ARCADE MACHINE

(Аркада)

Рангел Леонидов Плачков

Компютърни науки

1MI0800061

Задание:

Основната цел на проекта е да се направи система ,на която могат да се играят аркадни игри / компютърни игри. Машини ,като тази се използват предимно от деца и затова ще трбява да е здрава (физически) и интуитивна за използване. Игрите ще могат да се сменят с добавяне на код затова ще има софтуерна среда позволяваща програмист с основни знания по C++ да може да разработи игра (без да разбира от електроника, роботизирани системи и подобни).

Бюджет:

Електронни компоненти :210лв.

3д филамент:60лв.

Изгорели електронни компоменти :60лв.

Общо: 330лв.

Електронни компоненти:

1бр. Arduino mega 2560 PRO L **20лв.**

6бр. WS 2812 LED-Matrix 16x16 **30лв.**

5бр. Бутони с вграден pull-up резистор **0.80лв.**

2бр. Joystick Dual-axis XY module **4.50лв**

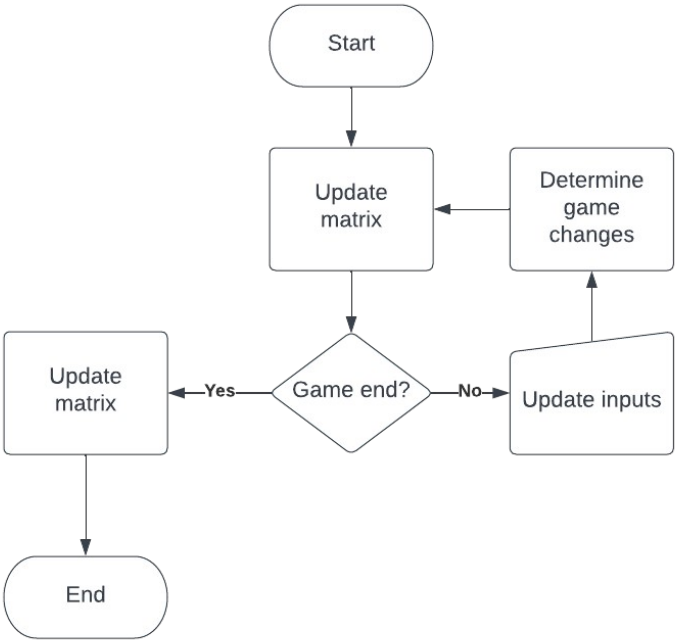
1бр. Захранващ блок 220VAC → 12DC 8.3A **27лв.**

2бр. Конвертор на напрежение от 12V 3A **10лв.**

Други (Джъмпери , Тенол, Болтове, Гайки ...) Общо:**10лв.**

Наръчник на програмиста:

Всички игри следват следните стъпни:



Въпреки ,че не е задължително да се спазва повечето игри имат схема поризходна на горната.

В този раздел ще се опишат най-важните функци ,които разработващият игри ще има нужда. За повече информация софтуерът е описан по-долу.

Arcade *example*; → Създава се клас Arcade с име „example”. Със създаването на този обект автоматично се дефинират всички константи , функции и член данни ,които хардуерът има нужда.

Cell *pixel*; → Създава се обект Cell с име pixel. Екранът на аркадата е съставен от 32x32 такива обекта.

Cell *matrix*[32][32]; → Създава се матрица от Cell-ове с име „matrix“.

example.**show**(matrix); -> **show()** е член функция на **Arcade**. С нея се отпечатва „matrix“ върху екрана на машината. Очакваният аргумент е **Cell** матрица 32x32.

example.**button(id)** → Връща 0(False) или 1(True) на база на това дали бутон с индекс **id** е натиснат.

example.**joystickX(id)** / *example*.**joystickY(id)** → Връщат стойности между 0 и 1023 на база позицията на joystick с индекс **id**.

example.**joystickD(id)** → Връща 0(False) или 1(True) на база на това дали вгреденият бутон на джойстик с индекс **id** е натиснат.

BUTTON_COUNT → Константа брой бутони.

JOYSTICK_COUNT → Константа брой джойстици.

Поддръжка:

В този раздел са описани проблеми ,които са наблюдавани да се получават периодично.

Проблем: Матрицата си променя яркоста няколко пъти в секунда. Матрицата има забавяне (0.1сек. ~ 0.5сек.) посредата на изпълнение на функция **show()**.

Решение: Причината за е че има недостиг на напрежение към Екрана. Причина за това може да е това потенциометъра на преобразувателите от 12V да се е разместил. С отверка може да се коригира. Целим се напрежението да е между 4.9V и 5V.

Проблем: По екрана има хаотично светене на пиксели. **Show()** се държи непредвидимимо.

Решение: Към Екранър се подава прекалено голямо напрежение и аналоговите сигнали започват да се четат по различен начин. Това лесно се коригира с отверка и калибриране на потемциометрите. Целим се напрежението да е между 4.9V и 5V.

Проблем: Едната половина на екрана свети по-ярко от другата.

Решение: Захранването е разделено на 2 с 2 преубразователя. Повсяка вероятност разликата в напрежението е станала усезаемо голяма. Лесно се коригира с калибриране на потенциометрите.

Проблем: Микроконтролера постоянно му свършва паметта.

Решение: Запазването на Cell матрица 32x32 отнема около 40% от паметна на микро контролера, така че лесно се вижда как това може да е проблем. Решение засега няма. Постарайте се да влизате в ограниченията. Рекурсията също може да доведе до проблеми. Не иделно решение е смъкване на игралното поле от 32x32 към 16x16.

Хардуер:

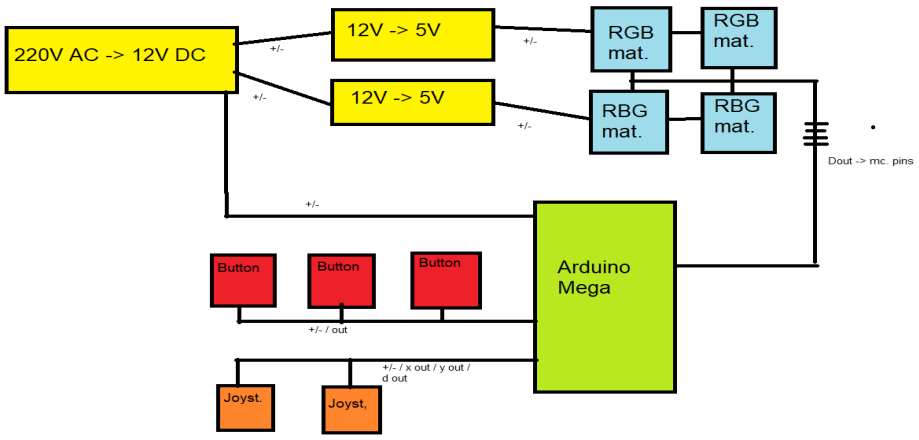
Основната част на хардуера е захранването на матриците. Всички останали елементи са достатъчно леки да се захранват от микроконтролера правейки работата с тях лесна.

Захранването на машината започва от захранващ блок от 220V променлив ток към 12V прав ток. От него захранваме микроконтролера ATMEGA 256 PRO L.

LED-Матриците имат нужда от 5V затова от захранващия блок има 2 отделни вериги. Всяка верига има конвертор на напрежение от 12V към 5V. И от конвертора са свързани 2 от 4те матрици.

Микроконтролера ATMEGA 256 PRO L е на практика Arduino MEGA и няма нужда от допълнителни драйвери или библиотеки за да се програмира. Към неко са Закачени всики бутони , джойстици и сигнални пинове на матриците. Причината на не работи с Arduino Uno е недостиг на памер.

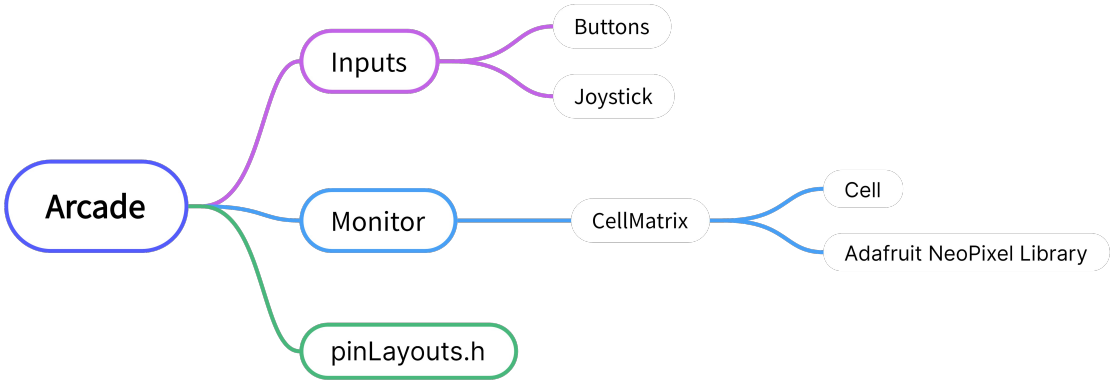
Микроконтролера е закрепен към платка предвидена за опростяване на схемите.



Матриците са WS2812 16x16 , работещи идентично с NeoPixel на Adafruit. https://www.adafruit.com/product/1487 (Линкът е към матрица 8x8 ,но използваме 16x16)

Сигналните пинове на матриците са свързани на четири отдлни пина въпреки че има теоретична възможност да са на един ,на практика сигнала не е достатъчно силен да стигне всичките 1024 пиксела.

Софтуер:



Програмата може да се раздели на 2 части ,а именно Monitor и Inputs.

Monitor има 4 обекта CellMatrix всеки от ,които управлява модул от пиксели 16x16. Големият клас Monitor управлява 4те модула. Основната цел на класа е да има фукнция която приема като параметър Cell [32][32] матрица и е отразява подообаващо.

Inputs има за цел лесен достъп до информацията от всяко входно устройство. То има матрици от обекти Button и Joystick които се съхраняват в матрица и съответно всеки модул има позижия(индекс) в матрицата.

Monitor и Inputs се обединяват в големият основен клас Arcade. В него се поема всяка декларация на пин и инициализирането на всеки „подготвителен“ код. От калса Arcade има достъп до много малък набор от функции независими една от друга диктуващи всички действия случващи се по ситемата.

клас Cell

Класът Cell има за цел да запазва данните за всеки отделен пиксел. В него има 3 член данни от тип "char", Причината да е char е да се използва само един байт място. На практика от тях се извличат числа все едно е Int. Класа има и статична константа (**MAX_VALUE**),която забранява RGB стойностите да премиинат определно ниво.

void setRed(int newRed)

Сетър ,който има за цел да сложи на m_Red член данната цвета newRed по начина:

m_Red = newRed % MAX_VALUE, функцията си има еквивалентите за зелено и синьо.

void set (int newRed, int newGreen, int newBlue)

void set (const String& color)

void set (const Cell& other)

Сетър който задава стойностите на трите стойности, алтернативно ако се подаде String и разпознае цвят ще зададе предварително набрани стойности. Ако не разпознае никакъв цвят ще направи пиксела (0,0,0). Ако се подаде друг Cell взима неговите член данни.

void clear()

Слага стойностите на пиксела на (0,0,0)

int getRed()

Връща m_Red съществуват и еквивалентните функции за зелено и синьо.

Cell()

Cell(int newRed, int newGreen, int newBlue)

Cell(const String& str)

Cell(const Cell& other)

Конструктори. Дефолтно стойностите са (0,0,0). Останалите изпълнява set() с параметъра.

Cell& operator=(const Cell& other)

Cell& operator(const String& other)

Оператор= изпълнява set() с даденият параметър. Връща (*this).

клас CellMatrix

Класът има за цел да обработва пикселите за един сегмент от 16x16 пиксела. Тези стойсности могат да се променят от **MAX_SIZE_X** и **MAX_SIZE_Y**. Освен тези константи има и **MAX_SIZE_X_ALL** и **MAX_SIZE_Y_ALL** , препрезентират големината на Целият дисплей (Съставен от всички отделни модули). Класът използва класа Cell също и библиотеката *Adafruit_NeoPixel* https://github.com/adafruit/Adafruit_NeoPixel ,статичната константа **PIXEL_COUNT** = MAX_SIZE_X * MAX_SIZE_Y.

int m_Pin

Класа има единствена член данна, която запазва пина към който е свързана. Стандартната(default) стойност е (-1) и това се използва за проверка ако се направи опит за управление на матрицата без здаден пин.

void setPin(int newPin)

Сетър на m_Pin.

CellMatrix()

Дефолтен конструктор даващ дефолтна стойност на m_Pin = -1.

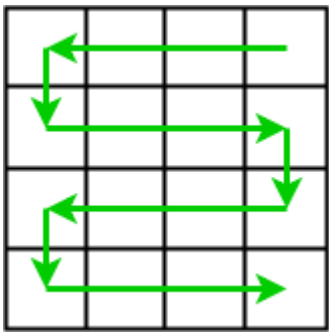
CellMatrix(int newPin)

Конструктор викащ setPin(newPin).

void applyMatrix(const Cell (&newMatrix)

[MAX_SIZE_X_ALL][MAX_SIZE_Y_ALL],int startX,int startY)

Функция приемаща за параметър Матрица с размерите на екрана (В случая 32x32) и параметри казващи от кой пиксел да се принтира. Поради фабричното подреждане и ротацията при запояване матриците се обхождат по следния начин



И се изпражат последователно към библиотеката *Adafruit_NeoPixel*.

(Източник на снимката + алгоритъм за обхождане : https://www.geeksforgeeks.org/print-matrix-in-snake-pattern-from-the-last-column/)

StartX и StartY се използват за посочване на най-горния-ляв пиксел от който да четем

голямата матрица. В сучай че m_Pin = -1 фнкцията се прекъсва.

void clear()

Изчиства всички пиксели на дисплея. Ако m_Pin = -1 неправи нищо. Сложност по време o(1).

клас Monitor

Класът има за цел менажирането на целият екран. В него има статични константи **MATRIX_IN_X** и **MATRIX_IN_Y** казващи по колко сегмента (от клас CellMatrix) ще има на ред и колкона. В случая са по 2. Освен това има и 2 статични константи

CELLS_IN_X = MATRIX_IN_X * CellMatrix::MAX_SIZE_X

CELLS_IN_Y = MATRIX_IN_Y * CellMatrix::MAX_SIZE_Y

Показващи колко пиксела има по ред и колко по колона.

CellMatrix m_Libs[MATRIX_IN_X][MATRIX_IN_Y];

Единствена член данна пазеща модулите а също и положението им един спрямо друг.

setPin(x,y,newPin)

Задава пин да избраният модул. *m_Libs[matX][matY].setPin(newPin)*.

void applyMatrix(const Cell (&newMatrix)[CELLS_IN_X][CELLS_IN_X)

Разделя голямата матрица на сегменти и ги изпраща на CellMatrix::applyMatrix() с подходящи StartX и StartY.

void clear()

Обходжа и извиква CellMatrix::clear() на всички елементи в член данната m_Libs;

клас Button

Класът има за цел да работи с бутоните на системата. В него има единствена член данна **m_pinOut** пазеща пиннът към който е свързао четенето на бутона. Дефолтната стойност е (-1)

int get()

Връща digitalRead(m_pinOut) т.е. 1 Ако е натиснат и 0 ако не е. При невъведен пин се връща (-1).

void declarePinMode()

задава m_pinOut да е OUTPUT в към микроконтролера.

void setPin(int newPin)

Сетър за пинът на бутона. След присвояване на стойноста се извиква declarePinMode();

Button()

Дефолтен конструктор задаващ стойност (-1) на пина.

Button(int newPin)

Конструктор извикващ setPin(newPin).

клас Joystick

Класът има за цел да работи с Джойстик. В него 3 член данни запазващи 3те пина ,от които се чете позиция X ,позиция У и дали е натиснато вграденото му копче.

void setPins(int newPin, int newX, int newY)

Задава стойности на съответните член данни и извиква declarePins().

declarePins()

Обявава на микроконтролера ,че 3те пинна са INPUT.

int getX() / **int getY()**

Връща (-1) ако пиннът не е зададен и analogRead(m_pinX) ако е. (Еквивалентно за m_pinY). Стойностите са от 0 до 1023.

int getD()

Връща (-1), ако не е зададен пиннът и 0 или 1 на база дали е натиснат вътрешният бутон на джойстика.

Joystick()

Дефолтен конструктор задаващ всички член данни за пинова да са (-1).

Joystick(int newPin, int newX, int newY)

Конструктор викващ setPins(newPin, newX, newY).

клас Inputs

Класът има за цел да управлява множество от бутони и джойстици едновременно. В него има две статични константи определящи броят на бутоните и броя на джойстиците.

В класа фигурират елементи от клас Button и Joystick.

BUTTON_COUNT ,JOYSTICK_COUNT

Класа има и две член данни ,а именно масив от Бутони и масив от Джойстици.

void setButtonPin(int newPin, int id) /

void setJoystickPin(int newPin, int newX, int newY, int id)

Функции имащи за цел да въведат пиновете на бутона/джойстика. Параметърът "id" показва кой е индекса на съответно Джойстика или Бутона в m_JoystickArr или m_ButtonArr. Ако се даде id по голямо от броя им (Определено от статичната константа) се дава стойност -1.

int getButton(int id);

int getJoystickX(int id);

int getJoystickY(int id);

int getJoystickD(int id);

Фукнции извикващи съответните им на бутоните или джойстиците.Id представлява иднекса на сензора. Ако Id е невалидно връща -1.

клас Arcade

Основният клас на системата. Класът има за цел лесно управление на входните и изходните устройства. Той е и просто структориран с цел лено писане на игри/програми.

pinLayouts.hpp

Файл съдържащ #define-ове за всички пинове.

Статичните константи са:

MATRIX_SIZE_X = Monitor::CELLS_IN_X

MATRIX_SIZE_Y = Monitor::CELLS_IN_Y

BUTTON_COUNT = Inputs::BUTTON_COUNT

JOYSTICK_COUNT = Inputs::JOYSTICK_COUNT

Определящи съответно размера на екрана и броя на входните усторйства.

void pinDeclaration()

Събрано на едно място всички декларации на всички пинове на всички входни и изходни устройства. Използва #define-овете от **pinLayouts.hpp**

Arcade()

Дефолтен конструктор. Извика pinDeclaration().

int button(int id)

Взима стойсноста на бутона с индекс (id). Ако стойноста е -1 значи има недефиниран пин или невалидно id.

int joystickX(int id)

int joystickY(int id)

Връщат позицията на джойстика с индекс id в интервал от 0 до 1023. Ако чиското е -1 значи пинът не е деклариран или id-то е грешно.

int joystickD(int id)

Връщат далие натиснат вътрешният бутон на джойстика с индекс id.Ако е 1 е натиснат , ако е 0 не е .Ако чиското е -1 значи пинът не е деклариран или id-то е грешно.

void show(const Cell (&newMatrix)[MATRIX_SIZE_X][MATRIX_SIZE_Y])const;

Печата матрицата върху екрана на играта.

void clear()

Слага стойностите на всички пиксели да са (0,0,0).

Проблеми повреме на разработка:

Основен проблем на проекта е захранването. Първоначално грешно описани параметри от страна на китайците се опитах да захранвам матриците с 12V резултатът беше че 2 матрици изгоряха. Захова се наложи да взема преубразовател към 5V. В последствие се оказа че един преубразовател не е достатъчен и се наложи да взема 2 и га свържа по две матрици на всеки. Въпреки това съм сложил софтуерна защита матциците да не се светкат с максимална стойност (255,255,255) на всеки пиксел. От една страма е ослепяващо ярко , от друга излъчва голяма топлина, и става undervoltage.

Следващ проблем беше свързването на матриците. Матриците имат възможнст да се свързват една след друга даваща теоретичната възможност да се управляват от един пин. На практика Един пин няма достатъчно силен сигнал да стигне до всичките матрици, а

само до 2. Поради улеснение в програмирането съм свързал матриците всяка на отделен пин.

Друг порблем с матриците е че от различни поръчки матриците светеха различно. Т. е. Имаше разминаване на цветовете. Има 3 матрици които не съм броил към цената напълно работещи но забележимо различни от сегашните.

Микроконтролера. Ардуино Uno е крайно неподходящ и като памет в него и като мощност за изчисление.

Джойстиците имат настоящ проблем с това че вътрешното им натискане неработи. (Предполаам фабричен дефект).

Библиотеката беше пренаписвана с цел намаляне на използваната памет. Предшната версия имаше член данна ,която пазеше всяка стойност на матрицита.

Идеи за подобрение:

Добавяне на няколко бъзера за мелодия повреме на игра.

Добавяне на четец на SD-карти. Идеята е вместо да се запазват игрите в кода на аркадата да може всяка карта да е отделна игра симулираща действието на смяна на касети/дискове с игри. Допълнително може да се запазват така резултати/рекорди на играчите ,нещо което понастоящем ще е ограничето предвид иначе ограничената памет на микроконтролера.

Захранване от батеии. Батерии от 3.7 волта ,които ще позволят аркадата да работи без контакт. Двата варианта на захранване трбява да са съвместими (Тоест и двата да ги има като избор).

Добавяне на библиотека за програмиста който работи по игрите. Често срещани функции или алгорити. Пример: функция чертаеща квадрати или триъгилници по екрана, функция засичаща време между натискания на бутон. Функция намираща радиуса от позицията на джойстика до центъра.