مقدمه:

این پروژه با استفاده از کتابخانه‌هایی مانند: OpenGL, Nuklear, GLM, GLFW, GLEW و ... (ما بقیه کتابخانه‌ها یا کم ارزش هستند یا نیازی به آشنایی آن در این PDF نیست) هست.
توی این پروژه هدف ساختن و مقایسه الگوریتم‌هایی برای رسیدن به مقصدی است؛ برای همین شبیه به یک بازی طراحی کردیم که مثلثی به دنبال شما می‌افتد و شما باید از فاصله خود را با آن حفظ کنید و واکنش‌های مثلث را تجزیه و تحلیل.(اگر به شما برسد، اتفاقی صورت نمی‌گیرد.)

خلاصه‌ای از پروژه:

- **نحوه اجرا برنامه و وارد کردن کتابخانه‌ها:**

اسم پروژه «Project-H&R» هست که در آن تمام فرآیند build توسط فایل CMake انجام می‌شود. داخل آن پوشه‌ای به اسم «src»، فایلی هست به اسم «includes.h» که در آن همه کتابخانه‌هایی(ممکن است نیاز شود) قرار داده شده.

- **پنجره:**

در همان پوشه فایلی برای ساخت پنجره به اسم «window.cpp» که همه مشخصات یک پنجره همچون:

۱. اندازه پنجره.
۲. نسخه مورد استفاده API OpenGL (4.6V)
۳. محاسبه FPS.
۴. دکمه خروج از پنجره.
۵. نوع پنجره.

قرار داده شده است.

- **خواندن اطلاعات Shader:**

در فایل «shader.hpp»، نحوه خوانش و اطلاعات رنگی و موقعیت رأس‌ها(Vertices) و تغییرات اندازه و جابه‌جایی صورت می‌گیرد و در قالب یک کلاس به اسم «shader» مورد استفاده قرار می‌گیرد.
فایل «shader.hpp» این گونه اطلاعات را از پوشه‌ای به اسم «shaders» که در آن ۲ فایل به اسم‌های «Mesh-Vert.vs» و «Mesh-Frag.fs» می‌خواند.

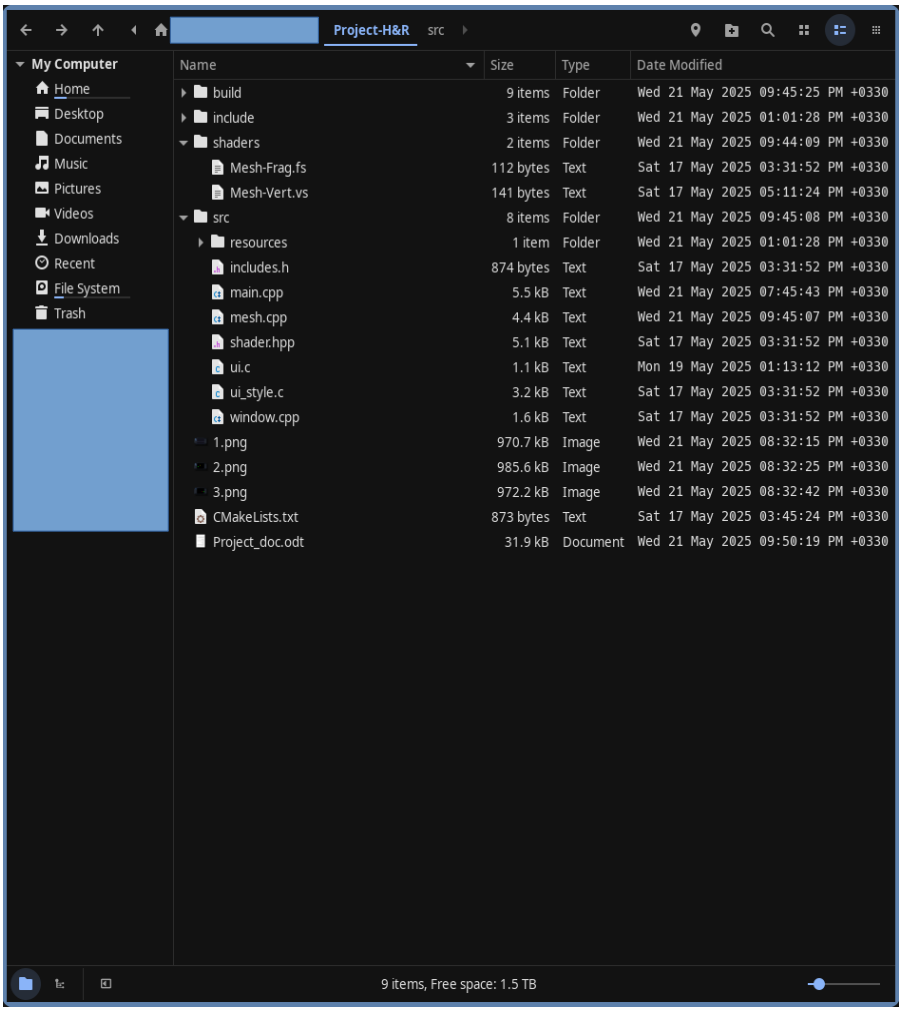
- **ساختن اشکال:**

در فایل «mesh.cpp» ایجاد اشکالی همچون مثلث و مربع صورت می‌گیرد که می‌توان در آن اندازه و موقعیت هر رأس(Vertex) و موقعیت دقیق اشکال و اضافه کردن رنگ به اجسام را مشخص کرد.

- **طراحی منو پروژه:**

در فایلی به اسم «ui.c» طراحی منو پروژه صورت می‌گیرد که تنظیمات ظاهری منو را می‌توان در فایل «ui_style.c» تغییر داد.

سرانجام همه کدها استفاده می‌شود در فایل «main.cpp».



کدهای برنامه:

اگر شما به فایل «main.cpp» نگاه کنید، تعداد زیادی خط کد می‌بینید که در این زمان زیاد اهمیت توضیح ندارند، چون فقط پیاده سازی می‌شوند.
در فایل، در تابع «main» اشکال را می‌سازیم.

```
// Create entities
// -------------------------------------------------------------------------
Mesh yellow_triangle;
yellow_triangle.make_triangle(0.5f, 0.5f);
yellow_triangle.set_color(1.0f, 1.0f, 0.0f, 1.0f);
yellow_triangle.set_position(-0.8f, 0.8f);
yellow_triangle.set_scale(0.2f);
float triangle_scale = 0.2f;

Mesh red_triangle;
red_triangle.make_triangle(0.5f, 0.5f);
red_triangle.set_color(1.0f, 0.0f, 0.0f, 1.0f);
red_triangle.set_position(-0.8f, 0.8f);
red_triangle.set_scale(0.2f);

Mesh green_square;
green_square.make_square(0.5f, 0.5f);
green_square.set_color(0.0f, 1.0f, 0.0f, 1.0f);
green_square.set_position(0.8f, -0.8f);
green_square.set_scale(0.2f);
float square_scale = 0.2f;
// -------------------------------------------------------------------------
```

و بعد متغییرهایی را می‌سازیم که موقعیت و سرعت اشکال را ذخیره کنند.

```
glm::vec2 triangle_pos(-0.8f, 0.8f);
glm::vec2 red_triangle_pos(-0.8f, 0.8f);
glm::vec2 player_pos(0.8f, -0.8f);
GLfloat movement_speed = 0.01f;
GLfloat follow_speed = 0.005f;
```

در آخر وارد حلقه اصلی می‌شویم و در آنجا، منوی پنجره و حرکت‌های مربع و پیدا کردن مسیر مثلث‌ها برای رسیدن به مربع (بر اساس الگوریتم‌های Greedy, Divide & Conquer) را می‌نویسیم.

```
// GUI
if (show_main_ui) {
    create_main_menu_ui(ctx, &show_main_ui, &divide, &greedy);
}
```

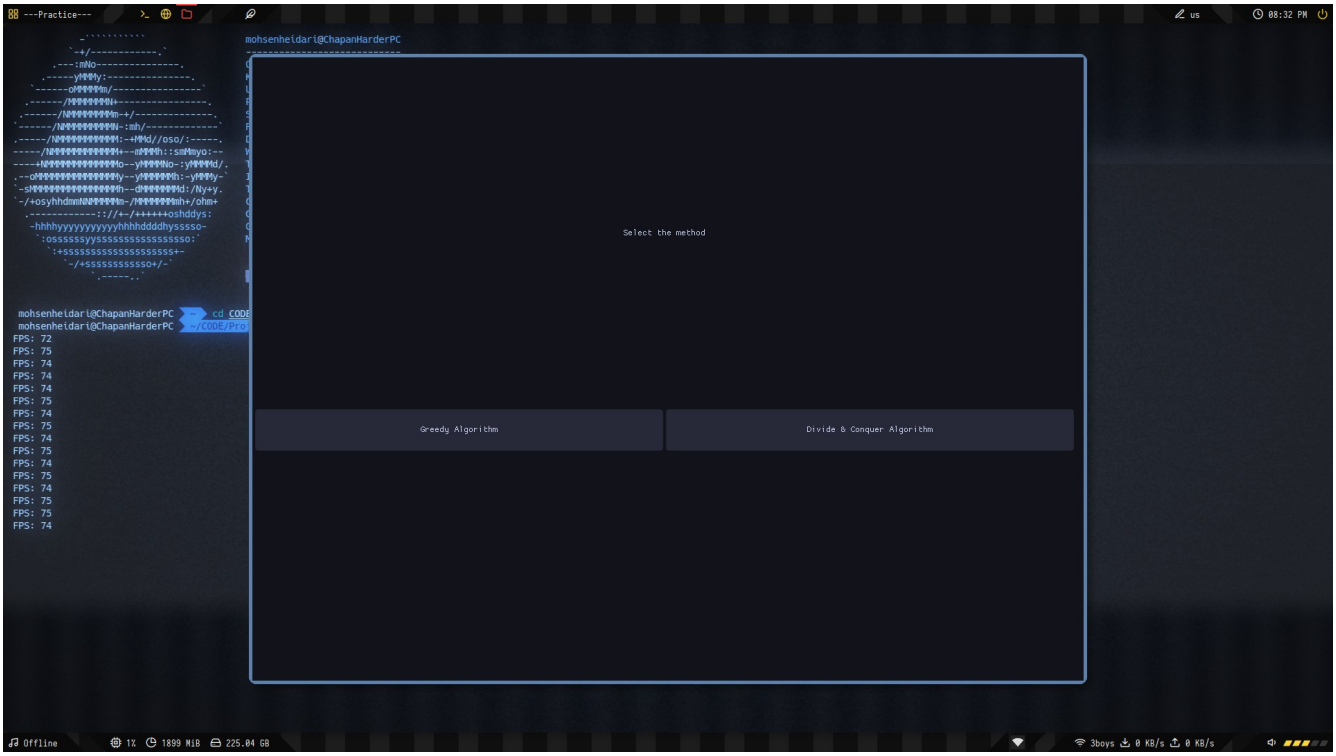
```
    // Divide and conquer approach to make triangle follow square
    // ------------------------------------------------------------------------
    if (divide) {
        glm::vec2 direction = player_pos - triangle_pos;
        float distance = glm::length(direction);

        if (distance > 0.01f) {
            direction = glm::normalize(direction);
            triangle_pos += direction * follow_speed;
            if (glm::length(player_pos - triangle_pos) < follow_speed) {
                  triangle_pos = player_pos;
            }
        }
        clampToBounds(triangle_pos, triangle_scale);
        yellow_triangle.set_position(triangle_pos.x, triangle_pos.y);

        create_back_option(ctx, &show_main_ui, &divide, &greedy);
        green_square.draw_square();
        yellow_triangle.draw_triangle();
    }
    // ------------------------------------------------------------------------

    // greedy approach to make triangle follow square
    // ------------------------------------------------------------------------
    if (greedy) {
            float dx = player_pos.x - red_triangle_pos.x;
            float dy = player_pos.y - red_triangle_pos.y;

            if (std::abs(dx) > std::abs(dy)) {
                red_triangle_pos.x += (dx > 0 ? follow_speed : -follow_speed);
            } else {
                red_triangle_pos.y += (dy > 0 ? follow_speed : -follow_speed);
            }

            if (glm::length(player_pos - red_triangle_pos) < follow_speed) {
                red_triangle_pos = player_pos;
            }

            clampToBounds(red_triangle_pos, triangle_scale);
            red_triangle.set_position(red_triangle_pos.x, red_triangle_pos.y);

            create_back_option(ctx, &show_main_ui, &divide, &greedy);
            green_square.draw_square();
            red_triangle.draw_triangle();
    }
    // ------------------------------------------------------------------------
```

نحوه اجرای برنامه:

برای اجرای برنامه، شما نیاز دارید که وارد پوشه «build» شوید در آنجا Terminal را به همان آدرس اجرا کنید دستور زیر را اجرا کنید.

```
make
./H&R.cm
```

برنامه به این شکل اجرا می‌شود.

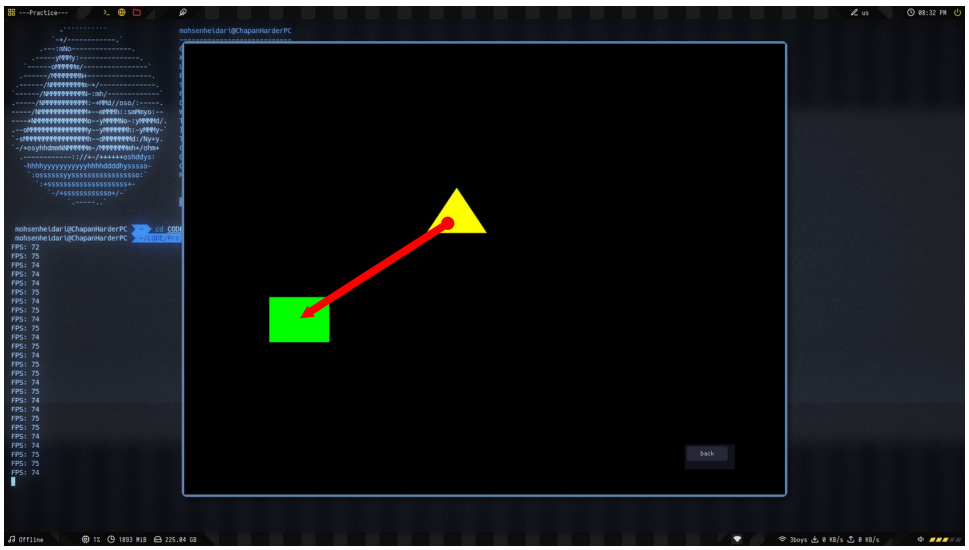


در اینجا ۲تا گزینه وجود دارد.(Greedy, Divide & Conquer)
هر کدام را بزنید، مثلثی شروع به دنبال کردن شما می‌کند ولی تفاوت زیادی وجود دارد که شما هم در اجرای آن متوجه این موضوع می‌شوید.

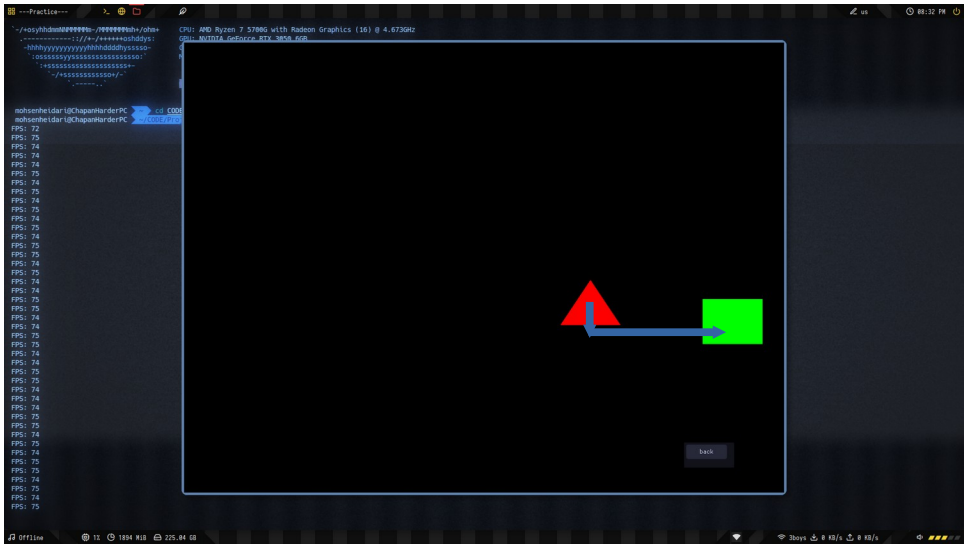
تفاوت‌های کلیدی:

شما بعد از اجرای برنامه، می‌بینید که مثلث زرد که با Divide & Conquer نوشته شده است به مراتب نرم‌تر و متناسب با حرکت‌های مربع، تغییر جهت می‌دهد؛ امّا در مثلث قرمز که با Greedy نوشته شده است این طوری نیست بلکه خیلی شکسته حرکت می‌کند.

- نمایی از Divide & Conquer:



- نمایی از Greedy:



دلایل این عمل چیست؟

۱. راهبردهای حرکت مثلث به سمت مربع:
   آ. کد Divide & Conquer:
   - مثلث در هر فریم‌مستقیماً‌به سمت مربع حرکت می‌کند (با محاسبه جهت**direction**).
   - حرکت‌نرم و مورب‌است (هم‌زمان در محور X و Y).

- سرعت حرکت ثابت (**follow_speed**) ولی جهت‌گیری پویا است.

ب. کد Greedy:

- مثلث‌فقط در یک محور(X یا Y) حرکت می‌کند.
- محور حرکت انتخاب‌شده،تفاضل بزرگ‌تربین مختصات X و Y مربع و مثلث است.
- حرکت‌شبیه به مسیرهای افقی/عمودی‌در بازی‌های دوبعدی قدیمی است (مثل پک‌من)

۲. طبیعت الگوریتم:

آ. کد Divide & Conquer:

- بهینه‌تراز نظر مسافت، چون کوتاه‌ترین مسیر (مورب) را انتخاب می‌کند.
- مناسب برای‌حرکت‌های طبیعی و روان.

ب. کد Greedy:

- ساده‌ترولی ممکن است مسیرهای غیربهینه (طولانی‌تر) ایجاد کند.
- مناسب برای‌حرکت‌های گسسته‌یا شبکه‌ای (مثل بازی‌های نوبتی).

جمع بندی:

معیار:	حریصانه (Greedy)	تقسیم و حل (Divide & Conquer)
نوع حرکت:	افقی/عمودی (گسسته)	مورب (طبیعی)
محاسبات:	سبک (ifساده)	سنگین‌تر (بردارها)
بهینه‌بودن مسیر:	خیر (مسیر طولانی‌تر)	بله (کوتاه‌ترین مسیر)
کاربرد:	بازی‌های قدیمی/ساده	بازی‌های مدرن

اگر به دنبال‌حرکت روان و طبیعی‌هستید،Divide & Conquer بهتر است. اگرسادگی و عملکرد سریع‌مهم است،Greedy گزینه بهتری است.