КОДИТЬ НА 3DS ПРОЩЕ, ЧЕМ ВЫ ДУМАЕТЕ.

ВВЕДЕНИЕ

Для того чтобы написать любую игру вам нужна **идея**, то к чему вы стремитесь, то что вас будет побуждать делать все новые и новые шаги на пути к мечте.

Идея - это главный двигатель прогресса вашего игрового проекта, именно она даёт вам цель того что вы хотите увидеть на экране вашей консоли. Без идеи вы будете мотаться из стороны в сторону за красивой картинкой и ломать голову за очередным неструктурированным кодом, не получая никакого прогресса в ваших возможностях, что не есть хорошо.

Этот учебник нацелен на программистов начального уровня, но базовые знания **языка программирования Си** не помешают. На этом закончим с предисловием, и вперёд к новым открытиям! ✊

ГЛАВА 1. ОСНОВЫ

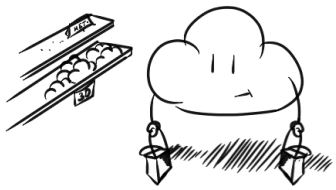
Для начало рассмотрим файл **main.c** в папке **source**.

#include

```
1  //AeroCrunch by CrithCraft Apr. 2019 (C) for 3ds
2
3  #include <citro2d.h>
4
5  #include <assert.h>
6  #include <string.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <time.h>
10
```

Здесь мы видим первые строки подключают основные библиотеки, т.к перед ними стоит **`#include`** - что значит подключить. **Библиотеки** состоят из функций. **Функция** в свою очередь это словесная форма действий, что должна сделать программа.

Проведём аналогию с реальной жизнью, вы дали себе команду **пойти в магазин за хлебом**. Данная затея состоит из нескольких действий. Нужно одеться, дойти до магазина, найти хлебный отдел и т.д, до момента когда вы окажетесь дома с булкой хлеба в руках. Действий действительно много, но их объединяет одна цель - сходить за хлебом. Если писать это как код то мы бы написали функцию **`СходитьЗаПокупками()`**, и в скобки написали бы значения функции, то есть зачем нужно сходить в магазин, за хлебом, значит функция будет выглядеть вот так **`СходитьЗаПокупками(“Хлеб”)`**.



- ***Настоятельно не рекомендую вставлять это в свой код, т.к. AeroCrunch не умеет ходить в магазин))***

НАПРИМЕР:

- *`C2D_SpriteSheetLoad(“romfs:/gfx/example.t3x”)`* – данная функция принадлежит библиотеке *`Citro2D`* и возвращает лист спрайтов который находится по пути: «*путь к вашему проекту*»*`/gfx/example.t3x`*

Прежде чем перейти далее я дам несколько коротких комментариев по строчкам которые вы видите ниже:

```
 9 ▼ #include <rendering/AeroCrunch.h>
10   #include <rendering/scenes_connect.h>
11
12   #define WIDTH    400
13   #define HEIGHT   240
14
15   int SaveScene_ID = 0;
16
17   // SpriteSheet Initilize
18   static C2D_SpriteSheet spriteSheet_players;
19   static C2D_SpriteSheet spriteSheet_map;
```

Первые две строчки как вы помните помните подключают библиотеки к файлу, но **есть отличие**. Здесь указан путь библиотеки, что значит что мы подсоединяем библиотеку которая указана по пути в тр. скобках.

НАПРИМЕР:

- *`#include <rendering/AeroCrunch.h>`* – такой записью мы подключаем файл библиотеки *`AeroCrunch.h`* , которая находится по пути: *«путь к вашему проекту»`/source/rendering/AeroCrunch.c`*

- ***Все ваши дополнительные файлы которые подключаются внутри файлов вашего проекта, должны находится внутри проекта.***

Все что идёт далее - **это объявление переменных.**

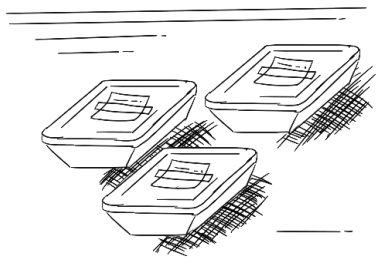
Переменные

Чтобы понять что такое переменные давайте представим аналогию с жизнью в виде **контейнеров с едой.** Для примера вы будете замужней женщиной с чадом в придачу. Вы собираете обед себе, мужу и сыну(или дочери) в разные контейнеры. Каждый контейнер имеет своё имя (Ира, Паша, Дима). В каждом контейнере лежит свой обед (Салат, Борщ, Пюре). Тогда у нас есть **три переменные:**

С именем **Ира** который содержит **салат** - Ира = ”салат”

С именем **Паша** который содержит **борщ** - Паша = ”борщ”

С именем **Дима** который содержит **пюре** - Дима = ”пюре”



Но помимо этого нужно помнить что любая переменная должна иметь определенный тип значений (число, буква или целый контейнер переменных). Вы же не положите нитки в контейнер с едой. Это аналогия с тем, когда вы в переменную для чисел пишете символы.

Для этого нам надо писать **тип переменной.** Он пишется до названия переменной и только при её обьявлении в коде.

Тип	Размер (бит)	Диапазон значений
bool	1	true/false
byte	8	0..255 (2^8)
sbyte	8	-128..127 ($\pm 2^7$)
short	16	-32768..32767
ushort	16	0..65535
int	32	-2147483648..2147483647 ($\pm 2^{31}$)
uint	32	0..4294967295 (2^{32})
long	64	-9223372036854775808..9223372036854775807 ($\pm 2^{63}$)
ulong	64	0..18446744073709551615 (2^{64})
decimal	128	$\pm 1.0*10^{-28}..\pm 7.9*10^{28}$
float	32	$\pm 1.5*10^{-45}..\pm 3.4*10^{38}$
double	64	$\pm 5.0*10^{-324}..\pm 1.7*10^{308}$
char	16	Один любой символ Unicode

НАПРИМЕР:

- `int SaveScene_ID = 0` – такой записью мы объявляем переменную **для чисел** `SaveScene_ID` и приравниваем ее к 0.
- `SaveScene_ID = 1` – такой записью мы поменяли значение в уже объявленной выше переменной присвоив ей значение равное 1.

ИСКЛЮЧЕНИЕ ПО НАПИСАНИЮ:

- `#define WIDTH 400` – `#define` переменные неизменяемы, значения подставляются заранее, всегда пишутся с верхним регистром и объявляются по такой структуре без = и `;`
- В данном примере мы обьявили `#define` переменную разрешения экрана **`WIDTH`**, дабы облегчить жизнь в написании будущего кода.

ДОПОЛНИТЕЛЬНО:

- *`static`* – данный элемент, при постановке его до типа переменной (или функции), объявляет вашу переменную (или функцию) статичной. Если говорить о 3DS коде, то данную вставку мне приходилось подставлять для объявления переменных листа графики (тип **`C2D_SpriteSheet`**). Статичные переменные могут присвоить значение только **один раз**. После этого они становятся не редактируемые!

int main()

```
20
21 int main(int argc, char* argv[]) {
22
23         // Prepare Scene
24         Aer_PrepareScene();
25
26         // Set upper and bottom screen to Graphics
27         C3D_RenderTarget* top = C2D_CreateScreenTarget(GFX_TOP, GFX_LEFT);
28         C3D_RenderTarget* bottom = C2D_CreateScreenTarget(GFX_BOTTOM, GFX_LEFT);
29
30         // Set spritesheet
31         spriteSheet_players = C2D_SpriteSheetLoad("romfs:/gfx/sprites_players.t3x");
32         if (!spriteSheet_players) svcBreak(USERBREAK_PANIC);
33         spriteSheet_map = C2D_SpriteSheetLoad("romfs:/gfx/map.t3x");
34         if (!spriteSheet_map) svcBreak(USERBREAK_PANIC);
35
```

Далее следует объявление функции main которое возвращает числовое значение - **`int`**. Функции могут возвращать значение тех типов которые вы прописали ранее как тип функции. Для того чтобы указать какое значение должно посылать функция нужно записать: *`return`* **`значение`**

Данную особенность функции можно использовать чтобы например приравнивать переменные к функциям с выставленными вами значениями. Функция после выполнения ввернет полученное ей значение и присвоит его в переменную.

НАПРИМЕР:

- *`SaveScene_ID = Aer_ReadSave(0)`* – такой записью мы приравниваем **`SaveScene_ID`** к значению которое выдаст функция **`Aer_ReadSave()`** (чтение сохранения) под значением **`0`**, что по логике моей библиотеки подразумевает **`слот 0`**, отсылка на старые добрые игры из 90! :D

- ***Внимание, загрузка и запись сохранений `Aer_ReadSave()` и `Aer_WriteSave()`, пока находятся в разработке и пока отсутствуют в первой публичной версии библиотеки `AeroCrunch`.***

ИСКЛЮЧЕНИЕ ПО НАПИСАНИЮ:

- *`void`* – тип функции который ничего не возвращает, в функции такого типа вам не нужно будет писать `return`, удобно кстати если вам нужно написать функцию нацеленную только на выполнение действий.

Далее в скобках идёт объявление переменных, это как раз те переменные которые вы вводите в скобках при вызове функции. Действия указанные для выполнения функции записывают внутри фигурных скобок **`{}`**

С функциями и переменными покончили, перейдём к основному коду.

Первая функция **`Aer_PrepareScene()`** подготавливает сцену на которой и будет все строится - фундамент, основа понимаете как вам будет удобнее.

Далее с помощью **`C3D_RenderTarget`** мы объявляем то что мы будем рисовать графику в верхнем и нижнем экране.

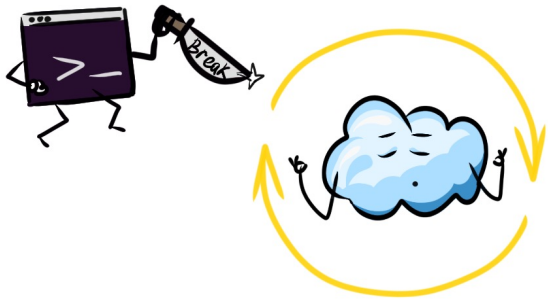
Следующие строки это присваивание переменной листа спрайта **`spriteSheet_players`** значению функции, которая и занимается экспортом изображения в код - **`C2D_SpriteSheetLoad()`**, далее последует проверка, здесь прописано условие, что если отсутсвует лист спрайтов (**`!spriteSheet_Player`**), то с нашей стороны вызываем панику системы - **`svcBreak(USERBREAK_PANIC)`** (это в будущем поможет нам с обработкой текстур).

while()

```
35
36      while (aptMainLoop())
37      {
38              // Exit to Homebrew launcher
39              hidScanInput();
40              if (hidKeysDown() & KEY_START)
41                  break;
42
43              // Begin render frame the scene
44              C3D_FrameBegin(C3D_FRAME_SYNCDRAW);
45
46              // Draw top screen
47              Aer_RenderScreen(top);
48              LoadMap(SaveScene_ID, spriteSheet_players, spriteSheet_map, 0);
49
50              // Draw bottom screen
51              Aer_RenderScreen(bottom);
52              LoadMap(SaveScene_ID, spriteSheet_players, spriteSheet_map, 1);
53
54              // End render frame the scene
55              C3D_FrameEnd(0);
56      }
57
```

Вы когда нибудь задавались вопросом, до какого момента работает программа?

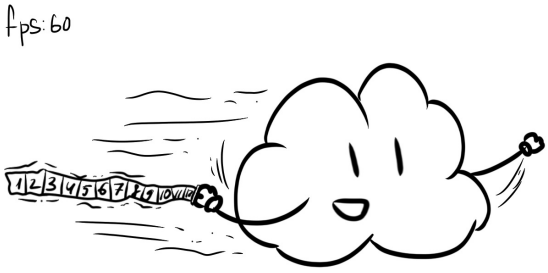
Цикл **while()** подразумевает, все что будет находится в нем будет происходит бесконечно долго, до момента когда условие в скобках будет выполнено. В нашем случае условие действует бесконечно долго, это значит что единственный выход завершить его, командой **break**.



Первая строка в нашем цикле **hidScanInput()** нацелена на отслеживание клавиш. Она отслеживает какие и как клавиши нажимаете (например как вы держите ***зажатой* кнопку A**).

Далее идёт проверка на условие зажата ли клавиша **hidKeysDows()**, ***и*(&)** зажата ли кнопка **Start(KEY_START)**. Если условие выполняется то **прерываем цикл жизни программы**.

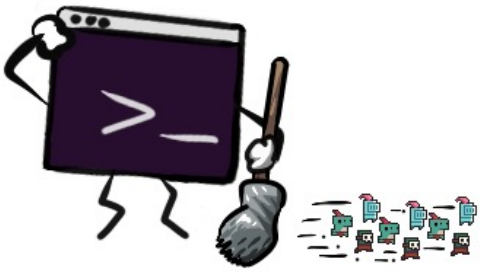
Следущая функция даёт команду на начало отрисовку кадра **C3D_FrameBegin()**. Не думаю что стоит объяснять что такое кадры и их частота в играх, к сожалению я не нашёл никаких функций по их регилировке,



скажу только лишь, что ваша homebrew игра, пока процессор не загружен, работает в **`60 fps`**.

После этого прописываем загрузку карты для верхнего и нижнего экрана. Для этого с помощью функции **`Aer_RenderScreen()`** - прорисовка экрана. Сначала включаем прорисовку вверхнего экрана, позже загружаем карту **`LoadMap(номер карты, лист с текстурой карты, лист с текстурой игрока, номер экрана)`**, также поступаем с нижним экраном.

Мы отрисовали комнаты, теперь нам нужно закрыть отрисовку кадра **`C3D_FrameEnd()`**. На этом мы закрываем цикл. Но на этом код ещё не окончен. При нажатии **`кнопки Start`** мы завершаем цикл программы, а кто будет убирать текстуры и библиотеки из памяти консоли. Этим нам и нужно заняться в последних строках кода.



Очистка памяти консоли и завершение main.c кода

```
57
58          // Clearing after closing
59          C2D_SpriteSheetFree(spriteSheet_players);
60          Aer_DefineLibs();
61          return 0;
```

После долгих мучений, мы с трудом добрались до стадии закрытия приложения, в случае его удачного завершения. Пункт первый, очищаем память от наших текстур **`C2D_SpriteSheetFree`**(**`лист спрайтов`**), данная функция призванна очистить память от заранее объявленного нами листа спрайтов указанного в скобках.

Далее ничего более писать не нужно **`AeroCrunch`** сделает это за вас. Функция **`Aer_DefineLibs()`** призвана закончить работу с библиотеками и некоторыми другими составляющими одной функцией. После этого, нам нужно известить 3DS, что все прошло окей и послать 0 в завершении main функции **`return 0`**.

На этом все в следующем томе который выйдет в пятницу на следующие недели я уже затрону файл **`map_001.c`** файл в котором и реализуется первая карта. Также будут затронуты графические функции **`AeroCrunch`** работа со спрайтами и их данными, а также покажу возможность реализации анимации под ваш определенный такт.

Удачи, до пятницы! 👋👋👋