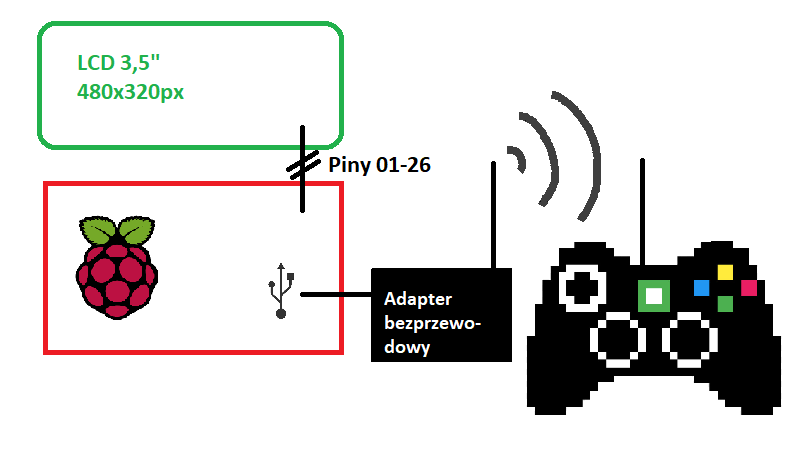
|  |  |  |  |
| --- | --- | --- | --- |
| **Laboratorium z przedmiotu Systemy wbudowane (SW)** | | | |
|  | | | |
| Karta projektu – zadanie 7 | | | |
|  | | | |
| *Nazwa projektu:* | | | |
|  | | | |
| *Prowadzący:* | *Autorzy (tylko nr indeksu):* | *Grupa dziekańska:* |  |
| ***Ocena:*** |  |
|  | | | |
| *Cel projektu:* | | | |
|  | | | |
| *Schemat:* | | | |
|  | | | |
| *Wykorzystana platforma sprzętowa, czujniki pomiarowe, elementy wykonawcze:* | | | |

# 1. Cel i zakres projektu

Celem projektu było stworzenie gry typu Snake, która ma działać na Raspberry Pi 3 B+. Aby uniknąć konieczności podłączania zewnętrznego monitora (co jest mało praktyczne w przypadku komputerów jednopłytkowych SBC) użyto dotykowego wyświetlacza LCD podłączanego przez GPIO, co razem z Raspberry Pi tworzy jedną zgrabną bryłę w obudowie.  
Aby nasza retro konsola była bardziej atrakcyjna stworzyliśmy skrypt który po zaimportowaniu do naszej gry obsługuje bezprzewodowego pada od Xboxa 360.

# 2. Schemat ideowy i połączeniowy



# 3. Projekt, a realizacja

Udało się zrealizować początkowe założenia projektu. Wyświetlacz wyświetla całe okno gry, a pad od Xboxa działa również bez zarzutu.  
W realizacji napotkaliśmy jednak problem natury sprzętowej, o czym nie zdawaliśmy sobie sprawy na początku, ale Raspberry Pi nie posiada podzespołów które zapewniają płynność rozgrywki do której przyzwyczaiły nas współczesne gamingowe komputery. Przypomina to rzeczywiście typowe retro konsole jak na przykład Game Boy.  
Jeśli chodzi o rozwój tego projektu, to na pewno możnaby dodać nowe funkcjonalności do naszej gry (dla przykładu tablicę wyników). Inną drogą rozwoju mogłoby być dodanie zupełnie innych gier i stworzenie konsoli na której można odpalać różne produkcje.

# 4. Najważniejsze fragmenty kodu z komentarzami.

→ Rozpoczęcie obsługi danego pada w naszej grze:

if pygame.joystick.get\_count() > 0:

    joystick = pygame.joystick.Joystick(0)

    joystick.init()

→ Główna cześć gry:

def newGame():

    global snake

    snake = Snake()

    global cur\_dir

    cur\_dir = LEFT

    global score

    score = 0

    global foods

    foods = []

    mainLoop()

def mainLoop():

    while True:

        handleEvents(pygame.event.get())

        addFoodInRandomPositionAndTime()

        drawGameWindow()

        pygame.time.wait(1000//GAME\_FPS)

→ Klasa Segment która reprezentuje pojedynczy blok na ekranie (składa się z nich sam wąż, jak i jedzenie):

class Segment:

    def \_\_init\_\_(self, x: int, y: int, dir: int):…

def draw(self, color=C\_WHITE):…

def update(self):…

→ Klasa Snake która reprezentuje poruszającego się po ekranie węża:

class Snake:

#inicjalizacja 2 segmentowego węża

def \_\_init\_\_(self):…

#metoda wydłużająca węża po zjedzeniu jedzenie

def addSegment(self):…  
 #metoda wykrywajaca czy wąż nie uderzył w siebie

def detectCollision(self):…  
 #metoda wykrywająca czy zostało zjedzone jedzenie

def detectFood(self):…  
 #metoda pomocnicza która pozwala tylko raz zjeść dany klocek  
 #z jedzeniem i zapobiega zawieszeniu się gry w pętli nieskończonej

def deleteLockInFlag(self):…  
 #metoda rysująca węża i aktualizująca jego cechy w każdej klatce

def drawAndUpdate(self, dir: int):…

→ Klasa Food reprezentująca pojawiające się na ekranie jedzenie:

class Food:

def \_\_init\_\_(self):

def drawAndUpdate(self):

→ Globalna metoda pozwalająca na pojawianie się nowych klocków z jedzeniem dla węża:

def addFoodInRandomPositionAndTime():

    maxFoodsInGameBoard = 1

    actualFoodsInGameBoard = len(foods)

    randomParameter = 30

    if actualFoodsInGameBoard < maxFoodsInGameBoard:

        #Add one food if board is empty

        if actualFoodsInGameBoard == 0:

            foods.append(Food())

        #Add next food to board sometimes

        if random.randint(0, randomParameter) == 1:  
       foods.append(Food())

Kod ten dodaje zawsze jedzenie w momencie gdy snake zje z planszy ostatni element, oraz w losowych momentach dodaje na planszy segment z jedzeniem o ile zmieści się w maksymalnym limicie dostępnego jedzenie na planszy.

→ Funkcja obsługująca zdarzenia input/output:

def handleEvents(events):

→ Funkcja rysująca okno gry w każdej klatce:

def drawGameWindow():

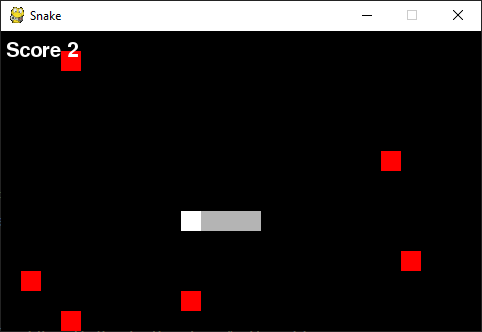
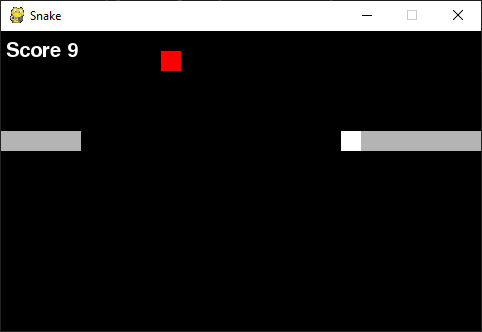
→ Funkcja wyświetlająca okno z powiadomieniem dla gracza:

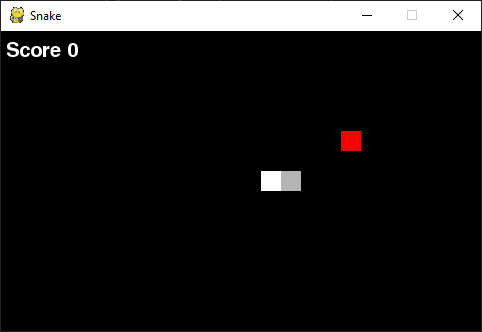
def displayMessageBox(subject, content):

# 5. Zdjęcie fizycznego urządzenia oraz połączeń

X

# 6. Zrzuty z ekranu z aplikacji





# 7. Podsumowanie i wnioski

X