**Документация**

«Стрелялка 2D»

.

Минск, 2021

**Содержание**

[Введение 3](#_Toc88698006)

[1. Содержание функции main 4](#_Toc88698007)

[1.1. Функция startMenu() 4](#_Toc88698008)

[1.2. Главная модульная часть 6](#_Toc88698009)

[1.2.1. Функция draw() 6](#_Toc88698010)

[1.2.2. Функция input() 8](#_Toc88698011)

[1.2.3. Функция logicForAI() 10](#_Toc88698012)

[1.2.3.1. Стрельба ИИ 11](#_Toc88698013)

[1.2.3.2. Движение ИИ, выбор цели 11](#_Toc88698014)

[1.2.4. Функция logic() 13](#_Toc88698015)

[1.2.4.1. Движение персонажей 13](#_Toc88698016)

[1.2.4.2. Стрельба 15](#_Toc88698017)

[1.2.4.3. Бонусы 17](#_Toc88698018)

[1.2.4.3.1. Бонус на тройной выстрел 18](#_Toc88698019)

[1.2.4.3.2. Бонус на дополнительные очки 19](#_Toc88698020)

[1.2.4.4. Система уровней игры 19](#_Toc88698021)

[1.2.4.5. Блокирующая стена 20](#_Toc88698022)

[1.2.5. Функция winner() 22](#_Toc88698023)

[2. Блок-схема 25](#_Toc88698024)

[3. Псевдокод 26](#_Toc88698025)

[Выводы 28](#_Toc88698026)

# **Введение**

Консольные приложения, да и, впрочем, все приложения строятся с помощью алгоритма и стиля программирования. Программа написана в стиле модульного программирования. В алгоритме моего приложения лежат 3 основные функции и несколько функций облегчающих код.

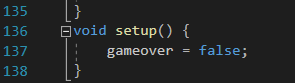
Одна из трех функций отвечает за постоянное обновление консоли и заполнением ее символами, обозначающих поля, персонажей, бонусов, стен и выстрелов. Вторая функция отвечает за отслеживание нажатия кнопок управления. Третья функция отвечает за всю логику игры, за обработку нажатия кнопок, да и за многое другое. Также есть вторая функция, отвечающая за логику. Только она отвечает за логику примитивного искусственного интеллекта.

# **1. Содержание функции main**

Функция main ­—  главная функция, в которой содержатся все функциональные части.

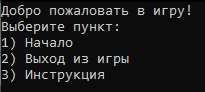
Изначально я поменял кодировку консоли с помощью функций SetConsoleCP() и SetConsoleOutputCP() c Windows-866 на кодировку Windows-1251. Для реализации такого, нужно подключить библиотеку windows.h, что и сделано вначале приложения. Это позволяет правильней отслеживать нажатые кнопки и подключить в приложение русский язык.

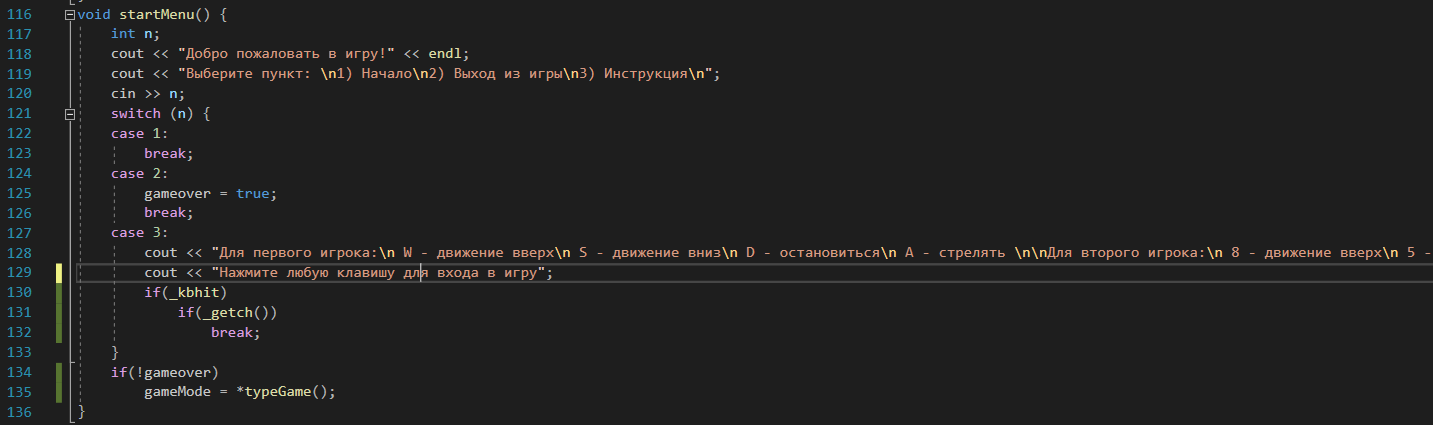
Далее идет функция setup(). Она дает указание приложению, что игра готова для запуска. В функции присваивает значение глобальной булевой переменной gameover, которая даст знать, когда закончится игра, на false.



## **1.1. Функция startMenu()**

Следующая функциональная часть после setup() ­—  startMenu(). Эта функция отвечает за выведение начального меню приложения.





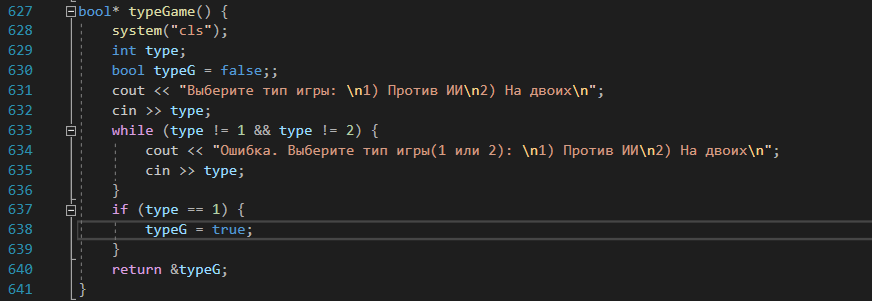
Изначально создается целочисленная переменная “n”. Она будет использоваться для выбора пункта меню выше. За отслеживание значения “n” отвечает оператор switch().

Если n == 1, мы пойдем в case 1. Там мы просто выйдем из switch() и пойдем выбирать тип игры.

Если n == 2, мы пойдем в case 2. Тут переменной gameover присвоится значение true, что означает выход из игры. Следующая строка означает выход из switch().

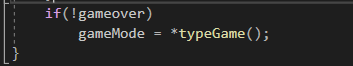
Если n == 3, нам выведется с помощью стандартного потока вывода cout инструкция по управлению персонажами. Далее, заранее подключив библиотеку conio.h мы с помощью функций \_kbhit() и \_getch() ждем нажатия на любую клавишу. Если оно последовало, то мы перейдем к выбору типа игры.

После оператора switch() стоит условный оператор if, в котором проверяется, gameover == false? Если нет, то действие перенесется в функцию winner(). Если да, то мы будем выбирать тип игры с помощью указателя на функцию typeGame().



Тип функции typeGame bool, логический. Так как в конце мы должны возвращаем значение адреса, где будет храниться значения типа игры.

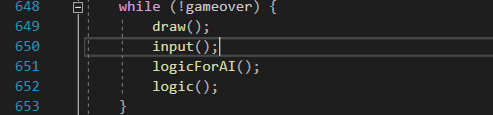
Изначально мы очистим консоль при помощи функции system() с параметром «“cls”». Потом в консоль будет выведено меню, где мы попросим выбрать: 1 – игра в одиночку против Искусственного Интеллекта, 2 – игра вдвоем с другом. Пользователь будет присваивать значение целочисленной переменной type. Далее будет проверка на правильное введение type. Если игрок не ввел 1 или 2, система укажет на ошибку и попросит ввести число заново. В следующем действии мы будем проверять, если type == 1, то мы присвоим булевой переменной typeG значение true, если нет, тогда ничего не присваиваем, а просто оставляем значение false. Потом уже возвращаем значение адреса на typeG.



Далее мы возвращаемся в функцию startMenu(), где заранее созданная глобальная булевая переменная gameMode заполучит значения разыменованного адреса функции typeGame().

На этом работа функции startMenu() заканчивается.

## **1.2. Главная модульная часть**

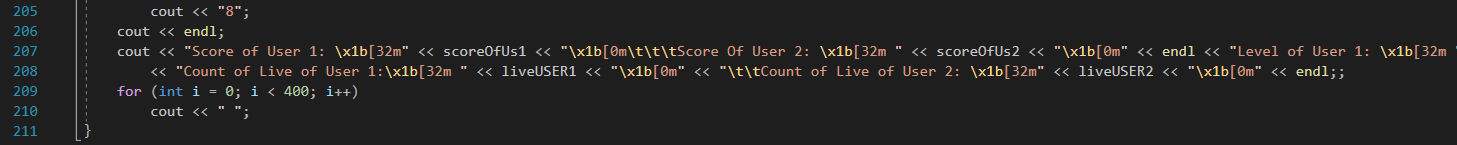


Цикл while, с проверкой на gameover == false, —  главный модуль приложения, в котором будут происходить все главные функции: зарисовка поля, отслеживания нажатия и логики приложения.

### **1.2.1. Функция draw()**







Функция draw() создана при помощи типа данных void или, как называют, пустого типа данных. Она отвечает за зарисовку поля. Отслеживает координаты персонажей, стен, бонусов, и, если они совпадают, она зарисовывает символ, означающий персонажа, или стены, или бонуса.

Отслеживание происходит, благодаря конструкциям if, if-else, else if, и двум циклам, причем один вложен в другой. Заранее были созданы целочисленные переменные, означающие длину и ширину поля. При этом длина поля — константное значение(был использован модификатор const).



Изначально функция draw() очищает терминал с помощью функции system(“cls”). Далее с помощью цикла for(142 строчка кода) идет зарисовка верхней границы. После чего мы переходим на новую строку и подключаем новый цикл for(стр. 145), в нем инициализируем переменную i, которая будет отвечать за ось Y на поле. В этом же цикле подключаем второй цикл for(стр. 146), В нем инициализируем переменную j, которая будет отвечать за ось X на рисунке. И, когда будет j == 0 или j == width – 1, мы будем рисовать символ границы(символ границы —  «8»). Следующие операторы else if будут отслеживать координаты персонажа 1 и персонажа два, бонус тройного выстрела и бонус дополнительных очков, стены, персонажа 1 и персонажа два во время попадания в них патроном.

Изменения цвета персонажей, пули возможно, благодаря командам оболочек. Например: *\x1b[33m%\x1b[0m*. Где \x1 — вид написания(жирный, подчеркнутый и так далее), b[33 — цвет оформления(менять только число), m – элемент, который будем красить.

Ко многим операторам else if мы еще вернемся в дальнейшем. А пока на строчке кода 204 работает цикл for, который отвечает за зарисовку нижней границы символом «8».

Также под основным игровым окном есть табличка содержащая количество очков, количество жизней и уровень двух пользователей. Сделана при помощи стандартного потока вывода *cout.* Все значения получает из функции logic().



### **1.2.2. Функция input()**

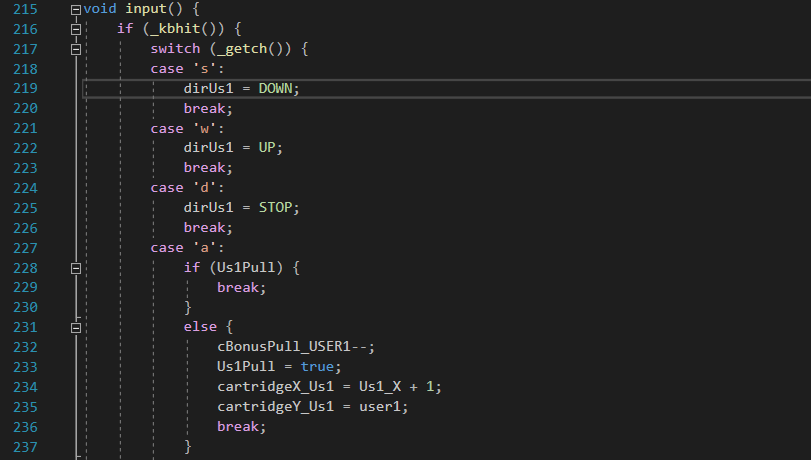
Функция inpit() — вторая главная функция, без которой не было бы всей анимации движения, выстрела. Она отслеживает все нажатые кнопки, обрабатывает их и отправляет данные в функцию logic().

Для отслеживания подключена библиотека conio.h. Благодаря функции \_kbhit() мы узнаем, что кнопка нажата, а с помощью функции \_getch() узнаем, какая кнопка была нажата. \_getch() лежит в операторе switch().

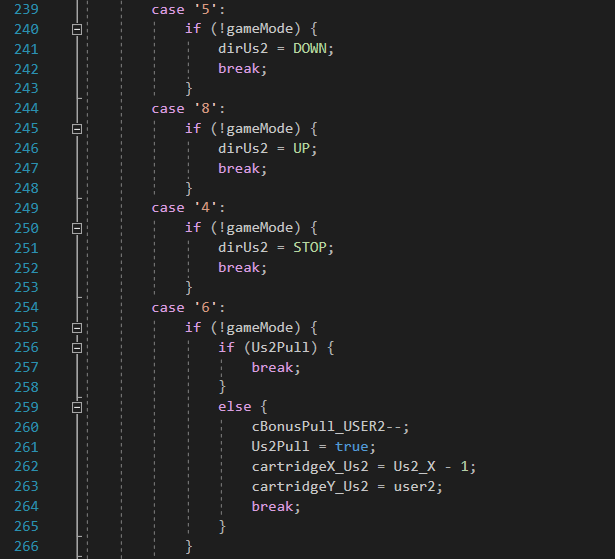
Заранее было создано сложный тип данных, перечисление(enum). Оно хранит в себе 3 значения: UP, DOWN, STOP. Которые означают состояния движения персонажа относительно оси Y.



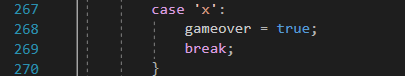
Создал свой перечислительный тип данных DirectionUs. А потом создал переменные dirUs1 и dirUs2 на основе перечисления.



Вернемся к switch(), который отслеживает, какая кнопка на клавиатуре была нажата. Если игрок 1 нажал «W», dirUs1 = UP, то есть персонаж 1 пойдет вверх, для второго игра аналогично будет, только должна быть нажата «8» и dirUs2 = UP. Если игрок 1 нажал «S», а игрок 2 — «5»., dirUs1 и dirUs2 получат значения DOWN. От нажатия на «D» или «4» положения игроков получат значения STOP. А, если игрок 1 нажал на «А» и пуля еще не выпущена, функция input() присвоит булевой переменной Us1Pull значение true, что означает, что игрок 1 выстрелил. Также присвоятся значения переменным, означающих координаты пуль. Пуля будет иметь начальные координаты перед персонажем. Для второго персонажа выстрел будет происходить тоже самое. Только, если игрок выбрал режим игры «Против ИИ», то от нажатия кнопок управления для второго игрока ничего не будет происходить. Там находится условие, которые блокирует нажатие кнопок.



При нажатии на кнопку «Х» input() присвоит значение переменной gameover true и выйдет из оператора switch(), что значит, что игра закончена. И действие перенесется в функцию winner().



### **1.2.3. Функция logicForAI()**

Функция logicForAI() будет работать, если только gamemode == true, то есть игрок выбрал, что хочет играть против искусственного интеллекта.

Эта функция, работает на обычных операторах if, else if, if-else. Уже компьютер выбирает, когда ехать вверх, когда ехать вниз, когда преследовать игрока, когда преследовать бонус, когда стрелять.

#### **1.2.3.1. Стрельба ИИ**

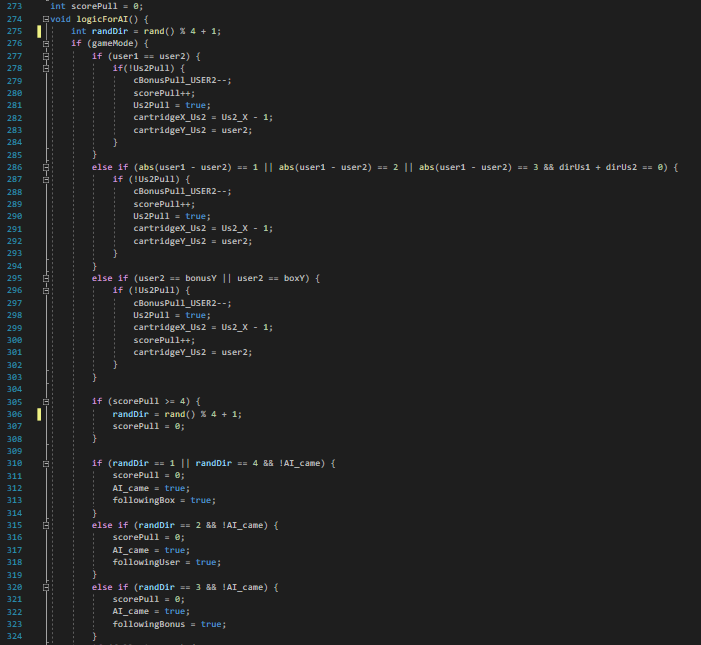
Стрелять персонаж будет лишь тогда, когда координата по У игрока 1 будет совпадать с координатой по У второго игрока или же отличаться на 1, 2 или 3 по модулю(Это сделано для большего шанса попадания в персонажа противника). Также ИИ будет стрелять, точнее передаст функции logic(), что можно стрелять, когда координата по Y игрока 2 совпадет с координатой по Y любого бонуса. Также, если ИИ попал в одну точку несколько раз, он поменяет свою цель на случайную, то есть, либо на бонус или игрока. На бонус, дающий дополнительные очки, двойной шанс.

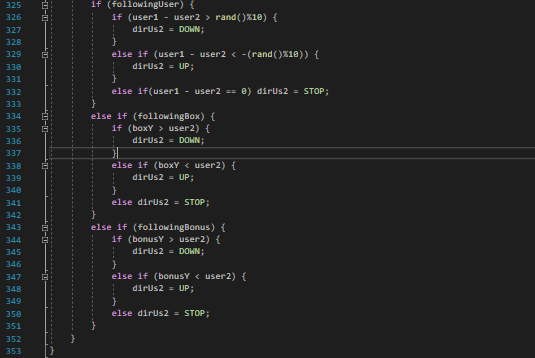
#### **1.2.3.2. Движение ИИ, выбор цели**

Теперь ехать вверх или вниз, или вообще остановится будет отвечает не функция input(), а функция logicForAI(). Цель, которую будет преследовать игрок выбирается случайно. Изначально создается локальная целочисленная переменная randDir, которая изначально равно случайному числу от 1 до 4(Сделано при помощи функции rand()).

Заранее создаем несколько булевых переменных: AI\_came, чтобы знать, преследует ли кого-нибудь или что-нибудь наш ИИ в данный момент; FollowingBox, чтобы знать: он преследует один из бонусов; FollowingUser, чтобы знать, преследует ли он игрока 1; FollowingBonus, чтобы знать, преследует ли он второй бонус.

Если одна из переменных активна, система будет стараться подогнать позицию бота по Y под позицию Y его цели. То есть меняя значение dirUs2.





### **1.2.4. Функция logic()**

Функция logic() — самая важная функция в приложении, без которой не было бы никакой анимации, системы уровней, бонусов, препятствий.

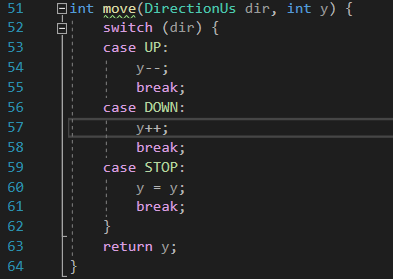
Эта функция содержит много подфункций, которые помогают сократить код и убрать одинаковые части кода, заменив их на одну функцию.

#### **1.2.4.1. Движение персонажей**

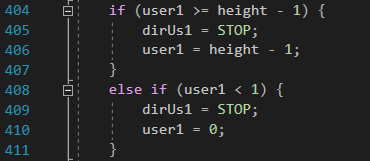


Переменные user1 и user2 — означают положения персонажа 1 и персонажа 2 на оси Y. Эти переменные будут изменятся относительно того, чему равно dirUs1 или dirUs2.

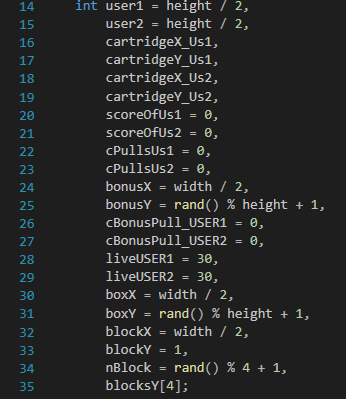
Функция move() состоит из 2 параметров, у первого параметра тип данных DirectionUs, У второго параметра тип данных целочисленный. С помощью switch() мы получаем направление персонажа, если ему нужно вверх в поле, то позиция персонажа по Y уменьшиться. Если нужно вниз, то позиция увеличится на 1. Если нужно остановится то, позиция по Y будет равна текущей позиции. И в конце функция возвращает значение для новой позиции Y персонажа.



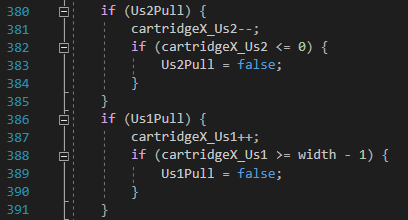
Если персонаж выйдет за пределы границы, сработает условие, которое, если выходит за верхнюю границу, вернет на позицию 0 и остановит персонажа или, если выходит за нижнюю границу, вернет на позицию height - 1 и остановит персонажа.



#### **1.2.4.2. Стрельба**

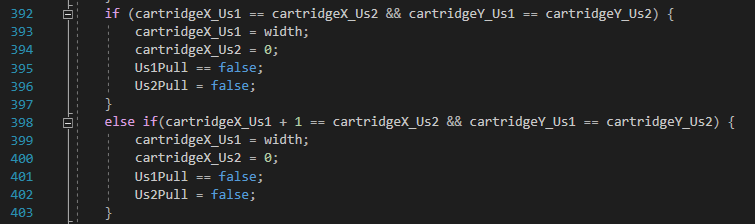


Заранее на строчках от 16 до 19 созданы переменные, которые означают координаты по Х и У пуль.



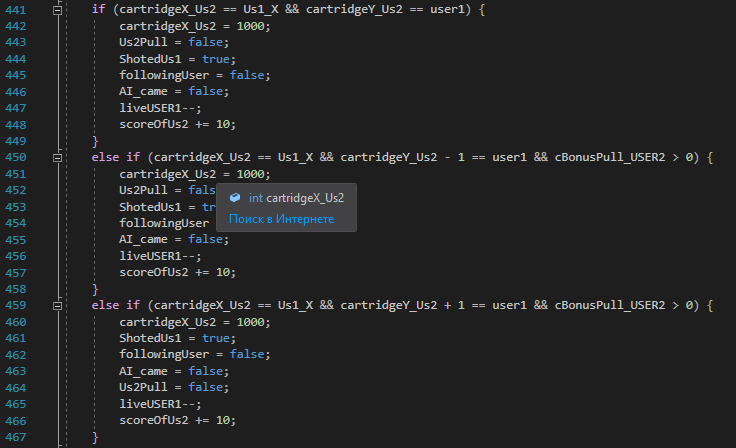
В функции input() мы отслеживаем, нажали ли игроки на кнопку выстрела, если да, то Us1Pull или Us2Pull примут значение true. И если у них значение истинно, то для пули создадутся и их координаты по Х будут изменяться на один в сторону противника. Но, когда пуля дойдет до границы, ты пуля исчезнет и игрок снова сможет стрелять.

Если координаты по Х и У пули первого игрока совпадают с координатами пули второго игрока. То пули исчезнут. И игроки снова смогут стрелять.



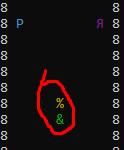
Попадание будет засчитано, если координаты пули будут равны координатам противника. Это проверяет if и else if. Если игрок взял бонус, который дает выстрел тремя пулями, то else if проверяет на попадание одной из пуль в игрока. Если режим игры «Против ИИ», то после попадания перестанет преследовать игрока.

Также после попадания игроку, который попал, начисляется 10 очков, а игрок, в которого попали, теряет 1 жизнь.



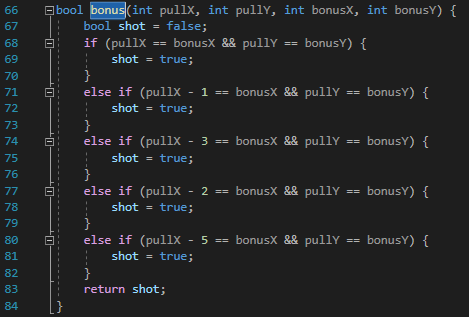
#### **1.2.4.3. Бонусы**

Есть два типа бонусов. Один дает тройной выстрел, а второй дает дополнительный очки для улучшения персонажа.



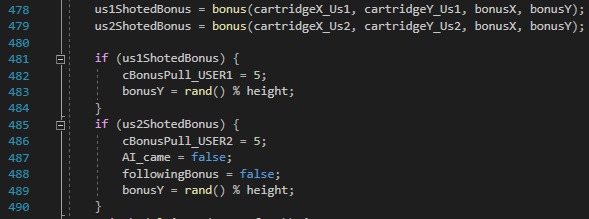
Попадание в бонус будет рассчитывать функция bonus(). Она содержит 4 параметра. Первые два отвечаю за координаты пули по Х и У, а вторые два параметра за координаты бонуса.

В функции создана локальная логическая переменная shot, что означает попал ли игрок в бонус или нет. Изначальное ее значение — false. Но в случае множественных проверок shot может получить значение true, то есть дать знать, что игрок попал по бонусу. Проверяется, сходятся ли координаты пули и бонуса. И для случая, когда уровень игрока побольше, может быть такое, что пуля будет перебегать координату по Х бонуса, поэтому, если совпадет координата по Х отличающая на 1, 2, 3 или 5, то бонус будет взят. В конце функции просто идет возращение значения переменной shot.



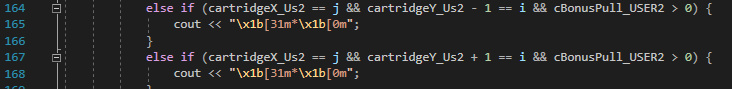
##### **1.2.4.3.1. Бонус на тройной выстрел**

Если, допустим, игрок 1 попал в оранжевый символ « % », то булевая переменная us1ShotedBonus получит значение true. И, если значение true, игроку дастся 5 бонусных выстрелов, а также бонус поменяет свое местоположения по оси Y с помощью функции rand() в пределах высоты поля. А для игрока 2 будет сообщено, что если режим игры «Против ИИ», то он может перестать преследовать цель и поменять ее.

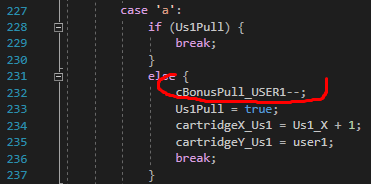


Бонусные пули будут привязаны к основной, только их координаты по Y будут отличаться на 1. Бонусные пули не могут собирать бонусы.

Зарисовкой бонусных пуль будут заниматься специальные else if, которые будут сверять, равны ли координаты и есть ли попытки на бонусные пули.



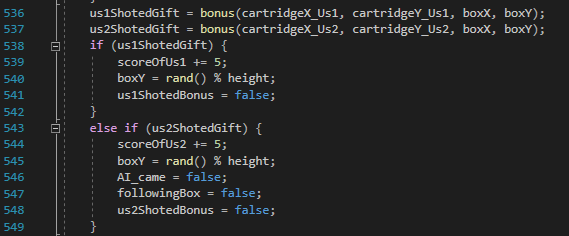
Бонусные пули будут тратиться, когда игрок нажмет на кнопку выстрела и функции input() обработает это.



##### **1.2.4.3.2. Бонус на дополнительные очки**

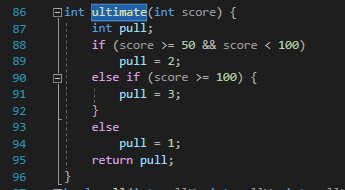
Важный бонус для прокачки персонажа. Бонус обозначается зеленым знаком амперсанта. При попадании в бонус дается 5 очков для улучшения.

Проверка на попадание происходит так же, как и с предыдущим бонус, то есть, с помощью функции bonus(). Если игрок попал в бонус, переменная scoreOfUs1 или, для игрока 2, scoreOfUs2, хранящая в себе очки игрока, увеличится на 5. А координата ящика по Y, которая хранится в переменной boxY, поменяется на случайную в пределах высоты поля. А для режима «Против ИИ»: бот после попадания в этот бонус потеряет цель попасть в этот ящик. А после переменная us1ShotedBonus получит значение false. И чтобы снова получить значение true, нужно попасть в этот бонус еще раз.

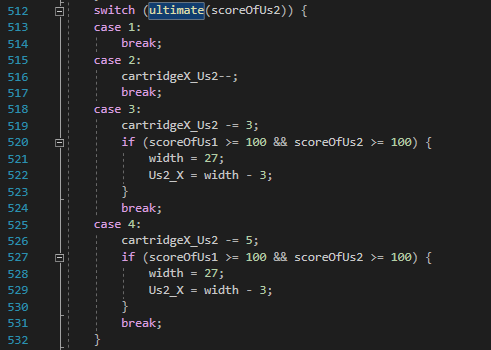


#### **1.2.4.4. Система уровней игры**

Уровень персонажа будет зависеть от его количества очков. Определять уровень игрока будет функция ultimate(). В этой функции есть один параметр score, то есть количество очков игрока. Если их больше 50 и меньше 100, уровень будет 2, если больше 100 очков, будет 3 уровень, а если ни один из вариантов не подошел, будет уровень 1. Уровень будет влиять на скорость изменения координаты по Х пули.



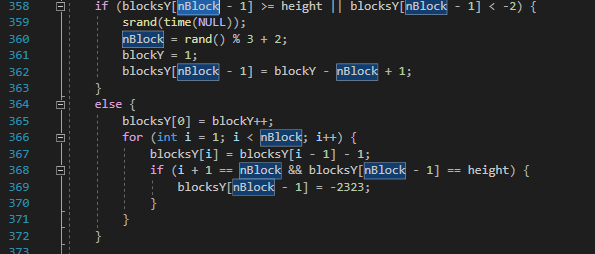
Функция ultimate() стоит в операторе switch(). И там в зависимости от уровня игрока сработают разные операторы case. Если уровень 1, то увеличения скорости никакого не будет. Если уровень 2, то увеличение будет дополнительно еще на 1, то есть, скорость будет 2х. Если уровень 3, то изменение скорости будет на 3, а также, если у обоих игрок уровень больше или равен 3, карта увеличится в два раза по ширине, поэтому значение width не константа. Если уровень игрока 4, то увеличение скорости будет 6х.



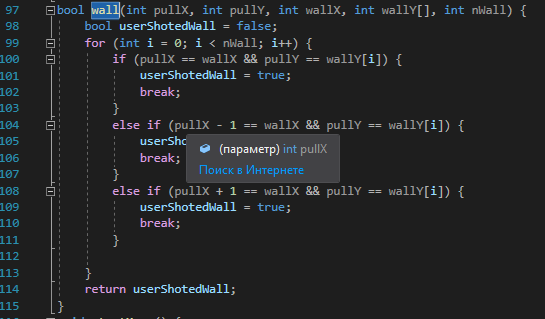
#### **1.2.4.5. Блокирующая стена**

Блокирующая стена периодически появляется на поле. Ее длина постоянно разная. Длина Хранится в массиве blocksY[]. Она может быть от 1 до 4 элементов. В начале функции logic() идет проверка: последний блок стены за границей или нет. Если да, то мы меняем количество блоков стены, а для первого блока ставим координату по Y 1. По Х координаты стены, бонусов всегда находятся посередине поля. А координата по Y последнего блока будет равна координате первого блока, уменьшенного на количество блоков, плюс 1.

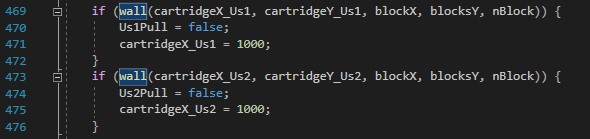
А, если стена находится в пределах области длины, то координата первого блока стены, будет увеличивать на 1, относительно начальной координаты, а координаты последующих блоков будут изменятся на один относительно предыдущего блока. И, если следующая итерация в цикле будет уже последней и последний блок зашел за пределы, последний блок получит координату -2323, это для того, чтобы обновиться стену.



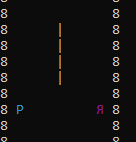
Попаданием патрона в стену занимается функция wall(). Тип этой функции логический. В конце она вернет true или false, что означает, попал ли игрок в стену. Тут работает цикл, в котором проверяется, попала ли пуля хотя бы в один из блоков стены.



И, и если игрок попал в один из блоков стены, булевая переменная Us1Pull примет значение false, что отменит выстрел и снова разрешит сделать новый.



В приложении стена выглядит, как модульные символы.

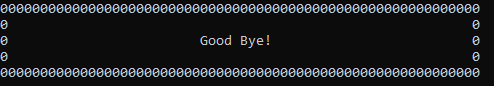


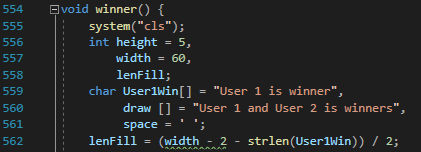
### **1.2.5. Функция winner()**

Функция winner() — модульная часть, которая включится, когда игра будет завершена, то есть, переменная gameover == true.



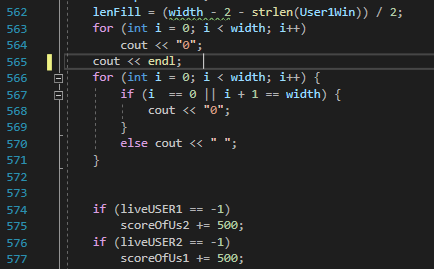
Изначально функция очищает терминал для заполнения его объявлением победителя или прощанием с игроками.





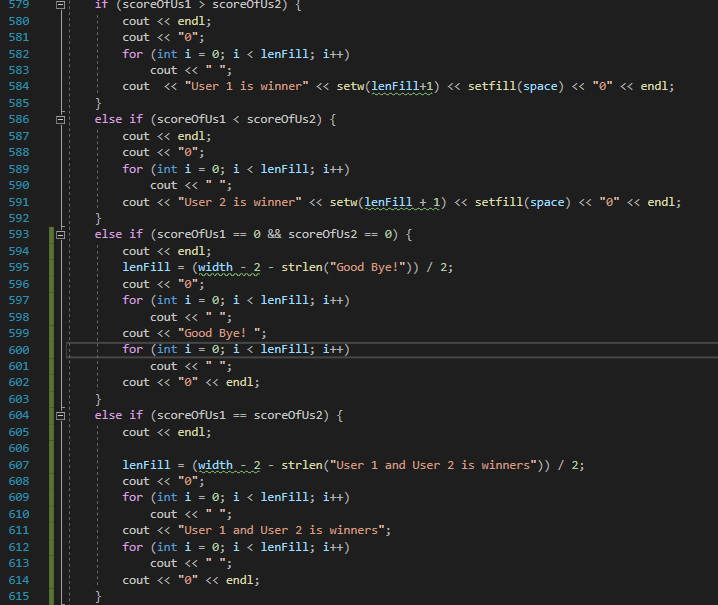
Далее мы задаем длину и ширину поля, где будет располагаться сообщение, а также инициализируем переменную space, присвоив ей значение « ‘ ’ »(пробел). Также создаем переменную ,которая будет обозначать область для заполнения пробелами lenFill. Еще из массива char создаем строки с объявлением победителя в строках 559-560.

lenFill будет равна не занимаемому текстом пространству.

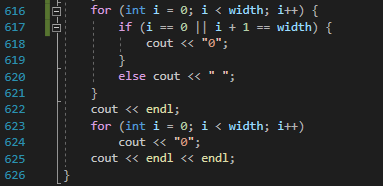


В строке 563 с помощью цикла for идет зарисовка верхнего поля для таблички. Далее после перехода на следующую строку будет идти зарисовка левого и нижнего поля(строки 566 – 571). Далее идет проверка, если первый игрок убил второго, то ему дастся +500 очков, если второй, то ему дастся +500 очков.

Далее будет зарисовка таблички относительно того, сколько очков у первого игрока и сколько очков у второго игрока. Если у первого игрока очков больше, то мы перейдем на новую строчку, нарисуем левую границу, заполним лишнее пространство пробелами через цикл for. И выведем надпись User 1 is winner, а далее, подключив библиотеку iomani, с помощью функции setw() установили место для заливки, а с помощью функции setfill() заполнили место пробелами. И в конце поставили знак правой границы. Выведения таблички, что игрок 2 победил, будет сделан по такому же алгоритму, как и выведение того, что первый игрок победил. Другие таблички тоже, будут выводится также, только заполнение после надписи будет происходит с помощью цикла for, не используя библиотеку iomanip.

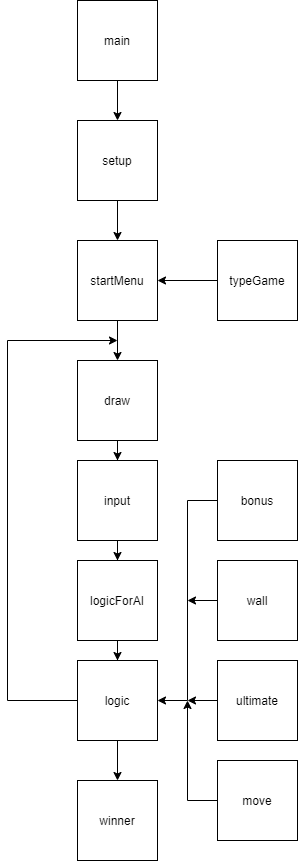


И в конце мы создадим пустую строку с помощью цикла for, заполненную по краям символами границы, а внутри пробелами. И в конце, также с помощью цикла for зарисуем нижнюю границу.



И после всего этого игра завершается.

# **2. Блок-схема**



Блок-схема сделана в виде функциональных частей программы.

# **3. Псевдокод**

Начало программы

Запуск начального меню

Выбор пункт меню

Если 1

Переход к выбору режима

Если 2

Выход из приложения

Если 3

Выведение инструкции

Переход к выбору режима

Конец Если

Выбор режима

Если 1

Подключить функцию logicForAI(), отвечающую за бота

Включить игру

Если 2

Разрешить управление игроку 2

Включить игру

Начало цикла. Если !gameover

Подключение функции draw()

Зарисовка поля, персонажей

Подключение функции input()

Отслеживать нажатые кнопки

Если игра против ИИ

Переход в функции logicForAI

Построить план, чтобы добраться до цели и выстрелить в нее

Отправить значения в функцию logic()

Если игра против второго игрока

Переход в функцию logic()

Обработка данных полученных из функций logicForAI(), input()

Переход к проверке цикла

Переход в функцию winner().

Вывод поля победителя.

Конец программы.

# **Выводы**

Для создания меню, перед входом в игру, или отслеживанием, куда будет двигаться персонаж, лучший способ — конструкция switch – case.

Для создания ИИ, зарисовки в нужных местах, проверки каких-то значений лучший способ — условные операторы if, else if, if-else.

Для создания двумерного поля, зарисовки его лучший способ — использовать цикл и вложенный в него цикл.

Для перебора массива лучше использовать обычный цикл For.

Для постоянного вызова функций лучшее средство — цикл while.

Для более стабильной работы кода, облегчения кода лучшее средство — использовать функции.

Для реализации проекта, нужно построить алгоритм своих действий, какие функции будут участвовать и, что они будут делать.

Также для больших проектов лучше называть переменные максимально логическими именами.