



STAY
ON
TARGET

STAR WARS

Pouya

Omidi 40212358002

basic programming
fall 1402



Tavakoli 40212358011

Dr. H. Bashiri



نیم نگاهی به پروژه 2 2

STRUCT

MAIN FUNCTION

DEFINECHRT FUNCTION

6 SAVECHART FUNCTION 6

MOVEMENT FUNCTION

ATTACK FUNCTION

9 STAR_QUOTES FUNCTION 9



شوخی ساختی بنام : star wars

در این بازی شما میتوانید یک سفینه را هدایت کنید تا بدون اینکه آسیب ببینه دشمنان کوهکشان را نابود کنه و ناجی دنیا شوید (ز)

در این مینی پروژه از مفاهیم مختلف همچون تابع و استراکت استفاده شده است تا بتوانیم به بهترین شکل ممکن شبیه سازی از بازی داشته باشیم و بهینه ترین برنامه ممکن را بنویسیم.
تابع هایی که استفاده شدند :

- *defineChart*
- *SaveChart*
- *movement*
- *attack*
- *star_quotes*

در ادامه به بررسی تک تک اجزا و سیستم ساخته شده در این نسخه کلون شده از بازی میپردازیم.

همچنین شما میتوانید تمام روند ساخت این برنامه را از لینک زیر مشاهده کنید :



```
struct GameData
{
    int ship_x, ship_y;
    int startX, startY;
    int table1[10][2];
    int Heal = 3;
    int enemy = 10;
};
```

چرا از struct استفاده کردیم ؟!

با توجه به اینکه برنامه نیاز داشت تا اطلاعات کلیدی از تابعی به تابع دیگه انتقال پیدا کنے و حجم این اطلاعات زیاد بود تصمیم گرفتیم از struct استفاده کنیم تا کد ما ساختاری منظم داشته باشد.

از طرفی این اطلاعات وابسته به هم هستند و همگی در روند بازی بصورت مستقیم تاثیرگذارند بنابراین استفاده از این ساختار به ذهنمان رسید .

این ساختار شامل چه اطلاعاتی است ؟



- موقعیت سفینه خودمان.
- موقعیت های سفینه های دشمن.
- موقعیت اولیه سفینه خودمان.
- تعداد جوئی که داریم.
- تعداد دشمنان.

```

int main()
{
    GameData gameData;
    srand(time(0));
    defineChart(gameData);

    gameData.startX = gameData.ship_X;
    gameData.startY = gameData.ship_y;

    while (true)
    {
        system("CLS");
        cout << "          STAR      WARS" << endl;
        // title of the game

        SaveChart(gameData);
        cout << "HEAL : " << gameData.Heal << endl;
        cout <<

"Do you want to Attack or Move? A or M. you can press Q
to quit"
<< endl;
        char attack_move = toupper(getch());

        switch (attack_move)
        {

            case 'A':
                attack(gameData);
                break;

            case 'M':
                movement(gameData);
                break;

            case 'Q':
                return 0;

            default:
                cerr << "invalid input!!!" << endl;
                system("pause");
        }

        if (gameData.Heal == 0)
        {

            system("CLS");
            star_quotes(0);
            break;
        }
        else if (gameData.enemy == 0)
        {
            system("CLS");

            cout << "YOU WON!!!" << endl;
            star_quotes(1);
            break;
        }
    }

    return 0;
}

```

این تابع قلب برنامه ما هست و سایر توابع تحت شرایطی که خواهیم گفت در این قسمت صدا زده میشوند.

در ابتدای این تابع دیتا های بازی که گفته شد انتقال پیدا میکنند به تابع اصلی تا بتوانیم آن را به توابع دیگه انتقال دهیم.

اهم چنین تابعی وجود دارد که فقط یکبار صدا میشود که جلوتر درباره عملکرد آن توضیح خواهیم داد.

در این تابع حلقه while قرار داده شده است که وظیفه دارد تا زمانی که تعداد دشمنان به صفر نرسیده و یا جون ما به صفر نرسیده توابع را براساس ورودی هایی که به آن میدهیم صدا بزند و بازی ادامه پیدا کند.

هدلینگ ورودی ها هم از طریق switch انجام میشود که شما L یا R و یا Q را وارد میکنید که هر کدام توابعی را که در جلوتر توضیح خواهیم داد را صدا میکنند و اگر چیزی جز آنها وارد شود به کاربر میگوید که ورودی آنها تعریف نشده و دوباره در حلقه از کاربر خواسته میشود تا ورودی جدیدی وارد کند.

R

L

Q

در این تابع ابتدا دیتا های ما به ان فرستاده میشوند.

ابتدا داخل حلقه را توضیح میدهیم:

داخل حلقه ابتدا موقعیت های سفینه خودمان بصورت رندوم انتخاب میشه.

سپس در حلقه ای دیگر که درون حلقه قبلی وجود دارد موقعیت های سفینه دشمن بصورت رندوم انتخاب میشود.

در این حلقه ما باید بررسی میکردیم که زمانی که موقعیت های دشمن در حال ساخته شدن هست اول از همه موقعیت تکراری نداشته باشیم و دوم اینکه موقعیت دشمن با سفینه ما یکسان نشود و سوم اینکه در یک ردیف نباید بیشتر از 9 سفینه وجود داشته باشه بنابراین این شرط را در حلقه گذاشتیم تا زمانی که این شرط برقرار نیستند موقعیت های جدیدی برای دشمن های ما پیدا کند.

و در انتهای این تابع حلقه های تو در تویی قرار دادیم تا تعداد کل دشمن هارا بشمارد و اگر این تعداد به 10 نمیرسید حلقه اولیه تکرار شود تا در نهایت ما مطمئن باشیم که 10 دشمن داریم.

```
int defineChart(GameData &gameData)
{
    int enemy_count = 0;

    while (enemy_count != 10)
    {
        // Generates random coordinates for spaceship
        gameData.ship_x = rand() % 10;
        gameData.ship_y = rand() % 10;

        // Set of stars to ensure you have exactly 10 stars
        int starsSet[10][2];

        // Keep track of the number of enemies in each row
        int rowEnemyCount[10] = {0};

        // Generate random coordinates for enemies
        for (int i = 0; i < 10; i++)
        {
            int x, y;

            do
            {
                x = rand() % 10;
                y = rand() % 10;
            } while ((x == gameData.ship_x && y == gameData.ship_y) ||
                     (rowEnemyCount[y] >= 9) ||
                     (i > 0 && (x == starsSet[i - 1][0] && y == starsSet[i - 1][1])));

            starsSet[i][0] = x;
            starsSet[i][1] = y;

            gameData.table1[i][0] = x;
            gameData.table1[i][1] = y;

            rowEnemyCount[y]++;
        }

        for (int i = 0; i < 10; i++)
        {
            for (int k = 0; k < 10; k++)
            {
                // Check if the current position is a star or a spaceship
                for (int l = 0; l < 10; l++)
                {
                    if (l == gameData.table1[i][0] && k == gameData.table1[i][1])
                    {
                        enemy_count++;
                        break;
                    }
                }
            }
        }
    }

    return 0;
}
```



این تابع دقیقاً مثل تابع قبلی دیتا هارا میگیرد.

```
int SaveChart(const GameData &gameData)
{
    for (int i = 0; i <= 10; i++)
    {
        for (int j = 0; j < 10; j++)
            cout << " ---";

        cout << endl;

        if (i <= 9)
        {
            for (int k = 0; k <= 10; k++)
            {
                bool isStar = false;
                bool isSpaceShip = false;
                // Check if the current position is a star or a space
                ship
                for (int l = 0; l < 10; l++)
                {
                    if (i == gameData.table1[1][0] && k == gameData.
                    table1[1][1])
                    {
                        isStar = true;
                        break;
                    }
                    else if (i == gameData.ship_x && k == gameData.
                    ship_y)
                    {
                        isSpaceShip = true;
                        break;
                    }
                }
                if (isStar)
                {
                    cout << "| * ";
                }
                else if (isSpaceShip)
                {
                    cout << "| @ ";
                }
                else
                {
                    cout << "| - ";
                }
            }
            cout << endl;
        }
        return 0;
}
```

اما در این تابع ما تغییراتی بر روی دیتا ها انجام نمیدهیم و این تابع صرفاً از حلقه های متعددی ساخته شده تا مپ بازی را هر زمان که صدا زده شد چاپ کند.

هرچند دو شرط بسیار مهم وجود دارد که آن هم این است که در چاپ کردن مپ متوجه بشود که کجا مپ را سفینه دشمن قرار دهد و کجا سفینه خودمان.

اول از همه در تابع قبلی ما مختصات را با تمام شرایط بدست اوردیم. حال در اینجا همانطور که از کد مشخص است ما می‌گوییم هر موقع که به مختصات مثل دشمن رسیدی مقدار بولیین آن را `true` قرار بده و از حلقه خارج بشو.

سپس در خط های بعدی میبینیم که اگر این مقدار `true` بود سفینه دشمن را چاپ کن.

حال میتوان همین مورد بالا را به تمام مختصات تعمیم داد و درواقع با مفهوم استقرار میتوان به درستی عملکرد آن پی برد.



```

void movement(GameData &gameData)
{
    cout <<
    " Please enter w, a, s, d to move or q to quit " << endl;
    char input;

    input = toupper(getch());
    switch (input)
    {
    case 'W':
        if (gameData.ship_x > 0)
        {
            gameData.ship_x--;
        }
        break;
    case 'A':
        if (gameData.ship_y > 0)
        {
            gameData.ship_y--;
        }
        break;
    case 'S':
        if (gameData.ship_x < 9)
        {
            gameData.ship_x++;
        }
        break;
    case 'D':
        if (gameData.ship_y < 9)
        {
            gameData.ship_y++;
        }
        break;
    case 'Q':
        cout << "Quitting the game." << endl;
        break;
    default:
        cout << "Invalid input!!!";
        break;
    }
    for (int i = 0; i < 10; i++)
    {

        if (gameData.ship_x == gameData.table1[i][0] &&
gameData.ship_y == gameData.table1[i][1])
        {
            gameData.table1[i][0] = -10;
            gameData.table1[i][1] = -10;
            gameData.Heal--;
            gameData.enemy--;
            gameData.ship_x = gameData.startX;
            gameData.ship_y = gameData.startY;
        }
    }
}

```

همانطور که از اسم تابع مشخص است این تابع وظیفه حرکت دادن سفینه ما را بر عهده دارد.

هرچند در این تابع هیچ تابعی تحت این عنوان که مپ را برای ما چاپ کند صدا زده نمیشود و همچنین خود تابع هم این قابلیت را ندارد.

در این تابع ما صرفا مختصات سفینه خودمان و دشمن را تحت شرایطی میتوانیم تغییر دهیم و زمانی که این تغییرات انجام شود با استفاده از تابع در صفحه قبل میتوان آن را به مختصات جدید چاپ کرد.

با استفاده از `switch` ما برای هر کدام از کلید ها تعریف کردیم که چه عملکردی دارند و همچنین با استفاده از شرط این امکان که سفینه از مپ خارج بشود را از آن گرفتیم و کاربر دوباره باید بگوید که چه کاری میخواهد بکند.

اما از طرفی شروط دیگری هم وجود دارد که آن هم این است که اگر سفینه ما با دشمن برخورد کرد که به عبارتی همان برابری مختصاتشان است اتفاقات زیر بیافتد:

- دشمن حذف شود.
- از جون ما کم شود.
- از تعداد دشمن ها کم شود.
- سفینه به مکان اولیه که بازی از آن جا شروع شد باز گردد.

نکته ای که درباره حذف دشمن وجود داره اینه که مختصات آن برابر (10- و 10-) قرار میگیرد که در این صورت هیچگاه جزو تمامی شرط های برنامه قرار نمیگرد و بنابراین چاپ نمیشود.





و اما میرسیم به تابع حمله.
این تابع هم مانند تابع حرکت فقط مختصات را تغییر میدهد و بصورت مستقیم چیزی چاپ نمیکند.

ما امکان حمله به دو سمت چپ و راست را داریم که با استفاده از switch عملکرد هر کدام از حمله ها مشخص شده است.

تنها چالش در این قسمت این است که باید حواسمون میبود که وقتی حمله ای صورت میگیرد اولین دشمن که قبل یا بعد از سفینه خودمون وجود دارد از بین بره.

که الگوریتم آنرا هم با استفاده از حلقه ها و دیتاپی که داشتیم توانستیم بدست بیاریم.

عملکرد حلقه ها به این صورت است که از مختصات خود شروع میکند و یکی یکی به سمت چپ یا راست میروند و اگر دشمن وجود داشت مختصات انرا برابر (10- و 10-) قرار میده تا همونطور که در صفحه قبل توضیح داده شد آن را حذف کند و از طرفی از حلقه خارج میشود تا باقی دشمن هارو که بعد از آن هستند را با یک حمله حذف نکند.

و بدیهی است که از تعداد دشمن های ما کم میشود.



```

void attack(GameData &gameData)
{
    int at_count = 0;

    cout <<
    "where do you want to attack?(you can attack Left with (L)
    and Right with (R)."
    << endl;

    char attack_direction = toupper(getch());
    switch (attack_direction)
    {
        case 'L':
            for (int j = gameData.ship_y - 1; j >= 0; j--)
                // Iterate over columns to the left of the spaceship
                {
                    if (at_count == 1)
                        break;
                    for (int i = 0; i < 10; i++)
                        // Iterate over rows
                        {
                            if (gameData.table1[i][0] == gameData.
                            ship_x && gameData.table1[i][1] == j)
                                {
                                    gameData.table1[i][0] = -10;
                                    gameData.table1[i][1] = -10;
                                    gameData.enemy--;
                                    at_count = 1;
                                    break;
                                }
                        }
                }
            break;

        case 'R':
            for (int j = gameData.ship_y + 1; j < 10; j++)
                // Iterate over columns to the right of the spaceship
                {
                    if (at_count == 1)
                        break;
                    for (int i = 0; i < 10; i++)
                        // Iterate over rows
                        {
                            if (gameData.table1[i][0] == gameData.
                            ship_x && gameData.table1[i][1] == j)
                                {
                                    gameData.table1[i][0] = -10;
                                    gameData.table1[i][1] = -10;
                                    gameData.enemy--;
                                    at_count = 1;
                                    break;
                                }
                        }
                }
            break;

        default:
            cout << "Invalid attack direction!" << endl;
    }
}

```

```

void star_quotes(int qnum)
{
    if (qnum == 1)
    {
        srand(time(0));
        int index = rand() % 5;
        string quotes[5] = {"May the Force be with you.",
        "The Force Is What Gives A Jedi His Power.",
        "Wars not make one great ~ YODA ",
        "Your focus determines your reality", "You were the chosen one"};
        cout << quotes[index];
    }
    if (qnum == 0)
        cout << "just letting you know that YOU ARE A BIG LOSER... " << endl
        << "but let the past die . " << endl
        << "Kill it, if you have to. " << endl
        << "That's the only way to become what you are meant to be.";
}

```

بردن یا باختن

مسئله این است !!!

GAME OVER

این تابعی اضافی است که در انتهای بازی با توجه به اینکه شما بردید یا باختید یک جمله انگیزشی می‌گه !!!

عملکردش هم اینگونه هست که اگر ببری مقدار تابع برابر 1 داده می‌شود که در این صورت بصورت رندوم یکی از جملات به شما نمایش داده می‌شود.

و همچنین اگر بیازی مقدار تابع برابر 0 داده می‌شده که در این صورت برای اینکه امیدتون رو از دست ندید بهتون یادآوری می‌کنه که گذشته دیگه گذشته و نیاز نیست خیلی بهش فکر کنید.

