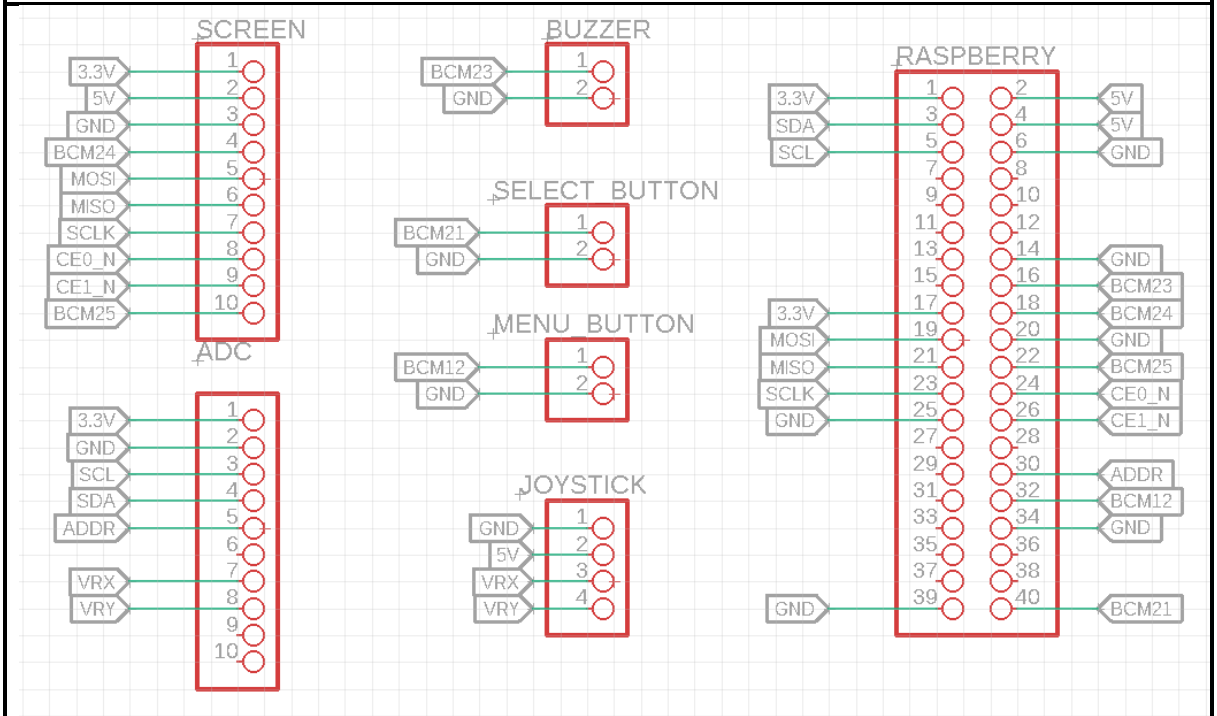
[Z7] KARTA PROJEKTU

Grupa laboratoryjna: L10/L4		
	Prowadzący zajęcia:	

Mini Arcade Machine

CEL PROJEKTU	*Stworzenie miniaturowej maszyny do gier inspirowanej maszynami arkadowymi z lat 80'. Maszyna posiada 3 rodzaje gier logicznych: Minesweeper, Memory oraz Simon Says. Model maszyny jest wydrukowany w drukarce 3D. Maszyna posiada joystick i 2 przyciski. Wbudowany jest również buzzer wydający efekty dźwiękowe oraz grający melodię.*

SCHEMAT POGLĄDOWY (UPROSZCZONY)



WYKORZYSTANA PLATFORMA SPRZĘTOWA, ELEMENTY POMIAROWE I WYKONAWCZE	*Raspberry Pi 4B, Thumb Joystick, przyciski Tact Switch, buzzer pasywny, wyświetlacz LCD TFT*

1. Cel i zakres projektu.

Celem projektu było stworzenie miniaturowej maszyny do gier, inspirowanej maszynami arkadowymi z lat 80'. W wydrukowanej na drukarce 3D 'maszynie' znajduje się Raspberry Pi 4B, do której podłączone są urządzenia wyjścia oraz wejścia, takie jak: joystick, przyciski, buzzer, wyświetlacz. Całość zasilana jest za pomocą dedykowanego zasilacza USB C dla Raspberry Pi 4B.

Maszyna posiada 3 gry: „Minesweeper", „Simon Says", „Match-2" oraz tabele najlepszych wyników dla każdej z gier. Minesweeper, czyli saper to gra, której celem jest znalezienie bomb rozłożonych na planszy. Jako wskazówka, co do położenia bomb, reszta kafelek na planszy posiada liczbę, odpowiadającą liczbie bomb w sąsiedztwie danego kafelka. Im szybciej gracz odkryje wszystkie kafelki niezawierające bomb, tym lepszy wynik osiągnie. „Simon Says" to gra na czas, której celem jest odegranie melodii, naśladując podświetlone kafelki. Wynik liczony jest na podstawie ilości odegranych nut. „Match-2" to gra polegająca na zapamiętywaniu położenia kafelków odpowiedniego koloru oraz połączeniu dwóch kafelków o tych samych kolorach. Wynik jest liczony na podstawie czasu, ile graczowi zajęło połączenie wszystkich kolorów. Gry zostały napisane w języku programowania Python z użyciem biblioteki PyGame. PyGame jest prostą biblioteką stworzoną z myślą o tworzeniu gier w języku Python. Do przechowywania wyników użyliśmy bazy danych w SQLite. Pomimo małego rozmiaru przechowywanych wyników, zdecydowaliśmy się na użycie SQLite ze względu na prostotę zapytań i obsługi danych w porównaniu z tak zwaną „hardcodowaną" obsługą danych w pliku 'txt'. Do interakcji z grą służą: joystick – do akcji „lewo", „prawo", „góra" i „dół" oraz dwa przyciski – niebieski – akcja „zatwierdź", czerwony – akcja „menu". Buzzer został wykorzystany jako informacja zwrotna dla gracza. W zależności od akcji, jaką gracz wykona, buzzer zagra krótką melodię: radosną, sugerującą wygraną grę lub smutną, sugerującą koniec gry. Dodatkowo buzzer reaguje krótkim dźwiękiem podczas wciskania przycisku interakcji.

Maszyna posiada w menu gry opcję bezpiecznego wyłączenia. Należy użyć tej opcji przed odłączeniem od zasilania, aby zabezpieczyć system przed korupcją plików.

Wykorzystany sprzęt:

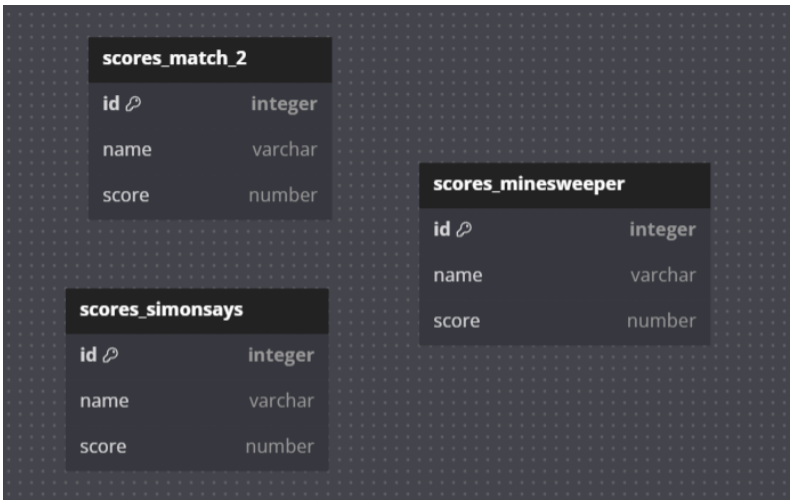
- Raspberry Pi 4B 4GB RAM
- Wyświetlacz TFT 2.8"
- Thumb Joystick
- Konwerter ADC ADS1115
- Przyciski Tact Switch - x2
- Buzzer pasywny
- Zasilacz USB C 5.1V/3A do Raspberry Pi
- Przewody połączeniowe

Wykorzystane oprogramowanie:

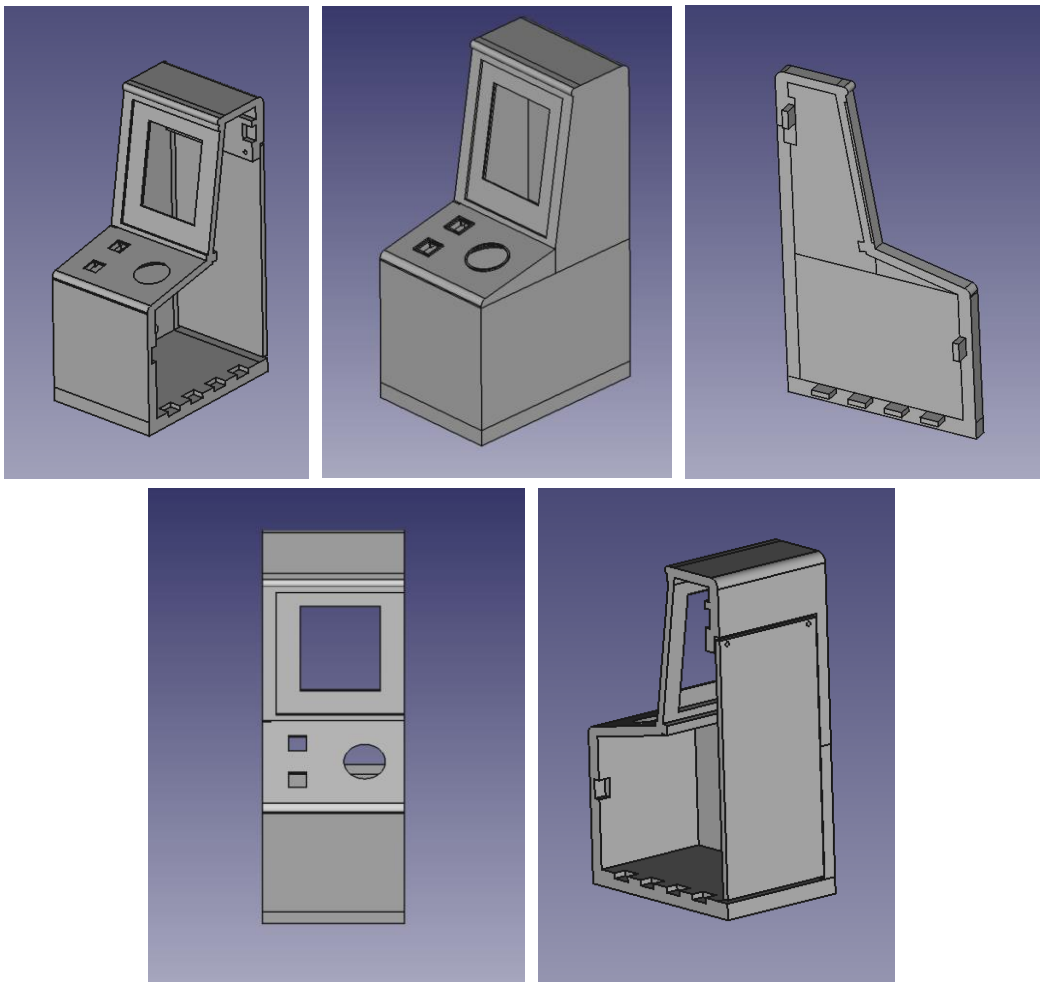
- Visual Studio Code – programowanie i prototypowanie logiki gier
- PyCharm Proffesional Edition – komunikacja i przesyłanie oprogramowania do Raspberry Pi 4B
- FreeCAD – modelowanie 3D obudowy

2. Schemat.

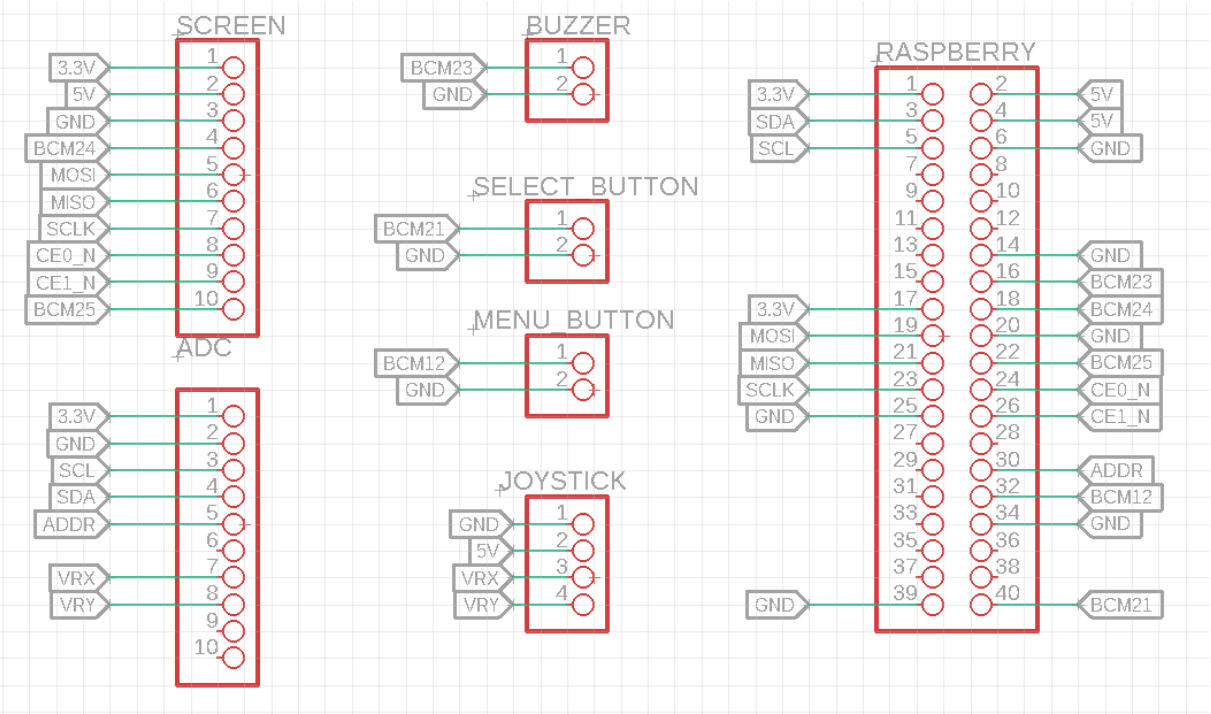
Schemat bazy danych:



Model 3D obudowy:



Schemat połączeniowy:



3. Zał oż enia projektowe a ich realizacja.

W założeniach projektowych planowaliśmy użyć przenośnego zasilania w postaci koszyka na akumulatory, jednakże ze względu na czas, jaki mogliśmy przeznaczyć na projekt oraz ryzyko jakie mogło nieść ze sobą zasilanie o niepewnym poziomie napięcia (ze względu na rozładowanie, np. możliwość błędnego odczytu wartości na joysticku przy operowaniu na niższym poziomie napięcia) postanowiliśmy stworzyć naszą maszynę bardziej zbliżoną oryginałom i uzależnić jej działanie od dostępu do gniazdka. Sprzęt zasilany jest poprzez dedykowaną do Raspberry Pi 4 ładowarkę USB-C.

Planowaliśmy również zaprojektować płytkę PCB do naszego projektu, jednak po dłuższym rozważeniu tej opcji uznaliśmy, że nie jest opłacalne poświęcać czas i pieniądze na dedykowaną płytkę, jako że cały nasz układ udało się połączyć zwykłymi przewodami z Raspberry Pi 4 w roli płytki PCB.

Projekt ma ogromne możliwości dalszego rozwoju. Nasz obecny projekt zajmuje niewielką frakcję obecnej pamięci, zostawiając sporo miejsca na możliwe przyszłe gry. Kombinacja joysticka i towarzyszących mu dwóch przycisków pozwala na przeróżne implementacje gier. Obecna implementacja reaguje wyłącznie na ruchy joysticka w kierunkach prawo/lewo/góra/dół. Można to zmienić w kodzie, aby pozwolić na odczyt pełnego zakresu ruchów joysticka, odkrywając nowe możliwe gry, czy techniki do wykorzystania w implementacji.

4. Listing kodu.

```
# Fragment pliku input.py zawierający obsługę urządzeń wejścia
# (joystick, przyciski)
def setup():
    # Inicjalizacja magistrali I2C
    i2c = busio.I2C(board.SCL, board.SDA)
    # Inicjalizacja konwertera ADC
    ads = ADS.ADS1115(i2c)

    # Inicjalizacja przycisków menu i interakcji
    global select_button
    global menu_button
    select_button = Button(19, bounce_time=0.1)
    menu_button = Button(26, bounce_time=0.1)

    # Przypisanie zmiennej do pinów, z których odczytujemy wartość na joysticku
    global y_axis
    global x_axis
    y_axis = AnalogIn(ads, ADS.P0)
    x_axis = AnalogIn(ads, ADS.P1)
    # Zmienna, która zawiera zwracaną wartość z joysticka:
    # NONE - joystick w stanie spoczynku
    # LEFT, RIGHT, UP, DOWN (lewo, prawo, góra, dół)
    # STANDBY - z joysticka została pobrana wartość, ale gracz nie sprowadził
joysticka do stanu spoczynku
    global joystick_value
    joystick_value = NONE

# Pobór wartości z przycisków
def get_button_input():
    if select_button.is_pressed:
        return SELECT
    if menu_button.is_pressed:
        return MENU
    return NONE


# Fragment pliku buzzer.py zawierającego obsługę buzzera

# Zainicjalizowanie buzzera pasywnego na pinie 26
def setup():
    global joystick_value
    buzzer = TonalBuzzer(26)
```

```
# Szczęśliwa melodia. Grana, kiedy gracz wygra/wpisze swoje imię do tabeli
# Kod polega na włączeniu buzzera na pewnym tonie, poczekaniu danego czasu
# I na końcu wyłączeniu buzzera
def play_happy_melody():
    buzzer.play(Tone('C5'))
    sleep(0.1)
    buzzer.play(Tone('E5'))
    sleep(0.1)
    buzzer.play(Tone('F5'))
    sleep(0.5)
    buzzer.stop()
    return


# Fragment pliku database.py zawierającego obsługę bazy danych tj. tabeli
wyników

# po uprzednim sprawdzeniu czy gracz osiągnął wynik godny tabeli i wpisaniu
imienia przez gracza, ta funkcja zapisuje do tabeli match_2 wynik z gry Match-2
# data - tuple w formie (imie_gracza, wynik)
def add_match_2_score(data):
    cursor.execute("SELECT COUNT(*) FROM match_2")
    number_scores = cursor.fetchone()
    cursor.execute("INSERT INTO match_2 (name, score) VALUES (?, ?)", (data[0],
data[1]))

    # limit najlepszych wyników to MAX_SCORES = 6
    # jeśli już było 6 wyników to usunie ten najgorszy
    if number_scores[0] >= MAX_SCORES:
        cursor.execute("""DELETE FROM match_2 WHERE id = (select * from (
                        SELECT id FROM match_2 ORDER BY score ASC limit 6,1)
                        AS t)""")
    conn.commit()


# Plik game.py:
# klasa Game odpowiadająca za cały przebieg rozgrywki

class Game:
    def __init__(self) -> None:
        os.environ["XDG_RUNTIME_DIR"] = "/tmp"
        # inicjalizacja biblioteki PyGame i obsługi czcionek
        pygame.init()
        pygame.font.init()
```

```
        # Załadowanie ekranu, czcionki, zegara
        self.screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_WIDTH))
        self.font = pygame.font.SysFont('arial', 24)
        self.clock = pygame.time.Clock()

        # GameStateManager - służy do obsługi zmiany scen
        # (gry, tabela wyników, ekran wpisywania imienia, wybór gry)
        self.game_state_manager = GameStatemanager('select')
        self.game_state_manager.setup(self)

        self.is_running = True
        # Inicjalizacja scen
        self.scenes = {'select': SelectScene(self.screen,
self.game_state_manager, self.font),
                       'match2': Match2(self.screen, self.game_state_manager,
self.font),
                       'simonsays': SimonSays(self.screen,
self.game_state_manager, self.font),
                       'minesweeper': MinesweeperScene(self.screen,
self.game_state_manager, self.font),
                       'leaderboard': LeaderboardScene(self.screen,
self.game_state_manager, self.font),
                       'entername': EnterNameScene(self.screen,
self.game_state_manager, self.font)}

    def run(self):
        while self.is_running:
            # Zaktualizuj wartość pobieraną z joysticka
            input.update_joystick()
            # Jeśli gracz kliknie przycisk MENU i nie znajduje się
            # w scenie wyboru gier to przenieś go do wyboru gier
            if input.get_button_input() == input.MENU and
self.game_state_manager.get_state() != 'select':
                self.game_state_manager.change_state('select')
            else:
                # Zaktualizuj input w danej scenie
                self.scenes[self.game_state_manager.get_state()].update()

            # Wyświetl obecną scenę
            self.scenes[self.game_state_manager.get_state()].run()
            pygame.display.update()
            # Ogranicz ilość FPS (klatek na sekundę)
            self.clock.tick(FPS)


# Plik match2.py - wyświetlanie i logika gry Match-2
```

```
class Match2(Scene):
    def __init__(self, display, game_state_manager, font) -> None:
        super().__init__(display, game_state_manager, font)

        self.time_text = self.font.render('Time: 0000', False, (255, 255, 255))

        # Dostępne kolory w grze, skopiowane dwa razy (para o takim samym
kolorze)
        colors = ["#66CCFF", "#F2973D", "#FF0000", "#FF99FF",
                        "#330066", "#0242BA", "#66FF66", "#FFFF57"]
        self.colors = colors + colors

        # Wskaźnik na pole, na którym jest gracz
        self.pointer = fig.Pointer(SQUARE_SIZE, ROWS, COLUMNS)

    # Funkcja wywoływana przy zmianie scen
    def enter(self):
        self.remaining = COLUMNS * ROWS / 2 # ile par zostało do odkrycia

        shuffle(self.colors) # pomieszanie kolorów
        # Dostępne kolory przedstawione jako macierz COLUMNS * ROWS
        self.available_colors = [self.colors[i * COLUMNS:(i + 1) * COLUMNS] for
i in range(ROWS)]

        self.squares = []
        # Zakrycie kwadratów, których nie odkryliśmy
        self.placeholder = fig.Square(0, 0, SQUARE_SIZE, '#757575')
        # 0 - nic, 1 - zaznaczony, 2 - odkryty
        self.squares_state = [[0 for _ in range(COLUMNS)] for _ in range(ROWS)]

        # Wypełnienie macierzy kwadratami z odpowiednimi kolorami
        for i in range(ROWS):
            y = i * (SQUARE_SIZE + 12) + 36
            l = []
            for j in range(COLUMNS):
                x = j * (SQUARE_SIZE + 12) + OFFSET_X
                square = fig.Square(x, y, SQUARE_SIZE,
self.available_colors[i][j])
                l.append(square)
            self.squares.append(l)

        # Przygotowanie zmiennych czasowych
        self.state_time = time.time()
        self.start_time = 0
        self.current_time = 0
```

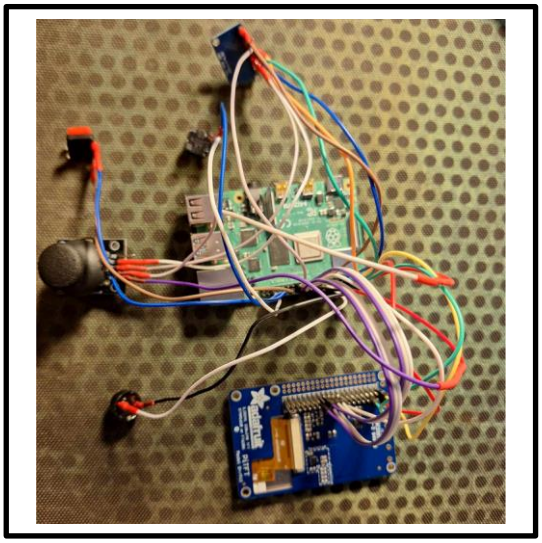
```
        self.state = 'show_all' # show-all – pokaż pary na starcie
                                # game - rozgrywka
                                # show – pokaż odkrytą parę

        self.selected = None

    # Obsługa inputu
    def update(self):
        if self.state != 'game':
            return
        # Przesuń wskaźnik
        joystick = input.get_joystick_input()
        if joystick == input.LEFT:
            self.pointer.move(-1, 0)
        elif joystick == input.RIGHT:
            self.pointer.move(1, 0)
        elif joystick == input.UP:
            self.pointer.move(0, -1)
        elif joystick == input.DOWN:
            self.pointer.move(0, 1)
        elif input.get_button_input() == input.SELECT:
            # Krótka melodia reakcji na kliknięcie kwadratu
            music_thread = threading.Thread(target=buzzer.play_select_melody,
daemon=True)
            music_thread.start()

            # Sprawdzanie czy para jaką gracz zaznaczył jest dobra
            current_row = self.pointer.current_index[0]
            current_col = self.pointer.current_index[1]
            if(self.squares_state[current_row][current_col] == 0):
                self.squares_state[current_row][current_col] = 1

                if(self.selected != None):

                    current_color =
self.available_colors[current_row][current_col]

                    prev_square_color =
self.available_colors[self.selected[0]][self.selected[1]]
                    if(current_color == prev_square_color):
                        # Ustawienie kwadratów na ‘odkryte’
                        self.squares_state[current_row][current_col] = 2
                        self.squares_state[self.selected[0]][self.selected[1]] =
2

                        self.remaining -= 1 # Odkryto pare
                    else:
```

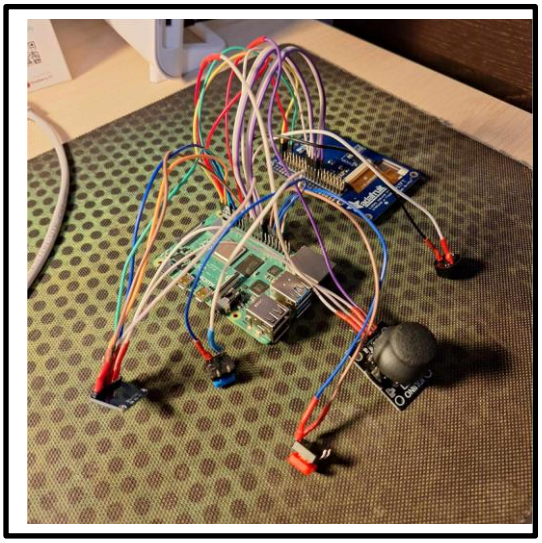
```
                        self.squares_state[current_row][current_col] = 0
                        self.squares_state[self.selected[0]][self.selected[1]] =
0

                    self.state_time = time.time()
                    self.state = 'show'
                else:
                    self.selected = (current_row, current_col)

            elif(self.squares_state[current_row][current_col] == 1):
                print('unselecting')
                self.selected = None
                self.squares_state[current_row][current_col] = 0
    def run(self):
        match self.state:
            case 'show_all':
                self.update_show_all()
            case 'game':
                self.update_game()
            case 'show':
                self.update_show()
            case _:
                pass

    # Pokazanie na starcie jak wygląda plansza
    def update_show_all(self):
        if time.time() - self.state_time > SHOW_ALL_TIME:
            self.state = 'game'
            self.start_time = time.time()
            return

        self.display.fill(pygame.Color('black'))

        for i in range(ROWS):
            for j in range(COLUMNS):
                self.display.blit(self.squares[i][j].surf, self.squares[i][j])

    # Widok gry
    def update_game(self):
        self.display.fill(pygame.Color('black'))

        for i in range(ROWS):
            y = i * (SQUARE_SIZE + 12) + 36
            for j in range(COLUMNS):
                x = j * (SQUARE_SIZE + 12) + OFFSET_X
                if(self.squares_state[i][j] == 0): # Zakryte - szary kwadrat
```

```
                    self.display.blit(self.placeholder.surf, (x, y))
                elif(self.squares_state[i][j] == 1): # Zaznaczone, pokaż kolor
                    self.display.blit(self.squares[i][j].surf,
self.squares[i][j])

        self.update_time()
        self.display.blit(self.time_text, (60, 2))

        pygame.draw.rect(self.display, pygame.Color('white'), self.pointer.rect,
4)

    # Jeśli gracz zaznaczy parę kwadratów, pokaż przez chwilę te kwadraty
    def update_show(self):
        if time.time() - self.state_time > SHOW_TIME:
            self.selected = None

            # Jeśli to była ostatnia para:
            # - puść melodię wygranej,
            # - sprawdź czy gracz dostał się do najlepszych wyników,
            # - przenieś gracza do menu głównego lub do sceny wprowadzania
imienia do tabeli,
            if self.remaining == 0:
                music_thread = threading.Thread(target=buzzer.play_happy_melody,
daemon=True)
                music_thread.start()
                if db.is_better_match_2(self.current_time):
                    self.game_state_manager.enter_name_scene('match_2',
self.current_time)
                    return

                self.game_state_manager.change_state('select')
                return

            self.state = 'game'
            return

        self.display.fill(pygame.Color('black'))

        for i in range(ROWS):
            y = i * (SQUARE_SIZE + 12) + 36
            for j in range(COLUMNS):
                x = j * (SQUARE_SIZE + 12) + OFFSET_X
                # Wyświetl zaznaczoną parę
                if ((i, j) == self.selected) or ([i, j] ==
self.pointer.current_index):
```

```
                    self.display.blit(self.squares[i][j].surf,
self.squares[i][j])
                elif (self.squares_state[i][j] == 0):
                    self.display.blit(self.placeholder.surf, (x, y))

        self.update_time()
        self.display.blit(self.time_text, (120 - self.time_text.get_width() //
2, 2))

    # Aktualizacja czau i jego tekstu
    def update_time(self):
        self.current_time = time.time() - self.start_time

        temp_string = f'{self.current_time:.1f}'.zfill(6)
        self.time_text = self.font.render(f'Time: {temp_string}', False, (255,
255, 255))
```

5. Zdjęcia zrealizowanego układu.





6. Podsumowanie i wnioski.

Nasz projekt zakończył się sukcesem. Połączyliśmy temat zastosowania podstaw systemów wbudowanych z naszą pasją do gier. Udało nam się osiągnąć oczekiwany efekt, zarówno pod kątem estetycznym jak i kątem programistycznym. Gry sprawiają przyjemność podczas grania, brak błędów psujących rozgrywkę. Wygląd gier jest prosty, ale i intuicyjny. Kod jest czytelny. Architektura programu jest łatwa do zrozumienia i rozszerzenia o większą ilość gier. Program działa płynnie i bez problemów. Obudowa jest estetyczna i odzwierciedla wygląd oryginalnych maszyn arkadowych.

Poznaliśmy różne formy komunikacji pomiędzy sprzętami. Do komunikacji komputera z Raspberry Pi używaliśmy protokołu SSH w celu wysyłania komend oraz protokołu SMTP do edytowania i przesyłania plików. W naszym projekcie wykorzystaliśmy urządzenia takie jak analogowy joystick i wyświetlacz. Wyświetlacz działa na interfejsie urządzeń peryferyjnych (SPI). Joystick natomiast najpierw przesyła sygnał do konwertera sygnału analogowego na cyfrowy (ADC), a następnie konwerter łączy się z Raspberry Pi na magistrali I2C. Nasz projekt wykorzystuje również proste urządzenia wejścia, tj. przyciski Tact switch, oraz proste urządzenia wyjścia, tj. buzzer pasywny.

Biblioteka PyGame świetnie się sprawdziła w naszym projekcie. Jej prostota pozwalała na szybką i schludną implementację gier. Mogliśmy również testować oprogramowanie uprzednio na komputerze, a następnie przetłumaczyć logikę klawiszy komputerowych na logikę wejść w urządzeniu.

Wyzwaniem dla nas był projekt od strony sprzętowej. To było nasze pierwsze doświadczenie z Raspberry Pi, więc na bieżąco musieliśmy aktualizować naszą wiedzę na temat tego mini komputera. Trudnym zadaniem też było poprawne podłączenie wyświetlacza do Raspberry Pi. Informacje w internecie na temat naszego wyświetlacza były znikome lub nieaktualne.