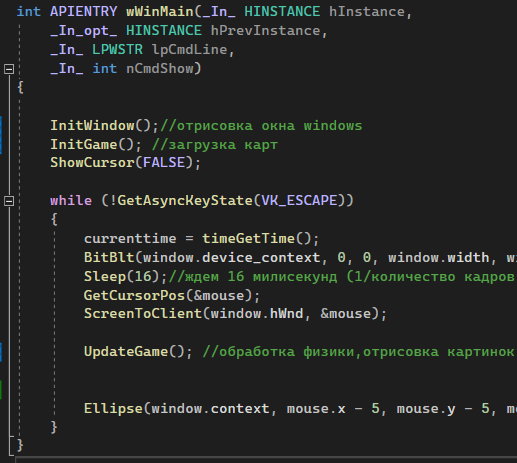
**Документация на ECS и что поменялось после**

**рефакторинга**

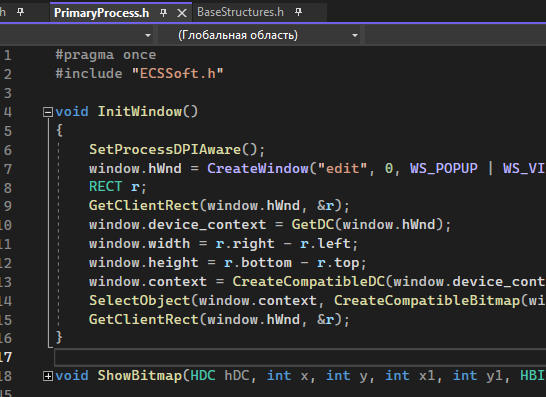
**Рефакторинг**

Рабочая область main очень сильно уменьшилась (очень советую не писать все что связано с логикой там для этого появился специальный заголовочный файл)

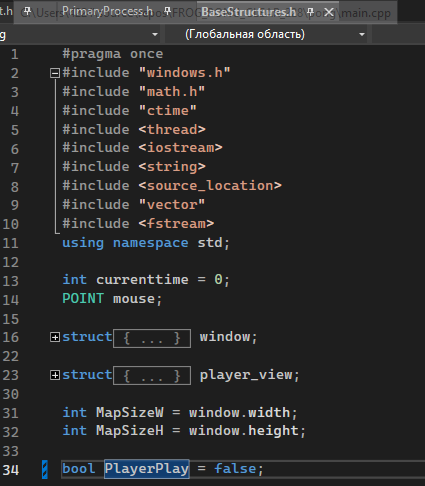


Для операция которые требуют постоянной отрисовки кадра появился метод “UpdateGame”

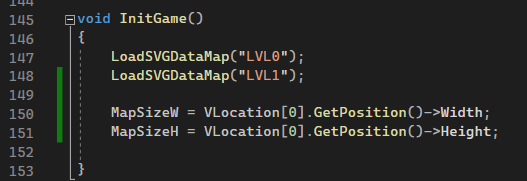
Файл “PrimaryProcess.h” хранит базовые методы для отрисовки Windows



Файл “BaseStructures.h” хранит базовые структуры и базовые заголовочные файлы для работы



Файл “GameFileSystem.h” так же выполняет функцию загрузки, но с очень большим апгрейдом, так же функция загрузки игры перенесена в этот файл

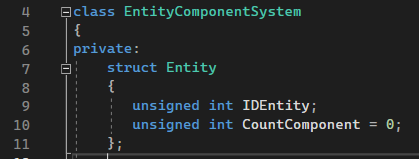


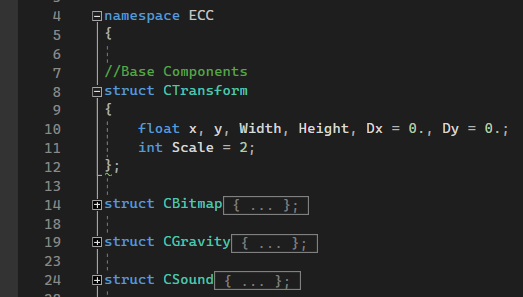
Старые заголовочные файлы были удалены поскольку весь функционал был перенесён на новую архитектуру

**Архитектура Entity Component System + ArcheType**

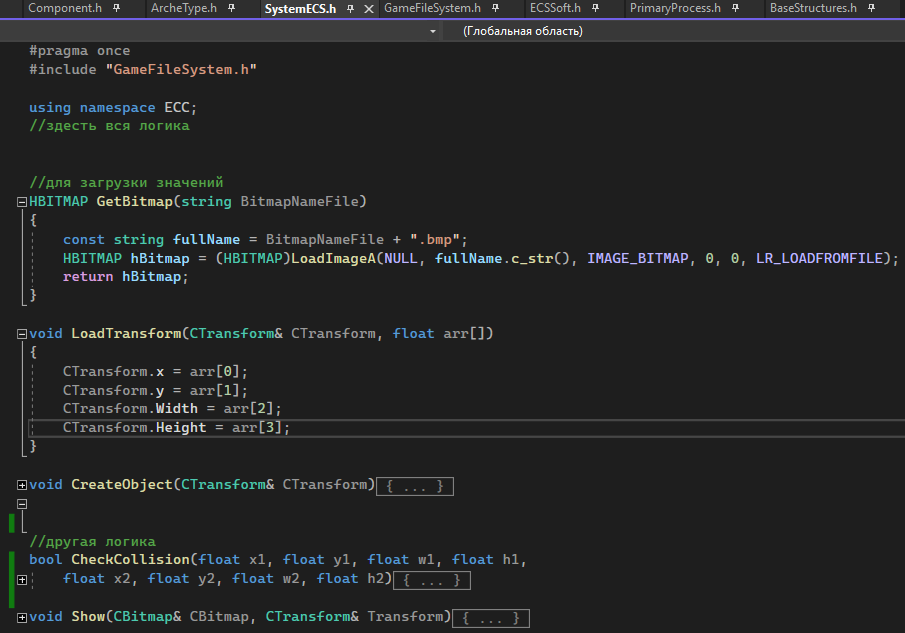
**Краткое пояснение за архитектуру**

Entity - это структура, которая знает только о том какой у него идентификатор и сколько к нему подключено компонентов “ECSSoft.h”

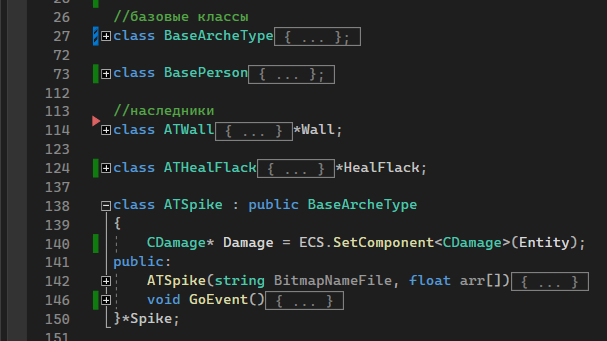


 Component - это стандартные структуры данных, которые содержат исключительно столько данные (подключены к пространству имён ECC) “Component.h”

System - Системы это методы которые решают различные задачи, они подключаются к архитектурам для дальнейшей работы объектов, архетипов “SystemECS.h”



ArcheType - Архетип это объект класса, который подключает к себе компоненты через ECS и так же подключает к себе системы “ArcheType.h”

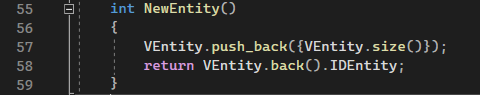


**Устройство работы ECS**

Есть множество подходов в создании ECS, но принцип один должны быть 2 контейнера, один хранит все “Entity”, а другой все “Component”, для этого я использовал контейнеры “vector”



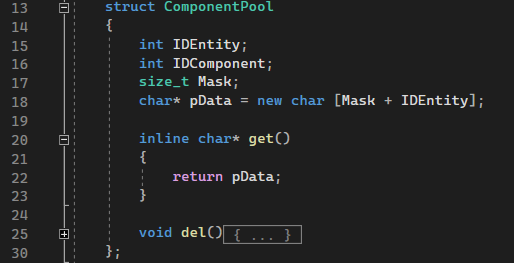
Следующим этапом будет создание сущности для этого описана функция “NewEntity” она добавляет сущность в список и сразу же присваивает индекс этой сущности



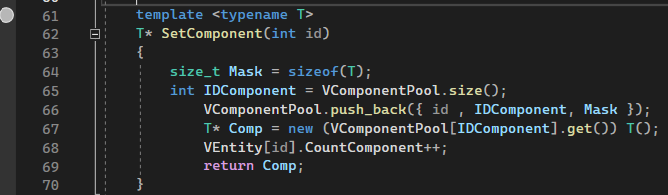


Теперь самое сложное это добавление компонента к сущности для этого создан шаблонный метод который получает размер в байтах компонента, получает свой идентификатор, и добавляет себя в массив к компонентам при этом у нас есть структура как должен выглядеть компонент внутри

**Структура компонета**



**Операция присвоения компонента**

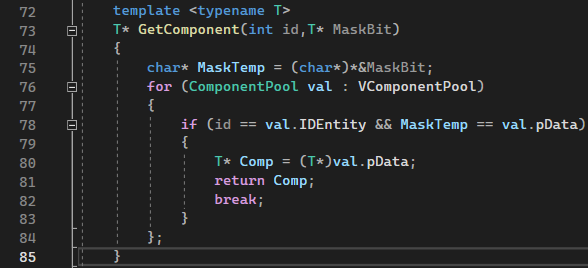


Немного поподробнее про 67 строчку вы можете заметить, что мы создаём новый экземпляр компонента из “кучи” при этом мы вызываем функцию которая описана выше на 20 строчке, функция “get()” отдает уникальную “bitmap” преобразованную из маски тоесть размера класса + идентификатора сущности при этом мы возвращаем объект обратно, и дополнительно еще в сметчике компонентов у сущности делаем инкремент компонента.

**Как мы подключаем компонент**



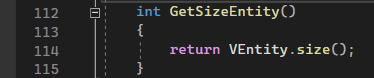
**Получение компонента**

Нам часто прийдётся получать нужный нам компонент поэтому для этого есть шаблонный метод “GetComponent” изначально он преобразует объект компонента который мы хотим получить в ссылку то есть в “bitmask”, затем бы при помощи цикла находим нужный нам компонент и возражаем его обратно

**Как мы получаем компонент**

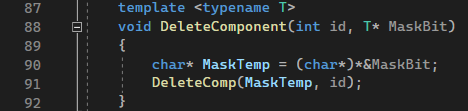


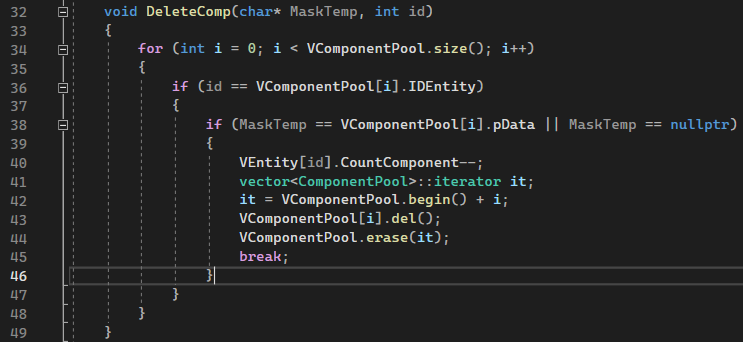
Для удобства есть функция получения всего количества сущностей

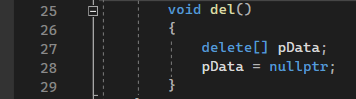


**Удаление и очистка памяти**

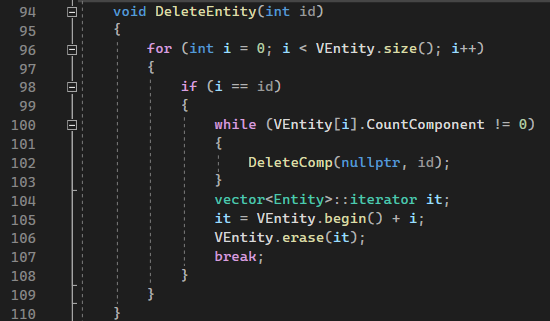
Начнём с удаления компонента он работает по такому же принципу как и нахождение компонента, но только теперь он удаляет его для этого мы вызываем шаблонный метод “DeleteComponent”



А после переходим в функцию “DeleteComp” В этой функции рассмотрено сразу 2 варианта удаления как 1 компонента так и множества

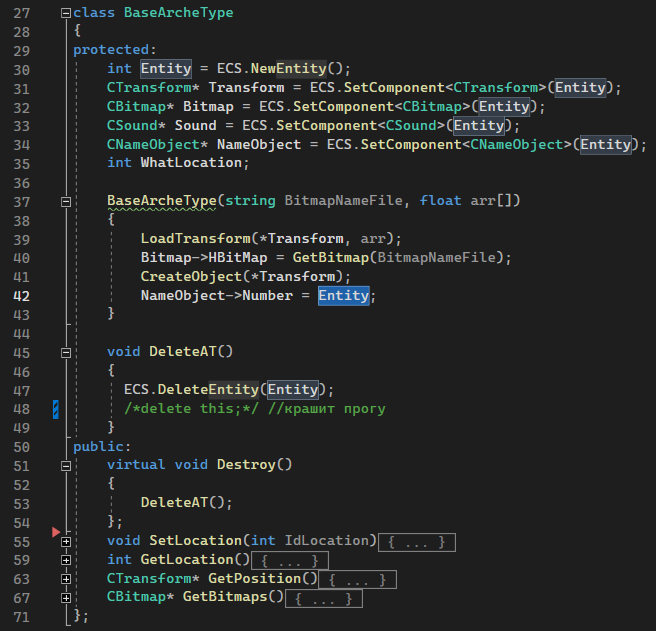


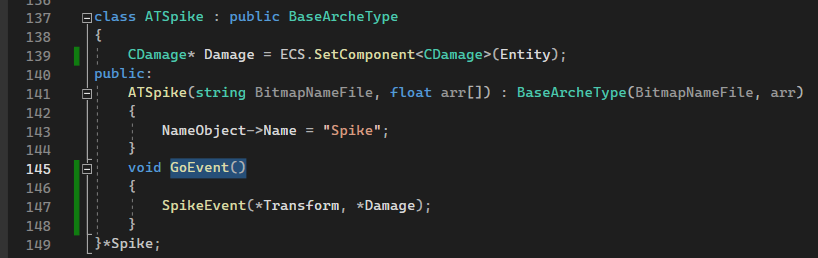
Так же есть метод удаления сущности и всех его компонентов сразу для этого мы так же используем функцию “DeleteComp”

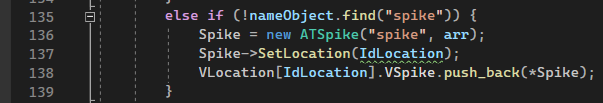


**Теперь как этим всем пользоваться простой алгоритм**

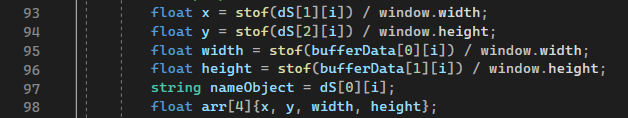
К примеру вам нужно создать новый игровой объект, допустим хилку, если персонаж ее подберёт то отхилится, а хилка пропадёт

1. Создаем класс “в файле ArcheType.h” ATHealFlack Наследуемся от базового архетипа “BaseArcheType” - это архетип который содержит в себе методы конструктор который уже сразу указывает позицию объекта, записывает его картинку и создает образ объекта в мире, и так же присваем ему номер идентификатора по Entity и так же имеет несколько открытых функций для удобства 
2. Если нам нужен компонент, которого нету в базовом классе мы его добавляем, и сознаем конструктор в котором указываем имя объекта, так же теперь мы можем создать функцию через которую объект может выполнять какие либо задачи, допустим это будет ивент (это конечно не настоящий ивент из паттерна observer, но это просто пока так будет) для этого сознаем метод GoEvent() и в него уже передаем систему которую хотим запустить



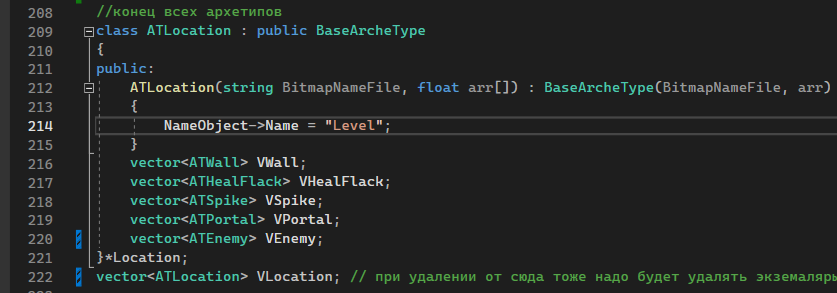
1. Создаём объект мы так же через файловую систему если этот объект будет создаваться в игровом мире для это мы теперь используем вот такую конструкцию 

Arr мы указываем это теперь позиция x,y,w,h объекта (так просто проще читается)



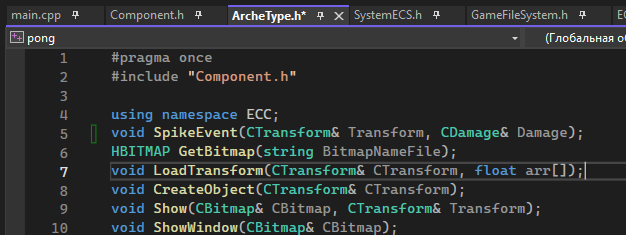
1. Можно заметить что объект шипов на 137 строчке куда то еще добавился в массив это массивы в локации который так же является архетипом если этот объект будет не в единственном экземпляре то для него нужно будет создать новый массив

Вот здесь



1. Мы все создали все добавили теперь нужно написать логику к объекту для этого в файле “SystemECS.h” создаем функцию которая будет работать с нашим объектом в качестве аргументов мы будем принимать компоненты этого объекта они уже были в базовом архетипе так что они уже есть и у нашего объекта

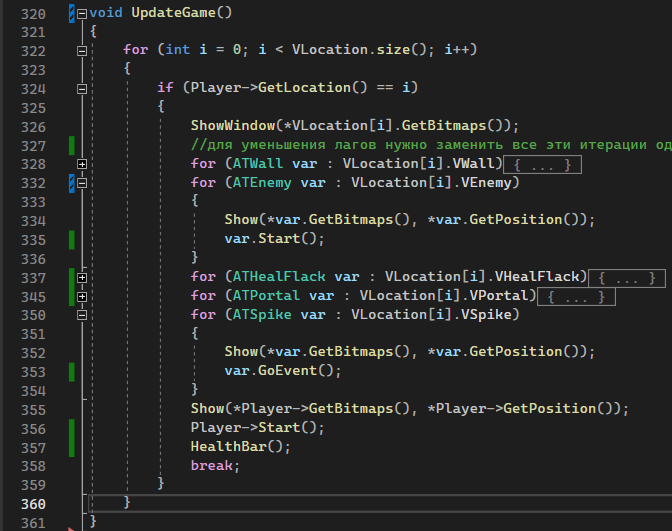


Есть один нюанс при создании всегла когда вы содаете метод вы должны его переопределить в файле “ArcheType.h” для того чтобы система могда видеть массив “vector<ATLocation> VLocation”

1. **Теперь мы пишем обычную логику которую должен выполнять данный объект как у нас было и раньше мы проверяем прикосновение с игроком если оно было мы вызываем получаем урон и так далее отпрыгиваем и все такое все это изложено внутри логики**

1. Последний этап запуск объекта

Для этого создан специальный метод в самом низу файла системы, данный метод перебирает все объекты именно этой локации, и выполняет их обрисовку по тому же принципу что и раньше с клипингом, так же он выполняет ивенты если мы это укажем, и так же выполняет расчёт физики персонажей



**НОВАЯ КРУТАЯ ФИЧА И НОВЫЕ ОСОБЕННОСТИ РЕДАКТОРА КАРТ**

Теперь можно делать много локация и переходить по ним через порталы для этого в редакторе нужно указать в названии портала в ту локацию куда он будет телепортировать, место положение куда будет телепортировать пока еще не указывает, но в другую локу телепортнет нормально главное раставить порталы на картах

Телепорт в локу 1



Телепорт в локу 0

