

Wymagania wstępne i Instrukcja użytkownika gry w statki

Adam Młyńczak i Michał Partyka

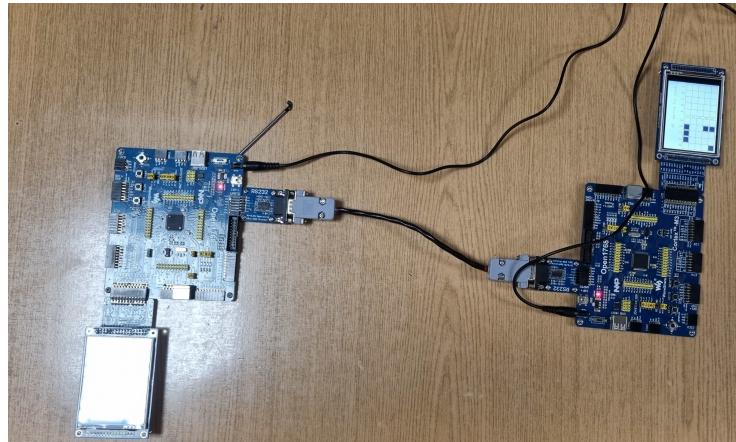
25 stycznia 2024

Wymagania wstępne

Projekt zakładał stworzenie gry w statki (ew. Okręty), wraz z komunikacją pomiędzy dwoma zestawami dla dwóch graczy.

Do zrealizowania projektu konieczne są następujące elementy elektroniczne:

- Dwie płytki Open1768,
- Dwa wyświetlacze ILI9325 - wyświetlanie planszy gry,
- Moduł UART RS232, podłączony pod UART0 – komunikacja pomiędzy płytami.



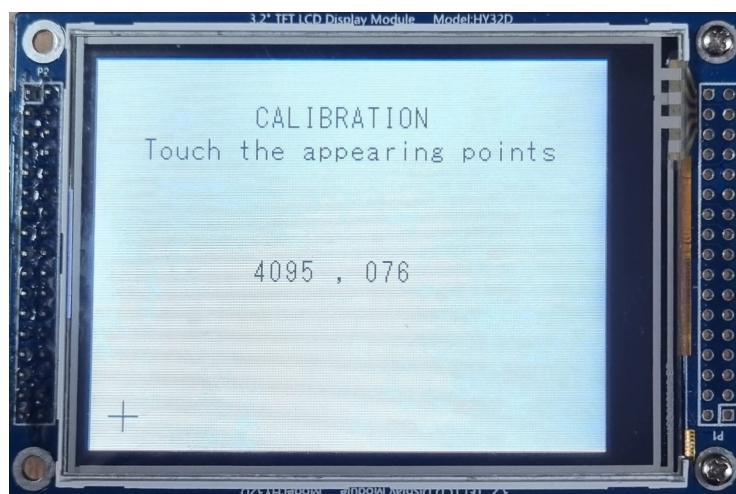
Rysunek 1:

Założenia projektu zostały zrealizowane.

Instrukcja użytkownika

2.1. Kalibracja

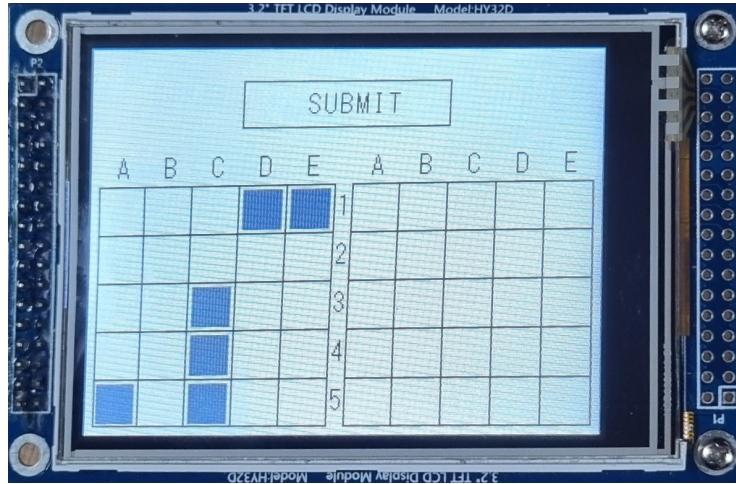
Przy każdorazowym włączeniu/restarcie płytka zostaniemy poproszeni o kliknięcie za pomocą rysika pojawiających się krzyżyków, w celu kalibracji. Bedziemy musieli zrobić to 3 razy, w 3 miejscach (prawy dolny róg, prawy górny róg, lewy dolny róg). Na środku ekranu znajdują się aktualne współrzędne dla kliknięcia, w celu ułatwienia stwierdzenia czy ekran przechwycił nasze dotknięcie.



Rysunek 2:

2.2. Ustawienie statków

W kolejnym kroku naszym zdaniem jest zatwierdzenie ustawienia statków. Klikając key2 na płytce zostanie wygenerowana nowa mapa z nowym rozmieszczeniem statków na planszy. Aby zatwierdzić wybór, należy dotknąć przycisk SUBMIT na ekranie.



Rysunek 3:

2.3. Rozpoczecie rozgrywki

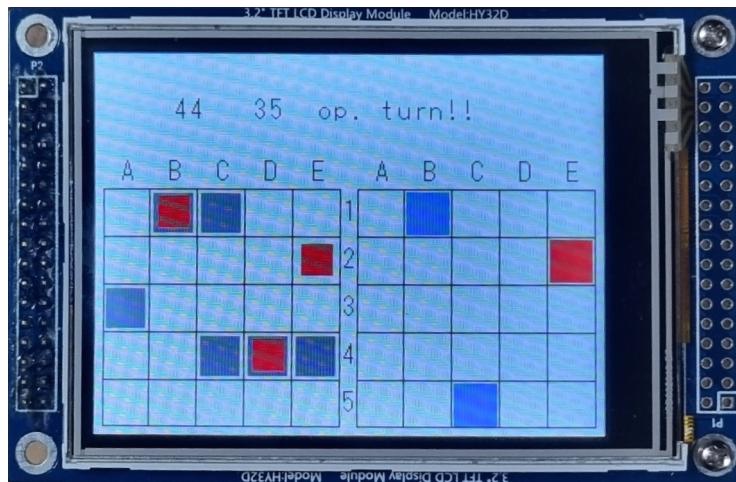
Po zatwierdzeniu ustawienia statków przycisk **SUBMIT** zamieni się na przycisk **READY**, który w momencie, w którym oboje z graczami będą gotowi, gracz rozpoczynający (ustalenie tego należy do graczy) musi dotknąć.

2.4. Rozgrywka

W trakcie wzajemnego strzelania przez graczy na ekranie pojawiają się następujące instrukcje:

- **wait** – pojawia się, gdy zawodnik strzelający czeka na pozwolenie od drugiej płytki na wykonanie strzału (czekamy, aż płytką przeciwnika zacznie odbierać informacje o współrzędnych),
- **op. turn!** - jest na ekranie, gdy nasz przeciwnik wykonuje swój strzał,
- **shoot!** - nakaz strzelania dla gracza,
- **waiting for check** – po naszym strzałe czekamy na informacje od drugiego gracza o tym, czy strzał był trafiony czy nie,
- **checking...** - nasza płytka sprawdza, gdzie przeciwnik strzelił i czy trafił,
- **ship hit** – informacja o trafionym statku,
- **miss! / missed!** - informacje o nietrafionym strzałe.

Na koniec jednej tury program dodatkowo sprawdza informacje, czy należy zmienić gracza strzelajacego (w przypadku nietrafionego strzału) i dodatkowo czy należy przerwać rozgrywke - czy któryś z zawodników już trafił wszystkie statki przeciwnika.



Rysunek 4:

2.5. Zakończenie rozgrywki

Na koniec gry w zależności od tego, czy zawodnik wygrał czy przegrał, na ekranie zostanie wypisany napis **YOU WIN!** lub **YOU LOST** :(.

Opis kodu

3.1. main.c

Plik **main.c** można podzielić na trzy sekcje. W pierwszej z nich znajdują się funkcje inicjujące, potrzebne do działania wyświetlacza oraz ekranu dotykowego. W drugiej części znajdują się operacje przygotowujące gracza pod rozgrywkę. Są to kalibracja ekranu, inicjalizacja wszystkich zmiennych dla **struct Player** oraz ustawienie statków. W ostatniej sekcji występuje już sam algorytm gry. W nieskończonej petli (która może zostać przerwana jeżeli zmienna **gameOn** będzie równa *false*) znajduje się funkcja **shoot** obsługująca strzelanie pomiędzy graczami. Po przerwaniu petli program zakończy rozgrywkę informując zawodników o jej wyniku.

3.2. delay.h

W pliku **delay.h** znajdują się funkcje **void SysTickHandler()** oraz **void delay(int t)** obsługujące opóźnienie, które w niektórych etapach naszego kodu są potrzebne. Wykorzystujemy tutaj SysTick timer, który przy każdym przerwaniu SysTick zwiększa wartość zmiennej **msTicks** o jeden. W funkcji **delay** w momencie, w którym wartość tej zmiennej dojdzie do wyznaczonego **t**, program dalej wykonuje swoje działania.

3.3. usart.h

W tej bibliotece ustalamy używanie **Driver_USART0** jako naszej komunikacji. Dodatkowo znajdują się tu funkcje **USART_Init** odpowiedzialne za inicjalizację i ustalenie potrzebnych do działania parametrów oraz **USART_DeInit** dzięki której wyłączamy komunikacje tym modułem.

3.4. lcd.h/c

W tych dwóch plikach znajdują się funkcje do rysowania prostych kształtów na wyświetlaczu LCD. W funkcji **drawRectangle**, która przyjmuje współdzielne (x oraz y) dwóch przeciwnieległych wierzchołków prostokąta oraz kolor tegoż prostokąta, rysowany jest rzeczywisty prostokąt za pomocą funkcji **lcdWriteReg** z biblioteki **Open1768_LCD.h** do obsługi wyświetlacza. Następna funkcja jest **writeSign**, w której korzystając z biblioteki **asciiLib.h** oraz wcześniej już przywołanej biblioteki 'pisane sa' litery. Funkcja ta przyjmuje współdzielne początku litery oraz jej kolor. Funkcja, która na niej bazuje jest **writeString**, w której wypisujemy na ekran ciąg znaków. Ostatnia funkcja zadeklarowana w tej bibliotece jest **clearText**, która wyczyszcza pole pod nowy napis.

3.5 game.h/c

W tej bibliotece znajdują się wszystkie funkcje odpowiedzialne za działanie gry. Jako zmienne globalne są tutaj zainicjalizowane **static bool startFlag**, dzięki której płytka wie, czy w tej turze ma strzelać czy czekać na odpowiedź z drugiej płytki, oraz **static ARM_DRIVER_USART * USARTdrv** - wskaźnik na używany przez nas moduł komunikacji.

Do obsługi i informacji dla gracza o swojej planszy czy też planszy przeciwnika zdefiniowane zostają tutaj dwa obiekty typu *struct*: **Board**, z polami **int squares[5][5]** - przedstawienie planszy jako tablicy dwuwymiarowej, **int max_hits** - maksymalna liczba trafień dla planszy oraz **int hits** - liczba trafionych pól; **Player**, z dwoma polami typu **Board** - boardPlayer (własna plansza) oraz boardOpponent (plansza przeciwnika) oraz z polem typu **bool** o nazwie **win** - domyślnie jest to *false*, w przypadku wygranej jest to zmieniane.

3.5.1. void calibrate(float *arr)

Funkcja odpowiedzialna za kalibrację. Rysuje ona po kolei krzyżyki w rogach ekranu, zczytuje współrzędne ekranu dotykowego i przelicza proporcje (funkcje liniowa) pomiędzy ekranem dotykowym a wyświetlaczem. Parametry funkcji przypisuje do przekazywanej tablicy `*arr` i przypisuje w kolejności: a1, b1, a2, b2 (pionowe oraz poziome przeliczanie).

3.5.2. int calc(int xy, float a, float b)

Funkcja przeliczająca ze współrzędnych ekranu dotykowego na wyświetlacz. Przyjmuje ona współrzędną (x lub y) ekranu dotykowego i za pomocą wcześniejszej kalibracji współczynniki *a* oraz *b* funkcji liniowej.

3.5.3. void ustawStatkiRand(Player *p)

Funkcja odpowiedzialna za ustawienie statków na planszy dla gracza (**Player *p**). Ustawiane są w kolejności: trzymasztowiec, dwumasztowiec oraz jednomasztowiec, gdzie współrzędne dla pierwszego pola statku są losowane za pomocą funkcji `rand()` z biblioteki `stdlib.h`. Przed ustawieniem każdego ze statków ustalana jest jego orientacja (pionowa lub pozioma) oraz sprawdzana jest legalność jego ustawienia za pomocą przywołanej już funkcji `isLegal`.

3.5.4. bool isLegal(int x, int y, Board *b)

Jest to funkcja potrzebna do sprawdzenia, w polu o współrzędnych (x, y) na planszy można postawić statek (przy dwu- oraz trzymasztowcach sprawdzane są odpowiednio dwa i trzy pola). Przy możliwości postawienia w tym miejscu okretu funkcja zwraca *true*, w innym wypadku *false*.

3.5.5. void drawBoard(Board *board)

Rysowanie planszy dla gracza. Funkcja przyjmuje *board* i rysuje mając na uwadze wartości elementów tablicy dwuwymiarowej reprezentującej te plansze.

3.5.6. void start(float *tab)

Odpowiedzialna za początek rozgrywki, gracze ustalają wtedy kolejność gry (za pomocą dotknięcia przycisku *READY*). Funkcja przyjmuje tablice z współczynnikami funkcji przeliczenia współrzędnych ekranu dotykowego na współrzędne wyświetlacza.

3.5.7. bool shoot(float *tab, Player *player)

Obsługa wzajemnego strzelania graczy. Najpierw płytka "odbierająca" wysyła informacje o tym, że czeka na strzał. Następnie zawodnik strzelający musi dotknąć pola, w które chce oddać swój strzał i wysyła informacje (wynik z `przelicz(int x, int y)`) do drugiego gracza, który już wcześniej czeka na te liczby.

Po tym następuje sprawdzenie czy jest to pole trafione czy chybione i zostaje wysłane przez płytke informacja zwrotna z ta wiadomością. Po przeprocesowaniu przez obie płytki wszystkich potrzebnych informacji, w przypadku nietrafionego strzału gracze zamieniają się tura (zmiana **startFlag** z *true* na *false* lub odwrotnie), a następnie sprawdzane jest, czy gra nie powinna się już zakończyć z funkcją **checkWin**. Obsługa przerwania wykonywania funkcji **shoot** znajduje się w pliku **main.c**.

3.5.8. int przelicz(int x, int y)

W tej funkcji przyjmowane są dwa parametry: współrzędne dotkniecia na ekranie lcd (przeliczone już z współrzędnych ekranu dotykowego) i przeliczane są one na współrzędne na planszy 5x5. Zwracany int jest w postaci liczby dwucyfrowej, gdzie liczba dziesiątek to kolumny (współrzędne 1-5), a liczba jedności to wiersze (współrzędne A-E).

3.5.9. void drawHit(int xy)

Rysowanie strzałów przeciwnika na naszej planszy - ze współrzędnych w postaci liczby dwucyfrowej.

3.5.10. void drawX(int xy)

W przypadku trafionego strzału, funkcja ta zaznaczy to na planszy mając podane współrzędne zwrócone przez funkcję **przelicz**.

3.5.11. void drawVoid(int xy)

Ta funkcja robi to samo co **drawX**, tylko dla strzału nietrafionego.

3.5.12. bool checkWin(Player *player)

Sprawdzenie, czy dany gracz wygrał/przegrał. Dla obu plansz przechowywanych dla gracza sprawdzana jest relacja pomiędzy *max_hits* oraz *hits*. Przy ich równości jest zwracana odpowiednia wartość, jak również jest zmieniana wartość dla *win* dla zawodnika.

3.5.13. void end(Player *player)

Zakończenie gry, funkcja sprawdza wartość pola *win* i wypisuje informacje o tym na ekran.