Urządzenia peryferyjne

Prowadzący zajęcia: dr inż. Jan Nikodem
Termin zajęć: Czwartek TN, godz. 14:10

Osoby wykonujące ćwiczenie: Mateusz Gawłowski, Bartosz Szymański	**Oznaczenie grupy:** B
Tytuł ćwiczenia: JOYSTIK STEROWANIE KURSOREM	**Data wykonania ćwiczenia:** 18.01.2024

Spis treści

1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z zasadami działania urządzeń typu joystik oraz implementacja programu z wykorzystaniem biblioteki DirectInput. Następnie należy napisać program wczytujący nazwę joystika, odczytujący zmianę położenia drążka oraz stan przycisków, program zastępujący działanie myszy oraz program pozwalający na rysowanie przy pomocy joystika.

2. Wstęp teoretyczny

Do nawiązywania komunikacji między sprzętem a jądrem systemu operacyjnego służy dedykowany sterownik urządzenia. Sterownik ten pełni funkcję programu odpowiedzialnego za obsługę danego urządzenia, działając jako pośrednik pomiędzy urządzeniem a systemem operacyjnym. System operacyjny Windows zawiera szeroki zakres sterowników obsługujących różnorodne urządzenia, co umożliwia natychmiastowe rozpoczęcie pracy z nowo podłączonym sprzętem. W trakcie zajęć, kontrolery były podłączane do złącza USB w komputerze, jednakże my zdecydowaliśmy się na wykorzystanie gniazda USB 3.0. Nowy standard jest kompatybilny w dół z USB 2.0 oraz 1.1, co gwarantuje poprawne wykrycie kontrolerów bez żadnych problemów. Początkowo używaliśmy aplikacji internetowej Hardware Tester, umożliwiającej sprawdzenie poprawności wykrywania ruchu drążka w joysticku oraz funkcji wszystkich przycisków.

Jedną z istotnych cech portu USB jest jego zgodność ze standardem Plug and Play, co oznacza, że komputer może obsługiwać urządzenia peryferyjne natychmiast po ich podłączeniu, bez potrzeby ingerencji użytkownika w konfigurację sprzętową komputera. Architektura USB składa się z serwera (hosta), wielu portów USB oraz podłączonych do nich urządzeń. Standard USB umożliwia również zasilanie podłączonych urządzeń napięciem 5V, co eliminuje potrzebę dodatkowego zasilania dla podłączonego joysticka. Należy jednak pamiętać, że maksymalny prąd w obwodzie wynosi 500mA (dla standardu 2.0), więc urządzenia wymagające większego prądu będą potrzebować dodatkowego źródła zasilania. Urządzenia zgodne ze specyfikacją USB 2.0 mogą osiągać maksymalną szybkość przesyłania danych wynoszącą 480 Mbit/s (60 MB/s), choć rzeczywista szybkość zależy od konkretnego urządzenia.

W trakcie programowania skorzystaliśmy z biblioteki DirectInput, dedykowanej obsłudze urządzeń wejściowych, takich jak klawiatura, mysz, joystick czy gamepad. Biblioteka ta zawiera funkcje umożliwiające odczyt danych z tych urządzeń w różny sposób (bezpośredni lub buforowany) oraz pozwala na przyporządkowanie konkretnych akcji do określonych przycisków (Action Mapping). API to jest głównie przeznaczone do tworzenia gier komputerowych, symulacji oraz innych interaktywnych aplikacji dla systemu Windows.

3. Użyte oprogramowanie

- Visual Studio IDE

4. Rozwiązania zadań wraz z krótkim opracowaniem

4.1 Odczytywnie nazwy zainstalowanego joysticka

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.ComponentModel;
4.  using System.Data;
5.  using System.Drawing;
6.  using System.Linq;
7.  using System.Text;
8.  using System.Threading;
9.  using System.Windows.Forms;
10. using SharpDX.DirectInput;
11.
12. namespace DirectInputNamespace {
13.     public partial class Selector : Form {
14.
15.         SharpDX.DirectInput.DirectInput directInput = new SharpDX.DirectInput.Direct
    Input();
16.         List<DeviceInstance> deviceList = new List<DeviceInstance>();  // tworzymy l
    iste znalezionych urzadzen
17.         public Selector() {
18.             InitializeComponent();
19.
20.             var devices = directInput.GetDevices(DeviceType.Joystick, DeviceEnumerat
    ionFlags.AttachedOnly);  // 'zlap' wszystkie urzadzenia typu joystick
21.             var devices2 = directInput.GetDevices(DeviceType.Gamepad, DeviceEnumerat
    ionFlags.AttachedOnly);  // 'zlap' wszystkie urzadzenia typu gamepad
22.
23.             foreach(DeviceInstance instance in devices) { // dodaj kazdy joystick do
     listy znalezionych urzadzen
24.                 deviceList.Add(instance);
25.                 listBox1.Items.Add(instance.InstanceName);
26.             }
27.
28.             foreach(DeviceInstance instance in devices2) { // dodaj kazdy gamepad do
     listy znalezionych urzadzen
29.                 deviceList.Add(instance);
30.                 listBox1.Items.Add(instance.InstanceName);
31.             }
32.         }
33.
34.         private void listBox1_SelectedIndexChanged(object sender, EventArgs e) {  //
     wlacz przyciski tylko wtedy kiedy wybierzemy urzadzenie z listy
35.             button1.Enabled = true;
36.             button2.Enabled = true;
37.             button3.Enabled = true;
38.         }
39.
40.         private void button1_Click(object sender, EventArgs e) {  // wybralismy emul
    ator myszki
41.             this.Hide();
42.             int wybranyKontroler = listBox1.SelectedIndex;  // zaznaczone urzadzenie
     staje sie naszym kontrolerem
43.             Joystick joystick = new Joystick(directInput, deviceList.ElementAt(wybra
    nyKontroler).InstanceGuid);
44.             joystick.Acquire();
45.
46.             EmulatorMyszy emulatorMyszy = new EmulatorMyszy(joystick);  //dzieki tem
    u wlaczamy mozliwosc emulacji myszy
47.             new Thread(new ThreadStart(emulatorMyszy.WlaczEmulacje)).Start();
48.             MouseEmulation me = new MouseEmulation();  // dzieki temu otworzy sie ok
    ienko mowiace o trwajacej emulacji myszki
49.             me.ShowDialog();
```

```
50.         }
51.
52.         private void button2_Click(object sender, EventArgs e) {  // wybralismy edyt
    or graficzny -> rysowanie
53.             this.Hide();
54.             int wybranyKontroler = listBox1.SelectedIndex; // zaznaczone urzadzenie
    staje sie naszym kontrolerem
55.             Joystick device = new Joystick(directInput, deviceList.ElementAt(wybrany
    Kontroler).InstanceGuid);
56.             device.Acquire();
57.             Canvas canvas = Canvas.CreateCanvas(true);  // dzieki temu otworzy sie o
    kno graficzne w ktorym mozna rysowac
58.             Paint paint = new Paint(canvas, device); // dzieki temu wlaczamy mozliwo
    sc rysowania za pomoca kontrolera
59.             paint.InputThread();
60.             this.Close();
61.         }
62.
63.         private void button3_Click(object sender, EventArgs e) {  // wybralismy test
    owanie dzialania kontrolera
64.             this.Hide();
65.             int wybranyKontroler = listBox1.SelectedIndex; // zaznaczone urzadzenie
    staje sie naszym kontrolerem
66.             Joystick joystick = new Joystick(directInput, deviceList.ElementAt(wybra
    nyKontroler).InstanceGuid);
67.             joystick.Acquire();
68.             Canvas canvas = Canvas.CreateCanvas(false);  // dzieki temu otworzy sie
    okno gdzie gdzie będziemy widzieli zmiany wartości osi X, Y, Z oraz przycisku fire
69.             canvas.dodajJoystick(joystick);
70.             TestKontrolera testKontrolera = new TestKontrolera(canvas, joystick);  /
    / dzieki temu wlaczamy mozliwosc testowania kontrolera
71.             testKontrolera.InputThread();
72.             this.Close();
73.         }
74.     }
75. }
```

Opracowanie

Poniższy kod to implementacja programu w języku C# przy użyciu biblioteki SharpDX, umożliwiającego obsługę urządzeń wejściowych, takich jak joysticki i gamepady. Poniżej znajduje się krótki opis działania poszczególnych fragmentów kodu:

1. Linie 1-10: Importowanie niezbędnych bibliotek.

2. Linie 12-75: Definicja przestrzeni nazw oraz klasy `Selector`, która dziedziczy po klasie `Form` z Windows Forms.

- Linie 15-16: Inicjalizacja obiektu `DirectInput` oraz listy `deviceList` przechowującej informacje o znalezionych urządzeniach.
- Linie 20-31 Znalezienie i dodanie do listy urządzeń typu joystick i gamepad. Następnie wyświetlenie nazw tych urządzeń w kontrolce `listBox1`.
- Linie 34-38: Obsługa zdarzenia zmiany zaznaczenia w kontrolce `listBox1`. Przyciski są aktywowane tylko wtedy, gdy użytkownik wybierze urządzenie z listy.
- Linie 40-49: Obsługa zdarzenia kliknięcia przycisku "Emulator myszki". Tworzenie obiektu `Joystick` na podstawie wybranego urządzenia, akwizycja joysticka, utworzenie obiektu

`EmulatorMyszy`, uruchomienie wątku obsługującego emulację myszy, oraz otwarcie okna informacyjnego.

- Linie 52-60: Obsługa zdarzenia kliknięcia przycisku "Edytor graficzny -> rysowanie". Podobnie jak poprzedni przycisk, tworzy obiekt `Joystick`, akwizycja joysticka, utworzenie obiektu `Canvas` i `Paint` umożliwiających rysowanie za pomocą kontrolera, a następnie zamknięcie bieżącego okna.
- Linie 63-72: Obsługa zdarzenia kliknięcia przycisku "Testowanie działania kontrolera". Analogicznie do poprzednich przypadków, tworzy obiekt `Joystick`, akwizycja joysticka, utworzenie obiektów `Canvas` i `TestKontrolera` umożliwiających testowanie działania kontrolera, a następnie zamknięcie okna.
- Linie 74-75: Zakończenie definicji klasy.

4.2 'TestKontrolera' - ilustrująca działanie joysticka

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading;
6.  using SharpDX.DirectInput;
7.
8.  namespace DirectInputNamespace {
9.      class TestKontrolera {
10.         private Canvas canvas;
11.         private Joystick joystick;
12.         public TestKontrolera(Canvas canvas, Joystick joystick) {
13.             this.canvas = canvas;
14.             this.joystick = joystick;
15.         }
16.         public void InputThread() {
17.             canvas.showCross = true;  // aby pokazal sie wskaznik
18.             while(canvas.Visible) {
19.                 int x = joystick.GetCurrentState().X;  // sczytujemy obecna pozycje
    X joysticka
20.                 int y = joystick.GetCurrentState().Y;  // sczytujemy obecna pozycje
    Y joysticka
21.                 // zmienna aby ograniczyc mozliwosci poruszania sie wskaznika
22.                 int maxInput = (1 << 16) - 1;  // przesuniecie bitowe o 16 pozycji w
     lewo i pozniej odjecie 1
23.                 // wywolanie metody poruszania wskaznikiem (pozycja x, pozycja y), c
    iagle wyliczana
24.                 canvas.poruszWskaznikiem(x * canvas.SzerokoscOkna / maxInput, y * ca
    nvas.WysokoscOkna / maxInput);
25.                 Thread.Sleep(10);
26.             }
27.         }
28.     }
29.
30. }
```

Opracowanie

Poniższy kod to implementacja klasy `TestKontrolera` w języku C# przy użyciu biblioteki SharpDX, służącej do testowania działania kontrolera poprzez poruszanie wskaznikiem na oknie graficznym. Poniżej znajduje się krótki opis działania poszczególnych fragmentów kodu:

1. Linie 1-6: Importowanie niezbędnych bibliotek.

2. Linie 8-28: Definicja przestrzeni nazw oraz klasy `TestKontrolera`.

   - Linie 10-14: Prywatne pola klasy `canvas` i `joystick`, które przechowują obiekty `Canvas` i `Joystick` odpowiednio. Konstruktor klasy przyjmuje te obiekty jako argumenty i przypisuje do odpowiednich pól.
   - Linie 16-26: Metoda `InputThread`, która jest wątkiem odpowiedzialnym za ciągłe monitorowanie stanu joysticka i poruszanie wskaznikiem na oknie graficznym.
   - Linia 17: Ustawienie flagi `showCross` w obiekcie `canvas` na `true`, aby wyświetlić wskaźnik.
   - Linie 18-26: Pętla while, która działa dopóki okno `canvas` jest widoczne. Wewnątrz pętli:
   - Linie 19-20: Pobranie obecnej pozycji X i Y joysticka za pomocą metody `GetCurrentState()`.
   - Linia 22: Ustalenie maksymalnej wartości `maxInput`, aby ograniczyć możliwości poruszania się wskaznika.
   - Linie 24: Wywołanie metody `poruszWskaznikiem` z obiektu `canvas`, przekazując przeskalowane wartości X i Y.
   - Linia 25: Odczekanie 10 milisekund, aby uniknąć nadmiernego obciążenia wątku.
   - Linia 27: Zakończenie metody `InputThread`.
   - Linie 30-31: Zakończenie definicji klasy.

4.3 Edytor graficzny – rysowanie przy pomocy Joysticka

4.3.1 Okno edytora graficznego

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.ComponentModel;
4.  using System.Data;
5.  using System.Drawing;
6.  using System.Linq;
7.  using System.Text;
8.  using System.Threading.Tasks;
9.  using System.Windows.Forms;
10. using SharpDX.DirectInput;
11.
12. namespace DirectInputNamespace {
13.     public partial class Canvas : Form {
14.         private System.Threading.Thread myThread;
15.         private Joystick joystick;
16.         public bool czyEdytorGraficzny = false;  // zmienna dzieki ktorej wiemy czy
    chcemy uzywac edytora graficznego czy nie
17.
18.         float x = 0, y = 0;
```

```
19.         public bool showCross = false;  // domyslnie wskaznik w ekranie graficznyj j
    est wylaczony
20.
21.         private Canvas() {
22.             InitializeComponent();
23.         }
24.
25.         public void dodajJoystick(Joystick joystick) {
26.             this.joystick = joystick;
27.         }
28.
29.
30.         public static Canvas CreateCanvas(bool czyEdytorGraficzny) {  // jako argume
    nt podajemy zmienna false albo true
31.             Canvas canvas = null;
32.             System.Threading.Thread thread = new System.Threading.Thread(() => {  //
     watek w ktorym dziala caly edytor graficzny
33.                 canvas = new Canvas();
34.                 canvas.czyEdytorGraficzny = czyEdytorGraficzny;  // wartosc bool bed
    zie utrzymana przez caly watek
35.                 canvas.ShowDialog();
36.             });
37.             thread.Start();
38.             while(canvas == null) {
39.                 System.Threading.Thread.Sleep(10);
40.             }
41.             canvas.myThread = thread;
42.
43.             return canvas;
44.         }
45.
46.         public int SzerokoscOkna { get { return pictureBox1.Width; } }  // zwracamy
    szerokosc naszego okna graficznego
47.         public int WysokoscOkna { get { return pictureBox1.Height; } }  // zwracamy
    wysokosc naszego okna graficznego
48.
49.         private void Timer1_Tick(object sender, EventArgs e) {  // z kazdym kolejnym
     tickiem watku, sprawdzamy
50.             // czy zostalo cos zmienione, jezeli tak to odswiezamy, aby zobaczyc zmi
    any inaczej to omijamy
51.             if(updated) {
52.                 updated = false;
53.                 Refresh();
54.             }
55.
56.             if(!czyEdytorGraficzny) {  // jezeli nie chcemy uzywac edytora graficzne
    go, wtedy
57.                                        // zamiast niego pojawiaja sie pola tekstowe
    i etykiety
58.                                        // uzywane podczas testowania kontrolera
59.                 if(joystick.GetCurrentState().Buttons[0]) { // jezeli wcisniemy przy
    cisk 1, czyli fire
60.                     textBox2.Text = "Wciśnięty";
61.                 }
62.                 else {
63.                     textBox2.Text = "Nie wciśnięty";
64.                 }
65.                 // pokazuje aktualny stan osi Z -> slider
66.                 textBox1.Text = String.Format("{0}%", (float)joystick.GetCurrentStat
    e().Z / ((1 << 16) - 1) * 100);
67.                 // pokazuje aktualny stan osi Y
68.                 textBox4.Text = String.Format("{0}%", (float)joystick.GetCurrentStat
    e().Y / ((1 << 16) - 1) * 100);
69.                 // pokazuje aktualny stan osi X
70.                 textBox3.Text = String.Format("{0}%", (float)joystick.GetCurrentStat
    e().X / ((1 << 16) - 1) * 100);
```

```
71.             }
72.             else {  // jezeli uzywamy edytora graficznego to chowamy wszystkie niepo
    trzebne pola
73.                 textBox1.Hide();
74.                 textBox2.Hide();
75.                 textBox3.Hide();
76.                 textBox4.Hide();
77.
78.                 label1.Hide();
79.                 label2.Hide();
80.                 label3.Hide();
81.                 label4.Hide();
82.             }
83.
84.         }
85.         private void PictureBox1_Paint(object sender, PaintEventArgs e) { // metoda
    dzieki ktorej pojawia sie nam nasz pedzel
86.             lock(bmp) {
87.                 e.Graphics.DrawImage(bmp, 0, 0);
88.             }
89.             using(SolidBrush pedzel = new SolidBrush(Color.Black)) { // nasz kursor
    jest czarny
90.                 e.Graphics.FillEllipse(pedzel, x - 4, y - 4, 2 * 4, 2 * 4);  // jest
     mala elipsa, a w tym przypadku kolem
91.             }
92.         }
93.
94.         public void WyczyscEkran() {  // metoda dzieki ktorej czyscimy nasz ekran
95.             bmp = new Bitmap(this.pictureBox1.Width, this.pictureBox1.Height);  // t
    worzymy bitmape naszego okna w ktorym bedziemy pracowali
96.             lock(bmp) {
97.                 using(Graphics g = Graphics.FromImage(bmp))
98.                 using(SolidBrush pedzel = new SolidBrush(Color.White)) {  // szybko
    ustawiamy kolor pedzla na bialy
99.                     g.FillRectangle(pedzel, 0, 0, SzerokoscOkna, WysokoscOkna);  //
    wypelniamy cale okno bialym kolorem
100.                    }
101.                    updated = true;  // ustawiamy flage aby edytor zauwazyl zmian
    y i odswiezyl sie
102.                }
103.            }
104.            public void Rysuj(float x, float y, float r) {  // metoda dzieki ktor
    ej mozemy rysowac
105.                lock(bmp) {
106.                    using(Graphics g = Graphics.FromImage(bmp))
107.                    using(SolidBrush pedzel = new SolidBrush(Color.Green)) {  //
    pedzel ustawiamy na kolor zielony, aby rysowac na zielono
108.                        g.FillEllipse(pedzel, x - r, y - r, 2 * r, 2 * r);  //  t
    ym razem nasze kolo/elipsa zalezy od wartosci r
109.                    }
110.                    updated = true;  // dzieki temu edytor sie odswiezy
111.                }
112.            }
113.
114.            public void poruszWskaznikiem(float x, float y) {  // metoda dzieki k
    torej mozemy poruszac wskaznikiem
115.                this.x = x;
116.                this.y = y;
117.                updated = true;  // zmiana flagi aby edytor sie odswiezyl
118.            }
119.
120.            private void Canvas_Load(object sender, EventArgs e) {
121.            }
122.        }

}
```

4.3.2 Rysowanie w oknie graficznym:

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.  using System.Threading;
7.  using SharpDX.DirectInput;
8.
9.  namespace DirectInputNamespace {
10.     class Paint {
11.         // ustawienie poczatkowych wartosci
12.         private float posX = 0, posY = 0;
13.         private float mouseSpeed = 100;
14.         private Canvas canvas;
15.         private Joystick joystick;
16.         public Paint(Canvas canvas, Joystick device) {
17.             this.canvas = canvas;
18.             this.joystick = device;
19.         }
20.         public void InputThread() {
21.             canvas.WyczyscEkran();  // przygotowujemy nasz ekran do rysowania
22.             while(canvas.Visible) { // jezeli canvas jest widoczny, czyli po prostu
    istnieje
23.                 int inputOffset = (1 << 15) - 1;
24.                 int x = joystick.GetCurrentState().X - inputOffset;
25.                 int y = joystick.GetCurrentState().Y - inputOffset;
26.
27.                 float xOffset = (float)x / inputOffset;
28.                 float yOffset = (float)y / inputOffset;
29.
30.                 int slider = joystick.GetCurrentState().Z;
31.                 mouseSpeed = 150 / ((1 << 16) - 1) + 1;
32.                 posX += xOffset * mouseSpeed;
33.                 posY += yOffset * mouseSpeed;
34.                 // wyzej tak jak wszedzie, odczytywanie i wyliczanie wartosci osi X,
     Y, Z joysticka
35.
36.                 // zebysmy nie wyszli poza nasze okno
37.                 if(posX > canvas.SzerokoscOkna) {
38.                     posX = canvas.SzerokoscOkna;
39.                 }
40.                 if(posX < 0) {
41.                     posX = 0;
42.                 }
43.                 if(posY > canvas.WysokoscOkna) {
44.                     posY = canvas.WysokoscOkna;
45.                 }
46.                 if(posY < 0) {
47.                     posY = 0;
48.                 }
49.
50.                 // uzywamy przycisku 1 na joysticku do rysowania
51.                 if(joystick.GetCurrentState().Buttons[0]) {
52.                     slider /= 1000;
53.                     // slider w tym przypadku zmienia wartosc 'r' elipsy/kola, czyli
     dzieki temu mozemy rysowac
54.                     // ciensze lub gruszbe linie, powiekszajac lub zmniejszajac nasz
     pedzel
55.                     canvas.Rysuj(posX, posY, slider);
56.                 }
57.                 // jezeli wcisniemy przycisk 3 na joysticku, czyscimy ekran
58.                 if(joystick.GetCurrentState().Buttons[2]) {
59.                     canvas.WyczyscEkran();
```

```
60.                 }
61.                 canvas.poruszWskaznikiem(posX, posY);  // za kazdym razem mozemy po
    prostu poruszac wskaznikiem/pedzlem
62.
63.                 Thread.Sleep(10); // jak nic nie robimy, to usypiamy watek, mozemy g
    o potrzebowac
64.             }
65.         }
66.     }
67. }
```

Opracowanie

Canvas Class:

Poniższy kod to implementacja klasy `Canvas` w języku C# przy użyciu biblioteki SharpDX, służącej do zarządzania oknem graficznym. Poniżej znajduje się krótki opis działania poszczególnych fragmentów kodu:

1. Linie 1-10: Importowanie niezbędnych bibliotek.

2. Linie 12-122: Definicja przestrzeni nazw oraz klasy `Canvas` dziedziczącej po klasie `Form` z Windows Forms.

- Linie 14-19: Prywatne pola klasy `myThread`, `joystick`, `czyEdytorGraficzny`, `x`, `y` oraz `showCross` przechowujące informacje o wątku, joysticku, trybie edytora graficznego, pozycji X i Y, oraz widoczności wskaźnika.

- Linie 21-23: Prywatny konstruktor, inicjalizujący komponenty formularza.

- Linie 25-26: Metoda `dodajJoystick`, przypisująca obiekt joysticka do pola `joystick`.

- Linie 30-43: Metoda statyczna `CreateCanvas`, tworząca nowy obiekt `Canvas` w osobnym wątku, zależnie od argumentu `czyEdytorGraficzny`.

- Linie 46-47: Właściwości zwracające szerokość i wysokość okna graficznego.

- Linie 49-84: Obsługa zdarzenia ticka timera (`Timer1_Tick`), odpowiadającego za odświeżanie widoku.

- Linie 85-91: Obsługa zdarzenia rysowania na `PictureBox` (`PictureBox1_Paint`), umożliwiająca wyświetlanie wskaźnika i obrazu.

- Linie 94-121: Metody `WyczyscEkran`, `Rysuj` i `poruszWskaznikiem`, odpowiedzialne za czyszczenie ekranu, rysowanie na ekranie oraz poruszanie wskaźnikiem.

- Linia 120: Pusta metoda `Canvas_Load`.

Paint Class:

Poniższy kod to implementacja klasy `Paint` w języku C# przy użyciu biblioteki SharpDX, służącej do rysowania na oknie graficznym za pomocą joysticka. Poniżej znajduje się krótki opis działania poszczególnych fragmentów kodu:

1. Linie 1-7: Importowanie niezbędnych bibliotek.

2. Linie 9-66: Definicja przestrzeni nazw oraz klasy `Paint`.

   - Linie 12-18: Prywatne pola klasy `posX`, `posY`, `mouseSpeed`, `canvas` i `joystick` przechowujące informacje o pozycji X i Y, prędkości myszy, obiekcie `Canvas` i joysticku odpowiednio.

   - Linie 20-30: Konstruktor klasy, przyjmujący obiekty `Canvas` i `Joystick` jako argumenty.

   - Linie 20-65: Metoda `InputThread`, odpowiadająca za ciągłe monitorowanie stanu joysticka i rysowanie na ekranie w odpowiedzi na ruchy joysticka.

Zarówno w klasie `Canvas`, jak i `Paint` można zauważyć wykorzystanie biblioteki SharpDX do obsługi urządzeń wejściowych (joysticka) oraz rysowania na ekranie. Wspólnie tworzą funkcjonalność umożliwiającą interakcję z graficznym interfejsem użytkownika za pomocą kontrolera.

5. Wnioski

Realizacja ćwiczenia w znaczący sposób wpłynęła na nasz poziom rozumienia tematu w obszarze komunikacji oraz obsługi urządzeń zewnętrznych takich jak joystick czy gamepad. Podczas trwania laboratorium udało się utworzyć spełniający założenia zadania kod, który został przedstawiony i krótko opisany powyżej.

6. Literatura

Poprzez literaturę należy rozumieć wszelkie materiały potrzebne do zrozumienia założeń laboratoriów, sposobu realizacji ćwiczenia oraz źródeł pomocy podczas rozwiązywania problemów napotkanych w trakcie realizacji zajęć.

   - Gamepad Tester: https://hardwaretester.com/gamepad