

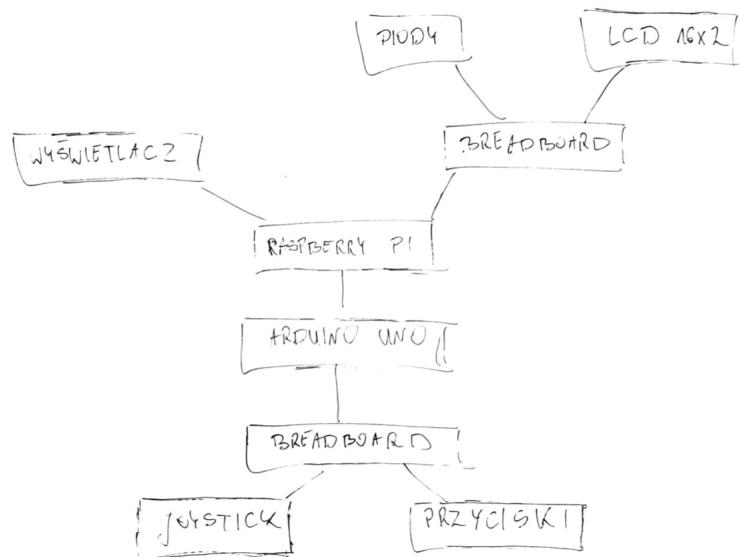
Laboratorium z przedmiotu Systemy wbudowane (SW)

Prowadzący mgr inż. Ariel Antonowicz	Autorzy 145323, 145396	Grupa I1.2
Nazwa projektu “Space Invaders“		Ocena

1 Cel projektu

Celem projektu jest stworzenie klonu klasycznej gry “Space Invaders“ na platformie Raspberry Pi ze sterowaniem oraz HUD zrealizowanym za pomocą urządzeń wejścia/wyjścia.

2 Wstępny schemat



3 Wykorzystana platforma sprzętowa, czujniki pomiarowe, elementy wykonawcze:

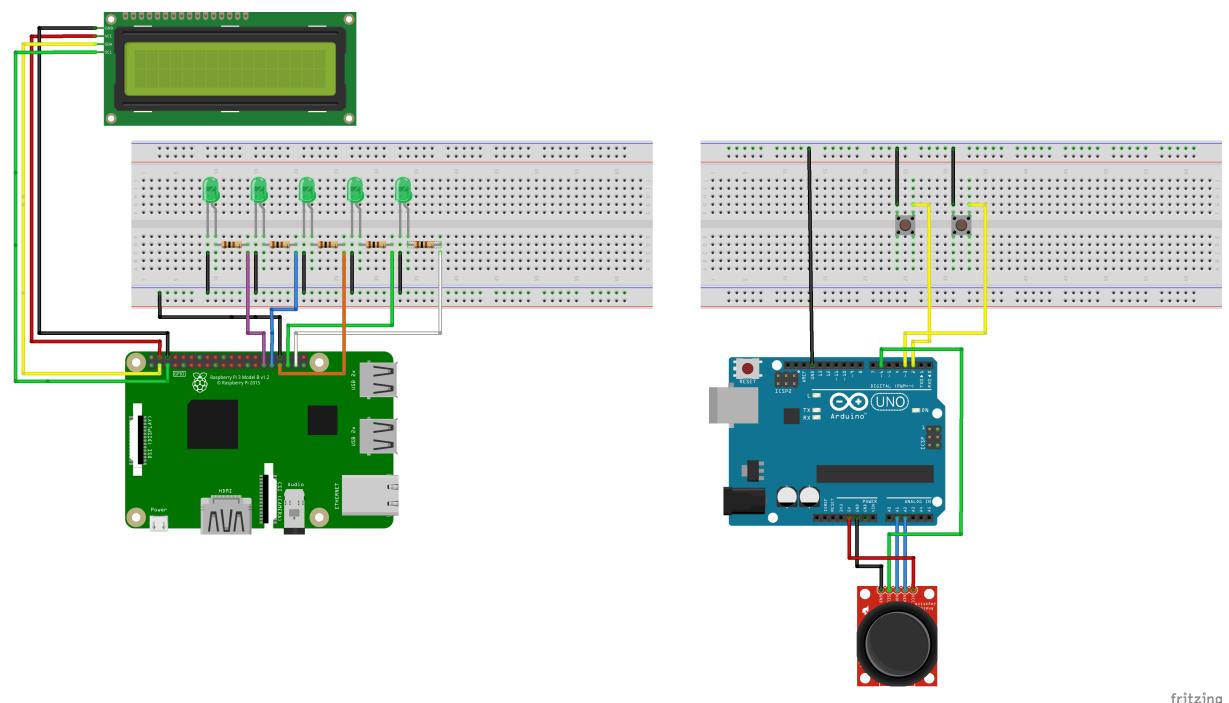
- Raspberry Pi 3,
- Wyświetlacz do Raspberry Pi (lub wyświetlimy grę na prywatnym komputerze przy pomocy VNC),
- Trzy diody LED (poziom życia),
- Wyświetlacz LCD 16x2 (obecny poziom gry oraz liczba zdobytych punktów),
- Arduino UNO (odbieranie sygnałów wejściowych od użytkownika),
- Joystick oraz dwa przyciski (sterowaniem statkiem).

4 Cel i zakres projektu

Projekt składał się z następujących etapów:

1. Napisanie kodu gry typu "Space Invaders",
2. Podłączenie do mikrokontrolera Arduino UNO przycisków oraz joysticka,
3. Podłączenie do minikomputera Raspberry Pi 3 diod oraz wyświetlacza LCD,
4. Połączenie urządzeń poprzez USB,
5. Uruchomienie gry na systemie Raspberry,
6. Przekazywanie do gry parametrów wejściowych poprzez Arduino, umożliwiając prowadzenie rozgrywki,
7. Odbieranie od gry odpowiednich informacji i przekazywanie ich do urządzeń wyjściowych (Raspberry),
8. Optymalizacja gry oraz uporządkowanie kodu,
9. Finalny test funkcjonalności.

5 Schemat



Uwagi do schematu:

- Brakuje połączenia pomiędzy Raspberry oraz Arduino poprzez USB,
- Brakuje połączenia pomiędzy Raspberry oraz monitorem poprzez port HDMI,
- Wykorzystany w schemacie joystick jest inny niż użyty przez nas w projekcie, oryginalny posiada osobne porty VCC oraz GND dla każdego z trzech portów X, Y, Z, natomiast podłączenia do wejść analogowych oraz wejścia cyfrowego w Arduino są zgodne z rzeczywistością.

Powyżej wymienione braki wynikają z ograniczeń oprogramowania Fritzing.

6 Projekt a realizacja

6.1 Gra

Projekt rozpoczęliśmy zgodnie z planem, od napisania kodu gry przy wykorzystaniu biblioteki PyGame. Miała ona polegać na zestrzeliwaniu kolejnych fal przeciwników i zbieraniu jak największej liczby punktów. Zaprogramowaliśmy trzy ustawienia wrogów, które są losowo wybierane na początku każdej fali. Dodatkowo każda kolejna fala porusza się szybciej od poprzedniej, co liniowo zwiększa trudność rozgrywki.

Problemy pojawiły się przy testowaniu gry na platformie Raspberry. Na komputerze osobistym gra osiągała płynne 60 klatek na sekundę, co niestety okazało się nieosiągalne na słabszym urządzeniu. Wprowadzone optymalizacje, takie jak obniżona rozdzielcość okienka gry, prostsze tekstury czy mniejsza liczba przeciwników, nie przyczyniły się do osiągnięcia oczekiwanych rezultatów. Pomimo niezadowalającej nas wydajności gra jest w pełni grywalna, utrzymując około 30 klatek na sekundę.

Prostota gry oraz wprowadzone optymalizacje mogą sugerować słabą wydajność samej biblioteki PyGame na urządzeniu Raspberry, dlatego w przypadku dalszego rozwijania projektu napisalibyśmy grę w innym środowisku graficznym.

W grze zaimplementowaliśmy licznik wrogich statków, które minęły statek gracza. Jeżeli wartość licznika osiągnie wartość równą pięć - gra zostaje zakończona. Niestety ze względu na ograniczenia czasowe nie udało nam się zaprezentować tej informacji na fizycznym urządzeniu wyjściowym. W przypadku większych zasobów czasowych skorzystalibyśmy z dodatkowych diod LED, w celu pokazania wartości licznika.

Poniższy kod przedstawia główną pętlę gry. Na podstawie obecnego stanu rozgrywki wartości określające poziom gry, liczby zdobytych punktów oraz poziom życia zostają przekazane do Raspberry. Na podstawie stanu urządzeń wejściowych statek gracza przesuwa się w odpowiednim kierunku, zostaje oddany strzał lub następuje reset gry. W trakcie pętli sprawdzane są również warunki zakończenia rozgrywki, następuje generacja nowej fali przeciwników (jeżeli poprzednia została zakończona) oraz statki przeciwników zostają przesunięte.

```
1 while run:  
2     clock.tick(FPS)  
3     writer.write_to_screen(level=level, points=int(player.score))  
4     writer.set_leds_based_on_health(health=player.health)  
5  
6     redraw_window()  
7  
8     if lives <= 0 or player.health <= 0:  
9         lost = True  
10        lost_count += 1  
11  
12        if lost:  
13            if lost_count > FPS * 3:  
14                run = False
```

```

15         else:
16             continue
17
18     #Creation of new wave
19     if len(enemies) == 0:
20         randlevel = random.randint(0,2)
21         for i in range(enemyAmount[randlevel][0]):
22             enemy=Enemy(enemyCordsx[randlevel][0][i],enemyCordsy[randlevel][0][i],"blue")
23                 enemies.append(enemy)
24                 for i in range(enemyAmount[randlevel][1]):
25                     enemy=Enemy(enemyCordsx[randlevel][1][i],enemyCordsy[randlevel][1][i],"green")
26                         enemies.append(enemy)
27                         for i in range(enemyAmount[randlevel][2]):
28                             enemy=Enemy(enemyCordsx[randlevel][2][i],enemyCordsy[randlevel][2][i],"red")
29                                 enemies.append(enemy)
30                                 level += 1
31                                 enemy_vel+=0.25
32
33     for event in pygame.event.get():
34         if event.type == pygame.QUIT:
35             run= False
36
37     x, y, z, b1, b2 = reader.get_input()
38
39     #Player movement
40     if x < 400 and player.x - player.vel > 0: #left
41         player.x -= player.vel
42     if x > 800 and player.x + player.vel + player.get_width() < WIDTH: #right
43         player.x += player.vel
44     if y > 800 and player.y-player.vel > 0: #up
45         player.y -= player.vel
46     if y < 400 and player.y+player.vel + player.get_height() + 20 < HEIGHT: #down
47         player.y += player.vel
48     if b1 == 0:
49         player.shoot()
50     if b2 == 0:
51         run = False
52
53     #Enemy movement
54     for enemy in enemies[:]:
55         enemy.move(enemy_vel)
56         enemy.move_lasers(laser_vel,player)
57
58         if random.randrange(0,2*60) == 1:
59             enemy.shoot()
60
61         if collide(enemy,player):
62             player.health -= 20
63             enemies.remove(enemy)
64             player.score += 5
65         elif enemy.y + enemy.get_height() > HEIGHT:
66             lives -= 1
67             enemies.remove(enemy)
68
69     player.move_lasers(-laser_vel,enemies)
70

```

6.2 Urządzenia wejścia

Pierwsze elementy układu, które postanowiliśmy połączyć to przyciski (odpowiadające za strzelanie statkiem i reset gry), oraz joystick (umożliwia ruch statkiem). Niemożliwe było podłączenie ich bezpośrednio do Raspberry, ponieważ w odróżnieniu od Arduino, nie posiada ono pinów analogowych, niezbędnych do podłączenia joysticka. W celu uproszczenia układu zdecydowaliśmy się podłączyć wszystkie urządzenia wejścia do Arduino.

Przyciski podłączyliśmy bez wykorzystania rezystorów, korzystając z wewnętrznego *pull-up* w kodzie za pomocą INPUT_PULLUP. Joystick podłączyliśmy do dwóch pinów analogowych (horizontalny oraz wertykalny ruch), oraz jednego cyfrowego (wciśnięcie galki). Do każdego urządzenia poprowadziliśmy również uziemienie.

Do Arduino wgraliśmy program, odczytujący wartości określające stan przycisków oraz joysticka, a następnie za pomocą poniższej klasy odczytywaliśmy dane podczas rozgrywki.

```
1 class ArduinoInputReader:
2     def __init__(self):
3         self.ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
4         self.ser.reset_input_buffer()
5
6     def get_input(self):
7         data = self.ser.readline().decode('ascii').rstrip()
8         self.ser.reset_input_buffer()
9
10    try:
11        x, y, z, b1, b2 = [int(i) for i in data.split(',')]
12    except:
13        x, y, z, b1, b2 = 512, 512, 1, 1, 1
14
15    return x, y, z, b1, b2
```

6.3 Urządzenia wyjścia

Następnym etapem pracy polegał na podłączeniu urządzeń wyjścia, czyli diod i wyświetlacza do Raspberry. Finalnie liczba diod oznaczała poziom aktualnego zdrowia gracza (100% - 5 diod zapalonych, 0% - 0 diod zapalonych), natomiast na wyświetlaczu pojawiała się informacja o aktualnym poziomie (fali przeciwników) i punktach, zdobywanych za pokonywanie wrogów.

```
1 class RaspberryOutputWriter:
2     def __init__(self, led_ports=(5, 6, 13, 19, 26)):
3         self.lcd = I2C_LCD_driver.lcd()
4
5         self.lcd.lcd_clear()
6         self.write_to_screen(level=5, points=0)
7         self.led_ports = led_ports
8
9     # Initialize LEDs
10    GPIO.setmode(GPIO.BCM)
11    GPIO.setwarnings(False)
12
13    for led_port in led_ports:
14        GPIO.setup(led_port, GPIO.OUT)
15
16    for led_port in led_ports:
17        GPIO.output(led_port, GPIO.HIGH)
18
19    def write_to_screen(self, level, points):
20        self.lcd.lcd_display_string("Level: {}".format(level), 1, 0)
21        self.lcd.lcd_display_string("Points: {}".format(points), 2, 0)
22
23
```

```

24 def set_leds_based_on_health(self, health):
25     n = health // 20 if health > 0 else 0
26
27     for led_port in self.led_ports:
28         GPIO.output(led_port, GPIO.LOW)
29
30     for led_port in self.led_ports[:n]:
31         GPIO.output(led_port, GPIO.HIGH)

```

6.4 Połączenie układów

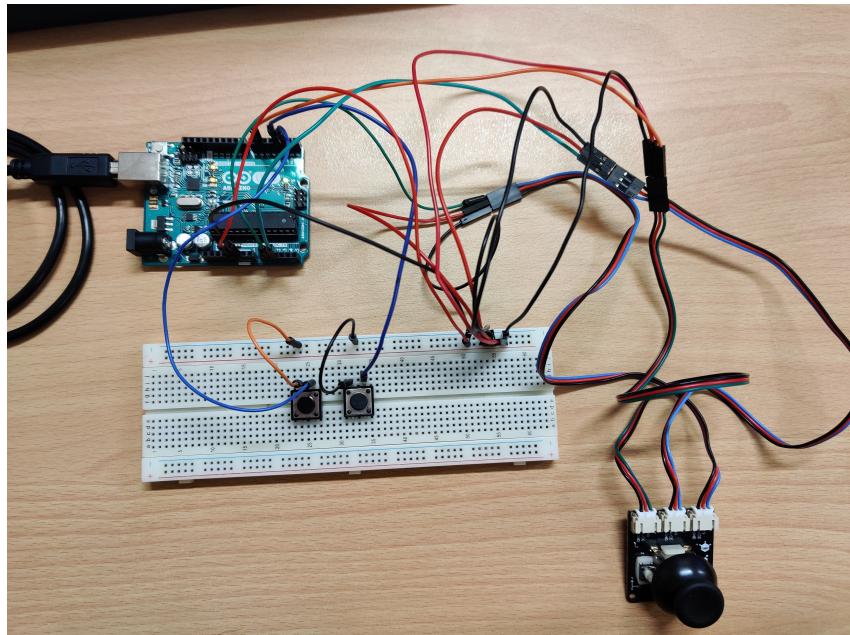
Po zbudowaniu układów odpowiedzialnych za dane wejściowe jak i wyjściowe, nadszedł czas na złączenie ich w jeden system. W tym celu między Raspberry a Arduino poprowadzony został kabel USB, umożliwiający niezbędną komunikację. Kod wprowadzony na Arduino, wypisujący dane na monitorze portu szeregowego, wygląda następująco:

```

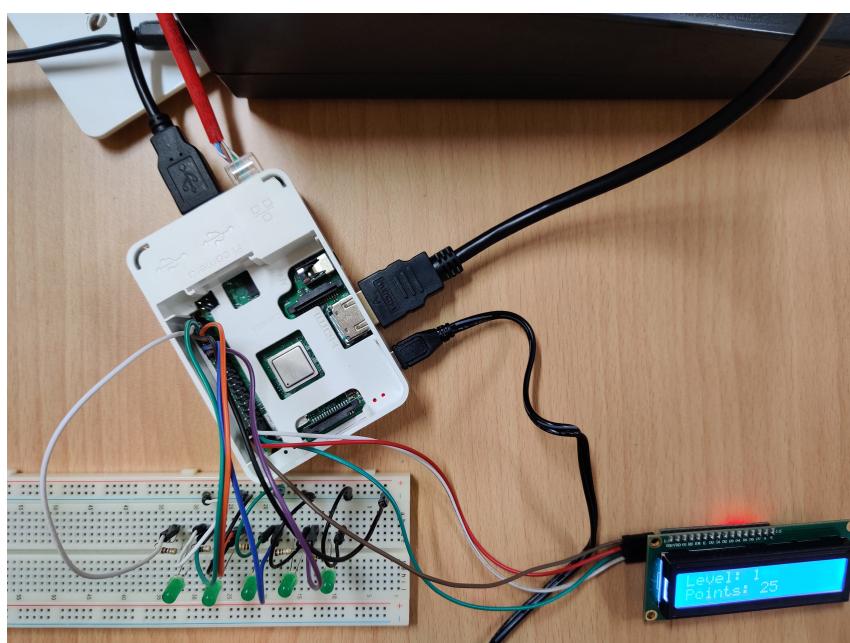
1 int JoyStick_X = 1; // A1
2 int JoyStick_Y = 2; // A2
3 int JoyStick_Z = 6;
4
5 const int buttonPin1 = 2;
6 const int buttonPin2 = 3;
7
8 void setup()
9 {
10     pinMode(JoyStick_Z, INPUT);
11     pinMode(buttonPin1, INPUT_PULLUP);
12     pinMode(buttonPin2, INPUT_PULLUP);
13
14     Serial.begin(9600);
15
16     delay(100);
17 }
18 void loop()
19 {
20     int x, y, z, b1, b2;
21
22     x = analogRead(JoyStick_X);
23     y = analogRead(JoyStick_Y);
24     z = digitalRead(JoyStick_Z);
25     b1 = digitalRead(buttonPin1);
26     b2 = digitalRead(buttonPin2);
27
28     Serial.print(x ,DEC);
29     Serial.print(",");
30     Serial.print(y ,DEC);
31     Serial.print(",");
32     Serial.print(z ,DEC);
33     Serial.print(",");
34     Serial.print(b1,DEC);
35     Serial.print(",");
36     Serial.print(b2,DEC);
37     Serial.print('\n');
38     delay(100);
39 }

```

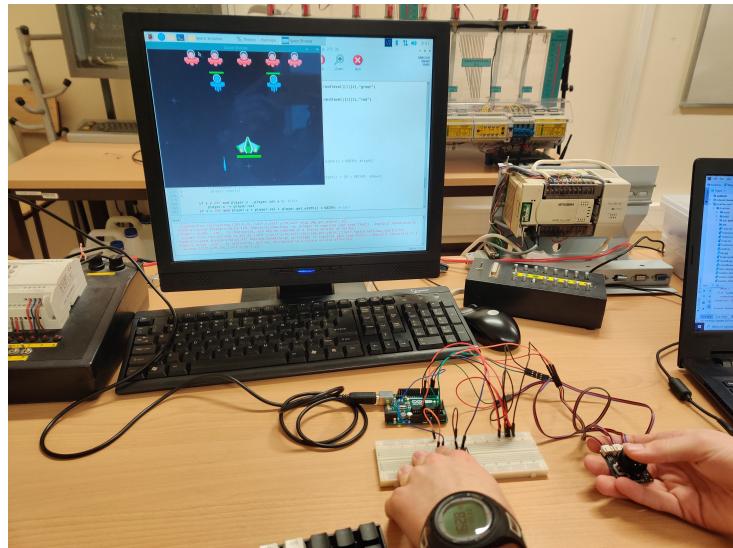
7 Zdjęcia fizycznego urządzenia/połączeń



Rysunek 1: Arduino wraz z urządzeniami wejścia - przyciskami oraz joystickiem

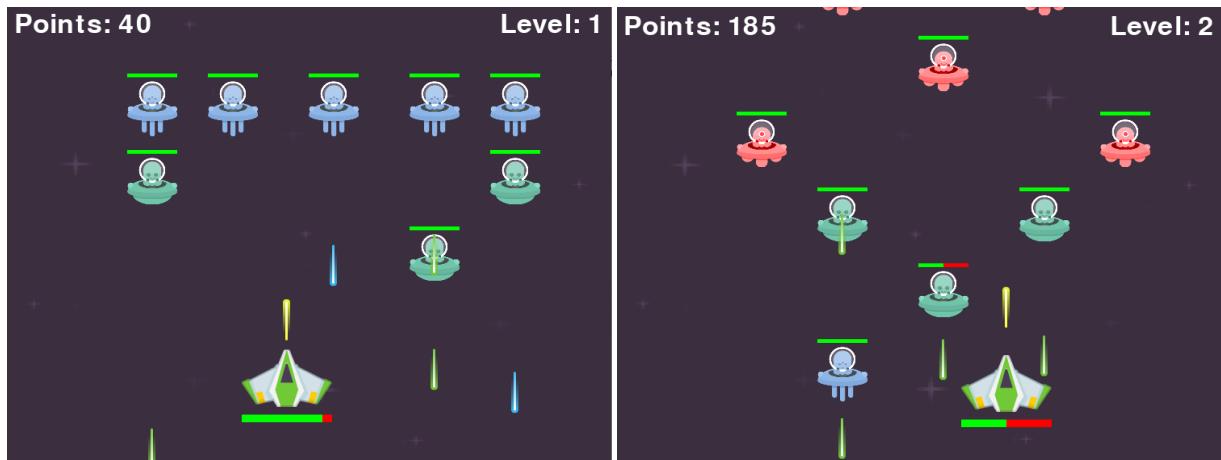


Rysunek 2: Raspberry wraz z urządzeniami wyjścia - diodami LED oraz wyświetlaczem LCD



Rysunek 3: Finalny test funkcjonalności

8 Zrzuty z ekranu aplikacji



9 Podsumowanie i wnioski

System jest w pełni sprawny i funkcjonalny, jednak jest też miejsce na wprowadzenie pewnych ulepszeń. Z pewnością tworząc system tego typu następny raz, wzięlibyśmy bardziej pod uwagę możliwości wydajnościowe narzędzi Raspberry, aby w pełni zwiększyć komfort gry. Pomimo tego jesteśmy zadowoleni z efektu - udało nam się osiągnąć cele, które przed sobą postawiliśmy, a wynik naszej pracy spełnia nasze oczekiwania.