České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství
Obor: Aplikace softwarového inženýrství

Grafický simulátor Ozobot Evo

Graphical simulator for Ozobot Evo

BAKALÁŘSKÁ PRÁCE

Vypracoval: Martin Švamberg
Vedoucí práce: Ing. Josef Nový
Rok: 2020

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství Akademický rok 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	**Martin Švamberg**
Studijní program:	**Aplikace přírodních věd**
Obor:	**Aplikace softwarového inženýrství**
Název práce česky:	**Grafický simulátor pro Ozobot Evo**
Název práce anglicky:	**Graphical Simulator for Ozobot Evo**

Pokyny pro vypracování:

1. Analyzujte možnosti robota Ozobot Evo a nástroje poskytované jeho výrobcem.
2. Navrhněte nástroj pro simulaci chování Ozobota využívající framework Qt.
3. Navržený nástroj implementujte.
4. Implementovaný nástroj otestujte.

Doporučená literatura:

[1] LAZAR, G. and PENEA, R. *Mastering Qt 5 - Second Edition.* 2018. ISBN 978-1788995399.

[2] BAKA, B. *Getting Started with Qt 5.* 2019. ISBN 978-1789956030.

[3] ENG, L. Z. *Qt5 C++ GUI Programming Cookbook - Second Edition.* 2019. ISBN 978-1789803822.

[4] Ozobot [online]. [cit. 2019-08-11]. Dostupné z: https://ozobot.com/

[5] OzoBlockly [online]. [cit. 2019-08-11]. Dostupné z https://ozoblockly.com/

Jméno a pracoviště vedoucího práce:

Ing. Josef Nový
Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, ČVUT v Praze

Jméno a pracoviště konzultanta:

–

.....
vedoucí práce

Datum zadání bakalářské práce: 11. 10. 2019

Termín odevzdání bakalářské práce: 7. 7. 2020

Doba platnosti zadání je dva roky od data zadání.

.....
garant oboru

.....
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
-1-

.....
děkan

V Praze dne 11. 10. 2019

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Děčíně dne .....

.....
Martin Švamberg

Poděkování

Děkuji vedoucímu práce Ing. Josefu Novému za svědomité vedení mé bakalářské práce a neocenitelné rady při její tvorbě.

Martin Švamberg

Název práce:
Grafický simulátor Ozobot Evo

Autor: Martin Švamberg

Studijní program: Aplikace přírodních věd
Obor: Aplikace softwarového inženýrství
Druh práce: Bakalářská práce

Vedoucí práce: Ing. Josef Nový
Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze

Abstrakt: Tato bakalářská práce se zabývá vytvořením aplikace pro simulaci chování nástroje Ozobot evo, používaného jako nástroj k výuce základů programování a rozvíjení logického myšlení. K vytvoření této desktopové aplikace byl použit framework Qt 5. Výsledkem je stabilní aplikace otestována studenty. V průběhu práce je popsán postup vývoje, testování a ukázka finálního návrhu simulátoru.
Klíčová slova: Ozobot , Evo, simulace, Qt 5, Výuka

Title:
Graphical simulator for Ozobot Evo

Author: Martin Švamberg

Abstract: This bachelor's thesis focuses on creating application for simulating behavior of tool Ozobot Evo, which is used for education of basic programing and development of logical thinking. For creation of this desktop application was used framework Qt 5. As result we have stable and student tested application. Thesis describes procces of developing, testing and in the end shows finished simulation tool.
Key words: Ozobot , Evo, simulation, Qt 5, Education

Obsah

Úvod **11**

1 Analýza možností robota Ozobot Evo a nástrojů poskytnutých výrobcem **13**
1.1 Technické parametry a informace 13
1.2 Možnosti programování a použití 13
1.2.1 Čtení kódu z předkreslené čáry 13
1.2.2 Webový nástroj pro programování ozobota 21
1.2.3 Android aplikace Ozobot Evo 31

2 Navržení nástroje pro simulaci chování Ozobot Evo využívající framework Qt **33**
2.1 Uživatelé nástroje 33
2.2 Volba implementačního nástroje 33
2.3 První grafický návrh rozhraní 34

3 Implementace nástroje **36**
3.1 Popis tříd 36
3.1.1 Třída MainWindow 38
3.1.2 Třída Mapování 38
3.1.3 Třída Form2_Kreskody 38
3.1.4 Třída Vykresleni 46
3.1.5 Třída Ozobot 48
3.2 Finální grafický návrh rozhraní 53

4 Otestování implementovaného nástroje **56**
4.1 Testující 56
4.2 Testování První verze 56
4.2.1 První testující 57
4.2.2 Druhý testující 57
4.2.3 Třetí testující 58
4.2.4 Čtvrtý testující 58
4.2.5 Výsledek testování první verze 59
4.3 Testování Finální verze 59
4.3.1 První testující 59
4.3.2 Druhý testující 60
4.3.3 Třetí testující 60

4.3.4 Čtvrtý testující 61
4.3.5 Shrnutí testování 61

Závěr **62**

Literatura **63**

Přílohy **64**

A Uživatelský manuál **64**

B Ukázka pdf exportu **67**

C Obsah CD **70**

Úvod

Cílem této bakalářské práce je vytvoření grafického simulátoru učebního nástroje robota Ozobot Evo. Ozobot Evo se využívá k učení základních principů programování na středních školách, a zejména základní školy o něj začaly jevit zájem. Důvodem, proč jsem si vybral toto téma je pomoc se zjednodušením testování vytvořených programů studenty, nebo vyučujícími.

Pomocí grafického simulátoru bude odstraněna nutnost mít pro otestování vždy přítomného robota Ozobot Evo. Umožní to tak možnost práce z domova a také rovnou otestovat výsledek před nahráním programu do robota. Zároveň to pomůže s možným nedostatkem počtu jednotlivých Ozobotů na daných pracovištích škol.

Prvním z důležitých kroků je prozkoumání programovacího prostředí poskytnutého výrobcem Ozobot Evo. Je tedy potřeba posoudit a prozkoumat všechny možnosti, kterých je schopen. A to buď na stránkách výrobce, nebo přímo v jejich programovacím prostředí.

To je velmi podstatné pro navržení grafické simulátoru samo o sobě, jelikož musí obsáhnout přinejmenším nejdůležitější aspekty možností Ozobot Evo. Dále je kladen velký důraz na zpětnou kompatibilitu se stránkami výrobce. Hlavním účelem simulátoru totiž není jen otestování již vypracovaného kódu v programovacím prostředí od výrobce, ale také možnost vytvoření základních programů přímo v něm. A proto je potřeba zajistit bezproblémový import i export kódu.

Výsledná platforma by měla být uživatelsky přívětivá, hlavně s ohledem na případný nižší věk uživatelů, zejména tedy žáků nižšího stupně základních škol. Ať už při využívání návrhářské části programu nebo vizualizace již zhotovených programů. Je samozřejmostí, že vzniklá simulace by měla být jasná a vykreslená vždy tak, aby z ní byly jasně vidět i případné chyby uživatelem napsaného kódu.

K implementaci grafického simulátoru bude použit framewok Qt. K jeho správnému využití bude potřeba prostudovat si vhodné materiály, a to buď ve formě odborné literary k němu sepsané, nebo přečtením dokumentace na stránkách výrobce. Zejména tedy podklady týkající se vykreslování grafiky s bližším zaměřením na pixelovou grafiku, ve které budou simulace vykreslovány.

Simulátor bude také obsahovat možnosti ukládat a načítat uživatelem vytvořené soubory. Je tedy potřeba vyhodnotit, v jakém formátu budou soubory ukládány. Pro učitele bude navíc přidána funkce exportu do pdf pro pozdější tisk.

V neposlední řadě po vytvoření grafického simulátoru dojde k jeho otestování.

Uživatelé testující hotový program by měli být složeni z různých věkových kategorií a měli by být rozděleni i dle jejich předchozích zkušeností s nástrojem Ozobot Evo.

Dle uvedených kriterií by se jednalo o žáky základních a středních škol. Nejnižší doporučený věk pro testování je třetí třída základních škol. Testovat necháme samozřejmě i učitele (pro zpětnou vazbu od vyučujících).

Všem, kteří budou testovat simulátor, budou kladeny dotazy ohledně jeho funkčnosti. Jedná se o náročnost pochopení operace s vyvinutým nástrojem, přesnost a jasnost jednotlivých simulací v porovnání s fyzickým provedením programu robotem a také zhodnocení zpětné kompatibility s programovacím prostředím vyvinutým vývojáři Ozobota Evo.

Kapitola 1

Analýza možností robota Ozobot Evo a nástrojů poskytnutých výrobcem

1.1 Technické parametry a informace

Nástroj Ozobot Evo je malý programovatelný robot o rozměrech 3 x 3 x 3.5 (výška, šířka a hloubka v cm). Podovozek obsahuje sedm senzorů pro rozpoznávání a čtení dráhy. Na přední a zadní straně má dva senzory přiblížení.

Robot obsahuje šest malých led světel - pět zepředu a jedno z vrchu. K nabíjení se používá konektor Micro-USB typu B. Nabízí propojení s mobilní aplikací od vývojářů pomocí Bluetooth.

1.2 Možnosti programování a použití

Možnosti nástroje ozobot Evo se dají rozdělit na tři hlavní skupiny, dle kterých ho lze programovat nebo používat. Dělíme je na čtení kódu předkreslené čáry speciálními fixami, ovládání ozobota pomocí aplikace od vývojářů a poslední: spouštění programu předem naprogramovaného na stránkách výrobce. Každý ze způsobů má své možnosti a využití, ale také omezení. Každý způsob je níže podrobně popsán.

1.2.1 Čtení kódu z předkreslené čáry

Prvním a nejlepším způsobem pro seznámení se s nástrojem Ozobot Evo je čtení čárového kódu. Čtení kódu je umožněno díky senzorům, kterými je robot vybaven. Rozezná černou, červenou, zelenou a modrou barvu. Důležité je kreslit čáry na bílé pozadí a používat fixy od výrobce, jejichž barvy ozobot bezpečně pozná. Lze využít i fix od jiných výrobců, ty však nezaručují správné rozeznání kódu a je tedy lepší se držet těch se značkou "ozobot".

Programování pomocí předkreslené čáry má striktní použití. Prostého následování

čáry lze dosáhnout např. nakreslením úsečky z bodu A do bodu B. Příklad úsečky na obrázku níže 1.1. Nezáleží, jakou barvu pro tento příklad použijeme. Obecně se však doporučuje používat k tomuto účelu vždy barvu černou, ostatní se totiž v určitých kombinacích používají k identifkaci rozličných akcí ozobota.

Obrázek 1.1: Ukázka ideální úsečky pro ozobota.

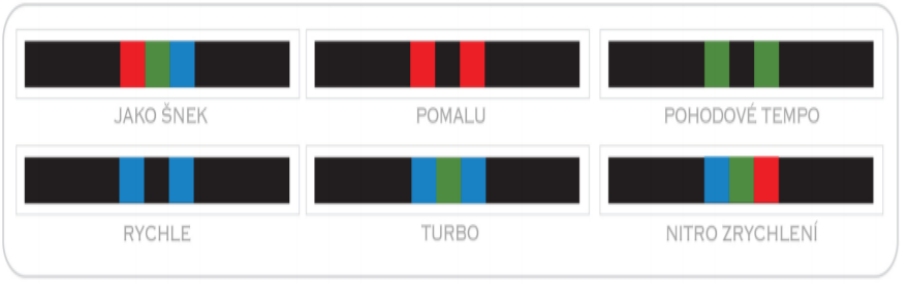
I přes velmi dobré senzory pro čtení se může v případě špatně nakreslené čáry stát, že robot neporozumí kódu (nerozezná ho) a neprovede ho. Špatnou čarou se myslí např. velmi tenká čára, nebo naopak velmi tlustá čára (širší jak samotný ozobot), přerušovanost plnosti barvy a nakreslení ostrých zatáček. V takovém případě ozobot většinou zůstane v půlce zatáčky, nebo vyjede z trasy a zastaví se. Doporučená a vyzkoušená šířka čáry je půl centimetru a při použití zatáček se doporučuje co nejvíce blížit k pravému úhlu.

Pro dosažení jiných operací, než jen následování čáry, se používají skupiny příkazů - barevných kódů. Každý z těchto příkazů, nenachází-li se na zakončení čáry, je buď čtyřmístný nebo třímístný. Na zakončení čáry se umisťují dvoumístné příkazy a většinou se týkají ukončení programu. I vícemístné kódy lze umístit na konec čáry, většinou se ale neprovedou správně. Jeden půlcentimertový čtverec představuje jednu část příkazu.

Skupiny příkazů jsou: rychlost, směr, časovač, super pohyby, výhra/východ a počitadla. Názvy skupin příkazů napovídají jejich funkčnost. Pro řízení čárovými kódy se dají využívat jen tyto výrobcem předdefinované skupiny příkazů, které ozobot umí rozeznat a provést. Je zde důležité dbát na správnost kreslení. Pokud přeženeme šířku či délku některého z částí(čtverců) příkazu, ozobot ho nerozezná a příkaz neprovede. Při přejetí jakékoliv barvy ozobot zasvítí horním světlem přečtenou barvu. To slouží i ke zpětné kontrole v případě, že příkaz není správně proveden.

První skupina (viz. obrázek 1.2) se, jak její název „rychlost“ napovídá, zabývá nastavováním rychlosti ozobota. Je zde na výběr ze šesti stupňů rychlostí, od nejpomalejší po nejrychlejší, jmenovitě: šnečí tempo, pomalu, procházka, rychle, turbo a nitro boost. Výchozí rychlostí ozobota je "procházka", tedy střední rychlost. Pro projevení vyšších rychlostí je potřeba dát ozobotovi prostor na rozjezd v podobě delší černé čáry po příkazu.

Je to kvůli tomu, aby stačil dosáhnout dané rychlosti. Dalé také není doporučeno dávat další příkazy hned po nejvyšších rychlostech. Např. pokud by po nastavení rychlosti nitro boost následoval s velmi malým odstupem libovolný další příkaz, ozobot by ho nemusel stačit včas zaregistrovat a přečíst správně.



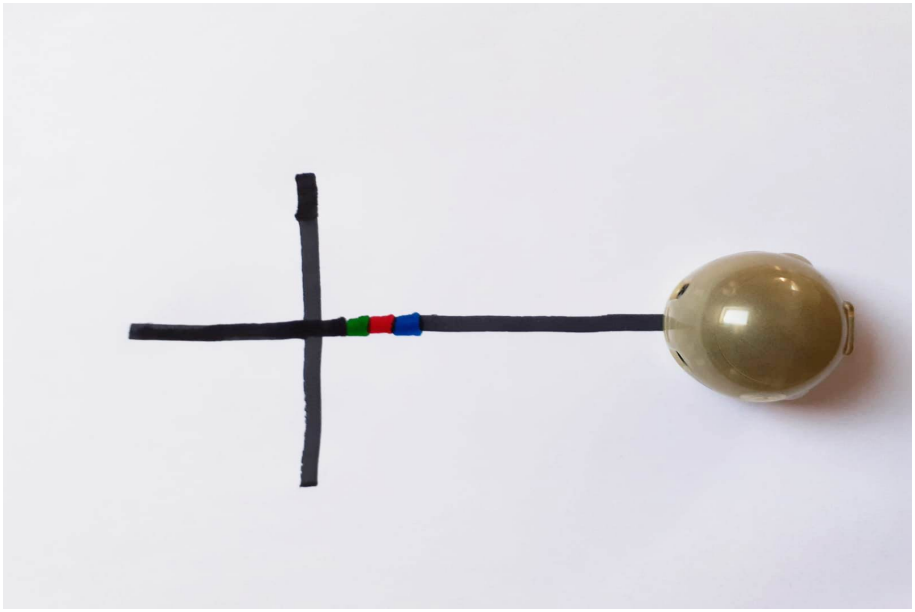
Obrázek 1.2: Přehled skupiny rychlost

Druhá skupina (viz. obrázek 1.3) je skupina směr. Ta se zabývá veškerým určováním směrů pohybů a otáčení. První trojice, jmenovitě jdi doleva, jdi rovně a jdi doprava, se využívá při rozhodování pohybů na křižovatkách (viz obrázek 1.4). Bez nich se ozobot rozhoduje náhodně. Těmito příkazy dosáhneme vyšší kontroly jeho pohybu.

Další trojice je velmi zajímavá k vyzkoušení. Opět jmenovitě: přeskoč čáru doleva, přeskoč čáru rovně a přeskoč čáru doprava. Nejedná se však doslova o žádné skákání, toho ozobot není schopen, nýbrž o přepojení na jinou čáru. To znamená, že po přečtení příkazu se ozobot v daném směru snaží ujet určitou vzdálenost po "prázdnu", dokud nenajde další čáru (viz. obrázek 1.5). V případě, že žádnou nenajde, se program ukončí.

Poslední dvojice příkazů z této skupiny se zabývá otáčením ozbota. První z nich se používá, když chceme robota otočit v půlce čáry zpět (tedy o 180 stupňů) a nechat ho, aby pokračoval v cestě. Druhý příkaz se umisťuje speciálně na konec čáry za stejným účelem. Lze tak při nakreslení příkazu na oba konce čáry dosáhnout nekonečného cyklu v podobě ozobota jezdícího z jednoho konce na druhý.

Obrázek 1.3: Přehled skupiny směr



Obrázek 1.4: Ukázka zahnutí doprava na křižovatce.

Obrázek 1.5: Ukázka přeskočení z jedné čáry na druhou.

Třetí skupina (viz. obrázek 1.6) s názvem časovače má jen tři příkazy. Prvním z nich je "zapnutí časovače (30s odpočítávání)". Po jeho přečtení začne ozobot odpočítávat od 30 do 0, během té doby ale pokračuje v následování čáry a čtení příkazů. Ve chvíli, kdy se časovač dostane na nulu ozobot v rychlém intervalu zasvítí světly a vypne se. Druhý příkaz slouží jednoduše k vypnutí časovače. Třetím příkazem je příkaz "pauza (3 sec.)". Ozobot po tomto příkazu pozastaví svou činnost na 3 vteřiny, než pokračuje dále.

Obrázek 1.6: Přehled skupiny časovače

Čtvrtá skupina (viz. obrázek 1.7) je mezi žáky základních škol nejoblíbenější - Cool pohyby. Ozobot po přečtení jednoho ze čtveřice těchto příkazů vykoná neobvyklý pohyb. Slouží hlavně k ukázce čeho je ozobot schopný. Po příkazu "tornádo"se začne ozobot na místě s narůstající rychlostí otáčet o 360 stupňů.

Vykonání příkazu "kličkuj"se dá popsat jako vykonání funkce sinus s takovým rozmezím, při kterém se ozobot moc neodchýlí od čáry a po dokončení se napojí zpět na následování. "Zatoč se"se od "tornádo"liší v tom, že otočka proběhne jednou a ve stejné rychlosti, rychlost tedy nenarůstá. Můžeme si všimnout, že tento příkaz je pozpátku "tornádo"a obráceně.

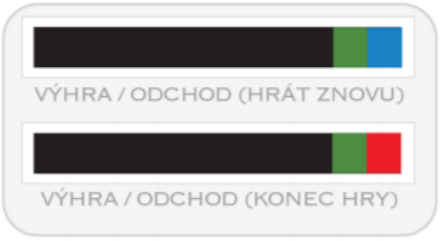
Poslední příkaz z této skupiny je "chůze po zpátku". Průběhem tohoto příkazu je otočení ozobota o 180 stupňů, poté jízda pozpátku po dobu jedné vteřiny a následné otočení o 180 stupňů ozobota zpět do původního směru.

Obrázek 1.7: Přehled skupiny cool pohyby

Předposlední skupina (viz. obrázek 1.8) má název výhra/výjezd a slouží k zobrazení dvou různých předprogramovaných animací. Příkaz "výhra/odchod (hrát znovu)"spouští animaci "úspěchu", ozobot po jejím dokončení následuje čáru (pokud není kód nakreslený na konci).

Příkaz "výhra/odchod (konec hry)"se liší jen v tom, že po dokončení animace "úspěchu"se ozobot vypne. Většinou se používá na konci čáry.

Poslední skupina počítadla(viz obrázek 1.9) nabízí čtyři různá počítadla, která můžeme použít. Při spuštění příkazu "Počítej křižovatky"se počítadlo nastaví na 0 a na každé projeté křižovatce ozobot k počitadlu přičte. Může to být křižovatka ve tvaru "T"nebo "+". Po projetí pěti křižovatek robot vykoná animaci "hotovo", zasvítí červeně a přestane následovat čáru.



Obrázek 1.8: Přehled skupiny výhra/východ

Počítadlo "Počítej zatáčky"se od prvního příkazu liší v tom, že počitadlo nám započítává jen zatáčky, tedy místa ve tvaru písmene "L". U obou kódů se započítávájí, jak přikazané směry, tak i ty náhodně vybrané ozobotem.

Třetím počítadlem je počítadlo barev. To započítáva změny barev, tedy pokud ozobot přejede např. z červené čáry na čáru modrou, k počítadlu se přičte. Barevné úseky, které jsou kratší jak 2 centimetry se nezapočítávají. Nepočítají se ani barevné přechody na černou barvu čáry. Po přečtení pěti barevných přechodů čar se robot zastaví.

Další počítadlo, počítadlo bodů, je ze všech nejjednodušší na vysvětlení i na vyzkoušení. Spolu se spouštěcím příkazem k němu patří ještě dva další příkazy "Bod +1"a "Bod -1". Ozobot při přejetí spouštěcího příkazu nastaví počítadlo na pět bodů. Následně při přečtení příkazu "Bod -1"odečte jeden bod. Ve chvíli, kdy je počítadlo na nule, vykoná ozobot animaci "hotovo"a skončí s následováním čáry.

Použitím příkazu "Bod +1"můžeme k počítadlu přičítat, ale nikdy stav nebude více jak pět bodů. Resetovat toto počítadlo se dá buď zapnutím a vypnutím ozobota, anebo novým přečtením spouštěcího příkazu.

Obrázek 1.9: Přehled skupiny počítadla

Toto jsou všechny skupiny příkazů kreslení čáry, které může ozobot vykonat.

1.2.2 Webový nástroj pro programování ozobota

Programovat ozobota můžeme také pomocí webové aplikace od výrobce na stránkách ozoblockly.com. Na této stránce je možno si vytvořit účet a uložit do paměti profilu až 12 uživatelem vytvořených programů. Jakékoliv další programy je nutno buď přeuložit přes starší nebo stáhnout do počítače/tabletu. Editor nabízí pět úrovní obtížnosti programování (viz. obrázek 1.10), kdy každá další úroveň přidává nové možnosti a příkazy, případně upravuje ty, co byly v předchozí úrovni.

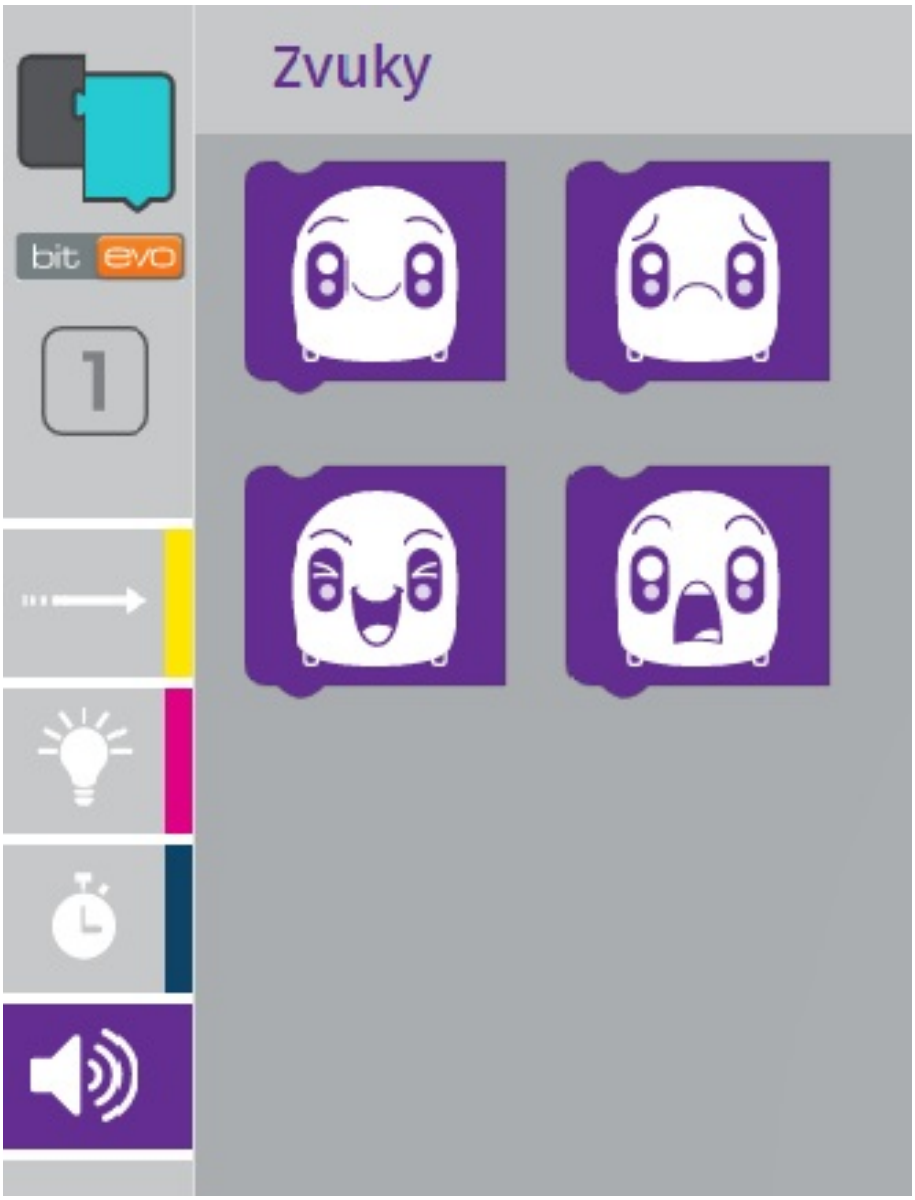
Jednotlivé programy se tvoří částmi ve formě dílků , které se k sobě připojují a ozobot je pak chronologicky vykonává.

Obrázek 1.10: Ukázka úrovní editoru

První úroveň

První úroveň s českým názvem „Nováček“ je pro nejmenší děti a příkazy jsou znázorněny jen obrázky bez textu (viz. obrázek 1.11). Obsahuje čtyři skupiny: Pohyb, světelné efekty, čekání a zvuky. S touto první úrovní lze robotovi nastavit barvu světla, pohyby vpřed nebo vzad ve vzdálenosti 1-10cm ve dvou různých rychlostech, čekání nebo-li pauza na 1-5 vteřin a jako poslední může ozobot přehrát čtveřici zvuků: radost, smutek, smích a překvapení. Tato úroveň se nejvíce hodí pro nejmenší, protože si rovnou asociují příkaz s tím, co ozobot provede. Používá se hlavně pro třetí třídu základních škol k rozvíjení logiky dítě.

Příkazy pohybu, pauzy a coolpohybů, které si žák vyzkoušel na papír si může naprogramovat pomocí této první úrovně. Pokud by však chtěl naprogramovat další příkazy, bude se muset posunout na vyšší úrovně. Ty už nejsou koncipované formou obrázků, ale textově s popiskem, co má ozobot udělat.



Obrázek 1.11: Ukázka skupiny zvuky

Druhá úroveň

Druhá úroveň se liší od té první tím, že přechází z obrázků na slovní popisky a přidává novou skupinu cykly. Tuto úroveň proberu více detailně, protože je stavebním kamenem pro ty nadcházející, které pak přidávají nové příkazy.

Na obrázku 1.12 je vidět ukázka příkazů ze skupiny pohyby. U příkazu „Pohyb“ lze navolit hned tři věci najednou. Směr, kterým ozobot pojede (opět vpřed nebo vzad). Vzdálenost, kterou ozobot ujede, kterou můžeme přesně navolit mezi 1-10 kroky, kde jeden krok je přibližně 1 cm. A jako poslední je výběr jedné ze čtyř rychlostí - pomalu, středně rychle, rychle a velmi rychle. Oproti první úrovni máme tohle vše v jednom příkazu.

Druhý příkaz „Otočit“, u kterého jsou k dispozici na výběr 3 dvojice možností. První: mírně vlevo a mírně vpravo, je otočení daným směrem o 45°. Druhá dvojice: vlevo a vpravo, je otočení daným směrem o 90°. Třetí dvojice: Otočení o 180° vlevo nebo vpravo.

Další je jeden z tzv. super pohybů „Cikcak“, u kterého máme možnost navolit jakou rychlostí ho ozobot provede. Rychlosti jsou stejné jako u příkazu „Pohyb“. Příkaz „Brusle“ je druhý ze série super pohybů. U něj ozobot provede pohyb podobný zabruslení. Lze si u něj opět navolit rychlost, ale také jestli má zabruslit dopředu, nebo dozadu.

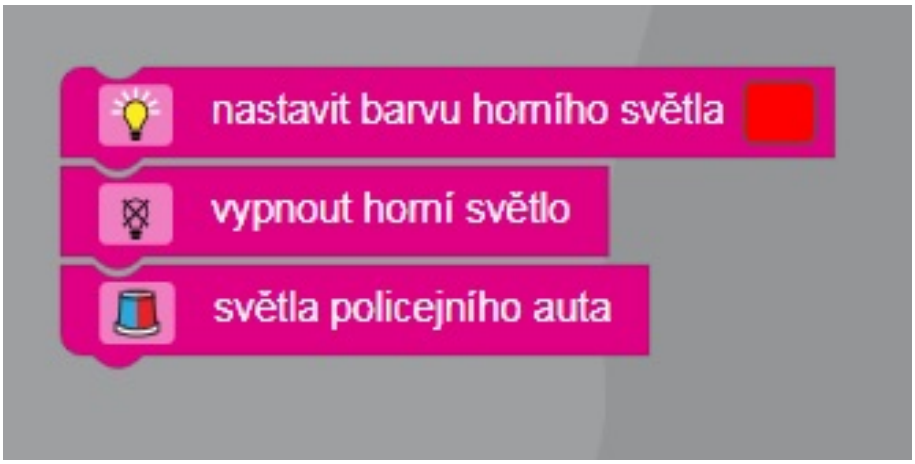
Poté na obrázku můžeme vidět „Otočit“, tento stejnojmenný příkaz jako ten již popsaný výše, se liší v tom, že jde o provedení otočky o 360°. A to buď doleva, nebo doprava.

Zbývající dva příkazy se týkají nařízení ozobotovi jezdit v kruhu, a to buď v kruhu o malém průměru, nebo o velkém. U obou můžeme nastavit jejich rychlost, jízdu kupředu či pozpátku, doleva nebo doprava a dobu trvání provádění příkazu. Doba je od jedné do pěti sekund.

Obrázek 1.12: Ukázka skupiny rychlost

Ze skupiny světla jsou vybrané na obrázku 1.13 tři principem jednoduché příkazy. První je nastavení horního světla ozobota na jednu z 12 barev: bílá, žlutá, fialová, tyrkysová, limetkově zelená, oranžová, světle fialová, světle modrá, zelená, červená, růžová a modrá. Ve vyšších úrovních lze namíchat barvu podle RGB. Druhý příkaz horní světlo ozobota, jak jeho název napovídá, vypne.

Třetí je jedna z vybraných přednastavených barevných sekvencí. V případě světel policejního auta ozobot střídavě zapíná modrou a červenou na předních světlech. Různé barevné sekvence pak spouší při zbylých příkazech: duha, semafor, disco, vánoční strom a ohňostroj.



Obrázek 1.13: Ukázka skupiny světla

Další sady příkazů Časování a Cykly jsou zasazené do jednoho obrázku - 1.14. V druhé úrovni je pro časování vyhrazen jen jeden příkaz, a to příkaz „počkat“, u kterého navolíme jednoduše od 1 do 10 vteřin dobu čekání před provedením dalšího příkazu.

Cykly představují opakování kódu, který je umístěn do jejich těla. V této urovni nám vývojáři nabízí buď cyklus, u kterého můžeme navolit počet opakování od 2 do 10, anebo nekonečný cyklus. V programátorském světě se tyto cykly dají přirovnat k použití cyklu while, což je vidět ještě více ve vyšších úrovních.

Ozobot Evo také umí zahrát/říci určitá slova a zvuky. Všechno, co ozobot řekne nebo zahraje, je v angličtině. Na obrázku 1.15 jsou vidět čtyři hlasové skupiny. První blok pojmenovaný „Přehrát“ přehrává zvuk jedné ze čtyř emocí: šťastný, smutný, překvapený a smích.

Příkaz „Říci barvu“ umožňuje vybrat uživateli jednu z 8 barev pro ozobota k zahrání. Jedná se o stejné barvy, které umí ozobot rozsvítit, vyjma barev jako "limetková zelená, světle modrá apod.". Dále umí zahrát jeden ze čtyř směrů - vpřed, vzad, vlevo a vpravo. A jako poslední v této úrovni umí ozobot říci číslo od jedné do deseti.

Obrázek 1.14: Ukázka skupiny časování a cykly

Obrázek 1.15: Ukázka skupiny zvuky

Třetí úroveň

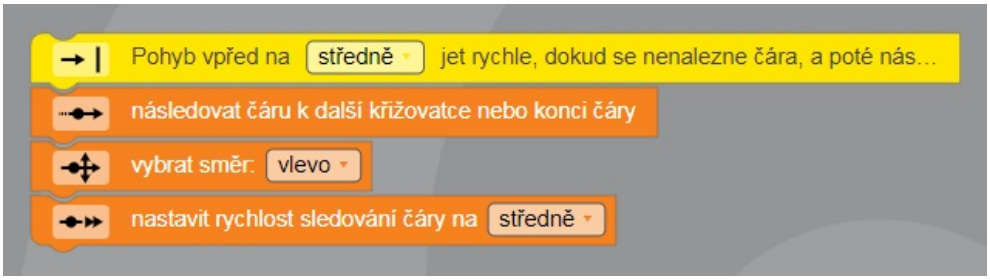
Třetí úroveň nám přidává tři nové skupiny příkazů a pár nových příkazů do stávajících skupin, vše ostatní již popsané zůstává stejné. Tři velké skupiny, které nám přibyly, jsou „Jízda po čáře“, „Senzory“ a „Logika“.

První si popíšeme skupinu „Senzory“, zbylé dvě skupiny spolu totiž úzce souvisí a budou popsány po sobě. Ozobot má ve předu a vzadu dva senzory. Vlevo vpředu, vlevo vzadu, vpravo vpředu a vpravu vzadu. V této úrovni je pro nás připraveno zjednodušené objekt vpředu, objekt vzadu a kombinace - objekt vpředu a vzadu.

Pokud ozobot senzory zaznamená nějaký objekt (např. ruku nebo papírovou kra-

bicí), tak provede uživatelem určenou akci viz. obrázek 1.16. Je také důležité, aby se v případě použití všech tří příkazů zadal kombinovaný příkaz jako první. Jinak by se dříve provedl jen příkaz s objektem před nebo vzad, přestože bychom dali něco naráz před i za ozobota. Je to kvůli chronologickému provádění příkazů.

Obrázek 1.16: Ukázka skupiny senzory



Obrázek 1.17: Ukázka skupiny jízda po čáře

V této úrovni můžeme ozobotovi nastavit i chování na předkreslené čáře. Na obrázku 1.17 je ukázka čtyř příkazů. První z nich je stejný jako příkaz „Skoč rovně“, který se dá nakreslit. Využívá se, když chceme přejít plynule na následování čáry během provádění příkazu.

Dalším blokem kódu dosáhneme, že ozobot bude následovat nakreslenou čáru buď do jejího konce, nebo do křižovatky. Blok kódu „Vybrat směr“ je opět stejný jako sada příkazů, která je popsána již v barevných kódech. Můžeme vybrat směr vlevo, vpravo, rovně a zpět.

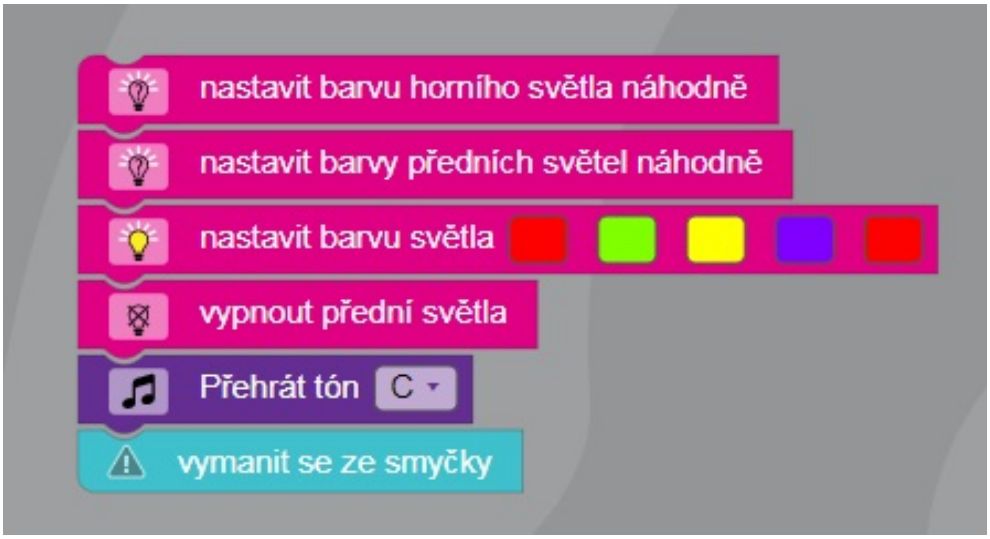
Poslední z nich vybírá rychlost, jakou ozobot po čáře pojede. Stejně jako u všech předchozích příkazů se dají navolit čtyři rychlosti.

Skupina logika se týká logického rozhodování ozobota na čáře. Sestává ze tří podmínek, které můžeme použít viz. obrázek 1.18. Můžeme pomocí nich zjistit, jakou má barvu křižovatka nebo konec čáry, jaká je barva povrchu a zda existuje cesta vpřed, vlevo nebo vpravo.

Protože se testují barvy, které můžeme nakreslit, jedná se jen o modrou, červenou, zelenou a černou. Testování barvy povrchu nabízí i barvu bílou - tato volba lze použít při kontrole, jestli ozobot na nějaké čáře je, anebo jestli je jen na papíře.

Obrázek 1.18: Ukázka skupiny logika

Všechny tři příkazy mají i rozšíření „Jinak“ nebo-li programátorské else. Např. pokud barva povrchu je zelená - jeď na další křižovatce rovně, jinak - zahni na další křižovatce doleva.



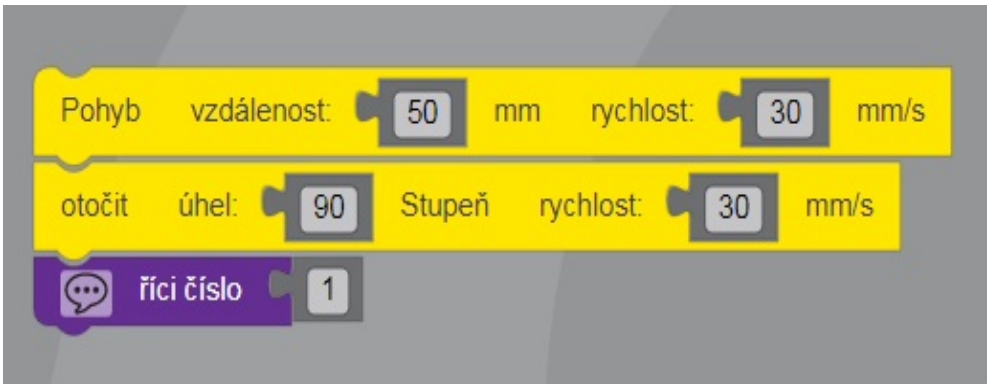
Obrázek 1.19: Ukázka přírůstků do skupin

Jako poslední nám přibylo v této úrovni pár příkazu již do existujících skupin viz. obrázek 1.19. Můžeme nyní nastavit barvu předních světel. Barvy se vybírají stejně

jako u horního světla. Zároveň barvu všech světel lze nastavit náhodně a také i najednou vypínat. Pro skupiny cykly přibyl příkaz „vymanit se ze smyčky“, který nám dokáže přerušit cyklus, i kdyby byl nekonečný. Je to programátorské break, využívá se však více až ve vyšší úrovni.

Čtvrtá úroveň

Čtvrtá úroveň přináší hned několik změn a přibližuje se mnohem více k programátorskému světu. Ozobot zvládá pracovat s číselnými proměnnými v rozsahu od -127 do 127. Hlavní změna této úrovně je, že všechna nastavení, která měla doposud omezení (např. kroky při příkazu „Pohyb“ byly omezeny maximálně na 10), lze nyní nahradit číslem z ozobotova rozsahu. Viz. obrázek 1.20.



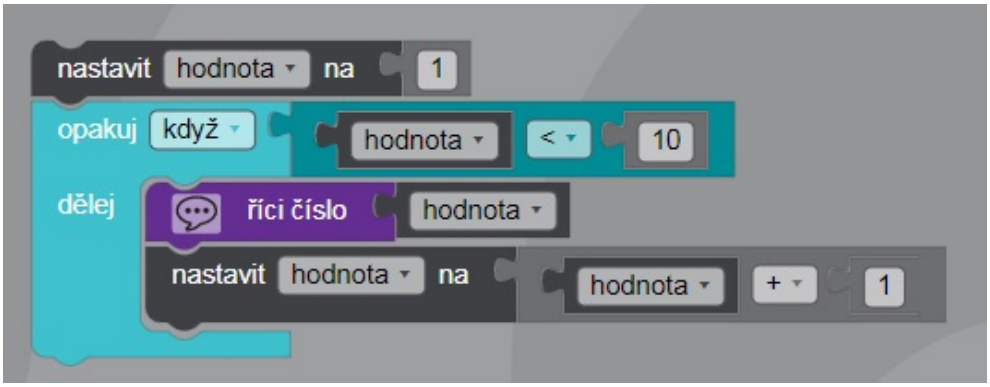
Obrázek 1.20: Ukázka změn 4. úrovně

U barev můžeme nastavit barvy podle RGB modelu, musíme však přepočítat na rozsah ozobota. Tato změna nastavení se projevuje u všech dosavadních příkazů.

Tato úroveň dále přináší uživatelské proměnné, použití matematiky a možnost vytvářet funkce. Do proměnných můžeme ukládat čísla nebo barvy. Ozobot neumí proměnné typu string(řetězec). Po vytvoření můžeme s proměnnou různě pracovat. Vzhledem k tomu, že ozobot má jen dva typy proměnných používáme převážně proměnné číselné. Ať už pro nastavení opakování cyklu, nebo nastavení různých parametrů příkazů. Zároveň lze měnit jejich hodnotu i v průběhu programu na základě nějaké akce.

K matematickým operacím s proměnnými využijeme novou skupinu „ matematika“. Kromě základních matematických operací(např. sčítání, dělení ...) nám nabízí i testování sudosti nebo lichosti proměnné, výběr náhodného čísla, získání absolutní hodnoty a získání zbytku po dělení.

Na obrázku 1.21 je ukázka vytvoření proměnné a zahrání čísla od 1-10 pomocí while cyklu.

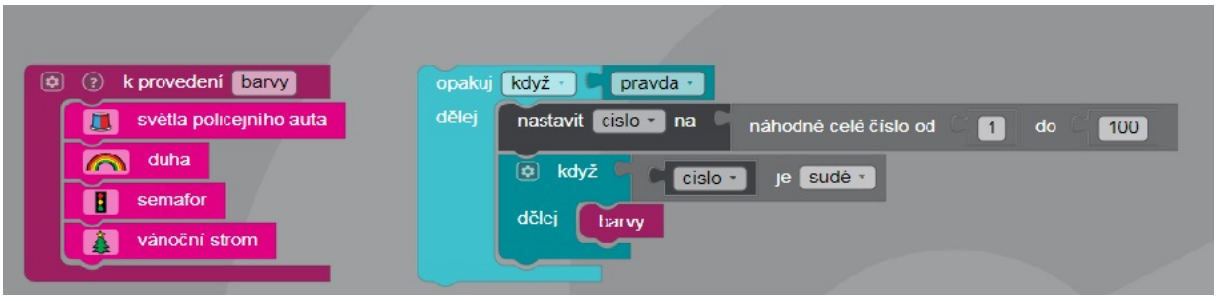


Obrázek 1.21: Ukázka vytvoření a použití proměnné

Na tomto příkladu můžeme vidět, že z obyčejného opakování 2 až 10 jsme mohli vytvořit příklad jednoduchého while cyklu s podmínkou na začátku. Oproti minulé úrovni prošly změnou i podmínky a je možné nyní testovat mnohem více, než jen barvu povrchu a čáry.

Další novou skupinou přidanou v této úrovni je skupina „ funkce“. Logika je stejná jako v programátorském světě, ať už se na ně díváme jako na lepší členění kódu či odstranění duplicity. Funkce máme dvojího typu: funkce s návratovou hodnotou a funkce bez návratové hodnoty. Ty bez návratové hodnoty mají účel vykonání určité série příkazů. Například pokud v kódu často používáme jednu stejnou světelnou sekvenci, nahradíme je tímto typem funkce.

Funkce s návratovou hodnotou se nám pak hodí v případě programů jako kalkulačka, kdy funkci předáme dvě hodnoty, které chceme např. sečíst a dostat výsledek, který pak můžeme uložit do proměnné a dále s ním podle potřeby pracovat.



Obrázek 1.22: Ukázka práce s funkcí bez výstupu

Pátá úroveň

Poslední úroveň nám přidává čtyři skupiny: Ukončení, Tlačítko, Pole a List. Tato úroveň je prozatím nejnovější, je však možné, že v budoucnosti přibudou ještě další.

Skupina „ Ukončení“ se dá velice jednoduše popsat jako série příkazů která určuje, co dělat po ukončení našeho programu. Možnosti ozobota jsou po ukončení provedení programu následující: vypnout, přepnout do režimu spánku nebo pokračování jízdy po nakreslené čáře. Zároveň máme k dispozici příkaz s názvem „ selhání“, po jehož spuštění provede ozobot krátkou animaci, při které zasvítí modře a červeně a následně se vypne.

Další skupina „ Tlačítko“ zahrnuje akce spojené se stiskem vypínacího/zapínacího tlačítka na ozobotovi. Pomocí příkazů této skupiny můžeme zachytávat počet stisknutím tlačítka a uložit je například i do proměnné. Můžeme to pak využít např.kdybychom dělali program, který má různé módy a na základě počtu stisku tlačítka bychom mězi nimi přepínali.

Zbývající dvě skupiny jsou přesně tím, čím jsou v programovacích jazycích. S úpravou pro ozobota můžeme mít tedy pole barev nebo pole čísel. To stejné platí pro kontejner List. S těmito novými skupinami se dají dělat už poměrně složité věci a posouvá se jimi používání ozobota na jinou úroveň.

1.2.3 Android aplikace Ozobot Evo

Android aplikace pro robota Ozobot Evo nabízí několik možností. Pro naše účely popisu ovládání a programování ozobota nám však stačí její dva hlavní aspekty - přímé ovládání ozobota a spouštění námi napsaných programů.

Obrázek 1.23: Ukázka vstupu do aplikace

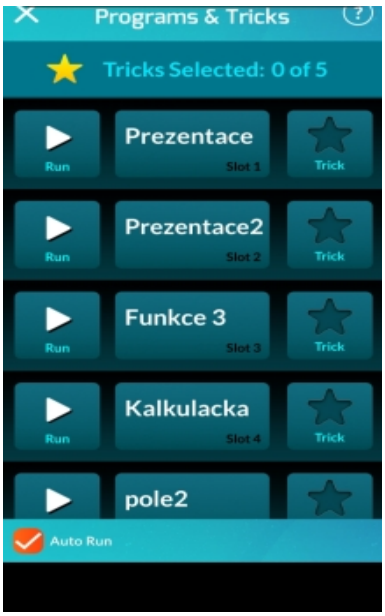
Aplikace se připojuje k ozobotovi přes bluetooth a v novější verzi aplikace vyžaduje i zapnutí polohy, bez kterého se k ozobotovi nepřipojí. Zároveň je mnohem náročnější a některé starší dotykové mobily ji nezvládají spustit. Tablety s aplikací většinou problémy nemají.

Do aplikace se pak přihlašujeme stejným uživatelským účtem, jako se přihlašujeme do webové aplikace a máme tedy přístup ke svým programům.

První funkce umožňuje ovládat ozobota v podstatě jako auto na ovládání. Řídíme jeho směr a měníme mu rychlost. Zároveň je na výběr několik přednastavených světelných módů a možnost nastavení odstínů všech jednotlivých světel ozobota samostatně. Slouží to k různým hrám, obzvlášť menším dětem se toto ovládání líbí.

Druhá a velice užitečná funkce této aplikace je spouštění uložených programů. Bez této funkce bychom museli každý program manuálně nahrávat přiložením ozobota k monitoru na nahrávací místo, jak je vidět ve webové aplikaci. Můžeme tedy spouštět po sobě několik nezávislých programů bez potřeby nahrávat jednotlivé programy pokaždé znovu.

Obrázek 1.24: Ukázka ovládacího rozhraní ozobota



Obrázek 1.25: Ukázka rozhraní pro spuštění programů

Kapitola 2

Navržení nástroje pro simulaci chování Ozobot Evo využívající framework Qt

2.1 Uživatelé nástroje

Pro vytvoření adekvátního nástroje je zapotřebí zhodnotit, kdo jej bude užívat. Jak již bylo zmíněno, tento nástroj bude využíván zejména žáky základních škol a jejich vyučujícími. Následně pak na technických kroužcích, které se zaměřují také spíše na žáky základních škol. Simulátor má využití i pro studenty středních škol, ale vzhledem k tomu, že ho budou využívat i mladší žáci, je zapotřebí ho přizpůsobit právě jim.

Hlavní cílová skupina jsou tedy žáci základních škol. Výsledný design nástroje by kvůli tomu měl být v rámci možností co nejjednoduší a nejpřehlednější. Musí obsahovat jednoduché prvky ovládání pro studenta a zároveň funkce pro učitele. Mezi ty patří hlavně PDF export. Důraz je kladen na jasně provedenou vizualizaci simulace. Cílem je dosáhnout co nejpřesnějšího výsledku v porovnání s fyzicky provedeným programem.

2.2 Volba implementačního nástroje

Simulátor Ozobot Evo je implementovaný jako desktopová aplikace, zcela napsaná ve frameworku Qt 5. Implementačních nástrojů je na výběr vždy několik. Jedním z hlavních kritérií pro výběr bylo, zdali chceme desktopovou aplikaci, nebo aplikaci webovou.

Jak již bylo zmíněno, simulátor je napsán jako desktopová aplikace, důvodů této volby bylo několik. Prvním je odstranění potřeby internetu při používání aplikace. Druhý důvod je jedna rychlá instalace. Ve školách lze nainstalovat jednoduše jednou na všechny počítače v příslušných učebnách.

Třetí důvod je využití frameworku Qt 5, se kterým mám osobní zkušenost. Je vyučován na naší fakultě (Fakulta jaderná a fyzikálně inženýrská), a jsem s ním tedy dobře obeznámen.

2.3 První grafický návrh rozhraní

Simulátor je rozdělen na dvě části. První část je jednoduchá uvítací obrazovka viz. obrázek 2.1 pro vstup do programu. Zde byl záměr uvést uživatele do programu, avšak zároveň uvítací obrazovku nepřeplňovat různými možnostmi a nastaveními. Obsahuje tedy jen jedno tlačítko, které nás přesune dále do simulátoru.

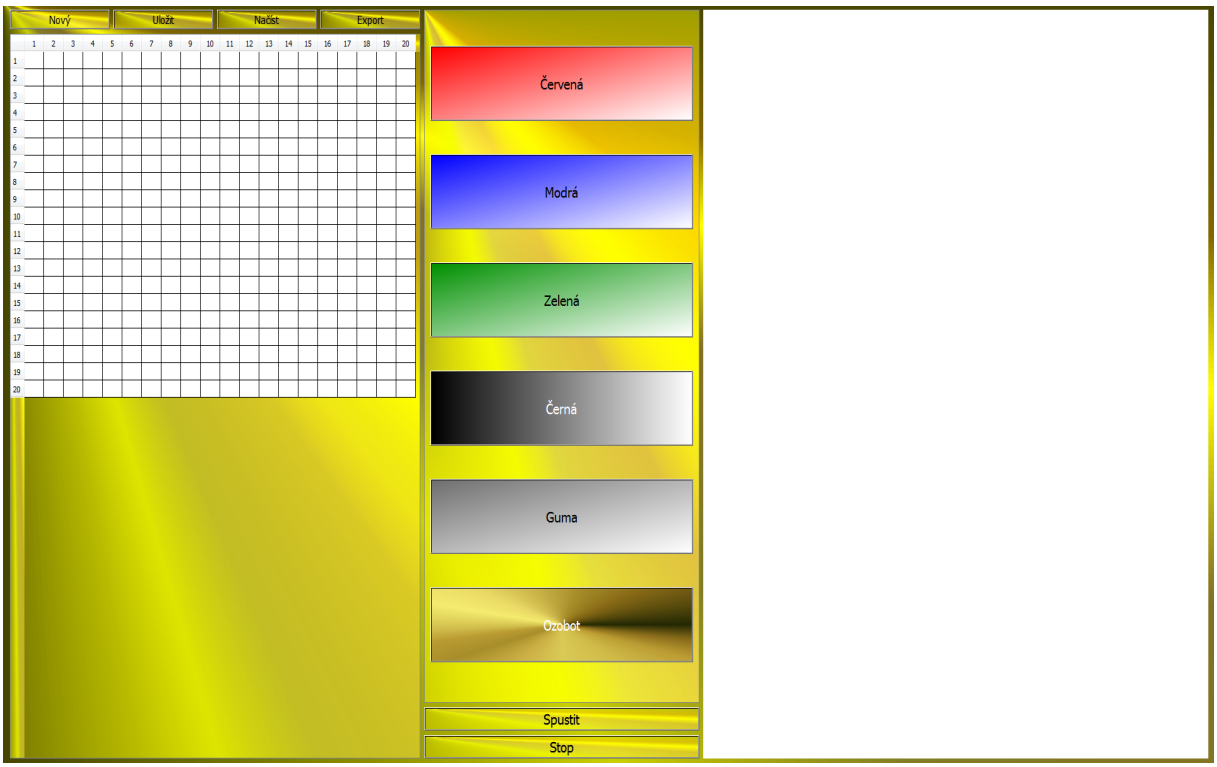
Pro barvu první verze rozhraní byl vybrán přechod ze zlaté do žluté barvy. Myšlenkou bylo zaujmout zejména menší studenty. Obrázek obsahuje ukázku kreslených kódů a fix od výrobce ozobota.

Obrázek 2.1: Ukázka vstupu do aplikace

Po kliknutí na tlačítko „Spustit“ se přesuneme do hlavního rozhraní simulátoru viz. obrázek 2.2. Je zde zvoleno stejné barevné schéma jako v úvodní obrazovce. Hlavní rozhraní simulátoru je rozděleno na tři části. Levá část je tvořena jednoduchým menu a tabulkou 20x20. Menu obsahuje tlačítka „Nový“, „Uložit“, „Načíst“ a „Export“. Tabulka je určena k navrhování tratě pro ozobota.

Prostřední část je tvořena nástroji tlačítek s barvami, které ozobot umí přečíst při čtení čar - červenou, modrou, zelenou a černou. Následně máme i tlačítko „Guma“ pro mazání nakreslené čáry. Zlaté tlačítko „Ozobot“ slouží k umístění ozobota na

výchozí pozici, ze které začne vykonávat program. Poslední částí nástrojů jsou spouštěcí a zastavovací tlačítko, které slouží ke spuštění a zastavení simulace. Barevné kódy a ozobot se umisťují do mřížky v levé části obrazovky.



Obrázek 2.2: První vzhled simulátoru

Prává část je tvořena plátnem, do kterého se promítá tabulka z levé části. Zároveň zde probíhá výsledná simulace, která je vykreslována pixelovou grafikou. Tlačítka z menu ukládají a načítají CSV výstup tabulky. Čtrvté tlačítko „Export“ ukládá do pdf formátu výstup plátna.

Při návrhu rozhraní zde byla původně myšlenka oddělit pravou část s plátnem do samostaného okna. Od toho se vzhledem k přehlednosti a následné implementaci upustilo. Vznikl tak jednotný návrh sestávající jen ze dvou oken: úvodního okna a hlavního okna.

Návrh hlavního okna je co nejvíce zjednodušen pro snadné využívání i těmi nejmenšími studenty. Je zde snaha vyhnout se dlouhému rozbalovacímu menu a místo toho zajistit hlavní funkce viditelně a přehledně přímo k dispozici.

Kapitola 3

Implementace nástroje

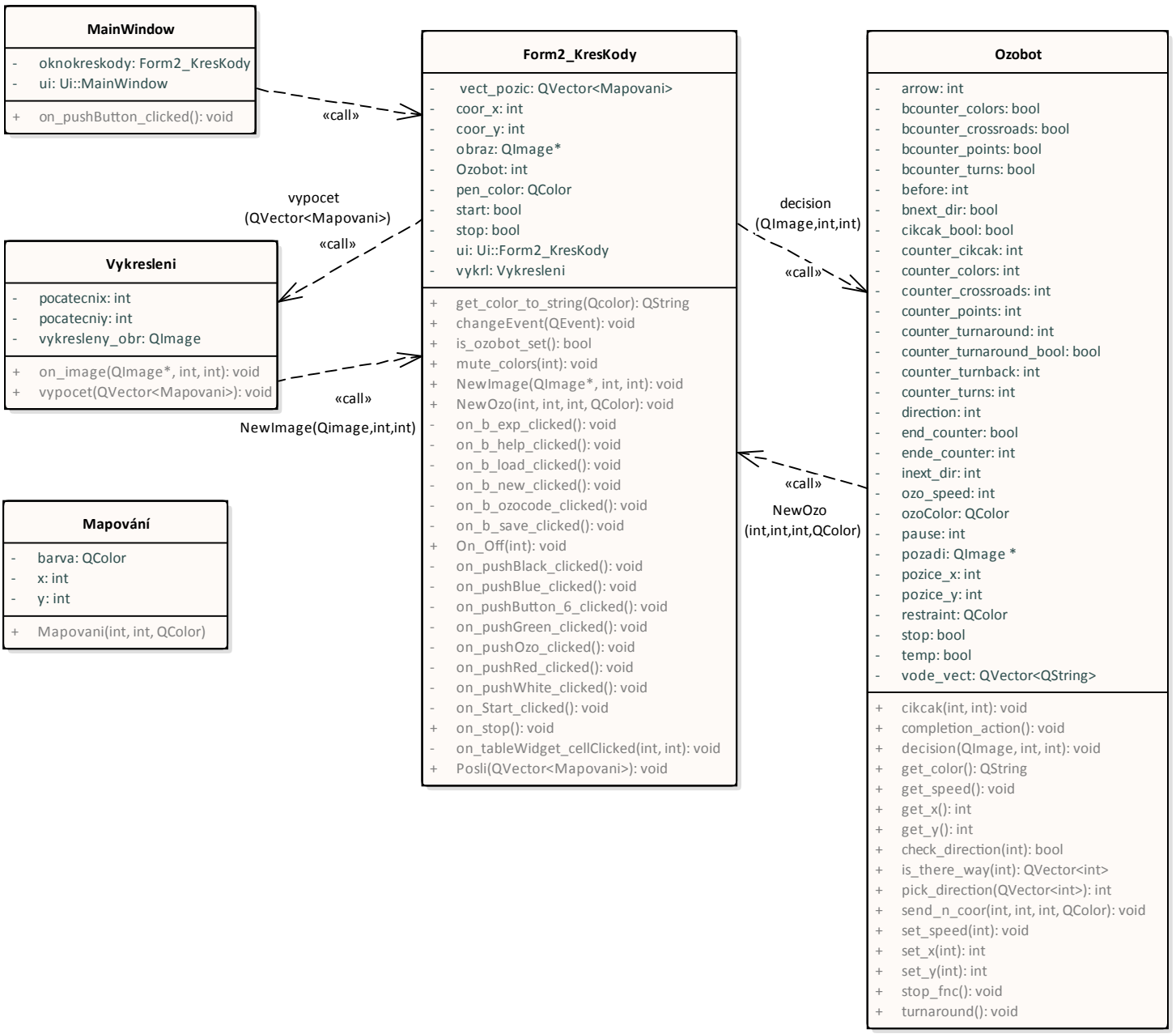
3.1 Popis tříd

Detailní class diagram simulátoru se všemi proměnnými a metodami je vidět na obrázku 3.1. Níže budou popsány jednotlivé třídy s jejich funkčností. Celkově tedy máme pět tříd: MainWindow, Mapovani, Form2_Kreskody, Vykresleni a Ozobot.

Třídy MainWindow a Form2_Kreskody obsluhují grafické rozhraní, které bylo ukázáno v předchozí kapitole. Třída „Form2_Kreskody“ dále spouší na samostatných vláknech metody tříd:

1. Vykresleni
   (a) získává se plátno simulace k zobrazení
2. Ozobot
   (a) získávají se data o pohybu ozobota

Obrázek 3.1: Class diagram



3.1.1 Třída MainWindow

Tato třída je hlavní třídou implementující funkcionalitu úvodní obrazovky. Graficky je reprezentována spouštěcí obrazovkou simulátoru viz. obrázek A.1. Účelem této třídy je zobrazení úvodního okna a přesměrování do těla simulátoru.

Obsahuje jednu privátní metodu „void on_pushButton_clicked()“. Tato metoda při stisknutí tlačítka „Spustit“ otevře okno těla simulátoru a uzavře spouštěcí obrazovku viz. 3.1. Tělo simulátoru je reprezentováno třídou Form2_Kreskody.

```
void MainWindow::on_pushButton_clicked(){
    this->close();
    oknokreskody = new Form2_KresKody();
    oknokreskody->show();
}
```

Ukázka kódu 3.1: Funkce void on_pushButton_clicked()

3.1.2 Třída Mapování

Tato třída slouží jako datový typ pro kontejner typu QVector, který se používá při načítání dat z objektu TableWidget. Mapovani obsahuje tři proměnné a konstruktor bez parametrů. Při zvolení této třídy jako datového typu lze uložit pozici x,y a barvu buňky. Používá se ve třídě Form2_Kreskody.

```
class Mapovani
{
public:
    int x;
    int y;
    QColor barva;
    Mapovani();
};
```

Ukázka kódu 3.2: Třída mapování

3.1.3 Třída Form2_Kreskody

Tato třída je graficky reprezentována jako tělo simulátoru viz. obrázek A.2. Celkový soupis všech proměnných a metod je vidět v class diagramu. Pomocí metod této třídy získáváme požadavky uživatele (většinou akce tlačítek), které dále zpracováváme.

Při zpracování požadavků volá i metody dalších dvou tříd: Vykresleni, Ozobot. Při prvním spuštění je potřeba inicializovat objekt TableWidget a nastavit barvu pozadí buňek na bílou.

Je to z důvodu pozdějšího načítání všech barev pozadí buněk TableWidgetu, kdy prádná „nebarevná“ způsobí chybu. Dále se nastaví výchozí barva pera na černou, kterou se bude vybarvovat pozadí buňek při kliknutí myší viz. obrázek 3.3.

```
ui->setupUi(this);
    pen_color=Qt::black;
    for(int i=0;i<ui->tableWidget->rowCount();i++)
    {
        for(int j=0;j<ui->tableWidget->columnCount();j++)
        {
        ui->tableWidget->setItem(i, j, new QTableWidgetItem);
        ui->tableWidget->item(i, j)->
                                    setBackground(Qt::white);
        }
    }
```

Ukázka kódu 3.3: Inicializace tabulky

Pozdější změna barvy pera probíhá pomocí kliknutí na jedno z barevných tlačítek, které mění hodnotu proměnné „pen_color“. Příklad je uveden na obrázku 3.4, ke je změna barvy pera na červenou.

Zcela obdobným způsobem jsou napsané i ostatní metody pro měnění barev, přičemž akce tlačítka „Guma“ mění barvu pera na bílou. Umístění ozobota se v TableWidgetu označuje žlutou barvou.

```
void Form2_KresKody::on_pushRed_clicked()
{
   pen_color=Qt::red;
}
```

Ukázka kódu 3.4: Změna barvy tlačítka

Velmi důležitá metoda je „ Form2_KresKody::on_tableWidget_cellClicked“, která zpracovává akci kliknutí na buňku v TableWidgetu. Tato metoda nastaví barvu pozadí buňky na aktuálně vybranou barvu pera. Dále projde všechny buňky TableWidgetu (20x20) a ukládá je do kontejneru QVector typu třídy„Mapovani“. Umožňuje nám tak uložit pozici x,y a barvu pozadí každé buňky. Vznikne nám tak Qvector o 400 položkách.

Poté metoda spustí na jiném vlákně výpočetní funkci ze třídy „Vykresleni“ a předá ji jako parametr QVector s daty o buňkách. Vykreslovací funkce, jmenovitě „vypocet“, vrátí QImage odpovídající datům z TableWidgetu spolu s výchozími

body x a y pro ozobota. S těmi dále pracuje metoda této třídy „Form2_KresKody::NewImage“ viz. 3.6.

```
connect(&vykrl,&Vykresleni::on_image,
this,&Form2_KresKody::NewImage);
QFuture<void> ini = QtConcurrent::run
                    (&this->vykrl,&Vykresleni::vypocet,
                    QVector<Mapovani>(vect_pozic));
```

Ukázka kódu 3.5: Propojení Form2_KresKody a Vykresleni

```
void Form2_KresKody::NewImage(QImage* img, int x, int y)
{
    QPixmap pm(QPixmap::fromImage(*img));
    obraz=img;
    coor_x=x;
    coor_y=y;
    QPainter p(&pm);
    p.setRenderHint(QPainter::Antialiasing, true);
    QPen pen(Qt::darkBlue, 2);
    p.setPen(pen);
    QBrush brush(Qt::white);
    p.setBrush(brush);
    p.drawEllipse(x, y, 40, 40);
    ui->label->setPixmap(pm);
}
```

Ukázka kódu 3.6: Metoda Form2_KresKody::NewImage

Pro zobrazení obsahu tabulky TableWidget se v metodě použije QLabel umístěný v grafickém rozhraní, operující jako plátno. Z QImage získané ze třídy Vykreslení se vytvoří QPixmap, která se pomocí příkazu „ui->label->setpixmap“ předá QLabelu, který ji zobrazí.

Do pixmapy se dále pomocí QPainteru přikreslí elipsa, znázorňující ozobota. Díky tomu, že vygenerování QImage běží na samostatném vlákně, se změny projevují na QLabelu téměř okamžitě a zároveň nezpomalují žádné jiné akce. Prostředí simulátoru dál plně reaguje bez zpoždění.

Další je metoda, která zahajuje spuštění animace: Form2_KresKody::on_Start_clicked. Ta se spustí kliknutím na tlačítko „Spustit“ v grafickém rozhraní. Pro spuštění simulace musí platit dvě věci: TableWidget není prázdný (existuje čára/kód) a ozobot je umístěný v tabulce. Zdali je ozobot umístěný, kontroluje metoda „Form2_KresKody::is_ozobot_set“.

V případě, že jedna z těchto podmínek není splněna, se uživateli objeví na obrazovce QMessage box s upozorněním. Dokud nejsou obě podmínky splněny, uživatele nepustí dál. Podoba obou QMessageboxů je vidět v následující podkapitole 3.3.

Ve chvíli, kdy jsou obě podmínky splněny začíná animace. Z implementačního hlediska to znamená výpočet měnění pozice elipsy znázorňující ozobota. Tento výpočet opět běží na samostném vlákně, aby nezamrzalo celé rozhraní a bylo možné ho nadále používat. Pro výpočet pozicí a čtení akcí ozobota se používají metody obsažené ve třídě „Ozobot“. Budou tedy popsané níže v podkapitole dané třídy.

Pro zamezení klikání do tabulky TableWidget a tlačítek menu za běhu animace se pomocí metody „Form2_KresKody::On_Off“ nastavuje TableWidget a téměř všechna tlačítka na disabled. Výjimkou jsou pouze tlačítka „Nápověda“, „Ozokódy“ a „Zastavit“. Při stisku tlačítka „Zastavit“ na grafickém rozhraní se všechno opět nastaví na enabled.

Na obrázku 3.7, na kterém je ukázáno propojení tříd můžeme vidět, že data předáváná metodě „Ozobot::decision“ jsou vypočítaný QImage a coordinace x,y ozobota. Tato metoda nám vrátí změněnou pozici ozobota, směr vykreslované šipky (intová proměnná) určující jeho směr a barvu, kterou ozobot svítí. S těmito získanými daty pracuje metoda „Form2_KresKody::NewOzo“.

```
On_Off(1);
connect(&ozo,&Ozobot::send_n_coor,
    this,&Form2_KresKody::NewOzo);
connect(this,&Form2_KresKody::on_stop,&ozo,
    &Ozobot::stop_fnc);
QFuture<void> run = QtConcurrent::run
    (&this->ozo,&Ozobot::decision,obraz,coor_x,coor_y);
```

Ukázka kódu 3.7: Propojení tříd Form2_KresKody a Ozobot

Postup je víceméně stejný jako u metody „Form2_KresKody::New_Img“. Vytvoří se QPixmap z QImage (vypočítaného ve třídě Vykresleni) a poté se přikreslí pomocí QPainteru elipsa znázorňující ozobota.

Rozdíl je v tom, že elipsa nyní mění i barvu podle povrchu plátna a vykresluje se uvnitř také šipka ukazující směr ozobota. Vše se předá QLabelu k vykreslení na plátno.

Chování ozobota je tedy zcela určeno metodou „Ozobot::decision“, která běží na samostném vlákně. Metoda „Form2_KresKody::New_Ozo“ slouží pouze k překreslení plátna pomocí získaných dat. Animace je ve stavu běhu do doby, než je stisknuto tlačítko „Zastavit“ a spuštěna k němu příslušná metoda.

Zbývající metody této třídy se starají o obsluhu tlačítek menu. Jmenovitě se nám jedná o akce tlačítek: Nový, Uložit, Načíst, Export, Ozokódy a Nápověda. Nejjednodušší z hlediska implementace byla operace tlačítka „Nový“, kdy se znovu inicializuje TableWidget do původního stavu a smaže se Qpixmapa QLabelu. S výjimkou řádku se smazáním Qpixmapy se jedná o stejný kód jako na na obrázku **??**.

Tělo metody tlačítka „Uložit“ je ukázáno na obrázku 3.8. Do souboru o formátu .csv se ukládají buňky TableWidgetu, respektive jejich barva pozadí. Pro zjednodušení ukládání a poté zpětného načítání se uloží barvy buňek v podobě zástupných

číslic.

Tento převod zprostředkuje metoda „Form2_KresKody::get_color_to_string“ během procházení aktuálních buňek TableWidgetu.

Program nabídne běžný ukládací dialog, kde si uživatel může vybrat umístění souboru. Ve výchozím stavu je to složka csv/ v kořenovém adresáři programu. Po uložení dostaneme csv soubor s tabulkou 20x20 naplněnou číslicemi 0 až 5, kde v tomto pořadí reprezentují barvy: černá, bílá, modrá, zelená, červená a žlutá (umístění ozobota).

Metoda tlačítka „Načíst“ provádí přesně opačný process. Při stisknutí uživateli zobrazí běžný načítací dialog, který používá k prvnímu zobrazení stejnou výchozí složku jako ukládací metoda.

Při čtení uloženého souboru si do kontejneru QVector typu QString načteme postupně všech dvacet řádků tabulky viz. obrázek 3.9. Dále pak ve dvou cyklech typu for, procházíme jednotlivé řádky znak po znaku a převádíme je z číslic na příslušnou barvu. Do QVectoru typu „Mapovani“ si pak ukládáme pozice x,y a načtenou barvu.

Výsledný QVector předáme dále metodě ve třídě „Vykresleni“, která nám vypočte nový QImage. Ten opět předá metodě „Form2_KresKody::New_Img“, která nám z něj vytvoří QPixmap a předá ji QLabelu.

Princip je tedy stejný jako u metody „Form2_KresKody::on_tableWidget_cellClicked“ s tím rozdílem, že nezpracováváme přímo buňky TableWidgetu, ale načítáme si je z tabulky v csv souboru.

```
void Form2_KresKody::on_b_save_clicked()
{
    QString filename = QFileDialog::getSaveFileName
     (this,'Uloz CSV soubor' , "./csv/ozo_kod_1.csv",
              "CSV soubory (.csv);Vsechny soubory (*)", 0, 0);
    QFile data(filename);
    if(data.open(QFile::WriteOnly |QFile::Truncate))
    {
        QTextStream output(&data);
        for(int i=0;i<ui->tableWidget->rowCount();i++)
        {
            for(int j=0;j<ui->tableWidget->columnCount();j++)
            {
                QString text=get_color_to_string
            (ui->tableWidget->item(j,i)->backgroundColor());
                output << text;
            }
            output << "\n";
        }
    }
}
```

Ukázka kódu 3.8: Ukládání do csv souboru

```
QTextStream in(&inputfile);
QVector<QString> line;
QString text;
vect_pozic.clear();
Mapovani mapa;
while(!in.atEnd())
{
  in >> text;
  line.append(text);
}
```

Ukázka kódu 3.9: Načítání csv souboru do QVectoru

Metoda „on_b_exp_clicked“ se provádí při stiknutí tlačítka „Export“ a zprostředkovává export plátna do formátu PDF. Před vytvářením pdf souboru se zkontroluje, zdali nemáme prázdné plátno. V případě prázdného plátna se zobrazí uživateli vyskakovací okno s upozorněním.

Poté si vytvoříme z plátna dočasný obrázek formátu PNG. Pomocí QPdfWriteru si připravíme stránku ve formátu A4, do které pomocí QPainteru vykreslíme připravený obrázek plátna. Pak si pomocí běžného ukládací dialogu vybereme místo, kam chceme pdf soubor uložit. Výchozím místem je složka pdf/ v kořenovém adresáři aplikace. Tělo metody je ukázáno na obrázku 3.10.

```
QFile file("./pdf/docasny_soubor.PNG");
file.open(QIODevice::WriteOnly);
obraz->save(&file, "PNG");
QString filename = QFileDialog::getSaveFileName
(this, "Uloz PDF soubor", "./pdf/ozo_kod_1.pdf",
"PDF soubory (.pdf);Vsechny soubory (*)", 0, 0);
QPdfWriter pdfWriter(filename);
pdfWriter.setPageSize(QPageSize(QPageSize::A4));
QPainter painter(&pdfWriter);
painter.setFont(QFont("Times", 10));
painter.drawText(200,200,
 "Vygenerovano simulatorem Ozobot Evo v.1.0.");
QImage img("./pdf/docasny_soubor.PNG");
painter.drawImage(QRect
(0,1000,pdfWriter.logicalDpiX()*8.0,
pdfWriter.logicalDpiY()*9.5), img);
file.remove();
```

Ukázka kódu 3.10: Metoda pro export

Tlačítko „Ozokódy“ a „Nápověda“ se z hlediska implementace liší jen zobrazovaným obsahem. Obě metody zobrazí v novém okně připravený obrázek a kódem jsou

téměř identické. Princip je takový, že si vytvoříme okno s QLabelem, kterému jako QPixmapu předáme uložený PNG obrázek. A to (buď) obrázek s ozokódy, nebo obrázek s nápovědou k správnému použití simulátoru.

```
void Form2_KresKody::on_b_ozocode_clicked()
{
    QLabel * label_img = new QLabel (this);
    label_img->setWindowFlags(Qt::Window);
    QImage img("./obr/ozokody.PNG");
    QPixmap pm(QPixmap::fromImage(img));
    label_img->setGeometry(QStyle::alignedRect
                           (Qt::LeftToRight,
    Qt::AlignCenter, QSize(696,520),
        qApp->desktop()->availableGeometry()));
    label_img->show();
}
```

Ukázka kódu 3.11: Metoda pro zobrazení ozokódů

Poslední dvě zbývající metody této třídy jsou: Form2_KresKody::changeEvent a Form2_KresKody::mute_colors. Jedná se o pomocné metody operující s grafikou rozhraní. První z nich ošetřuje změny velikosti nadpisu v grafickém rozhraní při maximalizaci a minimalizaci okna simulátoru.

Druhá je volána již dříve zmíněnou metodou „On_off“ a slouží k zešedivění tlačítek pro výběr barvy při spuštění animace. Důvodem je, že i při nastavení atributu disabled na true se pozadí tlačítek nezatmaví - nezešedne. Metoda nám tuto funkci zajišťuje pro větší přehlednost uživatelského rozhraní.

3.1.4 Třída Vykresleni

Účelem této třídy je vytvoření QImage z QVectoru typu Mapovani. Tento QVector se získává metodou třídy „Form2_KresKody“ a obsahuje x,y a barvu všech buňek TableWidgetu (20x20) z grafického rozhraní.

Třída obsahuje pouze jednu metodu a jeden signál. Metoda „Vykresleni::vypocet“ slouží k vygenerování QImage. Nejdřív se vytvoří prázdný Qimage 800x800 pixelů, což je velikost pozadí plátna (QLabelu). Pro zakreslování do QImage se používají čtyři for cykly do sebe zanořené. První dva slouží k pohybu po ose x a y, kde prochází celý obrázek. Zbylé dva slouží k vybarvení místa určitou barvou.

Abychom dosáhli přesného zobrazení tabulky 20x20 v obrázku o rozměrech 800x800, kreslíme jednotlivé buňky do obrázku jako čtverec o rozměrech 40x40. Zde na obrázku 3.12 lze vidět ukázku čtyř cyklů. V Qvectoru je obsažený pro nás nejdůležitejší údaj - barva.

Barvu získáváme tak, že procházíme všech 400 prvků QVectoru, kdy každých 20 prvků představuje jeden řádek. Každý jeden prvek obsahuje barvu buňky a ta se použije při příkazu „setPixel“. Tato část metody je ukázána na obrázku 3.13.

Po vykonání všech čtyř for cyklů získáme překreslený QImage a výchozí pozici x,y elipsy znázorňující ozobota. Vše pak předáme metodě „Form2_KresKody::New_Img“, která si z QImage vytvoří QPixmapu a nastaví ji QLabelu. Předání probíhá emitováním signálu „on_image(vykresleny_obr,pocatecnix,pocatecniy)“. Spojení mezi těmito dvěma třídami je popsáno v podkapitole 3.1.3.

```
QRgb barvas =qRgb(255,0,255);
    int c=0;
    vykresleny_obr=new QImage(800,800,QImage::Format_RGB32);
    for(int a=0;a<800;a=a+40)
    {
        for(int b=0;b<800;b=b+40)
        {
            for(int i=0;i<40;i++)
            {
                for(int j=0;j<40;j++)
                {
```

Ukázka kódu 3.12: Ukázka for cyklů

```
if(mapik[c].barva==Qt::black)
 {
        barvas=qRgb(0,0,0);
 }
else  if(mapik[c].barva==Qt::white)
{
        barvas=qRgb(255,255,255);
}
else  if(mapik[c].barva==Qt::darkGreen)
{
        barvas=qRgb(0,102,0);
}
else  if(mapik[c].barva==Qt::blue)
{
        barvas=qRgb(0,0,255);
}
else   if(mapik[c].barva==Qt::red)
{
        barvas=qRgb(255,0,0);
}
else  if(mapik[c].barva==Qt::yellow)
{
        barvas=qRgb(0,0,0);
        pocatecnix=j+a-39;
        pocatecniy=i+b-39;
}
vykresleny_obr->setPixel(j+a,i+b,barvas);
                        .
```

Ukázka kódu 3.13: Ukázka tvoření QImage

3.1.5 Třída Ozobot

Tato třída obsluhuje chování simulace ozobota jako takové, tedy simulaci čtení kódu a jeho provedení. Hlavní metodou této třídy je metoda „Ozobot::decision“, která jako zdroj dat dostane QImage plátna a počáteční souřadnice x a y ozobota.

Z této metody běžící na samostaném vlákně se volají všechny ostatní pomocné metody třídy Ozobot. Dokud běží animace, emituje zpět novou pozici x a y, směr šipky, a barvu ozobota do metody „Form2_KresKody::New_Ozo“.

Metoda Ozobot::decision

Základní princip rozhodování této metody je vyobrazen na obrázku 3.2. Při stisknutí tlačítka „Spustit“ v grafickém rozhraní se zavolá tato metoda. Běží tak dlouho, dokud není stiknuto tlačítko „Zastavit“, nebo dokud ozobot nenarazí na konec dráhy.

Nejdříve se pomocí „Ozobot::is_there_way“ zjistí, zda existuje alespoň jeden směr, kterým může ozobot jet. Z hlediska implementace to znamená podívat se v QImage plátna na všechny čtyři směry o 40 pixelů od pozice x a y ozobota. Pokud barva pixelů není bílá, existuje cesta. Existující směr se uloží do QVectoru „dir“.

Pokud máme alespoň jeden směr, vybereme ho. Pokud máme více možných směrů, vybere se směr náhodně pomocí metody „Ozobot::pick_direction“. Po vybrání směru, kterým ozobot ujede se uloží barva povrchu, na které aktuálně stojí. Barvy se ukládají do QVectoru a ve chvíli, kdy se velikost vectoru rovná čtyřem, zavolá se metoda pro vykonání kódu „Ozobot::check_code“ - tato metoda bude zvlášť popsána níže.

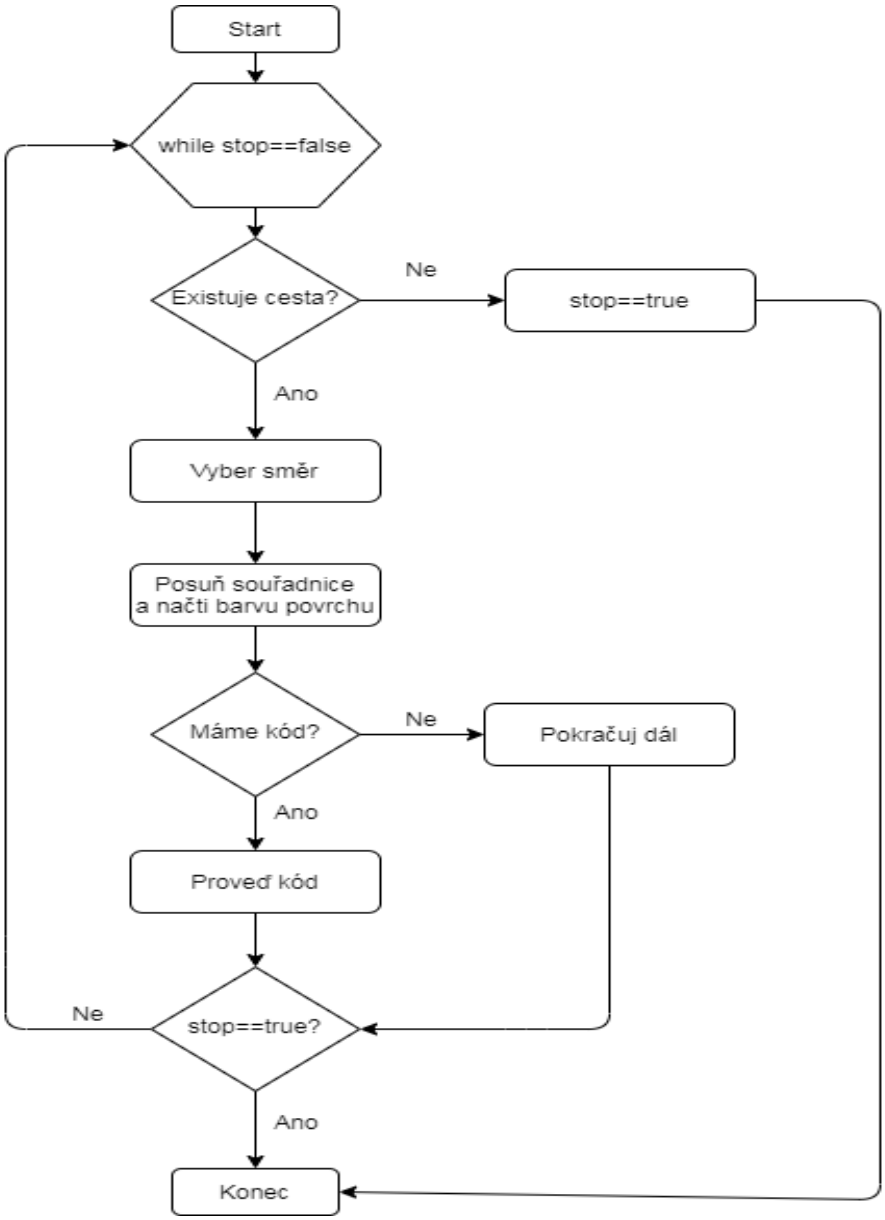
V případě, že by byl přečten kód určující konkrétní směr, musíme zkontrolovat zda daný směr existuje - to zajišťuje metoda „Ozobot::check_direction“. Pokud cesta tímto směrem není možná, metoda vrátí false a posun ozobota v tomto kole cyklu neproběhne. Vizuálně zpoždění vidět není.

Po úspěšném určení směru, načtení barvy a případném vykonání kódu se posune ozobotova souřadnice x nebo y o 40 pixelů. Nastavíme směr šipky na směr, ve kterém ozobot jede a emitujeme signál „Ozobot::send_n_coor“. Metoda „Form2_KresKody::New_Ozo“ vykreslí ozobota na nových souřadnicích, s barvou povrchu a šipkou ukazující směr cesty. Takto probíhá každý jednotlivý posun ozobota po dráze.

Pro vizuálně plynulý posun se rozloží emitování pomocí for cyklu, kde posun po 40 pixelech rozdělíme do 10 kroků cyklu viz. obrázek 3.14.

```
for(int  i=1;i<=10;i++)
{
emit  send_n_coor(lastx+i*(this->get_x()-lastx)/10,
lasty+i*(this->get_y()-lasty)/10,this->arrow,this->ozoColor);
QThread().currentThread()->msleep(this->get_speed()/10);
}
lastx=this->get_x();
lasty=this->get_y();
}
```

Ukázka kódu 3.14: Emit pomocí for cyklu



Obrázek 3.2: Diagram metody Ozobot::decision

Metoda Ozobot::check_code

Tato metoda zpracovává všechny kódy, přes které ozobot v animaci přejede. Jejím parametrem je řetězec složený ze čtyř prvků QVectoru „dir“, jenž obsahuje barvy povrchu, po kterých ozobot přejel. Tento řetězec se pak pomocí podmínek if a elseif testuje, zdali odpovídá nějakému barevnému kódu ozobota.

Protože ozobot čte tří až čtyřmístné kódy, musely by se používat dvě různé metody, které by kontrolovaly, zda máme třímístný nebo čtyřmístný kód. Proto byl všem třímístným kódům přidán sufix černé barvy, aby se tak mohl testovat vždy „čtyřmístný kód“.

Zde je potřeba dodat, že první načtená barva je vždy jiná než černá. Z důvodu toho aby se netestovali řetězce typu „black;black;black;black;“, nebo „black;black;black;red;“. V různých kombinacích dráh a kódů, které může uživatel vytvořit, by se pak takto nedalo zajistit, že se kód provede. Proto první načtená barva bude vždy červená, modrá či zelená.

Příklad testování, zdali máme kód je na obrázku 3.15. Zde je vidět, že barvy se do QVectoru dir přidávají ve tvaru „barva;“. Tento konkrétní příklad ukazuje nastavení rychlosti ozobota, které se projeví jako větší opoždění vykreslení pobyhu na plátně.

Pokud nesouhlasí řetězec s prvním testovaným kódem, přesouvá se pomocí podmínky else if k dalšímu testování zbylých implementovaných kódů. Když podmínka s řetězcem souhlasí, provede se daný kód a vymaže se QVector načtených barev pro opětovnou možnost načítání barev.

V případě, že kód je špatně zadan, například „red;blue;black;black;“ se pouze promaže Qvector „dir“ a pokračuje se v provádění animace.

```
if(code=="red;green;blue;black;")
    {
        this->set_speed(400);
        temp=false;
        code_vect.clear();
    }
```

Ukázka kódu 3.15: Kontrola zdali máme kód

Délka a náročnost provedení kódů se liší. Příkazy ozobota „Cik-Cak“ a „Pirueta“ mají své vlastní metody. Důvodem je, že provedení těchto kódů je na čtyři kroky. Příkaz „Cik-Cak“, který je popsán v první kapitole, změní směr šipky ozobota s jeho pohybem v daném směru čtyřikrát. Implementace musela být přizpůsobena ke skutečnosti, že ozobotův směr jízdy je náhodně vybírán.

Na obrázku **??** je vidět volání metody „Ozobot::cikcak“, která nám vrací směr šipky. Při přečtení kódu je nastavena proměnná „cikcak_bool“ na true. Provedení metody se pak volá v těle „Ozobot::decision“ před emitováním změněné polohy ozobota. Metoda se postupně volá čtyřikrát, kdy jí vždy předáme aktuální směr ozobota a číslo kroku. Po provedení všech čtyř kroků se proměnná „cikcak_ bool“

nastaví opět na false.

Na obrázku 3.16 je ukázáno tělo metody „Ozobot::cikcak“, kde používáme switch k určení směru šipky. Směr šipky se liší vždy na základě směru, kterým ozobot jede a v jakém kroku provedení příkazu je.

```
else if(this->cikcak_bool)
{
this->counter_cikcak++;
this->arrow=this->cikcak(this->direction,
                         this->counter_cikcak);
  if(this->counter_cikcak==4)
  {
    this->cikcak_bool=false;
    this->counter_cikcak=0;
  }
}
```

Ukázka kódu 3.16: Volání metody cic-cak

```
int Ozobot::cikcak(int direction,int count)
{
    switch(direction)
    {
    case 1:
        switch(count)
        {
        case 1: return 5;
        case 2: return 6;
        case 3: return 5;
        case 4: return 6;
        }
        break;
    case 2:
        switch(count)
        {
        case 1: return 7;
        case 2: return 8;
        case 3: return 7;
        case 4: return 8;
        }
        break;
    case 3:
        switch(count)
        {
        case 1: return 9;
        case 2: return 10;
        case 3: return 9;
        case 4: return 10;
        }
        break;
    case 4:
        switch(count)
        {
        case 1: return 11;
        case 2: return 12;
        case 3: return 11;
        case 4: return 12;
        }
        break;

    }
}
```

Ukázka kódu 3.17: Tělo metody cik-cak

3.2 Finální grafický návrh rozhraní

Rozhodnutí zanechat jen dvě okna se nezměnilo a zůstalo tedy logikou stejné. Nová uvítací obrazovka prošla změnou jen v podobě barvy. Při prvním testování rozhraní nástroje vznikly připomínky ke křiklavosti rozhraní při opakovaném testování (viz. Kapitola Otestování implementovaného nástroje).

Hlavní barva byla změněna na jemnější a méně křiklavou barvu - světle šedivou. Změnu můžeme vidět na obr. A.1. Funkčnost obrazovky zůstala stejná jako u první verze.

Obrázek 3.3: Ukázka upraveného vstupu do aplikace

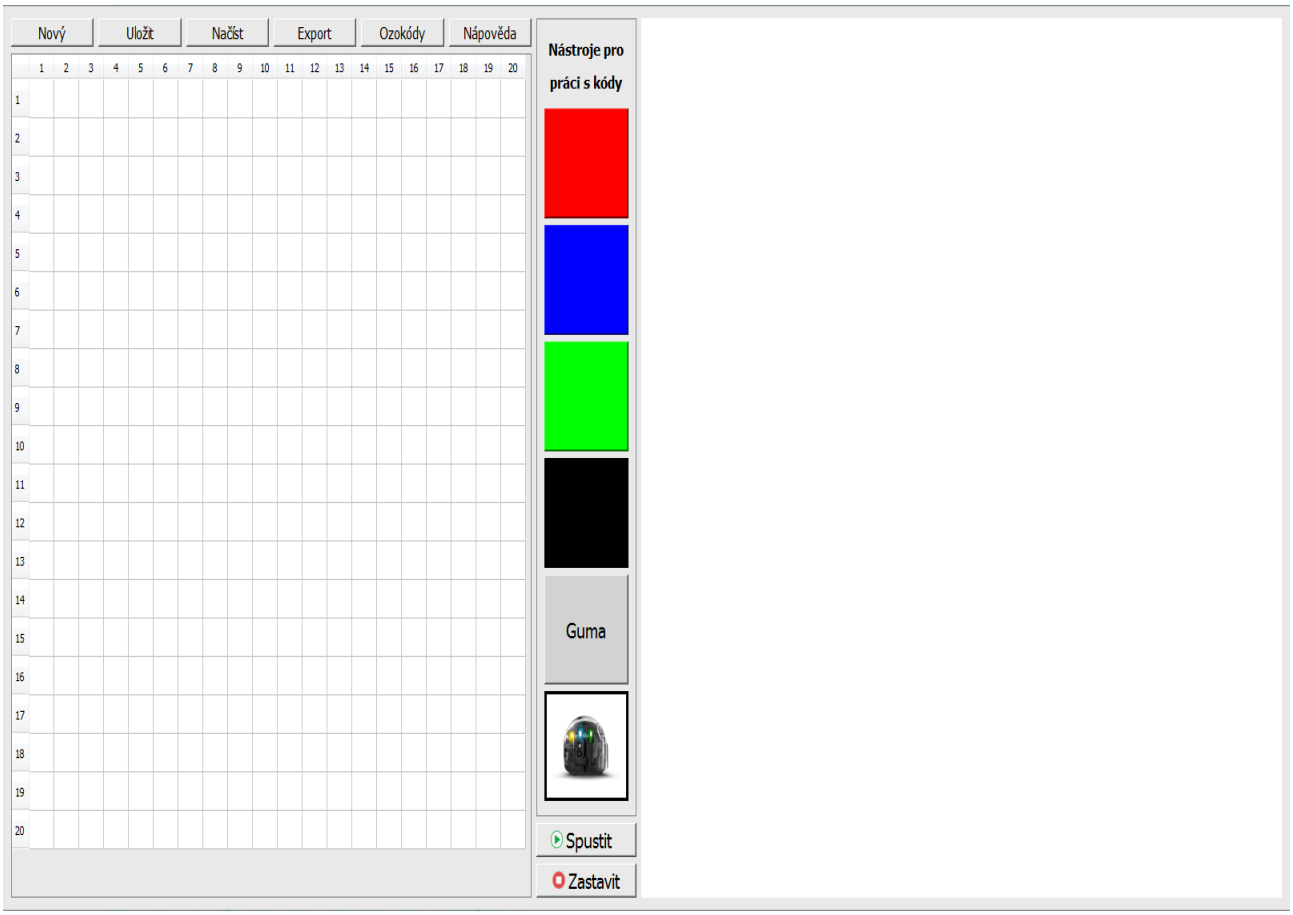
U hlavního rozhraní došlo k více změnám než jen k pouhé změně barvy. U změn šlo zejména o zlepšení funkčnosti pro uživatele. Po testování první verze rozhraní bylo k dispozici několik různých připomínek a nápadů. Výsledná verze rozhraní je vidět na obr A.2.

Do menu byly přidány dvě nové položky „Ozokódy“ a „Nápověda“. Tlačítka se díky zvýšení počtu zmenšila a nepůsobila tak roztažené, což byl problém zejména na monitorech s vysokým rozlišením - hlavní okno se totiž přizpůsobuje svou velikostí monitoru. Tabulka zůstala bodově stále 20x20, ale rozměry se přizpůsobila co nejvíc velikosti plátna. Šlo zde o to, aby nevypadala v porovnání s plátnem tak zmenšeně.

Ozokódy zobrazují v novém okně výpis všech kódů ozobota. Používá se k tomu oficiální výpis od výrobce, přeložený do čestiny. Nápověda nám zobrazí výpis se základním popisem aplikace simulátoru. Obsahuje tipy, jak simulátor správně používat, aby se předešlo chybám při používání. Chyby jsou však většinou ošetřeny

vyskakovacími okny, jak bude popsáno níže.

V prostřední části byly odstraněny názvy barev a přibyl nadpisek „Nástroje pro práci s kódy“ pro větší přehlednost. Z grafického hlediska se tlačítka zmenšila a tlačítka „Ozobot“, „Spustit“ a „Zastavit“ dostala navíc nový vzhled v podobě obrázku.

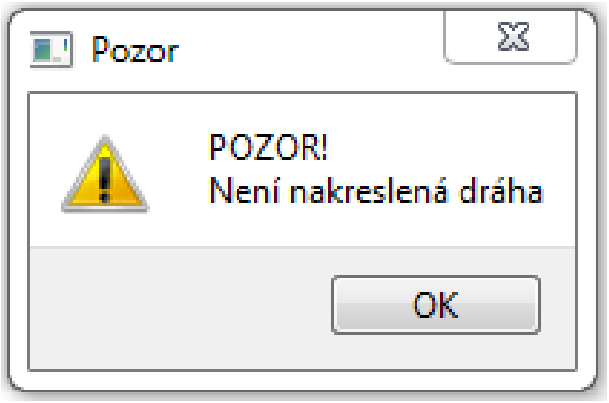


Obrázek 3.4: Finální vzhled simulátoru

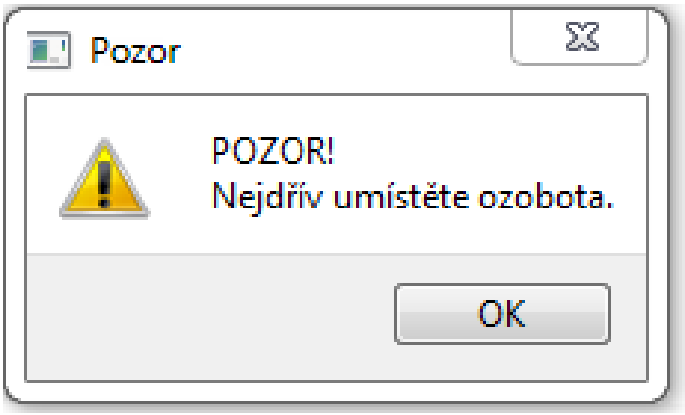
Po grafické a uživatelské stránce nám přibylo ještě pár vlastností. První z nich jsou vyskakovací okna v případě, že uživatel nepřipraví vše správně pro animaci. První vyskakovací okno se spustí ve chvíli, kdy nemáme do tabulky vložený žádný kód a klikneme na tlačítko „Spustit“ nebo „Export“. Simulátor nám nedovolí exportovat prázdné pdf. Vyskakovací okno je vidět na obrázku 3.5.

Druhé vyskakovací okno už se spouští jen při kliku na tlačítko „Spustit“. Pokud je splněno kritérium, že máme do tabulky umístěnou čáru, simulátor už pouze kontroluje, zda je umístěný na čáře i ozobot. Pokud není, zobrazí se vyskakovací okno viz. obr 3.6. Tlačítko export nevyžaduje pro vytvoření do pdf umístěného ozobota, protože jeho pozici do pdf nepřekresluje.

Poslední velká grafická změna probíhá opět při kliknutí na spouštěcí nebo zastavovací tlačítko. Během vykonávání animace se deaktivovala levá a prostřední část kromě tlačítek „Ozokódy“, „Nápověda“ a „Zastavit“. Nebylo to však podtržené vizuálně, což vedlo k pokusům umístit do tabulky nový kód, přestože simulace nebyla



Obrázek 3.5: Vyskakovací okno při nezadaném kódu



Obrázek 3.6: Vyskakovací okno při neumístěném ozobotovi

zastavena. V této finální verzi všechny deaktivované části zešednou, dokud není simulace zastavena. Po zastavení simulace části opět nabydou zpět svých barev a funkčnosti.

Testování finální verze pomocí těchto uživatelských a grafických kontrol proběhla mnohem úspěšněji než původní verze simulátoru. Průběh testování je detailněji popsán v kapitole 4.

Kapitola 4

Otestování implementovaného nástroje

Po dokončení implementace nástroje následovalo jeho otestování. Testovány byly dvě verze: úplně první verze rozhraní viz. kapitola 2 a finální verze rozhraní viz. předchozí kapitola. Pro testování simulátoru byl vybrán průřez věkovými kategoriemi. Simulátor testovaly tři děti ve věku základní školy a jeden vyučující.

4.1 Testující

Vzhledem k GDPR nebudou uvedena jejich jména, ale jen jejich věk a třída, do které chodí. V případě učitele je uveden stupeň, na kterém učí. Dále budou uvedeny pouze výsledky testování a názor na první a finální verzi simulátoru.

První žačka je ze třetí třídy základní školy ve věku devíti let. Druhý žák ve věku jedenácti let je z páté třídy základní školy, zároveň chodil na kroužek s ozoboty. Třetí žák je ze šesté třídy základní školy, věk třináct let, s ozoboty měl menší zkušenost. Poslední testující byl učitel střední školy, ve věku dvaceti šesti let.

4.2 Testování První verze

Na rozdíl od finální popsané verze neobsahovala první verze vyskakovací okna s upozorněními, v menu chyběly tlačítka „Nápověda“ a „Ozokódy“ a byla implementována jen půlka kódů, většinou třímístných. Zároveň ještě nebylo implementováno plynulé vykreslování ozobota, pohyboval se pouze najednou skoky po 40 pixelech.

4.2.1 První testující

Design

Žačce se design pozadí zlatého svítivého přechodu líbil. Měla menší potíže si zvyknout na používání tlačítka „Ozobot“, jelikož barevně splývalo s pozadím, zároveň graficky neevokovalo ozobota. Vyjádřila stížnost k příliš malé tabulce. Průběh animace se jí i přes trhaný pohyb ozobota líbil.

Práce se simulátorem

Jak již bylo zmíněno, žačce se nelíbila malá tabulka na zadávání dráhy ozobota, ale se zadáváním kódu jako takovým neměla po chvilce žádné potíže. Přes upozornění, aby nezadávala dráhu do vnitřního okraje tabulky, občas danou buňku zaplnila, což vedlo ke spadnutí programu.

V implementační kapitole kontrolujeme cestu o 40 pixelů napřed, což v okrajích tabulky způsobovalo pád simulátoru, protože byl kontrolovaný index out of range. Dále byl k simulaci přiložen papír s vytisknutými ozokódy, bez nich by musela použít internet.

Po dokončení animace nastala situace, kdy se snažila přidat dráhu, ale nestiskla „Stop“ a nešlo tedy klikat na tlačítka. Nastavení disabled nezešedivělo barevná tlačítka a nebylo tak tedy dobře poznat, že na ně nelze kliknout.

4.2.2 Druhý testující

Design

Barevný přechod se žákovi třetí třídy moc nelíbil. Navrhl, že by barva mohla být například modrá nebo zelená, jelikož tyto barvy se mu líbí víc. Z animace ozobota byl trochu zmatený, očekával spíše 3D animaci a ne 2D. Zároveň se mu moc nelíbil trhaný pohyb ozobota. Tlačítka „Spustit“ a „Stop“ se mu zdála stejně jako tabulka moc malá.

Práce se simulátorem

S přiloženým papírem ozokódů neměl problém vytvořit dráhu a vyzkoušet některé příkazy. Zastával názor, že má moc malou plochu, na kterou může vytvářet dráhu pro ozobota. Stejně jako u testované žačky občas zaplnil i okrajové buňky, což vedlo ke spadnutí simulátoru. Občasně zapomínal umístit ozobota, jelikož se neobjevilo žádné upozornění při spouštění simulace.

Se stisknutím tlačítka „Stop“ pro zpřístupnění přidávání další dráhy neměl po chvíli problém. Testoval dále i menu simulátoru, kde vytvořil několik různých uložených souborů typu csv, které pak různě načítal. Zkoušel i prázdný export prázdné

stránky pdf, která také způsobila pád aplikace.

4.2.3 Třetí testující

Design

Nejstarší žák měl ze všech testovaných studentů nejvíce doplňujích dotazů a návrhů. Co se barvy pozadí simulátoru a tlačítek týče, barevné přechody neměly moc velký úspěch. Také poukázal na to, že je zbytečné k barvám dopisovat popisky s jejich názvy.

Problém s typem simulace 2D neměl, jen se mu nelíbil trhaný pohyb. U provedení příkazu pro nastavení nejvyšší rychlosti byl spokojenější.

Práce se simulátorem

Ze začátku měl problém stisknout nejdříve tlačítko „Stop“. Podle jeho názoru nebylo na první pohled vidět, že jsou tlačítka a tabulka vypnuté. Přibližně jednou, nebo dvakrát zaplnil i okrajové buňky, což opět vedlo k pádu simulátoru.

Zmínil se, že by bylo dobré, aby ho simulátor nenechal kliknout na okrajové buňky. Zároveň mu chyběly ozokódy přímo v aplikaci, přestože mu nevadilo je číst z papíru.

4.2.4 Čtvrtý testující

Design

Design přechodů zlaté barvy se opět setkal s neúspěchem. Vyučující navrhl alternativu v méně křiklavé barvě, která by při častém používání nebila tolik do očí. Stejně jako studentům se mu příliš nezamlouvala malá tabulka v kontrastu s velkým plátnem. Názvy barev na tlačítkach nejsou potřeba a animace by mohla být mnohem plynulejší.

Napadlo ho přidat k tlačítku ozobot obrázek pro ujasnění pro nejmladší žáky základních škol. Všiml si dále, že při jednom směru je šipka trochu jiná než ostatní šipky. Tlačítkové menu se mu líbilo, zejména z důvodu používání simulátoru dětmi základních škol.

Práce se simulátorem

S upozorněním, aby nezaplňoval vnitřní okraje tabulky neměl problém se jejich zaplňování vyhnout. Nicméně poznamenal, že by bylo dobré, aby ho na to simulátor upozornil, případně ho tam vůbec nenechal kliknout. Stejně tak jako upozornění u pdf exportu o nutnosti existenci dráhy.

Dále by dle jeho názoru bylo užitečné mít v programu po ruce ozokódy, aby se opravdu nemusel použít internet k jejich dohledání, nebo se nemusely tisknout na papír. U vytištění výstupu z pdf exportu se přišlo na špatný odstín zelené barvy. Všechny ostatní barvy ozobot rozpoznal v pořádku.

4.2.5 Výsledek testování první verze

Z nasbíraných poznatků při testování vyplynula potřeba změnit a přidat několik částí simulátoru - jak v rámci designu, tak v rámci funkčnosti uživatelského rozhraní.

Design

- změnit barvu pozadí
- odebrat názvy barev z tlačítek
- více zvýraznit disabled stav tlačítek
- zvětšit velikost tabulky v poměru k plátnu
- zvýšit plynulost animace

Práce se simulátorem

- přidat vyskakovací okna s upozorněními
- do menu přidat tlačítko nápovědu
- přidat do simulátoru ozokódy
- upravit zelenou barvu pro pdf výstup

4.3 Testování Finální verze

Finální verze prošla řadou změn, které byly založeny na zpětné vazbě od testujících. Výsledný design můžeme vidět v kapitole 3.3. Finální verzi opět testovali stejní studenti a vyučující.

4.3.1 První testující

Design

Vzhledem k tomu, že se žačce původní barva pozadí líbila kvůli barevnosti , shledala novou barvu pozadí poněkud mdlou. Líbila se jí naopak změna barevného přechodu tlačítek na plné barvy. Změna velikosti tabulky měla pozitivní ohlas.

Zároveň se jí líbilo nové grafické „zešednutí“ všech barevných tlačítek ve stavu disabled a jejich opětovné „rozsvícení“. Plynulejší animace zlepšila hodnocení simulátoru.

Práce se simulátorem

Ošetření vyskakovacími okny a znemožnění zakreslení barev do okrajových buňěk pomohlo k odstranění padání simulátoru. Po přidání obrázku ozobota na tlačítko pro umístění ozobota se zlepšila její orientace v simulátoru.

4.3.2 Druhý testující

Design

Žák očekával spíše jinou barvu, než šedivou, ale celkový nový vzhled rozhraní hodnotil kladně.

Práce se simulátorem

Díky vyskakovacím oknům s upozorněními na chybějícího ozobota nebo nezadanou dráhu se zjednodušila pro studenta práce se simulátorem. Odstranilo se padání programu při přidání dráhy do okrajových buňek.

Student dále vyzkoušel téměř všechny implementované příkazy bez velkých potíží. Pokud se příkaz nepodařilo vykonat, jednalo se většinou pouze o chybně zadaný kód, nikoliv o chybu simulátoru. Tento problém byl také odstraněn, když si student zvykl používat tabulku s ozokódy přidanou do menu.

4.3.3 Třetí testující

Design

Novým designem byly odstraněny nedostatky, které žák k designu simulátoru měl. Napadlo ho ještě přidat barevné akce při kliknutí, nebo přejetí tlačítek myší. Dále, při přejetí nad tlačítkem „Ozobot“, změnit obrázek tlačítka na gif s ozobotem. Plynulejší animace byla přijata velmi kladně.

Práce se simulátorem

Ovládání dle jeho názoru bylo mnohem přehlednější než u první verze. Student se pokusil způsobit pád programu jako v případě první verze, ale ošetření vyskakovacími okny tomu zabránilo. Napadlo ho navíc přidat do simulátoru nahrávání a ukládání videa simulace.

4.3.4 Čtvrtý testující

Design

Vyučující ocenil změnu grafického rozhraní a světle šedivou barvu pozadí. Dle jeho názoru je barva z dlouhodobého hlediska při opakovaném používání simulátoru méně rušivá. U komponentů rozhraní ocenil jejich úměrnost. Vznikl zde dotaz, zda by nebylo lepší vytvořit tabulku 20x30 místo 20x20.

Práce se simulátorem

Stejně jako u studentů se podle něj zlepšila uživatelská zkušenost se simulátorem. Ocenil možnost zobrazení, čímž se odstranila potřeba ozokódy tisknout nebo je vyhledávat na internetu. U výstupu pdf exportu byla upravena zelená barva, kterou již ozobot rozpoznal.

Jediná výtka, která vznikla k vytištěnému výstupu byla, že by obraz plátna na papíru formátu A4 mohl mít o něco větší rozměry. Celkový názor na simulátor byl takový, že je užitečnou výukovou pomůckou.

4.3.5 Shrnutí testování

Díky připomínkám a nápadům testujících se podařilo vytvořit z první verze aplikace stabilní aplikaci. Byly odstraněny nalezené problémy, kdy simulátor padal. Došlo ke zlepšení vykreslení simulace a zlepšení uživatelského rozhraní a designu celého simulátoru.

Zároveň jsem získal několik nápadů pro přidání nových možností aplikace do budoucna. Celkové hodnocení finální verze testujících bylo velmi kladné.

Závěr

Hlavním cílem této práce bylo vytvořit uživatelsky přívětivou aplikaci pro simulaci výukového nástroje ozobot evo a zpřístupnit tak i výuku na školách, které nástroj používají k výuce základů programování a rozvíjení logického myšlení studentů. Zároveň zajistit odstranění případného nedostatku ozobotů na počet studentů a možnost zkoušení programů před spuštěním na ozobotovi.

Výsledkem práce je otestovaná stabilní aplikace vhodná použití z pohodlí domova nebo ve školních počítačových učebnách. K jejímu použití není potřeba internet, nepotřebuje žádné doplňující nástroje, vše je v ní obsažené. Umožňuje přípravu výukových materiálů pro vyučující, kteří používající ozobota evo. Dalším pozitivem je, že se studenti mohou připravovat sami doma, bez nutnosti vlastnit fyzicky ozobota evo.

V budoucnu by se aplikace měla vyvíjet směrem ke zvládnutí simulace nejen nakreslené dráhy a kreslených kódů, ale i kódů naprogramovaných na oficiálních stránkách výrobce - rovnou je i upravovat a nabízet export a import mezi těmito dvěma platformami. Výsledkem bude aplikace kompletně simulující všechny možnosti nástroje ozobot evo.

Literatura

[1] BAKA, Benjamin. *Getting Started with Qt 5*. 2019. ISBN 978-1789956030.

[2] ENG, Lee Zhi. *Qt5 C++ GUI Programming Cookbook*. Second Edition. 2019. ISBN 978-1789803822.

[3] FIŠER, Jiří. *Qt - programování v příkladech*. Ústí nad Labem: Univerzita J.E. Purkyně v Ústí nad Labem, 2003.

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

[5] LAZAR, Guillaume a PENEA ROBIN. *Mastering Qt 5*. Second Edition. 2018. ISBN 978-1788995399.

[6] VOIDREALMS. C++ Qt 122 - QtConcurrent Run a thread with signals and slots. *Youtube.com* [online]. 20.12.2014 [cit. 2020-04-10]. Dostupné z: https://www.youtube.com/watch?v=tvpC8UrPpZ4&list=LLwPnUC9v0_7GLHFDWu2E8AA&index=37&t=0s

[7] *Ozobot* [online]. [cit. 2020-01-12]. Dostupné z: https://ozobot.com/

[8] *Ozoblockly* [online]. [cit. 2020-03-21]. Dostupné z: https://ozoblockly.com/

Příloha A

Uživatelský manuál

Postup pro správné použití simulátoru.

Spuštění

- Po spuštění .exe souboru se zobrazí spouštěcí obrazovka simulátoru (viz. obrázek A.1)
- Klikněte na tlačítko „Spustit“
- Zobrazí se prostředí simulátoru (viz. obrázek A.2)

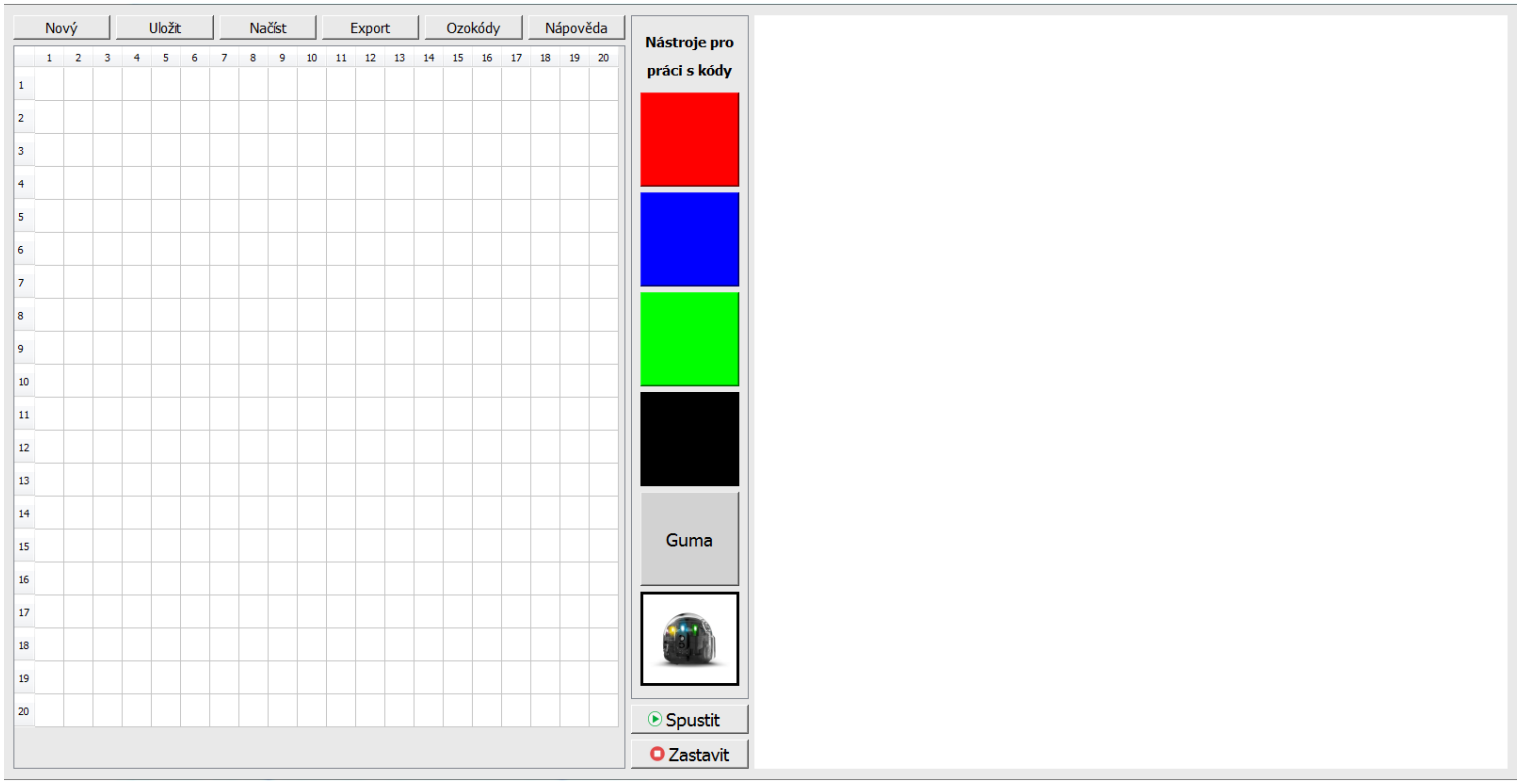
Start/stop simulace

- Pro vytvoření dráhy a přidání ozobota:
  - Vyberte barvu dráhy kliknutím na jedno z barevných tlačítek v sekci „Nástroje pro práci s kódy“
  - Klikněte na libovolné pole v tabulce v levé části obrazovky (vyjma okrajových)
  - Pro vymazání barvy políčka v tabulce klikněte na tlačítko „guma“ a poté na políčko, které chcete smazat
  - Po vytvoření dráhy klikněte na tlačítko s obrázkem ozobota
  - Klikněte do vyplněného pole v tabulce pro umístění ozobota
- Simulaci spustíte kliknutím na tlačítko „Spustit“
- Simulaci zastavíte kliknutím na tlačítko „Zastavit“

Obrázek A.1: Spouštěcí obrazovka

Menu simulátoru

- Kliknutí na tlačítko „Nový“ vyčistí prostředí simulátoru (promaže tabulku a plátno)
- Kliknutí na tlačítko „Uložit“ zobrazí ukládácí dialog a uloží data z tabulky do csv souboru
- Kliknutí na tlačítko „Načíst“ zobrazí načítací dialog a načte data z csv souboru do tabulky
- Kliknutí na tlačítko „Export“ zobrazí ukládací dialog a uloží plátno v pravém rohu do formátu pdf
- Kliknutí na tlačítko „Ozokódy“ zobrazí v novém okně implementované ozokódy
- Kliknutí na tlačítko „Nápověda“ zobrazí v novém okně stručnou nápovědu k simulátoru

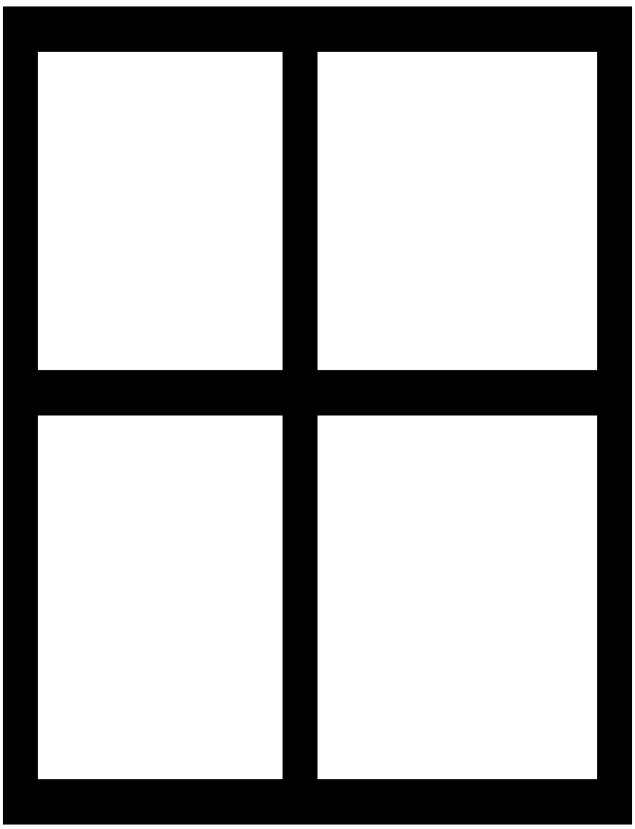


Obrázek A.2: Prostředí simulátoru

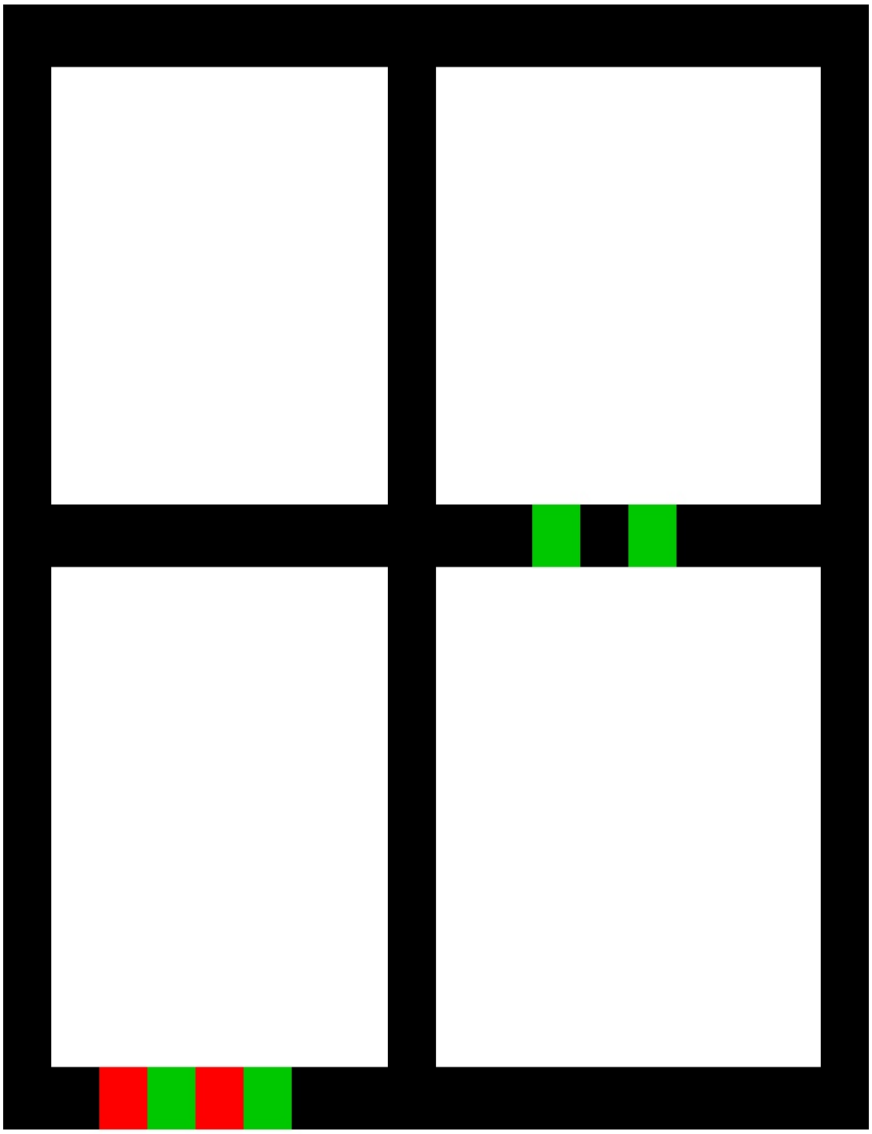
Příloha B

Ukázka pdf exportu

Vygenerováno simulátorem Ozobot Evo v.1.0.



Vygenerováno simulátorem Ozobot Evo v.1.0.



Příloha C

Obsah CD

- Složka A
  - Zdrojový kód simulátoru
- Složka B
  - Zkompilovaný program simulátoru ke spuštění
- Složka C
  - Bakalářská práce ve formátu pdf