

Střední průmyslová škola a Vyšší odborná škola, Písek, Karla Čapka 402, Písek

18-20-M/01 Informační technologie

## Maturitní práce

**Elektromechanická hra ->**

**samořídící šachovnice**

Téma číslo 4.

autor:

**Václav Zíka, B4.I**

vedoucí maturitní práce:

**Mgr. Milan Janoušek**

Písek 2024/2025

# **Licenční smlouva o podmírkách užití školního díla**

ve smyslu zákona č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění (dále jen „AZ“), uzavřená mezi smluvními stranami:

## **1. Autor práce: Václav Zíka**

bytem Na Spravedlnosti 974/36, Písek 39701, Česká republika  
datum narození: 29. 12. 2005  
(dále jen „autor“)

**a**

## **2. Nabyvatel: Střední průmyslová škola a Vyšší odborná škola, Písek, Karla Čapka 402, Písek**

397 11 Písek, Karla Čapka 402  
(dále jen "nabyvatel")  
zastoupená ředitelem školy: Ing. Jiří Uhlík

## **Článek 1**

### **Vymezení pojmu**

- 1.1 Školním dílem dle §60 AZ se pro účely této smlouvy rozumí dílo vytvořené žákem/studentem ke splnění školních nebo studijních povinností vyplývajících z jeho právního postavení ke škole.
- 1.2 Licencí se pro účely této smlouvy rozumí oprávnění k výkonu práva školní dílo užít v rozsahu a za podmínek dále stanovených.

## **Článek 2**

### **Dílo**

- 2.1 Předmětem této smlouvy je poskytnutí licence k užití školního díla – maturitní práce.

Název práce (dále jen „dílo“): Elektromechanická hra -> samořídící šachovnice  
vedoucí práce: Mgr. Milan Janoušek  
odevzdané nabývateli v elektronické formě dne 31. 3. 2025.

2.2 Autor prohlašuje, že:

- vytvořil dílo, specifikované touto smlouvou, samostatnou vlastní tvůrčí činností;
- při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími;
- dílo je dílem původním;
- neposkytl třetí osobě výhradní oprávnění k užití díla v rozsahu licence poskytnuté nabývateli dle této smlouvy před podpisem této smlouvy;
- je si vědom, že před zamýšleným poskytnutím výhradního oprávnění k užití díla v rozsahu licence poskytnuté nabývateli dle této smlouvy třetí osobě, je povinen informovat tuto třetí osobu o skutečnosti, že již poskytl nevýhradní licenci k užití díla nabývateli.

2.3 Dílo je chráněno jako dílo dle autorského zákona a dle zákona č. 89/2012 Sb., občanský zákoník (dále jen „Občanský zákoník“).

### Článek 3

#### Poskytnutí licence

3.1 Licenční smlouvou autor poskytuje nabývateli oprávnění k výkonu práva dílo užít pro účely výuky na Střední průmyslové škole a Vyšší odborné škole, Písek, Karla Čapka 402, a pro vnitřní potřebu školy, ze které neplyne škole hospodářský výsledek.

3.2 Licence je poskytována pro celou dobu trvání autorských a majetkových práv k dílu.

3.3 Autor poskytuje nabývateli oprávnění užít dílo způsoby podle 3.1 neomezeně.

3.4 Autor poskytuje nabývateli oprávnění užít dílo bezúplatně za splnění podmínky, že nabývatel nebude užívat dílo za účelem dosažení zisku a nebude-li v budoucnu dohodnuto písemně jinak.

## **Článek 4**

### **Údaje o autorství**

4.1 Nabyvatel se zavazuje, že uvede údaje o autorství autora dle Autorského zákona.

## **Článek 5**

### **Poskytnutí licence**

5.1 Pokud to není v rozporu s oprávněnými zájmy nabyvatele, licence je poskytována jako nevýhradní. Nabyvatel je oprávněn postoupit tuto licenci třetí osobě a udělovat podlicence za splnění podmínek uvedených v § 48 Autorského zákona.

5.2 Autor může své dílo užít či poskytnout jinému licenci, není-li to v rozporu s oprávněnými zájmy nabyvatele, za podmínky, že nabyvatel (dle této licenční smlouvy) je oprávněn po autoru školního díla požadovat, aby přiměřeně přispěl na úhradu nákladů, tak, jak je stanoveno v § 60 odst. 3 zákona Autorského zákona.

5.3 Nabyvatel není povinen dílo užít.

5.4 Nabyvatel je oprávněn dílo spojovat s jinými díly i zařadit dílo do díla souborného. Autor dává svolení k tomu, aby nabyvatel pořídil pro účely užití uvedené v této smlouvě překlad díla.

5.5 V případě, že z díla plyne hospodářský výsledek autorovi nebo nabyvateli, rozdelení zisku bude řešeno dodatkem k této smlouvě.

## **Článek 6**

### **Výpověď smlouvy**

6.1 Každá smluvní strana může smlouvu kdykoliv písemně vypovědět bez udání úvodu.

6.2 Výpověď musí být učiněna doporučeným dopisem doručeným druhé smluvní straně. Výpovědní lhůta je stanovena na dva měsíce a začíná běžet prvním dnem kalendářního měsíce následujícího po měsíci, v němž byla výpověď doručena druhé smluvní straně.

6.3 Autor nahradí Nabyvateli škodu, která mu odstoupením od smlouvy podle odstavce 1 a 2 vznikla. Účinky odstoupení nastanou nahrazením škody nebo poskytnutím dostatečné jistoty.

## Článek 7

### Závěrečná ustanovení

- 7.1 Smlouva je sepsána ve dvou vyhotoveních s platností originálu, která budou vložena do dvou výtisků díla (práce), z toho nabyvatel i autor obdrží po jednom vyhotovení.
- 7.2 Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem a občanským zákoníkem v platném znění, popř. dalšími právními předpisy.
- 7.3 Smlouva byla uzavřena podle svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoli v tísni a za nápadně nevýhodných podmínek.
- 7.4 Smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Písku dne 31. 3. 2025

Autor: \_\_\_\_\_

Nabyvatel: \_\_\_\_\_

## Anotace

Maturitní práce se zabývala tvorbou samořídící šachovnice, jejíž cílem bylo kompletně simulovat protihráče. Ať už se jedná o vymýšlení protitahu či o samotný manuální posun figurky. Šachovnice je ovládána mikrokontrolérem Arduino UNO. Pomocí magnetických spínačů umístěných na PCB detekujeme pozici figurek na hracím poli. Tyto informace následně zpracováváme Arduinem a skrze dva krokové motory a elektromagnet realizujeme tahy figurek. Pro zjištění ideálního příštího tahu využíváme MiniMax algoritmus, jenž je schopný zevaluovat danou situaci a skrze programovou logiku udat příkazy pro pohyb figurek. Celá samořídící šachovnice je vyrobena z dřevěné konstrukce, do které je umístěna deska plošného spoje s elektronikou. Šachové figurky použité v projektu jsou vytvořené pomocí 3D tisku.

**Klíčová slova:** Samořídící šachovnice, automatizace, programové řízení

## Annotation

Graduation thesis focuses on construction of self-controlling chessbord. The goal of the work is to completely simulate opponent. Within the simulation, we should be able to come up with ideal counter move and manual movement of the piece. The chessboard is controlled by a microcontroller Arduino UNO. We use reed switches situated on a PCB to detect position of chess pieces. These information we process with Arduino and through two stepper motors and an electromagnet we realize moves. To detect ideal next move we use MiniMax algorithm, which is able to evaluate given situation and through program logic set signals for stepper motors. The self-controlling chessboard is made out of wooden construction inside which is installed printed circuit board with necessary electronics. Chess pieces used in project have been made by 3D print.

**Key words:** Self-controlling chessboard, automatization, program control

## Poděkování

Chtěl bych poděkovat panu Mgr. Janouškovi za podporu a vedení práce. Za jeho odborné rady a konzultace ohledně možných řešení. Také bych chtěl vyjádřit dík panu Táborovi, Kutilovi, Kolpovi a panu Vondrákovi.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>9</b>
<b>2</b>	<b>Teoretická část</b>	<b>10</b>
2.1	Arduino . . . . .	10
2.2	Deska plošného spoje . . . . .	10
2.3	Elektronické součástky . . . . .	11
2.3.1	Magnetický spínač . . . . .	11
2.3.2	Multiplexor . . . . .	11
2.3.3	Stabilizátor . . . . .	12
2.3.4	Elektromagnet . . . . .	12
2.3.5	Krokový motor . . . . .	12
2.4	Mini-Max algoritmus . . . . .	12
<b>3</b>	<b>Praktická část</b>	<b>14</b>
3.1	Elektrické zapojení . . . . .	14
3.1.1	PCB deska . . . . .	14
3.1.2	Výběr řídící jednotky . . . . .	16
3.2	Detekce obsazenosti políček . . . . .	16
3.2.1	Zapojení a princip . . . . .	16
3.2.2	Programový kód . . . . .	18
3.3	Pohyb figurek na hracím poli . . . . .	24
3.3.1	Elektromagnet . . . . .	24
3.3.2	Krokové motory . . . . .	26
3.3.3	Šachový tah . . . . .	30
3.3.4	Braní figury . . . . .	33
3.4	Algoritmus vytvořeného programu . . . . .	35
3.4.1	Variabilizace hry . . . . .	35
3.4.2	Průběh hry . . . . .	36

3.4.3	Možnost hry s protihráčem . . . . .	37
3.5	Mechanické provedení . . . . .	38
3.5.1	Herní část šachovnice . . . . .	39
<b>4</b>	<b>Závěr</b>	<b>40</b>
	<b>Přílohy</b>	<b>41</b>
<b>A</b>	<b>Samořídíc šachovnice</b>	<b>42</b>
A.1	schema_sachovnice.pdf . . . . .	42
A.2	navrh_herniho_pole.pdf . . . . .	42
A.3	samoridici_sachovnice.mov . . . . .	42

# Kapitola 1

## Úvod

Nápad pro automatickou šachovnici jsem dostal při hledání nového projektu, který bych si doma zvládl sestrojit. Chtěl jsem, aby projekt obsahoval, jak část softwarovou, kterou jsem se do té doby primárně zabýval, tak i část mechanickou, kterou by bylo nutné vyrobit.

Napadlo mě vytvořit nějakou deskovou hru pro více hráčů. V té době jsem měl ve velké oblibě šachy, a proto vznikla idea automatické šachovnice. Začal jsem nákresem na papír a tím jsem získal základní představu o projektu. Šachovnice by se skládala ze tří hlavních částí:

**Konstrukční část** Konstrukční část se skládá ze samotné dřevěné konstrukce šachovnice, do které bude nutné umístit mechanismus pro pohyb figurkami. V rámci toho také její opracování a nadesignování projektu. Tvorba šachových políček a rozhraní pro jednoduché ovládání.

**Elektrotechnická část** Ta obsahuje systém pro detekci figurek na šachovnici. Ten by se dal vytvořit pomocí mnoha způsobů, ale v práci budu popisovat řešení pomocí desky plošného spoje. Dále vytvoření pohybové soustavy pomocí krokových motorů a umístění elektromagnetu, který bude s figurkami pohybovat. Posledním krokem je sestrojení systému, který dovolí hráči nastavit jakou obtížnost bude mít oponent.

**Softwarová část** Z hlediska softwaru je nutné vytvořit kód, který spojí všechny části dohromady. Je nutné transformovat signály tak, aby s nimi bylo možné pracovat. Z hlediska kódu je nutné vytvořit rozhraní pro komunikaci mezi mikrokontrolérem a součástkami. Je zapotřebí umožnit evaluaci dat o pozicích figurek a předat je algoritmu, který nám bude schopen vymyslet další tah.

# Kapitola 2

## Teoretická část

V této kapitole si popíšeme technologie, které byly zapotřebí při tvorbě projektu. Získáme k nim teoretický základ nutný k porozumění šachovnice.

### 2.1 Arduino

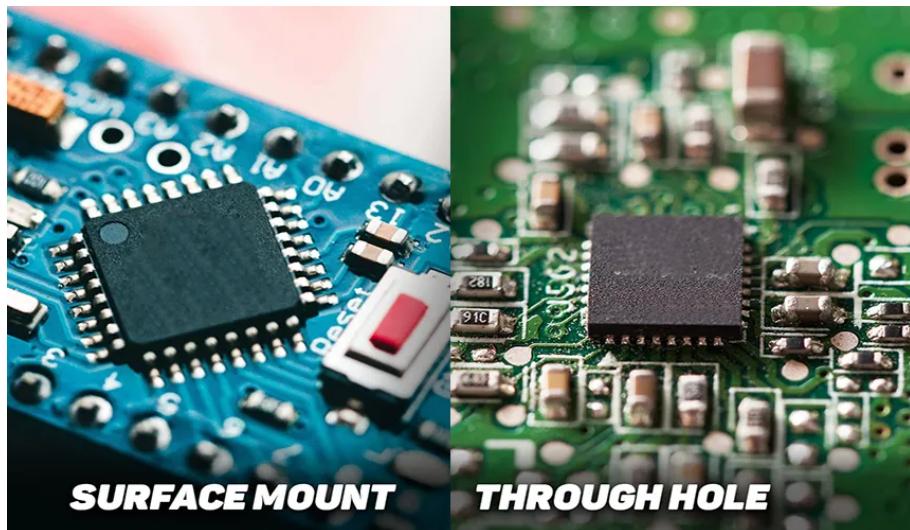
Arduino je platforma pro tvorbu různých elektronických projektů. Zabývá se hardwarem a softwarem a obě tyto části vytváří jako open-source. To je jeden z důvodů velkého rozšíření mikrokontrolerů Arduino.

Mikrokontrolér Arduino je malý počítač založen na jednom čipu. Arduino vyrábí mnoho různých řad jejich projektů, ale mezi nejznámější se řadí Arduino UNO či Mega. Arduino desky na sobě mají široké spektrum možných vstupů, výstupů a senzorů. Velkou výhodou těchto desek je cena. Ta se pohybuje v řádech stovek korun.

Arduino desky se primárně programují pomocí platformy Arduino IDE. To je další open-source program, který má na základě popularity mezi lidmi i velkou podporu knihoven s různými zaměřeními.

### 2.2 Deska plošného spoje

PCB deska, jak je zkratkou nazývána, nebo také DPS je velmi rozšířená elektronická součástka. Desky jsou vyrobené z izolačních materiálů s pájecími poli, kterým se říká podložky a elektrickými spoji nazývanými stopy. Na PCB desky se připájí potřebné součástky. Tyto součástky jsou spolu pak propojeny na základě schématu pro tvorbu PCB. Jsou dvě primární technologie montáže součástek na desku. Je tady technologie průchozích otvorů (THT, Through-Hole Technology), nebo povrchové montáže (SMD, Surface-Mounted Devices) viz. obr. 2.1. Při povrchové montáži se napájí součástky přímo na desku s tím, že



Obrázek 2.1: Rozlišení SMT a THT na PCB desce

jsou na ní připravené jednotlivé stopy. U THT jsou v desce připravené otvory, do kterých součástky umísťujeme.

## 2.3 Elektronické součástky

### 2.3.1 Magnetický spínač

Tento typ spínače je elektronický obvod aktivovaný pomocí magnetického pole. Nejbežnější je konstrukce pomocí dvou feromagnetických kovů umístěných kousek od sebe ve skleněné kapsli. V momentě, kdy je ke spínači přiložen magnet se kovy spojí a začnou vodit. Magnetické spínače jsou například využívány pro detekci uzavření dveří.

### 2.3.2 Multiplexor

Tato elektronická součástka slouží k přepínání vstupů na jeden výstup na základě řídících signálů. Multiplexor je realizován integrovanými obvody. Pomocí této součástky jsme například schopni zvětšit počet vstupů na určitém zařízení o počet vstupů na multiplexoru, s tím že do zařízení zapojíme pouze multiplexor. Tento postup je ideální v momentě, kdy máme zařízení s nedostatkem vstupů pro dané využití.

### **2.3.3 Stabilizátor**

Stabilizátor nám zajišťuje výstupní nápětí bez ohledu na změny výstupního proudu či vstupního napětí. Stabilázátory se dělí na lineární parametrický, lineární zpětnovazební a spínací zpětnovazební. Stabilizátory fungují na principu regulace odporu pomocí tranzistoru na základě referenčního napětí a napětí žádaného. Ke stabilizátoru se přidávají kondenzátory, které eliminují šum a stabilizují výstup. Lineární stabilizátory regulují napětí, tím že fungují jako proměnný odpor mezi vstupem a výstupem.

### **2.3.4 Elektromagnet**

Elektromagnetem se rozumí cívka pomocí, které jsme schopni vytvořit dočasné magnetické pole. Jádro této cívky je z magneticky měkké oceli. V momentě, kdy začneme cívkou vést proud, začne se okolo ní vytvářet magnetické pole. Velikost a polaritu tohoto pole jsme schopni ovládat.

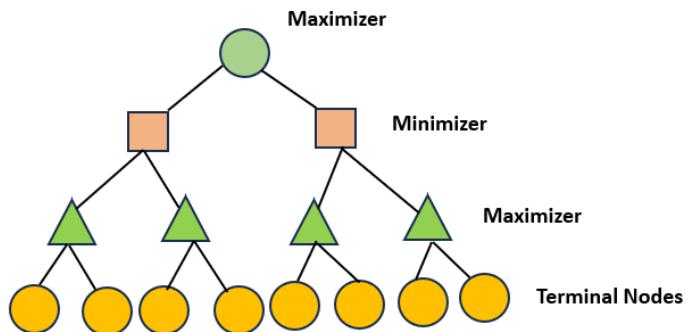
### **2.3.5 Krokový motor**

Krokový motor je zařízení, které nám dovolí na základě elektromagnetických pulsů přesně rotovat. V motoru je ozubené kolo, které můžeme otáčet na základě polových dvojic. Pomocí pulsování jednotlivá pole přitáhnou ozubené kolo a na základě tohoto jevu, můžeme motor otočit o přesný počet kroků. Rychlosť rotování jsme schopni korigovat na základě změn frekvence aktivace polí. To realizujeme skrze ovladače krovových motorů.

## **2.4 Mini-Max algoritmus**

Tento typ algoritmu je fundamentálním konceptem umělé inteligence a teorie her. Mini-Max má za cíl zminimalizovat možnou ztrátu na základě analýzy nejhorších možných scénářů, neboli min, a těch nejlepších, max. Ve hře pro dva hráče si algoritmus vytvoří dvě interní entity. Ta první se jmenuje Maximizer a cílí získat nejvyšší možné skóre. Oproti ní druhá, Minimizer, má za cíl, co nejvíce skóre Maximizer snížit. Toho docílí tak, že evaluuje všechny možné tahy obou hráčů zároveň.

## Minimax Algorithm



Obrázek 2.2: Minimax algoritmus

Z počátku si Mini-Max vytvoří tzv. herní strom. Účelem tohoto stromu je reprezentovat všechny možné tahy, které mohou v daný moment ve hře nastat. Dalším krokem je, že u tzv. závěrečných stavů (terminal nodes) určí skóre. Skoře se určuje na základě kritérií daných v kódu a postupně ve stromu postupuje výše a určuje skóre dál. Poté, co Mini-Max určí skóre pro všechny možné stavy, projde jednotlivé scénáře. V kole Maximizera volí variantu s nejvyšším možným skórem. Naopak v kole minimizera tu s nejnižším. Podle nejvyššího možného celkového skóre algoritmus určí příští zvolený blok.

# Kapitola 3

## Praktická část

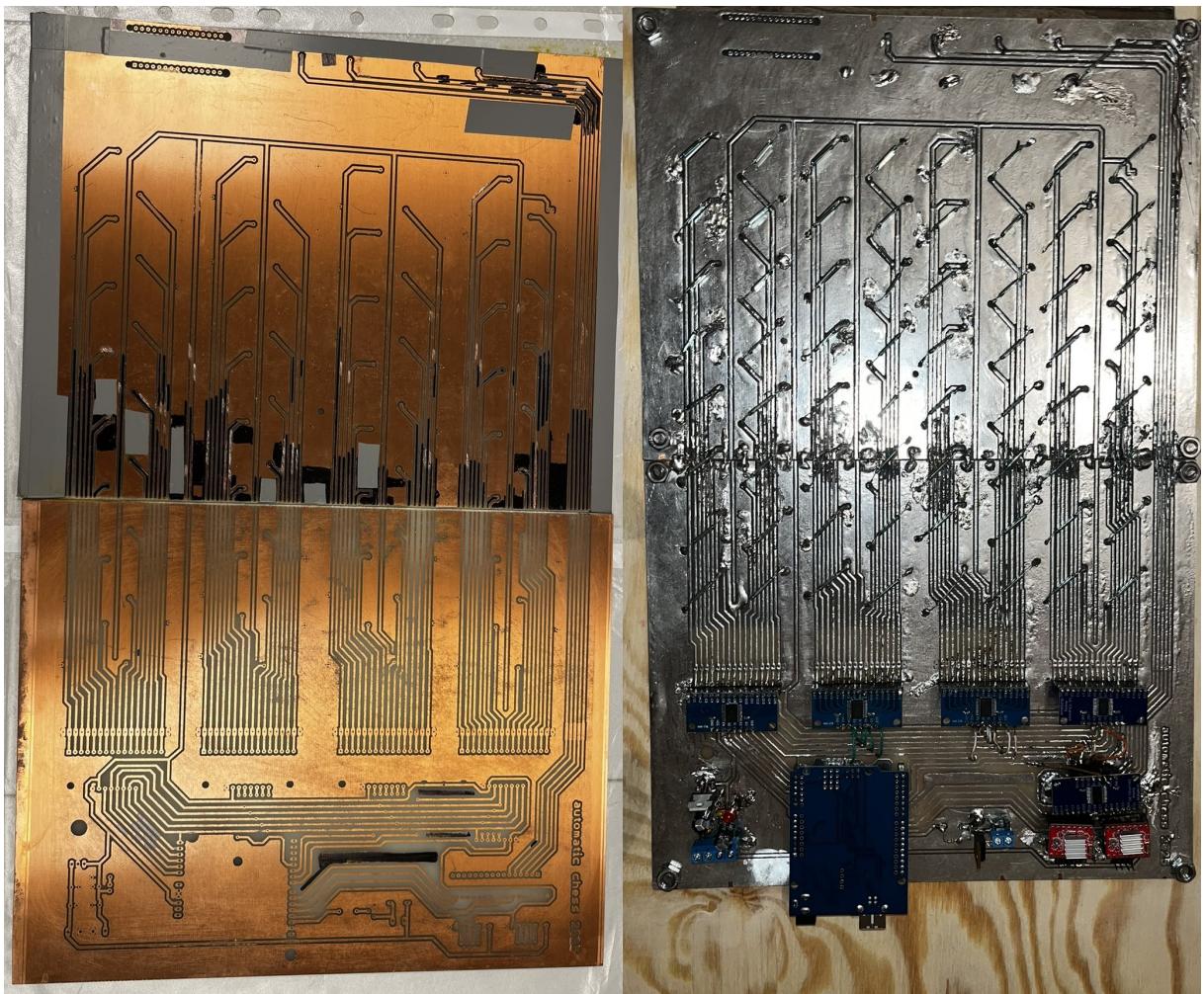
### 3.1 Elektrické zapojení

Pro projekt bylo zapotřebí vymyslet efektivní a spolehlivý způsob propojení. Zároveň však způsob nesměl bránit detekci figurek pomocí spínačů. Nabízela se spoustu možností, ale z důvodu velkého množství součástek jsem se rozhodl pro dle mě nejčistší řešení, a to desku plošného spoje.

#### 3.1.1 PCB deska

Deska plošného spoje bylo řešením, které se nabízelo kvůli spolehlivosti a zároveň tím, že deska je velmi tenká a její materiál nijak nebrání průchodu magnetického pole. Na základě mého schématu v programu EasyADA jsem si desku nechal sestrojit. Schéma šachovnice najdete v přílohách pod názvem schema\_sachovnice.pdf. Z důvodu výrobních možností a rozměrů desky byla deska sestrojena na dvě poloviny. Tyto poloviny jsem spájel dohromady. Deska byla vyrobena technologií SMT, takže jsem na ní povrchově připájal piny pro všechny potřebné součástky. A to:

- 69 jazýčkových magnetických kontaktů
- 5 analogových multiplexorů, transistor a různé drobné zařízení
- Mikrokontroler Arduino UNO
- Dva A4988 ovladače pro krokové motory
- Stabilizátor a piny pro vstupní napětí
- Napájení pro motory a elektromagnety



Obrázek 3.1: Porovnání PCB desky z výroby a osázené

Desku jsem si následně nechal pocínovat. Dvě části jsem pomocí propojek spojil dohromady. Během práce jsem posléze při detekci figurek musel řešit velký problém, a to se spolehlivostí desky. Magnetické spínače nefungovaly spolehlivě. Důvodem byly narušené cesty a také křekhost spínačů. Ty během pájení velmi snadno praskly.

Tyto problémy jsem řešil za pomocí multimetru a posléze detekcí pomocí počítače. Na desce jsem postupně zkontoval všechny cesty a v momentě, kdy špatně vodily jsem na ně napájel vrstvu cínu. Pokud problém spočíval v propojení dvou vedlejších cest, vzal jsem nůž a cín manuálně odstranil. Největším úskalím bylo manuální propojení dvou desek dohromady. Tato část desky vyžadovala časově velmi úpornou manuální péči.

Desky se mi po opravách podařilo dostat do spolehlivého stavu. Jednou věcí, která mi však projekt stížila bylo, že nespolehlivost cest na desce a spínačů jsem objevil až v

momentě, kdy jsem programoval detekci figurek. V tuto chvíli jsem však již, alespoň mohl využít napojení na multiplexory a pomocí kódu je kontrolovat skrze konzoly na počítači. To usnadilo proces oprav.

Deska je napájena externím 18V zdrojem. Na desce se pomocí stabilizátoru pro většinu součástek nastaví napětí 5V, avšak pro motory a elektromagnet zůstává napětí 18V.

### 3.1.2 Výběr řídící jednotky

Mikrokontrolér, který jsem si vybral pro samořídící šachovnici je Arduino UNO. Toto Arduino splňovalo veškeré nutné požadavky pro ovládání a zároveň to byl mikrokontroler, který jsem měl z počátku projektu doma a mohl jsem na něm zprvu testovat. Arduino je napájeno 5V z PCB desky a ovládá celou šachovnici. Arduino je naprogramováno variantou jazyku C++, ale o tom budu psát podrobněji později v práci. Do mikrokontroleru jsou skrze multiplexory zapojeny všechny magnetické spínače. Arduino si pomocí programové logiky přepíná jaký z multiplexorů bude v dané chvíli aktivní, jelikož data multiplexorů proudí po stejných cestách. Dále jsou do Arduina zapojené drivery motorů a transistor, který slouží k ovládání elektromagnetu.

## 3.2 Detekce obsazenosti políček

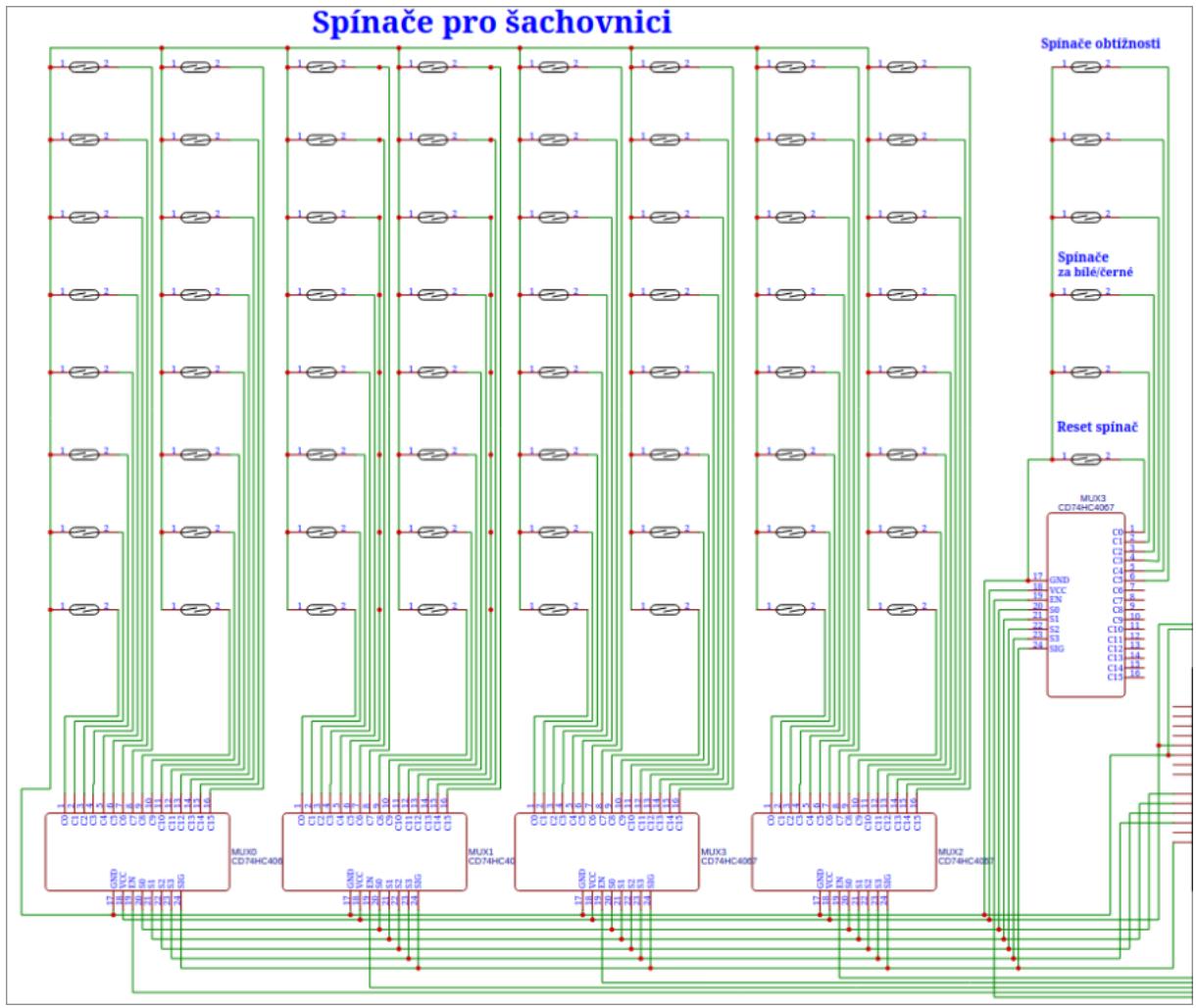
V této kapitole popíši, na jakém principu se v šachovnici detekují pozice figurek. Vyšvětlím jednotlivé zapojení multiplexorů a kód, který je ovládá. Spolehlivé detekování pozice figurek je stěžejním bodem této práce, jelikož z něho vychází další operace projektu.

### 3.2.1 Zapojení a princip

Aby samořídící šachovnice fungovala je zapotřebí vědět pozici figurek na herním poli. Nabízí se dvě řešení. Detekce pomocí kamer a zpracování obrazu a pomocí magnetických spínačů. Řešení pomocí kamer jsem zavrhl, jelikož by to znamenalo nějaké rameno na šachovnici s kamerou. To mi nepřipadal ideální. Magnetické spínače nám zachovají čistý vzhled šachovnice. Jelikož však na šachovnici máme 64 políček, potřebujeme 64 spínačů.

Tolika piny naše Arduino Uno nedisponuje, a tak je zapotřebí využít nějaký systém pomocí, kterého počet vstupů snížíme.

Po průzkumu a práci na řešení jsem se inspiroval z jednoho projektu dostupného na internetu [9]. V tomto projektu bylo využito multiplexorové řešení. Zapojíme piny spínačů do multiplexoru, a ten pak čteme pomocí řídící jednotky. Pomocí tohoto postupu jsme schopni zredukovat 64 pinů na 4 multiplexory. V této práci jsem jich využil pět, jelikož poslední multiplexor se stará o volbu obtížnosti a barvy.



Obrázek 3.2: Zapojení magnetických spínačů na desce

Multiplexory, které jsem využil mají 16 datových vstupů, 4 vstupy adresové, nápajení, zem, zapínací pin EN a výstup SIG. Výstupy multiplexorů, jak je vidět na schématu viz. obr. 3.2, jsou zapojeny do stejných cest na PCB desce. Stejně tak výstup SIG je pro všechny multiplexory stejný. Toto není problém, jelikož v kódu využívám pin Enable (en). Jestliže je tento pin v logické jedničce, je multiplexor vypnutý. V momentě, kdy je v logické nule, je zapnutý. Na základě tohoto principu jsme schopni přepínat přepínače samotné a tím ještě více zminimalizovat počet nutných vstupů do Arduina UNO.

### 3.2.2 Programový kód

Jak už jsme zmínili, šachovnice je ovládána mikrokontrolerem Arduino UNO, jenž se programuje variantou C++. Kód pro ovládání multiplexorů nám musí zajistit jejich

přepínání pomocí Enable pin a stabilní detekci pozic. První věc, kterou musíme v kódu nastavit jsou Enable piny. Po zapnutí nastavím Enable pin jako OUTPUT a pomocí for loopu si procyclujeme všechny Enable piny multiplexorů a dáme na ně logickou jedničku. Tato část kódu je v setup() funkci kódu. Ta se spustí pouze po spuštění mikrokontroleru.

```

1 // Disabling Multiplexers (When high - does nothing)
2 for (int i = 0; i < 5; i++) {
3     pinMode(muxEnable[ i ] , OUTPUT);
4     digitalWrite(muxEnable[ i ] , HIGH);
5 }
6
7 for (int i = 0; i < 4; i++) {
8     pinMode(muxAddr[ i ] , OUTPUT);
9     digitalWrite(muxAddr[ i ] , LOW);
10}
```

## Čtení hodnot pomocí getReedValues()

Pro čtení hodnot využíváme funkci getReedValues. V ní nejprve zaktivujeme multiplexor tím, že mu nastavíme pin na 0. Poté začneme na jeho adresové piny vysílat binarní kombinace od 0 do 15. Toho docílíme opět pomocí for cyklu a využití modulo funkce (v kódu jako %). Data následně ukládáme do našeho interního variablu muxValues a poté, co deaktivujeme multiplexor a jeho adresové vstupy, překopírujeme pomocí funkce memcpy hodnoty do proměnné recordedReedValue. Ta nám slouží pouze pro ukládání nejnovější možné detekce pozic. Pozice každé druhé řady zapisujeme oproti řadě první obráceně. To je z důvodu zapojení na PCB desce a realizováno podmínkou ve funkci.

```

1 void getReedValues( int targetMuxAddress , int from , int to ) {
2     // Splitting to two rows because row of chessboard has 8 pieces
3     int muxValues[ 8 ];
4
5     // Activating MUX
6     digitalWrite(targetMuxAddress , LOW);
7     for (int j = from; j < to; j++) {
8         // Checking MUX combinations
9         digitalWrite(muxAddr[ 0 ] , j % 2 );
10        digitalWrite(muxAddr[ 1 ] , j / 2 % 2 );
```

```

11     digitalWrite(muxAddr[2] , j / 4 % 2);
12     digitalWrite(muxAddr[3] , j / 8 % 2);
13
14     int reedValue = digitalRead(muxOutput);
15
16     if (j > 7) {
17         muxValues[j - 8] = reedValue;
18     } else {
19         muxValues[to - j - 1] = reedValue;
20     }
21 }
22 // Resetting MUX
23 digitalWrite(muxAddr[0] , LOW);
24 digitalWrite(muxAddr[1] , LOW);
25 digitalWrite(muxAddr[2] , LOW);
26 digitalWrite(muxAddr[3] , LOW);
27
28 digitalWrite(targetMuxAddress , HIGH);
29 memcpy(recordedReedValue , muxValues , sizeof(recordedReedValue));
30 }
```

Jelikož je SIG pin multiplexorů nastavený jako INPUT\_PULLUP, což je nutné pro detekci zda spínač spíná zem či je rozpojený, dostávám všechny rozpojené spínače jako jedničky. Ty seplé jako nuly. Ve výsledku může výstup v konzoli po zahájení hry vypadat takto.

```

    8 - 0 0 0 0 0 0 0 0
    7 - 0 0 0 0 0 0 0 0
    6 - 1 1 1 1 1 1 1 1
    5 - 1 1 1 1 1 1 1 1
    4 - 1 1 1 1 1 1 1 1
    3 - 1 1 1 1 1 1 1 1
    2 - 0 0 0 0 0 0 0 0
    1 - 0 0 0 0 0 0 0 0
    ----A B C D E F G H

```

## Ukládání všech pozic

Předchozí funkce nám zajišťuje přečtení hodnot multiplexoru na základě daných parametrů. V kódu máme další pomocnou funkci setCurrentBoard(). Cílem této funkce je projít všechny řady šachovnice od první až po osmou a uložit je do mikrokontroleru. V mikrokontroleru je opět pomocí funkce memcpy překopíruje do dvouřadého listu boardValues, který obsahuje všechny pozice figurek.

```

1 void setCurrentBoard() {
2     getReedValues( muxEnable[0] , 0 , 8 );
3     memcpy( boardValues[0] , recordedReedValue , sizeof(boardValues[0]) );
4
5     getReedValues( muxEnable[0] , 8 , 16 );
6     memcpy( boardValues[1] , recordedReedValue , sizeof(boardValues[1]) );
7     // ..... another five reads
8     getReedValues( muxEnable[3] , 8 , 16 );
9     memcpy( boardValues[7] , recordedReedValue , sizeof(boardValues[7]) );

```

## Detekce přemístění figurek

Předchozí funkce v této kapitole nám vysvětlují, jak šachovnice data čte. Nadcházející funkce je však zodpovědná za samotnou detekci změny pozic, při kterých uživatel vezme figurku do ruky a přendá ji na jiné políčko. Tato funkce je napsána bloku loop(). To je opět platformou Arduino daná funkce, která se po spuštění již zmíněné funkce setup()

neustále opakuje.

Abychom zjistili, zda se událo přemístění figurky musíme pravidelně čist pozice hodnot a porovnávat je z přechozími. To děláme pomocí while cyklu, který se opakuje do doby, než jsou splněné podmínky uvedené v závorkách. Podmínky, které jsou v kódu nastavené vyžadují, aby list proměnných move obsahoval čtyři hodnoty. Neboli hodnotu čísla a písmena odkud se figurka přemístila a hodnoty kam. Jak zjistíme hodnoty, které ukládáme do move, vysvětlím dále v textu. Nejprve tedy uložíme hodnoty pozic do proměnné boardValuesMemory a následně přejdeme do cyklu. V tomto cyklu se neustále volá funkce detectBoardMovement() až do splnění zmíněných podmínek.

```
1 // Player's turn => waiting for movement
2 setCurrentBoard();
3 memcpy(boardValuesMemory, boardValues, sizeof(boardValuesMemory));
4 while (!move[0] || !move[1] || !move[2] || !move[3]) {
5     detectBoardMovement();
6 }
```

## Funkce detectBoardMovement()

Pomocí této funkce detekujeme, zda bylo uskutečněno přemístění figurek. Tato funkce je volána cyklicky z while loopu dokud nenajdeme čtyři hodnoty pro proměnnou move. Např. e2e4 by nám značilo pohyb z počátku hry pěšce o dvě místa z e2 na políčko e4. Funkce porovnává současné hodnoty pozic v proměnné boardValues, které nastaví pomocí funkce setCurrentBoard() s hodnoty boardValuesMemory uloženými v hlavním cyklu před zahájením while cyklu.

```
1 void detectBoardMovement() {
2     bool fromChange = false;
3     setCurrentBoard();
4
5     for (int i = 0; i < 8; i++) {
6         for (int j = 0; j < 8; j++) {
7             if (boardValuesMemory[i][j] != boardValues[i][j] && boardValues[i][j]
8                 ]) {
9                 // Position of piece has changed
10                delay(1000);
```

```

10     setCurrentBoard();
11     if (boardValues[i][j]) {
12         // Piece was here before
13         fromChange = true;
14         move[0] = letterTranslate[j];
15         move[1] = numberTranslate[i];
16     }
17 }
18 }
19 }
20
21 // For loop for 1-0...
22
23 if (!fromChange || (move[2] && !move[0])) {
24     setCurrentBoard();
25     memcpy(boardValuesMemory, boardValues, sizeof(boardValuesMemory));
26     resetMove();
27 }
28 delay(1000);
29 }
```

Funkce pomocí for loopu prochází hodnoty pozic a tam postupně porovnává každé políčko s každým dokud nenajde nějakou změnu. V případě, že všechny hodnoty jsou stejné jako předtím, funkce se s vteřinovým zpožděním spustí znovu. V případě změny však pomocí kódu detekujeme zda šlo o změnu z 0 na 1 či naopak.

**Změny z 0 na 1** Jelikož logická jednička označuje prázdné políčko, tato změna značí odkud se figurka přemístila. Zapíše se tedy do move na první a druhé pozici.

**Změny z 1 na 0** Značí, že políčko bylo dříve prázdné a nyní se na něj umístila nová figurka. Zapíše se tedy do listu move na třetí a čtvrtou pozici

V momentě, kdy naplníme všechny čtyři místa v listu move, zruší se while cyklus v hlavní funkci programu a kód může postupovat dále. Ve funkci na konci kontrolujeme situaci, kdy se projde celý cyklus a najde pouze pozici, kam se figurka přemístila ale ne odkud. Také hlídáme situaci, kdy se žadný tah nenajde. V takových případech, tah

resetujeme a nastavíme novou hodnotu pozic spínačů do boardValuesMemory. Poté je ve funkci vteřinové zpoždění, které zajišťuje větší spolehlivost detekce při přesunu figurek.

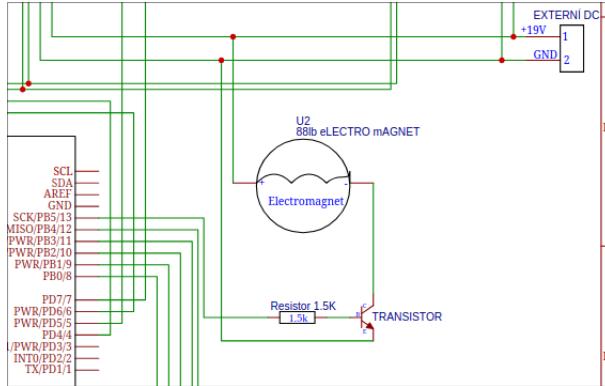
### 3.3 Pohyb figurek na hracím poli

Realizaci šachových tahů nám zajišťují dva krokové motory a elektromagnet. Pro účel práce bylo nutné umožnit XY pohyb elektromagnetu uvnitř šachovnice. Soustavu bylo nutné umístit dovnitř konstrukce a zajistit, aby byl elektromagnet velmi blízko PCB desce a tím i figurkám.

#### 3.3.1 Elektromagnet

Samotné přemístění figurky je realizováno pomocí elektromagnetu. Ten po přivedení proudu vytvoří elektromagnetické pole, které začne působit na magnetickou šachovou figurku nad ním, a tím s ní můžeme hýbat. Během tvorby řešení jsem musel zajistit, aby elektromagnetické pole nebylo příliš velké a neovlivňovalo ostatní figurky. Zprvu ovšem nastal jiný problém. Elektromagnet, i když zapnutý, nebyl schopen figurkou pohnout. Tato situace nastala na základě faktu, že pro velkou účinost elektromagnetu je nutné zminimalizovat prostor mezi elektromagnetem a přitahovaným tělesem. V situaci, kdy jsem používal elektromagnet se silou 150N stále to nebylo dostatečné. Problém jsem vyřešil vyvýšením elektromagnetu v konstrukci a zasazením jej blíže k plošnému spoji. Po těchto úpravách mi postačil, elektromagnet se silou 50N. Avšak ani tento elektromagnet by s figurkami nepohnul, jestliže by do nich nebyly umístěny magnetické prvky. Pokud by do figurky byla umístěna prostá ocel, nedokázal by jí elektromagnet přemístit.

Elektromagnet, který v práci využívám je v parametrech u prodejce uváděn na 12V. Já do něj však vysílám 18V z důvodu lepší spolehlivosti. Abychom však elektromagnet mohli ovládat, musel jsem vymyslet způsob, jakým do elektromagnetu těchto 18V budu vpouštět pouze po zaslání signálů. Na to využívám transistor. Transistor použitý v práci po přivedení signálu na bázi, přepne vstup z kolektoru na emitor. Na kolektoru mám přivedenou zem a po zaslání signálů na bázi, vyšlu zem skrze emitor do elektromagnetu. Ten je z jeho druhého vstupu připojen k napětí a po přivedení země, začne působit. Po ukončení signálu na bázi se elektromagnet vypne.



Obrázek 3.3: Ovládání elektromagnetu pomocí transistoru

Kód, který transistor ovládá je velmi jednoduchý. Ve funkci makeMove, o které budu v dalších kapitolách hovořit, je booleovská proměnná magnetActivated. V případě, že magnet potřebuji, nastavím jí jako true a pomocí podmínky v této funkci vyšlu na pin báze transistoru (magnetPin) logickou jedničku. Na konci funkce po provedení pohyby, magnet vždy vypínám.

```

1 // Enabling the electromagnet if neccesary
2 if (magnetActivated) {
3     digitalWrite(magnetPin, HIGH);
4 }
```

Dále je důležitá kalibrace elektromagnetu. Pro správný průběh programu je zapotřebí, aby elektromagnet byl vždy na začátku hry na stejném místě. Toto místo je mírně vedle pozici a1. Ke kalibraci využíváme funkci resetMagnet a setupMagnet. Přemístění provedeme, tak že po zapnutí samořídící šachovnice, umístíme na pozice ovládacích spínačů tři figury. Obsadíme pozice pro volbu hry za bílé, černé a první obtížnost.

```

1 while (!controlValues[4] && !controlValues[5] && !controlValues[6]) {
2     getReedValues(muxEnable[4], 0, 8);
3     memcpy(controlValues, recordedReedValue, sizeof(controlValues));
4     digitalWrite(dirPinX, LOW);
5     digitalWrite(stepPinX, HIGH);
6     delayMicroseconds(motorDelay + 600);
7     digitalWrite(stepPinX, LOW);
8     delayMicroseconds(motorDelay + 600);
9     magnetX = 1;
10 }
```

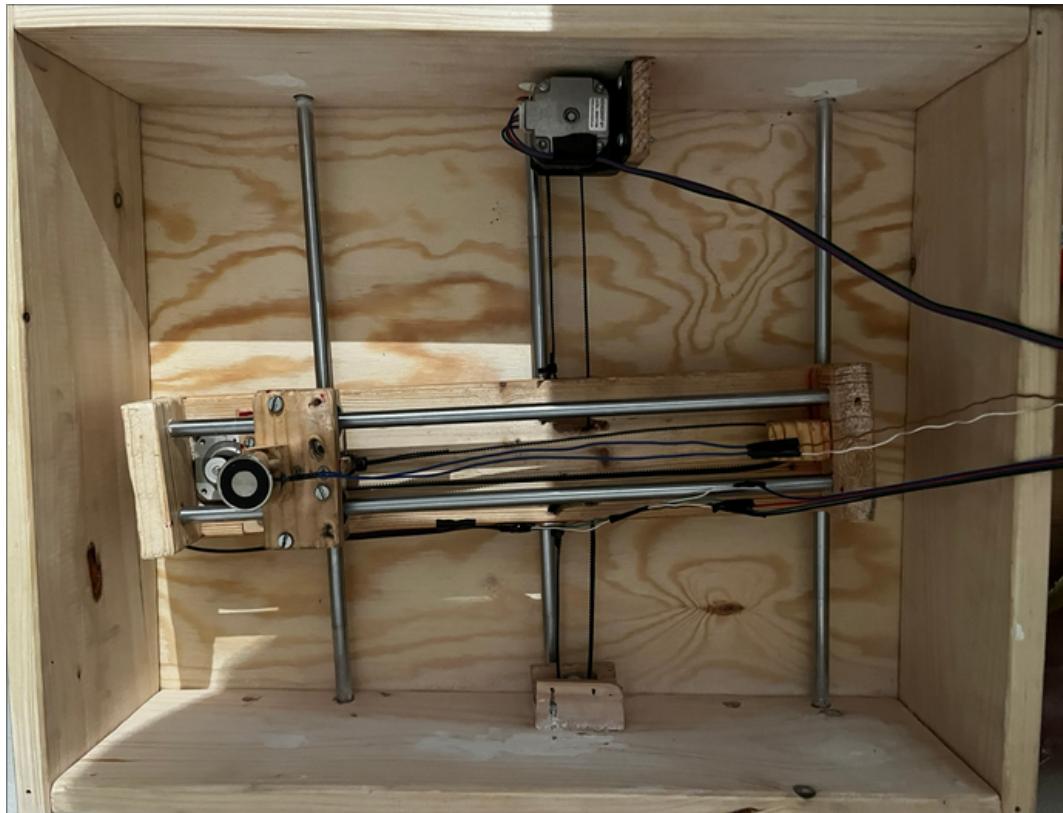
Kód začne přesouvat elektromagnet doleva až do doby, kdy odstraníme figury ze spínače pro obtížnost. Poté se začne elektromagnet přesouvat směrem dolů. Cílovou pozici poznáme pomocí zvuku, které motory při pohybu dodávají. V momentě, kdy magnet dosáhne cílové ať už X či Y souřadnice, začne zvuk motoru zintenzivňovat. Elektromagnet eventuálně pomocí funkce setupMagnet, posuneme mírně směrem vpravo a tím zajistíme start na pozici a1.

### 3.3.2 Krokové motory

Pro zajištění pohybu využívám dva krokové motory NEMA17 s driverem A4988. Driver krokových motorů slouží jako mezikrok mezi programem a kódem a zajišťuje pohodlné ovládání motorů. Do těchto driverů přivádíme po desce, jak 18V, tak 5V. Dále na nich nastavujeme směr pohybu a vysíláme signály, pomocí kterých pak motor hýbe s konstrukcí. Pro zajištění správného pohybu bylo nutné zajistit dostačné napětí. To je dle specifikací 8 až 35V. Po otestovaní jsem zjistil, že 18V bylo pro projekt ideální. Abych zamezil přehřívání ovladačů, nalepil jsem na ně z vrchu chlazení, které je s nimi prodáváno.

### Konstrukce

Konstrukce je realizována pomocí soustavy dvou motorů. Během řešení této problematiky jsem se inspiroval tímto projektem[10]. První motor je umístěn zevnitř boční stěny šachovnice. Na motor jsem umístil vhodnou řemenici a řemen natáhl na protější stěnu. Zde jsem si ze dřeva sestrojil ústrojí pro další řemenici. Pod řemenem vedou tři vodicí tyče, do kterých jsem umístil druhý motor s konstrukcí. Řemen prochází touto konstrukcí a při otáčení řemenice, tak pomocí řemenu pohybujeme na Y ose i s konstrukcí uvnitř šachovnice. Vnitřní konstrukce je umístěna na třech ložiscích, kterými prochází vodicí tyče. V této vnitřní dřevěné konstrukci je vyříznut otvor pro druhý krokový motor a zde je postupováno podle podobného postupu. Vedou zde dvě vodicí tyče na X ose. Na těchto vodicích tyčích jsou dvě ložiska, na která jsem si sestrojil dřevěnou plošinku připravenou pro umístění magnetu. Magnet je na plošině vyvýšen pro zajištění spolehlivého přenosu figur a kabely jsou vyvedeny na PCB desku. Na druhé straně je opět ústrojí pro řemenice a řemen je proveden skrze plošinu s elektromagnetem.



Obrázek 3.4: Konstrukce krokových motorů

### Programový kód

Funkci, která ovládá motory jsem nazval moveMagnet. Tato funkce bere tři vstupní parametry. Směr (direction) a vzdálenost (distance). Pro přehlednost nerozlišuje směry pomocí if podmínek, ale pomocí switche. V něm nastavím, jaký blok kódu se provede, jestliže směr bude odpovídat hodnotě za "case". Hodnota směru je dána jako pozice v numerické klávesnici. Tedy 4 jako pohyb vlevo, 2 pohyb směrem dolů a např. 9 jako diagonální pohyb doprava nahoru.

```
1 void moveMagnet( int direction , int distance ) {  
2     switch ( direction ) {  
3         // ...  
4         case 4:  
5             // Left  
6             digitalWrite ( dirPinX , LOW ) ;  
7             for ( int i = 0; i < boxLength * distance ; i++ ) {  
8                 digitalWrite ( stepPinX , HIGH ) ;
```

```

9      delayMicroseconds(motorDelay);
10     digitalWrite(stepPinX, LOW);
11     delayMicroseconds(motorDelay);
12 }
13 magnetX = magnetX - distance;
14 break;
15 case 9:
16 // Diagonal right up
17 digitalWrite(dirPinY, HIGH);
18 digitalWrite(dirPinX, HIGH);
19 for (int i = 0; i < boxLength * distance; i++) {
20     digitalWrite(stepPinY, HIGH);
21     digitalWrite(stepPinX, HIGH);
22     delayMicroseconds(motorDelay);
23     digitalWrite(stepPinY, LOW);
24     digitalWrite(stepPinX, LOW);
25     delayMicroseconds(motorDelay);
26 }
27 magnetX = magnetX + distance;
28 magnetY = magnetY + distance;
29 break;
30 }
31 // ...

```

Samotný kód, který vysílá signály ovladači je v bloku pod case. Nejprve určíme směr pohybu. Buď logickou nulou či jedničkou. Následně začneme pomocí for cyklu vysílat pulzy do motoru. Vždy nejdříve aktivujeme nutný stepPin. Pak zahájíme prodlevu. Tato prodleva nám určuje rychlosť otáčení. Čím menší je, tím rychleji se motory točí. Poté stepPin vypneme a znova zahájíme prodlevu. Tento cyklus se opakuje podle žádané vzdálenosti, kterou násobíme délkou jednoho políčka.

## Realizace tahu dle souřadnic

Abychom byli schopni zahrát šachový tah musíme ho transformovat do souřadnic. Zde si vysvětlíme funkci, která tah realizuje.

Tato funkce se jmenuje makeMove a bere pět vstupních parametrů, XY souřadnice

odkud a kam a dále, zda má být elektromagnet zapnutý. Věc, kterou jsem určil tak, aby se provedla jako první, je diagonální pohyb. Pomocí testování různých vztahů na šachovnici, jsem přišel na vzorec, který nám určuje diagonální pohyb.

$$|y - \text{targetY}| == |\text{targetX} - x|$$

Jestliže v situaci vztah platí, provedeme diagonální pohyb na místo a ukončíme funkci pomocí return. Ve funkci dále rozlišujeme pohyb diagonální doprava/doleva nahorů či dolů. Jaký potřebujeme zjistíme porovnáním x a targetX.

```

1   if ( abs(y - targetY) == abs(targetX - x) ) {
2       // Diagonal movement when moving piece
3       if(y > targetY) {
4           // Diagonal down
5           if (x < targetX) {
6               // Right
7               moveMagnet(3, targetX - x);
8           } else {
9               // Left
10              moveMagnet(1, targetX - x);
11          }
12      } else {
13          if (x < targetX) {
14              // Right
15              moveMagnet(9, targetX - x);
16          } else {
17              // Left
18              moveMagnet(7, targetX - x);
19          }
20      }
21      return;
22  }
```

Pokud pohyb dle souřadnic není diagonální, pohneme s figurkou po X či Y ose. Proces je podobný jako předtím, akorát zde neaplikujeme vzorec ale pouze porovnáváme. Využíváme zde dřívě zmíněnou funkci moveMagnet, kterou dáváme jako požadovanou urazenou vzdálenost rozdílem mezi x a targetX. Nejprve provedeme pohyb vertikální a až poté horizontální.

```

1 // Move magnet to starting position on Y
2 if (y < targetY) {
3     // Move up
4     moveMagnet(8, y - targetY);
5 } else if (y > targetY) {
6     // Move down
7     moveMagnet(2, targetY - y);
8 }
```

### 3.3.3 Šachový tah

Algoritmus, který nám určuje tahy oponenta je udáván v tomto formátu šachovém formátu, např. e2e4. Tento fakt nám komplikuje proces tahu oponenta. Musíme totiž transformovat tento příkaz do příkazu, kterému rozumí krokové motory.

Tuto transformaci provádíme ve funkci makeChessMove. Ta má jako vstup daný set znaků, který si přeloží do instrukcí. K překladu využíváme C funkci strchr, které nám udá pozici v listu. K ní pak přičteme jedničku a máme požadovanou X souřadnici tahu. Pro "e" by to tedy bylo 5. Y souřadnici pouze změníme z textu na číslo tím, že od něj odečteme textovou nulu. Následně tyto souřadnice předáme do funkce makeMove, kterou jsme vysvětlili v předešlé kapitole.

```

1 void makeChessMove(char givenMove[5]) {
2     char aiLetterTranslate[8] = "abcdefghijklmnopqrstuvwxyz";
3
4     int fromX = strchr(aiLetterTranslate, givenMove[0])-aiLetterTranslate+1;
5     int fromY = givenMove[1] - '0';
6
7     int toX = strchr(aiLetterTranslate, givenMove[2])-aiLetterTranslate+1;
8     int toY = givenMove[3] - '0';
9     // ...
```

Po těchto operacích teprve zahajíme požadovaný přesun figurky. Přemístíme magnet z jeho současných souřadnic na souřadnice figurky, s kterou má pohnout a poté, co tam dorazí aktivujeme elektromagnet a přemístíme magnet i s figurou na souřadnice cílové. V momentě, kdy není nutný žádný speciální typ pohybu, realizujeme ho takto.

```
1 // Other pieces
```

```

2 makeMove(magnetX, magnetY, fromX, fromY, false);
3 makeMove(fromX, fromY, toX, toY, true);

```

Přenos na cílové souřadnice musíme dále rozlišit na základě různých situací, které mohou nastat. Šachovnice si musí umět poradit s pohybem koně i v situaci, kdy kůň nemá prostor na pohyb, s rošádou a běžným šachovým pohybem.

## Pohyb koně

Ve funkci makeChessMove musíme nějakých způsobem pohyb koně detekovat. V rámci práce jsem postupným testováním přišel na tento vzorec, kterým pohyb detekujeme.

$$[fromY - toY] \neq [toX - fromX] \quad \&\& \quad ([fromX - toX] == 1 \mid\mid [fromX - toX] == 2)$$

Pohyb koně řeším ve funkci handleHorseMovement. Rozlišuji ho na horizontální a vertikální. O jaký se jedná zjistím opět pomocí absolutní hodnoty. Jestliže je rozdíl počátečního a konečného X rovné 1, jedná se o pohyb vertikální. V případě, že je to 2, tak pohyb horizontální. Horizontální pohyb se vykonává principiálně stejným způsobem jako vertikální, a tak dále popíší pouze ten vertikální.

```

1 if (!isPlaceOccupied(fromX, awayY)) {
2     // Empty in front him
3     makeMove(fromX, fromY, fromX, awayY, true);
4     makeMove(fromX, awayY, toX, toY, true);
5     return;
6 }

```

Ve funkci opět kontrolujeme pozice před koněm. V bloku nad tímto textem je kód pro řešení situace, kdy je pole přímo před koněm volné. Pak pouze standartně vyjedeme. V momentě, kdy je jsou volná místa na stranách, přemístěme koně diagonálně do nich a až poté na cílové místo.

Další situací, kterou jsem v práci musel vyřešit je když kůň nemá žádný prostor pro pohyb. Tato situace nastane například pokud chce algoritmus zahrát pohyb koně jako první tah. Jako řešení jsem vytvořil kód, který nejprve uhne s figurkou před koněm, poté přemístí koně a figurku vrátí.

```

1 int awayY = fromY - 1;
2 int awayPawnY = fromY - 3;

```

```

3 if (fromY < toY) {
4     awayY = fromY + 1;
5     awayPawnY = fromY + 3;
6 }
7
8 if (awayPawnY > 0 && awayPawnY < 9) {
9     // Move pawn away
10    makeMove(fromX, fromY, fromX, awayY, false);
11    makeMove(fromX, awayY, fromX, awayPawnY, true);
12
13    // Move horse to place
14    makeMove(fromX, awayPawnY, fromX, fromY, false);
15    makeMove(fromX, fromY, toX, toY, true);
16
17    // Move pawn back to place
18    makeMove(toX, toY, fromX, awayPawnY, false);
19    makeMove(fromX, awayPawnY, fromX, awayY, true);
20 }

```

Nejprve nastavíme hodnoty `awayY` a `awayPawnY`. `Away Y` nám určuje s jakou figurkou chceme uhýbat a `awayPawnY` nám udává, kam s níuhneme. Poté zkontrolujeme, zda pozice, kam chceme figurku blokující pohyb koně přemístit, je v rámci šachového pole. Poté s níuhneme na pozici `awayPawnY`. Koně přesuneme na pozici figurky a pak na cílové místo. Figurku z `awayPawnY` vrátíme na původní pozici.

## Rošáda

V šachu mohou nastat pouze 4 různé šachové rošády. V kódu jsem v podmínce tyto situace zadefinoval a kontroluji je při `makeChessMove`.

```

1 if (fromX == 5 && (toX == 7 || toX == 3) &&
2 (toY == 1 && fromY == 1 || fromY == 8 && toY == 8)) {
3     // Castling
4     handleCastling(fromX, fromY, toX, toY);
5 }

```

Ve funkci `handleCastling` rošádu řeším pomocí uhybání figurek, které rošádě brání. S funkcionalitou rošády mám v práci mírný problém a to z důvodu malého prostoru pro

manipulaci s figury na poslední a první řadě šachovnice. I přesto jsem realizoval kód, který se o její provedení pokouší. Rozděluji si rošádu na dva typy a popíši, jak řeším rošádu za černou barvu.

**Rošáda doleva** Nejprve se podívám zda před cílovou pozicí věže po rošádě je nějaká figurka. Jestliže ano, přemístím jí dopředu. Poté na místo této figurky posunu krále. Dalším krokem je přemístění věže na její cílovou pozici rošády. Poté diagonálním pohybem umístím krále vedle ní.

**Rošáda doprava** Zde je situace odlišná, jelikož jsme schopni využívat 9. horizontální souřadnice šachovnice. Nejprve vyjedu s vězí na tuto souřadnici mimo pole a zde jí posunu o políčko níže. Poté nad ní umístím krále a věž diagonálním pohybem posunu vedle něj. Posledním krokem je posunutí obou figurek o jednu pozici doleva.

### 3.3.4 Braní figury

V kódu braní šachové figury realizujeme právě ve funkci makeChessMove. Pomocí funkce isPlaceOccupied zjistíme zda jsou cílové souřadnice obsazené či volné.

```
1 bool isPlaceOccupied(int x, int y) {
2     if (!boardValues[y-1][x-1]) {
3         return true;
4     }
5     return false;
6 }
```

V případě, že funkce vrátí true, musíme danou figuru odstranit. To provádíme tak, že ji přesuneme mimo šachovnici neboli na 9. horizontální souřadnici. Do funkce makeMove dáme nejprve současné souřadnice magnetu a poté cílové, kde se nachází figurka. Na braní figurek jsem vytvořil funkci handlePieceTaking. Tato funkce má jako vstupní parametry souřadnice odkud a kam.

```
1 void handlePieceTaking(int fromX, int fromY, int toX, int toY) {
2     // Prepare magnet to taken piece
3     makeMove(magnetX, magnetY, toX, toY, false);
4
5     int movingInRow = toY;
```

```

6   for (int blockRightX = toX; blockRightX < 9; blockRightX++) {
7     // Move taken piece out of chess board
8     if (isPlaceOccupied(blockRightX + 1, movingInRow)) {
9       if (movingInRow != 0) {
10         if (!isPlaceOccupied(blockRightX, movingInRow - 1)) {
11           // If piece bellow empty, move there and then continue right
12           makeMove(blockRightX, movingInRow, blockRightX+1,movingInRow-1,1);
13           movingInRow = movingInRow - 1;
14           continue;
15         }
16       }
17
18     if (movingInRow != 8) {
19       if (!isPlaceOccupied(blockRightX, movingInRow + 1)) {
20         // If piece above empty, move there and then continue right
21         makeMove(blockRightX, movingInRow, blockRightX+1, movingInRow+1,1);
22         movingInRow = movingInRow + 1;
23         continue;
24       }
25     }
26   }
27
28   // If empty, move it right
29   makeMove(blockRightX, movingInRow, blockRightX + 1, movingInRow, true);
30 }
31 }
```

Problematika této funkcionality je, že musíme zajistit volný průchod figury na pravou stranu. Nemůžeme figuru jednoduše přemístit na 9. souřadnici doprava, jelikož by mohla narazit do nějaké jiné šachové figury umístěné na políčkách vpravo.

Programový kód funguje takovým způsobem, že se nejprve podívá o jednu pozici doprava, jestliže je volná, přesune tam figuru pomocí funkce makeMove. V momentě, kdy volná není, podívá se o jednu pozici doprava výše. Jestliže je tato pozice volná, přesune se celý cyklus o Y souřadnici výše a opět se pokouší pohnout doprava. Stejným způsobem funkce kontroluje i pozici pod figurkou. Tímto procesem se figura ve většině běžných šachových situací dostane mimo herní pole. Jsou zde i scénaře, ve kterých by se jí to

nepodařilo, ale z důvodu jejich zřídkého vyskytnutí je neřeším.

## 3.4 Algoritmus vytvořeného programu

Algoritmus pro šachovou hru se odehrává v hlavním cyklu programu. V této části práci si vystvětlíme, jak probíhá od samotného spuštění, po tvorbu tahů oponenta, až po ukončení hry.

### 3.4.1 Variabilizace hry

To realizujeme pomocí pátého kontrolního multiplexoru. Tento multiplexor má pouze pět spínačů, a to

**První dva spínače** Hra za bílé či černé figurky

**Další tři** Tyto spínače slouží pro zvolení obtížnosti hry.

V kódu variabilizaci sledujeme pomocí podmínek ve funkci setControlMux, která se volá hned na začátku hry. Nejprve přečteme hodnoty spínačů, poté určíme parametry. V momentě, kdy uživatel zvolil barvu a jeden ze spínačů obtížnosti je spuštěný, zahájíme hru přepnutím proměnné gameStarted na true.

```
1 getReedValues(muxEnable[4], 0, 8);
2 memcpy(controlValues, recordedReedValue, sizeof(controlValues));
3
4 // ...
5 if (!controlValues[5]) {
6     playingAsWhite = false;
7     Q = 8;
8     O = 8;
9     K = 8;
10    R = 8;
11    k = 8;
12 }
13
14 if (!controlValues[4]) {
15     difficulty = 0;
```

```

16 }
17 // ...
18
19 if (( !controlValues[6] || !controlValues[5]) &&
20 (!controlValues[4] || !controlValues[3] || !controlValues[2])) {
21     // Checking if mandatory user info is set
22
23     //Serial.println("Game started");
24     setupMagnet();
25     gameStarted = true;
26 }
```

V případě, že si hráč zvolí začínát hru za černé, je nutné fyzicky prohodit pozice figurek na šachovém poli. Poté začne šachový algoritmus prvním tahem. Ten v kódu provádím jednak nastavním hodnot Q, O, K, R, k na 8 a poté tímto blokem. Ten se provede v případě, že playingAsWhite je nastavný na false a není zaevidován žádný předchozí pohyb algoritmu, neboli je to první tah hry. Pak zavolám Mini-Max funkci getAiMove s prázdnou hodnotou a nový tah provedu.

```

1 if (playingAsWhite == false && lastMoveAI[0] == lastMoveAI[1]) {
2     // AI starting as BLACK
3     setCurrentBoard();
4     memcpy(boardValuesMemory, boardValues, sizeof(boardValuesMemory));
5
6     getAIMove("");
7     makeChessMove(lastMoveAI);
8     resetMove();
9 }
```

### 3.4.2 Průběh hry

V hlavním cyklu programu, po splnění proměnné gameStarted probíhá hra. Kód v cyklu primárně využívá již výše popsané funkce. Zde si ukážeme jejich aplikaci. Samotná záZNAM části šachové partie je v přílohách pod názvem samoridici\_sachovnice.mov. Nejprve získáme hodnoty spínačů pomocí setCurrentBoard, následně tyto hodnoty uložíme do paměti a spustíme detekci pohybu. V případě, že uživatel hraje za bílé figurky, spustí

se funkce detectBoardMovement a běží až do doby, kdy hráč s nějakou figurkou pohně. Tento pohyb se uloží do proměnné move, a ta se předá MiniMax algoritmu pro šachovou hru. Ten rozeberu dále v práci.

```

1 // User's turn => waiting for movement
2 setCurrentBoard();
3 memcpy(boardValuesMemory, boardValues, sizeof(boardValuesMemory));
4 while (!move[0] || !move[1] || !move[2] || !move[3]) {
5     detectBoardMovement();
6 }
7
8 // AI turn
9 getAIMove(move);

```

MiniMax vyhodnotí zda, byl tah uživatele validní a následně zda je hra u konce či ještě pokračuje. V případě, že tah nebyl validní, což se může stát chybou uživatele a přepnutím jiných spínačů. Vyřešíme to resetem tahu pomocí funkce resetMove.

Dále nám algoritmus vrátí ideální tah pro oponenta uložený pod externí proměnnou lastMoveAI. Ta obsahuje šachový tah, v šachovém formátu. V práci jsem již implementoval řešení funkční makeChessMove. Tuto funkci zde zavoláme a tah se provede. Po provedení tahu opět vyresetujeme proměnnou move a cyklus spustíme od začátku.

```
1 makeChessMove(lastMoveAI);
```

### 3.4.3 Možnost hry s protihráčem

Šachovou hru s protihráčem realizujeme pomocí MiniMax algoritmu. Tento šachový algoritmus od pana H. G. Mullera [11]. Program je dostupný pod Open source licencí a pro Arduino ho přepsal pan Diego Cueva. Tento algoritmus jsem objevil v jedné realizaci projektu šachovnice [9] a jeho využití jsem si přepsal pro své potřeby.

Kód pro algoritmus je v odlišném souboru, a to v MiniMax.h a MiniMax.cpp. Samotný kód algoritmu je extrémně optimalizován za účelem nejvyšší možné efektivity a délky. Jména proměnných i funkcí jsou zde zkracována, na co nejkratší a kód je díky tomu těžko srozumitelný. V kódu se ideální tah získá pomocí zavolání funkce D.

```
1 r = D(-I, I, Q, O, 1, 3); /* Think & do*/
```

Funkce D obsahuje výpočet Mini-Maxu na základě parametrů pro výhru oponenta. Tento výpočet je velmi koplikovaný a na stránkách pana Mullera uvedených ve zdrojích vysvětlený dopodrobna. Já zde vysvětlím části, které byly důležité pro projení s mým kódem.

Důležitá je zde funkce getAiMove, která má za vstup šachový tah. Algoritmus si tah rozebere a opět provede výpočet. Zde můžeme upravit obtížnost pod proměnnou v kódu určenou jako velké T. Dále je pro nás důležitý status hry. Ten ukládáme do externí proměnné z hlavního souboru gameStatus, kde ji kontrolujeme. V případě výhry ji stanovíme na 2 a v případě prohry na 1.

```
1 if (! ( r > -I + 1)) {  
2     Serial.println ("Lose ");  
3     gameStatus = 1;  
4 }
```

Poslední částí, kterou bych chtěl zmínit je funkce, která zvaliduje a překopíruje daný šachový tah do proměnné lastMove. Zárověn ještě stanoví validMove, což nám určuje zda byl pohyb legální.

```
1 strcpy (lastMove , c); /* Valid human movement */  
2 validMove = true;
```

## 3.5 Mechanické provedení

Šachovnici bylo zapotřebí umístit do pevného krytu, který bude elektrické součástky chránit před poškozením. Já jsem se rozhodl pro dřevo. Dřevo bylo pro mě volba z důvodu dobrých možností úprav a opracování. Dále se mi také velmi zamlouvalo zachovat dojem klasické šachovnice, která je běžně také dřevěná. Box se skládá ze čtyř zakladních dřevěnných desek. Ze shora a ze spoda je šachovnice uzavřena tenkými překližkami. Samořídící šachovnici jsem v závěru projektu nalakoval ochranou vrstvou. Na šachovnici byly také vypracovány designové prvky a číslování.

Horní překližka má v sobě zasazenou PCB desku, připevněnou pomocí 8 šroubů, které zárověn společně s cínem drží oba kusy desky pohromady. Do desky jsou vyvedeny kabely od motoru a elektromagnetu. Napajení je realizováno bočním otvorem do šachovnice.



Obrázek 3.5: Vzhled herní části šachovnice

### 3.5.1 Herní část šachovnice

Pro vzhled šachovnice jsem si v programu InkScape vytvořil návrh šachových políček, jenž lze naleznout v přílohách pod názvem navrh\_herniho\_pole.pdf. Na pravou část návrhu jsem umístil kontrolní spínače pro volbu hry. Návrh jsem si nechal vytisknout na pevný papír, jenž jsem na desku přilepil. Pro šachovnici jsem si nechal vytisknout vhodné šachové figurky pomocí 3D tisku a do nich vložil malé magnety.

# Kapitola 4

## Závěr

Samořídící šachovnice byla v rámci práce zrealizována. Je schopna detekovat figurky umístěné na jejím herním poli. Rešení byla deska plošného spoje s potřebnými součástkami. Deska obsahuje 69 magnetických spínačů, které skrze multiplexory čteme pomocí řídící jednotky. Jako řídící jednotku jsem si zvolil Arduino UNO, které dostatečně plní všechny potřebné úlohy práce. Šachovnice s mírnými odchylkami umožnuje šachovou hru proti oponentovi, kterého simuluje algoritmus.

Pro účely řízení jsem vytvořil program, který je schopen ovládat všechny potřebné části šachovnice. Program je schopen řešit situace, které během hry mohou nastat. Dokáže zařídit odpověď na šachový tah hráče, který rozpozná na základě sepnutí spínačů. Pro tento účel využívá algoritmus Mini-Max, který pomocí výpočtu zjistí ideální odpovídající tah. Následně tento tah transformuje do signálů, které jsou schopné vykonat krokové motory a přemístí požadovanou šachovou figurku na odpovědné místo.

Samořídící šachovnici jsem zkonstruoval do dřevěné skříně, ve které jsou umístěny všechny potřebné prvky. Je do ní z boční strany přivedeno odpovídající napětí. Toto napětí nám stabilizátor umožní variovat pro motory, elektromagnet a ostatní součástky.

Dovnitř šachovnice jsem sestrojil sestavu krokových motorů, která je schopna provádět pohyb po XY souřadnicích. Tento mechanismus je sestrojen pomocí umístění vnitřní plošiny na vodicí tyče v konstrukci. Touto plošinou je proveden řemen zakončený řemenicí. Na horizontální ose je podobný mechanismus. Je na ní však použit elektromagnet, který pohybuje s figurkami.

Pro projekt jsem zařídil šachové figurky vytvořené pomocí 3D tisku, do kterých jsem umístil malé magnety. Návrh pro herní pole jsem vytvořil a následně ho nechal vytisknout na vhodný materiál. Ten je připevněný na desku plošného spoje.

# Seznam obrázků

2.1	Rozlišení SMT a THT na PCB desce . . . . .	11
2.2	Minimax algoritmus . . . . .	13
3.1	Porovnání PCB desky z výroby a osázené . . . . .	15
3.2	Zapojení magnetických spínačů na desce . . . . .	18
3.3	Ovládání elektromagnetu pomocí transistoru . . . . .	25
3.4	Konstrukce krokových motorů . . . . .	27
3.5	Vzhled herní části šachovnice . . . . .	39

# Příloha A

## Samořídící šachovnice

A.1 schema\_sachovnice.pdf

A.2 navrh\_herniho\_pole.pdf

A.3 samoridici\_sachovnice.mov

# Literatura

- [1] CONRAD.CZ. Arduino® » Praktický mikrokontrolér pro individuální spínací a řídicí úlohy. <https://www.conrad.cz> [online]. ©2024 [cit. 29. 2. 2024]. Dostupné z: <https://www.conrad.cz/cs/clanky/elektromechanika/arduino.html?srsltid=AfmBOoq-DVdYd1lkV0hVguOg5cvqM7O721czoRtYbZBji4hT3SLgxRoY>
- [2] BOTLAND.CZ. Deska PCB – co to je?. <https://botland.cz/> [online]. ©2023 [cit. 13. 3. 2023]. Dostupné z: <https://botland.cz/blog/deska-pcb-co-to-je/>
- [3] UK.RS-ONLINE.COM. reed-switches-guide. <https://uk.rs-online.com> [online]. ©2023 [cit. 1. 2. 2023]. Dostupné z: <https://uk.rs-online.com/web/content/discovery/ideas-and-advice/reed-switches-guide>
- [4] DUBNO.CZ. VY\_32\_INOVACE\_CTE\_2.MA\_13\_Multiplexory. <https://dubno.cz/> [online]. ©2012 [cit. 1.8.2012]. Dostupné z: [https://dubno.cz/images/stories/dum/8.sablonu/24/pdf/VY\\_32,\\_INOVACE\\_CTE\\_2.MA\\_13\\_Multiplexory.pdf](https://dubno.cz/images/stories/dum/8.sablonu/24/pdf/VY_32,_INOVACE_CTE_2.MA_13_Multiplexory.pdf)
- [5] WIKIPEDIA.CZ. Stabilizátory napětí. <https://www.wikipedia.cz> [online]. ©2024 [cit. 22.2.2024]. Dostupné z: [https://cs.wikipedia.org/wiki/Stabiliz%C3%A1tor\\_nap%C4%9Bt%C3%AD](https://cs.wikipedia.org/wiki/Stabiliz%C3%A1tor_nap%C4%9Bt%C3%AD)
- [6] LASTMINUTEENGINEERS.COM. 28byj48-stepper-motor-arduino-tutorial/. <https://lastminuteengineers.com> [online]. ©2025 [cit. 23. 3. 2025]. Dostupné z: <https://lastminuteengineers.com/28byj48-stepper-motor-arduino-tutorial/>
- [7] CS.WIKIPEDIA.ORG. Elektromagnet. <https://cs.wikipedia.org/> [online]. ©2025 [cit. 25. 1. 2025]. Dostupné z: <https://cs.wikipedia.org/wiki/Elektromagnet>
- [8] GEEKSFORGEEKS.COM. mini-max-algorithm-in-artificial-intelligence. <https://www.geeksforgeeks.org/> [online]. ©2024 [cit. 27.8.2024]. Dostupné z: <https://www.geeksforgeeks.org/mini-max-algorithm-in-artificial-intelligence/>

- [9] HTTPS://WWW.INSTRUCTABLES.COM. Automated-Chessboard.  
*https://www.instructables.com* [online]. ©2022 [cit.2.1.2022]. Dostupné z:  
<https://www.instructables.com/Automated-Chessboard/>
- [10] HTTPS://PROJECTHUB.ARDUINO.CC. maguerero/automated-chess-board-67db6f.  
*https://projecthub.arduino.cc* [online]. ©2018 [cit.24.10.2018]. Dostupné z:  
<https://projecthub.arduino.cc/maguerero/automated-chess-board-67db6f/>
- [11] HOME.HCCNET.NL. /h.g.muller/max-src2. *https://home.hccnet.nl/* [online]. ©2025  
[cit.25.3.2025]. Dostupné z: <https://home.hccnet.nl/h.g.muller/max-src2.html>
- [12] PCBA-MANUFACTURERS.COM. SMT vs THT. 2023 [online]. ©23. 3. 2023  
[cit.<https://www.pcba-manufacturers.com/smt-vs-tht/>]. Dostupné z:
- [13] STORE.ARDUINO.CC. Arduino UNO pinout. 2025 [online]. ©19. 3. 2025  
[cit.<https://store.arduino.cc/en-cz/products/arduino-uno-rev3>]. Dostupné z:
- [14] HTTPS://WWW.TPOINTTECH.COM. minimax-algorithm-in-python.  
*https://www.tpointtech.com/* [online]. ©2025 [cit.23.3.2025]. Dostupné z:  
<https://www.tpointtech.com/minimax-algorithm-in-python>