ГУАП

КАФЕДРА № 53

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| кандидат технических наук |  |  |  | Кузнецов Виталий Александрович |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ДОПОЛНИТЕЛЬНОМ ЗАДАНИИ |
| №1 ОПТИМАЛЬНЫЙ ПЕРСОНАЖ/КОМАНДА |
|  |
| по курсу: ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 5136 |  | , |  | Д. Л. Мягков |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

Оглавление:

[Исходные данные задачи 3](#__RefHeading___Toc11772_844291809)

[Задача 4](#__RefHeading___Toc11774_844291809)

[Описание реализованных функций 5](#__RefHeading___Toc11776_844291809)

[Листинг программы. 11](#__RefHeading___Toc11778_844291809)

[Вывод. 12](#__RefHeading___Toc11780_844291809)

# Исходные данные задачи

Исходные данные задачи:

1. Персонаж обладает несколькими характеристиками:

− Количество наносимого им урона противнику в секунду. Персонаж наносит урон все время, кроме времени, на которое он выведен из строя. Если персонаж наносит урон нескольким соперникам, указанный урон наносится каждому.

− Количество соперников, которым наносится урон. Не более 3.

− Здоровье персонажа. Наносимый урон вычитается из текущего здоровья, при достижении значения 0, персонаж выбывает.

− Способность выводить из строя одного соперника. Задается двумя значениями в секундах: первое - на которое соперник выводится из строя, второе – интервал вывода из строя. Второе как минимум в двое больше первого. Персонаж выводит из строя, как только удается это сделать.

2. Персонажи могут сражаться один против одного, и в командах по 3 персонажа. Победой команды считается, когда все персонажи противоположной команды выбыли. Каждого персонажа можно выбирать только один раз (не зависимо от того, в какой команде уже выбран этот персонаж).

# Задача

А. Оптимальный выбор 1х1.

Дан набор персонажей с произвольно заданными характеристиками, не менее 10 персонажей. Соперник выбрал одного персонажа. Выбрать среди оставшихся персонажей тех, кто побеждает персонажа соперника с указанием времени сражения и процента оставшегося здоровья. В случае если таких персонажей несколько, то ранжировать их по: времени сражения или проценту (от первоначального) оставшегося здоровья. Вариант ранжирования (сортировки задается) как входной параметр.

В. Оптимальная команда.

Дан набор персонажей с произвольно заданными характеристиками, не менее 10 персонажей. Соперник выбрал трех персонажей. Собрать команду из трех персонажей среди оставшихся таких, которые побеждают команду соперника с указанием времени сражения. В случае если вариантов таких команд несколько, то ранжировать их по времени сражения. Считать, что персонажи наносят урон и выводят из строя в первую очередь соперника с наибольшим уроном.

# Описание реализованных функций

Функции для обеих задач:

1. input\_count\_people()
   1. unsigned short input\_count\_people()
   2. {
   4. short n = 10, n\_max = 10;
   5. cout << "Введите количество человек, не менее 10 и не более 20: ";
   6. while (true)
   7. {
   8. n = input\_value(rand() % n\_max + 10);
   9. if (n >= 10&& n <= 20)
   10. break;
   11. else
   12. cout << "Количество человек не может быть < 10 и > 20, \nвведите другое число: ";
   13. }
   14. erase\_past\_output({0,0});
   15. cout << "Количество человек =" << setw(count\_char(n\_max)) << n << endl;
   16. return n;
   17. }

Запускает бесконечный цикл пока пользователь не введет верное количество людей или пробел

1. input\_value(rand() % n\_max + 10)
   1. template <typename T>
   2. T input\_value(T a)
   3. {
   4. string str;
   5. getline(cin, str);
   6. if (str[0] == '\0')
   7. return a;
   8. return get\_number\_input\_after<T>(str);
   9. }
   10. Функция, которая принимает число, которое она вернет если пользователь ничего не введет. Иначе возвращается функция get\_number\_input\_after<T>(str)
2. get\_number\_input\_after<T>(str)
   1. template <typename T>
   2. T get\_number\_input\_after(string str)
   3. {
   4. while (true) // цикл продолжается до тех пор, пока пользователь не введет корректное значение
   5. {
   6. if (check<T>(str) == 1)
   7. {
   8. return str\_to\_number<T>(str);
   9. }
   10. getline(cin, str);
   11. }
   13. }

Функция запускает цикл пока пользователь не введет число того типа, которого запросила функция. Проверяется ввод, если человек ввел число, то возвращается преобразованное число из строки. Если нет, то человек снова вводит число.

1. bool check(string str)
   1. template <typename T>
   2. bool check(string str)
   3. {
   4. bool k = 1;
   5. string s = typeid(T).name();
   6. if ((str.find('-') < 1 || str.find('-') == -1) && ((str.find('.') < str.size() - 1 && str.find('.') > 0) || str.find('.') == -1))
   7. {
   8. if (str.length() > 1)
   9. {
   10. int count\_minus = 0, count\_point = 0;
   11. for (int i = 0; i < str.length(); i++)
   12. {
   13. if (str[i] == '-')
   14. {
   15. count\_minus++;
   16. }
   17. if (str[i] == '.')
   18. {
   19. count\_point++;
   20. }
   21. }
   22. if ((str[0] == '0' && str[1] != '.') || count\_point > 1 ||
   23. (str[0] == '-' && str[1] == '.') ||
   24. (str[0] == '-' && str[1] == '0' && str[2] != '.') || count\_minus > 1)
   25. {
   26. k \*= 0;
   27. std::cout << "!!Вы ввели не число!!" << std::endl << "Введитe число: ";
   28. }
   29. }
   30. else
   31. if (str == "-")
   32. {
   33. k \*= 0;
   34. std::cout << "!!Вы ввели не число!!" << std::endl << "Введитe число: ";
   35. }
   36. if (k == 1)
   37. {
   38. if (s == "double")
   39. {
   40. for (int i = 0; i < str.length(); i++)
   41. {
   42. if ((str[i] < '0' || str[i] > '9') && str[i] != '.' && str[i] != '-')
   43. {
   44. k \*= 0;
   45. std::cout << "!!Вы ввели не число!!" << std::endl << "Введитe число: ";
   46. break;
   47. }
   48. }
   49. }
   50. if (s == "int")
   51. {
   52. for (int i = 0; i < str.length(); i++)
   53. {
   54. if ((str[i] < '0' || str[i] > '9') && str[i] != '-')
   55. {
   56. k \*= 0;
   57. std::cout << "!!Вы ввели не целое число!!" << std::endl << "Введитe целое число: ";
   58. break;
   59. }
   60. }
   61. }
   63. }
   65. }
   66. else
   67. {
   68. k \*= 0;
   69. std::cout << "!!Вы ввели не число!!" << std::endl << "Введитe число: ";
   70. }
   72. if (s == "bool")
   73. {
   74. if ((str[0] < '0' || str[0] > '1') || str.size() > 1)
   75. {
   76. k \*= 0;
   77. std::cout << "!!Вы ввели не булево число!!" << std::endl << "Введитe булево число: ";
   78. }
   79. }
   80. return k;
   81. }
2. T str\_to\_number(std::string str)
   1. template <typename T>
   2. T str\_to\_number(std::string str)
   3. {
   4. T a = 0; //само число
   5. int index = 0, k = 1;
   6. if (str[0] == '-')
   7. {
   8. k = -1;
   9. str.erase(0, 1);
   10. }
   11. if (str.find('.') != -1)
   12. {
   13. index = str.find('.');
   14. str.erase(str.find('.'), 1);
   15. }
   16. else
   17. index = str.length();
   18. for (int i = 0; i < str.length(); i++)
   19. {
   20. a = a \* 10 + (str[i] - '0');
   21. }
   22. return k \* (a / pow(10, (str.length() - index)));
   23. }

Задача 1.

1. Функция find\_winner\_solo находит массив победителей.
   1. Принимает исходный массив персонажей, персонажа противника.
   2. Создает фантомов.
   3. Идет по всем перс. исходного массива.
   4. Для каждого персонажа обновляет время и фантомов.
   5. Запускает цикл while, пока здоровье нашего персонажа или персонажа противника минус урон не станет меньше нуля.
   6. Каждую итерацию увеличивает счетчик времени
   7. Дальше идет один из главных алгоритмов:
      1. Если время вывода из строя соперника (time) больше нуля, то персонаж способен выводить из строя соперника, значит идем дальше.
         1. Если интервал равен времени
            1. тогда времени фантома присваивается время настоящего персонажа.
         2. Если время фантома больше нуля
            1. тогда время фантома уменьшается, фантом противника становится выведенным из строя, урон фантома противника становится равен нулю, интервал фантома нашего становится равен интервалу настоящего.
         3. Иначе
            1. фантом противника становится снова в строю, урон фантома противника становится равен урону настоящего противника, интервал уменьшается
      2. То же самое происходит и с фантомом противника, только меняются переменные
   8. Дальше выводится время, время боя персонаж нашего становится равно времени, здоровье фантома нашего уменьшается на урон противника и наоборот.
   9. Дальше печатается строка характеристик нашего персонажа и персонажа противника.
   10. После цикла while мы смотрим, если здоровье нашего фантома минус урон противника равно нулю, значит мы убираем из массива этого персонажа и уменьшаем i на 1.
   11. Иначе процент оставшегося здоровья итого персонажа становится равен здоровье.

Задача 2

1. Функция void find\_winner\_team(vector <vector < Character >>& teams, vector < Character >& team\_enemy)
   1. Принимает массив команд
   2. Идет по всем командам
   3. Идем по похожему алгоритму как в задаче 1, сначала находим урон всех персонажей потом отнимаем урон от здоровья. Только тут нужно перебрать сначала всех наших персонажей и для каждого персонажа перебрать всех противников, а потом наоборот.

И тут возникает проблема того, что каждый раз когда мы работает с каждым персонажем нужно сортировать массив, чтобы находить людей с максимальным уроном. Но если все нули, то мы сортируем исходя из начальных значений реального урона.

Потом нужно уменьшить у всех здоровье и если здоровье стало меньше или равно нулю, то человек умирает.

Если здоровье всех в командах равно нулю, то выходим из цикла. Если наших, то мы проиграли, иначе мы выиграли, добавляем в выходной массив нашу команду.

# Листинг программы.

https://github.com/DaniilMiagkovOk/Dop\_task\_1.git

# Вывод.

Программа принимает на вход массив персонажей, потом либо находит победителей в боях 1 на 1, либо победителей в командных боях.