TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Aleks Savioja 185247IACB

Elektromehaaniline Tetris

Bakalaureusetöö

Juhendaja: Peeter Ellervee
PhD

Tallinn 2024

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Aleks Savioja

13.05.2024

Annotatsioon

Lõputöö eesmärk on huviprojekti korras kokku panna mängitav elektromehaanilisel kuvaril töötav tetrisemäng. Käesolevas töös valmib Arduino miktrokontrolleril jooksev tetrisemäng, mis on mängija poolt puldiga juhitav ning kajastub korrektselt mängu olekut *flip-dot* kuvaril. Töös kirjeldan ning põhjendan riistvara valikuid. Lisaks seletan lahti enda Tetrisemängu tarkvara, mis on kirjutatud C++ programmeerimiskeeles ning põhineb objektorienteeritud mudelil ja selgitan suurimaid takistusi, mis seisnesid Arduino mälu puudustes.

Lõputöö tulemusena valmis eraldiseisev puldiga juhitav elektromehaaniline tetrisemängu süsteem, mida kasutajal on toiteallika olemasolul võimalik iseseisvalt mängida.

Lõputöö on kirjutatud Eesti keeles ning sisaldab teksti 27 leheküljel, 6 peatükki, 24 joonist, 7 tabelit.

Abstract

Electromechanical Tetris

The purpose of this thesis is to develop a functional and playable Tetris game that is showcased on a electromechanical flip-dot display. The thesis provides a basic overview for the hardware choices, their functionalities, compatibilities and justifications. Giving a more detailed explanation on the inner workings of the flip-dot technology. The main focus revolves around the software development in C++, using object-oriented programming principles, and the complications that came with the limitations of Arduino Uno memory.

As of result of this thesis a working tetris game is developed that runs on a Arduino UNO microcontroller, is controllable through a simple 8 button gamepad and is presented to the user by using a flip-dot display. The game is playable for the user provided that they have access to an electrical outlet.

The thesis is in Estonian and contains 27 pages of text, 6 chapters, 24 figures, 7 tables.

Lühendite ja mõistete sõnastik

Pentominos	Pusle lauamäng
Tetromino	Tetrise mängujupp
flip-dot	Pöördplaat
NES	Nintendo
UNO	Arduino UNO Rev3 SMD
IDE	Integreeritud arenduskeskkond
LIFO	*Last In, First Out*. Viimane sisse, esimene välja - pinu haldamis meetod
Stack	Pinu
Heap	Kuhi
UART	*Universal asynchronous receiver / transmitter*, universaalne asünkroonne vastuvõtja / saatja
Non-Return-to-Zero	Modulatsiooni kodeerimisskeem
Driver	Draiver
Receiver	Vastuvõtja
USB	*Universal serial bus,* universaalne jadasiin
Object-oriented programming	Objektorienteeritud programmeerimine
Debugging	Silumine
DI	*Driver Input,* Draiveri sisend
RI	*Receiver Input,* Vastuvõtja sisend
DE	*Driver Enable,* Draiveri lubamine
RE	*Receiver Enable,* Vastuvõtja lubamine
GND	*Ground,* Maandus
Tx	*Serial Transmit Pin,* jadaedastus viik
DIP switch	DIP lüliti
USB Host Shield	Arduino lisamoodul, mis võimaldab USB seadmete ühenduse mikrokontrollerile

Sisukord

1 Sissejuhatus..... .10
2 Tetrise lühitutvustus..... .11
2.1 Tetrise mängu mõistmine..... .11
2.1.1 Mängu reeglid ja piirangud.....12
3 Riistvara valikud.....13
3.1 Kuvar..... .13
3.1.1 Flip-dot..... .14
3.1.2 Kuvari kontrolleri kontseptsioon.....16
3.1.3 Kuvari tehnilised andmed.....17
3.2 Juhtimiseks mikrokontroller.....19
3.2.1 Arduino.....20
3.2.2 Arduino mälu.....20
3.3 MAX485.....22
3.3.1 RS-485 Jadaliides..... .22
3.4 Juhtpult..... .24
3.5 USB Host shield..... .25
3.6 Toiteplokk.....25
3.7 Toitemoodul.....25
4 Tarkvara arendamine..... .27
4.1 Programmeerimiskeel ja -keskkond..... .27
4.2 Üldarhitektuur.....27
4.3 Tarkvara klassid.....28
4.4 Mängu kulg.....29
4.5 Väljakutsed ja lahendused..... .30
4.6 Optimeerimine.....33
5 Kokkupanek.....34
5.1 Riisvara vahelised ühendused.....34
5.2 Süsteemi testimine.....35

5.3 Tulemused.....35
6 Kokkuvõte..... .37
Kasutatud kirjandus..... .38
Lisa 1– Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks..... .40
Lisa 2 – Vooskeem..... .41
Lisa 3 – Klassi diagramm..... .42
Lisa 4 – Muud optimeeringud..... .43
Lisa 5 – Elektriskeem..... .46
Lisa 6 – Dip lülitite seadistus..... .47
Lisa 7 – Riistvara pildid.....48

Jooniste loetelu

Joonis 1. Kuvatõmmis Electronika 60 Tetrise mängust. [2].....11
Joonis 2: Füüsiline pilt Alfa-Zeta *flip-dot* kuvarist..... .14
Joonis 3. *Flip-dot* indikaatorite rida [5].....15
Joonis 4. *Flip-dot* indikaatorite aktiveerimise seletus [7].....17
Joonis 5. Arduino ARV miktrokontrolleri andmemälu kaardistus [13]..... .21
Joonis 6: Arduino vabamälu lugemiseks kasutatav funktsioon, kus loetakse *stack*'i ja *heap*'i vahelist ruumi [13].....22
Joonis 7. Poolduplekс [17].....23
Joonis 8. UART formaat [18].....24
Joonis 9: Moodulite plokkskeem.....28
Joonis 10: Esialgne ressursimahukas massiiv..... .31
Joonis 11: Optimeeritud massiiv..... .31
Joonis 12: Kuvari andmepaketti koostamine koodis..... .33
Joonis 13: Komplekteeritud süsteem.....36
Joonis 14: Vooskeem.....41
Joonis 15: Klassi diagramm.....42
Joonis 16: Komponentide omavahelised ühendused..... .46
Joonis 17: Esimese paneeli seadistus - aadress: 0x01, edastuskiirus: 57600.....47
Joonis 18: Teise paneeli seadistus - aadress: 0x02, edastuskiirus: 57600..... .47
Joonis 19. Mean Well LPV-60-24 [19]..... .48
Joonis 20. Arudino UNO rev3 SMD [11].....48
Joonis 21: MAX485 moodul [14].....48
Joonis 22. Toitemoodul DC/DC step-down 2.5..40V/1.25..35V 2A [20]..... .49
Joonis 23: USB Host shield 2.0 [21]..... .49
Joonis 24: USB NES mängupult [22].....49

Tabelite loetelu

Tabel 1. Alfa-Zeta *flip-dot* kuvari tehnilised andmed [8].....18
Tabel 2. Dip Switch - kontrolleri ülekande kiiruste konfiguratsioon [8].....18
Tabel 3. Dip switch - kontrolleri aadressi konfigureerimine [8]..... .18
Tabel 4: *Flip-dot* kuvari käsklused [8]..... .19
Tabel 5. Arduino UNO Rev3 SMD - Tehnilised andmed [12]..... .20
Tabel 6. Mean Well LPV-60-24 Tehnilised andmed [19].....25
Tabel 7. Kolmemõõtmelise masiivi optimeerimine..... .31

1 Sissejuhatus

Tetris on 1980ndatel aastatel loodud pusle videomäng, mis saavutas kiirelt rahvusvahelise kuulsuse. Üles ehitatud lihtsatele reeglitele, kerge ning kaasahaarav mäng on üks kõige ikoonilisemaid ning enimmüüdud mänge, mis iial loodud. Lapsepõlve nostalgiast inspireerituna otsustasin lõputöö mahus luua huviprojektina retro-stiilis mehaanilise tetrisemängu, mis hõlmab väljakutseid algajale riistvara arendajale ja programmeerijale.

Käesoleva bakalaureusetöö eesmärgiks on luua funktsioneeriv ning mängitav füüsiline koopia elektromehaanilisest tetrisest, mida kajastatakse *flip-dot* kuvaril *Arduino* mikrokontrolleri käskluste alusel ning on mängupuldiga juhitav.

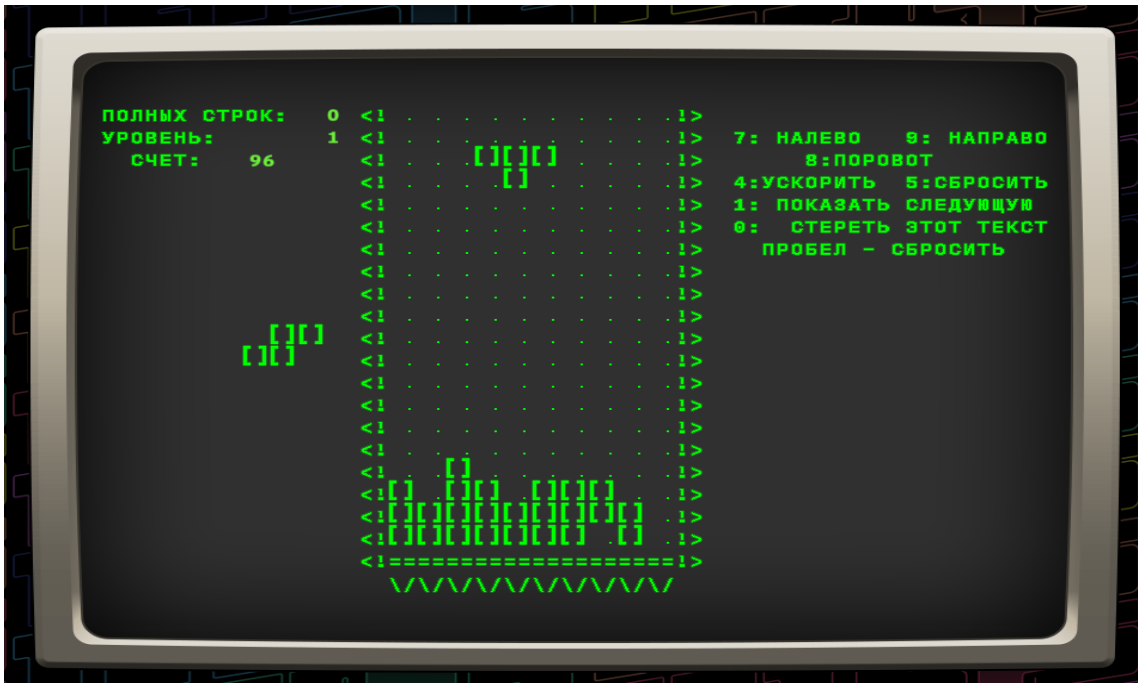
- Loon Arduino mikrokontrollerile töötava ning optimeeritud Tetrisemängu.
- Lisan Arduino'le juhtpuldi, millega mängijal on võimalik mängutükke liigutada.
- Tagan Arduino ja *flip-dot* kuvari omavahelise töötamise, kus mikrokontroller on suuteline koostama/saatma korrektseid andmepakette kuvarile ning veendun kogu mängusüsteemi terviklikus töötamises.

Lisaks seletan lahti riistvara valikud ja tööpõhimõtted ning annan ülevaate nende tehnilistest andmetest, kirjeldan tarkvara arenduskäiku C++ keeles, metoodika valikuid ning lahkan suurimaid takistusi, mis tulenesid mikrokontrolleri piiratud sisemälust.

Töö tulemusena valmib mängitav tetrise prototüüp, mille olekut kajastatakse elektromehaanilisel *flip-dot* kuvaril ning on kasutaja poolt juhitav.

2 Tetrise lühitutvustus

Tetrise lõi 1984ndal aastal Nõukogude Liidu tarkvarainsener Alexey Pajitnov. Inspireerituna enda lemmikust lauamängust *Pentominos*'t ning enda lemmikspordist tennisest tuli ta nimele Tetris [1]. Esimene versioon loodi *Electronika 60* mikroarvutil, mis saavutas kiirelt ülemaailmse kuulsuse. Tänapäevaks on Tetris kõige porditum arvutimäng, omades palju erinevaid mängu versioone, millel kõigil on oma reeglid. Allpool esitatud esimese Tetrise versiooni pilt (Joonis 1). Tegemist pole originaaliga, vaid veebibrauserisse porditud versiooniga, mis on kõigile ligipääsetav [2].



Joonis 1. Kuvatõmmis Electronika 60 Tetrise mängust. [2]

2.1 Tetrise mängu mõistmine

Tetrise põhikontseptsioon on langevate kujundite, mida nimetatakse *tetromino*'deks, juhtimine. Eesmärk on koguda punkte, paigutades *tetromino*'d nii, et need täidaksid mänguruudustiku horisontaalse joone. Kui rida on täielikult täidetud, kustutatakse see ja

kõik selle kohal olevad tükid liigutatakse vastavalt kustutatud reale, luues rohkem ruumi uutele kujunditele.

2.1.1 Mängu reeglid ja piirangud

Igal Tetrise versioonil on oma spetsiifilised reeglid ja piirangud, kuid selles projektis keskendun NES Tetrise reeglitele. Valik põhineb enda isiklikust kogemusest mainitud versiooniga.

- **Mänguväli**: Koosneb 20x10 plokist koosnevast väljast.
- **Tükkide genereerimine**: Mäng peab olema suuteline genereerima vajadusel *tetromino* plokke ruudustiku ülaossa.
- **Tükkide liikumine**: Mängija peab saama liigutada ja pöörata *tetromino* plokke.
- **Ruudustiku piirangud**: *Tetromino'*d ei tohi saada liikuda mänguvälja piiridest väljapoole.
- **Tükkide lukustamine**: Kui *tetromino* puudutab mänguvälja põhja või teist lukustunud tükki, lukustub see oma kohale. Seejärel genereeritakse mängijale mänguvälja ülaosas uus tükk.
- **Sunnitud liikumine**: *Tetromino*'d langevad automaatselt mänguvälja põhja suunas ning nende langemise kiirus kasvab vastavalt sellele, kui palju ridu mängija täidab ning vabastab.
- **Ridade vabastamine**: Kui mänguvälja horisontaalne rida täidetakse, peab see rida kaduma ja kõik selle kohal olevad plokid liigutatakse vastavalt madalamale.
- **Mängu lõpp**: Mäng lõppeb, kui *tetromino*'d kuhjuvad mänguvälja ülaossa ja uue tüki asetamiseks pole ruumi.
- **Taaskäivitamine**: Peale mängu lõppemist peab mäng olema mängija poolt taaskäivitatav.
- **Järgmise tüki aken:** Väike aken kuvari paremas servas näitab järgmist *tetromino* tükki, mis ilmub, andes mängijale aega mõtlemiseks.

3 Riistvara valikud

Tetrise riisvara valikuteks said järgmised komponendid: Alfa-Zeta elektromehaaniline *flip-dot* kuvar (Joonis 2) [3], *Arduino UNO R3 SMD* mikrokontroller (Joonis 20) [11], MAX485 transmissiooni- moodul (Joonis 21) [14], Mean Well 24V toiteallikas (Joonis 19) [19], toitemoodul DC/DC *step-down* (Joonis 22) [20], *USB host shield* (Joonis 24) [21], mängupult (Joonis 23).

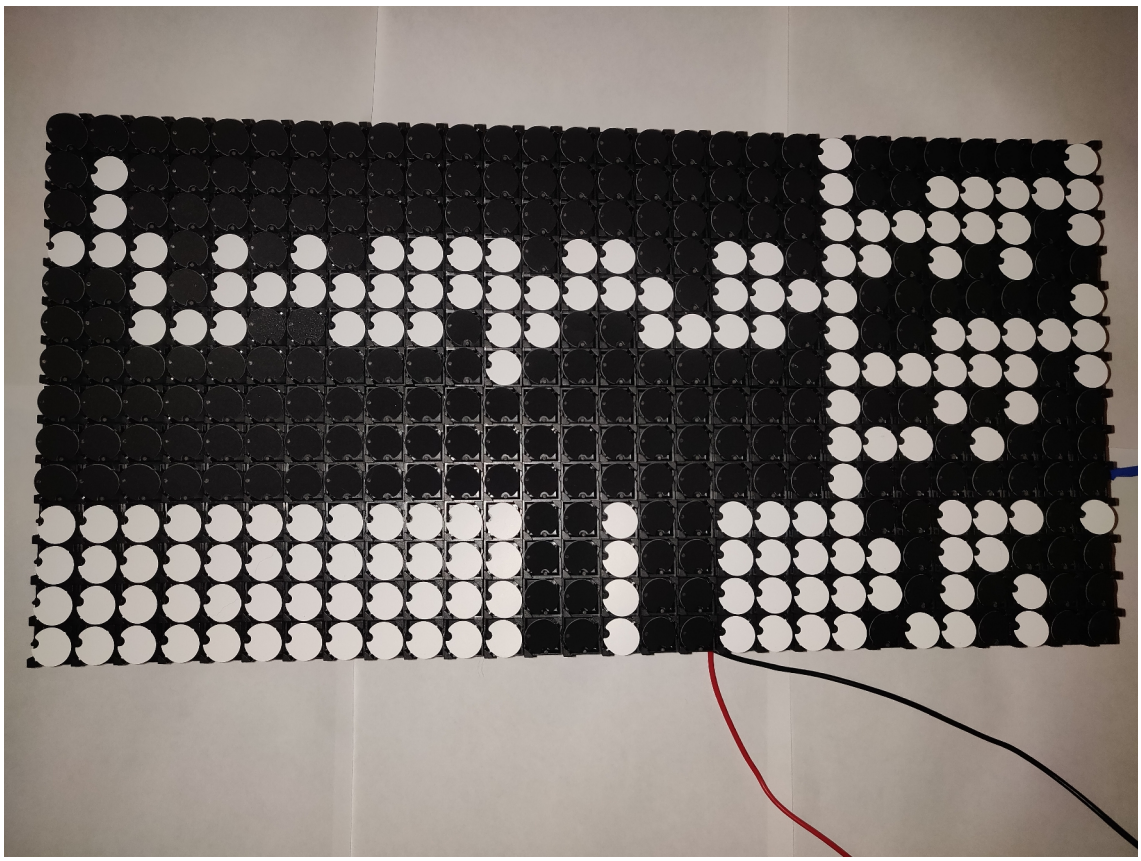
3.1 Kuvar

Nõuded tootele:

- Peab selgelt näitama mängu olekut mehaanilise väljundi kaudu, mis on mängijale hõlpsasti loetav.
- Suurus peaks mugavalt mahutama Tetrise mänguruudustikku, pakkudes samal ajal täiendavat ruumi järgmise mängutüki ja praeguse taseme tutvustamiseks.
- Kuigi see peaks pakkuma piisavalt nähtavust, ei tohiks kuvar olla liiga suur.
- Lisaks on sujuva mängimise tagamiseks oluline, et kuvaril oleks piisavalt suur värskendussagedus.

Valik:

Otsustasin Alfa-Zeta *flip-dot* kahepaneelilise kuvari kasuks, kuna see oli saadaval valmisehitatul kujul, vastas kõigile minu nõuetele ning oli aksepteeritava tarneajaga. Kuvari pilt esitatud allpool Joonis 2. Alternatiivsed valikud olid panna ise kokku mehaaniline kuvar kas lukustussolenoididest või servomootoritest. Lukustussolenoididega tehes oleks projekt läinud liiga kalliks ning servomootoritega oleks kuvar tulnud ebapraktiliselt suur. Lisaks oleks mõlemal juhul kuvari kokkupanek võtnud kõvasti rohkem aega, osutunud palju keerukamaks ning ei oleks mahtunud lõputöö ajaraami.



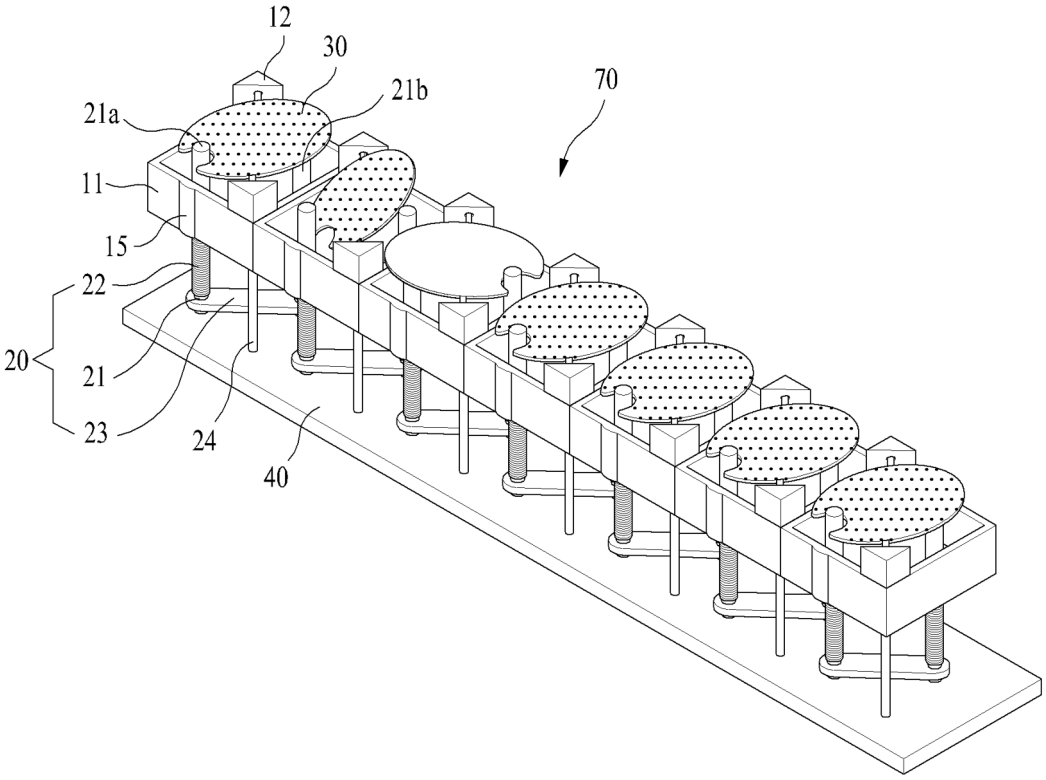
Joonis 2: Füüsiline pilt Alfa-Zeta *flip-dot* kuvarist

3.1.1 Flip-dot

Süsteemi tähelepanuväärseim element on tema elektromehaaniline kuvar. Nimelt on tegu suhteliselt vana ja teatud tehnoloogiaga, mida põhiliselt kasutati infotabloodena erinevates börsi-, transpordi- ja muudes ettevõtetes. Tehnoloogia arenguga vahetati aja möödudes aegunud kuvarid uuemate ning odavamate LED ja LCD ekraanide vastu, mis pakkusid paremaid resolutsioone, selgemat pilti, vähemat hooldamist ja olid energiasäästlikumad [4]. Sellegipoolest on tegu huvitava süsteemiga, mille visuaalne esteetika ning meelepärane heli muudavad selle köitvaks valikuks kunsti- ja huviprojektideks.

Kuvari terviklik visuaalne pilt koosneb kahest 28x7 (moodustades kokku 28x14 kuvari) paneelist, mis koosnevad indikaatoritest. Antud komponendi paremaks kirjeldamiseks lisatud järgmisele lehel Joonis 3. Iga indikaator koosneb mitmest võtmeosast: U-kujulisest metallsüdamikust (pildil 20), mis suudab säilitada magnetilisi omadusi,

püsimagnetist, juhtmekeerdudest koosnevast mähisest (pildil 22), pöörlevast kettast (pildil 30) ning plastkorpusest (pildil 11), mis kinnitab kõik elemendid kokku[5].



Joonis 3. *Flip-dot* indikaatorite rida [5]

Ketas:

Põhitähtsusega element ekraanil, mille tunnusjoon on kahevärviline disain. Tüüpiliselt üks pool on kas valge või kollane, et saavutada hea kontrast ning tuua esile kettad, teine pool aga must. Lähemal ülevaatusel on võimalik näha, et ketas koosneb kolmest kihist: välisest värvikihist, läbipaistvast plastikust keskmisest aluskihist ning mustast püsimagneti kihist.

Püsimagnet:

Püsimagnet asub plastkorpuse põhjas ning selle ülesanne on ketta pöördemehhanismi soodustada. Magnet kiirendab ketta liikumist, saavutades sujuvama ning efektiivsema

oleku vahetuse. Püsimagneti eemaldamine korpusest vähendab märkimisväärselt konstruktsiooni töötamist niivõrd, et ketta olekut ei pruugita muuta.

Plastkorpus:

Plastkorpuse ainus funktsioon on tagada erinevate *flip-dot* kuvari võtmeosade omavaheline ühilduvus.

Juhtmekeerdmähis:

Juhtmekeerdmähis, mis asetseb korpuse all, on mähitud ümber hea magnetilise püsivusega U-kujulise metallsüdamiku. See konfiguratsioon on mõeldud selleks, et muuta metallsüdamik püsimagnetiks mähise pingestamisel, andes sellele magnetpooluse.

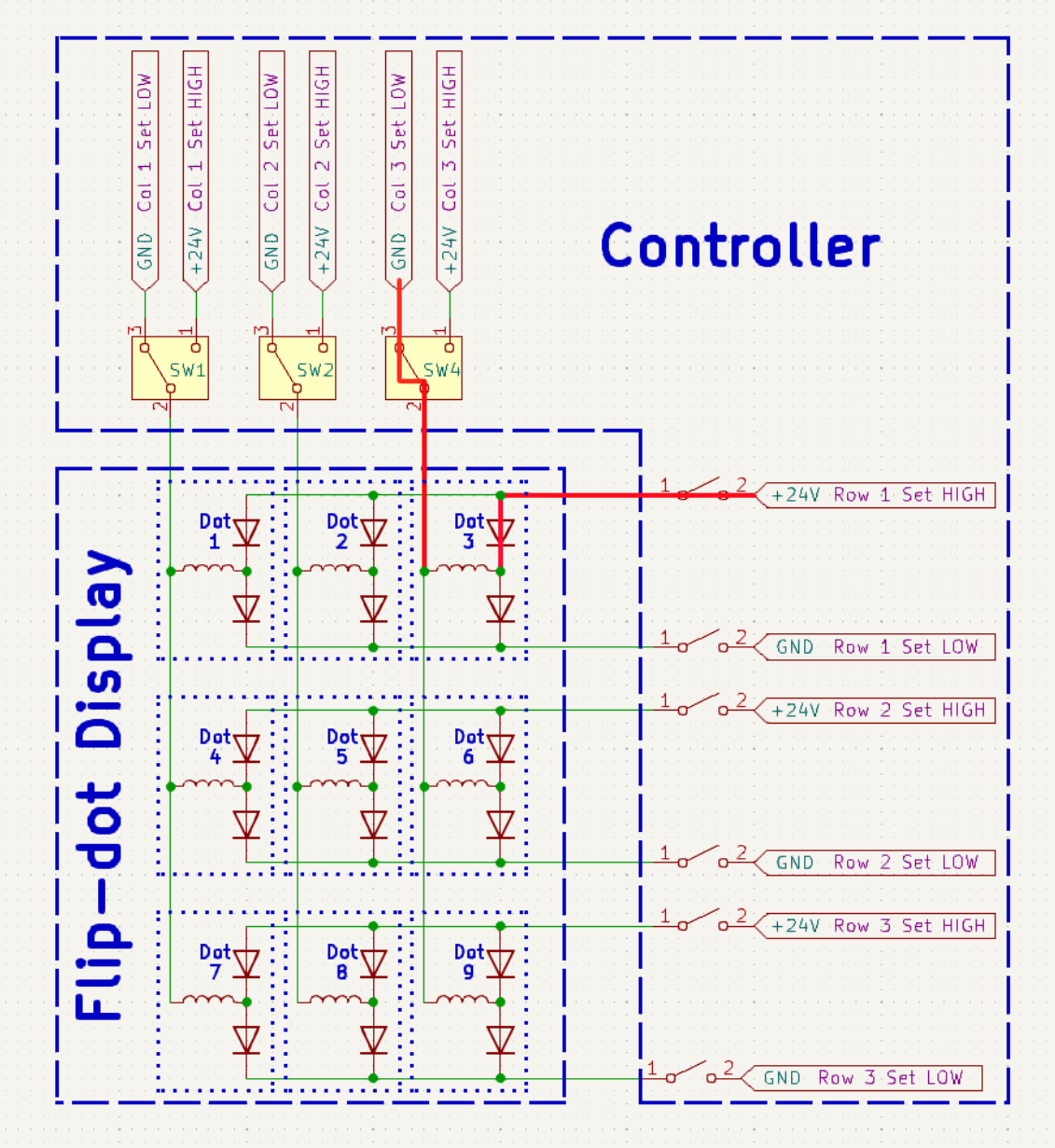
U-kujuline metallsüdamik:

Kõrge remanentsusega elektromagneetiline südamik, mis magnetpooluste muutumisel pöörab ketast [6].

3.1.2 Kuvari kontrolleri kontseptsioon

Üldjuhul on *flip-dot* kuvari indikaatordetailid paigutatud ruudustikku. Vältimaks soovimatute punktide aktiveerimist kuvari pingestamisel on iga indikaatori mähis varustatud dioodipaariga. Joonis 4 jälgides on võimalik simuleerida joonisel 'dot 3' aktiveerimist eeldusel, et 'row 1 set high' ning 'col 3 set low'.

Kontrolleri põhikontseptsioon ongi indikaatorite aktiveerimine õigel asukohal, mis tagatakse sisendite/väljundite manipuleerimisega õigetel ridadel ja veergudel [7].



Joonis 4. *Flip-dot* indikaatorite aktiveerimise seletus [7]

3.1.3 Kuvari tehnilised andmed

Ohutu töö tagamiseks on tootja poolt kaasa antud andmeleht [8], mis tagab ekraani korrektse käivitamise. Lisaks on andmelehel ka välja toodud kontrolleri seadistamiseks vajalik informatsioon. Kontrolleri juhtimiseks peab kasutama RS-485 andmeedastus standardit. Tabel 1 kirjeldab töötamiseks vajaminevaid andmeid, Tabel 2 kirjeldab ülekande kiiruse seadistamist, Tabel 3 kirjeldab kontrolleri aadressi seadistamist, Tabel 4 kirjeldab kuvari juhtimisprotokolli käsklusi.

Tabel 1. Alfa-Zeta *flip-dot* kuvari tehnilised andmed [8]

Sisendpinge	24V
Maksimaalne voolutarve	680mA
Ajakulu terve paneeli pööramiseks	160ms
Soovitatavad toiteallikad	1A / panel @ 24V

Tabel 2. Dip Switch - kontrolleri ülekande kiiruste konfiguratsioon [8]

Value	**ON**			**Speed**
	1	**2**	**3**	
0	↓	↓	↓	N/A
1	↑	↓	↓	N/A
2	↓	↑	↓	N/A
3	↑	↑	↓	9600
4	↓	↓	↑	19200
5	↑	↓	↑	38400
6	↓	↑	↑	57600
7	↑	↑	↑	9600
	OFF			

Tabel 3. Dip switch - kontrolleri aadressi konfigureerimine [8]

Position	**Meaning**
0 - 5	Address in binary code (natural)
6	Magnetizing time: OFF – 500 us (default) , ON – 450 us

Tabel 4: *Flip-dot* kuvari käsklused [8]

Käsklus	Andmebaitide kogus	Värskendus	Kuvari paneelide konfiguratsioon
0x82	0	Jah	Vorming(0x80, 0x82, 0x8F) – Kõikide ühendatud kuvarite värskendamine
0x83	28	Jah	28x7
0x84	28	Ei	28x7
0x87	7	Jah	7x7
0x88	7	Ei	7x7
0x92	14	Jah	14x7
0x93	14	Ei	14x7

3.2 Juhtimiseks mikrokontroller

Nõuded tootelt:

- Peab olema suuteline iseseisvalt ja pidevalt Tetrise mängu jooksutama seni, kui sellel on toide.
- Peab olema suuteline edastama mängu oleku andmeid kuni 15 kaadrit sekundis.
- Peab võimaldama andmeedastust RS-485 jadaliini kaudu, kas lisa mooduliga või iseseisvalt.
- Peab pakkuma sisestusvõimalusi olemasoleva mängukontrolleri integreerimiseks või kohandatud kontrolleri loomiseks.
- Peab olema taskukohane ja kulutõhus.

Valik:

Otsustasin Arduino Uno R3 SMD kasuks. Mikrokontrolleri valikut on praktilisuse mõttes raske seletada. Alternatiivsed valikud, nagu näiteks ESP-WROOM-32 või STM32 Blue Pill (valikuid on palju), on igas mõttes paremad: Olles samas hinnaklassis, pakuvad kõik konkureerivad mikrokontrollerid rohkem mälu ja suuremaid kiiruseid [9], [10]. Minu põhjus oli lihtne: omasin ühte ning see vastas nõuetele. Hoolimata sellest, et see vajas kõvasti rohkem optimeerimist tahtsin olemasolevaga hakkama saada.

3.2.1 Arduino

Arduino on kasutajasõbralik elektroonikaplatvorm, mis võimaldab kasutajal lugeda sisendeid ja teisendada need väljunditeks. Üldjuhul on antud mikrokontroller piisav kõiksugu lihtsate projektide loomiseks.

Arudino UNO rev3 SMD – tehnilised andmed esitatud tabelis (Tabel 5).

Tabel 5. Arduino UNO Rev3 SMD - Tehnilised andmed [12]

Mikrokontroller	ATmega328P 16MHz
Tööpinge	5V
Sisendpinge (soovitatav)	7-12V
Digitaalsed I/O viigud	14, millest 6 pakuvad PWM-väljundit
Analoogsisendi viigud	6
DC vool I/O viigu kohta	20 mA
Flash Memory	32 KB (ATmega328P)
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)

3.2.2 Arduino mälu

ATmega328P on Harvardi arhitektuurile baseeruv AVR mikrokontroller, kus juhised ja andmed salvestatakse erinevatesse mäluplokkidesse ning neile pääseb ligi ainult erinevatelt sideliinidelt [13].

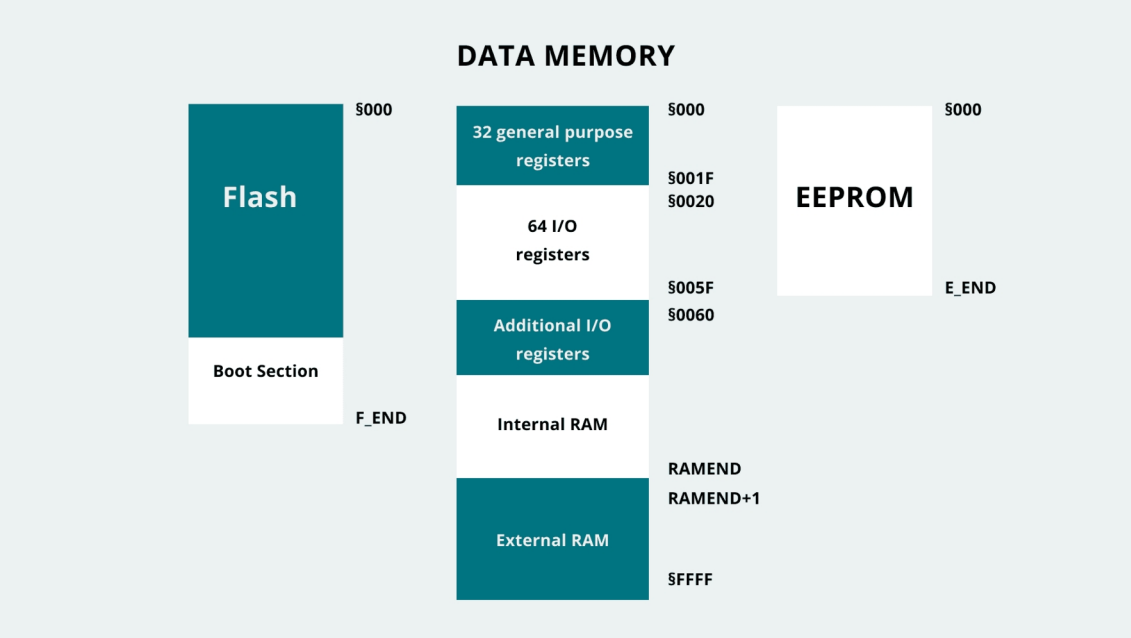
Mälutüübid võib jagada kahte erinevasee kategooriasse: muutmälu (SRAM) ja püsimälu (Flash memory). Püsimälu on säiliva omadusega andmete ladustaja, kuhu salvestatakse kogu kompileeritud koodi käsklused, säilitades nende olemasolu isegi sisendtoite puudumisel. See tähendab, et kogu üleslaetud programm on alati olemas ning Arduino säilitab oma funktsionaalsuse ka peale igat taaskäivitamist. Muutmälu aga tegeleb

peamiselt programmi tööajal tekkivate muutujatega ning ei säilita enda andmeid pärast sisendtoite kadumist. Muutmälu hõlmab endas ka kahte kriitilist mälu regiooni: stack ja heap.

Stack on enda olemuselt organiseeritud kui *Last in, First out* (LIFO) struktruurina [13] ning haldab endas programmi tööajal tekkinud funktsioonikutseid ja lokaalseid muutujaid, mis ka funktsiooni lõppemisel koheselt eemaldatakse.

Heap aga tegeleb dünaamiliselt eraldatud mäluga, mida kontrollitakse selliste funktsioonidega nagu `malloc()` või `new()`, pakkudes võimalust andmete käitlemiseks, mille suurused on kompileerimise ajal teadmata.

Järgnevalt esitatud pilt (Joonis 5) näitab Arduino AVR mikrokontrollerite andmemälu kaardistust, kus selgesti näha mäluplokkide eraldatust ning sektsioonide aadresse.



Joonis 5. Arduino ARV miktrokontrolleri andmemälu kaardistus [13]

Seoses *stack*'i ja *heap*'i kasvamise suunast kontrollitakse vabamälu nende vahelise suuruse tagastamisega. *Stack* laieneb muutmälu ülalosast allapoole samal ajal kui *heap* aga kasvab alt üles. Lahutades *stack*'i viimase aadressikuvari *heap*'i viimasest (Joonis 6), on võimalik saada hetkeseisu vabamälu [13].

```
int freeRam() {
      extern int __heap_start,*__brkval;
      Int v;
      return (int)&v - (__brkval == 0
                        ? (int)&__heap_start : (int) __brkval);
}
```

Joonis 6: Arduino vabamälu lugemiseks kasutatav funktsioon, kus loetakse *stack*'i ja *heap*'i vahelist ruumi [13]

Mõned optimeerimismeetodid on kasutada püsimälu andmete salvestamiseks võtmesõna PROGMEM'i abil ning kasutada *Serial* käsklustega käsikäes stringi ümbrist F(). Samuti on tungivalt soovitatud kasutada *stack* mälu üle *heap*'i, sest esimene on killustuskindel [13]. Ilmselgelt tuleks vähendada ka igasugu massiivide suurused nii minimaalseks kui võimalik. Tuleks ka kasutada väiksemahulisi andmetüüpe, kus võimalik.

3.3 MAX485

Nõue tootelt:

- Peab olema suuteline UART-jadasignaali teisendama RS485-diferentsiaalsignaaliks.

Valik:

Kuna antud komponendi valikul oli väga vähe nõudeid ning sobiks põhimõtteliselt iga alternatiiv, siis soetasin lihtsalt esimese kõige kättesaadavama mooduli.

MAX485 on väikese võimsusega RS-485 saatja/vastuvõtja moodul [14], mida kasutatakse UART-jadasignaali teisendamiseks RS-485 tüüpi diferentsiaalsignaaliks. Kuna antud mooduli tööpinge ühildub Arduino UNO'ga (5V), saab seda kasutada UNO andmete töötlemiseks.

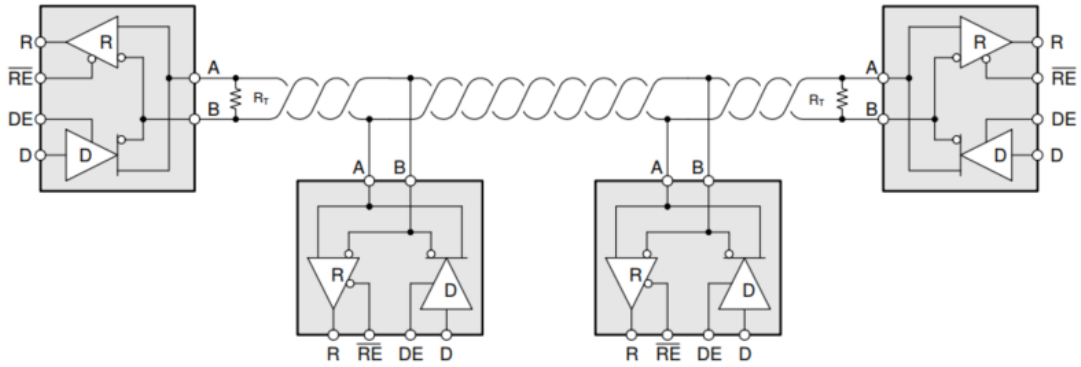
3.3.1 RS-485 Jadaliides

Saavutamaks tõrgeteta töötamise on oluline mõista kuvari ja mikrokontrolleri vahelist suhtlemist. RS-485 on standard, mis määratleb sidesüsteemides kasutatavate diferentsiaalsignaalide elektrilised omadused. Täpsemalt nõuab see andmete usaldusväärseks tõlgendamiseks kahe signaaliliini A ja B vahel minimaalset diferentsiaalpinget ±200 mV. Lisaks peaks iga üksiku signaaliliini (A või B) sisendpinge maanduse suhtes jääma vahemikku -7 V kuni +12 V [15].

RS-485 Standard võimaldab kahesuunalist jadasidet mitme seadmega võrgus, kasutades juhtme keerdpaari ülekandeliinina. *Driver* kodeerib ühe jadaside sisendi kaheks signaaliks, muutes need üksteisele vastupidisteks. Seejärel *receiver*, jälgides signaalide potentsiaalide vahet, rekonstrueerib originaalsignaali. Seda tüüpi signaali kasutamisel on omad eelised. Kuna originaalsignaali rekonstrueerimine toimub alati kahe signaali diferentsiaali järgi, siis on välisteguritel seda raske mõjutada.

RS-485 standard toetab kahte peamist andmeedastusrežiimi: poolduplekс ja täisduplekс.

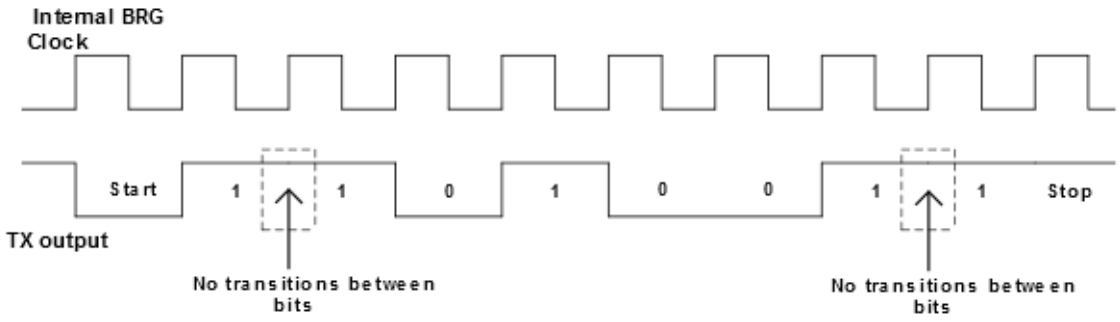
Poolduplekс: Andmeedastusrežiimis kasutatakse ühte keerdpaari kõikide seadmete omavaheliseks ühendamiseks (Joonis 7). Kõik seadmed võivad võrguprotokollil suhelda, võttes endale kas ülem- või alamrolli. Konfliktide vältimiseks saab aga andmeid korraga edastada vaid üks seade, samal ajal kui teised kuulavad [16]. Protokolli nõudeid järgides edastatakse andmeid vaheldumisi, tuginedes seadmete adresseerimisele. Igal seadmel on oma kindel aadress ning ainult adresseeritav seade reageerib andmeliinil olevatele andmetele. Üldjuhul manustatakse aadress vastavalt protokolli nõuetele otse andmeraamidesse.



Joonis 7. Poolduplekс [17]

Täisduplekс: Andmeedastusrežiimis kasutatakse kahte keerdpaari. Tavaliselt on üks ülemseade, mille *driver'i* väljund on ühenduses, ühe keerdpaariga alamseadme *receiver'i* sisenditega ning alamseadmete *driver*'ite väjundid on omakorda ühendatud teise keerdpaariga ülemseadme *receiver*'i sisendiga. Selline seadistus võimaldab andmeid edastada samaaegselt, tagades pideva kahesuunalise kommunikatsiooni [17].

Tüüpiliselt kasutatakse RS485 jadaandmete edastamiseks UART-protokolli. Universaalse asünkroonse vastuvõtja/saatja protokolli puhul ei kasutata eraldi ajastussignaali, vaid andmete kodeerimiseks tuginetakse oma sisemisele kellale, mille edastuskiirus tuleb eelnevalt ühiselt seadistada. Andmeid edastatakse *Non-Return-to-Zero* vormingus, kus kõrge pinge tähistab loogilist '1' ja madal loogilist '0'. Andmete edastus algab tühikäigul jooksva signaali madalaks muutmisega, mida nimetatakse *Start bit*'iks, sellele järgneb 7 – 8 andmebitti, valikuliselt 1 paarsus bit ning lõppeb signaali kõrgeks muutmisega, mida nimetatakse *stop bit*'iks (Joonis 8). [16], [18].



Joonis 8. UART formaat [18]

3.4 Juhtpult

Nõuded tootelt:

- Peab võimaldama vähemalt 4 erinevat sisendsignaali.
- Peab olema ühildatav kasutatava mikrokontrolleriga

Valik:

Retro-stiilse meeleolu tagamiseks sai valitud ka vastav pult. Tegemist on NES puldi [22] odava koopiaga, mis on ühendatav läbi USB pesa. Puldil on kokku 8 nuppu, millest reaalselt kasutatakse viite. Arduino puuduva USB-pesa kompenseerimiseks lisatakse sellele *USB Host shield*, tagades mängupuldi ühendamise võimaluse.

3.5 USB Host shield

Tegemist on Arduino lisamooduliga [21], mis kinnitatakse Arduino peale viikude omavahelise ühendamisega andes nõnda võimaluse ühendada USB-seadmeid mikrokontrolleriga. Seda on võimalik konfigureerida projekti mängupuldi väljundite korrektseks lugemiseks. Kasutamiseks tuleb Arduino'le peale laadida ka USB Host Shield 2.0 teek [23].

3.6 Toiteplokk

Nõuded tootelt:

- Peab pakkuma 24V väljundpinget ja vähemalt 2A voolu

Valik:

Valisin kõige taskukohasema hinnaga toote.

Toiteplokina kasutatakse LPV-60-24[19], millest piisab, et jooksutada nii *Flip-dot* ekraani kui ka Arduino't. Järgneval (Tabel 6) tabelil väljatoodud toiteploki tehnilised andmed.

Tabel 6. Mean Well LPV-60-24 Tehnilised andmed [19]

Sisend pinge	90-264VAC
Väljund pinge	24VDC
Vooluvahemik	0-2.5
Võimsus	60W

3.7 Toitemoodul

Nõue tootelt:

- Peab 24VDC muutma 7-12VDC'ks

Valik:

Valisin esimese kättesaadava komponendi.

Seoses vajadusega Arduino’t varustada toiteallikaga kasutatakse DC konverterit. Täpsemalt on kasutusel toitemoodul DC/DC *step-down* 2.5..40V/1.25..35V 2A [20]. Võimaldab vajaminevat astet 24V pealt 9V peale.

4 Tarkvara arendamine

Tarkvara peaks täpselt emuleerima Tetrise mängu, olles samal ajal piisavalt optimeeritud tõhusaks toimimiseks mäluga mikrokontrolleril. Samuti peaks mäng järgima alampeatükis 2.1.1 välja toodud piiranguid ja reegleid.

4.1 Programmeerimiskeel ja -keskkond

Kuna koodi arendus toimub Arduino põhjal, piirdub programmeerimiskeel peamiselt C++ või täpsemalt C++ lihtsustatud versiooniga, mille lisafunktsioonid on kohandatud Arduino kontrollimise jaoks. C++ kasutamine võimaldab ühilduvust '*Object-Oriented Programming*' mudeliga, mida kasutatakse kogu koodi loomise protsessis. See tagab paremini loetava ja hooldatava koodi, muutes vigade tuvastamise ja kõrvaldamise lihtsamaks.

Arduino'l on programmeerimiseks ja koodi oma seadmetesse üleslaadimiseks enda integreeritud arenduskeskkond. Kuigi antud IDE on funktsionaalne valik, otsustasin kirjutada koodi *Visual Studio Code*'is kasutades Platform.io raamistikku, mis toetab Arduino API-t ja võimaldab näiteks koodi otse Arduino mikrokontrollerile mugavalt kompileerida ning laadida [24]. Lisaks pakub *Visual Studio Code* reaalajas vigade tuvastamist, parandades Arduino IDE'ga võrreldes arenduse sujuvust.

4.2 Üldarhitektuur

Süsteem koosneb mitmest üksteisega suhtlevast moodulist. Iga moodul vastutab konkreetse aspekti eest, tagades tõhusa toimimise. Allpool on ülevaade põhikomponentidest:

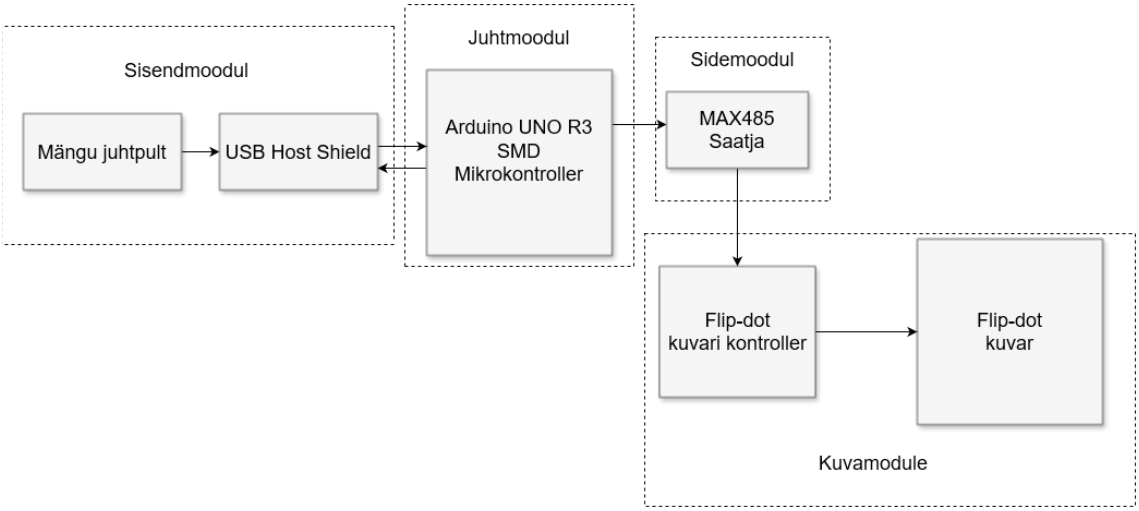
1. Sisendmoodul: Väljastab mängupuldi nupuvajutuste andmeid.

2. Sidemoodul: Juhtmooduli poolt saadetud andmete teisendamine RS-485 standard nõuetele ning nende saatmine andmeliinile.

3. Juhtmoodul: Antud moodul vastutab kogu mängu loogika eest, muutes mängu olekut vastavalt sisendmooduli poolsetele käsklustele, varustab sidemoodulit korrektse mängu olekuga ning kontrollib vajadusel mängu lõpetamist ja taaskäivitamist.

4. Kuvamoodul: Eraldiseisev süsteem, mis võtab vastu sidemooduli poolt saadetavad andmed ning vastutab nende andmete tõlkimise ja kajastamise eest.

Moodulite vahelised ühendused esitatud ka järgneva plokkskeemiga (Joonis 9).



Joonis 9: Moodulite plokkskeem

4.3 Tarkvara klassid

Mäng koosneb viiest erinevast klassist, millest igaüks sisaldab oma loogikat:

Grid: Haldab mänguruudustikku, kehtestades ruudustiku piirangud, kuvades aktiivseid tükke ja salvestades lukustatud tükid passiivsesse ruudustikusse. Aktiivne ja passiivne ruudustik määravad koos mängu oleku. Aktiivset ruudustikku värskendatakse igal korral, kui tükke liigutatakse või pööratakse.

TetrominoBlock: Majutab aktiivset tükki ning haldab selle liikumist ja pöörlemist. Samuti hõlbustab see vajadusel tükkide vahetamist. Vahetamisel valitakse tükk juhuslikult.

Controller: Seadistab vastavalt USB host shield 2.0 teegi järgi ennast parsijaks ning seejärel tegeleb mängupuldi andmete korrektse lugemisega, pakkudes ligipääsu töödeldud andmetele. Kood baseerub suuresti teegiga kaasa tulnud näidetele [23].

GameLogic: Toimib põhikomponendina, mis seob kõik programmikoodi klassid omavahel, kutsudes esile vastavate klasside meetodeid. See klass asetab tükke ruudustikule ja juhib plokkide liikumist mängija sisendi põhjal, kontrollib piiridest väljumise tingimusi, juppide omavahelist haakumist, haldab mängu lõppemise stsenaariume ja taaskäivitab mängu. Samuti vastutab see mängu oleku andmete teisendamise ja saatmise eest Display klassile.

Display: Teisendab mängu oleku andmeid õigesse formaati ning seejärel saadab MAX485 kaudu need andmeliinile.

Klassi diagramm lisatud: Lisa 3 – Klassi diagramm.

4.4 Mängu kulg

Mängu kulg on üsna lihtne. Mäng töötab lõputu tsüklina, seni kuni lõppemise olekut pole väljastatud. See tsükkel on jaotatud kolmeks kihiks, millest igaüks toimib erinevate intervallidega, tagamaks sisendi ja mängu vahel sujuvat suhtlust.

1. kiht on sisendi lugemiseks ja selle salvestamiseks.

2. kiht on sisenditöötlus. Töötleb 1. kihist salvestatud sisendit ja sooritab mängija käskude alusel vastavad toimingud.

3. kiht tagab sunnitud liikumise. Sõltumata kasutaja sisendist, liigutatakse aktiivset *tetromino* tükki automaatselt mänguvälja põhja suunas, hoides mängu dünaamilise ja väljakutsuvana. Antud kihi intervalli pikkust muudetakse vastavalt mängu tasemele.

Vooskeem lisatud: Lisa 2 – Vooskeem

4.5 Väljakutsed ja lahendused

Arduino vähese mälu olemus esitas tetrise mängu arendamisel mitmeid väljakutseid. Omades ainult 2KB SRAM-i ning vajadus kiire ja tõhusa täitmiskiiruse järele, pidi programm mängu oleku ja loogika salvestamiseks peamiselt SRAM-ile toetuma, mis tagas kiirema andmete hankimise.

Esimene väljakutse hõlmas kolmemõõtmeliste massiivide kasutamist *tetromino* ruutude kõikvõimalikke positsioonide salvestamiseks (Joonis 10), millega sai Arduino mälu kiirelt täis. Optimeerimiseks läksin üle kahemõõtmelistele massiividele, kasutades ruutude positsioonide kodeerimiseks biti vormingut (Joonis 11). See töötas alguses, kuid ebaõnnestus neljanda tüki definitsiooni lisamisel SRAM-i piirangute tõttu. Lahendusena kasutasin PROGMEM direktiivi, et salvestada globaalsete *tetromino* tükkide definitsioonid püsimällu. Kuna uusi tükke on vaja ainult siis, kui aktiivne tükk lukustatakse, ei tohiks suhteliselt harv juurdepääs püsimällu negatiivselt mõjutada mängitavust.

Järgnevalt (Tabel 7) on näidatud eelpool mainitud optimeerimis lahendust.

Tabel 7. Kolmemõõtmelise masiivi optimeerimine

```
static const byte L[4][4][4] = {
        {
            {0, 0, 1, 0},
            {1, 1, 1, 0},
            {0, 0, 0, 0},
            {0, 0, 0, 0}
        },
        {
            {0, 1, 0, 0},
            {0, 1, 0, 0},
            {0, 1, 1, 0},
            {0, 0, 0, 0}
        },
        {
            {0, 0, 0, 0},
            {0, 1, 1, 1},
            {0, 1, 0, 0},
            {0, 0, 0, 0}
        },
        {
            {0, 1, 1, 0},
            {0, 0, 1, 0},
            {0, 0, 1, 0},
            {0, 0, 0, 0}
        }
    };
```

Joonis 10: Esialgne ressursimahukas massiiv

```
static const byte L[4][4] PROGMEM
= {
    {0b0010,0b1110,0b0000,0b0000},
    {0b0100,0b0100,0b0110,0b0000},
    {0b0000,0b1110,0b1000,0b0000},
    {0b1100,0b0100,0b0100,0b0000}
};
```

Joonis 11: Optimeeritud massiiv

Silumise vältel tekkis veel üks probleem. Saadaoleva mälu jälgimiseks printisin selle väärtuse pärast iga tsükli kordamist. See tundus esialgu kasulik, kuna see näitas piisavalt ruumi täiendavate globaalsete muutujate jaoks. Sellest hoolimata jooksis mäng peale neljanda tüki definitsiooni lisamist kokku. Probleem seisnes selles, et prinditud mälu peegeldas *heap* ja *stack* vahelist ruumi alles pärast seda, kui ajutised muutujad, mis loodi funktsioonide sees iga tsükli kordamise ajal, *stack*'ist kustutati. See tähendab, et vabastati mälu, mis oli ajutiste muutujate ja funktsiooni kutsete poolt hõivatud tekitades mulle valearusaama, et Arduino'l oli piisavalt mälu täiendavate definitsioonide lisamiseks. Mainitud probleem toob esile tööajal programmi mälu kasutamise arvestamise tähtsuse.

Süsteemi testimise käigus selgus, et mängupult põhjustas vahetevahel kogu süteemi tardumise paariks sekundiks. Tardumiste vaheline aeg oli ebakorrapärane: vahepeal jooksis mäng probleemideta minuteid, teine kord esines tõrge tihedamini. Samuti ei õnnestunud viga kuidagi ise esile kutsuda, ei leidnud ühtegi seost puldi sisenditel ja

tardumisel. Kuna algselt testitud analoogpulgaga sääraseid probleeme ei tekkinud, siis järeldan, et probleem seisnes üliodava hinnaga puldis.

Kuvari jaoks oli vaja koostada korrektsed andmepaketid näitamaks täpset Tetrise mänguolekut. Pakettide koostamine toimus Display klassi sees, kus võeti vastu GameLogic klassi poolt saadetud 10-biti pikkused andmed masiivi *newData* kujul. Antud andmeid tuli jagada kahe kuvari paneeli vahel. Selleks koostasin, toetudes tootja andmelehele(Tabel 4), kaks erinevat masiivi: *dataScreen1* ja *dataScreen2*. Masiivi esimene element on paketi päis 0x80, teine element sisaldab endas käsutüüpi, kolmas näitab kuvari paneeli aadressi, millele järgneb 28 andmebaiti ning lõpetuseks viimane element sisaldab lõpubaiti.

Paneelid on jagatud kaheks võrdseks osaks, kus mõlemad paneelid kontrollivad neile ettenähtud 28x7 *flip-dot* indikaatoreid. Kuna Tetrise mänguvälja mõõtmed on 20x10, siis vastavalt sellele jaotasin pakettide koostamisel *newData* andmed kaheks bitinihke abil, salvestades kõrgemad 7 andmebiti väärtused õigetele dataScreen1 ridadele ja madalamad 3 õigetele dataScreen2 ridadele. Utiliseerisin bitimaske, et tagada juurdepääs ainult soovitud bittidele. Seejärel saadetakse vastavalt koodi käsklustele andmed edastusliinile, mis peaksid jõudma kuvarini, kus õigete andmete tõlgendamisel kajastub kuvaril korrektne mänguolek. Allpool on näidatud paketi komplekteerimine koodis (Joonis 12).

```
  // Header :
dataScreen1[0] = 0x80;
dataScreen2[0] = 0x80;
// Command :
dataScreen1[1] = 0x83;
dataScreen2[1] = 0x83;
// Address :
dataScreen1[2] = 0x01;
dataScreen2[2] = 0x02;

// Screen Data Pack. Just some "beautification" of the display :
for(int i = 0; i < 8; i++) {
    dataScreen1[i+3] = ~(info1[i]) & dataMask1;
    dataScreen2[i+3] = ~(info2[i]) & dataMask1;
}

// Actual Game grid value reflection :
for(int i = 0; i < 20; i++){
    uint16_t temp = newData[i] >> 3;
    dataScreen1[i+11] = (temp & dataMask1);
    temp = newData[i] << 4;
    dataScreen2[i+11] &= 0x0F;
    dataScreen2[i+11] |= (temp & dataMask2);
}

// End bytes :
dataScreen1[31] = 0x8F;
dataScreen2[31] = 0x8F;
```

Joonis 12: Kuvari andmepaketti koostamine koodis

4.6 Optimeerimine

Paljud harjumused, mis tulenesid arvutirakenduste arendamisest, tõid kaasa üsnagi suure vajaduse optimeerimisele. Esiteks asendati võimalusel suuremad andmetüübid väiksemate vastu. Masiivide suurusi vähendati, kus võimalik. Püsimällu salvestati võimalikult vähe andmeid, kuid kui suuremate masiivide puhul see nõudmine tekkis, tehti seda PROGMEM märksõna alusel. Lisaks kui masiivide andmeid oli vaja kopeerida, utiliseeriti memcpy() käsklust.

Optimeerimise käigus õppisin paljutki ning eriti huvi pakkus biti-kujuliste andmete töötlemine. Kõikide parandustega võideti ~30% SRAM mälu, mis on küllaltki märkimisväärne. Vajadusel oleks võimalik programmikoodi veelgi optimeerida, peegeldades globaalsete muutujate määratlusi, mille arvelt vähendatakse liigsete bitinihete kasutamist. Lisan mõned lahendused ka Lisa 4 – Muud optimeeringud nimekirja.

5 Kokkupanek

Jagasin kokkupaneku kolme etappi: Riisvara vahelised ühendused, süsteemi testimine ning tulemused. Riistvara komplekteeriti kaasaskantavuse tagamiseks puidust korpusesse.

5.1 Riisvara vahelised ühendused

Ühendustele läheneti järgnevalt:

- Arduino mikrokontrollerile laaditakse koostatud Tetrisemängu tarkvara.
- Kuvari 24V sisendid ühendatakse toiteploki LPV-60 24V väljundiga. Antud 2 paneeliga kuvar vajab voolutarvet kuni 1A paneeli kohta, mida kasutatud toiteplokk katab. Tabel 6.
- Arduino’t varustatakse 9V sisendiga kasutades DC/DC konverterit, mis teisendab 24V väljundi 9V peale.
- MAX485 edastusmoodul ühendatakse Arduino’ga järgnevalt: MAX485 DI viik UNO Tx viiguga, MAX485 DE, RE ja VCC viigud UNO 5V viiguga, MAX485 GND viik UNO GND viiguga. Seejärel ühendatakse MAX485 A ja B signaalid mõlema *flip-dot* kontrolleri +485 ja -485 jadaliiniga, luues pooldupleks ühenduse Arduino mikrokontrolleri ja *flip-dot* kuvari vahel. Seoses mittevajamineva kahepoolse infoedastusega, seadistatakse UNO ainult edastajaks.
- *Flip-dot* kuvari paneelidele antakse aadressid binaarses koodis. Paneel 1 saab aadressiks 1 ja paneel 2 saab aadressiks 2. See konfiguratsiooni saavutatakse 8DIP lüliti lülitamisel. Lisa 6 – Dip lülitite seadistus.
- Kuvari paneelide edastuskiirus seadistatakse 57600 bit/s 3DIP lüliti alusel, järgides tootja andmelehel olevaid andmeid.

- Arduino varustatakse *USB Host Shield*'iga, seejärel ühendatakse mängupult läbi USB pesa.
- Süsteem lülitatakse sisse läbi pistikupesa.

Komponentide omavahelised ühendused esitatud ka elektriskeemina (Joonis 16) peatükis Lisa 5 – Elektriskeem.

5.2 Süsteemi testimine

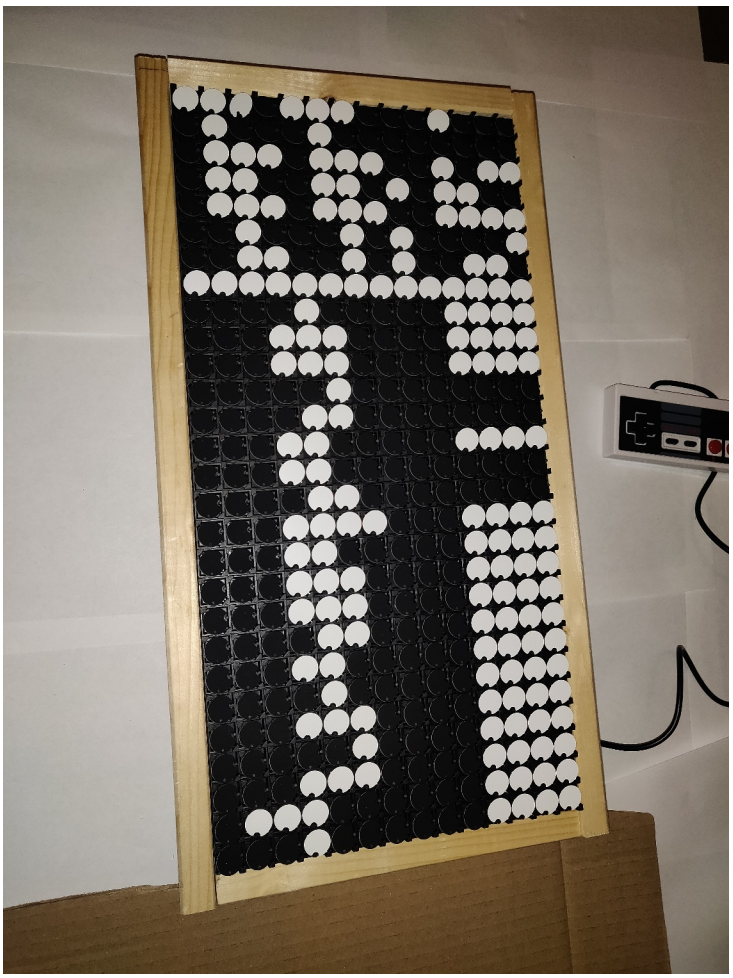
Süsteemi testitakse oodatud tulemuste saamiseks. Kontrollimine toimub käsitsi läbiproovimise alusel, testitakse kõiki mängu reegleid ja piiranguid.

Kontrollitakse mängutükkide liikumist ja pööramist, veendutakse mängupiiride olemasolus, uute tükkide genereerimises, sunnitud liikumises, ridade vabastamises, järgmise tüki kuvamises selleks ettenähtud aknas, tükkide lukustamises ning mängu õigeaegsest lõppemisest ja taaskäivitamisest.

5.3 Tulemused

Kokkupaneku tulemusena valmis ootustele vastav mehaanilise väljundiga töötav süsteem. Arduino mikrokontroller kajastab korrektselt Tetrise mänguolekut elektromehaanilisele *flip-dot* kuvarile mängupuldi sisendite alusel.

Süsteem pandi kokku puidust korpusesse ning on mängimiskõlbilk (Joonis 13).



Joonis 13: Komplekteeritud süsteem

6 Kokkuvõte

Käesolevas töös kajastati riistvara valikuid, seletades lahti *flip-dot* kuvari, Arduino mikrokontrolleri, nende jaoks vajaminevad toiteallikad, MAX485 edastusmooduli ja kasutaja jaoks mängupuldi ning tagati nende omavaheline kokkusobivus.

Samuti kirjeldati Tetrise tarkvara põhifunktsionaalsust, piiratud mäluga kaasnevaid optimeeringuid, kuvari jaoks korrektsete andmete ümber vormistamist ning nende andmete baasil õigete pakettide koostamist ja andmeliinile saatmist.

Lõpetuseks komplekteeriti kõik riistvara komponendid olles taganud nende omavahelise ühilduvuse. Lisaks viidi läbi põhjalik testimine, tagamaks mängusüsteemi terviklik töötamine.

Lõputöö tulemusena valmis Arduino mikrokontrollerile optimeeritud programmikoodil jooksev *flip-dot* kuvaril kajastuv puldiga juhitav elektromehaaniline Tetrisemängu prototüüp, mis on kasutaja poolt mängitav.

Kasutatud kirjandus

[1] “About Tetris” Tetris. Kasutatud: Mai. 2, 2024 [Võrgus.] Saadaval: https://tetris.com/about-us

[2] “Tetris E60” Tetris. Kasutatud: Mai. 9, 2024 [Võrgus.] Saadaval: https://tetris.com/tetris-e60/

[3] “Flip-Dot Boards XY5” Alfa-Zeta. Kasutatud: Aprill. 2, 2024 [Võrgus.] Saadaval: https://flipdots.com/en/products-services/flip-dot-boards-xy5/

[4] J. Tucker “The Wireless Age for Digital Destination Signage Arrives” Metro magazine. Kasutatud: Mai. 5, 2024 [Võrgus.] Saadaval: https://www.metro-magazine.com/10007970/the-wireless-age-for-digital-destination-signage-arrives

[5] Flip Dot Display Element using Electromagnet and Assembly Module thereof by 전병삼. (2015, Feb. 4) Patent KR101487904B1 [Võrgus] Saadaval: https://patents.google.com/patent/KR101487904B1/en

[6] “How it works” Alfa-Zeta. Kasutatud: Mai. 2, 2024 [Võrgus.] Saadaval: https://flipdots.com/en/electromagnetic-flip-disc-technology-how-it-works/

[7] Frederic L “Matrix layout and controller's job” Hackaday.io. Kasutatud: Mai. 3, 2024 [Võrgus.] Saadaval: https://hackaday.io/project/159415-flip-dot-display-diy-controller/log/149305-matrix-layout-and-controllers-job

[8] Andmeleht, XY FLIPDOT PANELS USER MANUAL, Alfa-Zeta, PL. Saadetud meili teel. [Võrgus.] Saadaval : https://github.com/alekssavioja/Tetris_Flipdot

[9] Andmeleht, ESP32-WROOM-32, https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[10] Andmeleht, STM32F103C8T6 Blue Pill, https://stm32-base.org/assets/pdf/regulators/TX6211B.pdf

[11] Arduino UNO Rev 3 SMD, https://store.arduino.cc/products/arduino-uno-rev3-smd

[12] Andmeleht, Arduino UNO Rev 3 SMD, https://docs.arduino.cc/hardware/uno-rev3-smd/#tech-specs

[13] Arduino Memory Guide, Arduino.cc, Kasutatud: Mai. 5, 2024 [Võrgus.] Saadaval: https://docs.arduino.cc/learn/programming/memory-guide/

[14] MAX485 Moodul, https://hobbycomponents.com/wired-wireless/663-max485-rs485-transceiver-module

[15] RS-485, Electrical Characteristic of Henerators and Receivers for Use in Balanced Digital Multipoint Systems, TIA/EIA-485-A, March 3, 1998 https://www.mikrocontroller.net/attachment/428561/eia485.pdf

[16] J. Kelly "RS-485 Serial Interface Explained" Cuidevices.com. Kasutatud: Mai. 6, 2024 [Võrgus.] Saadaval: https://www.cuidevices.com/blog/rs-485-serial-interface-explained

[17] K. Wang, H. Liu "RS-485 Basics Series" ti.com. Kasutatud: Mai. 6, 2024 [Võrgus.] Saadaval: https://www.ti.com/lit/wp/slla545/slla545.pdf?ts=1715249872037

[18] M. Wyman, C. Best, Microchip Technology Inc. Kasutatud: Mai. 6, 2024 [Võrgus.] Saadaval: https://onlinedocs.microchip.com/pr/GUID-167CA20A-2C0F-4CBC-A693-9FD032B9B193-en-US-1/index.html?GUID-39D5E45F-A0F3-4832-AFEC-DD769377CA0A

[19] Toiteplokk Mean Well LPV-60-24 [Võrgus.] Saadaval: https://www.oomipood.ee/product/lpv_60_24_toiteplokk_24vdc_2_5a_60w_ip67_mean_well

[20] Toitemoodul DC/DC step-down 2.5..40V/1.25..35V 2A LED indikaatoriga [Võrgus.] Saadaval: https://www.oomipood.ee/product/oky3502_1_toitemoodul_dc_dc_step_down_2_5_40v_1_25_35v_2a_led_indik

[21] USB Host Shield Arduino'le [Võrgus.] Saadaval: https://www.tinytronics.nl/en/communication-and-signals/serial/usb/usb-host-shield-arduino-compatible

[22] NES USB mängupult [Võrgus.] Saadaval: https://www.aliexpress.com/i/4000623559149.html

[23] O. Mazurov, A. Glushchenko, K.S. Lauszus, A. Kroll, guruthree, Y. Akagawa, A. Vink. USB Host shield 2.0 library. Github [Võrgus.] Saadaval: https://github.com/felis/USB_Host_Shield_2.0

[24] Platform.io [Võrgus.] Saadaval: https://docs.platformio.org/en/latest/frameworks/arduino.html

[25] Projekti lähtekood [Võrgus.] Saadaval: https://github.com/alekssavioja/Tetris_Flipdot

Lisa 1– Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Aleks Savioja

1 Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Elektromehaaniline Tetris" mille juhendaja on Peeter Ellervee

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

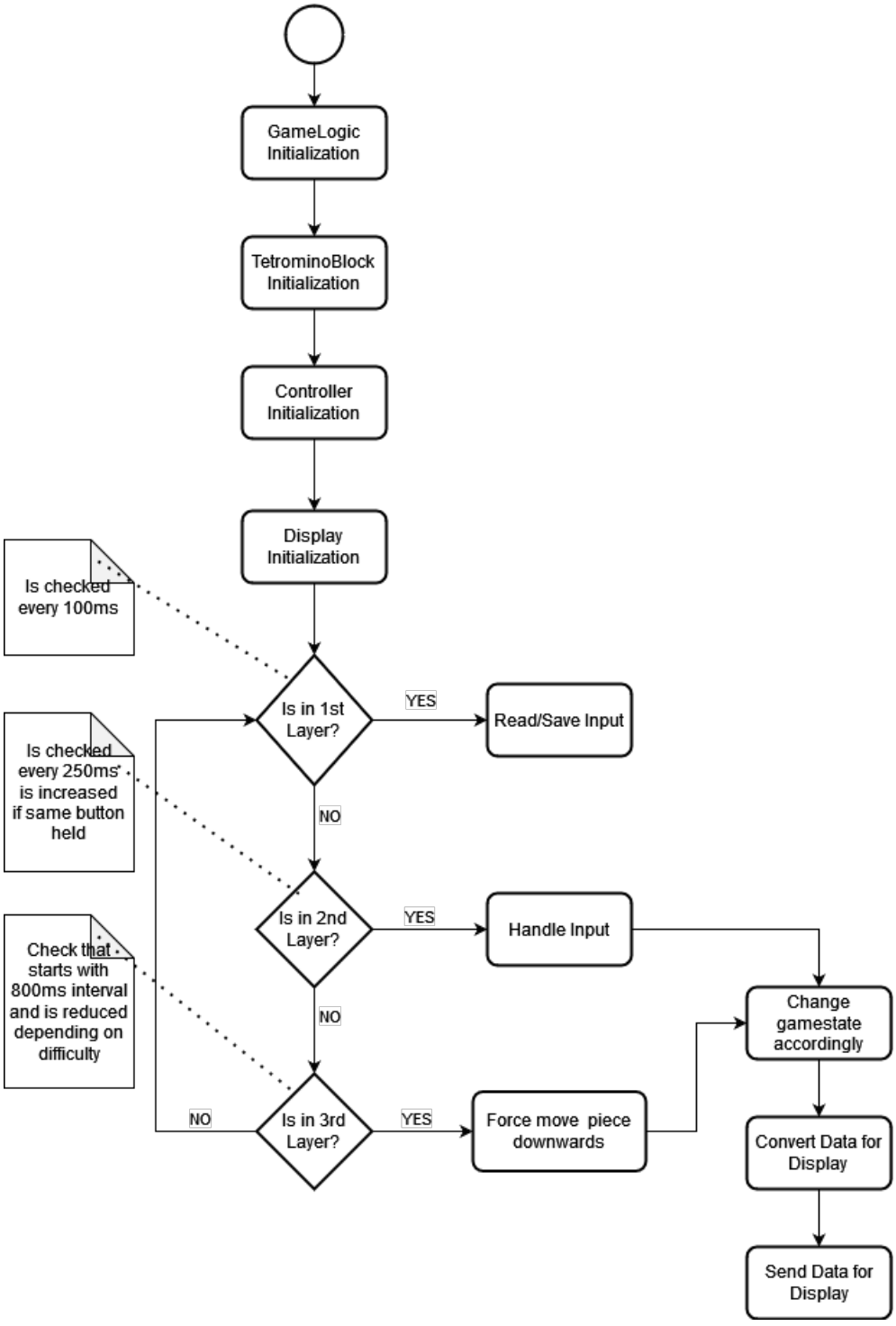
2 Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3 Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

13.05.2024

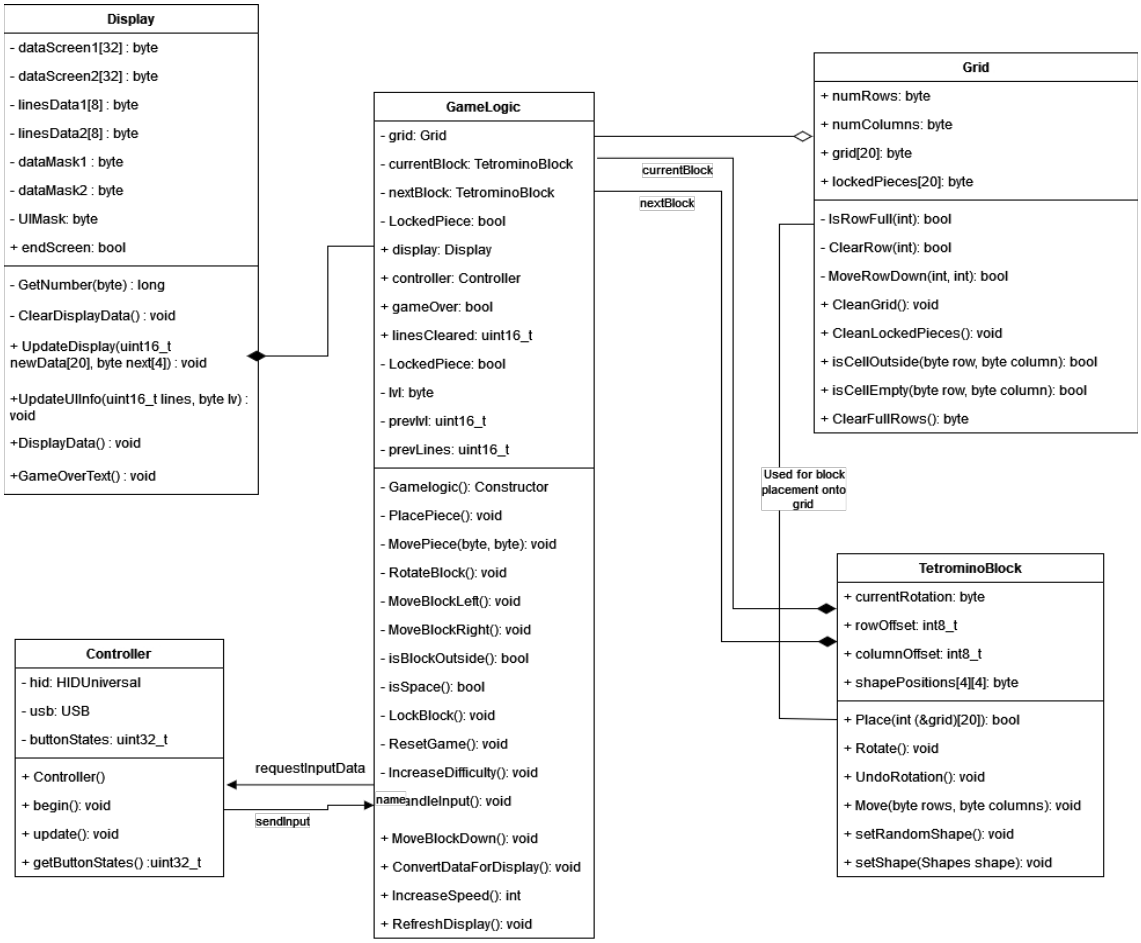
1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Vooskeem



Joonis 14: Vooskeem

Lisa 3 – Klassi diagramm



Joonis 15: Klassi diagramm

Lisa 4 – Muud optimeeringud

Vanad näited:

```
byte grid[20][10];
// == 200bytes == 1600bit
byte lockedPieces[20][10];
// == 200bytes == 1600bit
byte shapePositions[4][4][4];
// == 64bytes == 512bit
```

// Tükkide lisamine mänguväljakule.

```
bool TetrominoBlock::Place(byte (&grid)[20][10])
{
    for (byte row = 0; row < 4; row++)
    {
        for (byte column = 0; column < 4; column++)
        {
            if (shapePositions[currentRotation][row][column] == 1)
            {
                grid[row + rowOffset][column + columnOffset] = 1;
            }
        }
    }
    return true;
}
```

// Aktiivse tüki koordinaatide leidmine

```
void GameLogic::ConvertDataForDisplay()
{
    uint16_t displayData[20] = {0};

    for (byte x = 0; x < grid.numRows; x++)
    {
        for (int y = 0; y < grid.numColumns; y++)
        {
            if (grid.grid[x][y] || grid.lockedPieces[x][y])
            {
                displayData[x] |= 1 << (grid.numColumns - 1 - y);
            }
        }
    }

    byte nextBlockDATA[4] = {0};
    for (byte i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if(nextBlock.shapePositions
              [nextBlock.currentRotation][i][j]== 1)
```

```
            {
                nextBlockDATA[i + 1] |= (1 << (3 - j));
            }
        }
    }
    display.UpdateDisplay(displayData, nextBlockDATA);
}


// Mänguoleku teisendamine kuvarile sobivateks andmeteks

void GameLogic::ConvertDataForDisplay()
{
    uint16_t displayData[20] = {0};

    for (byte x = 0; x < grid.numRows; x++)
    {
        for (int y = 0; y < grid.numColumns; y++)
        {
            if (grid.grid[x][y] || grid.lockedPieces[x][y])
            {
                displayData[x] |= 1 << (grid.numColumns - 1 - y);
            }
        }
    }

    byte nextBlockDATA[4] = {0};
    for (byte i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if(nextBlock.shapePositions
                  [nextBlock.currentRotation][i][j] == 1)
            {
                nextBlockDATA[i + 1] |= (1 << (3 - j));
            }
        }
    }
    display.UpdateDisplay(displayData, nextBlockDATA);
}
```

Uued näited:

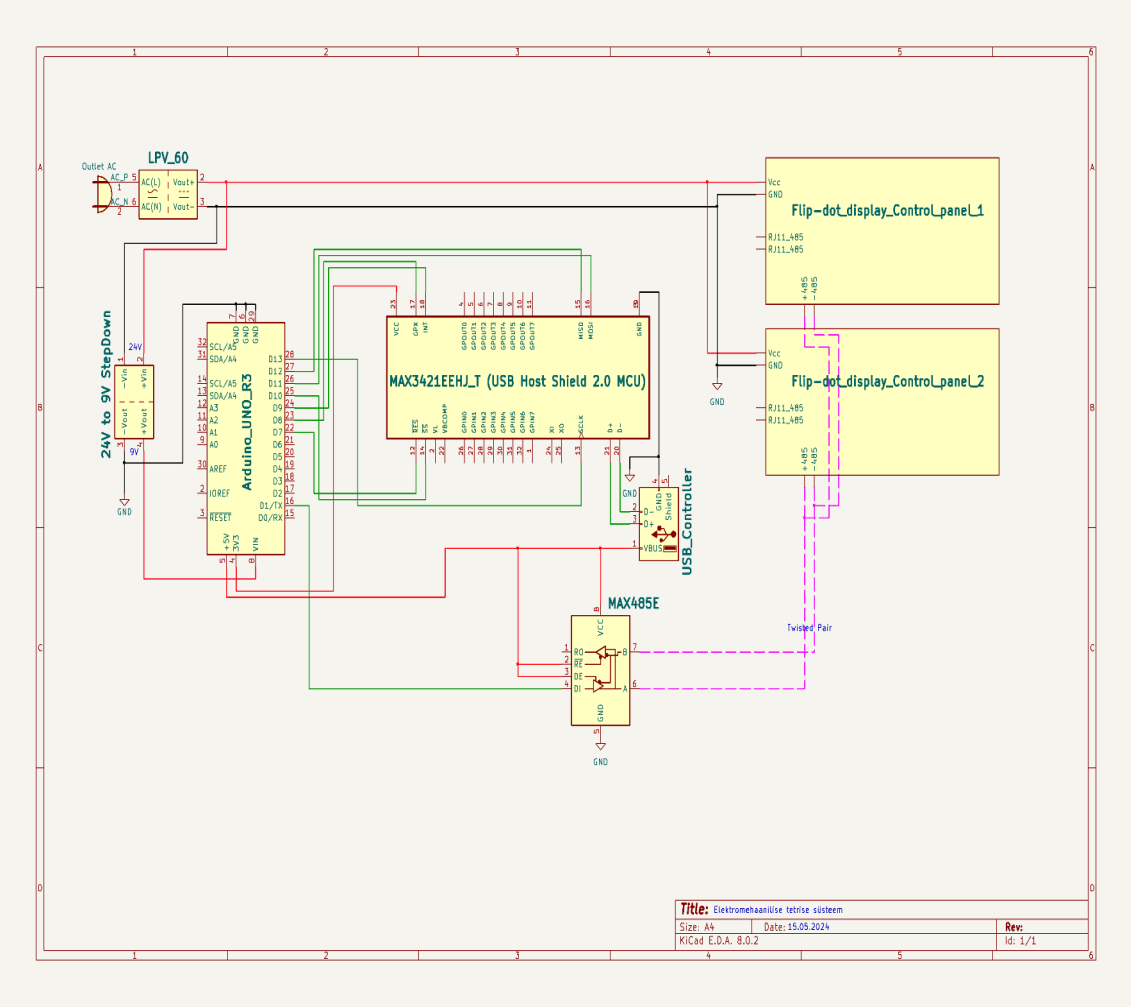
```
int grid[20];
// == 40bytes == 320bit
byte lockedPieces[20];
// == 40bytes == 320bit
byte shapePositions[4][4];
// == 16bytes == 128bit
```

// Tükkide lisamine mänguväljakule.

```
bool TetrominoBlock::Place(int (&grid)[20])
{
    for (int8_t row = 0; row < 4; row++)
    {
      for (int8_t column = 0; column < 4; column++)
        {
          if ( (shapePositions[currentRotation][row] &
                  (1 << (3 - column)) ) >= 1 )
```

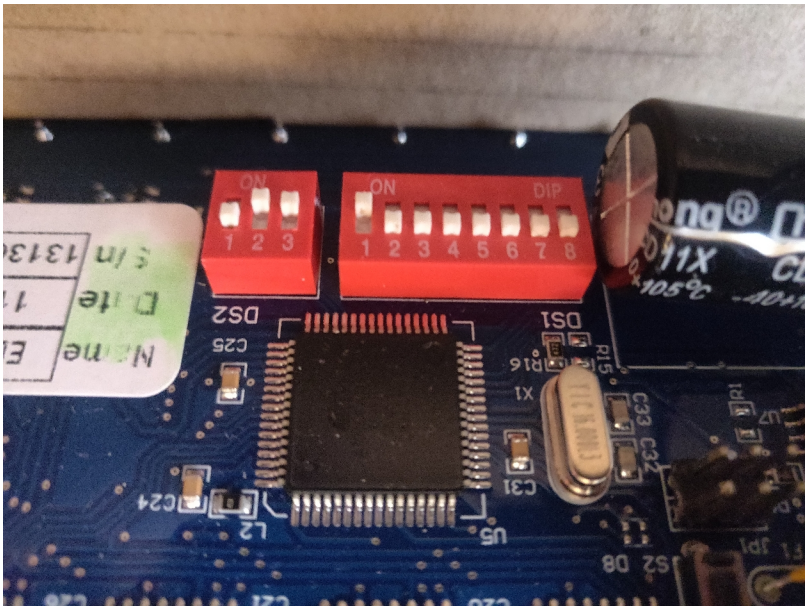
```
            {
                grid[row + rowOffset] |=
                  (1 << (9 - column - columnOffset));
            }
        }
    }
    return true;
}

// Aktiivse tüki koordinaatide leidmine

if (currentBlock.shapePositions[currentBlock.currentRotation][i] &
   (1 << (3 - j))) > 0)
{
    x = i + currentBlock.rowOffset;
    y = j + currentBlock.columnOffset;
}

// Mänguoleku teisendamine kuvarile sobivateks andmeteks

void GameLogic::ConvertDataForDisplay()
{
    uint16_t displayData[20] = {0};
    for(byte row = 0; row < grid.numRows; row++)
    {
        displayData[row] |= (grid.grid[row] | grid.lockedPieces[row]);
    }

    byte nextBlockDATA[4] = {0};
    for (byte i = 0; i < 4; i++)
    {
        nextBlockDATA[i] |=
        nextBlock.shapePositions[nextBlock.currentRotation][i];
    }

    display.UpdateDisplay(displayData, nextBlockDATA);
}
```

Lisa 5 – Elektriskeem

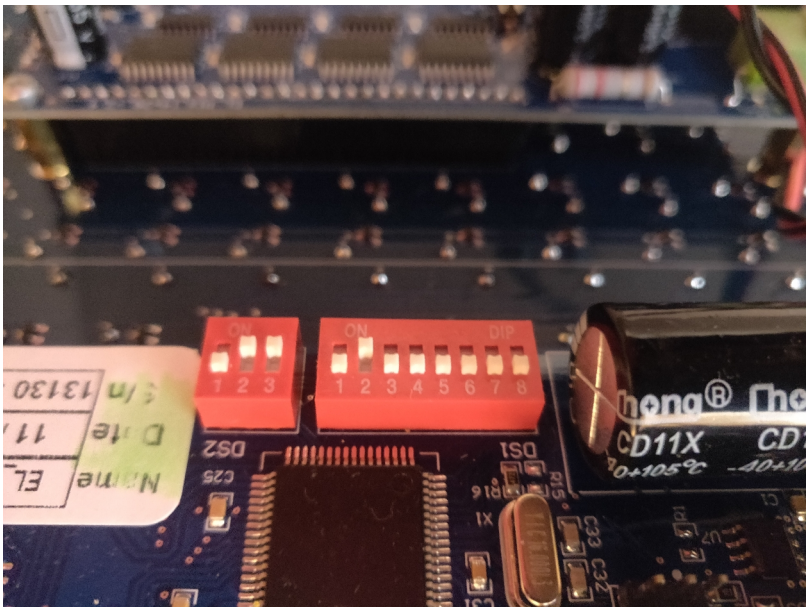


Joonis 16: Komponentide omavahelised ühendused

Lisa 6 – Dip lülitite seadistus



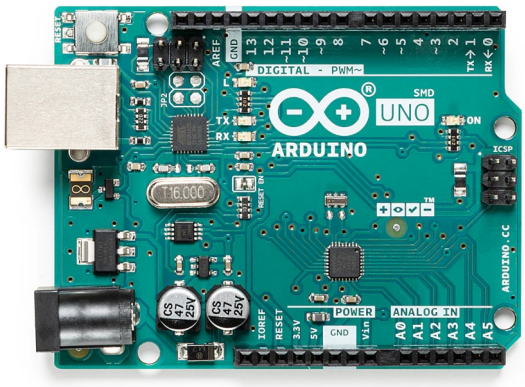
Joonis 17: Esimese paneeli seadistus - aadress: 0x01, edastuskiirus: 57600



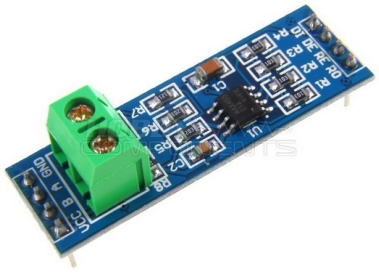
Joonis 18: Teise paneeli seadistus - aadress: 0x02, edastuskiirus: 57600

Lisa 7 – Riistvara pildid

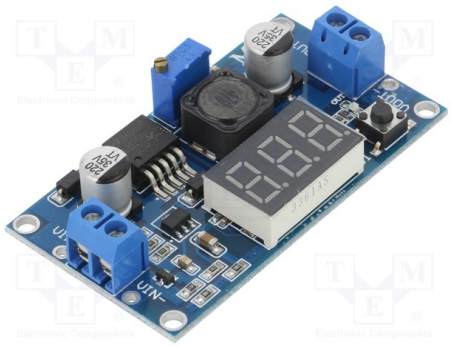
Joonis 19. Mean Well LPV-60-24 [19]



Joonis 20. Arudino UNO rev3 SMD [11]



Joonis 21: MAX485 moodul [14]



Joonis 22. Toitemoodul DC/DC step-down 2.5..40V/1.25..35V 2A [20]

Joonis 23: USB Host shield 2.0 [21]

Joonis 24: USB NES mängupult [22]