**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

**КУРСОВ ПРОЕКТ**

*ПО ПРАКТИКУМ ПО ПРОГРАМИРАНЕ НА МИКРОКОНТРОЛЕРИ*

Студент: Одобрил:

Илиян Антов доц. д-р инж.

Фак. №: 101220020 Серафим Табаков

София

2023 г.

**Задание:**

**Да се разработи програмен код за микроконтролер TI MSP430FR6989 (в комбинация с развойната платка LaunchPad development kit и booster board платката BOOSTXL-EDUMKII), позволяващ използването му като контролер за компютърна игра:**

1. Да се прочитат и обработват по подходящ начин състоянията на:

* Джойстика (позиция по X и по Y)
* Бутоните (двата големи бутона на booster board платката + бутона на джойстика)

1. Информацията от контролера да се изпраща по подходящ начин към компютърна програма чрез серийна комуникация през UART интерфейс.
2. Да се осигури възможност за получаване на информация от компютърна програма (например резултат от играта) чрез серийна комуникация през UART интерфейс и получената информация да се записва и визуализира на LCD екрана на booster board платката.

**Да се разработи компютърна игра “Space Invaders”, демонстрираща възможностите на микроконтролерното устройство:**

1. Да се осигури възможност за свързване към микроконтролерно устройство посредством серийна връзка през UART интерфейса.
2. Да се създадат основните модули на играта:

* Играч (Player)
* Противник (Enemy)
* Снаряд (Projectile)
* Ниво (Level)

1. Да се създадат няколко демонстративни нива чрез свързване на модулите по подходящ начин и добавяне на меню за край на играта.
2. Получената информация от контролера да се прочита и обработва, като се преобразува в команди, които управляват позицията и състоянието на Играча (Player).
3. Резултът от играта да се изпраща към контролера за визуализация при всяка негова промяна (при успешно приключване на нивото, при край на играта).

**Съдържание**

[I. Увод 4](#_Toc134319538)

[II. Обобщен блоков алгоритъм 5](#_Toc134319539)

[III. Ръководство за потребителя 6](#_Toc134319540)

[1. Подготовка за стартиране на играта 6](#_Toc134319541)

[2. Стартиране на играта 6](#_Toc134319542)

[3. Геймплей 7](#_Toc134319543)

[4. Край на играта 8](#_Toc134319544)

[IV. Ръководство за програмиста 9](#_Toc134319545)

[1. Функции на микроконтролерната програма 9](#_Toc134319546)

[2. Функции на компютърната програма 10](#_Toc134319547)

[2.1. Модул “Controller” 10](#_Toc134319550)

[2.2. Модул “Game” 11](#_Toc134319551)

[2.3. Модул “Projectile” 11](#_Toc134319552)

[2.4. Модул “Player” 12](#_Toc134319553)

[2.5. Модул “Enemy” 12](#_Toc134319554)

[2.6. Модул “Level” 13](#_Toc134319555)

[2.7. Главна програма (Модул “Main”) 15](#_Toc134319556)

[V. Програмен код 16](#_Toc134319557)

[1. Програмен код на микроконтролера 16](#_Toc134319558)

[2. Програмен код на компютърната програма 21](#_Toc134319559)

[2.1. Модул “Controller” 21](#_Toc134319562)

[2.2. Модул “Game” 23](#_Toc134319563)

[2.3. Модул “Projectile” 23](#_Toc134319564)

[2.4. Модул “Player” 24](#_Toc134319565)

[2.5. Модул “Enemy” 26](#_Toc134319566)

[2.6. Модул “Level” 27](#_Toc134319567)

[2.7. Главна програма (Модул “Main”) 32](#_Toc134319568)

# Увод

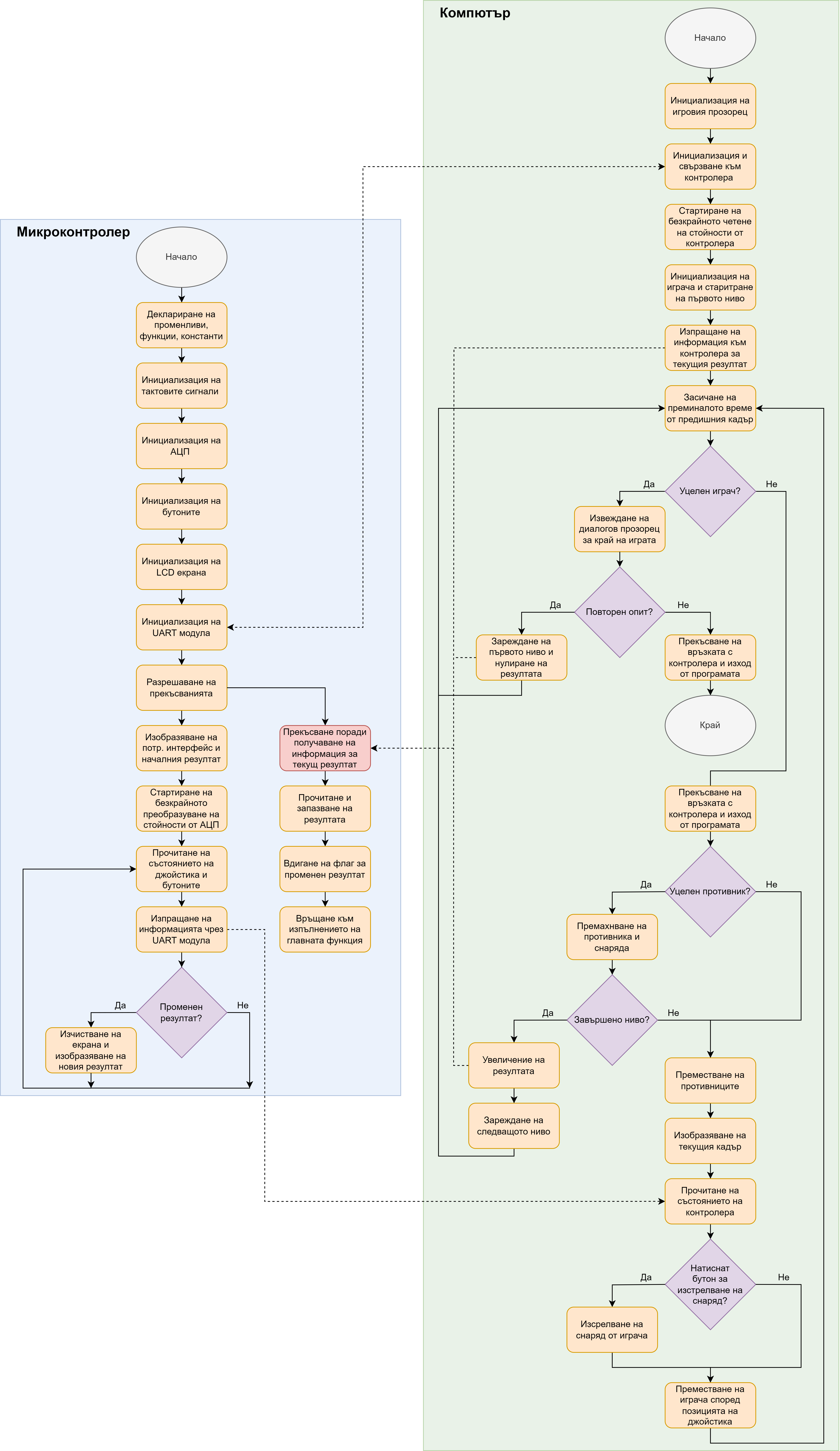
В днешни дни, игровите контролери са може би най-разпространеният аксесоар в индустрията на видеоигрите. На практика всяка игрова конзола е снабдена с поне един контролер, а в много случаи и с по няколко. Производителите предалагат вариации на тези продукти и самостоятелно, като предоставят възможност за използването им и с компютри, мобилни устройства и т.н.

Настоящият курсов проект има за цел реализацията на един такъв контролер, подходящ за използване с компютър. Контролерът ще бъде снабден с джойстик, бутони и LCD екран, на който може да се визуализира информация за играта. Свързването на контролера към компютъра ще става посредством USB кабел. Обменът на информация ще се извършва посредсвом серийния UART интерфейс.

За целите на устройството е използван микроконтролерът MSP430FR6989. За улеснение на разработката е използвана развойната платка LaunchPad development kit, която осигурява лесно свързване към изводите на микроконтролера и предоставя някои допълнителни функционалности (като например JTAG модул за лесно зареждане на фърмуеър и дебъг). Използвана е също booster board платката BOOSTXL-EDUMKII, която е снабдена с нужните за реализация на устройството модули – джойстик с бутон, два специално изведени тактилни бутона и вграден LCD екран.

За демонстрация на работата на контролера ще бъде реализирана компютърна игра “Space Invaders”. Тя ще се състои от няколко демонстративни нива, като всяко от нивата ще съдържа играч, управляван от контролера, както и противници, управлявани от самата игра. Посоката и скоростта на движение на играча ще се определят според положението на джойстика на контролера. Изстрелването на снаряди от играча ще става при натискане на един от бутоните на контролера. При стартиране на играта ще се извежда диалогов прозорец, позволяващ на потребителя да избере контролера, към който желае да се свърже. При успешно завършване на нивото, играта ще изпраща текущия резултат към микроконтролера, който ще го визуализира на своя LCD екран.

# Обобщен блоков алгоритъм



# Ръководство за потребителя

## Подготовка за стартиране на играта

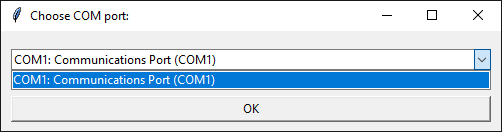
Преди да стартира играта, потребителят трябва да свърже посредсвом USB кабел игровия контролер (MSP430FR6989 LaunchPad с прикачен BOOSTXL-EDUMKII модул) към компътъра и да изчака инициализацията на всички модули на микроконтролера. Инициализацията е приключила, когато на LCD дисплея на контролера се изобрази потребителския интерфейс (фиг. 3.1.)



**Фиг. 3.1.** Инициализация на контролера

## Стартиране на играта

Играта се стартира чрез изпълнение на файла *main.py*. След стартиране, на екрана се визуализира диалогов прозорец, чрез който потребителят трябва да избере комуникационния порт, към който е свързан контролера (фиг. 3.2.). Това става чрез избор на една от опциите от пaдащото меню и натискане на бутона OK.



**Фиг. 3.2.** Диалогов прозорец за избор на комуникационен порт

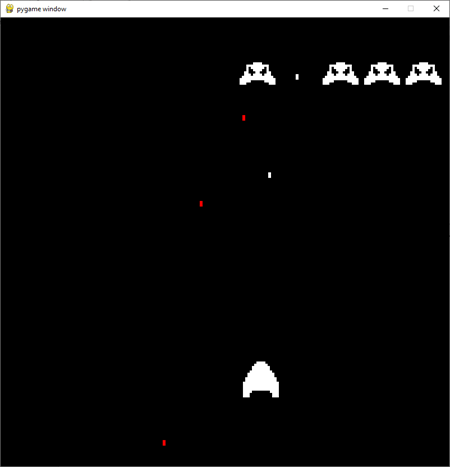
## Геймплей

След успешно свързване с контролера, на екрана се изобразява игровия прозорец със заредено първо ниво и играта започва (фиг. 3.3.). Противниците са разположени в горната част на екрана и се придвижват наляво-надясно и надолу към играча. Всеки от противниците периодично изстрелва червен снаряд към играча.



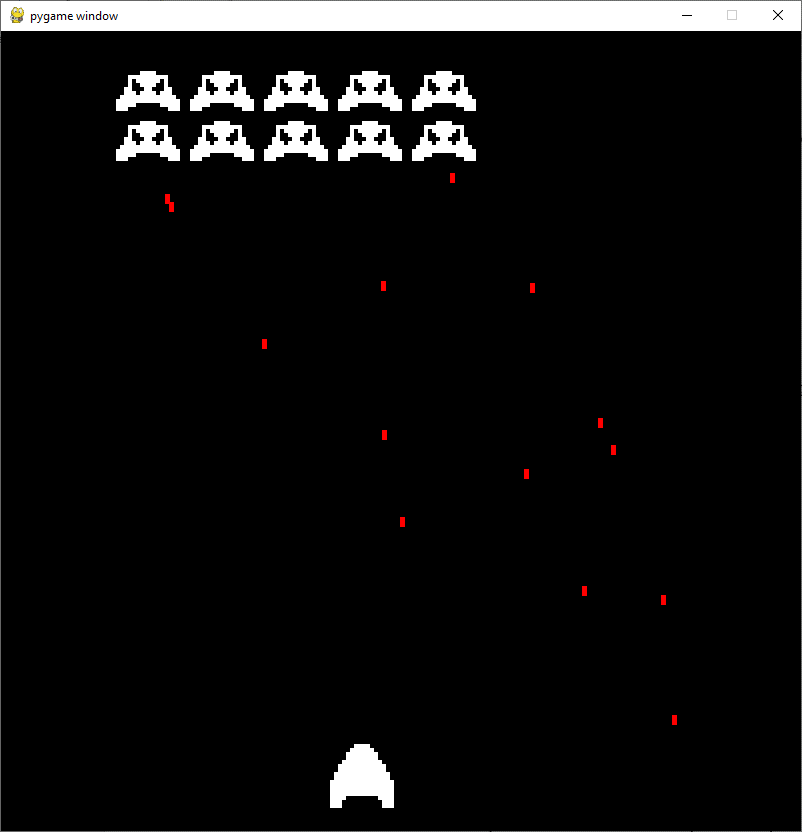
**Фиг. 3.3.** Първо ниво на компютърната игра

Управлението на играча става чрез джойстика на контролера. При натискане на бутона S2, играчът изстрелва бял снаряд нагоре към противниците. При уцелване на който и да от противниците, той се унищожава (фиг. 3.4.).



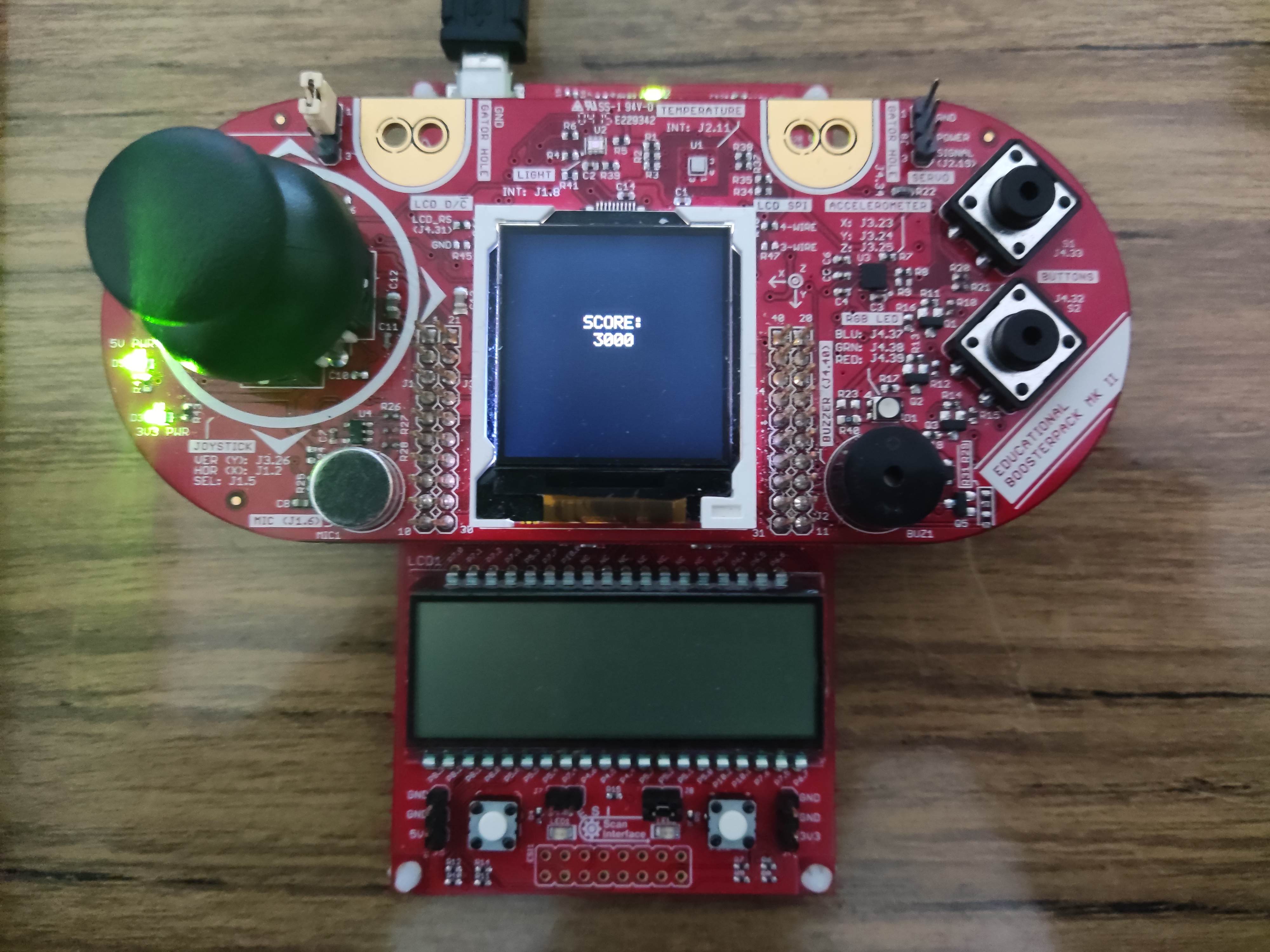
**Фиг. 3.4.** Изстрелване на снаряди от играча

Ако всички противници в настоящото ниво бъдат унищожени, автоматично се зарежда следващото и играта продължава (фиг. 3.5.)



**Фиг. 3.5.** Второ ниво на компютърната игра

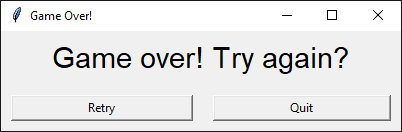
При зареждане на първото ниво, резултатът започва от 0 точки. Всяко успешно завършено ниво увеличава този резултат с 1000 точки \* номера на нивото (1000 точки за първо ниво, 2000 точки за второ ниво и т.н.). Точките, събрани от настоящата игра, са винаги изобразени на LCD екрана на контролера (фиг. 3.6.)



**Фиг. 3.6.** Резултат от играта

## Край на играта

Ако играчът се сблъска с противнически (червен) снаряд или директно в противник, играта приключва и на екрана се изобразява диалогов прозорец за край на играта (фиг. 3.7.). Играчът може да избере дали желае да опита отново (чрез бутон Retry) или да изключи играта (чрез бутон Quit).



**Фиг. 3.7.** Диалогов прозорец за край на играта

# Ръководство за програмиста

## Функции на микроконтролерната програма

void init\_clock\_system(void)

* Инициализира цифровия осцилатор с честота 16MHz и го задава като източник на SMCLK

void init\_ADC(void)

* Инициализира аналоговите изводи за X и Y осите на джойстика
* Конфигурира и разрешава аналогово-цифровия преобразувател ADC12\_B

void init\_buttons(void)

* Инициализира изводите на микроконтролера, към които са свързани бутоните

void init\_LCD(void)

* Инициализира LCD екрана на booster board платката
* Пресмята размера на полето, в което ще се изобразява резултата от играта

void init\_UART(void)

* Инициализира UART модула на микроконтролера
* Разрешава прекъсванията при получаване на информация по UART

void init(void)

* Забранява стражевия таймер
* Изключва високото импедансно състояние на изводите на микроконтролера
* Извиква всички останали (разгледани по-горе) инициализационни функции
* Разрешава прекъсванията глобално

void display\_UI(void)

* Изобразява потребителския интерфейс върху LCD екрана

void display\_score(void)

* Изчертава черен правоъгълник върху визуализирания в момента резултат, като по този начин го изтрива
* Изобразява записания в променливата score резултат

int main(void)

* Инициализира микроконтролера
* Изобразява потребителския интерфейс върху LCD екрана
* Изобразява първоначалния резултат върху LCD екрана
* Стартира четенето на стойности от АЦП в режим repeated sequence of channels (няколко канала, многократно)
* В безкраен цикъл извършва следното:
  + Прочита регистрите на АЦП, в които е запазена информацията за позицията на джойстика по X и Y
  + Прочита състоянията на изводите на микроконтролера, към които са свързани бутоните
  + Изпраща събраната информация към компютърното приложение посредсвом UART интерфейса
  + Ако резултатът от играта е бил променен от последната итерация на цикъла, изтрива предишния резултат и визуализира новия

\_\_interrupt void USCI\_A1\_ISR(void)

* Представлява прекъсване (interrupt), което се изпълнява при получаване на стартов бит на входящия извод на UART интерфейса
* Прочита флага за прекъсването
* Ако флагът представлява стартов бит, прочита входящата информация от UART интерфейса и я записва в променливата score
* Вдига флагът за наличие на промяна в резултата
* Изчиства флага за прекъсване

## Функции на компютърната програма



### Модул “Controller”

class Controller

* Служи за свързване и комуникация с микроконтролерната програма

def \_\_init\_\_(self)

* Инициализира променливите на инстанцията на класа

def connect() -> serial.Serial

* Изобразява диалогов прозорец за избор на COM порта, към който е свързан контролерът
* Осъществява връзка с контролера на избрания от потребителя порт
* Връща обект, представляващ въпросната връзка

def disconnect(self)

* Прекратява връзката с микроконтролера

def read\_input(self)

* В безкраен цикъл извършва следното:
  + Ако флагът за прекъсване на четенето е вдигнат, прекратява изпълнението си
  + Прочита входящата информация от UART интерфейса
  + Обработва получената информация и я запазва в съответните променливи, отразяващи текущото състояние на контролера

def write\_score(self, score: int)

* Изпраща резултата от играта (променлива score) към микроконтролера посреством серийната връзка

### Модул “Game”

class Game

* Служи за задаване на характеристиките на играта
* Задава максималния брой кадри в секунда (fps)
* Задава размера на игровия прозорец
* Задава шанса на всеки противник да изстреля снаряд по време на кадъра
* Създава игровия прозорец

### Модул “Projectile”

class Projectile

* Служи за представяне на снаряд

def \_\_init\_\_(self, pos\_x: int, pos\_y: int)

* Инициализира променливите на инстанцията на класа
* Задава началната позиция на снаряда по X и Y в зависимост от параметрите pos\_x и pos\_y

def draw(self)

* Изобразява снаряда върху игровия прозорец

def move(self, frame\_time: int, direction: str) -> bool

* Премества снаряда нагоре или надолу (в зависимост от параметъра direction), като отчита изминалото време от предишния кадър (параметър frame\_time), за да се осигури постоянна скорост на движение, независима от броя кадри в секунда
* Връща True ако снарядът е напуснал игралния прозорец, в противен случай връща False

### Модул “Player”

class Player(pygame.sprite.Sprite)

* Служи за представяне на играч

def \_\_init\_\_(self)

* Инициализира променливите на инстанцията на класа

def draw(self, frame\_time: int)

* Премества нагоре всеки от снарядите, които играчът е изстрелял
* Ако някой от снарядите е напуснал игровия прозорец, го унищожава
* Изобразява снарядите върху игровия прозорец
* Изобразява играча върху игровия прозорец

def move(self, dx: int, dy: int, frame\_time: int)

* Пресмята посоката и скоростта на движение на играча в зависимост от положението на джойстика по X (параметър dx) и по Y (параметър dy)
* Премества играча, като отчита изминалото време от предишния кадър (параметър frame\_time), за да се осигури постоянна скорост на движение, независима от броя кадри в секунда

def shoot(self)

* Създава снаряд в горната част на модела на играча и го добавя към списъка с изстреляни снаряди

### Модул “Enemy”

class Enemy(pygame.sprite.Sprite)

* Служи за представяне на противник

def \_\_init\_\_(self, spawn\_x: int, spawn\_y: int)

* Инициализира променливите на инстанцията на класа
* Задава началната позиция на противника по X и Y в зависимост от параметрите spawn\_x и spawn\_y

def draw(self, frame\_time: int)

* Генерира случайно число, което определя дали по време на този кадър въпросният противник ще изстреля снаряд и ако да, създава снаряд в долната част на модела на противника и го добавя към списъка с изстреляни снаряди
* Премества надолу всеки от снарядите, които противникът е изстрелял
* Ако някой от снарядите е напуснал игровия прозорец, го унищожава
* Изобразява снарядите върху игровия прозорец
* Изобразява противника върху игровия прозорец

def move(self, dx: int, dy: int, frame\_time: int)

* Пресмята посоката и скоростта на движение на противника в зависимост от параметрите dx и dy
* Премества противника, като отчита изминалото време от предишния кадър (параметър frame\_time), за да се осигури постоянна скорост на движение, независима от броя кадри в секунда

### Модул “Level”

class Level

* Служи за представяне на ниво от играта

def \_\_init\_\_(self, number: int)

* Инициализира променливите на инстанцията на класа
* Създава списък с противници за съответния номер ниво (параметър number)

def draw(self)

* Изобразява всички противници в нивото върху игровия прозорец

def spawn\_enemies(level\_number: int, enemy\_row\_length: int) -> list[Enemy]

* Създава списък с противници и ги разполага по подходящ начин върху игралното поле
* Общият брой противници зависи от номера на нивото (параметър level\_number), а броят противници на ред се определя от параметъра enemy\_row\_length
* Връща генерираният списък с противници за съответното ниво

def move\_enemies(self, frame\_time: int)

* Премества всеки от противниците в нивото към текущата посока на движение
* Ако някой от противниците е достигнал левия или десния край на игровия прозорец, премества всички противници един ред надолу и обръща посоката на движение

def \_find\_leftmost\_enemy\_x(self) -> int

* Връща координатите по X на най-левия (все още неразрушен) противник в нивото

def \_find\_rightmost\_enemy\_x(self) -> int

* Връща координатите по X на най-десния (все още неразрушен) противник в нивото

def check\_enemy\_hit(self, projectile: Projectile) -> bool

* Проверява дали някой от противниците в нивото е уцелен от даден снаряд (параметър projectile) и ако да, премахва противника от нивото
* Връща True ако е открит уцелен противник и False в противен случай

def check\_player\_hit(self, player: Player) -> bool

* Проверява дали даден играч (параметър player) е уцелен от който и да било противников снаряд и ако да, връща True
* Проверява дали играчът не се е сблъскъл с който и да било противник и ако е, връща True
* Ако играчът не е бил уцелен от нищо, връща False

def check\_completion(self) -> bool

* Връща True ако списъкът с противници в нивото е празен и False в противен случай

def game\_over\_dialog()

* Изобразява диалогов прозорец за край на играта
* Ако потребителят избере бутона Quit, прекъсва серийната комуникация с микроконтролера и изключва програмата

def game\_over(tkinter\_root, controller: Controller)

* Затваря диалоговия прозорец за край на играта
* Спира четенето на информация от UART интерфейса
* Прекъсва връзката с микроконтролера
* Затваря игровия прозорец
* Изключва програмата

### Главна програма (Модул “Main”)

def main()

* Изпълнява се при стартиране на програмата
* Инициализира игровия прозорец и таймера за време на кадрите
* Свързва се към микроконтролерното устройство
* Създава нишка, която паралелно чете постъпващата на серийния UART интерфейс информация от контролера
* Създава играч
* Зарежда първото ниво
* Изпраща първоначалния резултат от играта към контролера
* В безкраен цикъл извършва следното:
  + Записва изминалото време от предишния кадър
  + Проверява дали играчът не е бил уцелен и ако да, изобразява диалогов прозорец за край на играта
  + Проверява дали някой от противниците не бил уцелен и ако да, го премахва от нивото
  + Проверява дали всички противници в нивото са били уцелени и ако да, повишава резултата, изпраща го към контролера и зарежда следващото ниво
  + Изтрива предишния кадър
  + Премества всички противници в нивото
  + Изобразява всички противници и противникови снаряди върху игровия прозорец
  + Изобразява играча и всички негови снаряди върху игровия прозорец
  + Изобразява кадъра
  + Проверява дали не е натиснат бутона за изход от програмата и ако да, изключва програмата
  + Проверява дали бутонът за изстрелване на снаряд на контролера е натиснат и ако да, изстрелва снаряд
  + Прочита състоянията на бутоните на контролера
  + Прочита позицията на джойстика на контролера
  + Интерполира позицията на джойстика към стойности на отместване по X и Y на играча
  + Премества играча

# Програмен код

## Програмен код на микроконтролера

#include <stdio.h>

#include <driverlib.h>

#include <math.h>

#include "grlib.h"

#include "Crystalfontz128x128\_ST7735.h"

#include "uartstdio.h"

#define MAX\_STR\_SIZE    20

volatile char score[20] = {"#####"};

volatile int score\_changed = 0;

/\* Graphic library context \*/

Graphics\_Context g\_sContext;

Graphics\_Rectangle score\_text\_box;

void init\_clock\_system(void) {

    CS\_setDCOFreq(CS\_DCORSEL\_1, CS\_DCOFSEL\_4); //DCO = 16 MHz

    CS\_initClockSignal(CS\_SMCLK, CS\_DCOCLK\_SELECT, CS\_CLOCK\_DIVIDER\_1); //SMCLK = DCO clock

}

void init\_ADC(void) {

    // Setup the analog pin connected to the X axis of the joystick

    GPIO\_setAsInputPin(GPIO\_PORT\_P9, GPIO\_PIN2);

    GPIO\_setAsPeripheralModuleFunctionInputPin(GPIO\_PORT\_P9, GPIO\_PIN2, GPIO\_TERNARY\_MODULE\_FUNCTION);

    // Setup the analog pin connected to the Y axis of the joystick

    GPIO\_setAsInputPin(GPIO\_PORT\_P8, GPIO\_PIN7);

    GPIO\_setAsPeripheralModuleFunctionInputPin(GPIO\_PORT\_P8, GPIO\_PIN7, GPIO\_TERNARY\_MODULE\_FUNCTION);

    // Initialize the ADC

    ADC12\_B\_initParam adc\_init\_struct;

    adc\_init\_struct.sampleHoldSignalSourceSelect = ADC12\_B\_SAMPLEHOLDSOURCE\_SC;

    adc\_init\_struct.clockSourceSelect = ADC12\_B\_CLOCKSOURCE\_SMCLK;

    adc\_init\_struct.clockSourceDivider = ADC12\_B\_CLOCKDIVIDER\_1;

    adc\_init\_struct.clockSourcePredivider = ADC12\_B\_CLOCKPREDIVIDER\_\_1;

    adc\_init\_struct.internalChannelMap = ADC12\_B\_MAPINTCH0;

    ADC12\_B\_init(ADC12\_B\_BASE, &adc\_init\_struct);

    // Enable the ADC

    ADC12\_B\_enable(ADC12\_B\_BASE);

    // Setup the sample and hold module

    ADC12\_B\_setupSamplingTimer(ADC12\_B\_BASE,

                               ADC12\_B\_CYCLEHOLD\_16\_CYCLES,

                               ADC12\_B\_CYCLEHOLD\_4\_CYCLES,

                               ADC12\_B\_MULTIPLESAMPLESENABLE);

    // Setup the memory registers that will store X axis values

    ADC12\_B\_configureMemoryParam XMemoryParam = {0};

    XMemoryParam.memoryBufferControlIndex = ADC12\_B\_MEMORY\_0;

    XMemoryParam.inputSourceSelect = ADC12\_B\_INPUT\_A10;

    XMemoryParam.refVoltageSourceSelect = ADC12\_B\_VREFPOS\_AVCC\_VREFNEG\_VSS;

    XMemoryParam.endOfSequence = ADC12\_B\_NOTENDOFSEQUENCE;

    XMemoryParam.windowComparatorSelect = ADC12\_B\_WINDOW\_COMPARATOR\_DISABLE;

    XMemoryParam.differentialModeSelect = ADC12\_B\_DIFFERENTIAL\_MODE\_DISABLE;

    ADC12\_B\_configureMemory(ADC12\_B\_BASE, &XMemoryParam);

    // Setup the memory registers that will store Y axis values

    ADC12\_B\_configureMemoryParam YMemoryParam = {0};

    YMemoryParam.memoryBufferControlIndex = ADC12\_B\_MEMORY\_1;

    YMemoryParam.inputSourceSelect = ADC12\_B\_INPUT\_A4;

    YMemoryParam.refVoltageSourceSelect = ADC12\_B\_VREFPOS\_AVCC\_VREFNEG\_VSS;

    YMemoryParam.endOfSequence = ADC12\_B\_ENDOFSEQUENCE;

    YMemoryParam.windowComparatorSelect = ADC12\_B\_WINDOW\_COMPARATOR\_DISABLE;

    XMemoryParam.differentialModeSelect = ADC12\_B\_DIFFERENTIAL\_MODE\_DISABLE;

    ADC12\_B\_configureMemory(ADC12\_B\_BASE, &YMemoryParam);

}

void init\_buttons(void) {

    // S1

    GPIO\_setAsInputPin(GPIO\_PORT\_P3, GPIO\_PIN0);

    // S2

    GPIO\_setAsInputPin(GPIO\_PORT\_P3, GPIO\_PIN1);

    // Joystick button

    GPIO\_setAsInputPin(GPIO\_PORT\_P3, GPIO\_PIN2);

}

void init\_LCD(void) {

    Crystalfontz128x128\_Init(); // Initializes display driver

    Crystalfontz128x128\_SetOrientation(LCD\_ORIENTATION\_UP); // Set default screen orientation

    Graphics\_initContext(&g\_sContext, &g\_sCrystalfontz128x128); // Initializes graphics context

    Graphics\_setForegroundColor(&g\_sContext, GRAPHICS\_COLOR\_WHITE);

    Graphics\_setBackgroundColor(&g\_sContext, GRAPHICS\_COLOR\_BLACK);

    GrContextFontSet(&g\_sContext, &g\_sFontFixed6x8);

    Graphics\_clearDisplay(&g\_sContext);

    score\_text\_box.xMin = 0;

    score\_text\_box.xMax = 128;

    score\_text\_box.yMin = floor(60 - ((float)Graphics\_getStringHeight(&g\_sContext) / 2));

    score\_text\_box.yMax = ceil(60 + ((float)Graphics\_getStringHeight(&g\_sContext) / 2));

}

void init\_UART(void) {

    // Initialize the UART module

    UARTStdioInit();

    // Enable and clear RX interrupts

    EUSCI\_A\_UART\_enableInterrupt(EUSCI\_A1\_BASE, EUSCI\_A\_UART\_STARTBIT\_INTERRUPT);

    EUSCI\_A\_UART\_clearInterrupt(EUSCI\_A1\_BASE, EUSCI\_A\_UART\_STARTBIT\_INTERRUPT\_FLAG);

}

void init(void) {

    // Stop the watchdog timer

    WDT\_A\_hold(WDT\_A\_BASE);

    // Disable the GPIO power-on default high-impedance mode to activate previously configured port settings

    PMM\_unlockLPM5();

    FRAMCtl\_configureWaitStateControl(FRAMCTL\_ACCESS\_TIME\_CYCLES\_1); //Needed for DCO = 16 MHz

    // Initialize the clock system

    init\_clock\_system();

    // Initialize the ADC used for reading joystick values

    init\_ADC();

    // Initialize all buttons

    init\_buttons();

    // Initialize the LCD screen

    init\_LCD();

    // Initialize the UART module user for sending and receiving information from the computer app

    init\_UART();

    // Enable interrupts globally

    \_\_enable\_interrupt();

}

void display\_UI() {

    // Clear the display

    Graphics\_clearDisplay(&g\_sContext);

    // Display the basic UI

    Graphics\_drawStringCentered(&g\_sContext, (int8\_t \*)"SCORE:", AUTO\_STRING\_LENGTH, 64, 50, OPAQUE\_TEXT);

}

void display\_score() {

    // Draw a black box over the old score

    Graphics\_setForegroundColor(&g\_sContext, GRAPHICS\_COLOR\_BLACK);

    Graphics\_fillRectangle(&g\_sContext, &score\_text\_box);

    // Display the current score

    Graphics\_setForegroundColor(&g\_sContext, GRAPHICS\_COLOR\_WHITE);

    Graphics\_drawStringCentered(&g\_sContext, (int8\_t \*)score, AUTO\_STRING\_LENGTH, 64, 60, OPAQUE\_TEXT);

}

int main(void) {

    // Initialize all modules

    init();

    // Display the LCD UI

    display\_UI();

    // Display the initial score

    display\_score();

    // Start reading joystick X and Y values repeatedly

    ADC12\_B\_startConversion(ADC12\_B\_BASE,

                            ADC12\_B\_MEMORY\_0,

                            ADC12\_B\_REPEATED\_SEQOFCHANNELS);

    while(1) {

        // Extract the current controller state

        long joystickX = ADC12\_B\_getResults(ADC12\_B\_BASE, ADC12\_B\_MEMORY\_0);

        long joystickY = ADC12\_B\_getResults(ADC12\_B\_BASE, ADC12\_B\_MEMORY\_1);

        long button1 = GPIO\_getInputPinValue(GPIO\_PORT\_P3, GPIO\_PIN0);

        long button2 = GPIO\_getInputPinValue(GPIO\_PORT\_P3, GPIO\_PIN1);

        long buttonj = GPIO\_getInputPinValue(GPIO\_PORT\_P3, GPIO\_PIN2);

        // Create a standard string with the current controller state and send it via the UART interface

        UARTprintf("x:\%d y:\%d but1:\%d but2:\%d butj:\%d \n\r", joystickX, joystickY, button1, button2, buttonj);

        // If the score has changed since the last iteration, display the new value and reset the flag

        if(score\_changed) {

            display\_score();

            score\_changed = 0;

        }

        \_\_delay\_cycles(1000);

    }

}

#pragma vector = USCI\_A1\_VECTOR

\_\_interrupt void USCI\_A1\_ISR(void) {

    // Get the UART RX interrupt status

    uint16\_t rx\_status;

    rx\_status = EUSCI\_A\_UART\_getInterruptStatus(EUSCI\_A1\_BASE, EUSCI\_A\_UART\_STARTBIT\_INTERRUPT\_FLAG);

    // If there is incoming data, read and save it to the score variable

    if(rx\_status) {

        UARTgets(score, 20);

    }

    // Raise the score change flag

    score\_changed = 1;

    \_\_delay\_cycles(20000);

    // Clear the UART RX interrupt

    EUSCI\_A\_UART\_clearInterrupt(EUSCI\_A1\_BASE, EUSCI\_A\_UART\_STARTBIT\_INTERRUPT\_FLAG);

}

## Програмен код на компютърната програма



### Модул “Controller”

import tkinter

from tkinter import \*

from tkinter.ttk import Combobox

import serial

import serial.tools.list\_ports

class Controller:

    def \_\_init\_\_(self):

        self.connection = Controller.connect()

        self.x = 2048

        self.y = 2048

        self.but1 = 0

        self.but2 = 0

        self.butj = 0

        self.stop = False

    @staticmethod

    def connect() -> serial.Serial:

        # Create the dialog box window

        dialog\_root = Tk()

        dialog\_root.title("Choose COM port:")

        dialog\_root.geometry("500x100")

        # Get a list of available serial devices on all COM ports

        available\_comports = serial.tools.list\_ports.comports()

        # Format the COM port devices as "<name>: <description>"

        port\_choices = [(port + ": " + desc) for port, desc, hwid in sorted(available\_comports)]

        port\_choice = StringVar()

        port\_choice.set(port\_choices[1])

        # Create a combo box widget from the list

        combo\_box = Combobox(dialog\_root, values=port\_choices, textvariable=port\_choice, state="readonly")

        combo\_box.pack(padx=10, pady=10, fill=tkinter.X, expand=True)

        # Create a confirmation button

        button = Button(dialog\_root, text="OK", command=dialog\_root.destroy)

        button.pack(padx=10, fill=tkinter.X, expand=True)

        # Display the dialog and wait for confirmation

        dialog\_root.mainloop()

        # Extract the chosen COM port name from the combo box

        target\_port = port\_choice.get().split(":")[0]

        # Establish a connection with the chosen device

        serial\_connection = serial.Serial(port=target\_port,

                                          baudrate=9600,

                                          timeout=1)

        if not serial\_connection.isOpen():

            serial\_connection.open()

        # Return the connection object

        return serial\_connection

    def disconnect(self):

        self.connection.close()

    def read\_input(self):

        # Read the incoming input indefinitely

        while True:

            # If the stop variable is set, exit the method

            if self.stop:

                return

            try:

                # Read one line of input and remove all carriage returns and new lines

                raw\_data = self.connection.read\_until(b"\r\n")

                raw\_data\_str = raw\_data.decode("utf-8").replace("\n", "").replace("\r", "").strip()

                # Convert the input into a list

                raw\_data\_split = raw\_data\_str.split(" ")

                # Convert the input list into dictionary

                raw\_data\_dict = {}

                for raw\_data in raw\_data\_split:

                    name\_value = raw\_data.split(":")

                    raw\_data\_dict[name\_value[0]] = name\_value[1]

                # Set the class properties according to the input

                self.x = int(raw\_data\_dict["x"])

                self.y = int(raw\_data\_dict["y"])

                self.but1 = int(raw\_data\_dict["but1"])

                self.but2 = int(raw\_data\_dict["but2"])

                self.butj = int(raw\_data\_dict["butj"])

            except:

                pass

    def write\_score(self, score: int):

        # Write the game score to the controller

        self.connection.write(data=str(score))

### Модул “Game”

from collections import namedtuple

import pygame

class Game:

    # Set the maximum frames per second for the game

    max\_fps = 1000

    # Set the display size in pixels

    Display = namedtuple("Display", "x y")

    display\_size = Display(800, 800)

    # Set the game score

    score = 0

    # Set the chance for an enemy to shoot (coefficient)

    enemy\_shoot\_chance = 0.001

    # Create the game screen

    screen = pygame.display.set\_mode(size=display\_size)

### Модул “Projectile”

import pygame

from Game import Game

class Projectile:

    default\_size = (5, 10)

    default\_color = (255, 255, 255)

    default\_speed = 400

    def \_\_init\_\_(self, pos\_x: int, pos\_y: int):

        self.size = Projectile.default\_size

        self.color = Projectile.default\_color

        self.speed = Projectile.default\_speed

        self.pos = pygame.math.Vector2(pos\_x, pos\_y)

        self.rect = pygame.rect.Rect(pos\_x, pos\_y, self.size[0], self.size[1])

    def draw(self):

        # Draw the projectile on the screen

        pygame.draw.rect(surface=Game.screen,

                         color=self.color,

                         rect=self.rect)

    def move(self, frame\_time: int, direction: str) -> bool:

        # Move the projectile up based on the movement speed of the entity and the frame time (to ensure constant speed)

if direction == "up":

            self.pos.y -= self.speed \* (frame\_time / 1000)

        # Move the projectile down based on the movement speed of the entity and the frame time (to ensure constant speed)

        elif direction == "down":

            self.pos.y += self.speed \* (frame\_time / 1000)

        # Restrict Y movement to the game screen

        if self.pos.y < 0 or self.pos.y > Game.display\_size.y:

            return True

        # Move the projectile to the new position

        self.rect.top = self.pos.y

        return False

### Модул “Player”

import os

import pygame

from Game import Game

from Projectile import Projectile

class Player(pygame.sprite.Sprite):

    image = pygame.image.load(os.path.join('imgs', 'player.png'))

    default\_size = (16 \* 4, 16 \* 4)

    default\_pos = (368, 700)

    default\_speed = 400

    # Center deadzone for player movement to compensate for controller inaccuracy

    controller\_deadzone = (-50, 50)

    def \_\_init\_\_(self):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.size = Player.default\_size

        self.image = pygame.transform.scale(Player.image, self.size)

        self.speed = Player.default\_speed

        self.projectiles = []

        self.pos = pygame.math.Vector2(Player.default\_pos[0], Player.default\_pos[1])

        self.rect = self.image.get\_rect()

        self.rect.topleft = self.pos.x, self.pos.y

    def draw(self, frame\_time: int):

        # Move all previously shot projectiles up

        for projectile in list(self.projectiles):

            projectile\_destroyed = projectile.move(frame\_time, "up")

            # If the projectile was destroyed (reached the end of the screen/hit enemy), remove it from the list

            if projectile\_destroyed:

                self.projectiles.remove(projectile)

            # Display the projectile

            projectile.draw()

        # Draw the player on the screen

        Game.screen.blit(self.image, self.rect)

    def move(self, dx: int, dy: int, frame\_time: int):

        # If the X movement is within the center deadzone, remove it

        if self.controller\_deadzone[0] < dx < self.controller\_deadzone[1]:

            dx = 0

        # If the Y movement is within the center deadzone, remove it

        if self.controller\_deadzone[0] < dy < self.controller\_deadzone[1]:

            dy = 0

        # Create a vector pointing to the direction of travel

        movement\_vector = pygame.math.Vector2(dx, dy)

        if movement\_vector.length() == 0:

            return

        # Scale the vector based on the movement speed of the entity and the frame time (to ensure constant speed)

        movement\_vector.scale\_to\_length(self.speed \* (frame\_time / 1000))

        # Move the position based on the vector

        self.pos.x += movement\_vector.x

        self.pos.y += movement\_vector.y

        # Restrict X movement to the game screen

        if self.pos.x < 0:

            self.pos.x = 0

        elif self.pos.x > (Game.display\_size.x - self.rect.width):

            self.pos.x = (Game.display\_size.x - self.rect.width)

        # Restrict Y movement to the game screen

        if self.pos.y < 0:

            self.pos.y = 0

        elif self.pos.y > (Game.display\_size.y - self.rect.height):

            self.pos.y = (Game.display\_size.y - self.rect.height)

        # Move the player to the new position

        self.rect.topleft = self.pos

    def shoot(self):

        # Create a white projectile in the top middle of the player

        projectile = Projectile(self.rect.midtop[0], self.rect.midtop[1])

        self.projectiles.append(projectile)

### Модул “Enemy”

import os

import random

import pygame

from Game import Game

from Projectile import Projectile

class Enemy(pygame.sprite.Sprite):

    image = pygame.image.load(os.path.join('imgs', 'enemy.png'))

    default\_speed = 200

    default\_size = (16 \* 4, 10 \* 4)

    default\_pos = (0, 0)

    min\_x = Game.display\_size.x / 5

    def \_\_init\_\_(self, spawn\_x: int, spawn\_y: int):

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.size = Enemy.default\_size

        self.image = pygame.transform.scale(Enemy.image, self.size)

        self.speed = Enemy.default\_speed

        self.projectiles = []

        self.rect = self.image.get\_rect()

        self.rect.topleft = (spawn\_x, spawn\_y)

        self.pos = pygame.math.Vector2(self.rect.x, self.rect.y)

    def draw(self, frame\_time: int):

        # Generate a random number that will determine if the enemy will shoot this frame

        random\_num = random.randint(0, 1 / Game.enemy\_shoot\_chance)

        # Only shoot a projectile if the generated number is 0

        if random\_num == 0:

            # Create a red projectile in the bottom middle of the enemy

            projectile = Projectile(self.rect.midbottom[0], self.rect.midbottom[1])

            projectile.color = (255, 0, 0)

            self.projectiles.append(projectile)

        # Move all previously shot projectiles down

        for projectile in list(self.projectiles):

            projectile\_destroyed = projectile.move(frame\_time, "down")

            # If the projectile was destroyed (reached the end of the screen/hit player), remove it from the list

            if projectile\_destroyed:

                self.projectiles.remove(projectile)

            # Display the projectile

            projectile.draw()

        # Draw the enemy on the screen

        Game.screen.blit(self.image, self.rect)

    def move(self, dx: int, dy: int, frame\_time: int):

        # Create a vector pointing to the direction of travel

        movement\_vector = pygame.math.Vector2(dx, dy)

        if movement\_vector.length() == 0:

            return

        # Scale the vector based on the movement speed of the entity and the frame time (to ensure constant speed)

        movement\_vector.scale\_to\_length(self.speed \* (frame\_time / 1000))

        # Move the position based on the vector

        self.pos.x += movement\_vector.x

        self.pos.y += movement\_vector.y

        # Restrict X movement to the game screen

        if self.pos.x < 0:

            self.pos.x = 0

        elif self.pos.x > (Game.display\_size.x - self.rect.width):

            self.pos.x = (Game.display\_size.x - self.rect.width)

        # Restrict Y movement to the game screen

        if self.pos.y < 0:

            self.pos.y = 0

        elif self.pos.y > (Game.display\_size.y - self.rect.height):

            self.pos.y = (Game.display\_size.y - self.rect.height)

        # Move the enemy to the new position

        self.rect.topleft = self.pos

### Модул “Level”

import tkinter

from tkinter import Tk, Button

import pygame

from Controller import Controller

from Enemy import Enemy

from Game import Game

from Player import Player

from Projectile import Projectile

class Level:

    # Set the offset (in pixels) between enemies in the level

    enemy\_offset = 10

    # Set the maximum number of enemies in a row

    enemy\_row\_length = 5

    def \_\_init\_\_(self, number: int):

        self.num = number

        # Set the initial movement direction of the enemies

        self.movement\_direction = "right"

        # Spawn enemies according to the level number

        self.enemies = self.spawn\_enemies(level\_number=number,

                                          enemy\_row\_length=Level.enemy\_row\_length)

    def draw(self, frame\_time: int):

        # Draw all enemies

        for enemy in self.enemies:

            enemy.draw(frame\_time)

    @staticmethod

    def spawn\_enemies(level\_number: int, enemy\_row\_length: int) -> list[Enemy]:

        # Create an empty list for Enemy objects

        enemies = []

        # Create a row of enemies for each level number

        for i in range(level\_number):

            # Place the first enemy in the row at the minimum X

            current\_x = Enemy.min\_x

            # Place the first enemy in the row at the right Y position depending on the row number

            current\_y = i \* (Enemy.default\_size[1] + Level.enemy\_offset)

            # Create enemy objects for the current row

            for j in range(enemy\_row\_length):

                # Create an Enemy object and add it to the list

                enemy = Enemy(spawn\_x=current\_x,

                              spawn\_y=current\_y)

                enemies.append(enemy)

                # Move the spawn point for the next enemy

                current\_x += Enemy.default\_size[0] + Level.enemy\_offset

        # Return the list of all enemies  in the level

        return enemies

    def move\_enemies(self, frame\_time: int):

        # If the enemies are currently moving right

        if self.movement\_direction == "right":

            # Find the X of the rightmost enemy in the level and spawn a copy of it

            rightmost\_enemy\_x = self.\_find\_rightmost\_enemy\_x()

            rightmost\_enemy = Enemy(spawn\_x=rightmost\_enemy\_x,

                                    spawn\_y=0)

            # Move the copy of the rightmost enemy one more step right

            rightmost\_enemy.move(dx=Enemy.default\_speed,

                                 dy=0,

                                 frame\_time=frame\_time)

            # If the enemies have reached the end of the screen, move them one row down

            if rightmost\_enemy.rect.right >= Game.display\_size.x:

                for enemy in self.enemies:

                    enemy.pos.y += Enemy.default\_size[1]

                    enemy.rect.top = enemy.pos.y

                # Change the direction of enemy travel

                self.movement\_direction = "left"

            # If the enemies have not reached the end of the screen, move them one step right

            else:

                for enemy in self.enemies:

                    enemy.move(dx=Enemy.default\_speed,

                               dy=0,

                               frame\_time=frame\_time)

        # If the enemies are currently moving left

        elif self.movement\_direction == "left":

            # Find the X of the leftmost enemy in the level and spawn a copy of it

            leftmost\_enemy\_x = self.\_find\_leftmost\_enemy\_x()

            leftmost\_enemy = Enemy(spawn\_x=leftmost\_enemy\_x,

                                   spawn\_y=0)

            # Move the copy of the leftmost enemy one more step left

            leftmost\_enemy.move(dx=-Enemy.default\_speed,

                                dy=0,

                                frame\_time=frame\_time)

            # If the enemies have reached the end of the screen, move them one row down

            if leftmost\_enemy.rect.left <= 0:

                for enemy in self.enemies:

                    enemy.pos.y += enemy.default\_size[1]

                    enemy.rect.top = enemy.pos.y

                # Change the direction of enemy travel

                self.movement\_direction = "right"

            # If the enemies have not reached the end of the screen, move them one step left

            else:

                for enemy in self.enemies:

                    enemy.move(dx=-Enemy.default\_speed,

                               dy=0,

                               frame\_time=frame\_time)

    def \_find\_leftmost\_enemy\_x(self) -> int:

        # Find the X of the current leftmost enemy rectangle and return it

        min\_x = min(enemy.rect.x for enemy in self.enemies)

        return min\_x

    def \_find\_rightmost\_enemy\_x(self) -> int:

        # Find the X of the current rightmost enemy rectangle and return it

        max\_x = max(enemy.rect.x for enemy in self.enemies)

        return max\_x

    def check\_enemy\_hit(self, projectile: Projectile) -> bool:

        # Iterate through the list of enemies in the level

        for enemy in self.enemies:

            # If the projectile has collided with an enemy, destroy the enemy and return True

            if enemy.rect.colliderect(projectile):

                self.enemies.remove(enemy)

                return True

        # If no collision was detected, return False

        return False

    def check\_player\_hit(self, player: Player) -> bool:

        # Iterate through the list of enemies in the level

        for enemy in self.enemies:

            # If the player has collided with an enemy, return True

            if enemy.rect.colliderect(player):

                return True

        # Iterate through the list of enemies in the level

        for enemy in self.enemies:

            # Iterate through the list of projectiles for each enemy

            for projectile in enemy.projectiles:

                # If the player has collided with a projectile, return True

                if projectile.rect.colliderect(player):

                    return True

        # If no collision was detected, return False

        return False

    def check\_completion(self) -> bool:

        # If no enemies are left in the level, return True

        if not self.enemies:

            return True

        # If there are still non-destroyed enemies in the level, return False

        else:

            return False

    @staticmethod

    def game\_over\_dialog(controller: Controller):

        # Create the dialog box window

        dialog\_root = Tk()

        dialog\_root.title("Game Over!")

        dialog\_root.geometry("400x100")

        # Create a label with the game over text

        game\_over = tkinter.Label(dialog\_root, text="Game over! Try again?", font=("Arial", 22))

        game\_over.pack(padx=10, fill=tkinter.X, expand=True)

        # Create a retry button that closes the dialog box

        retry\_button = Button(dialog\_root, text="Retry", command=dialog\_root.destroy)

        retry\_button.pack(padx=10, pady=10, fill=tkinter.X, side=tkinter.LEFT, expand=True)

        # Create a quit button that exits the game

        quit\_button = Button(dialog\_root, text="Quit", command=(lambda: Level.game\_over(dialog\_root, controller)))

        quit\_button.pack(padx=10, pady=10, fill=tkinter.X, side=tkinter.RIGHT, expand=True)

        # Display the dialog and wait for input

        dialog\_root.mainloop()

    @staticmethod

    def game\_over(tkinter\_root, controller: Controller):

        # Close the tkinter GUI

        tkinter\_root.quit()

        # Stop the input thread

        controller.stop = True

        # Disconnect the controller

        controller.disconnect()

        # Close the pygame window

        pygame.quit()

        # Exit the program

        exit()

### Главна програма (Модул “Main”)

import threading

from collections import namedtuple

import numpy

import pygame

from Controller import Controller

from Game import Game

from Level import Level

from Player import Player

def main():

    # Initialize the pygame module and clock

    pygame.init()

    clock = pygame.time.Clock()

    # Initialize the controller module

    controller = Controller()

    # Connect to the controller

    controller.connect()

    # Create a thread for reading of controller input

    controller\_input\_thread = threading.Thread(target=Controller.read\_input, args=(controller,))

    controller\_input\_thread.start()

    # Save the current states of all controller buttons

    ButtonStates = namedtuple("ButtonStates", "but1, but2, butj")

    controller\_button\_states = ButtonStates(controller.but1, controller.but2, controller.butj)

    # Initialize the player module

    player = Player()

    # Load the first level

    level = Level(1)

    # Write the game score to the controller

    controller.write\_score(Game.score)

    # Main loop

    while True:

        # Get the time it took to render the previous frame

        frame\_time = clock.tick(Game.max\_fps)

        # Check if the player was hit by an enemy projectile

        if level.check\_player\_hit(player):

            # Display the game over dialog

            Level.game\_over\_dialog(controller)

            # If the "Retry" button was pressed, reset the score and write it to the controller

            Game.score = 0

            controller.write\_score(Game.score)

            # Load the first level

            level = Level(1)

            # Go to the next iteration of the main loop

            continue

        # Check if any enemies were hit by player projectiles

        for projectile in player.projectiles:

            hit = level.check\_enemy\_hit(projectile)

            if hit:

                # Destroy the player projectile

                player.projectiles.remove(projectile)

                # Check if the level was completed

                if level.check\_completion():

                    # Increase the score and write it to the controller

                    Game.score += level.num \* 1000

                    controller.write\_score(Game.score)

                    # Load the next level

                    level = Level(level.num + 1)

                    # Empty the list of player projectiles

                    player.projectiles = []

                    # Go to the next iteration of the main loop

                    continue

        # Draw the player, enemies and projectiles on the game screen

        Game.screen.fill((0, 0, 0))

        level.move\_enemies(frame\_time)

        level.draw(frame\_time)

        player.draw(frame\_time)

        # Display the new frame

        pygame.display.update()

        # Get a list of pygame events

        events = pygame.event.get()

        for event in events:

            # If the "QUIT" event was received

            if event.type == pygame.QUIT:

                # Stop the input thread

                controller.stop = True

                # Disconnect the controller

                controller.disconnect()

                # Close the pygame window

                pygame.quit()

                # Exit the program

                exit()

        # Save the current states of all controller buttons

        controller\_button\_states = (controller.but1, controller.but2, controller.butj)

        # If the "shoot" button was pressed (where it was not previously), shoot a projectile from the player

        if controller.but2 == 0 and controller\_button\_states[1] != 0:

            player.shoot()

        # Map the X and Y movement values received from the controller (numbers between 0 and 4096) to values used by

        # the player movement function (numbers between -2048 and 2048)

        x\_move = numpy.interp(controller.x, [0, 4096], [-2048, 2048])

        y\_move = numpy.interp(controller.y, [0, 4096], [-2048, 2048])

        # Move the player according to the input received from the controller

        player.move(dx=x\_move,

                    dy=-y\_move,

                    frame\_time=frame\_time)

if \_\_name\_\_ == "\_\_main\_\_":

    # Call the main function

    main()