

GYMNÁZIUM JÍROVCOVA 8

MATURITNÍ PRÁCE

Protokoly TCP/IP

Alex Olivier Michaud

vedoucí práce: Dr.rer.nat. Mgr. Michal Kočer

V Českých Budějovicích 14. února 2023

2022/2023

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně s vyznačením všech použitých pramenů.

V Českých Budějovicích dne podpis

Alex Olivier Michaud

Abstrakt

Cílem této práce je poskytnout přehled základů komunikace na úrovni TCP/IP. Důraz je kladen na návrh vlastního aplikačního protokolu založeného na rodině protokolů TCP/IP a jeho následnou implementaci. Práce upozorní na klíčové aspekty komunikace v rámci TCP/IP a představí návrh a implementaci nového protokolu pro použití v síťové komunikaci. Ověření funkčnosti protokolu proběhne na jednoduché videohře napsané v Pythonu.

Klíčová slova

TCP/IP, Protokol, Sítě, Klient-server, Python, MicroPython

Poděkování

Rád bych vyjádřil hlubokou vděčnost Dr.rer.nat. Michalu Kočerovi, vedoucímu mé seminární práce, za jeho odborné vedení, podporu a povzbudivou pomoc po celou dobu mého výzkumu. Chtěl bych také upřímně poděkoval své rodině za její vytrvalou podporu a za korektury mé práce.

Obsah

I Základy komunikace aplikací na úrovni TCP/IP	2
1 Protokoly TCP/IP	3
1.1 historie TCP/IP	3
1.2 Základy komunikace aplikací na úrovni TCP/IP	4
1.3 Principy TCP/IP	4
1.4 Vrstva sítového rozhraní	4
1.5 Sítová vrstva	5
1.6 Transportní vrstva	5
1.7 Aplikační vrstva	6
1.7.1 Relační vrstva	6
1.7.2 Prezentační vrstva	6
1.7.3 Aplikační vrstva	7
1.8 Slovník protokolů a technologií	7
1.8.1 I2C	7
1.8.2 Wi-Fi	8
1.8.3 ARP	9
1.8.4 IP	9
1.8.5 TCP	10
1.8.6 DHCP	11
1.8.7 JSON	11
II Návrh aplikačního protokolu rodiny TCP/IP	12
1.9 Implementace protokolu	13
1.10 Popis vybraných nástrojů pro řešení	14

1.10.1	Python	14
1.10.2	MicroPython	14
1.10.3	Raspberry Pi Pico W	14
1.10.4	Tříosý akcelerometr GY-291 s ADXL345	15
1.10.5	Modul <code>socket</code>	15
1.10.6	Modul <code>network</code>	16
1.10.7	Modul <code>json</code>	16
1.10.8	Moduly <code>machine</code> a <code>ADXL345</code>	17
1.10.9	Modul <code>random</code>	17
1.10.10	Modul <code>pygame</code>	17
1.10.11	Modul <code>multiprocessing</code>	18
1.11	Popis řešení	18
1.11.1	Server	18
1.11.2	Client	21
1.11.3	Diskuze řešení	22
1.12	Využití dat	23
1.12.1	Návrh videohry	23
1.12.2	Popis videohry	23
1.12.3	Popis řešení videohry	24
1.12.4	Diskuze videohry	26
Bibliografie		27
Přílohy		29
A Zdrojový kód serveru		30
B Zdrojový kód klienta		32
C Zdrojový kód videohry		33

Úvod

Komunikace je základním aspektem moderních technologií a protokol TCP (Transmission Control Protocol) je jedním z klíčových protokolů, které umožňují tuto komunikaci. V této práci se budeme zabývat základy komunikace na úrovni TCP/IP a se zaměřením na jeho praktické využití.

V první části práce uvedeme přehled základů komunikace a úlohy a vlastnosti sítových protokolů při komunikaci.

V další části popíšeme návrh a vývoj vlastního aplikačního protokolu, který je založen na rodině protokolů TCP/IP. Půjde o vývoj jednoduchého aplikačního protokolu, který je určen k usnadnění komunikace mezi mikropočítačem a osobním počítačem.

V hlavní část práce se zaměříme na praktickou implementaci navrženého protokolu na konkrétním scénáři sítové komunikace. K tomu použijeme mikropočítač a osobní počítač, aby se demonstrovala funkčnost protokolu v reálném prostředí.

Na závěr provedeme zhodnocení navrženého protokolu, diskusi jeho silných stránek a omezení a navrhнем směry dalšího výzkumu a vývoje.

Cílem této práce je přispět k pochopení protokolu TCP a jeho praktického použití na konkrétním komunikačním scénáři. Práce má být prakticky zaměřená a pomoci studentům získat hlubší znalosti o protokolech TCP.

Část I

Základy komunikace aplikací na úrovni TCP/IP

1 Protokoly TCP/IP

1.1 historie TCP/IP

V roce 1966 se povedlo v USA Bobu Taylorovi úspěšně sehnat finance od Charlese Marii Herzfelda, ředitele ARPA,¹ na projekt ARPANET, který měl umožnit přístup k počítačům na velké vzdálenosti. V dalsích třech letech se rohodlo o počátečních standardech pro identifikaci, autentizaci uživatelů, přenos znaků a kontroly a roku 1969 byl ARPANET poprvé použit firmou BBN. [History], [History3], [History2], [ARPNET]

Při dalším výzkumu a pokusech o vytvoření nového modelu ARPANET, dva vědci Robert Elliot Kahn a Vinton Gray Cerf vytvořili nový model, kde hlavní zodpovědnost za spolehlivost byla předána uživateli místo sítě. Tímto roku 1974 vznikl nový protokol Transmission Control Program, který byl vydán v RFC² 675 s názvem Specification of Internet Transmission Control Program. Ale tato verze nebyla až do roku 1981 funkční, poté byla zprovozněna verzí 4. Je standardizována pomocí RFC 791 - Internet Protocol(IP) a RFC 793 Transmission Control Protocol(TCP). [History], [History3], [History2], [ARPNET], [IP], [TCP], [Rules]

TCP i IP, prošlo s postupem času velkým vývojem, kdy vznikalo stovky aktualizací. Například roku 1994 vzniklo Internet Protocol next generation (IPng), který zavádí IP verzi 6. Nyní se aktivně používá 10+ variant TCP na Linuxu. MacOS a Windows je má zavedeno jako výchozí nastavení. [History3], [History2], [History], [IP]

¹Nyní známo jako DARPA(Defense Advanced Research Projects Agency) je výkonná moc ministerstva obrany Spojených států amerických, které je pověřena vývojem technologií pro vojenské účely

²žádost o komentáře - označuje dokumenty popisující internetové protokoly

1.2 Základy komunikace aplikací na úrovni TCP/IP

TCP/IP je rodina protokolů, která umoňuje komunikaci uzel³ a to pomocí end-to-end⁴ principu a specifikováním toho jak by data měla být připravena, adresována, přenášena, směrována a přijmána. Tyto protokoly jsou nejčastěji děleny do čtyř úrovní Link, Internet, Transport a Application. [zakladykomunikace1], [zakladykomunikace2], [zakladykomunikace3]

1.3 Principy TCP/IP

TCP/IP stojí na několika zásadních principech jako klient-server, encapsulace, stateless a robustnost. [Princip5], [Princip4]

Client-server princip je vztah, kde jeden uzel požádá o službu nebo prostředek druhý uzel.

V TCP/IP modelu je uživatel klient(je mu poskytována služba) a další počítač je server.

[zakladykomunikace1], [Princip5], [Princip1], [Princip3]

Encapsulace je princip, který používá abstraktní dělení TCP/IP do čtyř úrovní. V každé takové úrovni se k původním datům přidávají další data, tak aby mohly být odesány přes síť.

Opačný proces, kdy uživatel se snaží dostat data se nazývá deencapsulace. [zakladykomunikace1] [Princip5]

Rodina TCP/IP protokolů je nazývána jako stateless. Tento princip říká, že jakákoliv žádost o službu od uživatele je nezávislá na té předchozí. Toto umožňuje lepší plynulost sítě, jelikož síťové cesty mohou být používány nepřetržitě. [Princip5]

Robustnost je princip, který dbá na to, aby uživatel neposílal žádná data, která by mohla způsobit problém druhému uživateli při procházení TCP vrstvami. Zároveň se snaží předvídat vše co dostane od druhého uživatele, co by mohlo způsobit problém a s případnými problémy nakládá liberálně. [Princip5], [Princip6]

1.4 Vrstva síťového rozhraní

Vrstva síťového rozhraní je nejnižší úroveň TCP/IP, dělí se na další dvě podkategorie, a to fyzická a logická. [Princip5], [Link4]

Na fyzické úrovni jsou všechna zařízení, kabely a etc., která konkrétně posílají bity. Protokoly

³bod přerozdělení nebo koncový bod komunikace

⁴snaží se o to, aby důležité role sítě byly řešeny konečným úzlem

na této úrovni jsou standardizovány IEEE⁵, například jsem patří protokol Ethernet⁶, Wi-Fi, etc. [Princip5], [Link2], [Link3]

Další součástí link úrovně je logická část, tato úroveň protokolů spojuje pouze síťový segment⁷ a posílá takzvané frame pouze v LAN(locální síť). Toto propojení zajišťuje pomocí různých protokolů, jako například ARP(Address Resolution Protocol), který umožňuje switchy, aby rozpoznal MAC adresy zařízení. Tato část se dále dělí na podčásti a to LLC a MAC podčást. LLC podčást umožňuje adresování a kontrolu logické části. Dále specifikuje mechanismy, pro zařízení, které adresují a kontroluje data, která jsou vyměněna mezi zařízeními. MAC podčást má zodpovědnost za možnost přístupu k mediu (CSMA/CD), nebo tento problém řeší pomocí MAC adres. [Princip5], [Link2], [Link3], [Link4], [Link5], [Link6]

1.5 Síťová vrstva

Síťová vrstva, v referenčním modelu TCP/IP známá také jako vrstva 2, je zodpovědná za směrování a předávání paketů v sítích. Jedná se o důležitý level, který umožňuje zařízením v různých sítích vzájemně komunikovat, díky němuž může fungovat internet. [sit1], [sit2]

Jedním z hlavních úkolů síťové vrstvy je směrování, které zahrnuje rozhodování o vhodné cestě pro každý paket na základě jeho cíle. K určení nejlepší cesty se používají různé metody a algoritmy pro směrování, od jednoduchých statických metod až po adaptivnější přístupy, které mohou zohlednit různé faktory v síti. [sit1], [sit2]

Kromě směrování je síťová vrstva zodpovědná také za realizaci předávání paketů v mezilehlých uzlech podél zvolené cesty a také za řízení toku dat a prevenci přetížení sítě. Hraje také klíčovou roli při propojování různých sítí, což umožňuje bezproblémovou komunikaci mezi nimi. [sit1], [sit2]

1.6 Transportní vrstva

Transportní vrstva, v referenčním modelu TCP/IP známá také jako vrstva 3, je důležitou součástí procesu síťové komunikace. Je zodpovědná za zajištění spolehlivé komunikace mezi aplikačními procesy běžícími na různých hostitelích v síti. [tran1], [tran2]

Mezi hlavní úkoly transportní vrstvy patří oprava chyb, segmentace a desegmentace dat a

⁵Institute of Electrical and Electronics Engineers

⁶kably s kroucenou dvojlinkou

⁷část počítačové sítě

zajištění doručení dat ve správném pořadí. K provádění těchto úkolů používá protokoly, jako je protokol TCP (transmission control protocol) a UDP (user datagram protocol). [tran1], [tran2], [tran3]

Transportní vrstva, která se v modelu OSI nachází mezi síťovou vrstvou (vrstva 3) a aplikační vrstvou (vrstva 7), zajišťuje koncové spojení mezi zdrojovým a cílovým hostitelem. To jím umožňuje komunikovat bez rušení jinými síťovými komponenty. [tran1], [tran2], [tran3]

Souhrnně řečeno, transportní vrstva shromažďuje segmenty zpráv z aplikační vrstvy a přenáší je do sítě, kde jsou znova sestaveny a doručeny do aplikační vrstvy cílového hostitele. Je nezbytnou součástí procesu síťové komunikace, poskytuje spolehlivé transportní služby vyšším vrstvám a umožňuje aplikacím komunikovat mezi sebou napříč sítí. [tran1], [tran2], [tran3]

1.7 Aplikační vrstva

Aplikační vrstva je v referenčním modulu TCP/IP určena číslem čtyři, ale jelikož je pro mou práci nejdůležitější tak si tuto vrstvu rozdělíme podle podrobnějšího modelu ISO/OSI, kde aplikační vrstva se dělí na vrstvu relační, prezentační a aplikační.

1.7.1 Relační vrstva

Relační vrstva je zodpovědná za navazování, udržování a ukončování komunikačních relací mezi dvěma koncovými body. Zajišťuje, aby komunikace mezi dvěma koncovými body byla spolehlivá a probíhala hladce, i když dojde k chybám nebo přerušení na nižších vrstvách. Například pomocí synchronizace je umožněno do posílaných dat přidat kontrolní body. Díky těmto bodům, v případě chyby, si příjemce může znova vyžádat poslání dat od určitého bodu. Zároveň umožňuje, aby na stejné přenosové lince probíhalo více komunikačních relací současně, které mohou být duplexní, poloduplexní, či simplexní. [session1], [session2]

1.7.2 Prezentační vrstva

Prezentační vrstva je zodpovědná za doručování, formátování a šifrování informací při jejich předávání mezi různými systémy a aplikacemi. [presentation1], [presentation2]

Prezentační vrstva zajišťuje, aby syntaxe a sémantika přenášených zpráv byla standardizována a ve správném formátu. Odpovídá za integraci všech různých formátů do standardizo-

vané podoby pro efektivní komunikaci a za kódování zpráv z formátu závislého na uživateli do společného formátu a naopak pro komunikaci mezi různými systémy. Pro vytvoření těchto formátu se používají různé serializace, jako například XML či TVL, které umožňují efektivní přenos složitých datových struktur. [presentation1], [presentation2], [presentation3]

Kromě serializace je prezentační vrstva zodpovědná také za šifrování a dešifrování dat. To se často provádí za účelem ochrany citlivých informací při jejich přenosu po sítích a může se provádět na různých vrstvách síťového zásobníku v závislosti na konkrétních požadavcích aplikace nebo protokolu. [presentation1], [presentation2], [presentation3]

1.7.3 Aplikační vrstva

Aplikační vrstva se dělí na dva prvky a to CASE⁸ a SASE⁹. Ty jsou určeny pro lehčí vytváření aplikací, protože poskytují stavební bloky, se kterými aplikace mohou pracovat. CASE poskytuje služby pro aplikační vrstvu a požaduje služby od vrstvy relací. Zatímco SASE poskytuje specifické aplikační služby, jako je přenos souborů, vzdálený přístup k databázi a zpracování transakcí. [application1], [application2], [application3]

Dále aplikační vrstva poskytuje několik funkcí, ty umožňují uživatelům snadný přístup k datům a manipulaci s nimi. Dovoluje uživatelům odesílat a přijímat e-maily, přistupovat k souborům na vzdáleném počítači a spravovat je, přihlašovat se jako vzdálený hostitel a přistupovat k informacím o různých službách. Poskytuje také protokoly, které zajišťují softwaru odesílat a přijímat informace a prezentovat uživatelům smysluplná data. [application1], [application2], [application3]

1.8 Slovník protokolů a technologií

Nyní zde popíšeme několik různých protokolů, ty jsou později konkrétně použité v naší praktické části. Jsou řazeny podle vrstev od nejnižší po nejvyšší a technologie jsou až za nimi.

⁸Common Application Service Element

⁹Specific Application Service Element

1.8.1 I2C

I2C¹⁰ je protokol používaný pro komunikaci mezi čipy. Tento protokol umožňuje připojit se do sběrnicového rozhraní zabudovaného do zařízení pro sériovou komunikaci. [i2c4], [i2c1], [i2c3]

Funguje na principu SDA¹¹ a SCL¹² rozhraní. SDA je využito na komunikaci a přenos dat mezi zařízeními a SCL je užito na přenos hodin. Dále se mezi zařízeními dohodne role Master nebo Slave. Počet obou zařízení Master i Slave je omezen počtem adres. Ale pro naše účely pracujeme s módem pouze jednoho Master zařízení. [i2c4], [i2c1], [i2c3], [i2c4]

Master zařízení zahájí komunikaci tím, že v kanále SDA změní napětí z vysokého na nízké, a obráceně u kanálu SCL. Nyní Master zařízení pošle Slave zařízením adresu, pokud ta souhlasí, Slave zařízení pošle ACK¹³ zprávu. Nyní je komunikace navázána. Dále pokaždé když dostane Master zařízení rámec s daty od Slave zařízení, posílá mu na zpět ACK zprávu. [i2c4], [i2c3]

Výhody této komunikace jsou v jednoduchosti zapojení, v počtu zařízení v Slave roli a ve spolehlivosti a dostupnosti na mnoha zařízeních. [i2c1], [i2c4]

1.8.2 Wi-Fi

Wi-Fi je označení zařízení pro přenos dat s danými specifikacemi, která prošla v minulosti testováním jedním ze členů organizace **Wireless Ethernet Compatibility Alliance** (WECA), dnes s názvem **Wi-Fi Alliance**. Tato zařízení využívají standard IEEE 802.11, který umožňuje bezdrátově sdílení dat. Nejpoužívanějšími standardy Wi-Fi jsou 802.11b a 802.11a. [WiFi1], [WiFi2]

Tyto standardy využívají rádiové vlny k přenosu informací mezi zařízeními a směrovačem prostřednictvím specifických frekvencí. V závislosti na množství přenášených dat lze využít dvě rádiové frekvence: 2,4 GHz a 5 GHz. Standard 802.11b využívá frekvenci 2,4 GHz, zatímco standard 802.11a využívá frekvenci 5 GHz. [WiFi1]

Wi-Fi také využívá různé architektonické postupy, přičemž pro naše účely jsou nejdůležitější přístupové body (AP) a antény¹⁴. Metoda AP zahrnuje použití stacionárního přístupového

¹⁰Inter-Integrated Circuit

¹¹Serial Data

¹²Serial Clock

¹³acknowledgement

¹⁴Ta sice nepoužívá standarty 802.11, nýbrž vzniká kolaborací různých firem, ale pro účely práce má stejně využití, jako wifi

bodu, který funguje jako základní rádiová stanice a datový most, ten je obvykle připojený k síti prostřednictvím technologie Ethernet. Tento přístupový bod také nastavuje potřebná bezpečnostní opatření. [WiFi1], [Pruvodce]

Technologie antén se často využívá v otevřených venkovních prostorách, kde je nutná komunikace na velké vzdálenosti, navíc se využívají i bleskojistky a přídavné antény. [Pruvodce]

1.8.3 ARP

Protokol ARP¹⁵ je protokol používaný v LAN¹⁶ k určení fyzické adresy zařízení pomocí adresy síťové vrstvy (např. IP adresy). Když chce zařízení komunikovat s jinými ve stejné síti LAN, potřebuje znát cílovou fyzickou adresu, aby mu mohlo poslat data. Má však k dispozici pouze IP adresu cílového zařízení. Zde přichází na řadu protokol ARP. [arp3], [arp1]

Aby zdrojové zařízení zjistilo fyzickou adresu toho cílového, rozešle všem uzelům v síti LAN paket s požadavkem ARP. Paket obsahuje IP adresu cílového zařízení a žádost o jeho fyzickou adresu. Paket obdrží všechny uzly v síti LAN, ale pouze to se shodnou IP adresou odpoví svou fyzickou adresou. Tato odpověď je odeslána zpět zdroji ve formě paketu odpovědi ARP.

[arp3], [arp2]

Zdrojové zařízení pak uloží fyzickou adresu cílového zařízení do své mezipaměti ARP, což je tabulka, která mapuje IP adresy na fyzické adresy. Tímto způsobem může použít fyzickou adresu z mezipaměti pro budoucí komunikaci s cílovým zařízením, místo aby musel vysílat požadavek ARP pokaždé, když chce odeslat data. Mezipamět ARP má hodnotu časového limitu, která udává dobu, po kterou zůstane fyzická adresa v mezipaměti, než ji bude třeba obnovit. [arp1], [arp2], [arp3]

1.8.4 IP

Internetový protokol (IP) je klíčovou součástí internetu, která je zodpovědná za směrování datových paketů mezi zařízeními v síti. [IP2], [IP3], [IP4]

Jedním z hlavních úkolů protokolu IP je přenášet datové pakety, nazývané IP datagramy, přes mezilehlé uzly k jejich cíli. K tomu protokol IP využívá informace o topologii sítě, takzvaně směrovací informace, které rozhodují o dalším směru přenosu IP datagramu. Tento proces se nazývá směrování. [IP2], [IP3], [IP4]

¹⁵Address Resolution Protocol

¹⁶Lokální síť

Protokol IP pracuje s abstraktními adresami, takzvanými adresami IP, což jsou 32bitová čísla, která identifikují zařízení v síti. Tyto adresy se používají k určení cesty, kterou mají datové pakety projít, aby dosáhly svého cíle. Aby bylo možné přenášet data mezi zařízeními v různých sítích, spoléhá protokol IP na překlad síťových adres. Protokol IP slouží k převodu mezi adresami IP a fyzickými adresami, například adresami sítě Ethernet. [IP2], [IP3], [IP4]

Protokol IP může pracovat ve spolehlivém nebo nespolehlivém režimu. Ve spolehlivém režimu je protokol IP zodpovědný za správné doručení datových paketů a podnikne kroky k opravě případných chyb. V nespolehlivém režimu protokol IP jednoduše zahodí všechna poškozená data a pokračuje dál, opravu chyb přenechá protokolům vyšších vrstev. Nespolehlivý režim je obecně efektivnější, protože snižuje dobu spojenou s opravou chyb. [IP2], [IP3], [IP4]

Kromě směrování datových paketů a adresování poskytuje protokol IP také možnosti fragmentace a opětovného sestavení. To umožňuje protokolu IP přenášet datové pakety, které jsou větší než maximální přenosová jednotka sítě, jejich rozdelením na menší pakety a jejich opětovným sestavením v cíli. [IP1], [IP2], [IP3], [IP4]

1.8.5 TCP

Protokol TCP¹⁷ je základní součástí internetu a zajišťuje spolehlivý přenos dat mezi zařízeními. Je to protokol zaměřený na spojení, což znamená, že navazuje a udržuje spojení mezi zařízeními nebo aplikacemi, dokud nedokončí výměnu dat. [Pruvodce], [TCP], [TCP1]

Protokol TCP je zodpovědný za rozdelení původní zprávy do paketů, jejich očíslování a předání vrstvě IP k transportu do cílového zařízení. Dále se stará o přenos případných odložených paketů, spravuje řízení toku a zajišťuje, aby všechny pakety dosáhly svého cíle.

[Pruvodce], [TCP], [TCP1]

K navázání spojení mezi zařízením a serverem používá protokol TCP třícestný handshake, který zajišťuje současný přenos více spojení soketů TCP v obou směrech. Zařízení i server musí před zahájením komunikace synchronizovat a potvrdit pakety a poté mohou vyjednávat, oddělovat a přenášet spojení soketů TCP. [Pruvodce], [TCP], [TCP1], [TCP2]

Jednou z hlavních výhod protokolu TCP je jeho spolehlivost. Je navržen tak, aby zajistil, že všechny pakety dosáhnou svého cíle, i když se některé pakety během přenosu ztratí. Toho dosahuje opakováním přenosem ztracených paketů a kontroluje zda nedošlo k chybám. To

¹⁷Transmission Control Protocol

z něj činí ideální protokol pro aplikace, vyžadující spolehlivý a bezchybný přenos, jako je e-mail a přenos souborů. [Pruvodce], [TCP], [TCP1]

1.8.6 DHCP

Protokol DHCP¹⁸ je síťový protokol, který umožňuje automatizovat proces, získávání IP adres a dalších konfiguračních informací v LAN. DHCP je součástí client/server architektury, kde v síti se vyskytuje DHCP server, ten čeká na požadavky a po dotazu automaticky vydá konfigurační informace. Zároveň po určité době si vezme danou IP adresu zpátky a přidá jí opět do databáze s volnými IP adresami. [DHCP1], [DHCP2], [DHCP4]

Kromě přidělování IP adres poskytuje DHCP také další konfigurační informace, jako je subnet¹⁹, adresa výchozí brány a DNS. Protokol DHCP je standardem IEEE, vychází ze staršího protokolu BOOTP (bootstrap protocol). Ale protokol BOOTP funguje pouze v sítích IPv4, což z něj činí protokol zastaralý. [DHCP1], [DHCP2], [DHCP3] , [DHCP4]

1.8.7 JSON

JSON²⁰ je způsob serializace dat, který je jednoduše pochopitelný lidmi, a zároveň zařízeními, díky čemu se prosadil, jako jeden z nejpoužívanějších způsobů pro serializaci dat. Tohoto dosáhl pomocí již zmíněné jednoduchosti, oproti ostatním formátům, jako například XML, který byl uživatelsky silně nepřívětivý. [JSON1], [JSON2], [JSON3]

Pochází z programovacího jazyka JavaScript, ve kterém kopíruje formu JavaScript objektu. Ten je ale zároveň snadno interpretovatelný v ostatních programovacích jazycích, díky tomu je to univerzální nástroj pro přenos dat mezi zařízeními a aplikacemi. [JSON1], [JSON2], [JSON3], [JSON4]

¹⁸Dynamic Host Configuration Protocol

¹⁹Subnet je logickým rozdělením sítě IP na více menších síťových částí.

²⁰JavaScript Object Notation

Část II

Návrh aplikačního protokolu rodiny TCP/IP

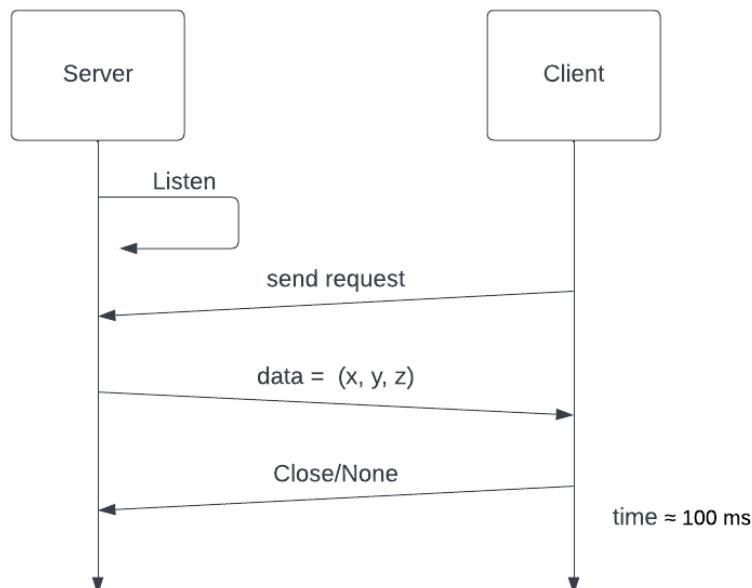
1.9 Implementace protokolu

Protokol²¹ je navržen tak, aby umožnil klientovi vyžádat si data od serveru s akcelerometrem a server zaslal požadovaná data klientovi. Toto je zajištěno tím, že server neustále poslouchá na daném portu. A kdykoliv přijde od klienta požadavek `send`, tak server odešle data. Pokud klient již data nechce, může poslat zprávu `close` nebo přestat posílat požadavky, což způsobí, že server spojení uzavře.

Abychom ověřily tuto implementaci. Nejdříve nastavíme server a klienta, kteří implementují tento protokol. Necháme klienta poslat serveru požadavek na data. Ověříme zda server odešle klientovi požadovaná data. Dále necháme klienta odeslat zprávu `close` nebo přestat posílat požadavky a pozorujeme zda se tak stane.

Pokud se ověří daná implementace můžeme předpokládat, že Protokol je funkční.

Obrázek 1.1: sekvenční diagram protokolu



²¹soubor pravidel, podle kterých probíhá elektronická komunikace nebo datový přenos, mezi dvěma (či více) konečnými body.
Více zde [\[slovnik\]](#)

1.10 Popis vybraných nástrojů pro řešení

1.10.1 Python

Jazyk použitý pro implementaci protokolu je Python. Zvolili jsme si tento jazyk z několika důvodů. Python má širokou standardní knihovnu, která obsahuje řadu modulů pro síť a komunikaci, například sokety, které usnadňují vytvoření Protokolu. Python má čistou a čitelnou syntaxi, která usnadňuje psaní a pochopení kódu. Navíc je to dynamicky typovaný jazyk, což znamená, že v kódu nemusíte uvádět typy proměnných. To může výrazně usnadnit psaní a debug kódu. To je několik důvodů, proč byl vybrán oproti ostatním jazykům, jako například C, C++, Java a Go.[**python**]

1.10.2 MicroPython

Dalším využitým nástrojem je MicroPython. MicroPython je implementace Pythonu, která je navržena pro snadné použití a provoz na malých mikrokontrolérech, jako ESP8266, ESP32 a Raspberry Pi Pico W. Tohoto dokázala díky speciální optimalizaci, která spočívá ve vymázaní většiny standardní knihovny a přidání modulů pro specifickou práci s mikrokontroléry. Mezi tyto moduly patří moduly pro podporu sítí, interakci se senzory, displeji a dalšími hardwarovými komponentami.[**micropython**]

1.10.3 Raspberry Pi Pico W

Hardwarové řešení je Raspberry Pi Pico W, která se hodí pro vytváření prototypů a experimentování. Snadno se používá a lze ji programovat v různých jazycích, včetně jazyků MicroPython a C. Díky tomu je skvělou volbou pro projekty, v nichž chcete rychle a snadno vytvářet prototypy a testovat své nápadы. Dále má vestavěné rádio 2,4 GHz, které lze použít pro komunikaci WiFi, a na připojení Bluetooth Low Energy (BLE). Má i řadu digitálních a analogových vstupů a výstupů, které lze využít k propojení se senzory a dalšími zařízeními. Podporuje protokol I2C²², který je důležitý pro práci s akcelometrem. [**raspberry**]

²²Inter-Integrated Circuit

Obrázek 1.2: Raspberry Pi Pico W



1.10.4 Tříosý akcelerometr GY-291 s ADXL345

Jako data pro přenos jsme vybrali data z tříosého akcelerometru GY-291 s ADXL345. Tento senzor je přesný, cenově dostupný a používá se k měření pohybu a orientace. Měření probíhá ve 3 osách X, Y a Z a měří zrychlení v rozmezí -16 g až +16 g. Měření lze snadno získat, díky připojení akcelerometru GY-291 k mikrokontroléru pomocí rozhraní I2C. Tento senzor je kompatibilní s mikrokontroléry, jako je Arduino, Raspberry Pi a mnoho dalších.

Senzor je také schopen detektovat orientaci v prostoru díky integrovanému gyroskopu. Jeho nejčastější použití spočívá v úkolech, jako stabilizace kamery nebo detekce pohybu v prostoru pro různé aplikace.

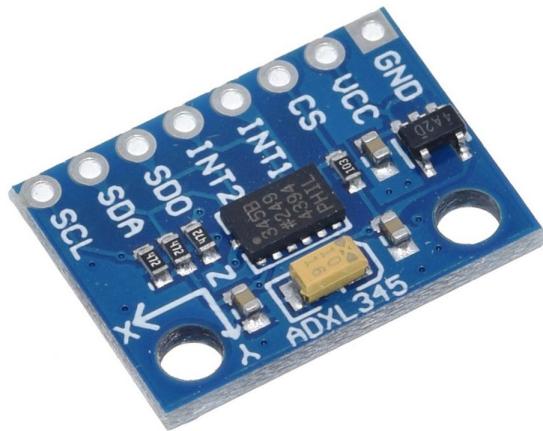
Další možnosti pro sběr dat byl například senzor MPU-6050, který má sběrnici dat v 6 rozměrech, ten kombinuje akcelometr v 3 osách a gyroskop také ve třech osách. Je určen na velmi přesné měření, které zde nebylo nutné. Další volbou byl senzor BNO055 ten kombinuje akcelometr, gyroskop a magnetometr ve 3 osách, ten je určen na přesnou orientaci v prostoru, které také nebyla nutná. [\[gyroskop\]](#), [\[MPU\]](#), [\[BNO\]](#)

1.10.5 Modul `socket`

Modul `socket` je vestavěný modul do jazyka Python, který poskytuje rozhraní pro práci se sockety, včetně jejich vytváření a používaní pro přijímání a odesílání dat po síti. Podporuje různé rodiny adres včetně `AF_INET` (IPv4) a `AF_INET6` (IPv6). Podporuje také řadu typů soketů, včetně `SOCK_STREAM` (TCP) a `SOCK_DGRAM` (UDP). [\[socket\]](#)

Díky tomuto modulu lze provádět spousty standardních akcí očekávaných od sítí, například

Obrázek 1.3: Tříosý akcelerometr GY-291 s ADXL345



navazovat spojení se servery, odesílat a přijímat data a uzavírat spojení, pokud již nejsou potřeba.

Alternativy k tomuto modulu jsou moduly `AsyncIO` a `PySocket`. `AsyncIO` je standardní knihovna určena na asynchronní programování a mimo to podporuje i užití asynchronních socketů. Tato knihovna je vhodná pro aplikace vyžadující vysoký výkon a škálovatelnost. Tato knihovna nebyla vybrána z důvodu neznalosti asynchronního programování. `PySocket`, je modul třetí strany, který funguje na bázi `socket` modulu, jedná se pouze o jeho zjednodušení. `PySocket` nebyl vybrán z toho důvodu, že není implementován v MicroPythonu a není to standardní knihovna v Pythonu. [`async`], [`pysocket`]

1.10.6 Modul `network`

Modul `network` je jeden z klíčových modulů v MicroPythonu, je určen pro možnost připojit se, odesílat zprávy a přijímat data přes síť. Poskytuje celou řadu síťových možností, konkrétně v protokolu je využit na připojení se k síti. [`network`]

1.10.7 Modul `json`

Modul `json` je vestavěný modul v jazyce Python a implementaci MicroPython. Poskytuje funkce pro práci s daty ve formátu JSON. Díky `json` knihovně je možné převádět data JSON na objekty jazyka Python a naopak. Jeho funkce umožňují jednoduchou serializaci dat a odeslání přes sockety [`json`]

Alternativou ke knihovně `json` je modul `ujson`. Tento modul je napsán v Pythonu a poskytuje rychlejší služby za využití méně paměti. Pro náš protokol, který je určen pro přenos

malých dat, toto nebylo nutné. [**ujson**]

1.10.8 Moduly `machine` a `ADXL345`

Modul `machine` je vestavěná knihovna v MicroPythonu určená pro práci s hardwarovými obvody, jako jsou časovače, I/O²³ piny a I2C rozhraní, které je podstatné pro práci s GY-291 s ADXL345. Tohoto modulu dále využívá modul třetí strany [**machine**]

ADXL345, ten je užitečný pro snadnou práci s ADXL345. Modul získává z paměti informace o stavu zařízení a data naformátuje do stavu použitelného v aplikacích. K modulu `machine` existuje alternativa modul `pyb`, ten ale musí být doinstalován do MicroPythonu. K `ADXL345` modulu v momentu implementace nebyla alternativa. [**pyb**], [**adxl**]

1.10.9 Modul `random`

`random` je vestavěná Python knihovna, ta umožňuje generovat pseudo náhodná čísla a provádět náhodné výběry. Nejvíce uplatnění tento modul najde při tvorbě her, či dalších programů, kde je potřeba generovat pseudo náhodné výsledky. [**random**]

Alternativou k modulu `random` je modul `secrets`. `Secretes` je modul, určený hlavně na generování kryptograficky bezpečných čísel a řetězců, což není nutné pro tvorbu videoher. [**secretes**]

1.10.10 Modul `pygame`

`pygame` je sbírka různých Python modulů. Dohromady tvoří jeden celek, který je určen ke tvorbě videoher. Uživatel má možnost zobrazovat obraz, pouštět zvuk, či pracovat s uživatelským vstupem. Díky těmto možnostem můžete tvořit celé hry v Pythonu. [**pygame**]

`pygame` dále vyčnívá díky své dobré implementaci, kde hlavní funkce jsou tvořeny v jazyce C a Assembly. Tato implementace zaručuje dostatečnou rychlosť kodů, která je při hrách nutná. Na to navazuje i snadná práce s více jádry najednou. Dále je pygame určen pro mnoho operačních systémů a mnoho prostředí. Toto vše z něj činí dobrý multifunkční nástroj, který je snadno přenositelný.

Další výhodou modulu `pygame` je důraz na jednoduchost použití a ovládání. Potřebná doba pro naučení se práci s ním je nízká, proto je velmi vhodný k testování protokolů.

²³input/output

Alternativa k `pygame` je například `pyglet`. `pyglet` je knihovna určená na práci se zobrazěním oken a hraním zvuku. Na rozdíl od `pygame` má menší podporu pro vytváření videoher. Dalším důležitým rozdílem je, že `pyglet` používá jiný program pro vykreslování grafiky, ten je určen spíše pro 3D grafiku. [[pygame](#)], [[pyglet](#)]

1.10.11 Modul `multiprocessing`

Modul `multiprocessing` je Python modul určený pro paralelní provádění kódů na několika jádrech. Umožňuje vytvářet několik samostatných procesů, které mohou běžet současně na jednom zařízení, či dokonce na několika síťově připojených počítačích. Tato funkce je užitečná pro výpočetně náročné úkoly a pro programy, kde je nutné, aby několik programů běželo ve stejnou chvíli. K tomu `multiprocessing` přidává možnosti sdílet data mezi těmito programy pomocí různých datových typů. [[multiprocess](#)]

Alternativa k modulu `multiprocessing` je modul `threading`. Hlavní rozdíl mezi nimi je, že `threading` používá threads a `multiprocessing` používá procesy²⁴. Ty se liší tím, že threads používají jedno jádro a procesy využívají více jader. Z toho vyplývají rozdíly ve funkčnosti. U procesů je náročné sdílet data, ale v případě chyby se ukončí pouze jeden proces a ne celý program. V případě chybovosti u threadu se ukončí celý program. Problém se sdílením dat je ale snadno řešitelný, díky implementaci modulu `multiprocessing`, a je tak vhodný pro můj protokol. [[threading](#)]

1.11 Popis řešení

1.11.1 Server

Zdrojový kód pro server A.1 je celý napsán v MicroPythonu. Kód začíná importem modulů, které jsou popsány v předešlé kapitole. Pokračujeme nastavením proměnných užitých později v programu. Proměnná `HOST` je využita na nastavení rozsahu, ve kterém bude server přijímat dotazy od klienta. Proměnná `PORT` je využita na nastavení portu, na kterém se server bude pohybovat. Nakonec zavedeme funkci pro připojení k internetu.

```
1| import socket
2| import network
```

²⁴proces je instance vytvořená `multiprocessing` modulem

```

3 from machine import Pin, I2C, SoftI2C
4 import ADXL345 # https://github.com/DFRobot/micropython-dflib/
    tree/master/ADXL345
5 from json import dumps
6
7 PORT = 8880
8 HOST = "0.0.0.0"
9
10 def do_connect(ssid, password):
11     import network
12     wlan = network.WLAN(network.STA_IF)
13     wlan.active(True)
14     if not wlan.isconnected():
15         print('Connecting to network... ')
16         wlan.connect(ssid, password)
17         while not wlan.isconnected():
18             pass
19     print(f'Network config: {wlan.ifconfig()}')

```

Zdrojový kód 1.1: server.py

Nyní si zavedeme hlavní část programu, pro který jsme vytvořili blokové schéma pro jeho lepší pochopení.

Jak je vidět na schématu, nejdříve navážeme spojení se sítí díky funkci `do_connect`. Pokračujeme vytvořením objektu `i2c`, předáme ho objektu `ADXL345`, díky kterému později budeme získávat souřadnice. Nakonec pokračujeme vytvořením socketu, v něm nastavíme typ protokolu. Na síťové vrstvě protokol IP a na transportní vrstvě protokol TCP. Dále nastavíme socketu jeho port číslo a rozmezí, na němž bude přijímat dotazy. Nakonec nastavíme, aby daný program poslouchal a bral maximálně 100 clientů v jeden moment.

```

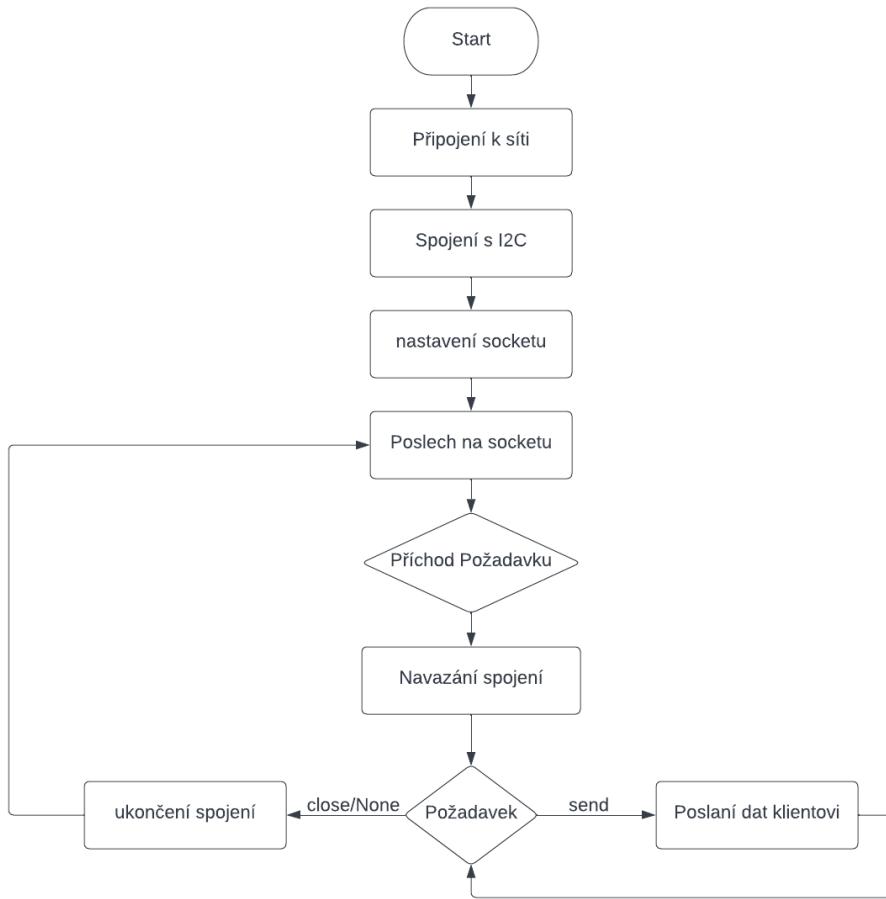
1 if __name__ == '__main__':
2     do_connect("SSID", "password")
3     i2c = SoftI2C(scl=Pin(17), sda=Pin(16), freq=10000)
4     adx = ADXL345.ADXL345(i2c)
5     soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6     soc.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
7     soc.bind((HOST, PORT))
8     soc.listen(100)

```

Zdrojový kód 1.2: server.py

Nyní pokračujeme hlavní programovou smyčkou, tato smyčka je nekonečná. Ta čeká na dotaz klienta. Když se client dotáže, program spustí další smyčku. V té program přijme data posланé od klienta a dekóduje je. A nyní podle typu dat se rozhodne, jaký bude další postup.

Obrázek 1.4: blokový diagram serveru



Pokud nejsou žádná data, nebo dávají dotaz na `close`, program uzavře spojení a nastaví socket tak, aby v případě nutnosti mohl být použit co nejdříve znova. Když data dávají dotaz na `send`, program díky objektu `ADXL345` zjistí souřadnice akcelometru. Tyto data dá do datové struktury `tuple` a datovou strukturu serializuje způsobem JSON. Dále stejná data dá do kódování UTF-8 a pošle je přes socket klientovi.

```

1  while True:
2      print("wating for request")
3      conn, address = soc.accept()
4      while True:
5          data = conn.recv(2048)
6          data = data.decode("utf-8")
7          print(data)
8          if data == None:
9              conn.close()
10             soc.setsockopt(socket.SOL_SOCKET, socket.
11                           SO_REUSEADDR, 1)
  
```

```

11         break
12     if data == "send":
13         x = adx.xValue
14         y = adx.yValue
15         z~= adx.zValue
16         data = dumps((x, y, z))
17         conn.sendall(data.encode("utf-8"))
18     elif data == "close":
19         print("close")
20         conn.close()
21         soc.setsockopt(socket.SOL_SOCKET, socket.
22                         SO_REUSEADDR, 1)
23         break

```

Zdrojový kód 1.3: server.py

1.11.2 Client

Celý zdrojový kód klienta B.1 je napsán v Pythonu. Kód začína importem modulů a po-kračuje stanovením globálních proměnných. Nastaví proměnné PORT a IP, ty musí souhlasit s nastavením serveru.

```

1 import socket
2 from json import loads
3
4 PORT = 8880
5 IP = "192.168.102.197"

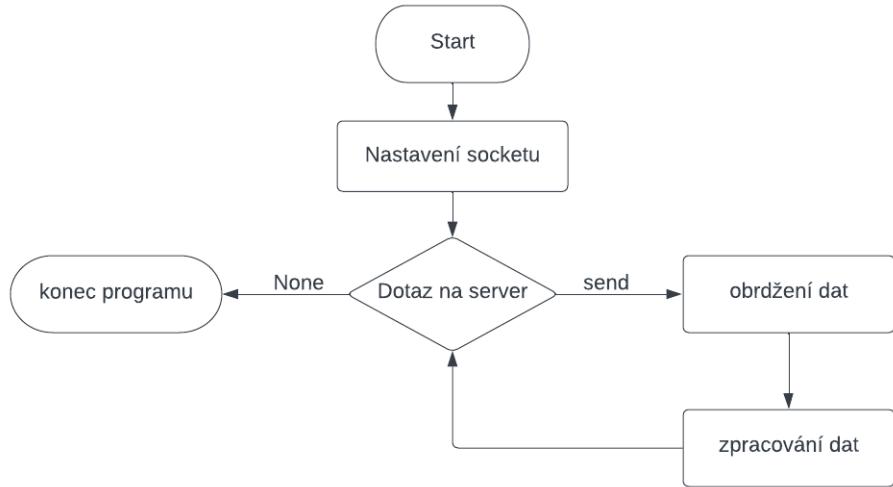
```

Zdrojový kód 1.4: client.py

Nyní se zaměříme na hlavní kód programu, je napsán ve funkci z důvodu jeho pozdějšího použití, jako proces. Zároveň má jeden argument queue, to je objekt modulu `multiprocessing`, ten je později využit na sdílení dat mezi procesy. Pokračujeme hlavním programem. Nejdříve nastavíme typ protokolu. Na síťové vrstvě protokol IP a na transportní vrstvě protokol TCP.

Dále definujeme hlavní smyčku. Ta se ihned po spuštění dostane do další smyčky. V této smyčce se nejdříve serveru pošle požadavek na `send`. Dále klient vyzkouší, jestli mu server poslal data. V případě že data nejsou přijata, vydá se chybové hlášení o ztracení kontaktu a klient se pokusí znova navázat kontakt. Jestli jsou data přijata, klient je dekóduje. Přeformátuje je z JSON formátu do formátu datové struktury `tuple`. Dále tyto data vloží do řady. Toto opakuje do té doby, než se program přeruší.

Obrázek 1.5: blokový diagram klienta



```

1 def client(queue):
2     s~= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3     s.connect((IP, PORT))
4     while True:
5         print("Sending")
6         while True:
7             msg = "send"
8             s.sendall(msg.encode("utf-8"))
9             try:
10                 data = s.recv(1024)
11             except socket.error:
12                 print("Connection lost")
13                 continue
14             data = data.decode("utf-8")
15             data = loads(data)
16             queue.put(data)
  
```

Zdrojový kód 1.5: client.py

1.11.3 Diskuze řešení

Výsledek je funkční a je proveden pomocí zamýšlené implementace. Toto umožní uživateli, který vlastní zařízení Raspberry Pi Pico W s ADXL345, účinně přenášet data o poloze přes WiFi, k různým účelům.

Limitací ze strany serveru je několik. Program umožňuje pouze dva typy dotazů a to `send` a `close`. Proto může dojít k tomu, že klient pošle jiný dotaz, a s tímto dotazem nemusí být naloženo adekvátně. Dále kód nemá implementované žádné mechanismy proti chybám,

nebude proto přiraven na selhání, popřípadě na různé krajní situace. Následně i přestože je komunikace pouze v lokální síti, měla by být šifrována. Dalším úzkalím může být, že kód není optimalizován na výkonnost, či práci s pamětí. Pokud tedy uživatel bude chtít použít zařízení s menší výkonností nežli Raspberry Pi Pico W, může se dostat do problémů. A v poslední řadě, server může pracovat pouze s akcelerometrem ADXL345 a nepodporuje jiné modely akcelerometrů.

Limitací ze stran klienta je také několik. Jedním z hlavních omezení je, že program pracuje pouze s jednou přednastavenou IP adresou, optimálnější by bylo, kdyby ji bylo možné určit dynamicky. Dalším problémem je, že klient není připraven na případné chyby, proto může snadno dojít k jeho selhání. Důkazem toho je předpoklad klienta, že server je spuštěný, pokud by nebyl, klient automaticky selže. Nakonec, klient po obdržení dat neodešle serveru žádnou potvrzovací zprávu, takže server neví, zda klient data úspěšně přijal.

Další směr vývoje může spočívat nejdříve v implementaci systému proti chybám a selháním a uskutečnění šifrování. Následný vývoj může spočívat ve vytvoření automatického nastavení IP adresy serveru, tak aby to více vyhovovalo klientské straně. Konečný vývoj by umožňoval využití co nejširšího počtu zařízení. Toho se dá dosáhnout vytvořením serveru tak, aby podporoval co nejvíce zařízení.

1.12 Využití dat

1.12.1 Návrh videohry

Pro ukázku praktického využití dat jsme si vybrali projekt založený na vytvoření videohry. Základní princip hry spočívá ve vesmírné lodi, která se vyhýbá meteoritům, a zároveň se snaží sbírat mince. Uživatel pohybuje s Raspberry Pi Pico W s ADXL345, pomocí toho ovládá lod na obrazovce a snaží se vyhýbat meteoritům. Po každém zásahu vemírné lodi meteoritem, se hráči ubere jeden život. Zároveň se hráči počítá skóre, které se zvyšuje sběrem mincí.

1.12.2 Popis videohry

Hráč se nejdříve objeví na hlavní obrazovce. Odtud má dvě možnosti. Může se podívat na celkové nejlepší skóre, které kdy nahrál. Další možnost je spustit hru. Po spuštění hry se objeví na obrazovce, kde se na dolní liště pohybuje vesmírná loď. Ta kopíruje pohyby hráče se zařízením na ovládání. Ihned po spuštění se na horní listě také objeví meteority

a mince, které putují k dolní liště, pod náhodným sklonem s náhodnou rychlostí. Dále se v levém horním rohu objeví celkový počet získaných mincí. V pravém horním rohu se ukazují obrázky ve tvaru srdce, které znázorňují životy hráče. Po každém nárazu hráče do meteoritu jedno srdce změní barvu na šedou, tím signalizuje ztrátu jednoho života. Životy jsou celkem tři.

1.12.3 Popis řešení videohry

Po spuštění se program rozdělí do dvou procesů. A to proces, ve kterém se nachází hra a proces klienta.

```
1 if __name__ == "__main__":
2     queue = Queue()
3     menu_proc = Process(target=main_menu, args=(queue,))
4     client_proc = Process(target=client, args=(queue,))
5     client_proc.start()
6     menu_proc.start()
7     client_proc.join()
8     menu_proc.join()
```

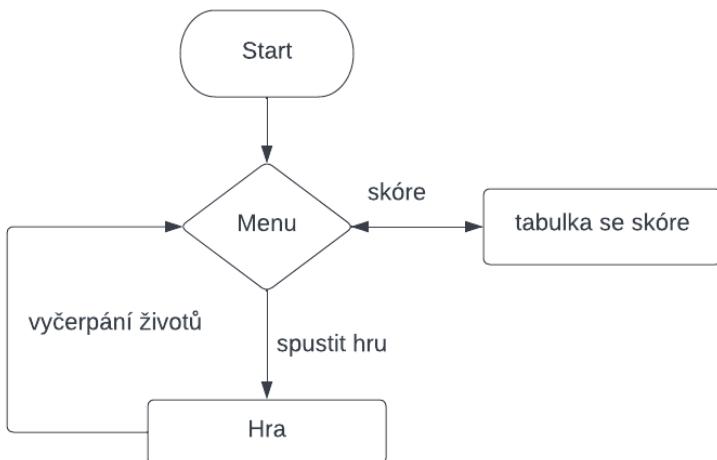
Zdrojový kód 1.6: game.py

Hra se dále dělí na několik funkcí, kde každá znázorňuje jednu obrazovku. Konkrétně jsou tři obrazovky, a to obrazovka se skóre, hlavní obrazovka a obrazovka videohry.

Klient se mezitím spojí se serverem a začne generovat data. Oba tyto procesy mezitím sdílejí objekt `queue`, který značí řadu. Do tohoto objektu klient přidává data získané od serveru, a jelikož proces se hrou běží 60-krát za sekundu, tak ihned data vyjmeme ze řady a použije je pro ovládání vesmírné lodi. Druhá možnost je, že data vyjmeme z řady a zahodí je, aby se loď nezačala pohybovat náhodně, pokud hráč znova spustí hru.

Nyní se dostáváme k herní logice. Ta se celá nachází v obrazovce s videohrou, konkrétně v její smyčce, která probíhá 60-krát za sekundu. Kvůli této rychlosti, se program nejdříve podívá, jestli objekt `queue` není prázdný. Pokud je prázdný, program pokračuje. Pakliže je v ní nějaký prvek, program vyjmeme data z řady a začne vypočítávat rychlost. To konkrétně pomocí změny proměnné s názvem `ship_speed`. Jestli se uživatel dostane za hranici levé lišty, program začne lodí přidávat takovou hodnotu, aby loď pokračovala na pravou stranu a naopak. Toto je napsáno proto, aby se loď nezasekla na žádné z bočních stran. Dále program zkonzroluje velikost dat, a to konkrétně, jestli data jdou do záporných, či kladných hodnot.

Obrázek 1.6: blokový diagram videohry



Jestliže jsou hodnoty záporné, loď se začne pohybovat levým směrem a naopak. Hodnota rychlosti se dále vypočítává pomocí lineární funkce, a to z toho důvodu, aby ovladač měl větší citlivost na prudké pohyby ovladačem, či naopak jemné pohyby.

```

1 if queue.empty() is False:
2     data = queue.get()
3     print(data)
4     if ship_rect.left > W:
5         ship_speed -= 2
6     elif ship_rect.left < 0:
7         ship_speed += 2
8     elif data[2] > 0:
9         ship_speed += 0.01 * data[2]
10    elif data[2] <= 0:
11        ship_speed += 0.01 * data[2]

```

Zdrojový kód 1.7: game.py

Dále začneme využívat rychlosť lodi. Nejdříve znova zkonzolujeme, že loď není mimo obrazovku, pokud loď mimo obrazovku je, tak její rychlosť nastavíme na nula. Tím zamezíme možnosti vyjetí lodi mimo obrazovku. Nyní přičteme k objektu vesírmé lodi vypočítanou rychlosť a pokračujeme v dalším doladění pohybu. Prvně když chtěl hráč rychle změnit směr pohybu lodi, tak se pohybovala pomalu, jelikož se rychlosť nasčítávala. Pro tento problém jsme zavedli následné řešení. Nejdříve zkonzolujeme jestli stará pozice byla záporná, a pokud zároveň byla nynější rychlosť kladná, tak rychlosť vynulujeme. Tím může loď ostře změnit svůj směr, a to i obráceně. Dále jestli rychlosť lodi, kvůli nasčítaní byla moc veliká tak

program rychlosť zmenší, aby byla snadnejšia ovladateľná.

```
1 if 0 <= ship_rect.left + ship_speed <= W - 90:
2     ship_rect.left += ship_speed
3     if old_position < 0 < ship_speed:
4         ship_speed -= ship_speed
5     elif old_position > 0 > ship_speed:
6         ship_speed += ship_speed
7     if ship_speed > 200:
8         ship_speed -= 100
9 else:
10    ship_speed = 0
```

Zdrojový kód 1.8: game.py

1.12.4 Diskuze videohry

Videohra dává bližší důkaz o reálném použití dat pro různé aplikace. Je možné díky ní si přiblížit odezvu protokolu a jeho schopnosti.

Ačkoli hra slouží jako užitečný nástroj, má určitá omezení. Hlavním omezením je její implementace, zejména pokud jde o logiku použitou pro výpočet rychlosti kosmické lodi. Ta by mohla být zlepšena, aby se zvýšila její citlivost a celková uživatelská přívětivost. Kromě toho se objevuje problém s pohybem ostatních objektů ve hře, ty mají tendenci opakovat určité pohyby nebo se nesrazí se spodní částí obrazovky.

Pro zlepšení hratelnosti by se budoucí vývoj mohl zaměřit na zdokonalení logiky pohybu vesmírné lodi a současně řešit pohyb ostatních objektů ve hře. Je však důležité poznamenat, že protokol má potenciál využití i mimo hru, například v robotice nebo při řízení stability a detekci pádů.

Závěr

V této práci jsme se zabývali základy komunikace na úrovni TCP/IP a zaměřili jsme se na praktické aplikace protokolu TCP (Transmission Control Protocol). Nejdříve jsme rozbrali, jakým způsobem zařízení komunikují, a dále protokoly, které jim toto usnadňují. Popsali jsme návrh a vývoj vlastního aplikačního protokolu založeného na rodině protokolů TCP/IP s cílem usnadnit komunikaci mezi mikropočítačem a osobním počítačem. Poté jsme demonstrovali funkčnost tohoto protokolu v reálném scénáři pomocí mikropočítače a osobního počítače.

Prostřednictvím naší praktické implementace jsme byli schopni vyhodnotit navrhovaný protokol a určit jeho silné stránky a omezení. Zjistili jsme, že protokol je schopen efektivně navázat a udržovat komunikaci mezi oběma zařízeními s minimálním zpožděním. Zaznamenali jsme však také, že by bylo možné provést další zlepšení. Hlavně co se týče odolnosti systému proti chybám a jeho zabezpečení.

S ohledem na tato zjištění navrhujeme, aby se budoucí výzkum v této oblasti zaměřil na odstranění zjištěných omezení, konkrétně v oblasti bezpečnosti a dynamičnosti protokolu. Kromě toho by bylo vhodné prozkoumat využití tohoto protokolu v dalších komunikačních scénářích, například v robotice, či při detekci pádů.

Celkově tato práce přispěla k pochopení protokolu TCP a jeho praktickému využití v konkrétním komunikačním scénáři. Doufáme, že poslouží jako cenný zdroj informací pro studenty a odborníky z praxe, kteří se zajímají o protokoly TCP, a že bude inspirací pro další výzkum v této oblasti.

Seznam obrázků

1.1	sekvenční diagram protokolu	13
1.2	Raspberry Pi Pico W	15
1.3	Tříosý akcelerometr GY-291 s ADXL345	16
1.4	blokový diagram serveru	20
1.5	blokový diagram klienta	22
1.6	blokový diagram videohry	25
C.1	hlavní obrazovka	41
C.2	obrazovka videohry	42
C.3	obrazovka skóre	42
C.4	destička s Raspberry Pi W	43

Přílohy

A Zdrojový kód serveru

```
1
2
3 import socket
4 import network
5 from machine import Pin, I2C, SoftI2C
6 import ADXL345 # https://github.com/DFRobot/micropython-dflib/
    tree/master/ADXL345
7 from json import dumps
8
9 PORT = 8880
10 HOST = "0.0.0.0"
11
12
13 def do_connect(ssid, password):
14     import network
15     wlan = network.WLAN(network.STA_IF)
16     wlan.active(True)
17     if not wlan.isconnected():
18         print('Connecting to network...')
19         wlan.connect(ssid, password)
20         while not wlan.isconnected():
21             pass
22     print(f'Network config: {wlan.ifconfig()}')
23
24
25 if __name__ == '__main__':
26     do_connect("SSID", "password")
27     i2c = SoftI2C(scl=Pin(17), sda=Pin(16), freq=10000)
28     adx = ADXL345.ADXL345(i2c)
29     soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
30     soc.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
31     soc.bind((HOST, PORT))
32     soc.listen(100)
33     while True:
34         print("wating for request")
35         conn, address = soc.accept()
```

```
36     while True:
37         data = conn.recv(2048)
38         data = data.decode("utf-8")
39         if data == None:
40             conn.close()
41             soc.setsockopt(socket.SOL_SOCKET, socket.
42                             SO_REUSEADDR, 1)
43             break
44         if data == "send":
45             x = adx.xValue
46             y = adx.yValue
47             z~= adx.zValue
48             data = dumps((x, y, z))
49             conn.sendall(data.encode("utf-8"))
50         elif data == "close":
51             print("close")
52             conn.close()
53             soc.setsockopt(socket.SOL_SOCKET, socket.
54                             SO_REUSEADDR, 1)
55             break
```

Zdrojový kód A.1: server.py

B Zdrojový kód klienta

```
1 import socket
2 from json import loads
3
4 PORT = 8880
5 HOST = "0.0.0.0"
6 IP = "192.168.102.197"      #Default raspberry pi ip address
7
8
9 def client(queue):
10     s~= socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11     s.connect((IP, PORT))
12     while True:
13         print("Sending")
14         while True:
15             msg = "send"
16             s.sendall(msg.encode("utf-8"))
17             try:
18                 data = s.recv(1024)
19             except socket.error:
20                 print("Connection lost")
21                 continue
22             data = data.decode("utf-8")
23             data = loads(data)
24             queue.put(data)
```

Zdrojový kód B.1: client.py

C Zdrojový kód videohry

```
1 """
2 author = Alex Olivier Michaud
3 This is the main file of the game, it is used to start the game,
4     show the score and exit the game.
5 The game is a space shooter like, where you have to avoid the
6     asteroids
7 to navigate the ship it uses an accelerometer, connected to a
8     raspberry pi
9 """
10 # TODO: reunite the language of the code to english
11 import pygame
12 from sys import exit
13 from random import randint, uniform
14 from multiprocessing import Process, Queue
15 from client import client
16 import sqlite3
17
18 # Global variables for window size
19 H = 800
20 W = 1000
21
22 def score(queue):
23     """
24         This function is used to show the score in the menu of the
25             game
26         :param queue: the main queue of the game to share resources
27         :return: None
28     """
29     global H, W
30     pygame.init()
31     screen = pygame.display.set_mode((W, H))
32     # pygame.image.load("menu.png")
33     myFont = pygame.font.Font("8-BIT WONDER.ttf", 30)
34     pygame.display.update()
35     # database of score
36     conn = sqlite3.connect('score.db')
```

```

33     c = conn.cursor()
34     c.execute("CREATE TABLE IF NOT EXISTS score (score integer)")
35     c.execute("""SELECT MAX(score) FROM score""")
36     data = c.fetchone()
37     print(data)
38     conn.close()
39     if data[0] is None:
40         data = 0
41     run = True
42     while run:
43         screen.blit(pygame.image.load("img/menu.png"), (-10, -90))
44         # create text
45         text_1 = myFont.render("Best Score", True, (0, 0, 0))
46         text_2 = myFont.render("Return", True, (0, 0, 0))
47         text_score = myFont.render(str(data[0]), True, (0, 0, 0))
48         # Creating text rectangles
49         text_1_rect = text_1.get_rect(midtop=(W / 2, 300))
50         text_2_rect = text_2.get_rect(midtop=(W / 2, 400))
51         text_score_rect = text_score.get_rect(midtop=(W / 2, 350))
52         # blit text
53         screen.blit(text_1, text_1_rect)
54         screen.blit(text_2, text_2_rect)
55         screen.blit(text_score, text_score_rect)
56         for event in pygame.event.get():
57             if event.type == pygame.QUIT:
58                 run = False
59                 queue.put("exit")
60                 exit()
61             if event.type == pygame.MOUSEBUTTONDOWN:
62                 if text_2_rect.collidepoint(event.pos):
63                     run = False
64                     main_menu(queue)
65         pygame.display.flip()
66     return
67
68
69 def main_menu(queue):
70     """
71     This function is used to show the main menu of the game,
72         which is used to start the game,
73     show the score and exit the game
74     :param queue: the main queue of the game to share resources
75     :return: None
76     """
77     global H, W
78     pygame.init()

```

```

79 # font is from https://www.dafont.com/8bit-wonder.font
80 myFont = pygame.font.Font("8-BIT WONDER.ttf", 30)
81 pygame.display.update()
82 run = True
83 while run:
84     # the queue is used here, because the client is always
85     # running, so if you restart the game,
86     # the spaceship would otherwise move in a random
87     # direction, so if you empty the queue,
88     # the spaceship will not move, until you move it, and the
89     # [0,0,0] is
90     # here, so you will start the game with the spaceship in
91     # the middle of the screen
92     queue.get()
93     if queue.empty():
94         queue.put([0, 0, 0])
95     # create background
96     screen.blit(pygame.image.load("img/menu.png"), (-10, -90))
97     # create text
98     text_1 = myFont.render("Avoid Asteroids", True, (0, 0, 0))
99     text_2 = myFont.render("Press tab to start", True, (0, 0, 0))
100    text_3 = myFont.render("Check the score", True, (0, 0, 0))
101    # Creating text rectangles
102    text_1_rect = text_1.get_rect(midtop=(W / 2, 300))
103    text_2_rect = text_2.get_rect(midtop=(W / 2, 350))
104    text_3_rect = text_3.get_rect(midtop=(W / 2, 400))
105    # blit text
106    screen.blit(text_1, text_1_rect)
107    screen.blit(text_2, text_2_rect)
108    screen.blit(text_3, text_3_rect)
109    for event in pygame.event.get():
110        if event.type == pygame.QUIT:
111            run = False
112            queue.put("exit")
113            exit()
114        if event.type == pygame.KEYDOWN:
115            if event.key == pygame.K_SPACE:
116                run = False
117                App(queue)
118            if event.type == pygame.MOUSEBUTTONDOWN:
119                if text_2_rect.collidepoint(event.pos):

```

```

120                     score(queue)
121             pygame.display.flip()
122
123
124 def App(queue):
125     """
126         This function is used to show the game, and to control the
127             game
128     :param queue:
129     :return:
130     """
131
132     global H, W
133     pygame.init()
134     screen = pygame.display.set_mode((W, H))
135     pygame.display.set_caption("Space Invaders")
136     clock = pygame.time.Clock()
137     font = pygame.font.SysFont("Arial", 30)
138
139     # import images
140     sky_surface = pygame.image.load("img/1.png")
141     ship = pygame.image.load("img/ship.png").convert_alpha()
142     meteorite = pygame.image.load("img/meteorite.png").
143         convert_alpha()
144     meteorite_2 = pygame.image.load("img/meteorite_2.png").
145         convert_alpha()
146     coin = pygame.image.load("img/coin.png").convert_alpha()
147     heart = pygame.image.load("img/heart_for_game.png").
148         convert_alpha()
149     dead_heart = pygame.image.load("img/die_heart_game.png").
150         convert_alpha()
151
152     # scale images
153     ship = pygame.transform.scale(ship, (90, 90))
154     meteorite = pygame.transform.scale(meteorite, (90, 90))
155     meteorite_2 = pygame.transform.scale(meteorite_2, (90, 90))
156     coin = pygame.transform.scale(coin, (90, 90))
157     heart = pygame.transform.scale(heart, (50, 50))
158     dead_heart = pygame.transform.scale(dead_heart, (50, 50))
159
160     # coordinates
161     ship_x = 450
162     meteorite_x = randint(0, 1000)
163     meteorite_y = 0
164     meteorite_x_2 = randint(0, 1000)
165     meteorite_y_2 = 0
166     coin_x = randint(0, 1000)
167     coin_y = 0
168
169     # rectangles

```

```

164     ship_rect = ship.get_rect(midbottom=(ship_x, 800))
165     meteorite_rect = meteorite.get_rect(midbottom=(meteorite_x,
166         meteorite_y))
167     meteorite_rect_2 = meteorite_2.get_rect(midbottom=(
168         meteorite_x_2, meteorite_y_2))
169     coin_rect = coin.get_rect(midbottom=(coin_x, coin_y))
170     sky_surface_rect = sky_surface.get_rect(topleft=(0, 0))
171     heart_rect = heart.get_rect(topright=(W, 0))
172     heart_rect_2 = heart.get_rect(topright=(W - 50, 0))
173     heart_rect_3 = heart.get_rect(topright=(W - 100, 0))
174     dead_heart_rect = dead_heart.get_rect(topright=(W, 0))
175     dead_heart_rect_2 = dead_heart.get_rect(topright=(W - 50, 0))
176     dead_heart_rect_3 = dead_heart.get_rect(topright=(W - 100, 0))
177
178     # speed
179     ship_speed = 0
180     meteorite_speed = 5
181     meteorite_speed_2 = 5
182     coin_speed = 5
183     # speed in y
184     meteorite_speed_y = 0
185     meteorite_speed_y_2 = 0
186     coin_speed_y = 0
187
188     # number of coins
189     coins = 0
190     # number of lives
191     lives = 3
192
193     # text for coins
194     coins_text = font.render(f"Coins: {coins}", True, (255, 255,
195         255))
196
197     # previous position
198     old_position = 0
199
200     run = True
201     while run:
202         for event in pygame.event.get():
203             if event.type == pygame.QUIT:
204                 queue.put("exit")
205                 pygame.quit()
206                 exit()
207
208                 # spawning objects
209                 screen.blit(sky_surface, (0, 0))
210                 screen.blit(ship, ship_rect)
211                 screen.blit(meteorite, meteorite_rect)

```

```

209     screen.blit(meteorite_2, meteorite_rect_2)
210     screen.blit(coin, coin_rect)
211     screen.blit(coins_text, (10, 10))
212     screen.blit(heart, heart_rect)
213     screen.blit(heart, heart_rect_2)
214     screen.blit(heart, heart_rect_3)
215
216     # checking the number of lives and spawning dead hearts
217     if needed:
218         if lives == 2:
219             screen.blit(dead_heart, dead_heart_rect)
220         if lives == 1:
221             screen.blit(dead_heart, dead_heart_rect)
222             screen.blit(dead_heart, dead_heart_rect_2)
223         if lives == 0:
224             screen.blit(dead_heart, dead_heart_rect)
225             screen.blit(dead_heart, dead_heart_rect_2)
226             screen.blit(dead_heart, dead_heart_rect_3)
227             # updating the number of score in the database
228             conn = sqlite3.connect("score.db")
229             cursor = conn.cursor()
230             cursor.execute("CREATE TABLE IF NOT EXISTS score (
231                 score integer)")
232             cursor.execute("INSERT INTO score VALUES (?)", (coins
233                 ,))
234             conn.commit()
235             conn.close()
236             run = False
237             ###game over###
238             main_menu(queue)
239
240             if queue.empty() is False:
241                 data = queue.get()
242                 print(data)
243                 if ship_rect.left > W: # if the ship is out of the
244                     screen
245                         ship_speed -= 2
246                 elif ship_rect.left < 0: # if the ship is out of the
247                     screen
248                         ship_speed += 2
249                 elif data[2] > 0:
250                     ship_speed += 0.01 * data[2] # linear function
251                         to represent the speed of the ship
252                 elif data[2] <= 0:
253                     ship_speed += 0.01 * data[2]
254
255             # TODO: make the ship movement more smooth
256             if 0 <= ship_rect.left + ship_speed <= W - 90: # if the
257                 ship is in the screen

```

```

251     ship_rect.left += ship_speed
252     if old_position < 0 < ship_speed: # better movement,
253         if the ship is moving to the right
254             # it resets the speed, thus making the movement
255             smoother
256             ship_speed -= ship_speed
257     elif old_position > 0 > ship_speed: # same as above
258         but for the left
259             ship_speed += ship_speed
260     if ship_speed > 200: # if the speed is too high, it
261         resets it
262             ship_speed -= 100
263     else:
264         ship_speed = 0
265
266     # moving objects, random sleep, random spawn, random
267     # speed
268     if sky_surface_rect.colliderect(meteorite_rect) is False:
269         if meteorite_rect.left == 0: # if its 0 I~would get
270             a division by 0 error
271             meteorite_rect.left = 1
272         meteorite_speed_y = uniform(-1 * (1000 /
273             meteorite_rect.left), 1000 / (
274                 1000 - meteorite_rect.left)) # linear
275                     function to represent the direction of the
276                     meteorite
277         meteorite_speed = randint(3, 10) # random speed in
278         meteorite_rect.top = 0 # respawn the meteorite at
279             the top of the screen
280
281     if sky_surface_rect.colliderect(meteorite_rect_2) is
282     False:
283         if meteorite_rect_2.left == 0: # if its 0 I~would
284             get a division by 0 error
285             meteorite_rect_2.left = 1
286         meteorite_speed_y_2 = uniform(-1 * (1000 /
287             meteorite_rect_2.left), 1000 / (
288                 1000 - meteorite_rect_2.left)) # linear
289                     function to represent the direction of the
290                     meteorite
291         meteorite_speed_2 = randint(3, 10) # random speed in
292             x
293         meteorite_rect_2.top = 0 # respawn the meteorite at
294             the top of the screen
295
296     if sky_surface_rect.colliderect(coin_rect) is False:
297         if coin_rect.left == 0: # if its 0 I~would get a
298             division by 0 error
299             coin_rect.left = 1

```

```

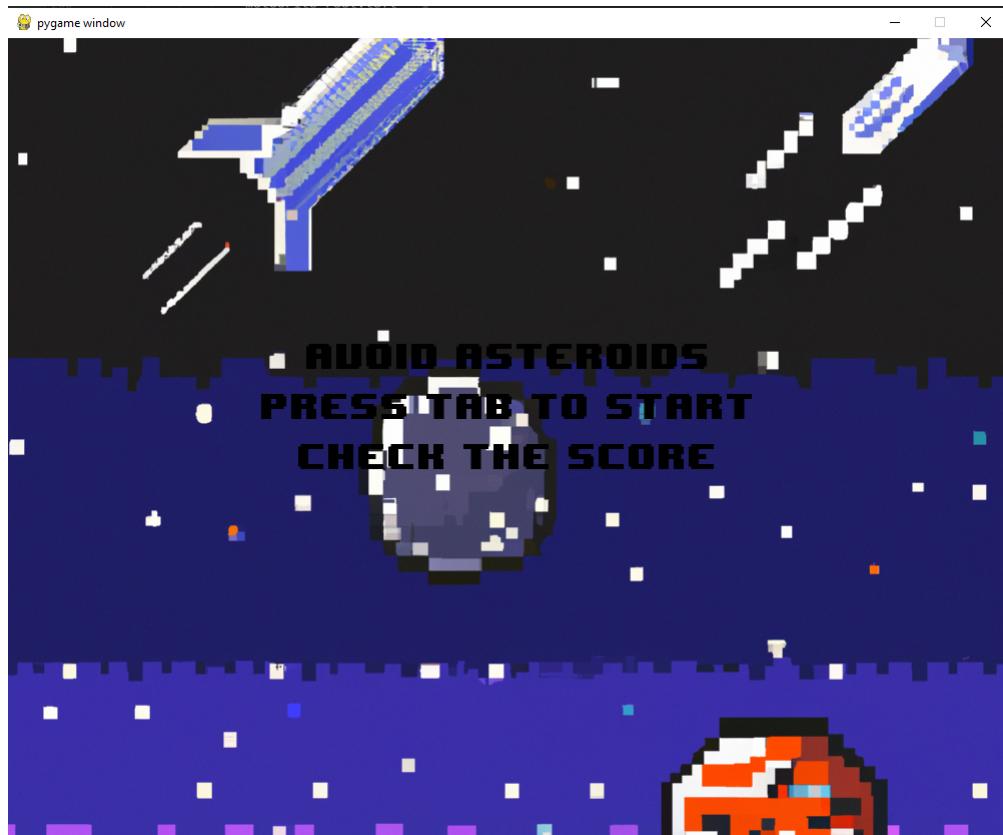
282     coin_speed_y = uniform(-1 * (1000 / coin_rect.left),
283                             1000 / (
284                                 1000 - coin_rect.left)) # linear function to
285                                 represent the direction of the meteorite
286     coin_speed = randint(3, 10) # random speed in x
287     coin_rect.top = 0 # respawn the meteorite at the top
288                                 of the screen
289
290     # moving objects
291     meteorite_rect.top += meteorite_speed
292     meteorite_rect_2.top += meteorite_speed_2
293     coin_rect.top += coin_speed
294     # moving objects in y. m,
295     meteorite_rect.left += meteorite_speed_y
296     meteorite_rect_2.left += meteorite_speed_y_2
297     coin_rect.left += coin_speed_y
298
299     # collisions with the ship, if the ship collides with the
300     # meteorite, it loses a life
301     # if the ship collides with the coin, it gets a coin and
302     # updates the score
303     if meteorite_rect.colliderect(ship_rect):
304         lives -= 1
305         meteorite_rect.top = 0
306
307     if meteorite_rect_2.colliderect(ship_rect):
308         lives -= 1
309         meteorite_rect_2.top = 0
310
311     if coin_rect.colliderect(ship_rect):
312         coins += 1
313         coins_text = font.render(f"Coins: {coins}", True,
314                                 (255, 255, 255))
315         coin_rect.top = 0
316
317         pygame.display.update()
318         clock.tick(60)
319
320
321 if __name__ == "__main__":
322     queue = Queue() # object to communicate between the
323                     processes,
324     # the queue is used for the data from the accelerometer the
325     # format is [x, y, z]
326     menu_proc = Process(target=main_menu, args=(
327         queue,)) # process for the menu, it interacts with the
328                     functions App() and Score(),
329     # so these processes are not needed
330     client_proc = Process(target=client, args=(

```

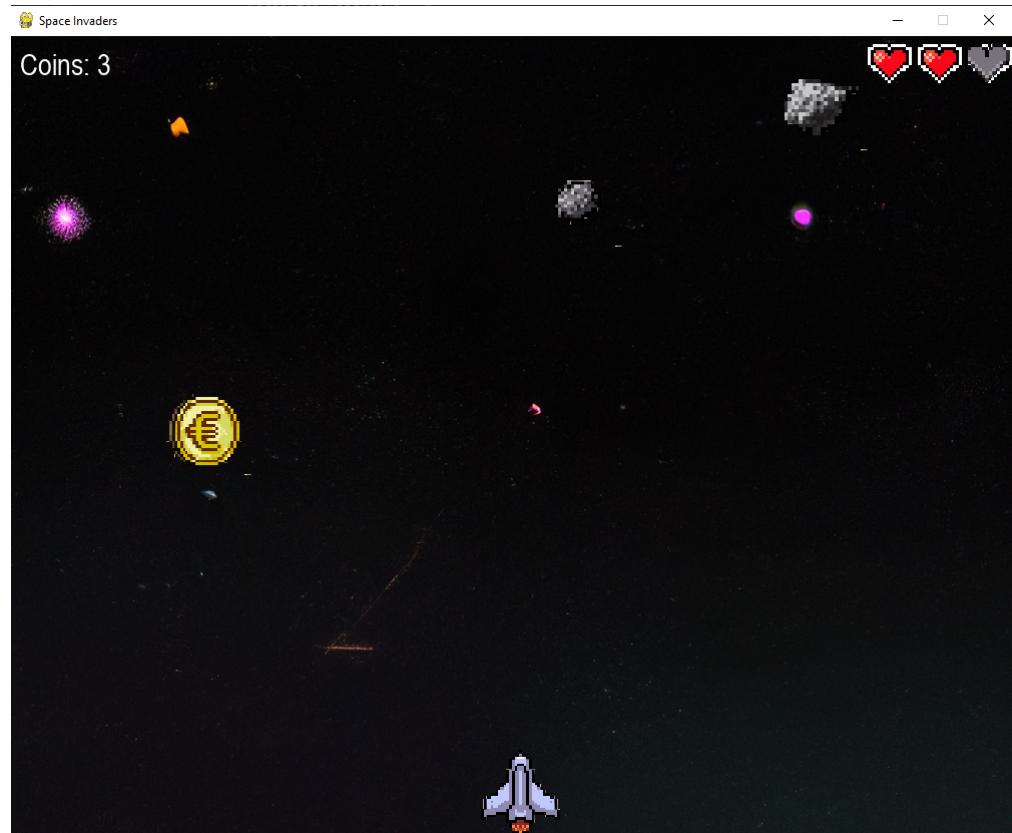
```
322         queue,)) # process for the accelerometer, it sends the
323         # data to the queue,
324     # it is imported from the client.py file
325     client_proc.start()
326     menu_proc.start()
327     client_proc.join()
328     menu_proc.join()
329     ##TODO: if the process menu_proc is closed, the client_proc
330     # processes should be closed too
```

Zdrojový kód C.1: hra.py

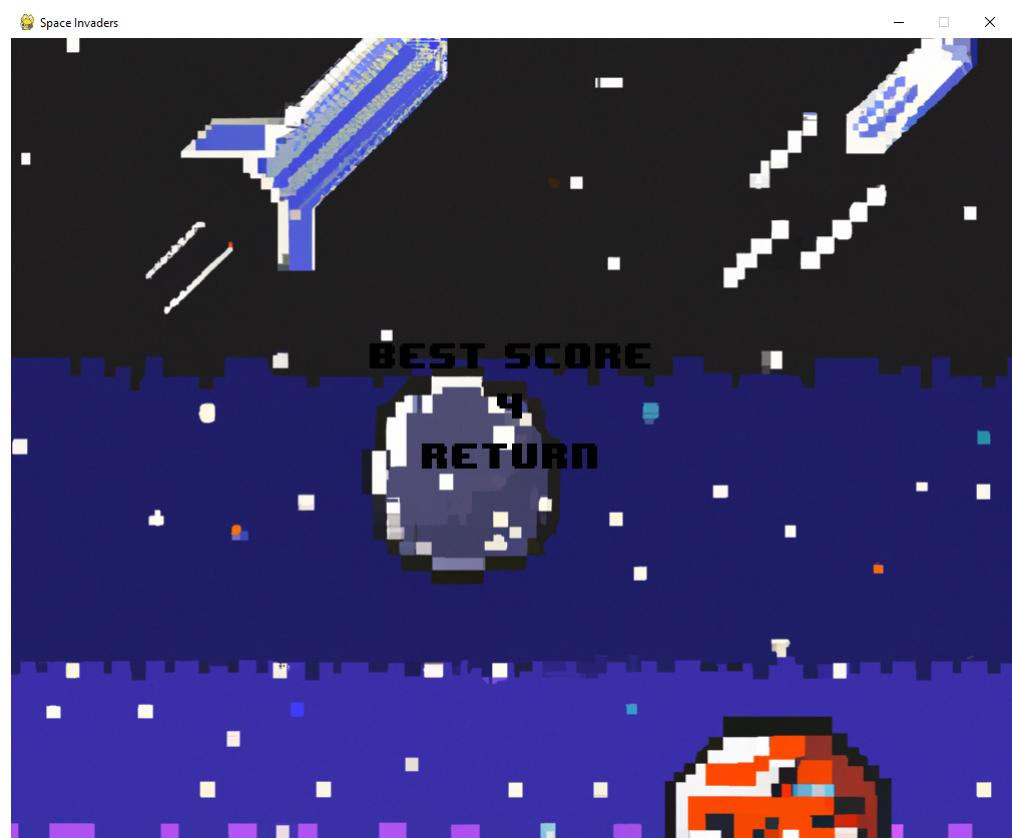
Obrázek C.1: hlavní obrazovka



Obrázek C.2: obrazovka videohry



Obrázek C.3: obrazovka skóre



Obrázek C.4: destička s Raspberry Pico W

