VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NÁVRH A VÝVOJ EDUKAČNÍHO HERNÍHO TERMINÁLU PRO MINIHRY NAPROGRAMOVANÉ V MATLABU

DESIGN AND DEVELOPMENT OF AN EDUCATIONAL GAMING TERMINAL FOR MINI-GAMES PROGRAMMED IN MATLAB

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Petr Černý

AUTHOR

VEDOUCÍ PRÁCE — Ing. Martin Appel, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Petr Černý**
Studijní program:	Mechatronika
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. Martin Appel, Ph.D.**
Akademický rok:	2023/24

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Návrh a vývoj edukačního herního terminálu pro minihry naprogramované v Matlabu

Stručná charakteristika problematiky úkolu:

V rámci výuky programování v Matlabu dostávají studenti úkoly naprogramovat minihry, jako jsou například had, pong a podobně. Abychom zvýšili jejich motivaci k programování těchto her, plánujeme v naší laboratoři Mechlab instalovat herní automat. Na tomto automatu budou moci studenti spustit a otestovat své vytvořené hry. Zařízení bude vybaveno joystickem a sadou tlačítek pro ovládání. K identifikaci studentů bude sloužit RFID čtečka karet. Součástí tohoto systému bude také aplikace, která umožní přidávat nové hry vytvořené studenty.

Cíle bakalářské práce:

– Návrh a vývoj fyzického zařízení
– Integrace RFID čtečky pro uživatelskou identifikaci
– Vývoj software pro správu her a uživatelského rozhraní
– Testování a validace systému

Seznam doporučené literatury:

GREPL, Robert. Kinematika a dynamika mechatronických systémů. Brno: Akademické nakladatelství CERM, 2007. ISBN 978-80-214-3530-8.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2023/24

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

Abstrakt

Tato práce se zabývá tvorbou zařízení, které má sloužit, jako pracovní pomůcka pro studenty učící se programování. Během práce vznikla uživatelská aplikace, umožnující vložit naprogramovanou hru studenta a následně jí spustit. Aplikace také udržuje databázi uživatelských účtů pomocí identifikace uživatelů, která probíhá díky RFID technologii a studentských ISIC karet. Pro ovládání zařízení byla vytvořena deska plošných spojů, komunikující s RFID čtečkou a počítačem.

Summary

This bachelor's thesis deals with creating a device, suitable as a learning module for students learning programming. While creating this thesis a user application was created that is able to handle importing a game programmed by student and running it. The application maintains a database containing a list of users ands uses RFID technology and ISIC cards for identification. For controlling the device a printed circuit board was created that communicates with RFID reader and PC.

Klíčová slova

MATLAB, OOP, Parallel computing toolbox, RFID, HTML, DPS

Keywords

MATLAB, OOP, Parallel computing toolbox, RFID, HTML, PCB

Bibliografická Citace

ČERNý, P. *Návrh a vývoj edukačního herního terminálu pro minihry naprogramované v Matlabu*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2024. 56 s., Vedoucí diplomové práce: Ing. Martin Appel, Ph.D..

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a že jsem uvedl veškeré použité zdroje a literaturu.

Petr Černý

Brno . . . .

Rád bych poděkoval vedoucímu bakalářské práce Ing. Martinu Appelovi, Ph.D. za cenné rady a připomínky.

Petr Černý

Obsah

1 Úvod

Herní průmysl je důležitou součástí dnešního světa. I když arkádové hry jsou už dnes spíše zastaralou technologií, stále bývají prvním větším projektem studentů učících se programování. Ne jinak tomu je na oboru Mechatronika, kde studenti mají v prvním ročníku možnost, v případě zájmu, programovat pod vedením svojí vlastní hru. Tyto hry studenti programují v programovacím jazyku MATLAB, jehož základy se v prvním ročníku vyučují.

Tento programovací jazyk umožňuje psaní kódu pomocí objektově orientovaného programovaní, OOP, nicméně tento přístup je v jazyku MATLAB poměrně málo využíván. To je pravděpodobně zapříčiněno tím, že OOP bylo do MATLABu přidáno relativně nedávno, v roce 2008, oproti ostatním programovacím jazykům, kterých součástí bývá už od druhé poloviny 20 století. Taktéž je MATLAB používán spíše inženýry, analytiky a statiky pro rychlé prototypování a datovou analýzu, což nejsou typické odvětví pro použití OOP. Nicméně OOP je velmi důležitým odvětvím programovaní, a často velmi usnadňuje psaní kódu.

1.1 Stanovené cíle

Tato práce si stanovuje za cíl být primárně pomůckou pro studenty v rámci jejich studia programování. Podpořit studenty v učení programování pomocí objektů, a možná zvýšit také jejich motivaci k tomu, aby se do projektu zahrnující naprogramování vlastní hry pustili.

V rámci této práce by mělo vzniknou zařízení, fungující jako arkádový herní automat, který by dovoloval studentům vložit jejich naprogramovanou hru a otestovat jí. Dále by měl obsahovat uživatelskou aplikaci, která by na tomto zařízení běžela a umožňovala spuštění a hraní her s možností ukládání skóre a udržování tabulek nejlepších hráčů. Pro ukládání skóre je důležité, aby zařízení mělo možnost vytvoření vlastních uživatelských profilů a nějakou formu identifikace, ideálně pomocí RFID. Posledním bodem zadání poté bylo testování a validace zařízení.

Obrázek 1.1: Finální podoba zařízení

2 Rešerše

Tato kapitola se zabývá primárně objektově orientovaným programováním (OOP), nástrojem Parallel Computing Toolbox v jazyku MATLAB, a RFID komunikací. Tyto témata bylo potřeba prostudovat pro tvorbu této bakalářské práce.

2.1 Objektově-orientované programování

S tím jak se postupně vyvíjí technologie obvykle také stoupá komplexita řešení. Často pak příliš komplexní řešení zapříčiní výskyt chyb. Poté bývá zapotřebí přesunout se k novým způsobům řešení. Jedním z relativně nových přístupů k řešení programátorských problémů je psaní kódu pomocí *Objektově orientované programování*, OOP. [1]

„In the old days, programmers starting a project would sit down almost immediately and start writing code. However, as programming projects became large and more complicated, it was found that this approach did not work very well. The problem was complexity.“[1]

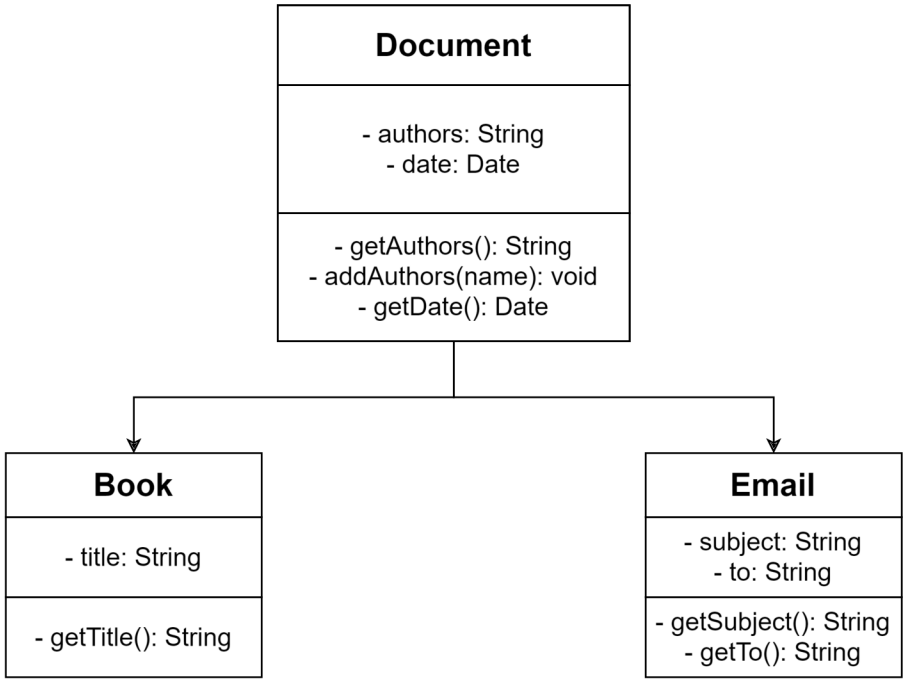
OOP se poté stalo velmi populární a dnes je neodmyslitelnou součástí programování. Poskytuje nový nadhled nad programováním, a místo psaní programu jako sérii kroků, které jsou vykonávány po sobě, se OOP realizuje pomocí objektů, které mají určité vlastnosti a mohou provádět určité akce. [2] Výsledkem toho je kód, který je čistší, má lepší čitelnost, je více stabilní a lehčeji udržovatelný. [3]

2.1.1 Terminologie

Hlavními termíny doprovázející OOP jsou *Objekt* a *Třída*. [4] Třída je něco jako šablona, definuje konkrétní vlastnosti a akce. Objekt je poté instance této třídy, má dané vlastnosti a může vykonávat dané akce, ale každý objekt se může od jiného objektu v těchto vlastnostech lišit. Například můžeme mít třídu kancelářská židle, která má vlastnost barvu, a poté dva objekty pánskou kancelářskou židli, jejíž barva je černá a dámskou kancelářskou židli, jejíž barva je červená. Jedná se o různé objekty stejné třídy.

Dalším důležitým termínem je *Dědičnost*, která umožňuje dědit třídám vlastnosti a akce jiných tříd a vytvářet tak komplexní struktury tříd. Třída vytvořená tak, že dědí od jiné, nadřazené třídy, pak získává všechny, nebo jen některé, vlastnosti a akce od této nadřazené třídy. Schéma příkladu dědičnosti lze vidět na obrázku 2.1. Nadřazená třída *Document* má vlastnosti *authors* a *date*. Podřadné třídy *Book* a *Email*, které dědí z nadřazené třídy, poté taky mají tyto vlastnosti, ale navíc má každá definované, ještě své unikátní, *title*, respektive *subject* a *to*.

Posledním termínem vhodným zmínit je *zapouzdření*. Jedná se o stav, kdy jakákoliv interakce s objektem je nutně prováděna skrz jeho akce, k tomuto určené, a je omezen celkový přístup k vnitřku objektu. To má za následek lepší čitelnost kódu a hlavně bezpečnost, tedy nemůže se stát, že by se například přepsala hodnota objektu z místa, které by k tomu nebylo určené.



Obrázek 2.1: Diagram dědičnosti

2.1.2 OOP v MATLABu

Programovací jazyk MATLAB také podporuje psaní programů pomocí OOP. Struktura použití je podobná jako u jiných programovacích jazyků, ale rozdílem oproti jiným je rozlišování dvou druhů tříd.

MATLAB definuje různé datové typy, jako je například vektor, matice, double, tedy definuje význam hodnot, kterých smí proměnné takto definované nabývat. Prvním druhem tříd, které MATLAB rozlišuje je tzv. *Value class*. Jedná se o datový typ. Pokud se vytvoří objekt, který je typu Value class a následně se vytvoří kopie, tato kopie se stane novým nezávislým objektem.

Druhá třída je *Handle class*. Zde se jedná o referenční typ. Pokud se vytvoří objekt typu Handle class a následně zkopíruje, vytvoří se pouze reference na původní objekt. Změna jednoho objektu ovlivňuje i druhý, protože se jedná o stejný objekt. [5]

```
% Example of Value vs Handle class behavior

handleObj1 = HandleClassExample();
handleObj2 = handleObj1;
% handleObj2 references the same object as handleObj1

valueObj1 = ValueClassExample();
valueObj2 = valueObj1;
% valueObj2 is a copy of valueObj1;
% changes to valueObj2 do not affect valueObj1
```

2.2 Paralelní vlákna

Paralelní programování je schopnost rozdělit obsáhlý program do více menších programů a ty poté vykonávat současně za účelem zrychlení běhu programu, který běží zároveň na několika jádrech počítače.

Programovací jazyk MATLAB standardně v jeho základní verzi nepodporuje paralelní programování, ale nabízí řešení v podobě *Parallel Computing Toolboxu*. Toolbox je sada funkcí a tříd, které poskytují nástroje pro specifické téma, například právě paralelní výpočty. [6]

2.2.1 Hlavní funkce

Součástí Parallel Computing Toolboxu je několik funkcí umožnující paralelní programování.

První z nich je funkce *parfor*, jedná se o paralelní verzi for smyčky, která dovoluje jednotlivým iteracím běžet současně.

Funkce *parfeval* umožňuje asynchronní exekuci pomocí tzv. paralelních workerů.

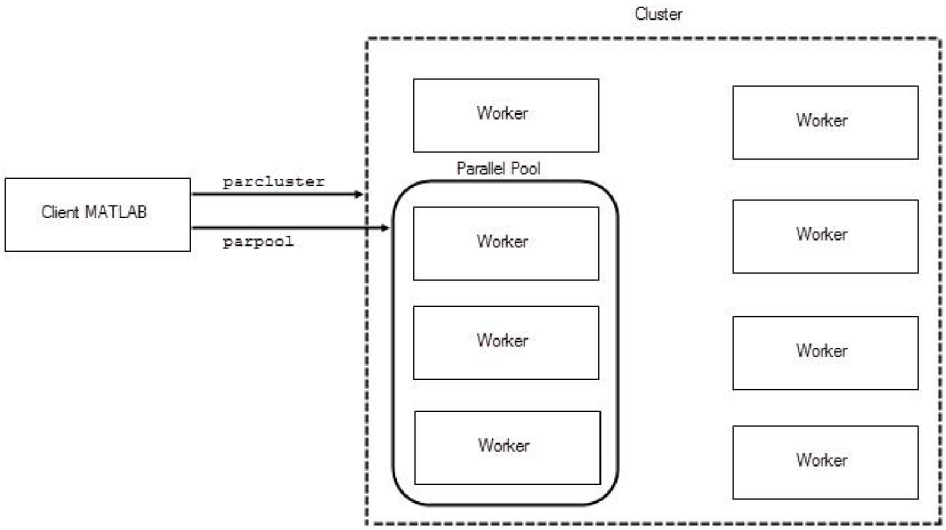
Funkce *spdm* dovoluje vykonat nějakou část kódu na všech paralelních workerech, které se nachází v tzv. parallel pool.

V předešlém popisu se objevily termíny *parallel worker* a *parallel pool*. *Paralelní worker* je výpočetní proces, který běží bez grafického rozhraní. Dá se přirovnat k spuštění nového procesu MATLABu v novém okně, který je nezávislý na tom hlavním. Paralelní worker obsahuje vlastní workspace a vlastní proměnné. *Paralelní Pool* je poté seskupení těchto workerů, které jsou připraveni na nějaký výpočet. Počet workerů uvnitř paralelního poolu se může libovolně definovat v mezích daných několika faktory, primárně se jedná o počet jader počítače a limit licence Parallel Computing Toolboxu. Paralelní pool lze vidět na obrázku 2.2. Pomocí funkce *parpool* se vytvoří skupina workerů. Ještě širším pojmem je *cluster*, zde se většinou jedná o spolupráci několika počítačů. [7]

2.2.2 Parfeval

Funkce *parfeval* umožňuje vykonat funkci v pozadí pomocí paralelního poolu, bez toho aby blokovala hlavní proces MATLABu. Z pravidla se tato funkce používá a je výhodná pro simulace, které zabírají hodně času, zpracování dat nebo komplexní výpočty. Zde poté bývá výhodné nečekat na ukončení těchto procesů, ale raději nechat je pracovat v pozadí a data si vyzvednout po ukončení.

K tomuto účelu vrací funkce tzv. *Future* objekt. Tento objekt obsahuje informace o běžící paralelní funkci, a pokud je potřeba nějaké interakce s paralelně běžící funkcí je nutné provést tak přes tento objekt. Primárně pomocí něho můžeme sledovat stav funkce, tedy jestli stále probíhá, popřípadě funkce skončila, dále lze funkci předčasně zrušit nebo získat výsledek.



Obrázek 2.2: MATLAB Parallel pool [7]

2.2.3 Komunikace mezi paralelními workery

Často je potřeba získat z paralelně běžící funkce data dříve než po ukončení výpočtu, popřípadě je potřeba reagovat na nějaké podmínky při chodu. Zde bývá zapotřebí vytvořit mezi workerem, který vykonává paralelně běžící funkci, a hlavním klientem nějakou formu komunikace.

Samotná funkce *parfeval* neumožňuje komunikaci, mezi těmito stranami. Nicméně můžeme využít objektů, které jsou součástí parallel pool, *DataQueue* a *PollableDataQueue*. Jedná se o objekty, které umožňují komunikaci mezi klientem a paralelním workerem i během toho, co je prováděn výpočet, a můžeme tak například získat mezi-výpočty a ty poté použít v klientovi, mezitím co se čeká na dokončení hlavních výsledků.

Oba objekty jsou velmi podobné, DataQueue umožňuje asynchronní komunikace a PollableDataQueue umožňuje synchronní komunikaci.

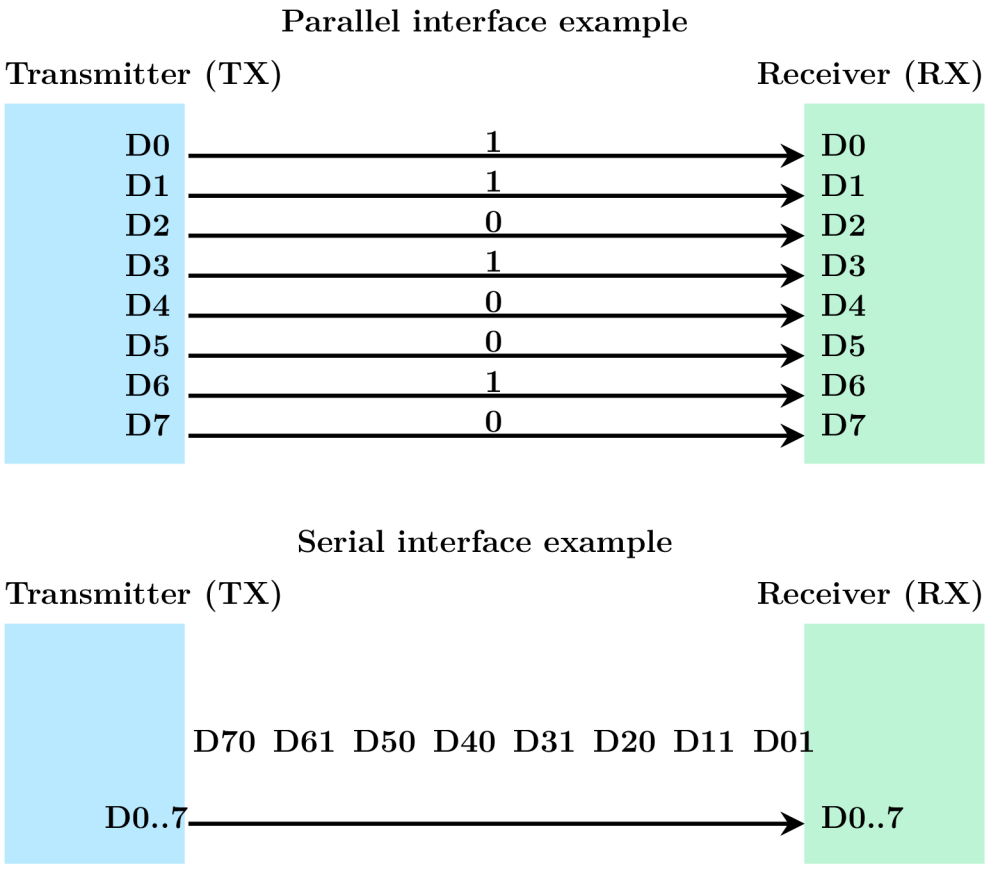
Objekty se nejprve vytvoří a následně vloží jako vstup do funkce *parfeval*. Následně můžeme využít funkcí *send* a *poll*, pro objekt PollableDataQueue. Nebo funkce *afterEach* a *send* pro objekt DataQueue. [8]

2.3 Sériová komunikace

Sériová komunikace je jedním z nejdůležitějších aspektů digitální elektroniky a počítačového inženýrství. Komunikace mezi periferiemi a centrální procesorovou jednotkou (CPU) může probíhat buďto paralelně, kde se jednotlivé bity posílají naráz, nebo sériově, kde se posílají bity postupně za sebou. Rozdíl mezi těmito typy lze vidět na obrázku 2.3.

Při zasílání dat mezi zařízeními lze zaslat pomocí komunikační linky buď logická 1 nebo logická 0. Tedy v případě zaslání například slova *Ahoj*, z jednoho zařízení do druhého, je nutné zaslat data pomocí binárního kódu a ASCII tabulky, kde každé písmeno odpovídá nějakému číslu. Pomocí sériové komunikace by se tedy muselo zaslat následujícího řetězce.

$$1000001\ 1101000\ 1101111\ 1101010 \tag{2.1}$$



Obrázek 2.3: Sériová vs paralelní komunikace [9]

2.3.1 Interpretace dat

Při zasílání dat, která by vypadala jako na obrázku 2.4 by nemusel být problém s interpretací dat a lze vidět, že se jedná o řetězec *01010101*. Pokud by se ale zasílali data, jako na obrázku 2.5 tak se zde objeví problém s interpretací, jelikož nelze jasně určit, jestli se jedná o řetězec dat *10*, *1100* nebo například *11110000*. Je potřeba zavedení způsobu, kterým se odliší délka jednoho bitu. Toho lze dosáhnout dvěma způsoby, první možností je přidání další linky, takzvaná *Clock* linka, která udává hodinový signál, zde se jedná o *Synchronní komunikaci*. Druhou možností je definování interního počitadla, na kterém se shodnou obě strany a obě by tak věděly, po jak dlouhou dobu se bude posílat jeden bit. Zde se jedná o *Asynchronní komunikaci*.

Obrázek 2.4: Interpretace sériové komunikace část 1 [10]

Obrázek 2.5: Interpretace sériové komunikace část 2 [10]

Obrázek 2.6: Synchronní vs Asynchronní sériová komunikace [10]

2.3.2 Přenosové módy

Další možností rozdělení komunikace je z hlediska módu přenosu, to znamená které strany, a kdy můžou komunikovat.

První možností je mód *Simplex*, jedná se o jednostrannou komunikaci, tedy pouze jedno zařízení může zastávat funkci vysílání, druhé zařízení je poté schopné pouze přijímání dat.

Druhou možností je mód *Half Duplex*. Tento mód vyjadřuje způsob komunikace, kdy možnost vysílat i přijímat mají obě zařízení, ale pouze jedno zařízení může v daný čas vysílat.

Poslední možností je *Full Duplex*, který umožňuje obou zařízením vysílat i přijímat, a to i ve stejný čas.[11]

2.3.3 Další termíny sériové komunikace [11]

Koncem této sekce je potřeba ještě definovat další důležité termíny z hlediska komunikace.

Baud Rate vyjadřuje přenosovou rychlost, kterou jsou data posílána mezi oběma zařízeními. Jedná se o jednotku, která vyjadřuje kolik bitů za sekundu se odešle, bps. Tyto hodnoty jsou standardizované, například, 1200, 2400, 4800, 9600, 115200 a vycházejí z minulosti, kdy první video terminály komunikovali rychlostí 300 bps, což zhruba odpovídalo 30 vteřinám, z této hodnoty se poté vycházelo a proto jsou všechny dnešní standardní hodnoty násobkem 300.

Synchronizační bity jsou bity, které označují začátek a konec komunikace.

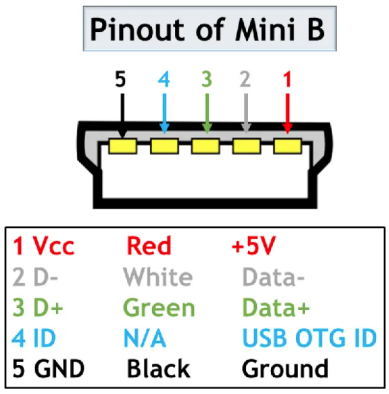
2.3.4 Komunikační protokoly

Komunikační protokol je souhrn pravidel, kterými se musí komunikace řídit, tak aby obě strany mohly dosáhnout bezproblémové komunikace. V následujících sekcích budou popsány protokoly, které byly použity v mé bakalářské práci.

USB protokol

Universal Serial Protocol, USB, se poprvé objevil v PC a slouží pro připojení externích zařízení. Důvodem vzniku protokolu bylo sjednocení konektorů. USB protokol byl poprvé představen ve verzi USB 1.0, a od jeho uvedení prošel protokol různými změnami a úpravami a momentálně je poslední verzí USB 4.0. V této práci se využívalo protokolu USB 2.0 a tedy bude více popsán.

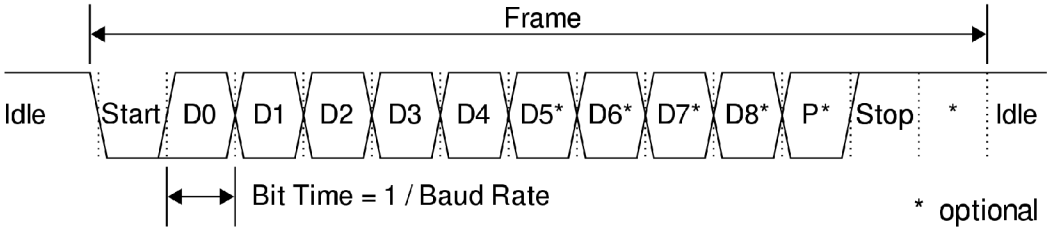
Tento standard vyšel na trh v roce 2000 a označuje se jako Hi-speed USB. Jeho přenosová rychlost je 480 Mbps. Na obrázku 2.7 lze poté vidět pinout konektoru USB-B mini. Jak je vidět tato verze protokolu využívá 4 nebo 5 vodičů. 2 vodiče slouží pro napájení externího zařízení. Data se posílají v USB protokolu pomocí diferenciálního páru, tj. dva komplementární signály pomocí dvou vodičů, D+ a D-. Diferenciálního páru se využívá kvůli zmenšení šumu, a také odolnější komunikaci na delší vzdálenosti. Konektory typu B poté využívají ještě 5 vodič, ID, který slouží pro USB-OTG, tedy aby se mohlo zařízení připojit, jako periferie, ale sloužit také jako master. Komunikace tohoto protokolu je poté typu Half-duplex. [12]



Obrázek 2.7: Konektor USB-B mini [13]

UART protokol

Uninversal Asynchronous Receiver/Transmitter, UART, je asynchronní komunikace a je možné použít všechny módy přenosu, tedy full-duplex, half-duplex i simplex. UART modul se skládá z datového registru a posuvného registru, který postupně posílá nebo přijímá data. Na obrázku 2.8 lze vidět složení každé UART komunikace. Nejdříve se zasílá Start bit, následně 5-8 datových bitů, volitelný parity bit, pro kontrolu znehodnocených dat a nakonec Stop bit, indikující konec komunikace. [14]



Obrázek 2.8: Sekvence UART komunikace [15]

2.4 Identifikační systémy

V moderním světě je kladen velký důraz na robustní a bezpečné identifikační systémy. Byly vyvinuty různé technologie, které zastávají široké spektrum potřeb od udržování stavu zásob až po zajištění bezpečnosti pomocí kontrolovaného přístupu.

Revoluci identifikačních systémů zahájil vývoj čárového kódu a stal se rychle velmi populární, nicméně jeho nevýhody, jako omezený počet místa k uložení dat, donutily k vývoji dalších modernějších technologií, kterým se staly kontaktní karty, výhodné jednak svojí velikostí, a jednak možností uložení násobně většího počtu dat. Logický krok dopředu odtud byly poté bezkontaktní karty, které ideálně nepotřebují vlastní napájení, ale získají ho taktéž bezdrátově ze čtecího zařízení. Tímto vznikají Radio-frequency identification systems, RFID. [16]

Na obrázku 2.9 můžeme vidět jednoduché srovnání identifikačních systémů používaných v dnešní době. Lze vidět, že oproti čárovému kódu, který zvládne uložit do stovky bajtů, dokáží RFID systémy uložit až 64 tisíc bajtů.

RFID systémy hrají důležitou roli v identifikaci studentů na univerzitách. Velmi často se využívají studentské karty, které využívají technologie RFID. V Evropě jsou velmi používané studentské karty firmy ISIC, které jsou rozšířené do 98 zemí [17]. Tyto karty využívá i univerzita VUT, která momentálně využívá karty používající protokol Mifare.

System parameters	Barcode	OCR	Voice recognition	Biometry	Smart card	RFID systems
Typical data quantity (bytes)	1–100	1–100	–	–	16–64 k	16–64 k
Data density	Low	Low	High	High	Very high	Very high
Machine readability	Good	Good	Expensive	Expensive	Good	Good
Readability by people	Limited	Simple	Simple	Difficult	Impossible	Impossible
Influence of dirt/damp	Very high	Very high	–	–	Possible (contacts)	No influence
Influence of (optical) covering	Total failure	Total failure	–	Possible	–	No influence
Influence of direction and position	Low	Low	–	–	Unidirectional	No influence
Degradation/wear	Limited	Limited	–	–	Contacts	No influence
Purchase cost/reading electronics	Very low	Medium	Very high	Very high	Low	Medium
Operating costs (e.g. printer)	Low	Low	None	None	Medium (contacts)	None
Unauthorised copying/modification	Slight	Slight	Possible* (audio tape)	Impossible	Impossible	Impossible
Reading speed (including handling of data carrier)	Low ~4 s	Low ~3 s	Very low >5 s	Very low >5–10 s	Low ~4 s	Very fast ~0.5 s
Maximum distance between data carrier and reader	0–50 cm	<1 cm Scanner	0–50 cm	Direct contact**	Direct contact	0–5 m, microwave

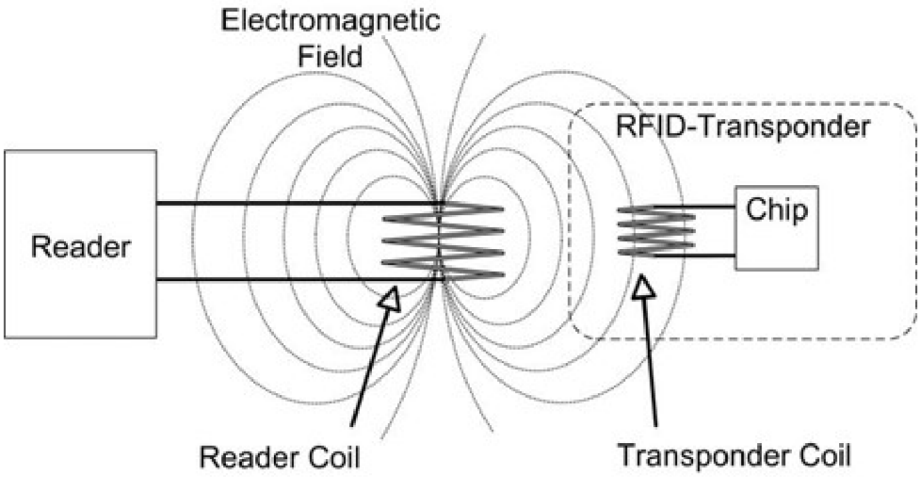
Obrázek 2.9: Srovnání různých identifikačních systémů [16]

2.4.1 RFID

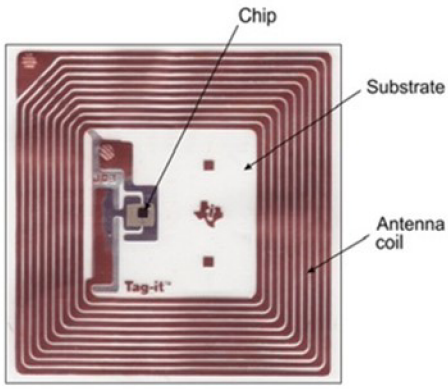
RFID technologie je bezdrátová komunikace, která využívá elektromagnetické nebo elektrostatické pole v pásmu radiových vln a slouží pro identifikaci a komunikaci s objekty. Každý RFID systém je tvořen dvěma komponenty, *transpondér*, ten se nachází na objektu, který má být identifikován a může být různých tvarů, z různých materiálů a různé velikosti. Skládá se z mikročipu pro ukládání paměti a antény. Ukázku typického RFID transpondéru, který se používá jako nálepky, je ukázán na obrázku 2.11.

Čtecí zařízení, je druhým komponentem potřebným pro RFID komunikaci, je složitější oproti transpondéru a skládá se z RF transceiveru, kontrolní jednotky, která se stará o komunikaci, antény a zpravidla také nějakého sériového rozhraní pro další komunikaci s nadřazeným systémem.

Schéma funkčnosti RFID systému je zobrazeno na obrázku 2.10.



Obrázek 2.10: Princip funkce RFID systému [18]



Obrázek 2.11: Vnitřek RFID transpondéru [19]

RFID systémy se dají rozdělit podle několika kritérií. První dělení je z hlediska pracovní frekvence a dělí se na systémy *Nízké frekvence*, které obvykle pracují s frekvencí kolem 130 kHz a jejich čtecí vzdálenost je typicky nižší než 10 cm.

Systémy *Vysoké frekvence* pracují na frekvenci 13,56 MHz a jejich maximální čtecí vzdálenost je do 1 metru.

Systémy *Ultra vysoké frekvence* pracují na frekvenci od 300 MHz do 3 GHz a lze s nimi komunikovat typicky až na 12 metrů. [16]

Dalším dělení je z hlediska energie RFID transpondéru, ty mohou být buď *pasivní*, které nemají vlastní zdroj energie a je nutné energii dodat z čtecího zařízení. Transpondéry *aktivní* mají vlastní zdroj energie a tedy nepotřebují spoléhat na čtecí zařízení. Zároveň existuje kombinace těchto možností a existují tzv. *semi-pasivní* transpondéry, které obsahují baterii, ta ale většinou dodává energii pouze mikročipu a nedodává energii pro přenos signálu. [16]

3 Řešení

Tato kapitola se bude zabývat samotným řešením této bakalářské práce. Nejprve vývojem uživatelské aplikace sloužící jako pomůcka pro studenty v programování jejich her, a také pro testování a hraní těchto her. Dále vývojem desky plošných spojů implementující sběr uživatelských vstupů a RFID čtečku a v poslední řadě tvorbu samotného fyzického zařízení.

3.1 Uživatelská aplikace

Jedním z hlavních aspektů této bakalářské práce byl návrh a vytvoření uživatelské aplikace. Tato aplikace by měla sloužit jako pracovní pomůcka pro studenty, kteří by si chtěli vyzkoušet naprogramovat vlastní hru pomocí OOP a následně ji otestovat. Aplikace umožňuje jednoduché vkládání vlastních naprogramovaných her, vytvoření uživatelského profilu a udržování skóre u jednotlivých her.

Samotná aplikace je rozdělena do dvou části, frontend a backend. Frontend aplikace je vytvořen v jazyce HTML a jeho funkčnost bude popsána v sekci 3.2. Backend aplikace je tvořen v jazyce MATLAB. Finální podobu aplikace lze vidět na obrázku 3.7.

3.1.1 Struktura programu

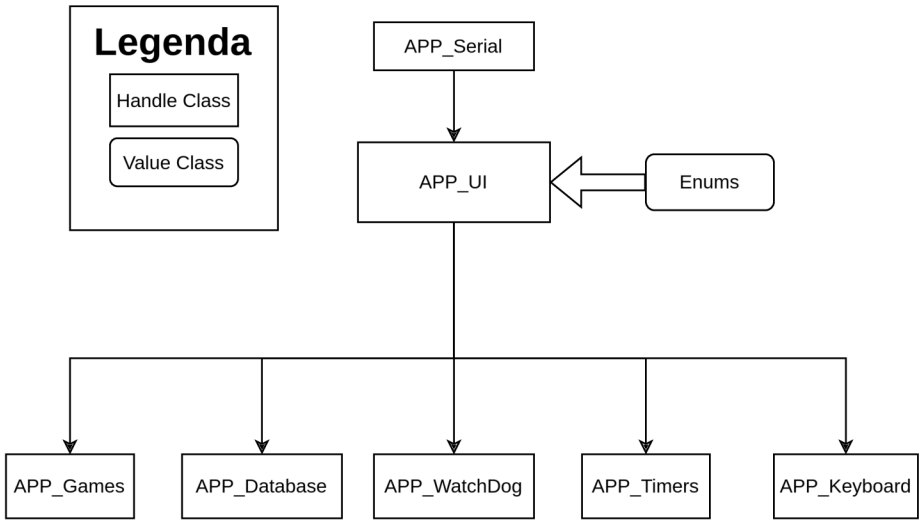
Aplikace je naprogramována pomocí OOP, které bylo popsáno v sekci 2.1. Programování pomocí objektů bylo zvoleno, jelikož výrazně ulehčuje práci a vytváření uživatelských aplikací v jazyce MATLAB.

Samotný program je rozdělený do celkem 8 tříd, kde každá obsahuje funkce a vlastnosti týkající se nějakého okruhu aplikace. Toto výrazně zlepšuje čitelnost a přehlednost kódu. Struktura aplikace lze vidět na obrázku 3.1 a skládá se z 7 tříd typu handle a jedné třídy typu value. Jednotlivé třídy budou popsány v následujících sekcích.

3.1.2 Třída APP_UI

Hlavním bodem aplikace je třída *APP_UI*. Tato třída má primárně na starost vytvoření a interakci s grafickým uživatelským rozhraním naprogramovaným v jazyku HTML, dále volá metody ostatních tříd v případě potřeby, a také slouží pro interakci s externě naprogramovanou hrou studenty, kteří můžou používat veřejné funkce a vlastnosti, které tato třída taktéž obsahuje.

Celá aplikace je rozdělena do tří samostatných panelů, tedy MATLAB objektů typu *uipanel*, které mezi sebou přepínají v závislosti na aktuální potřebě. Tyto panely jsou *Panel_Main*, který zobrazuje hlavní menu aplikace, tedy výběr her, konzoly pro zobrazování hlášek, logo hry, autora a QR kód s odkazem na autorův GitHub.



Obrázek 3.1: Struktura programu

Dalším panelem je *Panel_Game*, tento panel obsahuje prostředí pro zobrazení hry a tabulky vítězů.

Posledním panelem je *Panel_NewUser*, který se zobrazí při vytváření nového uživatelského profilu.

Důležitou funkcí této třídy je také zpracování uživatelských vstupů, na základě těchto vstupů poté komunikuje s frontendem aplikace, aby se mohl vzhled aplikace aktualizovat, popřípadě třída dále přeposílá uživatelské vstupy do studenty naprogramované hry, kde si buďto student vstupy může zpracovat sám, nebo použít již zpracované vstupy.

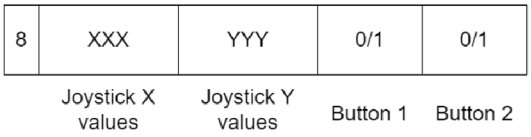
Třída APP_UI má také na starost spuštění hry. Spuštění hry probíhá ve dvou fázích, první fází je uložení reference na třídu se hrou a následné zavolání konstruktoru hry. Druhou fází je poté vytvoření časovače, který ve stanoveném intervalu volá funkci hry, popsaný v sekci 3.1.7, která se chová jako hlavní smyčka. Jak bude popsáno v sekci 3.1.4 hlavní třída s vytvořenou hrou musí mít stejný název jako adresář hry, to je kvůli tomu aby se podle tohoto názvu zavolal z třídy APP_UI správný soubor, který obsahuje hru.

3.1.3 Třída APP_Serial

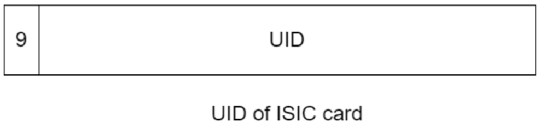
Třída *APP_Serial* má na starost komunikaci s deskou plošných spojů. Po připojení desky k počítači pomocí USB se vytvoří virtuální COM port, ke kterému se MATLAB připojí a následně vytvoří callback funkci. Tato funkce je zavolána po každé přijaté zprávě, která přijde z desky plošných spojů a následně zprávu zpracuje.

Z desky plošných spojů mohou přicházet data dvojího typu. Prvním typem je zasílaný řetězec uživatelských vstupů, tedy analogové hodnoty odečtené z joysticku, zde se zasílají data od 0 do 100, pro každou osu, X a Y. Hodnota 50 znamená, že je osa joysticku v klidové poloze, hodnoty blížící se 0 a 100 poté indikují vychýlení joysticku. Dále řetězec obsahuje logickou hodnotu tlačítek. Strukturu tohoto řetězce lze vidět na obrázku 3.2, kde první hodnota *8* indikuje, o který typ řetězce se jedná.

Druhý typ řetězce se zašle pokaždé, když RFID čtečka detekuje studentskou ISIC kartu v blízkosti. Tento řetězec je zobrazen na obrázku 3.3 a obsahuje pouze hodnotu *9*, která opět indikuje, o který typ zprávy se jedná, a dále samotné UID ISIC karty.



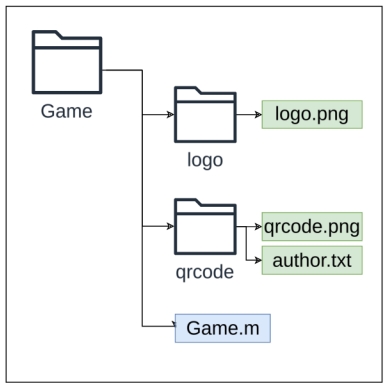
Obrázek 3.2: Řetězec dat



Obrázek 3.3: Řetězec ID

3.1.4 Třída APP_Games

Třída *APP_Games* vznikla, jelikož bylo potřeba implementovat jednoduchou možnost vkládání vlastních her. Toho bylo docíleno vytvořením sdílené složky, například pomocí služby Google Drive, která je umístěna na disku počítače. K této složce má MATLAB aplikace přístup a pozná, pokud se do složky importuje nová složka s hrou. Složky se studenty naprogramovanými hrami musí dodržovat předem definovaná pravidla pro správnou funkčnost. Tato struktura je zobrazena na obrázku 3.4. Pravidla, která musí složka s hrou dodržovat jsou primárně ta, že se složka musí jmenovat stejně jako MATLAB soubor, který obsahuje hru. Pokud by tak nebylo, aplikace by hru nebyla schopná správně načíst. Další pravidla nejsou povinná, pouze doporučená a hra by měla fungovat bez problémů i bez jejich dodržení. Tedy že složka může obsahovat další dvě složky *logo*, která obsahuje PNG nebo JPG soubor s logem hry, a *qrcode*, která obsahuje PNG nebo JPG soubor s QR kódem s odkazem na autorův GitHub a zároveň textový soubor *author.txt*, který obsahuje jméno autora.



Obrázek 3.4: Struktura složky s hrou

Třída *APP_Games* poté sdružuje funkce pro interakci s touto sdílenou složkou, obsahující hry. Při inicializaci aplikace tato třída uloží všechny adresovatelné složky uvnitř do vektoru a udržuje si tak přehled o stavu složky s hrami. Následně třída zobrazí logo a QR kód hry, která je ve vektoru první. Třída poté definuje časovač, který periodicky

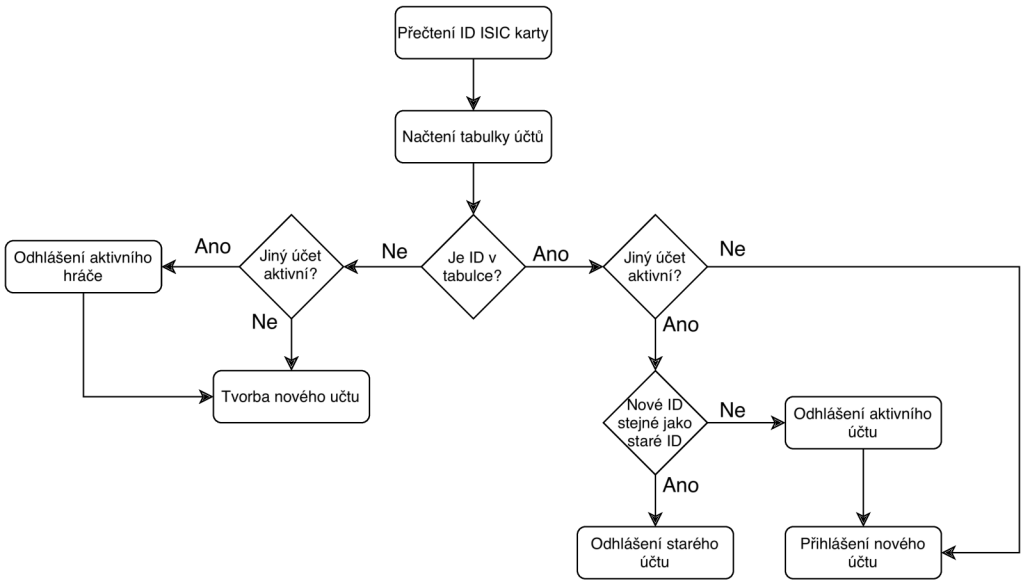
kontroluje stav složky a sleduje, jestli se nezměnila velikost. Pokud ano třída opět načte všechny nové adresovatelné složky a nahradí původní vektor. Časovač kontroluje tuto složku každou 1 vteřinu, tak aby reakce na vložení hry nebyla velmi pomalá, ale zároveň časovač příliš neblokoval běh aplikace.

3.1.5 Třída APP_Database

Pro aplikaci je důležitým úkolem udržování databáze uživatelských profilů. Tyto profily se vytváří pomocí studentských ISIC karet, a následně je UID ISIC karty přiděleno vytvořenému profilu. Tato třída, třída *APP_Database*, obsahuje funkce pro interakci se souborem, který obsahuje databázi uživatelských profilů, pro zpracování načtených dat z ISIC karty a pro ukládání skóre jednotlivých her.

Při prvním načtení ISIC karty je uživatel povinen vytvořit si uživatelský profil zadáním jména, které poté bude zobrazeno v tabulce nejlepších hráčů. Toto jméno je poté přiřazeno danému UID odpovídající ISIC kartě a uloženo do tabulky v textovém souboru, který je uložen v adresáři MATLAB aplikace. Při opětovném načtení ISIC karty se tabulka prohledá, a po zjištění shody načte příslušné jméno z tabulky. Schéma této smyčky je zobrazeno na obrázku 3.5.

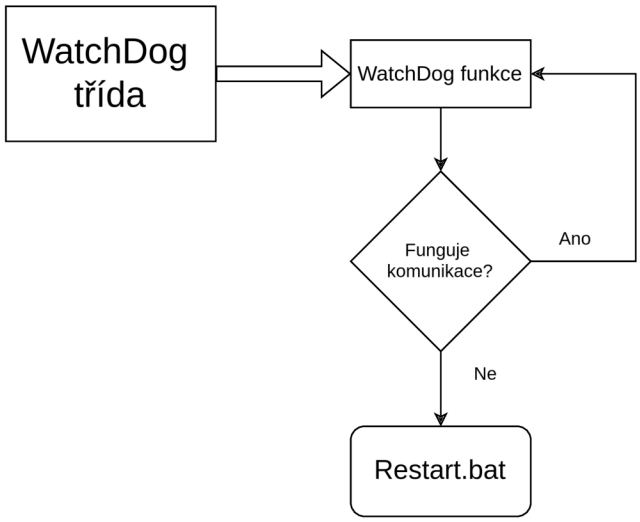
Ukládání skóre jednotlivých her je provedeno zavoláním příslušné funkce této třídy. Uložení skóre si ovládá každá hra samostatně a autor hry ji může provést kdykoliv. Po zavolání funkce se v adresáři hry vytvoří textový soubor *score.txt*, popřípadě otevře, pokud je už soubor vytvořen. Tento soubor poté obsahuje tabulku, kde ke každému profilu, který hru hrál, je přiřazeno jeho maximální skóre. Pokud se zavolá tato funkce, skóre z dané hry se uloží k účtu, který je momentálně aktivní. Skóre se vždy přepíše pouze pokud je lepší než dosavadní skóre.



Obrázek 3.5: Schéma načtení nového uživatele

3.1.6 Třída APP_WatchDog

Při testování studenty naprogramovaných her může docházet k různým chybám, jednou z nich je například zacyklení v nekonečné smyčce. Jako ochrana před touto situací se jeví použití nějaké formy časovače, který by hlídal, jak dlouho běží jedna smyčka hry. Takovýto přístup by bezpochyby fungoval, pokud by MATLAB umožňoval nějakou formu paralelních vláken. Nicméně standardní verze MATLABu nepodporuje paralelní programování a MATLAB objekt *Timer* by se tedy při zacyklení v nekonečné smyčce také zacyklil a nedal se použít.



Obrázek 3.6: Schéma funkčnosti třídy APP_WatchDog

Možností jak tento problém vyřešit bylo použití *Parallel Computing Toolboxu*, tedy sadě funkcí a tříd umožňující chod programu paralelně. Pomocí tohoto toolboxu lze využít paralelně běžícího procesu, který bude komunikovat s třídou APP_WatchDog. Pokud se program zacyklí v nekonečné smyčce tak přestane třída APP_WatchDog komunikovat s paralelně běžícím procesem, a po určité době, co je komunikace neaktivní zavolá paralelně běžící proces batch soubor, tedy sadu příkazů co se mají po sobě vykonat, který restartuje celou MATLAB aplikaci.

Paralelně běžící proces je vytvořen funkcí, která je spuštěna pomocí funkce *parfeval*, která byla popsána v sekci 2.2. Tato funkce očekává na zaslané data, a po každé přijaté zprávě funkce vynuluje počitadlo. Pokud se počitadlo nevynuluje a dosáhne určité hodnoty, tak funkce zavolá soubor *Restart.bat*.

3.1.7 Třída APP_Timers

Třída *APP_Timers* sdružuje všechny použité časovače, které se v aplikaci používají. Zároveň definuje funkce pro bezpečnou manipulaci s těmito časovači. Jednotlivé časovače, které jsou použity a jejich funkce jsou následující.

SteppingTimer je časovač, který ovládá krokování hry a student si může libovolně měnit periodu časovače, a tím tedy určovat tempo hry.

Jelikož se data z joysticku zasílají velmi rychle, byl by problém s interpretací těchto dat. Proto byla nutnost tyto data, respektive reakce na ně, uměle zpomalit. Pokud je překročena hranice, která označuje vychýlení joysticku, tak se spustí relevantní časovač *RepeatTimerX* nebo *RepeatTimerY*, který ovládá reagování aplikace, čím déle je joystick držen v jedné poloze, tím se zrychluje perioda časovače na maximální hodnotu.

WatchDogTimer určuje frekvenci zasílání dat do paralelně běžící funkce, která kontroluje chod aplikace, jak bylo popsáno v 3.1.6.

GamesTimer určuje frekvenci kontroly složky s hrami, pro případ importování nové hry.

Některé časovače se zapínají a vypínají velmi často, a mohlo by se stát, že by se stala chyba a snažil by se například vypnout časovač, který už je vypnutý. Kvůli tomuto byly definované funkce pro bezpečnou manipulaci, příklad této funkce pro zapnutí některých časovačů je zobrazen níže. Vstupem do této funkce je hodnota typu *enum*, která odpovídá časovači, který chceme zapnout. Poté funkce zkontroluje stav časovače a zapne ho pouze pokud se nachází časovač ve vypnutém stavu. Obdobně toto je definované pro vypnutí časovačů.

```
this.Timers.startMyTimer(Enums.GamesTimerE);
```

```
        function startMyTimer(this, index)
            switch(index)
                case Enums.GamesTimerE
                    if(strcmp(this.GamesTimer.Running, 'off'))
                        start(this.GamesTimer);
                    end
                case Enums.SteppingTimerE
                    if(strcmp(this.SteppingTimer.Running, 'off'))
                        start(this.SteppingTimer);
                    end
            end
        end
```

3.2 Frontend aplikace

Jak již bylo výše zmíněno, je aplikace rozdělena na dvě části, backend aplikace byl již popsán, frontend aplikace bude popsán v této sekci. Frontend aplikace byl naprogramován pomocí programovacího jazyku HTML a Javascript. Tato možnost byla zvolena, protože HTML umožňuje výrazně větší modifikaci a vzhled aplikace poté může vypadat mnohem lépe. MATLAB standardně podporuje vložení HTML objektů, a tedy to výrazně ulehčilo implementaci této možnosti.

Nejprve se naprogramuje zdrojový kód v HTML, který obsahuje objekt, který chceme vložit do MATLABu. V MATLABu se poté vytvoří objekt *uihtml*, se kterým se pracuje stejně jako s normálním prvkem MATLABu, které se používají pro tvorbu uživatelských aplikací, například *uipanel*. Vstupem do tohoto objektu je cesta ke zdrojovému kódu obsahují HTML objekt. Zároveň takto naprogramovaný objekt v HTML dokáže jednoduše komunikovat s MATLAB aplikací.

Byly vytvořeny celkem čtyři HTML soubory. *Index.html*, který obsahuje zdrojový kód pro vzhled hlavního menu. *Loading.html* obsahuje načítací obrazovku. *Leaderboard.html* obsahuje tabulku vítězů a *NewUser.html* obsahuje vzhled, který se zobrazí při vytváření nového uživatelského profilu.

3.2.1 Hlavní menu

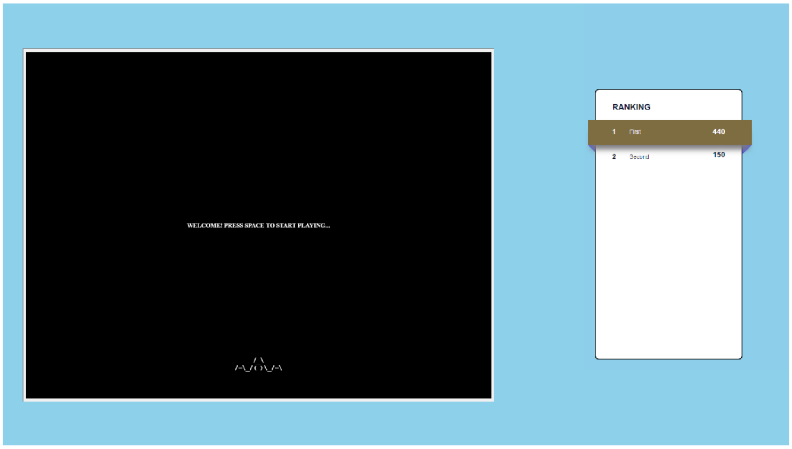
Hlavní menu aplikace se skládá z několika objektů. Ve středu se nachází tabulka s výběrem hry a prostor pro zobrazení loga hry. Důležitou částí je konzole, která slouží primárně pro zobrazování chybových hlášek, popřípadě zobrazuje jméno aktivního uživatelského profilu. Tato konzole se nachází v levém spodním rohu. V pravém rohu je poté prostor pro zobrazení QR kódu, který by měl odkazovat na GitHub autora hry, a také jméno autora. Samotný vzhled lze vidět na obrázku 3.7.

Obrázek 3.7: Hlavní menu

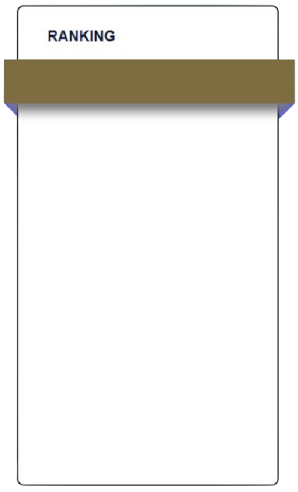
3.2.2 Leaderboard

Tabulka vítězů, která zobrazuje nejvyšší skóre, je taktéž naprogramovaná pomocí HTML jako samostatný objekt. Je součástí panelu *Panel_Game*, popsaného v sekci 3.1.2.

Při každém spuštění hry se načtou v části aplikace naprogramované v MATLABu, data z tabulky udržované pomocí textového dokumentu v adresáři hry, a tyto data se z MATLABu zašlou do HTML kódu tabulky. Zde se poté přepíší data a jména podle aktuálních dat. Vzhled samotné tabulky lze vidět na obrázku 3.9. Zakomponovaná tabulka do panelu je zobrazena na obrázku 3.8.



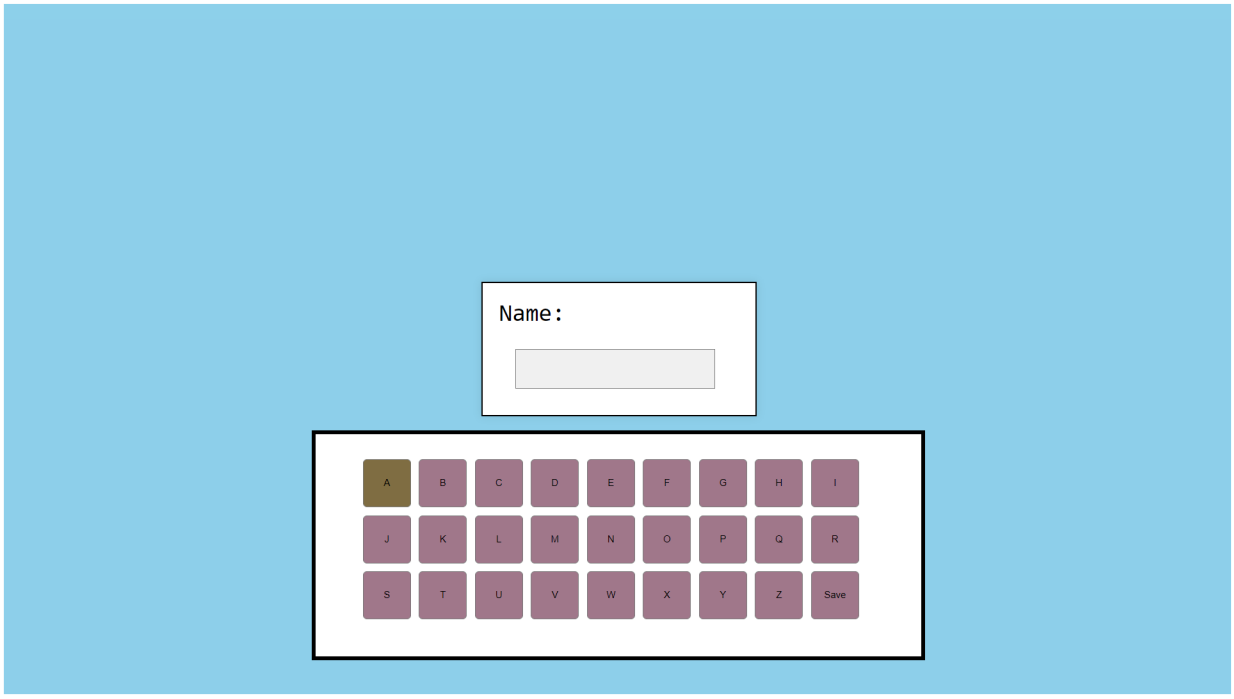
Obrázek 3.8: Zobrazení panelu s hrou



Obrázek 3.9: Tabulka

3.2.3 Tvorba nového uživatele

Při načítaní ISIC karty je uživatel nucen vytvořit si uživatelský profil, pokud se přihlašuje poprvé. V této situaci se přepne aplikace na panel vytvoření nového profilu, jak lze vidět na obrázku 3.10. Tento panel se skládá z textového pole a z virtuální klávesnice. V HTML je naprogramovaná pouze část tohoto panelu, konkrétně pozadí a pole pro textové pole. Virtuální klávesnice a textové pole je vytvořeno v MATLAB kvůli jednoduchosti.



Obrázek 3.10: Panel pro vytváření nového uživatelského účtu

3.2.4 Komunikace mezi MATLABEM a HTML

Často bývá potřeba komunikace mezi HTML zdrojovým kódem a kódem v MATLABu, i pro tento případ jsou v MATLABu implementované jednoduché funkce pro komunikaci. Při vytváření objektu *uihtml* v MATLABu se definuje i funkce *HTMLEventReceivedFcn*, která se zavolá pokud přijdou nějaká data z HTML strany, který je definovaný jako vstup do objektu *uihtml*. V této funkci se data dále zpracují podle takzvaného event názvu.

Do HTML se data dají poslat pomocí funkce *sendEventToHTMLSource*, do které vstupy jsou objekt *uihtml*, se kterým je komunikace žádaná, dále název eventu, který identifikuje typ zaslaných zpráv, a popřípadě další vstup můžou být i samotná data, které ovšem nejsou povinná. Data je ideální zasílat pomocí datového typu *JSON*. Toto ovšem není standardní datový typ MATLABu, ovšem MATLAB obsahuje funkci pro jednoduché převedení jiných datových typů, například vektoru, na typ JSON, tato funkce je *jsonencode*.

Ve zdrojovém kódu HTML, respektive pomocí programovacího jazyku Javascript, který se stará o logickou část HTML kódu, se vytvoří takzvaný *event listener*, který

čeká na příchozí data ze strany MATLABu a poté zavolá funkci podle jednotlivých názvů eventů. V těchto funkcích se poté data zpracují. Pro zaslaní dat zpět do MATLABu lze použít funkce *sendEventToMATLAB*.

V této práci je hlavní komunikace mezi třídou UI a zdrojovým kódem hlavního menu, index.html. Komunikace probíhá pomocí několika eventů. Event *ValueChanged* je zaslán z MATLABu, pokud se změní počet načtených her, a je potřeba přidat nebo odebrat název hry z nabídky. Event *JoystickData* se zasílá pokaždé, když uživatel hýbe s joystickem a nachází se v hlavním menu, tak aby se mohl v něm pohybovat, v HTML kódu se poté upravuje vzhled na základě poslaných dat, kde se zasílají data ve formě dvou indexů, řádek a sloupec. Event *ConsoleMessage* zasílá do HTML kódu zprávu, kterou je potřeba zobrazit v konzoly. A poslední event *AuthorChanged*, zasílá jméno autora hry, který je potřeba přepsat.

Zdrojový kód HTML komunikuje zpět pouze na začátku komunikace, kdy zašle data pozice a velikosti rámu připraveného pro loga her, tak aby MATLAB mohl automaticky nastavit pozici, kam umístit obrázek loga a QR kódu při změně monitoru. Také pokud se změní vybraná hra, HTML zašle MATLABu event s názvem hry, aby mohl MATLAB změnit logo.

3.3 API pro tvorbu vlastní hry

Třída *APP_UI* obsahuje také sadu veřejných vlastností a funkcí, ke kterým má student při tvorbě jeho hry přístup, a které umožňují úpravy této třídy. Pro tyto funkce a vlastnosti vznikl v rámci této bakalářské práce i dokument popisující jejich funkce více do detailu, zde jsou pouze lehce popsané.

Mezi veřejné vlastnosti patří *Axes*, jedná se o objekt typu *uiaxes*, ve kterém se bude vykreslovat hra. Student může libovolně upravovat tento objekt, tak aby fungoval správně pro jeho hru.

Další vlastností je *AxesPanelSize*, což je vektor, který obsahuje hodnoty velikosti objektu *uipanel*, ve kterém se nachází objekt *Axes*. Slouží pro vyčtení šířky a výšky, a vymezuje tak maximální rozměr, kterého může objekt Axes nabývat.

Veřejné funkce jsou následující.

```
function saveScore(Score)
```

Tato funkce umožňuje uložení skóre k aktivnímu účtu. Pokud není načtený žádný účet, tato funkce nic neudělá.

```
function backToMainMenu()
```

Funkce slouží pro návrat do hlavního menu

```
function setTimerFreq(timerFreq)
```

Funkce slouží pro nastavení periody časovače, která ovládá krokování hry.

```
function enableButtonsIRQ(enableArr)
```

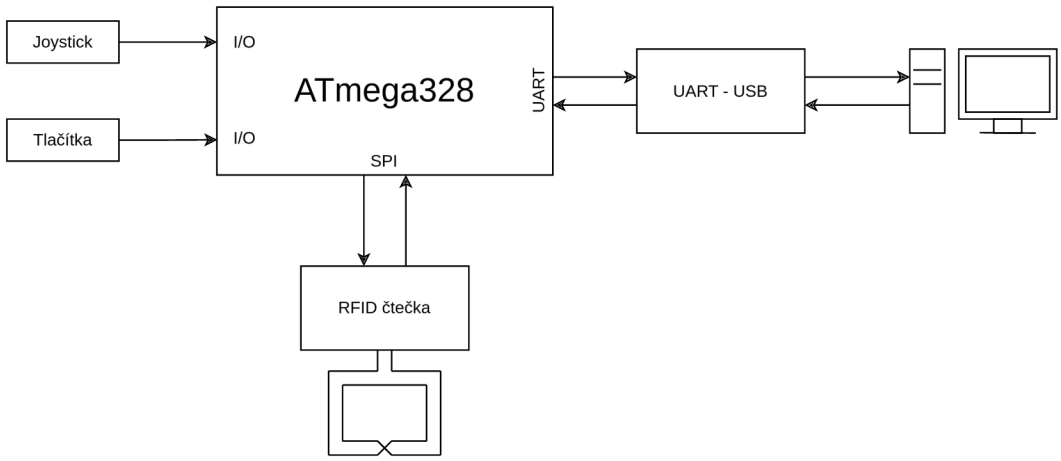
Funkce slouží pro specifikování, kterých uživatelských vstupů, respektive Up, Down, Left, Rigth, Enter, Exit bude hra využívat. Pro povolené vstupy je nutné definovat odpovídající funkce ve hře

```
function sendJoystickValue()
```

Tato funkce slouží pro přepnutí interpretace dat, standardně se data z joysticku zpracovávají v třídě APP_UI. Pokud se zavolá tato funkce, budou do hry zasílány přímo hodnoty z joysticku a student si je bude moct zpracovat sám.

3.4 Deska plošných spojů

Důležitou částí této bakalářské práce bylo vytvoření desky plošných spojů, jejíž úkol by měl primárně být sběr dat z joysticku a tlačítek, a také komunikace s RFID čtečkou a následně uživatelskou aplikací. Schéma funkčnosti desky je zobrazeno na obrázku 3.11



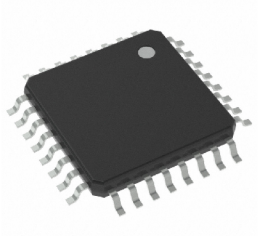
Obrázek 3.11: Schéma desky plošných spojů

3.4.1 Výběr komponentů

Prvním úkolem byl výběr komponentů, které budou tvořit desku plošných spojů. Na mikrokontroler, jakožto hlavní bod desky, nebyl kladen příliš velký nárok. Jedná se o poměrně jednoduché úkony, a proto byl zvolen mikrokontroler ATmega328, obrázek 3.12, který je součástí populárních vývojových desek Arduino, na kterých byl vyvíjen prototyp desky, bylo tedy logické zvolit tuto variantu.

Dalším výběrem byl USB-UART převodník, zde byl zvolen převodník FT232RL od firmy FTDI, jelikož podporuje virtuální COM porty, a tedy by neměl být problém s drivery na počítači při komunikaci.

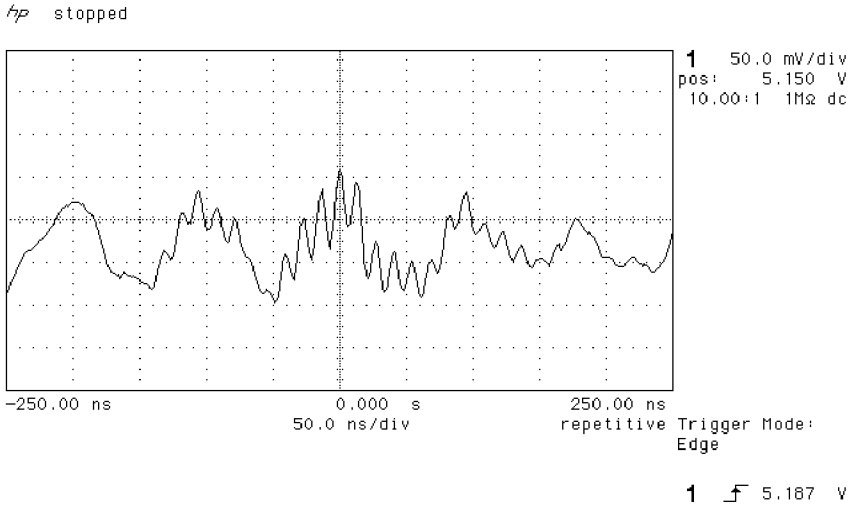
Důležitou volbou byla RFID čtečka, zde byl zvolen RFID modul RC522, založený na integrovaném obvodu MFRC522 od firmy NXP. Pro tuto čtečku je potřeba přidat na desku 3,3V regulátor. Pro tuto volbu byl zvolen low dropout regulátor MIC5504.



Obrázek 3.12: Mikrokontroler ATmega328P-AU [20]

3.4.2 Tvorba desky

Samotný návrh desky plošných spojů byl vytvořen pomocí programu Altium. Část schématu obvodu desku lze vidět na obrázku 3.14, která obsahuje obvod mikrokontroleru a převodníku. Napětí je dodáváno desce pomocí USB kabelu z počítače. Ovšem toto napětí bývá velmi zašuměné, obrázek 3.13 ukazuje měření pomocí osciloskopu, kde lze vidět šum USB portu z počítače. K filtrování tohoto napětí je vhodné použití feritových perliček a kondenzátorů, tedy ve své podstatě LC filtru. Hodnoty tohoto filtru byly zjištěny z datasheetu převodníku FT232RL. [21] Zároveň je vhodné použití ESD ochrany USB linky.



Obrázek 3.13: Šum USB portu [22]

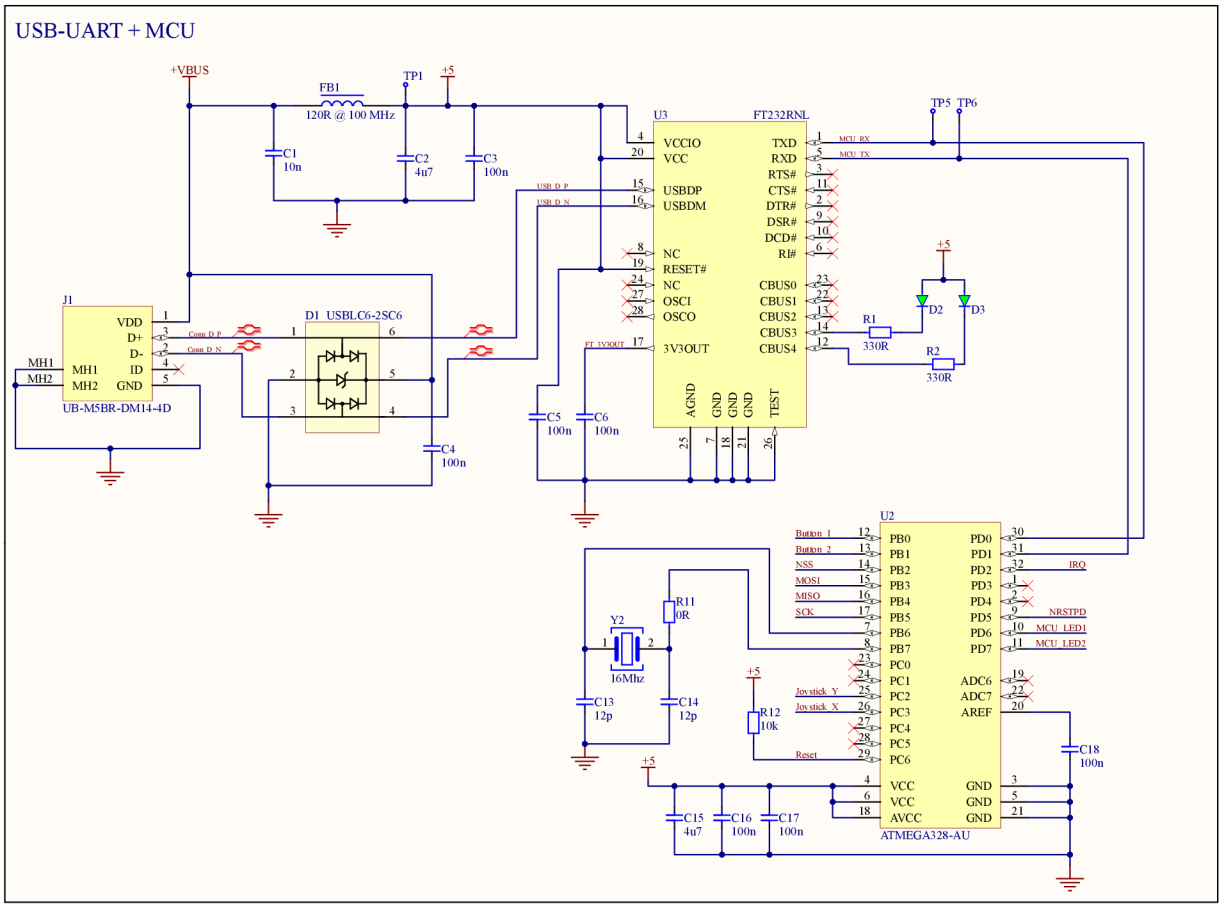
Integrovaný obvod FT232RL je zapojený podle datasheetu [21] do zapojení USB bus power configuration, tedy zapojení, kdy převodník pracuje na napětí přivedeném pomocí USB linky. Diody D2 a D3 indikují stav komunikace integrovaného obvodu, tedy stavy transfer a receive.

U zapojení mikrokontroleru je důležité zapojení takzvaných *decoupling* kondenzátorů, které slouží jako přímý dodavatel energie pro mikrokontroler v případě potřeby a udržují konstantní napětí, které mikrokontroler potřebuje. Dále je k mikrokontroleru připojen externí 16 MHz krystal, který ovládá časování mikrokontroleru. ATmega328 sice obsahuje interní krystal, nicméně tyto interní krystaly bývají velmi nepřesné, oproti externím.

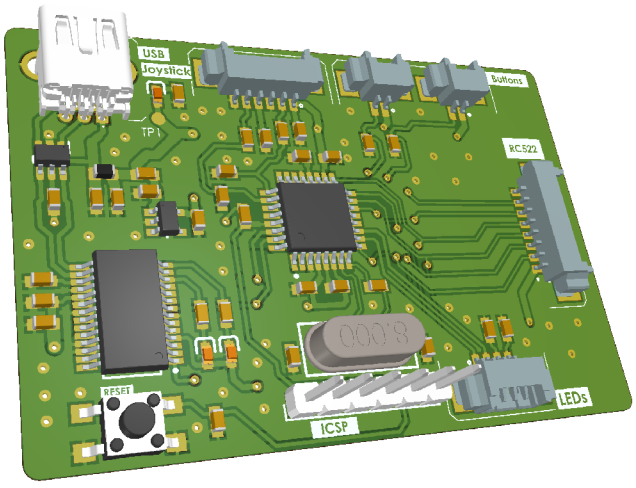
Dále je potřeba myslet na to, jakým způsobem se bude mikrokontroler programovat. Standardní způsob programování mikrokontrolerů ATmega328 je pomocí ICSP, In-Circuit Serial Programming. [23]

Dále je potřeba filtrovat data z joysticku pomocí jednoduchého RC filtru. Poslední částí desky je 3,3V regulátor a konektor pro připojení RFID čtečky.

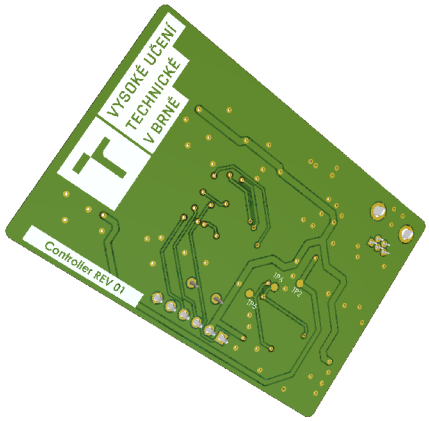
Finální deska je zobrazena na obrázcích 3.15 a 3.16, kde lze vidět 3D model desky.



Obrázek 3.14: Schéma části obvodu mikrokontroleru a převodníku



Obrázek 3.15: Přední strana desky



Obrázek 3.16: Zadní strana desky

3.5 Program mikrokontroleru

Program mikrokontroleru byl napsán pomocí programu *PlatformIO* v jazyce C. Program je rozdělen do několika souborů, kdy každý soubor sdružuje funkce a datové typy pro jednotlivé použité periferie. Tyto jednotlivé soubory budou popsané v následujících sekcích.

3.5.1 GPIO

Jedná se o soubor zdrojového kódu *gpio.c* a hlavičkového souboru *gpio.h*. Soubor sdružuje funkce pro nastavení a ovládání GPIO pinů. Každý GPIO pin je nastaven pomocí datového typu struct *gpio_pin*, který obsahuje číslo pinu a dále je nutné specifikovat port, port bývá označen písmenem, kvůli tomuto byl vytvořen datový typ enum, který definuje jednotlivé porty. Tato hodnota je také součástí datového typu *gpio_pin*. Jeho definice je následující.

```
typedef struct{
    int pin;
    port_enum port;
}gpio_pin;
```

Dále je potřeba vytvořit funkce pro nastavení módu pinu, tedy jestli se jedná o INPUT nebo OUTPUT pin. Toho je dosaženo pomocí registrů *DDRx*, kde x označuje, o který port se jedná. [23] Vstupem do této funkce je poté pointer, tedy ukazatel. Tento ukazatel ukazuje na pin, který má být ovládán, tedy na *gpio_pin*, a dále jestli se pin nastaví jako OUTPUT nebo INPUT.

```
    void gpio_set_mode(const gpio_pin *pin, mode_enum mode)
    {
        switch(pin->port){
            case port_enum::B:
                DDRB |= (mode << pin->pin);
                break;
            case port_enum::C:
                DDRC |= (mode << pin->pin);
                break;
            case port_enum::D:
                DDRD |= (mode << pin->pin);
                break;
        }
    }
```

Dalšími funkcemi jsou nastavení hodnoty pinu, které jsou definované jako OUTPUT, a přečtení hodnoty pinů, které jsou definované jako INPUT. Funkce jsou vytvořeny obdobně jako funkce pro nastavení módu pinu. Zde lze využít registrů *PORTx*, respektive *PINx*. A pomocí bitových operací provedeme nastavení nebo přečtení hodnoty. Příklad nastavení hodnoty pinu B2 na logickou jedničku lze vidět pomocí rovnice 3.1. A příklad pro přečtení stejného pinu lze vidět pomocí rovnice 3.2.

$$\text{PORTB} \mid = \ (1 << 2); \tag{3.1}$$

$$\text{val} \ = \text{PINB\&}(1 << 2); \tag{3.2}$$

3.5.2 ADC

Soubory *adc.c* a *adc.h* nastavují použití A/D převodníku, který slouží primárně pro čtení analogových hodnot joysticku, a definují funkci pro přečtení hodnoty na analogovém pinu. ADC se nastavuje tak, že reference napětí, vůči kterému porovnává převodník čtené napětí, je hodnota napětí na pinu AREF, ten je v tomto případě přímo spojen s napětím mikrokontroleru, tedy 5V, kvůli stabilizaci napětí a lepším výsledkům konverze je k pinu AREF připojen externí kondenzátor. Dále se nastavuje prescaler, který ovládá dobu čtení hodnoty.

Funkce pro přečtení analogové hodnoty funguje tak, že se nastaví bit registru *ADCSRA*, který zahájí konverzi. Dále program čeká na změnu tohoto bitu, která indikuje dokončení konverze. Jelikož by se hodnota nevešla do 8-bitového registru je přečtená hodnota rozdělena na horní a dolní bajt a tyto bajty jsou vyčteny z příslušných registrů.

3.5.3 UART

Soubory *uart.c* a *uart.h* nastavují použití UART periferie a definují funkce pro přijímání a odesílání dat. Nejdříve se nastavuje hodnota BAUDRATE na 9600, to je standardní využívaná hodnota.

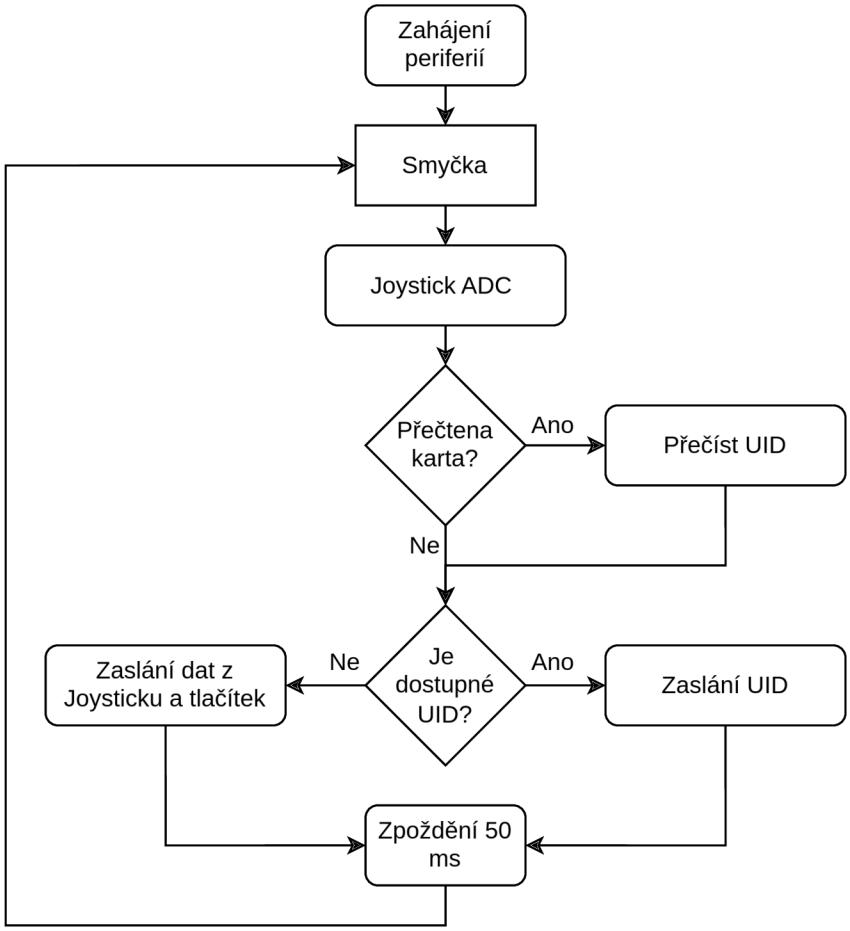
Pro zasílání dat jsou definované dvě funkce, do první funkce vstupuje řetězec charakterů, které jsou určené pro odeslání, tato funkce následně prochází celý řetězec a pro každý jednotlivý charakter, což odpovídá 8 bitům, zavolá druhou funkci, která zašle tento jeden charakter.

Přijímání dat není pro tuto práci potřebné, nicméně UART byl nastaven tak, aby při přijmutí jednoho charakteru zaznamenal interrupt, neboli přerušení, v tomto přerušení se následně tento charakter uloží do pole. Po příchodu konečného charakteru indikujícího konec zaslané zprávy přerušení nastaví k tomu určenou proměnnou do stavu indikující příjem zprávy, ta se poté může dále zpracovat v hlavní smyčce programu.

3.5.4 Main

Soubor *main.c* obsahuje hlavní smyčku programu, taktéž jako inicializaci ostatních struktur a souborů popsaných dříve v této sekci.

Hlavní smyčka se skládá primárně z čtení analogových hodnot joysticku a z čtení přítomnosti ISIC karty. Stav tlačítek se zjišťuje v přerušení, kvůli takzvanému *debouncingu*, tedy odskoku. To je stav, kdy mechanický prvek, například tlačítko, při sepnutí odskočí od kontaktu, a tak způsobí vícenásobné nechtěné sepnutí. Z tohoto důvodu je stav tlačítek vyčítán každých zhruba 1 ms v přerušení a jeho hodnota uložena, pokud je stav tlačítek sepnutý delší dobu indikuje to náběžnou hranu sepnutí tlačítka, které už je s největší pravděpodobností opravdu sepnuté. Každý běh smyčky je vyslán příkaz pro navázání komunikace s ISIC kartou a pouze, pokud je nějaká v blízkosti, tak je přečteno a uložené UID karty, blíže popsané v sekci 3.6. Celé schéma programu lze vidět na obrázku 3.17.



Obrázek 3.17: Schéma main.c

3.6 RFID protokol

Pro použití RFID čtečky bylo potřeba vytvořit knihovnu, respektive upravit dostupnou knihovnu, pro moje použití. Knihovna má na starost komunikaci s integrovaným obvodem MFRC522, který je součástí RFID čtečky RC522, použité v mé práci. Dále má knihovna na starost inicializaci komunikace s ISIC kartou a přečtení UID této karty.

Pro tento populární model existuje spousta knihoven splňující tento úkol, nicméně tyto knihovny je potřeba upravit tak, aby fungovali s mým programem. Konkrétně v mé práci se vycházelo z knihovny MFRC522 [24]. Tato knihovna je vytvořena pomocí OOP a je poměrně složitá. Pro moji práci byla převzata kostra knihovny, konkrétně komunikace s integrovaným obvodem MFRC522, která byla upravena, tak aby nebylo využito objektů.

Hlavní strukturou knihovny je struktura *MFRC_t*, součástí které jsou dva GPIO piny definované pomocí struktury *gpio_pin* popsané v sekci 3.5.1. Tyto piny ovládají resetování integrovaného obvodu a výběr obvodu pro komunikaci. Dále struktura obsahuje pole *UID*, které slouží pro uložení identifikačního čísla ISIC karty, hodnotu *size*, která určuje, jak je dané UID dlouhé, standard definuje délku 4, 7 nebo 10 bajtů. Původně byla definovaná pouze velikost 4 bajty, nicméně kvůli tomu jak je RFID populární by už nemohla být zaručena unikátnost identifikátoru, a proto se velikosti rozšířily. [25] Poslední hodnota, kterou struktura obsahuje je *sak*, neboli select acknowledgement, tato hodnota je zaslána po úspěšném navázání komunikace.

```
typedef struct{

    const gpio_pin CE;
    const gpio_pin RST;

    uint8_t Uid[10];
    uint8_t size;
    uint8_t sak;

}MFRC_t;
```

Integrovaný obvod MFRC522 funguje na základě stavového automatu [26], a může se nacházet v několika stavech, které určují jakou funkci obvod vykonává, tyto stavy lze vidět na obrázku 3.18. Nejdůležitější je stav *transceive*. V tomto stavu obvod zasílá data zapsaná do FIFO fronty obvodu a přijímá data, která následně také ukládá do FIFO fronty. Pokud tedy chceme odeslat nějaká data přes anténu, je potřeba nejdříve uvést integrovaný obvod do stavu *Idle*, následně naplnit FIFO frontu obvodu daty, která chceme odeslat, poté uvést obvod do stavu *transceive* a pomocí registru k tomu určeného zahájit odeslání dat. MFRC522 následně odešle data a zahájí příjem, zároveň umožňuje nastavení přerušení, které indikuje dokončený příjem. Po tomto je možné vybrat data z FIFO fronty, do které obvod data uložil.

Command	Command code	Action
Idle	0000	no action, cancels current command execution
Mem	0001	stores 25 bytes into the internal buffer
Generate RandomID	0010	generates a 10-byte random ID number
CalcCRC	0011	activates the CRC coprocessor or performs a self test
Transmit	0100	transmits data from the FIFO buffer
NoCmdChange	0111	no command change, can be used to modify the CommandReg register bits without affecting the command, for example, the PowerDown bit
Receive	1000	activates the receiver circuits
Transceive	1100	transmits data from FIFO buffer to antenna and automatically activates the receiver after transmission
-	1101	reserved for future use
MFAuthent	1110	performs the MIFARE standard authentication as a reader
SoftReset	1111	resets the MFRC522

Obrázek 3.18: Stavy MFRC522 [26]

3.6.1 Zahájení komunikace

Důležité z hlediska RFID bylo zahájení komunikace mezi čtečkou, dále PCD (proximity coupling device), a transceiverem, dále PICC (proximity integrated circuit card). Tato komunikace je popsána standardem ISO 14443, konkrétně jak probíhá zahájení komunikace popisuje subsekce tohoto standardu ISO 14443-3, která se ještě dále děli na typ A a typ B. V této práci byl použit typ A. [27]

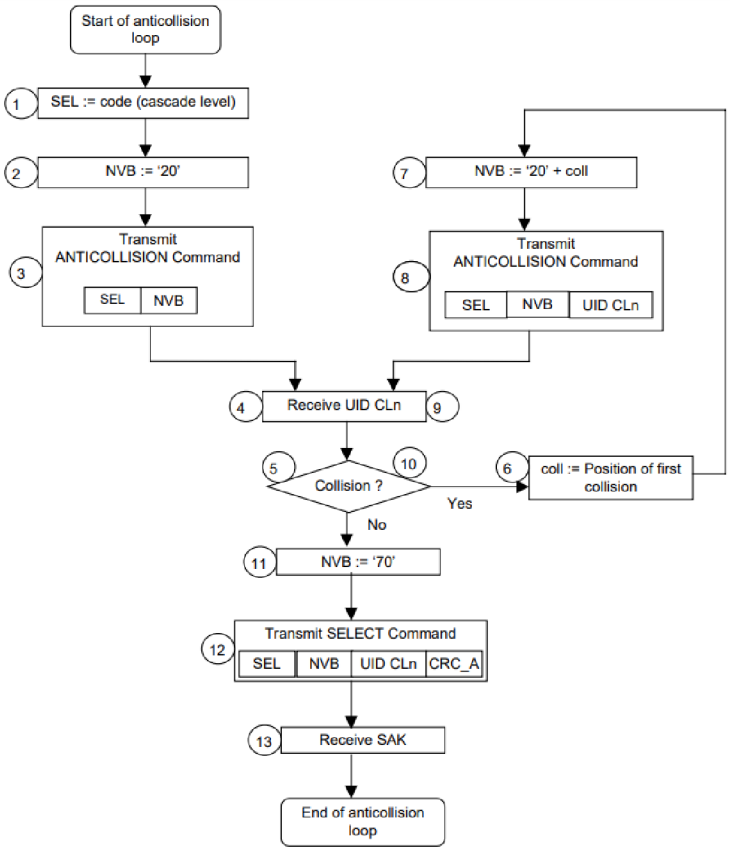
Pro PICC je definováno několik stavů, ve kterých se může nacházet, a to jsou *IDLE*, *READY*, *ACTIVE* a *HALT*. Samotná inicializace komunikace je rozdělena na dvě části *REQUEST* a *ANTICOLLISION* smyčka. PCD nejprve vysílá opětovně příkaz REQUEST, pokud PICC zachytí tento příkaz odpoví na něj příkazem ATQA, Answer to request type A. Tento příkaz obsahuje informace o PICC, konkrétně délku UID. Po odeslání tohoto příkazu přechází všechny PICC v okolí do stavu READY. Dále následuje ANTICOLLISION smyčka, jejíž průběh je definován standardem, schéma lze vidět na obrázku 3.19. PCD a PICC si mezi sebou během ANTICOLLISION smyčky zasílají buffer, jehož struktura je na obrázku 3.20.

PCD zašle na začátku smyčky dva bajty složené z SEL, Select příkaz, který označuje stupeň ANTICOLLISION smyčky pro získání UID delšího 4 bajtů a NVB, Number of valid bits, tento bajt určuje počet zaslaných bajtů a bitů. Vrchní 4 bity tohoto bajtu určují počet zaslaných bajtů a spodní 4 bity určují počet zbylých bitů. Minimální hodnota NVB v hexadecimální soustavě je 0x20, maximální poté 0x70, což určuje počet zaslání celého buferu.

Na tento příkaz odpoví všechny PICC v okolí svým UID. Pokud je přijato více než jedno UID jedná se o kolizi. PCD následně určí pozici první kolizi a zvolí jeden z možných PICC, kterému pošle zprávu složenou z SEL, NVB a části UID před kolizí odpovídající PICC, se kterým chce PCD komunikovat.

Na tento příkaz následně odpoví pouze relevantní PICC zasláním zbytku svého UID. Pro navázání komunikace zašle PCD příkaz, kdy definuje NVB jako 0x70, indikující zaslání celého buferu. PICC odpoví příkazem SAK, ze kterého lze poznat, jestli je UID kompletní nebo ne. Pokud není UID kompletní, ANTICOLLISION smyčka se opakuje pouze s jiným kaskádovým bajtem SEL. Pokud je UID kompletní PICC přechází do stavu ACTIVE a může začít komunikace mezi PCD a PICC.

Komunikace s ISIC kartou je ale šifrovaná a není možné s ní tedy komunikovat, to ovšem nebylo potřeba, jelikož pro tuto práci stačilo vyčíst z karty její UID.



Obrázek 3.19: Anticollision smyčka [27]

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
SEL	NVB	UID0	UID1	UID2	UID3	BCC/SAK	CRCA	CRCB

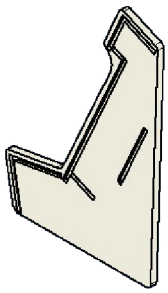
Obrázek 3.20: Buffer

3.7 Fyzické zařízení

Pro tvorbu celého zařízení byl vybrán jako vhodný materiál dřevo, konkrétně MDF desky. Tyto desky se snadno obrábí a zároveň není potřeba žádné finální povrchové úpravy. Desky byly obrobeny pomocí CNC frézky, a aby mohly být obrobeny bylo potřeba vymodelovat 3D model.

3.7.1 Model

Model zařízení byl vymodelován pomocí programu Inventor. Skládá se z několika desek, které se do sebe jednoduše zasunou. Boční strany modelu, jedna z nich je zobrazena na obrázku 3.21, obsahují drážky, do kterých zapadají osazení na prostředních deskách. Celkový model je zobrazen na obrázku 3.23. Pomocí tohoto modelu poté byly vygenerovány data pro CNC frézku. Obrázek 3.22 ukazuje již fungující a sestavené fyzické zařízení. Model byl vytvořen tak, aby se do něho dal zasunout počítačový monitor, na kterém se bude zobrazovat uživatelská aplikace.



Obrázek 3.21: Boční strana modelu

Obrázek 3.22: Fyzické zařízení z přední strany

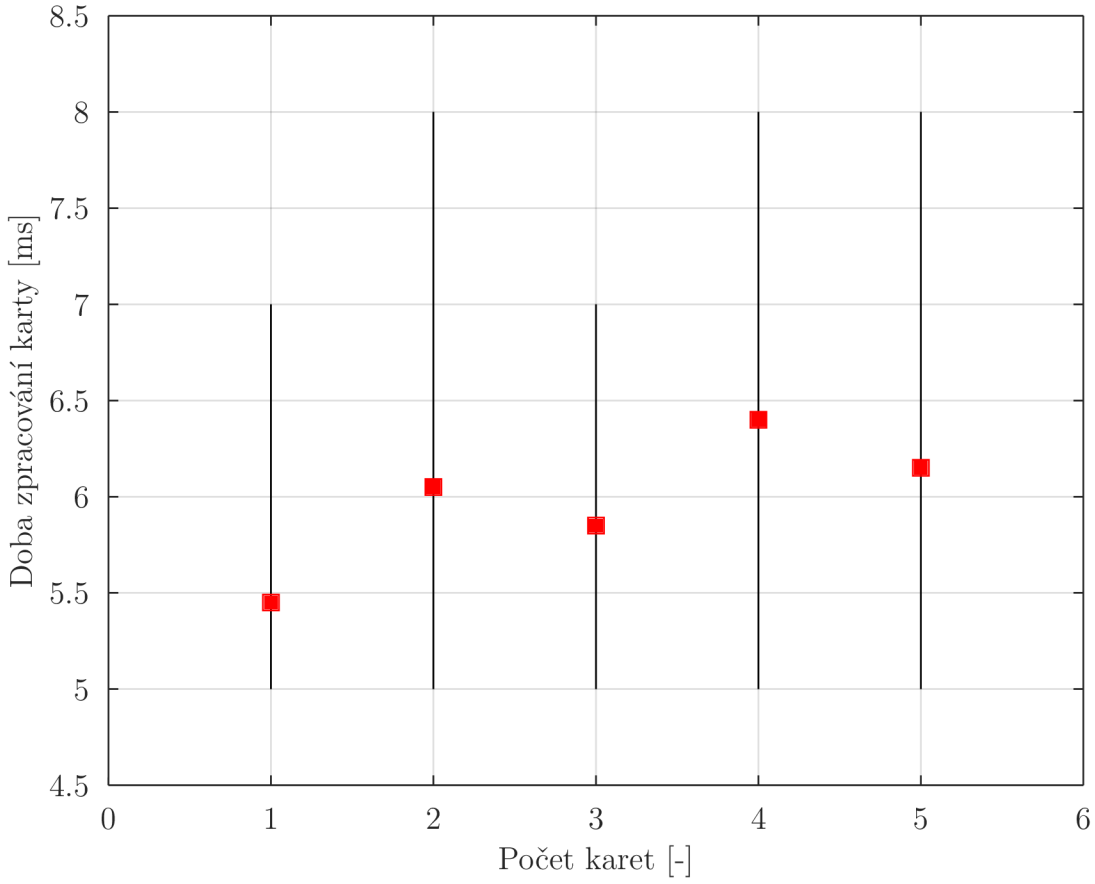
Obrázek 3.23: Model zařízení

3.8 Testování a validace zařízení

Pro ověření správné funkčnosti zařízení bylo potřeba provést testování. Zařízení bylo primárně testováno studenty, kteří s tímto zařízením strávili nějaký čas, a poté poskytly cenné poznatky pro úpravu této práce.

3.8.1 RFID testování

Nejdříve bylo ale pro zajištění správné funkčnosti testována RFID čtečka. Primárním cílem tohoto testu bylo zjištění funkcionality *Anticollision* funkce, popsané v sekci 3.6. Tedy funkce čtečky reagovat, pokud je v její blízkosti více karet, které se snaží komunikovat. Byla provedena série opakovaných měření, kde se postupně přikládal větší počet ISIC karet do blízkosti čtečky a sledovala odezva systému. Čas se měřil v programu mikrokontroleru před přečtením karty a po přečtení ID karty, která byla vybrána. Měření je znázorněno na obrázku 3.24 a je z něj patrné, že větší počet karet mírně prodlužuje průměrnou dobu potřebnou k přečtení ID jedné z karet a ověřilo, že čtečka funguje správně i pro větší počet ISIC karet v blízkosti.



Obrázek 3.24: Test doby zpracování žádosti o komunikaci pro různý počet ISIC karet

3.8.2 Poznatky studentů

Z poznatků, které měli studenti k této práci vyvstalo několik změn, které byly provedeny pro zlepšení práce. Mezi ně patřilo.

Neintuitivnost ovládání při prvním použití, kdy uživatelé nevěděli, jakým způsobem, a kterými tlačítky se v menu navigovat a vyžadovalo to nějakou dobu, než přišli na správné ovládání. Pro zlepšení tohoto problému byly přidány do konzole vysvětlivky, které se zobrazí vždy při návratu do hlavního menu. Tyto vysvětlivky popisují jakým způsobem v menu navigovat. Dále byl upraven vzhled tlačítek v uživatelské aplikaci, tak aby barvy jednotlivých tlačítek v aplikaci odpovídaly těm fyzickým, umístěným na těle zařízení.

Další poznatek byl ohledně uživatelských profilů, kdy v aplikaci není důrazně viditelný aktivní uživatelský profil, a uživatelé si poté někdy nejsou jisti, který je zrovna aktivní. Pro vyřešení tohoto problému byla do aplikace přidána kolonka, která je viditelná v hlavním menu, která zobrazuje jméno aktivního uživatelského profilu. Také byl přidán vysvětlující text do konzole při spuštění, který vysvětluje funkčnost profilu, a tedy to, že skóre v jednotlivých hrách se ukládá pouze pokud je aktivní nějaký profil. Provedené změny v souvislosti s hlavním menu aplikace jsou zobrazeny na obrázku 3.25.

V souvislosti s uživatelskými profily byl další poznatek ohledně jejich tvorby. Kdy při tvorbě uživatelských profilů chyběla funkcionalita umazaní písmena ve virtuální klávesnici. Tento problém byl vyřešen přidáním dalšího tlačítka, fungujícího jako *zpět*, které umaže poslední charakter uložený v textovém poli jména.

Obrázek 3.25: Upravené hlavní menu aplikace podle poznatků

4 Závěr

Začátek práce se věnuje rozšíření znalostí ohledně důležitých témat, které byly relevantní pro tvorbu této práce. A tvoří tak teoretický úvod této práce. Následně byly popsány jednotlivé části práce, které byly vytvořeny. Konkrétně se jedná o uživatelskou aplikaci, desku plošných spojů a v poslední řadě samotné fyzické zařízení.

První částí práce byla tvorba uživatelské aplikace. Ta byla rozdělena do dvou částí, frontendu, který se zabývá vzhledem aplikace, programovaným v jazyce HTML, CSS a Javascript. A dále backendu, který je naprogramovaný v jazyce MATLAB a zabývá se logickou částí aplikace a komunikací s deskou plošných spojů a studenty naprogramovanou hrou.

V kapitole 3.1 je popsána struktura, která byla navržena pro backend uživatelské aplikace. Tato struktura byla rozdělena do jednotlivých funkčních bloků kvůli přehlednosti. Kdy každý blok zastává část aplikace a sdružuje vlastnosti a funkce jednotlivých logických bloků.

V kapitole 3.2 je popsána tvorba vzhledu uživatelské aplikace a způsob komunikace mezi zdrojovým kódem frontendu a backendu.

Uživatelská aplikace umožňuje studentům vložit jejich naprogramovanou hru. Jako součást práce vznikl také manuál, obsažený v příloze A, který by měl studentům pomoci v programování jejich her. Manuál obsahuje popis veřejných vlastností a funkcí, které může student využívat pro tvorbu, dále popisuje strukturu adresáře hry a strukturu samotné hry. V poslední řadě obsahuje manuál také příklad, který ukazuje naprogramovanou jednoduchou hru.

Importování her probíhá uložením adresáře hry na sdílený disk, ze kterého poté aplikace načítá data, jak bylo popsáno v kapitole 3.1.4.

Důležitou vlastností uživatelské aplikace je identifikace uživatelů a tvorba uživatelských účtů. Pomocí studentských ISIC karet, které fungují na principu RFID, 2.4.1, jsou studenti schopni přihlásit se ke svým účtům přiložením karty. Při prvním přiložení karty si student vytvoří uživatelský účet, který je přiřazen k ID číslu ISIC karty a uložen do databáze, tento proces byl popsán v kapitole 3.1.5.

K jednotlivým uživatelským účtům se poté ukládají skóre samotných her, tak že pro každou hru vytvoří aplikace textový soubor udržující tabulku. Aplikace poté tuto tabulku při spuštění hry načítá a zobrazuje uživateli.

Dále je v kapitole 3.4 popsán návrh a tvorba desky plošných spojů, která má za úkol sběr dat z tlačítek a joysticku, komunikaci s RFID čtečkou a komunikaci s uživatelskou aplikací. Program mikrokontroleru, který je rozdělen do jednotlivých souborů, každý zastávající chod jednotlivých periferií, je popsán v kapitole 3.5. Důležitou částí programu mikrokontroleru byla komunikace s RFID čtečkou, která je popsána v kapitole 3.6.

Zařízení bylo testováno nezávisle na sobě studenty, kteří poskytli zpětnou vazbu, díky které se následně upravila primárně aplikace, tak aby její ovládání bylo intuitivnější a lépe pochopitelné pro uživatele, který zařízení ovládá poprvé. Zároveň byla testována funkčnost

RFID čtečky při přiložení více ISIC karet. Kde se prokázalo, že při detekci více ISIC karet v blízkosti čtečky se mírně prodlouží průměrná doba čtení karty. Tímto způsobem byla otestována také funkčnost čtečky a její ANTICOLLISION smyčky, popsané v kapitole 3.6.

Cíle této práce, které byly popsané úvodem, konkrétně návrh a vývoj fyzického zařízení, integrace RFID čtečky pro uživatelskou identifikaci, vývoj software pro správu her a uživatelského rozhraní, a testování a validace systému, tak byly splněny.

4.1 Náměty na další vývoj zařízení

Během řešení této práce se naskytlo několik dalších námětů, jak by se dalo zařízení rozšířit. Některé jsou níže popsány a mohou posloužit pro další vývoj zařízení.

Jelikož některé hry, které byly už v minulosti naprogramované studenty, a které se používaly pro testování tohoto zařízení obsahovaly zvukové nahrávky, bylo by vhodné zařízení rozšířit o audio modul.

Při tvorbě zařízení se ukázalo, že pouze dvě tlačítka jsou poměrně omezující při programování her, bylo by tedy žádoucí, aby se zařízení rozšířilo o další sadu uživatelských tlačítek.

Uživatelská aplikace by se dala rozšířit o statistiky jednotlivých her, například celková odehraná doba, žebříček oblíbenosti. Tyto statistiky spolu s jednotlivými tabulkami nejlepších hráčů a databází uživatelů by pak mohly být na samostatném internetovém webu, který by mohl přehledně zobrazovat tyto informace a umožňovat například hodnocení jednotlivých her.

Literatura

[1] LAFORE, Robert. *Object-Oriented Programming in C++*. Fourth Edition. Sams, 2001. ISBN 0672323087.

[2] *Object-oriented programming* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2024-05-08]. Dostupné z: `https://en.wikipedia.org/wiki/Object-oriented_programming`.

[3] *Advantages of Object Oriented Programming (OOP)* [online]. c2024. [cit. 2024-05-08]. Dostupné z: `https://www.theknowledgeacademy.com/blog/advantages-of-object-oriented-programming/`.

[4] *Difference Between Object And Class* [online]. 2023. [cit. 2024-05-08]. Dostupné z: `https://www.geeksforgeeks.org/difference-between-class-and-object/`.

[5] *Comparison of Handle and Value Classes* [online]. c1994-2024. [cit. 2024-05-08]. Dostupné z: `https://www.mathworks.com/help/matlab/matlab_oop/comparing-handle-and-value-classes.html`.

[6] *Parallel Computing Toolbox* [online]. c1994-2024. [cit. 2024-05-08]. Dostupné z: `https://www.mathworks.com/products/parallel-computing.html`.

[7] *Run Code on Parallel Pools* [online]. c1994-2024. [cit. 2024-04-07]. Dostupné z: `https://www.mathworks.com/help/parallel-computing/run-code-on-parallel-pools.html`.

[8] *Asynchronous Parallel Programming* [online]. c1994-2024. [cit. 2024-05-08]. Dostupné z: `https://www.mathworks.com/help/parallel-computing/asynchronous-parallel-programming.html?s_tid=CRUX_lftnav`.

[9] *Serial communication* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2024-04-07]. Dostupné z: `https://en.wikipedia.org/wiki/Serial_communication`.

[10] *Synchronous vs Asynchronous Protocols* [online]. [B.r.]. [cit. 2024-04-07]. Dostupné z: `https://www.labcenter.com/blog/sim-synch-asynch-protocols/`.

[11] *Serial Communication Protocols* [online]. 2019. [cit. 2024-05-09]. Dostupné z: https://circuitdigest.com/tutorial/serial-communication-protocols.

[12] *USB* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2024-05-09]. Dostupné z: https://en.wikipedia.org/wiki/USB%5C#2.0.

[13] *10 USB Pinout Explained* [online]. 2024. [cit. 2024-04-07]. Dostupné z: https://www.etechnophiles.com/usb-pinout-ports-connectors/?utm_content=cmp-true.

[14] *Universal asynchronous receiver-transmitter* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2024-05-09]. Dostupné z: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter.

[15] *Elements of a UART frame including optional bits* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2024-05-09]. Dostupné z: https://commons.wikimedia.org/wiki/File:UART_Frame.svg.

[16] FINKERZELLER, Klaus. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication*. 2010. vyd. WILEY, 2010. ISBN 978-0-470-69506-7.

[17] *ISIC* [online]. c2022. [cit. 2024-04-17]. Dostupné z: https://www.isic.org/.

[18] *What to Know about RFID* [online]. [B.r.]. [cit. 2024-04-07]. Dostupné z: https://www.cdotech.com/what-to-know-about-rfid/.

[19] *RFID Chips* [online]. c2024. [cit. 2024-04-07]. Dostupné z: https://www.chipsetc.com/rfid-chips.html.

[20] *ATmega328-AU* [online]. [B.r.]. [cit. 2024-04-07]. Dostupné z: https://www.digikey.cz/en/products/detail/microchip-technology/ATMEGA328-AU/2271029.

[21] *FT232RNL Datasheet* [online]. c2024. [cit. 2024-05-09]. Dostupné z: https://ftdichip.com/wp-content/uploads/2024/04/DS_FT232RN.pdf.

[22] *Filtering the 5V USB power supply line* [online]. 2015. [cit. 2024-04-06]. Dostupné z: https://andybrown.me.uk/2015/07/24/usb-filtering/.

[23] *ATmega328P Datasheet* [online]. c2015. [cit. 2024-05-09]. Dostupné z: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[24] BALBOA, Miguel. *Arduino library for MFRC522*. GitHub, 2023. Dostupné také z: https://github.com/miguelbalboa/rfid.

[25] *Types of UID in contactless RFID card systems* [online]. c2024. [cit. 2024-05-09]. Dostupné z: https://www.rfidcard.com/types-of-uid-rfid-card/.

[26] *MFRC522 Datasheet* [online]. 2016. [cit. 2024-05-09]. Dostupné z: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf.

[27] *Identification cards — Contactless integrated circuit(s) cards — Proximity cards - ISO/IEC 144443*. 1. vydání. 2001.

Seznam zkratek

OOP	Objektově orientované programování
USB	Universal serial bus
UART	Universal asynchronous receiver/transmitter
RFID	Radio frequency identification system
GPIO	General purpose input/output
ADC	Analog digital converter
UID	Unikátní identifikátor
SAK	Select Acknowledgement
SEL	Select
NVB	Number of valid bits
PCD	Proximity coupling device
PICC	Proximity integrated circuit card
API	Application programing interface

Přílohy

A Manuál pro tvorbu vlastní hry

A.1 User interface (APP_UI) třída

Třída APP_UI je vytvořena pro umožnění tvorby vlastních her pomocí OOP (objektově orientovaného programování) a pro spuštění her na zařízení k tomu určenému. Primárně bude sloužit jako pracovní pomůcka pro studenty, pro lepší pochopení OOP a testování jejich her.

Třída poskytuje rozhraní pro zobrazování hry, zpracování uživatelského ovládání a udržování databáze se skóre jednotlivých her.

Tato dokumentace slouží pro přehled veřejných metod a vlastností, které může student využít pro tvorbu vlastní hry.

A.2 Veřejné vlastnosti

Třída APP_UI obsahuje dvě vlastnosti, ke kterým má tvůrce hry přístup. Tyto vlastnosti jsou následující:

- **Axes**

Jedná se o objekt typu *uiaxes*, ve kterém se bude vykreslovat hra, k tomuto objektu má tvůrce přístup a může ho libovolně upravovat.

- **AxesPanelSize**

Jedná se o vektor, který obsahuje hodnoty velikosti objektu uipanel, ve kterém je objekt Axes, umístěn. Slouží pro vyčtení šířky a výšky, a vymezuje tak maximální rozměr, kterého může objekt Axes nabývat.

Struktura vlastnosti AxesPanelSize je následující:

$$\text{AxesPanelSize} = [\text{UIPanelWidth, UIPanelHeight}] \tag{4.1}$$

A.3 Veřejné metody

```
function saveScore(Score)
```

Popis
Funkce slouží k uložení skóre uživatele, pouze pokud je aktivní nějaký uživatelský profil.

Parametry

- Score - Hodnota skóre

```
function backToMainMenu()
```

Popis
Funkce slouží pro návrat do hlavního menu, bez uložení skóre, to je nutné udělat samostatně funkcí *saveScore*.

Parametry

- Funkce nemá žádné vstupní parametry.

```
function setTimerFreq(timerFreq)
```

Popis
Funkce slouží pro nastaví periody časovače, který ovládá krok hry.

Parametry

- timerFreq - Hodnota časovače [s]
  - Minimální hodnota je 0,01 s

```
function enableButtonsIRQ(enableArr)
```

Popis
Funkce slouží pro specifikování, kterých uživatelských vstupů bude hra využívat.

Parametry

- enableArr - Logický vektor, tj vektor obsahující jedničky nebo nuly, který odpovídá vektoru [Up Down Left Right Enter Exit]. Logická jednička určuje povolení daného vstupu a indikuje plánované použití. Pak je důležité definovat následující funkce, odpovídající těm vstupům, které jsou povoleny:

```
function BtnUpPressed()
function BtnDownPressed()
function BtnLeftPressed()
function BtnRightPressed()
function BtnEnterPressed()
function BtnExitPressed()
```

```
function sendJoystickValue()
```

Popis

V klasickém režimu nezasílá třída APP_UI přímé hodnoty z joysticku, ale pouze zasílá, který směr je aktivní. Pokud si uživatel přeje získávat z joysticku přímo tyto hodnoty, je možné je získat zavoláním této funkce. Je potřeba brát na paměti, že třída APP_UI má naprogramováno umělé zpoždění, tak aby se nevyvolávaly funkce aktivního směru příliš rychle za sebou, tato funkcionalita bude následně na uživateli, aby jí naprogramoval. Pokud bude tato funkce zapnutá, je potřeba místo funkcí BtnUpPressed, BtnDownPressed, BtnLeftPressed, BtnRightPressed definovat ve svém objektu funkci

```
function JoyValues(X, Y)
```

, která bude přijímat hodnoty z joysticku pokaždé, když přijdou z mikrokontroleru, tj. zhruba každých 50 ms.

Parametry

- Funkce nemá žádné vstupní parametry.

A.4 Struktura třídy hry

Hlavní třída hry musí dodržovat určitá pravidla pro správnou funkčnost.

Konstruktor objektu bude přijímat handle, referenci, na třídu APP_UI, který je vhodné si uložit do své proměnné, aby mohla třída hry následně volat veřejné metody.

Dále musí mít třída definovánu metodu *runFrame*, ve které bude probíhat hlavní smyčka hry s frekvencí definovanou pomocí metody *setTimerFreq*.

Nakonec musí mít třída definovány metody pro reagovaní na uživatelské vstupy, které bude používat. Pomocí funkcí:

```
function BtnUpPressed()
function BtnDownPressed()
function BtnLeftPressed()
function BtnRightPressed()
function BtnEnterPressed()
function BtnExitPressed()
```

Je vhodné, aby metoda *BtnExitPressed* byla využita pro vrácení do hlavního menu.

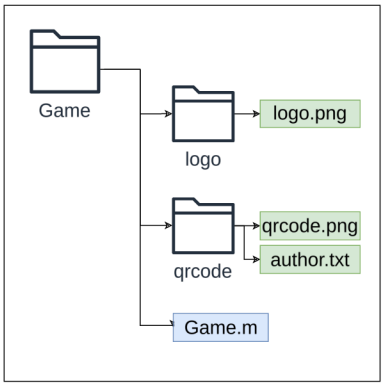
A.5 Struktura složky

Adresář hry má definovanou strukturu a bude uložen na sdíleném disku.

Hra bude v aplikaci pojmenována podle názvu této složky. Uvnitř složky se následně musí nacházet minimálně jeden soubor, a to MATLAB skript se třídou hry, který se bude jmenovat stejně jako složka hry.

Následně je možnost vytvořit podsložku logo, ve které bude jeden soubor s příponou .JPG nebo .PNG, který obsahuje logo hry, které bude zobrazené v prostředí.

Také je možnost vytvořit podsložku qrcode, ve které se můžou nacházet soubory author.txt, který obsahuje jméno autora a soubor s příponou .JPG nebo .PNG, který obsahuje QR kód, s odkazem na GitHub autora. Schéma finální podoby složky je na obrázku 4.1.



Obrázek 4.1: Struktura složky

A.6 Příklad

```
classdef Pong < handle
    properties
        hUI             %property for storing handle for UI
        axes            %property for storing handle for axes
        Ball
        PadlleL
        PadlleR
    end

methods
    %input for constructor is handle for UI object
        function this = Pong(hUI)
            this.hUI = hUI;     %storing handle for UI
            this.axes = hUI.Axes;   %storing handle for axes
            %setting the size of plot area
            set(this.axis, 'XLim', [0, Width], 'YLim', [0, Height]);
            %I want to use dimensions UP/DOWN and buttons ENTER/EXIT
            hUI.enableButtonsIRQ([1 1 0 0 1 1]);
            hUI.setTimerFreq(0.01); %setting the timer freqeuncy for 0.01 s
        end

    %main loop function
        function runFrame(this)
            if (this.GameStart)     %wait for starting the game
                %main loop
            end
        end
```

```
        %functions for handling user input
        function BtnUpPressed(this)
            this.PadlleR.moveUp();
        end

        function BtnDownPressed(this)
            this.PadlleR.moveDown();
        end

        function BtnEnterPressed(this)
            this.GameStart = 1;
        end

        function BtnExitPressed(this)
            this.hUI.backToMainMenu();
        end
    end
end
```