Задание: разработать игру «змейку».

В данном примере мы с вами разберем как, написать игру змейка с использованием классов и функций.

Для этого нам понадобиться немного теории и Visual Studio с установленными пакетами для разработки консольных приложений .NET Framework с использованием языка программирования C#.

Скачать можно с официального сайта [Microsoft](https://visualstudio.microsoft.com/ru/vs/).

Ну что ж, приступим, для создания змейки нам понадобиться, знание:

1. синтаксиса языка программирование C#
2. Умение определять, создавать, объявлять переменные разных типов
3. Создавать классы и работать с ними

В конце этого документа будет предоставлено задание для самостоятельной работы.

Синтаксис языка программирование C# основан на идентификаторах и классах.

**Идентификатор** - это имя, используемое для идентификации класса, переменной, функции или любого другого пользовательского элемента. Основные правила для классов именования в C # заключаются в следующем:

* Имя должно начинаться с буквы, за которой может следовать последовательность букв, цифр (0-9) или подчеркивание. Первый символ в идентификаторе не может быть цифрой.
* Он не должен содержать какое-либо встроенное пространство или символ, например? - +! @ #% ^ & \* () [] {}. ; : "'и \. Однако можно использовать символ подчеркивания (\_).
* Это не должно быть ключевое слово C #.

**Ключевые слова** - зарезервированные слова, предопределенные компилятору C #. Эти ключевые слова не могут использоваться в качестве идентификаторов. Однако, если вы хотите использовать эти ключевые слова в качестве идентификаторов, вы можете префикс ключевого слова с помощью символа @.

В C # некоторые идентификаторы имеют особое значение в контексте кода, такие как get и set, называются контекстуальными ключевыми словами.

В следующей таблице перечислены зарезервированные ключевые слова и контекстные ключевые слова в C #

**Зарезервированные ключевые слова**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| abstract | as | base | bool | break | byte | case |
| catch | char | checked | class | const | continue | decimal |
| default | delegate | do | double | else | enum | event |
| explicit | extern | false | finally | fixed | float | for |
| foreach | goto | if | implicit | in | in (generic modifier) | int |
| interface | internal | is | lock | long | namespace | new |
| null | object | operator | out | out (generic modifier) | override | params |
| private | protected | public | readonly | ref | return | sbyte |
| sealed | short | sizeof | stackalloc | static | string | struct |
| switch | this | throw | true | try | typeof | uint |
| ulong | unchecked | unsafe | ushort | using | virtual | void |
| volatile | while |  |  |  |  |  |

**Классы** являются основным типом в языке C#. Класс представляет собой структуру данных, которая объединяет в себе значения (поля) и действия (методы и другие функции-члены). Класс предоставляет определение для динамически создаваемых экземпляров класса, которые также именуются объектами. Классы поддерживают механизмы наследования и полиморфизма, которые позволяют создавать производные классы, расширяющие и уточняющие определения базовых классов.

Новые классы создаются с помощью объявлений классов. Объявление класса начинается с заголовка, в котором указаны атрибуты и модификаторы класса, имя класса, базовый класс (если есть) и интерфейсы, реализуемые этим классом. За заголовком между разделителями { и } следует тело класса, в котором последовательно объявляются все члены класса.

Следующий код является простым примером объявления класса с именем Point:

public class Point

{

public int x, y;

public Point(int x, int y)

{

this.x = x;

this.y = y;

}

}

Экземпляры классов создаются с помощью оператора **new**, который выделяет память для нового экземпляра, вызывает конструктор для инициализации этого экземпляра и возвращает ссылку на экземпляр. Следующие инструкции создают два объекта Point и сохраняют ссылки на них в две переменные:

Point p1 = new Point(0, 0);

Point p2 = new Point(10, 20);

Приступим к работе.

Для начала нам нужно создать само поле и сделать так, что бы оно было квадратным для этого создаем

int n = 10;

int m = 10;

где n и m – размерность двумерного массива в котором будет бегать наша змейка

Далее создадим переменные для помещения фрукта на «стол»

int xz = 0, yz = 0, xv = 0, yv = 0, size = 1;

ConsoleKeyInfo f;

string move = "";

bool eat = false;

String[,] map = new string[n, m];

int[] masx = new int[size];

int[] masy = new int[size];

Тут же определим объект класса ConsoleKeyInfo для считывания информации клавиши(управление) определим что наша «змейка» еще ничего не съела, создадим поле с параметрами 10\*10 и создадим два массива по x и y с размером 1.

Далее объявим переменную типа bool для того, что бы машина понимала, когда игра начинается, а когда заканчивается

bool game = true;

Приготовления для змейки закончены, теперь пришло время перейти к классам и начать с ними работать.

Создаем класс **Game** и в нем создаем функцию Otrisovka

public static void Otrisovka(int n, int m, ref int xz, ref int yz, ref int yv, ref int xv, ref string[,] map) – эта функция нужна для того что бы мы могли создать поле для змейки, в нем нужно прописать

Random rand = new Random(); // функцию случайной генерации с обьектом rand

for (int i = 0; i < n; i++) // от рисовать границы поля

{

for (int j = 0; j < m; j++)

{

map[i, j] = " ";

if (i == 0 || j == 0 || i == n - 1 || j == m - 1)

map[i, j] = "#";

}

}

yz = n - 2; xz = 1;

map[yz, xz] = "X"; // поставить на поле «голову» змеи

while (true) // и создать в нем «пищу»

{

xv = rand.Next(1, m - 2);

yv = rand.Next(1, n - 2);

if (xv == xz && yv != yz || xv != xz && yv == yz || xv != xz && yv != yz)

{

break;

}

}

map[yv, xv] = "\*";

После создания поля, нам необходимо заставить змейку двигаться, для этого давайте вернемся в класс **Programm** и там создадим управление для змейки

if (Console.KeyAvailable == true)

{

f = Console.ReadKey();

if (f.Key == ConsoleKey.UpArrow)

if (move != "Down")

move = "Up";

if (f.Key == ConsoleKey.DownArrow)

if (move != "Up")

move = "Down";

if (f.Key == ConsoleKey.RightArrow)

if (move != "Left")

move = "Right";

if (f.Key == ConsoleKey.LeftArrow)

if (move != "Right")

move = "Left";

}

Такое создание движение сработает только 1 раз, и отреагирует только на 1 нажатие, заметьте, что в нажатиях есть еще 1 обработчик исключений, т.к мы не хотим что бы наша змейка съела сама себя или начала двигаться хвостом вперед(змеи не умеют пятиться) по этому мы создадим бесконечный цикл, для этого управления.

while (game == true)

{

Console.WriteLine();

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

Console.Write(map[i, j]);

}

Console.WriteLine();

}

if (Console.KeyAvailable == true)

{

f = Console.ReadKey();

if (f.Key == ConsoleKey.UpArrow)

if (move != "Down")

move = "Up";

if (f.Key == ConsoleKey.DownArrow)

if (move != "Up")

move = "Down";

if (f.Key == ConsoleKey.RightArrow)

if (move != "Left")

move = "Right";

if (f.Key == ConsoleKey.LeftArrow)

if (move != "Right")

move = "Left";

}

}

Вот вроде бы управление и оно бесконечное, готово для нашей игры, но при нажатии на клавиши ничего не меняется, следовательно нам нужно как то изменить направление движения, для этого вернемся в класс **Game** и создадим там функцию, где ключевыми параметрами для нас будут xz,yz, map и строка move. По строке move мы и будем менять направление движения змейки с изменением ее координат в массиве, а что бы змейка не дублировалась, пустые части, где наша змейка кончается, заменяем на пустые клетки.

public static void Move(int n, int m, ref int xz, ref int yz, string move, ref string[,] map)

{

map[yz, xz] = " ";

switch (move)

{

case "Up":

if (yz - 1 >= 0) yz -= 1;

break;

case "Down":

if (yz + 1 < n) yz += 1;

break;

case "Left":

if (xz - 1 >= 0) xz -= 1;

break;

case "Right":

if (xz + 1 < m) xz += 1;

break;

}

map[yz, xz] = "X";

}

Теперь необходимо вызвать эту самую функцию, вернемся в класс Programm и вызовем функцию перемещения

Game.Move(n, m, ref xz, ref yz, move, ref map);

Теперь наша змейка может передвигаться, НО ей нечего кушать и по этому давайте создадим ей порционный генератор еды.

Вернемся в класс **Game** где мы создаем функции для нашей игры и создадим там функцию по «сотворению еды»

public static void EatGenerate(int n, int m, ref string[,] map, ref int xv, ref int yv, int xz, int yz, ref bool eat, int[] masx, int[] masy)

{

if (xv == xz && yv == yz)

{

eat = true;

Random rand = new Random();

bool s = true;

while (s == true)

{

xv = rand.Next(1, m - 2);

yv = rand.Next(1, n - 2);

int p = 0;

if (xv != xz && yv == yz || yv != yz && xv == xz || xv != xz && yv != yz)

{

for (int i = 0; i < masx.Length; i++)

{

if (xv != masx[i] && yv == masy[i] || yv != masy[i] && xv == masx[i] || xv != masx[i] && yv != masy[i])

p++;

if (p == masx.Length)

s = false;

}

}

}

map[yv, xv] = "\*";

}

}

Попробуйте сами разобраться в этой функции, и напиши ответ в отчете о работе или добавьте комментарии при составлении отчета.

После того как мы создали функцию по поставке еды для нашего питомца, мы должны заставить его работать, вернемся в класс **Programm** и вызовем функцию

Game.EatGenerate(n, m, ref map, ref xv, ref yv, xz, yz, ref eat, masx, masy);

Вроде бы все хорошо, но если хорошо присмотреться, то в коде нехватает движения всей змейки целиком, наша голова передвигается, а вот остальное тело нет, нужно это исправить, давайте создадим для этого функцию в классе **Game**

public static void Tail(ref int[] masx, ref int[] masy, ref string[,] map, ref bool eat, ref int size, int xz, int yz)

{

if (eat == true)

{

size++;

Array.Resize<int>(ref masx, size);

Array.Resize<int>(ref masy, size);

eat = false;

}

map[masy[size - 1], masx[size - 1]] = " ";

for (int i = masx.Length - 1; i > 0; i--)

{

masx[i] = masx[i - 1];

masy[i] = masy[i - 1];

}

masx[0] = xz;

masy[0] = yz;

for (int i = 0; i < size; i++)

{

map[masy[i], masx[i]] = "X";

}

}

В этой функции мы обрабатываем изменение размеров змейки при съедании фрукта, с помощью изменение размеров массива который представляет питомца. После этого заполняем массив заново и обрисовываем его.

Давайте вызовем нашу функцию для отрисовки хвоста змейки.

Game.Tail(ref masx, ref masy, ref map, ref eat, ref size, xz, yz);

Поздравляю ваша игра почти готова, ей не хватает только одного. Любая игра должна закончиться победой или поражением. В нашем случае игра бесконечная. Давайте сделаем так, что бы игра имела конец, условиями поражения будут встреча змейки с частью себя, если змейка врезается в ограждение, условием победы будет только одно: заполнить все поле змейкой.

Вернемся в наш с вами знакомый класс **Game** и создадим там две функции поражение и победы.

Условие поражения:

public static void Lose(int n, int m, int[] masx, int[] masy, int xz, int yz, ref bool game, int size)

{

if (xz == 0 || xz == m - 1 || yz == 0 || yz == n - 1)

{

game = false;

Console.WriteLine("You LOSE!!!");

}

for (int i = 1; i < size; i++)

{

if (masx[i] == xz && masy[i] == yz)

{

game = false;

Console.WriteLine("You LOSE!!!");

}

}

}

Условие победы:

public static void Win(int n, int m, string[,] map, ref bool game)

{

int p = 0;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

if (i != 0 || j != 0 || i != n - 1 || j != m - 1)

{

if (map[i, j] == "X")

{

p++;

}

}

}

}

if (p == map.Length - n \* 2 - m \* 2 + 4)

{

game = false;

Console.WriteLine("You WIN!!!");

}

}

Теперь вызовем эти функции друг за другом в классе **Programm**

**Если** вы все правильно сделали, то теперь вы можете сыграть в эту замечательную игру.

**Если** вы конечно, можете реагировать и нажимать клавиши на очень большой скорости. Играю в эту змейку почти невозможно среагировать и вовремя нажать клавишу, потому что цикл **while** выполняется очень и очень быстро, и нужно затормозить его выполнение.

Для этого воспользуемся таким средством как остановка работы процессора(Sleep)

System.Threading.Thread.Sleep(400);

Эту часть кода нужно прописать в самом конце цикла **While** и теперь вы можете наслаждаться своей прекрасной игрой.

Задания для самостоятельно доработки игры «Змейка»

Функциональные требования:

- Управление змейкой должно происходить по клавишам (вверх, вниз, влево, вправо).

- Змейка должна непрерывно двигаться в сторону, куда нажал пользователь, по нажатии на другую клавишу, змейка должна поменять свое движение в сторону нажатой клавиши

- Скорость и размер поля должен меняться в зависимости от выбранной пользователем сложности игры

легко[поле10х10, скорость медленная]

нормально[поле15х15,скорость нормальная]

сложно[после20х20,скорость быстрая]

Требования к оформлению:

- Поле должно быть очерченным и понятным для пользователя

- Змейка должна обладать головой и телом (символы ASCII)

- «Фрукты» должны так же иметь свои собственные символы (ASCII)

Требования к выполнению работы(общ)

- Работа должна быть выполнена на функциях и классах

- Поле и змейка не должны «мерцать» при движении

Требования к отчету:

- Файл с отчетом должен содержать: титульный лист, задание, результат работы (скриншоты), листинг программы с подробными комментариями, файл Visio Studio с блок-схемой работы программы.