

PROJEKT

WIZUALIZACJA DANYCH SENSORYCZNYCH

Prosta gra zręcznościowa - Piłeczka

Martyna Żukowska , 252877



Prowadzący:

dr inż. Bogdan Kreczmer

Katedra Cybernetyki i Robotyki
Wydziału Elektroniki, Fotoniki i
Mikrosystemów
Politechniki Wrocławskiej

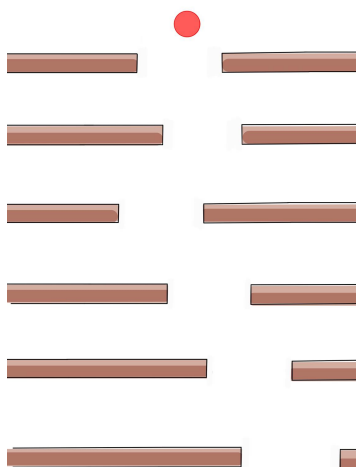
16 czerwca 2022

Spis treści

1	Charakterystyka tematu projektu	1
2	Podcele i etapy realizacji projektu	1
3	Specyfikacja finalnego produktu	2
4	Terminarz realizacji poszczególnych podcelów (z dokładnością do 1 tygodnia)	2
5	Projekt graficznego interface użytkownika	4
6	Opis gry	5
6.1	Początkowe ustawienia	5
6.2	Gra	5
6.3	Rozpoczęcie nowej gry	6
7	Konfiguracja Stm32	6
8	Komunikacja	7
9	Ramka danych	8
10	Gra	9
10.1	Początki OpenGL	9
10.2	Udoskonalenie Ustawień	10
10.3	Poruszanie się piłeczki przy pomocy płytki	10
10.4	Skalowanie	11
10.5	Tłumaczenie	11
10.6	Losowe generowanie przeszkód	13
10.7	Kolizja	13
10.8	Życia	13
10.9	Podsumowanie	14
11	Dokumentacja w doxygenie	15
12	Test działania aplikacji	17

1 Charakterystyka tematu projektu

Tematem projektu jest prosta gra zręcznościowa 2D. Na ekranie wyświetlana będzie 'spadająca piłka', którą gracz będzie mógł sterować przy pomocy płytki z żyroskopem obracając moduł płytki w prawo, bądź w lewo. Z dołu do góry będą przemieszczać się przeszkody, które gracz będzie starał się ominąć poruszając się (Rysunek1). W momencie, gdy gracz wpadnie na przeszkodę i nie uda mu się jej uniknąć więcej niż 3 razy, gra zostanie zakończona. Punkty w grze będą naliczane na podstawie ilości ominiętych przeszkód.



Rysunek 1: Podgląd możliwego wyglądu gry

2 Podcele i etapy realizacji projektu

Bardziej szczegółowe przedstawienie zagadnień związanych z danym tematem. Wyodrębnienie podcelów.

Lista podcelów:

- Przegląd literatury i zasobów Internetu związanych z tematem projektu
- Projekt układu elektronicznego (schemat ideowy)
- Wykorzystanie mikroprocesora STM32F429 na płytce uruchomieniowej STM32F429IDISC1
- Odbieranie danych z 3-osiowego żyroskopu MEMS L3GD20 i przekazywanie ich w odpowiedniej formie do wizualizacji
- Stworzenie prostego obiektu poruszającego się zgodnie z poruszaniem płytką.
- Próba zminimalizowania dryfu poprzez implementację filtra górnoprzepustowego.
- Dodanie przeszkód poruszających się z dołu do góry
- Wykrywanie kolizji na wizualizacji w momencie wpadnięcia piłeczki na przeszkodę przez piłeczkę. (3 możliwe na grę)
- Wprowadzenie restart gry przy kolizji.

3 Specyfikacja finalnego produktu

Finalna wizualizacja będzie:

- z rozsądną dokładnością pokazywała umieszczenie piłeczki na planszy, zgodnie z danymi z żyroskopu.
- rozpoczynała się od początku w momencie kolizji z napływającymi przeszkodami.
- prezentowała się jako przystępna prosta gra zręcznościowa.

Możliwa rozbudowa wizualizacji w 3D, w zależności od dostępnego czasu i szybkości postępowania projektu.

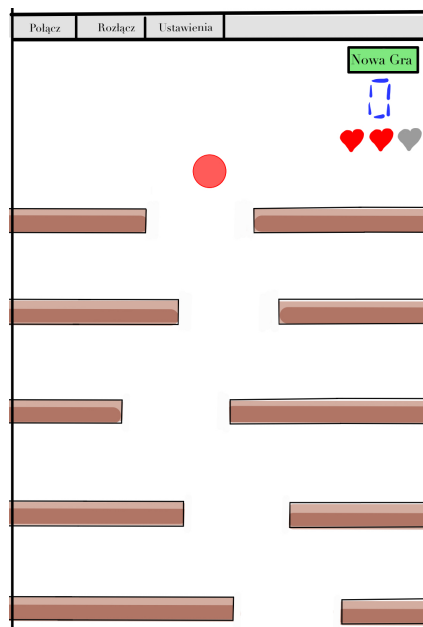
4 Terminarz realizacji poszczególnych podcelów (z dokładnością do 1 tygodnia)

- 29 marca 2022 – zakończenie przeglądu materiałów związanych z danym tematem
- 5 kwietnia 2022 – schemat układu elektronicznego, wstępna konfiguracja płytki STM.
- 12 kwietnia 2022 – Pojawienie się prostego obiektu w wizualizacji, zapoznanie się ze środowiskiem, pierwsze kroki projektu. W tym momencie pierwsze elementy powinny już znajdować się w wizualizacji.
- 26 kwietnia 2022 – Odczyt danych z żyroskopu, próba przekierowania ich w odpowiedniej formie.
- 10 maja 2022 – Poruszanie się piłeczki na wizualizacji zgodnie z danymi z żyroskopu. Na początku proste poruszanie obiektem, w późniejszych etapach poruszanie nim zgodnie z odbieranymi danymi.
- 17 maja 2022 – Próba eliminacji dryfu, najprawdopodobniej przy pomocy filtra górnoprzepustowego. W razie niepowodzenia wypróbowanie innych możliwości.
- 24 maja 2022 – Dodanie napływających przeszkód w wizualizacji. Na początek pojawienie się pojedynczych elementów, w następnych krokach powinny się pojawiać w odpowiednich miejscach i poruszać ku górze z odpowiednią prędkością.
- 31 maja 2022 – Wprowadzenie kolizji w momencie wpadnięcia piłeczki na przeszkodę. Pojawienie się komunikatu w razie kolizji.
- 7 czerwca 2022 – Restartowanie gry w momencie wpadnięcia piłeczki na przeszkodę. Dodatkowo testowanie płynności całej gry.
- 14 czerwca 2022 – Możliwa rozbudowa prostej gry zręcznościowej

5 Projekt graficznego interface użytkownika

Główne okno programu przedstawione zostało na rysunku nr 5. Składa się ono z 3 części:

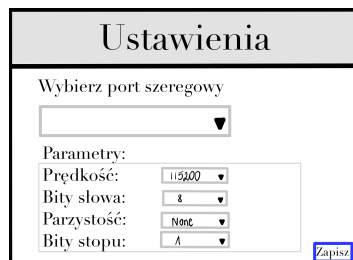
- Okno z główną zawartością gry zręcznościowej. Główna część wizualizacji. Tu będą wygenerowane prostokątne przeszkody , które będą dostatecznie płynnie poruszały się z dolnej, do górnej części okna. Gracz może poruszać się w prawo lub w lewo piłeczką sprawnie ich unikając. Ma w tym celu do dyspozycji całe dostępne okienko
- W prawym górnym rogu znajdują się przycisk do obsługi gry : NOWA GRA , który umożliwia, po jego wciśnięciu, wygenerowanie od początku nowej gry (tracąc jej aktualny stan)
- Dodatkowo poniżej widać 3 możliwe życia, ratujące przed zakończeniem gry w momencie kolizji (kolor świadczy o tym, czy dalej je posiadamy: czerwony-tak , szary-nie). Wyświetlany jest też wynik gracza w formie punktów uzyskanych w trakcie gry.
- W górnym menu programu widać zakładkę:
 - Połącz (łączy aplikację z wyszukanym portem)
 - Rozłącz (rozłącza aplikację z mikroprocesorem)
 - Ustawienia (wyświetla okno do modyfikacji ustawień połączenia portu)



Rysunek 3: Szkic wyglądu aplikacji

Okno ustawień połączenia

Jest ono stworzone do wyboru portu szeregowego. Można w nim zmieniać parametry połączenia, takie jak prędkość transmisji, ilość bitów słowa, parzystość, czy ilość bitów stopu. Wygląd tego okna został zaprezentowany na rysunku 4.



Rysunek 4: Szkic wyglądu okna ustawień

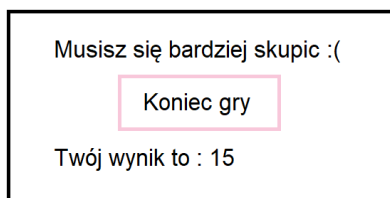
6 Opis gry

6.1 Początkowe ustawienia

Po stworzeniu okna na samym początku należy połączyć się z mikroprocesorem. Do wyboru portu szeregowego, z którym chcemy się połączyć służy specjalne okno uruchamiane z głównego menu gry poprzez **Ustawienia**. W nim można zmieniać parametry połączenia, jak zostało to pokazane na rysunku 4. Dane przypisane w ustawieniach są zapisywane i po wciśnięciu w przycisk **Połącz** w głównym menu można zaczynać rozgrywkę. Bez początkowego wyboru ustawień i połączenia się przez odpowiedni port gra będzie niemożliwa.

6.2 Gra

Po konfiguracji można przejść do docelowej gry. Piłeczka oraz przeszkody pojawią się w momencie wciśnięcia przycisku **Nowa Gra**. Po pojawieniu się elementów w okienku gra startuje, wtedy, gracz może się poruszać w prawo lub w lewo unikając przeszkód napływających z dołu. Po każdej ominiętej przeszkodzie gracz zyskuje jeden punkt. Po ewentualnym wpadnięciu na przeszkodę, w przypadku gdy gracz posiada jeszcze dostępne życia, traci jedno z nich. W momencie gdy życia się mu skończą, a doszło do kolizji pojawia się komunikat zakończenia gry, na którym dodatkowo wyświetla się komunikat podsumowujący ilość zebranych punktów w czasie rozgrywki, jak zostało to pokazane na rysunku 7.



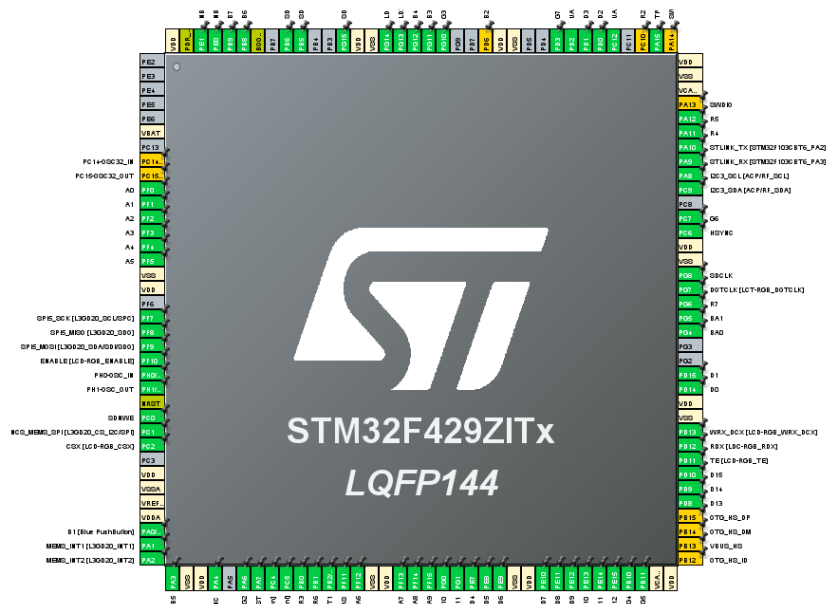
Rysunek 5: Okno po ukończeniu gry

6.3 Rozpoczęcie nowej gry

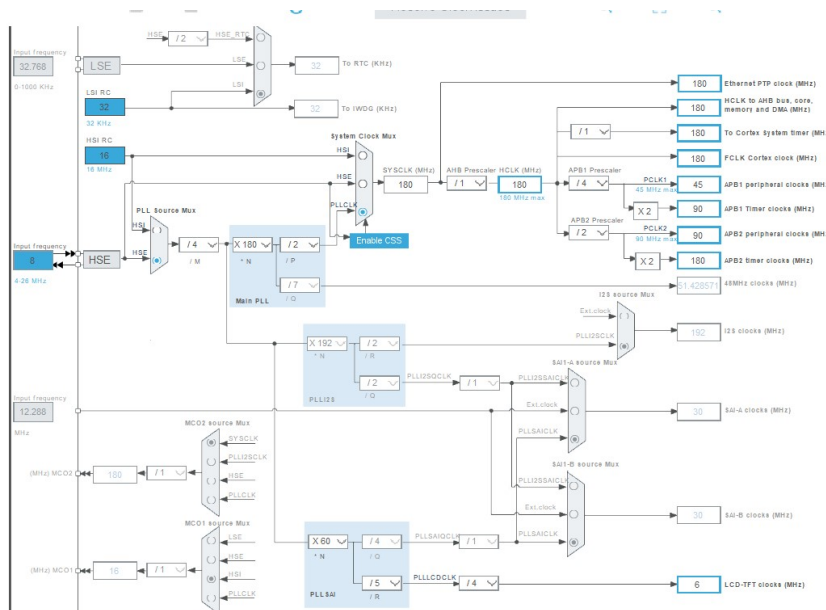
Po zakończeniu gry, rozpoczęcie nowej następuje po ponownym wciśnięciu przycisku Nowa gra. Jest to również możliwe w trakcie trwającej rozgrywki, wtedy całość resetuje się, wraz z zebranymi punktami i straconymi zyciami.

7 Konfiguracja Stm32

Konfiguracja mikrokontrolera odbyła się w programie STM32CUBE_IDE



Rysunek 6: Konfiguracja wyjść mikrokontrolera w programie STM32CubeMX

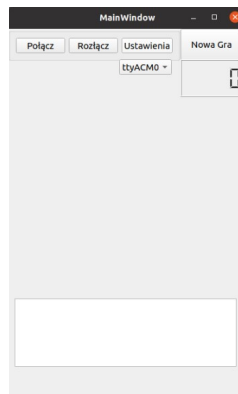


Rysunek 7: Konfiguracja zegarów mikrokontrolera

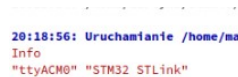
Została ona skonfigurowana tak aby dane z żyroskopu zostały przekierowana do funkcji `printf()` z wykorzystaniem interfejsu USART, tak aby ostatecznie zostały odebrane przez komputer już w aplikacji.

8 Komunikacja

Uruchamiamy aplikację i widzimy jej początkowy wygląd. (Patrz rysunek ?? Do wyboru portu szeregowego, z którym chcemy się połączyć służy przycisk **Ustawienia**. Po jego naciśnięciu w okienku niżej wyświetlają się podstawowe informacje o danym porcie do którego podłączona jest STMka. To znaczy sama nazwa mikrokontrolera oraz nazwa portu lokalizacji. W późniejszych etapach będzie też możliwość zmiany parametrów połączenia, takich jak prędkość transmisji, ilość bitów słowa, parzystość, czy ilość bitów stopu.

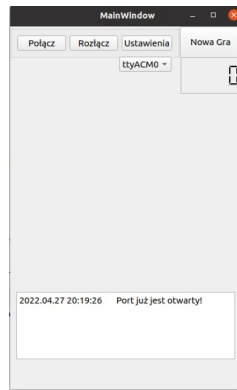


Rysunek 8: Podgląd Aplikacji



Rysunek 9: Podgląd Terminala

W kolejnym etapie następuje połączenie z mikrokontrolerem. W tym momencie otrzymujemy komunikat powodzenia lub porażki operacji. Połączenie pokazane na rysunku 10 i rysunek 11

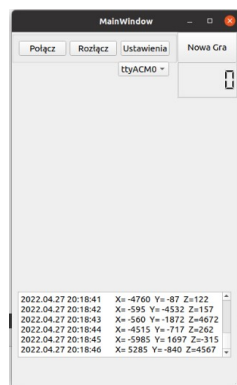


Rysunek 10: Podgląd Aplikacji

```
28:19:22: Uruchamianie /home/martyna/Studia/sem5/ND5/b
info
"ttyACM0" "STK32 STLink"
Połącz
"/dev/ttyACM0"
Otwarcie portu szeregowego się nie powiodło!
```

Rysunek 11: Podgląd Terminala

Na koniec, przy tym etapie pracy, obieramy dokładne dane z żyroskopu w aplikacji (rysunek 12 i rysunek 13).



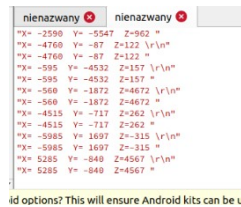
Rysunek 12: Podgląd Aplikacji

9 Ramka danych

Ramka danych wygląda następująco:

- Prosty napis "M" od imienia autora, do ułatwienia znalezienia początku ramki
- Pierwsze 2 liczby, to odczyty z osi X i Y
- Ostatnie znaki to suma kontrolna i znak końca linii

$$M - wartoscX - wartoscY - xx -$$

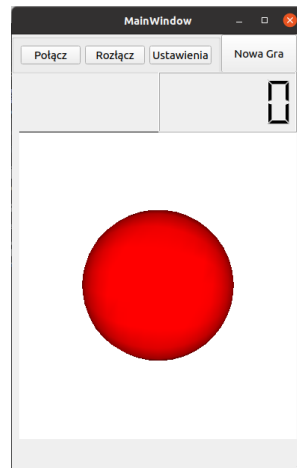


Rysunek 13: Podgląd Terminala

10 Gra

10.1 Początki OpenGL

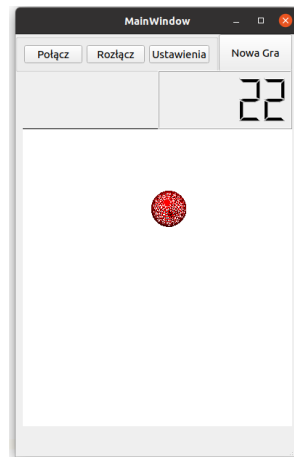
Do projektu została wykorzystana biblioteka OpenGL i na jej bazie został stworzony widget "gra", który po dołączeniu do odpowiedniej klasy umożliwiał rysowanie docelowej kulki w aplikacji w 3D. Na tym etapie głównie stawiano nacisk na zapoznanie się z tą biblioteką. Celem tego etapu było narysowanie prostej kulki, którą byłam w stanie umieścić w wybranym miejscu i nią obracać. Jak widać na Rysunku 14, udało się. Poruszanie obiektem było możliwe w sferze 3D dzięki wykorzystaniu macierzy translacji, a obracanie nim przez dodanie odpowiedniego kąta w macierzy rotacji, który cały czas zwiększano w aplikacji.



Rysunek 14: Podgląd głównego okienka

Następnie rozwinięty został przycisk **Nowa gra**. Po jego naciśnięciu zmieniała się wartość naliczonych punktów oraz wprowadzane były zmiany w rysowanej kulce. Testowane tu były dwa warianty:

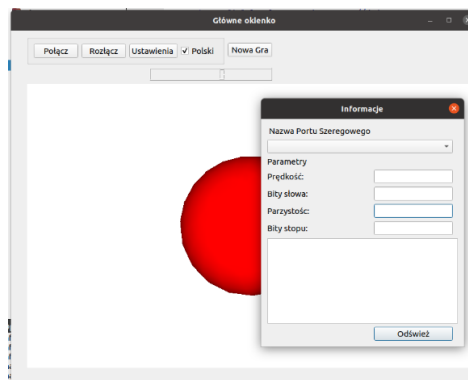
- Po dodaniu sztywnych wartości X i Y, kręciła się ona po wyznaczonym zadanym okręgu, traciła wypełnienie i zmieniała swój rozmiar. (Rysunek 15)
- Po wpisaniu do wartości X i Y wartości z żyroskopu przez fakt że dane te nie zostały jeszcze odpowiednio znormalizowane czasami znikająca z zakresu wyświetlanej powierzchni, co będzie musiało być ulepszone i zmienione w kolejnych etapach projektu. Jednak tak samo traciła ona wypełnienie i zmieniała swój rozmiar.



Rysunek 15: Działanie aplikacji po wciśnięciu przycisku **Nowa Gra**

10.2 Udoskonalenie Ustawień

W następnym etapie projektu stworzono kolejne okienko w aplikacji, które wyskakuje po kliknięciu przycisku **Ustawienia** (Rysunek 16). Wcześniej wszystkie dane dotyczące komunikacji płytka-komputer pokazywały się w głównym oknie. Teraz wypisują się w nowo powstałym okienku **Informacje**. Dodatkowo zostały wypisane informacje dotyczące komunikacji z Portem (ukazują się one po wciśnięciu przycisku **odśwież**). Na dole tego okienka jest pole tekstowe, gdzie wyświetlają się informacje z ramki danych dopiero w momencie połączenia się z STMką, dopiero po wciśnięciu przycisku **Półłącz** w głównym oknie aplikacji. Jest to możliwe przez to, że nowo powstałe okienko **Dialog** jest niezależne i nie blokuje działań w głównym oknie.



Rysunek 16: Podgląd okienka **Informacje**

10.3 Poruszanie się piłeczki przy pomocy płytki

Tu został zastosowany filtr górnoprzepustowy tak aby wyeliminować dryf na tyle ile to możliwe. Wykorzystano do tego `Timer6`, dzięki czemu była możliwość wykonywać działanie filtru w tle, przez przerwania. Kawałek kodu przedstawiającego działanie filtru na Rysunku 17.

W następnych krokach poprzez przekierowanie zoptymalizowano dane odczytywane z żyroskopu tak aby piłeczka rysująca się w widgecie, była widoczna. W tym momencie wartości `X` piłeczki były z zakresu od około 20 do około 80.

```

#include "FirstOrderIIR.h"

int FirstOrderIIR_Init(FirstOrderIIR_t *filter, float alpha, float beta_0, float beta_1)
{
    if(alpha > 1.0f || alpha < 0.0f)
        return -1;

    filter->alpha = alpha;
    filter->beta_0 = beta_0;
    filter->beta_1 = beta_1;
    filter->out = 0.0f;
    filter->previous_in = 0.0f;

    return 0;
}

float FirstOrderIIR_Update(FirstOrderIIR_t *filter, float in)
{
    filter->out = filter->beta_0 + in + filter->beta_1 * filter->previous_in + filter->alpha * filter->out;
    filter->previous_in = in;

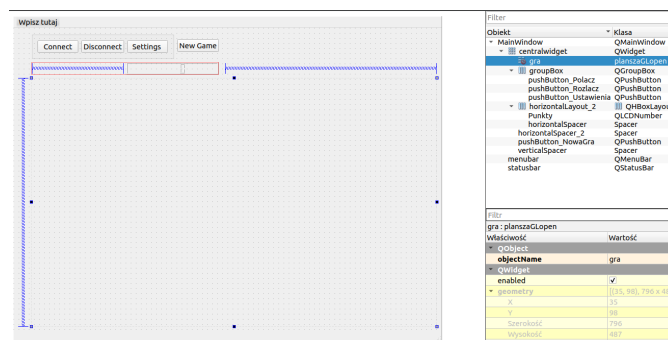
    return filter->out;
}

```

Rysunek 17: Fragment kodu na płytce

10.4 Skalowanie

Poprawiono wszystkie widoki aplikacji dodając do nich sprężyny i ustawiając parametry w taki sposób, aby była możliwość rozszerzenia okienka gry, przy czym aby powiększało się jedynie pole w którym jest wyświetlana gra, a nie wszystkie panele w menu. Nadano przy tym nowe rozmiary startującego okienka tak, aby piłeczka miała jak największy zakres ruchu, zanim zniknie z wyświetlanego pola. Podgląd zmian wykonanych w pliku .ui na Rysunku 18. Poprzez zaznaczone ustawienia, w momencie grania w grę można skalować okienko zmieniając przy tym wielkość kuleczki i adaptując grę do nowego rozmiaru okienka.



Rysunek 18: Podgląd środowiska Qt Creator z wgrany plikiem mainwindow.ui

10.5 Tłumaczenie

Zostało dodane tłumaczenie do odpowiednich komunikatów i przycisków. Przez to że w pliku głównym main.cpp uzależniłam to od ustawień systemu na którym jest uruchamiana gra, zmieniłam wszystkie nazwy na angielskie, aby były zgodne z systemem. Następnie chcąc dodać tłumaczenie ich na język polski, dodałam plik *nienazwany_PL.ts* w Qt Linguist oraz dołączyłam go odpowiednio w pliku .pro. Aby przetłumaczyć dane wyświetlające się w aplikacji w momencie połączenia się aplikacji i płytki dodałam funkcje:

QString nazwa = tr("tekst_do_przetlumaczenia");

do każdego wywołującego się komunikatu. Po wywołaniu lupdate czyli uaktualnieniu tłumaczenia środowisko znalazło mi 15 nowych elementów do przetłumaczenia. Otworzono

plik *nienazwany_PL.ts* w Qt Linguist odpowiednio dodano polskie tłumaczenie odpowiednim elementom (Rysunek 20). Następnie po skompilowaniu tłumaczenia pokazał się komunikat że poprawnie przetłumaczono 15 komunikatów i nie wykryto żadnego nowego, co pokazuje Rysunek 19

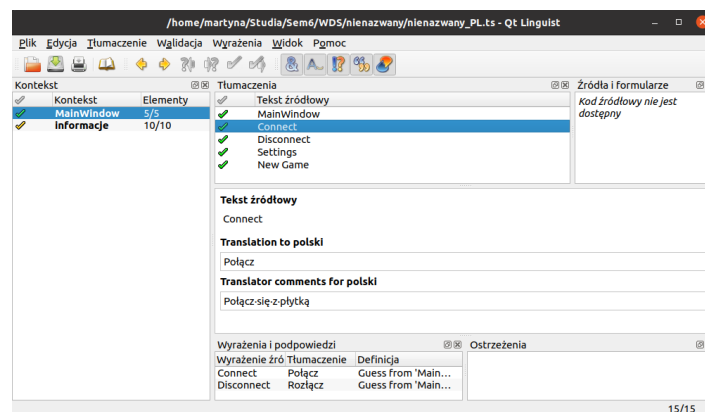
Ostatecznie uzależniono całość od check boxa, gdzie w zależności od tego czy jest zaznaczony język polski, czy nie następuje tłumaczenie całej aplikacji.

```

Komunikaty ogólne  [Filtr]
Info: creating stash file /home/martyna/Studia/Sem6/WDS/nienazwany/.qmake.stash
Updating 'nienazwany_PL.ts'...
Found 15 source text(s) (0 new and 15 already existing)
Zakończono: "/home/martyna/Qt/5.15.2/gcc_64/bin/lupdate"
Starting external tool: "/home/martyna/Qt/5.15.2/gcc_64/bin/lrelease /home/martyna/Studia/Sem6/WDS/nienazwany/nienazwany.pro"
Info: creating stash file /home/martyna/Studia/Sem6/WDS/nienazwany/.qmake.stash
Updating '/home/martyna/Studia/Sem6/WDS/nienazwany/nienazwany_PL.qm'...
Wygenerowano 15 tłumaczeń (przetłumaczonych 15, nieprzetłumaczonych 0)
Zakończono: "/home/martyna/Qt/5.15.2/gcc_64/bin/lrelease"

```

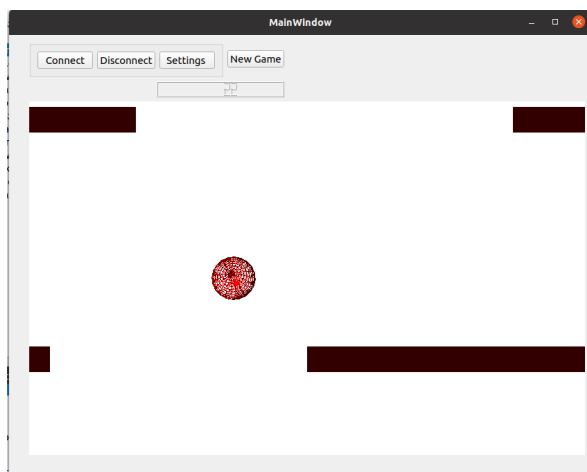
Rysunek 19: Podgląd terminala w środowisku Qt Creator



Rysunek 20: Podgląd środowiska Qt Linguist

10.6 Losowe generowanie przeszkód

Dodawanie przeszkód ostatecznie uzależniono od szerokości i wysokości wyświetlającego się okienka, po to aby przy zwiększaniu lub zmniejszeniu go nie było problemu z wyświetlaniem przeszkód. Na początku stworzono 4 przeszkody które składały się z prawej i lewej strony przeszkody. Do tego stworzono nową klasę `ob`, która tworzyła obiekt odpowiedzialny za każdą nową przeszkodę. Odpowiednie funkcje generowały losowe długości przeszkody z lewej i prawej strony, tak aby przejście gry nie było takie oczywiste. Gdy się udało stworzyć odpowiednie obiekty i wygenerować je w aplikacji, zaczęłam je przesuwając z odpowiednią prędkością do "góry". W kolejnym etapie tak zmodyfikowałam zmienną `gap` zależną od wysokości `Y` każdej z narysowanej przeszkody w taki sposób, aby przeszkody nie nachodziły na siebie i mogły generować się w cały czas, to znaczy w momencie gdy pierwsza przeszkoda dotarła do brzegu okienka to jej wartość `Y` zmieniała się na mniejszą o `gap` od przeszkody nr 4, a jej szerokości były na nowo generowane. Na okienku pokazywane jest tylko od 2 do 3 przeszkód w jednym momencie co daje możliwość reakcji na napływające przeszkody i zostawić sobie zapas na usuwanie i tworzenie nowych. (Przykładowy podgląd na Rysunku 21)



Rysunek 21: Podgląd Gry

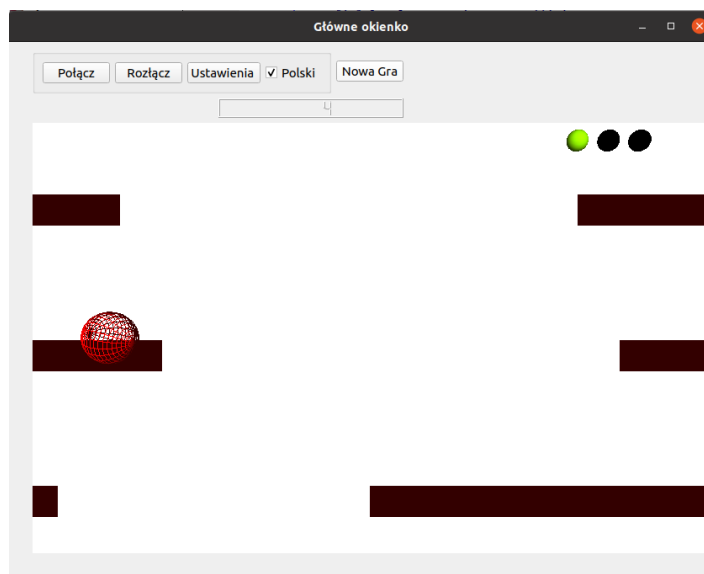
10.7 Kolizja

Kolizja została dodana w prosty sposób. Z racji tego, że przeszkoda podzielona jest na 2 części prawą i lewą, to na początku sprawdzamy odległość wierzchołków od środka piłeczki części przeszkody po lewej, a później po prawej stronie. W momencie gdy dostane długość wektora między punktami porównujemy go do długości promienia piłeczki, przez co jestem w stanie stwierdzić, czy w tym aspekcie dochodzi do kolizji. Na koniec sprawdzam czy piłeczka nie znajduje się w polu obu części przeszkód.

10.8 Życia

Na początku w założeniach projektu ustalone były 3 możliwe życia, przed zakończeniem rozgrywki. Dlatego zostały dodane 3 zielone kulki symbolizujące 3 życia. Zmieniono kod: w trakcie wystąpienia kolizji, zamiast resetu gry, zmniejszała się zmienna odpowiadająca za ilość żyć. Dopiero w tym momencie gdy zrównała się ona z zerem dochodzi do

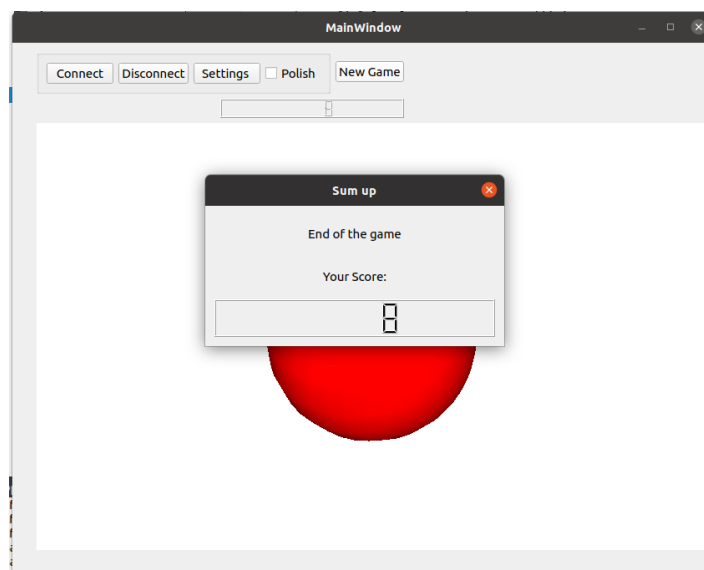
resetu rozgrywki. Wraz z każdą kolizją, kolor kulki życia zmieniał się na czarny, co symbolizuje jego utratę. Ostatecznie po utracie ostatniej szansy i dochodzi do zakończenia rozgrywki, pojawia się okienko **Podsumowania**.



Rysunek 22: Podgląd Gry

10.9 Podsumowanie

Stworzone zostało nowe okienko gdzie ukazuje się informacja o zakończeniu rozgrywki oraz wynik gracza. Zaprezentowane zostało to na Rysunku 23. Powstało ono analogicznie jak okienko informacyjne, również z zachowaniem zasad rozszerzania okienka. Tak samo tekst pokazujący się w okienku **Podsumowanie** został przetłumaczony na język polski.



Rysunek 23: Podgląd okienka podsumowanie

11 Dokumentacja w doxygenie

Dokumentacja dla całego projektu została stworzona w doxygenie. Opisano w niej wszystkie klasy i metody. Dokumentacja do całego projektu zostanie dodana paczce wraz z raportem.

My Project

Collaboration diagram for planszaGLOpen:

Public Member Functions

- planszaGLOpen** (QWidget *parent=0)
- initializeGL ()**
Funkcja inicjalizująca. More...
- paintGL ()**
Funkcja która cały czas generuje obiekty w Widżecie Tu tworzone jest 6 losowych przeszkód „napływających z dołu” tylko i wyłącznie w momencie gdy zmienna pomocnicza jest równa 0. More...
- resizeGL (int w, int h)**
Funkcja odpowiedzialna za skołowanie. More...
- int sprawdz_kolizje (float x, float y, double R, int live)**
Funkcja odpowiedzialna za sprawdzanie kolizji piłka-przeszkody. More...
- void losuj_nowe_przeszkody (int los, GLfloat pom2)**
Funkcja odpowiedzialna za generowanie długości wszystkich nowych przeszkód. More...
- void przenieś_przeszkody (GLfloat pom3, float gap)**
Funkcja odpowiedzialna za przeniesienie przeszkód. More...
- int sprawdz_Score (int wynik)**
Funkcja odpowiedzialna sprawdzenie czy pokonalismy dane przeszkody. More...
- void draw_live (int live, float ang)**
Funkcja odpowiedzialna sprawdzenie czy pokonalismy dane przeszkody. More...

Public Attributes

- double **R**
- float **x**
- float **y**
- float **z**
- bool **wired**
- int **tryb**
- int **wynik**
- float **ang**
- float **zjedz**
- int **punkty**

Rysunek 24: Podgląd fragmentu dokumentacji planszaGLOpen

przeszkoda Class Reference

Zawiera definicję klasy przeszkody. More...

#include <ob.h>

Public Member Functions

- przeszkoda** (float x, float y, float dlugosc, float dlugosc2, float wysokosc, GLfloat Width1, GLfloat Width2, GLfloat Height)
Konstruktor. More...
- przeszkoda ()**
Zwykły konstruktor. More...
- void draw** (GLfloat pom, float zjedz)
Funkcja sprawiająca wygenerowanie się przeszkody w widżecie. More...
- void losuj** (int los, GLfloat pom)
Funkcja równie przeszkody o różnej długości. More...
- przeszkoda & operator=** (const przeszkoda &pom)
Przełączenie operatora =. More...
- int IFCollisionDetect** (float X_ball, float Y_ball, double r)
Funkcja sprawdzająca kolizje przeszkody z napływającą piłeczką Po kolei sprawdzane są wierzchołki części przeszkody z prawej i lewej strony, a następnie w polu figury. More...
- int IFScore** (float Y_ball)
Funkcja do zliczanie punktów za pokonane przeszkody. More...
- void Dodaj_gap** (int ile)
Funkcja do dodawania przerwy na początku gry. More...

Public Attributes

- float **Xpos**
- float **Ypos**
- float **width1_f**
- float **width2_f**
- float **height_f**
- GLfloat **width1**
- GLfloat **width2**
- GLfloat **height**
- float **gap**

Rysunek 25: Podgląd fragmentu dokumentacji przeszkoda

Member Function Documentation

◆ Dodaj_gap()

```
void przeszkoda::Dodaj_gap ( int ile )
```

Funkcja do dodawania przerwy na początku gry.

Parameters

[in] **ile** o ile zwiększyć początkową przerwę

◆ draw()

```
void przeszkoda::draw ( GLfloat pom,  
                        float   zjedz  
                        )
```

Funkcja sprawiająca wygenerowanie się przeszkody w widgecie.

Parameters

[in] **pom** potrzebny do zdefiniowania szerokości okienka

[in] **zjedz** do wiadomości o ile przesunięta powinna być przeszkoda w osi Y (efekt przesuwających się przeszkód)

Rysunek 26: Podgląd fragmentu dokumentacji przeszkoda

12 Test działania aplikacji

Dokonano testu na całej aplikacji:

1. Podłączono płytkę do komputera.
2. Włączono aplikację.
3. Sprawdzono tłumaczenie przez zmianę języka na Polski → poprawnie przetłumaczono aplikację.
4. Sprawdzono informacje o porcie w ustawieniach przy pomocy przycisku **odśwież** - wykryto port i dobrze wypisano informacje.
5. Test komunikacji płytka-komputer przez przycisk **Połącz**(Rysunek 27)→ czytanie wartości z ramki.
6. Test rozpoczęcia gry kliknięcie przycisku **Nowa Gra** (Rysunek 28). → poprawne pojawienie się przeszkód, życ i poruszającej się piłeczki.
7. Test poruszania się piłeczki:
 - Rysunek 29 w lewo.
 - Rysunek 30 w prawo.

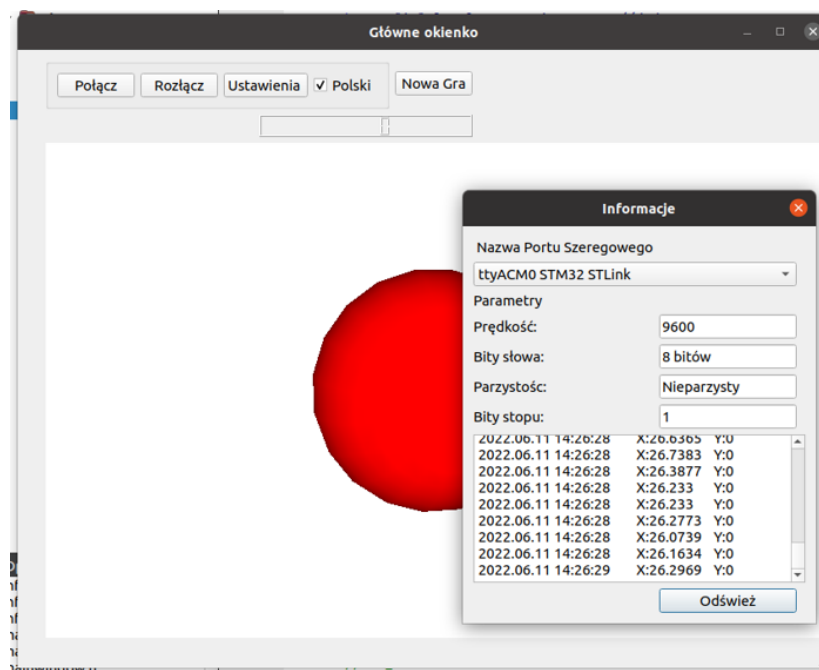
Ogólne poprawne działanie.

8. Test kolizji → poprawne zarejestrowanie wpadnięcia na przeszkodę na Rysunku 31 pokazane przez utratę życia.
9. Test skończenia się życ (Rysunek 32) → wpadnięcie na przeszkodę przy posiadaniu tylko jednego życia.
10. Test okienka **Podsumowanie** → poprawnie pojawienie się okienka oraz wypisanie ilości zdobytych punktów.
11. Test powrotu na język angielski po zakończeniu gry (Rysunek 33) → poprawne pojawienie się komunikatu w języku angielskim i wyniku gry.

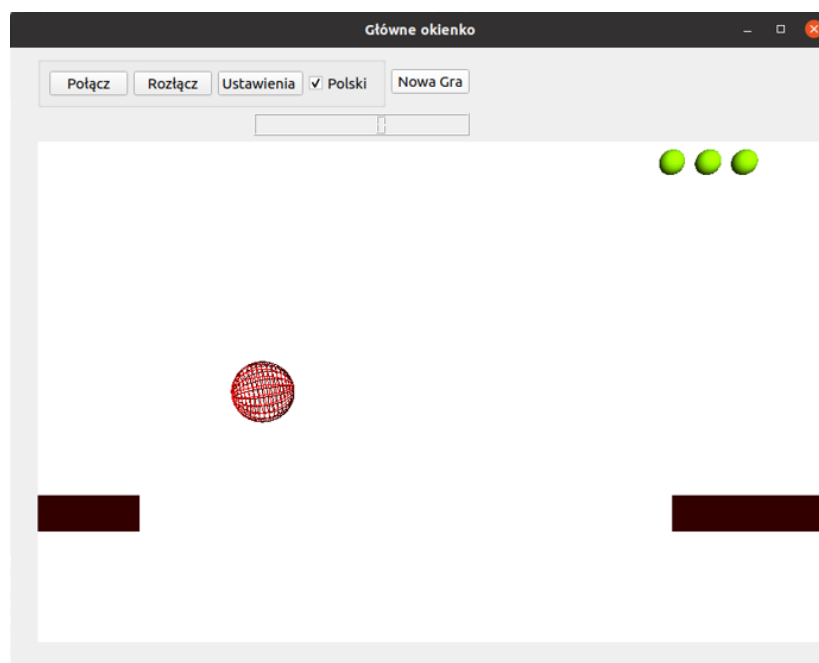
Wszystkie testy zakończyły się powodzeniem, gra jest przejrzysta i działa poprawnie, więc można założyć, że projekt został zakończony sukcesem. :)

Dokumentacja została dodana do ostatecznego raportu.

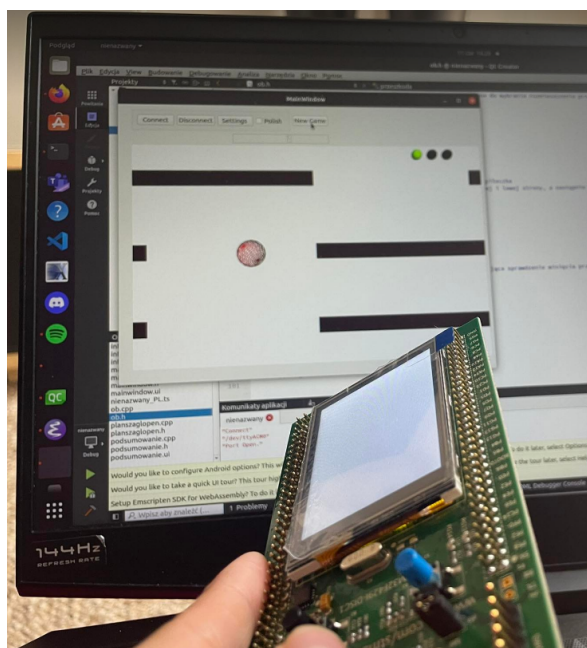
Na koniec warto zauważyć, że finalny efekt aplikacji jest ładnie podobny do początkowych projektów wyglądu aplikacja, a wszystkie jego założenia zostały wykonane.



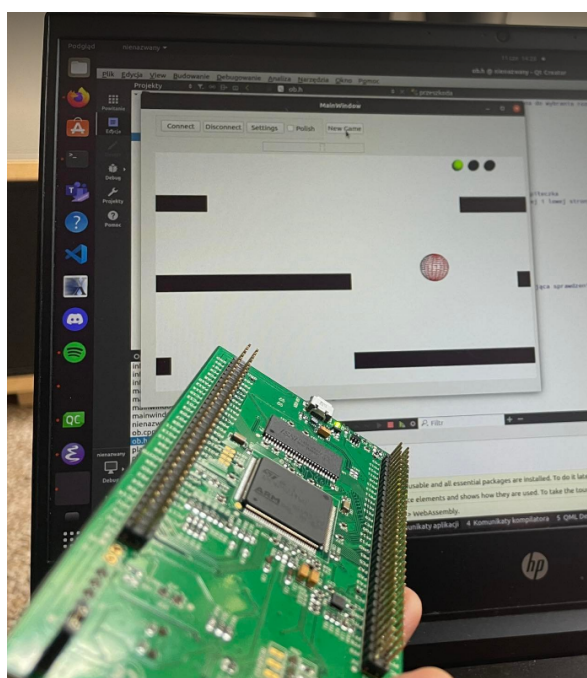
Rysunek 27: Pokazanie testu obierania danych z płytki



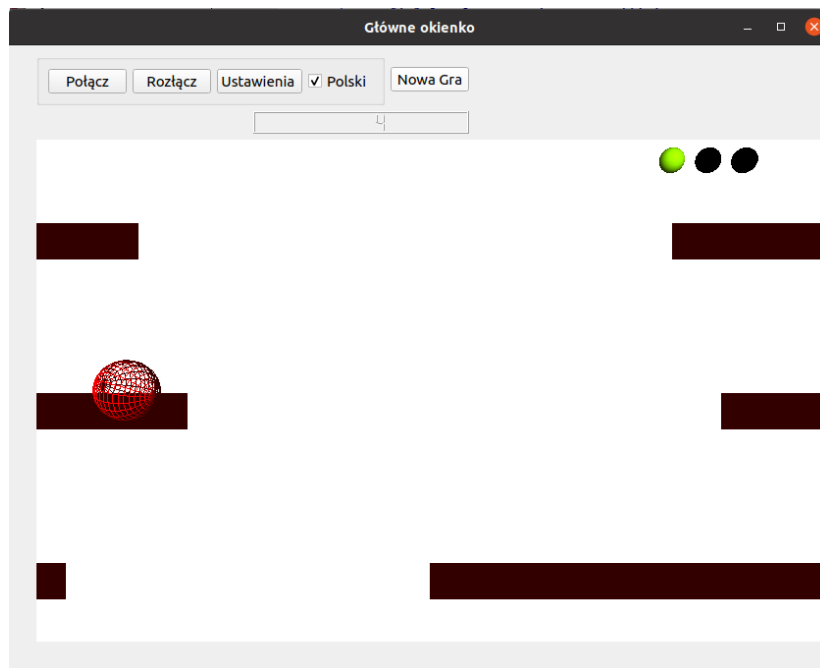
Rysunek 28: Pokazanie testu rozpoczęcia gry



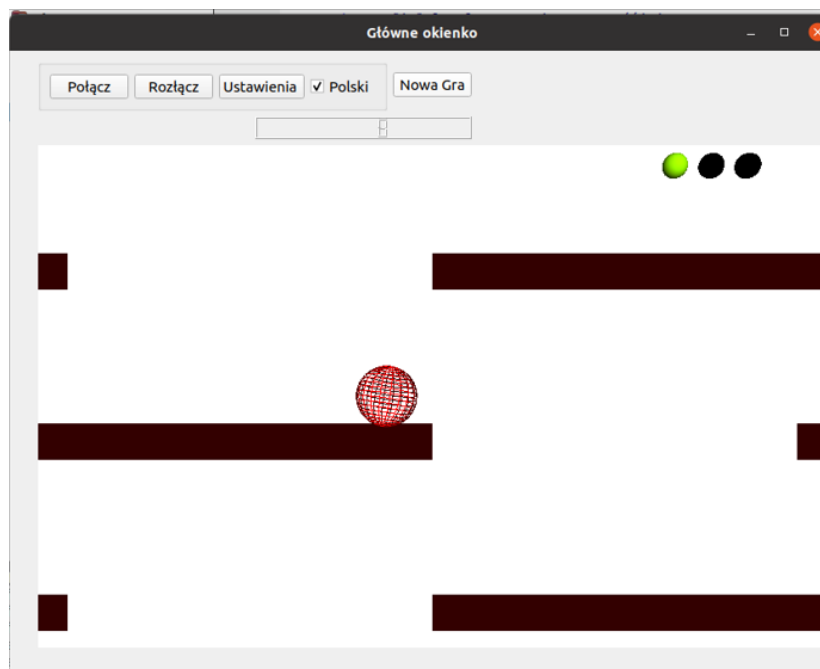
Rysunek 29: Pokazanie testu przesuwania pileczki w lewo za pomocą płytki



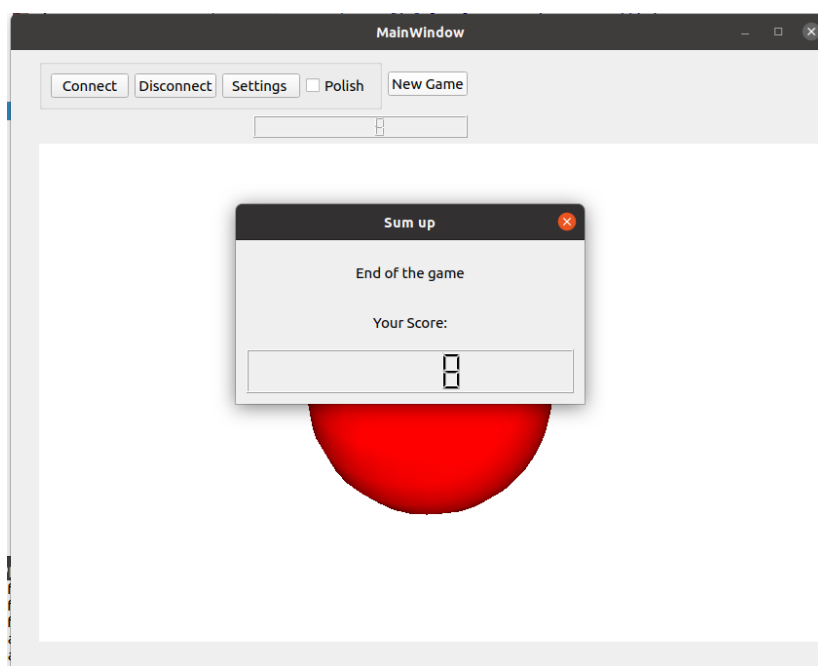
Rysunek 30: Pokazanie testu przesuwania pileczki w prawo za pomocą płytki



Rysunek 31: Pokazanie testu kolizji i utraty życia



Rysunek 32: Pokazanie gry chwili przed końcem gry



Rysunek 33: Test podsumowania gry