**Содержание:**

**1.Введение1**

1.1. Цели1

1.2. Задачи1

1.3. История создания “змейки”1

1.4. План разработки приложения1

**2.Инструкция к игре**2

2.1 Меню2

2.2 Игровой процесс2

**3.Техническая документация к игре3**

3.1. Псевдокод3

3.2. Блок-схема8

3.2.1. Функция main8

3.2.2. Функция setup9

3.2.3. Функция draw10

3.2.4. Функция input11

3.2.5 Функция logic12

**4.Заключение13**

1. **Введение.**

**1.1. Цели.**

Целью данной мини-игры являлось повторение игрового процесса всем известной игры “змейка”, при этом, используя исключительно язык С/С++. При выполнении данного задания мы пользовались командами, которые были изучены на лекциях и в ходе выполнения лабораторных работ.

**1.2. Задачи.**

Перед собой мы поставил следующие задачи:

- Повторить gameplay игры “Змейка”;

- Разработать меню;

- При разработке использовать известные для нас команды ЯП С/С++;

- При необходимости, дополнительно изучить команды, которые помогут реализовать приложение.

**1.3. История создания игры “Змейка”.**

Игра “Змейка” – самая популярная мобильная игра на границе тысячелетий. Можно смело предположить, что на сегодняшний день трудно найти людей, которые бы в неё не играли или, хотя бы, не слышали про неё. История данной игры началась за несколько лет до появления первых мобильных телефонов. В 1977 году компания Gremlin выпустила игровой автомат Hustle, в которой пользователю нужно было управлять “змейкой”.

**1.4. План разработки приложения.**

Создание мини-игры SNAKE мы разбили на два этапа:

1) Разработать меню;

2) Разработать gameplay самой игры.

1. **Инструкция к игре.**

**2.1 Меню.**

В игровом меню пользователю предоставляется 4 варианта выбора:

- New game (новая игра). Это переход непосредственно к gameplay нашей мини-игры.

-Instruction. В этом разделе меню указаны основные команды для управления змейкой.

-Exit. При нажатии на этот пункт меню, пользователь выйдет из консольного окна.

Переключение между пунктами меню происходит при помощи стрелок (cursor control keys) вверх (up) и вниз (down). Для выбора того или иного варианта меню используется клавиша enter. Также для досрочного выхода из консольного окна можно использовать клавишу Esc.

**2.2 Игровой процесс.**

Во время игрового процесса пользователь может использовать клавиши для передвижения – это ‘w’, ’s’, ’a’, ’d’. Для выхода из игры – ‘x’. Игра происходит до тех пор, пока змейка не коснется своего хвоста. Во всех остальных случаях игра будет продолжаться.

За каждую съеденную \*, у змейки увеличивается хвост, а к score(счёт) прибавляется 10 очков.

В случае, если змейка врезается в границу игрового поля – змейка появляется с противоположной стороны и игра продолжается.

1. **Техническая документация к игре.**

**3.1 Псевдокод.**

1) Создание и объявление глобальных переменных, таких как: размеры игрового поля, создание игровых объектов (змейка и фрукты), переменную типа enum для создания переменных передвижения змейки, переменную типа bool, массивы для хвоста нашей змейки (по координатам X и Y) и т.д.

2) Функция main начинается с функции для подключения русского языка, функции для генерации случайных чисел, установки цветов для фона консоли и символов, отчистки экрана.

3)Вызов функции showLogo(), которая выводит название игры – SNAKE. В данной функции у меня вызывается еще одна функция – gotoxy, которая позволяет задать координаты расположения надписи, в ней же содержится переменная COORD, которая и содержит координаты, которые будут переданы через параметры, а функция SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), p) это реализует. Так же функция Sleep (1200) позволяет в течении 1.2с демонстрировать в консоли эту надпись.

5) Создание переменных для создания меню: const int num\_menu\_items = 4, т.к. в меню я планирую создать 4 пункта, int activeMenuItem = 0, переменная для “активного” пункта меню, int ch=0 хранит нажатие клавиши, bool exit = false для выхода из цикла меню игры.

6)Далее начинается цикл while(!exit).

7)Внутри цикла вызываем функцию showMenu(), в которой написано меню, при помощи вызова нескольких команд cout, а перед этим отчистка консольного окна system(“cls”).

8)Далее вызов функции gotoxy(0, activeMenuItem). Данная функция описана выше. В параметрах передали два значения (по оси X и Y). По оси X курсор у нас всегда находится в самом первом ряду – координаты которого равны 0, а значение Y изменяется в зависимости от выбора пункта меню пользователем.

9) ch=\_getch(); При помощи данной функции присваиваем переменной ch значение кода нашей клавиши. При нажатии стрелок сначала идёт 224 и потом код клавиши стрелки, поэтому далее идет проверка условия, которая описана в п.10

10) Если ch==224 , то записывается второй код клавиши. Т.к. в буфер присылаются два кода, один из них – 224.

11)Конструкция switch(ch), которая обрабатывает значение нажатой клавиши, подробнее описывается в следующих пунктах.

12)Если код клавиши == 27, то exit=true //при нажатии клавиши esc выход из консоли

13) Если код клавиши == 72, то activeMenuItem-- //при нажатии стрелки вверх

14) Если код клавиши ==80, то activeMenuItem++ //при нажатии стрелки вниз

15) Если код клавиши == 13, то это нажатие клавиши enter. Внутри данного “кейса” мы создаём несколько if-ов. Подробнее в следующих пунктах.

16)если activeMenuItem==0 //выбор первого варианта меню – new game, то п.17, иначе переходим к п.62

17)Вызываем функцию setup();

18)Функция setup содержит значение переменной gameOver = false; dir = STOP, начальное положение координаты нашей змейки(по X и Y): это высота и ширина деленная на два, при помощи вызова функции rand(), переменным fruitX и fruity присваивается случайные значения в пределах нашего игрового поля. Как можно догадаться, это будущие объекты, которые наша змейка будет “съедать”, score = 0.

19) Далее в main создаем цикл while(!gameOver).

20)Внутри данного цикла вызываем функцию draw ();

21)Функция draw содержит команды, которые позволяют нам создавать видимые границы игрового поля и непосредственно змейку и фрукты. Подробнее в следующих пунктах.

22) Очищаем терминал окна консоли system(“cls”).

23)Рисуем верхнюю границу поля при помощи цикла for, который выполняется до тех пор, пока локальная переменная не станет равной значению width+1. В качестве границы мы будем использовать октоторп. Который мы будем выводить командной cout.

24) Рисуем боковые границы. В цикле for(int i; i<height;i++) мы создаём вложенный цикл for(int i; i<width; i++). Внутри этих циклов создаём несколько проверок условий.

25) Если (j==0 || j == width-1), то будет выводиться #, при помощи команды cout, иначе переходим к п.26

26) Если (j==y && j == x), то будет выводиться ‘0’, - это тело нашей змейки, иначе переходим к п.27.

27) Иначе если (i==fruitY && j == fruitX), то будет выводиться ‘\*’, это наши “фрукты”, иначе переходим к п.28.

28) Иначе создаем локальную переменную bool print = false. Создаём еще один вложенный цикл, который будет выполнятся пока k<nTail.

29) Внутри цикла у нас проверка условия: если(tailX[k]==j && tailY[k]==i), то print = true и выводим при помощи cout символ ‘о’, который будет служить у нас в качестве хвоста нашей змейки.

30)Если(!print), то выводим пробел.

31)Выход из основного цикла.

32)Создание еще одного цикла, в котором у нас будет выводиться наш текущий результат при помощи переменной score и нижняя граница игровой зоны. Выполнятся будет до тех пор, пока локальная переменная не станет меньше ширины нашего поля. Заполняться будет так же символом – ‘#’.

33) Команда cout с подсказкой, что выйти из игры можно, нажав клавишу ‘x’.

34) Для того, что бы змейка не передвигалась с большой скоростью по полю, вызываем функцию Sleep(100);, которая сделает задержку в 1/10с.

35) Далее в основной (main) функции вызываем функцию input ();

36) В функции input() мы создаём конструкцию if(\_kbhhit()). Функция “внутри” if-а отслеживает нажатия пользователя, не закрывая терминал. Возвращает true, если пользователь нажал кнопку на клавиатуре.

37) Если пользователь нажал клавишу ‘a’, то dir = LEFT; //При нажатии клавиши ‘a’, переменной dir присвоится значение LEFT. В последующих пунктах аналогично.

38) Если пользователь нажал клавишу ‘d’, то dir = RIGHT;

39) Если пользователь нажал клавишу ‘w’, то dir = UP;

40) Если пользователь нажал клавишу ‘s’, то dir = DOWN;

41) Если пользователь нажал клавишу ‘x’, то gameOver = true; //при нажатии на клавишу ‘x’ выход из игры.

42) В основной функции main вызываем функцию logic();

43) В функции logic проработана вся основная логика нашей игры. Создаём несколько новых переменных для хвоста нашей змейки. Int prevX=tailX[0]; int prevY=tailY[0], int prev2x, prev2y.

44)tailX[0]=x, tailY[0]=y; Нулевым элементам массива присваиваем значение X и Y.

45) Создаём цикл for(int i = 1; i<nTail; i++).

46) Внутри цикла настраиваем “поведение” нашего хвоста.

47)Создаём конструкцию switch(dir).

48)При нажатии клавиши LEFT: x--; т.е. наша змейка будет сдвигаться по оси x влево, поэтому используем дикремент и уменьшаем значение переменной X. Далее аналогично.

49) При нажатии клавиши RIGHT: x++;

50) При нажатии клавиши UP: y--;

51) При нажатии клавиши DOWN: y++;

52)Далее создаём несколько условий для того, чтобы наша змейка при “пересечении” игровой границы, появлялась с другой стороны.

53) Если (x>=width-1), то переменной x присвоится значение 0, иначе переход к п.54.

54) Иначе если (x<0), то переменной x присвоится значение width-2, иначе переход к п.55.

55) Если (y>=height), то переменной y присвоится значение 0, иначе переход к п.56.

56) Иначе если (y<0), то переменной y присвоится значение height-1, иначе переход к п.57.

57) Создаём цикл for, который будет выполняться до того, пока i<nTail. Внутри цикла содержится условие, в котором описывается, что произойдет, если змейка “съест” свой хвост.

58)Если(tailX[i]==x && tailY[i]==y), то gameOver = true, т.е. осуществится выход из игры. Далее происходит вызов функции showLose().

59)В функции showLose() у нас происходит вызов функции gotoxy(), для задания координатов нашей надписи, которая выводится при помощи команды cout и содержит надпись “Вы проиграли”. Далее вызов функции Sleep(1000), которая позволит в течении секунды демонстрировать данную надпись в консоли.

60) В функции logic() создаём еще одно условие, в котором будет описано, что происходит, когда змейка съедает фрукт.

61) Если (x == fruitX && y == fruitY), то к переменной score добавляется 10, координаты для фрукта заново генерируются (как это было в пункте 18), значение переменной nTail увеличивается на один.

62) В функции main переходим к следующему условию. Иначе если (activeMenuItem == 1), то на экран выводится инструкция к игре. Отчищаем консоль system(“cls”) и выводим при помощи нескольких cout-ов инструкцию на экран консоли, иначе переход к п.63.

63) Иначе если (activeMenuItem == 2), то вывод информации про разработчика, иначе переход к п.64.

64) Иначе если(activeMenuItem == 3), то выполняется выход из консоли. Внутри условия: exit = true.

65) Далее при помощи условий cоздаём ограничения перехода по пунктам меню, чтобы пользователь переключался только между пунктами меню.

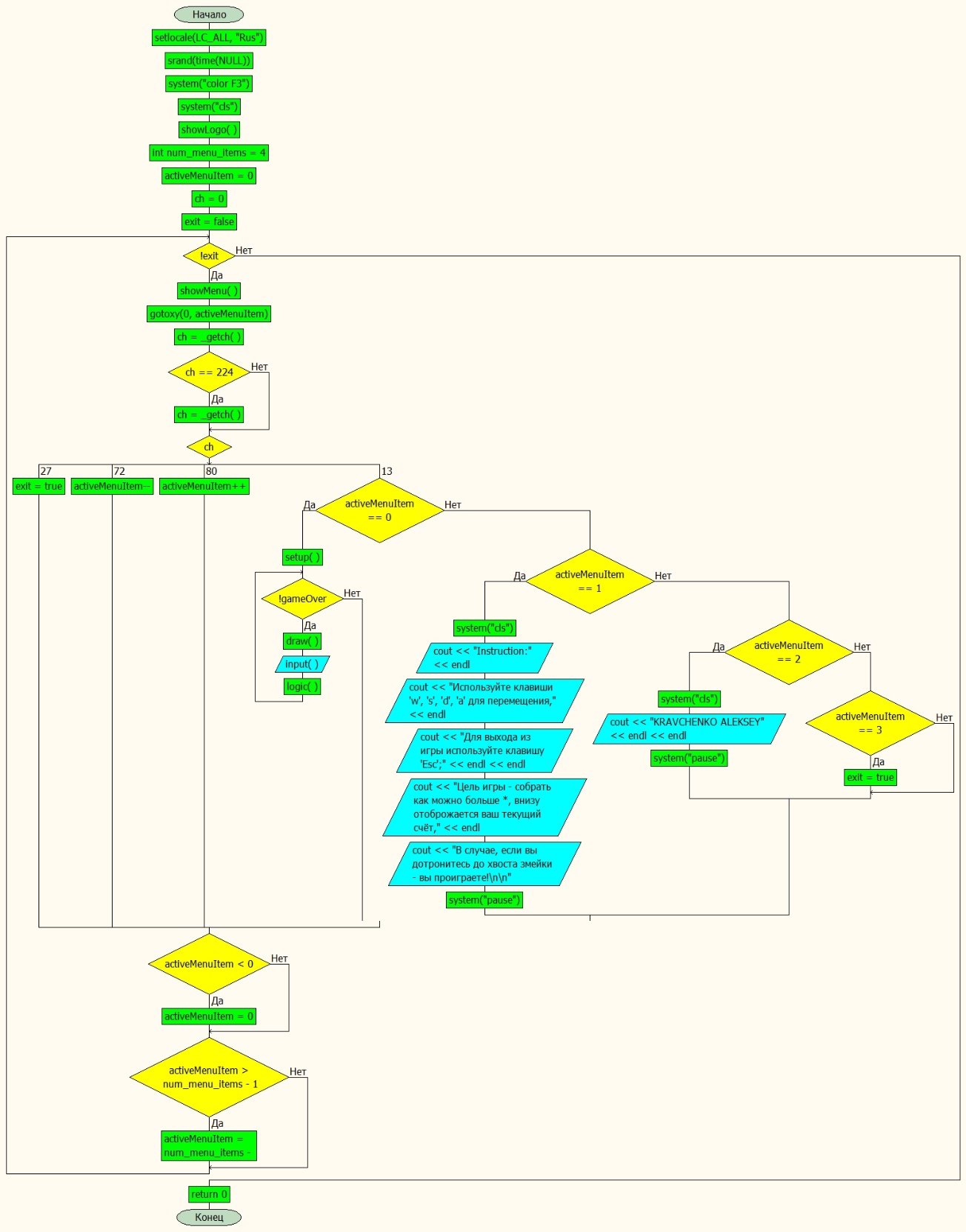
66) Если (activeMenuItem<0), то переменной activeMenuItem присваивается значение 0.

67) Если (activeMenuItem > num\_menu\_items-1), то activeMenuItem = num\_menu\_items - 1;

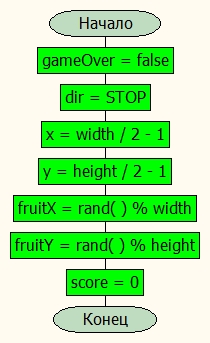
68)Конец программы.

**3.2. Блок-схема.**

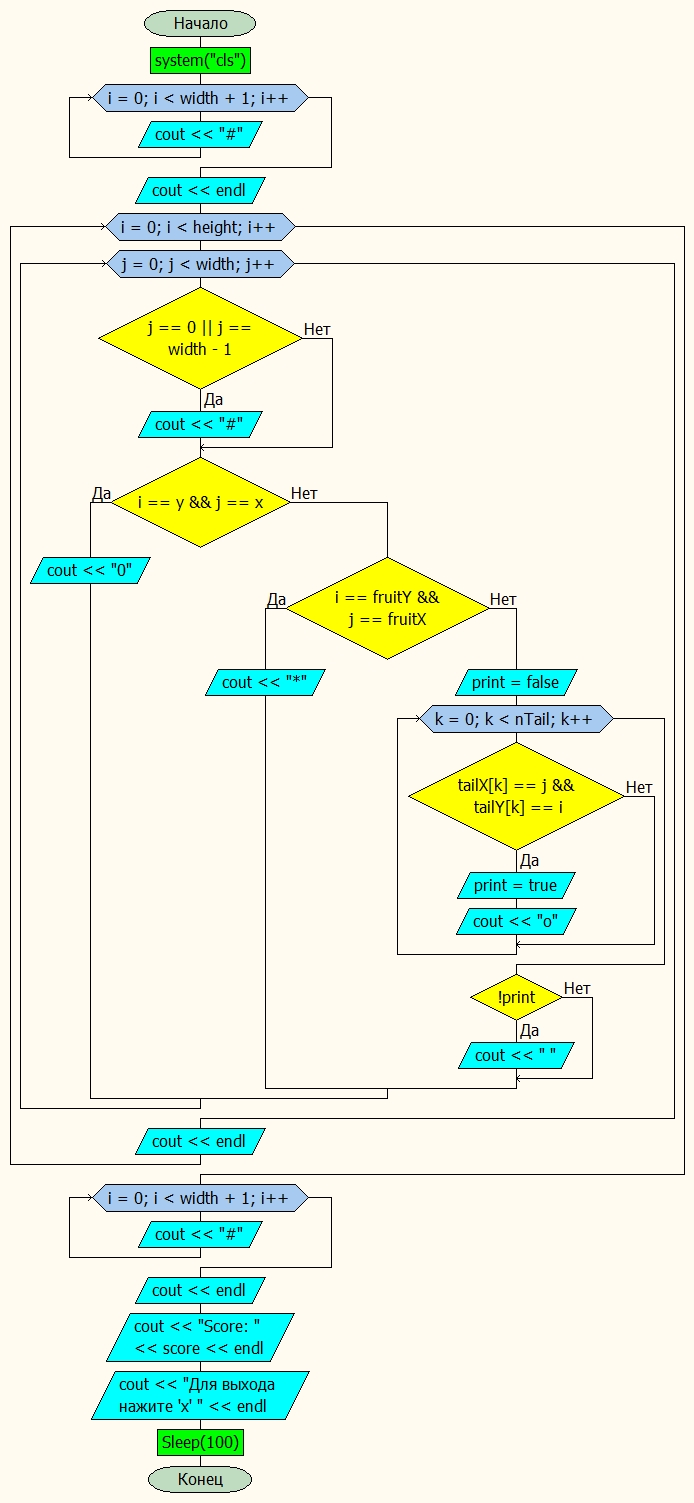
**3.2.1 Функция main.**



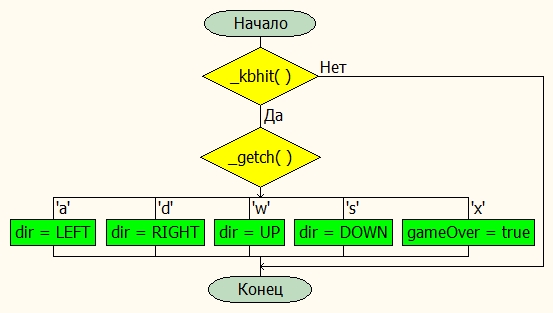
**3.2.2. Функция setup.**



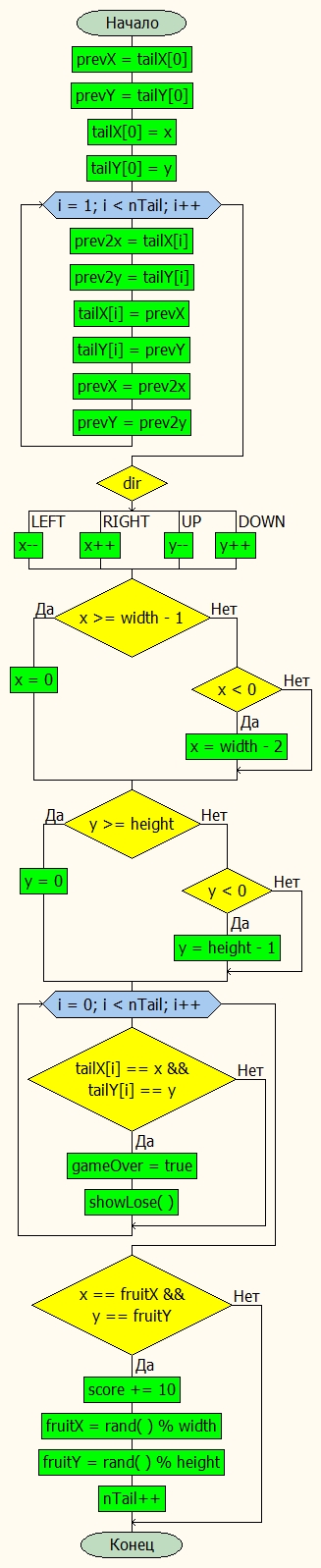
**3.2.3 Функция draw.**



**3.2.4. Функция input.**



**3.2.5. Функция logic**



**Заключение:**

В ходе выполнения данной работы, я познакомился и освоил некоторые команды ЯП С/С++. Научился создавать меню, изменять цвет в консоли, создавать динамику в 2D пространстве. Пользоваться функциями отчистки консольного окна, функцией отслеживания нажатия клавиш на клавиатуре, функцией для задания координат и др. При помощи ASCII-символов нарисовали графические объекты.

При выполнении программы сделали следующие **выводы**:

1. Для постоянного вызова функций удобнее использовать цикл while;
2. Чтобы код легко читался, необходимо следовать рекомендациям по хорошему стилю программирования;
3. Модульное программирование, создание дополнительных функций -повышает эффективность нашего кода и программы;
4. При планировании задачи, лучше всего заранее строить алгоритм, после чего, приступать непосредственно к работе.