VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

DIGITÁLNÍ INFORMAČNÍ PANELY PRO HRU KUŽELKY

DIGITAL INFORMATION PANELS FOR THE GAME OF SKITTLES

DIPLOMOVÁ PRÁCE
MASTER' THESIS

AUTOR PRÁCE — Bc. Ludvík Szelke
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Radomil Matoušek, Ph.D.
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Ludvík Szelke**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Radomil Matoušek, Ph.D.**
Akademický rok:	2022/23

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 diplomové práce:

DIGITÁLNÍ INFORMAČNÍ PANELY PRO HRU KUŽELKY

Stručná charakteristika problematiky úkolu:

Informační panely jsou ideálními prvky pro prezentaci nejrůznějších informací či reklamních sdělení. V uvedené práci půjde o návrh systému k distribuci informací do selektivně definovaných panelů připojených pomocí ethernetu pro sport kuželky. Programově půjde primárně o back-end aplikaci, náznakem o front-end design. Předpokládají se základní zkušenosti s nějakým programovacím jazykem a pochopení principu tvorby aplikací klient-server.

Cíle diplomové práce:

- Rešerše stavu věcí v kontextu hry kuželky.
- Softwarová implementace server aplikace.
- Softwarová implementace klient aplikace.
- Praktické zkoušky zařízení a reálné otestování na dvou panelech.

Seznam doporučené literatury:

CHROBOCZEK, Martin, 2013. Grafická uživatelská rozhraní v Qt a C++: [plně kompatibilní s Qt 5]. Brno: Computer Press. ISBN 978-80-251-4124-3.

JOHN, Jiří, 2001. Bowling a kuželky. Praha: Grada. ISBN 80-247-9048-3.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku.

V Brně, dne 18. 10. 2022

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Tato diplomová práce se zabývá návrhem aplikací sloužící pro ovládání, řízení a zobrazování všech informacích potřebných pro plynulý provoz a hraní hry evropské kuželky. V první části této práce je představena historie digitálních informačních panelů a kuželkářského sportu, společně s vysvětlením pravidel pro návrh aplikací. Praktická část práce představuje počáteční vývoj aplikací. Postupně jsou představeny všechny aplikace, vysvětlena jejich struktura, a nakonec i jejich otestování.

ABSTRACT

This thesis deals with the design of applications used to control, manage and display all the information needed for the smooth operation and play of the game of European bowling. In the first part of this thesis, the history of digital dashboards and the sport of bowling is presented, along with an explanation of the rules for designing applications. The practical part of the thesis presents the initial development of the applications. All the applications are introduced in turn, their structure is explained, and finally their testing is presented.

KLÍČOVÁ SLOVA

Server, klient, front-end, back-end, kuželky.

KEYWORDS

Server, client, front-end, back-end, skittles.

ÚSTAV AUTOMATIZACE A INFORMATIKY

2023

BIBLIOGRAFICKÁ CITACE

SZELKE, Ludvík. *DIGITÁLNÍ INFORMAČNÍ PANELY PRO HRU KUŽELKY.* Brno, 2023. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/149212. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Radomil Matoušek.

PODĚKOVÁNÍ

Chtěl bych poděkovat všem, kteří mi pomohli tuto diplomovou práci realizovat, jmenovitě vedoucímu práce panu doc. Ing. Radomilu Matouškovi, Ph.D. za konzultace a rady při navrhování a pak také rodičům za jejich podporu při studiu.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestně právních důsledků.

V Brně dne 20. 5. 2023 ……

Bc. Ludvík Szelke

OBSAH

1 ÚVOD

Tato diplomová práce se zabývá tvorbou digitálních informačních panelů ke hře kuželky a obsahuje tři kapitoly. První kapitola je věnována přehledu současného stavu poznání v této problematice. Druhá kapitola prezentuje návrh a tvorbu klient server aplikací. V poslední kapitole je provedeno testování aplikací. Výsledky testování jsou shrnuty v závěru práce.

Cílem této diplomové práce je zpracování rešerše, která se zaměřuje na historický vývoj kuželek, současný stav hry z hlediska pravidel, typu disciplín. Dále jsou tu prezentovány jisté odlišnosti od bowlingu a také jakým způsobem se vyvíjelo zobrazování výsledků ve sportu.

Dále se rešerše věnuje digitalizaci a informačnímu věku společně se stručnou historií vývoje digitálních informačních panelů. V poslední části současného stavu poznání je vysvětlení jistých architektur sítí společně i s vysvětlením front-end a back-end výrazů.

Dalším cílem je vytvoření implementace server aplikace společně s klient aplikacemi. Tyto aplikace jsou zodpovědné za nastavení, ovládání a fungování hry. Této části je věnována třetí kapitola práce, kde je postupně prezentována tvorba jedné server aplikace a dalších třech klient aplikací.

Všechny tyto aplikace pak dohromady vytvářejí celý systém, díky kterému je možné hrát hru kuželky. Je zde představen back-end i front – end všech aplikací, pospáno, jak fungují i vysvětlené chování aplikací. Na konci této kapitoly se nachází vylepšení grafického prostředí aplikací a následné testování všech programů.

Součástí této práce je i příloha soubor zip, ve které jsou všechny skripty programů, jak před vylepšením grafického rozhraní, tak i potom. Součástí jsou i vlastní vytvořené obrázky piktogramů do zobrazovací aplikace.

2 PŘEHLED SOUČASNÉHO STAVU POZNÁNÍ

2.1 Historie kuželkářského sportu

Není úplně přesně známo, kdy hra kuželky vznikla. Dnešní vědci hledají odpovědi až někde v babylonské říši, což je cca 4500 let před n. l. Nicméně první stopu o hře podobné kuželkám zanechal básních Homér slavný antický spisovatel eposů Illias a Odysea přibližně ve 12. století před n. l. [2] Podle jeho zmínky se původně házelo vzduchem nebo i po zemi vhodnými kameny na vzdálený cíl.

Od doby starých Řeků se jejich hra postupně vyvíjela a šířila po evropském kontinentu. Bohužel svojí jednoduchostí a oblíbeností se z hry velmi rychle stal hazard, tím pádem bývala snaha kuželky omezovat, či dokonce zakazovat. Nejen církev ve středovku, ale i některé organizace moderních státu se o toto snažili [1]. To dokazuje i první dochovaný dokument o kuželkách, který byl zákaz této hry vydaný městskou radou v Jihlavě roku 1261 a poté také později ve 14. století, což dokazuje, že v naší zemi byly kuželky také velmi oblíbené jako hra i jako hazard [2]. Dále hru do kuželek zakazoval římský císař a český král Karel IV. V roce 1841 přibližně o 500 let později začal platit zákaz hry do devíti kuželek v americkém státě Connecticut. Obyvatelům se toto nařízení nelíbilo, a tak k deváté kuželce přidali desátou, poupravili pravidla a dali tak vzniknout nové hře bowlingu [1].

Co to chlapci hrajou? — kolik je k tomu kuželek třeba? Spočítej je! — Kolik je tady koulí? — Kdyby všechny kuželky byly postaveny a jedna se porazila, kolik by jich ještě stálo? — kdyby se 2 kuželky porazily? — kdyby 3 kuželky atd. — Zakrej prstem ty, které zůstaly stát; kolik tady leží kuželek poražených? — kolik jich tady ještě stojí? — Kdyby ležely 3, kolik by jich stálo? — kdyby ležely 4? atd. — Udělej tolik čárek, co je tady kuželek! — Spočítej je! — Přidej 1 čárku, která znamená kouli! — Kolik máš tady čárek? — Smaž 1 čárku a řekni: 1 od 10, zbyde? — anebo: 10 bez 1 jest . . .? Smaž 2 čárky, 3 čárky atd. — Udělej ještě jednou 10 čárek! — Teď smazuj po dvou čárkách, a říkej 2 od 10, zbyde? — anebo 10 bez 2 jest . . .? atd. — Spočítej také kuželky a koule dohromady; kolik to bude kusů?

Slušná hra je dobrá — tělu k sesilení;
Ale hrát jen smíme,— až je po učení.

Obr. 1: Zmínka kuželek v učebnici matematiky [1]

Později už nešlo o hazard, ale o sport, konkrétně kuželkářský sport. Historicky byl sport rozdělen na dvě odvětví, kuželky a bowling. V Evropě se za sport považovaly spíše kuželky, kdy se počet kuželek ustáli na devíti, v americké verzi to bylo deset kuželek. [1] První kuželkářská sportovní organizace v Evropě vznikla v roce 1885 v Drážďanech. V roce 1926 byla založena mezinárodní kuželkářská organizace IBA (International Bowling Association) ve Stockholmu, ve které naší zemi zastupoval až do roku 1931 německý svaz.

Toto vedlo ke snahám vytvořit vlastní kuželkářskou organizaci, a tak dne 11. ledna 1937 vznikla Asociace československého sportu kuželkářského dále jen ASČK. Již v dubnu téhož roku měla organizace 620 členů z 31 klubů. Organizace ASČK však zanikla na začátku 2. světové války, a proto vznikl Svaz českých kuželkářů (SČK) a SAZK na Slovensku. S koncem války opět obě organizace zanikly. S nástupem komunistického režimu byla ustanovená nová vrcholná organizace Československá asociace kuželkářského sportu (ČAKS). Ta i díky několikanásobnému přejmenování přežila režim až do vzniku samostatné České republiky.

V roce 2001 došlo k reorganizaci, právě díky nastupujícímu konkurenčnímu bowlingu a vznikl nový vrcholný orgán Česká kuželkářská a bowlingová federace, pod kterou kuželkářský sport spravuje Česká kuželkářská asociace a bowling řídí Česká bowlingová asociace.

Během těchto let se staly kuželky významným sportem naší země a naši kuželkáři získali hodně sportovních úspěchů. Mezi lety 1955 a 2021 získali 46 zlatých medailí z mistrovství světa a nespočet dalších úspěchů [2].

Obr. 2: Znak České kuželkářské asicoace [3]

2.2 Pravidla kuželkářského sportu

2.2.1 Pravidla kuželek

Pro někoho, kdo nezná kuželky ve stručnosti, je velice důležité, aby koule při hodu byla vypuštěna na náhozovou desku a přímou cestou porazila co nejvíce kuželek. Délka dráhy je stanovena 19,5 metrů, kde na konci jsou v pravidelných rozestupech postaveny kuželky do kosočtverce, konkrétně 9 kuželek. Každý hráč hází určitý počet hodů za daný čas a výsledkem je součet všech shozených kuželek. Více detailněji je to stanoveno v pravidlech kuželkářského sportu. [1]

Pravidla kuželkářského sportu jsou rozdělena na 5 částí. Část první týkající se všeobecných ustanovení, část druhá o základních pravidlech hry, část třetí o povinnostech a právech účastníků soutěží, dopingu a závěrečná ustanovení. Pouze druhá část je pro tuto diplomovou práci důležitá, ale zkráceně v první části je jasně ustanoveno, jakým způsobem probíhají soutěže, co se organizace, vybavení jak kuželny, tak hráčů týče. Z názvu zbylých částí už vyplívá, co obsahují.

Pro různý počet hodů je i různá hrací doba Po překročení hrací doby se už hráči nepočítají žádné další body. Důležitý je i chybný hod, který nastává pokaždé pokud minete kuželky nebo při opakovaném přešlapu.

Přešlap je v kuželkách velice specifický. První přešlap je dovolen, při čemž se vám hod i spadlé kuželky počítají, ale rozsvítí se varování o přešlapu a každý další se už nepočítá a je brán jako chybný hod. [4]

Dalším důležitým článkem jsou přestupky. Je jich celá řada například, položení koule mimo náhozovou desku, úmyslná nebo nevědomá hra do nepřipraveného automatického stavěče kuželek nebo nesportovní chování. Za každý přestupek vás může rozhodčí napomenout nebo vám i udělit žlutou katru. Dvě žluté potom znamenají červenou a vyloučení jako ve fotbale [4].

Ve všech disciplínách představuje každá regulérně poražená kuželka jeden bod. Výkon hráče je dán součtem všech dosažených bodů. Ne každá shozená kuželka je ovšem bod pro hráče. Záleží na mnoha faktorech, které se musí vzít v úvahu a rozlišovat.

Kuželky mohou být poraženy řádným způsobem:

- Přímo koulí.
- Nárazem jedné kuželky do druhé.
- Závěsnou lankem jiné kuželky.
- Kuželky mohou být poraženy neregulérním způsobem:
- Kuželky poražené koulí odraženou od bočního mantinelu.
- Kuželky poražené koulí od zadní stěny.
- Kuželky poražené v době, kdy automatický systém detekce spadlých kuželek nebyl připraven ke hře.
- Kuželky poražené částmi automatického stavěče na kuželky.

Hráč se při hodu nesmí dotknout země rukou nebo kolenem. Koule se musí valit po zemi od začátku hrací plochy. Koule nesmí narazit do mantinelů nebo žlábků. [4]

Obr. 3: Strike [5]

2.2.2 Disciplíny

V kuželkářském sportu se používají tyto disciplíny:

- **Plné**
  - hráč hází každým hodem do plného stavu devíti kuželek;
- **Dorážková**
  - hráč po prvním hodu do plných doráží dalšími hody kuželky, které zůstaly stát, po jejich doražení se celý způsob opakuje;
- **Sdružená**
  - hráč hází polovinu hodů do plných a polovinu hodů v dorážkové (dále jen „hs“).

Sdružená disciplína se dělí:

- 200 hs se hraje přes čtyři dráhy. Hráč hraje na každé dráze nejprve 25 hodů do plných a pak 25 hodů do dorážkových.
- 120 hs se hraje přes čtyři dráhy. Hráč hraje na každé dráze nejprve 15 hodů do plných a pak 15 hodů do dodorážkových.
- 100 hs se hraje přes dvě dráhy. Hráč hraje na každé dráze nejprve 25 hodů do plných a pak 25 hodů do dorážkových.
- 60 hs se hraje přes dvě dráhy. Hráč hraje na každé dráze nejprve 15 hodů do plných a pak 15 hodů do dorážkových.
- 40 hs se hraje na jedné dráze. Hráč hraje na každé dráze nejprve 20 hodů do plných a pak 20 hodů do dorážkových.
- 30 hs se hraje na jedné dráze. Hráč hraje na každé dráze nejprve 15 hodů do plných a pak 15 hodů do dorážkových.
- 20 hs se hraje na jedné dráze. Hráč hraje na každé dráze nejprve 10 hodů do plných a pak 10 hodů do dorážkových.

Jednotlivé sdružené disciplíny lze vhodně kombinovat, disciplíny hrané přes čtyři dráhy lze odehrát i na dvou dráhách [4].

2.3 Rozdíl mezi bowlingem a kuželkami

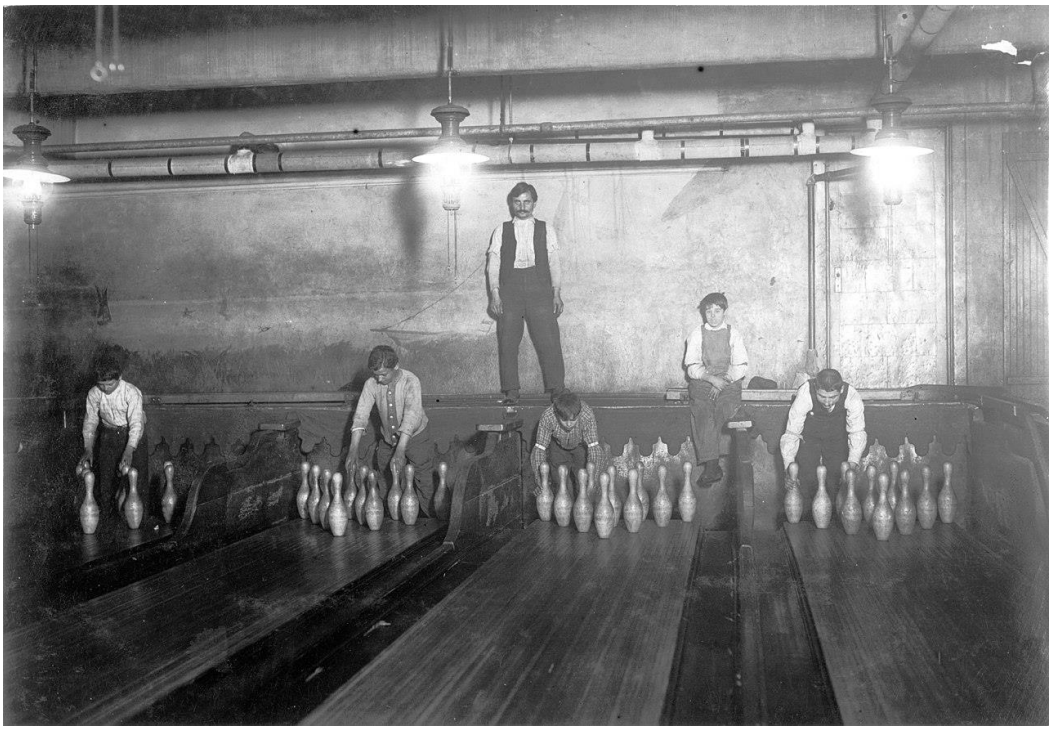
Bowling a kuželky mají odlišná pravidla a systém skórování. V bowlingu se používá skórování na základě bodů, kde každý hod má své hodnoty. Kuželky se obvykle počítají počtem poražených kuželek, kde každá poražená kuželka představuje jeden bod [6]. Také rozestavení kuželek je jiné, v bowlingu je do trojúhelníku 10 kuželek a v kuželkách do kosočtverce 9 kuželek.

V bowlingu se používají speciální kuželky, které mají standardní velikost a tvar. Mají výšku asi 38 cm a váží kolem 1,6 kg. Naopak v kuželkách se používají kuželky různých velikostí a tvarů, které se mohou lišit podle regionálních pravidel.

V bowlingu má hráč obvykle několik hodů na jedno kolo, zatímco v kuželkách se obvykle hodí jen jednou [7]. Herní doba v bowlingu je také delší, s hráči, kteří střídavě házejí na stejnou dráhu, zatímco v kuželkách se jednotliví hráči střídají na různých drahách.

2.4 Historie zobrazování výsledků ve sportu

Ukazování výsledků ve sportu má dlouhou historii, která sahá až do starověku. Jedním z prvních známých příkladů jsou olympijské hry, které se poprvé konaly ve starověkém Řecku v roce 776 před naším letopočtem. Na těchto prvních hrách byli vítězové odměňováni olivovými věnci, zatímco poražení nedostávali nic [8].



Obr. 4: Pinboys – obsluha kuželkových her [16]

Jedna z prvních známých zpráv o používání výsledkové tabule přišla v roce 1895 na fotbalovém zápase na Harvardu. Skóre bylo tvořeno z čísel na kusu dřeva, zavěšených na hřebíku a měněných ručně osobou sledující hru. Tyto typy tabulek byly převládající během 20. a 30. let 20. století s přidáním hodin a příležitostnou reklamou [9].

Postupem času se způsob zobrazování výsledků ve sportu vyvíjel, ve 20. století byly zavedeny výsledkové tabule a elektronické displeje. Například v kuželkách byly výsledkové tabule původně aktualizovány ručně osobami známými jako "pin boys". S příchodem automatizace a elektronických bodovacích systémů se však tento proces stal efektivnějším a přesnějším.

Dnes se ve sportech, jako je bowling a kuželky, mohou výsledky v reálném čase zobrazovat na obrazovkách pro hráče i diváky. Na těchto displejích se často zobrazují podrobné informace o skóre každého hráče a také statistiky, jako je například počet striků.

Kromě elektronických displejů se moderní sporty při zobrazování výsledků ve velké míře spoléhají také na analýzu dat a online platformy. Patří sem nástroje pro sledování a analýzu údajů o výkonech, stejně jako webové stránky a mobilní aplikace, které umožňují fanouškům sledovat živé výsledky a sledovat své oblíbené týmy a hráče.

2.5 Historie digitálních panelů

Digitální informační panely, známé také jako *digital signage*, mají dlouhou a fascinující historii, která sahá až do počátků výpočetní techniky. Tyto panely prošly vývojem od jednoduchých monochromatických displejů až po dnešní interaktivní obrazovky s vysokým rozlišením, které mohou zobrazovat širokou škálu obsahu.

Historie digitálních informačních panelů sahá až do 70. let 20. století, kdy byly vyvinuty první elektronické displeje. Tyto displeje sloužily především k zobrazování základního textu a jednoduché grafiky a běžně se používaly na letištích a nádražích k zobrazování jízdních řádů a informací o odjezdu vlaků.

Obr. 5: Digitální panelů společnosti NYCTrainSign [17]

V 80. a 90. letech 20. století se digitální informační panely začaly ve větší míře používat v komerčním prostředí. Částečně to bylo způsobeno vývojem pokročilejších zobrazovacích technologií, jako jsou plazmové a LCD obrazovky, které umožnily zobrazovat složitější obsah [10].

Na počátku 20. století vedl nástup internetu a vývoj webových systémů pro správu obsahu (CMS) k revoluci v oblasti digitálních informačních tabulí. Díky webovému systému CMS bylo možné vzdáleně spravovat digitální displeje odkudkoli na světě, což usnadnilo a zefektivnilo nasazení sítí digital signage na více místech.

Dnes se digitální informační panely používají v nejrůznějších prostředích, od maloobchodních prodejen a nákupních center až po letiště, nádraží a sportovní stadiony. Používají se k zobrazování reklamy, informací o cestě, zpráv a aktuálního počasí a řady dalšího obsahu.

2.6 Digitalizace a informační věk

Digitalizace, tedy proces převodu analogových informací do digitální podoby, je hlavní hnací silou informačního věku [10]. Právě tento proces vedl k významným změnám ve způsobu života, práce a komunikace a otevřel nové příležitosti pro podniky i jednotlivce.

Za počátek informačního věku se obvykle považuje konec 20. století, kdy se rozšířily osobní počítače a internet. [11] Od té doby se množství vytvářených a sdílených digitálních informací exponenciálně zvýšilo a dnes žijeme ve světě, kde jsou digitální informace všudypřítomné.

Jednou z hlavních výhod digitalizace je, že umožňuje snadnější přístup k informacím a jejich sdílení. To má zásadní vliv na způsob komunikace v osobním i profesním životě. Podnikům také usnadnila shromažďování a analýzu dat, která lze využít ke zlepšení produktů, služeb a zkušeností zákazníků.

Obr. 6: Digitalizace [15]

Další výhodou digitalizace je, že umožnila vznik nových obchodních modelů a odvětví, například vzestup elektronického obchodování a online tržišť změnil maloobchodní průmysl, zatímco rozvoj digitálních streamovacích služeb narušil tradiční mediální a zábavní průmysl [12].

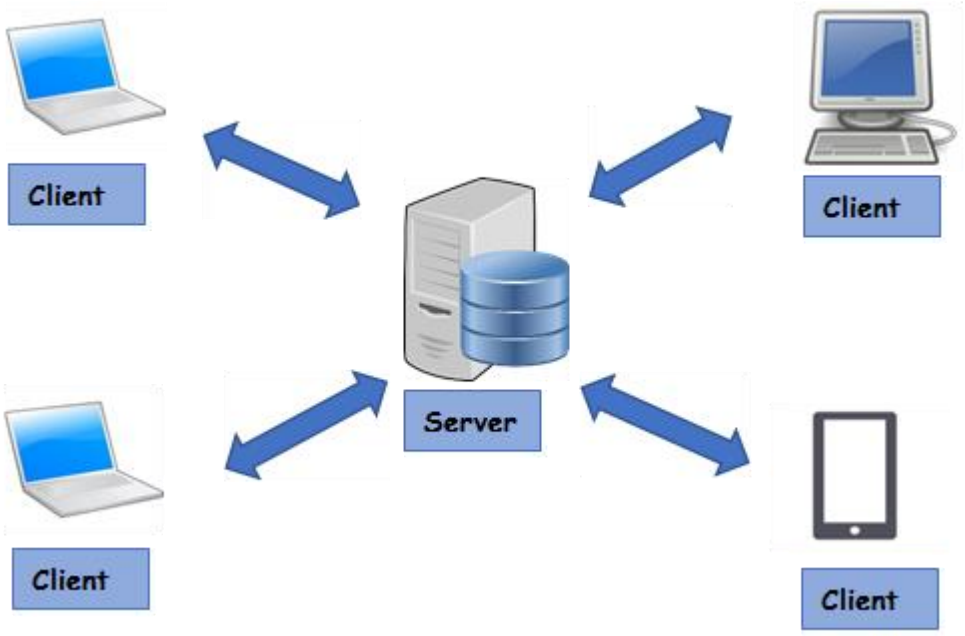
Digitalizace však přináší i některé výzvy, například obavy o soukromí a bezpečnost údajů. S tím, jak se na internetu shromažďuje a sdílí stále více osobních údajů, roste potřeba tyto údaje chránit a zajistit, aby nebyly zneužity.

2.7 Architektura server – klient

Architektura server-klient, známá také jako architektura klient-server, je model, ve kterém klientská zařízení a serverová zařízení komunikují prostřednictvím sítě. V tomto modelu klient posílá požadavky na server, který pak odpovídá požadovanými informacemi nebo službami [18].

Základními součástmi architektury server-klient jsou klientská zařízení, serverová zařízení a síť, která je spojuje. Klientským zařízením může být počítač, chytrý telefon,

tablet nebo jakékoli jiné zařízení, které požaduje informace nebo služby od serveru. Serverovým zařízením může být vyhrazený server nebo virtuální server, na kterém jsou umístěny aplikace nebo služby přístupné klientům prostřednictvím sítě [19].
Mezi výhody architektury server-klient patří lepší zabezpečení, škálovatelnost a snadnější správa zdrojů. V tomto modelu je server odpovědný za správu zdrojů, jako jsou databáze, soubory a aplikace, zatímco klient je odpovědný za interakci s těmito zdroji [20].



Obr. 7: Architektura klient – server [20]

2.8 Back – end

Vývoj back-endových aplikací zahrnuje tvorbu komponent webových stránek nebo aplikací na straně serveru. Tyto komponenty zahrnují server, databázi a aplikační logiku, které umožňují fungování webové stránky nebo aplikace [21].

Základními jazyky používanými při vývoji back-end aplikací jsou Java, Python, Ruby a PHP. Tyto jazyky se používají k vytváření skriptů a aplikací na straně serveru, které zpracovávají data, spravují databáze a aplikační logiku [22].

Mezi klíčové aspekty vývoje back-endových aplikací patří škálovatelnost, bezpečnost a výkon. Vývojáři musí zajistit, aby aplikace zvládala velké množství dat a provozu, byla zabezpečená proti vnějším hrozbám a optimalizovaná z hlediska výkonu [23].

2.9 Front – end

Front-end design je proces vytváření uživatelského rozhraní a prostředí webových stránek nebo aplikací. Zahrnuje použití různých nástrojů a technologií k vytvoření vizuálně přitažlivého a funkčního rozhraní, které uživatelům umožňuje interakci s webem nebo aplikací.

Mezi základní součásti front-end designu patří HTML, CSS a JavaScript. Jazyk HTML se používá k vytvoření struktury stránky, zatímco CSS slouží ke stylizaci stránky a jejímu vizuálnímu zatraktivnění. JavaScript se používá k přidání interaktivity a funkčnosti stránky [22] .

Mezi důležité aspekty návrhu front-endu patří uživatelský komfort, přístupnost a výkon. Návrháři musí zajistit, aby se webové stránky nebo aplikace snadno používaly a navigovaly, byly přístupné všem uživatelům a rychle se načítaly [23] .

3 VLASTNÍ ŘEŠENÍ

3.1 Návrh architektury systému

V současném informačním prostředí je časté potřebovat spolupráci mezi různými aplikacemi a systémy. V případě systému hry evropské kuželky, který zahrnuje čtyři aplikace (server, ovládací aplikace, PLC aplikace a zobrazovací aplikace) je důležité navrhnout vhodnou architekturu, která umožní efektivní komunikaci a spolupráci mezi těmito aplikacemi.

3.1.1 Jednotlivé návrhy aplikací architektury:

Server

- Server bude představovat centrální uzel, který zajišťuje komunikaci mezi všemi aplikacemi.
- Bude obstarávat příjem a odesílání zpráv mezi aplikacemi a zajišťovat jejich správné směřování.
- Serverová aplikace bude zodpovědná za zpracování dat a rozhodování o dalších akcích na základě obsahu přijatých zpráv.

Ovládací aplikace

- Ovládací aplikace bude určena pro ovládání zařízení v systému hry evropské kuželky.
- Bude komunikovat se serverovou aplikací pomocí zpráv obsahujících instrukce pro ovládání a nastavení zařízení.

PLC aplikace

- PLC aplikace bude sloužit jako simulace komunikace mezi serverem a PLC řízením automatického stavěče kuželek.
- Bude sledovat stav automatu a posílat informace o automatu serveru.
- Po vyzvání zašle žádané informace o stavu automatu.
- Zobrazovací aplikace
- Zobrazovací aplikace bude sloužit k vizualizaci dat a stavu systému.

3.1.2 Návrh komunikace mezi aplikacemi v systému

Naše systémové aplikace by měly být navrženy tak, aby spolupracovaly a vyměňovaly si informace prostřednictvím serveru. Zde je podrobnější popis komunikace a mechanismů:

Ovládací aplikace – Server:

- Ovládací aplikace by měla komunikovat se serverem prostřednictvím asynchronního přístupu.
- Při odesílání instrukcí nebude blokovat hlavní vlákno a nepočká na okamžitou odpověď.
- Místo toho ovládací aplikace bude vytvářet asynchronní požadavky na server a bude pokračovat v dalším provádění kódu.
- Server by měl přijímat tyto asynchronní požadavky a zpracovávat je na pozadí bez blokování jiných operací.

Server – PLC:

- Komunikace mezi serverem a PLC aplikací by měla být asynchronní, což znamená, že aplikace bude stále čekat na zprávu od serveru.
- Při předávání instrukcí od serveru by se mělo vytvořit asynchronní volání, které neblokuje běh serverového vlákna a čeká na zprávu od serveru.
- Tím bude zajištěno správné vykonání požadovaných operací na PLC a následné předání výsledků zpět serveru.

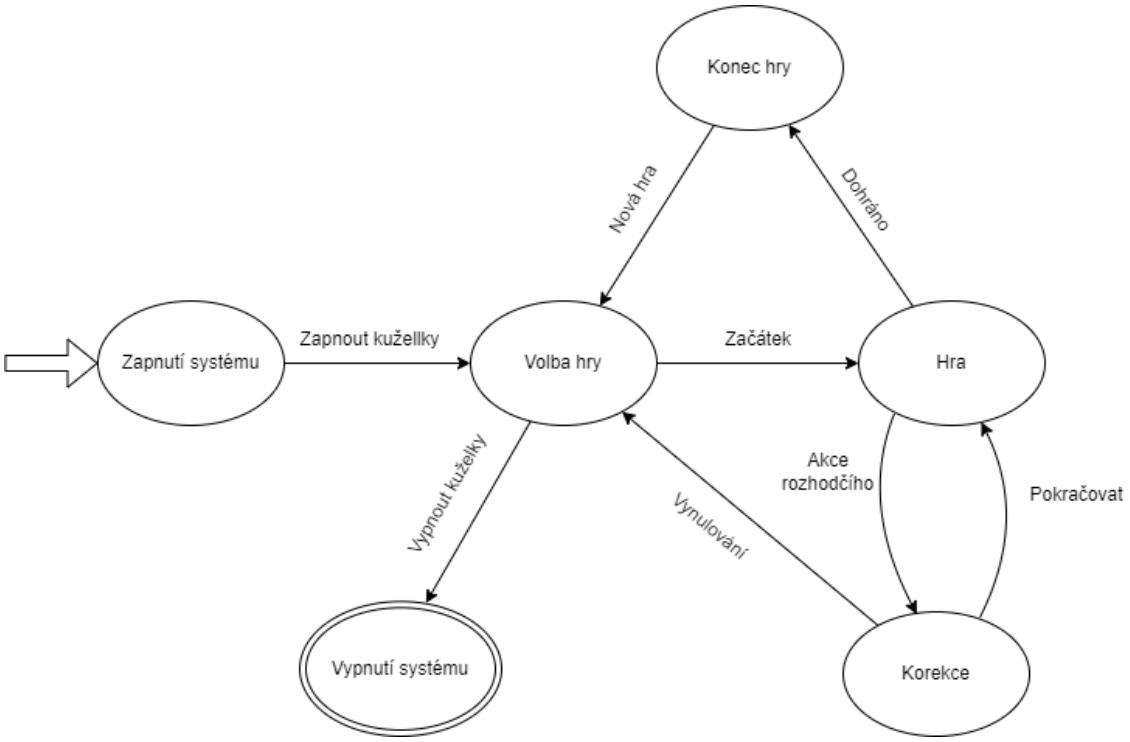
PLC – Server:

- Komunikace mezi PLC aplikací a serverem by nejspíše měla být asynchronní kvůli rychlejšímu a neblokujícímu přijmu informací, ale vzhledem k tomu že my budeme pouze simulovat její chování můžeme použít synchronní komunikaci.
- PLC aplikace po vyzvání příkazem odešle data o stavu automatického stavěče kuželek na server bez nutnosti okamžité odpovědi.
- Server bude přijímat tato synchronní data, zpracuje je a bude pokračovat v dalších operacích.

Server – Zobrazovací aplikace:

- Komunikace mezi serverem a zobrazovací aplikací bude synchronní.
- Server po obdržení příkazu k odeslání dat o stavu systému do zobrazovací aplikace, pošle data
- Zobrazovací aplikace bude přijímat asynchronním způsobem tyto zprávy a aktualizovat své rozhraní pro uživatele.

3.1.3 Stavový diagram systému



Obr. 8: Stavový diagram hry

Na obrázku je návrh stavového automatu ke kuželkám. V tomto případě máme několik stavů. Nejprve se po startu systému automat dostane do stavu Volby hry. V tomto stavu se mohou navolit různé údaje o typu hry, kuželkách atd. Dále se automat může dostat příkazem vypnout kuželky do stavu Vypnutí systému. Nebo příkazem začátek do stavu Hra. Zde bude probíhat algoritmus hry kuželek a po jeho ukončení se příkazem dohráno automat dostane do stavu Konec hry, kde by mohlo být vyhlášení výsledků. Z tohoto stavu se pak dá příkazem nová hra dostat zpět do stavu Volby hry. Pokud bude algoritmus stále probíhat může hru přerušit příkaz akce rozhodčího a tím se automat dostane do stavu Korekce. Ve stavu Korekce bude možné upravit hodnoty hry, jako jsou například počet bodů počet chyb atd. a také bude možné provést dva příkazy. Příkazem start se automat vrátí do stavu Hra, kde bude pokračovat v hraní a příkazem vynulování se bude moci vrátit do stavu Volba hry, například pokud by bylo potřeba vypnout systém.

3.2 Serverová aplikace

Tato kapitola popisuje tvorbu serverové aplikace a obsahuje výpisy ze souboru `app.py`, který se nachází v příloze A. Serverová aplikace je nedílnou součástí digitálního panelového systému používaného při hře evropské kuželky. Slouží jako centrální uzel pro komunikaci mezi třemi klientskými aplikacemi: ovládací aplikací, aplikací displeje a aplikací PLC. Tyto aplikace dohromady tvoří systém, který by měl vylepšovat tradiční hru evropské kuželky.

3.2.1 Popis aplikace

Server aplikace používá knihovny pro komunikaci socket a pro převádění libovolných objektů do seriových dat JSON.

Kód definuje třídu serveru, která zajišťuje komunikaci s klienty. Inicializuje tři sokety: `ovladaci_socket`, `zobrazovaci_socket` a `plc_socket`. Tyto sokety slouží k přijímání dat od klientů a odesílání odpovědí zpět.

Třída serveru má metody pro příjem dat z `ovladaci_socket` a `plc_socket` a pro odesílání dat do `zobrazovaci_socket` a `plc_socket`. Tyto metody používají funkci `receive_from_client()` pro příjem dat od klientů a funkci `sendall()` pro odesílání dat klientům.

Součástí kódu je také třída `HerniPrikazy`, která se stará o zpracování přijatých dat. Obsahuje metody pro zpracování různých typů dat, jako jsou herní parametry, příkazy a informace o hráči. Metoda `process_data()` analyzuje přijatá data JSON a podle toho aktualizuje příslušné proměnné.

V této třídě je také naimplementován herní algoritmus pomocí nekonečné smyčky `while`. Třída serveru obsahuje metodu `overeni_prikazu()`, která průběžně přijímá data z `ovladaci_socket` a zpracovává je pomocí třídy `HerniPrikazy`. Kontroluje také, zda není zadán příkaz stop nebo jiné příkazy pro provedení konkrétních akcí.

Celkově serverová aplikace nastavuje sokety pro komunikaci, přijímá data od klientů, zpracovává je a odesílá příslušné odpovědi. Poskytuje rámec pro zpracování různých typů příkazů a správu stavu hry.

3.2.2 Implementace server aplikace

Serverová aplikace je implementována pomocí programovacího jazyka Python a pro síťovou komunikaci se spoléhá na modul `socket`. Zde je příklad inicializace serveru:

```
import socket
import json
import time

HOST = '127.0.0.1'

OVLADACI_PORT = 5000
PLC_PORT = 5001
ZOBRAZOVACI_PORT = 5002

# Inicializace atributů
typ_hry = None
pocet_hodu = 0
hody_zbytek = 0
prumerny_pocet_kuzelek = 0
cas = None
typ_kuzelek = None
jmeno_hrace = None
jmeno_klubu = None
vlastni_postaveni = None
prikaz = None
kuzelky = None
pozice_kuzelek = None
rychlost = 0
pocet_chyb = 0
soucet_kuzelek = 0
soucet_kuzelek_celkem = 0
celkovy_bodovy_zisk = 0
faul = 0
opak_faul =0
```

Výpis 1: Inicializace serveru

Ve výše uvedeném úryvku kódu importujeme knihovny `socket`, `json` a `time`. Server má nastavenou proměnnou `HOST` na hostitelské jméno nebo IP adresu počítače, na kterém běží server.

Definuje samostatná čísla portů pro různé typy spojení: `OVLADACI_PORT` pro připojení ovládací aplikace, `ZOBRAZOVACI_PORT` pro připojení k zobrazovací aplikaci a `PLC_PORT` pro připojení k PLC aplikaci.

Server také inicializuje některé proměnné pro sledování stavu, například „`počet_kuzelky`" (kuželeky), „`rychlost`" (rychlost) a „`pocet_chyb`" (počet chyb).

3.2.3 Zpracování dat od klientů

Serverová aplikace přijímá data z klientských aplikací a odpovídajícím způsobem je zpracovává. Třída `HerniPrikazy` obsahuje metody `handle_data` a `process_data`, které zpracovávají příchozí data. Zde je příklad, jak probíhá zpracování dat:

```
def handle_data(self, data):
        parsed_data = json.loads(data)
        self.process_data(parsed_data)


    def process_data(self, parsed_data):
        global faul, opak_faul, typ_hry, pocet_hodu, hody_zbytek, prumerny_po
jmeno_hrace, jmeno_klubu, trenink_cas, trenink_pocet_hodu, trenink_typ_hodu,
pozice_kuzelek, rychlost, pocet_chyb, soucet_kuzelek, soucet_kuzelek_celkem,

        if isinstance(parsed_data, dict):
            print(parsed_data)
            if "typ_hry" in parsed_data:
                typ_hry = parsed_data["typ_hry"]

                if typ_hry == "zakladni":
                    if "jmeno_hrace" in parsed_data:
                        jmeno_hrace = parsed_data["jmeno_hrace"]
                    else:
                        print("Chybějící informace o jméně hráče.")
                        return

                    if "jmeno_klubu" in parsed_data:
                        jmeno_klubu = parsed_data["jmeno_klubu"]
                    else:
                        print("Chybějící informace o jméně klubu.")
                        return
                    if "cas" in parsed_data:
                        cas = float(parsed_data["cas"])
                    else:
                        print("Chybějící informace o čase.")
                        return
```

Výpis 2: Zpracování dat

V metodě `process_data` nejdříve nastavuje pomocí příkazu `global` herní atributy, aby byly globální pro celou aplikaci. Pomocí metod `handle_data` a `process_data` zpracovává serverová aplikace různé typy dat přijatých od klientů. Kontroluje typ dat a na základě obsahu dat provádí konkrétní akce. Například pokud data obsahují pole `"kuzelky"`, serverová aplikace aktualizuje atribut `kuzelky` přijatou hodnotou. Podobně zpracovává i další typy dat, například `"rychlost" (rychlost)` a `"pocet_chyb" (počet chyb)`.

```
elif "prikaz" in parsed_data:
                prikaz = parsed_data["prikaz"]

                if prikaz == "oprava":
                    if "pocet_hodu" in parsed_data:
                        hody_zbytek = parsed_data["pocet_hodu"]
                    else:
                        print("Chybějící informace o počtu hodů.")
                        return

                    if "cas" in parsed_data:
                        cas = float(parsed_data["cas"])
                    else:
                        print("Chybějící informace o čase.")
                        return
                    # ...
                if prikaz == "karta":
                    print(f"Přijat příkaz: Karta")

                elif prikaz == "start":
                    print(f"Přijat příkaz: Start")

                elif prikaz == "vynulovat":
                    print(f"Přijat příkaz: Vynulovat")

                elif prikaz == "stop":
                    print(f"Přijat příkaz: Stop")

                elif prikaz == "nepocitej_hod":
                    print(f"Přijat příkaz: Nepočítej kuželky")
```

Výpis 3: Ukázka části metody rozlišování zprávy

Také jsou tu možnosti pro rozlišení příkazu od ovládací aplikace, kdy až na opravu stačí pouze potvrzení o přijetí příkazu. V příkazu oprava už se dopředu počítá, že spolu s ním se obdrží i určitá data pro opravení vnitřních atributů.

Po zpracování dat serverová aplikace aktualizuje stav hry a na základě zpracovaných dat provede všechny potřebné akce. Tyto akce mohou zahrnovat aktualizaci skóre, sledování počtu hodů, výpočet průměru a další.

3.2.4 Zpracování komunikace s klientem

Serverová aplikace používá ke komunikaci s klientskými aplikacemi sokety. Pro každého klienta nastaví samostatné sokety a naslouchá příchozím spojením. Zde je příklad, jak server zpracovává komunikaci s klienty:

Při metodě `start_server` nejprve označuje proměnné jako globální a poté vytváří serverová aplikace samostatné sokety pro každou klientskou aplikaci (ovládání, zobrazování a PLC). Váže sokety k zadanému hostiteli a portu a pak naslouchá příchozím připojením.

```
def start_server(self):
        global faul, opak_faul,typ_hry, pocet_hodu, hody_zbytek, prumerny_pocet_kuzelek, cas,
jmeno_hrace, jmeno_klubu, trenink_cas, trenink_pocet_hodu, trenink_typ_hodu, vlastni_postaveni
pozice_kuzelek, rychlost, pocet_chyb, soucet_kuzelek, soucet_kuzelek_celkem, celkovy_bodovy_zi

        prikazy = HerniPrikazy()

        self.ovladaci_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.plc_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.zobrazovaci_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        self.ovladaci_socket.bind((HOST, OVLADACI_PORT))
        self.ovladaci_socket.listen()


        self.plc_socket.bind((HOST, PLC_PORT))
        self.plc_socket.listen()

        self.zobrazovaci_socket.bind((HOST, ZOBRAZOVACI_PORT))
        self.zobrazovaci_socket.listen()

        print(f"Server běží na adrese {HOST}:{OVLADACI_PORT} pro ovládací aplikaci.")
        print(f"Server běží na adrese {HOST}:{PLC_PORT} pro PLC aplikaci.")
        print(f"Server běží na adrese {HOST}:{ZOBRAZOVACI_PORT} pro zobrazovací aplikaci.")
```

Výpis 4: Metody serverové aplikace

Metoda `__init__` je speciální metoda ve třídách Pythonu, která se automaticky volá při vytváření instance třídy. V případě třídy Server tato metoda inicializuje proměnné socket, `ovladaci_socket`, `zobrazovaci_socket` a `plc_socket` na hodnotu `None`. Tím, že tyto proměnné zpočátku nastaví na hodnotu `None`, třída umožňuje zkontrolovat, zda byly inicializovány, nebo ne. Tyto proměnné budou aktualizovány a budou jim přiřazeny objekty soketů při spuštění serveru v metodě `start_server`.

```
def __init__(self):
        self.ovladaci_socket = None
        self.zobrazovaci_socket = None
        self.plc_socket = None

    def receive_from_client(self, client_socket):
        data = client_socket.recv(1024)
        return data.decode()
```

Výpis 5: Konstruktor __init__

Metoda `receive_from_client` přebírá parametr `client_socket`, který představuje soketové spojení s konkrétním klientem. Tato metoda přijímá data od klienta voláním metody `recv` na soketu. Očekává, že data budou kódována jako bajty a budou mít maximální velikost 1024 bajtů. Přijatá data jsou vrácena po dekódování pomocí metody `decode`, přičemž se předpokládá, že data byla odeslána jako řetězec. To umožňuje serveru přijímat a zpracovávat zprávy nebo informace zaslané klientem.

Metoda `receive_ovladaci` je zodpovědná za příjem dat ze soketu `ovladaci_socket`, který představuje soketové spojení s klientem ovládací aplikace. Přijímá příchozí spojení od klienta pomocí metody `accept_socket`. Adresa klienta je vypsána jen pro informaci o přijetí. Poté zavolá metodu `receive_from_client`, kterou jsme definovali dříve, aby přijala a dekódovala data odeslaná klientem. Očekává se, že přijatá data budou ve formátu JSON, proto se analyzují pomocí metody `json.loads`

```
def receive_ovladaci(self):
        ovladaci_client, ovladaci_address = self.ovladaci_socket.accept()
        print(f"Připojen klient_ovladaci z {ovladaci_address}")
        data = self.receive_from_client(ovladaci_client)
        parsed_data = json.loads(data)
        #ovladaci_client.close()
        return parsed_data
```

Výpis 6: Metoda receive_ovladaci

Metoda `send_zobrazovaci` je zodpovědná za odeslání zprávy do `zobrazovaci_socket`, který představuje socketové spojení s klientem zobrazovací aplikace. Nejprve vypíše zprávu, která má být odeslána. Poté přijme příchozí spojení od klienta a vypíše jeho adresu. Parametr zprávy je zakódován a odeslán klientovi pomocí metody `sendall` a socketu.

```
def send_zobrazovaci(self, message):
        print(message)
        zobrazovaci_client, zobrazovaci_address = self.zobrazovaci_socket.accept()
        print(f"Připojení k klient_zobrazovaci z {zobrazovaci_address}")
        zobrazovaci_client.sendall(message.encode())
        #zobrazovaci_client.close()
```

Výpis 7: Metoda send_zobrazovaci

Metoda `send_to_plc` je zodpovědná za odeslání zprávy do soketu `plc_socket`, který představuje soketové spojení s klientem plc aplikace. Funguje podobně jako metoda `send_zobrazovaci`, ale odesílá zprávu klientovi PLC. Zpráva se vypíše, přijme se příchozí spojení od PLC klienta a vypíše se adresa klienta. Zpráva je zakódována a odeslána klientovi pomocí metody `sendall` a socketu.

```
def send_to_plc(self, message):
        print(message)
        plc_client, plc_address = self.plc_socket.accept()
        print(f"Připojí k klient_plc z {plc_address}")
        plc_client.sendall(message.encode())
        #zobrazovaci_client.close()
```

Výpis 8: Metoda send_to_plc

Metoda `receive_from_plc` je zodpovědná za příjem dat ze soketu `plc_socket`, který představuje soketové spojení s klientem PLC aplikace. Přijímá příchozí spojení od klienta a vypisuje jeho adresu. Poté zavolá metodu `receive_from_client`, která je definována dříve, aby přijala a dekódovala data odeslaná klientem PLC. Podobně jako u metody `receive_ovladaci` se očekává, že přijatá data budou ve formátu JSON, a analyzují se pomocí metody `json.loads`.

```
def receive_from_plc(self):
        plc_client, plc_address = self.plc_socket.accept()
        print(f"Připojen klient_plc z {plc_address}")
        data = self.receive_from_client(plc_client)
        parsed_data = json.loads(data)
        #zobrazovaci_client.close()
        return parsed_data
```

Výpis 9: Metoda receive_from_plc

Metoda ověření příkazu

Metoda `overeni_prikazu` začíná definicí sady globálních proměnných používaných v celé třídě.

1. Přidá se instance třídy `HerniPrikazy`.
2. Pomocí `self.ovladaci_socket.setblocking(False)` se nastaví `self.ovladaci_socket` do neblokujícího režimu. To umožňuje programu pokračovat ve vykonávání i v případě, že nepřichází žádné spojení od klienta ovládací aplikace.
3. Metoda inicializuje některé proměnné, včetně `start_time`, `elapsed_time` a `ovladaci_data`.
4. Uvnitř cyklu `while` se metoda pokusí přijmout spojení od klienta ovládací aplikace pomocí `self.ovladaci_socket.accept()`. Pokud nedojde k žádnému příchozímu spojení, je zachycena výjimka `BlockingIOError` a je vypočten uplynulý čas.
5. Pokud je navázáno spojení a jsou přijata data od klienta, jsou načtena a analyzována jako JSON pomocí `json.loads()`. Poté je `plc_client`

uzavřen. Pokud je navázáno spojení a jsou přijata data od klienta, jsou načtena a analyzována jako JSON pomocí `json.loads()`. Poté je `plc_client` uzavřena.

6. `If ovladaci_data not None`, jsou přijatá data zpracována voláním `prikazy.process_data(ovladaci_data)`.
7. Metoda kontroluje hodnotu proměnné `prikaz` (získanou ze zpracovávaných dat) a na základě její hodnoty provádí různé akce.
8. Pokud je `prikaz "stop"`, odešle klientovi zobrazovací aplikace zprávu stop pomocí `self.send_zobrazovaci()` a čeká na příjem dat od klienta pomocí `self.receive_ovladaci()`. Pokud jsou přijatá data `"oprava"`, odešle je zpět klientovi zobrazovací aplikace pomocí `self.send_zobrazovaci()`.
9. Pokud je `prikaz "vynulovat"`, vynuluje různé globální proměnné týkající se hry.
10. Pokud je `prikaz "karta"`, zvýší proměnnou `pocetkaret` a na základě hodnoty `pocetkaret` odešle zprávu zobrazovací aplikaci.
11. Pokud je prikaz `"nepocitej_hod"`, nastaví proměnnou `opak_faul` na hodnotu 1 a pošle odpovídající zprávu klientovi zobrazovací aplikace.
12. Pokud není splněna žádná z výše uvedených podmínek, vypíše "jiný příkaz".
13. Po zpracování přijatých dat se smyčka přeruší a provádění metody se ukončí.

```
def overeni_prikazu(self):
        global faul, opak_faul, typ_hry, pocet_hodu, hody_zbytek, prumerny_pocet_kuzelek, cas,
jmeno_hrace, jmeno_klubu, trenink_cas, trenink_pocet_hodu, trenink_typ_hodu, vlastni_postaveni,
pozice_kuzelek, rychlost, pocet_chyb, soucet_kuzelek, soucet_kuzelek_celkem, celkovy_bodovy_zis

        prikazy = HerniPrikazy()

        self.ovladaci_socket.setblocking(False)  # Nastavení socketu na neblokující režim

        start_time = time.time()
        elapsed_time = 0
        ovladaci_data = None
        pocetkaret = 0

        while elapsed_time < 5 and ovladaci_data is None:
            try:
                ovladaci_client, ovladaci_address = self.ovladaci_socket.accept()
                print(f"Připojen klient_ovladaci z {ovladaci_address}")
                data = self.receive_from_client(ovladaci_client)
                ovladaci_data = json.loads(data)
                ovladaci_client.close()
            except BlockingIOError:
                elapsed_time = time.time() - start_time
```

Výpis 10: Kroky 1 až 6

```
if ovladaci_data is not None:
                prikazy.process_data(ovladaci_data)
                if prikaz == "stop":
                    message = json.dumps({"prikaz": "stop"})
                    self.send_zobrazovaci(message)
                    data = self.receive_ovladaci()
                    prikazy.process_data(data)
                    if data == "oprava":
                        self.send_zobrazovaci(data)

                elif prikaz == "vynulovat":
                    typ_hry = None
                    pocet_hodu = 0
                    hody_zbytek = 0
                    prumerny_pocet_kuzelek = 0
                    cas = None
                    typ_kuzelek = None
                    jmeno_hrace = None
                    jmeno_klubu = None
                    vlastni_postaveni = None
                    prikaz = None
                    kuzelky = None
                    pozice_kuzelek = None
                    rychlost = 0
                    pocet_chyb = 0
                    soucet_kuzelek = 0
                    soucet_kuzelek_celkem = 0
                    celkovy_bodovy_zisk = 0
                    faul = 0
                    opak_faul =0
                    print("vynulovány hodnoty")
                    vynulovani = json.dumps({"prikaz": "stop","prikaz": "c
"0","rychlost": "0", "chyby": "0", "soucet_kuzelek": "0", "soucet_kuzelek_
"kuzelky":"000000000","faul":"0","opak_faul":"0","karta":"0","automat":"nc
)
                    self.send_zobrazovaci(vynulovani)
```

Výpis 11: Metoda receive_from_plc zbylé kroky

Ve výpisu 12 je implementován postup operací, který se provede po zavolání metody karta.

```
elif prikaz == "karta":
                    pocetkaret = pocetkaret + 1
                    print(pocetkaret)
                    if pocetkaret == 1:
                        message = json.dumps({"karta": "1"})
                        self.send_zobrazovaci(message)
                    elif pocetkaret < 1:
                        message = json.dumps({"karta": "2"})
                        self.send_zobrazovaci(message)


                elif prikaz == "nepocitej_hod":
                    opak_faul = 1
                    message = json.dumps({"opak_faul": "1"})
                    self.send_zobrazovaci(message)

                else:
                    print("jiný příkaz")
                break
```

Výpis 12: Metoda receive_from_plc rozpoznávání příkazů

3.2.5 Návrh algoritmu hry

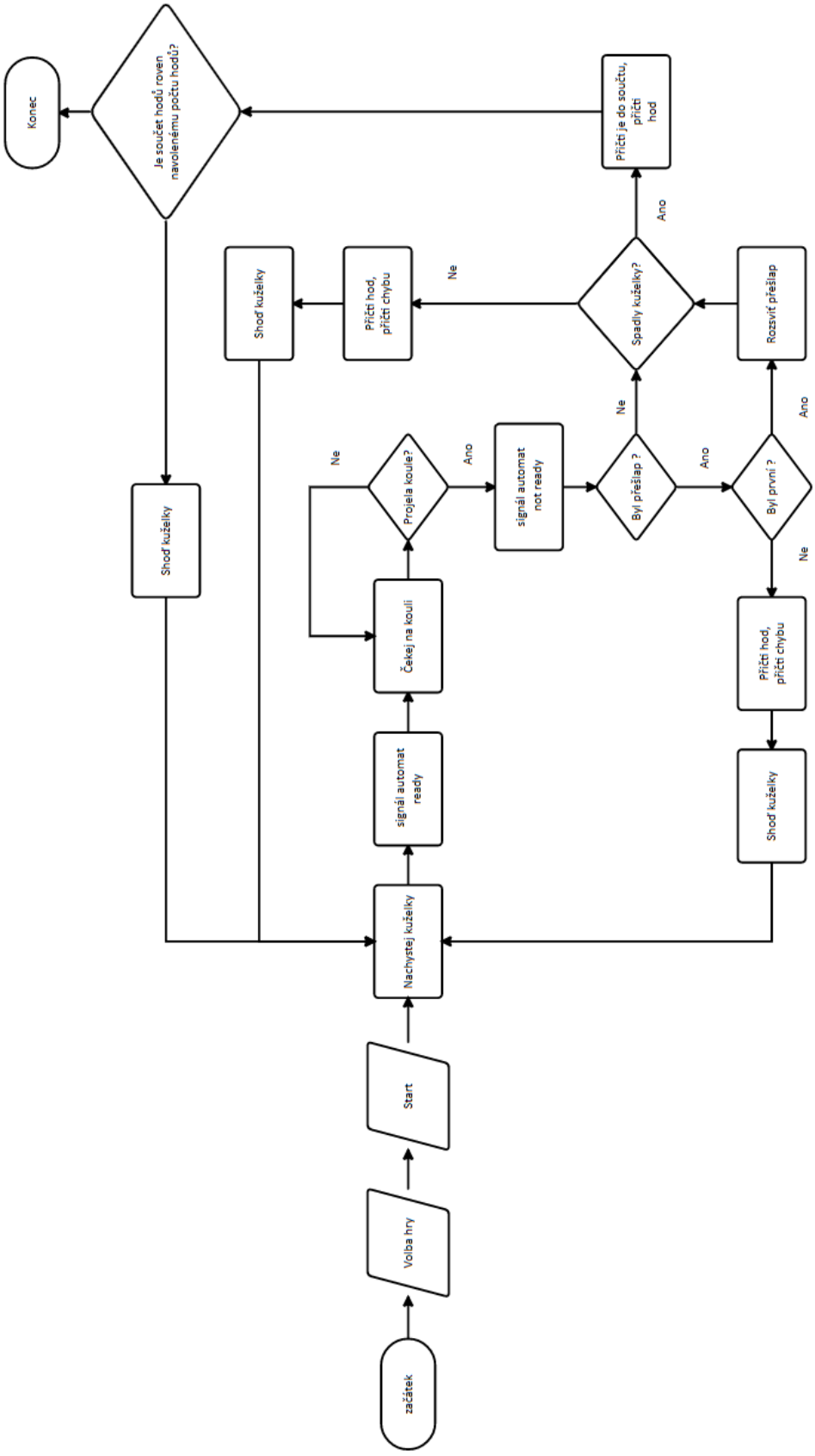
Toto je původní návrh algoritmu chodu hry:

- Odeslání informací o hře z ovládací aplikace.
- Na základě typu a počtu hodů se pošle zpráva do PLC.
- Odpověď od PLC s označením `"automat ready"`.
- Předání zprávy `"automat ready"` do zobrazovací aplikace.
- Příkaz start:
- Odeslání příkazu start do zobrazovací aplikace
- Smyčka: dokud není `"zbyle_hody"` 0:
    - Odeslání zprávy `"stav": "faul"` do PLC.
    - Odpověď PLC s hodnotou faulu.
    - Odeslání zprávy `"stav": "rychlost"` do PLC.
    - Odpověď PLC s hodnotou rychlosti.
    - Odeslání zprávy `"rychlost": "23.7"` do zobrazovací aplikace.
    - Odeslání zprávy `"stav": "kuzelky"` do PLC.
    - Odpověď PLC s hodnotou `"kuzelky"`.
    - Odeslání zprávy `"kuzelky"` do zobrazovací aplikace.
    - Odeslání zprávy `"automat": "notready"` do zobrazovací aplikace.

- Výpočty a úpravy v příkazech.
- Odeslání vypočítaných dat do zobrazovací aplikace.
- Posílání zpráv do PLC na základě typu kuželek a počtu hodů.
- Odpověď od PLC s označením `"automat": "ready".`
- Předání zprávy `"automat": "ready"` do zobrazovací aplikace.
- Odeslání zprávy `"stav": "faul"` do PLC (znovu).

Dále by měl server umět kdykoliv reagovat na jisté příkazy:

- Příkaz stop:
- Odeslání příkazu stop do zobrazovací aplikace.
- Oprava dat uvnitř serveru nebo vynulování dat uvnitř serveru.
- Pokud je vyžadováno vynulování, vrácení na začátek.
- Ukončení procesu.
- Příkaz oprava:
- Nastavení a úprava všech výpočetních hodnot v aplikaci
- Posláni příkazu do serverové aplikace
- Příkaz karta
- Podle počtu karet zvolit žlutá nebo červená.
- Odeslat informace o kartách do serverové aplikace.
- Příkaz vynulovat
- Nastavit úplně všechny hodnoty na výchozí.
- Příkaz nepočítat hod
- Zajistit, aby následující hod nebyl uznán.



Obr. 9: Vývojový diagram algoritmu průběhu hry.

3.2.6 Implementace algoritmu

V následující části se nachází ukázky a vysvětlení kódu algoritmu chodu hry. Kód začíná nekonečnou smyčkou pomocí `while True`, což znamená, že hra bude pokračovat donekonečna, dokud nebude explicitně ukončena. V rámci smyčky algoritmus provádí následující akce:

- Přijímá nastavení parametrů hry z ovládací aplikace pomocí funkce `receive_ovladaci()`.
- Zpracuje přijatá data pomocí funkce `prikazy.process_data()`.
- Odešle herní data zobrazovací aplikaci ve formátu JSON pomocí funkce `send_zobrazovaci()`.
- Odešle příkaz do PLC aplikace ve formátu JSON pomocí funkce `send_to_plc()`.
- Přijme data z PLC aplikace pomocí funkce `receive_from_plc()`
- Odešle přijatá data z PLC aplikace do zobrazovací aplikace ve formátu JSON.
- Přijme příkaz "start" od ovládací aplikace a zpracuje jej.
- Odešle příkaz "start" do zobrazovací aplikace.
- Spustí vnořenou smyčku pro házecí cyklus hry.

```
# Algoritmus hry
        while True:
            #data o hre z ovladaci aplikace
            ovladaci_data = self.receive_ovladaci()
            prikazy.process_data(ovladaci_data)

            #Odeslat data o hre do zobrazovací aplikace
            message = json.dumps({"jmeno_hrace": jmeno_hrace, "jmeno_klubu": jmeno_k
hody_zbytek})
            self.send_zobrazovaci(message)

            #Odeslat prikaz do PLC
            pozice_kuzelek = "000000000"
            plc_message = json.dumps({"priprav_kuzelky": "000000000"})
            self.send_to_plc(plc_message)


            #Přijmout automat ready z PLC
            plc_data = self.receive_from_plc()
            print(plc_data)

            #Poslat automat ready do zobrazovaci aplikace
            message = json.dumps(plc_data)
            self.send_zobrazovaci(message)

            #Přijmout start
            ovladaci_data = self.receive_ovladaci()
            print(ovladaci_data)
            prikazy.process_data(ovladaci_data)

            #Poslat start do zobrazovaci aplikace
            message = json.dumps(ovladaci_data)
            self.send_zobrazovaci(message)
```

Výpis 13: Jednotlivé kroky algoritmu

Vzhledem k tomu, že algoritmus je kvalitně okomentován také tomu, že příkazy pro komunikaci mezi klienty a serverem jsou téměř totožné, budeme se už zabývat pouze složitějšími operacemi.

```
#logika faulu/přešlapu a opakovaného faulu detekovaného
plif faul == 0:
      message = json.dumps({"faul":"0"})

      self.send_zobrazovaci(message)
  elif faul == 1:
      message = json.dumps({"faul":"1"})
      #Poslat faul do zobrazovaci aplikace
      self.send_zobrazovaci(message)
  elif faul > 1:
      message = json.dumps({"opak_faul":"1"})
      #Poslat opak_faul do zobrazovaci aplikace
      self.send_zobrazovaci(message)
  # ...
  #kontola jestli nepřisel přikaz od ovladaci aplikace
  self.overeni_prikazu()
```

Výpis 14: Logika faulu/přešlapu

V této části algoritmu probíhá rozhodování o zaslání typu faulu. Nejprve by také bylo důležité říct, že okno `opak_faul` může být při hře voláno dvěma způsoby, jeden jako opakovaný faul tzn., že hráč udělá 2x přešlap, ale také jako příjem příkazu `nepocitej_kuzelky` od ovládací aplikace.

Funkce `if` zde kontroluje právě první možnost, jestli byl přešlap, jestli byl první nebo jestli byl druhý. Naopak spodní řádek volá metodu `overeni_prikazu()`, která právě slouží k obdržení příkazu např. `nepocitej_kuzelky`, `karta`, `stop` a `vynulovat`.

```
#logika sčítání shozených kuželek pro danný typ hry
# pokud se předchozí postavení nerovná novému
if pozice_kuzelek != kuzelky:
    if typ_kuzelek == "plne":
        soucet_kuzelek = kuzelky.count("1")

    elif typ_kuzelek == "dorazkove":
        soucet_kuzelek= kuzelky.count("1")- pozice_kuzelek.count("1")

    elif typ_kuzelek == "sdruzene":
        if hody_zbytek >= pocet_hodu/2:
            soucet_kuzelek = kuzelky.count("1")

        else:
            soucet_kuzelek=kuzelky.count("1")-
pozice_kuzelek.count("1")
```

Výpis 15: Logika shozených kuželek

V další části algoritmu je implementována logika sčítání spadlých kuželek pro daný typ kuželek. PLC aplikace posílá údaj o shozených kuželkách formou stringu jedniček a nul, kdy číslo jedna reprezentuje shozenou kuželku a nula stojící. V proměnné `pozice_kuzelky` je uložený poslední stav shozených kuželek. Příkaz `count()` spočítá uvnitř proměnné kolikrát string obsahuje číslo 1. Důležité je si uvědomit, že při dorážce pokud prvním hodem se shodí například 5 kuželek a druhým po něm další 2, tak v logice posílání přijde 5 kuželek a 7 kuželek shozených, proto se musí jednotlivé county od sebe odečíst.

```
#logika sčítání shozených kuželek pro danný typ hry
# pokud se předchozí postavení rovná novému
elif pozice_kuzelek == kuzelky:
    print(pozice_kuzelek)
    print("stejná pozice")
    print(typ_hry)
    if typ_kuzelek == "plne":
        soucet_kuzelek = kuzelky.count("1")
        print("jiná pozice")
    elif typ_kuzelek == "dorazkove":
        soucet_kuzelek=0
        print("stejna pozice")
    elif typ_kuzelek == "sdruzene":
        if hody_zbytek >= pocet_hodu/2:
            soucet_kuzelek = kuzelky.count("1")
            print("jiná pozice")
        else:
            soucet_kuzelek=0
            print("stejna pozice")
```

Výpis 16: Logika sčítání shozených kuželek

Ve výpisu je vytvořena logika sčítání kuželek, která nastane, pokud se oba hody jak předchozí `pozice_kuzelek` tak `kuzelky` rovnají. Dále je také uvedena podmínka, která říká, že pokud je `opak_faul`, což znamená buď 2x přešlap nebo přijatý příkaz `nepocitej_kuzelky`, tak je `soucet_kuzelek` nula.

```
tních hodnot
    hody_zbytek = int(hody_zbytek) - 1
    pocet_hodu = int(pocet_hodu)
    soucet_kuzelek_celkem = soucet_kuzelek_celkem + soucet_kuzelek
    celkovy_bodovy_zisk = celkovy_bodovy_zisk + soucet_kuzelek
    prumerny_pocet_kuzelek = soucet_kuzelek_celkem / ((pocet_hodu) - (hody_zbytek))
    if soucet_kuzelek == 0:
        pocet_chyb = int(pocet_chyb) + 1
    else:
        pocet_chyb = pocet_chyb

    #Poslání vypočítaných dat do zobrazovaci aplikace
    message = json.dumps(plc_data)
    self.send_zobrazovaci(message)
    vypoc_data = json.dumps({"prumer": prumerny_pocet_kuzelek,
                             "chyby": pocet_chyb,
                             "soucet_kuzelek": soucet_kuzelek,
                             "soucet_kuzelek_celkem": soucet_kuzelek_celkem,
                             "body_vsech_her": celkovy_bodovy_zisk,
                             "zbyva_hodu": hody_zbytek})
    self.send_zobrazovaci(vypoc_data)
```

Výpis 17: Výpočet ostatních hodnot

Tato část kódu se stará o výpočet všech hodnot do zobrazovací aplikace a následné poslání hodnot do zobrazovací aplikace.

```
# logika resetování opakovaného faulu
if faul != 0:
    faul = 1

opak_faul = 0
message = json.dumps({"opak_faul":"0"})
self.send_zobrazovaci(message)


#Zhasni rozsvícene kuželky
if typ_kuzelek == "Pln\u00e9":
    message = json.dumps({"kuzelky":"000000000"})
    self.send_zobrazovaci(message)
elif typ_kuzelek == "Dor\u00e1\u017ekov\u00e9":
    if kuzelky == "111111111":
        message = json.dumps({"kuzelky":"000000000"})
        self.send_zobrazovaci(message)

elif typ_kuzelek =="Sdru\u017een\u00e9":
    if hody_zbytek >= pocet_hodu/2:
        message = json.dumps({"kuzelky":"000000000"})
        self.send_zobrazovaci(message)
    else:
        if kuzelky == "111111111":
            message = json.dumps({"kuzelky":"000000000"})
            self.send_zobrazovaci(message)
```

Výpis 18: Logika resetování opakovaného faulu

Ve výpisu je implementována logika resetování `opak_faul`, což je nutné provést po každém hodu, protože opakovaný faul svítí pouze v případech, kdy se hráč dopustil, nějakého přestupku, a proto by se mu neměl hod počítat.

Následuje operace zhasínání labelu ukazujících, jestli kuželka spadla nebo ne.

```
#Podmínka postavení kuželek
if hody_zbytek !=0:
    if typ_kuzelek == "Pln\u00e9":
        plc_message = json.dumps({"priprav_kuzelky": "000000000"})
    elif typ_kuzelek == "Dor\u00e1\u017ekov\u00e9":
        if kuzelky.count("1") == 9:
            plc_message = json.dumps({"priprav_kuzelky": "000000000"})
        else:
            plc_message = json.dumps({"priprav_kuzelky": kuzelky})
    elif typ_kuzelek == "Sdru\u017een\u00e9":
        if hody_zbytek <= pocet_hodu/2:
            if kuzelky != "111111111":
                plc_message = json.dumps({"priprav_kuzelky": kuzelky})
            else:
                plc_message = json.dumps({"priprav_kuzelky": "000000000"})
        else:
            plc_message = json.dumps({"priprav_kuzelky": "000000000"})
    # else:
    #     plc_message = json.dumps({"priprav_kuzelky": vlastni_postaveni})
```

Výpis 19: Podmínka postavení kuželek

Poslední důležitý proces je posílání PLC aplikaci informace pro postavení kuželek na další hod. Jak už bylo v předchozích metodách znázorněno, opět jde o rozdíl mezi typem kuželek. Pokud se hraje do plných, je string obsahující informace pořád stejný, jakmile ale budeme hrát hru do dorážkových, musíme čekat, dokud neshodíme všechny kuželky a až pak je postavit znova.

3.3 Ovládací aplikace

3.3.1 Popis ovládací aplikace

V této kapitole je popsána tvorba klientské ovládací aplikace a obsahuje výpisy ze souboru `klient_ovladaci.py`, který se nachází v příloze A. Tato aplikace je napsaná v jazyce Python s využitím knihovny `PyQt6` pro tvorbu grafického základního a zatím testovacího uživatelského rozhraní (GUI). Kód definuje několik tříd, které představují konkrétní okna a widgety aplikace pro ovládání hry.

Hlavní část kódu začíná s definicí třídy `GameSelectionWindow`, která je hlavním oknem aplikace. Tato třída vytváří hlavní okno a obsahuje tlačítka pro výběr typu hry: Základní hra, Sportovní hra a Tréninková hra. Při kliknutí na tlačítko se zavolá příslušná metoda pro zobrazení okna s podrobnostmi daného typu hry.

Každý typ hry (Základní hra, Sportovní hra, Tréninková hra) má svou vlastní třídu, která definuje okno se specifickými prvky a funkcionalitou. Například třída `BasicGameWindow` definuje okno pro základní hru, které obsahuje vstupní pole pro délku hry a počet hodů, tlačítka pro různé akce (Start, Stop, Vynulovat atd.) a další prvky pro volbu typu hry.

Každá třída má také metody pro obsluhu událostí, jako je kliknutí na tlačítka. Tyto metody volají funkci `send_data`, která pomocí soketů odesílá data na server. Data jsou ukládána do formátu JSON před odesláním.

Kromě toho jsou v kódu definovány další třídy pro specifické funkcionality, jako je okno pro opravy (`CorrectionWindow`) nebo okno pro vlastní pozice kuželek (`CustomPositionWindow`). Tyto třídy obsahují metody pro ovládání a přenos dat na server.

3.3.2 Implementace ovládací aplikace

Začněme prozkoumáním fragmentu kódu, který je zodpovědný za odesílání dat z řídicí aplikace na server:

```
import sys
import json
from PyQt6.QtWidgets import QApplication, QMainWindow,
QWidget, QVBoxLayout, QLabel, QLineEdit, QPushButton,
QCheckBox, QButtonGroup, QRadioButton, QComboBox
import socket

HOST = '127.0.0.1'  # Adresa serveru
PORT = 5000  # Port serveru
#PORT = 1234

def send_data(data):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((HOST, PORT))
        json_data = json.dumps(data)
        s.sendall(json_data.encode())
        print('Data odeslána na server:')
```

Výpis 20: Odesílání dat na server

Na tomto výpisu je znázorněno, že aplikace importuje knihovny `sys`, `json` a `socket` Funkce `send_data` naváže spojení se serverem pomocí zadaného hostitele a portu. Převede data do formátu JSON a odešle je na server.

3.3.3 Výběr hry:

Ovládací aplikace vytváří okno pro výběr hry, kde si uživatelé mohou vybrat typ hry. Okno se skládá ze tří tlačítek: `"Základní hra"`, `"Sportovní hra"` a `"Tréninková hra"`.

```
basic_button = QPushButton("Základní hra")
basic_button.clicked.connect(self.show_basic_game_details)
layout.addWidget(basic_button)

sport_button = QPushButton("Sportovní hra")
sport_button.clicked.connect(self.show_sport_game_details)
layout.addWidget(sport_button)

training_button = QPushButton("Tréninková hra")
training_button.clicked.connect(self.show_training_game_details)
layout.addWidget(training_button)
```

Výpis 21: Implementace tlačítka „Základní hra“

Po kliknutí na tlačítko `"Základní hra"` se zavolá funkce `show_basic_game_details`, která otevře nové okno určené speciálně pro základní hru.

3.3.4 Okno základní hry:

Okno základní hry umožňuje uživatelům zadávat podrobnosti o hře, například dobu trvání hry a počet hodů. Obsahuje také přepínače pro výběr typu hry. Níže je výňatek z kódu, který zpracovává okno základní hry.

```
class BasicGameWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Základní hra")

        layout = QVBoxLayout()

        duration_label = QLabel("Délka hry:")
        self.duration_input = QLineEdit()
        layout.addWidget(duration_label)
        layout.addWidget(self.duration_input)

        rolls_label = QLabel("Počet hodů:")
        self.rolls_input = QLineEdit()
        layout.addWidget(rolls_label)
        layout.addWidget(self.rolls_input)

        #Výběr typu hry
        self.radioButtonGroup = QButtonGroup()

        self.full_radioButton = QRadioButton("plne")
        self.collision_radioButton = QRadioButton("dorazkove")
        self.combined_radioButton = QRadioButton("sdruzene")

        self.radioButtonGroup.addButton(self.full_radioButton)
        self.radioButtonGroup.addButton(self.collision_radioButton)
        self.radioButtonGroup.addButton(self.combined_radioButton)

        layout.addWidget(self.full_radioButton)
        layout.addWidget(self.collision_radioButton)
        layout.addWidget(self.combined_radioButton)


        submit_button = QPushButton("Potvrdit")
        submit_button.clicked.connect(self.submit)
        layout.addWidget(submit_button)
        #...
```

Výpis 22: Třída BasicGameWindow

Třída `BasicGameWindow` představuje okno základní hry. Obsahuje vstupní pole pro dobu trvání a hody a přepínače pro výběr typu hry. Uživatelé mohou zadat požadované hodnoty a odeslat je pomocí poskytnutých tlačítek. Níže je ukázka metod, které zpracovávají tlačítka start, stop a karta. Takových to funkcí je tu více a jsou totožné i v ostatních třídách

```
def start_game(self):
        #print("Start")
        game_details = {"prikaz":"start"}
        print(json.dumps(game_details))
        send_data(game_details)

    def stop_game(self):
        #print("Stop")
        game_details = {"prikaz":"stop"}
        print(json.dumps(game_details))
        send_data(game_details)

    def show_karta(self):
        #print("Karta")
        game_details = {"prikaz":"karta"}
        print(json.dumps(game_details))
        send_data(game_details)
```

Výpis 23: Metoda Start_game

3.3.5 Okno sportovní hry:

Ve třídě `SportGameWindow` je implementováno okno sportovní hry a je podobné základnímu oknu hry, ale obsahuje vstupní pole pro zadaní jména hráče a jejich klubu a také volbu typu turnaje.

```
class SportGameWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Sportovní hra")

        layout = QVBoxLayout()

        name_label = QLabel("Vaše jméno:")
        self.name_input = QLineEdit()
        layout.addWidget(name_label)
        layout.addWidget(self.name_input)

        team_label = QLabel("Název týmu:")
        self.team_input = QLineEdit()
        layout.addWidget(team_label)
        layout.addWidget(self.team_input)

        rolls_label = QLabel("Typ hodů:")
        self.rolls_input = QComboBox()
        self.rolls_input.addItems(["20 SH", "30 SH", "40 SH",
        layout.addWidget(rolls_label)
        layout.addWidget(self.rolls_input)
```

Výpis 24: Část implementace volby typu turnajových her

3.3.6 Okno tréninkové hry:

Okno tréninkové hry je opět velice podobné oknu základní hry, je tu ale jeden rozdíl, a to ve volbě vlastního postavení, která odkáže na okno nové pozice.

3.3.7 Opravné okno:

Okno pro opravu poskytuje rozhraní pro opravu chyb nebo úpravu detailů hry.

```
class TrainingGameWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Tréninková hra")

        layout = QVBoxLayout()

        duration_label = QLabel("Délka hry:")
        self.duration_input = QLineEdit()
        layout.addWidget(duration_label)
        layout.addWidget(self.duration_input)

        rolls_label = QLabel("Počet hodů:")
        self.rolls_input = QLineEdit()
        layout.addWidget(rolls_label)
        layout.addWidget(self.rolls_input)
```

Výpis 25: Ukázka kódu představující část implementace opravného okna

Třída `CorrectionWindow` představuje okno pro opravu parametrů hry. Obsahuje vstupní pole pro provádění oprav, jako jsou úprava počtu bodu, hodů, času a dalších.

3.3.8 Okno vlastní pozice:

Poslední třídou je `CustomPositionWindow`, která představuje okno pro zadávání vlastních pozic. Obsahuje vstupní pole plné tlačítek s čísly reprezentující kuželky, ty slouží pro zadávání údajů o vlastní pozici kuželek.

Okno vlastní pozice umožňuje uživatelům zadat vlastní pozice jednotlivých kuželek.

```
class TrainingGameWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Tréninková hra")

        layout = QVBoxLayout()

        duration_label = QLabel("Délka hry:")
        self.duration_input = QLineEdit()
        layout.addWidget(duration_label)
        layout.addWidget(class CustomPositionWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Vlastní postavení kuželek")
        layout = QVBoxLayout()
        label = QLabel("Vyberte nové postavení kuželek")
        layout.addWidget(label)

        self.buttons = []

        for i in range(9):
            button = QPushButton(str(i + 1))
            button.setCheckable(True)
            layout.addWidget(button)
            self.buttons.append(button)

        submit_button = QPushButton("Potvrdit")
        submit_button.clicked.connect(self.submit)
        layout.addWidget(submit_button)

        self.setLayout(layout)self.duration_input)

        rolls_label = QLabel("Počet hodů:")
        self.rolls_input = QLineEdit()
        layout.addWidget(rolls_label)
        layout.addWidget(self.rolls_input)
```

Výpis 26: Příklad úryvku kódu okna vlastní pozice obsahující tlačítka.

3.4 PLC aplikace

Tato kapitola popisuje tvorbu klientské PLC aplikace a obsahuje výpisy ze souboru `klient_plc.py`, který se nachází v příloze A. Jedná se o aplikaci v jazyce Python, která komunikuje se serverem pomocí soketů. Je importován modul `socket` a modul `time`. Jde o náhradu za komunikaci s PLC

3.4.1 Fungování aplikace

Aplikace definuje funkci `send_data`, která přijímá datový parametr. Tato funkce naváže spojení se serverem pomocí zadaného HOST a PORT. Převede data do formátu JSON, odešle je na server a vypíše zprávu, že data byla odeslána.

Aplikace definuje funkci `receive_message`, která přijímá zprávy ze serveru. Naváže spojení se serverem pomocí zadaného HOST a PORT, přijme data ze serveru (až 1024 bajtů), dekóduje je z bajtů na řetězec a vypíše přijatá data.

Hlavní funkce je vstupním bodem aplikace. Zobrazuje zprávu, která oznamuje, že aplikace je připravena ke komunikaci se serverem.

Uvnitř hlavní smyčky (`while True`) aplikace volá funkci `receive_message`, která každou sekundu kontroluje příchozí data ze serveru pomocí `time.sleep(1)`.

Aplikace vyzve uživatele k zadání zprávy JSON, kterou chce odeslat na server. Pokud uživatel zadá "exit", smyčka se přeruší a aplikace se ukončí. Zadaná data jsou poté odeslána na server pomocí funkce `send_data`.

Hlavní smyčka pokračuje, opakovaně kontroluje příchozí data ze serveru a odesílá vstup uživatele, dokud se uživatel nerozhodne ukončit činnost.

Aplikace umožňuje obousměrnou komunikaci mezi klientem a serverem, přičemž klient může odesílat zprávy JSON a přijímat zprávy ze serveru. Časový modul slouží k zavedení prodlevy mezi kontrolou příchozích dat ze serveru.

```python
import json
import time
from socket import socket, AF_INET, SOCK_STREAM

HOST = '127.0.0.1'
PORT = 5001

def send_data(data):
    with socket(AF_INET, SOCK_STREAM) as s:
        s.connect((HOST, PORT))
        json_data = data
        s.sendall(json_data.encode())
        print('Data sent to the server.')

def receive_message():
    # Připojení k serveru a příjem zprávy
    server_address = ('127.0.0.1', 5001)  # Adresa a port serveru
    client_socket = socket(AF_INET, SOCK_STREAM)
    client_socket.connect(server_address)
    data = client_socket.recv(1024).decode()
    print(data)

def main():
    print("Ready to communicate with the server.")

    while True:
        # Kontrola příchozích dat každou sekundu
        receive_message()
        time.sleep(1)

        data = input("Enter the JSON message to send (or 'exit' to quit): ")

        if data == "exit":
            break

        print(data)
        send_data(data)

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

Výpis 27: Kod PLC aplikace

3.5 Zobrazovací aplikace

3.5.1 Popis zobrazovací aplikace:

Tato kapitola popisuje tvorbu klientské zobrazovací aplikace a obsahuje výpisy ze souboru `Klient_zobrazovaci.py`, který se nachází v příloze A. Je zde popsána zobrazovací aplikace pro hru kuželky, která slouží k zobrazování aktuálních výsledků při kuželkové hře. Aplikace je navržena pro příjem dat od serveru a okamžité zobrazení těchto dat pomocí základního testovacího grafického uživatelského rozhraní. Aplikace je napsána v jazyce Python s využitím knihoven `PyQt6` a `socket`

Aplikace je rozdělena do dvou tříd, které slouží k oddělenému zpracování různých částí funkcionalit. Hlavní třídou je `BowlingStatusWindow`, která představuje hlavní okno aplikace a obsahuje různé widgety pro zobrazení informací.

Grafické rozhraní aplikace je navrženo pomocí knihovny PyQt6 a obsahuje několik klíčových prvků pro zobrazení výsledků hry kuželky. Hlavní okno aplikace obsahuje textová pole pro zobrazení informací o hráči, klubu, stavu kuželek, automatu, faulu, opakovaném faulu, počtu karet, rychlosti koule, průměru shozených kuželek na hod, zbývajícím čase a počtu chyb.

Aplikace komunikuje se serverem pomocí síťového soketu a přijímá data ve formátu JSON. Po přijetí dat jsou jednotlivé části aplikace aktualizovány na základě těchto dat. Například přijetí nového jména hráče způsobí aktualizaci pole pro zobrazení jména hráče v aplikaci.

3.5.2 Implementace zobrazovací aplikace:

Aplikace začíná importem potřebných modulů a knihoven:

```
import sys
from PyQt6.QtWidgets import QApplication, QLabel, QVBoxLayout, QWidget
from PyQt6 import QtCore
from PyQt6.QtCore import QThread, QTimer, pyqtSignal
import json
from socket import socket, AF_INET, SOCK_STREAM
```

Výpis 28: Import modulů

Tyto moduly poskytují funkce pro vytvoření grafického rozhraní, zpracování vláken, správu časovačů, komunikaci se serverem a zpracování dat JSON.

3.5.3 Vytvoření vlákna přijímače dat:

Pro nepřetržitý příjem dat ze serveru bez blokování hlavního uživatelského rozhraní je implementována třída `DataReceiverThread` jako podtřída `QThread`. Toto vlákno se připojuje k serveru, přijímá data a vysílá signál vždy, když jsou přijata nová data.

```
class DataReceiverThread(QThread):
    data_received = pyqtSignal(str)

    def run(self):
        while True:
            server_address = ('127.0.0.1', 5002)  # Adresa a port serveru
            client_socket = socket(AF_INET, SOCK_STREAM)
            client_socket.connect(server_address)
            data = client_socket.recv(1024).decode()
            self.data_received.emit(data)
            client_socket.close()
```

Výpis 29: Metoda přijímání dat

V metodě `run` vlákna `DataReceiverThread` probíhá nepřetržitý proces příjmu dat. Navazuje spojení se serverem, přijímá data pomocí soketu a vysílá signál `data_received` s přijatými daty.

3.5.4 Definice stavového okna kuželek:

Třída `BowlingStatusWindow` je hlavním oknem aplikace. Dědí od `QWidget` a obsahuje různé widgety `QLabel` pro zobrazení různých informací týkajících se hry kuželky. Obsahuje také rozvržení pro vertikální uspořádání widgetů.

```
class BowlingStatusWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Stav kuželek")
        layout = QVBoxLayout()

        # Jméno hráče
        self.player_label = QLabel()
        layout.addWidget(self.player_label)

        # Jméno klubu
        self.club_label = QLabel()
        layout.addWidget(self.club_label)

        # Okno zobrazující stav kuželek
        self.bowling_pins = []
        for i in range(9):
            pin_label = QLabel(str(i + 1))  # Číslo kuželky
            pin_label.setStyleSheet("background-color: transparent; border-radius:
bold")
            pin_label.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
            self.bowling_pins.append(pin_label)
            layout.addWidget(pin_label)
```

Výpis 30: Třída pro vytvoření hlavního okna

Metoda `__init__` nastaví titulek okna, vytvoří widgety `QLabel` pro zobrazení jména hráče, názvu klubu a dalších informací a uspořádá je do vertikálního rozvržení pomocí `QVBoxLayout`.

3.5.5 Aktualizace uživatelského rozhraní:

Třída `BowlingStatusWindow` poskytuje několik metod pro aktualizaci uživatelského rozhraní na základě přijatých dat. Tyto metody zpracovávají aktualizaci jmen hráčů, názvů klubů, stavu kuželek, stavu automatu, faulů, zobrazení karet, rychlosti koule, průměrného počtu kuželek na hod, zbývajícího času, počtu chyb, celkového počtu kuželek shozených hráčem, zbývajících hodů, kuželek shozených v aktuálním hodu a celkového počtu kuželek shozených v aktuální hře.

```
def set_pole_jmeno_hrace(self, jmeno_hrace):
        self.player_label.setText(jmeno_hrace)

    def set_pole_jmeno_klubu(self, jmeno_klubu):
        self.club_label.setText(jmeno_klubu)


    def set_stav_kuzelek(self, stav):
        if len(stav) != 9:
            print("Neplatný stav kuželek.")
            return

        for i, hodnota in enumerate(stav):
            kuzele_label = self.bowling_pins[i]
            if hodnota == '1':
                kuzele_label.setStyleSheet("background-color: green; borde
bold")

            elif hodnota == '0':
                kuzele_label.setStyleSheet("background-color: transparent;
weight: bold")

            else:
                print("Neplatná hodnota kuželky.")
```

Výpis 31: Příjem pro nastavení dat

Tyto metody aktualizace přijímají přijatá data jako vstup a aktualizují příslušné widgety `QLabel` novými hodnotami.

3.5.6 Spuštění komunikace a příjem dat:

Metoda `start_communication` třídy `BowlingStatusWindow` inicializuje vlákno pro příjem dat a připojí jeho signál `data_received` ke slotu `receive_message`. Tato metoda spustí vlákno a umožní komunikaci se serverem.

```
def start_communication(self):
        self.thread = DataReceiverThread()
        self.thread.data_received.connect(self.receive_message)
        self.thread.start()
```

Výpis 32: Start komunikace

Metoda `start_communication` vytváří instanci vlákna `DataReceiverThread`, připojí jeho signál `data_received` ke slotu `receive_message` a spustí vlákno, které začne přijímat data ze serveru.

3.5.7 Zpracování přijatých dat:

Metoda `receive_message` třídy `BowlingStatusWindow` je zodpovědná za zpracování přijatých dat. Analyzuje data JSON a na základě přijatých informací aktualizuje uživatelské rozhraní.

```
def receive_message(self, data):
        print(data)

        # Zpracování přijaté zprávy
        try:
            message = json.loads(data)
            for typ, hodnota in message.items():
                if typ == 'kuzelky':
                    self.set_stav_kuzelek(str(hodnota))
                elif typ == 'faul':
                    self.set_faul(hodnota == '1')
                elif typ == 'opak_faul':
                    self.set_opak_faul(hodnota == '1')
                elif typ == 'karta':
                    self.set_pole_karty(int(hodnota))
                elif typ == 'rychlost':
                    self.set_rychlost_koule(float(hodnota))

                #...


                elif typ == 'automat':
                    self.set_stav_automatu(hodnota)
                elif typ == 'jmeno_hrace':
                    self.set_pole_jmeno_hrace(str(hodnota))
                elif typ == 'jmeno_klubu':
                    self.set_pole_jmeno_klubu(str(hodnota))
                elif typ == 'prikaz':
                    if hodnota == 'start':
                        self.start_timer()
                    elif hodnota == 'stop':
                        self.stop_timer()
                else:
                    print("Neznámý typ informace.")
        except json.JSONDecodeError:
            print("Chybný formát JSON.")
```

Výpis 33: Metoda receive_message

Metoda `receive_message` analyzuje přijatá data pomocí `json.loads` a extrahuje potřebné informace. Poté zavolá odpovídající metody `update`, aby aktualizovala uživatelské rozhraní o nové informace.

3.6 Úprava grafického rozhraní aplikací

Tato kapitola popisuje úpravu mnou vytvořených aplikací. Veškeré soubory včetně piktogramů .ui souborů jsou nahrány do složky `po úpravě GUI`, která je součástí přílohy `Diplomová práce Ludvík Szelke – programy.zip`

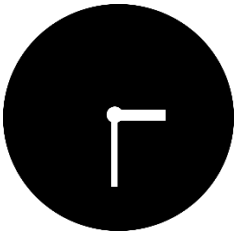
K úpravě grafického zobrazení aplikací bylo použito softwaru QT Designer, což je vizuální nástroj vyvinutý společností Qt Development Frameworks, který umožňuje vytvářet grafické uživatelské rozhraní (GUI) pro aplikace napsané v jazyce Qt. V jeho prostředí se dá velmi jednoduše nakreslit vámi chtěné grafické rozhraní, které později umí program uložit do .ui souboru. Při tvorbě je velice důležité dbát na dodržování jmen všech prvků grafického rozhraní, aby nové GUI bylo stejně funkční jako předchozí.

Tento soubor UI soubor je obvykle ve formátu XML a obsahuje informace o různých prvcích rozhraní, jako jsou tlačítka, textová pole, seznamy, dialogová okna a další. Každý prvek má své vlastnosti, jako je pozice, velikost, text, barva a události, které jsou s ním spojeny. UI soubor slouží jako šablona, která definuje strukturu grafického rozhraní. Existují jisté programy, které vám pomocí této šablony umějí vytvořit GUI v daném programovacím jazyku.

3.6.1 Postup předělání grafického rozhraní

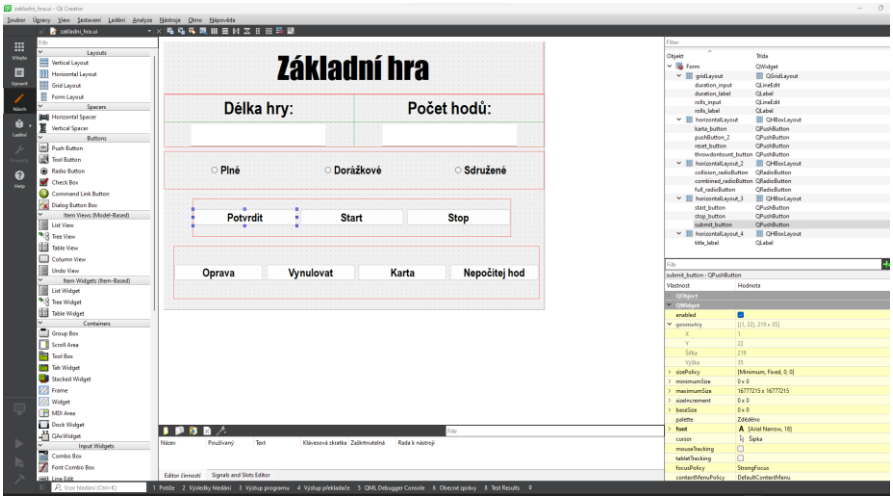
Při úpravě grafického rozhraní aplikace jsem postupoval následovně:

1. Vytvoření piktogramu pro pochopení údajů v programu Inkscape.



Obr. 10: Piktogram času

2. Vytvoření návrhu v Qt Designeru.



Obr. 11: Ukázka tvorby Grafického rozhraní v Qt Designeru

1. Uložení UI soboru.
2. Vytvoření nového GUI v novém skriptu pomocí příkazu uic.

```
python -m PyQt5.uic.pyuic -x zobrazovaci.ui -o zobrazovaci_ui.py
```

Výpis 34: Příkaz pro vytvoření GUI v novém skriptu

3. Importování nového skriptu do ovládací aplikace

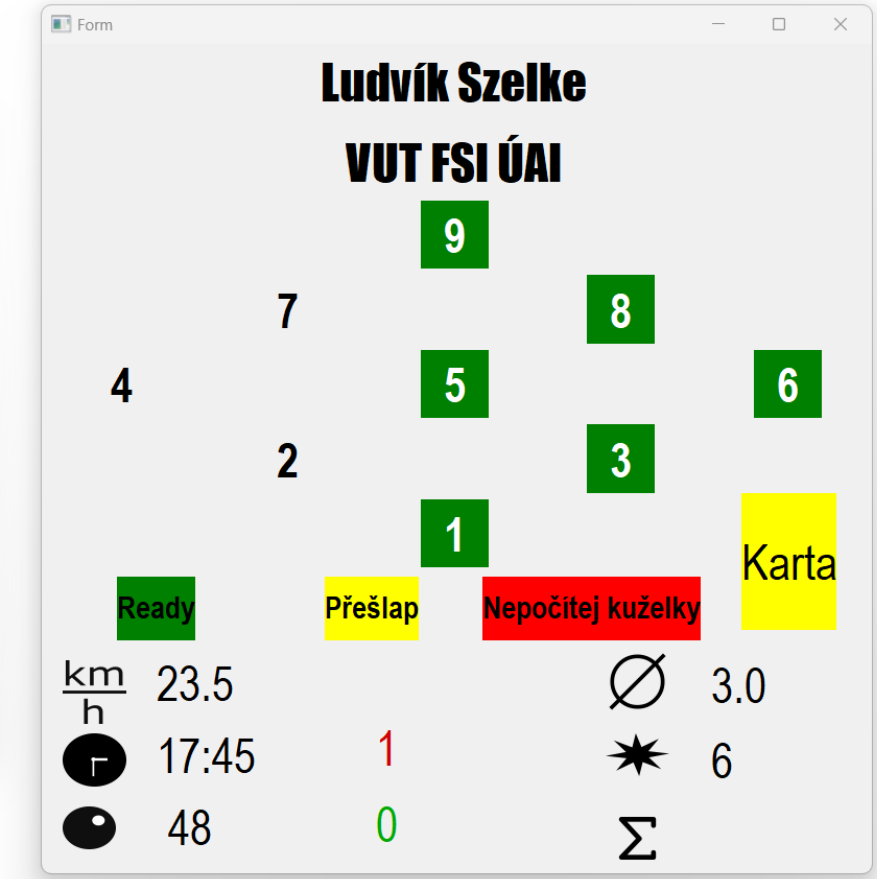
```
from PyQt5 import uic
with open('zobrazovaci_ui.py', 'w', encoding='utf-8') as f:
    uic.compileUi('zobrazovaci.ui', f)

from zobrazovaci_ui import Ui_Form
```

Výpis 35: Importování GUI z python souboru

4. Vymazání všech původních nefunkčních údajů například původní Qlabel, QpushButton atd. o předchozím GUI.

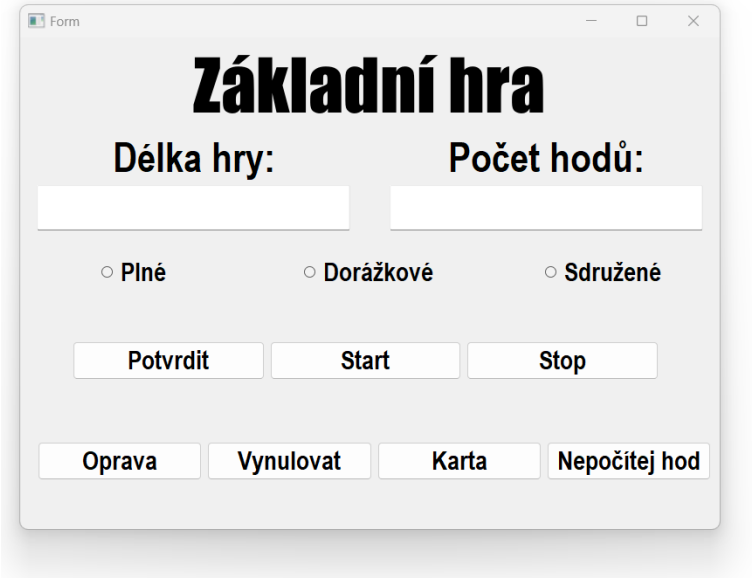
3.6.2 Nové grafické rozhraní obrazovky zobrazovací aplikace



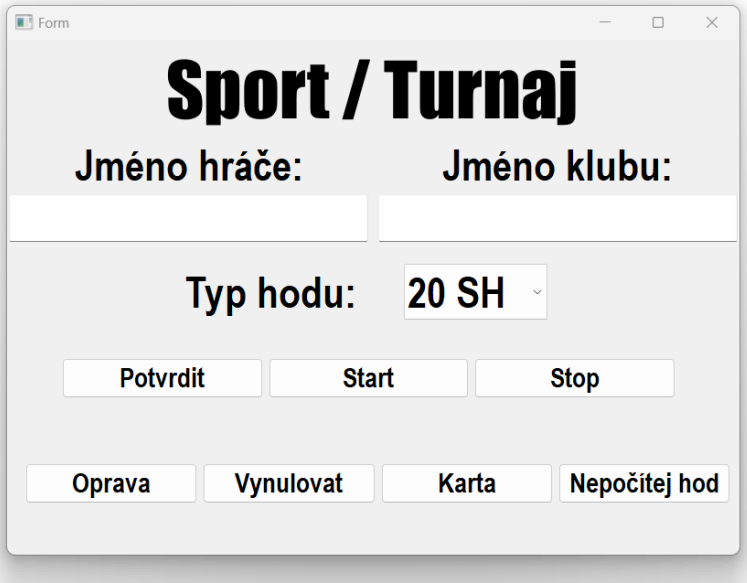
Obr. 12: Nové GUI zobrazovací aplikace

3.6.3 Nové grafické rozhraní obrazovek ovládací aplikace

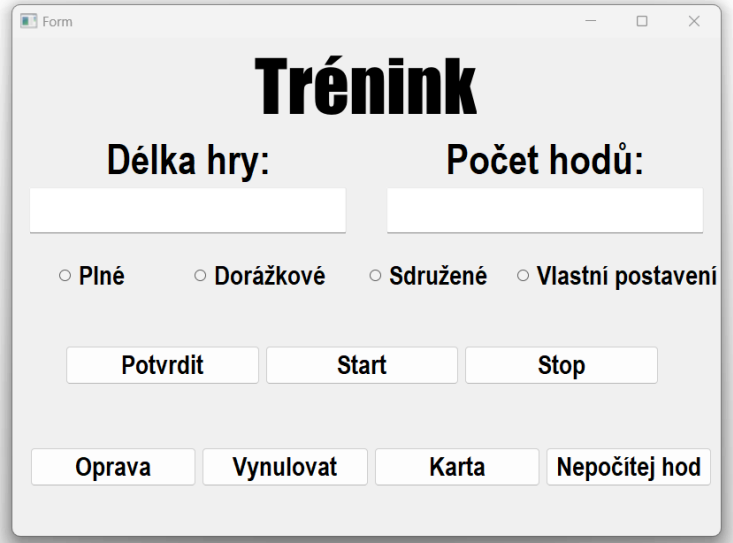
Obr. 13: Nové GUI hlavní obrazovky ovládací aplikace



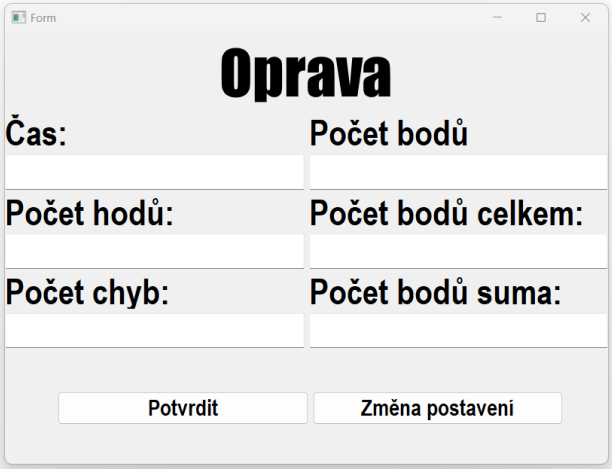
Obr. 14: Nové GUI obrazovky základní hry



Obr. 15: Nové GUI obrazovky Sportovní hry



Obr. 16: Nové GUI obrazovky Tréninkové hry



Obr. 17: Nové GUI obrazovky Oprava

Obr. 18: Nové GUI obrazovky Vlastní postavení

4 TESTOVÁNÍ

4.1 Praktické zkoušky

4.1.1 Před úpravou GUI

Během vývoje aplikací bylo nutné mnohokrát zapínat a vypínat aplikace, analyzovat chyby, opravovat kód tak, aby aplikace pracovaly, jak mají. V této fázi vývoje si stojím za tím, že aplikace spolu zvládají komunikovat bez problémů v daném pořadí. Ovládací aplikace umí po stisknutí tlačítek posílat všechny předdefinované zprávy, a pokud je server aktivní a běží metoda přijímaní dat, všechna data umí přijmou, zpracovat a uložit.

Zobrazovací aplikace, která je nastavena tak, že pořád naslouchá, jestli nedorazil od serveru příkaz, pracuje více méně bezchybně, s výjimkou toho, že neumí zobrazit údaj o celkovém počtu shozených kuželek za všechny hry.

Server aplikace pracuje přesně tak, jak byla nastavena. Postupně čeká na jednotlivé příkazy. Všechny je umí zpracovat a odeslat klientům.

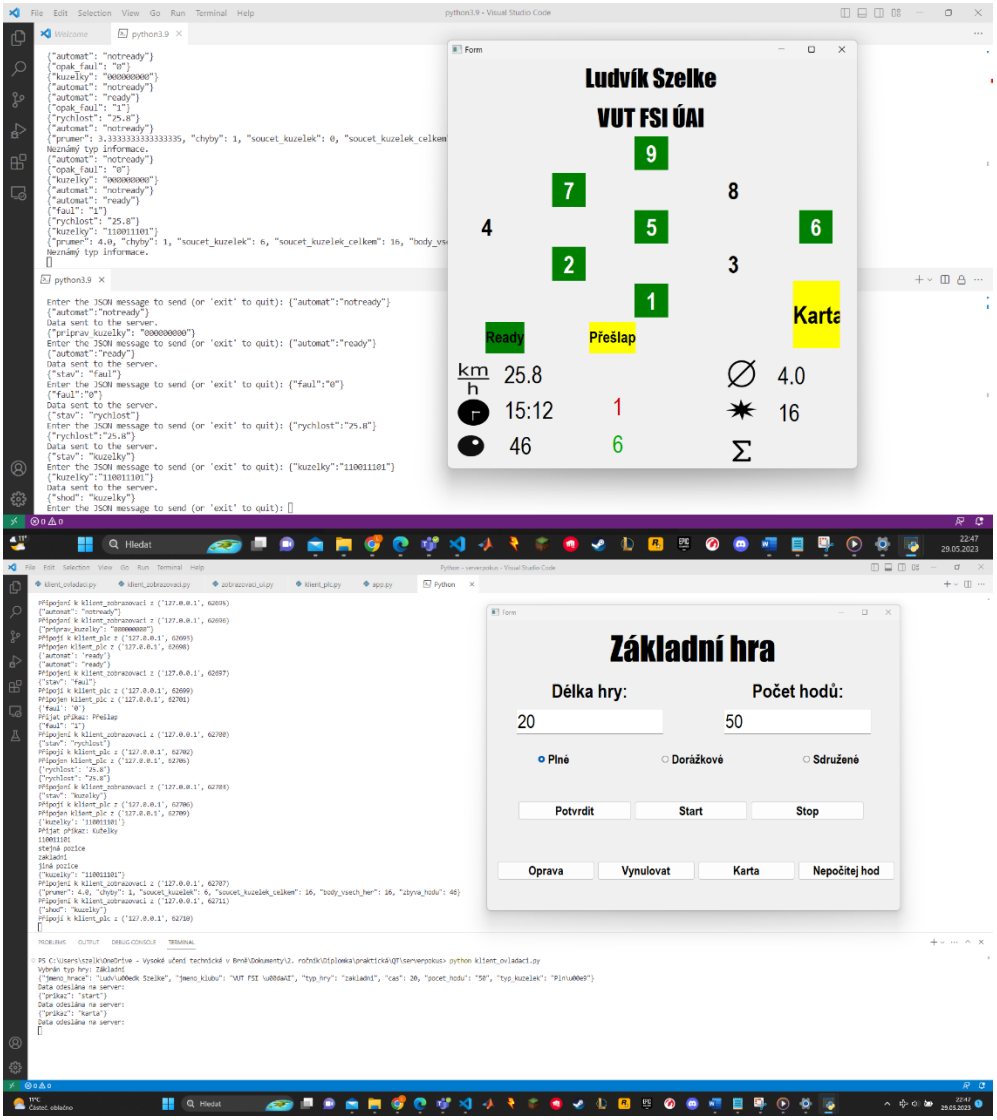
4.1.2 Po úpravě GUI

Při úpravě grafického rozhraní mohlo vlivem přejmenování názvu buttonu nebo labelu dojít k porušení nějaké funkčnosti. U zobrazovacího panelu k tomu nedošlo, má celkem málo vstupních informací, a proto i po úpravě GUI funguje bezchybně, až na problém se zobrazováním celkového počtu bodu za všechny hry.

Toto už se nedá říci o ovládací aplikaci. Ta byla před úpravou GUI bezchybná, bohužel nyní některá tlačítka nereagují. Výjimkou je ovšem obrazovka Základní hry, která funguje stále stejně. Na obrazovce Trénink bylo změněno tlačítko volby vlastního postavení, protože takto to dává větší logiku při volbě herních údajů, a proto je zde nutná úprava.

Server aplikace nemá GUI, a proto se nemusíme bát, že by fungovala jiným způsobem než předtím.

Na následujícím obrázku je ukázáno praktické fungování všech aplikací. Je možné, si všimnou výpisy chodu zpráv v jednotlivých příkazových řádcích aplikací. Celkově lze říct, že původní programy fungují téměř bezchybně a programy po úpravě GUI budou potřebovat opravit názvy labelu nebo některých tlačítek, aby fungovaly tak jako před úpravou.



Obr. 19: Ukázka testování aplikací

5 ZÁVĚR

V rámci řešení této diplomové práce byl navržen koncepční systém programů pro hru evropské kuželky, který zajišťuje ovládání hry, hraní hry a zobrazování dat hry. Je nutné upozornit, že se jedná stále o programy ve fázi vývoje. Prvotním krokem bylo teoretické navržení systému programů, společně s teoretickým návrhem jejich komunikace. Dále následovala teoretická část návrhu společné komunikace programů a teoretický návrh algoritmu hraní hry.

Dalším krokem byla již samotná implementace programů společně, s podrobným popisem a vysvětlením všech metod a funkcí, pomocí výpisů ze skript jednotlivých programů. Nejprve se takto popsal princip funkce a implementace server programu, následoval klient ovládací aplikace, klient PLC aplikace a jako poslední byl představen klient zobrazovací aplikace. Poslední část vývoje spočívala v přepracování původního grafického rozhraní ovládací aplikace a zobrazovací aplikace. V této části je popsán postup tvorby nových grafických rozhraní a na závěr je zde prezentován přehled jednotlivých nových obrazovek programů.

V poslední kapitole této práce bylo provedeno testování, kde se pomocí simulování určitých typů her testovalo chování aplikací, které běžely na stejném zařízení. Toto bylo provedeno z důvodu časové tísně, ale jsem přesvědčen, že pokud by bylo nutné testovat aplikace na separátních zařízeních, byla by pouze provedena změna IP adres v počátečním nastavení aplikací a systém by měl pracovat úplně stejným způsobem. Testování odhalilo několik chyb finální ovládací aplikace, které jsou způsobeny přepracováním grafického rozhraní aplikace. Zjednodušeně řečeno pasivní prvky finální aplikace, jako jsou například údaje o názvu tlačítek, nejsou shodné s původním pojmenováním těchto tlačítek, vlivem například překlepů při tvorbě. Tento problém má velice jednoduché řešení, a to kontrolu a následnou opravu jmen tlačítek. Dále bylo zjištěno, že zobrazovací aplikace nezobrazuje data o celkovém počtu shozených kuželek za všechny hry. Toto je s velkou pravděpodobností způsobeno chybným odesíláním dat typu JSON ze serveru, protože při příjmu vypočítaných dat od serverové aplikace bylo vytištěno do konzole zobrazovací aplikace chybové hlášení: „Neznámý typ informace“. Tato chybová hláška říká, že nebylo rozpoznáno některé z klíčových jmen řetězce JSON, pravděpodobně to o celkovém počtu shozených kuželek za všechny hry.

Co se týče činnosti serverové aplikace, tak pracuje přesně tak, jak je nastavena. Zde by bylo potřeba provést několik změn, například vložit algoritmus do své vlastní metody společně s některými operacemi v algoritmu, jako např. výpočet hodnot nebo příprava kuželek, pro lepší čtení a přehlednost kódu. Dále by pro průmyslovou distribuci bylo nutné doimplementovat stavový automat, který by sloužil k lepšímu chodu serverové aplikace. Nejspíše by bylo dobré také upravit jazyk názvů metod a proměnných v programu, aby se lépe chápala struktura programu v zahraničí. Všechny programy, které byly při této diplomové práci vytvořeny se nachází v příloze A.

SEZNAM POUŽITÉ LITERATURY

[1] WIKIPEDIE. Kuželky [online]. [cit. 29. května 2023]. Dostupné z: <https://cs.wikipedia.org/wiki/Ku%C5%BEelky>. --- obrázek starý kuželky

[2] KUŽELKY.cz. [online]. [Cit. 29. května 2023 Dostupné z: https://www.kuzelky.cz/o-kuzelkach/]

[3] KUŽELKY.cz. [online]. [Cit. 29. května 2023] Dostupné z: https://www.kuzelky.cz/logo/ - obrázek loga

[4] KUŽELKY.cz. Pravidla koulového sportu [online]. [Cit. 29. května 2023] Dostupné z: https://www.kuzelky.cz/dokumenty/predpisy/Pravidla-kuzelkarskeho-sportu.pdf

[5] Create Vista. Stock photo: Bowling strike shows skittles game [online]. [Cit. 29. května 2023]. Dostupné z: https://create.vista.com/cs/unlimited/stock-photos/32854081/stock-photo-bowling-strike-shows-skittles-game/ ----obrázek strike

[6] OLYMPIJSKÝ TÝM ČR. "Sport." [online]. [Cit. 29. května 2023]. Dostupné z: https://www.olympijskytym.cz/sport/67. [Cit. 29. května 2023]

[7] Kuželky nebo bowling? Bowlingář se na kuželkách nechytne, tvrdí hráči [online]. [cit. 2023-05-29]. Dostupné z: https://www.irozhlas.cz/sport_ostatni-sporty/kuzelky-nebo-bowling-bowlingar-se-na-kuzelkach-nechytne-tvrdi-hraci_201701092056_nsramkova

[8] Olympic History: from the home of Zeus in Olympia to the modern Games [online]. [cit. 2023-05-29]. Dostupné z: https://olympics.com/ioc/ancient-olympic-games/history

[9] Scoreboard Nissen Scoremaster "Cadet." [online]. [cit. 2023-05-29]. Dostupné z: https://americanhistory.si.edu/collections/search/object/nmah_1372122

[10] The History Of Digital Signage [online]. [cit. 2023-05-29]. Dostupné z: https://www.skykit.com/blog/history-digital-signage/

[11] The History Of Digital Signage [online]. [cit. 2023-05-29]. Dostupné z: https://digitalsignagepress.com/blog/history-digital-signage/

[12] What is Digitalization? [online]. [cit. 2023-05-29]. Dostupné z: https://www.walkme.com/glossary/digitalization/

[13] Digital revolution [online]. [cit. 2023-05-29]. Dostupné z: https://www.techopedia.com/definition/23371/digital-revolution

[14] THE ADVANTAGES AND DISADVANTAGES OF DIGITALISATION [online]. [cit. 2023-05-29]. Dostupné z: https://startsmarter.co.uk/the-advantages-and-disadvantages-of-digitalisation/

[15] Digitalizace podnikání – nutnost dnešní doby [online]. [cit. 2023-05-29]. Dostupné z: digitalizace: https://blog.newlogic.cz/digitalizace-podnikani-nutnost-dnesni-doby/

[16] WICKES HINE, Lewis . *1:00 A.M. Pin boys working in Subway Bowling Alleys, New York (State).* [online]. duben 1910 [vid. 2023-05-29]. Dostupné z: https://commons.wikimedia.org/wiki/File:Pinboys_nclc.04636.jpg

[17] NEUMAN, William. How Long Till Next Train? The Answer Is Up in Lights. *The New York Times* [online]. 2007 [vid. 2023-05-29]. ISSN 0362-4331. Dostupné z: https://www.nytimes.com/2007/02/17/nyregion/17train.html

[18] Architektura klient-server (Client–server model) [online]. [cit. 2023-05-29]. Dostupné z: https://managementmania.com/cs/architektura-klient-server

[19] KLIENT-SERVER [online]. [cit. 2023-05-29]. Dostupné z: https://site-pc.webnode.cz/klient-server/

[20] CLIENT-SERVER ARCHITECTURE [online]. [cit. 2023-05-29]. Dostupné z: https://medium.com/@nityachampion/client-server-architecture-e9806f7538a9

[21] Back-End System [online]. [cit. 2023-05-29]. Dostupné z: https://www.techopedia.com/definition/1405/back-end-system

[22] FRONTEND VS. BACKEND [online]. [cit. 2023-05-29]. Dostupné z: https://www.czechitas.cz/blog/frontend-vs-backend?gclid=Cj0KCQjwmtGjBhDhARIsAEqfDEcNP7ASzLWiCidkWuxiXskkyPl2-UZm5gCwmpzB2tsJlAjCAIo9IkoaAoE9EALw_wcB

[23] Frontend VS Backend – What's the Difference? [online]. [cit. 2023-05-29]. Dostupné z: https://www.freecodecamp.org/news/frontend-vs-backend-whats-the-difference/

SEZNAM OBRÁZKŮ

SEZNAM VÝPISŮ

SEZNAM PŘÍLOH

A Diplomová práce Ludvík Szelke - programy.zip

SEZNAM ZKRATEK

IBA	International Bowling Association
ASČK	Asociace československého sportu kuželkářského
SČK	Svaz českých kuželkářů
ČAKS	Československá asociace kuželkářského sportu
CMS	Content Management Systém – Systém správy obsahu
PLC	Programmable logic controller – Programovatelný logický ovladač
JSON	JavaScript Object Notation – Zápis objektů v jazyce JavaScript