ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

КУРСОВ ПРОЕКТ

ПО ПРАКТИКУМ ПО ПРОГРАМИРАНЕ НА МИКРОКОНТРОЛЕРИ

Студент:
Илиян Антов
Фак. №: 101220020

Одобрил:
доц. д-р инж.
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)

2. Информацията от контролера да се изпраща по подходящ начин към компютърна програма чрез серийна комуникация през UART интерфейс.

3. 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:

4. Да се осигури възможност за свързване към микроконтролерно устройство посредством серийна връзка през UART интерфейса.

5. Да се създадат основните модули на играта:
   - Играч (Player)
   - Противник (Enemy)
   - Снаряд (Projectile)
   - Ниво (Level)

6. Да се създадат няколко демонстративни нива чрез свързване на модулите по подходящ начин и добавяне на меню за край на играта.

7. Получената информация от контролера да се прочита и обработва, като се преобразува в команди, които управляват позицията и състоянието на Играча (Player).

8. 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
II. Обобщен блоков алгоритъм 5
III. Ръководство за потребителя 6
1. Подготовка за стартиране на играта 6
2. Стартиране на играта 6
3. Геймплей 7
4. Край на играта 8
IV. Ръководство за програмиста 9
1. Функции на микроконтролерната програма 9
2. Функции на компютърната програма 10
2.1. Модул “Controller” 10
2.2. Модул “Game” 11
2.3. Модул “Projectile” 11
2.4. Модул “Player” 12
2.5. Модул “Enemy” 12
2.6. Модул “Level” 13
2.7. Главна програма (Модул “Main”) 15
V. Програмен код 16
1. Програмен код на микроконтролера 16
2. Програмен код на компютърната програма 21
2.1. Модул “Controller” 21
2.2. Модул “Game” 23
2.3. Модул “Projectile” 23
2.4. Модул “Player” 24
2.5. Модул “Enemy” 26
2.6. Модул “Level” 27
2.7. Главна програма (Модул “Main”) 32

I. Увод

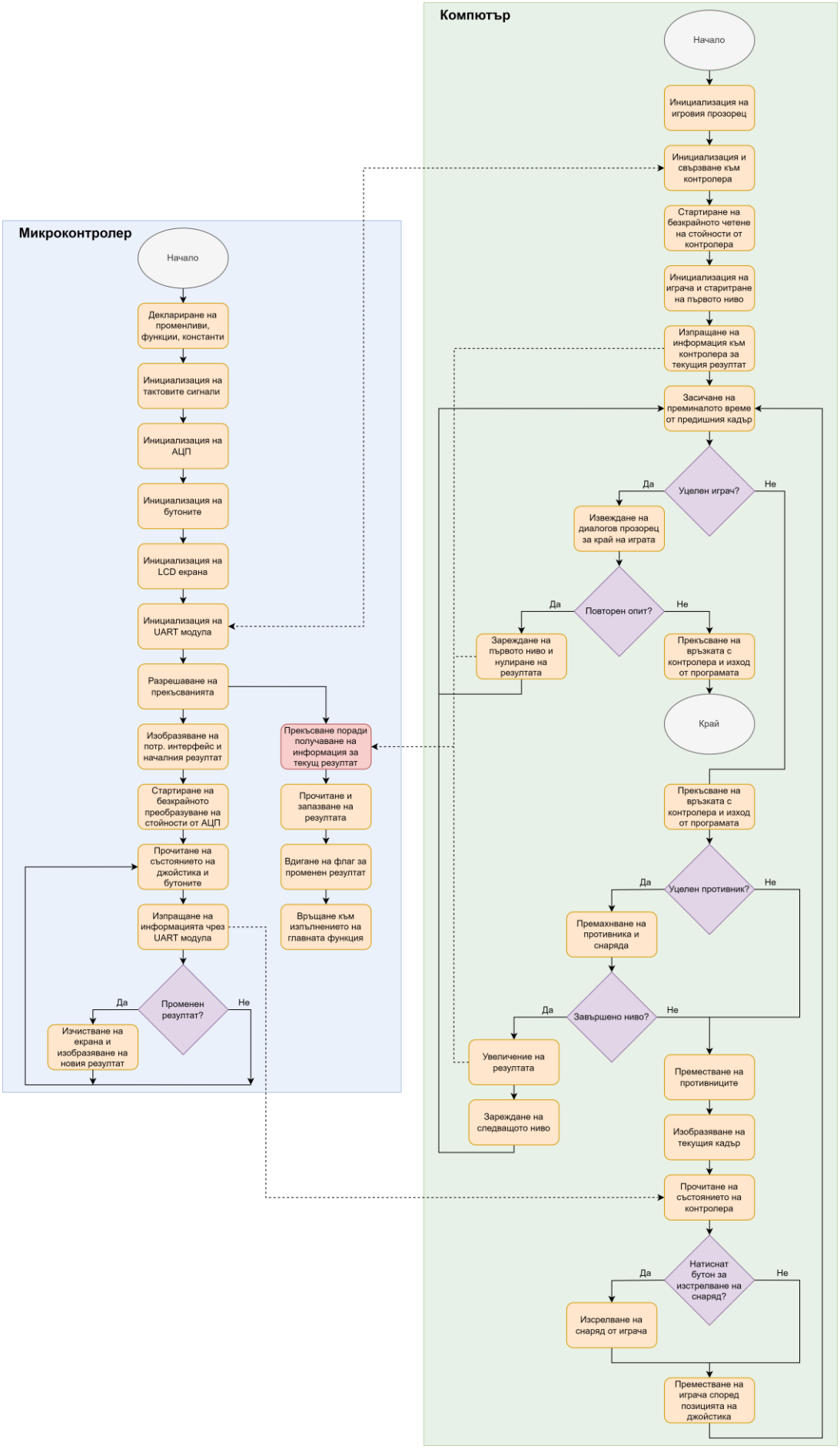
В днешни дни, игровите контролери са може би най-разпространеният аксесоар в индустрията на видеоигрите. На практика всяка игрова конзола е снабдена с поне един контролер, а в много случаи и с по няколко. Производителите предалагат вариации на тези продукти и самостоятелно, като предоставят възможност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

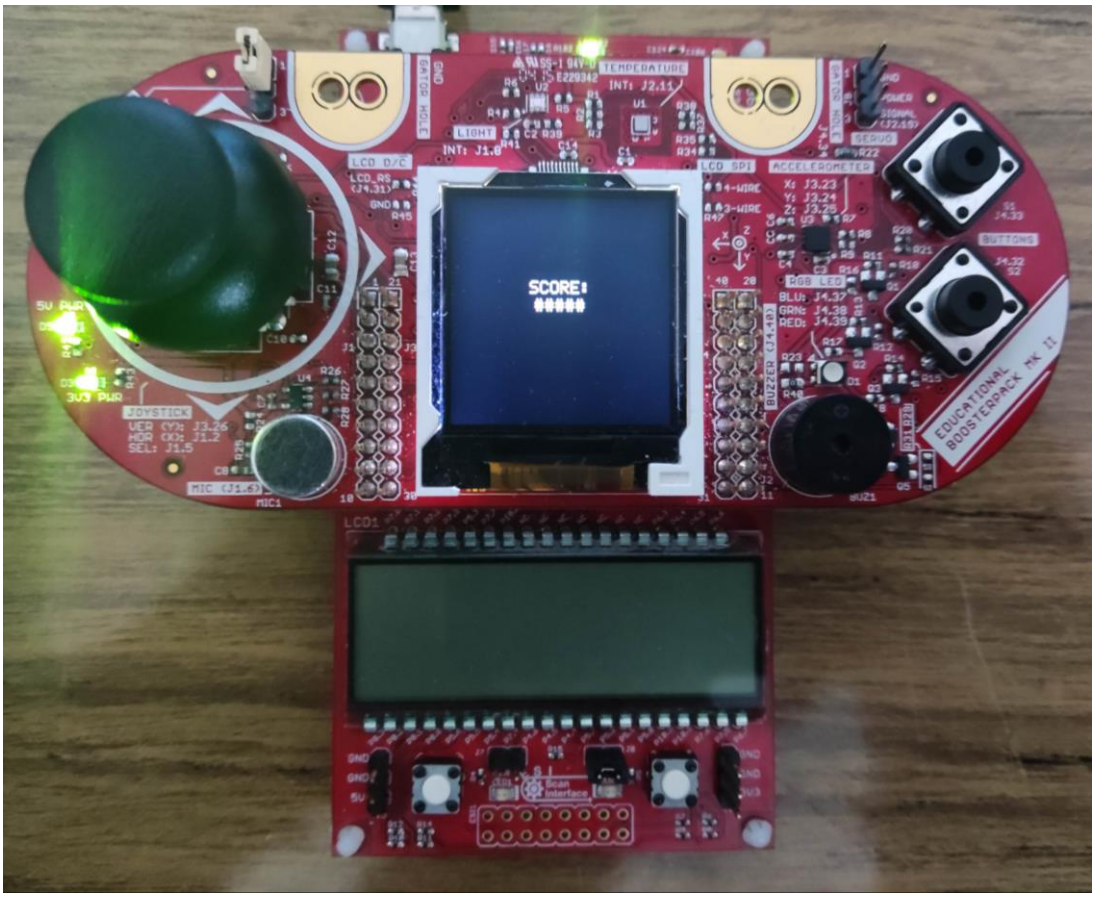
II. Обобщен блоков алгоритъм



III. Ръководство за потребителя

1. Подготовка за стартиране на играта

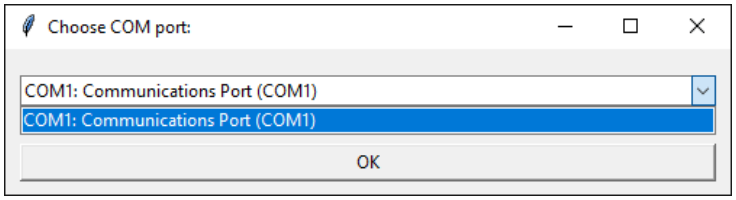
Преди да стартира играта, потребителят трябва да свърже посредсвом USB кабел игровия контролер (MSP430FR6989 LaunchPad с прикачен BOOSTXL-EDUMKII модул) към компътъра и да изчака инициализацията на всички модули на микроконтролера. Инициализацията е приключила, когато на LCD дисплея на контролера се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



Фиг. 3.1. Инициализация на контролера

2. 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 падащото меню и натискане на бутона OK.



Фиг. 3.2. Диалогов прозорец за избор на комуникационен порт

3. 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.)

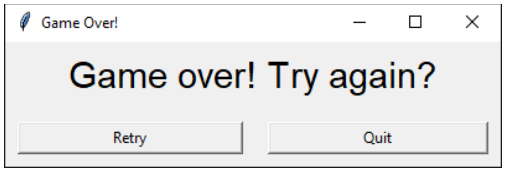
Фиг. 3.5. Второ ниво на компютърната игра

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

4. 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

IV. Ръководство за програмиста

1. 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

2. Функции на компютърната програма

2.1. 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 посредством серийната връзка

2.2. 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

2.3. 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

2.4. 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

2.5. 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

2.6. 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

2.7. 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

V. Програмен код

1. 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);
}
```

2. Програмен код на компютърната програма

2.1. 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))
```

2.2. 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)
```

2.3. 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
```

2.4. 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)
```

2.5. 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
```

2.6. 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()
```

2.7. 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()
```