vector<int>v(2000,INT\_MIN); // создаем вектор, который будет вмещать в себя все “ходы” игроков.

\*\* Инициализируем все значения(2000) вектора INT\_MIN.

int game(int x){

if(v[x] != INT\_MIN){ // возвращаем возможные “ходы”.

return v[x];

}

if(x >= 1000){ //Очевидно, что если число больше 1000, то мы должны вернуть 0.

return 0;

}

vector<int>neg,tmp={game(x+100),game(x\*2)}; // определяем рекурсивно все возможные “ходы”.

for(int x: tmp){ // “кладем ходы” в вектор.

if(x<=0){

neg.push\_back(x);

}

}

*// основной код функции*

int res; **// результат**

if(!neg.empty()){ **// если вектор neg не пустой, то мы должны положить в результат**

res = -\*max\_element(neg.begin(),neg.end())+1;

**// Берем максимальное из отрицательных значений или 0, т.к. Наличие отрицательного(или 0) числа говорит нам о том, что в этой комбинации есть победные значение , чтобы сходить в эту позицию, нам нужно вернуть число по модулю и добавить к нему 1.**

} else {

res = -\*max\_element(tmp.begin(),tmp.end());

**// если в векторе tmp нет отрицательных, то игрок в позиции x и y проигрывать и его цель затянуть игру, то есть его цель – проиграть за максимальное количество шагов.**

}

return v[x] = res; // возвращаем элемент

}

Пример:

Позиция x y

Tmp = {1,**-1**, 2, 1} // вектор содержит отрицательные значения -1, это означает, что игрок, делающий ход из позиции x y, победит, потому что он будет ходить в -1 , тем самым заведет второго игрока в проигрышную позицию.

// значит, что функция вернет 2, И значит, что в позиции x y игрок выигрывает, сделав максимум 2 ходы

Tmp = {1, **1**, 2, 1}} // вектор не содержит отрицательные значения, это означает, что игрок, делающий ход из позиции x y, проиграет , его цель – как можно дальше отнять проигрыш, поэтому мы берем максимальное значение.

// значит, что функция вернет -2. И значит, что в позиции x y игрок проиграет, сделав максимум 2 ходы