



Zadání diplomové práce

Název:	Cloudový bot na hru Metin2
Student:	Bc. Daniel Jankech
Vedoucí:	Ing. Jaroslav Kuchař, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

V dnešní době existuje velké množství řešení pro automatické ovládání hier (tzv. bot) za různými účely, jsou ale primárně určeny pro lokální počítač. V rámci této závěrečné práce je cílem se zaměřit na cloudové řešení pro zvolenou hru: Metin2. Práce bude využívat znalosti z computer-vision, cloudových služeb a serverových aplikací. Na komunikaci s botem bude vytvořený interface využívající naši vystavenou API.

- Preskúmajte podobné riešenia existujúcich open-source botov všeobecne, aj pre zvolenú hru Metin2.
- Preskúmajte možnosti cloudových služeb podporujících computer vision za účelom spustenia bota a jeho ovládania.
- Navrhnete sadu úloh, ktoré sú vhodné k automatizácii.
- Navrhnete architekturu celého riešenia, použité technológie a bota pre automatizáciu činností.
- Implementujte bota, ktorý bude podporovať zvolené činnosti.
- Výsledné riešenie otestujte.
- Diskutujte o riešení, o jeho výhodách a nevýhodách a prípadne obecné využiteľnosti.

Diplomová práce

CLOUDOVÝ BOT NA HRU METIN2

Bc. Daniel Jankech

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedúci: Ing. Jaroslav Kuchař, Ph.D.
4. mája 2023

České vysoké učení technické v Praze
Fakulta informačních technologií

© 2021 Bc. Daniel Jankech. Všetky práva vyhradené.

Táto práca vznikla ako školské dielo na FIT ČVUT v Prahe. Práca je chránená medzinárodnými predpismi a zmluvami o autorskom práve a právach súvisiacich s autorským právom. Na jej využitie, s výnimkou bezplatných zákonných licencií, je nutný súhlas autora.

Odkaz na túto prácu: Jankech Daniel. *Cloudový bot na hru Metin2*. Diplomová práca. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Pod'akovanie	vii
Vyhlásenie	viii
Abstrakt	ix
Úvod	1
1 Analýza a návrh riešenia	3
1.1 Načrtnutie aktuálnej situácie	3
1.2 Existujúce riešenia	3
1.3 Počítačové videnie - detekcia objektov	4
1.3.1 OpenCV	5
1.3.2 Tradičné techniky detekcie objektov	5
1.3.3 Techniky detekcie objektov založené na deep-learningu	7
1.3.4 Haar Feature-based Cascade Classifiers	8
1.3.5 Identifikácia možných problémov	9
1.4 Cloud bot ako stavový stroj	10
1.4.1 Návrh úloh	10
1.4.2 Návrh dosiahnuteľných stavov a prechodné podmienky	11
1.4.3 Požiadavky na cloudové služby	13
1.4.4 Prieskum cloudových providerov	13
1.4.5 Potrebné nástroje k tvorbe vlastného Windows ISO obrazu	14
1.4.6 API	18
1.4.7 Databáza	19
1.5 Architektúra	21
1.5.1 Funkčné požiadavky	21
1.5.2 Nefunkčné požiadavky	21
1.5.3 Návrh databázových modelov	22
1.5.4 Ovládanie viacerých cloudových inšancií	23
2 Realizácia	25
2.1 Python	25
2.2 Detekcia objektov	25
2.2.1 Príprava vstupných dát	25
2.2.2 Tréning	28
2.3 Bot	33
2.3.1 Herné ovládanie	33
2.3.2 Štatistické údaje	34
2.3.3 Flask API	34
2.3.4 Interface na ovládanie bota	36
2.4 Dockerizácia služieb	37
2.4.1 Dockerfile a docker-compose	38
2.5 Cloudové riešenie	39

2.5.1	Voľba cloudového providera	39
2.5.2	Vnorená virtualizácia	39
2.5.3	Konfigurácia siete	40
2.5.4	Tvorba snímky systému Windows	41
2.5.5	Spustenie bota a pridružených služieb	43
3	Testovanie riešenia	45
3.1	Statické testovanie kódu	45
3.2	Testovanie efektivity bota	46
4	Diskusia	49
5	Záver	51
A	Zoznam skratiek	53
	Obsah príloženého média	57

Zoznam obrázkov

1.1	Typy featur [18]	9
1.2	Stavový diagram	13
1.3	Tabuľka úloh	22
1.4	Tabuľka herných údajov	23
1.5	Tabuľka herných štatistík	23
2.1	OpenCV anotačný nástroj	27
2.2	Vzorka č. 1	28
2.3	Vzorka č. 2	28
2.4	Výledky tréningu č.1	30
2.5	Výledky tréningu č.2	31
2.6	Postman interface	36
2.7	Typy featur [18]	39
2.8	Zakázaná vnorená virtualizácia [35]	40
2.9	Povolená vnorená virtualizácia [35]	40
2.10	Kontrola bežiacich kontajnerov	41
2.11	Kontrola bežiacich kontajnerov	43

Zoznam tabuliek

1.1	Tabuľka porovnania cloudových poskytovateľov	14
3.1	Testovací beh č. 1	46
3.2	Testovací beh č. 2	47
3.3	Testovací beh č. 3	47

Zoznam výpisov kódu

2.1	Vytváranie pozitívnych a negatívnych snímok obrazovky	26
2.2	Skript na tvorbu popisných súborov	27
2.3	Ukážka neg.txt súboru	27
2.4	Spustenie anotačného nástroja	27

2.5	Ukážka pos.txt súboru	27
2.6	Tvorba vektorového súboru	28
2.7	Tréningová konfigurácia č.1	29
2.8	Výstup tréningovej konfigurácie č.1	29
2.9	Tréningová konfigurácia č.2	30
2.10	Výstup tréningovej konfigurácie č.2	30
2.11	Tréningová konfigurácia č.3	31
2.12	Výstup tréningovej konfigurácie č.3	31
2.13	Ukážka mapovania kláves	33
2.14	Metóda <code>press_key</code>	33
2.15	Ukážka API volania č.1	35
2.16	Ukážka API volania č.2	35
2.17	API Dockerfile	38
2.18	Docker-compose	38
2.19	Kontrola otvorených portov	41
2.20	Tvorba install.wim súboru	42
2.21	Cesta k install.wim	42
2.22	Cesta k install.wim	42
2.23	Build docker obrazu bota	43
2.24	Spustenie bota	43
3.1	Spustenie bota	45
3.2	Spustenie bota	45
3.3	Spustenie bota	45
3.4	Spustenie bota	46
3.5	Spustenie bota	46

*Rád by som poďakoval vedúcemu práce, Ing. Jaroslav Ku-
chařovi, Ph.D., za veľmi užitočné rady, rýchlu odozvu a časovú fle-
xibilitu pri vypracovaní tejto práce.*

Vyhlásenie

Vyhlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým usmernením o dodržiavaní etických zásad pri vypracovaní vysokoškolských prác.

Beriem na vedomie, že sa na moju diplomovú prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Z. z. Autorský zákon v znení neskorších predpisov, najmä skutočnosť, že České vysoké učení technické v Prahe má právo na uzavretie licenčnej zmluvy na použitie tejto práce ako školského diela podľa § 60 odst. 1 citovaného zákona.

V Prahe dňa 4. mája 2023

.....

Abstrakt

Táto diplomová práca navrhuje a realizuje vývoj cloudového bota pre online hru Metin2, ktorý bude určený na vykonávanie úloh, ako je napríklad farmárčenie v hre a získavanie herných zdrojov. Bot bude umiestnený v cloude, čo zabezpečí horizontálnu škálovateľnosť a spoľahlivosť. Vývoj bota pre Metin2 v cloude však predstavuje niekoľko výziev, vrátane návrhu bota, ktorý bude komunikovať s herným klientom nezistiteľne, pokúsi sa pohybovať v prostredí hry na nerozoznanie od ľudského správania a zabezpečí škálovateľnosť a spoľahlivosť v prostredí cloudu. Táto práca skúma technológie a postupy, ktoré možno použiť na vývoj takéhoto bota, a navrhuje riešenia na prekonanie týchto výziev. Navrhovaný cloudový bot pre hru Metin2 má za cieľ zlepšiť herný zážitok hráčov tým, že im umožní efektívnejší postup v hre.

Kľúčová slova cloud, bot, hry, automatizace, počítačové vidění, python

Abstract

This master thesis proposes the development of a cloud-based bot for the massively multiplayer online role-playing game (MMORPG) Metin2, which will be designed to perform tasks such as farming for in-game resources and completing quests. The bot will be hosted in the cloud, providing horizontal scalability and reliability. However, developing a cloud-based bot for Metin2 poses several challenges, including designing the bot to interact with the game's client undetectably, try to navigate the game's environment indistinguishably from human behavior, and ensure scalability and reliability in a cloud-based environment. This thesis explores the technologies and techniques that can be used to develop such a bot and proposes solutions to overcome these challenges. The proposed cloud-based bot for Metin2 aims to improve the gaming experience for players by enabling them to progress more efficiently through the game.

Keywords cloud, bot, games, automatization, computer-vision, python

Úvod

Herný priemysel prešiel v posledných rokoch obrovskou transformáciou. Vďaka technologickému pokroku sa online hry stali dostupnejšími ako kedykoľvek predtým. Hráči z celého sveta sa môžu spojiť a komunikovať v reálnom čase, vďaka čomu sú online hry medzi hráčmi veľmi obľúbené. Jednou z takýchto hier je Metin2, masívne multiplayerová online hra na hrdinov (MMORPG), ktorú vyvinula spoločnosť Ymir Entertainment Co. Ltd.

Ako každá online hra, aj Metin2 si vyžaduje serverovú infraštruktúru na podporu jej prevádzky. V posledných rokoch sa cloud computing stal obľúbenou voľbou vývojárov hier, ponúka mnoho výhod vrátane škálovateľnosti, spoľahlivosti a cenovej efektívnosti.

Bežným javom je aj používanie botov v online hrách. Boti sa môžu používať na vykonávanie opakujúcich sa úloh, ako je napríklad nejaký spôsob farmárčenia pre herné zdroje, bez akejkoľvek potreby ľudského zásahu. To umožňuje hráčom sústrediť sa na iné nie tak ľahko zautomatizovateľné činnosti a postupovať tak v hre rýchlejšie a efektívnejšie.

V tejto diplomovej práci navrhujeme a následne realizujeme vývoj cloudového bota pre hru Metin2. Bot bude navrhnutý tak, aby vykonával nami zadané úlohy, ako napríklad vyššie spomenuté farmárčenie herných prostriedkov, čo v preklade do praxe bude znamenať automatické vyhľadanie a následne zničenie kameňov Metin. Bot bude umiestnený v cloude, čo umožní horizontálnu škálovateľnosť a spoľahlivosť.

Vývoj cloudového bota pre Metin2 predstavuje niekoľko výziev. Po prvé, bot musí byť schopný komunikovať s herným klientom tak, aby ho *anti-cheat* mechanizmy v hre neodhalili. Po druhé, bot musí byť schopný pohybovať sa v prostredí hry a vykonávať úlohy spôsobom, ktorý je na nerozoznanie od ľudského správania. A nakoniec, bot musí byť navrhnutý tak, aby bol škálovateľný a spoľahlivý v cloudovom prostredí.

V tejto práci budeme skúmať rôzne technológie a techniky, ktoré sa dajú použiť na vývoj bota pre hru Metin2 nasadeného do cloudového prostredia. Budeme tiež diskutovať o výzvach spojených s vývojom takéhoto bota a navrhujeme riešenia na prekonanie týchto výziev.

Celkovo je cieľom tejto magisterskej práce prispieť do oblasti online hier návrhom vývoja cloudového bota pre Metin2. Bota sme navrhli tak, aby zlepšil zážitok hráčov z hry tým, že im umožní efektívnejší postup v hre najmä prostredníctvom automatizácie získavania herných zdrojov.

Analýza a návrh riešenia

1.1 Načrtnutie aktuálnej situácie

Metin2 je masívne multiplayerová online RPG hra (MMORPG), ktorá bola prvýkrát vydaná v Kórei v roku 2004 a odvtedy sa stala populárnou po celom svete. V Metin2 si hráči môžu vytvoriť postavy a preskúmať rozprávkový svet plný príšer, rôznych výziev a iných hráčov.

Jedným z hlavných cieľov hry je zvyšovať úroveň svojho charakteru a získavať nové vybavenie zabíjaním príšer a plnením úloh. To však môže byť časovo náročný a opakujúci sa proces, ktorý si vyžaduje veľa repetitívnej práce, čo niektorí hráči považujú za únavné a otravné. Na vyriešenie tohto problému sa hráči obracajú na používanie botov, čo sú softvérové programy určené na automatizáciu procesu farmenia alebo iných činností. Týchto botov možno naprogramovať tak, aby vykonávali zadané činnosti, ako je zabíjanie príšer, zbieranie predmetov a plnenie úloh, bez toho, aby sa hráč musel aktívne zapájať do hry.

Zatiaľ čo niektorí hráči považujú takýchto botov pre hru Metin2 ako spôsob, ktorým ušetriť čas a spríjemniť si hru, iní ich považujú za podvádzanie a neférovosť voči ostatným hráčom. Používanie botov v online hrách je často v rozpore s pravidlami hry a môže viesť k sankciám, ako je pozastavenie účtu alebo jeho zablokovanie.

Je dôležité poznamenať, že používanie botov v online hrách nie je ojedinelé pre hru Metin2 a je bežným problémom aj v mnohých iných online hrách. Vývojári hier často prijímajú opatrenia na zabránenie používaniu botov, napríklad určitá forma *captcha* verifikácie (In-game captcha), pri podozrivom správaní administrátori serveru zablokujú účty hráčom. Z etického dôvodu vývoj nášho bota bude prebiehať na neaktívnom serveri, kde táto aktivita neznevýhodňuje žiadnych iných hráčov.

Stručne povedané, farmárske roboty Metin2 sú softvérové programy, ktoré automatizujú proces farmárčenia alebo iných činností v online hre Metin2. I keď niektorí hráči ich môžu považovať za užitočné, ich používanie sa často považuje za podvádzanie a môže viesť k sankciám zo strany správcov herného servera.

1.2 Existujúce riešenia

V tejto kapitole si preskúmame open-source riešenia Metin2 botov. Prieskum je dôležitý najmä z dôvodu dôkladnej analýzy našej problematiky, určenia funkčných a určenia nefunkčných požiadaviek. Rád by som začal tým, že podľa mojich znalostí cloudová implementácia zatiaľ neexistuje, všetky nájdené projekty sú vytvorené tak, aby bežali lokálne. Cloudové riešenie je však len forma nasadenia, priblížime si teda niekoľko z nich.

Prvým takýmto projektom je **nounix-metin2-farmbot**, ktorý funguje na princípe injekcie herného enginu.

Injekcia herného enginu je typ techniky exploitácie softvéru, ktorá zahŕňa manipuláciu s jadrovým kódom za cieľom získať neoprávnený prístup k údajom hry alebo ovládať jej správanie. Funguje tak, že využíva zraniteľnosti alebo slabé miesta, napríklad pretečenie vyrovnávacej pamäte alebo únik pamäte, na účely injektovania vlastného kódu do pamäte hry. Tento injektovaný kód sa potom môže použiť na manipuláciu s dátami alebo správaním hry, napríklad na úpravu atribútov hráča alebo obídenie anti-cheat opatrení[1].

Injekcia môže byť riskantná a nezákonná činnosť, pretože zahŕňa úpravu kódu hry spôsobom, ktorý nebol zamýšľaný jej vývojármi. Okrem toho vývojári hier často používajú opatrenia proti takýmto podvodom a bezpečnostné opatrenia na odhalenie a zabránenie injekcie samotnej, takže použitie tejto techniky má najčastejšie za následok blokáciu herného účtu.

Je dôležité poznamenať, že injektovanie herného enginu sa nie vždy používa na škodlivé účely. Niektorí *modderi* používajú túto techniku na vytváranie vlastných rozšírení do hry (inak módov), ktoré zlepšujú hrateľnosť alebo pridávajú do hry nové funkcie.

Táto implementácia teda priamym spôsobom ovplyvňuje herné mechaniky a dovoľuje hráčovi zachádzať za limity jeho charakteru. Ako som však už vyššie spomenul, takýto typ botov je veľmi ľahko detekovaný, keďže priamo interaguje s jadrovým kódom klienta. Ak teda Metin2 server využíva akýkoľvek anti-cheat systém, ktorý kontroluje herné súbory, konto je automaticky zablokované.

Druhým veľmi podobným projektom je **notsaki-Mt2-Stone-Farm-Bot**. Tento projekt už je veľmi podobný svojou metodikou nášmu projektu. Využíva image-processing na automatické farmenie kameňov Metin. Na detekciu herných objektov, využíva metódu template-matching, o ktorej si povieme viac v sekcii **Tradičné techniky detekcie objektov**. Bot je taktiež navrhnutý ako stavový stroj. Funkcionalita tohto bota, je veľmi podobná našej zamýšľanej funkcionalite, hlavným rozdielom však zostáva metóda detekcie objektov a taktiež metóda nasadenia.

1.3 Počítačové videnie - detekcia objektov

Počítačové videnie je oblasť výskumu, ktorá sa zameriava na umožnenie strojom interpretovať a pochopiť vizuálne údaje z okolitého sveta [2]. S rýchlym pokrokom v oblasti strojového učenia a počítačového hardvéru sa počítačové videnie v posledných rokoch stáva čoraz dôležitejšou oblasťou výskumu a vývoja [3]. Jednou z najdôležitejších úloh v oblasti počítačového videnia je detekcia objektov, ktorá zahŕňa identifikáciu a lokalizáciu objektov záujmu v obraze alebo videoprenose [4].

Detekcia objektov má množstvo aplikácií v rôznych oblastiach okrem toho herného, ktorý by som povedal, je práve menšinovou. Sú to oblasti vrátane robotiky, autonómneho riadenia, zdravotníctva a športovej analýzy. Napríklad pri autonómnej jazde je detekcia objektov rozhodujúca pre identifikáciu a sledovanie iných vozidiel, chodcov a taktiež aj prekážok v prostredí. V zdravotníctve sa detekcia objektov môže použiť na zisťovanie anomálií na lekárskejších snímkach, napríklad nádorov alebo iných nálezov [5].

Detekcia objektov je náročná úloha vzhľadom na zložitosť a variabilitu vizuálnych údajov, ako aj v mnohých aplikáciách potrebu pre výkon v reálnom čase [4]. V priebehu rokov výskumníci vyvinuli rôzne prístupy na riešenie tejto výzvy, vrátane tradičných techník počítačového videnia a metód založených na *deep-learningu* [6]. Tradičné techniky počítačového videnia sa pri detekcii objektov spoliehajú na ručne vytvorené funkcie a heuristiku, zatiaľ čo metódy založené na hlbokom učení využívajú neuronové siete na automatické učenie funkcií z údajov [3].

V tejto práci sa okrem Cloudového nasadenia systémov zameriame aj na detekciu objektov v hernom svete, pričom poskytneme prehľad tejto oblasti vrátane tradičných techník a techník založených na hlbokom učení, hodnotiacich metrík a ich praktických aplikácií. Budeme tiež diskutovať o obmedzeniach súčasných prístupov a poukážeme na možnosti budúceho výskumu v

oblasti detekcie objektov. Budem rád ak na konci tejto práce bude mať čitateľ hlbšie pochopenie detekcie objektov a jej potenciálneho vplyvu v rôznych aktuálnych oblastiach [2].

1.3.1 OpenCV

OpenCV (Open Source Computer Vision Library) je softvérová knižnica počítačového videnia a strojového učenia s otvoreným zdrojovým kódom. Je napísaná v jazyku C++ a poskytuje zbierku vysoko výkonných algoritmov a nástrojov na spracovanie obrazu a videa, detekciu a extrakciu črt, rozpoznávanie objektov, strojové učenie a ďalšie. Medzi kľúčové funkcie OpenCV patria: Medzi kľúčové funkcie OpenCV patria:

- **Spracovanie obrazu a videa** - Poskytuje širokú škálu algoritmov na spracovanie obrazu a videa vrátane filtrovania, transformácie, segmentácie a detekcie a extrakcie prvkov.
- **Detekcia a rozpoznávanie objektov** - Obsahuje nástroje na detekciu a rozpoznávanie objektov vrátane Haarových kaskád, HOG (histogram orientovaných sklonov) a modelov založených na *deep-learningu*.
- **Strojové učenie** - OpenCV obsahuje množstvo nástrojov strojového učenia na úlohy, ako je klasifikácia, regresia a klastrovanie.

V našej práci budeme využívať OpenCV ako knižnicu, pomocou ktorej budeme detekovať herné objekty a následne s nimi interagovať. V ďalšej sekcii si povieme niečo viac o tradičných a neskôr aj menej tradičných technikách detekcie objektov.

1.3.2 Tradičné techniky detekcie objektov

Tradičné techniky detekcie objektov sa spoliehajú na ručne vytvorené funkcie a heuristiku. Tieto metódy často zahŕňajú sériu krokov spracovania obrazu, ako je detekcia hrán, segmentácia a extrakcia prvkov, po ktorých nasleduje klasifikátor na identifikáciu objektov [2]. V tejto časti sa budeme zaoberať niektorými tradičnými technikami na detekciu objektov.

1.3.2.1 Feature-based object detection

Feature-based object detection alebo aj Detekcia objektov na základe decriptorov je typ techniky počítačového videnia, ktorá sa používa na identifikáciu objektov na obrázkoch alebo videách. Základnou myšlienkou detekcie objektov na základe decriptorov je extrahovať z obrazu súbor príznakov alebo teda deskriptorov, ktoré možno použiť na identifikáciu objektu nášho záujmu. Tieto vlastnosti sú často založené na lokálnych *patternoch* alebo štruktúrach v obraze, ako sú hrany, rohy alebo škvrnky.

Proces detekcie objektov na základe deskriptorov zvyčajne zahŕňa niekoľko krokov. Najprv sa spracuje obrázok alebo snímka videa s cieľom identifikovať potenciálne oblasti, ktoré môžu obsahovať objekt záujmu. Tieto oblasti sa často identifikujú pomocou segmentačných techník, ktoré rozdeľujú obraz na samostatné oblasti na základe farby, textúry alebo iných vizuálnych charakteristík.

Z každej kandidátskej oblasti sa extrahuje súbor znakov alebo deskriptorov. Tieto znaky môžu byť založené na rôznych technikách, ako napríklad *Scale Invariant Feature Transformation* (SIFT), *Speeded-Up Robust Features* (SURF) alebo histogram orientovaných gradientov (HOG). Tieto prvky sú často navrhnuté tak, aby boli odolné voči zmenám osvetlenia, mierky a orientácie[7].

Nakoniec sa extrahované vlastnosti použijú na priradenie kandidátskych oblastí k modelu. Týmto modelom môže byť súbor šablón alebo štatistický model, ktorý opisuje vzhľad objektu v

rôznych polohách a svetelných podmienkach. Proces porovnávania môže zahŕňať porovnanie extrahovaných znakov so znakmi v modeli alebo použitie algoritmov strojového učenia na naučenie klasifikátora, ktorý dokáže rozlíšiť objekt záujmu od iných objektov na obrázku[7].

Celkovo je detekcia objektov na základe deskriptovo výkonnou technikou, ktorú možno použiť na identifikáciu objektov v širokom spektre aplikácií vrátane robotiky, monitorovania a autonómnych vozidiel.

1.3.2.2 Template matching

Template matching je populárny prístup k detekcii objektov v počítačovom videní, pri ktorom sa *template* alebo šablona priraduje k väčšiemu obrázku alebo snímke videa s cieľom identifikovať výskyt našeho *templatu*. *Template* je zvyčajne malý obrázok, ktorý predstavuje objekt záujmu a porovnáva sa s väčším obrázkom alebo videosnímkou na rôznych miestach a v rôznych mierkach[2].

Proces porovnávania *templatov* zahŕňa niekoľko krokov. Najprv sa vyberie *template*, ktorý predstavuje objekt záujmu. Tento *template* sa môže vytvoriť ručne alebo automaticky pomocou vyššie spomenutých rôznych techník, ako je detekcia hrán, detekcia škvŕn alebo strojové učenie[8].

Potom sa teda *template* porovnáva s väčším obrazom alebo snímku videa na rôznych miestach a v rôznych mierkach. To sa vykonáva posúvaním *templatu* po väčšom obraze alebo videosnímkou a výpočtom miery podobnosti medzi *templatu* a porovnávanou časťou väčšieho obrazu alebo videosnímkou. Miera podobnosti môže byť založená na technikách, ako je krížová korelácia, súčet štvorcových rozdielov alebo normalizovaná korelácia[8].

Nakoniec sa miesta vo väčšom obraze alebo videosnímkou, ktoré majú najvyššie skóre podobnosti so šablónou, označia za potenciálne prípady výskytu objektu záujmu.

Template matching je jednoduchý a účinný prístup na detekciu objektov na obrázkoch alebo videách, má však obmedzenia. Jedným z hlavných obmedzení je, že je citlivý na zmeny osvetlenia, orientácie a mierky, čo môže mať za následok falošne pozitívne alebo vynechané detekcie.

Napriek svojim obmedzeniam sa *template matching* stále používa v mnohých aplikáciách počítačového videnia, ako je robotika, monitorovanie a lekárske snímanie.

1.3.2.3 HOG-based object detection

Detekcia objektov založená na metóde HOG (Histogramy orientovaných gradientov) je technika počítačového videnia, ktorá sa používa na detekciu objektov na obrázkoch extrahovaním deskriptora histogramu orientovaných sklonov. Deskriptor HOG zachytáva rozloženie smerov lokálnych hrán v obraze alebo snímke videa, ktoré možno použiť na rozlíšenie rôznych objektov na základe ich tvaru a vzhľadu.

Proces detekcie objektov na základe HOG zvyčajne zahŕňa niekoľko krokov, ako je výber súboru trénovaných obrázkov, ktoré obsahujú objekt záujmu, extrakcia deskriptorov - príznakov HOG z týchto obrázkov, tréning klasifikátora na týchto deskriptoroch a použitie trénovaného klasifikátora na detekciu objektu v nových obrázkoch alebo videosnímkach[9].

Detekcia objektov založená na HOG sa použila v širokej škále aplikácií, ako je detekcia tváre, detekcia chodcov a rozpoznávanie objektov.

Dalal a Triggs predstavili detekciu objektov založenú na HOG vo svojom článku z roku 2005 "Histogramy orientovaných gradientov pre detekciu ľudí"[10]. V tomto článku použili deskriptor príznakov HOG na detekciu chodcov na snímkach a dosiahli v tom čase špičkový výkon.

Odvtedy sa detekcia objektov založená na HOG široko rozšírila v komunite počítačového videnia a uplatnila sa v mnohých ďalších aplikáciách okrem detekcie ľudí. Girshick a kol. navrhli variant detekcie objektov na základe HOG s názvom Region-based Convolutional Neural Networks (R-CNN), ktorý dosiahol špičkový výkon v úlohách detekcie objektov a sémantickej segmentácie (Girshick a kol., 2014)[9].

1.3.2.4 Obmedzenia tradičných techník

Tradičné techniky počítačového videnia majú pri úlohách detekcie objektov obmedzenia. Tieto techniky sa spoliehajú na ručne vytvorené prvky, ktoré nemusia byť odolné voči zmenám mierky, rotácie a osvetlenia. Okrem toho tieto techniky môžu vyžadovať rozsiahle ladenie parametrov na dosiahnutie dobrého výkonu, čo môže byť časovo náročné a výpočtovo nákladné. V posledných rokoch sa preto na detekciu objektov čoraz viac využívajú techniky založené na *deep-learningu*.

Celkovo sa tradičné techniky počítačového videnia používajú už mnoho rokov na riešenie problémov detekcie objektov. Hoci tieto techniky majú určité obmedzenia, položili základ pre vývoj pokročilejších techník vrátane prístupov založených na *deep-learningu*.

1.3.3 Techniky detekcie objektov založené na deep-learningu

Techniky detekcie objektov založené na hlbokom učení získali v posledných rokoch značnú popularitu vďaka svojej schopnosti automaticky sa učiť príznaky z dát. Tieto metódy využívajú neurónové siete na extrakciu vlastností z obrazov a predpovedanie prítomnosti a umiestnenia objektov. Jednou z prvých metód detekcie objektov založených na hlbokom učení je R-CNN, ktorá používa sieť návrhu regiónov na generovanie potenciálnych oblastí objektov a konvolučnú neurónovú sieť na ich klasifikáciu a spresnenie [11]. Ďalšou populárnou metódou detekcie objektov založenou na hlbokom učení je You Only Look Once (YOLO), čo je jednotný systém detekcie objektov v reálnom čase, ktorý využíva jednu neurónovú sieť na predpovedanie ohraničujúcich polí a pravdepodobností tried priamo z obrazu[4].

1.3.3.1 Single Shot Detector metóda

SSD alebo Single Shot Detector je metóda detekcie objektov založená na hlbokom učení, ktorá využíva jednu konvolučnú neurónovú sieť na predpovedanie triedy a umiestnenia objektov v obraze. Táto sieť ako vstup berie celý obrázok a ako výstup vytvára súbor ohraničujúcich boxov a k nim priradených pravdepodobností tried. To je v protiklade s inými metódami detekcie objektov, ktoré na detekciu objektov používajú viacero stupňov, ako napríklad rodina metód CNN založených na regiónoch (R-CNN) [12, 13].

Metóda SSD využíva na extrakciu príznakov zo vstupného obrazu kostrovú sieť, ktorá sa potom používa na predpovedanie tried objektov a ohraničujúcich boxov. Sieť na extrakciu príznakov je zvyčajne predtrénovaná konvolučná neurónová sieť, ako napríklad VGG alebo ResNet o ktorej si onedlho povieme viac [12, 13].

Používa viacero konvolučných vrstiev rôznych veľkostí na detekciu objektov v rôznych mierkach a pomeroch strán. Každá konvolučná vrstva je zodpovedná za detekciu objektov s určitým rozsahom veľkosti a pomerom strán. Výsledné mapy príznakov sa potom používajú na predpovedanie triedy a umiestnenia objektov pomocou súboru konvolučných filtrov [12, 13].

Táto metóda taktiež používa aj predvolené boxy, čo sú vopred definované ohraničujúce boxy rôznych mierok a pomerov strán, ktoré sa používajú na predpovedanie polohy objektov. Predvolené boxy sú definované v každej vrstve mapy prvkov a ku každému boxu je priradená sada predpovedaných pravdepodobností tried[12, 13].

Celkovo je metóda SSD známa svojou rýchlosťou a presnosťou a bola použitá na širokú škálu úloh detekcie objektov vrátane detekcie chodcov, vozidiel a tvárí.

1.3.3.2 You Only Look Once metóda

š

You Only Look Once (YOLO) je populárna metóda detekcie objektov založená na hlbokom učení, ktorá využíva jednu neurónovú sieť na predpovedanie triedy a umiestnenia objektov v obraze[4].

Metóda YOLO rozdelí vstupný obraz na sieť polí a predpovedá hraničné boxy a pravdepodobnosti tried pre každé políčko. Každý ohraničujúci box je definovaný svojimi stredovými súradnicami, šírkou, výškou a súvisiacimi pravdepodobnostnými triedami. Metóda YOLO používa na tieto predpovede jednu konvolučnú neurónovú sieť, čo umožňuje detekciu objektov v reálnom čase na zariadeniach s nízkou spotrebou energie[4].

Jednou z kľúčových výhod metódy YOLO je jej účinnosť. Keďže vyžaduje iba jeden priamy prechod cez neurónovú sieť, môže vykonávať detekciu objektov v reálnom čase, a to aj na zariadeniach s nízkou spotrebou. Okrem toho je metóda YOLO schopná detekovať objekty v rôznych mierkach a pomeroch strán pomocou viacerých *anchor boxov*, čo sú vopred definované ohraničujúce boxy rôznych veľkostí a tvarov, ktoré sa používajú na predpovedanie polohy objektov[4].

Používa sa v širokom spektre aplikácií vrátane detekcie chodcov, vozidiel a tvárí. Je však známe, že má problémy s detekciou malých objektov a objektov so zložitými tvarmi, pretože architektúra jednej siete nemusí byť schopná zachytiť všetky potrebné detaily[4].

Celkovo je metóda YOLO rýchlym a efektívnym spôsobom vykonávania detekcie objektov v reálnom čase, vďaka čomu je vhodná pre aplikácie, ktoré vyžadujú rýchlu a presnú detekciu, ako je napríklad autonómne riadenie a video dohľad.

1.3.3.3 RetinaNet metóda

RetinaNet je metóda na detekciu objektov založená na hlbokom učení, ktorú v roku 2018 predstavili výskumníci zo spoločnosti Facebook AI Research. RetinaNet bol navrhnutý na riešenie bežného problému metód detekcie objektov, ktorý spočíva v tom, že majú tendenciu zápasíť s detekciou objektov v rôznych mierkach[14].

Metóda RetinaNet to dosahuje pomocou siete s pyramídou funkcií (FPN), ktorá kombinuje funkcie z viacerých konvolučných vrstiev s rôznym priestorovým rozlíšením. To umožňuje modelu detekovať objekty vo viacerých mierkach a znížiť počet falošne pozitívnych výsledkov[15].

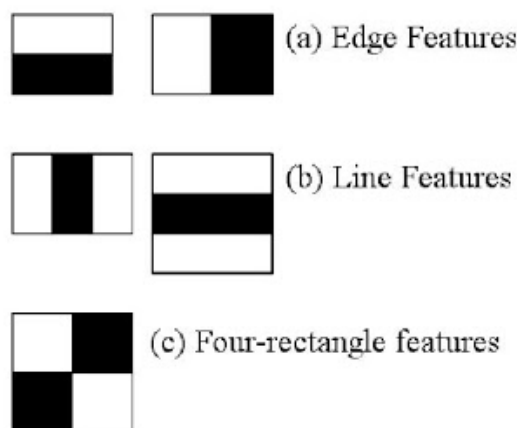
Okrem toho RetinaNet zavádza novú stratovú funkciu s názvom Focal Loss, ktorá je navrhnutá na riešenie problému nevyváženosti tried, ktorý nastáva, keď je v obraze veľa oblastí pozadia a len niekoľko oblastí obsahujúcich objekty. Funkcia Focal Loss priradzuje vyššie váhy ťažkým príkladom, čo sú oblasti, ktoré sú nesprávne klasifikované s vysokou istotou, čím sa zefektívňuje proces tréningu[16].

Technológia RetinaNet využíva aj jednostupňový prístup detekcie, podobný metóde YOLO, pri ktorom sa návrhy ohraničujúcich polí a klasifikácia objektov vykonávajú v jednom priechoode neurónovej siete dopredu. Vďaka tomu je model rýchly a efektívny, čo ho robí vhodným pre aplikácie v reálnom čase[14].

RetinaNet dosiahla špičkové výsledky v niekoľkých referenčných testoch detekcie objektov vrátane COCO a PASCAL VOC a bola použitá v širokom spektre aplikácií vrátane samojazdiacich automobilov, robotiky a video monitorovania.

1.3.4 Haar Feature-based Cascade Classifiers

Haar Feature-based Cascade Classifiers sú populárnou metódou na detekciu objektov, ktorú predstavili Viola a Jones v roku 2001. Metóda používa Haar-like features, čo sú obdĺžnikové príznaky, ktoré počítajú rozdiel medzi súčtom hodnôt pixelov v bielych a čiernych oblastiach objektu[Viola, 17].



■ Obr. 1.1 Typy featur [18]

Táto metóda sa používa na tréovanie kaskádového klasifikátora, čo je séria klasifikátorov, ktoré sú tréované na detekciu čoraz zložitejších prvkov. Kaskádový klasifikátor funguje tak, že na oblasť obrazu sa postupne aplikuje séria binárnych klasifikátorov, pričom každý klasifikátor určuje, či oblasť obsahuje objekt záujmu[Viola, 17].

Metóda je známa svojou účinnosťou a presnosťou, vďaka čomu je vhodná pre aplikácie, ktoré vyžadujú detekciu objektov v reálnom čase, ako je napríklad detekcia tváří v kamerách a bezpečnostných systémoch. Okrem toho sa táto metóda dá tréovať na relatívne malých súboroch údajov, čo z nej robí obľúbenú voľbu pre aplikácie, kde sú údaje obmedzené[Viola, 17].

Jedným z obmedzení metódy kaskádových klasifikátorov na báze Haarových príznakov je, že má problémy s detekciou objektov v rôznych mierkach a orientáciách, ako aj objektov so zložitými tvarmi. Okrem toho si metóda vyžaduje starostlivé ladenie parametrov na dosiahnutie optimálneho výkonu[Viola, 17].

Napriek týmto obmedzeniam zostávajú kaskádové klasifikátory založené na Haarových prvkoch populárnou metódou na detekciu objektov a používajú sa v širokom spektre aplikácií vrátane detekcie tváří, chodcov a vozidiel[Viola, 17].

V našej práci sme sa rozhodli použiť túto metódu, na základe viacerých faktorov. Jedným z faktorov bola jej schopnosť dosiahnuť vysokú presnosť aj s malým vzorkom vstupných dát, ktoré sme museli manuálne zozbierať.

1.3.5 Identifikácia možných problémov

Existuje niekoľko problémov, s ktorými sa možno stretnúť pri používaní nami zvolenej metódy detekcie objektov. Niektoré z nich zahŕňajú:

- 1. Falošne pozitívne výsledky** - Jedným z hlavných problémov pri kaskádových klasifikátoroch založených na Haarových príznakoch je, že môžu generovať falošne pozitívne výsledky, čo znamená, že klasifikátor detekuje objekt, ktorý neexistuje. Môže k tomu dôjsť z viacerých dôvodov, ako sú napríklad rozdiely v osvetlení, tieň alebo šum na obrázku.
- 2. Citlivosť na škálovanie a rotáciu** - Ďalším obmedzením kaskádových klasifikátorov založených na Haarových funkciách je, že môžu byť citlivé na zmeny mierky a rotácie. To znamená, že klasifikátor nemusí byť schopný odhaliť objekty, ktoré sú menšie alebo väčšie ako tréované údaje, alebo objekty, ktoré sú naklonené alebo otočené.
- 3. Obmedzené na detekciu špecifických typov objektov** - Kaskádové klasifikátory založené na Haarových príznakoch sú zvyčajne tréované na detekciu špecifických typov objektov, ako

sú napríklad tváre alebo autá. Pri použití na detekciu iných typov objektov nemusia dosahovať dobré výsledky.

Všetky vyššie uvedené možné problémy však dokáže vyriešiť dostatočne dobrý tréning modelu s ktorým súvisí veľkosť vstupných dát a odladenie parametrov. Práve preto bude tréning nášho modelu kľúčovou časťou úspešnej detekcie objektov.

1.4 Cloud bot ako stavový stroj

Cloud computing sa stal populárnou formou poskytovania výpočtových zdrojov na požiadanie prostredníctvom internetu. Umožnil organizáciám využívať výhody distribuovaných výpočtových systémov, ako sú škálovateľnosť, spoľahlivosť a nákladová efektívnosť, bez toho, aby museli spravovať základnú infraštruktúru. Správa a orchestrácia cloudových zdrojov však môže byť náročná, najmä v zložitých a dynamických prostrediach[19].

Jedným z prístupov k riešeniu tejto výzvy je používanie stavových strojov, čo sú modely, ktoré opisujú správanie systému definovaním súboru stavov a udalostí, ktoré spúšťajú prechody medzi týmito stavmi. Stavové stroje sa široko používajú v rôznych oblastiach, ako je softvérové inžinierstvo, riadiace systémy a siete, na modelovanie komplexných systémov a riadenie ich správania[20].

V poslednom čase rastie záujem o používanie stavových strojov v cloud computingu na automatizáciu správy a orchestrácie cloudových zdrojov. Jedným zo sľubných prístupov je používanie cloudových botov, čo sú softvérové programy, ktoré komunikujú s cloudovými zdrojmi prostredníctvom API a vykonávajú automatizované úlohy na základe vopred definovaných pravidiel a zásad. Pomocou cloudového bota ako stavového stroja je možné modelovať správanie cloudového systému a automatizovať jeho správu škálovateľným a efektívnym spôsobom[5].

Cieľom tejto práce je navrhnúť a implementovať bota ako stavový stroj, ktorý dokáže spracovať celý rad udalostí a prechodov v prostredí cloudu.

1.4.1 Návrh úloh

Pri návrhu úloh sme premýšľali, ktoré činnosti sú najviac repetitívne v hre Metin2 a zároveň zautomatizovateľné. Pre začiatok sme teda navrhli 2 úlohy:

- **Automatické plášte statočnosti** - Aktivácia pláštú statočnosti spôsobí, že všetky príšery v okolí hráča sa stanú agresívnymi a začnú útočiť. Týmto ich hráč k sebe vie privolať veľké množstvo príšer naraz, bez potreby ich privolať jednotlivo. V praxi tento úkon vyzerá tak, že hráč neustále stláča jednu z kláves, na ktorej tento plášť statočnosti má uložený. Samozrejme vykonávať jeden klik, niekoľko krát za sekundu, počas niekoľkých hodín nieje ideálne. Zvolili sme teda tento úkon ako prvú úlohu k automatizácii. Túto úlohu budeme považovať za splnenú, pokiaľ bot dokáže automaticky klikať ľubovoľnú klávesu v určených intervaloch.
- **Vyhľadanie a ničenie kameňov Metin** - V hre Metin2 sú tieto kamene hlavným zdrojom príjmov. Môžeme si ich predstaviť ako príšeru, ktorá sa síce nehýbe, ale má veľmi vysoký počet životov. Vždy keď kameňu Metin uberieme 5% maximálneho života, zrodí okolo nás príšery, ktoré automaticky útočia. Po zničení kameňa metin, nás však čaká sladká odmena vo forme herných zdrojov. Táto metóda ich zisku je bezkonkurenčná oproti akýmkoľvek iným metódam. Bot musí byť schopný tieto kamene vyhľadať, pokiaľ niesú v dohľade herného charakteru, zmení pozíciu a hľadá ďalej. Pokiaľ však úspešne detekuje takýto kameň, pokúsi sa ho zničiť a následne zozbierať herné zdroje. Túto úlohu považujeme za splnenú, pokiaľ celý tento proces bot dokáže vykonávať automaticky.

1.4.2 Návrh dosiahnuteľných stavov a prechodné podmienky

Pri návrhu dosiahnuteľných stavov je kľúčová definícia funkcionality nášeho bota. Na základe nami navrhnutých úloh, ktoré bot bude automatizovať, sme navrhli nasledujúce stavy:

1. INITIALIZING
2. POLLING_NEXT_STATE
3. SEARCHING
4. CHECKING_MATCH
5. MOVING
6. HITTING
7. COLLECTING_DROP
8. RESTART
9. STANDBY
10. ERROR

1.4.2.1 INITIALIZING

Stav INITIALIZING je prvý zo sekvencie stavov. Ako prvým dôležitým krokom je inicializácia databázy a pripojenie k nej. Ďalej nakoľko na posielanie inštrukcii do hry Metin2 je potrebná interakcia s *On-Screen-Keyboard*, je dôležité, ktoré okno systému má aktuálny *focus*. V tomto stave teda prvým krokom je aktivácia okna Metin2 a nasledujú kroky ako sú prihlásenie sa do účtu a zvolenie charakteru. V tento moment už sa nachádzame v hernom svete Metin2, ale predtým než začneme ničiť kamene, je dôležité aktivovať všetky požehnania pre náš charakter, ktoré ho spravia silnejším. Nasleduje teda aktivácia elixírov a ostatných konzumovateľných *itemov*. Po "nabuffovaní" charakter nasadneme na koňa a môžeme prejsť do ďalšieho stavu.

1.4.2.2 POLLING_NEXT_STATE

Názov nášho ďalšieho stavu hovorí za všetko ale predsalen si ho popíšeme. V predošlom stave sme inicializovali databázu a pripojili sa k nej, teraz zistíme či sa v našej DB nachádzajú inštrukcie. Ak áno tak s najväčšou pravdepodobnosťou ďalším stavom bude stav iný ako **SEARCHING**, po prečítaní tejto inštrukcie sa zmaže a pokračuje do prečítaného stavu. Ak však v DB žiadna inštrukcia nieje, bot pokračuje sekvenčne do ďalších stavov.

1.4.2.3 SEARCHING

V tomto stave prichádza na radu náš systém detekcie objektov. Obraz je snímaný niekoľko snímok za sekundu a náš model detekuje potencionálne zhody. Ak žiadne zhody niesú nájdené, charakter sa presunie na inú pozíciu, kde tento krok opakuje. Ak však nájdeme zhody presúvame sa do ďalšieho stavu.

1.4.2.4 CHECKING_MATCH

Teraz vyhodnotíme najlepšiu z nájdených zhôd na základe podobnosti oproti natrénovanému modelu. Vyhodnotíme presné súradnice X a Y na základe pozície obdĺžnika získaného pomocou detekcie objektov. Presunieme kurzor na stred tohto obdĺžnika a pomocou ľavého kliku sa charakter začne ku kamenu presúvať čo indikuje prechod do ďalšieho stavu.

1.4.2.5 MOVING

Stav MOVING je iba prechodným stavom, počas ktorého náš charakter cestuje medzi bodom A a B. Tento stav je však dôležitý pre identifikáciu neočakávanej kolízie s terénom alebo jedným z NPC, ak sa ku kameňu nedostaneme do 10s prejdeme do stavu **ERROR**, ak áno prejdeme do stavu **HITTING**

1.4.2.6 HITTING

Bot sa nachádza v tomto stave po dobu 60s, počas týchto 60s náš charakter neustále útočí do kameňa Metin a tým znižuje jeho HP (body života), po tomto čase je kameň zničený a sme pripravení prejsť do ďalšieho stavu.

1.4.2.7 COLLECTING_DROP

Momentálne sa nachádzame v ďalšom medzistave, kde náš charakter musí pozbierať veci zo zeme (práve kvôli týmto veciam ničíme kamene). Kombináciou niekoľkých klaves tieto veci pozbierame a môžeme prejsť do ďalšieho stavu.

1.4.2.8 RESTART

V tomto stave znova aktivujeme požehnanie, keďže po niekoľkých minútach ich efekt vyprchá a prechádzame do stavu **POLLING_NEXT_STATE**

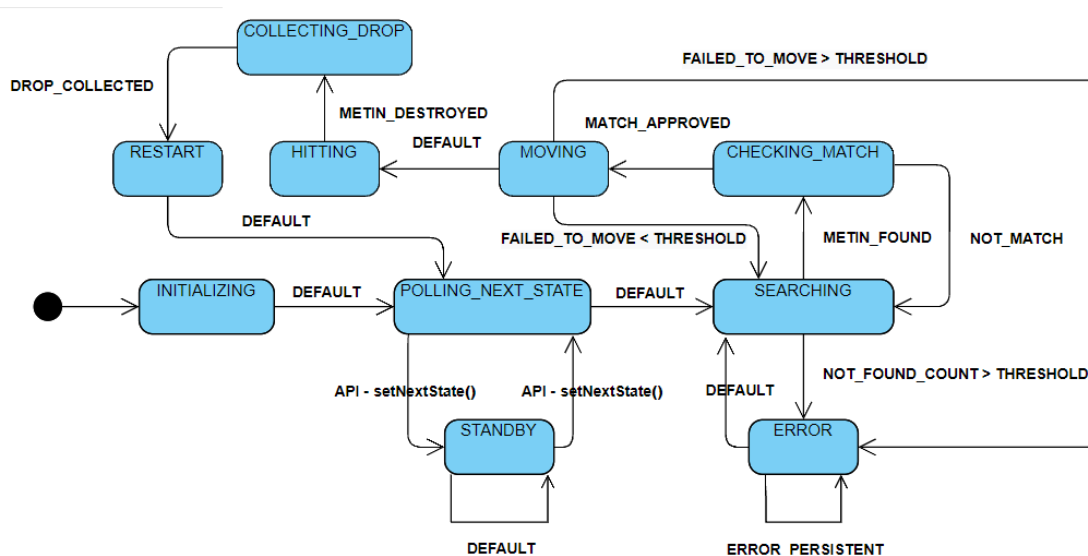
1.4.2.9 STANDBY

Do stavu **STANDBY** sa dokážeme dostať iba pomocou inštrukcie z našej API. Keďže bot beží v nekonečnom cykle, je dôležité mať akýsi stav v ktorom je zastavený. Tento stav je veľmi podobný stavu **POLLING_NEXT_STATE** až na jednu výnimku. Pokiaľ neexistuje ďalšia inštrukcia v DB, jednoducho nič nerobí. Dotaz na DB sa opakuje každých 30s.

1.4.2.10 ERROR

Do stavu **ERROR** nás môže dostať akýkoľvek s predošlých stavov, vždy pri prechode do tohto stavu si ukladáme aktuálny *timestamp*, pokiaľ je čas od predošlého erroru menej ako 5 minút, error je označený ako perzistentný a bot je zastavený. Pokiaľ je error nie-perzistentný, pokúsi sa začať od stavu **SEARCHING**

Prechod medzi stavmi je znázornený v stavovom diagrame, ktorý môžeme vidieť na obrázku nižšie.



■ Obr. 1.2 Stavový diagram

1.4.3 Požiadavky na cloudové služby

Pre úspešný beh nášho cloudového bota musíme zdefinovať niekoľko požiadaviek, ktoré musí nami zvolený provider splňovať.

- **Podpora pre systém Windows** - Bohužiaľ hra Metin2 je pomerne stará a preto podporuje iba operačný systém Windows.
- **Podpora pre grafické rozhranie** - Je potrebné kvôli rozpoznávaniu obrazu a interakciou s hrou.
- **Podpora vnorenej virtualizácie** - Nakoľko naša API a DB bežia cez Docker, je potrebná podpora vnorenej virtualizácie.
- **Obnova systému Windows z ISO Image** - V našej čistej inštalácii Windowsu je potrebné mať okrem hry nainštalovaných niekoľko nástrojov aby bot mohol úspešne vykonávať svoju funkciu. V ideálnom prípade teda obnovíme systém z ISO Image, ktorý už má všetku potrebnú funkcionálnosť nainštalovanú.
- **Bonus: Možnosť klonovať Windows inštancie** - V ideálnom prípade bez straty persistentných dát - V takomto prípade by predošlý bod nebol potrebný a veľmi by nám to zjednodušilo vytváranie ďalších inštancií nášho bota.

1.4.4 Prieskum cloudových providerov

Keďže čoraz viac informačných systémov beží v cloude, dopyt po cloudových provideroch raketovo stúpa. Výzvou pre každého vývojára, je nájsť správneho providera, ktorý spĺňa všetky požiadavky kladené jeho projektom. Takéto požiadavky definujeme v kapitole **Nefunkčné požiadavky**. V nasledujúcej tabuľke nájdeme užší výber cloudových poskytovateľov, ktorí všetky nefunkčné požiadavky aplikácie spĺňajú. Budeme ich teda porovnávať v ich všeobecných výhodách a nevýhodách:

Provider	Výhody	Nevýhody
Hyonix	<ul style="list-style-type: none"> - Vysoká rýchlosť pripojenia - Neobmedzený bandwidth - Bezkonkurenčne nízka cena - Vyhradená IP adresa - Dostupné viaceré lokácie dátových centier - Technická podpora 24/7 	<ul style="list-style-type: none"> - Pomerne nová firma - Obmedzené informácie o spoločnosti
Kamatera	<ul style="list-style-type: none"> - Plány začínajúce od 23\$ mesačne - Užívateľské hodnotenie 4.8/5 hviezdíčiek v kategórii Windows VPS hosting - Flexibilné plány a jednoduché škálovanie - Dostupné viaceré lokácie dátových centier 	<ul style="list-style-type: none"> - Obmedzené hodiny technickej podpory - Potrebné pokročilé technické zručnosti
InterServer	<ul style="list-style-type: none"> - Plný admin prístup k službe - 99.9% garancia dostupnosti - Dobré recenziue používateľov 	<ul style="list-style-type: none"> - Vyššie ceny oproti iným poskytovateľom
ChemiCloud	<ul style="list-style-type: none"> - Bezplatná pomoc pri migrácii - Dostupné viaceré lokácia dátových centier - Rýchly a spoľahlivý hosting 	<ul style="list-style-type: none"> - Obmedzené volne dostupné informácie o plánoch Windows VPS - Vyššie ceny oproti iným poskytovateľom

■ **Tabuľka 1.1** Tabuľka porovnania cloudových poskytovateľov

1.4.5 Potrebne nástroje k tvorbe vlastného Windows ISO obrazu

Windows ISO obraz je digitálny súbor, ktorý obsahuje kompletnú kopiu inštaláčného disku pre operačný systém Windows. ISO je formát súboru, ktorý nám umožňuje vytvoriť spúšťací disk alebo USB kľúč, pomocou ktorého následne nainštalujeme operačný systém.

ISO obraz obsahuje všetky súbory potrebné na inštaláciu a konfiguráciu systému Windows v novom alebo existujúcom počítači. Patria sem súbory operačného systému, systémové ovládače a rôzne softvérové nástroje, ktoré sú súčasťou inštalácie systému Windows.

Windows ISO Images sú k dispozícii na stiahnutie z oficiálnej webovej lokality spoločnosti Microsoft alebo iných webových adresách tretích strán, je dôležité uistiť sa, že súbor ISO je legitímny a nebol upravený, aby sa predišlo nakazeniu škodlivým softvérom alebo vírusom.

Náš bot je navrhnutý tak, aby mohli bežať jeho viaceré inštancie súčasne. K tvorbe takýchto inštancií máme dve možnosti prístupu:

1. Inštancie vytvárame z čistého Windows ISO Image a následne skriptom stiahneme a nainštalujeme všetky potrebné aplikácie a knižnice.
2. Inštancie vytvárame z vlastného Windows ISO Image s predinštalovanými nástrojmi a knižnicami.

V našej práci sme zvolili druhú možnosť a preto si v nasledujúcej kapitole popíšeme nástroje, ktoré budeme využívať pri vytváraní vlastného Windows ISO obrazu.

1.4.5.1 VirtualBox

VirtualBox je bezplatný virtualizačný open-source softvér, ktorý umožňuje používateľom spustiť viac operačných systémov súčasne na jednom fyzickom počítači. Vyvinula ho spoločnosť Oracle Corporation a je k dispozícii pre platformy Windows, MacOS, Linux a Solaris. Umožňuje používateľom vytvárať a spravovať virtuálne počítače, ktoré možno použiť na testovanie nového softvéru alebo na spúšťanie aplikácií v rôznych operačných systémoch. Poskytuje virtualizované

prostredie, ktoré izoluje hosťujúci OS od hostiteľského OS, čo používateľom umožňuje experimentovať s rôznymi konfiguráciami bez toho, aby ovplyvnili primárny hardvér. VirtualBox je vybavený celým radom funkcií, vďaka ktorým je obľúbenou voľbou pre vývojárov aj IT profesionálov. Medzi kľúčové funkcie VirtualBoxu patria[21]:

- **Podpora viacerých platforiem** - VirtualBox podporuje celý rad operačných systémov vrátane systémov Windows, MacOS, Linux a Solaris.
- **Správa snímok** - Umožňuje používateľom vytvárať snímky virtuálnych počítačov v rôznych fázach vývoja. Táto funkcia umožňuje používateľom v prípade potreby vrátiť sa k predchádzajúcemu stavu virtuálneho počítača.
- **Virtuálne siete** - Poskytuje funkcie virtuálnej siete, ktoré umožňujú virtuálnym počítačom komunikovať medzi sebou alebo aj s hostiteľským operačným systémom.
- **Škálovateľnosť** - VirtualBox možno použiť na správu rozsiahlych nasadení virtuálnych počítačov na viacerých hostiteľoch.

Keďže VirtualBox budeme používať pri vývoji, povieme si niečo k jeho inštalácii a konfigurácii. VirtualBox si môžeme stiahnuť z oficiálnej webovej lokality spoločnosti Oracle Corporation. Po stiahnutí inštaláčného balíka budeme postupovať podľa sprievodcu inštaláciou a nainštalujeme softvér do svojho počítača. Po inštalácii sme pripravení vytvoriť nový virtuálny počítač pomocou nasledujúcich krokov[21]:

- Kliknutím na tlačidlo "New"(Nový) vytvoríme nový virtuálny počítač.
- Zadáme názov virtuálneho počítača a vyberieme operačný systém a verziu, ktorá bude vo virtuálnom počítači nainštalovaná. Môžeme vybrať jednu z ponúkaných verzií alebo vytvoríme inštanciu z poskytnutého ISO súboru.
- Zvolíme adekvátne množstvo výpočtových zdrojov, ktoré budú pridelené virtuálnemu počítaču.
- Vyberieme možnosť ukladania pre virtuálny počítač vrátane veľkosti virtuálneho pevného disku.
- Podľa ostatných pokynov dokončíme inštaláciu virtuálneho operačného systému.

Po vytvorení virtuálneho počítača môžeme ďalej konfigurovať jeho nastavenia kliknutím pravým tlačidlom myši na virtuálny počítač a výberom položky "Settings"(Nastavenia). Týmto otvoríme dialógové okno, ktoré nám umožní upraviť konfiguráciu virtuálneho počítača vrátane jeho sieťových nastavení, zdieľaných priečinkov a mnoha ďalších.

K zhrnutiu, VirtualBox je výkonný a flexibilný virtualizačný softvér, ktorý používateľom umožňuje spustiť viacero operačných systémov súčasne na jednom fyzickom počítači. Poskytuje celý rad funkcií, vďaka ktorým je obľúbenou voľbou pre vývojárov a IT profesionálov. Obsahuje podporu viacerých platforiem, pridávania hostí, správy snímok, virtuálnych sietí a škálovateľnosti. Vďaka ľahko použiteľnému rozhraniu a komplexnej dokumentácii je VirtualBox ideálnym nástrojom pre každého, kto chce experimentovať s novým softvérom alebo testovať aplikácie na rôznych operačných systémoch. V našej implementácii, bude VirtualBox použitý na vytvorenie čistého Windows ISO image, ktorý obsahuje len operačný systém a nevyhnutné nástroje potrebné na beh nášho bota.

1.4.5.2 Disk Management

Nástroj *Disk Management* je integrovaný nástroj v operačných systémoch Windows, ktorý umožňuje používateľom spravovať jednotlivé partície a *volumes* na disku počítača. Poskytuje grafické používateľské rozhranie, ktoré uľahčuje vytváranie, formátovanie, zmenu veľkosti a odstraňovanie diskových partícií. Medzi kľúčové funkcie nástroja *Disk Management* patria:

1. **Vytváranie diskových partícií** - Používatelia môžu pomocou nástroja *Disk Management* vytvárať nové partície na svojich pevných diskoch alebo iných pamäťových zariadeniach. Môžu si vybrať veľkosť partície, zvoliť jej kódovanie a priradiť jej písmeno a názov.
2. **Formátovanie diskových partícií** - Nástroj *Disk Management* umožňuje používateľom formátovať svoje diskové partície pomocou rôznych súborových systémov vrátane NTFS, FAT32 a exFAT. Používatelia si tiež môžu vybrať veľkosť klastra a povoliť kompresiu na partícií.
3. **Rozšírenie a zmenšenie partícií** - Používatelia môžu pomocou rozšíriť alebo zmenšiť svoje diskové partície v závislosti od svojich potrieb. Môžu pridať nepridelené miesto do existujúcej partície alebo zmenšiť partíciu a vytvoriť tak voľné miesto.
4. **Zmeniť písmená diskov a cesty** - *Disk Management* umožňuje používateľom zmeniť písmeno disku. To môže byť užitočné, keď je písmeno disku v konflikte s iným zariadením alebo keď chcú používatelia efektívnejšie usporiadať svoje disky.
5. **Konvertovať disky** - Používatelia môžu konvertovať svoje disky medzi základnými a dynamickými diskami alebo medzi štýlmi diskových partícií MBR a GPT. To im môže umožniť používať funkcie, ako je RAID, alebo vytvoriť viac ako štyri primárne diskové partície.

V našom projekte tento nástroj bude slúžiť na vytvorenie virtuálnej partície, do ktorej vložíme všetky nainštalované programy a nástroje potrebné pre beh nášho bota.

1.4.5.3 Sysprep

Sysprep (skratka pre System Preparation - príprava systému) je nástroj dodávaný s operačnými systémami Windows, ktorý pripravuje počítač na duplikáciu, zobrazovanie a *imaging*. Odstraňuje informácie špecifické pre počítač, ako je názov počítača a bezpečnostný identifikátor (SID), čo umožňuje nasadenie obrazu do viacerých počítačov bez konfliktov. Sysprep je výkonný nástroj, ktorý umožňuje správcovi systému vytvárať vlastné obrazy operačných systémov Windows a konfigurovať ich na nasadenie do viacerých počítačov. Často sa používa v spojení s nástrojmi, ako sú Windows Deployment Services (WDS) a Microsoft System Center Configuration Manager (SCCM), na automatizáciu nasadenia vyššie spomenutých obrazov.

Sysprep tiež umožňuje správcovi systému konfigurovať nastavenia, ako sú sieťové nastavenia, regionálne nastavenia a ovládače. Tieto nastavenia sa dajú nakonfigurovať pred zachytením obrazu a potom sa použijú počas nasadenia.

Možno ho spustiť v dvoch režimoch: v režime grafického rozhrania a v režime príkazového riadka. V režime GUI môžu správcovia systému konfigurovať nastavenia prostredníctvom grafického používateľského rozhrania. V režime príkazového riadka môžu správcovia používať rozhranie naopak príkazového riadka k automatizácii celého procesu Sysprep a konfiguráciu nastavení. V nasledujúcich bodoch si uvedieme niekoľko dôvodov, prečo používať nástroj Sysprep:

- **Štandardizácia** - Umožňuje vytvárať štandardizované obrazy systému Windows, ktoré možno nasadiť vo viacerých počítačoch s rovnakými nastaveniami a konfiguráciami.
- **Úspora času** - Sysprep nám môže ušetriť veľa času tým, že automatizuje proces nasadenia a konfiguruje nastavenia pred vytvorením obrazu.
- **Prispôsobenie** - Umožňuje nám prispôbiť obrazy systému Windows tak, aby spĺňali špecifické požiadavky nášho projektu. To zahŕňa konfiguráciu nastavení, ako sú regionálne nastavenia, sieťové nastavenia a ovládače.
- **Zabezpečenie** - Odstránením informácií špecifických pre počítač pomáha Sysprep zabezpečiť, aby inštalácie systému Windows nasadené do iných počítačov boli bezpečné a neobsahovali žiadne citlivé informácie.

K zhrnutiu, Sysprep je výkonný nástroj, ktorý umožňuje správcovi systému pripraviť operačné systémy Windows na duplikáciu, *imaging* a nasadenie. Odstraňuje z inštalácie systému Windows informácie špecifické pre daný počítač, čo umožňuje nasadenie obrazu do viacerých počítačov bez konfliktov. Sysprep je základný nástroj pre správcov systému, ktorí potrebujú nasadiť štandardizované obrazy systému Windows vo viacerých počítačoch s rovnakými nastaveniami a konfiguráciami.

1.4.5.4 Windows PE

Windows Preinstallation Environment alebo aj **WinPE** je odľahčená verzia operačného systému Microsoft Windows, ktorá sa používa na nasadenie a konfiguráciu inštalácií systému Windows. Je navrhnutý na prevádzku v obmedzenom prostredí a poskytuje len základné funkcie, ako je správa súborov, sieťové pripojenie a rozdelenie disku[22].

WinPE je modulárny operačný systém, čo znamená, že správcovia systému môžu pridávať alebo odoberať komponenty a prispôbovať prostredie svojim špecifickým potrebám nasadenia. To môže zahŕňať prídanie ovládačov pre špecifický hardvér, sieťové nástroje alebo vlastné skripty. Po zavedení systému WinPE môžu správcovia systému použiť nástroje na nasadenie, ako napríklad WDS alebo SCCM, za účelom nasadenia obrazov systému Windows do viacerých počítačov[22].

WinPE je cenný nástroj pre správcov systému, ktorí potrebujú nasadiť a konfigurovať inštalácie systému Windows na viacerých počítačoch. Tu je niekoľko dôvodov, prečo správcovia systémov používajú WinPE[22]:

- **Flexibilita** - WinPE je modulárny operačný systém, ktorý sa dá prispôbiť špecifickým potrebám nasadenia. Správcovia systému môžu podľa potreby pridávať alebo odoberať jeho komponenty.
- **Kompatibilita** - Je kompatibilný so širokou škálou hardvéru a možno ho zaviesť z rôznych typov médií vrátane CD, DVD, jednotky USB a sieťového zavádzania.
- **Efektívnosť** - WinPE je ľahký operačný systém, ktorý sa dá rýchlo zaviesť a efektívne vykonáva úlohy nasadenia.
- **Prispôsobenie** - Správcovia systému môžu pomocou WinPE vytvoriť vlastné skripty nasadenia alebo pridať nástroje tretích strán na automatizáciu úloh nasadenia.

Na záver si to zhrnieme, WinPE je odľahčená verzia operačného systému Windows, ktorú správcovia systémov používajú na nasadzovanie a konfiguráciu inštalácií systému Windows. Poskytuje základné prostredie operačného systému, ktoré možno prispôbiť špecifickým potrebám nasadenia. WinPE je cenný nástroj pre správcov systému, ktorí potrebujú efektívne a rýchlo nasadiť obrazy systému Windows do viacerých počítačov[22].

1.4.5.5 AnyBurn

AnyBurn je bezplatný, *lightweight* a ľahko použiteľný softvér na napaľovanie diskov CD, DVD a Blu-ray. Je to všestranný nástroj, ktorý používateľom umožňuje vytvárať a napaľovať obrazy ISO, ripovať zvukové disky CD, kopírovať disky, vytvárať bootovacie disky USB a ďalšie. Niektoré z jeho funkcií sú:

- **Vypaľovanie súborov a priečinkov** - Umožňuje používateľom napaľovať súbory a priečinky na disky CD, DVD alebo Blu-ray. Používatelia môžu vybrať súbory alebo priečinky, ktoré chcú napaľovať, a jednoducho vytvoriť disk.
- **Vytváranie a napaľovanie obrazov ISO** - AnyBurn umožňuje používateľom vytvárať obrazy ISO zo súborov a priečinkov v počítači a potom ich napaľovať na disk. Je to užitočná funkcia na vytváranie záložných diskov alebo distribúciu softvéru.

- **Ripovanie zvukových diskov CD** - Ďalej umožňuje používateľom ripovať zvukové disky CD do obľúbených zvukových formátov, ako sú MP3, WAV, FLAC alebo APE.
- **Kopírovanie diskov** - AnyBurn umožňuje používateľom vytvoriť kópiu disku CD, DVD alebo Blu-ray. Táto funkcia môže byť užitočná na zálohovanie dôležitých údajov alebo vytváranie duplicitných kópií disku.
- **Vytváranie bootovateľných diskov USB** - Softvér môžeme použiť na vytvorenie bootovateľných USB diskov z obrazov ISO. Je to užitočná funkcia na inštaláciu operačných systémov alebo iného softvéru do počítača bez nutnosti fyzického diskového vstupu.

Nástroj *AnyBurn* nám bude slúžiť prevažne kvôli jeho funkcii **Vytváranie a napájanie obrazov ISO**. Som si istý, že na túto funkciu sa dá použiť množstvo iných nástrojov, no zvolili sme ho prevažne kvôli veľmi dobrým ohlasom a takisto kvôli už predošlej skúsenosti z praxe.

1.4.6 API

API je skratka pre aplikačné programovacie rozhranie a predstavuje súbor definovaných pravidiel, ktoré umožňujú rôznym aplikáciám vzájomne komunikovať. Rozhrania API fungujú ako sprostredkovatelia, ktorí spracúvajú prenosy údajov medzi systémami a umožňujú spoločnostiam otvoriť údaje a funkcie svojich aplikácií externým vývojárom tretích strán. Rozhrania API umožňujú programom komunikovať, získavať informácie alebo vykonávať určitú funkciu. Sprostredkujú komunikáciu medzi používateľmi (klientmi) a zdrojmi (servermi) a keď používatelia zadávajú požiadavky API alebo navštívia internetový obchod, očakávajú rýchlu odozvu. API tradične komunikujú v spoločnom JSON formáte. Existujú rôzne typy rozhraní API vrátane rozhraní API WebSocket a rozhraní API REST. WebSocket API podporuje obojsmernú komunikáciu medzi klientskymi aplikáciami a serverom, vďaka čomu je efektívnejšie ako REST API. Rozhrania API REST sú však najpopulárnejšie a najflexibilnejšie rozhrania API, ktoré sa dnes nachádzajú na webe.[23]

1.4.6.1 WebSocket API

WebSocket API je protokol, ktorý umožňuje obojsmernú komunikáciu medzi klientskymi aplikáciami a servermi prostredníctvom jedného dlhodobého spojenia. Poskytuje efektívnejší spôsob výmeny údajov ako tradičné spojenia HTTP, pretože sa vyhýba réžii neustáleho otvárania a zatvárania viacerých spojení. Pomocou *websocketov* môže klient aj server kedykoľvek začať komunikáciu a správy sa môžu odosielať hneď, ako sú k dispozícii, bez potreby techník pollingu alebo long-pollingu[24].

1. Polling - Technika komunikácie medzi klientom a serverom, pri ktorej klient opakovane žiada server o informácie vo vopred definovaných intervaloch. Klient posiela požiadavku na server v stanovenom časovom intervale a server odpovedá so všetkými aktualizáciami, ktoré mohli nastať od predchádzajúcej požiadavky. Polling môže byť užitočný pre aplikácie, ktoré vyžadujú pravidelné aktualizácie alebo oznámenia v reálnom čase, ako sú napríklad kanály sociálnych médií alebo novinky zo spravodajstva[25].
2. Long-polling - Umožňuje serveru posielať aktualizácie klientovi hneď, ako sú k dispozícii. Namiesto toho, aby klient opakovane žiadal o aktualizácie, klient odošle požiadavku na server a čaká, kým server odpovie na všetky nové údaje. Ak nie sú k dispozícii žiadne nové údaje, server drží spojenie otvorené, kým nie sú k dispozícii nové údaje, a vtedy server odpovie s novými údajmi. Táto technika je užitočná pre aplikácie, ktoré vyžadujú aktualizácie v reálnom čase, ako sú napríklad chatové aplikácie alebo online hry[26].

Jednou z hlavných výhod rozhrania WebSocket API je jeho nízka latencia, vďaka ktorej je ideálne pre aplikácie v reálnom čase, ako sú online hry, živý chat a finančné obchodovanie. Podporuje aj prenos binárnych údajov, ktorý môže byť v určitých prípadoch použitia efektívnejší ako textové formáty údajov[24].

WebSocket API podporuje väčšina moderných webových prehliadačov, ako aj technológie na strane servera, napríklad Node.js a Java Servlets. Často sa používa v kombinácii s inými webovými technológiami, ako sú HTML5, CSS a JavaScript, na vytváranie interaktívnych webových aplikácií[24].

1.4.6.2 REST API

REST API je skratka pre Representational State Transfer Application Programming Interface. Je to široko používaný architektonický štýl na navrhovanie webových rozhraní API, ktorý umožňuje výmenu údajov medzi systémami na internete. Rozhrania RESTful API sú založené na protokole HTTP a sú navrhnuté tak, aby pracovali s bezstavovým komunikačným protokolom, čo znamená, že každá požiadavka od klienta na server obsahuje všetky potrebné informácie na dokončenie požiadavky bez toho, aby sa vyžadoval akýkoľvek predchádzajúci kontext. Rozhrania API REST používajú štandardné metódy HTTP, ako sú GET, POST, PUT, DELETE atď. na vykonávanie rôznych operácií s vystavenými zdrojmi[27].

Rozhrania API REST sa v posledných rokoch stali veľmi populárnymi vďaka svojej jednoduchoosti, škálovateľnosti a flexibilitě. Vo veľkej miere ich využívajú webové a mobilné aplikácie na výmenu údajov s aplikáciami na strane servera[27].

Keďže náš bot je implementovaný ako stavový stroj, REST je postačujúcou voľbou architektonického štýlu našej API. Pre potreby cloudovej implementácie teda vytvoríme minimalistickú REST API, ktorá bude spĺňať nasledujúce funkcie

- Ovládanie bota
- Získ aktuálneho stavu bota
- Získ štatistických dát z DB

Túto funkcionality však nemôžeme získať bez implementácie databáze. V ďalšej kapitole si teda povieme niečo o niekoľkých typoch databáz a zvolíme ten, ktorý nám najviac bude vyhovovať.

1.4.7 Databáza

Výber správneho typu databázy je dôležitým rozhodnutím, ktoré môže výrazne ovplyvniť výkon a funkčnosť našej aplikácie alebo systému. Vzhľadom na množstvo dostupných typov databáz, z ktorých každý je určený na riešenie špecifických potrieb, môže byť výber správnej databázy náročnou úlohou. Toto rozhodnutie by malo vychádzať z povahy aplikácie alebo systému a požiadaviek na údaje, ktoré spravuje. Neschopnosť vybrať správnu databázu môže viesť k nízkemu výkonu, strate údajov a celkovému neoptimálnemu zážitku pre koncových používateľov.

V tejto kapitole sa budeme zaoberať rôznymi dostupnými typmi databáz a faktormi, ktoré by sa mali zohľadniť pri výbere správnej databázy pre vašu aplikáciu alebo systém. Preskúame aj výhody a nevýhody jednotlivých typov databáz a konkrétne prípady použitia, pre ktoré sú najvhodnejšie. S týmito vedomosťami následne vykonáme informované rozhodnutie o vhodnej databáze pre nášho bota.

1.4.7.1 MySQL

MySQL je jednou z najpoužívanejších relačných databáz s otvoreným zdrojovým kódom na svete. MySQL je systém správy databáz, ktorý na interakciu s údajmi uloženými v databáze používa

jazyk SQL (Structured Query Language). Je známy svojou škálovateľnosťou, spoľahlivosťou a jednoduchým používaním, vďaka čomu je obľúbenou voľbou pre malé aj veľké podniky.[28]

Jednou z hlavných výhod MySQL je jej kompatibilita s rôznymi operačnými systémami vrátane systémov Windows, Linux a MacOS. Vďaka tejto multiplatformovej podpore je flexibilnou voľbou pre podniky s rôznorodou IT infraštruktúrou. Okrem toho MySQL ponúka silné funkcie zabezpečenia údajov vrátane podpory šifrovania a bezpečného overovania používateľov, čo podnikom pomáha udržiavať ich údaje v bezpečí.[28]

Ďalšou významnou výhodou MySQL je podpora vysokovýkonných výpočtových systémov, vďaka čomu je ideálnou voľbou pre rozsiahle aplikácie, ktoré vyžadujú rýchle spracovanie a ukladanie údajov. MySQL využíva architektúru klient-server, ktorá umožňuje súčasný prístup viacerých používateľov k databáze, čo z nej robí vynikajúcu voľbu pre podniky s vysokou návštevnosťou alebo súbežnými reláciami používateľov.[28]

Celkovo je MySQL výkonný a všestranný systém správy databáz, ktorý ponúka celý rad funkcií pre podniky všetkých veľkostí. Vďaka svojej flexibilita, spoľahlivosti a cenovej dostupnosti je obľúbenou voľbou medzi vývojármi, ktorí chcú efektívne spravovať a uchovávať svoje údaje.

1.4.7.2 MongoDB

MongoDB je populárny databázový systém NoSQL orientovaný na dokumenty, ktorý bol predstavený v roku 2009. Namiesto tabuliek a riadkov ako v tradičných relačných databázach používa MongoDB kolekcie a dokumenty. Dokumenty pozostávajú z dvojíc kľúč-hodnota, ktoré sú základnou jednotkou údajov v MongoDB. Dokumenty môžu mať rôzne štruktúry a polia sa môžu v jednotlivých dokumentoch líšiť, vďaka čomu je MongoDB veľmi flexibilná a škálovateľná[29].

Tento databázový systém je navrhnutý tak, aby bol ako sme už povedali vysoko škálovateľný a dokázal ľahko spracovať veľké objemy údajov. Jeho automatické funkcie shardingu umožňujú rozdeliť údaje medzi viacero uzlov, čo umožňuje horizontálne škálovanie pri náraste údajov. To je veľká výhoda oproti tradičným relačným databázam, ktoré si často vyžadujú rozsiahlu optimalizáciu na spracovanie veľkých objemov údajov[29].

Ďalšou výhodou MongoDB je jej schopnosť spracovávať neštruktúrované údaje. Keďže dokumenty môžu mať rôznu štruktúru, je vhodná na ukladanie údajov v rôznych formátoch, ako sú JSON, BSON a XML. MongoDB dokáže spracovať aj geopriestorové údaje, vďaka čomu je obľúbenou voľbou pre aplikácie, ktoré vyžadujú lokalizačné služby[29].

MongoDB má však aj niektoré potenciálne nevýhody, ktoré treba zvážiť. Keďže ide o databázu NoSQL, neposkytuje rovnakú úroveň zhody s ACID (Atomicity, Consistency, Isolation, Durability) ako tradičné relačné databázy. To znamená, že nemusí byť vhodná pre aplikácie, ktoré vyžadujú prísnu konzistenciu údajov alebo podporu transakcií. Okrem toho má MongoDB strmšiu krivku učenia ako niektoré iné databázy kvôli svojmu jedinečnému dátovému modelu a dotazovaciemu jazyku[29].

Napriek týmto potenciálnym nevýhodám zostáva MongoDB obľúbenou voľbou pre mnohé aplikácie vďaka svojej flexibilita, škálovateľnosti a schopnosti spracovať neštruktúrované údaje. Široko sa používa v odvetviach, ako je elektronický obchod, sociálne médiá a vývoj mobilných aplikácií.

1.4.7.3 PostgreSQL

PostgreSQL, známy aj ako Postgres, je populárny open-source systém na správu relačných databáz (RDBMS), ktorý sa vyvíja už viac ako tri desaťročia. Pôvodne bol vydaný v roku 1989 ako výskumný projekt na Kalifornskej univerzite v Berkeley a odvtedy ho prijalo mnoho organizácií na celom svete. PostgreSQL je navrhnutý tak, aby zvládol širokú škálu pracovných záťaží, od malých aplikácií až po veľké systémy na podnikovej úrovni[30].

Jednou z kľúčových výhod PostgreSQL je jeho robustný súbor funkcií. Podporuje rôzne typy údajov vrátane skalárnych, polí a JSON. Poskytuje tiež pokročilé možnosti indexovania

a vyhľadávania vrátane fulltextového vyhľadávania a geopriestorových dotazov. Okrem toho sa PostgreSQL vyznačuje osvedčenou spoľahlivosťou a stabilitou a kladie veľký dôraz na integritu a konzistenciu údajov[30].

Ďalšou výhodou PostgreSQL je jeho škálovateľnosť. Bez problémov zvláda veľké súbory údajov a veľké objemy dát, vďaka čomu je obľúbenou voľbou pre webové aplikácie a dátové úložiská. PostgreSQL má tiež veľkú a aktívnu komunitu používateľov a vývojárov, ktorí prispievajú k jej neustálemu vývoju a údržbe[30].

Hoci má PostgreSQL mnoho silných stránok, má aj určité obmedzenia. Jedným z najväčších problémov pre niektorých používateľov je zložitosť jeho nastavenia a správy. V porovnaní s niektorými inými databázami môže byť konfigurácia a ladenie PostgreSQL na dosiahnutie optimálneho výkonu náročnejšie. Okrem toho niektorí používatelia môžu zistiť, že syntax jazyka SQL v databáze PostgreSQL nie je taká intuitívna ako v iných databázach, čo môže sťažovať prácu tým, ktorí tento jazyk nepoznajú[30].

Celkovo možno konštatovať, že hoci existujú aj iné vhodné databázy na implementáciu cloudových botov, PostgreSQL je vďaka robustnému súboru funkcií, spoľahlivosti, flexibilitě a s prihliadnutím na predošlú praktickú skúsenosť silným kandidátom. V našej konkrétnej implementácii, bude databáza slúžiť najmä ako médium medzi ovládaním bota prostredníctvom API a na ukladanie štatistických údajov. Musíme však myslieť aj na potencionálne rozšírenie funkcionality bota v budúcnosti a preto komplexná databáza ako je PostgreSQL je viac než vhodným kandidátom.

1.5 Architektúra

Architektúra bota bude vychádzať z funkčných a nefunkčných požiadaviek, ktoré si zadefinujeme v nasledujúcich kapitolách.

1.5.1 Funkčné požiadavky

Požiadavky špecifikujúce hlavnú funkcionality navrhovaného systému:

1.5.1.1 F1 - Automatické farmenie kameňov Metin

Bot detekuje a automaticky ničí kamene metin

1.5.1.2 F2 - Úschova štatistických údajov v DB

Bot posiela štatistické údaje do databázy

1.5.1.3 F3 - Ovládanie pomocou API

Bot umožňuje ovládanie pomocou API volaní

1.5.1.4 F4 - Rozhranie na jednoduché ovládanie bota

Bot disponuje jednoduchým grafickým rozhraním k funkcii ovládania - pravdepodobne bude použitý *Postman* alebo *Swagger* klient

1.5.2 Nefunkčné požiadavky

Požiadavky špecifikujúce hlavné obmedzenia kladené na systém:

1.5.2.1 N1 - Cloudové nasadenie na stroji so systémom Windows

Hra Metin2 je pomerne stará a preto podporuje iba operačný systém Windows

1.5.2.2 N2 - Dockerizácia služieb

Bot a ostatné služby je možné nasadiť ako docker kontajner

1.5.2.3 N3 - Škálovateľnosť

Inštancie bota je možné škálovať horizontálne

1.5.2.4 N4 - Vytváranie inštancií pomocou ISO snímky

K umožneniu horizontálneho škálovania, inštancie sú generované z predvytvoreného ISO obrazu systému

1.5.3 Návrh databázových modelov

Pred samotným návrhom databázových modelov je dôležité definovať požiadavky bota na našu databázu. Databáza sa bude dotazovať v nasledujúcich scenároch:

1. Bot získava z DB svoju nasledujúcu úlohu - Naša implementácia bola navrhnutá formou stavového stroja. Bot teda bude mať špecifický stav, v ktorom bude dotazovať na DB, za účelom získania ďalšej úlohy. Na základe výsledku tohto dotazu, je bot uvedený do špecifikovaného stavu.
2. Bot ukladá do DB štatistické údaje - Keďže bot beží v cloudovom prostredí, je veľmi dôležité uchovávať štatistické údaje, najmä z dôvodu kontroly efektivity našej imlementácie.
3. Bot získava z DB údaje, potrebné na prihlásenie sa na svoj charakter

Na základe týchto troch požiadaviek na našu databázu, vieme navrhnuť jej schému.

TaskTable
+ cloud_id: varchar
+ next_state: varchar
+ created_at: timestamp

■ Obr. 1.3 Tabuľka úloh

- cloud_id - Id cloudovej inštancie pre ktorú je určená úloha
- next_state- Udáva ďalší stav, ktorý bude nasledovať po stave POLLING_NEXT_STATE
- created_at - Čas vytvorenia úlohy, dôležité z dôvodu chronologického spúšťania úloh.

AccountTable
+ cloud_id: varchar
+ username: varchar
+ password: varchar

■ Obr. 1.4 Tabuľka herných údajov

- cloud_id - Id cloudovej inštancie vytvárajúcej záznam v databáze
- username - Prihlasovacie meno konta
- password - Zahashované prihlasovacie heslo konta

BotStatsTable
+ cloud_id: varchar
+ metin_count: int
+ error_count: int
+ session_begin: timestamp
+ session_end: timestamp
+ total_duration: double

■ Obr. 1.5 Tabuľka herných štatistík

- cloud_id - Id cloudovej inštancie vytvárajúcej záznam v databáze
- error_count - Počet situácií, kedy sa bot dostal do stavu ERROR
- metin_count - Počet zničených kameňov metin
- session_begin - Čas spustenia aktuálneho behu bota
- session_end - Čas ukončenia aktuálneho behu bota
- total_duration - Celkový čas aktuálneho behu bota

1.5.4 Ovládanie viacerých cloudových inštancií

Naša implementácia bude pracovať iba s jednou cloudovou inštanciou, avšak je navrhnutá tak, aby bola rozšíriteľná do verzie s viacerými bežiacimi inštanciami súčasne. Preto považujem za dôležité, opísať tento scenár aspoň po návrhovej stránke. Aby sme mohli ovládať viaceré cloudové inštancie nášho bota súčasne, potrebujeme splniť nasledujúce kroky:

1. Určíme počet inštancií, ktoré potrebujeme na základe našich požiadaviek. Aspoň rádomý odhad počtu je veľmi dôležitý, nakoľko bude z neho vychádzať voľba ostatných nástrojov spomenutých v tejto kapitole.

2. Zvolíme poskytovateľa cloudových služieb tak, aby spĺňal zadané nefunkčné požiadavky aplikácie a umožňoval jednoduché vytváranie a správu cloudových inštancií.
3. Keďže náš bot beží ako docker kontajner, použijeme nástroj na orchestráciu kontajnerov ako je napríklad Kubernetes alebo Docker Swarm. Tieto nástroje nám umožnia jednoducho nasadiť a škálovať viaceré inštancie nášho bota.
4. Je dôležité, aby každá inštancia bola nakonfigurovaná rovnako. Tvorbou vlastného Windows ISO Image, konfigurácia je persistentná naprieč jednotlivými inštanciami. V opačnom prípade môžeme použiť nástroj na správu konfigurácie, ako je napríklad Ansible, Chef alebo Puppet.
5. Na monitorovanie výkonu a zdravia inštancií, môžeme použiť monitorovací nástroj ako je napríklad Nagios, Zabbix alebo Prometheus.
6. Ako posledné je takisto veľmi dôležité logovanie a analýza logov. Na túto činnosť môžeme použiť napríklad ELK Stack (Elasticsearch, Logstash, Kibana), Graylog alebo Splunk.

2.1 Python

Python je vysokoúrovňový interpretovaný programovací jazyk známy svojou jednoduchosťou, ľahkým používaním a čitateľnosťou. V roku 1991 ho vytvoril Guido van Rossum a odvtedy sa stal jedným z najpopulárnejších programovacích jazykov súčasnosti. Python je open-source jazyk a má veľkú a aktívnu komunitu vývojárov, čo prispelo k jeho širokému používaniu[31].

Jedným z hlavných dôvodov popularity jazyka Python je jeho jednoduchosť pri učení. Jazyk má jednoduchú syntax, vďaka ktorej sa ľahko číta a píše. Vďaka tomu je výbornou voľbou pre začiatočníkov aj skúsených programátorov. Okrem toho má Python širokú škálu knižníc a rámcov, ktoré možno použiť na rôzne aplikácie vrátane dátovej vedy, strojového učenia, vývoja webových stránok a ďalších.

Python je tiež všestranný jazyk. Možno ho použiť na rôzne typy programátorských úloh, od jednoduchých skriptov až po zložité aplikácie. Vďaka tejto všestrannosti v kombinácii s jednoduchým používaním sa stal obľúbenou voľbou pre širokú škálu projektov.

Jednou z kľúčových vlastností jazyka Python je jeho dynamic typing. To znamená, že dátový typ premennej sa určuje počas behu, a nie v čase kompilácie. To umožňuje väčšiu flexibilitu a jednoduchosť použitia pri práci s údajmi, pretože programátor nemusí explicitne špecifikovať dátový typ premennej.

Ďalšou dôležitou vlastnosťou jazyka Python je podpora objektovo orientovaného programovania (OOP). Python umožňuje programátorom vytvárať triedy a objekty, ktoré možno použiť na zapuzdrenie údajov a správania. To uľahčuje vytváranie opakovane použiteľného kódu a zjednodušuje proces vývoja[31].

Na záver možno povedať, že Python je populárny programovací jazyk známy svojou jednoduchosťou, ľahkým používaním a všestrannosťou. Vďaka podpore dynamic typing-u a objektovo orientovaného programovania je výbornou voľbou pre širokú škálu projektov. Vďaka aktívnej komunite vývojárov a širokej škále knižníc a rámcov zostane Python pravdepodobne obľúbenou voľbou programátorov aj v nasledujúcich rokoch[31].

2.2 Detekcia objektov

2.2.1 Príprava vstupných dát

2.2.1.2 Tvorba popisných súborov

Ďalším krokom je vytvorenie popisných súborov. Popisné súbory sa líšia jednotlivito pre negatívne a pozitívne snímky obrazovky. V prípade negatívnych snímok, je dôležitá iba ich cesta. Vytvorili sme preto nasledujúci krátky skript.

■ Výpis kódu 2.2 Skript na tvorbu popisných súborov

```
def generate_negative_description_file(folder):
    with open('neg.txt', 'w') as f:
        for filename in os.listdir(folder):
            f.write(folder + filename + '\n')
```

Tento skript má za výsledok súbor v tomto tvare:

■ Výpis kódu 2.3 Ukážka neg.txt súboru

```
classifier/negative_2023_19_3_01/1679228560.jpg
classifier/negative_2023_19_3_01/1679228563.jpg
classifier/negative_2023_19_3_01/1679228565.jpg
classifier/negative_2023_19_3_01/1679228568.jpg
classifier/negative_2023_19_3_01/1679228570.jpg
...
```

V prípade pozitívnych snímok, nieje formát výstupného súboru tak jednoduchý. Pozitívne snímky totiž obsahujú nami hľadané objekty. Pozície týchto objektov teda musia byť zakódované v našom výstupnom súbore, k tomu využijeme integrovaný OpenCV anotačný nástroj. Pomocou tohto nástroja, manuálne označíme každý objekt a výstupom bude opisný súbor, obsahujúci cestu k snímke ale aj pozície všetkých obdĺžnikov obsahujúce naše objekty záujmu.

■ Výpis kódu 2.4 Spustenie anotačného nástroja

```
opencv_annotation.exe --annotations=POSITIVE.txt --images=IMAGES_PATH/
```



■ Obr. 2.1 OpenCV anotačný nástroj

Po úspešnej anotácii, získavame výstupný súbor v nasledujúcom formáte. Prvým údajom je cesta k snímke, nasledujúca cifra určuje počet anotovaných objektov a za ňou sú už ich súradnice.

■ Výpis kódu 2.5 Ukážka pos.txt súboru

```
positive_2023_19_3_01/1679228558.jpg 2 318 298 54 68 921 350 67 81
```

```
positive_2023_19_3_01/1679228585.jpg 1 1139 248 29 37
positive_2023_19_3_01/1679228621.jpg 1 881 211 36 38
...
```

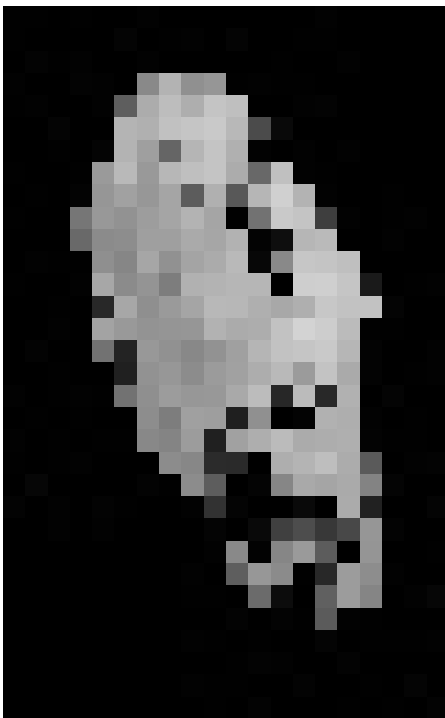
2.2.1.3 Tvorba vektorového súboru

Ďalším krokom spracovania pozitívnych snímok je vytvorenie vektorového súboru, ktorý bude už použitý na tréning nášho klasifikátora. Taktiež použijeme jeden z OpenCV nástrojov pomocou nasledujúceho skriptu.

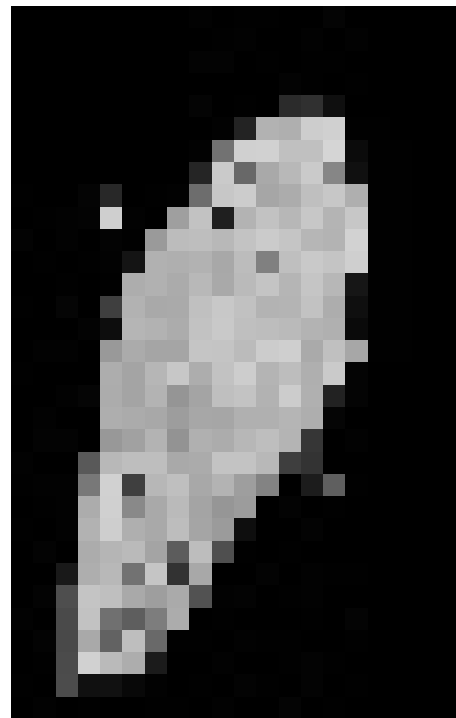
■ Výpis kódu 2.6 Tvorba vektorového súboru

```
opencv_createsamples.exe -info POSITIVE.txt -w 24 -h 24 -num 1000 -vec OUTPUT.vec
```

Ako môžeme vidieť, zadávame niekoľko parametrov, akými sú **-w** - tento parameter určuje minimálnu šírku detekovaného objektu v pixeloch, analogicky **-h** určuje výšku a parameter **-num** by mala byť hodnota väčšia ako celkový počet anotovaných objektov v súbore **POSITIVE.txt**. Bez akýchkoľvek nástrojov, nieje jednoduché pre človeka čítať vektorové súbory. Použijeme teda ďalší z OpenCV nástrojov na vizualizáciu našich výsledkov a získavame nasledujúce vzorky z našich snímok.



■ Obr. 2.2 Vzorka č. 1



■ Obr. 2.3 Vzorka č. 2

V tomto momente disponujeme všetkými potrebnými vstupnými dátami a nasleduje fáza tréningu nášho klasifikátora.

2.2.2 Tréning

K tréningu klasifikátora použijeme ďalší z nástrojov OpenCV. Ako môžeme nižšie vidieť, stretávame sa s niekoľkými novými argumentami, ktorým musíme zadať požadované hodnoty:

- numPos - Počet pozitívnych vzoriek použitých pri tréningu pre každú fázu klasifikátora.
- numNeg - Počet negatívnych vzoriek použitých pri tréningu pre každú fázu klasifikátora.
- numStages - Počet kaskádových štádií, ktoré sa majú trénovať
- minHitRate - Minimálna požadovaná miera úspešnosti pre každú fázu klasifikátora.
- maxFalseAlarmRate - Maximálna požadovaná miera chybných hlásení pre každú fázu klasifikátora.
- acceptanceRatioBreakValue - Tento argument sa používa na určenie toho, ako presne sa má náš model ďalej učiť a kedy sa má zastaviť. Dobrým pravidlom je trénovať najviac $10e-5$, aby sa zabezpečilo, že sa model na našich tréningových údajoch nepretrénuje.

Na tréningovanie nášho klasifikátora sme použili nasledujúce odporúčané konfigurácie:

■ Výpis kódu 2.7 Tréningová konfigurácia č.1

```
opencv_traincascade.exe -data PATH_TO_CLASSIFIER -vec POSITIVE.vec
                        -bg NEGATIVE.txt -w 24 -h 24
                        -numPos 400 -numNeg 200 -numStages 10
                        -miniHitRate 0.5 -maxFalseAlarmRate 0.5
```

■ Výpis kódu 2.8 Výstup tréningovej konfigurácie č.1

```
PARAMETERS:
cascadeDirName: classifier/cascade/
sampleHeight: 24
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: BASIC
Number of unique features given windowSize [24,24] : 162336

Stages 0-2 are loaded

===== TRAINING 3-stage =====
<BEGIN
POS count : consumed    400 : 401
NEG count : acceptanceRatio    35 : 0.000949436
Required leaf false alarm rate achieved. Branch training terminated.
```



■ Obr. 2.4 Výledky tréningu č.1

Ako môžeme vidieť, pri tréningu s touto konfiguráciou sme sa nedostali ďalej ako po úroveň 3, takýto výsledok je nedostačujúci, nakoľko nami požadovaná úroveň bola 10. Zvolíme teda prísnejšie kritéria tréningu. Nesprávnu konfiguráciu môžeme vidieť aj na obrázku vyššie, kde kameň metin je detekovaný iba 1x, aj keď v snímku je týchto kameňov 5. Okrem nesprávneho počtu detekovaných kameňov aj obdĺžnik, ktorý vykresľujeme, nie je na úplne správnom mieste a detekuje sa množstvo okolitého šumu.

■ **Výpis kódu 2.9** Tréningová konfigurácia č.2

```
opencv_traincascade.exe -data PATH_TO_CLASSIFIER -vec POSITIVE.vec
                        -bg NEGATIVE.txt -w 24 -h 24
                        -numPos 500 -numNeg 250 -numStages 15
                        -miniHitRate 0.5 -maxFalseAlarmRate 0.5
                        -acceptanceRatioBreakValue 0.0005
```

■ **Výpis kódu 2.10** Výstup tréningovej konfigurácie č.2

```
PARAMETERS:
cascadeDirName: classifier/cascade/
vecFileName: pos.vec
bgFileName: neg.txt
numPos: 500
numNeg: 250
numStages: 10
precalcValBufSize[Mb] : 1024
precalcIdxBufSize[Mb] : 1024
stageType: BOOST
featureType: HAAR
sampleWidth: 24
sampleHeight: 24
boostType: GAB
minHitRate: 0.995
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: BASIC
Number of unique features given windowSize [24,24] : 162336
```

```
Stages 0-5 are loaded

==== TRAINING 6-stage ====
<BEGIN
POS count : consumed    500 : 518
NEG count : acceptanceRatio    0 : 0
Required leaf false alarm rate achieved. Branch training terminated.
```



■ Obr. 2.5 Výsledky tréningu č.2

Pri tréningu s touto konfiguráciou sme sa dostali už ďalej a to po úroveň 6, takýto výsledok je však stále nedostačujúci, nakoľko nami požadovaná úroveň bola 10. Fakt, že konfigurácia je nesprávna, môžeme vidieť aj na obrázku vyššie. V tomto prípade, detekujeme už viac kameňov metin, ale stále detekujem množstvo okolitého šumu.

■ **Výpis kódu 2.11** Tréningová konfigurácia č.3

```
opencv_traincascade.exe -data PATH_TO_CLASSIFIER -vec POSITIVE.vec
                        -bg NEGATIVE.txt -w 24 -h 24
                        -numPos 320 -numNeg 640 -numStages 15
                        -miniHitRate 0.5 -maxFalseAlarmRate 0.5
                        -acceptanceRatioBreakValue 0.0005
```

■ **Výpis kódu 2.12** Výstup tréningovej konfigurácie č.3

```
PARAMETERS:
cascadeDirName: classifier/cascade/
vecFileName: pos.vec
bgFileName: neg.txt
numPos: 320
numNeg: 640
numStages: 10
precalcValBufSize[Mb] : 1024
precalcIdxBufSize[Mb] : 1024
acceptanceRatioBreakValue : 0.0001
stageType: BOOST
featureType: HAAR
sampleWidth: 24
sampleHeight: 24
boostType: GAB
minHitRate: 0.995
```

```
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: BASIC
Number of unique features given windowSize [24,24] : 162336

Stages 0-9 are loaded

===== TRAINING 10-stage =====
<BEGIN
POS count : consumed    320 : 323
NEG count : acceptanceRatio    0 : 0
```

Pomocou poslednej konfigurácie sme úspešne natrénovali klasifikátor až po úroveň 10, ktorá bola zvolená ako požadovaná. Parametre tejto konfigurácie môžeme teda považovať za správne a použijeme ich na tréning nášho modelu.

2.3 Bot

V nasledujúcej časti si priblížime navrhnutú implementáciu nášho bota, taktiež popíšeme použité metódy a postupy pri implementácii.

2.3.1 Herné ovládanie

Bohužiaľ anti-cheat hry Metin2 blokuje akékoľvek klávesové vstupy, ktoré nepochádzajú priamo z klávesnice. Z tohoto dôvodu využijeme natívnu aplikáciu systému Windows **On-Screen-Keyboard**. S využitím tejto metódy z dôvodu obchádzania anti-cheat systému však vzniká nový problém, ktorým je ako použijeme On-Screen-Keyboard. Riešenie je pomerne jednoduché, namapujeme súradnice jednotlivých kláves a myšou následne vykonáme klik na tejto pozícii.

■ Výpis kódu 2.13 Ukážka mapovania kláves

```
self.key_pos = {
    'space': (200, 210),
    'Fn': (20, 210),
    'Alt': (100, ),
    'Shift': (40, 180),
    'Caps': (20, ),
    'Tab': (20, 115),
    'Enter': (380, 140)
    ...
}
```

Nasleduje funkcia, ktorá vykonáva klik, podrží klávesu stlačenú, uvoľní ju alebo v kombinácii s klávesou Shift napíše veľké písmeno:

■ Výpis kódu 2.14 Metóda `press_key`

```
def press_key(self, button, mode='click', count=1):
    x, y = self.x, self.y
    if button not in self.key_pos.keys():
        if button.lower() in self.key_pos.keys():
            self.press_key(button.lower(), 'clickCapital')
            return
        else:
            raise Exception('Unknown key!')

    else:
        x += self.key_pos[button][0]
        y += self.key_pos[button][1]
        pyautogui.moveTo(x=x, y=y)
    if mode == 'click':
        for i in range(count):
            pyautogui.mouseDown()
            sleep(0.001)
            pyautogui.mouseUp()
    elif mode == 'clickCapital':
        for i in range(count):
            self.press_key('Shift')
            self.press_key(button)
    elif mode == 'down':
        pyautogui.mouseDown()
    elif mode == 'up':
        pyautogui.mouseUp()
```

Implementácia tejto funkcie nieje potrebná iba pre prípad využitia On-Screen-Keyboard, ale aj pre prípad kliknutia do stredu detekovaného herného objektu (kameňa Metin), kde týmto klikom naň zaútočíme.

2.3.2 Štatistické údaje

Pri návrhu databázového modelu, sme zvolili nasledujúce údaje, ktoré budeme ukladať do DB.

- `error_count` - počet situácií, kedy sa bot dostal do stavu ERROR
- `metin_count` - počet zničených kameňov metin
- `session_begin` - čas spustenia aktuálneho behu bota
- `session_end` - čas ukončenia aktuálneho behu bota
- `total_duration` - celkový čas behu bota

2.3.3 Flask API

Flask je populárny webový framework v jazyku Python, ktorý umožňuje vývojárom jednoducho vytvárať webové aplikácie. Jednou z kľúčových vlastností Flasku je podpora vytvárania webových API (Application Programming Interfaces) prostredníctvom knižnice Flask API. Flask API veľmi zjednodušuje tento proces, takže je vynikajúcou voľbou pre vývojárov, ktorí chcú vytvárať RESTful API.

Jednou z hlavných výhod používania Flask API je jeho jednoduché používanie. Flask API je postavená nad Flask-om, čo znamená, že vývojári môžu pri vytváraní API využívať existujúci ekosystém Flask. Ten zahŕňa podporu smerovania, spracovania požiadaviek a spracovania odpovedí, čo uľahčuje vytváranie rozhraní API, ktoré sú v súlade so zásadami návrhu RESTful API.

Flask API poskytuje aj celý rad funkcií, ktoré uľahčujú vytváranie bezpečných API. Podporuje autentifikáciu a autorizáciu, čo vývojárom umožňuje kontrolovať prístup k ich API. Okrem toho Flask API poskytuje podporu autentifikácie založenej na tokenoch, čo je bežný prístup na zabezpečenie API.

Ďalšou kľúčovou výhodou Flask API je jeho rozšíriteľnosť. Flask API poskytuje celý rad zásuvných modulov a rozšírení, ktoré možno použiť na rozšírenie funkčnosti knižnice. Patrí sem podpora pre dokumentáciu Swagger, verzovanie API a ďalšie.

Jednou z hlavných výhod vytvárania API pomocou Flask API je jeho výkon. Ako sme už vyššie spomenuli Flask API je postavená nad Flask-om. To znamená, že rozhrania API vytvorené pomocou Flask API môžu byť vysoko výkonné aj pri veľkom zaťažení.

Na záver možno povedať, že Flask API je výkonná knižnica, ktorá zjednodušuje proces vytvárania RESTful API pomocou jazyka Python. Vďaka jednoduchému používaniu, bezpečnostným funkciám a rozšíriteľnosti je vynikajúcou voľbou pre vývojárov, ktorí chcú vytvárať webové API. Okrem toho je vďaka svojmu výkonu vynikajúcou voľbou pre aplikácie s vysokou návštevnosťou. Vďaka aktívnej komunite vývojárov a pokračujúcemu vývoju bude Flask API pravdepodobne aj v nasledujúcich rokoch obľúbenou voľbou na vytváranie API v jazyku Python.[32]

Keďže náš bot je navrhnutý ako stavový stroj, architektúra našej API bude pomerne jednoduchá. V riešení sme implementovali nasledujúce volania:

- `setNextState` [**POST**] - Pomocou callu `setNextState` určíme špecifikovanej inštancii pomocou `cloud_id` ďalší stav do ktorého sa dostaneme pri nasledujúcej iterácii cyklu bota.

■ **Výpis kódu 2.15** Ukážka API volania č.1

```
@app.route('/nextState/', methods=['POST'])
def setNextState():
    nextState = request.get\_\_json()['nextState']
    cloud_id = request.get\_\_json()['cloud_id']
    conn = psycopg2.connect(db\_\_url)
    cur = conn.cursor()

    cur.execute(
        'CREATE TABLE IF NOT EXISTS StateTable ' \
        '(id serial PRIMARY KEY, next_state varchar(255), ' \
        'cloud_id varchar(255), created_at timestamp)')

    query = 'INSERT INTO StateTable ' \
            '(next_state, cloud_id, created_at) ' \
            'VALUES (%s, %s, %s)'

    cur.execute(query, (nextState, datetime.now()))
    conn.commit()
    cur.close()
    conn.close()

    return nextState
```

- `getCurrentSessionStats` [**GET**] - Ako už z názvu vyplýva, tento API call fetchne z DB aktuálne štatistické údaje nášho bota, v aktuálnom štádiu implementácie tieto štatistiky predstavujú zatiaľ len počet zničených kameňov metin danej inštancie.

■ **Výpis kódu 2.16** Ukážka API volania č.2

```
@app.route('/sessionStats/<string:id>', methods=['GET'])
def getCurrentSessionStats(id):
    conn = psycopg2.connect(db\_\_url)
    cur = conn.cursor()

    query = f'SELECT * FROM BotStats WHERE cloud_id = {id}' \
            f' ORDER BY created_at ASC LIMIT 1'

    cur.execute(query)
    result = cur.fetchone()
    return result
```

2.3.3.1 Postman

Postman je výkonný nástroj, ktorý možno použiť na vytvorenie dokumentácie pre naše API. Dokumentácia je pre každé API dôležitá, pretože umožňuje vývojárom pochopiť, ako s API pracovať a čo API dokáže. Pomocou nástroja Postman môžeme vytvoriť podrobnú a informatívnu dokumentáciu, ktorá vývojárom uľahčí prácu s našim API.[33]

Ak chceme vytvoriť dokumentáciu pomocou aplikácie Postman, musíme najprv vytvoriť kolekciu. Kolekcia je skupina volaní API, ktoré sú navzájom prepojené. Volania API môžeme v rámci kolekcie usporiadať do priečinkov, aby sme ľahšie našli konkrétne volania. Po vytvorení kolekcie môžeme začať dokumentovať svoje API pridávaním informácií ku každej požiadavke.[33]

V aplikácii Postman môžeme pridať dokumentáciu ku každej požiadavke pridaním popisu. Popis by mal obsahovať informácie o tom, čo požiadavka robí, aké parametre vyžaduje a ako bude vyzeráť odpoveď. Môžeme tiež pridať príklady, ako požiadavku použiť a ako bude vyzeráť

odpoveď. Tieto informácie sú dôležité pre vývojárov, ktorí sa snažia používať vaše rozhranie API.[33]

Ďalším dôležitým aspektom dokumentácie API je poskytovanie informácií o overovaní. Informácie o autentizácii môžeme do svojich požiadaviek v aplikácii Postman pridať pridaním hlavičky Authorization (Autorizácia) alebo pridaním parametrov do požiadavky. Do dokumentácie môžeme zahrnúť aj informácie o tom, ako získať kľúč API alebo prístupový *token*. [33]

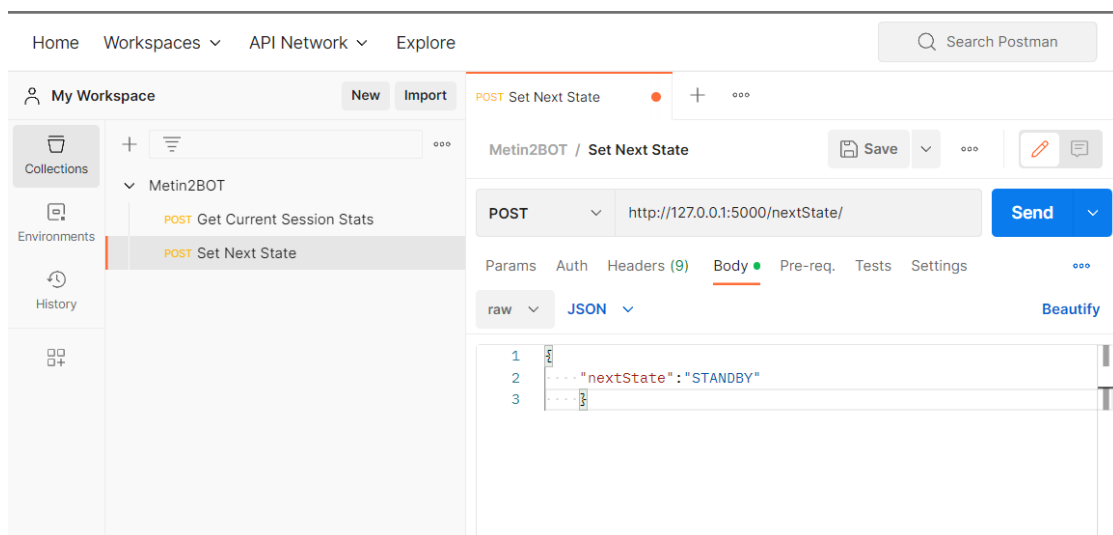
Okrem pridávania dokumentácie ku každej požiadavke môžeme vytvoriť aj komplexný dokument, ktorý slúži ako prehľad nášho API. Táto funkcia vygeneruje dokument HTML, ktorý obsahuje všetky požiadavky v našej kolekcii spolu s ich popisom a príkladmi. Dokumentáciu môžeme prispôsobiť pridaním ďalších informácií, ako sú napríklad zásady používania. [33]

Jednou z výhod používania aplikácie Postman na dokumentáciu API je to, že uľahčuje udržiavanie dokumentácie v aktuálnom stave. Keď vykonáme zmeny vo svojom rozhraní API, môžeme aktualizovať dokumentáciu v aplikácii Postman tak, aby tieto zmeny odrážala. Vývojári tak majú vždy prístup k najpresnejším a najaktuálnejším informáciám o vašom API. [33]

Celkovo je Postman vynikajúci nástroj na vytváranie dokumentácie. Vďaka jeho ľahko použiteľnému rozhraniu a výkonným funkciám môžete vytvoriť komplexnú a informatívnu dokumentáciu, ktorá vývojárom uľahčí pochopenie používania vášho API. Či už s dokumentáciou API len začínate, alebo hľadáte efektívnejší spôsob správy existujúcej dokumentácie, Postman určite stojí za zváženie. Napriek všetkým vyššie spomenutým faktom si však myslím, že tvorba dokumentácie kvôli dvom volaniam je zbytočná, považujem však za dôležité spomenúť tento výkonný nástroj, keďže pri pridávaní ďalších API volaní by sme túto technológiu určite využili.

2.3.4 Interface na ovládanie bota

Náš bot je ovládaný skrz jeho vystavenú API. V aktuálnej funkcionalite používame len 2 volania, popísané v sekcii **Flask API**. Považujem za zbytočné vytvárať vlastný interface za účelom použitia iba dvoch volaní. Použijeme teda vyššie spomenutý Postman interface, ktorý je jednoducho prístupný aj vo webovej verzii.



■ Obr. 2.6 Postman interface

2.4 Dockerizácia služieb

Docker je kontajnerová open-source platforma, ktorá umožňuje vývojárom vytvárať balíky aplikácií a služieb do kontajnerov. Kontajnery sú ľahké, samostatne spustiteľné jednotky, ktoré možno spustiť na akejkoľvek platforme bez potreby operačného systému. Poskytuje platformovo nezávislý spôsob nasadzovania a správy aplikácií, vďaka čomu je obľúbeným nástrojom pre cloud computing[34].

Používanie nástroja Docker v cloud computingu má niekoľko výhod, medzi ktoré patrí napr:

1. **Prenosnosť** - Kontajnery nástroja Docker možno spustiť na akejkoľvek platforme, čo uľahčuje presun aplikácií medzi vývojovými, testovacími a produkčnými prostrediami.
2. **Efektívnosť** - V porovnaní s tradičnými virtuálnymi počítačmi využívajú kontajnery Docker menej zdrojov, takže sú efektívnejšie a hospodárnejšie.
3. **Škálovateľnosť** - Docker uľahčuje škálovanie aplikácií nahor alebo nadol v závislosti od dopytu.
4. **Bezpečnosť** - Poskytuje bezpečné prostredie na spúšťanie aplikácií vďaka zabudovanej izolácii a sandboxu.

Architektúra Dockeru je založená na modeli klient-server, pričom klient komunikuje s Docker daemonom za účelom správy kontajnerov. Docker daemon beží na hosťujúcom počítači a spravuje kontajnery, zatiaľ čo klient môže byť spustený na akomkoľvek počítači, ktorý má prístup k daemonu[34].

Predtým, ako službu dockerizujeme, je nevyhnutné pochopiť architektúru služby a jej závislosti. To zahŕňa identifikáciu rôznych komponentov služby a ich interakcií, ako aj prípadných závislostí od externých služieb alebo knižníc.

Ak chceme službu dockerizovať, prvým krokom je vytvorenie image-u Docker. Docker image je snapshot služby alebo aplikácie a jej závislosti uložený v registri kontajnerov. Obraz možno použiť na vytvorenie kontajnerov, v ktorých sa služba spustí.

Vytvorenie dockeru image zahŕňa definovanie súboru Dockerfile, ktorý špecifikuje kroky potrebné na vytvorenie image-u. Súbor Dockerfile zvyčajne obsahuje pokyny na inštaláciu závislostí, konfiguráciu služby a nastavenie prostredia. Po vytvorení obrazu Docker ho možno použiť na spustenie kontajnerov.

Docker-compose je nástroj, ktorý sa používa na definovanie a spúšťanie viackontajnerových aplikácií Docker. Je to nástroj príkazového riadka, formou súboru YAML docker-compose definuje služby, ktoré tvoria aplikáciu, ich konfiguráciu a ich vzájomné vzťahy. Umožňuje vývojárom špecifikovať premenné prostredia, porty, zväzky a ďalšie konfigurácie potrebné na spustenie každej služby. Ďalej automatizuje proces spúšťania a zastavovania kontajnerov a zjednodušuje správu komplexných aplikácií, ktoré majú viacero závislostí a služieb.

Súbor docker-compose sa skladá z niekoľkých častí, vrátane:

1. **Verzia** - Určuje verziu formátu súboru docker-compose, ktorý sa má použiť.
2. **Services (služby)** - Definuje služby, ktoré tvoria aplikáciu, a ich konfiguráciu.
3. **Network (sieť)** - Určuje sieť, ktorú majú jednotlivé služby používať.
4. **Volumes** - Sú preferovaným mechanizmom na uchovávanie údajov generovaných a používaných docker kontajnermi .

Pomocou nástroja docker-compose môžu vývojári jednoducho nasadzovať, škálovať a aktualizovať svoje aplikácie a môžu jednoducho spravovať konfiguráciu každej služby. Môžu tiež jednoducho testovať svoje aplikácie v rôznych prostrediach, napríklad vo vývojovom, stagingovom a produkčnom, jednoduchou zmenou súboru docker-compose.

Celkovo je docker-compose výkonný nástroj, ktorý zjednodušuje nasadenie a správu komplexných aplikácií Docker. Umožňuje vývojárom sústrediť sa na vytváranie svojich aplikácií, a nie na správu infraštruktúry.

2.4.1 Dockerfile a docker-compose

Prvým krokom v dockerizácii našich služieb, je vytvorenie Dockerfile pre našu API. Ako prvé špecifikujeme používanú verziu Pythonu, ďalej nastavíme pracovný adresár na `/app`. Skopírujeme obsah aktuálneho adresára do kontajneru v `/app`. Nainštalujeme všetky závislosti a nastavíme *environment variables* pre Flask API. Ako posledné vystavíme port 5000 aby bolo možné k nemu prístupovať aj z vonkajška a spustíme Flask API.

■ Výpis kódu 2.17 API Dockerfile

```
FROM python:3.9-slim-buster

WORKDIR /app

COPY . /app

RUN pip install --no-cache-dir -r requirements_api.txt

ENV FLASK_APP=api.py

EXPOSE 5000

CMD ["flask", "run", "--host", "0.0.0.0"]
```

Ďalším krokom, je dockerizácia DB, ktorú vykonáme spoločne s vytvorením nášho docker-compose súboru. Keďže rôzne DB majú v dockeri preddefinované svoje *images*, nemusíme pre našu DB vytvárať samostatný Dockerfile. Ako prvé špecifikujeme docker verziu a zdefinujeme použité servisy. Servis *api* sa vytvorí na základe vyššie definovaného Dockerfile, aby naša API vedela prístupovať do DB, musíme zdefinovať DB url v premennej *environment*. Ďalším krokom je definícia samotnej DB, ktorú sme zvolili ako Postgres databázu. V definícii DB servisy zadáme prihlasovacie údaje a meno DB, ktorú budeme používať. Pomocou premennej *volumes* zabezpečí persistenciu dát. Ako posledné musíme vystaviť porty pre API a pre DB aby boli prístupné aj z vonkajšku.

■ Výpis kódu 2.18 Docker-compose

```
version: '3.9'
services:
  api:
    build: .
    ports:
      - "5000:5000"
    environment:
      - DATABASE_URL=postgres://user:password@db:5432/dbname
    depends_on:
      - db
  db:
    image: postgres
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: dbname
    volumes:
```

```

- db_data:/var/lib/postgresql/data
ports:
- "5432:5432"

volumes:
  db_data:

```

2.5 Cloudové riešenie

V tejto kapitole si viac priblížime naše cloudové riešenie. Budeme hovoriť o problémoch, s ktorými sme sa stretli a o potrebnej konfigurácii cloudu.

2.5.1 Voľba cloudového providera

Vychádzajúc zo sekcie **Prieskum cloudových providerov** a taktiež po dobrej predošlej skúsenosti sme zvolili ako nášho cloudového providera spoločnosť Hyonix. Tento provider poskytuje výhradne Windows VPS, verím že touto úzkou špecializáciou bude riešenie akýchkoľvek potencionálnych problémov o to rýchlejšie. Okrem Windows špecializácie, patria medzi najlacnejších providerov na trhu a poskytujú 24/7 podporu.

The screenshot shows the Hyonix cloud management interface. At the top, there is a navigation menu with options: Servers, Billing, Support, Affiliates, and Account. The main content area is titled 'Servers' and includes a search bar, a 'Sort Status' dropdown, and a table of active servers. The table has columns for Name, IP Address, Due Date, Pricing, Status, and Action. Two servers are listed: 'JANKEDAN-DIPLO HS-1/' and 'DIPLOMKA-V2 HS-2/'.

Name	IP Address	Due Date	Pricing	Status	Action
JANKEDAN-DIPLO HS-1/	167.94.158.156	2023-05-03	\$6.50 Monthly	Active	
DIPLOMKA-V2 HS-2/	188.210.236.45	2023-05-18	\$12.00 Monthly	Active	

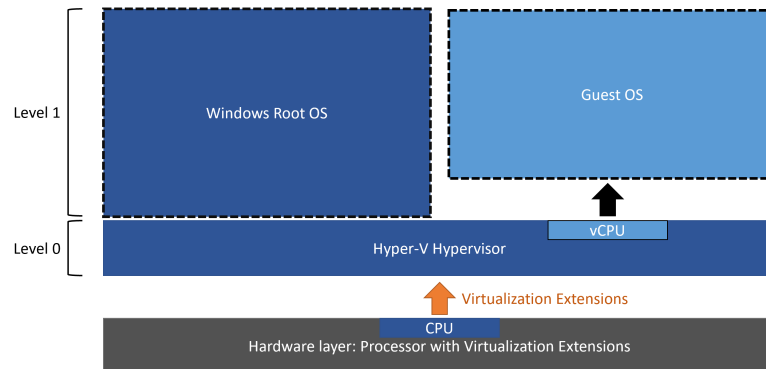
■ Obr. 2.7 Typy featur [18]

2.5.2 Vnorená virtualizácia

Ako prvý problém, na ktorý sme narazili bola zakázaná vnorená virtualizácia. Táto funkcia je dôležitá z dôvodu nasadenia našich dockerizovaných služieb. Docker využíva službu Hyper-V, ktorá poskytuje platformu na vytváranie a správu virtualizovaných prostredí vrátane operačných systémov, aplikácií a hardvérových prostriedkov.

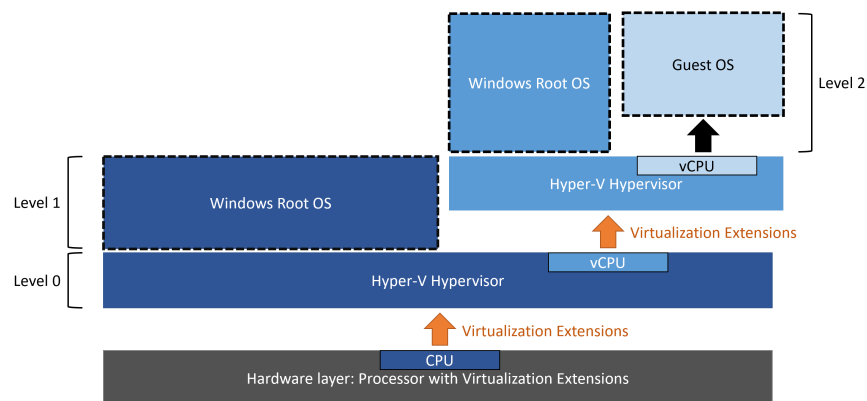
Vnorená virtualizácia je funkcia, ktorá umožňuje spustiť Hyper-V vo vnútri virtuálneho počítača Hyper-V. To je užitočné napríklad na spustenie emulátora telefónu Visual Studio vo virtuálnom počítači alebo na testovanie konfigurácií, ktoré si bežne vyžadujú niekoľko hostiteľov alebo v našom prípade vytvorenie docker kontajnerov v už existujúcom virtuálnom prostredí, ktorým je náš server[35].

Na nasledujúcom obrázku je znázornená technológia Hyper-V bez vnárania. Hypervízor Hyper-V preberá plnú kontrolu nad hardvérovými virtualizačnými schopnosťami (oranžová šípka) a nevystavuje ich hostujúcejmu operačnému systému[35].



■ Obr. 2.8 Zakázaná vnorená virtualizácia [35]

Na rozdiel od toho je na nasledujúcom diagrame zobrazený Hyper-V s povolenou vnorenou virtualizáciou. V tomto prípade Hyper-V vystavuje hardvérové virtualizačné rozšírenia svojmu virtuálnemu počítaču. Pri zapnutom vnáraní môže hostujúci virtuálny počítač nainštalovať vlastný hypervízor a spúšťať vlastné hostujúce virtuálne počítače[35].



■ Obr. 2.9 Povolená vnorená virtualizácia [35]

2.5.3 Konfigurácia siete

K umožneniu prístupu na naše API endpointy, budeme musieť nakonfigurovať sieťové nastavenia systému Windows.

1. Stlačíme tlačidlo Windows a vyhľadáme *Windows Defender Firewall with Advanced Security*

2. Pravým klikneme na Inbound Rules - Zvolíme *New Rule*
3. Rule Type - Zvolíme možnosť Port
4. Protocol and Ports - Zvolíme možnosť TCP a špecifikujeme port 5000, na ktorom beží naša API
5. Action - Ponecháme zvolenú možnosť *Allow the connection*
6. Profile - Tu môžeme ponechať všetky možnosti zakliknuté
7. Name - Zvolíme meno nášho pravidla a jeho popis

V tomto momente by mala sieť byť úspešne nakonfigurovaná, môžeme to overiť skontrolovaním otvorených portov. Do *Windows PowerShell* zadáme nasledujúci príkaz:

■ Výpis kódu 2.19 Kontrola otvorených portov

```
netstat -aon
```

Tu môžeme vidieť všetky aktívne pripojenia k nášmu PC, nás zaujímajú prevažne pripojenia v stave *LISTENING*, presnejšie existencia pripojenia s portom 5000 v tomto stave. Ako môžeme vidieť nižšie, takéto pripojenie sa tam naozaj nachádza, máme teda úspešne nakonfigurovanú sieť.

Proto	Local Address	Foreign Address	State	PID
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING	1276
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING	4
TCP	0.0.0.0:5000	0.0.0.0:0	LISTENING	24892
TCP	0.0.0.0:5040	0.0.0.0:0	LISTENING	6396
TCP	0.0.0.0:5432	0.0.0.0:0	LISTENING	24892

■ Obr. 2.10 Kontrola bežiacich kontajnerov

2.5.4 Tvorba snímky systému Windows

Snímku systému Windows vytvárame na záver z dôvodu uchovania konfigurácie siete a uchovania stiahnutým súborov. Jednou z možností ako ju vytvoríme je použiť nástroje ako sú napr. NTLite alebo WinReducer, pomocou týchto nástrojov vieme vytvoriť Windows ISO Image, ktorý obsahuje potrebné predinštalované programy.

Pri vytváraní Windows ISO snímky, sme sa riadili videom , ktoré vieme nájsť tu. Pre krátke zhrnutie tento proces vieme opísať do nasledovných bodov:

1. Stiahneme základný ISO Image systému Windows 10
2. Stiahneme VirtualBox a spustíme pomocou stiahnutého ISO image. Proces vytvárania virtuálneho počítača pomocou VirtualBoxu sme bližšie popisali v kapitole **Potrebné nástroje k tvorbe vlastného Windows ISO obrazu** .VirtualBox používame aby sme získali čistú systémovú inštaláciu, na ktorú následne nainštalujeme potrebné nástroje a knižnice.
3. Pomocou nástroja Disk Management, vytvoríme virtuálnu partíciu, ktorá bude obsahovať všetky nástroje, ktoré chceme mať v našom ISO obraze nainštalované.
4. Spustíme jeden z *Windows Deployment Tools* nástrojov - *Sysprep*, ktorý nám ďalej pripraví systém Windows do potrebného stavu. Po dokončení tohto kroku, by sa malo naše virtuálne prostredie vypnúť.

5. Vo VirtualBoxe zvolíme možnosť Storage a do prázdneho slotu mountneme náš Windows PE - malý operačný systém používaný na inštaláciu, nasadenie a opravu Windows systému.
6. Znova spustíme naše virtuálne prostredie, pomocou klávesy F12, naboostujeme systém z moun-tnutého virtuálneho disku Windows PE.
7. Ďalším krokom bude vytvorenie samotnej Windows ISO snímky, spustíme konzolu s Admin oprávneniami a vložíme nasledujúci príkaz:

■ **Výpis kódu 2.20** Tvorba install.wim súboru

```
dism /capture-image /ImageFile:"D:\install.wim" ^
/CaptureDir:C:\ /Name:Win10Apps /compress:max
```

8. Po dobehnutí predloženého príkazu prejdeme do zložky D:, kde nás bude čakať súbor *install.wim*, ktorý obsahuje nainštalované aplikácie, tento súbor môžeme otvoriť pomocou programu 7-zip a preskúmať jeho obsah.
9. Ďalším krokom bude presun súboru *install.wim* z virtuálneho prostredia, do prostredia nášho PC. Na tento proces použijeme medium, vo forme USB kľúča.
10. Posledným krokom bude vytvorenie Windows ISO Image, použijeme nástroj AnyBurn, ktorý nám umožní upraviť existujúci Windows ISO Image. V nástroji AnyBurn, otvoríme ten istý image, z ktorého sme vytvorili naše virtuálne prostredie a začneme ho upravovať. Tento image, zatiaľ nedisponuje nainštalovanými súborami. Je dôležité lokalizovať súbor xx, ktorý sa nachádza v ceste:

■ **Výpis kódu 2.21** Cesta k install.wim

```
sources/install.wim
```

alebo

■ **Výpis kódu 2.22** Cesta k install.wim

```
sources/install.esd
```

Odstránime príslušný súbor a nahradíme ho vyššie vytvoreným našim súborom *install.win*.

11. Potvrdíme zmeny v ISO súbore a nástroj AnyBurn vytvorí nový upravený Windows ISO Image s nainštalovanými aplikáciami.

Na záver je dôležité spomenúť, že pokiaľ sa nechceme dopustiť pirátstva, je dôležité mať platnú Windows licenciu. Licencia na systém Windows je totiž viazaná na hardvér počítača a spustenie systému Windows vo virtuálnom počítači je v podstate spustenie systému vo virtualizovanom hardvérovom prostredí. Ak teda chceme používať systém Windows vo virtuálnom počítači, potrebujeme platnú licenciu na systém Windows.

Z tohto pravidla existujú určité výnimky, napríklad ak máme licenciu Windows Enterprise s právami na virtualizáciu alebo ak používame virtualizačné riešenie, ktoré poskytuje vlastné licencie Windows (napríklad Azure Virtual Machines od spoločnosti Microsoft). Vo väčšine prípadov však na spustenie vo virtuálnom počítači budeme potrebovať samostatnú licenciu systému Windows.

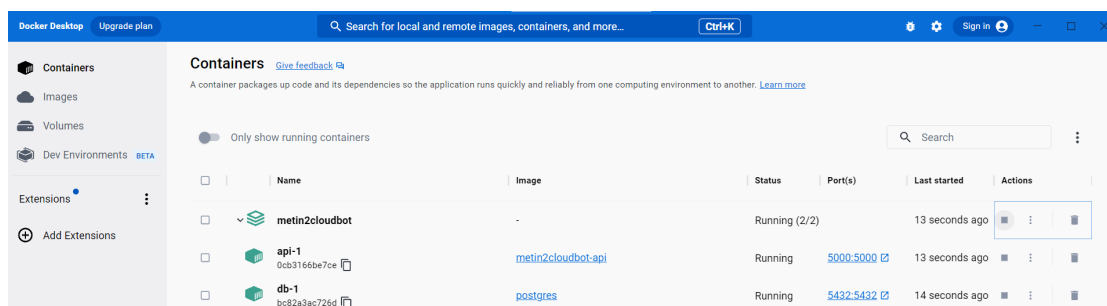
2.5.5 Spustenie bota a pridružených služieb

Spustenie nášho bota je veľmi jednoduché, najprv spustíme jeho zdockerizované služby, ktorými sú API a databáza. Keďže sme vytvorili vlastnú snímku Windows, ktorá obsahuje všetky potrebné súbory. Otvoríme *Windows Powershell* a prejdeme do adresára, ktorý obsahuje súbory nášho bota. V adresári spustíme nasledovný príkaz:

■ **Výpis kódu 2.23** Build docker obrazu bota

```
docker-compose up --build -d
```

Tento príkaz zbuildí príslušné docker obrazy a následne ich spustí. V prípade, že všetko prebehne hladko, vieme v aplikácii *Docker Desktop* skontrolovať beh našich kontajnerov.



■ **Obr. 2.11** Kontrola bežiacich kontajnerov

Ďalej nám zostáva už len spustiť bota samotného, nezakomponovali sme ho do *docker-compose.yml* súboru, keďže chceme aby API a databáza bežali nezávisle od behu bota. Bota spustíme jednoduchým príkazom v adresári:

■ **Výpis kódu 2.24** Spustenie bota

```
py main.py
```

Po spustení tohto príkazu, bot prevezme kontrolu nad vašou myškou, začne spracovávať obraz, následne vyhľadávať a fariť kamene Metin.

Testovanie riešenia

3.1 Statické testovanie kódu

Statické testovanie kódu je typ testovania, ktorý nezahŕňa spustenie kódu. Ide o proces vyhodnocovania bez jeho skutočného spustenia. Tento typ testovania sa môže vykonávať ručne alebo pomocou nástrojov. Môže pomôcť identifikovať chyby v ranom štádiu procesu vývoja, znížiť náklady na opravu chýb a zlepšiť celkovú kvalitu kódu. Je dôležitou súčasťou životného cyklu vývoja softvéru.

Na statické otestovanie nášho kódu sme použili nástroj *pylint*. Nainštalujeme ho pomocou nasledujúceho príkazu:

■ Výpis kódu 3.1 Spustenie bota

```
py -m pip3 install pylint
```

Následne ho spustíme na našom súbore *main.py*, keďže *main* je previazaný so všetkými ostatnými triedami, *pylint* rekurzívne otestuje aj ich. Spúšťame ho s flagom *-errors-only*, ktorý zabezpečí, že upozornení budeme iba na errorry.

■ Výpis kódu 3.2 Spustenie bota

```
pylint main.py
```

Spustením nástroja *pylint* získavame nasledujúci výstup:

■ Výpis kódu 3.3 Spustenie bota

```
***** Module main
main.py:1:0: C0114: Missing module docstring (missing-module-docstring)
main.py:11:0: C0116: Missing function or method docstring (missing-function-docstring)
main.py:48:26: E1101: Module 'cv2' has no 'addWeighted' member (no-member)
main.py:51:8: E1101: Module 'cv2' has no 'imshow' member (no-member)
main.py:55:14: E1101: Module 'cv2' has no 'waitKey' member (no-member)
main.py:60:12: E1101: Module 'cv2' has no 'destroyAllWindows' member (no-member)
main.py:66:0: C0116: Missing function or method docstring (missing-function-docstring)
main.py:98:0: C0116: Missing function or method docstring (missing-function-docstring)
main.py:116:0: C0116: Missing function or method docstring (missing-function-docstring)

-----
Your code has been rated at 6.91/10
```

Môžeme si všimnúť, že *pylint* nás upozornil na 2 typy chýb:

1. missing-function-docstring - Upozorňuje nás, že niektoré funkcie nemajú docstring, čiže chýba nám nejaká popisná funkcia. Toto upozornenie je iba na úrovni **WARNING**, môžeme ho teda ignorovať
2. no-member - Pylint má pocit, že cv2 neobsahuje vyššie vypísané funkcie. To však nieje pravda, keďže tieto funkcie používame v našej implementácii, túto chybu môžeme teda takisto ignorovať

Znova spustíme pylint príkazom ktorým ignorujeme vyššie spomenuté dve chybové hlášky:

■ Výpis kódu 3.4 Spustenie bota

```
pylint --disable=E1101 --errors-only main.py
```

Na výstupe dostávame nasledujúce hlásenie:

■ Výpis kódu 3.5 Spustenie bota

```
***** Module main
-----
Your code has been rated at 10/10 (previous run: 6.91/10, +3.09)
```

Statické testovanie týmto krokom ukončujeme a prechádzame do praktického testovania nasadenia nášho bota.

3.2 Testovanie efektivity bota

Rozhodli sme sa, že najlepšie otestujeme nášho bota praktickým nasadením. Nasadili sme teda jednotlivé natréňované modely na dobu 1h a pozorovali sme ich efektivitu počas troch behov. Štatistické údaje ktoré pozorujeme sú:

1. Počet zničených kameňov Metin - Dôležitý indikátor nakoľko našim cieľom je dosiahnuť čo najvyššie číslo v tejto metrike
2. Počet vstupov do ERROR stavu - Vysoká chybovosť môže poukázať na špecifický problém na ktorý bot naráža
3. Počet iterácii celého cyklu bota - Pokiaľ sa bot opakovane dostáva do stavu ERROR, jeho beh sa zastaví. Môžeme teda očakávať variáciu v tomto údaji

Detekčný model	Počet zničených kameňov	Počet vstupov do ERROR stavu	Počet iterácii cyklu bota
Model č.1	1	7	8
Model č.2	16	24	40
Model č.3	35	5	40

■ Tabuľka 3.1 Testovací beh č. 1

Z prvého behu pozorujeme nasledujúce fakty:

- **Model č.1** - Je viditeľne zlé natréňovaný, počas svojho behu zvládol zničiť iba 1 kameň Metin. Následne sa dostal do persistentného stavu ERROR a predčasne ukončil svoj beh
- **Model č.2** - Taktiež zle natréňovaný, dosiahol však podstatne lepšie výsledky oproti Modelu č.1. Podľa môjho názoru dosiahol až prekvapivo vysoký počet zničených kameňov Metin. Domnievam sa, že môže ísť o falošne-pozitívne výsledky.

Detekčný model	Počet zničených kameňov	Počet vstupov do ERROR stavu	Počet iterácii cyklu bota
Model č.1	0	7	7
Model č.2	20	20	40
Model č.3	39	1	40

■ **Tabuľka 3.2** Testovací beh č. 2

- **Model č.3** - Podľa očakávaní beh prebehol pomerne hladko.
- **Model č.1** - Potvrdil nám pozorovanie z predošlého behu, model je zle natrénovaný
- **Model č.2** - Model dosiahol v tomto behu lepšie výsledky, stále sa však dostáva príliš často do stavu ERROR. Je možné, že bot detekuje objekt, o ktorom si myslí že je kameň Metin, ako môžeme vidieť na obrázku 2.5 a zničí ho. Nejedná sa však o kameň metin ale iba o príšeru. Indikuje nám na to aj fakt, že bot sa príliš často dostáva do stavu ERROR. Pokiaľ budeme pozorovať podobné výsledky pri 3. pokusnom behu, berieme toto pozorovanie za potvrdené.
- **Model č.3** - Dosiahli sme lepšie výsledky než v predošlom behu.

Detekčný model	Počet zničených kameňov	Počet vstupov do ERROR stavu	Počet iterácii cyklu bota
Model č.1	0	7	7
Model č.2	16	24	40
Model č.3	37	3	40

■ **Tabuľka 3.3** Testovací beh č. 3

- **Model č.1** - Môžeme sebedovomo tvrdiť, že zničenie jedného kameňa v prvom behu bola náhoda alebo falošne-pozitívny výsledok.
- **Model č.2** - Zvýšená chybovosť potvrdzuje náš predpoklad, model je zle natrénovaný a prevažne sa jedná o falošne-pozitívne výsledky v metrike *Počet zničených kameňov*.
- **Model č.3** - Model dosiahol uspokojivé výsledky.

Počas posledných dvoch testovacích behov, boli potvrdené naše predpoklady. Modely č.1 a č.2 sú zle natrénované, nedosahujú požadované výsledky a pri opakovanom nasadení by bolo potrebné určite zmeniť tréningové parametre. Model č.3 dosahoval uspokojivé výsledky, nie však bezchybné. Dostával sa taktiež do chybového stavu, čo je indikátorom, nie úplne kvalitného tréningu.

Naša implementácia funguje takmer bezchybne, to však neznamená, že nieje priestor na akékoľvek vylepšenia. Oblasti, ktoré by takéto vylepšenie uvítali môžeme rozdeliť zhrnúť v nasledujúcich bodoch:

- 1. Detekcia zostávajúceho života kameňov Metin** - V súčasnosti naša implementácia používa pevne daný časový interval počas ktorého charakter útočí na kameň Metin. Tento interval by sme však mohli nahradiť detekciou zostávajúceho života daného kameňa. Keďže v hre Metin2 existujú rôzne typy postáv, každá postava ničí kamene inou rýchlosťou. Pri používaní bota s charaktermi s rôznou výbavou, levelom a triedou, nedosahujeme maximálnu efektivitu pevne stanoveným intervalom, preto je toto vylepšenie prvé na zozname.
- 2. Imitácia správania hráča** - Ako sme spomenuli, využívanie botov nie je všeobecne v MMORPG hrách povolené. Ak sa chceme vyvarovať blokácie našej postavy, bot by mal vedieť fungovať na nerozoznanie od hráča. V triviálnych scenároch môžeme hovoriť o spôsobe akým sa postava pohybuje a správa. Boti sa správajú veľmi lineárne, majú veľkú odozvu medzi jednotlivými úkonmi a preto oku skúseného hráča bežný bot neunikne. Okrem plynulých úkonov by určite bolo veľkým vylepšením ak by bot vedel komunikovať s hráčmi aspoň na veľmi elementárnej úrovni (bežný bot vám nikdy neodpíše na hernú správu), takáto vlastnosť by určite prispela k legitimnosti vašeho herného účtu.
- 3. Vyhýbanie sa hráčom** - Ďalším spôsobom ako predísť blokácii účtu je všeobecne vyhýbanie sa hráčom. Podobne ako sme trénovali bota na detekciu kameňov Metin, by sme mohli trénovať bota na detekciu hráčov v našom okolí. Pri úspešnej detekcii by sa náš charakter mohol presunúť ďalej od detekovaného hráča, aby predišiel nahláseniu a tým aj následnému prevereniu účtu.
- 4. Rozšírenie sady úloh k automatizácii** - V hernom svete Metin2 sú desiatky úkonov, ktoré môžu byť automatizované, ako napríklad ťaženie rudy, zabíjanie *bossov* alebo plnenie misií.
- 5. Optimalizácia použitého modelu na detekciu objektov** - Pri testovaní nasadeného bota sme zistili, že Model č.3, ktorý využívame vo finálnej verzii sa taktiež dostáva do chybového stavu. Považujem za dôležitú počet vstupov do takéhoto stavu minimalizovať a preto by som navrhol ďalšiu optimalizáciu nášho modelu alebo celkové pretrenovanie s inými parametrami.
- 6. Automatické generovanie inštancií bota podľa požiadavok užívateľa** - S touto komplexnou funkciou by sme dokázali naplno využívať cloudovú implementáciu nášho bota. Pri jej implementácii však vzrastá veľké množstvo výziev, ktorým by sme museli čeliť ako programátori. Nakoľko okrem v práci popísaných problémov, existuje niekoľko ďalších, ktoré sme ešte nespomenuli.

Hlavnou výhodou nášho bota oproti ostatným je jeho cloudové nasadenie. Teoreticky vieme púšťať desiatky až stovky jeho inštancií, čo by nám dalo bezkonkurenčnú výhodu oproti ostatným hráčom. Takýmto ťahom by sme však úplne zničili ekonomiku hry, čo považujem za nemorálne. Náš bot je teda využiteľný najmä hráčmi hry Metin2, ktorí nechcú tráviť dlhé hodiny repetitívnymi úlohami a chcú si spríjemniť hru. Automatizuje činnosť, ktorá zaberá hráčom najviac času a tou je ničenie kameňov Metin.



Kapitola 5

Záver

Naša práca sa zaoberala návrhom a následnou implementáciou Metin2 bota nasadeného v cloudovom prostredí. Prvým z cieľov bol prieskum existujúcich riešení. Z prieskumu sme zistili, že všetky riešenia bežia výhradne lokálne a používajú rôzne metódy detekcie herných objektov. Najčastejšie používaná knižnica naprieč všetkými projektami bola OpenCV, ktorú sme sa následne rozhodli použiť aj v našej implementácii. Preskúmali a porovnali sme rôzne metódy detekcie objektov a následne zvolili tú najvhodnejšiu vzhľadom na našu konkrétnu situáciu.

Ďalej sme navrhli sadu vhodných úloh k automatizácii. Prvou úlohou, bolo automatické používanie *Plášťov statočnosti*, ktorú sme splnili. Druhou úlohu, ktorá tvorila jadro našej práce, bola detekcia herných objektov, kameňov Metin a ich následné ničenie. Tento cieľ sme takisto splnili, výsledky môžeme pozorovať v sekcii **Testovanie riešenia**.

Vzhľadom na cloudové nasadenie nášho bota, bol dôležitý prieskum cloudových providerov, ktorí splňovali nami definované nefunkčné požiadavky aplikácie. Kľúčovou požiadavkou, bolo poskytovanie virtuálneho stroja so systémom Windows a sprístupneným grafickým rozhraním.

Architektúru riešenia sme navrhli tak, aby bola jednoducho rozšíriteľná do podoby, kde by našich botov bežalo paralelne desiatky. S touto myšlienkou, sme vytvorili vlastný Windows ISO obraz, z ktorého môžeme vytvárať jednotlivé inštancie virtuálnych počítačov. Ďalej sme navrhli a implementovali dosiahnuteľné stavy a prechodné podmienky nášho bota.

Cieľ práce sa teda podarilo splniť, výstupom je funkčný bot ako stavový stroj, ktorý je nasadený v cloudovom prostredí.

..... Dodatok A

Zoznam skratiek

API Application Programming Interface
REST Representational state transfer

Bibliografia

1. *Code Injection*. cheatengine.org. Dostupné tiež z: https://owasp.org/www-community/attacks/Code_Injection.
2. SZELISKI, R. *Computer vision: algorithms and applications* [online]. Springer Science Business Media, [b.r.].
3. LECUN, Y.; BENGIO, Y.; HINTON, G. (2015). *Deep learning* [online]. Nature, [b.r.].
4. REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. (2016). *You only look once: Unified, real-time object detection*. [Online]. [B.r.].
5. WANG, C.; CHEN, Q.; ZHAO, B.; ZHANG, L. (2020). *Bot-based intelligent cloud service management: A survey and future directions*. [Online]. [B.r.].
6. SAXENA, A.; DRIEMEYER, J.; NG, A. Y. (2009). *Robotic perception: A computer vision approach* [online]. Communications of the ACM, [b.r.].
7. LOWE, D. G. (2004). *Distinctive image features from scale-invariant keypoints*. [Online]. International Journal of Computer Vision, [b.r.].
8. GONZALEZ, R. C.; WOODS, R. E. (2018). *Digital image processing*. [Online]. [B.r.].
9. GIRSHICK, R.; DONAHUE, J.; DARRELL, T.; MALIK, J. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation*. [Online]. [B.r.].
10. DALAL, N.; TRIGGS, B. (2005). *Histograms of oriented gradients for human detection*. [Online]. [B.r.].
11. REN, S.; HE, K.; GIRSHICK, R.; SUN, J. (2015). *Faster R-CNN: Towards real-time object detection with region proposal networks*. [Online]. [B.r.].
12. LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S.; BERG, A. C. (2016). *SSD: Single shot multibox detector*. [Online]. [B.r.].
13. HUANG, L.; YANG, Y.; DENG, Y.; YU, Y. (2017). *DenseBox: Unifying landmark localization with end to end object detection*. [Online]. [B.r.].
14. LIN, T. Y. (2018). *Focal loss for dense object detection*. [Online]. [B.r.].
15. LIN, T. Y.; DOLLÁR, P.; GIRSHICK, R.; HARIHARAN, B.; BELONGIE, S. (2017). *Feature pyramid networks for object detection*. [Online]. [B.r.].
16. LIN, T. Y.; GOYAL, P.; GIRSHICK, R.; HE, K.; DOLLÁR, P. (2017). *Focal loss for dense object detection*. [Online]. [B.r.].
17. LIENHART, R.; MAYDT, J. (2002). *An extended set of Haar-like features for rapid object detection*. [Online]. [B.r.].

18. *Cascade Classifier*. opencv.org. Dostupné tiež z: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.
19. M., Armbrust; A., Fox; R., Griffith; D., Joseph A.; R., Katz; A., Konwinski; ZAHARIA, M. (2010). *A view of cloud computing*. [Online]. [B.r.].
20. G., Cassandras C.; (2008), Lafortune S. *Introduction to discrete event systems* [online]. [B.r.].
21. *VirtualBox Manual*. virtualbox.org. Dostupné tiež z: <https://www.virtualbox.org/wiki/Downloads#manual>.
22. *WinPE Intro*. learn.microsoft.com. Dostupné tiež z: <https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/winpe-intro?view=windows-11>.
23. *What is an API (application programming interface)?* ibm.com. Dostupné tiež z: <https://www.ibm.com/topics/api>.
24. *Websockets API*. developer.mozilla.org. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.
25. *Polling*. techtarget.com. Dostupné tiež z: <https://www.techtarget.com/whatis/definition/polling>.
26. *Long - Polling*. javascript.info. Dostupné tiež z: <https://javascript.info/long-polling>.
27. *What is rest API*. redhat.com. Dostupné tiež z: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>.
28. *MySQL Advantages and Disadvantages*. w3schools.blog. Dostupné tiež z: <https://www.w3schools.blog/mysql-advantages-disadvantages>.
29. *What is mongodb*. mongodb.com. Dostupné tiež z: <https://www.mongodb.com/what-is-mongodb>.
30. *PostgreSQL benefits and Challenges*. infoworld.com. Dostupné tiež z: <https://www.infoworld.com/article/3619531/postgresql-benefits-and-challenges-a-snapshot.html>.
31. *Python documentation*. python.org. Dostupné tiež z: <https://www.python.org/doc/>.
32. *Flask User's Guide*. palletsprojects.com. Dostupné tiež z: <https://flask.palletsprojects.com/en/2.3.x/>.
33. *Postman documentation*. learning.postman.com. Dostupné tiež z: <https://learning.postman.com/docs>.
34. *Docker docs*. docs.docker.com. Dostupné tiež z: <https://docs.docker.com/desktop/>.
35. *Nested Virtualisation*. learn.microsoft.com. Dostupné tiež z: <https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/user-guide/nested-virtualization>.

Obsah přiloženého média

	readme.txt.....	stručný popis obsahu média
	src	
	impl.....	zdrojové kódy implementace
	thesis.....	zdrojová forma práce ve formátu L ^A T _E X
	text.....	text práce
	thesis.pdf.....	text práce ve formátu PDF