Asteroid Avoider

Introducere

Proiectul reprezintă un joc arcade în care utilizatorul pilotează o navă spațială cu ajutorul unui joystick, având ca scop evitarea asteroizilor. Nivelul de dificultate crește dinamic în timp, iar numărul de vieți rămase este indicat vizual printr-un LED RGB (verde, galben sau roșu intermitent). Jucătorul poate activa un scut temporar cu utilizări limitate prin apăsarea joystick-ului, iar un buzzer pasiv oferă feedback auditiv, emițând semnale sonore la coliziunea cu un asteroid și în momentul de game over. La finalizarea sesiunii, se afișează timpul petrecut în joc și cel mai bun timp obținut, fiind actualizat când este cazul. Scopul jocului pentru utilizator este testarea reflexelor și obținerea unui timp de supraviețuire cât mai lung pentru a stabili un nou record.

Descriere generală

Hardware Design

Listă de componente:

Componentă	Descriere
ATmega328P Xplained Mini	Microcontrollerul principal care se ocupă de periferice și logica jocului. Cel mai bun timp va fi stocat în memoria EEPROM a acestuia.
Display TFT LCD 3.2 (Controller ILI9341)	Ecranul unde se afișează jocul prin protocolul SPI
Joystick HW-504	Modul principal de controlare a navei spațiale
LED RGB	Folosit pentru status de viață: 3 vieți - verde, 2 vieți - galben, 1 viață - roșu intermitent
Buzzer pasiv	Redă sunet în momentul coliziunii navei cu un asteroid sau la game over
Rezistențe	Folosite pentru limitarea curentului prin LED și pentru divizori de tensiune

Schema electrică:

Pinii nefolosiți sunt marcați cu un X.

Pentru comunicarea prin protocolul SPI cu ecranul LCD, se folosesc următorii pini:

- **D8 (RESET):** responsabil pentru a reporni și inițializa ecranul
- **D9 (DC):** folosit pentru a-i transmite ecranului dacă informația este o comandă sau un pixel care trebuie desenat pe ecran
- **D10 (CS):** selectează ecranul pentru a comunica cu acesta prin setarea valorii la LOW
- **D11 (MOSI):** pinul care transferă date de la master (ATmega328P) către slave (ecran)
- **D13 (SCK):** semnalul de ceas care dă ritmul exact pentru ca transferul de biți să fie perfect sincronizat

Deoarece nu se vor citi niciodată date de la ecran, nu se folosește pinul **D12 (MISO)**. Având în vedere că ecranul efectuează comunicarea SPI la 3.3V, iar pinii au un output de 5V, se folosește un divizor de tensiune pentru fiecare pin, utilizând rezistențe de 1K și 2K pentru a coborî tensiunea de la 5V la 3.33V. De asemenea, se folosește pinul **3V3 (LED)** al plăcii pentru a alimenta backlight-ul ecranului.

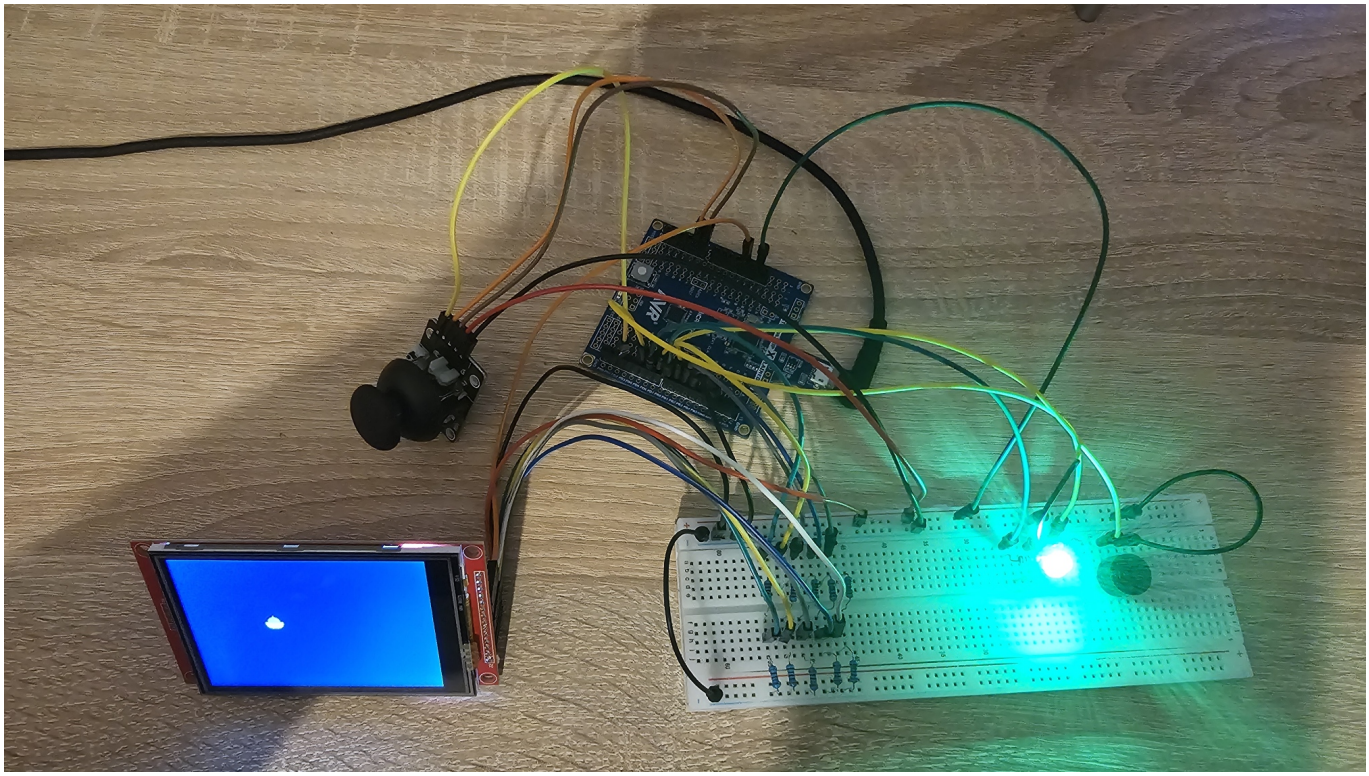
Pentru joystick, se folosesc următorii pini:

- **A0 (VRX):** citește valoarea analogică pentru axa X
- **A1 (VRY):** citește valoarea analogică pentru axa Y
- **D2 (SW):** citește starea butonului integrat în manetă; acest pin suportă întreruperi externe (INT0), care va fi configurat să genereze un interrupt request doar pe falling edge

Pentru LED-ul RGB, se folosesc pinii digitali **D5 (R)** și **D6 (G)** pentru a controla culoarea roșie, respectiv culoarea verde, folosind rezistențe de 330Ω pentru a limita curentul prin acesta. Deoarece avem nevoie numai de culorile roșu, verde și galben (roșu + verde), nu este necesară culoarea albastră, pinul aferent acesteia rămânând neconectat. Pentru situația în care jucătorul mai are o singură viață, se va folosi un timer care va alterna între HIGH și LOW pentru culoarea roșie.

Buzzerul pasiv va fi controlat de pinul digital **D7 (BUZZER)**, alternând între stările HIGH și LOW folosind timere pentru a reda sunetul.

Conectare componente + testare ecran LCD și LED RGB:



Nava spațială este afișată corect pe ecranul LCD, ceea ce validează comunicarea prin protocolul SPI. LED-ul RGB este funcțional, fiind activat pe culoarea verde, demonstrând configurarea corectă a pinilor de ieșire.

Software Design

Mediu de dezvoltare: Visual Studio Code + PlatformIO

Librăria folosită: https://github.com/lexus2k/ssd1306, care are suport și pentru ILI9341, folosit pentru desenarea pe ecran și funcționalitățile protocolului SPI. S-a ales această librărie, deoarece se poate folosi fără ecosistemul Arduino, folosește cât mai puține resurse și are implementări performante, fiind important pentru a actualiza ecranul cât mai repede posibil pentru un framerate cât mai bun.

Interacțiunea cu componentele hardware:

- **Joystick:** Se inițializează ADC-ul pentru cei doi pini analogici, folosind VCC ca referință și prescaler de 128. La citirea valorii ADC, pentru a preveni zgomotul electric și pentru ca nava spațială să nu se miște de una singură, s-a impus o toleranță de 50 astfel încât pentru orice valoare care se află în acea toleranță față de valoarea de mijloc (512) îi va corespunde viteza 0 pentru axa respectivă. De asemenea, pentru butonul joystick-ului, se inițializează pinul respectiv ca input și se activează întreruperea lui INT0 pe front descrescător. La fiecare apăsare, se va trata rutina lui INT0, care va seta o variabilă cu true, aceasta reprezentând că o cerere de apăsare a fost efectuată. Există o funcție care actualizează statusul apăsării joystick-ului, verificând dacă o cerere a existat. De asemenea, există funcții pentru a verifica dacă butonul a fost apăsat și pentru a dezactiva sau activa funcționalitatea butonului.
- **Systime:** Se folosește timer-ul 0 astfel încât la fiecare milisecundă, să se activeze o întrerupere de

comparație care numără câte milisecunde au trecut de la pornirea plăcii. Numărul de milisecunde este folosit pentru calcularea valorii de delta time pentru mișcarea jucătorului și a asteroizilor independent de framerate și pentru timere software.

- **RGB LED:** Se inițializează pinii led-ului ca output și timer-ul 1, dar oprit. Timer-ul acestuia este pornit în cazul în care se cere roșu intermitent. Întreruperea de comparație a acestuia va alterna starea logică a pinului corespunzător culorii roșii la fiecare 0.25 secunde. Stările valide ale led-ului sunt următoarele: GREEN, YELLOW, BLINKING RED, NONE. Starea NONE este folosită atunci când jucătorul nu mai are viață și led-ul este stins.
- **Buzzer:** Pentru a reda sunet, se folosește timer-ul 2, care va alterna rapid între HIGH și LOW pentru a reda frecvența specificată. De asemenea, avem funcționalitatea de a seta frecvența dorită și dacă să activăm sau nu buzzer-ul. Numărul de milisecunde de la systime este folosit pentru a verifica dacă mai trebuie să redea sunet în continuare sau să schimbe nota în cazul în care avem mai mult de o notă de redat, cum ar fi atunci când jucătorul a rămas fără vieți.

Componente software:

- **Player:** Jucătorul principal, adică nava spațială. El se poate deplasa în funcție de joystick. De asemenea, verifică statusul de invincibil, atunci când a fost lovit de un asteroid, acesta durând doar 1 secundă și statusul de scut, dacă poate să-l activeze și dacă îl are activat sau nu, scutul având o viață de doar 3 secunde. Scutul se activează prin apăsarea joystick-ului. Acesta mai verifică și coliziunile cu marginile hărții sau cu asteroizii. Dacă nu are activat scutul, asteroidul aplică un impuls navei și aceasta continuă să se miște din inerție, viteza de inerție scăzând în timp. Nava devine invincibilă și i se scade o viață, iar LED-ul RGB își schimbă starea. În caz contrar, asteroidul ricoșează din scut dacă se îndreaptă spre jucător sau jucătorul de oprește in dreptul asteroidului dacă asteroidul se îndepărtează de acesta. În modul invincibil, jucătorul va alterna între a apărea pe ecran și a nu aparea, pentru a observa vizual că este invincibil.
- **Asteroid:** Asteroidul este instanțiat inițial în afara ecranului, cât să atingă cercul de marginea hărții. Acesta se mișcă în linie dreaptă, viteza rămânând constantă, dacă nu este perturbat de un alt asteroid. Acesta se elimină în cazul în care ajunge din nou offscreen, după inițializare. În cazul în care există o coliziune între doi asteroizi, prima dată se modifică pozițiile înapoi în timp astfel încât cele două cercuri să fie tangente. Apoi, se aplică principiile coliziunii elastice pentru a modifica vitezele celor doi asteroizi. De asemenea, deoarece nu vrem să avem cercuri prea mari care ar suprasolicita comunicarea SPI, dar totuși să avem varietate, s-a decis ca fiecare asteroid să aibă una dintre cele trei raze: 7 pentru asteroid mic, 11 pentru asteroid mediu și 15 pentru asteroid mare. Încă un motiv pentru care s-a făcut acest lucru este menționat la partea de optimizări. Un edge case important de menționat este că atunci când se instanțiază un asteroid, este posibil ca acel asteroid care este offscreen să se ciocnească de un asteroid aproape de acesta și să devieze și să rămână offscreen mereu. Pentru a rezolva acest lucru, se pune un timer software scurt ca după primele 80 de milisecunde, dacă încă este offscreen, să se elimine.
- **Asteroid spawner:** Entitatea care se ocupă de a instanția asteroizi offscreen într-un mod random. Ea menține în memorie numărul mediu și deviația in milisecunde de așteptare pentru a lansa un asteroid, precum și viteza minimă și maximă pe care o poate avea. Are implementat și un sistem de ponderi pentru alegerea tipului de rază. Cu cât ponderea este mai mare față de restul, cu atât cresc șansele ca acea rază să fie selectată. De asemenea, se aleg valorile vitezei astfel încât tot cercul, la un moment dat, să fie vizibil pe ecran, neavând situații de a arunca un asteroid și mai adânc offscreen sau să apară foarte puțin într-un colț. La fiecare 10 secunde, se actualizează variabilele astfel încât să se spawneze mai des, asteroizi să fie mai rapid și să avem asteroizi cu raze mai mari.

Deoarece mulți asteroizi mari (cei cu raza 15) cu viteză mare ar rezulta în multe actualizări ale ecranului, pe măsură ce trece timpul, șansele să lanseze asteroizi mari scad. Pentru asteroizii mici, șansele scad mai repede. Astfel, jocul favorizează în late-game asteroizii de dimensiune medie. Numărul de actualizări de făcut a fost limitat la 20.

- **UI bar manager:** Este un manager care actualizează datele din bara de UI. În stânga, se afișează numărul de vieți pe care jucătorul le mai are. În dreapta, sunt afișate câte scuturi mai pot fi folosite, fiecare pătrățel reprezentând un scut.

Bucla de joc:

Prima dată se actualizează starea de apăsare a joystick-ului. După aceea, se verifică buzzer-ul, pentru a-l opri după un timp scurt dacă jucătorul a luat damage, sau pentru a alterna notele dacă rulează melodia de game over de 3 note. Dacă jucătorul a rămas fără vieți, starea jocului devine game over și se desenează ecranul de final. Aici se calculează timpul supraviețuit și se verifică în EEPROM dacă s-a bătut recordul de timp supraviețuit anterior, actualizându-se dacă este cazul. De asemenea, se pornește și muzica de final. În acest caz, se așteaptă apăsarea joystick-ului pentru a reseta jocul. Dacă jocul rulează în continuare, se calculează delta time (diferența dintre milisecunda actuală și milisecunda de la frame-ul anterior convertită în secunde). Se actualizează spawner-ul de asteroizi, iar apoi se actualizează poziția jucătorului, invincibilitatea, scutul și coliziunile acestuia cu marginile hărții, iar la final și poziția asteroizilor. Apoi, se verifică coliziunile între asteroizi, iar la final coliziunile asteroizilor cu jucătorul. Dacă jucătorul a fost lovit și mai are vieți, se pornește buzzer-ul pentru a reda sunetul de hit. La finalul buclei se face desenarea. Se desenează jucătorul, apoi asteroizii. În cazul în care un asteroid poate fi distrus pentru că a ieșit din ecran, el este eliminat din memorie prin copierea ultimului asteroid din vectorul de asteroizi în locul lui. În final, se actualizează datele din bara de UI și se desenează pe ecran.

Optimizări:

- Pe departe cea mai bună optimizare este **să comunicăm mai puțin cu ecranul prin SPI**. Astfel, nu redesenăm de fiecare dată toate elementele de pe ecran pentru că ar fi prea lent, ci **vom desena doar ce se schimbă de la un frame la altul**. Pentru jucător, în memoria flash avem bitmap-ul navei spațiale, având și culoarea fundalului incorporat în aceasta. Se desenează cel mult 2 dreptunghiuri, având culoarea fundalului, unu stânga/dreapta, altul sus/jos, peste părțile necomune dintre poziția veche și cea nouă. Apoi, desenăm din nou nava spațiala folosind bitmap-ul. Deoarece bitmap-ul are și culoarea fundalului, se vor actualiza și pixelii care aparțineau navei vechi. De asemenea, se elimină și efectul de flickering, deoarece un pixel nu își va schimba culoarea mai mult de o dată. Pentru asteroizi, fiind cercuri, parcurgem pe diametrul cercului aflat la poziția veche și verificăm dacă trebuie să ștergem pe verticală elementele necomune dintre cercul vechi și cercul nou. Trebuie desenate cel mult 2 linii verticale pentru a le elimina. Apoi, deoarece librăria nu are suport pentru desenarea de cercuri, se desenează cercul nou în felul următor: parcurgem fiecare punct de pe raza cercului și, profitând de simetrie verticală și orizontală, desenăm două linii verticale. Și în acest caz, eliminăm flickering-ul, din aceleași motive menționate anterior. Iar pentru bara de UI, actualizăm textul despre vieți în cazul în care se schimbă numărul de vieți al jucătorului și eliminăm câte un scut când se folosește această abilitate.
- O altă optimizare are legătură cu desenarea cercurilor, deoarece ea trebuie făcută la fiecare frame. Deoarece, pentru aflarea înălțimii cercului la un punct de pe diametru implică utilizarea radicalului, ceea ce ar fi lent, se folosesc niște **look-up table-uri pentru cele 3 raze suportate**. Fiecare reprezintă cea mai apropiată valoare întreagă de valoarea lui $\sqrt{R^2 - x^2}$, unde R reprezintă raza, iar x reprezintă distanța de la centru la un punct pe diametrul acestuia. Aceste tabele sunt stocate în memoria flash.

Rezultate Obţinute

Link Youtube: Asteroid Avoider - Proiect PM 2026

Se poate observa în videoclip că, într-o sesiune de joc, toate componentele hardware și software menționate funcționează.

Concluzii

Realizarea acestui proiect a fost o experiență foarte interesantă, dar și destul de provocatoare, mai ales când a venit vorba de partea grafică. Cea mai mare problemă a fost să implementez logica prin care să se șteargă și să se redeseneze strict porțiunile de pe ecran unde s-au mișcat obiectele de la un frame la altul, dar după ce am reușit să fac jocul să ruleze fluid, rezultatul a meritat tot efortul.

Download

Link GitHub: https://github.com/PredaAndrei3/AsteroidAvoider

Bibliografie/Resurse

Resurse Hardware:

Manual utilizare Display TFT LCD 3.2

ATmega328P Datasheet

Resurse Software:

SSD1306 Library

Coliziune elastică

Export to PDF