camera_x = 2 * x / (double)WIDTH - 1;
    ray->ray_dir_x = ray->dir_x + ray->fov_x * ray->camera_x;
    ray->ray_dir_y = ray->dir_y + ray->fov_y * ray->camera_x;
    ray->hit = 0;
    ray->map_x = (int)ray->pos_x;
    ray->map_y = (int)ray->pos_y;
    ray->delta_dist_x = fabs(1/ray->ray_dir_x);
    ray->delta_dist_y = fabs(1/ray->ray_dir_y);
    side_checker(ray);
    perform_dda(cub, ray);
    line_calculator(ray);
    set_image(cub, ray);
}
```

→ haritayı ekranın en sol hissindən gizmeye başlayacağınız. Camera-x'in değer aralığı [-1, 1]. Bu sayıla görüş açısıyla gorpınca

ışınların her birini elde etmiş oluyoruz

ekranın x eksenİ

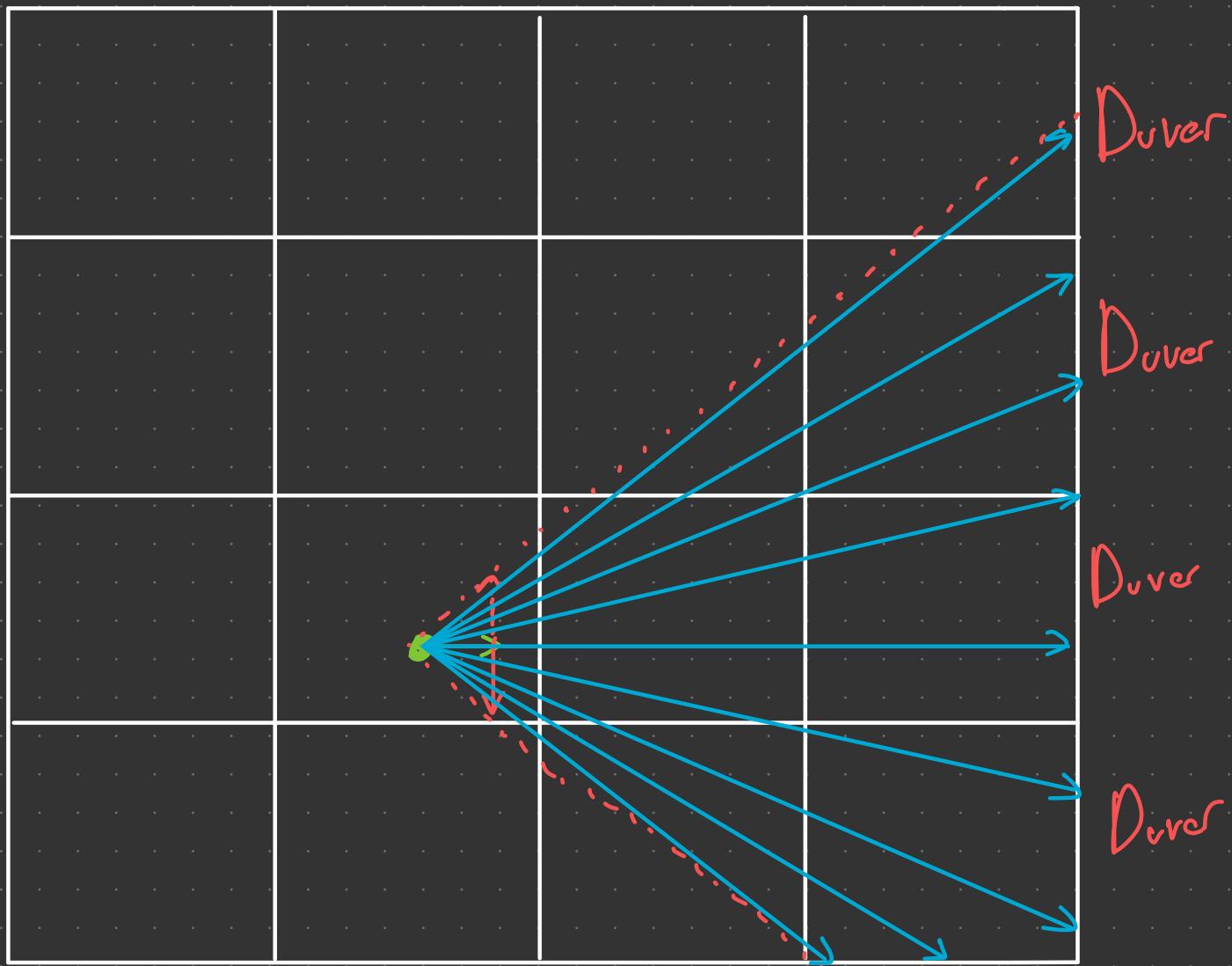


ışınların yönü = dir + fov x camera-x
ray-dir-x & ray-dir-y

hit \rightarrow duvara denk gelince l olacak

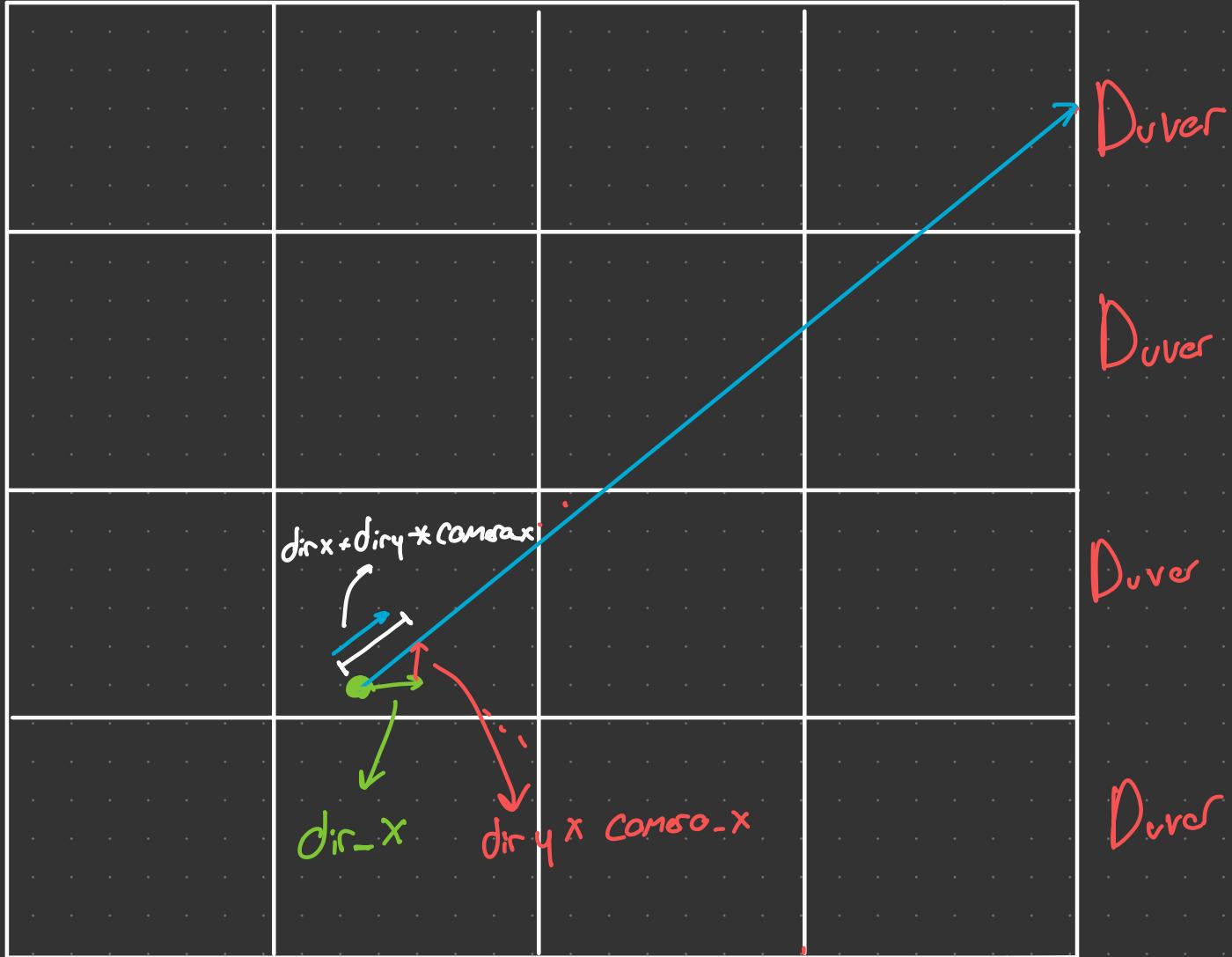
map: x, map-y \rightarrow haritada hangi koerde olduğumuzu tutuyor

Duvar

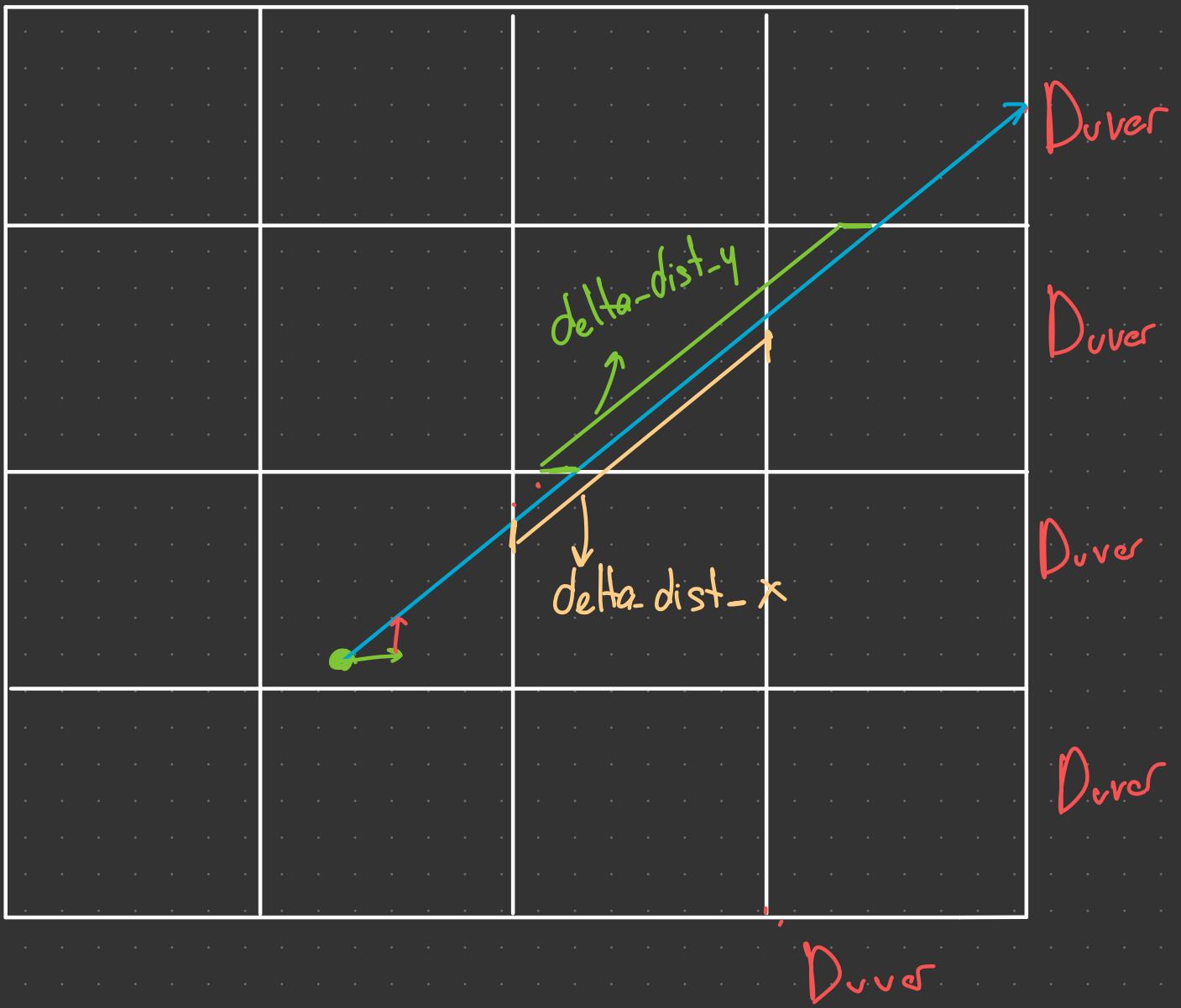


Duvar

Bu şekilde yatay eksendeki her bir piksel için 15'in gönderecektir (1920 hz)



İşlardan birini modelleyelim.



delta-dist_x : 1'sinin x ekseninde 1 birim gitmesi için
ilerlemesi gereken toplam mesafe

delta-dist_y : x 'in y ye göre olan

Üçgen
benzerliği

$$\begin{array}{c} \text{?} \\ | \\ \text{raydir_y} \\ \text{raydir_x} \end{array} \quad \text{delta-dist}_x \Rightarrow \frac{6}{1} = \frac{\text{raydir_y}}{\text{raydir_x}} \quad \Rightarrow ? = \frac{\text{raydir_y}}{\text{raydir_x}}$$

$$\Rightarrow \text{deltadist}_x^2 = 1^2 + 2^2 = 1 + \frac{\text{raydir}_y^2}{\text{raydir}_x^2} = \frac{\text{raydir}_x^2 + \text{raydir}_y^2}{\text{raydir}_x^2}$$

$$\Rightarrow \text{deltadist}_x = \frac{|\text{raydir}|}{\text{raydir}_x}$$

↓

$$\text{raydir}^2 = \text{raydir}_x^2 + \text{raydir}_y^2$$

Deltadist'in uzunluğu, 1'inin büyüklüğünün, 1'inin x eksenine uzaklığının bölümüyle orientili. fakat bir 1'inin büyüklüğünü târacagız. böylece işlem kolaylığı sağlanacak.

Soru: 1'in büyüküğü 1 olmaya bilir.

Cevap: Aynı işlərni 4 için uygularsak: .

$$\text{deltadist}_y = \frac{|\text{raydir}|}{\text{raydir}_y} \text{ gelir. ilerde bize bu iki}$$

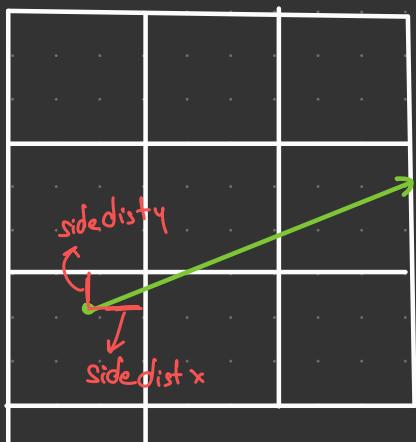
Sayıının oranı lazım, ikişinde de $|\text{raydir}|$ orta olduğundan ikisinin oranı $\frac{\text{raydir}_x}{\text{raydir}_y}$ yapılıyor. Yani $|\text{raydir}|$ sadelesiyor.

Side-checker(ray): Karakterin baktığı yöne göre hesaplamalar yapılır.

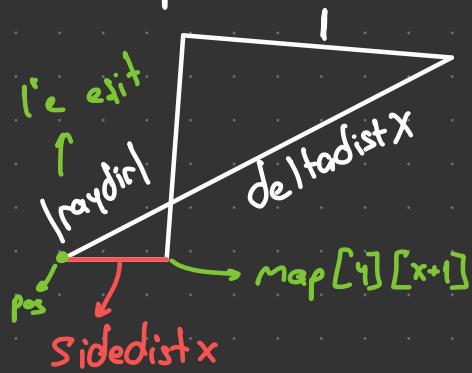


+x ve -y yönüne bakıyor. O zaman
step X = 1 ve step Y = -1 olmalı

sidedist: ilk hesaplanırken karakterimizin baktığı yöndeki en yakın kenara uzaklığını hesaplıyor.



→ hesapları



$$\frac{\text{sidedist } x}{\text{raydir}} = \frac{1}{\text{deltadist } x}$$

⇒ raydir

$$\frac{\text{deltadist}x}{\text{deltadist}y} = \frac{\text{raydir}y}{\text{raydir}x} \Rightarrow |\text{raydir}| \text{ doesn't matter} \rightarrow \frac{1}{\text{raydir}x} \& \frac{1}{\text{raydir}y}$$

```
//which box of the map we're in
int mapX = int(posX);
int mapY = int(posY);

//length of ray from current position to next x or y-side
double sideDistX;
double sideDistY;

//length of ray from one x or y-side to next x or y-side
double deltaDistX = (rayDirX == 0) ? 1e30 : std::abs(1 / rayDirX);
double deltaDistY = (rayDirY == 0) ? 1e30 : std::abs(1 / rayDirY);
double perpWallDist;

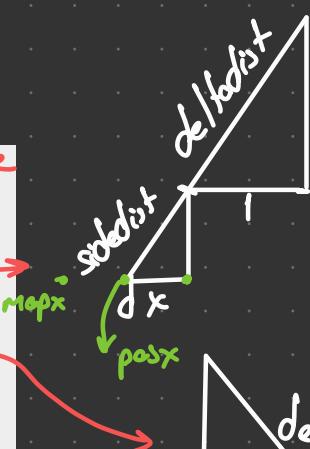
//what direction to step in x or y-direction (either +1 or -1)
int stepX;
int stepY;

int hit = 0; //was there a wall hit?
int side; //was a NS or a EW wall hit?
```

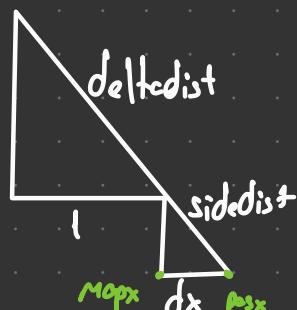
Hangi karede olduğunu gösterir

$\text{if } (\text{in } -x \text{ yönüse } -1; +x \text{ yönüse } +1)$

```
//calculate step and initial sideDist
if (rayDirX < 0) sıfırı sollu ise
{
    stepX = -1; sıfırı sollu ikerle
    sideDistX = (posX - mapX) * deltaDistX;
}
else
{
    stepX = 1;
    sideDistX = (mapX + 1.0 - posX) * deltaDistX;
}
if (rayDirY < 0)
{
    stepY = -1;
    sideDistY = (posY - mapY) * deltaDistY;
}
else
{
    stepY = 1;
    sideDistY = (mapY + 1.0 - posY) * deltaDistY;
}
```



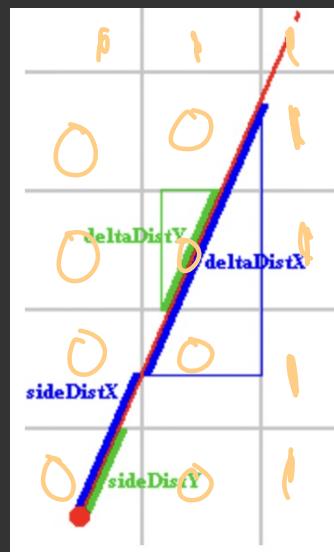
$$\text{sidedist} = d_x = \text{mapx} + 1 - \text{posx}$$



$$\text{sidedist} = \frac{d_x}{1} = \text{posx} - \text{mapx}$$

```
//perform DDA
while (hit == 0)
{
    //jump to next map square, either in x-direction, or in y-direction
    if (sideDistX < sideDistY)
    {
        sideDistX += deltaDistX;
        mapX += stepX;
        side = 0;
    }
    else
    {
        sideDistY += deltaDistY;
        mapY += stepY;
        side = 1;
    }
    //Check if ray has hit a wall
    if (worldMap[mapX][mapY] > 0) hit = 1;
}
```

giderken denk geldiği çizgi dikay ixs
1 kare ± x yönünde git

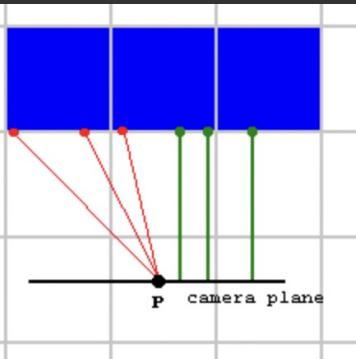


geriye ışının uzunluğunu bulmak kaldı.

İki noktası arası koordinat düzlemini hesabi

$$(x_{san} - x_{ilk})^2 + (y_{san} - y_{ilk})^2 = uzaklik^2 \text{ yaparsak}$$

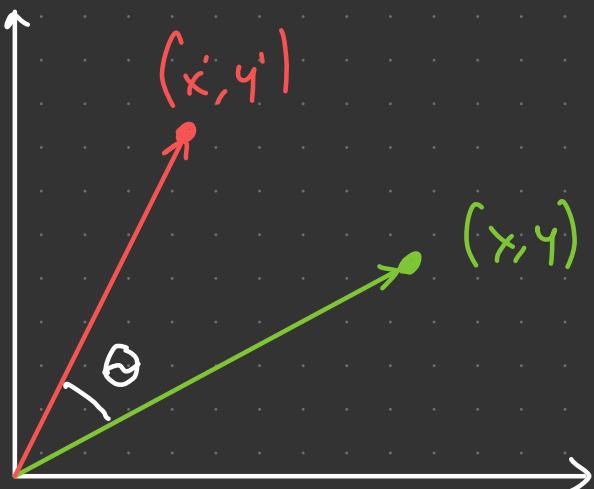
fisheye effect denilen bir görüntü alırız.



* kırmızı yerine yeşillesi
uzaklık olarch alacağız



Rotation Matrix



$$\rightarrow x' = x \cos \theta - y \sin \theta$$
$$y' = x \sin \theta + y \cos \theta$$