StarWars

گزارش کار پروژه نهایی

اسفند ماه ۱۴۰۲ – دانشگاه بوعلی سینا

مبانی کامپیوتر و برنامه سازی

استاد : دکتر بشیری

محمد متین سلیمانی

شماره دانشجویی : ۴۰۲۱۲۳۵۸۰۲۱

Git Hub:
https://github.com/M-M-Soleimani/Star-Wars

Gmail: m.m.soleimani84@gmail.com

فهرست :

معرفی بازی :

در این بازی یک سفینه خودی داریم که در ردیف اول از پایین قرار دارد و میتوان به چپ و راست حرکت کند و به سمت بالا گلوله شلیک کند و دارای سلامتی سه می باشد . در این بازی ما چهار نوع سفینه ی دشمن داریم که در ابعاد و سلامتی دارای تفاوت اند.

بازی در دو مود پیشرفته و پایه ای وجود دارد ، که کاربر حق انتخاب بین این دو را داراست.

بازیکن در ابتدای بازی سایز مپ ، حد نصاب امتیاز را وارد می کند در ادامه در صورت رسیدن به حد نصاب امتیاز بازیکن برنده می شود می تواند به صورت بدون محدودیت بازی را ادامه دهد. در صورت از دست دادن تمامی سلامتی ها شما می بازید.

چالش های اجرای پروژه :

چالش ها در سطح پیاده سازی :

۱. طراحی گرافیکی زمین بازی
۲. ماژولاریتی توابع
۳. عدم استفاده از متغیر های گلوبال
۴. بهینه کد زدن
۵. تعریف دو مود برای بازی
۶. حجم بالای کد

چالش های منطقی :

۷. زمان بر بودن حرکات
۸. ساده بودن بازی

توابع موجود در کد :

تابع **main** :

در این تابع ، تابع منو فراخوانی می کنیم.

```
65  int main()
66  {
67      while (Menu())
68      {
69      }
70      return 0;
71  }
```

تابع Menu:

تابعی که مسئول نمایش منو به کاربر جهت تعامل و ادامه ی روند بازی.

در این تابع با استفاده ازswich case یه ورودی از کاربر دریافت و او را به مرحله ی بعدی می برد.

این تابع دارای ۵ وضعیت است :

۱. انتخاب مدل بازی
۲. ادامه ی بازی قبلی
۳. اطلاعات سفینه ها و اجزای بازی
۴. تاریخچه ی بازی ها
۵. خروج

```
bool Menu()
{
    system("cls || clear"); // This function clears the console
    bool Invalid_Selection = false;
    do
    {
        // In the next few lines, the menu will be displayed on the console
        cout << "1- Game Mode " << endl
             << "2- Continue Game " << endl
             << "3- Description of SpaceShips " << endl
             << "4- History" << endl
             << "5- Exit " << endl;
        char user_selection;
        user_selection = getch(); // We get an option chosen by the user
        switch (user_selection)
        {
        case '1':
            if (Game_Mode())    // In this case, the game mode function is called
            {
                return false;
            }
            else
                return true;
            break;

        case '2':
            Continue_Game();    // The continue game function is called
            return true;
            break;

        case '3':
            SpaceShips_Description();   // The function of the description of spaceships is called
            return true;
            break;

        case '4':
            display_History();  // call the history function
            return true;
            break;

        case '5':
            return false;
            break;

        default:
            Invalid_Selection = true;
            system("cls || clear");                         // This function clears the console
            cerr << Red << "Invalid Selection !" << Reset << endl; // In this line, if an invalid choice is made by the user, an error will be displayed on the console
            break;
        }
    } while (Invalid_Selection);
    return false;
}
```

تابع Game_Mode :

تابعی که به کاربر اجازه می ده از بین دو مود بازی یکی را انتخاب کند.

این تابع نیز مانند تابع با از یک swich case تشکیل شده که انتخاب ها به شرح زیر است :

۱. مود پایه ای

۲. مود پیشرفته

۳. خروج

مقدار بازگشتی این تابع درست و نادرست است .

```
bool Game_Mode()
{
    system("cls || clear"); // This function clears the console
    bool Invalid_Selection = false;
    do
    {
        // In the next few lines, the menu will be displayed on the console
        cout << "1- Basic Mode " << endl
             << "2- Advanced Mode " << endl
             << "3- Exit " << endl;
        char user_selection;
        user_selection = getch(); // We get an option chosen by the user
        switch (user_selection)
        {
        case '1':
            Initializer_Basic();    // call the basic initializer function
            return true;
            break;

        case '2':
            Initializer_Advanced();  // // call the basic Advanced function
            return true;
            break;

        case '3':
            return false;
            break;

        default:
            Invalid_Selection = true;
            system("cls || clear");                          // This function clears the console
            cerr << Red << "Invalid Selection !" << Reset << endl; // In this line, if an invalid choice is made by the user, an error will be displayed on the console
            break;
        }
    } while (Invalid_Selection);
    return false;
}
```

تابع Initializer_Basic:

این تابع وظیفه ی آماده سازی برای مود پایه ای را دارد.این تابع با دریافت ابعاد مپ و حد نصاب امتیاز و تعریف متغیر ها بستر را برای فراخوانی تابع Run_game_basic مهیا می کند. این تابع مقدار بازگشتی ندارد.

```
void Initializer_Basic()
{
    system("cls || clear"); // This function clears the console
    bool Invalid_Selection = false;
    int map_size;
    do
    {
        Invalid_Selection = false;
        cout << "Enter the map size : "; // In these few lines, the dimensions of the playground are received from the user
        cin >> map_size;
        if (map_size <= 15)
        {
            Invalid_Selection = true;
            system("cls || clear");  // This function clears the console
            cerr << Red << "Invalid size ! (Map size should be bigger than 15) " << Reset << endl; // In this line, if an invalid choice is made by the user, an error will be displayed on the console
        }
    } while (Invalid_Selection);
    vector<vector<Map_Components>> map;
    map.resize(map_size, vector<Map_Components>(map_size)); // Creates a two-dimensional vector with the dimensions of map size * map size

    system("cls || clear"); // This function clears the console
    cout << "Enter the map size : " << map_size << endl;
    cout << "Enter the quorum for the win : "; // In these few lines, a quorum of points to win is received from the user
    int quorum_point;
    cin >> quorum_point;
    double point = 0;
    int spaceship_health;
    int Spaceship_position;
    int level;
    int Spaceship_type;
    vector<string> Enemies_history;
    Run_game_basic(map, map_size, quorum_point, point, spaceship_health, Spaceship_position, Enemies_history, level, Spaceship_type); // Calling the run game function
}
```

تابع Initializer_Advanced :

این تابع وظیفه ی آماده سازی برای مود پایه ای را دارد.این تابع با دریافت ابعاد مپ ، حد نصاب امتیاز ، تایین نوع سفینه ی خودی و تعریف متغیر ها بستر را برای فراخوانی تابع Run_game_Advanced مهیا می کند.

```
void Initializer_Advanced()
{
    system("cls || clear"); // This function clears the console
    bool Invalid_Selection = false;
    int map_size;
    do
    {
        Invalid_Selection = false;
        cout << "Enter the map size : "; // In these few lines, the dimensions of the playground are received from the user
        cin >> map_size;
        if (map_size <= 15)
        {
            Invalid_Selection = true;
            system("cls || clear");     // This function clears the console
            cerr << Red << "Invalid size ! (Map size should be bigger than 15) " << Reset << endl; // In this line, if an invalid choice is made by the user, an error will be displayed on the console
        }
    } while (Invalid_Selection);
    vector<vector<Map_Components>> map;
    map.resize(map_size, vector<Map_Components>(map_size)); // Creates a two-dimensional vector with the dimensions of map size * map size

    cout << "Enter the quorum for the win : "; // In these few lines, a quorum of points to win is received from the user
    int quorum_point;
    cin >> quorum_point;
    int spaceship_type;
    Invalid_Selection = false;
    do
    {
        string Spaceship_Type_name;
        Invalid_Selection = false;
        // In these few lines, we take the type of spaceship from the user
        cout << "Select the type of spaceship : " << endl
             << "1- spaceship 1" << endl
             << "2- spaceship 2" << endl
             << "3- spaceship 3" << endl;
        spaceship_type = getch() - 48;
        switch (spaceship_type)
        {
        case 1:
            system("cls || clear"); // This function clears the console
            cout << "Select the type of spaceship : spaceship 1 " << endl;
            Spaceship_Type_name = "spaceship 1"; // In this line, we assign the name of a type of spaceship to a string variable to be used in subsequent functions.
            break;

        case 2:
            system("cls || clear"); // This function clears the console
            cout << "Select the type of spaceship : spaceship 2 " << endl;
            Spaceship_Type_name = "spaceship 2"; // In this line, we assign the name of a type of spaceship to a string variable to be used in subsequent functions.
            break;

        case 3:
            system("cls || clear"); // This function clears the console
            cout << "Select the type of spaceship : spaceship 3 " << endl;
            Spaceship_Type_name = "spaceship 3"; // In this line, we assign the name of a type of spaceship to a string variable to be used in subsequent functions.
            break;

        default:
            Invalid_Selection = true;
            system("cls || clear");    // This function clears the console
            cerr << Red << "Invalid Selection !" << Reset << endl; // In this line, if an invalid choice is made by the user, an error will be displayed on the console
            break;
        }
    } while (Invalid_Selection);
    cout << "Enter the map size : " << map_size << endl;
    cout << "Enter the quorum for the win : " << quorum_point << endl;
    double point = 0;
    int spaceship_health;
    int Spaceship_position;
    int level;
    vector<string> Enemies_history;
    Run_game_Advanced(map, map_size, quorum_point, point, spaceship_health, Spaceship_position, Enemies_history, level, spaceship_type); // Calling the run game function
}
```

تابع positioning :

تابعی که با استفاده از تابع srand با استفاده از seed اولیه ی time شروع به تولید اعداد تصادفی برای تولید دشمن ها و مکان یابی آنها انجام می دهد.

تابع display :

تابعی که مسئول نمایش رنگی اجزای مپ می باشد.

```cpp
void display(vector<vector<Map_Components>> map, int map_size)
{
    system("cls || clear"); // This function clears the console
    for (size_t i = 0; i < map_size; i++)
    {
        for (size_t j = 0; j < map_size; j++)
        {
            if (map[i][j].color == "Bright_Green") // In the following conditions, the color for printing the map components is specified and they are displayed in the console
            {
                cout << Bright_Green << map[i][j].character << Reset;
            }
            else if (map[i][j].color == "Bright_Cyan")
            {
                cout << Bright_Cyan << map[i][j].character << Reset;
            }
            else if (map[i][j].color == "Bright_Blue")
            {
                cout << Bright_Blue << map[i][j].character << Reset;
            }
            else if (map[i][j].color == "Blue")
            {
                cout << Blue << map[i][j].character << Reset;
            }
            else if (map[i][j].color == "Magenta")
            {
                cout << Magenta << map[i][j].character << Reset;
            }
            else if (map[i][j].color == "Bright_Red")
            {
                cout << Bright_Red << map[i][j].character << Reset;
            }
            else
            {
                cout << White << map[i][j].character << Reset;
            }
        }
        cout << endl;
    }
}
```

تابع Move_Spaceship :

این تابع ابتدا با پیمایش بر روی سطر آخر سفینه ی کاربر را پیدا کرده و سپس با دریافت ورودی از کاربر سفینه را به چپ و راست حرکت می دهد و همچنین این تابع با فشرده شدن space بازی را متوقف و در صورت درخوتست کاربر آن را save می کند. مقدار بازگردانی شده ی این تابع یک استرینگ است که با استفاده از شروطی با کاربر به تعامل می پردازد.

تابع Move_Enemy_Spaceship :

این تابع با پیمایش مپ تمامی سفینه های دشمن درصورتی که بتوان آنها را جابجا کرد به پایین حرکت می دهد و درصورت اصابت گلوله به آنها سلامتی آنها را کاهش می دهد و یا در صورت برخورد با سفینه کاربر آنها را از بین می برد.

```
string Move_Spaceship(vector<vector<Map_Components>> &map, int map_size, int &Spaceship_position, int quorum_point, double point, int spaceship_health, vector<string> &Enemies_history, int &level, int spaceship_type)
{
    int User_Selection;
    User_Selection = getch();
    int row, colum;
    for (size_t i = 0; i < map_size; i++) // In these few lines we find the position of the spaceship
    {
        if (map[map_size - 1][i].name == "Spaceship")
        {
            row = map_size - 1;
            colum = i;
            Spaceship_position = colum;
            break;
        }
    }
    switch (User_Selection)
    {
    case 100:
        // If we choose the letter D, we move to the right
        if ((colum + 1) < map_size) // This condition checks not to leave the game map
        {
            map[row][colum + 1].name = map[row][colum].name;
            map[row][colum + 1].Health = map[row][colum].Health;
            map[row][colum + 1].damage = map[row][colum].damage;
            map[row][colum + 1].size = map[row][colum].size;
            map[row][colum + 1].color = map[row][colum].color;
            map[row][colum + 1].character = map[row][colum].character;

            map[row][colum].name = "empty";
            map[row][colum].Health = 0;
            map[row][colum].damage = 0;
            map[row][colum].size = 1 * 1;
            map[row][colum].color = "White";
            map[row][colum].character = "[ ]";
        }
        else
        {
            cerr << Red << "Invalid Move !" << Reset << endl; // In this line, if an invalid move is made by the user, an error will be displayed on the console
            Sleep(200);     // This function freezes the console for 200 milliseconds
        }
        return "The move was successful";
        break;

    case 97:
        // If we choose the letter A, we move to the left
        if (colum - 1 >= 0) //// This condition checks not to leave the game map
        {
            map[row][colum - 1].name = map[row][colum].name;
            map[row][colum - 1].Health = map[row][colum].Health;
            map[row][colum - 1].damage = map[row][colum].damage;
            map[row][colum - 1].size = map[row][colum].size;
            map[row][colum - 1].color = map[row][colum].color;
            map[row][colum - 1].character = map[row][colum].character;

            map[row][colum].name = "empty";
            map[row][colum].Health = 0;
            map[row][colum].damage = 0;
            map[row][colum].size = 1 * 1;
            map[row][colum].color = "White";
            map[row][colum].character = "[ ]";
        }
        else
        {
            cerr << Red << "Invalid Move !" << Reset << endl; // In this line, if an invalid move is made by the user, an error will be displayed on the console
            Sleep(200);   // This function freezes the console for 200 milliseconds
        }
        return "The move was successful";
        break;

    case 115:
        // If you press the S key, you will remain without moving
        break;

    case 27:    // By pressing the esc key, the game is saved and the program is closed
        save(map, map_size, quorum_point, point, spaceship_health, Enemies_history, level, spaceship_type);
        Save_History(level, point, spaceship_health, quorum_point);
        return "saved successfully";
        break;

    case 32:
        // In the next few lines, the game will be saved by pressing the space bar, then if you press the S key, the game will be saved and the game will be closed.
        int user_slection;
        do
        {
            system("cls || clear"); // This function clears the console
            cout << Bright_Yellow_new << "Press space to continue the game or press S to save" << Reset << endl;
            user_slection = getch();
            if (user_slection == 115)
            {
                save(map, map_size, quorum_point, point, spaceship_health, Enemies_history, level, spaceship_type);
                Save_History(level, point, spaceship_health, quorum_point);
                return "saved successfully";
                break;
            }
        } while (user_slection != 32 && user_slection != 115);
        return "play";
        break;

    default:
        cerr << Red << "Invalid Selection !" << Reset << endl; // In this line, if an invalid choice is made by the user, an error will be displayed on the console
        Sleep(200);   // This function freezes the console for 200 milliseconds
        break;
    }
    return "The move was successful";
}
```

تابع Shoot :

این تابع ابتدا با پیمایش مپ تمامی گلوله ها را در صورت امکان به بالا حرکت می دهد و سپس یک گلوله ی جدید ایجاد می کند.

```
void Shoot(vector<vector<Map_Components>> &map, int map_size, int &Spaceship_position, int spaceship_type)
{
    for (size_t i = 0; i < map_size; i++) // In the following few lines, if there is a bullet, and if possible, we move the bullet up
    {
        for (size_t j = 0; j < map_size; j++)
        {
            if (map[i][j].name == "bullet")
            {
                if (i > 0 && map[i - 1][j].name == "empty") // If possible, we move the bullet up
                {
                    map[i - 1][j].name = map[i][j].name;
                    map[i - 1][j].Health = map[i][j].Health;
                    map[i - 1][j].damage = map[i][j].damage;
                    map[i - 1][j].size = map[i][j].size;
                    map[i - 1][j].color = map[i][j].color;
                    map[i - 1][j].character = map[i][j].character;

                    map[i][j].name = "empty";
                    map[i][j].Health = 0;
                    map[i][j].damage = 0;
                    map[i][j].size = 1 * 1;
                    map[i][j].color = "White";
                    map[i][j].character = "[ ]";
                }
                else // Otherwise (the bullet is on the edge of the map) we remove it from the map
                {
                    map[i][j].name = "empty";
                    map[i][j].Health = 0;
                    map[i][j].damage = 0;
                    map[i][j].size = 1 * 1;
                    map[i][j].color = "White";
                    map[i][j].character = "[ ]";
                }
            }
        }
    }
    // We create a bullet in a few lines below
    map[map_size - 2][Spaceship_position].name = "bullet";
    map[map_size - 2][Spaceship_position].Health = 0;
    map[map_size - 2][Spaceship_position].damage = spaceship_type;
    map[map_size - 2][Spaceship_position].size = 1;
    map[map_size - 2][Spaceship_position].color = "Bright_Red";
    map[map_size - 2][Spaceship_position].character = "[^]";
}
```

تابع is_dead :

این تابع در مپ پیمایش می کند و در صورتی که عنصری به سلامتی 0 و یا کمتر رسیده باشد آن را پاک می کند.

```
684  void is_dead(vector<vector<Map_Components>> &map, int map_size)
685  {
686      // In the following few lines, we search all the elements of the map and if the health is less than 0, we delete it
687      for (size_t i = 0; i < map_size; i++)
688      {
689          for (size_t j = 0; j < map_size; j++)
690          {
691              if (map[i][j].Health <= 0 && map[i][j].name != "bullet")
692              {
693                  map[i][j].name = "empty";
694                  map[i][j].Health = 0;
695                  map[i][j].damage = 0;
696                  map[i][j].size = 1 * 1;
697                  map[i][j].color = "White";
698                  map[i][j].character = "[ ]";
699              }
700          }
701      }
702  }
```

تابع Show_information :

این تابع اطلاعاتی از قبیل :

۱. سایز مپ

۲. سلامتی سفینه

۳. حد نصاب امتیاز تایین شده توسط کاربر

۴. امتیاز لحظه ای

۵. لول

۶. و دو راهنمایی جهت save و puse بازی

```
704  void Show_information(int map_size, int quorum_point, double point, int spaceship_health, int level)
705  {
706      // In a few lines below we display the information of the game
707      cout << Yellow << "map size : " << map_size << "*" << map_size << Reset << endl;
708      cout << Green << "health : " << Reset;
709      for (size_t i = 0; i < spaceship_health; i++)
710      {
711          cout << Red << "*" << Reset;
712      }
713      cout << endl;
714      cout << Bright_Cyan << "Quorum of points : " << quorum_point << Reset << endl;
715      cout << Bright_Blue << "point : " << point << Reset << endl;
716      cout << Bright_Yellow << "level : " << level << Reset << endl;
717      cout << Bright_Yellow_new << "Select 'ecs' to exit and save" << Reset << endl;
718      cout << Bright_Yellow_new << "Press space to pause the game" << Reset << endl;
719  }
```

تابع save :

این تابع اطلاعات بازی و موقعیت اجزای بازی را در یک فایل به نام “game.txt” ذخیره می کند.

```cpp
void save(vector<vector<Map_Components>> map, int map_size, int quorum_point, double point, int spaceship_health, vector<string> &Enemies_history, int level, int spaceship_type)
{
    ofstream out("game.txt", ios ::out); // We create an output stream with the name "out"
    // In the following few lines, we save the game information in the file
    out << map_size << endl;
    out << quorum_point << endl;
    out << point << endl;
    out << spaceship_health << endl;
    out << level << endl;
    out << spaceship_type << endl;
    out << Enemies_history.size() << endl;
    for (size_t i = 0; i < Enemies_history.size(); i++)
    {
        out << Enemies_history[i] << " ";
    }
    out << endl;
    for (size_t i = 0; i < map_size; i++)
    {
        for (size_t j = 0; j < map_size; j++)
        {
            if (map[i][j].name != "empty")
            {
                out << i << " " << j << " " << map[i][j].name << " " << map[i][j].Health << " " << map[i][j].character << " " << map[i][j].color << " " << map[i][j].damage << " " << map[i][j].size << endl;
            }
        }
    }
    system("cls || clear"); // This function clears the console
    out.close();
}
```

تابع Continue_Game :

در این تابع با استفاده از فایلی که از بازی های قبلی باقی مانده است ، بازی را ادامه می دهیم.

در ابتدا در صورت باز نشدن فایل ارور مناسبی نمایش داده خواهد شد ، در ادامه درصورت صفر بودن سلامتی یا رسیدن به حد امتیاز اجازه ی ادامه بازی را نخواهید داشت زیرا که بازی به اتمام رسیده است.

```
751  void Continue_Game()
752  {
753      vector<vector<Map_Components>> map;
754      int map_size, quorum_point, spaceship_health;
755      int i, j;
756      double point;
757      int level;
758      int Spaceship_position;
759      int spaceship_type;
760      ifstream in; // We create an input stream with the name "in"
761      in.open("game.txt");
762      if (!in.is_open()) // If there is no appropriate error file, it will be displayed
763      {
764          cerr << Red << "Error opening file !" << Reset << endl;
765          Sleep(1000);
766      }
767      else // In the following few lines, we read the game information from the file
768      {
769          in >> map_size >> quorum_point >> point >> spaceship_health;
770          int Enemies_history_size;
771          in >> level;
772          in >> spaceship_type;
773          in >> Enemies_history_size;
774          vector<string> Enemies_history;
775          Enemies_history.resize(Enemies_history_size);
776          for (size_t i = 0; i < Enemies_history_size; i++)
777          {
778              in >> Enemies_history[i];
779          }
780          map.resize(map_size, vector<Map_Components>(map_size)); // Creates a two-dimensional vector with the dimensions of map size * map size
781          while (in >> i >> j >> map[i][j].name >> map[i][j].Health >> map[i][j].character >> map[i][j].color >> map[i][j].damage >> map[i][j].size) // Read them as long as there is information in the file
782          {
783              if (map[i][j].name == "Spaceship")
784              {
785                  Spaceship_position = j;
786              }
787          }
788          if (point >= quorum_point || spaceship_health <= 0)
789          {
790              cerr << Red << "You can not continue the game that has ended !" << Reset << endl;
791              Sleep(1000);
792          }
793          else
794          {
795              Run_game_basic(map, map_size, quorum_point, point, spaceship_health, Spaceship_position, Enemies_history, level, spaceship_type); // Calling the run game function
796          }
797          in.close();
798      }
799  }
```

تابع Run_game_basic :

این تابع مسئول اعمال صحیح منطق بازی و ایجاد گیم پلی بازی در حالت پایه ای است .

```
803 void Run_game_basic(vector<vector<Map_Components>> &map, int &map_size, int &quorum_point, double &point, int &spaceship_health, int &Spaceship_position, vector<string> &Enemies_history, int &level, int spaceship_type)
804 {
805     while (true && !End_Game(quorum_point, point, spaceship_health, Enemies_history, level)) // just for test
806     {
807         if (!Enemy_is_in(map, map_size))
808         {
809             positioning(map, map_size, spaceship_health, Enemies_history); // This function specifies the position of game elements
810         }
811         display(map, map_size);    // This function displays the game map
812         Show_information(map_size, quorum_ int &map_size paceship_health, level);     // call the function that displays game information
813         string temp = Move_Spaceship(map, map_size, Spaceship_position, quorum_point, point, spaceship_health, Enemies_history, level, spaceship_type);
814         if (temp == "saved successfully") // call the Move_Spaceship function
815         {
816             cout << Bright_Green << "Game saved !" << Reset;
817             break;
818         }
819         else if ( temp == "play")
820         {
821             continue;
822         }
823         else
824         Move_Enemy_Spaceship(map, map_size, spaceship_health, point, Enemies_history, spaceship_type); // Calling a function to move enemy spaceships
825         is_dead(map, map_size);       // Calling a function to check health of map map Components
826         Shoot(map, map_size, Spaceship_position, spaceship_type);       // Calling a function to fire bullets
827         is_dead(map, map_size);        // Calling a function to check health of map map Components
828         user_level(point, level);
829         save(map, map_size, quorum_point, point, spaceship_health, Enemies_history, level, spaceship_type); // Calling a function to save game data
830     }
831 }
```

تابع Enemy_is_in :

این تابع چک می کند که آیه سفینه ی دشمن در مپ وجود دارد یا خیر.

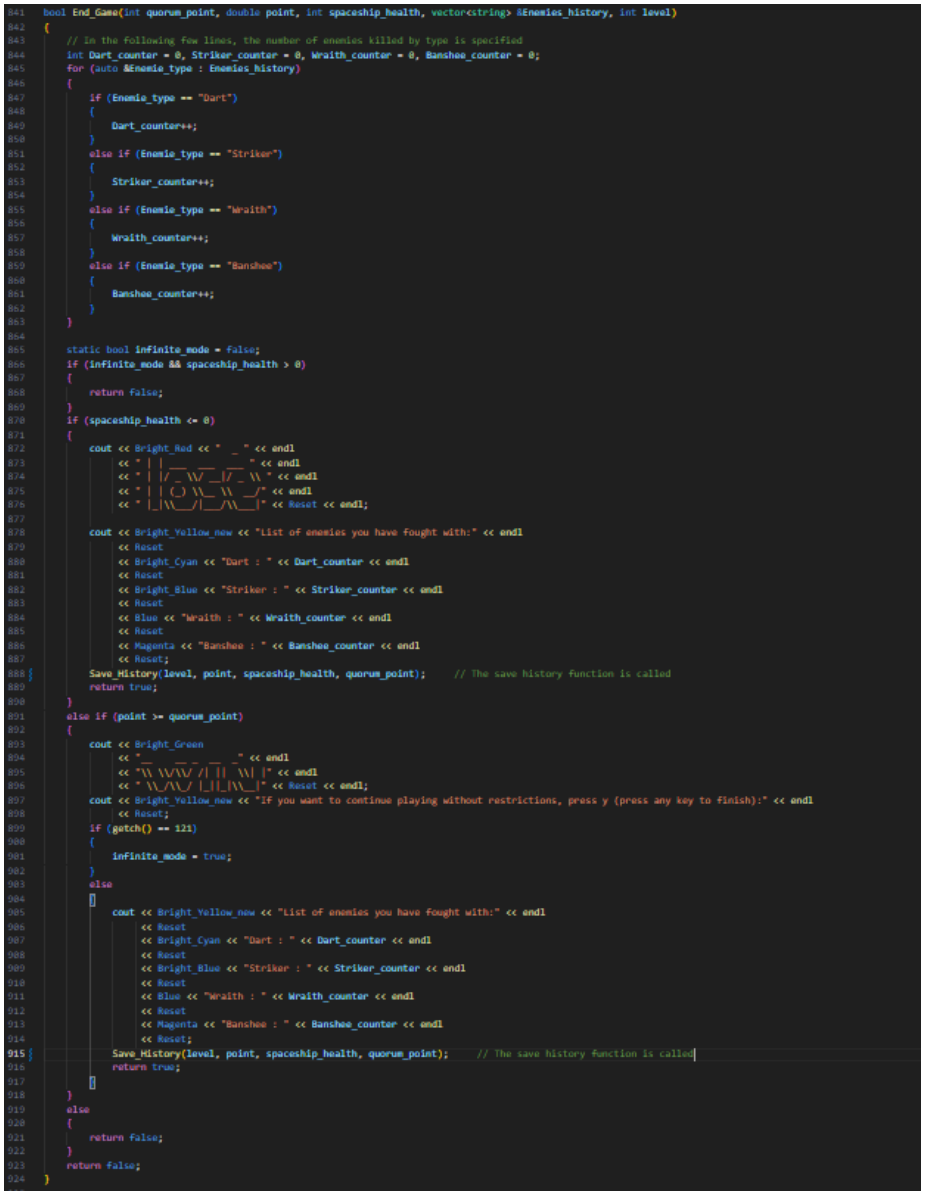
```
> void Run_game_basic(vector<vector<Map_Components>> &map, int &map_size, int &quorum_point, double &point, int &spaceship_health, int &Spaceship_position, vector<string> &Enemies_history, int &level, int spaceship_type)

bool Enemy_is_in(vector<vector<Map_Components>> &map, int &map_size)
{
    // In the following few lines, we navigate the map, if there is an enemy, it returns the true value
    for (size_t i = 0; i < map_size; i++)
    {
        for (size_t j = 0; j < map_size; j++)
        {
            if (map[i][j].name == "Dart" || map[i][j].name == "Striker" || map[i][j].name == "Wraith" || map[i][j].name == "Banshee")
            {
                return true;
            }
        }
    }
    return false;
}
```

تابع End_Game :

تابعی که با استفاده از سلامتی ، حد نصاب امتیاز و امتیاز فعلی وضعیت برد و باخت را مشخص می کند .

درصورت برد به کاربر می گوید که می خواهد به صورت بدون محدودیت به بازی ادامه دهد یا خیر.

و در آخر نتیجه را با استفاده از تابع ذخیره ی تاریخچه ذخیره می کند.



```
bool End_Game(int quorum_point, double point, int spaceship_health, vector<string> &Enemies_history, int level)
{
    // In the following few lines, the number of enemies killed by type is specified
    int Dart_counter = 0, Striker_counter = 0, Wraith_counter = 0, Banshee_counter = 0;
    for (auto &enemie_type : Enemies_history)
    {
        if (Enemie_type == "Dart")
        {
            Dart_counter++;
        }
        else if (Enemie_type == "Striker")
        {
            Striker_counter++;
        }
        else if (Enemie_type == "Wraith")
        {
            Wraith_counter++;
        }
        else if (Enemie_type == "Banshee")
        {
            Banshee_counter++;
        }
    }

    static bool infinite_mode = false;
    if (infinite_mode && spaceship_health > 0)
    {
        return false;
    }
    if (spaceship_health <= 0)
    {
        cout << Bright_Red << "  _ " << endl
             << " | | ___  ___  ___ " << endl
             << " | |/ _ \\/ _|/ _ \\ " << endl
             << " | | (_) \\__ \\  __/" << endl
             << " |_|\\___/|___/\\___|" << Reset << endl;

        cout << Bright_Yellow_new << "List of enemies you have fought with:" << endl
             << Reset
             << Bright_Cyan << "Dart : " << Dart_counter << endl
             << Reset
             << Bright_Blue << "Striker : " << Striker_counter << endl
             << Reset
             << Blue << "Wraith : " << Wraith_counter << endl
             << Reset
             << Magenta << "Banshee : " << Banshee_counter << endl
             << Reset;
        Save_History(level, point, spaceship_health, quorum_point);     // The save history function is called
        return true;
    }
    else if (point >= quorum_point)
    {
        cout << Bright_Green
             << "  ___  ___  ___  ___  _ " << endl
             << "\\ \\/\\/ /| || \\| |" << endl
             << " \\_/\\_/ |_||_|\\__|" << Reset << endl;
        cout << Bright_Yellow_new << "If you want to continue playing without restrictions, press y (press any key to finish):" << endl
             << Reset;
        if (getch() == 121)
        {
            infinite_mode = true;
        }
        else
        {
            cout << Bright_Yellow_new << "List of enemies you have fought with:" << endl
                 << Reset
                 << Bright_Cyan << "Dart : " << Dart_counter << endl
                 << Reset
                 << Bright_Blue << "Striker : " << Striker_counter << endl
                 << Reset
                 << Blue << "Wraith : " << Wraith_counter << endl
                 << Reset
                 << Magenta << "Banshee : " << Banshee_counter << endl
                 << Reset;
            Save_History(level, point, spaceship_health, quorum_point);     // The save history function is called
            return true;
        }
    }
    else
    {
        return false;
    }
    return false;
}
```

تابع user_level :

این تابع با یک عمل ریاضی ساده امتیاز رو به لول تبدیل می کند.

```
926    void user_level(int point, int &level)
927    {
928        level = point / 200;
929    }
```

تابع Run_game_Advanced :

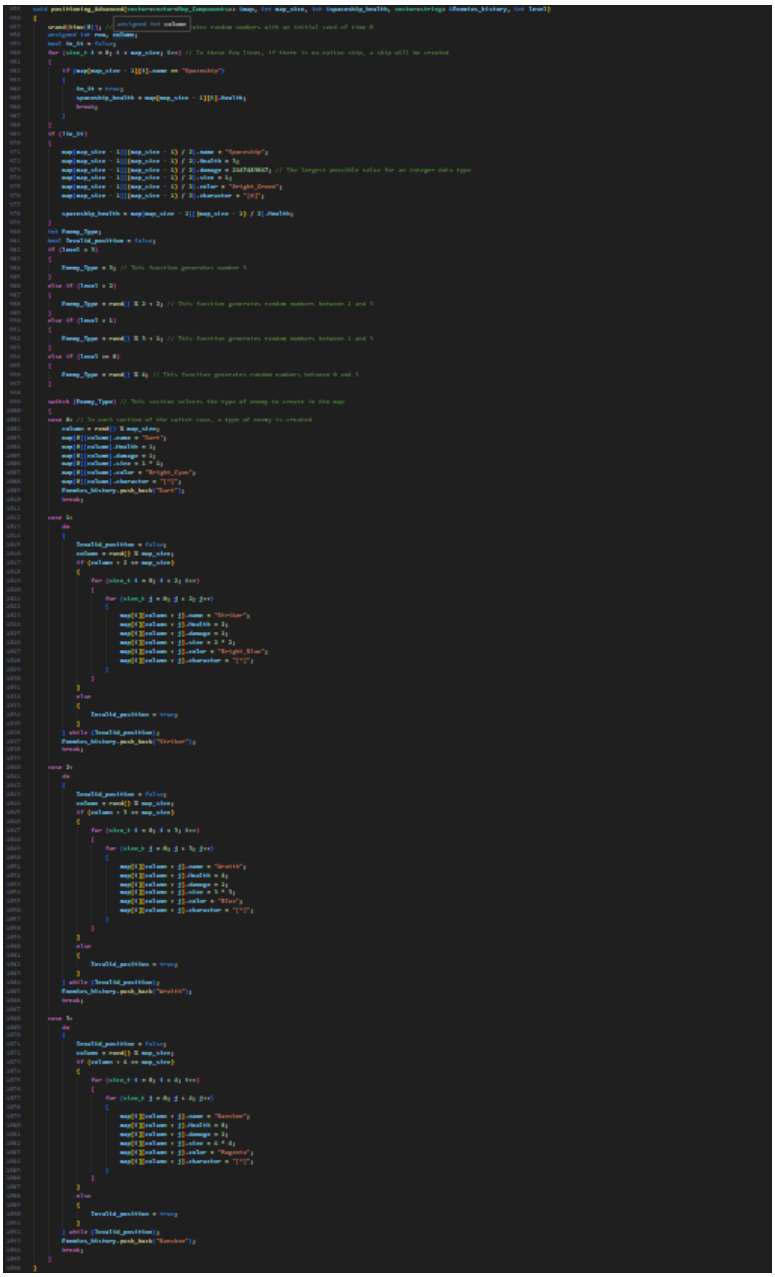
این تابع مسئول اعمال صحیح منطق بازی و ایجاد گیم پلی بازی در حالت پایه ای است.

```
939  void Run_game_Advanced(vector<vector<Map_Components>> &map, int &map_size, int &quorum_point, double &point, int &spaceship_health, int &Spaceship_position, vector<string> &Enemies_history, int &level, int spaceship_type)
940  {
941      while (true && !End_Game(quorum_point, point, spaceship_health, Enemies_history, level)) // just for test
942      {
943          if (!Enemy_is_in(map, map_size))
944          {
945              positioning_Advanced(map, map_size, spaceship_health, Enemies_history, level); // This function specifies the position of game elements
946          }
947          display(map, map_size);     // This function displays the game map
948          Show_information(map_size, quorum_point, point, spaceship_health, level);     // call the function that displays game information
949          string temp = Move_Spaceship(map, map_size, Spaceship_position, quorum_point, point, spaceship_health, Enemies_history, level, spaceship_type);
950          if (temp == "saved successfully") // call the Move_Spaceship function
951          {
952              cout << Bright_Green << "Game saved !" << Reset;
953              break;
954          }
955          else if ( temp == "play")
956          {
957              continue;
958          }
959          else
960          Move_Enemy_Spaceship(map, map_size, spaceship_health, point, Enemies_history, spaceship_type); // Calling a function to move enemy spaceships
961          is_dead(map, map_size);                                                                        // Calling a function to check health of map map Components
962          Shoot(map, map_size, Spaceship_position, spaceship_type);                                      // Calling a function to fire bullets
963          is_dead(map, map_size);                                                                        // Calling a function to check health of map map Components
964          user_level(point, level);
965          save(map, map_size, quorum_point, point, spaceship_health, Enemies_history, level, spaceship_type); // Calling a function to save game data
966      }
967  }
```

تابع positioning_Advanced :

تابعی که با استفاده از تابع srand با استفاده از seed اولیه ی time شروع به تولید اعداد تصادفی برای تولید دشمن ها و مکان یابی آنها انجام می دهد.

ولی تفاوت این تابع با تابع positioning در این است که این تابع با درنظر گرفتن لول دشمن ایجاد می کند.



تابع Save_History :

این تابع اطلاعات هر بازی را به صورت تاریخچه نگهداری می کند.

اطلاعاتی از قبیل :

۱. لول
۲. امتیاز
۳. سلامتی
۴. نتیجه ی بازی

```
1098  void Save_History(int level, double point, int spaceship_health, int quorum_point)
1099  {
1100      ofstream out("History.txt", ios ::out | ios ::app);
1101      out << "level : " << level << " - "
1102          << "point : " << point << " - "
1103          << "health : " << spaceship_health << " - ";
1104      if (point >= quorum_point)
1105      {
1106          out << "win" << endl;
1107      }
1108      else if (spaceship_health <= 0)
1109      {
1110          out << "lose" << endl;
1111      }
1112      else
1113      {
1114          out << "Continue" << endl;
1115      }
1116      out.close();
1117  }
```

تابع display_History :

این تابع مسئول نمایش تاریخچه ی بازی ها است.

```
1119  void display_History()
1120  {
1121      ifstream in;
1122      in.open("History.txt");
1123      string data;
1124      cout << Bright_Yellow_new << "History of games : " << Reset << endl;
1125      while (getline(in, data, '\n'))
1126      {
1127          cout << data << endl;
1128      }
1129      cout << Bright_Yellow_new << "Press one of the keys to return to the menu !" << Reset << endl;
1130      int temp = getch();
1131      in.close();
1132  }
```

تابع **SpaceShips_Description** :

این تابع اطلاعات انواع سفینه ها را نمایش می دهد.

```
1134  void SpaceShips_Description()
1135  {
1136      system("cls || clear"); // This function clears the console
1137      cout << Bright_Green << "SpaceShips : " << endl
1138           << "Health : 3 " << endl
1139           << "damage : infinity " << endl
1140           << "size : 1*1 " << endl
1141           << "character : [#] " << Reset << endl;
1142      cout << "_____________________________________" << endl;
1143      cout << Bright_Cyan << "Dart : " << endl
1144           << "Health : 1 " << endl
1145           << "damage : 1 " << endl
1146           << "size : 1*1 " << endl
1147           << "character : [*] " << Reset << endl;
1148      cout << "_____________________________________" << endl;
1149      cout << Bright_Blue << "Striker : " << endl
1150           << "Health : 2 " << endl
1151           << "damage : 1 " << endl
1152           << "size : 2*2 " << endl
1153           << "character : [*][*] " << endl
1154           << "            [*][*] " << Reset << endl;
1155      cout << "_____________________________________" << endl;
1156      cout << Blue << "Wraith : " << endl
1157           << "Health : 4 " << endl
1158           << "damage : 1 " << endl
1159           << "size : 3*3 " << endl
1160           << "character : [*][*][*] " << endl
1161           << "            [*][*][*] " << endl
1162           << "            [*][*][*] " << Reset << endl;
1163      cout << "_____________________________________" << endl;
1164      cout << Magenta << "Banshee : " << endl
1165           << "Health : 6 " << endl
1166           << "damage : 1 " << endl
1167           << "size : 4*4 " << endl
1168           << "character : [*][*][*][*] " << endl
1169           << "            [*][*][*][*] " << endl
1170           << "            [*][*][*][*] " << endl
1171           << "            [*][*][*][*] " << Reset << endl;
1172      cout << "_____________________________________" << endl;
1173      cout << Bright_Yellow_new << "Press one of the keys to return to the menu !" << Reset << endl;
1174      int temp = getch();
1175  }
```

کتابخانه های مورد استفاده :

کتابخانه ی iostream :

برای خواندن از ورودی و نوشتن در خروجی و به صورت کلی ارتباط با کاربر.

کتابخانه ی time.h و stdlib.h :

برای ایجاد seed برای تابع rand() که اعداد تصادفی تولید کند.

کتابخانه ی windows.h :

برای استفاده ار دستورات سیستمی مانند cls که باعث پاک شدن cmd می شود.

کتابخانه ی conio.h :

برای استفاده از دستوراتی مانند getch و...

در واقع یه هدر غیر از استاندارد اصلی برای دریافت وردی و خروجی در کنسول است.

کتابخانه ی vector :

برای استفاده از وکتور و توابع وابسته به آن.

کتابخانه ی string :

برای استفاده از استرینگ ها و توابع وابسته به آن.

کتابخانه fstream :

برای ایجاد یک جریان ورودی خروجی بر روی فایل است.

کتابخانه ی ncurses :

برای سازگاری با سیستم عامل لینوکس

برآمد های آموزشی :

۱. یادگیری ماژولار نویسی
۲. یادگیری نحوه ی ایجاد سازگاری با سیستم عامل های متفاوت
۳. یادگیری استفاده از توابع رندوم با seed اولیه ی زمانی
٤. استفاده از دستورات سیستمی
٥. یادگیری نحوه ی کامنت گذاری
٦. طراحی محیطی گرافیکی در کنسول
۷. مدیریت استفاده از متغیر های گلوبال
۸. یادگیری تعریف و استفاده از ساختمان های داده
۹. یادگیری فراخوانی توابع با استفاده از پوینتر
۱۰. یادگیری استفاده از گیت هاب
۱۱. یادگیری ایجاد جریان ورودی و خروجی روی فایل ها

سوال در استک اور فلو :

https://stackoverflow.com/questions/78088750/how-to-specify-in-c-to-perform-a-predetermined-task-after-a-certain-time-and-n