# Объекты

Объект представляет собой цельную сущность.

Объект описывает состояние и поведение этой сущности:

Поля и свойства объекта – это состояние этого объекта;

Методы – это действия этого объекта.

Рассмотрим просто пример:

|  |
| --- |
| class Player  {  public string Name { get; private set; }  public int HP { get; private set; }  public int Mana { get; private set; }  public int Strength { get; private set; }    public void Shownfo()  {  Console.WriteLine($"name {Name}");  }  } |

Это класс Player который описывает объект какого-то игрока. В самом простом случае он хранит имя и какую-то информацию об этом игроке. Но сейчас это не является цельной сущностью, по сути объект просто хранит информацию и всё и не умеет больше ничего, ни ходить, ни атаковать, ни обороняться (конечно бывают случаи, когда объекты создаются только для того что бы хранить информацию, но даже они как правило что-то умеют делать, к примеру: получать информацию и выводить её).

Давайте дополним нашего игрока парой базовых функций:

|  |
| --- |
| class Player  {  public string Name { get; private set; }  public int HP { get; private set; }  public int Mana { get; private set; }  public int Strength { get; private set; }  public void Attack(Enemy enemy)  {  enemy.HP -= Strength;  }  public void Move(Direction direction, int howMuch)  {  }  public void Defend()  {  }  public void Rest()  {  }  public void Shownfo()  {  Console.WriteLine($"name {Name}");  }  } |

|  |
| --- |
| enum Direction  {  forward,  back,  left,  right  } |

Пока оставим реализацию самих методов, и посмотрим на то, что у нас теперь умеет игрок:

Он умеет атаковвать врага, он умеет ходить в указаном направлении на указанное количество шагов, он умеет отдыхать и наконец он умеет защищаться.

Но в данной реализации, его не могут атаковать, потому что его поле здоровья приватное, а других методов для уменьшения здоровья у него нет.

Давайте исправим это:

|  |
| --- |
| class Player  {  private int baseHp = 100;  private int hp;  private double armor;  private bool defend = false;  public bool IsAlive { get; private set; } = true;  public string Name { get; private set; }  public int Mana { get; private set; }  public int Strength { get; private set; }  public double Armor  {  get  {  if (defend)  {  return armor \* 1.2;  }  return armor;  }  private set  {  armor = value;  }  }  public int HP  {  get  {  return hp;  }  set  {  if (value > Armor)  {  hp = value - (int)Armor;  }  if (hp < 0)  {  IsAlive = false;  }  }  }  public void Attack(Enemy enemy)  {  enemy.HP -= Strength;  }  public void Move(Direction direction, int howMuch)  {  }  public void Defend()  {  defend = true;  }  public void Rest()  {  HP += baseHp / 10;  }  public void Shownfo()  {  Console.WriteLine($"name {Name}");  }  } |

Мы добавили поле базового здоровья, свойство Брони, свойство живой ли игрок.

Обратите внимание, что все возможные действия, которые может совершать объект описаны именно внутри самого объекта! Объект ни как не зависит от внешних условий, всё что может влиять на состояние объекта, находится внутри объекта. Чтобы добавить возможные позитивные и негативные состояния на игрока (объект), можно добавить массив возможных способностей и воздействий.

Конечно класс на данный момент далеко не полный, но он описывает концепцию того, как должен быть построен объект и описывающий его класс.

Давайте напишем простую консольную игру.

У нас будет игрок, он будет ходить по лабиринту и уметь сражаться с противниками.

Для начала нарисуем нашего героя: для этого воспользуемся ASCII символами и для героя выберем символ под номером 2: ☻

static void Main(string[] args)

{

Console.Write((char)2);

}

Теперь на экране у нас появился символ нашего героя.

Следующая задача научить нашего героя передвигаться! Для передвижения, нам необходимо «удалить» его со старого места и нарисовать на новом.

Чтобы удалить: необходимо передвинуть курсор на место, где находится в данный момент наш «герой» и поставить туда пустой символ.

static void Main(string[] args)

{

int x = 0, y = 0;

Console.SetCursorPosition(x, y);

Console.Write((char)2);

Console.SetCursorPosition(x, y);

Console.Write(' ');

Console.Write((char)2);

Console.Read();

}

Вот таким простым способом мы научились передвигать нашего героя. Теперь надо научить его двигаться в разных направлениях и главное в том направлении, в котором хотим мы. Для этого вынесем передвижение героя в отдельный метод, а запрос данных с клавиатуры в цикл.

Обратите внимание, что координаты в консоли начинаются с левого верхнего угла и движение вверх, это минус от текущей координаты, движение вниз плюс к текущей координате, движение вправо это плюс, движение влево минус. Т.е. координаты 0,0 соответствуют верхнему левому углу.

class Program

{

static int x = 0;

static int y = 0;

static void Main(string[] args)

{

Console.SetCursorPosition(x, y);

Console.Write((char)2);

while(true)

{

int direction = 0;

if (Console.KeyAvailable)

{

ConsoleKeyInfo key = Console.ReadKey();

if (key.Key == ConsoleKey.LeftArrow)

direction = 0;

else if (key.Key == ConsoleKey.RightArrow)

direction = 1;

else if (key.Key == ConsoleKey.UpArrow)

direction = 2;

else if (key.Key == ConsoleKey.DownArrow)

direction = 3;

MoveHero(1, direction);

}

}

Console.Read();

}

static void MoveHero(int offset, int direction)

{

int nextX = x;

int nextY = y;

switch (direction)

{

case 0:

nextX -= offset;

break;

case 1:

nextX += offset;

break;

case 2:

nextY -= offset;

break;

case 3:

nextY += offset;

break;

default:

break;

}

Console.SetCursorPosition(x, y);

Console.Write(' ');

x = nextX;

y = nextY;

Console.SetCursorPosition(x, y);

Console.Write((char)2);

}

}

Теперь в главном цикле мы считываем клавиши с консоли (стрелки) и двигаем героя в зависимости от того, что мы нажали в методе MoveHero(). Но есть пара проблем, к примеру мы можем выйти за пределы карты и вторая проблема это передача в метод MoveHero() направления с помощью переменной int.

Чтобы исправить вторую проблемы, нам всего лишь надо ввести перечисление Direction:

enum Direction

{

left,

right,

up,

down

}

class Program

{

static int x = 0;

static int y = 0;

static void Main(string[] args)

{

Console.SetCursorPosition(x, y);

Console.Write((char)2);

while(true)

{

Direction direction = 0;

if (Console.KeyAvailable)

{

ConsoleKeyInfo key = Console.ReadKey();

if (key.Key == ConsoleKey.LeftArrow)

direction = Direction.left;

else if (key.Key == ConsoleKey.RightArrow)

direction = Direction.right;

else if (key.Key == ConsoleKey.UpArrow)

direction = Direction.up;

else if (key.Key == ConsoleKey.DownArrow)

direction = Direction.down;

MoveHero(1, direction);

}

}

Console.Read();

}

static void MoveHero(int offset, Direction direction)

{

int nextX = x;

int nextY = y;

switch (direction)

{

case Direction.left:

nextX -= offset;

break;

case Direction.right:

nextX += offset;

break;

case Direction.up:

nextY -= offset;

break;

case Direction.down:

nextY += offset;

break;

default:

break;

}

Console.SetCursorPosition(x, y);

Console.Write(' ');

x = nextX;

y = nextY;

Console.SetCursorPosition(x, y);

Console.Write((char)2);

}

}

Чтобы исправить первую проблему, нам придётся ввести стены! Пока нарисуем только габарит. Для стен мы будем использовать массив. В главную очередь потому что его в последствии удобно редактировать для рисования карты. И так начнём, наша первая карта выглядит так:

readonly char[,] levelData = {

{ '#', '#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','h',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', '#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#' }

};

Героя мы обозначим буквой h

Это двумерный массив символов, каждая ячейка которого что-то представляет, в данном случае в нём содержится три вида ячеек: герой, стена и пустая ячейка. Данный массив является readonly, что означает, что его нельзя изменять, он будет служить нам «шаблоном» начального запуска. Но для работы с игрой нам понадобиться другой массив, который будет необходимо заполнить для взаимодействия игрового процесса через него.

Давайте определим, как будут выглядеть наши символы:

Hero - ☻ (ASCII символ 2)

Wall - ▒ (ASCII символ 177)

C# работает с юникодом, а таблица символов ACII заканчивается на 127 символе, всё что выше, это расширенная таблица символов ASCII и она идёт в разрез с юникодом, поэтому мы будем использовать явную кодировку для отображения символов на экране.

static int heroRow = 0;

static int heroColumn = 0;

const int rowCounts = 15;

const int columnCounts = 35;

static readonly char heroSymbol = EncodingChar(2);

static readonly char wallSymbol = EncodingChar(177);

static char[,] levelData = new char[rowCounts, columnCounts];

static private char EncodingChar(byte numberSym)

{

Encoding encoder = Encoding.GetEncoding(437);

byte[] sym = { numberSym };

var symbol = encoder.GetString(sym)[0];

return symbol;

}

static void Initialize()

{

for (int row = 0; row < rowCounts; row++)

{

for (int column = 0; column < columnCounts; column++)

{

char symbol = levelDataInit[row, column];

switch (symbol)

{

case '#':

levelData[row, column] = wallSymbol;

break;

case 'h':

heroRow = row;

heroColumn = column;

levelData[row, column] = heroSymbol;

break;

default:

levelData[row, column] = symbol;

break;

}

}

}

}

static void Render()

{

Console.SetCursorPosition(0,0);

for (int row = 0; row < rowCounts; row++)

{

for (int column = 0; column < columnCounts; column++)

{

Console.Write(levelData[row, column]);

}

Console.WriteLine();

}

}

Мы добавили три новых метода: Initialize(), Render(), EncodingChar()

Последний нам нужен только для декодирования символов.

Метод инициализации нужен для первичного построения игрового массива, рендер будет работать в главном цикле нашей программы и каждый раз обновлять игровое поле.

Теперь надо немного поменять метод MoveHero()

static void MoveHero(int offset, Direction direction)

{

int nextRow = heroRow;

int nextColumn = heroColumn;

switch (direction)

{

case Direction.left:

nextColumn -= offset;

break;

case Direction.right:

nextColumn += offset;

break;

case Direction.up:

nextRow -= offset;

break;

case Direction.down:

nextRow += offset;

break;

default:

break;

}

levelData[heroRow, heroColumn] = ' ';

heroRow = nextRow;

heroColumn = nextColumn;

levelData[heroRow, heroColumn] = heroSymbol;

}

Теперь мы не работаем с консолью, мы работаем только с массивом.

Давайте добавим их в метод Main()

static void Main(string[] args)

{

Initialize();

while (true)

{

Render();

Update();

}

Console.Read();

}

public static void Update()

{

Console.CursorVisible = false; // чтобы не мерцал экран

Direction direction = 0;

if (Console.KeyAvailable)

{

ConsoleKeyInfo key = Console.ReadKey();

if (key.Key == ConsoleKey.LeftArrow)

direction = Direction.left;

else if (key.Key == ConsoleKey.RightArrow)

direction = Direction.right;

else if (key.Key == ConsoleKey.UpArrow)

direction = Direction.up;

else if (key.Key == ConsoleKey.DownArrow)

direction = Direction.down;

MoveHero(1, direction);

}

}

Движение героя мы вынесли в метод Update(), это будет основной метод действий нашей игры. Игру можно запустить, но наш герой всё равно проходит сквозь стены. Давайте теперь сделаем ограничение на проход сквозь стены.

static void MoveHero(int offset, Direction direction)

{

int nextRow = heroRow;

int nextColumn = heroColumn;

bool canMove = false;

switch (direction)

{

case Direction.left:

nextColumn -= offset;

break;

case Direction.right:

nextColumn += offset;

break;

case Direction.up:

nextRow -= offset;

break;

case Direction.down:

nextRow += offset;

break;

default:

break;

}

char nextSym = levelData[nextRow, nextColumn];

switch (nextSym)

{

case ' ':

canMove = true;

break;

default:

break;

}

if (canMove)

{

levelData[heroRow, heroColumn] = ' ';

heroRow = nextRow;

heroColumn = nextColumn;

levelData[heroRow, heroColumn] = heroSymbol;

}

}

Всё, теперь наш герой не умеет ходить сквозь стены!

Теперь самое время добавить врагов и оружие. Введём два типа врага: скелетов и орков.

Скелет: ☻ (ASCII символ 2) цвет белый

Орк: ☻ (ASCII символ 2) цвет зелёный

Кулак: Без символа, урон 2

Палка: (ASCII символ 47) цвет зелёный, урон 16

Дубинка: (ASCII символ 33) цвет зелёный, урон 24

Копьё: (ASCII символ 24) цвет зелёный, урон 32

Сабля: (ASCII символ 108) цвет зелёный, урон 40

А игрока сделаем жёлтым цветом.

static readonly char[,] levelDataInit = {

{ '#', '#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#' },

{ '#', ' ',' ',' ','#',' ','#','#',' ',' ',' ',' ',' ',' ','o',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','s',' ',' ',' ',' ',' ',' ',' ','#' },

{ '#', ' ','#',' ','o',' ',' ','#','o','#','#','#','#','#','#','#','#','#','#','#','#','#','#',' ','#','#','#','#','#','#','#','#','#',' ','#' },

{ '#', ' ','#','#','#','#','#','#',' ','#',' ',' ',' ','#',' ','#','3',' ',' ','s',' ',' ','#',' ','#','#','s','#','#',' ',' ',' ','#',' ','#' },

{ '#', ' ',' ','s',' ',' ',' ',' ',' ',' ',' ','#',' ','#',' ','#','#','#','#','#','#',' ','#',' ','#','s','4',' ',' ',' ','#','s','#',' ','#' },

{ '#', '#','#','#','#','#',' ','#',' ','#','#','#',' ','#',' ',' ',' ',' ',' ',' ','#',' ','#',' ','#','#','s','#','#',' ',' ',' ','#',' ','#' },

{ '#', '#',' ',' ','o','#',' ','#',' ','#',' ','#',' ','#','#','#','#',' ','#',' ','#',' ',' ',' ','#','#','#','#','#',' ','#',' ','#',' ','#' },

{ '#', ' ',' ','#',' ',' ',' ','#',' ','#',' ','o',' ',' ',' ',' ','#',' ','#','#',' ','#','#','#','#','#','#',' ','#',' ',' ',' ','#',' ','#' },

{ '#', '#','#','#','#','#','#','#',' ','#',' ','#','#','#','#',' ',' ',' ',' ',' ',' ',' ',' ',' ','#',' ',' ','s','#',' ','#',' ','#','s','#' },

{ '#', ' ',' ','2','#',' ',' ',' ',' ','#',' ',' ',' ',' ',' ',' ','#','#','#','#','#','#','#',' ','#',' ','#',' ','#',' ',' ',' ','#',' ','#' },

{ '#', ' ','#','#','#',' ','#','#','#','#',' ','#','#','#','#','#','#',' ',' ',' ','o',' ',' ',' ','#',' ','#',' ',' ',' ','#',' ',' ',' ','#' },

{ '#', ' ',' ','o',' ',' ',' ',' ',' ',' ',' ','#',' ',' ',' ',' ',' ',' ','#','#','#','#','#','#','#',' ','#','#','#','#','#','#','#','#','#' },

{ '#', '#','#','#','#','#','#','#','#','#','#','#',' ','#','#','#','#','#','#',' ',' ',' ','#','1','#',' ','#',' ','s',' ','#',' ',' ',' ','#' },

{ '#', 'h',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','#',' ',' ',' ','#',' ',' ',' ','#','s',' ',' ','#',' ','#' },

{ '#', '#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','#','e','#' }

};

Это наша новая карта.

Скелетов мы обозначили буквой s, а орков буквой о.

Но теперь у нас есть проблем: врагов много, а если мы хотим, чтобы они двигались, чтобы у каждого было своё здоровье, своё оружие?

Сейчас мы храним только две переменные и один метод для игрока (к ним добавляется цвет!), не создавать же для каждого существа на карте отдельные переменные? Это будет огромный код, к тому же очень одинаковый. Поэтому нам необходимо выделить всех существ в отдельные сущности, которые позволили бы нам достаточно просто работать с каждым из них.

Сделаем небольшую заготовку в новом файле Unit.cs

enum UnitType

{

None,

Hero,

Org,

Skeleton

}

enum WeaponType

{

None,

Fist,

Stick,

Club,

Spear,

Saber

}

class Weapon

{

public WeaponType Type { get; private set; }

public int Damage { get; private set; }

}

class Unit

{

public UnitType Type { get; private set; }

public char Symbol { get; private set; }

public int Row { get; private set; }

public int Column { get; private set; }

public int Health { get; private set; }

public Weapon Weapon { get; private set; }}

Мы создали классы Юнита (любого) и оружия. Класс Юнит хранит в себе тип юнита, его расположение, здоровье и оружие которое он носит.

В данный момент эти классы просто хранят в себе информацию удобным образом (что уже очень не мало и мы в принципе можем с этим работать), но всё же их надо доработать.

И ещё нам понадобиться вспомогательная структура, которая будет хранить в себе символы отображения и их цвет.

struct Symbols

{

public static readonly char emptySymbol = ' ';

public static readonly char heroSymbol = EncodingChar(2);

public static readonly char wallSymbol = EncodingChar(177);

public static readonly char skeletonSymbol = EncodingChar(2);

public static readonly char orgSymbol = EncodingChar(2);

public static readonly char fistChar = ' ';

public static readonly char stickChar = EncodingChar(47);

public static readonly char clubChar = EncodingChar(33);

public static readonly char spearChar = EncodingChar(24);

public static readonly char saberChar = EncodingChar(108);

public static readonly char heroMapSymbol = 'h';

public static readonly char orgMapSymbol = 'o';

public static readonly char skeletonMapSymbol = 's';

public static readonly char stickMapChar = '1';

public static readonly char clubMapChar = '2';

public static readonly char spearMapChar = '3';

public static readonly char saberMapChar = '4';

private static char EncodingChar(byte numberSym)

{

Encoding encoder = Encoding.GetEncoding(437);

byte[] sym = { numberSym };

var symbol = encoder.GetString(sym)[0];

return symbol;

}

public static char GetRenderSymbol(char symbol)

{

char sym = ' ';

switch (symbol)

{

case '#':

sym = wallSymbol;

break;

case 'h':

sym = heroSymbol;

break;

case 'o':

sym = orgSymbol;

break;

case 's':

sym = skeletonSymbol;

break;

case '1':

sym = stickChar;

break;

case '2':

sym = clubChar;

break;

case '3':

sym = spearChar;

break;

case '4':

sym = saberChar;

break;

default:

sym = symbol;

break;

}

return sym;

}

public static ConsoleColor GetRenderColor(char symbol)

{

ConsoleColor Color = ConsoleColor.White;

switch (symbol)

{

case '#':

Color = ConsoleColor.White;

break;

case 'h':

Color = ConsoleColor.Yellow;

break;

case 'o':

Color = ConsoleColor.Green;

break;

case 's':

Color = ConsoleColor.White;

break;

case '1':

Color = ConsoleColor.DarkYellow;

break;

case '2':

Color = ConsoleColor.DarkRed;

break;

case '3':

Color = ConsoleColor.DarkCyan;

break;

case '4':

Color = ConsoleColor.Cyan;

break;

default:

Color = ConsoleColor.Black;

break;

}

return Color;

}

public static char GetMapUnitSymbol(UnitType unit)

{

char unitChar = ' ';

switch (unit)

{

case UnitType.Hero:

unitChar = heroMapSymbol;

break;

case UnitType.Org:

unitChar = orgMapSymbol;

break;

case UnitType.Skeleton:

unitChar = skeletonMapSymbol;

break;

default:

unitChar = emptySymbol;

break;

}

return unitChar;

}

}

Так же нам придётся вынести игровое поле в отдельный вспомогательный класс, чтобы мы имели возможность работать с игровым полем.

В принципе мы уже можем отказаться от работы с символами, но оставим это на чуть позже, чтобы не затягивать ещё больше.

class Map

{

public static char[,] LevelData { get; set; }

private const int maxCountUnit = 35;

public static int UnitsCount { get; private set; }

public static Unit[] Units { get; private set; }

public static int HeroIndex { get; private set; }

public static void SetMap(int rowCounts, int columnCounts, char[,] levelDataInit)

{

Units = new Unit[maxCountUnit];

LevelData = new char[rowCounts, columnCounts];

SetData(rowCounts, columnCounts, levelDataInit);

}

private static void SetData(int rowCounts, int columnCounts, char[,] levelDataInit)

{

for (int row = 0; row < rowCounts; row++)

{

for (int column = 0; column < columnCounts; column++)

{

char symbol = levelDataInit[row, column];

LevelData[row, column] = symbol;

if (symbol == Symbols.heroMapSymbol || symbol == Symbols.orgMapSymbol || symbol == Symbols.skeletonMapSymbol)

{

if (symbol == Symbols.heroMapSymbol)

HeroIndex = UnitsCount;

Units[UnitsCount] = new Unit(symbol, row, column);

UnitsCount++;

}

}

}

}

public static bool GetUnit(int row, int column, out Unit unit)

{

bool found = false;

unit = null;

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

{

if (Units[i].Row == row && Units[i].Column == column)

{

unit = Units[i];

found = true;

break;

}

}

return found;

}

}

Что мы сделали в этом классе?

Этот класс хранит у нас полностью всю информацию о карте и юнитах которые находятся на карте, потому что их надо обрабатывать отдельно.

Мы задали максимальное количество существ на карте в 35 штук, и завели отдельную переменную для подсчёта сколько же у нас всего в данный момент хранится юнитов на карте. Также нам стоит выделить номер героя, чтобы можно было к нему обращаться.

Так как на карте мы не можем ни как получить доступ к объекту другого юнита кроме как по его координатам, то нам нужен метод поиска юнита.

В метод SetMap переехал метод инициализации массива из основного класса.

У нас поменялись методы инициализации и рендера, давайте посмотрим на них:

static Unit Hero;

static void Initialize()

{

Map.SetMap(rowCounts, columnCounts, levelDataInit);

int heroIndex = Map.HeroIndex;

Hero = Map.Units[heroIndex];

}

static void Render()

{

Console.SetCursorPosition(0, 0);

for (int row = 0; row < rowCounts; row++)

{

for (int column = 0; column < columnCounts; column++)

{

char sym = Symbols.GetRenderSymbol(Map.LevelData[row, column]);

ConsoleColor color = Symbols.GetRenderColor(Map.LevelData[row, column]);

Console.ForegroundColor = color;

Console.Write(sym);

}

Console.WriteLine();

}

}

Мы выделили отдельную переменную под героя и в инициализации теперь получаем его из метода класс Map.

В рендере мы теперь запрашиваем данные из класса Map, а символы для отрисовки из структуры Symbols.

Теперь надо до конца разобраться с оружием и юнитами:

class Weapon

{

const int fistDmg = 2;

const int stickDmg = 16;

const int clubDmg = 24;

const int spearDmg = 32;

const int saberDmg = 40;

public WeaponType Type { get; private set; }

public int Damage { get; private set; }

public Weapon(WeaponType type)

{

Type = type;

InitData();

}

public Weapon(char weapon)

{

Type = GetTypeFromChar(weapon);

InitData();

}

private WeaponType GetTypeFromChar(char weapon)

{

WeaponType type = WeaponType.Fist;

if (weapon == Symbols.clubMapChar)

{

type = WeaponType.Club;

}

else if (weapon == Symbols.stickMapChar)

{

type = WeaponType.Stick;

}

else if (weapon == Symbols.spearMapChar)

{

type = WeaponType.Spear;

}

else if (weapon == Symbols.saberMapChar)

{

type = WeaponType.Saber;

}

return type;

}

private void InitData()

{

switch (Type)

{

case WeaponType.Fist:

Damage = fistDmg;

break;

case WeaponType.Stick:

Damage = stickDmg;

break;

case WeaponType.Club:

Damage = clubDmg;

break;

case WeaponType.Spear:

Damage = spearDmg;

break;

case WeaponType.Saber:

Damage = saberDmg;

break;

}

}

}

Тут мы прописали какой урон будет наносить каждый тип нашего оружия.

Мы создали два конструктора:

Первый принимает на вход тип оружия, и в зависимости от типа производит инициализацию своих параметров: урон! )

Второй принимает на входит символ оружия и в зависимости от символа определяет тип, а дальше как в прошлом конструкторе.

class Unit

{

const int heroHealth = 400;

const int orgHealth = 60;

const int skeletonHealth = 80;

const WeaponType heroWeapon = WeaponType.Fist;

const WeaponType orgWeapon = WeaponType.Club;

const WeaponType skeletonWeapon = WeaponType.Saber;

public UnitType Type { get; private set; }

public char Symbol { get; private set; }

public int Row { get; private set; }

public int Column { get; private set; }

public int Health { get; private set; }

public Weapon Weapon { get; private set; }

public Unit(char unitSym, int row, int column)

{

Row = row;

Column = column;

Type = GetUnitFromChar(unitSym);

InitData();

}

private UnitType GetUnitFromChar(char unitSym)

{

UnitType type = UnitType.None;

if (unitSym == Symbols.heroMapSymbol)

{

type = UnitType.Hero;

}

else if (unitSym == Symbols.orgMapSymbol)

{

type = UnitType.Org;

}

else if (unitSym == Symbols.skeletonMapSymbol)

{

type = UnitType.Skeleton;

}

return type;

}

private void InitData()

{

switch (Type)

{

case UnitType.Hero:

Weapon = new Weapon(heroWeapon);

Health = heroHealth;

break;

case UnitType.Org:

Weapon = new Weapon(orgWeapon);

Health = orgHealth;

break;

case UnitType.Skeleton:

Weapon = new Weapon(skeletonWeapon);

Health = skeletonHealth;

break;

default:

break;

}

Symbol = Symbols.GetMapUnitSymbol(Type);

}

public void MoveUnit(ref char carrentSym, ref char nextSym, int row, int column)

{

if (Health <= 0)

{

return;

}

if (nextSym == Symbols.emptySymbol)

{

MoveToCell(ref carrentSym, ref nextSym, row, column);

return;

}

if (nextSym == Symbols.heroMapSymbol || nextSym == Symbols.orgMapSymbol || nextSym == Symbols.skeletonMapSymbol)

{

Map.GetUnit(row, column, out Unit enemy);

Attack(enemy);

}

if (Type == UnitType.Hero)

{

if (nextSym == Symbols.stickMapChar || nextSym == Symbols.clubMapChar || nextSym == Symbols.spearMapChar || nextSym == Symbols.saberMapChar)

{

Weapon = new Weapon(nextSym);

MoveToCell(ref carrentSym, ref nextSym, row, column);

return;

}

}

}

private void MoveToCell(ref char carrentSym, ref char nextSym, int row, int column)

{

carrentSym = Symbols.emptySymbol;

nextSym = Symbol;

Row = row;

Column = column;

}

public void Attack(Unit unit)

{

unit.GetDamage(Weapon.Damage);

}

public void GetDamage(int dmg)

{

Health -= dmg;

if (Health <= 0)

{

Map.LevelData[Row, Column] = Symbols.emptySymbol;

}

}

}

Класс Юнитов.

В нём мы определяем сколько здоровья у каждого из юнитов, какое оружие он носит и научили его атаковать и получать урон, давай посмотрим по ближе:

Конструктор тут один и он очень похожий на тот что был у оружия, получаем символ и координаты, чтобы потом в классе Map мы смогли найти нашего юнита.

По символу он определяет, что это за юнит и в зависимости от типа определяет его характеристики.

Теперь к методу «Ходить»

Чтобы ходить мы должны определить, что за клетка перед нами: пустая, стена, соперник, оружие.

Если пустая то всё очень просто, мы просто идём на эту ячейку. Так как мы получаем наши символы по ссылке, то мы их перезаписываем, а юниту присваиваем новые координаты

private void MoveToCell(ref char carrentSym, ref char nextSym, int row, int column)

{

carrentSym = Symbols.emptySymbol;

nextSym = Symbol;

Row = row;

Column = column;

}

Если мы встречаем на своём пути соперника, то мы должны его атаковать, сила атаки, это урон нашего оружия. Теперь нам надо найти объект нашего соперника, чтобы дать ему урон, что мы и делаем с помощью класс Map: находим и атакуем

if (nextSym == Symbols.heroMapSymbol || nextSym == Symbols.orgMapSymbol || nextSym == Symbols.skeletonMapSymbol)

{

Map.GetUnit(row, column, out Unit enemy);

Attack(enemy);

}

public void Attack(Unit unit)

{

unit.GetDamage(Weapon.Damage);

}

В методе атаки, мы получаем объект соперника и вызываем у него метод «получить урон» и в качестве аргумента передаём урон нашего оружия.

public void GetDamage(int dmg)

{

Health -= dmg;

if (Health <= 0)

{

Map.LevelData[Row, Column] = Symbols.emptySymbol;

}

}

В методе получить урон, мы отнимаем у нашего здоровья полученный урон, и смотрим сколько его осталось. Если осталось меньше нуля, то мы убираем его с карты.

Есть также блок специальный для героя:

if (Type == UnitType.Hero)

{

if (nextSym == Symbols.stickMapChar || nextSym == Symbols.clubMapChar || nextSym == Symbols.spearMapChar || nextSym == Symbols.saberMapChar)

{

Weapon = new Weapon(nextSym);

MoveToCell(ref carrentSym, ref nextSym, row, column);

return;

}

}

Если герой наткнулся на оружие, то мы заменяем ему его оружие и ставим его на эту клетку с оружием.

Теперь нашу игру можно запускать, двигаться и атаковать, дальше мы научим наших врагов двигаться и атаковать в ответ героя.

Полный код: <https://github.com/WolfyW/ConsoleGame>