Igraća konzola temeljena na mikroupravljaču

Golić, Marina

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

IGRAĆA KONZOLA TEMELJENA NA MIKROUPRAVLJAČU

Završni rad

Marina Golić

Osijek, 2023.

Sadržaj

1. UVOD

Zadatak ovog rada je razviti igraću konzolu temeljenu na mikroupravljaču. Mikroupravljači zbog napretka u njihovim karakteristikama danas imaju široku primjenu, pa tako i u razvijanju igraćih konzola. Neke od karakteristika koje današnji mikroupravljači imaju su: jezgra koja radi na frekvenciji i do 400MHz, veliki kapacitet pohrane podataka zbog velikih vrijednosti *FLASH*, *RAM* i *SRAM* memorije, razna komunikacijska sučelja, puno ulaznih i izlaznih portova, te mnogi mikroupravljači imaju ugrađeni DMA kontroler. Iz navedenih razloga, danas umjesto procesora jednostavnije igraće konzole unutar sebe imaju mikroupravljač. Prijenosne igraće konzole koje unutar sebe imaju mikroupravljač su Nitendo GAME&WATCH i *tamagotchi*. U ovom radu treba realizirati da se igraća konzola igra pomoću akcelerometra. U prvom dijelu rada će biti opisane komponente koje će se koristiti za razvijanje hardwarea igraće konzole, u drugom dijelu rada će biti opisano kako će se odabrane komponente spojiti sa mikroupravljačem, te problemi koji su nastali tijekom spajanja komponenata sa mikroupravljačem. Nakon toga u trećem poglavlju će biti opisani dijelovi programskog koda vezani uz izbornik i igre. U zadnjem poglavlju će biti prikazano i opisano kako radi igraća konzola, i kako se rukuje sa igraćom konzolom.

1.1. Zadatak rada

Zadatak završnog rada je razvoj i implementacija jednostavnijih računalnih igara u mikroupravljački sustav s mogućnošću kontrole akcelerometrom.

2. TEORIJSKA PODLOGA

2.1. TFT Zaslon

Za prikaz igraće konzole je odabran zaslon LCD TFT dijagonale 2.4 inča odnosno rezolucije 240 (visina)*320 (širina) piksela. Zaslon se sastoji od pozadinskog osvjetljenja i RGB zaslona. Istoimeni zaslon unutar sebe ima čip pomoću kojega je omogućena vrlo jednostavna kontrola piksela, kontrola je omogućena pomoću pisanja programskog koda u kojem se napiše raspon piksela koji predstavlja određenu boju. Odabrani zaslon je i osjetljiv na dodir. Izgled prednje i zadnje strane pločice zaslonskog modula koji će biti korišten u radu vidljiv je na slici 2.1.

Slika 2.1. Prikaz prednjeg i zadnjeg djela TFT LCD zaslona [1]

Tablica 2.1. Osnovne tehničke specifikacije TFT LCD zaslona [2]

Specifikacija zaslona	Vrijednosti
Rezolucija zaslona	240*320
Radni napon	3.3 V
Način rada	SPI i 8 bitni mod
IC sučelje	ILI9341
Izlaz	720 izlaza za uvor TFT MOS tranzistora 320 izlaza za upravljačke eletkrode
Podrška za Arduino razvojnu platformu	Da
Komunikacija	8/9/16/18 bitna paralelna 8080 komunikacija 6/16/18 bita RGB komunikacija 3 linije/ 4 linije SPI komunikacija
Način rada zalona	Full color mode – mod rada sa najvećim brojem boja (262 000 boja) Reduce color mode – smanjen broj boja (256 boja)

Odabrani zaslon ima serijsku komunikaciju, odnosno SPI komunikaciju (SPI, engl. *Serial Peripheral Interferance)*. SPI označava serijsko periferno sučelje, to je standard sinkrone komunikacijske sabirnice. SPI je razvila Motorola, SPI omogućava brzi prijenos podataka i ima jednostavnu implementaciju. SPI komunikacija odvija se u konfiguraciji *master/slave*, gdje jedan *master* može birati između jednog ili više *slave* uređaja. Poznata je kao četvero-žična serijska komunikacija i zahtjeva upotrebu četiri podatkovne žice na glavnom kontroleru. Iz glavnog uređaja izlazi SCLK odnosno signal serijskog takta, SLCK određuje u kojem će se trenutku (vremenskom intervalu) prenositi bitovi podataka. MOSI (engl. *Master Output, Slave Input*) predstavlja glavni izlaz za *master,* on preko sabirnice prenosi podatke s *master-a* na *slave,* MISO (engl. *Master Input Slave Output*) prenosi podatke sa *slave-a* na *master*. Dok *master* generira jedan takt signala, MISO i MOSI mogu istodobno prenositi *clock* signal kada je to potrebno i tako je omogućen *full-duplex* mod. *Full duplex* je način komunikacije kod koje se u istom trenutku i šalju i primaju podaci. *Slave select* određuje koji *slave* trenutno prenosi podatke na sabirnici i omogućava sabirnici da više *slave-ova* istovremeno šalju podatke po potrebi.

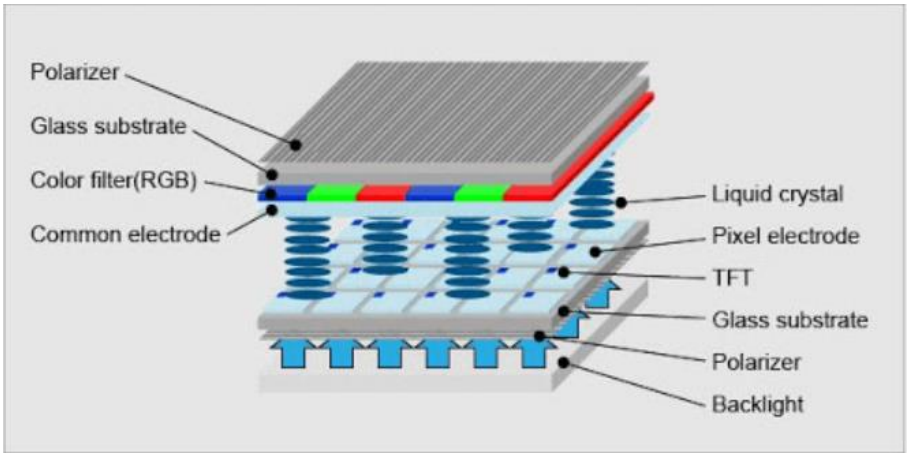
Kad korisnik implementira više *Slave select* pinova, svaki od tih pinova će dijeliti linije MISO,MOSI i SCLK, ali i svaki od njih u tom slučaju zahtjeva i zasebnu SS liniju. Ako neki pin ima zasebnu SS liniju to označava: ograničenje broja SS linija zbog izlaznog kapaciteta glavnog kontrolera, i nije potrebna formalna shema adresiranja. Možemo zaključiti da je SPI komunikacija namijenjena za uređaje koji zahtijevaju određeni broj linija.

SPI komunikacija gledana sa hardware-ske razine je vrlo jednostavna za primjenu, ova komunikacija nema nekakvo ograničenje što se tiče signala takta, pa je iz tog razloga vrlo lako postići i veće brzine prijenosa podataka te je iz tog razloga za razvijanje igrače konzole odabran ovaj zaslon. Ova komunikacija za rad ne zahtijeva visoki iznos napona napajanja, a to su iznosi od 3.3 V ili napon od 5 V.

Pomoću SPI komunikacije korisniku je omogućeno:

- Stvaranje različitih duljina riječi
- Dodavanje još *flow-control* pinova
- Omogućeno je istovremeno slanje podataka preko MISO i MOSI pinova
- Mana SPI komunikacije je da ako korisnici u programu ne navedu sve komponente može doći do nekompatibilnosti.

Struktura TFT LCD zaslona (engl. *Thin Film Transistor Liquid Crystal Display*) koja je prikazana na slici 2.2. se sastoji od dvije staklene ploče (engl. *Glass substrate*) . Sa vanjskih strana staklenih ploča se nalaze filterski slojevi koji se nazivaju polarizatori (engl. *polarizer*) . Staklena ploča ima filter pomoću kojega filtrira boje, a taj filter je (engl. *color filter*) RGB. Zajednička elektroda (engl. *common electrode*) zatvara put koji teče između dva sloja. Između dvije staklene ploče se nalaze tekući kristali (engl. *Liquid crystal*). Tekući kristali se pokreću kada dođe do razlike napona između stakla filtra u boji i TFT zaslona. TFT ili tankoslojni tranzistori, to su tranzistori s efektom polja. TFT staklo ima tankoslojnih tranzistora koliko ima i piksela. Piksel elektrode (engl. *Pixel electrode*) su vodiči koji usmjeravaju struju iz piksela. Razina pozadinskog osvjetljenja (engl. *backlight*) se određuje na temelju kretanja tekućih kristala koji stvaraju boju [3].

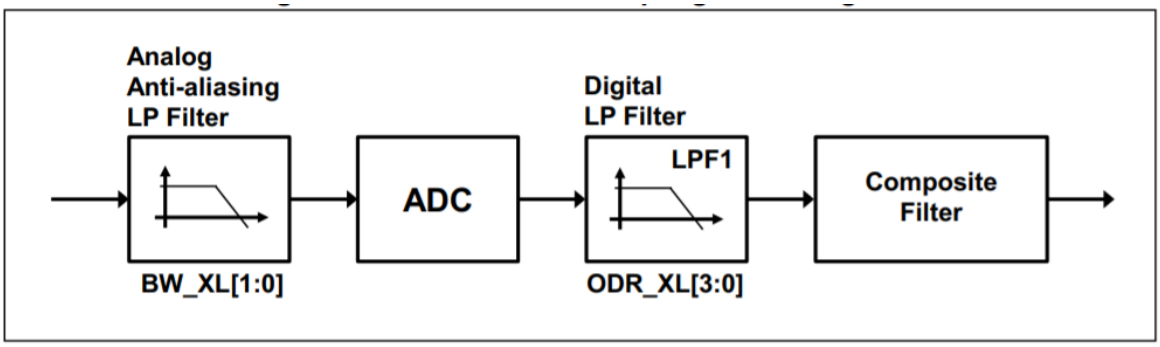


Slika 2.2. Prikaz strukture TFT LCD zaslona [3]

Ovaj zaslon je odabran između tri zaslona koji su bili planirani za ovaj projekt. Osim odabranog TFT LCD zaslona za realizaciju se trebao koristiti Nokia 5110 zaslon. Nokia 5110 je monokromatski zaslon, njegov nedostatak je taj što prikazuje ispis na zaslon u samo dvije boje, što nije dovoljno za neke od igara koje su razvijene. Rezolucija zaslona je 84x48 piksela što je vrlo mala rezolucija za neke zahtjevnije igre, ovaj zaslon ima također SPI komunikaciju. Za razvoj igraće konzole je ulazio u obzir i grafički zaslon rezolucije 128x64 koji ima ST7920 upravljač. Odabrani zaslon je bolji izbor od ostala dva koja su navedeni zato što je zaslon u boji, ima zaslon osjetljiv na dodir i veličina zaslona je dovoljna za igranje različitih igara, te ima na zadnjoj strani tiskane pločice utor za SD karticu, na SD karticu je moguće spremiti razvijene igre.

2.2. LSM6DS3 akcelerometar i žiroskop

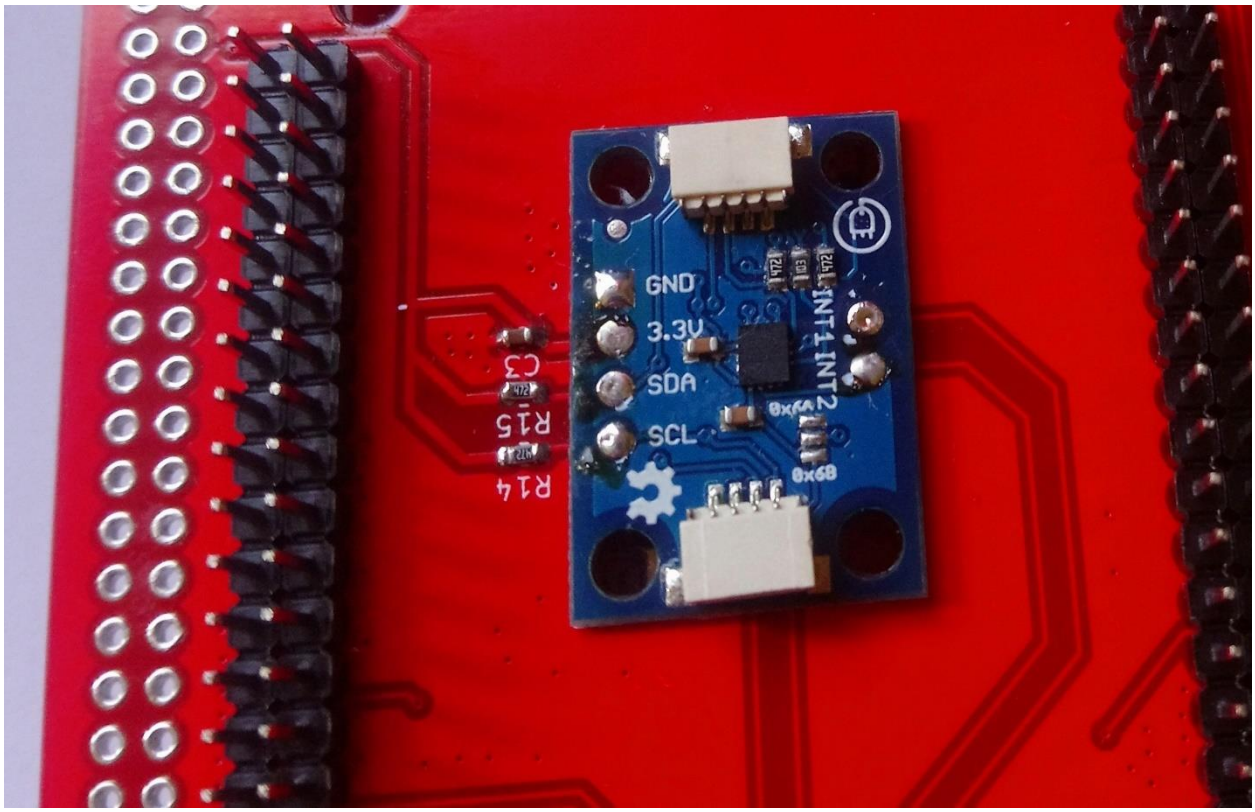
Akcelerometar je uređaj pomoću kojega se može izmjeriti analiza linearnog i kutnog ubrzanja. On mjeri silu ili više sila koji djeluju na njega, a ta sila se na njega odnosi kao kada bi se on nalazio u Kartezijevom sustavu, u sustavu su tri osi, te osi su X, Y i Z. Ako modul miruje, on će cijelo vrijeme očitavati silu koja u tom trenutku jedino djeluje na njega, a to je sila teža. Kad korisnik modul uzme ili na neki drugi način ga makne iz stanja mirovanja on istog trenutka izračunava sve sile koje djeluju na njega. Akcelerometar može detektirati i frekvencije niže od 100 kHz u ovom slučaju zato što se za komunikaciju između mikroupravljača i akcelerometra koristi I2C komunikacija[6], odnosno takt na SCL liniji iznosi 100 kHz. Iz tog razloga je akcelerometar jako osjetljiv na sile koje djeluju na njega. Ako se ubrzanje događa u bilo kojoj ravnini koja je suprotna od ravnine u kojoj je usmjeren senzor, akcelerometar će prikazivati negativnu vrijednost izmjerene veličine. Senzor može detektirati nagibe, udarce, tapkanje, korake itd. [4]. Ako na akcelerometar ne djeluje ni jedna druga sila osim sile teže dakle ako je u stanju mirovanja vrijednosti za X, Y i Z os će biti sljedeće: X = 0 g, Y = 1 g, i Z = 0 g, ako se akcelerometar nagne u lijevu stranu vrijednosti će biti: X = 1 g, Y = 0 g i Z = 0 g , a ako se akcelerometar nagne u desnu stranu vrijednosti će biti sljedeće: X = -1 g , Y = 0 g i Z = 0 g.[4]



Slika 2.3. Blok dijagram LSM6DS3 akcelerometra [5]

Na slici 2.3. je prikazan blok dijagram uzorkovanja akcelerometra, možemo vidjeti da blok dijagram ima kaskadu od 4 bloka. Prvi blok (engl. *Analog Anti-aliasing LP Filter*) je analogni niskopropusni filtar, drugi blok sa slike (ADC) je analogno digitalni pretvarač, treći blok sa slike (engl. *Digital LP Filter*) je digitalni niskopropusni filtar i četvrti blok (engl. *Composite Filter)* je kompozitni filtar. Analogni signal koji dolazi na ulaze akcelerometra filtrira se pomoću analognog nisko propusnog filtra prije nego što ga ADC pretvori u digitalni signal, digitalni LP filtar je jedino moguće koristiti u *High Performance* modu. Pomoću filtra nagiba se izračunava nagib akcelerometra, a može se koristiti i za detekciju buđenja i za aktivnost ili neaktivnost.

Pomoću žiroskopa se detektira i izračunava kutna brzina i orijentacija predmeta na koji je stavljen žiroskop. Mjerna jedinica koja se koristi za prikaz detektirane vrijednosti kutne brzine su stupnjevi u sekundi, ta vrijednost se vrlo malo mijenja tj. ne mijenja se puno s obzirom na promjenu temperature i tijekom nekog dužeg vremenskog perioda.



Slika 2.4. Prikaz odabranog akcelerometra LSM6DS3

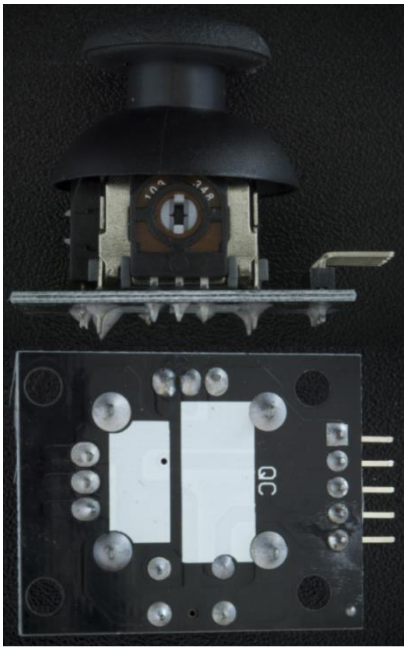
Akcelerometar koji je odabran je LSM6DS3 (slika 2.4), potrošnja energije odabranog akcelerometra je 0.9 mA u kombiniranom načinu rada, i 1.25mA u kombiniranom načinu rada visokih performansi do 1.6 kHz. Analogni napon napajanja od 1.71 V do 3.6 V. Akcelerometar ima raspon ubrzanja od +-2g/+-4g/+-8g/+-16g, taj raspon može izmjeriti i brzinom do 6700 uzoraka u sekundi. Akcelerometae ima raspon kutne brzine od +-125/+-245/+-500/+-1000/+-2000.[6]

2.3. Joystick

Joystick je uređaj koji je vrlo jednostavan, postoji analogna i digitalna izvedba *joysticka*. Najčešće se koristi u igraćim konzolama i u zrakoplovima. S ovim uređajem se lako upravlja, sastoji se od stupa koji je s gumenom ovojnicom pričvršćen na plastičnu podlogu. Ta plastična podloga je PCB, koji je vrlo jednostavno izveden, PCB ima nekoliko jednostavnije dizajniranih električnih krugova. Električni krugovi omogućavaju prijenos električne energije s jedne vodljive pozicije na drugu vodljivu poziciju.

Ako *joystick* stoji u nultom položaju odnosno ako nije maknut iz stanja mirovanja, svi pojedinačni krugovi na pločici su odspojeni, osim jednog koji detektira da je štap u stanju mirovanja. PCB ima nekoliko upravljačkih vodova, stup se može okretati u okrugu od 360 stupnjeva[6].

Spajanje modula sa mikroupravljačem je vrlo jednostavno, ovisno o tome da li je odabran analogni ili digitalni *joystick*. *Joystick* se najčešće povezuje s mikroupravljačem pomoću žica, mikroupravljač očitava vrijednosti u horizontalnoj i vertikalnoj osi, očitane vrijednosti su vrijednosti pozicija sa stupa *joysticka* na kojem se nalazi gljiva s kojom upravlja korisnik, to vrijedi za analogni *joystick* koji je prikazan na slici 2.5. Ako je odabran digitalni *joystick* on ima prekidač za uključivanje i isključivanje a pomoću toga se označava položaj stupa. Ako je štap rotiran i na nekoj je od pozicija (lijevo,desno,gore ili dolje) to je u digitalnom smislu logička jedinica tada bi se omogućio smjer ovisno o poziciji na kojoj je štap, ako je štap između dvije pozicije lijevo i prema gore onda se matematikom izračuna točna putanja.



Slika 2.5. Na slici je prikazan joystick koji se koristi u ovom projektu[7]

Analogni *joystick* ima različite moguće pozicije, to znači da on ima puno mogućih stanja, što omogućava bolju preciznost za razliku od digitalnih *joysticka* koji imaju osam digitalnih stanja. U ovom projektu je odabran analogni *joystick* baš upravo iz tog razloga jer ima veću preciznost i više mogućih stanja. Odabrani *joystick* ima ulazni napon koji može biti maksimalno do 5V.

2.4. Mikroupravljač

Mikroupravljač je u stvarnosti integrirani krug, a sama riječ mikroupravljač govori da on manipulira tj. on upravlja s ostalim dijelovima elektroničkog kruga. On upravlja s ostalim komponentama pomoću mikroprocesorske jedinice (MPU, engl. *Microprocessor Unit*), memorije i nekih perifernih uređaja. Danas svijet ne bi mogao funkcionirati bez mikroupravljača jer su napravljeni tako da su optimizirani za različite aplikacije koje zahtijevaju funkcionalnost obrade i da mogu komunicirati sa analognim, digitalnim ili elektromehaničkim komponentama, takav jedan uređaj je prikazan na slici 2.6. Mikroupravljač se više bavi okolnim hardware-om koji omogućuje mikroupravljaču upravljanje sustavom, a ne samo da izvršava upute i sprema podatke.

Mikroupravljač ima sljedeće komponente: središnju procesorsku jedinicu (CPU), neku vrstu trajne memorije, trenutnu memoriju, periferne uređaje i pomoćne komponente. Središnja procesorska jedinica izvodi aritmetičke operacije nad podacima, manipulira protokom podataka i generira upravljačke signale i to na način kako mu je zadao programer unutar programskog koda. Kompliciranе sklopove koji su napravljeni za rad središnje procesorske jedinice (CPU) dizajner sustava ne može vidjeti. Pojavom i razvojem C programskog jezika i razvojem integriranih okruženja se uvelike pojednostavilo i ubrzao razvoj mikroupravljača te također se povećala proizvodnja i danas se gotovo u sve uređaje mogu ugraditi. Programski kod najčešće pisan u strojnom programskom jeziku, taj programski kod govori središnjoj procesorskoj jedinici što ona treba raditi, taj kod se pohranjuje i čuva unutar trajne memorije (*FLASH* memorija). Trenutna memorija (RAM, engl. *Random Access Memory*) privremeno pohranjuje podatke odnosno podaci se iz nje brišu nakon što mikroupravljač izgubi napajanje odnosno kad se ugasi ili makne s napajanja. Privremeno čuvanje podataka se može izvesti i pomoću nekog od unutarnjih registara unutar CPU-a. Periferni moduli su najčešće moduli pomoću kojih mikroupravljač može jednostavno komunicirati i upravljati s uređajima koji su dio mikroupravljača. Primjer takvog modula je digitalno analogni pretvarač, analogno digitalni pretvarač i generator koji generira referentni napon.

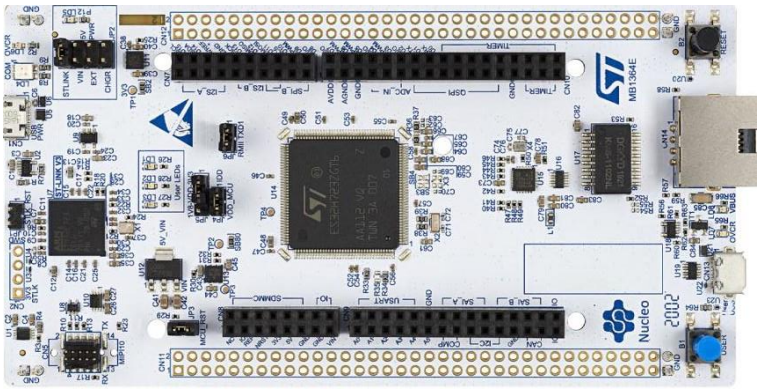
Periferni moduli mogu biti:

- Generiranje takta: unutarnji oscilator, kvarc kristal, fazno zaključana petlja.
- Vremenski interval: sat realnog vremena (RTC, engl. *Real Time Clock*), tajmer za opću namjernu, pulsno širinska modulacija (PWM, engl. *Pulse Width Modulation*), brojač događaja.
- Analogna obrada signala: operacijsko pojačalo, analogni komparator, analogno-digitalni pretvornik
- Ulazi i izlazi: sklop digitalnog ulaza i izlaza za općenitu upotrebu
- Serijska komunikacija: SPI, I2C, UART, USB

Slika 2.6. Primjer mikroupravljačke pločice[8]

Mikroupravljač koji je odabran za razvoj igraće konzole je razvojna pločica iz NUCLEO 144 serije, razvojna pločica je razvijena od strane tvrtke STMicroelectornics. Razvojna pločica na sebi ima mikroupravljač STM32H743ZIT6 kao što je i prikazano na slici 2.7. Razvojna pločica je odabrana iz razloga što omogućava vrlo brzo osvježavanje zaslona zato što ima ADC sa puno ulaznih kanala a to je u ovom slučaju potrebno jer to omogućava fluidnost tijekom igranja igara, zatim drugi razlog odabira ove razvojne pločice je veliki broj ulaznih i izlaznih pinova koji su potrebni za implementaciju svih komponenata koje su potrebne za razvoj igraće konzole, treći razlog je što omogućava veliku brzinu komunikacije sa modulima koji se spajaju na nju, zatim još

jedan od razloga što za programiranje nije potrebno imati programator zasebno jer na razvojoj pločici se već nalazi STLink.



Slika 2.7. Prikaz odabrane razvojne pločice

Tablica 2.4. Prikaz specifikacije odabrane razvojne pločice[9]

Specifikacije razvojne pločice	Vrijednosti
Jezgra	32bitna ARM Corteks-M7 jezgra
Frenkvencija	Do 480MHz
Memorija	Do 2MB FLASH memorije, 1MB RAM memorije
Ulazi/izlazi opće namjene	Do 168 I/O portova s mogućnosti prekida (*engl.* Interrutps)
Debugger	Ugrađeni STLink/V3 programator sa SWD konektorom
Napajanje fleksibilne pločice	ST-Link, USB VBUS ili vanjski izbori
Dva gumba	User i reset
Radni napon	1.62V do 3.6V
Komunikacijski protokoli	I2C, SPI, UART, USART, LIN, USB OTG interface
Broj ADC	3
Broj ulaznih kanala ADC-a	36 kanala
Rezolucija ADC-a	16-bit

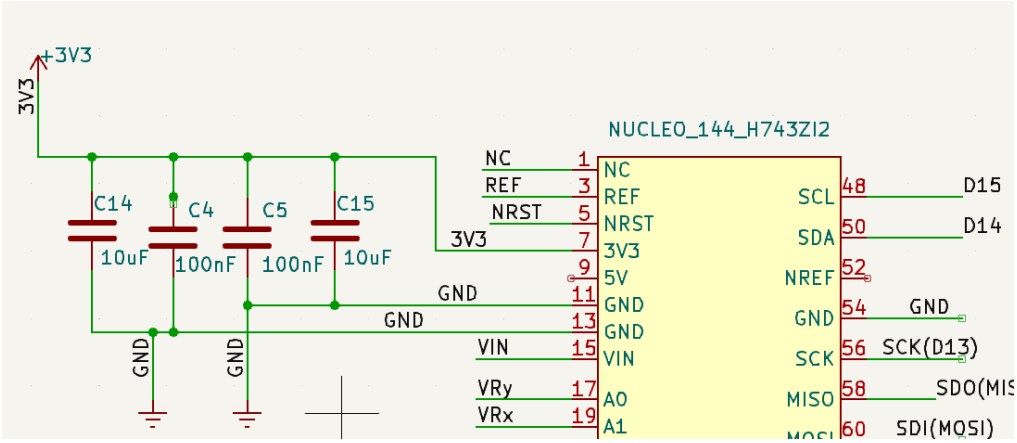
3. RAZVOJ HARDWARESKOG DIJELA IGRAĆE KONZOLE

3.1. Odabir na koji će se način napajati igraća konzola

Shema igraće konzole je razvijena unutar programa za automatizaciju elektroničkog dizajna KiCad 7.0. Igraća konzola u hardware-skom smislu se sastoji od sedam komponenata: razvojne pločice na kojoj je mikroupravljač STM32H743ZIT6, prekidača, LCD TFT zaslona, akcelerometra LSM6DS3, *joysticka*, baterije i tipke. Mikroupravljač je glavna jedinica na koju se spaja prekidač, LCD TFT zaslon, akcelerometar, *joystick*, baterija i tipke. Igraća konzola se može napajati pomoću baterije ili preko razvojne pločice koju smo direktno spojili pomoću USB kabela sa izvorom napajanja što može biti *laptop*, baterija ili neki drugi način napajanja. Baterija koja je odabrana za napajanje je litijeva baterija od 1200 mAh i 3.7 V. Baterija se spaja s igraćom konzolom pomoću JST konektora. Baterija također ima ugrađeni sustav zaštite od previsokog napona (odnosno sprječava prekomjerno punjenje), preniskog napona i kratkog spoja [11].

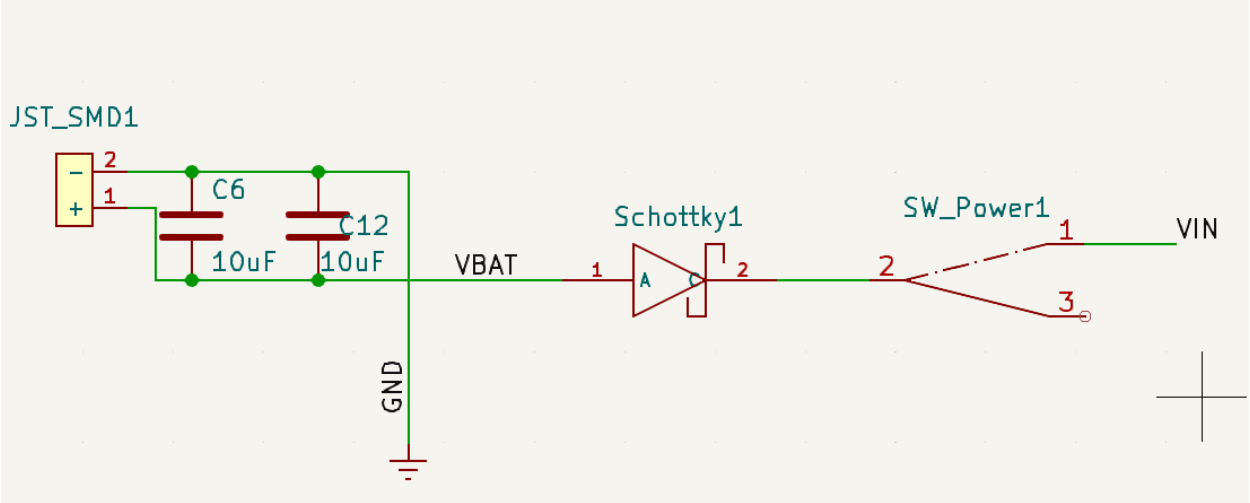
3.2. Spajanje mikroupravljača i prekidača

Mikroupravljač STMH743ZIT6 odnosno razvojna pločica iz Nucleo 144 serije koja se koristi za realizaciju igraće konzole radi na naponu od 3.3 V, te ima kristalni oscilator na 32.768 kHz.[10]. Prije nego što se spoji mikroupravljač sa prekidačem, na izvode napajanja odabranog mikroupravljača je potrebno staviti *decoupling* kondenzatore što možemo i vidjeti na slici 3.1.



Slika 3.1. Shematski prikaz spajanja *decoupling* kondenzatora na napajanje razvojne pločice

Decoupling kondenzatori su stavljeni iz razloga kako bi potisnuli visokofrekvencijske šumove u signalima, odnosno kako ne bi došlo do smetnji pri napajanju mikroupravljača. Spomenute smetnje mogu ometati pravilno napajanje mikroupravljača tako što može doći do pada u naponu napajanja ili čak se može i oštetiti izvod na mikroupravljaču ili sam mikroupravljač. Rješenje kako bi se izbjegle te smetnje i moguća oštećenja je da se između mase (minusa) i napajanja (plusa) stave *decoupling* kondenzatori. *Decoupling* kondenzatori potiskuju naponske špiceve u signalima, te na taj način se dobije stabilniji napon napajanja. Nakon dodavanja filtriranja kod napajanja mikroupravljača, potrebno je omogućiti da se igraća konzola može napajati preko mikroupravljača ili preko baterije što je prikazano na slici 3.2.



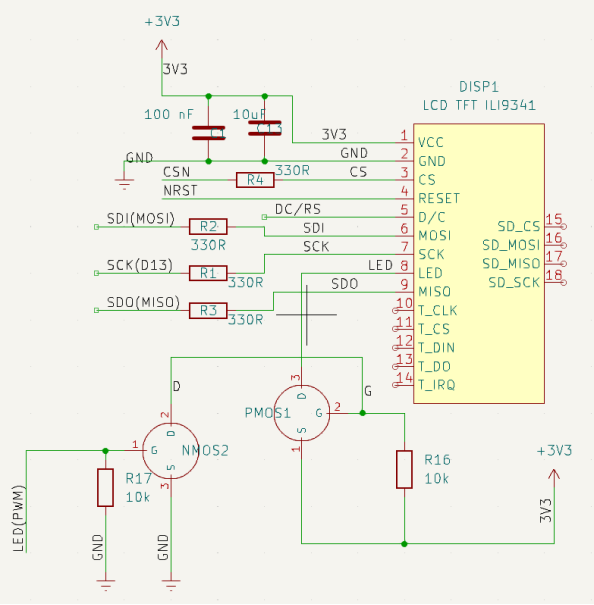
Slika 3.2. Shematski prikaz spajanja igraće konzole sa baterijom

Odabir da li će se konzola napajati preko razvojne pločice ili preko baterije je napravljeno pomoću *Schottky-eve* diode i prekidača što je i prikazano na slici 3.2. Ukoliko se igraća konzola napaja preko baterije onda je potrebno bateriju spojiti na JST konektor kod kojega su također stavljeni *decoupling* kondenzatori. Nakon kondenzatora na plus sa JST konektora odnosno baterije se spaja *Schottky* dioda. Dioda je stavljena iz tog razloga što će ona provesti napon samo ukoliko je prekidač spojen na JST konektor i to omogućava da se igraća konzola napaja preko baterije. Druga uloga dodavanja diode je što onemogućava protok struje u bateriju kad je prekidač u stanju da se konzola ne napaja preko baterije, jer u suprotnom da nema diode može doći do protoka struje u bateriju i baterija se može oštetiti. U slučaju da korisnik prekidač stavi u stanje da se igraća konzola napaja preko mikroupravljača, tada korisnik prvo mora premjestiti *jumper* na razvojnoj pločici na mjesto gdje piše STLink. Zatim uključi USB kabel u razvojnu pločicu.

U tom slučaju napon kroz diodu ne prolazi i ostatak kruga za napajanje preko baterije ne radi. Kada korisnik stavi prekidač u stanje da se konzola napaja preko mikroupravljača, napon sa prekidača je spojen na pin na mikroupravljaču odnosno na pin VIN. Izvod VIN je ulazni izvod na mikroupravljaču, što znači da je napon sa mikroupravljača spojen na taj izvod i igraća konzola ima izvor napajanja.

3.3. Spajanje mikroupravaljača sa LCD TFT zaslonom

Zaslon LCD TFT koji se koristi u realizaciji ovog projekta je spojen pomoću SPI komunikacije sa odabranom razvojnom pločicom na kojoj se nalazi mikroupravljač STM32H743ZIT6. Prije nego što se pinovi koji su potrebni za SPI komunikaciju spoje sa mikroupravljačem, potrebno je kod zaslona dodati *decoupling* kondenzatore između izvoda GND i 3.3 V. Na izvode: MOSI, MISO, SCK i CSN je bilo potrebno dodati otpornike u vrijednosti od 330Ω. Otpornici su spojeni između navedenih izvoda sa LCD TFT zaslona prema mikroupravljaču. Otpornike je bilo potrebno staviti zbog smetnji, te smetnje su zvonjava (engl. *ringing*) u signalu koji putuje od predajnika do prijamnika, odnosno u ovom slučaju od zaslona do mikroupravljača. Zbog zvonjave može nastati problem zbog lažnog prijelaza i cijeli krug se može oštetiti, pomoću otpornika se smanjuju špicevi kod zvonjave te na taj način ne može doći do prenapona. Osim navedenih pinova sa mikroupravljačem su povezani i pinovi NRST, DC/RS i LED pin kao što možemo i vidjeti na slici 3.3.



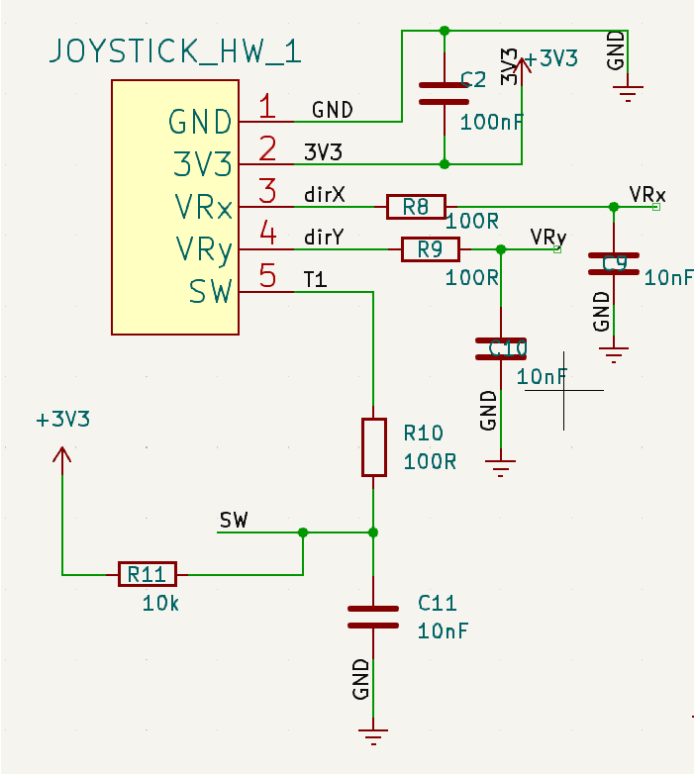
Slika 3.3. Shema spajanja LCD TFT RGB zaslona sa razvojnom pločicom

NRST pin je pin koji omogućava *reset* zaslona. DC/RS pin DC označava podatak (engl. *data*) , a RS odabrani registar (engl. *Register select*). LED pin je pin pomoću kojega se upravlja pozadinskim osvjetljenjem LCD TFT zaslona. Da bi se omogućilo upravljanje pozadinskim osvjetljenjem, potrebno je izvesti sklop koji će upravljati ili naponom ili strujom kroz svjetleće diode pozadinskog osvjetljenja. U ovom slučaju, odabrano je da će se upravljati naponom pomoću PWM modulacije. Prvi problem koji se pojavljuje je taj što izvod mikroupravljača nema dovoljno struje da upravlja s pozadinskim osvjetljenjem, stoga je potrebno koristiti tranzistor da bi se pojačala struja. Korišteni tranzistor je MOSFET P tipa (AO3401A). Razlog korištenja MOSFETa P tipa je taj što omogućava da se vrlo jednostavno prekida napajanje (jer pozadinsko osvjetljenje radi kada se na LED izvod dovede napon od 3.3 V). Kada se *gate* navedenog tranzistora spoji na GND, ukoliko se raspiše petlja za Ugs, dobiva se Ugs = -3.3 V, te je PMOS u stanju vođenja struje između D i S izvoda, jer je Ugs > Uth (napona praga vođenja). Ukoliko se na *gate* dovede 3.3 V i raspiše li se petlja za Ugs, dobiva se da je Ugs = 0V, odnosno Ugs < Uth i PMOS ne vodi. R16 predstavlja *pull-up* otpornik. Ovdje postoje mogući problem: logika je obrnuta, što predstavlja problem kod PWM modulacije, gdje ukoliko se postavi *duty cycle* na 100%, pozadinsko osvjetljenje ne svijetli, a ukoliko se postavi na 0%, pozadinsko osvjetljenje svijetli punim intenzitetom. Da bi se navedeni problem riješio, dodan je još jedan tranzistor, ovoga puta NMOS(CJ3400) između *gatea* PMOSa i GNDa. Ukoliko se na NMOS dovede 3.3 V, Ugs NMOSa je 3.3 V, što je veće od Uth (napona praga) i NMOS vodi, što znači da je napon na *gateu* drugog tranzistora približno 0V, time i Ugs ~ 3.3 V, dakle PMOS vodi. Ukoliko se *gate* NMOSa spoji na GND, njegov Ugs je tada 0V, odnosno Ugs < Uth i NMOS ne vodi, što znači da je *gate* PMOSa na 3.3 V, dakle Ugs PMOS iznosi 0V, što znači da PMOS ne vodi struju, a time i onemogućava rad pozadinskoga osvjetljenja. Na taj način je realizirana logika da na logičku jedinicu pozadinsko osvjetljenje radi, a na logičku nulu ne radi. Time je izvedeno da *duty cycle* PWMa nije obrnut. Otpornik R17 sa slike 3.3. je pull down otpornik koji definira stanje NMOSa kad nema nikakav signal na sebi.

Nakon što su dodane sve komponente koje su potrebne za PWM modulaciju na *drain* P-MOS tranzistora spaja se izvod sa LCD TFT zaslona, zatim na *gate* N-MOS tranzistora spaja se odabrani PWM izvod sa mikroupravljača što je i prikazano na slici 3.3.

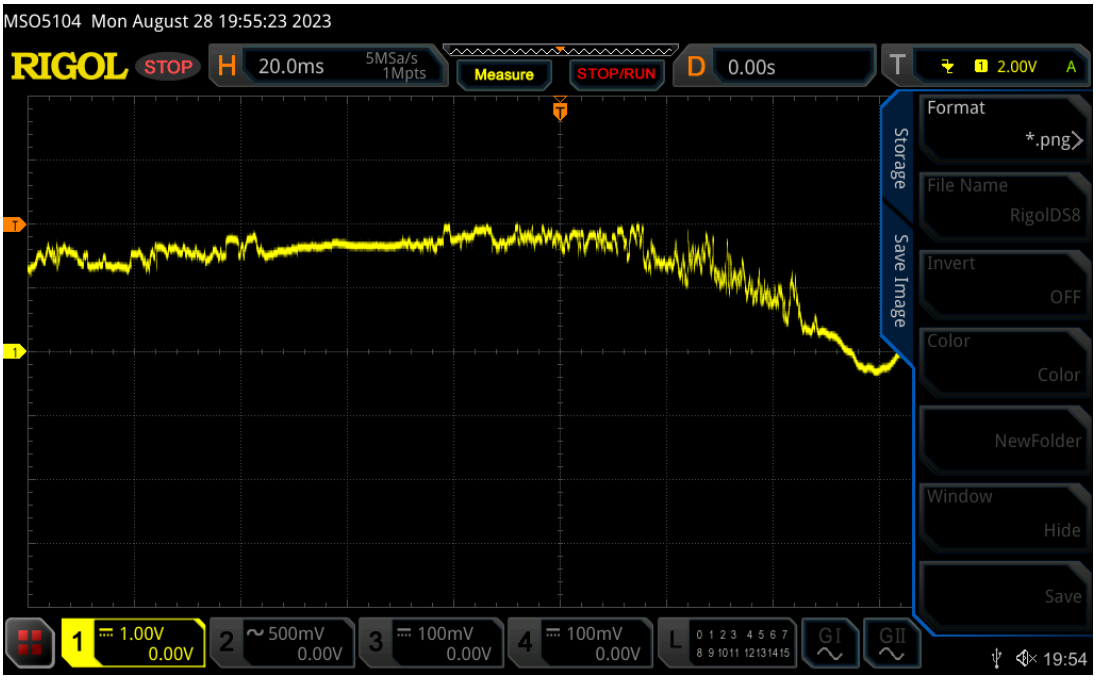
3.4. Spajanje mikroupravljača sa joystickom

Joystick koji je odabran za razvoj igraće konzole je potrebno spojiti s mikroupravljačem. *Joystick* se treba prvo spojiti na izvor napajanja. Odabrani *joystick* se inaće spaja na napon od 5V međutim u ovom slučaju se spaja na napon od 3.3 V, zato što odabrani mikroupravljač radi na naponu od 3.3 V. *Joystick* se spaja na izvor napajanja tako da između izvoda 3.3 V na *joysticku* i izvoda GND na *joysticku* se spoji *decoupling* kondenzator vrijednosti 100 nF. Mikroupravljač i *joystick* je potrebno spojiti pomoću tri linije, dvije linije su analogne i jedna je digitalna. Analogne linije su izvodi sa potenciometara koji se nalaze na dnu stupa *joysticka*. Digitalna linija je izvod sa tipke na *joysticku*. Izvodi sa potenciometra na *joysticku* koje je potrebno spojiti sa mikroupravljačem su izvod za horizontalnu os, te izvod za vertikalnu os i tipka. Prije nego što se izvodi spoje na mikroupravljač potrebno je na obadvije analogne linije dodati otpornik i kondenzator kao što je i prikazano na slici 3.4. Digitalna linija je tipka na *joysticku* koja se nalazi na sredini štapa, tipka ima dva moguća stanja, i izvod za tipku se spaja na digitalni izvod na mikroupravljaču.



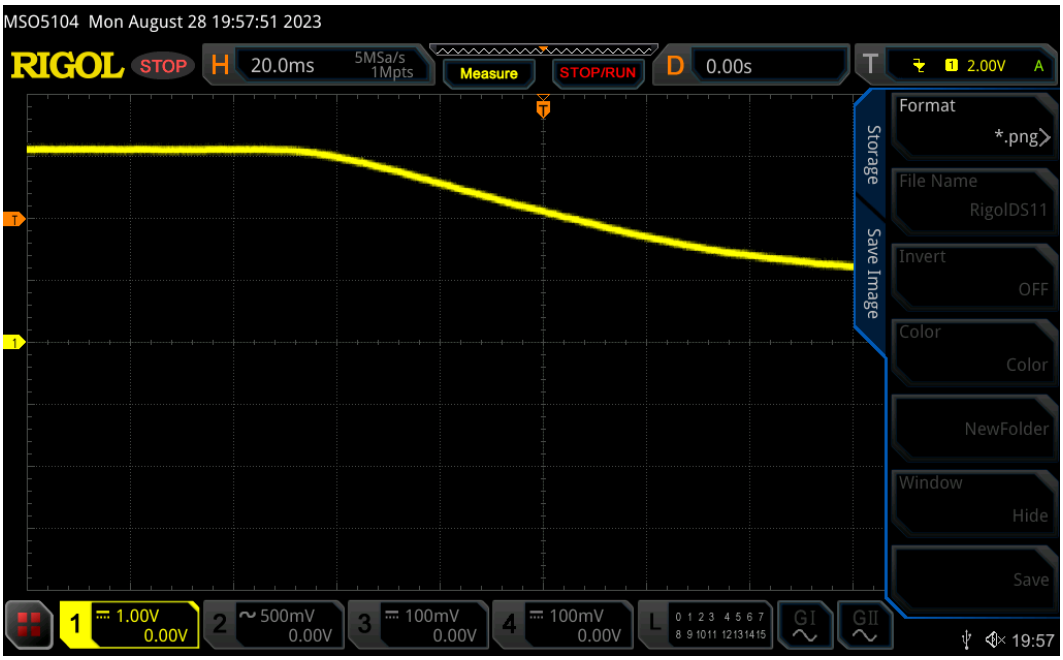
Slika 3.4. Shematski prikaz kako su spojeni *joystick* i mikroupravljač

Razlog dodavanja kondenzatora i otpornika na analogne izvode je filtriranje odnosno otklanjanje šuma s potenciometra *joysticka*. Dakle ukoliko se ne ukloni šum očitanje neće biti točno. Šum snimljen s potenciometra *joysticka* je prikazan na slici 3.5.



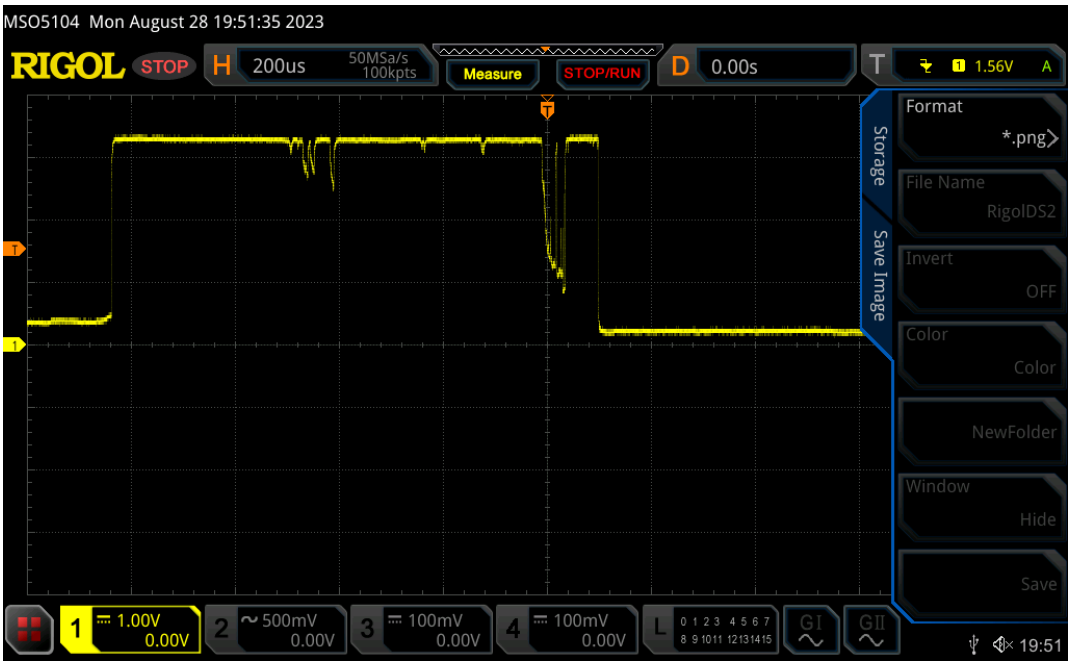
Slika 3.5. Prikaz snimljenog šuma sa potenciometra *joysticka* prilikom upravljanja gljivom *joysticka*

Spoj kondezatora i otpornika potreban za filtriranje koji je prikazan na slici 3.4. se naziva *low pass* filter odnosno niskopropusni filter. *Low pass* filter je potrebno dodati kako bi se otklonio šum koji je prikazan na slici 3.5. Šum je potrebno ukloniti iz razloga što mikroupravljač pomoću potenciometra, očitava točan kut pod kojim se trenutno nalazi *joystick* .Šum se treba ukloniti kako bi se dobile što preciznije vrijednosti pri pomicanju štapa na *joysticku*. Na slici 3.6. je prikazan snimljeni signal sa potenciometra *joysticka* nakon dodavanja *low pass* filtera.



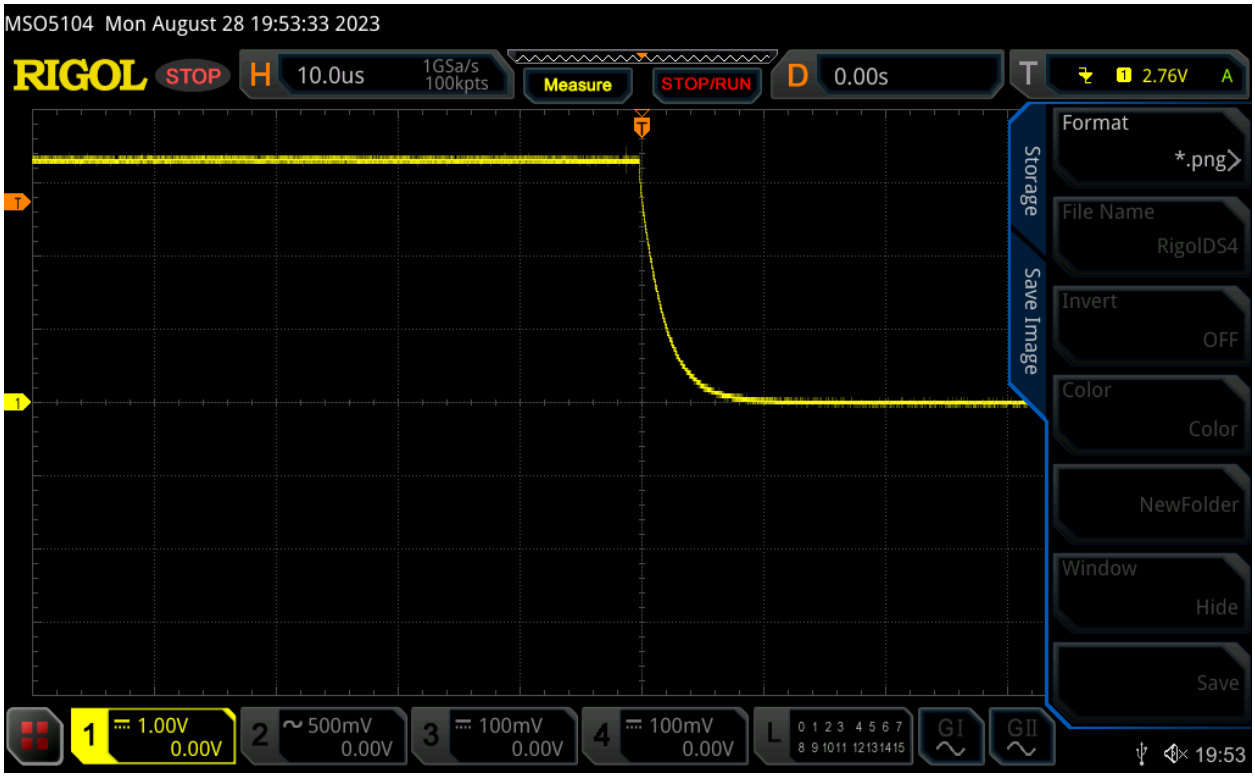
Slika 3.6. Prikaz snimljenog signala sa potenciometra na *joysticku* nakon dodavnja filtera

Nakon spajanja *joysticka* sa mikroupravljačem je uočeno da tipka na *joysticku* ima odskok (engl. *debonuce*) što je i prikazano na slici 3.7. Snimljeni signal na slici prikazuje odskok snimljen sa tipke, snimljeni odskoci su ustvari višestruki pritisci, u stvarnosti je to jedan pritisak. Odskok se može riješiti u programskom kodu ili hardware-ski.



Slika 3.7. Na slici je prikazan odskakujući signal snimljen sa tipke na *joysticku*

U ovom slučaju je odabrano hardware-sko rješenje. Rješenje je da se između *joysticka* i mikroupravljača spaja *debounce* RC filter. *Debounce* RC filter se sastoji od jednog otpornika i jednog kondenzatora [14]. Kondenzator se spaja u seriju sa otpornikom te se zatim taj spoj spaja paralelno sa pinom na mikroupravljaču (slika 3.4.). Uloga kondenzatora je da nakon što se kondenzator napuni, na dopušta struji da teče kroz njega te na taj način ima djelovanje kao razmak u stanjima tipke. Kondenzator na taj način onemogućava pojavu više stanja tipke, odnosno uklanja odskakanje gumba (engl. *bounce*). Međutim nije dovoljno staviti samo kondenzator, potrebno je staviti i otpornik kako bi se ograničila struja koja ulazi u kondenzator. Snimljeni signal sa tipke na *joysticku* nakon dodavanja kondenzatora i otpornika je prikazan na slici 3.8.

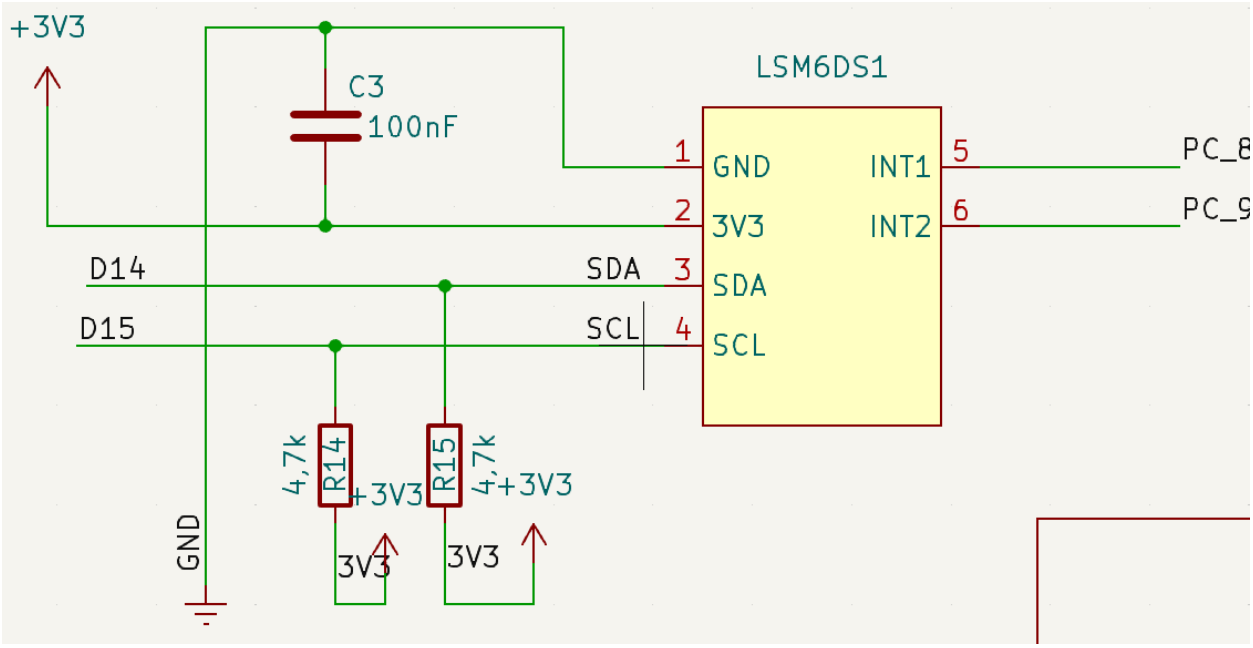


Slika 3.8. Na slici je prikazan signal sa tipke na *joysticku* nakon dodavanja *debounce* RC filtera

Zatim je potrebno nakon *debounce* RC filtra dodati još jedan otpornik, taj otpornik je *pull up* otpornik. *Pull up* otpornik je potrebno dodati kako bi se predefiniralo stanje tipke.

3.5. Spajanje mikroupravljača i LSM6DS3

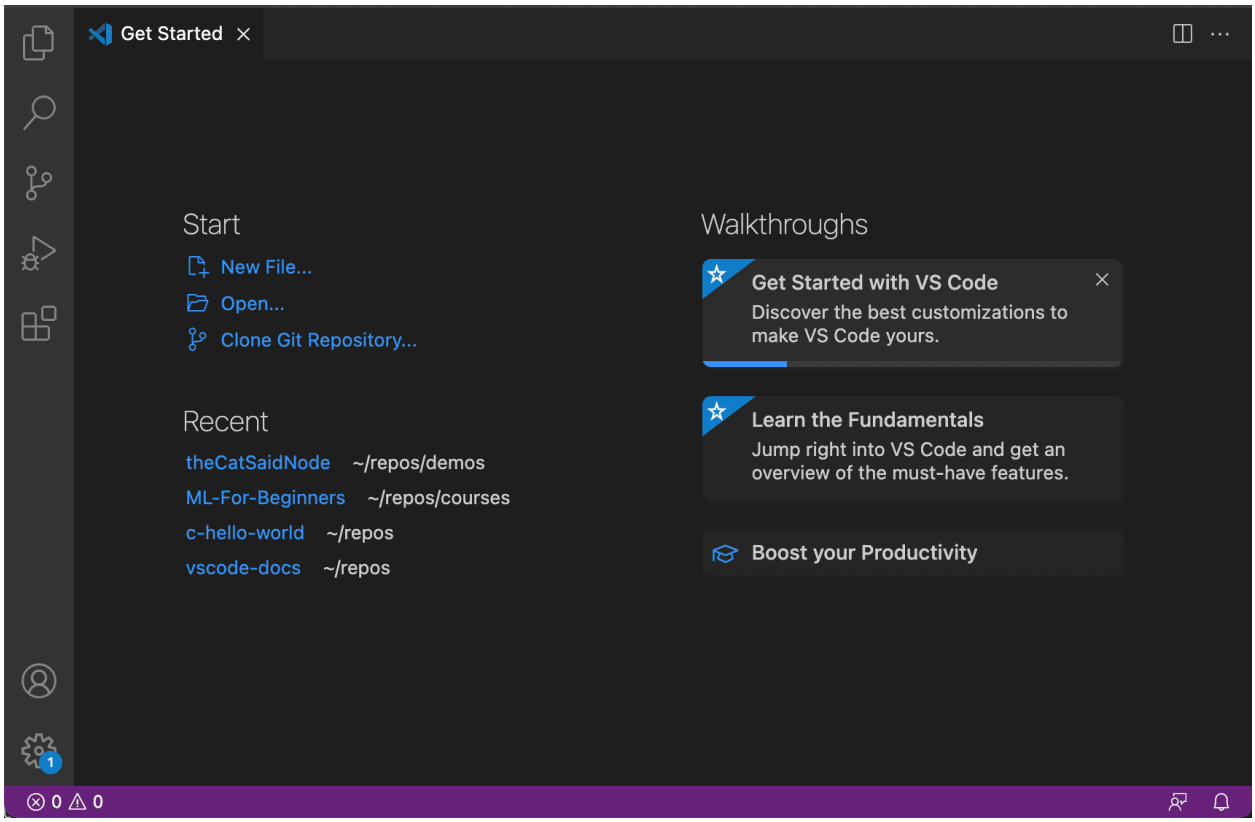
Mikroupravljač i odabrani akcelerometar LSM6DS3 su spojeni putem I2C komunikacije. Prvo treba između izvoda napajanja na linijama GND i 3.3 V akcelerometra staviti *decoupling* kondenzator. Zatim na liniju SDA i SCL treba staviti otpornike prema 3.3 V liniji. Otpornike je potrebno dodati jer akcelerometar nakon spajanja sa odabranim mikroupravljačem bez otpornika nije radio. Otpornici koji su stavljeni su *pull-up* otpornici. *Pull-up* otpornike je potrebno staviti prema I2C protokolu[15], jer svaka signalna linija ima *pull-up* otpornik iz razloga kako bi se linija stavila u predefinirano stanje logičke jedinice. Linije SDA i SCL je nakon spoja sa otpornikom potrebno spojiti na izvode razvojne pločice na kojoj se nalazi mikroupravljač STM32H743ZIT6 kao što je i prikazano na slici 3.9.



Slika 3.9. Prikaz sheme spajanja akcelerometra LSM6DS3 sa razvojnom pločicom

4. RAZVOJ PROGRAMSKOG DIJELA ZA IGRAĆU KONZOLU

Programski kod za programski dio igraće konzole je razvijen unutar *Visal Studio Code-a* koji je prikazan na slici 4.1.



Slika 4.1. Prikaz *Visual Studio Code* programskog okruženja

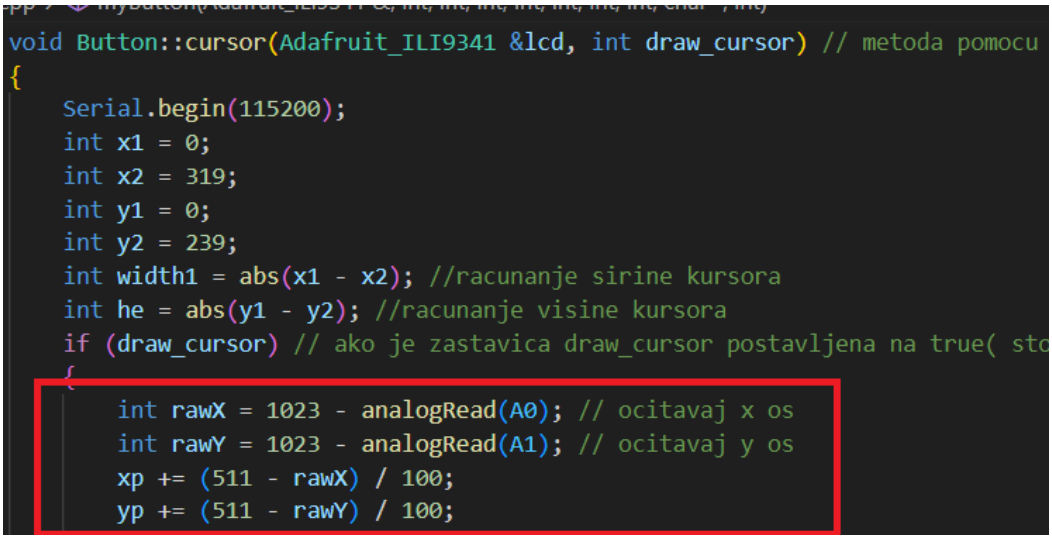
Programski kod unutar *Visual Studio Code-a* pisan je C/C++ programskim jezikom s podrškom za Arduino platformu. Programski dio igraće konzole se dijeli na dio vezan uz izbornik i dio vezan za igre. Izbornik i igre su podijeljeni u različite datoteke. Tijekom pisanja koda bilo je potrebno i koristiti biblioteke sa interneta kako bi pojedine komponente igraće konzole pravilno radile. Kako bi se razvijeni izbornik i razvijene igre mogle prikazivati na LCD TFT zaslonu prvo su kreirane dvije datoteke koje to omogućavaju. Te dvije datoteke su screen.h i screen.cpp, unutar *header* datoteke screen.h je potrebno uključiti biblioteku ILI9341_STM32.h [16]. Biblioteku je potrebno uključiti kako bi mikroupravljač mogao komunicirati sa zaslonom. Unutar *header* datoteke screen.h se nalazi klasa unutar koje se nalazi deklaracija metode koja je potrebna za ispisivanje na zaslon i za brisanje sa zaslona.

Izvan klase u *header* datoteci screen.h se nalazi još jedna funkcija. Ta funkcija se odnosi na osvježavanje zaslona koja se povezuje sa klasama za kuglicu i tipke. Unutar datoteke screen.cpp se nalazi definicija metode potrebne za ispis na zaslon i brisanje za zaslona i definicija metode za osvježavanje zaslona. Definicija metode za osvježavanje zaslona unutar sebe ima samo zastavicu koja je postavljena na *true*. Zatim unutar definicije metode za ispis na zaslon i brisanje sa zaslona ispituje da li je zastavica jednaka *true*. Ukoliko je zastavica jednaka *true* dogodit će se ispis na zaslon, brisanje za zaslona, i zaslon će se rotirati. Zatim klasa za tipku koja se nalazi unutar datoteke GUI.h i klasa za kuglicu koja se nalazi unutar datoteke ball.h u konstruktoru imaju pokazivač na funkciju kao argument, taj pokazivač pokazuje na adresu spomenute metode za osvježavanje zaslona. Svaka od spomenute dvije klase unutar sebe ima deklariranu metodu unutar čije se definicije nalazi pokazivač iz konstruktora. Zatim je tu metodu u kojoj se nalazi pokazivač potrebno pozivati unutar metodi gdje je potreban ispis i brisanje sa zaslona.

4.1.Razvoj izbornika

Programski kod za izbornik je pisan u dvije datoteke a to su GUI.h i GUI.cpp. Kako bi se izbornik i igre pravilno prikazali na zaslonu potrebno je uključiti datoteku za odabrani zaslon, a ta biblioteka je ILI9431_STM32.h[16]. Kako bi se mogle koristiti funkcije koje se nalaze unutar Arduino platforme potrebno je i uključiti datoteku Arduino.h[17]. Biblioteku Arduino.h je uvijek potrebno uključiti kako bi sve funkcije i makroi arduino platforme ispravno radile.

Unutar datoteke GUI.h odnosno *header* datoteke se nalaze deklaracije metoda, te metode su metoda vezana za tipke na zaslonu i metoda za kursor. Unutar GUI.cpp je prvo potrebno uključiti *header* datoteku screen.h kako bi se moglo ispisivati i brisati sa zaslona ono što je u programskom kodu namijenjeno da se ispisuje na zaslon. Metode koje se nalaze unutar datoteke GUI.cpp su definicije spomenutih metoda deklariranih unutar GUI.h datoteke. To su definicija metode za tipke i definicija metode za kursor. Korisnik se izbornikom služi pomoću *joysticka. Joystick* na LCD TFT zaslonu je prikazan pomoću kursora odnosno trokuta, korisnik kursorom upravlja pomoću *joysticka*. Upravljanje *joystickom* u programskom smislu se nalazi unutar GUI.cpp datoteke unutar metode za kursor. Unutar metode za kursor je kreirano da program čita vrijednosti sa analognih izvoda na *joysticku* što je i označeno na slici 4.2. , kako korisnik upravlja gljivom *joysticka*, kursor na zaslonu se pomiče ovisno o očitanoj vrijednosti. Mikroupravljač čita analogne vrijednosti sa analognih pinova *joysticka* kao što je i označeno na slici 4.2.



```
void Button::cursor(Adafruit_ILI9341 &lcd, int draw_cursor) // metoda pomocu
{
    Serial.begin(115200);
    int x1 = 0;
    int x2 = 319;
    int y1 = 0;
    int y2 = 239;
    int width1 = abs(x1 - x2); //racunanje sirine kursora
    int he = abs(y1 - y2); //racunanje visine kursora
    if (draw_cursor) // ako je zastavica draw_cursor postavljena na true( stc
    {
        int rawX = 1023 - analogRead(A0); // ocitavaj x os
        int rawY = 1023 - analogRead(A1); // ocitavaj y os
        xp += (511 - rawX) / 100;
        yp += (511 - rawY) / 100;
```

Slika 4.2. Prikaz programskog koda za očitavanje analognih vrijednosti sa odabranog *joysticka*

Pročitana analogna vrijednost sa *joysticka* u smjeru horizontalne osi može biti u rasponu vrijednosti od 0 do 1023, to također vrijedi za vertikalnu os. Ako korisnik gljivu *joysticka* drži u lijevu stranu očitane vrijednosti sa *joysticka* su 1023, dok ako korisnik drži gljivu u desnu stranu vrijednosti su od 0 do 5. Kako bi se kursor po zaslonu pravilno kretao očitane analogne vrijednosti treba oduzeti od broja 511 što je i označeno na slici 4.11. Broj 511 označava kad korisnik ne dira gljivu *joysticka*, odnosno tad kursor na zaslonu stoji. Oduzimanjem očitane vrijednosti od 511 dobijemo smjer kretanja kursora dakle da li će kretanje kursora biti u lijevu ili u desnu stranu. Unutar datoteke GUI.cpp se i nalazi definicija metode za tipke koje će se prikazivati na LCD TFT zaslonu. Unutar definicije je kreiran pravokutnik i tekst koji će se nalaziti unutar tog pravokutnika. Tekst reprezentira odabir u izborniku, tekst je kreiran tako da bude pozicioniran na sredinu tog pravokutnika. Poziciju je potrebno izračunati po horizontalnoj i po vertikalnoj osi.

```
int duljinatexta = strlen(mytext) * 6 * _textscale;
int xpos = (w / 2) + x - (duljinatexta / 2);
int ypos = (h / 2) + y - (_textscale * 8 / 2);
```

Slika 4.3. Prikaz programskog dijela koda pomoću kojega se računa duljina teksta i pozicija na kojoj će biti smješten tekst unutar kreiranog pravokutnika

Na slici 4.3. je prikazan način na koji se računa duljina teksta koji će se nalaziti unutar kreiranog pravokutnika. Kako bi se izračunala duljina upisanog teksta koristi se gotova funkcija *strlen* koja se nalazi u osnovnoj biblioteci C++ jezika.

U ovom slučaju nakon što se izračuna duljina teksta, tu duljinu treba prilagoditi ugrađenom fontu u AdafruitGFX *library-u*[18]. Taj font je 6 (širina) x 8 (visina) piksela, dakle izračunatu duljinu treba još skalirati. Za skaliranje je potrebno dobivenu duljinu teksta pomnožiti sa 6, 6 je duljina jednog znaka unutar dobivene duljine. Nakon što se dobivena duljina pomnoži sa 6 potrebno je još pomnožiti sa *_textscaleom* odnosno veličinom teksta. Nakon skaliranog (engl.*scale*) teksta potrebno je i tekst pozicionirati unutar kreiranog pravokutnika. Tekst se pozicionira po horizontalnoj i po vertikalnoj osi, tako da tekst bude u sredini kreiranog pravokutnika. Na slici 4.3. je prikazan izračun pozicioniranja. Pozicija teksta po horizontalnoj osi se računa tako da se širina pravokutnika podijeli sa 2, kako bi se odredila njegova sredina, zatim je potrebno dobivenu sredinu zbrojiti sa početnom pozicijom pravokutnika po horizontalnoj osi i dobivenom duljinom teksta podijeljenu sa 2.

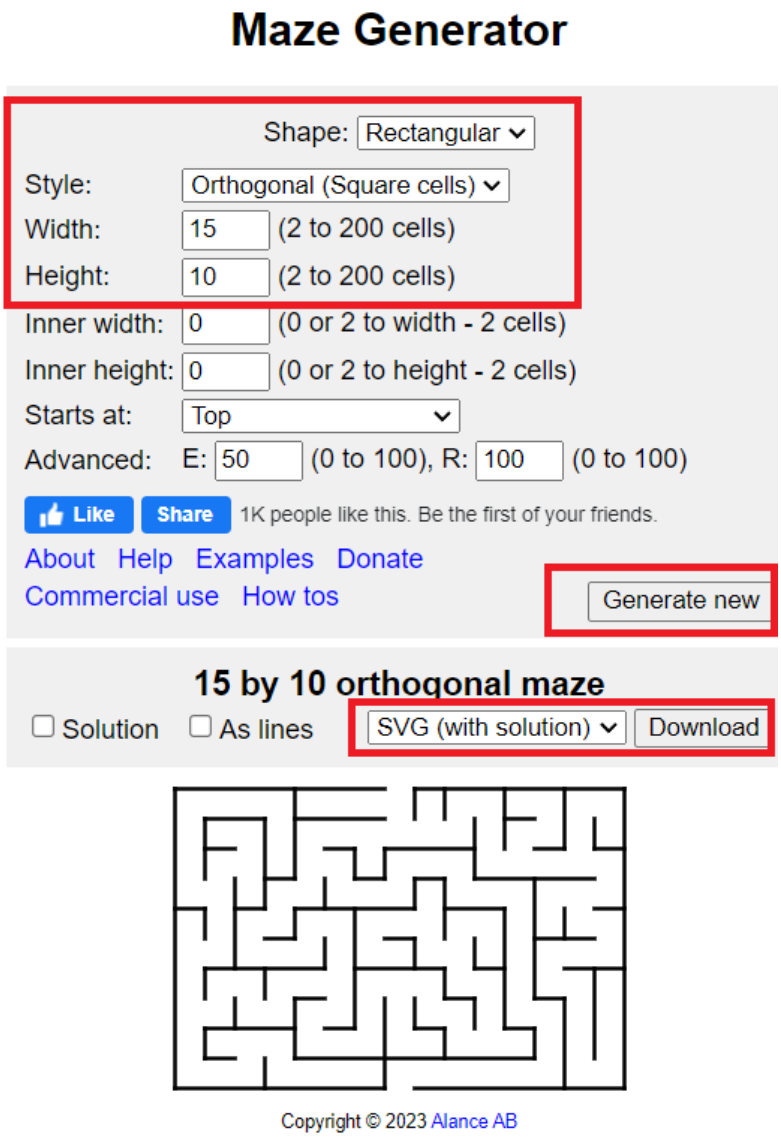
Pozicija teksta po vertikalnoj osi se računa tako da se prvo kreirana visina pravokutnika podijeli sa 2, zatim se rezultat dijeljenja zbroji sa početnom pozicijom iscrtavanja pravokutnika po vertikalnoj osi. Zatim se dobivenom zbroju još pridoda *_textscale* pomnožen sa 8 i podijeljen sa 2. *_textscale* se množi sa 8 zbog toga što je visina svakog znaka u tekstu jednaka 8. Unutar metode za kreiranje tipki se nalazi i ispitivanje kolizije između kursora i kreirane tipke. Metodu za tipke i za kursor je potrebno pozvati u glavnom kodu. Glavni kod iz kojeg se pokreće igraća konzola se nalazi unutar datoteke Igra.ino. Na početku datoteke Igra.ino je potrebno uključiti datoteku GUI.h te ostale biblioteke i datoteke.

Izbornik u Igra.ino je kreiran pomoću *switch case* slučajeva. *Switch case* u ovom slučaju je napravljen kao automat sa konačnim brojem stanja. Dakle unutar *switch case* slučaja se ispituje da li je došlo do kolizije između kreirane tipke i kursora te ako je to istina, tada se *switch case* prebacuje na sljedeći slučaj koji je definiran unutar uvjeta koji se ispituje. Nulti slučaj je prikaz glavnog izbornika odnosno tipaka i kursora u glavnom izborniku. Unutar nultog slučaja je potrebno pozvati metodu pomoću koje se iscrtavaju tipke u glavnom izborniku, također je potrebno pozvati i metodu za kursor. Ako se metoda za kursor ne pozove, kursor se neće ispisati na LCD TFT zaslon. Metodu za kursor potrebno je pozivati unutar svakog slučaja koji se odnosi na izbornik.

4.2.Razvoj igre labirint

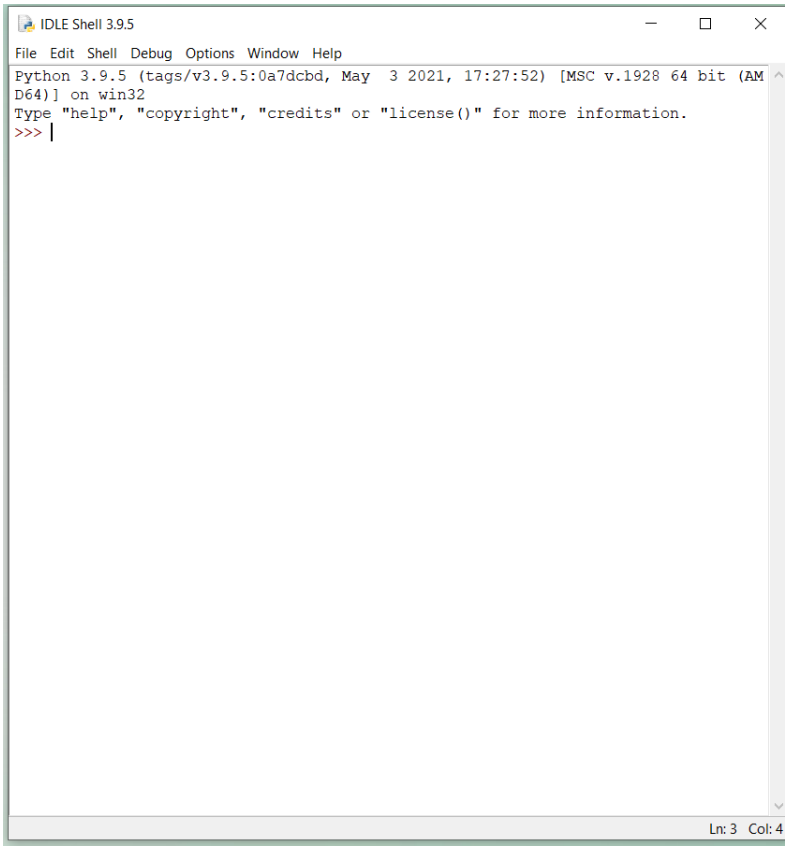
4.2.1. Razvoj igre labirint unutar Python IDLE okruženja

Razvoj igre labirint (engl.*maze*) je kreiran i raspodijeljen u više različitih datoteka. Za razvoj igre labirint (engl.*maze*) u programskom smislu prvo je bilo potrebno generirati labirint. Svaki labirint unutar igre je kreiran pomoću maze generator-a. *Maze Generator* je stranica na internetu koja služi za generiranje labirinata različite duljine i širine [19].



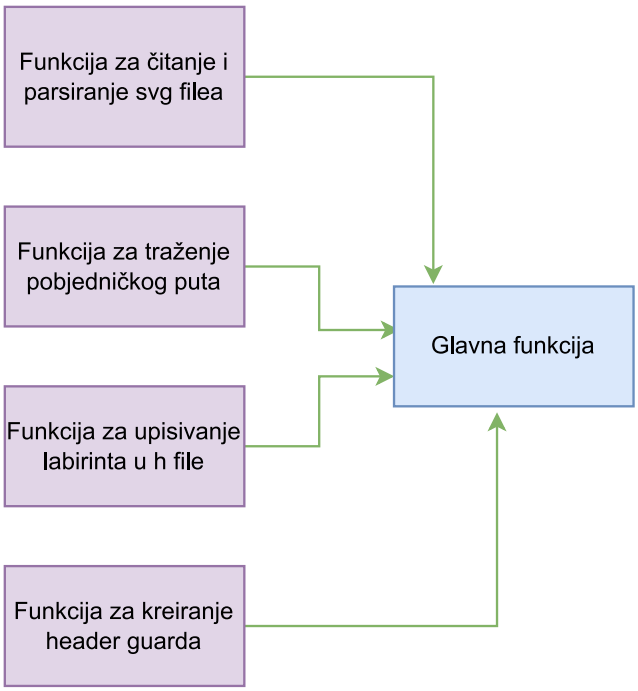
Slika 4.4. Na slici je prikazan Maze Generator[19]

Na slici 4.4. su označene postavke koje treba podesiti za generiranje labirinta. Kao što je i označeno treba odabrati oblik, stil, te visinu i širinu labirinta. Nakon što su postavke odabrane potrebno je odabrati i format pomoću kojega će se iscrtati željeni labirint, u ovom slučaju je korišten SVG format sa rješenjem[19]. SVG (engl.*Scalable Vector Graphic)* je format koji se koristi za prikaz vektorske grafike za web. Dok SVG format sa rješenjem omogućava i prikaz rješenja za uspješan izlazak iz labirinta. Labirint u SVG formatu sa rješenjem je potrebno *download-ati* i spremiti u obliku svg formata što je i prikazano na slici 4.4. Nakon što je labirint spremljen na računalo, labirint je potrebno prilagoditi odabranom LCD TFT zaslonu iz razloga kako bi se izgenerirani labirint pravilno iscrtao na zaslon. Ukoliko se labirint ne prilagodi zaslonu postoji mogućnost da se linije labirinta iscrtaju van zaslona i onda te linije korisnik ne vidi tijekom igre. Za *scale-anje* odnosno prilagođavanje labirinta zaslonu je odabrano da se napiše programski kod u *Python* programskom jeziku, odnosno skripta. Skripta za *scale-anje* je kreirana unutar IDLE (Python 3.9. 64-bit) okruženja koje je i prikazano na slici 4.5.



Slika 4.5. Prikaz IDLE(Python 3.9.5.) programskog sučelja

Spremljeni labirint u svg formatu je potrebno otvoriti i *scale-ati* unutar IDLE *Shell* okruženja. *Scale-anje* odnosno prilagođavanje labirinta zaslonu je raspodijeljeno u pet funkcija pisane *Python* programskim jezikom. Funkcije su prikazane na slici 4.6.



Slika 4.6. Prikaz međusobne povezanosti funkcija u *python* skripti

Prva funkcija sa slike 4.6. je funkcija za čitanje i parsiranje svg datoteke, dakle ta funkcija čita svg datoteku i parsira ju u obliku liste. Druga funkcija otvara svg file i traži ključne riječi kako bi našla početnu i završnu točku u labirintu. Zatim od se pronađenih završnih točaka kreira završna linija u labirintu. Nakon toga slijedi funkcija koja će upisati skalirani labirint u *header* datoteku. Na kraju je potrebno kreirati *header guard* iz razloga što se datoteka sa prilagođenim labirintom koja će biti *header* datoteka uključuje u C/C++ programski jezik.

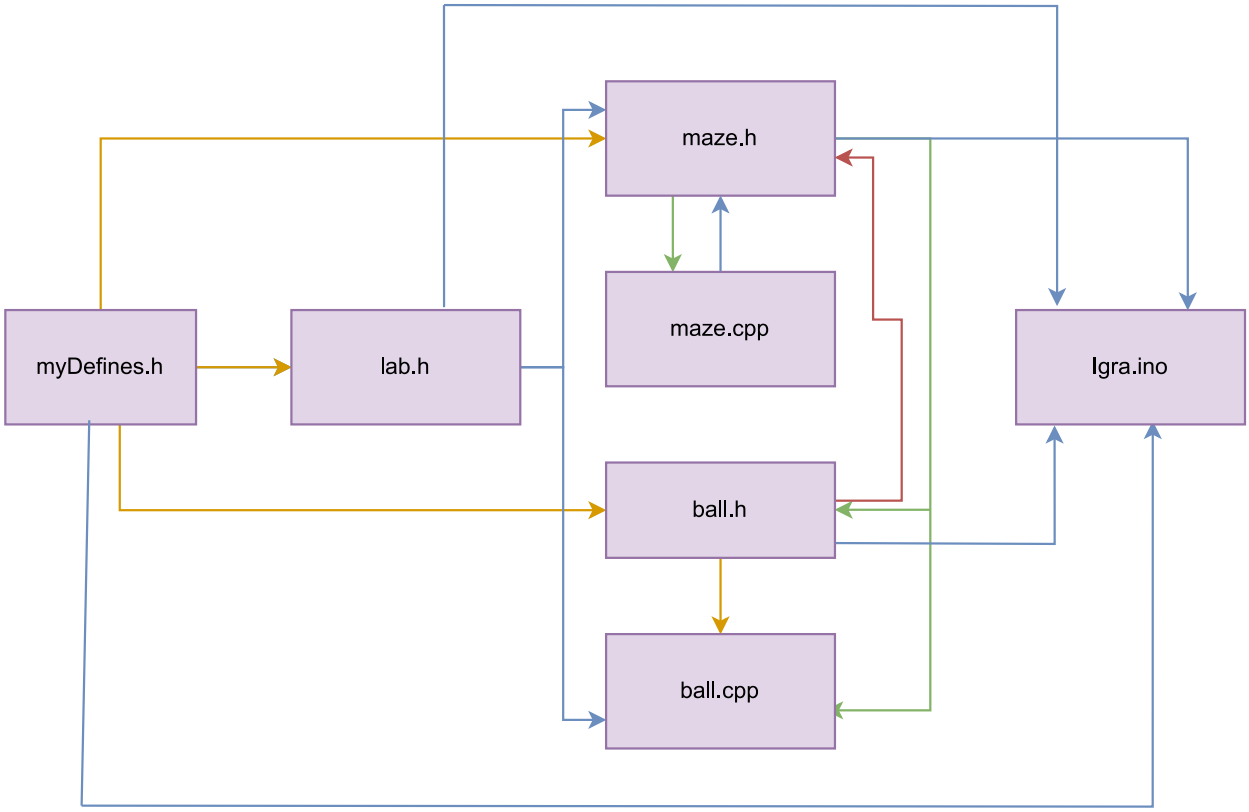
Nakon što su u *pythonu* napisane četiri funkcije sa slike 4.6, sve četiri funkcije je potrebno pozvati u glavnoj funkciji kako bi se one izvršile. Nakon što se program pokrene, program traži od korisnika unos koliko želi labirinata, kako želi da mu se zove labirint i gdje želi da mu se spremi skalirani labirint kao što možemo i vidjeti na slici 4.7.

```
*IDLE Shell 3.9.5*
File  Edit  Shell  Debug  Options  Window  Help
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Marina\Desktop\Labirintevi za igricu\Scale maze.py =====
Upisi koliko zelis imati labirinteva 5
Upisi gdje zelis da ti se spremi skalirani labirint lab.h
Upisi svg file Lab1.svg
Upisi naziv labirinta: lab1Lines
Upisi naziv podatka lab1
```

Slika 4.7. Na slici je prikazano što program traži da korisnik unese nakon što se pokrene programski kod

4.2.2. Razvoj igre labirint unutar Visual Studio Code-a

Nakon što je labirint izgeneriran i skaliran te spremljen u *header* datoteku, razvoj igre labirint se može nastaviti razvijati unutar *Visual Studio Code-a*. Razvoj igre labirint u *Visual Studio Codeu* je raspodijeljen u sedam datoteka. Navedene datoteke su: lab.h, maze.h, maze.cpp, ball.h, ball.cpp, myDefines.h i Igra.ino što možemo i vidjeti u dijagramu međusobne povezanosti koji je prikazan na slici 4.8.



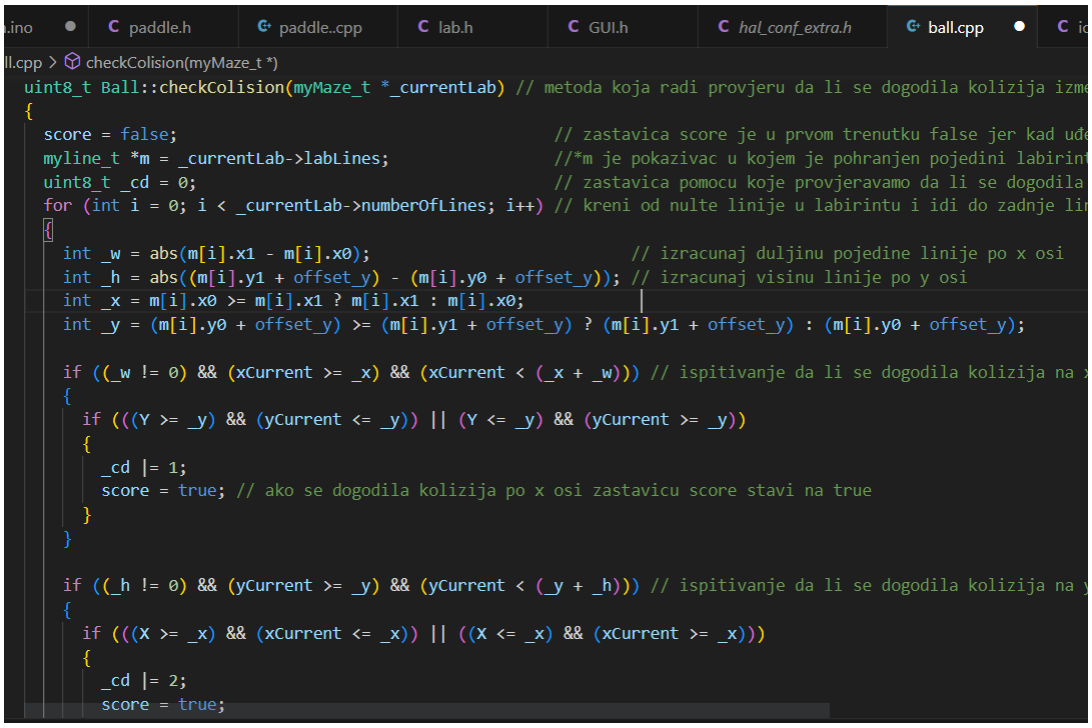
Slika 4.8. Prikaz međusobne povezanosti biblioteka u igri labirint

Prva datoteka je lab.h to je *header* datoteka unutar koje se nalaze labirintevi u obliku polja koji će se prikazivati na odabranom LCD TFT zaslonu. myDefines.h je datoteka unutar koje se nalaze kreirani tipovi podataka potrebni za opisi labirinta. Maze.h je *header* datoteka unutar koje se nalaze metode vezane za labirint, a to su metoda za iscrtavanje labirinta na zaslon, metoda za izračun slučajnog (engl. *random*) broja. Dobiveni slučajni broj je broj pod kojim se nalazi jedan od labirinata, zatim metoda za odabir težine igre. Datoteka maze.cpp sadrži sve definicije deklariranih metoda koje se nalaze unutar datoteke maze.h. Ball.h je *header* datoteka unutar koje je potrebno uključiti biblioteku SparkFunLSM6DS3.h [20], navedenu biblioteku je potrebno uključiti kako bi se mogao koristiti odabrani akcelerometar.

Unutar ball.h se nalaze metode potrebne za ispis i kretanje kuglice, metoda za provjeravanje kolizije između kuglice i linija u labirintu, metoda za iscrtavanje izlazne linije iz labirinta, metoda za provjeravanje kolizije izlazne linije i kuglice, i sve metode pomoću kojih se oblikuju tekstualni objekti u igri. Unutar datoteke ball.cpp je potrebno uključiti biblioteke vezane za odabrani zaslon, akcelerometar, i *header* datoteke što je i plavom bojom označeno na slici 4.10.

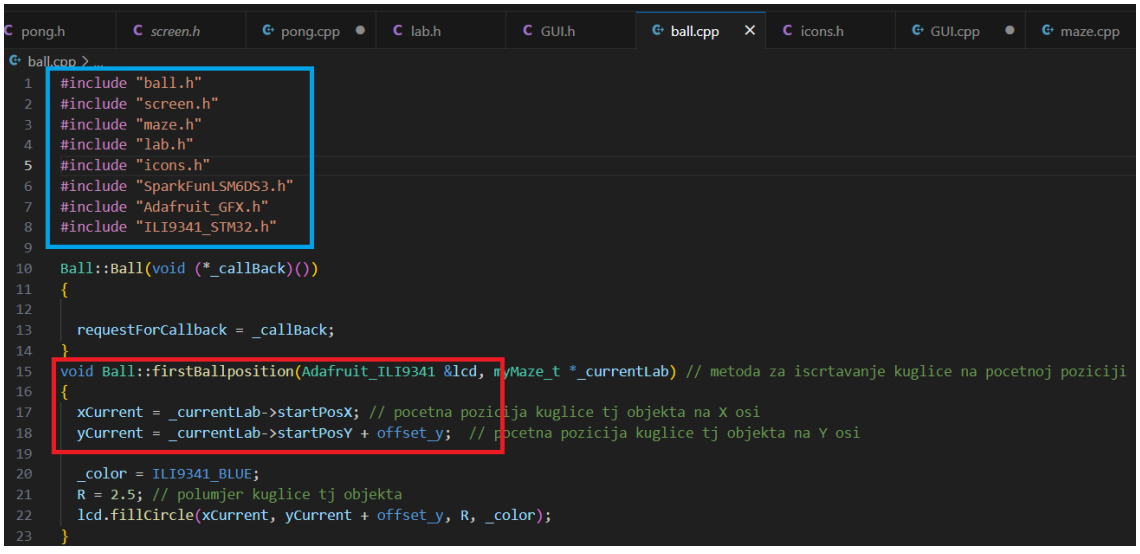
Datoteka ball.cpp sadrži sve definicije metoda koje su deklarirane unutar ball.h *header* datoteke. Izbornik i svaka kreirana igra se pokreću iz datoteke Igra.ino. Unutar datoteke Igra.ino se nalaze još tri metode potrebne za prikaz igre labirint na LCD TFT zaslonu. Te tri metode su metoda za tijek vremena u igri, metoda za iscrtavanje zaslona koji će se prikazivati na zaslonu nakon što korisnik dođe na izlaznu liniju u labirintu, te metoda *GameLoop* unutar koje se nalaze metode koje se trebaju stalno ponavljati i prikazivati na LCD TFT zaslonu tijekom igranja igre.

Najbitniji dijelovi koda unutar metodi potrebnih za razvoj igre labirint je dio koda potreban za provjeravanje kolizije između kuglice i linije unutar labirinta, i dio koda vezan za upravljanje akcelerometrom. Navedeni dio koda se nalazi unutar metode za provjeravanje kolizije između kuglice i labirinta a ta metoda se nalazi unutar datoteke ball.cpp a to je i prikazano na slici 5.9. Provjeravanje kolizije je kreirano tako da se prvo izračuna visina i širina svake linije u polju odnosno labirintu. Zatim se pomoću uvjeta ispituje da li je kuglica po horizontalnoj po vertikalnoj osi svoje pozicije dotaknula liniju unutar labirinta te da li se ta kolizija dogodila sa lijeve ili desne strane te linije. Ako su oba uvjeta ispunjena kolizija se dogodila, prikaz događanja kolizije je kreiran tako da se zastavica za koliziju prebacuje sa 0 na 1 odnosno sa laži na istinu. Zastavicu prije ispitivanja kolizije je potrebno podesiti na 0. Unutar uvjeta za ispitivanje kolizije se nalazi i zastavica vezana za uspjeh u igri (engl.*score*), dakle ako je uvjet za koliziju ispunjen zastavica za uspjeh je jednaka istini, i uspjeh u igri korisniku se smanjuje za određeni broj.



Slika 4.9. Programski dio koda unutar kojeg se nalazi ispitivanje kolizije između kuglice i pojedine linije u labirintu

Dio koda koji je također bitan je vezan uz odabrani akcelerometar. Kako bi korisnik mogao igrati igru odnosno kako bi mogao upravljati kuglicom u labirintu, potrebno mu je omogućiti upravljanje. Prvo se kuglica postavi na početnu poziciju po horizontalnoj osi odnosno po vertikalnoj osi unutar metode za iscrtavanje kuglice koja se zove *firstBallposition* što je i prikazano na slici 4.10.



Slika 4.10. Programski dio koda unutar kojeg su označene biblioteke koje je potrebno uključiti, te dio koda za postavljanje kuglice na početnu poziciju

Postavljanje početne pozicije kuglice po horizontalnoj osi i po vertikalnoj osi je prikazano na slici 4.10. Nakon što je kuglica postavljena na početak labirinta potrebno je omogućiti da se kuglica pomjera pomoću akcelerometra. Način kako omogućiti da se kuglica pomjera je da se čitaju vrijednosti sa SDA i SCL linije na akcelerometru, zatim te očitane vrijednosti pomnože sa 5, kako bi vrijednost ubrzanja bila veća. Zatim izračunatim vrijednostima za ubrzanje po treba pridružiti pozicije kuglice po horizontalnoj osi i po vertikalnoj osi. Prethodno opisano gibanje kuglice napisano u programskom kodu je prikazano na slici 4.11.

```
void Ball::updateBallposition(Adafruit_ILI9341 &lcd, LSM6DS3 myIMU)
{
  // varijable za akceleraciju kuglice
  float rawX = myIMU.readFloatAccelX();   //citanje vrijednosti pozicije akcelerometra po x osi
  float rawY = myIMU.readFloatAccelY();     //citanje pozicije akcelerometra po y osi


  velX = rawX * 5;
  velY = rawY * 5;

  X = xCurrent; //X je nova pozicija kuglice, novoj poziciji kuglice po x osi pridruzi staru pozic
  Y = yCurrent;

  xCurrent += velX;  //pokretanje kuglice pomocu akcelerometra po x osi
  yCurrent += velY;  //pokretanje kuglice pomocu akcelerometra po y osi


  lcd.fillCircle(xCurrent, yCurrent, R, _color); // ispisivanje kuglice na poetnoj poziciji, tu mo
  updateScreen();                                // pozivanje funkcije za update screena koja nam
}
```

Slika 4.11. Programski dio koda unutar kojeg se očitavaju vrijednosti sa akcelerometra i prikaz načina na koji je izvedno da se kuglica pomiče

4.3.Razvoj igre pong

Programski kod za igru pong je raspodijeljen u četiri datoteke unutar *Visual Studio Code-a*. Prva datoteka je *header* datoteka myDefines.h unutar koje se nalaze tipovi podataka koji su potrebni za kreiranje objekata u igri. Druga datoteka je *header* datoteka pong.h, unutar te datoteke prvo je potrebno uključiti tri biblioteke. Prva bilioteka je vezana za odabrani LCD TFT zaslon, ta biblioteka je ILI9341_STM32.h, druga biblioteka je vezana za akcelerometar SprakFunLSM6DS3.h [20]. Tu bliblioteku je potrebno uključiti kako bi korisnik mogao pomicati kuglicu u igri. I treća datoteka je Arduino.h[17]. Unutar *header* datoteke pong.h se nalaze i sve deklaracije metoda potrebnih za igru pong, a to su metoda za iscrtavanje prvog reketa (engl. *paddle*), drugog reketa (engl. *paddle*), metoda za kretanje reketa, metoda za iscrtavanje kuglice i kretanje kuglice, metoda za ispitivanje kolizije između kuglice i reketa, metoda za prikaz života u igri, metoda za oduzimanje života u igri, metoda za ispitivanje da li je došlo do kraja igre.

Treća datoteka je pong.cpp unutar koje je potrebno uključiti *header* datoteku screen.h kako bi se objekti igre mogli pravilno iscrtavati na zaslon i brisati sa zaslona. Datoteka pong.cpp sadrži definicije svih metoda deklariranih unutar pong.h *header* datoteke. Te zatim zadnja četvrta datoteka je Igra.ino iz koje se i pokreće kreirana igra, međutim unutar Igra.ino se nalaze još dvije funkcije vezane za igru pong. To su funkcija unutar koje se ponavljaju određene metode deklarirane unutar pong,h datoteke. To su metode koje se trebaju prikazivati tijekom cijelog vremena igranja igre, i funkcija koja kreira zaslon koji će se prikazati nakon što korisnik izgubi u igri sve živote. Bitniji dijelovi koda za kreiranje igre pong su dio za kretanje reketa s kojim upravlja korisnik što je i prikazano na slici 4.12.

```
void Pong::movePaddle1(Adafruit_ILI9341 &lcd, LSM6DS3 myIMU, myPaddle_t1 *p1) //veslo kojim upravlja korisnik
{

    float rawX = myIMU.readFloatAccelX(); //citanje vrijednosti akcelerometra po x osi
    float rawY = myIMU.readFloatAccelY();

    brzinax = rawX * 10; //brzina kretanja vesla po x osi, ocitanu vrijednost akceleracije pomnozi sa 10
    brzinay = rawY * 10;
    Y0 = p1->sy1;

    if (p1->sy1 <= 0) //ako je pozicija vesla manje ili jednako 0 odnosno ako je na rubu zaslona
    {
        p1->sy1 = 0; //pocetnu tocku iscrtavanja po y os vesla stavi na 0
    }

    if (p1->sy1 >= 190) //ako je pozicija vesla po y osi vece od 190
    {
        p1->sy1 = 190; //pocetnu tocku iscrtavanja vesla po y osi stavi da je jednaka 190, iz razloga sto mu je duljina 50
    {
    }
    p1->sy1 += brzinay; //poziciji vesla po y osi dodaj ubrzanje
    lcd.fillRect(p1->sx1, p1->sy1, p1->w, p1->h, _colorpaddle);
    updateScreenGame1();
}
```

Slika 4.12. Prikaz programskog dijela unutar kojeg je izvedeno da se veslo s kojim upravlja korisnik pomjera

Na slici 4.12. je prikazano kako se očitavaju vrijednosti sa akcelerometra. Zatim nakon očitanih vrijednosti, računa se ubrzanje, a to je kreirano tako da se očitane vrijednosti sa akcelerometra pomnože sa 10. Nakon što je ubrzanje izračunato potrebno je onemogućiti da se reket iscrtava van zaslona kao što je i prikazano na slici 4.12.

Na slici 4.13. je prikazan dio koda za ispitivanje kolizije između vesla i kuglice.

```
uint8_t Pong::checkCollisionPaddle(myPaddle_t1 *p1, myPaddle_t2 *p2, my_ballPong *b)
{
    uint8_t collision = 0;
    score = false;
    score2 = false;

    if (((b->startBallx - b->r) <= (p1->sx1 + p1->w)) && ((b->startBallx - b->r) > 0) && (b->startBally >= p1->sy1) && (b->startBally <= (b->startBally + p1->h)))
    {
        collision = 1;
        //Serial.println("KOLIZIJA!");
    }

    if (((b->startBallx + b->r) >= (p2->sx2)) && ((b->startBallx + b->r) > 0) && (b->startBally >= p2->sy2) && (b->startBally <= (b->startBally + p2->h)))
    {
        collision = 3;
    }
```

Slika 4.13. Dio programskog koda koji ispituje koliziju između kuglice i reketa sa kojim upravlja korisnik, i koliziju između kuglice i reketa sa kojim upravlja računalo

Prije ispitivanja uvjeta potrebno je zastavicu za koliziju staviti na 0, što označava da se kolizija u prvom trenutku nije dogodila. Prvi označeni uvjet se odnosi na koliziju između kuglice i reketa s kojim upravlja korisnik pomoću akcelerometra.

Uvjet je kreiran tako da se prvo usporedi da li je pozicija kuglice po horizontalnoj osi umanjena za njen polumjer manja od početne točke iscrtavanja reketa po horizontalnoj osi zbrojen sa njegovom širinom. Drugi uvjet je usporedba ako je pozicija kuglice po horizontalnoj osi umanjena za njen polumjer veća od 0. Treći uvjet je usporedba da li je pozicija kuglice veća ili jednaka od pozicije reketa po vertikalnoj osi, i da li je pozicija kuglice po vertikalnoj osi manja ili jednaka od pozicije kuglice po vertikalnoj osi zbrojena sa visinom reketa. Ako su svi navedeni uvjeti ispunjeni dogodila se kolizija, odnosno zastavica za koliziju je jednaka 1. Uvjet za reket sa kojim upravlja računalo ima iste uvjete kao i reket sa kojim upravlja korisnik. Na slici 4.14. su prikazani uvjeti sa kojima se ispituje da li je kuglica dotaknila rub sa lijeve strane zaslona i sa desne strane zaslona.

```
    if (((b->startBallx - b->r) <= (p1->sx1)) && ((b->startBally - b->r) <= 240) && (b->startBally >= 0))
    {
        collision = 2;
        score = true;
        //Serial.println("Game over");
    }


    if (((b->startBallx + b->r) >= 320) && ((b->startBally + b->r) <= 240) && (b->startBally >= 0))
    {
        collision = 4;
        score2 = true;
        //Serial.println("Game over");
    }

    return collision;
}
```

Slika 4.14. Programski kod za ispitivanje kolizije između kuglice i rubova zaslona

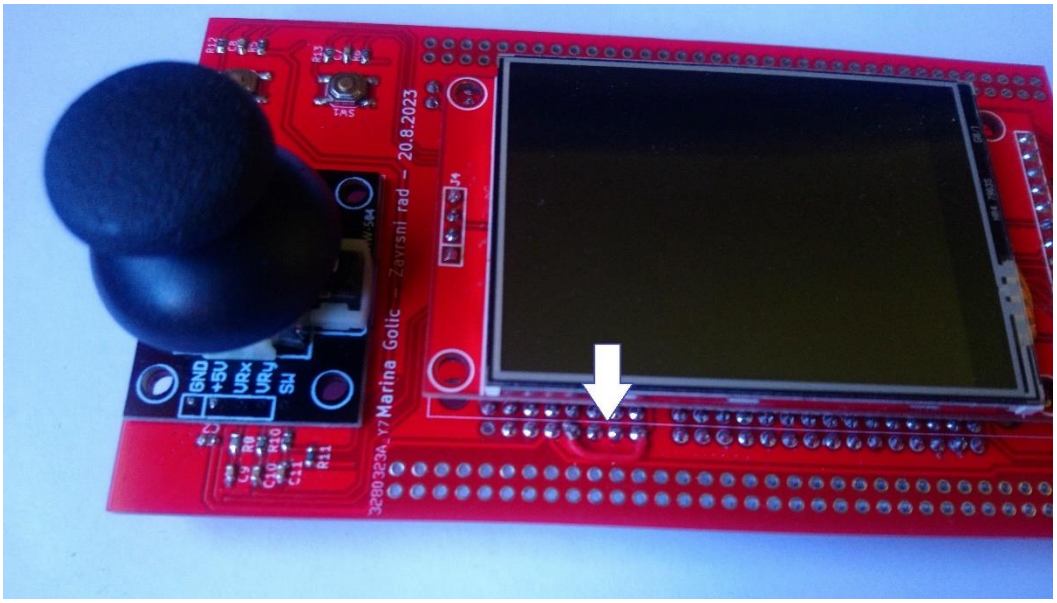
Način kako se kreće kuglica u igri pong je prikazano na slici 4.15. Prvo treba ispitati pomoću uvjeta da li je kuglica dotaknila gornji rub zaslona ili donji rub zaslona, te ukoliko je dotaknila jedan rub, kuglica mijenja smjer kretanja. Smjer kretanja treba promjeniti kako se kuglica ne bi nastavila kretati u istom smjeru prije kolizije sa rubom, ako se ne promjeni smjer kretanja kuglica će otići van zaslona. Isti uvjet treba ispitati i za desni i lijevi rub zaslona te promjeniti smjer kretanja kuglice. Nakon ispitanih uvjeta novoj poziciji kuglice je potrebno pridružiti staru poziciju kuglice, to treba napraviti kako bi se stalno *update-ala* pozicija kuglice, odnosno kako bi se stalno kuglica brisala na staroj poziciji i iscrtavala na novoj poziciji. Ako kuglica nije dotaknila niti jedan rub njen smjer kretanja je jednak početnom smjeru kretanja.

```
pong.cpp > moveBall(Adafruit_ILI9341 &, LSM6DS3, my_ballPong *)

void Pong::moveBall(Adafruit_ILI9341 &lcd, my_ballPong *b)
{
    Serial.begin(115200);

    if (b->startBallx > lcd.width() || b->startBallx < 0)
    {

        ballDirectionX = -ballDirectionX;
    }
    if (b->startBally > lcd.height() || b->startBally < 0)
    {

        ballDirectionY = -ballDirectionY;
    }
    newxb = b->startBallx;
    newyb = b->startBally;
    b->startBallx += ballDirectionX;

    b->startBally += ballDirectionY;

    // if((ballX > p2->sx2) && (ballX < (p2->w + p2->sx2)) && (ballY > p2->sy2) &&

    lcd.fillCircle(b->startBallx, b->startBally, b->r, ILI9341_CYAN);
}
```

Slika 4.15. Prikaz programskog koda koji opisuje način na koji se kuglica kreće u igri Pong

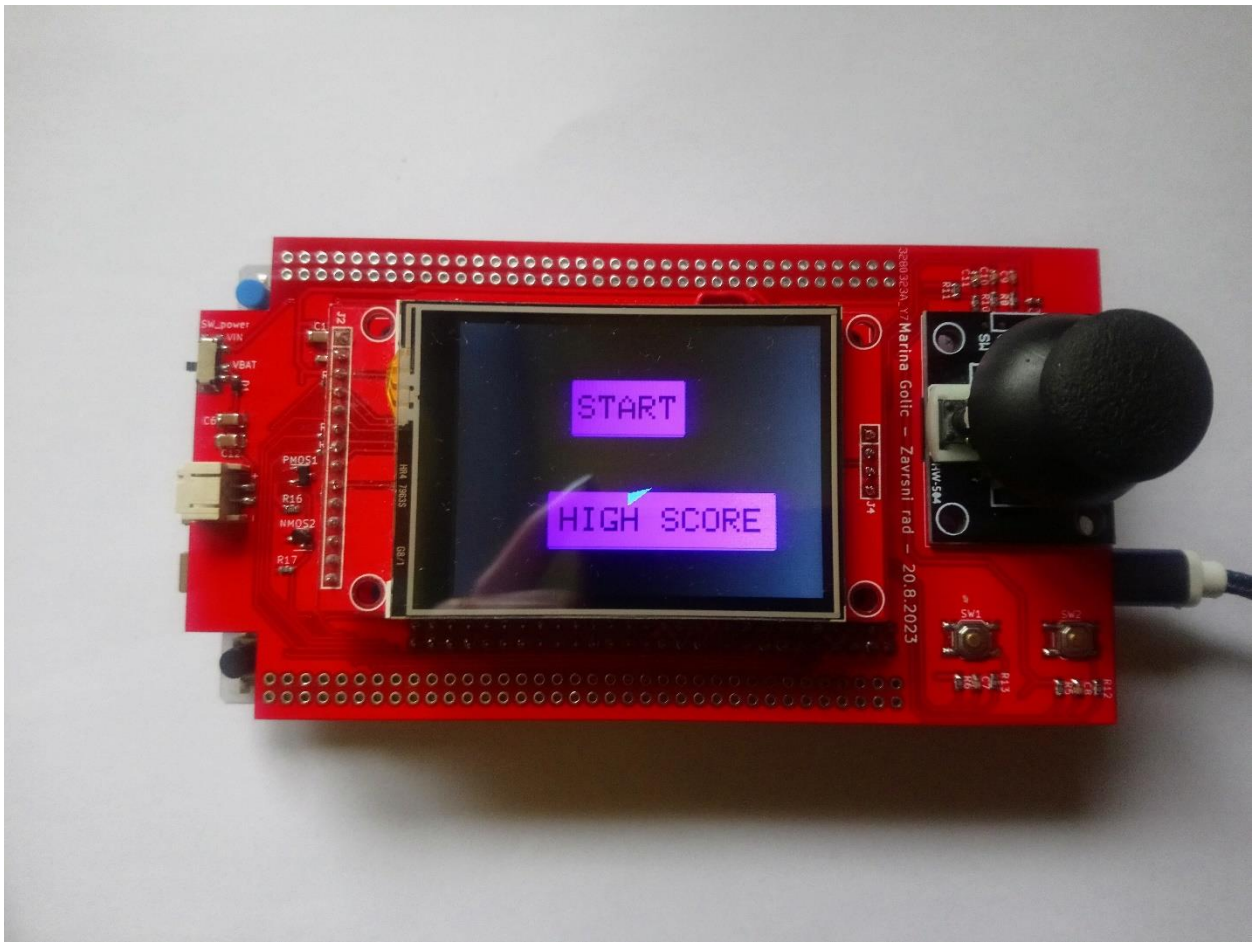
5. PRIKAZ KONAČNOG REZULTATA

Nakon što je dizajnirana tiskana pločica za igraću konzolu i nakon što je napravljena. Nakon što su svi moduli bili zalemljeni na tiskanu pločicu, prilikom odabira da se igraća konzola napaja pomoću baterije uočen je problem na LCD TFT zaslonu. Pozadinsko osvjetljenje na LCD TFT zaslonu nije radilo, naime rješenje problema je da pin sa prekidača ne bude spojen na pin VIN na mikroupravljaču nego na pin 5V. Problem je riješen tako što je na tiskanoj pločici dodana žica koja spaja pin sa prekidača na 5V, što je i prikazano pomoću strelice na slici 5.1.



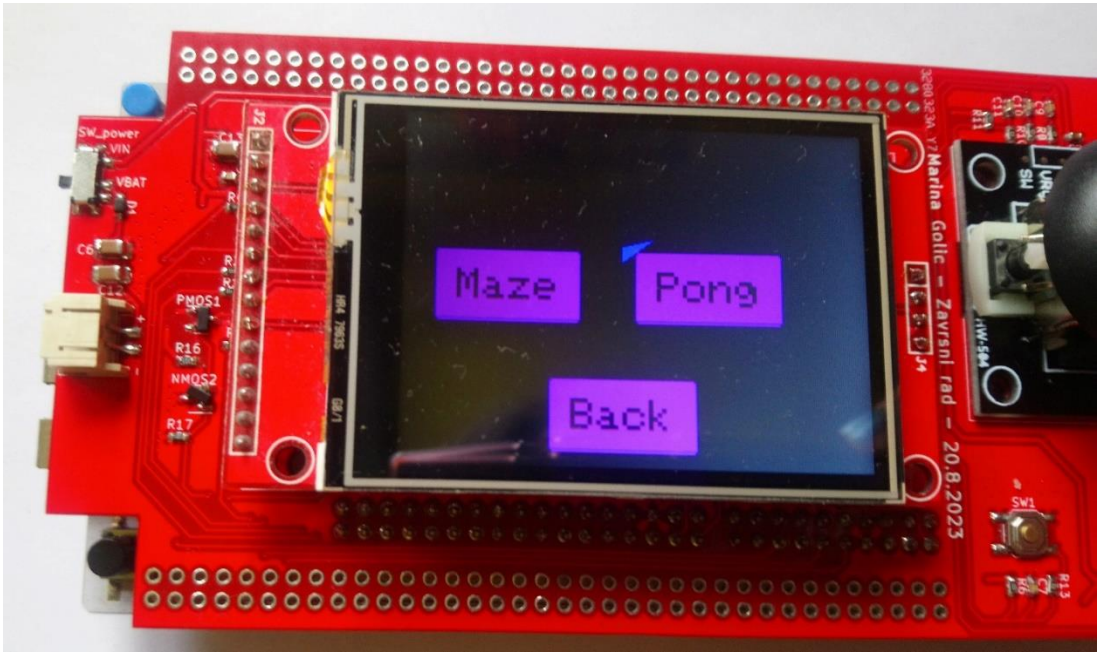
Slika 5.1. Prikaz na koji način je rješen problem

Korisnik pomoću prekidača odabire kako želi napajati igraću konzolu, postoje dva odabira. Prvi odabir da li će konzolu napajati preko litijeve baterije, drugi odabir je da li će konzolu napajati pomoću USB kabela koji može biti uključen u laptop ili neki drugi izvor napajanja. Ukoliko korisnik odabere da se konzola napaja preko baterije potrebno je na razvojnoj pločici prebaciti *jumper* na 5V, ukoliko se konzola napaja preko USB kabela *jumper* na razvojnoj pločici treba prebaciti na STLINK. Nakon što korisnik uključi igraću konzolu prikaže mu se glavni izbornik unutar kojega se nalaze dva izbora što možemo i vidjeti na slici 5.2.



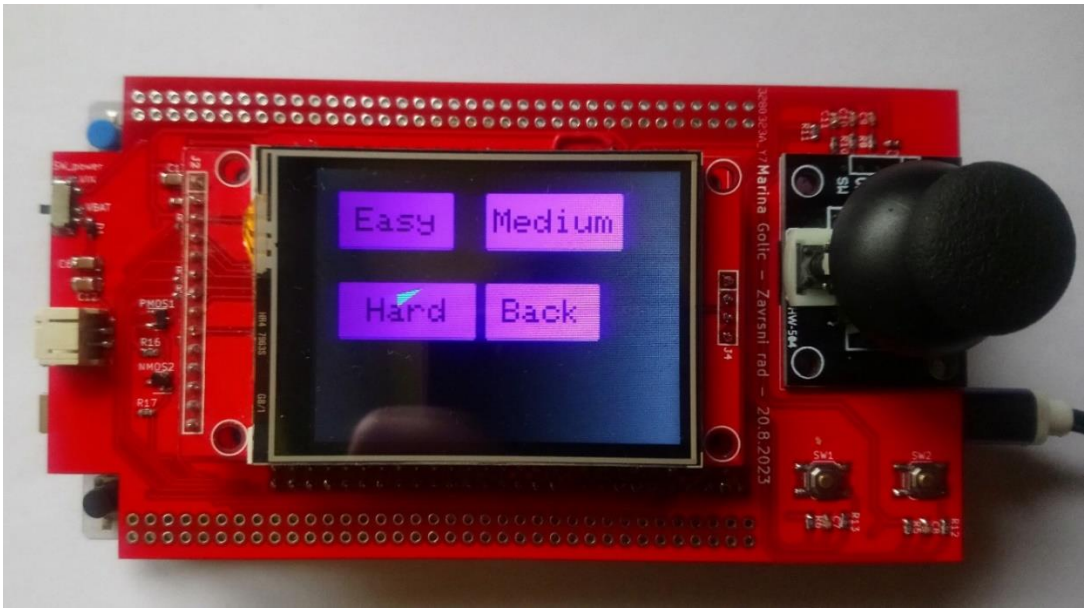
Slika 5.2. Prikaz glavnog izbornika igraće konzole

Navedena dva izbora su da li će ući u izbornik za odabir igre koju želi igrati i drugi izbor je uvid u postignute bodove tijekom igranja igara. Korisnik pomoću *joysticka* odabire jedan od dva izbora. Ako korisnik odabere prvi izbor otvara mu se izbornik unutar kojeg se nalaze tri izbora kao što je i prikazano na slici 5.3.



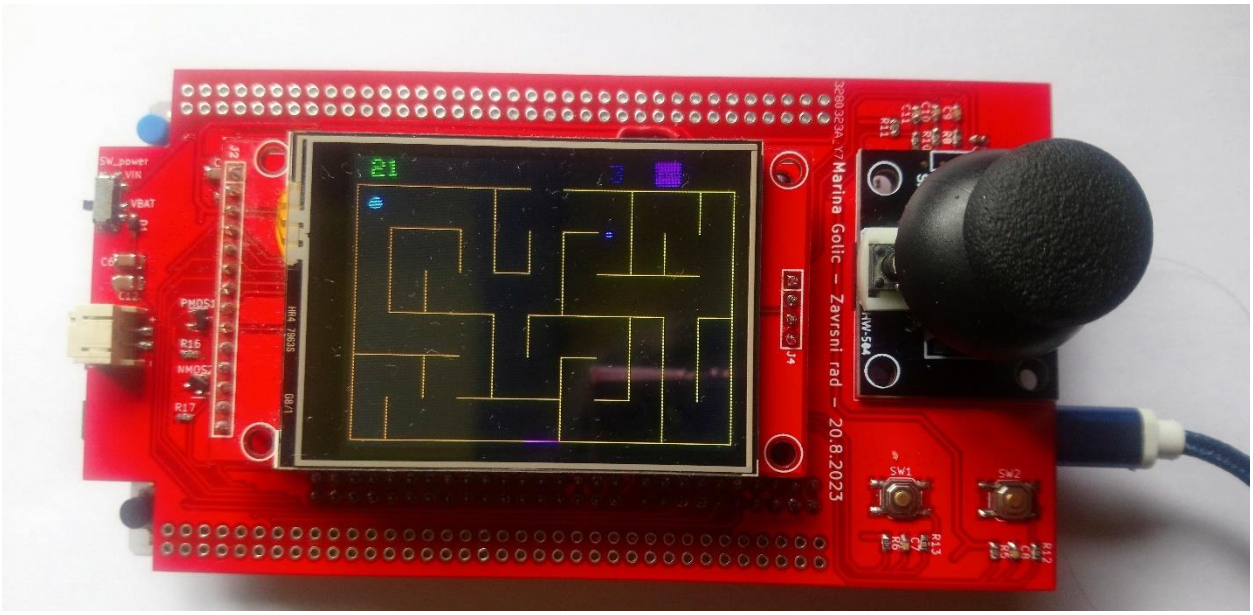
Slika 5.3. Prikaz izbornika sa izborom igara

Prvi izbor je igra labirint, drugi izbor je igra pong i treći izbor je povratak u glavni izbornik. Odabirom igre labirint otvara se ponovno izbornik unutar kojega postoje četiri odabira što je i prikazano na slici 5.4. Tri odabira su vezana za težinu igre, i četvrti je povratak u glavni izbornik.



Slika 5.4. Prikaz izbornika sa izborom težine igre nakon što korisnik klikne na igru labirint

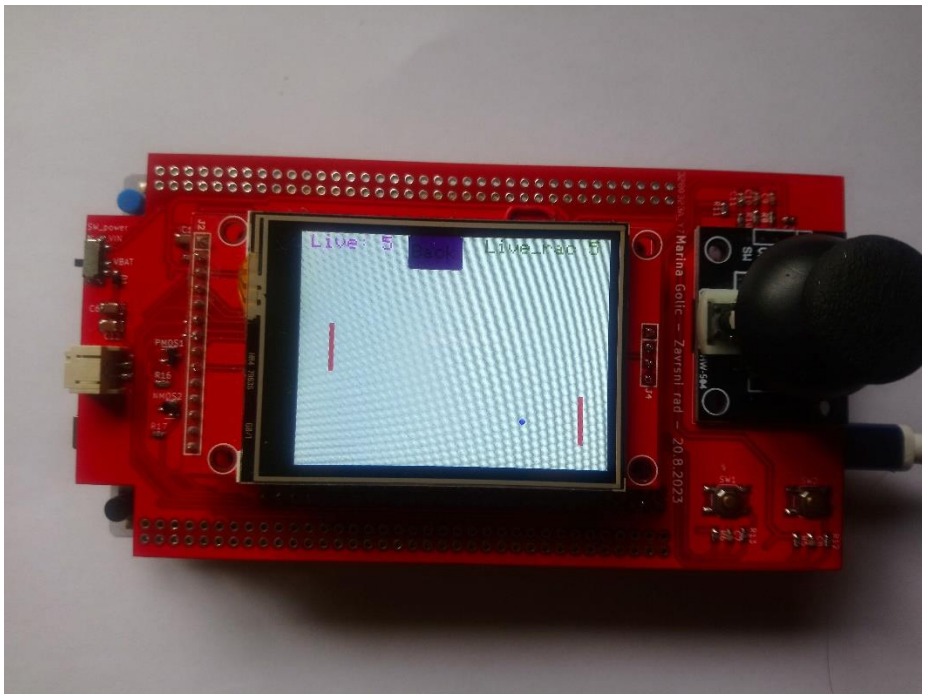
Ako korisnik odabere jednu od tri moguće težine igre otvara se zaslon na kojem se nalazi labirint iz odabrane težine igre (slika 5.5.), na zaslonu se i prikazuje vrijeme u igri i prikaz života u igri, korisnik pomoću akcelerometra LSM6DS3 igra igru.



Slika 5.5. Prikaz igre labirint

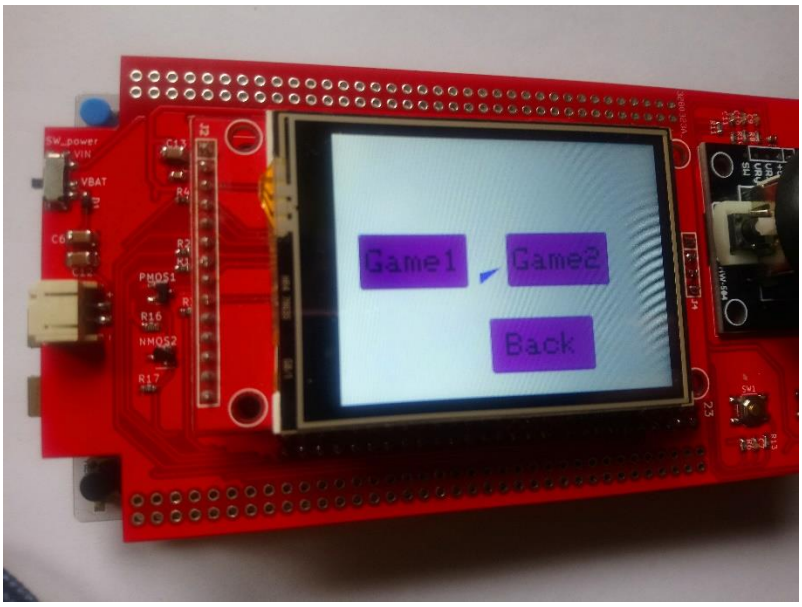
Nakon što korisnik dođe sa kuglicom do izlaza iz labirinta, trenutni labirint se briše, prikazuje se zaslon na kojemu se nalaze informacije o trenutnom broju bodova i trenutni broj života u igri koje ima korisnik, taj zaslon se prikazuje 5 sekundi. Nakon što 5 sekundi istekne, briše se zaslon sa informacijama i ispisuje se novi zaslon na kojem se nalazi novi labirint i kuglica se prikazuje na početku novog labirinta. Nakon što korisnik uspješno prijeđe sve labirinteve u odabranoj težini igre, ponovno se vraća u izbornik sa tri različite težine igre. Tijekom igranja igre vrijeme u igri teče. Korisnik život u igri može izgubiti na dva načina ako mu istekne vrijeme koje je predviđeno da je dovoljno za rješavanje labirinta, te ako korisnik tijekom igranja igre puno puta udari sa kuglicom liniju u labirintu. Ukoliko korisnik izgubi sve živote u igri, igra je završena i ponovno se prikazuje glavni izbornik. Ukoliko korisnik pomoću *joysticka* odabere tipku povratak (engl.*back*) vratit će se u glavni izbornik.

Odabirom na igru pong na zaslonu se prikazuje igra, te prikaz koliko života u igri trenutno ima korisnik, i koliko života u igri trenutno ima računalo što možemo i vidjeti na slici 5.6.



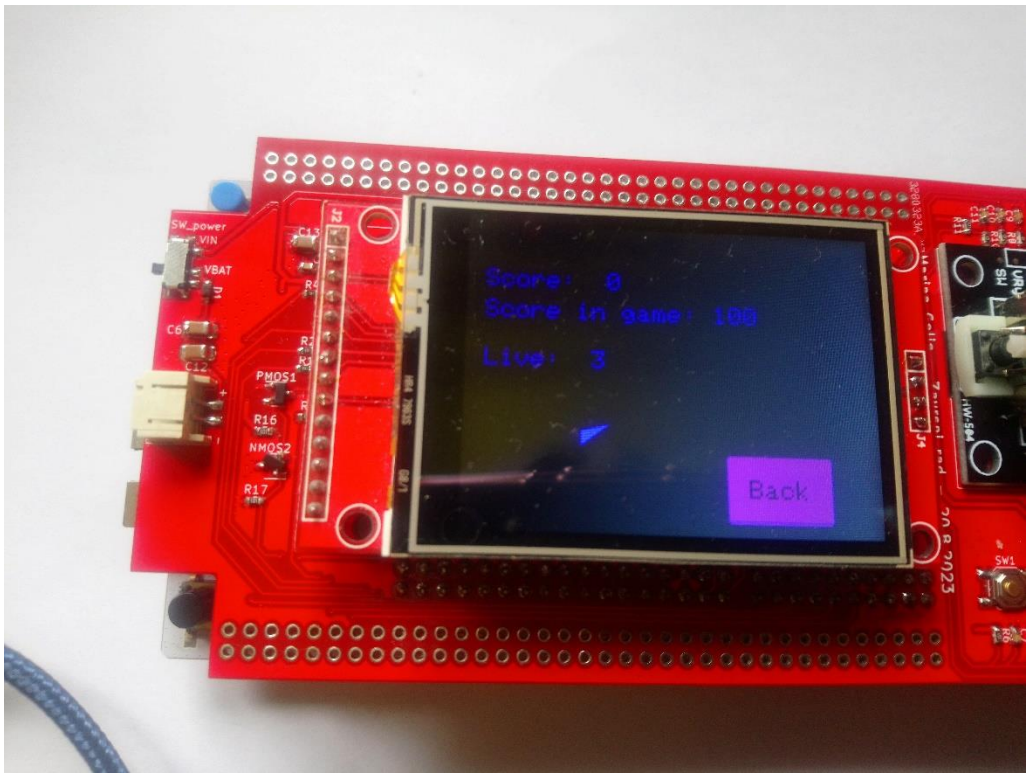
Slika 5.6. Prikaz igre pong

Korisnik može odmah započeti sa igranjem igre pomoću akcelerometra. Korisnik pomoću akcelerometra upravlja reketom, cilj je da korisnik uspije udariti sa reketom kuglicu, ako kuglica udari u rub zaslona koji se nalazi iza reketa korisnik gubi život u igri. Nakon što korisnik izgubi sve živote u igri, na zaslonu se ponovno prikazuje glavni izbornik. Ako korisnik odabere u glavnom izborniku da želi vidjeti postignute bodove iz igara, otvara mu se drugi izbornik unutar kojeg ima tri izbora, ta tri izbora su bodovi postignuti u igri labirint, bodovi postignuti u igri pong, te tipka koja omogućava povratak u glavni izbornik (slika 5.7.).



Slika 5.7. Prikaz izbornika za izbor uvida bodova

Ako korisnik odabere uvid u bodove iz bilo koje dvije navedene igre otvara mu se zaslon sa informacijama o postignutom rezultatu tijekom igranja odabrane igre(slika 5.8.).



Slika 5.8. Prikaz zaslona sa postignutim bodovima

6. ZAKLJUČAK

Igraća konzola je uspješno napravljena, zadatak rada je uspješno postignut. Korisnik može pomoću akcelerometra igrati igre koje su razvijene pomoću mikroupravljača. Korisnik osim što može igrati igre, može odabrati koju igru želi igrati i također može birati težinu igre pomoću *joysticka*. Omogućeno je i da se korisnik pomoću tipke može vratiti iz jednog izbornika u drugi, ili se može vratiti u izbornik tijekom igranja igre. Također je omogućen i uvid u postignute bodove tijekom igranja igara. Korisnik ima i mogućnost odabira kako želi napajati igraću konzolu. Odabir je omogućen pomoću prekidača. Postoje dva odabira a to su napajanje preko baterije ili preko USB kabela koji je spojen na razvojnu pločicu.

Igraća konzola se može unaprijediti na više načina. Neki od mogućih unaprjeđenja su da se razvije još igara na igraću konzolu te da se proširi mogućnost igranja igara na primjer da se igre mogu igrati pomoću *joysticka*, tipaka, ili sa žiroskopom. Može se i unaprijediti tako da se u igre implementira glazbena podloga. Nadalje može se i razviti da korisnik ima mogućnost i igrati igre koje će biti u 3D grafici. Sa hardware-ske strane je moguće da se trenutni zaslon zamjeni sa zaslonom veće rezolucije. Te također da se napravi tiskana pločica na kojoj će biti sam mikroupravljač dakle da ne bude na razvojnoj pločici, i zatim da se ta igraća konzola stavi u kućište koje se dizajnira u nekom od programa za 3D dizajn i isprinta na 3D printeru.

LITERATURA

[1] https://e-radionica.com/hr/2-4-display-240x320-touchscreen.html - Pristupljeno dana 8.7.2021.

[2] https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf - Pristupljeno dana 8.7.2021.

[3] https://www.orientdisplay.com/knowledge-base/tft-basics/how-does-tft-displays-work/ – Pristupljeno dana 9.7.2021.

[4] https://www.tme.eu/en/news/library-articles/page/22568/how-does-an-accelerometer-work-and-what-is-it-used-for/ - Pristupljeno dana 10.7.2021

[5] https://www.st.com/resource/en/datasheet/lsm6ds3.pdf - Pristupljeno dana 12.7.2021

[6] https://content.arduino.cc/assets/st_imu_lsm6ds3_datasheet.pdf

[7] https://www.arrow.com/en/categories/computer-products/computer-peripherals/joysticks - Pristupljeno dana 14.7.2021

[8] https://e-radionica.com/hr/joystick-modul-ps2.html - Pristupljeno dana 14.7.2021.

[9] https://store.arduino.cc/arduino-uno-rev3 - Pristupljeno dana 14.7.2021

[10] https://os.mbed.com/platforms/ST-Nucleo-H743ZI2/ - Pristupljeno dana 26.8.2023.

[11] https://soldered.com/product/li-ion-battery-1200mah-3-7v/ - Pristupljeno dana 26.8.2023.

[13] https://soldered.com/learn/hum-joystick-module-ps2/ - Pristupljeno dana 25.8.2023.

[14] https://digilent.com/reference/learn/microprocessor/tutorials/debouncing-via-rc-filter/start - Pristupljeno dana 23.8.2023.

[15] https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/ - Pristupljeno dana 23.8.2023.

[16]https://github.com/BornaBiro/ILI9341_STM32/blob/0f0e34f6c9e357c256a164717ed77867f84fb6ea/ILI9341_STM32.h#L120 – Pristupljeno dana 24.8.2023.

[17] https://github.com/arduino/ArduinoCore-avr/blob/master/cores/arduino/Arduino.h - Pristupljeno dana 25.8.2023.

[18] https://github.com/adafruit/Adafruit-GFX-Library - Pristupljeno dana 25.8.2023.

[19] https://www.mazegenerator.net/ - Pristupljeno dana 24.8.2023.

[20] https://github.com/sparkfun/SparkFun_LSM6DS3_Arduino_Library - Pristupljeno dana 29.8.2023.

[21] https://soldered.com/hr/learn/sto-je-to-mosfet/ - Pristupljeno dana 23.8.2023.

SAŽETAK

U ovom završnom radu je bilo potrebno omogućiti da korisnik igra igre pomoću akcelerometra. Prikaz razvijene igre labirint i igre pong se prikazuje na LCD TFT zaslonu iz tog razloga što to omogućava bolji korisnički doživljaj. Korisnik ima i mogućnost korištenja izbornika. Korisnik se izbornikom služi pomoću *joysticka*. Ako korisnik želi izaći iz igre to može učiniti pomoću tipke. Korisnik unutar izbornika ima i mogućnost uvida svojih postignutih bodova tijekom igranja igara. Omogućeno je da korisnik ima i odabir kako želi napajati igraću konzolu. Odabir je implementiran pomoću prekidača koji se nalazi na tiskanoj pločici na kojoj se nalazi zaslon, j*oystick*, tipke i akcelerometar. Omogućena su dva odabira načina napajanja , prvi odabir je napajanje preko baterije. Igraća konzola se može napajati pomoću baterije jer svi moduli koji su odabrani za razvoj igraće konzole rade na naponu od 3.3V. Baterija se uključi u konektor koji se nalazi na tiskanoj pločici, na taj način korisnik može koristiti igraću konzolu i kretati se. Drugi način je napajanje preko USB kabela, jedan dio kabela je uključen u izvor napajanja, a drugi dio kabela je uključen u razvojnu pločicu, ovaj način ne omogućava da se korisnik slobodno kreće tijekom igranja igre.

Ključne riječi : razvojna pločica NUCLEO 144, mikroupravljač STM32H743ZI2, akcelerometar, *joystick*, TFT RGB ILI9341 zaslon, labirint, pong, izbornik

ABSTRACT

In this final thesis, it was necessary to enable the user to play games using the accelerometer. The developed games, including a maze and Pong, are displayed on the LCD TFT screen to enhance the user experience. The user also has the option to navigate through a menu using a joystick. If the user wishes to exit a game, they can do so using a button. Within the menu, the user can also check their game scores. It's possible for the user to choose how to power the gaming console, with the selection implemented using switches located on the printed circuit board housing the screen, joystick, buttons, and accelerometer. Two power options are provided. The first option is battery power. The gaming console can be powered by a battery since all modules implemented for the development of the gaming console operate at a voltage of 3.3V. The battery is connected to a connector on the printed circuit board, allowing the user to use the gaming console and move around freely. The second option is power via a USB cable. One end of the cable is connected to a power source, while the other end is connected to the development board. This method does not allow the user to move freely during gameplay.

Keywords: NUCLEO 144 development board, STM32H743ZI2 microcontroller, accelerometer, joystick, TFT RGB ILI9341 screen, maze, Pong, menu.

ŽIVOTOPIS

Marina Golić je rođena u Osijeku 10.9.1995. godine. Pohađala je osnovnu školu Ivana Kukuljevića u Belišću te ju završila 2010. godine, iste godine upisuje srednju školu u Valpovu smjer komercijalist. Srednju školu je završila 2014. godine, i iste godine je položila državnu maturu. Stručni preddiplomski studij smjer Automatika na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek upisuje 2017. godine. Tijekom studiranja je sudjelovala 2018 godine na početničkoj IN2 Arduino radionici u organizaciji Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek.

Marina Golić

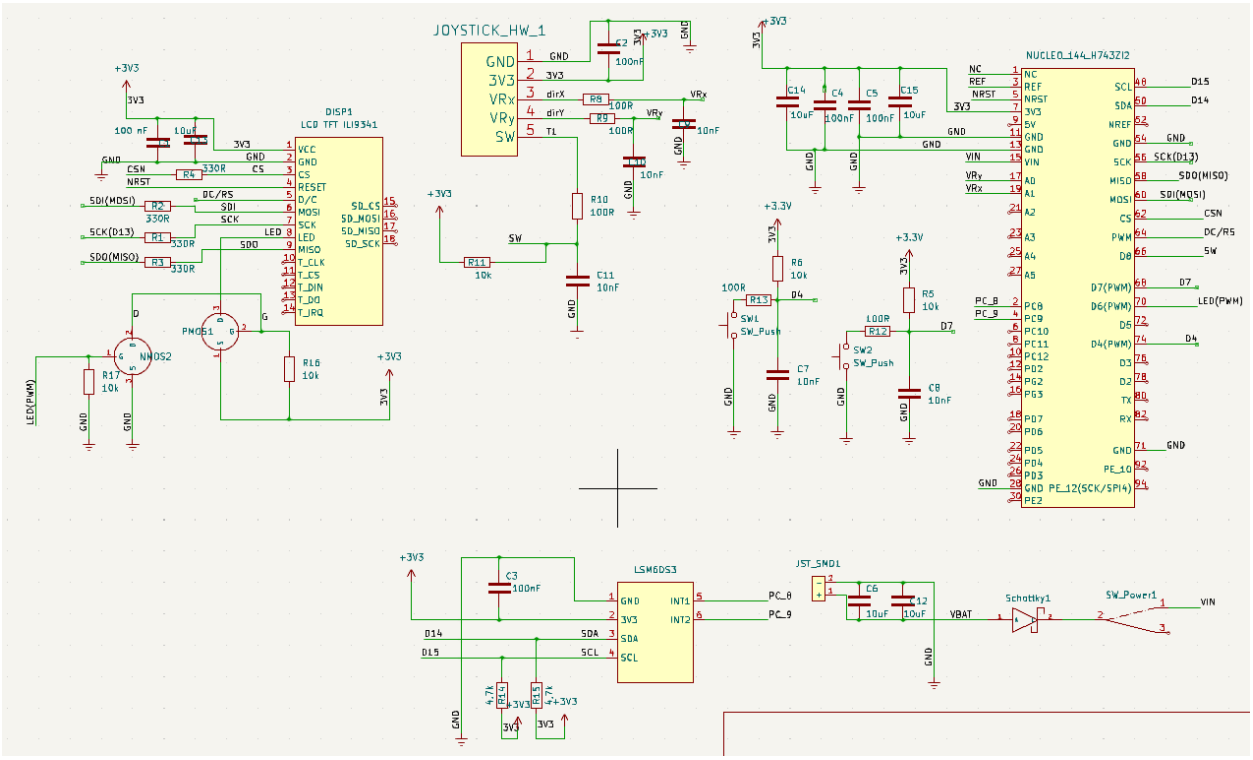
PRILOZI

Prilog 1. Programski kod za skaliranje labirinta pisan u Python programskom jeziku

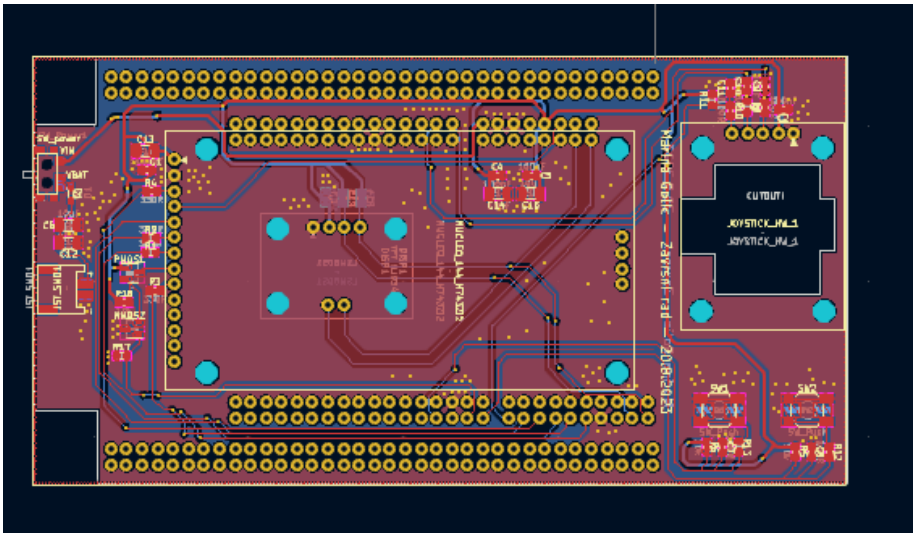
```
import re
from parse import parse


def parse_svg(fobj_in, displayw, displayh): #funkcija koja parsira svg file
    lista = []
    MyArray = []
    W = []
    w2= []
    Lista_endl = []
    endl = []
    end = []

    inFile = open(str(fobj_in), 'r') #otvori svg file i citaj ga
    oneLine = inFile.readline() #citaj liniju u otvorenom file-u
    while (len(oneLine) != 0): #sve dok ne dodes do kraja linije
        oneLine = oneLine.replace("\n", '') #makni sve prelaske u novi red
        oneLine = oneLine.replace("\r", '') #makni sve enter znakove
        oneLine = oneLine.replace(' ', '')  #makni sva prazna mjesta
        MyArray = parse('<linex1="{}"y1="{}"x2="{}"y2="{}"/>',oneLine) #PARSIRANJE ZA LINIJE, takav format zelimo dobiti jer je to ustvari x1,y1,x2,y2
        w2 = parse('<svgwidth="{}"height="{}"version="1.1"xmlns="http://www.w3.org/2000/svg">', oneLine) #PARSIRANJE ZA DULJINU I ŠIRINU LABIRINTA
        if MyArray != None: #ako argument iz liste je broj, taj argument stavi u listu
            lista.append(list(MyArray))
        if w2 != None:
            W.append(list(w2))

        oneLine = inFile.readline()  #čitaj liniju po liniju u fobj_in ustvari u svg fileu

    for i in range(0, len(W)): #petlja za širinu i duljinu labirinta
        width = (W[i][0]) #sirina labirinta
        height = (W[i][1]) #visina labirinta

    scaleX = int(width)/displayw  #racunanje scale faktora za horizontalnu os, sirinu labirinta podijeli sa sirinom zaslona
    scaleY = int(height)/displayh #racunanje scale faktora za vertikalnu os, visinu labirinta podijeli sa visinom zaslona

    inFile.close()  #zatvaranje filea za čitanje

    return lista, W, scaleX, scaleY

def search_str(fobj_in, lista, scaleX, scaleY): #Funkcija pomoću koje se traži pobjednicki put, odnosno startna pozicija u labirintu i pozicija koja oznacava izlaz iz labirinta
    inFile = open(str(fobj_in), 'r') #otvori i citaj svg file
    oneLine = inFile.readline() #citaj liniju u fileu
    lista_en = [] #inicijalizacija liste za pocetne tocke u pobjednickoj liniji
    e = [] #inicijalizacija liste za zavrsne tocke u pobjednickoj liniji
    d = []
    while (len(oneLine) != 0):
        word = "points=" #U svg fileu koji otvorimo trazimo kljucnu rijec points, jer se nakon te rjeci nalaze tocke koje oznacavaju pobjednicki put tj rjesenje za izlaz iz labirinta
        if word in oneLine:
            txt = str(oneLine) #Nakon sto pronademo kljucnu rijec, tu cijelu liniju gdje se ta rijec nalazi spremamo u varijablu txt
        oneLine = inFile.readline()

    startindex = txt.find("points=\"") #zatim trazimo index sa kojom pocinje rijec points
    print(startindex)

    txt = txt[startindex + 8:] #Za ispis podniza koji počinje u sredini niza i ispisuje se do kraja, možemo to učiniti uključivanjem samo broja indeksa ispred dvotočke, ovako:
    txt = txt.strip() #micemo razmake
    txt = txt.replace("/>", "")
    txt = txt.replace('"', '')
    lista_en.append(txt.split(' ')) #stavljamo txt u listu i odvajamo tocke, da mozemo lakse naci pocetnu i zavrsnu tocku
    firstpos = parse("{},{}", lista_en[0][0]) #Prva pozicija, lista_en[0][0] to su prvi x i y odnosno startna pozicija kuglice po x osi i po y osi
    startX = firstpos[0] # razdvajanje x i y osi
    startY = firstpos[1]

    exitpos = parse("{},{}", lista_en[0][-2]) #zadnjna tocka u labirintu, ujedno i exitLine
    exitX = exitpos[0]
    exitY = exitpos[1]
    exitB = exitY
    for i in range(0,2,1):
            e.append(list(lista[i])) #uzmi prve dvije linije iz liste i spremi ih u listu e
    de = e[0][2]  #u varijablu de spremi x2 od prve linije iz labirinta
    de2 = e[1][0] #uvarijablu de2 spremi x1 od druge linije iz labirinta

    duljina = int(de2) - int(de) #oduzmi x1 od druge linije od x2 iz prve linije, tako ces dobit razmak izmedu dvije linije u labirintu odnosno duljinu exit linije
    du = duljina/2 #podjeli taj razmak sa 2, to radis jer ti imas u pocetku od exit linije samo jednu tocku, i onda toj tocki dodajes taj razmak/2 odnosno oduzimas
    exitX1 = (int(exitX) + du) #exit tocki koja je na polovini razmaka dodaj polovicu razmaka
    exitX2 = (int(exitX) - du) #exit tocki koja je na polovini razmaka oduzmi polovicu razmaka
    inFile.close()

    return startX, startY, exitX1, exitY, exitX2, exitB, duljina


def save_h(outFile, lista, scaleX, scaleY, startX, startY,exitX2, exitY, exitX1, exitB, duljina ): #funkcija pomocu koje upisujemo parsirani labirint u h file

    NameMaze = input("Upisi naziv labirinta: ")
    NameDate = input("Upisi naziv podatka ")
    outFile.write(f"const myline_t {NameMaze}[] = " + "{")

    for i in range(0, len(lista)): #petlja za upis u h file svih linija iz svg filea
        outFile.write("{")
        outFile.write("{}, {}, {}, {}".format(
                int(float(lista[i][0])/scaleX),
                int(float(lista[i][1])/scaleY),
                int(float(lista[i][2])/scaleX),
                int(float(lista[i][3])/scaleY)
            ))
        outFile.write("}")
        if i != (len(lista) - 1):
            outFile.write(", ")
    outFile.write("};")
    outFile.write("\n")
    outFile.write(f"const myMaze_t {NameDate} = " + "{" )
    outFile.write(f"{myline_t*}{NameMaze} " + ",")
    outFile.write("{} , {} ".format(int(float(startX)/scaleX), int(float(startY)/scaleY))) #upis startne pozicije u h file
    outFile.write(" , {")
    outFile.write("{}, {}, {}, {} ".format(int((float(exitX1)/scaleX)) ,int(float(exitY)/scaleY), int((float(exitX2)/ scaleX)), int(float(exitB)/scaleY))) #upis zadnje linije u h file
    outFile.write("},")
    outFile.write(f"sizeof({NameMaze}) / sizeof(myline_t)" + "}" + ";")
    outFile.write("\n")


def headerguard(outFile):
    outFile.write("#ifndef __LAB_H__ \n") #Make header source in open file
    outFile.write("#define __LAB_H__ \n")
    outFile.write('#include "myDefines.h" \n')
    outFile.write('#include "ILI9341_STM32.h" \n')
```

```
def main():

    displayw = 319
    displayh = 219
    numberMaze = int(input("Upisi koliko zelis imati labirinteva "))
    fobj_out = input("Upisi gdje zelis da ti se spremi skalirani labirint ")
    outFile = open(fobj_out, 'w')
    outFile.write("\n")
    headerguide(outFile)


    for i in range(0, numberMaze):

        fobj_in = input("Upisi svg file ") #trazi od korisnika da upise file koji zeli da se skalira
        lista, W, scaleX, scaleY = parse_svg(fobj_in, displayw, displayh) #pozovi funkciju za parsiranje
        startX, startY, exitX1, exitY, exitX2, exitB, duljina = search_str(fobj_in, lista, scaleX, scaleY)#pozovi funkciju za trazenje pocetne i zavrsne tocke u labirintu
        save_h(outFile, lista, scaleX, scaleY, startX, startY, exitX1, exitY, exitX2, exitB, duljina) #pozovi funkciju za upis skaliranog labirinta u h file
    outFile.write("#endif")
    outFile.close()


if __name__ == "__main__":
    main()
```

Prilog 2. Shema igraće konzole



Prilog 3. PCB igraće konzole



Prilog 4. Programski kod pisan u C/C++ sa podrškom Arduino platforme

https://github.com/ladyM9/Game_in_C_program/tree/Game_version2

Prilog 5. Dijagram toka programskog koda