Úlohy pro výuku programovaní v prostředí robotického simulátoru

Bc. Pavel Ševčík

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Pavel Ševčík**
Osobní číslo:	**A22312**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Úlohy pro výuku programování v prostředí robotického simulátoru**
Téma práce anglicky:	**Programming Education Tasks in a Robotic Simulator Environment**

Zásady pro vypracování

1. Prostudujte robotické simulátory Webots a CoppeliaSim.
2. Vyberte simulátor, který se jeví jako vhodnější pro výuku programování v úvodních kurzech a kroužcích robotiky na FAI UTB.
3. Pro některé úlohy z těchto kurzů navrhněte 3D pracovní prostředí.
4. Implementujte ukázkový řídicí kód robota, který splní zadání úlohy.
5. Ke každé úloze vytvořte dokumentaci a návod k řešení.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. KIRTAS, M., K. TSAMPAZIS, N. PASSALIS a A. TEFAS. Deepbots: A Webots-Based Deep Reinforcement Learning Framework for Robotics. In: Ilias MAGLOGIANNIS, Lazaros ILIADIS a Elias PIMENIDIS, ed. Artificial Intelligence Applications and Innovations [online]. Cham: Springer International Publishing, 2020, s. 64–75. IFIP Advances in Information and Communication Technology. ISBN 978-3-030-49186-4.
2. MORDENTI, Andrea. Programming Robots with an Agent-Oriented BDI-based Architecture: Explorations using the JaCa and WeBots platforms. S.l.: LAP LAMBERT Academic Publishing, 2013. ISBN 978-3-659-33303-3.
3. STARY, Vadim a Lukas GACHO. Modelling and Simulation of Missile Guidance in WEBOTS Simulator Environment. In: 2020 19th International Conference on Mechatronics – Mechatronika (ME): 2020 19th International Conference on Mechatronics – Mechatronika (ME) [online]. Prague, Czech Republic: IEEE, 2020, s. 1–5 [vid. 2021-12-03]. ISBN 978--1-72815-602-6.
4. CARBONELL, Vanessa Cruz a Ricardo Andrés Castillo ESTEPA. Simulation of a Quadrupedal Bioinspired Modular Robot Using Webots. International Review on Modelling and Simulations (IREMOS) [online]. 2019, 12(2), 94–102. ISSN 2533-1701.
5. PACHECO, Julio a Francesc BENITO. Development of a Webots Simulator for the Lauron IV Robot. In: Proceedings of the 2005 conference on Artificial Intelligence Research and Development. NLD: IOS Press, 2005, s. 347–354. ISBN 978-1-58603-560-0.
6. MICHEL, Olivier. Webots: Symbiosis Between Virtual and Real Mobile Robots. In: Jean-Claude HEUDIN, ed. Virtual Worlds [online]. Berlin, Heidelberg: Springer, 1998, s. 254–263. Lecture Notes in Computer Science. ISBN 978-3-540--68686-6.

Vedoucí diplomové práce: **Ing. Tomáš Dulík, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne .....

podpis studenta

ABSTRAKT

Tato diplomová práce se zabývá porovnáním robotických simulátorů určených pro výuku robotických kroužků a na vysoké škole. Porovnáme zde dva robotické simulátory a vybereme ten vhodnější pro tyto účely. Dále se zde nachází vypracování úloh, které budou použity pro výuku.

Klíčová slova: robotický simulátor, Webots, CoppeliaSim, robot, program

ABSTRACT

This thesis deals with the comparison of robotic simulators designed for teaching robotics course and at the university level. We will compare two robotic simulators and select the more suitable one for these purposes. Additionally, the thesis includes the development of tasks that will be used for teaching.

Key words: robotic simulator, Webots, CoppeliaSim, robot, program

Tímto bych chtěl poděkovat panu Ing. Tomáši Dulíkovi, Ph.D. za vedení mé diplomové práce, jeho rady, pomoc i všechnu konzultaci a časy, kdy mi byl nápomocný i za jeho lidský a milý přístup a všechny vědomosti, které mi za dobu studia na této škole předal.

Dále bych chtěl poděkovat celé své rodině za trpělivost, kterou semnou měla při psaní této práce a za její neskonalou výdrž a milost. V neposlední řadě bych chtěl podekovat i svému drahému králíkovi Timmymu, který semnou přečkal tyto časy a mým drahým kamarádům a kolegům.

Všem Vám mockrát děkuji a velmi si Vás vážím!

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tématem této diplomové práce jsou úlohy pro výuku programování v prostředí robotického simulátoru. K výběru tohoto tématu mě vedly dlouholeté zkušenosti se stavbou a programováním robotů a s účastí na robotických soutěžích jak na území České republiky, tak na mezinárodní úrovni. Další motivací je mé vedení robotického kroužku na Fakultě aplikované informatiky ve Zlíně a pořádání celorepublikové soutěže RoboGames na naší škole. V oblasti robotiky i její výuky se používané nástroje a technologie vyvíjí raketovým tempem a nejde se zastavit na jednom místě. Je potřeba sledovat nové technologie i trendy – jak v robotice, tak u výukových nástrojů. Fenoménem posledních let se stávají právě robotické simulátory, kde lze robota vymodelovat, naprogramovat a dále s ním i například soutěžit. Tento trend se pokoušíme zavést i do našeho kroužku robotiky.

V této diplomové práci jsem vytvořil a prezentuji sadu příkladů, které budou sloužit jako nástroj pro vyučující robotických předmětů nebo kroužků pro studenty či žáky, kteří pomocí těchto úloh budou moci začít s modelováním a programováním robotů, popřípadě s rozvíjením jejich dovedností.

V kapitole 1 je obsaženo shrnutí technologií, používaných v kroužku robotiky na FAI UTB. Následně v kapitole 2 rozebírám disciplíny robotických soutěží, které dále budu modelovat a převádět do zadání úloh. V kapitolách 3 a 4 provádím průzkum a srovnání robotických simulátorů Webots a CoppeliaSim. Implementace řídicích kódů robotů pro splnění zadaných úloh se nachází v kapitolách 6 až 10 společně s vytvořením dokumentace a návodů řešení.

I. TEORETICKÁ ČÁST

1 ROBOTIKA VE VÝUCE

Již dlouhou řadu let se robotika neodmyslitelně stala součástí našich životů. Její vliv každým rokem roste, a proto je důležitým faktorem u žáků a studentů, kteří s robotikou začínají, zvolit správnou platformu, se kterou se budou studenti potýkat při jejich prvních krocích.

1.1 Robotické platformy využívané při výuce začátečníků na FAI

Již dlouhou řadu let se při výuce na FAI využívají roboti, sestavení ze stavebnic LEGO®. Jedná se o stavebnice, kde celým mozkem robota je řídicí jednotka, která se následně programuje. Tělo robota je sestaveno ze součástek stavebnice LEGO®, což dodává pro konstrukci robota širokou škálu možností.

Tato platforma poskytuje přístup k jednoduchým a interaktivním projektům, které studenti mohou vytvořit a následně sestavit a naprogramovat.

1.1.1 LEGO® Mindstorms® NXT

Verze NXT navázala na předchozí verzi Mindstorms RCX. Stavebnice NXT umožňovaly připojit 3 motory a 4 senzory. Byly nakoupeny a zařazeny do výuky oboru „Učitelství informatiky“ na FAI. Přibližně s nástupem této verze stavebnic vznikly v ČR a v zahraničí robotické soutěže, kde mohly být používány pouze tyto stavebnice.

1.1.2 LEGO® Mindstorms® EV3

Novější a vylepšená verze EV3 měla na rozdíl od předchozí řady rozšířený počet portů na 4 motory, což byla při komplexnějších stavbách robotů velmi vítaná možnost. Dalšími výhodami oproti NXT byly například vylepšené senzory. Také stavebnice EV3 byla nakoupena a je dodnes využívána ve výuce kroužku robotiky a 1. ročníku oboru „Inteligentní systémy s roboty“ na FAI UTB.

Řídicí jednotka FAI „LETS GO“

V prvních letech kroužku robotiky na FAI UTB byla v diplomové práce pana Davida Wunderlicha [8] vytvořena řídicí jednotka, která je kompatibilní s LEGO® Mindstorms® NXT a EV3 senzory a motory. Jednotka obsahuje CPU STM32F407 s firmware Espruino, jehož IDE umožňuje programování pomocí grafických bloků Google Blockly nebo v jazyce JavaScript.

1.1.3 LEGO® Spike a Mindstorms® Inventor

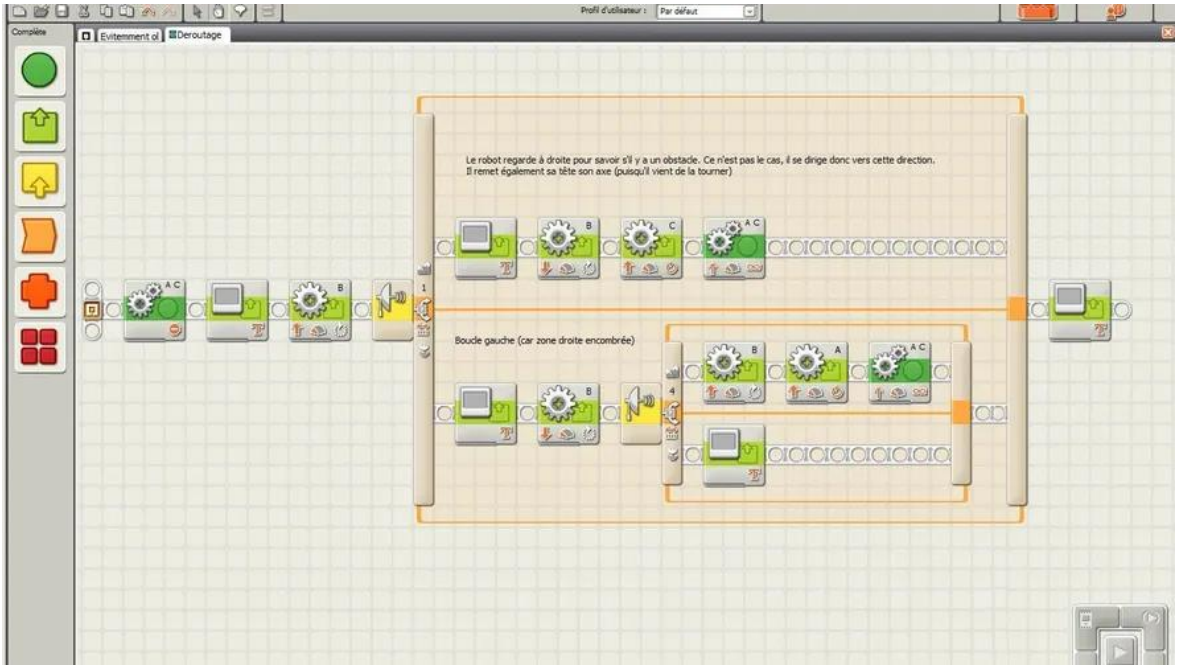
Velkým zlomem se stal příchod edice Spike a Mindstorms® Inventor. Dalo by se říct, že oproti verzím NXT a EV3 zde nezůstal kámen na kameni. Došlo k razantnímu zmenšení řídicí jednotky a ke změně tvaru i typu signálů portů. Tyto kroky vedly k tomu, že součásti této stavebnice již nadále nebyly kompatibilní s předešlými dvěma řadami stavebnic. Řídicí jednotka Spike je zcela shodná s verzí Inventor, rozdíl je pouze v jejich firmware a IDE. Zatímco IDE pro verzi Inventor vyžaduje instalaci desktopové aplikace, IDE pro verzi Spike je k dispozici jako online web, tj. instalaci aplikace nepotřebuje.
Obě IDE připomínají oblíbený Scratch a obsahují také možnost programovat roboty pomocí psaného kódu v pythonu. Dále je zde velmi užitečná varianta programování na mobilním telefonu či tabletu a přenosu programu do řídicí jednotky pomocí Bluetooth, možnost programování pomocí USB kabelu samozřejmě také zůstává. Celý robot může být bezdrátově ovládán pomocí mobilního zařízení či tabletu.

1.2 Vývojové prostředí a jazyky

V průběhu doby prošla vývojová prostředí velkou řadou změn. Jednotlivé verze mají naprosto odlišný styl tvorby programu. Jedinou výjimkou jsou verze Inventor a Spike, kde nyní aktuální varianta Spike používá stejné vývojové prostředí jako již ukončená varianta Mindstorms® Inventor.

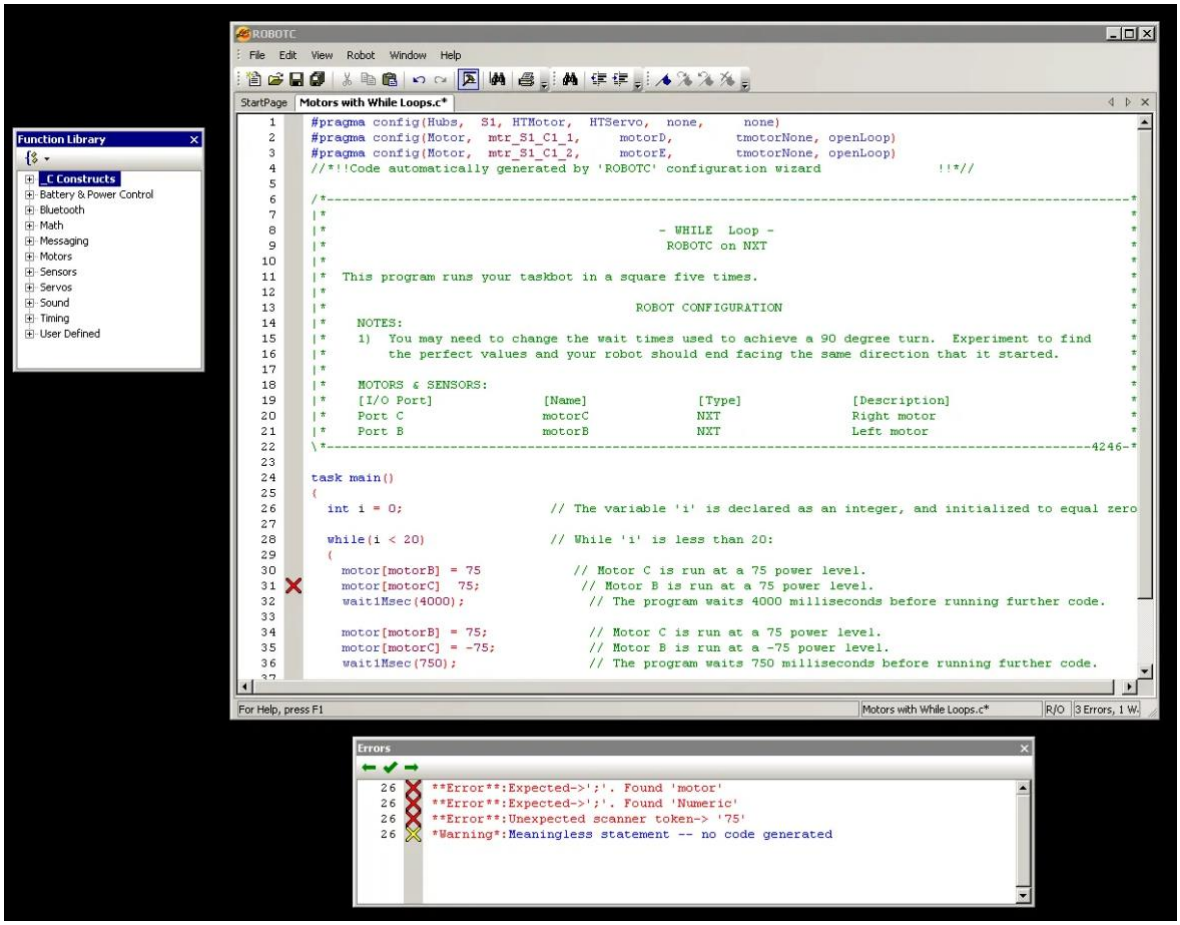
1.2.1 NXT

První generace nabízela dva primární druhy programování. Pro začátečníky jednodušším a více přívětivým se jevil jazyk NXT-G. Jednalo se o skládání bloků za sebe, čímž vznikal výsledný program. Tato varianta programování se těšila velké popularitě hlavně u mladších studentů a začínajících programátorů skrze svou jednoduchou syntaxi, kde každý blok reprezentoval daný úkon či krok, který má být proveden. Nevýhodou ovšem mohla být u rozsáhlejších projektů snížená přehlednost, skrze velikosti výsledného programu při skládání dlaždic za sebe. Další nevýhodou byl fakt, že při složitějších programech nešlo zacházet do hlubších detailů, tudíž program nemohl být tak moc propracovaný, jako by byl, kdyby byl psán textovým kódem.



Obrázek 1 - Jazyk NXT – G [1]

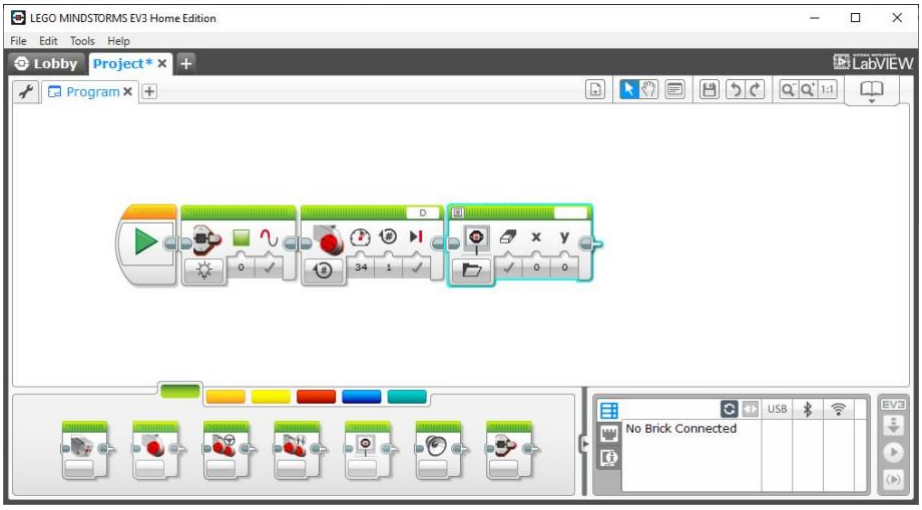
Druhou variantou, která šla u této řady využít, byl jazyk C. Tato varianta se nabízela spíše pro pokročilejší uživatele, kteří již za sebou měli určité základy programování a alespoň nějakou znalost jazyku C. Výhodou této varianty byla možnost více a detailněji pracovat s robotem a uživatel si mohl dovolit dělat více propracované programy.



Obrázek 2 - RobotC [2]

1.2.2 EV3

Druhá generace Mindstorms pod názvem EV3 přinesla do svého vývojového prostředí spíše designový upgrade. Prostředí bylo přepracováno a úpravou prošly i jednotlivé bloky. Tyto bloky změnily svůj design, ale prošly i formou rozšíření svých možností úprav, kde s bloky šly provádět rozsáhlejší operace.

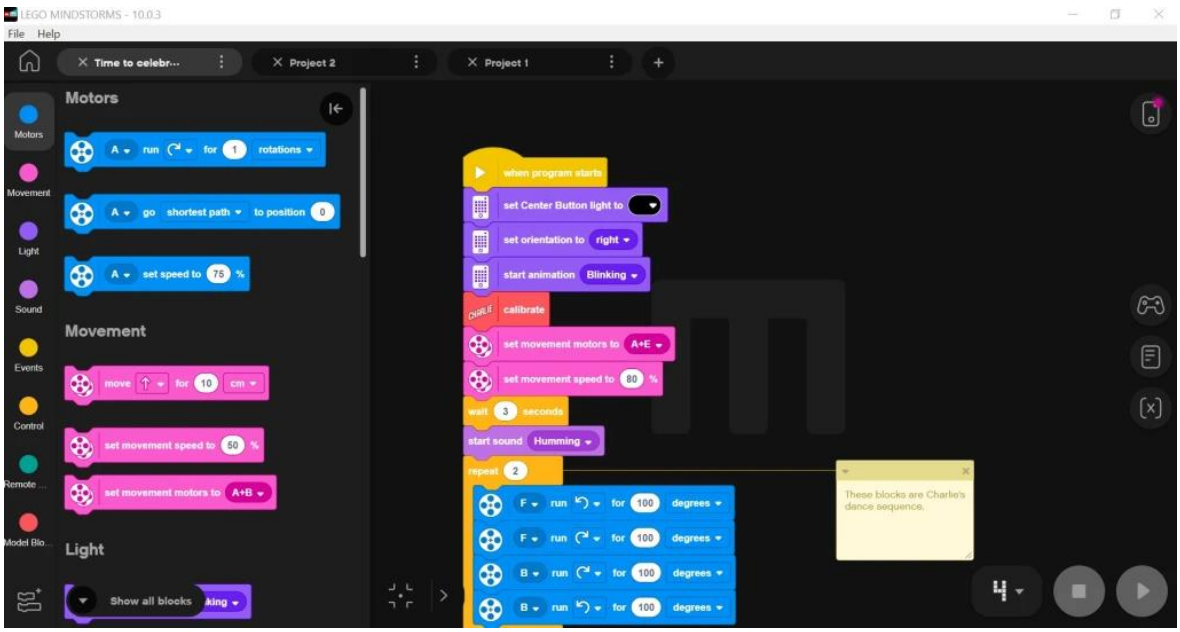


Obrázek 3 - EV3[3]

1.2.3 Inventor + Spike

Největší změnu přinesl příchod verze Inventor, kde došlo k úplnému předělání vývojového prostředí. Došlo k razantní změně platformy do stylu moderního fenoménu programování ve stylu Scratch nebo Google Blockly. Ne příliš kompaktní dlaždice se přeměnily v bloky, které se skládají pod sebe, což má za následek větší přehlednost a jednoduchost. Tento jednotný styl, který v dnešní době využívá většina programovacích platforem zaměřující se na mladší žáky má za důsledek sjednocení a zjednodušení vnímání programování, jelikož začínajícím programátorům usnadňuje přechod mezi jednotlivými platformami. Dalším krokem může být přechod na textové programování v Pythonu, jenž je taktéž v dnešní době velmi rozšířený a populární programovací jazyk. To může děti motivovat k programování v tomto jazyce.

Bezpochyby velkým přínosem a ulehčením je možnost naprogramovat robota pomocí tabletu či mobilního zařízení, jenž má za důsledek možnost větší mobility.



Obrázek 4 - Inventor + Spike IDE [4]

2 TYPY ÚLOH

Rozmanitost úloh při tvorbě robotů je opravdu obrovská. Cokoliv si daný jedinec usmyslí, lze vytvořit, sestavit, naprogramovat a následně přivést k robotímu životu. Základní a nejvíce rozšířené typy úloh a disciplíny na soutěžích, se kterými se studenti setkávají, jsou ovšem následující. Tyto úlohy naučí uživatele pracovat s hlavními a těmi nejdůležitějšími senzory, jak funguje fyzika v robotice a základní prvky algoritmizace.

2.1 Drag Race

Prvními kroky bývají pochopení chování robota. Přesně pro tento účel se hodí tato disciplína. Jednoduše řečeno je převzata inspirace z amerických závodů aut na rovné ploše. Úkolem robota je projet rovný úsek dráhy ohraničený mantinely s předem definovanou délkou za co nejkratší čas.

Na první pohled se může zdát, že jde o opravdu banální a jednoduchý úkol. Problém však nastává v té chvíli, kdy robot má ujet delší vzdálenost rovně a nezabočit. Zde se již setkáváme s prvním úskalím, a to vyřešit situaci, aby kola byla synchronizována.

Dalším přínosným prvkem tohoto typu zadání je pochopení a skládání mechanických částí tak, aby byly co nejvíce aerodynamické. Na řadu taktéž přichází převodování motoru. Naučit se postavit převodovku tak, aby byla optimálně vyřešena a robot díky ní mohl jet rychleji.

Po stránce programování zde uživatel pochopí a osahá si základní prvky programu, práce s motory a nastavování rychlosti, popřípadě udání určité délky, kterou má robot ujet.

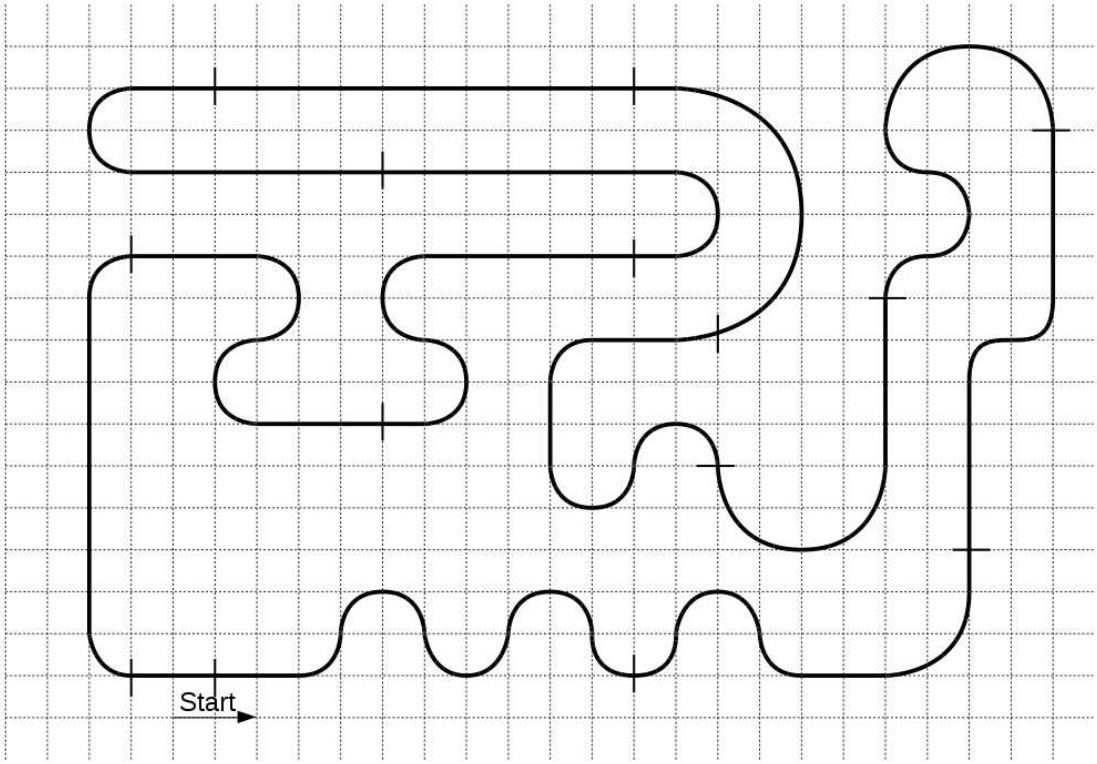
2.2 Line Follower

Nyní přecházíme k mírně komplexnějšímu typu zadání. Předpokládáme, že uživatel již ovládá práci s motory, chápe, jak fungují a je schopen naprogramovat robota do stavu, aby se rozjel.

Zadáním této úlohy je, aby robot následoval čáru na zemi, která mu definuje dráhu, kterou má projet. Na začátek doporučuji, aby se jednalo pouze o rovnou čáru skrze pochopení, jak tento princip funguje. Následně můžeme přejít na složitější obrazce, popřípadě přidávat křižovatky, ostré nebo dokonce hranaté zatáčky a další nástrahy či překážky.

Řešením této úlohy je použití čidla barvy nebo světla. Čidlo se umístí na spodní stranu robota tak, aby snímala podložku. Barva vedoucí linky je vždy černá a okolní podkladová plocha bílá. Díky odrazu robot dokáže rozpoznat barvu a následovat ji.

V této kapitole přidáváme již zmíněné čidlo a naučíme uživatele pracovat s dalším velmi podstatným prvkem. Po pochopení, jak funguje a pracuje toto čidlo, můžeme přejít na variantu s více čidly, kde využíváme prvky PID regulátoru.



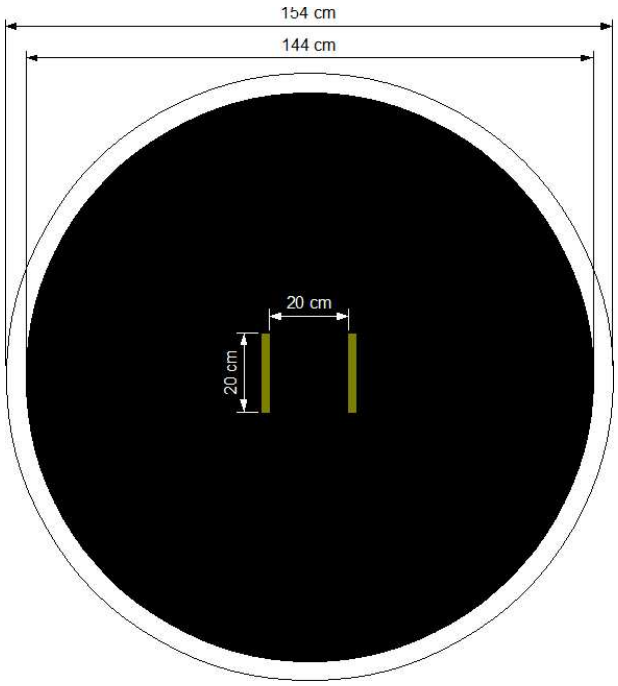
Obrázek 5 - LineFollower [5]

2.2.1 PID regulátor

Při řešení s PID regulátorem můžeme použít více čidel. Nejideálnější řešením jsou tři čidla. Tento regulátor pracuje se třemi složkami: Proporcionální (P), Integrální (I), Diferenciální (D). Proporcionální člen reaguje na aktuální odchylku postavení robota od ideální pozice čáry. Integrální člen koriguje systematické chyby a v neposlední řadě Diferenciální reaguje na rychlost změny odchylky. Následně dochází k ladění. Nastavujeme koeficienty pro P, I, D tak, aby robot sledoval co nejpřesněji čáru a co nejméně se odchyloval od trasy, kterou má následovat.

2.3 Sumo

Nejoblíbenější a zároveň nejpočetněji zastoupená disciplína robotických soutěžích. Vždy i ve výuce nejvíc nadchne a studenti dávají pozor a dávají si obzvláště záležet na propracování robota. Každý z nás si dozajista chtěl postavit svého bojového robota, podobného tomu v amerických filmech. Tady přichází ona možnost.
Hlavní podstatou, jak už z názvu vypovídá, je postavit svého sumo robota a co nejlépe ho odladit, naprogramovat a vybavit ho tak, aby byl ten, který se stane vždy vítězem.
Disciplína je založena na tom, že dva roboti se postaví k sobě zády, po dobu 5 sekund musí v nečinnosti stát a po uplynutí tohoto časového intervalu se otočit nejméně o 90 stupňů, najít svého protivníka a vytlačit jej z kruhu ven.
Zde se dostáváme k problematice po hardwarové části. Robot musí mít pevnou konstrukci, aby nebyl zničen protivníkem. Studenti tedy musí klást důraz na to, aby robot byl dobře postaven.
Na řadu přichází práce s dalším senzorem, a to ultrazvukovým čidlem. Toto čidlo slouží k detekci protivníka a získávání hodnot vzdálenosti. Po zkušenostech s prací s motory a barevným senzorem nám do skládačky zapadá i poslední veledůležitý člen z edice čidel. Tímto krokem student ovládá práci se všemi nejdůležitějšími prvky a je schopen složit a naprogramovat i takto komplexního robota.



Obrázek 6 - SUMO [5]

2.4 Micro Mouse

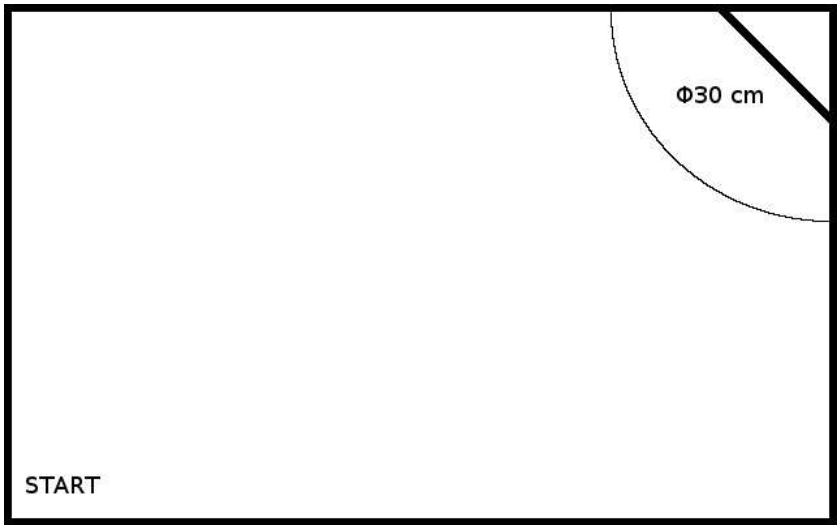
Po zvládnutí práce se všemi hlavními senzory a znalostmi naprogramování autonomního robota se můžeme přesunout na složitější příklady. Jedním z takových je právě micro mouse. Zadáním tohoto úkolu je projet v co nejkratším čase bludiště a dostat se do cíle. Záludnost tohoto úkolu spočívá v tom, že robot už musí mít sofistikovanější program, který zahrnuje složitější algoritmus na prohledávání trasy v prostoru. K tomu cíli může robot používat například schopnost detekce stěn, či pamatování si prostoru, kde již byl nebo jakou zatáčku jakým stylem projel a kam tato cesta vedla, či vedla do slepého zákoutí.

2.5 Robot uklízeč

Posledním typem úkolu, který si projdeme, je Robot uklízeč. Cílem této disciplíny je najít a převézt všechny kostky, které jsou umístěny na ploše, která má předem definované přesné rozměry a je ohraničená černým okrajem na bílém podkladu do vyznačeného úseku. Kostky nebo předměty, které mají být přemístěny, jsou v každém kole rozmístěny náhodně, aby se předešlo stavu, že program bude udělán tak, že pojede po přesně naprogramované trase. Tímto krokem je snaha o to, aby robot byl autonomní a rozhodoval se sám.

Předpoklady pro stavbu a naprogramování robota jsou již komplexnějšího charakteru, kdy robot musí být schopný prohledat prostor, detekovat předměty a přesunout je do určeného cíle.

Pokročilejší variantou je řešení, kdy robot musí kostky přemístit do krabice nebo zvednuté plošiny. Toto řešení vyžaduje složitější mechanické znalosti na postavení konstrukce a naprogramování obslužného softwarového řešení.



Obrázek 7 - Robot uklízeč [5]

3 WEBOTS

Tento robotický simulátor slouží primárně k rychlému a přesnému prototypování a následné simulaci robotů. Jsou zde možnosti robota vymodelovat, naprogramovat a následně pomocí simulace zjistit, zdali je kód v pořádku a jak se robot chová. Webots je považován za nesmírně užitečný nástroj, jelikož pomáhá vývojářům testovat algoritmy a roboty ve virtuálním prostředí s velkou přesností a následně podle simulace a jejich výsledků robota implementovat do reálného světa.
V posledních letech se tento program hojně začal využívat i na poli škol a ve výuce, kde si studenti mohou vyzkoušet sestavit a přivést robota k životu bez nutnosti kupovat drahé stavebnice nebo nutnosti robota fyzicky vlastnit.

3.1 Historie

Historie robotického simulátoru Webots sahá do roku 1996, kdy jej vynalezl a vyvinul pan doktor Olivier Michel ve Švýcarsku ve městě Lusanne [6]. Od roku 1998 je dále vyvíjen a zpravovaný společností Cyberbotics Ltd. V roce 2018 došlo k přechodu na open – source. Webots má širokou komunitu na známých platformách jako jsou například GitHub nebo Discord. Výhoda široké komunity má za následek, že uživatelé mohou rychle najít odpovědi na své otázky anebo sdílet své zkušenosti, což má za cíl být nápomocno pro ostatní uživatele. Cyberbotics taktéž poskytuje oficiální technickou podporu, popřípadě na vyžádání nabízí konzultace či školení [7].

3.2 Rozhraní

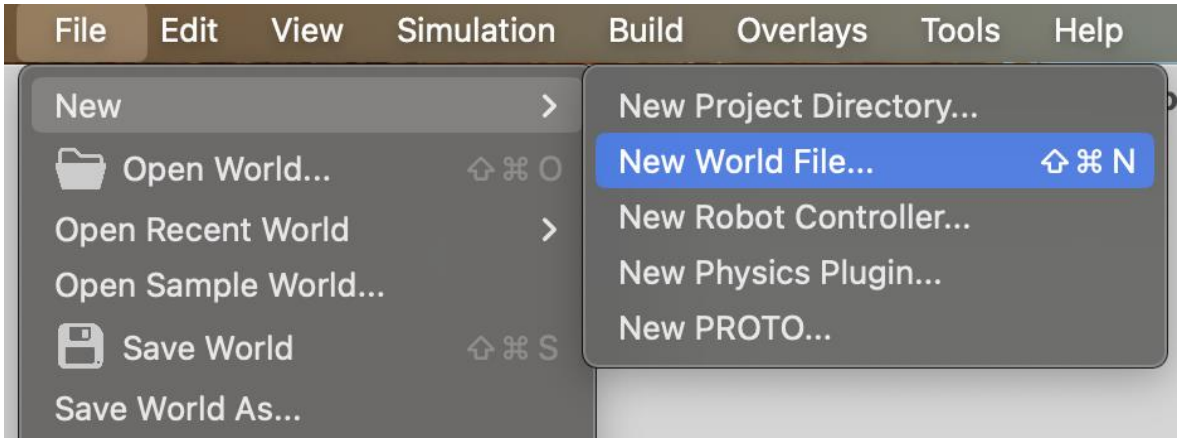
Prvním krokem, který uživatel musí provést, je stažení instalačního balíčku z internetu. Program lze stáhnout na těchto stránkách https://www.cyberbotics.com. Dostupný je pro Windows ve formátu instalačního balíčku .exe, pro operační systém macOS a počítače od firmy Apple v instalačním balíčku .dmg. V neposlední řade i pro platformu Linux. Je zde i možnost stáhnout si starší verzi softwaru [14].

Obrázek 8 - Instalační sada Webots

Po stažení námi vybraného balíčku pro instalaci na námi zvolený a správný operační systém se nám objeví na ploše, popřípadě v domovské nabídce ikona programu, viz. Obrázek 9. Po dvojitém poklikání ikony se aplikace spustí.

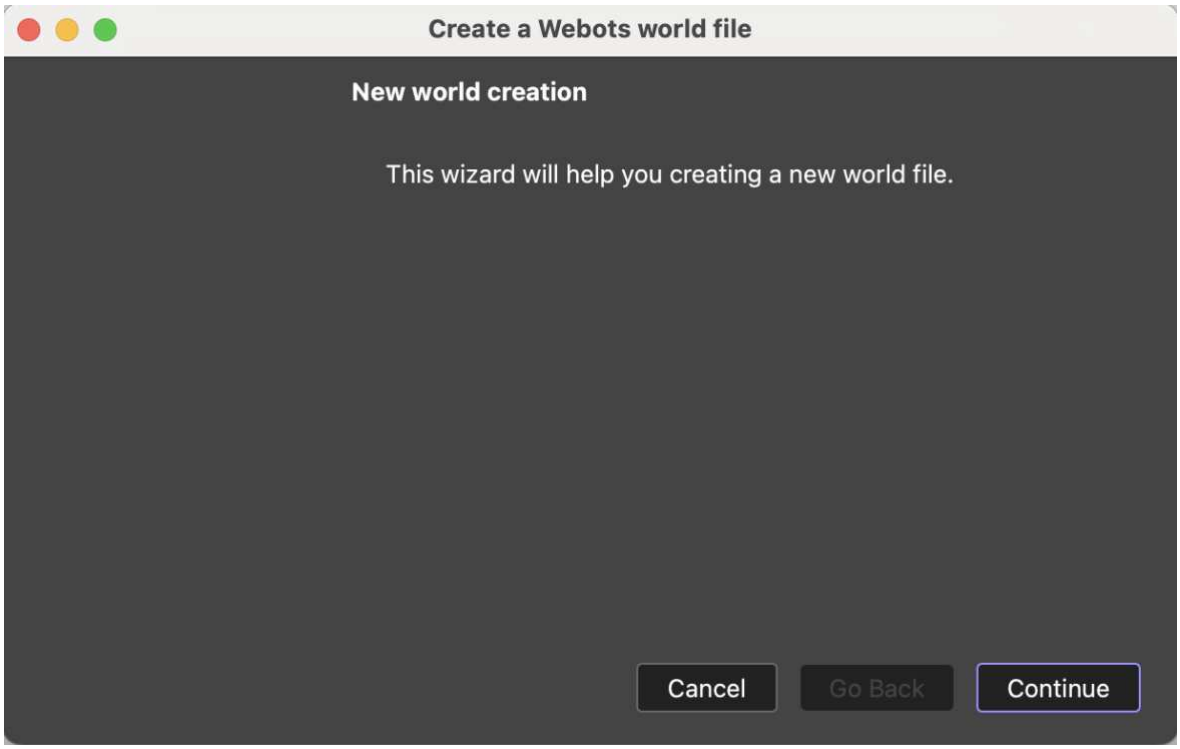
Obrázek 9 - Ikona Webots po instalaci

Po spuštění programu si jako první krok vytvoříme nový svět. Postupovat budeme pomocí lišty, kde v kolonce File přejdeme do New a zde přejdeme na možnost New World File a tuto možnost vybereme. File >> New >> New World File.



Obrázek 10 - Vytvoření nového světa

V následujícím kroku se zobrazí okno, které nás provede úvodním nastavením světa. Tento Wizard je velmi užitečný a přínosný, hlavně při prvních spuštěních programu, kdy je velmi nápomocný.



Obrázek 11 - Vytvoření nového světa – Wizard

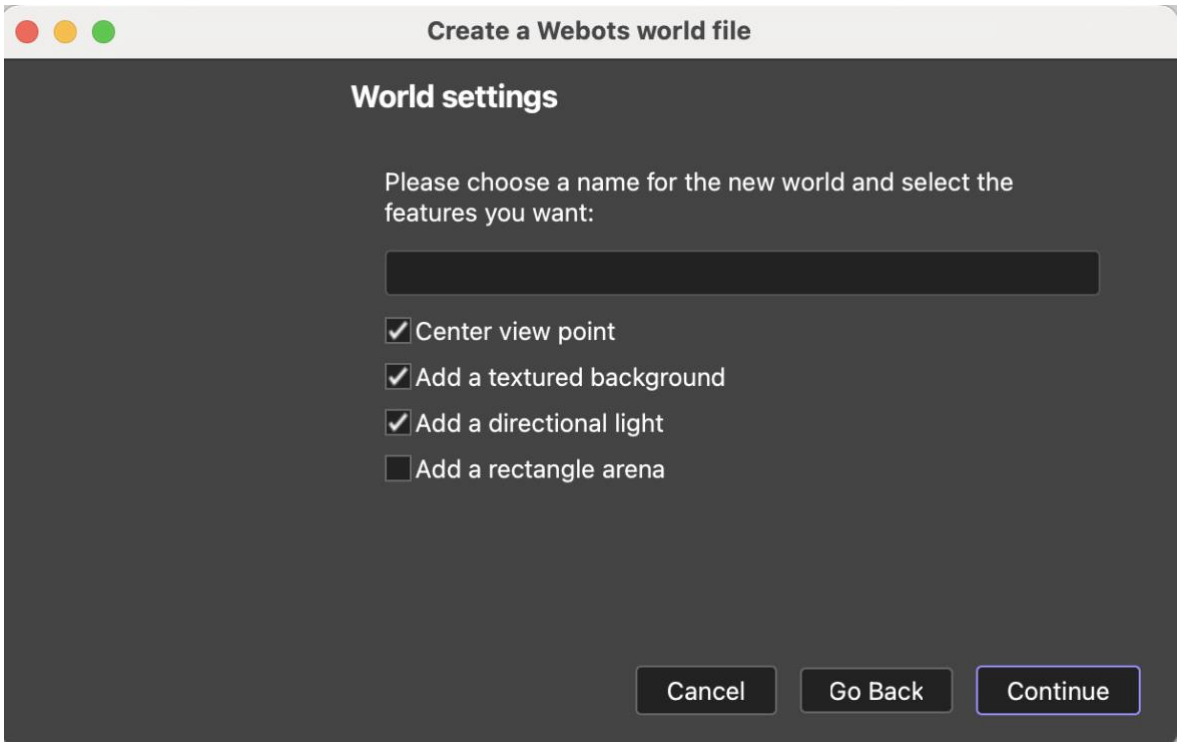
Zde odklikneme tlačítko Continue, které nás přesměruje na další krok vytváření našeho nového projektu.

Následující krok nás přesune k dalšímu nastavení našeho nového světa. Zde si v textovém poli zvolíme název našeho světa, se kterým budeme nadále pracovat. Pod tímto názvem se vytvoří složka v domovské složce s projekty tohoto programu. Pokud budeme chtít projekt otevřít znovu, budeme jej hledat pod tímto názvem.

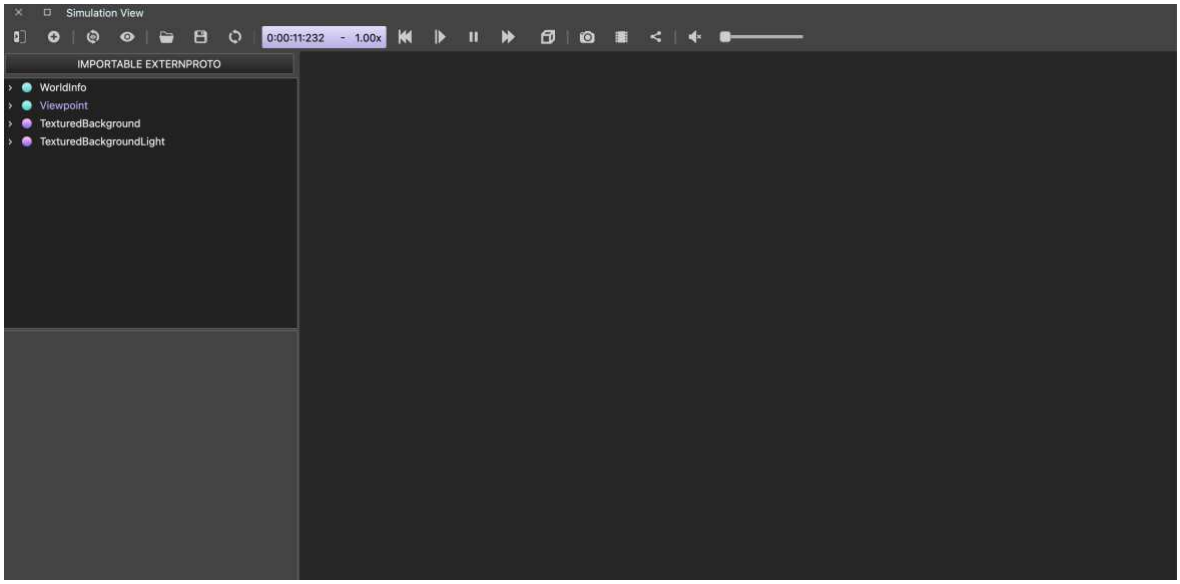
Doporučuji zde používat pouze slova bez háčků, čárek a podobných diakritických znamének a mezer. Použití diakritických znamének, mezer, či speciálních znaků by mohlo vést k problému se sestavením projektu.

Wizard automaticky zatrhne první tři možnosti, kterými jsou Center view point, Add a textured background a Add a directional light.

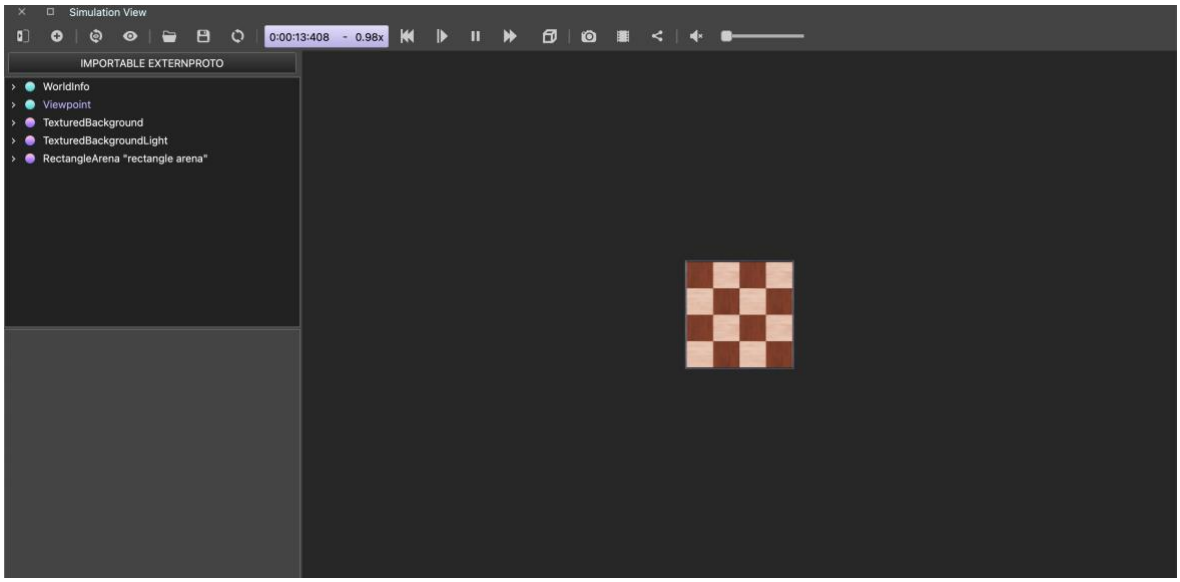
Doporučuji zde zatrhnout a zvolit možnost i poslední čtvrtou možnost Add a rectangle arena [10]. Při zvolení této možnosti se do projektu přidá i již předem vytvořená aréna.



Obrázek 12 - Vytvoření nového světa – Pojmenování a výběr možností



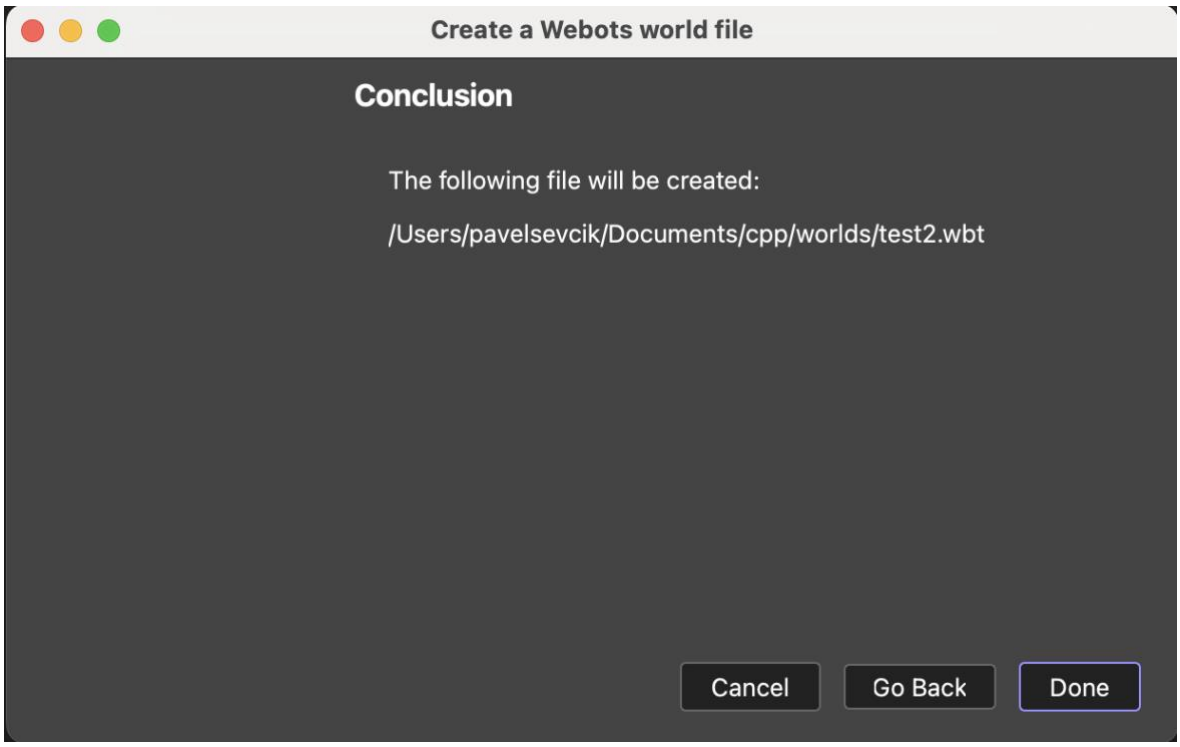
Obrázek 13 - Vytvoření nového světa – Defaultní nastavení bez Add a rectangle arena



Obrázek 14 - Vytvoření nového světa – Defaultní nastavení s „Add a rectangle arena“

Po pojmenování programu, vybrání možností a definování potřebných věcí se dostaneme na poslední dialogové okno, kde se zobrazí místo uložení našeho projektu.

Nyní již stačí pouze odkliknout tlačítko Done a vytvoří se náš první projekt a svět, ve kterém budeme dále pracovat [15].



Obrázek 15 - Vytvoření nového světa – Odkaz na repozitář

Po vytvoření nového světa se zobrazí rozšířené možnosti pro práci a úpravu se scénou, komponentami a dalšími položkami.

Vrchní lišta, která přibude po vytvoření světa se nachází na horní pozici a obsahuje tyto důležité prvky:

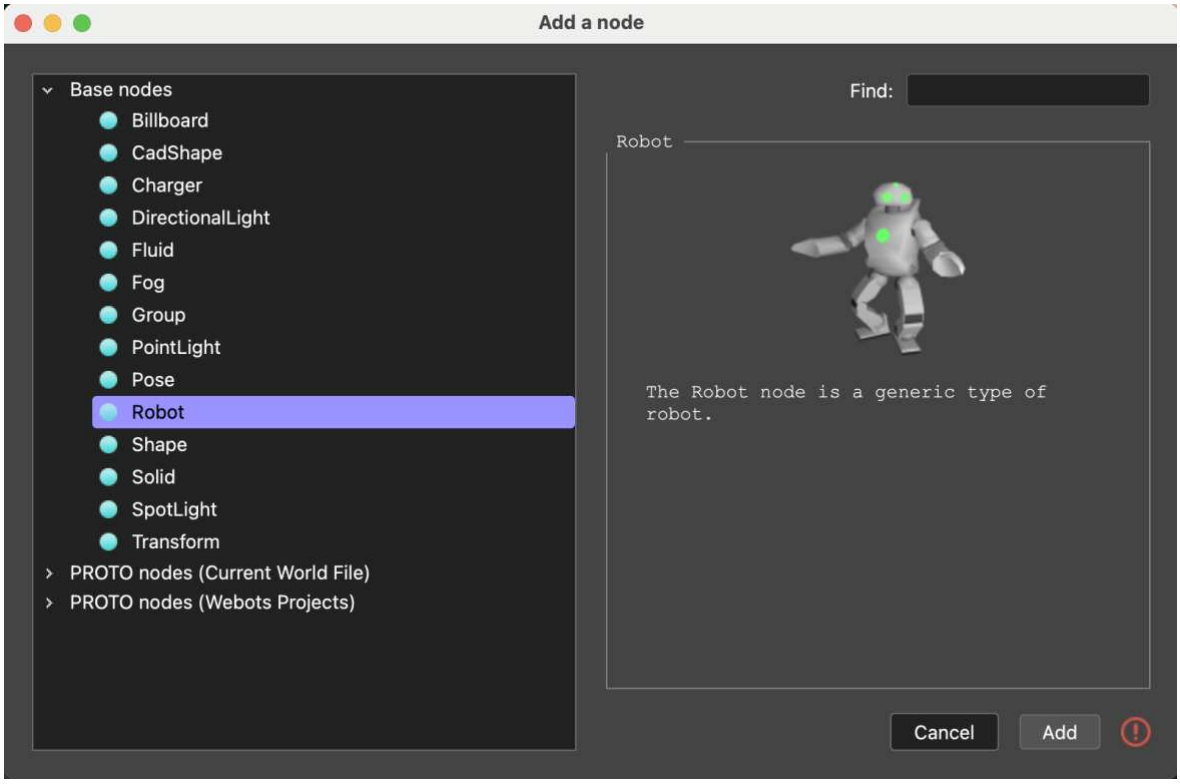
- **Řízení simulace**
  - **Play/Pause**: toto tlačítko slouží ke spuštění nebo pozastavení simulace
  - **Step**: spouštění simulace na jeden krok vpřed
  - **Reset**: slouží k resetování simulace
- **Nástroje pro řízení simulace**
  - **Hodiny**: přepíná simulaci do reálného času, kdy simulace běží synchronizovaně s reálným světem

  - **Simulation speed slider**: nastavování rychlosti simulace

- **Nástroje pro práci s pohledem**
  - **Zoom**: přibližuje nebo oddaluje scénu
  - **Rotate**: dochází k otáčení kamery
  - **Translate**: pohyb kamery v horizontálním směru
- **Nástroje pro práci se scénou**
  - **Select**: výběr objektů ve scéně
  - **Add new**: přidání nových objektů do simulace
- **Nástroje pro práci s objekty**
  - **Move**: přesunutí vybraného objektu
  - **Rotate**: otočení vybraného projektu
- **Možnosti sdílení**
  - **Printscreen**: možnost zachytit snímek obrazovky
  - **Video**: možnost pořídit video záznam obrazovky

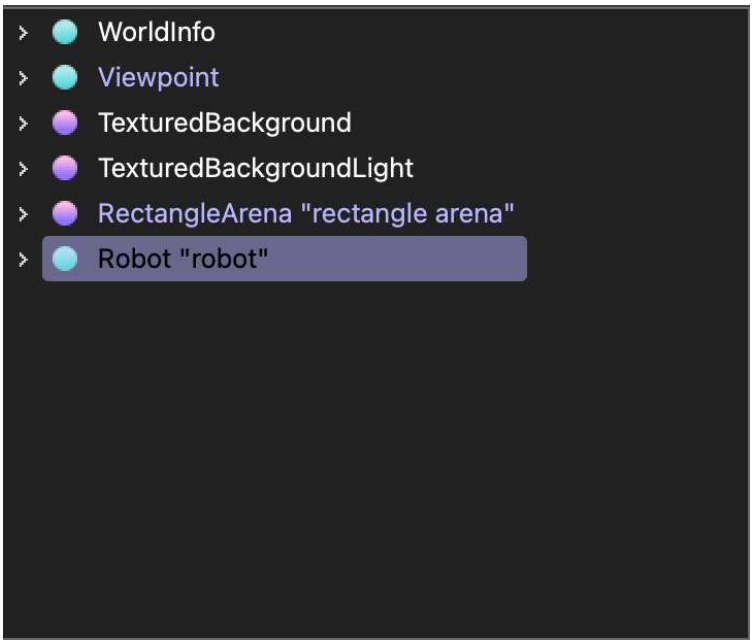
Obrázek 16 - Řídicí lišta

Dalším krokem je přidání robota, nebo různého tvaru. Tohoto docílíme tím, že v horní liště klikneme na tlačítko Add. Poté se zobrazí vyskakovací okno, kde rozbalíme nabídku Base nodes. Zde nám vyjede široká nabídka prvků, kterou můžeme použít, zvolíme si možnost Robot a následně přes tlačítko Add na pravé straně dole přidáme.



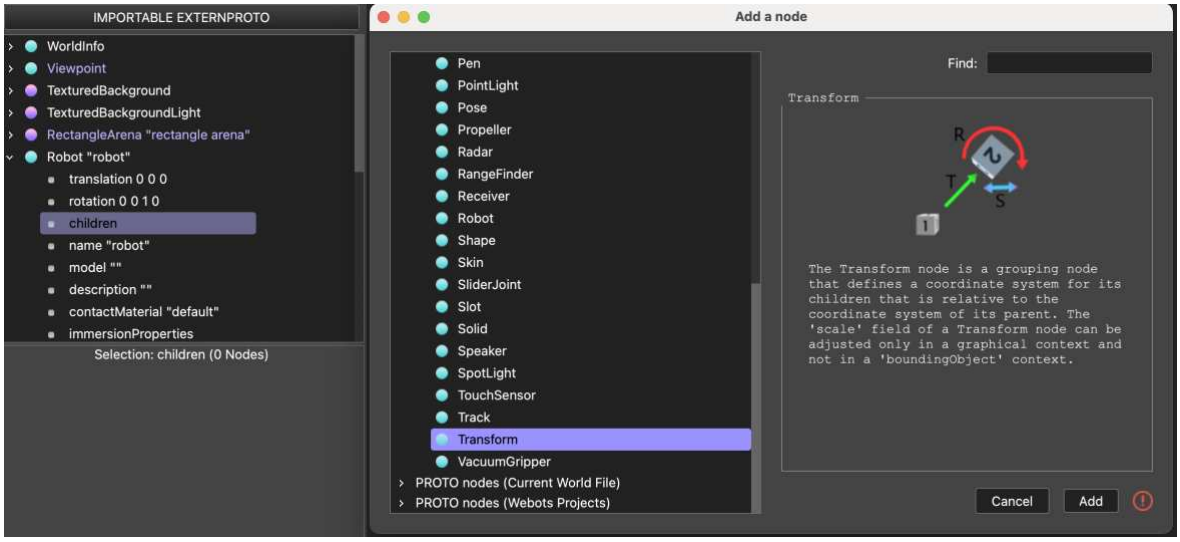
Obrázek 17 - Přidání tvarů

Nyní se nám v poli na levé straně obrazovky přidá nová položka pro editování, jenž je právě přidaný robot [16].

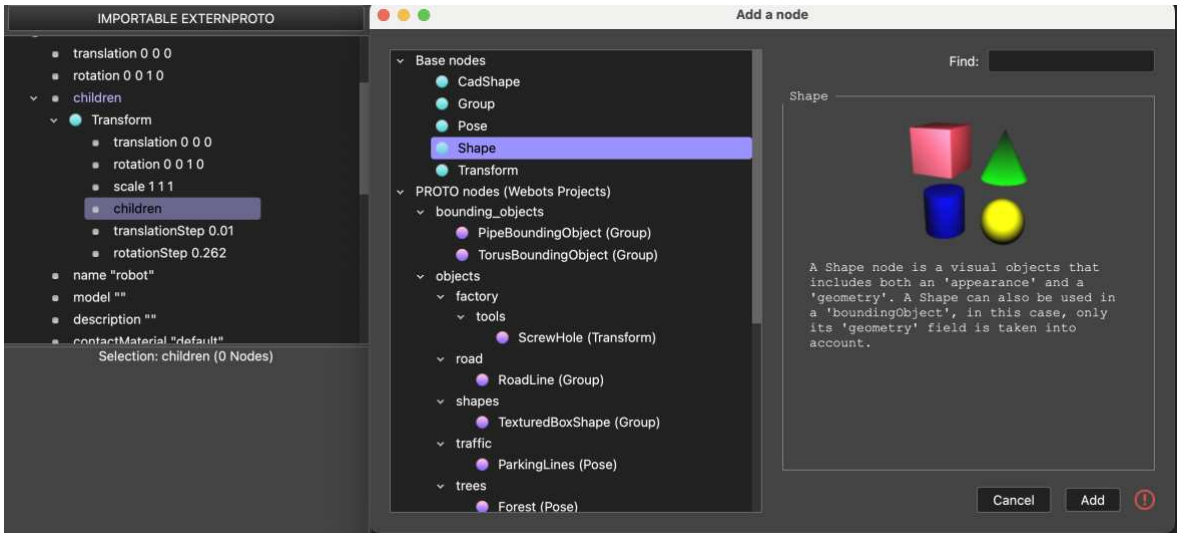


Obrázek 18 - Editace robota

Nyní si vytvoříme základní tělo robota, a to následujícími kroky. Rozklikneme nabídku Robot a v záložce si vybereme pole Children [17]. Po rozkliknuti tohoto pole se nám objeví okno s dalšími možnosti. Zde vybereme možnost Transform a následně tlačítkem Add přidáme do projektu.

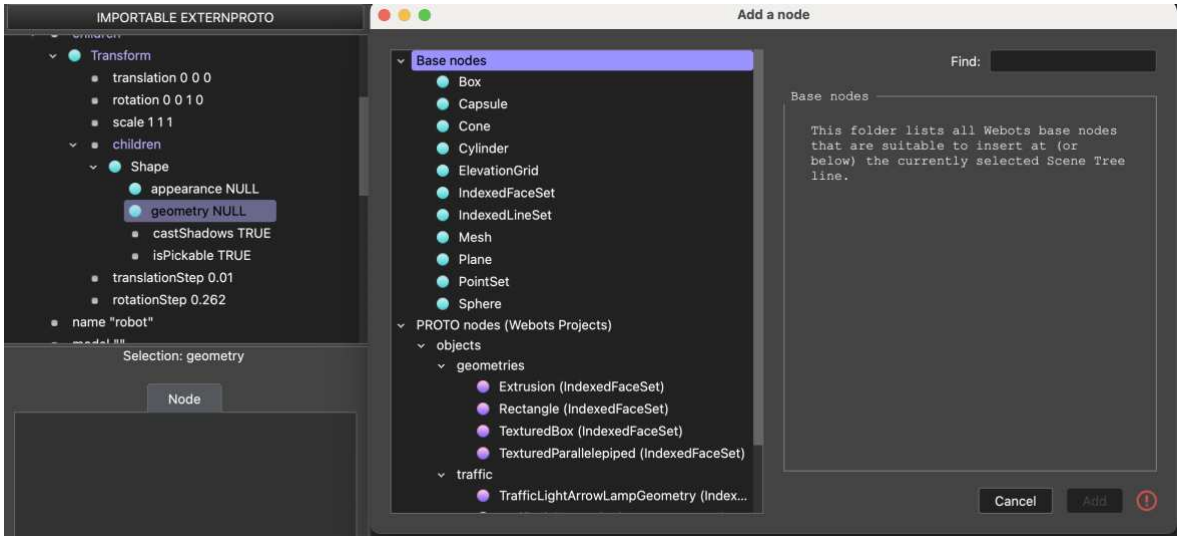


Obrázek 19 – Children



Obrázek 20 - Vložení tvaru

Posledním krokem, který si proberme v obecné teoretické části je výběr tvaru. Po přidání všech předešlých kroků se nám v nabídce dále zobrazí možnost Shape. Rozklikneme tuto nabídku a vybereme dále možnost Geomety Null. Tu dvojklikem otevřeme. Následně se nám zobrazí další okno s možnostmi. Zde vybereme námi požadovaný tvar tělesa a finálně přidáme kliknutím na tlačítko Add. U tohoto tělesa můžeme dále upravovat jeho vlastnosti a atributy.



Obrázek 21- Výběr tvaru

3.3 Platformy

Webots je navržen tak, aby byl co nejvíce kompatibilní s různými druhy operačních systémů. Hlavními platformami jsou Microsoft Windows, macOS a různé Linuxové distribuce. Tato široká škála podpory umožnuje uživatelům a technickým nadšencům používat Webots na většině běžně dostupných počítačových systémů. Tyto kroky vedou a jsou podstatné pro jeho použití v různorodých a širokých kruzích vzdělávacích metod ale i výzkumných prostředcích.

3.3.1 Windows

Na Windows platformě nabízí Webots plnou funkčnost a je velmi dobře optimalizovaný pro běh na této platformě. Konfigurace a instalace software je pro uživatele přívětivě vyřešena, což uživatelům umožnuje snadný přístup ke všem funkcím, kterými Webots disponuje. Stále je Windows velmi často preferován a používán ve školách, či průmyslových areálech, což z Webots dělá ideální volbu.

3.3.2 macOS

Program Webots je taktéž plně kompatibilní i pro nejnovější verze systému macOS, což činí ideální variantu pro prostředí, kde jsou produkty od Applu hojně využívány. Vývojáři Webots se zaměřili i na tuto část trhu, která pokrývá hned po Windows druhou nejrozšířenější příčku v počtu uživatelů. Běh na operačním systému od firmy Apple je hladký a bez nejmenších problémů.

3.3.3 Linux

Uživatelé Linuxu těží hlavně z otevřenosti a flexibilnosti, kterou Webots na této platformě nabízí. Linux je velmi často volbou a je preferován ve výzkumných institucích, či u programátorů pro jeho schopnost přizpůsobení a vlastní konfigurace, což umožnuje hlubší integraci s vlastními nástroji a aplikacemi.

3.4 Výhody

Jednou z hlavních výhod tohoto simulátoru je jeho možnost používat jej na všech hlavních platformách, jak na Windows, macOS, tak i na Linuxu. Činí to z programu velmi flexibilní pracovní nástroj, kdy může být používán velmi různorodou skupinou, popřípadě kdekoliv na školách nezávisle na platformě operačního systému, kterou uživatelé v dané instituci využívají. Další výhodou je ze začátku přívětivé grafické rozhraní a intuitivní ovládání u méně pokročilých projektů. Navigace se jeví jako logická a umožnuje rychlou a snadnou ovladatelnost hlavní pracovní plochy. Velmi bych ocenil širokou a velmi rozsáhlou knihovnu předem definovaných typů a různorodých konfigurací modelů, které se dají použít. At se jedná o základní tvary bloků, které jsou ideální a dostačující pro začátky a první sestavení robota a tím i vizualizaci toho, jak daný robot funguje a pro výukové materiály jsou dostačující, tak i následné pokročilé možnosti a konfigurace. Dalším obrovským plusem je široká možnost využití programovacích jazyků, jakožto jsou C, C++, Python, nebo Java. Tímto krokem je možnost programování robota velmi flexibilní a otvírá více prostoru větší skupině programátorů, než pokud by program byl upnut pouze na jeden programovací jazyk. Dále také za zmínku stojí široká komunita uživatelů, která může být velmi nápomocná v různých situacích, kdy si uživatel neví rady a může tedy vyhledat pomoc.

3.5 Nevýhody

I u tohoto velmi dobrého vývojového simulátoru narazíme na několik drobností, které stojí za to vytknout. Jednou z nich může být náročnost na výpočet. Pokud se jedná o výpočty náročnějších simulačních scén, modelů s vysokým počtem robotů, může být uživatel omezen hardwarovým vybavením, což může vést ke zpomalení simulace. Ve výhodách jsem zmínil intuitivní ovládání, ovšem pokud se pustíme do rozsáhlejších projektů, může docházet ke zmatečným situacím s velkým množstvím různorodých situací, hlavně u nových uživatelů.

4 COPPELIASIM

Robotický simulátor Coppeliasim je robotický simulátor, který slouží k modelování, vizualizaci a simulaci robotických systémů. Jedná se o poměrně uživatelsky přívětivé rozhraní, které nabízí širokou funkcionalitu, která umožnuje inženýrům, vývojářům nebo učitelům či studentům zkoumat a optimalizovat chování robotů v různorodých podmínkách nebo prostředích [11].

4.1 Historie

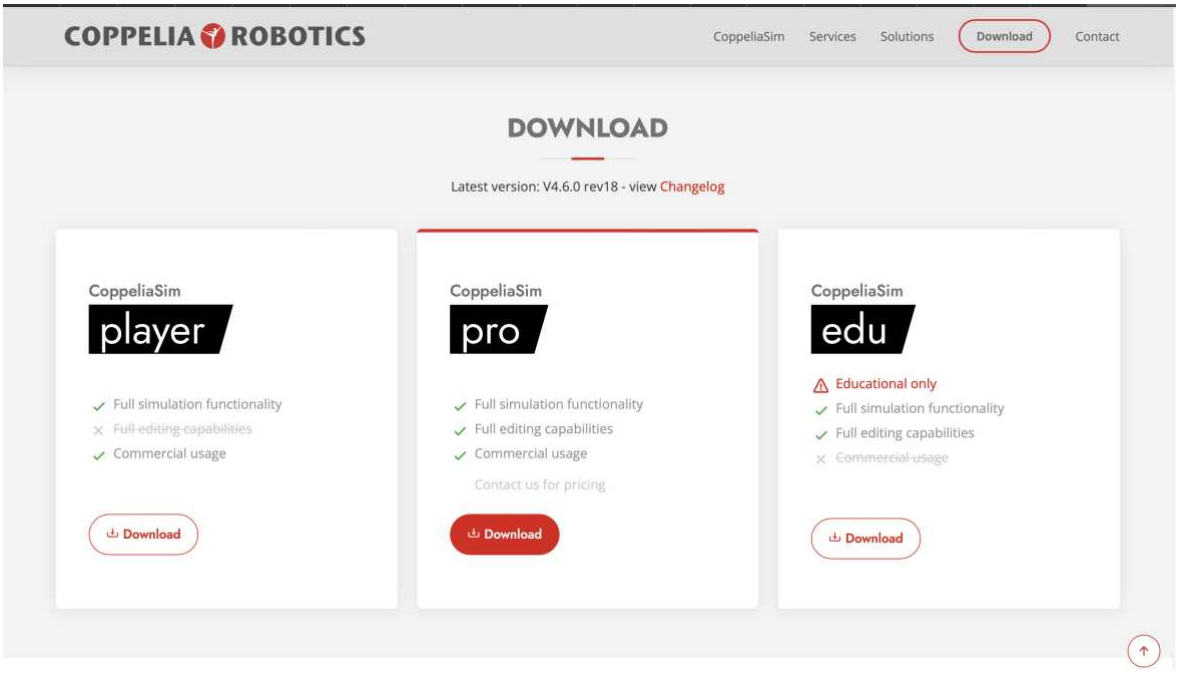
Tento robotický simulátor vznikl jako výsledek dlouhodobého zkoumání a inovací v oblasti robotiky. Za jeho původem a vznikem stojí laboratoř Coppelia Robotics, kde se výzkumníci a pracovníci věnují vývoji pokročilých technologií v oblasti automatizace a robotiky. Jeden z hlavních zakládajících členů a jeho tvůrců byl pan profesor doktor Marc Freese [12], který taktéž byl zakladatelem společnosti Coppelia Robotics. V letech na univerzitě se zaměřoval na vývoj nástrojů určených pro simulaci a testování robotických systémů. Jeho obrovskou snahou bylo vytvořit inovativní a komplexní prostředí, které by bylo nápomocné a umožnovalo detailněji modelovat a vytvářet chování robotů a provádět simulace.

Píše se rok 2008 a přichází velký milník. V tomto roce byla uvedena první verze tohoto robotického simulátoru na trh. Tato první verze obsahovala funkce a možnosti pro modelování a simulaci robotů a systémů. Postupem času přicházely postupné aktualizace a vylepšení, která navazovala a přizpůsobovala se pokrokům v oblasti robotiky a dalším a dalším požadavkům uživatelů. Tyto kroky přispívali k dalšímu rozvoji. Dalším obrovským milníkem a přelomovým krokem bylo připojení a integrace pro další programovací jazyky jakožto například jsou: Python, C, C++. Tento krok umožnil uživatelům flexibilnější přístup k simulacím a do velké míry pomohl k expanzi tohoto programu. Verze, kterou známe dnes, je výsledkem mnoha a mnoha let výzkumu a různých inovací. Poskytuje široké možnosti a funkce pro modelování a simulaci robotických systémů.

4.2 Rozhraní

Před začátkem používání tohoto robotického simulátoru musí uživatel provést pár kroků. Prvním krokem uživatele je přechod na webové stránky Coppelia Robotics, kde nalezne instalační balíčky potřebné pro instalaci programu. Jejich webové stánky se nachází na této webové adrese: https://www.coppeliarobotics.com .

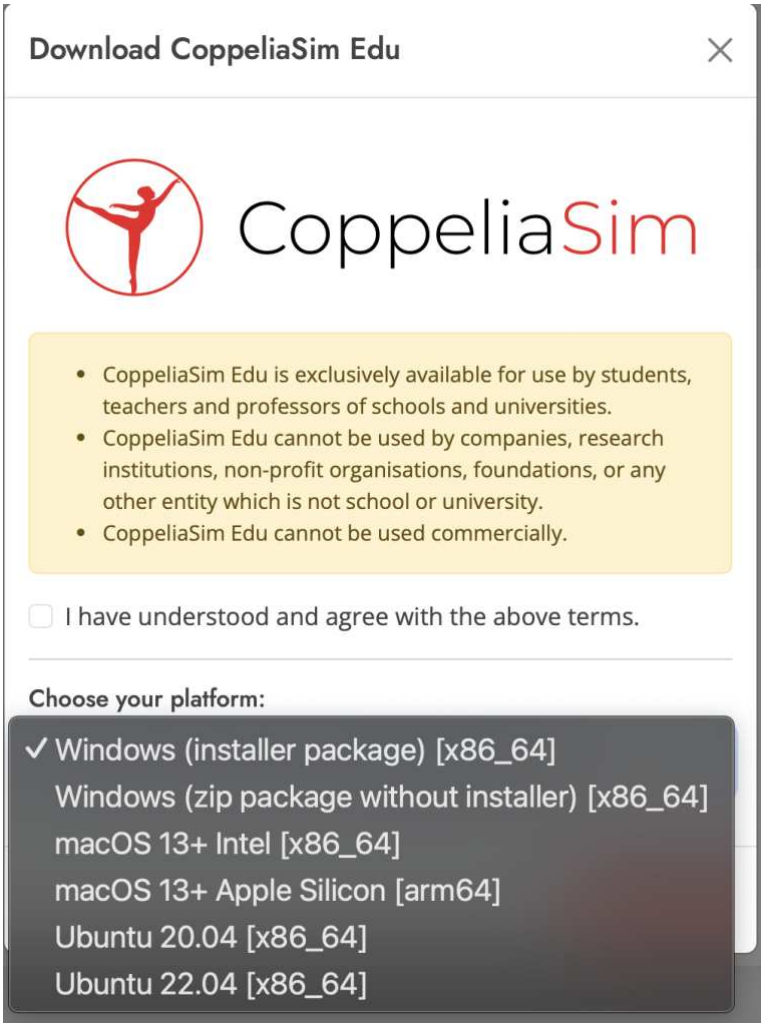
Zde si uživatel v horní liště vybere možnost Download. Po kliknutí na toto pole jej stránka přesměruje na možnosti výběru verzí pro stáhnutí.



Obrázek 22 - CoppeliaSim – Instalační sada

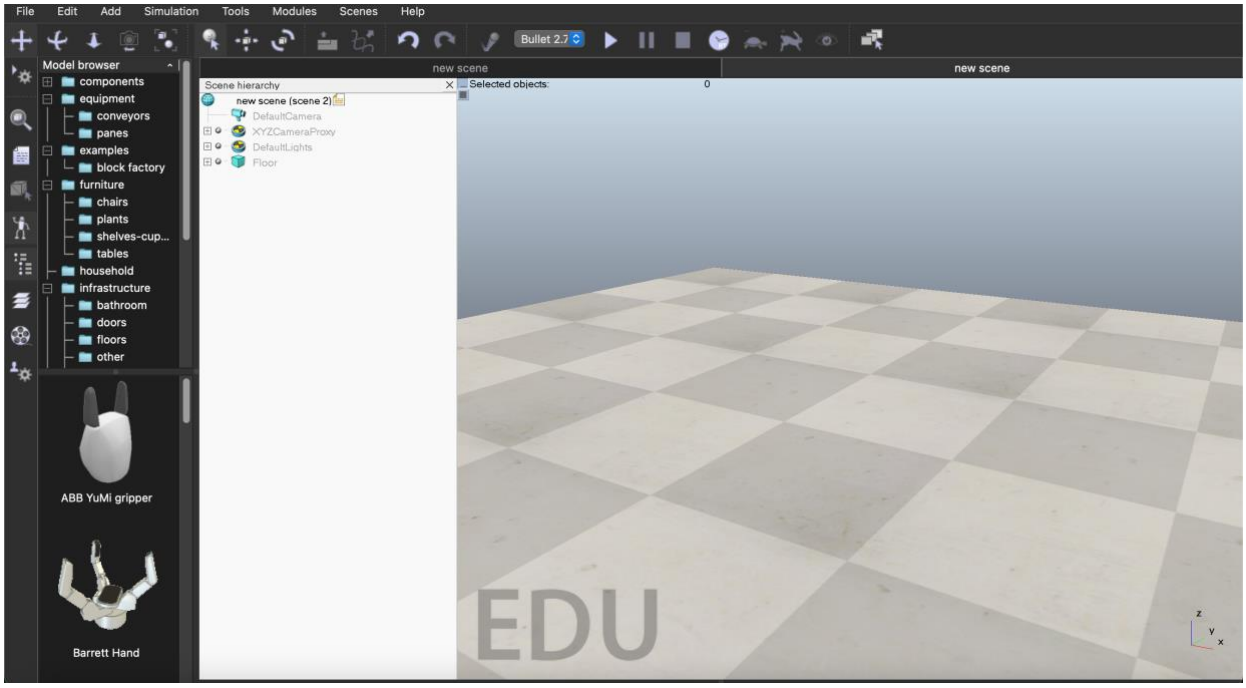
Zde se nám objeví několik různých variant a možností pro stažení. V našem případě sáhneme po variantě edu, která je zdarma a nabízí většinu obsahu. Tuto variantu ovšem nelze používat ke komerčním účelům a je určena pouze pro edukační účely.

Dalším krokem k nainstalování programu je výběr operačního systému. Aplikace Coppeliasim je multiplatformní a tím pádem podporuje více operačních systému.
Z podporovaných operačních systému jsou na výběr Microsoft, kde si můžete zvolit, zdali si stáhneme přímo instalační soubor, nebo .Zip balíček. Další podporovanou platformou je macOS, zde je na výběr z varianty nových čipů, které přímo vyrábí Apple pod označením M1 a vyšší ve variantě instalačního souboru Apple Silicona, nebo pro starší varianty MacBooku ještě s čipy od firmy Intel s označením instalačního balíčku Intel. V neposlední řade zde má zastoupení i Linux ve formě pro Ubuntu.



Obrázek 23 - Výběr platformy

Po úspěšném nainstalování aplikace a jejím prvním otevření se před námi objeví přímo pracovní plocha. Tato pracovní plocha je prázdná a můžeme na ní přidávat libovolné předměty z knihovny, kterou nabízí přímo aplikace. Velikost plochy můžeme různě upravovat, měnit její vzhled a podobné atributy. V následujících krocích si popíšeme další funkcionality programu a jak s nimi zacházet.



Obrázek 24 - CoppeliaSim – prostředí

Hlavní programová lišta nabízí nepřeberné množství nastavení, úprav, detailů a různých technik. Námi nejpoužívanější věci si nyní popíšeme [13].

Záložka File obsahuje základní uživatelské nastavení jakožto je Nová scéna, Otevření scény, Načtení modelu, Zavření scény, Uložení scény, Import, Export a vypnutí programu. Tyto položky budeme využívat poměrně často.

Záložka Add je taktéž velmi důležitá a nepostradatelná součást práce v tomto simulátoru. Obsahuje seznam dalších součástek, tvarů a těles, které budeme přidávat do scény kromě vybraných hotových modelů, které se nachází v levé části hlavní scény. V této záložce se primárně nachází základní tvary těles, které slouží k modelování pokročilejších robotů.

Záložka Simulation slouží primárně k práci a ovládání simulace.

V záložce Tools stojí za zmínku hlavně kolonka Scripts, která se využívá k psaní prgramů pro jednotlivé roboty.

Záložka Scenes slouží k přepínání scén, pokud máme otevřených více různých scén. Tímto můžeme jednoduše přepínat mezi jednotlivými scénami a nedochází ke ztracení.

Poslední záložka Help slouží k pomoci uživateli. Zde bych vyzdvihnul kolonku User manual, která uživatele přesměruje na webové stránky výrobce a zobrazí kompletní manuál a dokumentaci, jak pracovat, pokud si uživatel neví rady. Další velmi podstatnou a

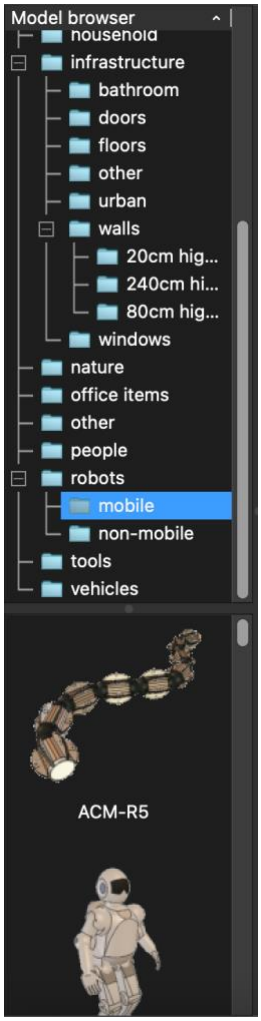
užitečnou vlastností, která se zde nachazí je kolonka Debug, která slouží k postupnému stepování programu, pokud uživatel hledá chybu.

Obrázek 25 - CoppeliaSim – Hlavní lišta

Ovládací lišta disponuje různorodou paletou ovládacích prvků. První set komponent je zaměřený na práci s pohledem kamery, jako je například její rotace, posunutí dopředu/ dozadu nebo zaměření na objekt, nebo práce s úhly. Další sada prvků pracuje s objekty. Zde je možnost vybrat objekt, přesouvat objekt na danou pozici, nebo s objektem rotovat. Další sadou jsou kroky dozadu a dopředu. Tyto kroky jsou velmi důležité, pokud uděláme chybu a chceme se vrátit o krok zpět nebo naopak. Poslední sadou je soubor komponentů pro práci s plynutím času při simulaci. Jsou zde komponenty na puštění simulace, zastavení, vypnutí, zpomalení, či zrychlení.

Obrázek 26 - Ovládací lišta

Další komponentou programu je Model browser. Tento seznam modelů se nachází po levé straně hlavní scény. Obsahuje nespočetné množství již předem definovaných modelů, které můžeme využít a rovnou přidat do svého projektu. Nachází se zde například přímo kompletní modely robotické ruky od renomovaných robotických firem, zabývající se jejich výrobou, vozítka určeny k naprogramování, nebo roboti pro mladší uživatele, které můžeme rozchodit a plnit s nimi úkoly, které zaujmou i děti. Dále se zde nachází například soubor zdí, které lze využít pro stavení bludišť, či budov a i takové detaily, jako jsou okna, dveře nebo jim podobné komponenty.



Obrázek 27 - Model browser

4.3 Platforma

Robotický simulátor CoppeliaSim je velmi adaptabilní a multiplatformní nástroj. Je navržen tak, aby byl kompatibilní s co největší paletou operačních systémů, které se dnes využívají. Díky těmto krokům je zajištěno, že je simulátor dostupný širokému spektru uživatelů.

4.3.1 Windows

CoppeliaSim je plně kompatibilní s operačním systémem Windows, a to i pro již starší a nyní již nepodporované verze jako je velmi oblíbený a dosud hojně využívaný Windows 7. Podporu má taktéž u novějších verzí jako je Windows 10, nebo nejnovější distribuce Windows 11.

4.3.2 macOS

Podporu má taktéž zajištěnou u operačních systému běžících na počítačích od firmy Apple s operačním systémem macOS. Zde je zajištěna podpora pro obě stále běžící architektury a jak pro nyní již nově nevyráběné MacBooky s architekturou a čipem od firmy Intel, tak i pro nejnovější výpočetní techniku s čipy pod označením M1 a vyššími řadami, které si Apple vyvíjí sám a přizpůsobuje pro ně svůj operační systém macOS.

4.3.3 Linux

CoppeliaSim běží a má podporu taktéž pro operační systémy Linux. Například pro verzi Debian. Linux není mezi širokou veřejností tak hojně využíván, jako dva předem zmíněné operační systémy, ale i přes to se rozhodli vývojáři této aplikace podporovat a stále vydávat nejnovější verze tohoto programu i pro tuto platformu. Linux se těší velké popularitě skrze jeho možné modifikace spíše u vývojářů, či programátorů a přesně na tuto skupinu je cílená verze pro Linux.

4.4 Výhody

Jednou z hlavních výhod a předností robotického simulátoru CoppeliaSim je jeho možnost a dovednost být použitelný napříč všemi hlavními operačními systémy. Tento fakt, že je aplikace multiplatformní umožnuje používat tento simulátor širšímu spektru uživatelů. Další nespornou výhodou je možnost využití široké škály programovacích jazyků. Mezi podporované programovací jazyky patří například: Python a C++. Další výhodou je použitelnost API pro programování a manipulaci s roboty. Podpora fyzikálních enginů je

taktéž skvělým faktorem, který je v tomto programu obsažen. Velmi milým a příjemným faktem je taktéž to, že vydavatelé nabízejí jako jednu ze stažitelných variant variantu Edu, která je určena pro výukové účely a je zdarma.

4.5 Nevýhody

Jako nevýhodu zde vidím rozsáhlost uživatelského prostředí, které se mi nejeví tolik adaptivní hlavně pro nové uživatele. Komplexnost rozhraní může mít za následky to, že se uživatel lehce ztratí a je následně zmaten. Požadavky na výkon hardwaru mohou být další nevýhodou, a to hlavně u větších nebo velmi rozsáhlých projektů na zařízeních, které nedisponují příliš velkým výkonem a jsou hardwarově omezeny, zde může dojít k problému s výpočetní kapacitou a simulace se můžou protahovat a zpomalovat při použití výpočtů složitějších fyzikálních interakcí.

5 POROVNÁNÍ A VÝBĚR SIMULÁTORU

Při výběru ideálního simulátoru, který bude použit pro výuku v hodinách robotiky Fakultě Aplikované Informatiky a v přidružených kroužcích, které se taktéž pořádají na území FAI, jsem se snažil vycházet z několika důležitých kritérií. Prvním z nich byla uživatelská přívětivost. Následujícími faktory byli například podpora programovacích jazyků, celková možnost integrace, dostupnost a přívětivost simulátoru.

Webots nabízí velmi intuitivní grafické uživatelské rozhraní a skvělou dokumentaci, tyto aspekty dokážou usnadnit novým uživatelům tohoto simulátoru integraci do procesu a celkové porozumění systému. Uživatelské rozhraní mi přijde relativně velmi dobře organizované a uzpůsobené. U simulátoru CoppeliaSim mi již uživatelské prostředí přijde o něco chaotičtější ve srovnání s Webots. Pro nové uživatele může být tento fakt do jisté míry odrazujícím pro první zkušenosti a začátky se simulátorem.

Je pravdou, že oba simulátory nabízejí relativně velmi dobře řešenou dokumentaci a uživatelskou komunitu, která je velmi široká a uživatelsky přívětivá. Webots ovšem disponuje širší uživatelskou základnou a tento fakt vede k tomu, že na internetu lze dohledat větší škála návodů, doporučení, tipů, či fórum, kde, pokud si neví rady, tak může nalézt odpověd na své otázky, či problémy.

Z hlediska programovacích jazyků oba tyto simulátory obstály na výbornou. V obou případech lze použít vícero programovacích jazyků a tím pádem se naskýtá větší příležitost pro oslovení širšího spektra programátorů. Mezi těmito jazyky se nachází masově rozšířené a populární jazyky, jakožto jsou C/C++ a Python,

Kde má ovšem CoppeliaSim mírně nad simulátorem Webots navrch je v možnosti integrace s externími aplikacemi. I Webots sice poskytuje dostatečnou možnost pro většinu aplikací, ovšem CoppeliaSim je na tom lépe. Jenže tento fakt nám není natolik přínosný do naší robotické výuky na univerzitě, z toho důvodu, že úlohy, které studenti vytváří jsou více zaměřeny na jiné typy úloh, jakožto jsou například robotické závody a jim podobné soutěže.

Těmito všemi aspekty a zpracováním, jsem vybral jakožto vítěze mezi těmito robotickými simulátory právě Webots, jelikož se jeví lépe pro účely základního vzdělávání a výuky jak v robotickém kroužku, tak i pro studenty na vysoké škole. Webots se jeví jako vhodnější možnost díky jeho jednoduššímu přístupu, široké škále vzdělávacích materiálu a mnohých dalších, což má za následek a může umožnovat rychlejší a méně komplikovaný vstup na pole robotiky.

II. PRAKTICKÁ ČÁST

6 ROBO SUMO

6.1 Zadání

Vymodelujte a naprogramujte robota, který je sestrojen a obstojí v robotické disciplíně ROBOSUMO.

Zadáním této disciplíny je, aby Váš robot vytlačil druhého soupeřova robota z kruhu dříve než on Vás. Při začátku soutěže musí roboti stát k sobě otočení zády a každý z robotů musí stát na své polovině hracího kruhu. Po odstartování soutěže musí robot nečinně stát po dobu alespoň 5 sekund a až po uplynutí tohoto časového intervalu se robot smí rozjet. Po rozjetí musí absolvovat otočku alespoň o 90 stupňů a následně smí vyjet vstříc svému soupeři. Kolo může trvat maximálně 3 minuty. Vyhrává robot, který vytlačí svého soupeře z ringu jako první.

6.2 Program

6.2.1 Definice hlaviček a konstant

V prvním kroku je potřeba importování potřebných knihoven a definovaní konstant, které budeme používat v programu. Využijeme knihovny „Robot“, „Motor“ , „DistanceSensor“ , „ColorSensor“ , které nám umožní práci s robotem, jeho motory a senzory. Dále použijeme konstanty „TIME_STEP“, „MAX_SPEED“, „WAIT_TIME“ , „TURN_TIME“ , „DISTANCE_THRESHOLD“ a „LINE_THRESHOLD“ , které definují základní parametry a vlastnosti robota, které budeme potřebovat. Tyto vlastnosti jsou například: čekání, maximální rychlost robota, či vzdálenost, na kterou robot bude útočit na protivníka.

```
1. #include <webots/Robot.hpp>
2. #include <webots/Motor.hpp>
3. #include <webots/DistanceSensor.hpp>
4. #include <webots/ColorSensor.hpp>
5. using namespace webots;
6. #define TIME_STEP 64
7. #define MAX_SPEED 6.28
8. #define WAIT_TIME 5000 // 5 sekund v milisekundách
9. #define TURN_TIME 1570 // Čas na otočení o 90 stupňů v
   milisekundách
10.     #define DISTANCE_THRESHOLD 500 // Práh pro detekci
   druhého robota (v milimetrech)
11.     #define LINE_THRESHOLD 800 // Práh pro detekci bílé
   čáry
```

6.2.2 Deklarace třídy „MyRobot“

V této části deklarujeme již zmíněnou třídu MyRobot, která dědí z třídy Robot, která je poskytovaná WeBots. Třída obsahuje členské proměnné pro motory a senzory. Dochází zde k aktivaci senzorů a dalších komponentů na našem robotu.

```
1. class MyRobot : public Robot {
2. private:
3.     Motor *leftMotor, *rightMotor;
4.     DistanceSensor *frontSensor, *leftSensor, *rightSensor;
5.     ColorSensor *lineSensor;
6. public:
7.     MyRobot() {
8.         leftMotor = getMotor("left wheel motor");
9.         rightMotor = getMotor("right wheel motor");
10.             frontSensor = getDistanceSensor("front
   sensor");
11.             leftSensor = getDistanceSensor("left sensor");
12.             rightSensor = getDistanceSensor("right
   sensor");
13.             lineSensor = getColorSensor("line sensor");
14.             frontSensor->enable(TIME_STEP);
15.             leftSensor->enable(TIME_STEP);
16.             rightSensor->enable(TIME_STEP);
17.             lineSensor->enable(TIME_STEP);
18.         }
```

6.2.3 Otáčení o 90 stupňů

Jak jsme již uvedli v první části práce, robot po startu musí provést otočku alespoň o 90 stupňů. Tato funkce slouží k vykonání tohoto úkolu. Dochází k nastavení pozice obou motorů tak, aby se jedno kolo točilo dopředu a druhé dozadu, což vede k rotaci robota

```
1. void turn90Degrees() {
2.     leftMotor->setPosition(TURN_TIME);
3.     rightMotor->setPosition(-TURN_TIME);
4.     while (step(TIME_STEP) != -1) {
5.         if (leftMotor->getPosition() == 0 && rightMotor-
   >getPosition() == 0)
6.             break;
7.     }
8. }
```

6.2.4 Hlavní smyčka

V hlavní smyčce robot provádí sérii akcí a příkazů. Počáteční určená doba 5 sekund určuje robotu, že zůstane statický do začátku soutěže. Po uplynutí této doby se robot otočí o definovaný úhel a začíná vyhledávat svého nepřítele. Pokud senzory detekují nepřítele na definovanou vzdálenost, robot se s maximální rychlostí rozjede proti němu a snaží se jej eliminovat vytlačením z kruhu. Pokud robot nepřítele nevidí, pokračuje v prohledávání prostředí. V případě, že narazí na bílý okraj kruhu, couvne a otočí se.

```
1. void run() {
2.     int start_time = getTime();
3.     bool found_enemy = false;
4.     while (step(TIME_STEP) != -1) {
5.         if (getTime() - start_time < WAIT_TIME) {
6.             // Čekání na začátek soutěže
7.             continue;
8.         } else if (!found_enemy) {
9.             // Otočení o 90 stupňů po uplynutí 5 sekund
10.                 turn90Degrees();
11.                 found_enemy = true;
12.             } else {
13.                 double leftSpeed = MAX_SPEED;
14.                 double rightSpeed = MAX_SPEED;
15.                 // Detekce druhého robota pomocí
   ultrazvukových senzorů
16.                 if (frontSensor->getValue() <
   DISTANCE_THRESHOLD ||
17.                     leftSensor->getValue() <
   DISTANCE_THRESHOLD ||
18.                     rightSensor->getValue() <
   DISTANCE_THRESHOLD) {
19.                     // Robot na dosah, zaútočit
20.                     leftSpeed = MAX_SPEED;
21.                     rightSpeed = MAX_SPEED;
22.                 } else {
23.                     // Pokračovat v hledání
24.                     leftSpeed = MAX_SPEED / 2; // Nižší
   rychlost pro pomalejší pohyb
25.                     rightSpeed = MAX_SPEED / 2;
26.                 }
27.                 // Detekce bílé čáry
28.                 if (lineSensor->getValue() >
   LINE_THRESHOLD) {
29.                     leftSpeed = -MAX_SPEED; // Couvání
30.                     rightSpeed = -MAX_SPEED;
31.                     // Otočení o 90 stupňů
32.                     turn90Degrees();
33.                     // Resetování proměnné pro nalezení
   nepřítele
34.                     found_enemy = false;
35.                 }
36.                 // Nastavení rychlostí motorů
37.                 leftMotor->setVelocity(leftSpeed);
38.                 rightMotor->setVelocity(rightSpeed);
39.             }
40.         }
41.     }
```

6.2.5 Funkce Main

Main je vstupním bodem programu. Vytváří instanci třídy MyRobot a spouští její hlavní smyčku run.

```
1. int main() {
2.     MyRobot robot;
3.     robot.run();
4.     return 0;
5. }
```

6.2.6 Kompletní kod v C++

```
1. #include <webots/Robot.hpp> // Zahrnutí hlavičkového souboru
   pro Webots robot
2. #include <webots/Motor.hpp> // Zahrnutí hlavičkového souboru
   pro pohybové motory
3. #include <webots/DistanceSensor.hpp> // Zahrnutí
   hlavičkového souboru pro ultrazvukové senzory
4. #include <webots/ColorSensor.hpp> // Zahrnutí hlavičkového
   souboru pro barevný senzor
5. using namespace webots;
6.
7. #define TIME_STEP 64 // Interval kroku simulace
8. #define MAX_SPEED 6.28 // Maximální rychlost motorů
9. #define WAIT_TIME 5000 // Čekací doba na začátek soutěže (5
   sekund)
10.     #define TURN_TIME 1570 // Čas potřebný pro otočení o 90
   stupňů
11.     #define DISTANCE_THRESHOLD 500 // Práh pro detekci
   druhého robota ultrazvukovými senzory
12.     #define LINE_THRESHOLD 800 // Práh pro detekci bílé
   čáry barevným senzorem
13.
14.     // Definice třídy MyRobot odvozené od třídy Robot
15.     class MyRobot : public Robot {
16.     private:
17.       Motor *leftMotor, *rightMotor; // Ukazatele na
   pohybové motory
18.       DistanceSensor *frontSensor, *leftSensor,
   *rightSensor; // Ukazatele na ultrazvukové senzory
19.       ColorSensor *lineSensor; // Ukazatel na barevný
   senzor
20.     public:
21.       MyRobot() {
22.         leftMotor = getMotor("left wheel motor"); //
   Inicializace levého motoru
23.         rightMotor = getMotor("right wheel motor"); //
   Inicializace pravého motoru
24.         frontSensor = getDistanceSensor("front sensor"); //
   Inicializace předního ultrazvukového senzoru
25.         leftSensor = getDistanceSensor("left sensor"); //
   Inicializace levého ultrazvukového senzoru
26.         rightSensor = getDistanceSensor("right sensor"); //
   Inicializace pravého ultrazvukového senzoru
27.         lineSensor = getColorSensor("line sensor"); //
   Inicializace barevného senzoru
28.         frontSensor->enable(TIME_STEP); // Zapnutí předního
   ultrazvukového senzoru
```

```
29.         leftSensor->enable(TIME_STEP); // Zapnutí levého
  ultrazvukového senzoru
30.         rightSensor->enable(TIME_STEP); // Zapnutí pravého
  ultrazvukového senzoru
31.         lineSensor->enable(TIME_STEP); // Zapnutí barevného
  senzoru
32.       }
33.
34.       // Metoda pro otočení robota o 90 stupňů
35.       void turn90Degrees() {
36.         leftMotor->setPosition(TURN_TIME); // Nastavení
  pozice levého motoru pro otočení
37.         rightMotor->setPosition(-TURN_TIME); // Nastavení
  pozice pravého motoru pro otočení
38.         while (step(TIME_STEP) != -1) { // Cyklus, dokud se
  motory nevrátí do výchozí pozice
39.           if (leftMotor->getPosition() == 0 && rightMotor-
  >getPosition() == 0)
40.             break; // Pokud jsou motory výchozí pozici,
  ukončit cyklus
41.         }
42.       }
43.
44.       // Hlavní metoda pro řízení robota
45.       void run() {
46.         int start_time = getTime(); // Časový údaj o
  začátku běhu programu
47.         bool found_enemy = false; // Příznak nalezení
  nepřítele
48.         while (step(TIME_STEP) != -1) { // Hlavní smyčka
  programu
49.           if (getTime() - start_time < WAIT_TIME) {
50.             // Čekání na začátek soutěže
51.             continue; // Pokračovat na další iteraci smyčky
52.           } else if (!found_enemy) {
53.             // Otočení o 90 stupňů po uplynutí 5 sekund
54.             turn90Degrees(); // Volání metody pro otočení o
  90 stupňů
55.             found_enemy = true; // Nastavení příznaku
  nalezení nepřítele
56.           } else {
57.             double leftSpeed = MAX_SPEED; // Nastavení
  počáteční rychlosti levého motoru
58.             double rightSpeed = MAX_SPEED; // Nastavení
  počáteční rychlosti pravého motoru
59.             // Detekce druhého robota pomocí ultrazvukových
  senzorů
60.             if (frontSensor->getValue() <
  DISTANCE_THRESHOLD ||
61.                 leftSensor->getValue() < DISTANCE_THRESHOLD
  ||
62.                 rightSensor->getValue() <
  DISTANCE_THRESHOLD) {
63.               // Robot na dosah, zaútočit
64.               leftSpeed = MAX_SPEED; // Nastavení maximální
  rychlosti levého motoru
65.               rightSpeed = MAX_SPEED; // Nastavení
  maximální rychlosti pravého motoru
66.             } else {
67.               // Pokračovat v hledání
68.               leftSpeed = MAX_SPEED / 2; // Snížení
  rychlosti levého motoru
```

```
69.                 rightSpeed = MAX_SPEED / 2; // Snížení
   rychlosti pravého motoru
70.             }
71.             // Detekce bílé čáry
72.             if (lineSensor->getValue() > LINE_THRESHOLD) {
73.               leftSpeed = -MAX_SPEED; // Nastavení záporné
   rychlosti levého motoru (couvání)
74.               rightSpeed = -MAX_SPEED; // Nastavení záporné
   rychlosti pravého motoru (couvání)
75.               // Otočení o 90 stupňů
76.               turn90Degrees(); // Volání metody pro otočení
   o 90 stupňů
77.               // Resetování příznaku nalezení nepřítele
78.               found_enemy = false;
79.             }
80.             // Nastavení rychlostí motorů
81.             leftMotor->setVelocity(leftSpeed); // Nastavení
   rychlosti levého motoru
82.             rightMotor->setVelocity(rightSpeed); //
   Nastavení rychlosti pravého motoru
83.           }
84.         }
85.       }
86.     };
87.
88.     // Hlavní funkce
89.     int main() {
90.       MyRobot robot; // Vytvoření instance třídy MyRobot
91.       robot.run(); // Spuštění hlavní metody pro řízení
   robota
92.       return 0; // Návratová hodnota programu
93.     }
```

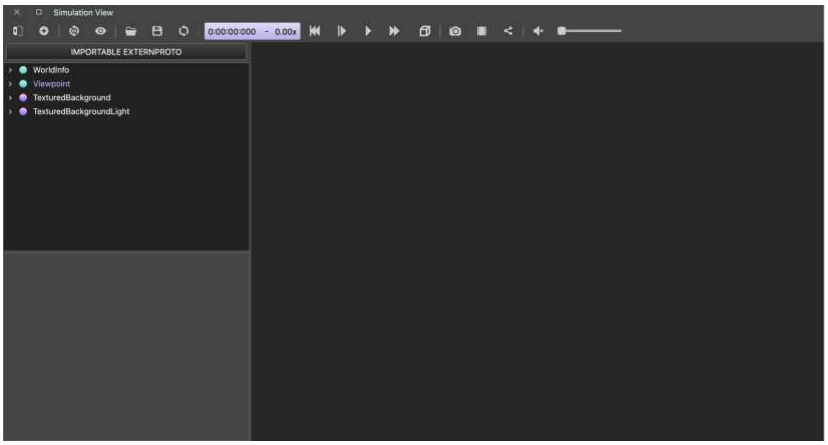
6.2.7 Kompletní kod v Pythonu

```
1. from controller import Robot  # Importuje třídu Robot z
   modulu controller
2.
3. TIME_STEP = 64  # Základní časový krok simulace
4. MAX_SPEED = 6.28  # Maximální rychlost motoru
5. WAIT_TIME = 5000  # Doba čekání na start v milisekundách (5
   sekund)
6. TURN_TIME = 1570  # Doba potřebná pro otočení o 90 stupňů v
   milisekundách
7. DISTANCE_THRESHOLD = 0.5  # Minimální vzdálenost pro detekci
   druhého robota v metrech
8. LINE_THRESHOLD = 800  # Hodnota senzoru pro detekci bílé
   čáry
9. robot = Robot()  # Vytvoří instanci robota
10.
11.     # Inicializace motorů a senzorů
12.     left_motor = robot.getMotor("left wheel motor")  #
   Získá levý motor
13.     right_motor = robot.getMotor("right wheel motor")  #
   Získá pravý motor
14.     front_sensor = robot.getDistanceSensor("front sensor")
   # Získá přední distanční senzor
15.     left_sensor = robot.getDistanceSensor("left sensor")  #
   Získá levý distanční senzor
16.     right_sensor = robot.getDistanceSensor("right sensor")
   # Získá pravý distanční senzor
```

```
    line_sensor = robot.getColorSensor("line sensor")  # Získá senzor pro detekci čáry

    # Aktivace senzorů
    front_sensor.enable(TIME_STEP)
    left_sensor.enable(TIME_STEP)
    right_sensor.enable(TIME_STEP)
    line_sensor.enable(TIME_STEP)

    def turn_90_degrees():
        left_motor.setPosition(TURN_TIME)  # Nastaví pozici levého motoru pro otočení
        right_motor.setPosition(-TURN_TIME)  # Nastaví pozici pravého motoru pro otočení
        while robot.step(TIME_STEP) != -1:  # Čeká na dokončení otočení
            if left_motor.getPosition() == 0 and right_motor.getPosition() == 0:
                break  # Pokud je otočení dokončeno, vyskočí z cyklu

    def run():
        start_time = robot.getTime()  # Získá startovní čas
        found_enemy = False  # Příznak pro nalezení nepřítele
        while robot.step(TIME_STEP) != -1:  # Hlavní smyčka
            if robot.getTime() - start_time < WAIT_TIME:
                continue  # Čeká na začátek soutěže
            elif not found_enemy:
                turn_90_degrees()  # Otočí se o 90 stupňů po uplynutí 5 sekund
                found_enemy = True
            else:
                left_speed = MAX_SPEED
                right_speed = MAX_SPEED
                if (front_sensor.getValue() < DISTANCE_THRESHOLD or
                    left_sensor.getValue() < DISTANCE_THRESHOLD or
                    right_sensor.getValue() < DISTANCE_THRESHOLD):
                    left_speed = MAX_SPEED  # Nastaví rychlost na maximum při detekci nepřítele
                    right_speed = MAX_SPEED
                else:
                    left_speed = MAX_SPEED / 2  # Sníží rychlost pro hledání
                    right_speed = MAX_SPEED / 2
                if line_sensor.getValue() > LINE_THRESHOLD:
                    left_speed = -MAX_SPEED  # Couvá při detekci bílé čáry
                    right_speed = -MAX_SPEED
                    turn_90_degrees()  # Otočí se o 90 stupňů
                    found_enemy = False  # Resetuje pro hledání nepřítele
                left_motor.setVelocity(left_speed)  # Nastaví rychlost levého motoru
                right_motor.setVelocity(right_speed)  # Nastaví rychlost pravého motoru

    run()  # Spustí hlavní funkci
```

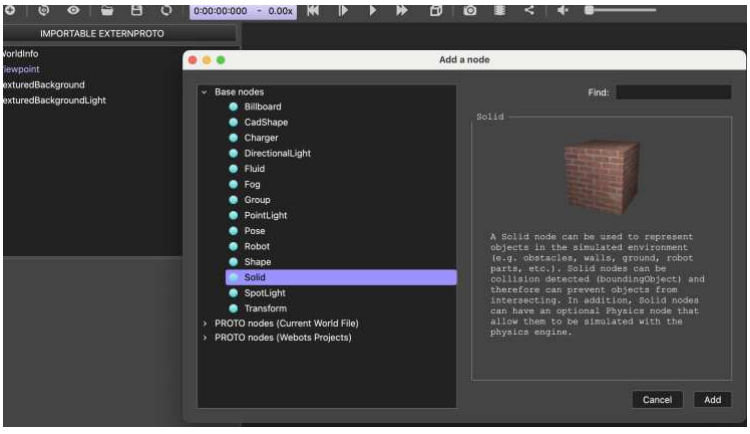
6.3 Model

V tomto případě při sestavování projektu uděláme mírnou změnu. Nepoužijeme předem definované nastavení, jako v minulých případech, kdy si do projektu necháme přidat RectangleArena, ale zvolíme možnost s prázdným prostředím. Možnosti prázdného prostředí využijeme z tohoto důvodu, že soutěžní disciplína SUMO se odehrává na podložce, která má tvar válce. Zde by nám předem vytvořená čtvercová aréna nebyla příliš platná. Pokud ovšem bychom vygenerovali projekt, kde by již aréna byla, nemusíme zoufat. Aréna jde jednoduchým způsobem smazat, a to tím stylem, že v nabídce po levé straně vybereme její komponentu a po stisknutí pravého tlačítka, se nám objeví možnost pro její odstranění.



Obrázek 28 - SUMO – Pracovní prostředí

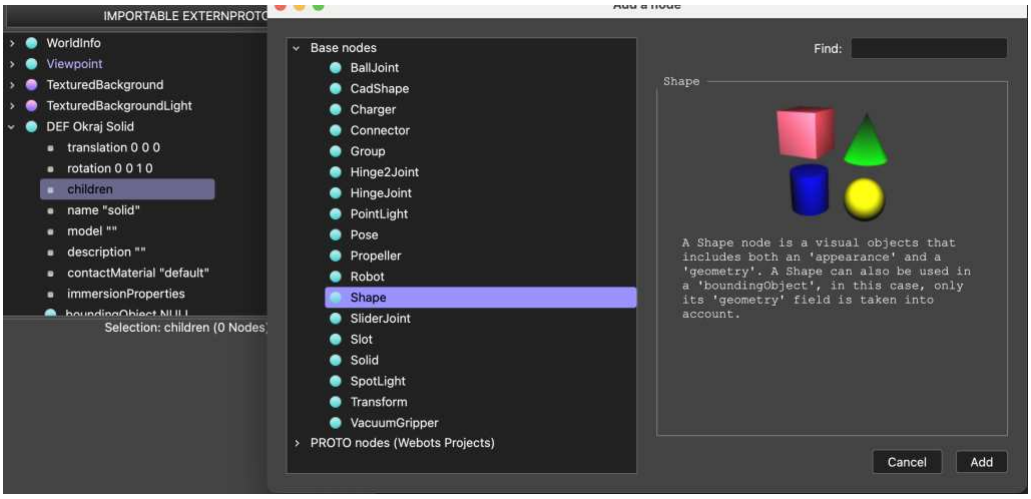
Nyní přes nabídku v horní liště a přes tlačítko Add začneme přidávat do našeho projektu potřebné komponenty ke stavbě kruhu. V nabídce vybereme možnost Solid a tu následně přidáme do projektu.



Obrázek 29 - SUMO – Tvorba kruhu

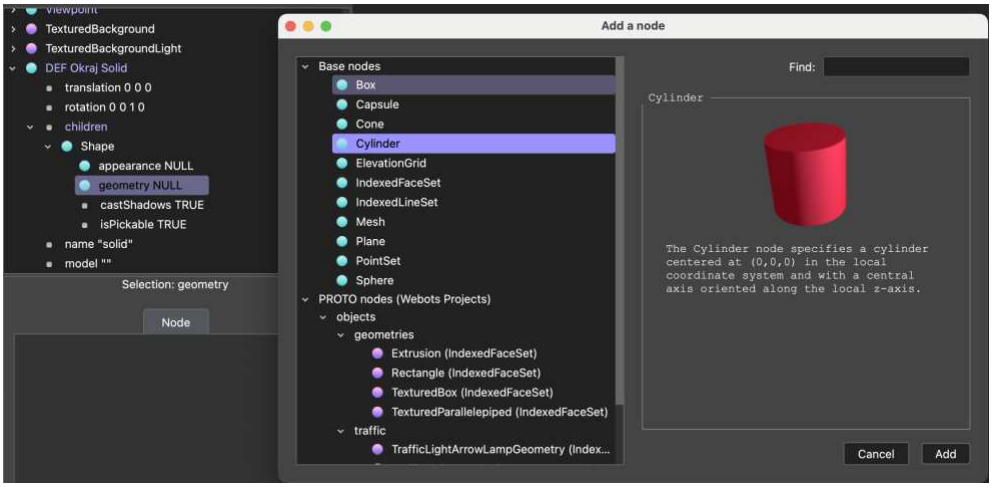
Nyní si doporučuji přejmenovat danou komponentu a to tím, že dvakrát poklikám na její hlavní funkci. Tento krok není povinný a lze pokračovat i bez něj, ovšem, když budeme pokračovat dále s více stejnými funkcemi, mohlo by docházet k tomu, že se vytvoří chaos a zmatek. Tímto tomuto problému zabráníme.

Dalším krokem, který musíme udělat, je přidání tvaru. Po rozkliknuti naší nyní již přejmenované funkce si najdeme kolonku children. Tuto kolonku rozklikneme a vybereme zde možnost Shape a přidáme ji doprojektu.



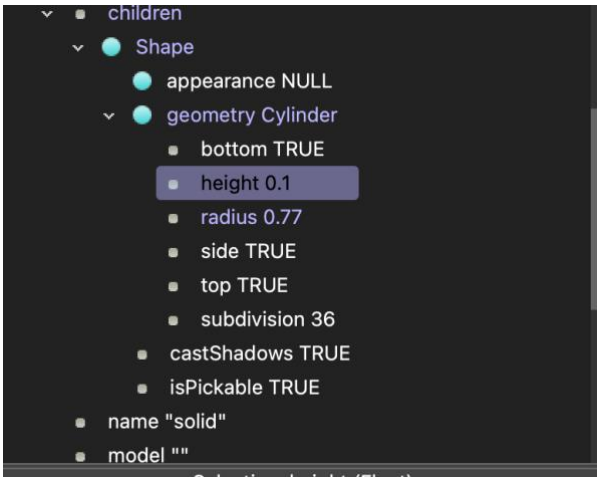
Obrázek 30 - SUMO – tvorba

Po přidání tvaru se nám objeví další možnost. Touto možností, kterou potřebujeme a budeme s ní nadále pracovat je možnost Shape. Tu si rozklikneme a najdeme si kolonku geometry. Po rozkliknutí kolonky geometry se nám objeví tabulka s možnostmi tvarů. Pro náš účel vybereme možnost Cylinder a vložíme jej do projektu.



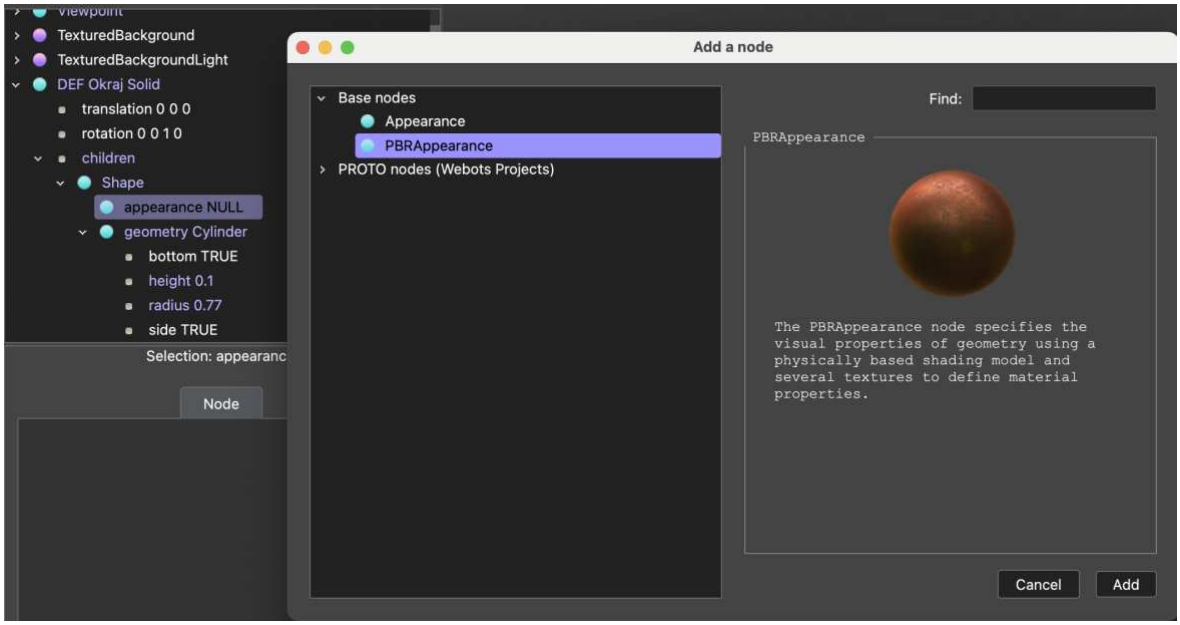
Obrázek 31 - SUMO – tvorba

Již máme přidaný první kruh, který potřebujeme jako základ. Nyní je na čase upravit jeho rozměry, tak aby odpovídali požadovanému zadání. Úpravu hodnot nalezneme po rozkliknutí geometry Cylinder pod možnostmi height, která nám změní výšku objektu a radius, který nám změní průměr objektu.



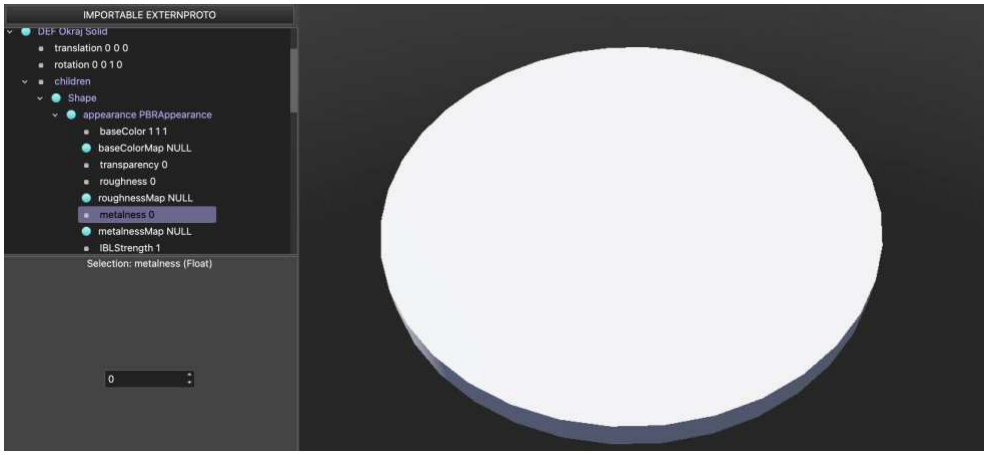
Obrázek 32- SUMO – tvorba velikosti

Nyní potřebujeme změnit barvu objektu z důvodu, že okraj kruhu musí být bílý a vnitřek kruhu černý. Tohoto docílíme krokem, že v možnostech Shape rozklikneme možnost appearace, zde zvolíme možnost PBRAppearence a tu přidáme do projektu.



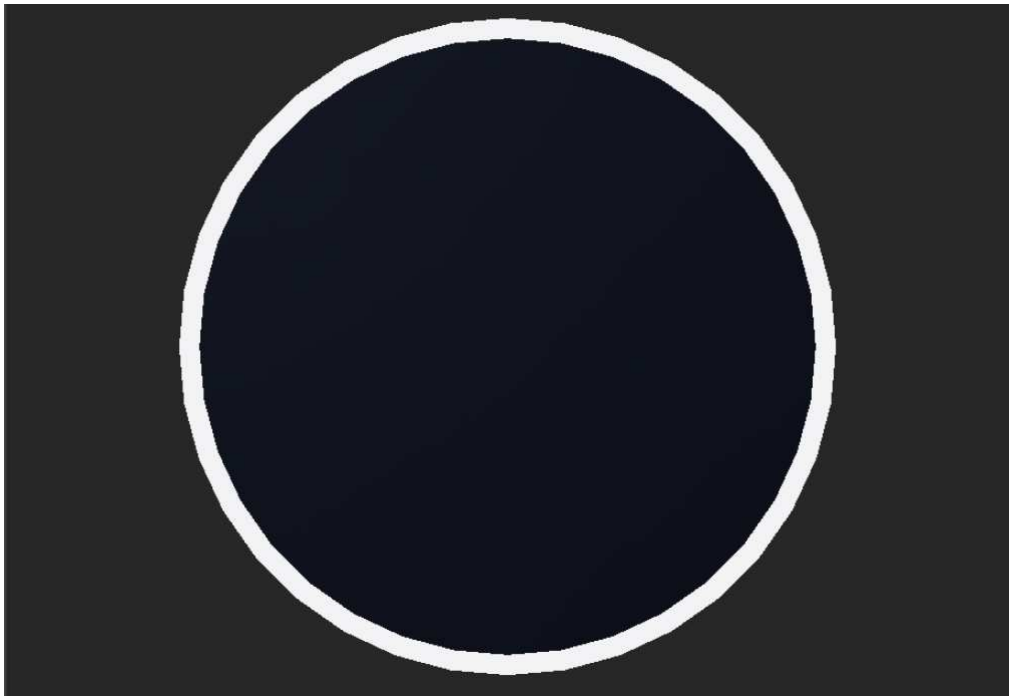
Obrázek 33 - SUMO – tvorba barva

Nyní nám již zbývá změnit barvu na čistě bílou a oddělat metallnes. Tyto kroky učiníme tak, že si rozklikneme položku appearance a přejdeme do kolonky color, kde si navolíme příslušnou barvu. V našem případě je jako první potřebná bíla tudíž filtry všech tří složek nastavme na 1. V dalším kroku změníme hodnotu u metallnes z 1 na 0 a tím dostaneme čistě bílý odstín kruhu.



Obrázek 34 - SUMO – tvorba, změna barvy

U vytváření vnitřní části kruhu, který bude mít černou barvu, budeme postupovat krok za krokem stejně, jako jsme postupovali v první části. Jediný rozdíl, který uděláme bude změna velikosti na menší, aby kruh mohl být usazen vevnitř a změna barvy na černou.



Obrázek 35 - SUMO – Výsledná podoba

7 LINE FOLLOWER

7.1 Zadání

Vymodelujte nebo použijte importovaného robota z prostředí Webots, který ujede v co nejrychlejším čase závodní dráhu určenou pro soutěž LineFollower.

Vašim úkolem je projet závodní dráhu určenou pro disciplínu LineFollower za co nejkratší dobu. Celkově máte na projetí čas stanovený na 3. minuty. Za tuto dobu můžete absolvovat libovolný počet pokusů. Jako finální čas se počítá ten, který bude nejrychlejší.

Na výběr máte ze dvou možných typů řešení tohoto úkolu.

Varianta 1:

Jedná se o jednodušší variantu, kdy používáte pouze jedno čidlo. Tato varianta je jednodušší z pohledu programování, avšak tento robot je pomalejší a méně přesný.

Varianta 2:

Varianta s více čidly. Nejlepší možnou cestou na řešení s více čidly je použití právě tří čidel. Pomocí těchto čidel vytvoříte PID regulátor. Toto řešení je z hlediska programovacího a testovacího složitější variantou ovšem výsledný robot je rychlejší a přesnější.

7.2 Program bez PID

7.2.1 Definice hlaviček a konstant

Prvním krokem našeho programu je vložení hlavičkových souborů pro WeBots abychom mohli využívat funkce a třídy, které jsou jimi poskytovány. Dále definujeme konstanty, které slouží pro nastavení hodnot. V našem případě budeme používat pro rychlost, a odraz světla

```
1. #include <webots/Robot.hpp>
2. #include <webots/Motor.hpp>
3. #include <webots/DistanceSensor.hpp>
4. #define SPEED 5.0 // Definice konstantní rychlosti pohybu
5. #define THRESHOLD 500 // Práh pro detekci černé čáry
6. #define TURN_THRESHOLD 300 // Práh pro detekci ostré zatáčky
7. #define CROSSROAD_THRESHOLD 800 // Práh pro detekci
   křižovatky
8. #define TURN_SPEED 2.5 // Rychlost pro zatáčení
9. #define TURN_TIME 100 // Čas potřebný k otočení v
   milisekundách
```

7.2.2 Hlavní funkce main()

Tato část slouží pro inicializaci robota. Nastavujeme zde motory a senzory robota. Dále zde nastavujeme parametry motorů a ktivujeme senzor. Proměnná inCrossroad slouží k detekci, zdali se robot nachází na křižovatce.

```
using namespace webots; // Použití jmenného prostoru Webots
int main() {
    Robot robot; // Vytvoření instance robota
    int timeStep = (int)robot.getBasicTimeStep(); // Získání základního časového kroku simulace
    Motor *leftMotor = robot.getMotor("left wheel motor"); // Připojení levého motoru
    Motor *rightMotor = robot.getMotor("right wheel motor"); // Připojení pravého motoru
    DistanceSensor *sensor = robot.getDistanceSensor("sensor"); // Připojení snímače vzdálenosti
    sensor->enable(timeStep); // Aktivace snímače vzdálenosti
    leftMotor->setPosition(INFINITY); // Nastavení nekonečné pozice pro kontinuální rotaci
        rightMotor->setPosition(INFINITY);
        leftMotor->setVelocity(0); // Inicializace motorů na nulovou rychlost
        rightMotor->setVelocity(0);
        bool inCrossroad = false; // Příznak, zda je robot na křižovatce
```

7.2.3 Hlavní smyčka programu

V hlavní smyčce programu čteme hodnoty ze senzorů a díky nim řídíme pohyb robota po čáře. Hodnoty rozhodují o tom, jaká akce bude provedena na základě čáry, křižovatky a obtížnosti zatáčky.

```
    while (robot.step(timeStep) != -1) { // Hlavní smyčka programu
        double sensorValue = sensor->getValue(); // Získání hodnoty ze snímače
        if (sensorValue < THRESHOLD) { // Pokud je detekována černá čára
            leftMotor->setVelocity(SPEED); // Pohyb rovně
            rightMotor->setVelocity(SPEED);
            inCrossroad = false; // Nenacházíme se na křižovatce
        } else if (sensorValue < CROSSROAD_THRESHOLD && !inCrossroad) { // Pokud je pravděpodobně detekována křižovatka
            leftMotor->setVelocity(TURN_SPEED); // Otočení vpravo
            rightMotor->setVelocity(-TURN_SPEED);
                robot.step(TURN_TIME); // Čekání na dokončení otočky
                inCrossroad = true; // Indikace, že jsme na křižovatce
```

```
            } else if (sensorValue >= CROSSROAD_THRESHOLD
&& inCrossroad) { // Pokud jsme ujeli od křižovatky
                inCrossroad = false; // Reset indikátoru
křižovatky
            } else { // Pokud dojde k ostré zatáčce nebo
ztrátě čáry
                if (sensorValue < TURN_THRESHOLD) { //
Pokud je detekována ostrá zatáčka
                    leftMotor->setVelocity(SPEED / 2); //
Zpomalení a otočení
                    rightMotor->setVelocity(-SPEED / 2);
                } else { // Pokud čára není detekována
                    leftMotor->setVelocity(-TURN_SPEED); //
Hledání čáry otáčením
                    rightMotor->setVelocity(TURN_SPEED);
                }
            }
        }
```

7.2.4 Konec programu

V posledním kroku je ukončena hlavní funkce main() a celý program. Návratová hodnota 0 ukazuje, že program proběhl v pořádku a bez chyb.

```
    return 0; // Konec programu
}
```

7.2.5 Celý program v C++

```
#include <webots/Robot.hpp> //
#include <webots/Motor.hpp> //
#include <webots/DistanceSensor.hpp> //
#define SPEED 5.0 // Definice konstantní rychlosti pohybu
#define THRESHOLD 500 // Práh pro detekci černé čáry
#define TURN_THRESHOLD 300 // Práh pro detekci ostré zatáčky
#define CROSSROAD_THRESHOLD 800 // Práh pro detekci
křižovatky
#define TURN_SPEED 2.5 // Rychlost pro zatáčení
#define TURN_TIME 100 // Čas potřebný k otočení v
milisekundách
    using namespace webots; // Použití jmenného prostoru
Webots
    int main() {
        Robot robot; // Vytvoření instance robota
        int timeStep = (int)robot.getBasicTimeStep(); //
Získání základního časového kroku simulace
        Motor *leftMotor = robot.getMotor("left wheel
motor"); // Připojení levého motoru
        Motor *rightMotor = robot.getMotor("right wheel
motor"); // Připojení pravého motoru
        DistanceSensor *sensor =
robot.getDistanceSensor("sensor"); // Připojení snímače
vzdálenosti
        sensor->enable(timeStep); // Aktivace snímače
vzdálenosti
        leftMotor->setPosition(INFINITY); // Nastavení
nekonečné pozice pro kontinuální rotaci
        rightMotor->setPosition(INFINITY);
```

```
20.         leftMotor->setVelocity(0); // Inicializace motorů
  na nulovou rychlost
21.         rightMotor->setVelocity(0);
22.         bool inCrossroad = false; // Příznak, zda je robot
  na křižovatce
23.         while (robot.step(timeStep) != -1) { // Hlavní
  smyčka programu
24.             double sensorValue = sensor->getValue(); //
  Získání hodnoty ze snímače
25.             if (sensorValue < THRESHOLD) { // Pokud je
  detekována černá čára
26.                 leftMotor->setVelocity(SPEED); // Pohyb
  rovně
27.                 rightMotor->setVelocity(SPEED);
28.                 inCrossroad = false; // Nenacházíme se na
  křižovatce
29.             } else if (sensorValue < CROSSROAD_THRESHOLD &&
  !inCrossroad) { // Pokud je pravděpodobně detekována
  křižovatka
30.                 leftMotor->setVelocity(TURN_SPEED); //
  Otočení vpravo
31.                 rightMotor->setVelocity(-TURN_SPEED);
32.                 robot.step(TURN_TIME); // Čekání na
  dokončení otočky
33.                 inCrossroad = true; // Indikace, že jsme na
  křižovatce
34.             } else if (sensorValue >= CROSSROAD_THRESHOLD
  && inCrossroad) { // Pokud jsme ujeli od křižovatky
35.                 inCrossroad = false; // Reset indikátoru
  křižovatky
36.             } else { // Pokud dojde k ostré zatáčce nebo
  ztrátě čáry
37.                 if (sensorValue < TURN_THRESHOLD) { //
  Pokud je detekována ostrá zatáčka
38.                     leftMotor->setVelocity(SPEED / 2); //
  Zpomalení a otočení
39.                     rightMotor->setVelocity(-SPEED / 2);
40.                 } else { // Pokud čára není detekována
41.                     leftMotor->setVelocity(-TURN_SPEED); //
  Hledání čáry otáčením
42.                     rightMotor->setVelocity(TURN_SPEED);
43.                 }
44.             }
45.         }
46.         return 0; // Konec programu
47.     }
```

7.3 Program s PID

7.3.1 Import knihoven a definice konstant

V první části kódu si naimportujeme knihovny a definujeme konstanty, které budeme používat v programu. Konstanty KP, KI, KD jsou parametry, které budeme dále využívat pro PID regulátor.

```
1. #include <webots/Robot.hpp>
2. #include <webots/Motor.hpp>
3. #include <webots/DistanceSensor.hpp>
```

```
#define TIME_STEP 64
#define KP 1.2
#define KI 0.02
#define KD 20
#define BASE_SPEED 10.0
using namespace webots;
```

7.3.2 Definice třídy s PID

V této části kódu definujeme PID regulátor. Konstruktor třídy inicializuje parametry pro proporcionalitu, integraci a derivaci. Metoda calculate přijímá žádanou hodnotu setpoint a aktuální hodnotu pv jako vstupy, dále vypočítá aktuální chybu error, integrační a derivační části. Nakonec vrací výstup regulátoru, který je následně použit.

```
class PIDController {
public:
    PIDController(double kp, double ki, double kd) : kp_(kp), ki_(ki), kd_(kd), prev_error_(0.0), integral_(0.0) {}
    double calculate(double setpoint, double pv) {
        double error = setpoint - pv;
        integral_ += error;
        double derivative = error - prev_error_;
        prev_error_ = error;
        return (kp_ * error) + (ki_ * integral_) + (kd_ * derivative);
         }
     private:
         double kp_;
         double ki_;
         double kd_;
         double prev_error_;
         double integral_;
     };
```

7.3.3 Main funkce

Hlavní funkce main inicializuje robota a jeho komponenty, a to včetně motorů a senzorů. PIDController je inicializován s předem definovanými komponenty. Ve smyčce while, která se opakuje jsou nejprve čteny hodnoty ze senzorů na základě, kterých je vypočítaná chyba. Tato chyba je dále použita v PID regulátoru k výpočtu výstupu, který je následně použit k nastavení, tak aby sledoval čáru.

```
int main() {
Robot robot;
PIDController pid(KP, KI, KD);
Motor *leftMotor = robot.getMotor("left wheel motor");
Motor *rightMotor = robot.getMotor("right wheel motor");
leftMotor->setPosition(INFINITY);
rightMotor->setPosition(INFINITY);
leftMotor->setVelocity(0);
rightMotor->setVelocity(0);
```

```
10.     DistanceSensor *leftSensor =
   robot.getDistanceSensor("left sensor");
11.     DistanceSensor *middleSensor =
   robot.getDistanceSensor("middle sensor");
12.     DistanceSensor *rightSensor =
   robot.getDistanceSensor("right sensor");
13.     leftSensor->enable(TIME_STEP);
14.     middleSensor->enable(TIME_STEP);
15.     rightSensor->enable(TIME_STEP);
16.     while (robot.step(TIME_STEP) != -1) {
17.     double leftValue = leftSensor->getValue();
18.     double middleValue = middleSensor->getValue();
19.     double rightValue = rightSensor->getValue();
20.     // Zpracování hodnot senzorů a výpočet odchylky
21.     double error = leftValue - rightValue;
22.     // Vypočítáme PID výstup
23.     double pidOutput = pid.calculate(0, error);
24.     // Nastavení rychlosti motorů
25.     leftMotor->setVelocity(BASE_SPEED - pidOutput);
26.     rightMotor->setVelocity(BASE_SPEED + pidOutput);
27.     }
28.     delete leftMotor;
29.     delete rightMotor;
30.     delete leftSensor;
31.     delete middleSensor;
32.     delete rightSensor;
33.     return 0;
34.     }
```

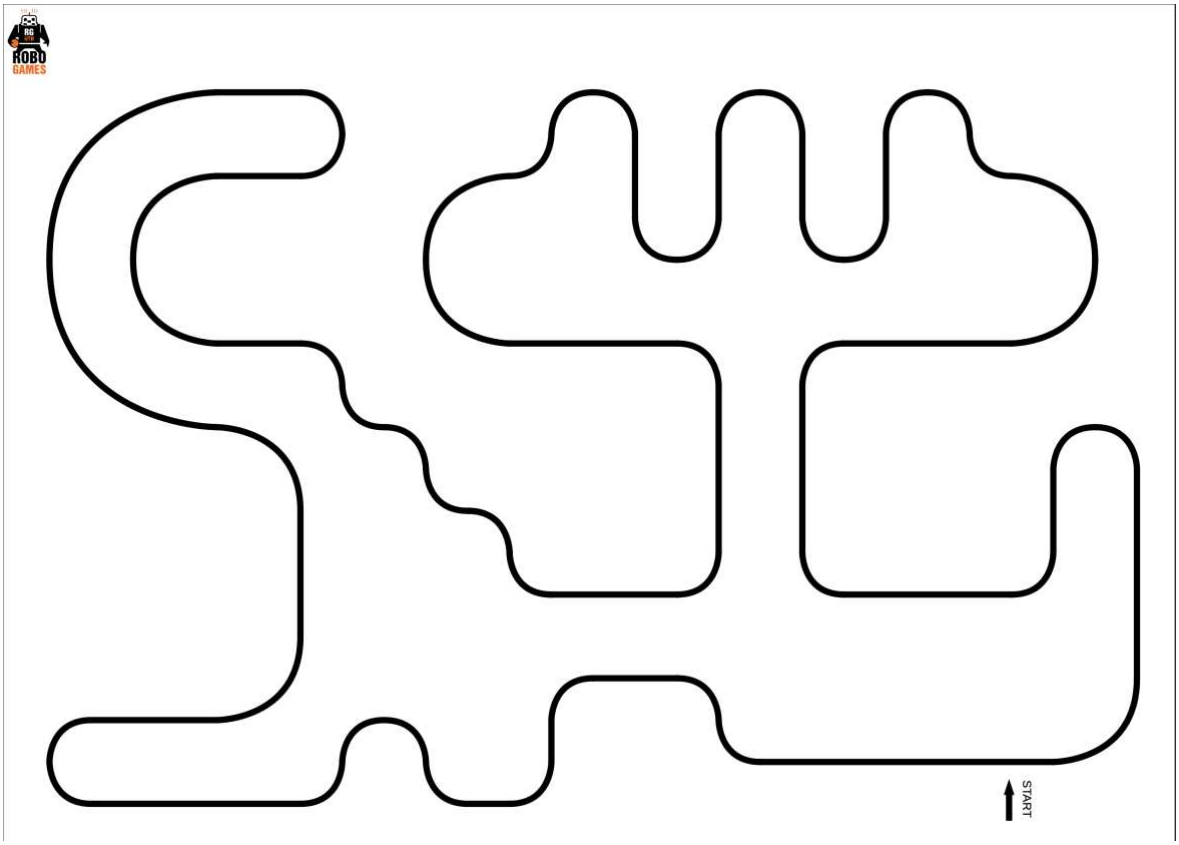
7.3.4 Celý kod v C++

```
1. #include <webots/Robot.hpp>
2. #include <webots/Motor.hpp>
3. #include <webots/DistanceSensor.hpp>
4. // Časový krok simulace (v milisekundách)
5. #define TIME_STEP 64
6. // Konstanty PID regulátoru
7. #define KP 1.2
8. #define KI 0.02
9. #define KD 20
10.     // Rychlost motorů
11.     #define BASE_SPEED 10.0
12.     using namespace webots;
13.     class PIDController {
14.     public:
15.     PIDController(double kp, double ki, double kd) :
   kp_(kp), ki_(ki), kd_(kd), prev_error_(0.0), integral_(0.0)
   {}
16.     double calculate(double setpoint, double pv) {
17.     double error = setpoint - pv;
18.     integral_ += error;
19.     double derivative = error - prev_error_;
20.     prev_error_ = error;
21.     return (kp_ * error) + (ki_ * integral_) + (kd_ *
   derivative);
22.     }
23.     private:
24.     double kp_;
25.     double ki_;
26.     double kd_;
27.     double prev_error_;
28.     double integral_;
```

```
29.     };
30.     int main() {
31.     Robot robot;
32.     PIDController pid(KP, KI, KD);
33.     Motor *leftMotor = robot.getMotor("left wheel motor");
34.     Motor *rightMotor = robot.getMotor("right wheel
  motor");
35.     leftMotor->setPosition(INFINITY);
36.     rightMotor->setPosition(INFINITY);
37.     leftMotor->setVelocity(0);
38.     rightMotor->setVelocity(0);
39.     DistanceSensor *leftSensor =
  robot.getDistanceSensor("left sensor");
40.     DistanceSensor *middleSensor =
  robot.getDistanceSensor("middle sensor");
41.     DistanceSensor *rightSensor =
  robot.getDistanceSensor("right sensor");
42.     leftSensor->enable(TIME_STEP);
43.     middleSensor->enable(TIME_STEP);
44.     rightSensor->enable(TIME_STEP);
45.     while (robot.step(TIME_STEP) != -1) {
46.     double leftValue = leftSensor->getValue();
47.     double middleValue = middleSensor->getValue();
48.     double rightValue = rightSensor->getValue();
49.     // Zpracování hodnot senzorů a výpočet odchylky
50.     double error = leftValue - rightValue;
51.     // Vypočítáme PID výstup
52.     double pidOutput = pid.calculate(0, error);
53.     // Nastavení rychlosti motorů
54.     leftMotor->setVelocity(BASE_SPEED - pidOutput);
55.     rightMotor->setVelocity(BASE_SPEED + pidOutput);
56.     }
57.     delete leftMotor;
58.     delete rightMotor;
59.     delete leftSensor;
60.     delete middleSensor;
61.     delete rightSensor;
62.     return 0;
63.     }
```

7.4 Model

Jako předlohu pro zadání úkolu LineFollower jsem si zvolil oficiální zadání dráhy, které se používá při soutěžích RoboGames. Jedná se jednodušší dráhu, kterou by měl projet skoro každý závodník.



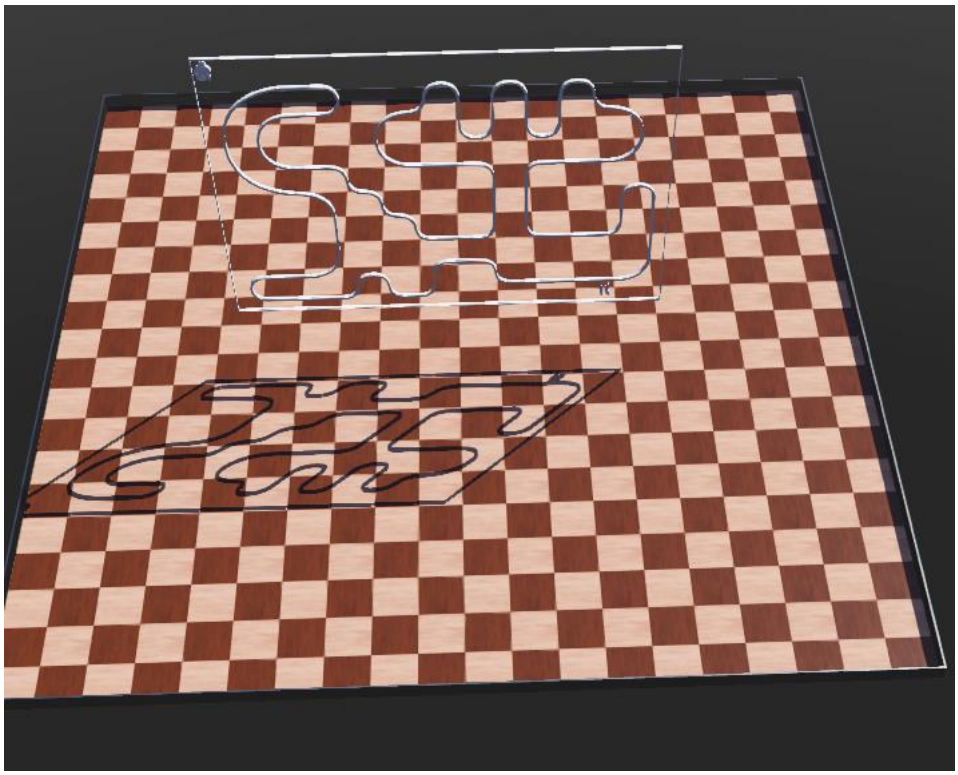
Obrázek 36 - LineFollower – Předloha

U tohoto typu modelování úkolu je postup mírně složitější a budeme potřebovat pomocné nástroje třetích stran pro vytvoření kostry mapy. Jakožto první krok jsem si stáhnul oficiální zadání trasy. Následně jsem použil online editor na převod z formátu .PNG na formát .SVG. Tento krok jsme učinili, abychom mohli následně převést objekt na další formát. Jako další potřebný krok jsem použil online nástroj TinkerCad. Lze použít i Blender, či další alternativy, kde lze pracovat s 3D modely. Já osobně rád používám TinkerCad pro jeho všestrannost a přívětivost. Zde jsem si importoval již zmíněný převedený formát souboru .SVG a upravil jej do požadované velikosti. Následně jsem jej stáhnul ve formátu .STL se kterým budeme dále pracovat již v simulátoru Webots.

Nyní již máme vše potřebné pro editaci dráhy a přejdeme do simulátoru Webots.

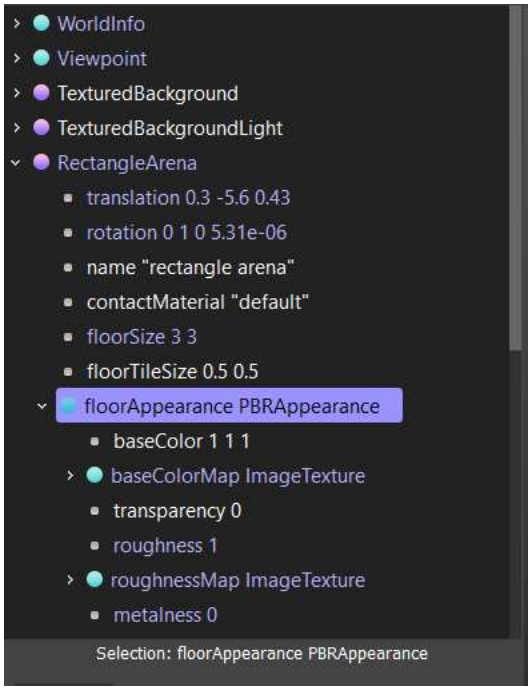
V simulátoru Webots si založíme nový projekt. Vybereme zde možnost přidat čtvercovou arénu a projekt založíme. Po založení projektu budeme potřebovat naimportovat námi převedený projekt do formátu .STL.

Tyto kroky uděláme přes hlavní lištu programu, kde si otevřeme záložku file a nalezneme kolonku import. Tu otevřeme a následně přes nové editační okno, které nám vyskočí vybereme cestu k souboru, kde ji máme uloženou. Nyní již jen akci potvrdíme a náš objekt se nám přidá do projektu.



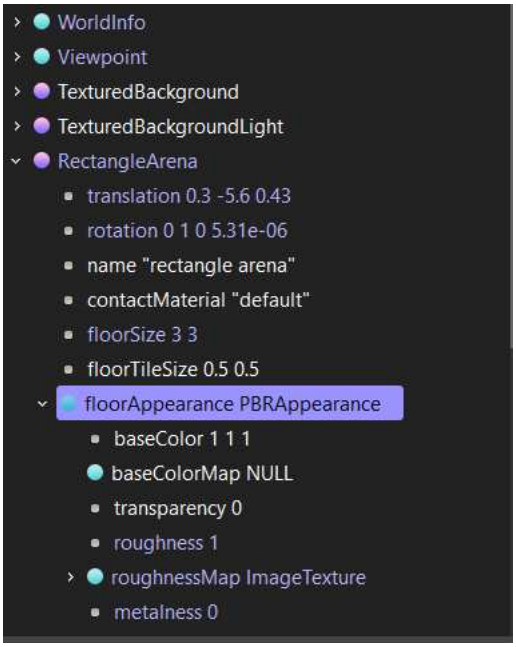
Obrázek 37 - LineFollower – Vložení dráhy

Nyní potřebujeme změnit pozadí naší čtvercové arény na bílou barvu, jelikož robot reaguje na černou barvu a okolí musí být bíle, aby nedocházelo k rušení senzorů. Tohoto kroku dosáhneme tak, že si rozklikneme pole RectangleArena a následně nalezneme pole floorAppereance. Na toto pole klikneme pravým tlačítkem myši a nalezneme možnost konvertování na base nodes. Toto potvrdíme a odemknou se nám rozšířené možnosti editaci arény.



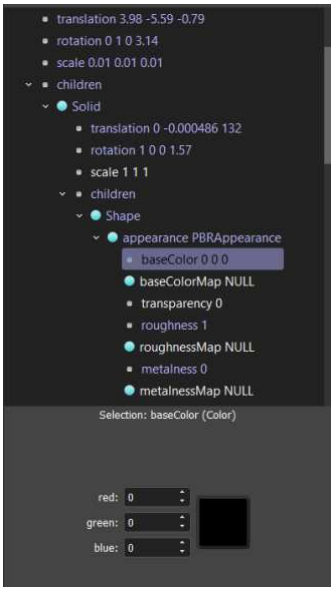
Obrázek 38 - LineFollower – Změna pozadí 1

Nyní si v rozšířené nabídce nalezne kolonku floorAppearance PBRAppearence. Na tuto kolonku opět najedeme pravým tlačítkem myši a vyjedou nám další možnosti editace. Zde opět resetujeme nodes a dostaneme finální bílou barvu pozadí, kterou pro toto zadání potřebujeme.



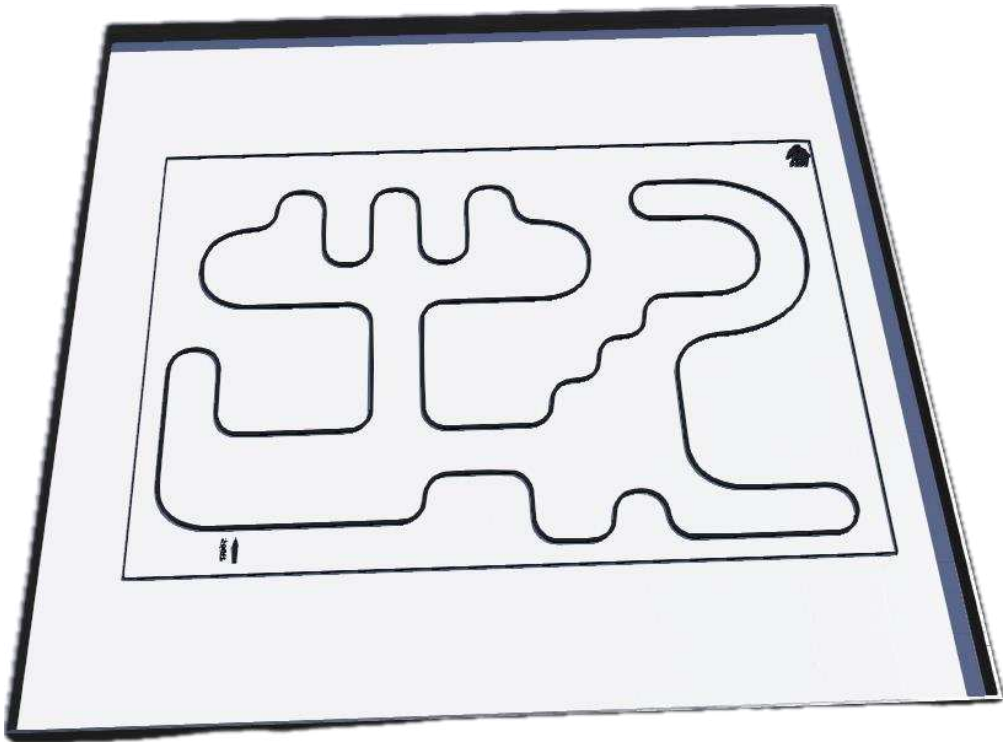
Obrázek 39 - LineFollower – Změna pozadí 2

Po finální úpravě hracího pole na bílou barvu musíme ještě přebarvit čáru na černou barvu. Zde si otevřeme kolonku, která je mateřskou pro naši vloženou čáru, otevřeme kolonku children, následně kolonku shape a zde přes možnost baseColor změníme barvu na černou.



Obrázek 40 - LineFollower – Změna barvy čáry

Po všech těchto úpravách dostaneme finální verzi naší závodní dráhy pro disciplínu LineFollower.



Obrázek 41 - LineFollower – Finální podoba

8 ROBOT UKLÍZEČ

8.1 Zadání

Vymodelujte nebo importujte již hotového robota z prostředí Webots, který uklidí celou hrací plochu a odveze co nejvíce bloků do vyhrazeného koridoru za co nejrychlejší čas.

Vašim úkolem bude uklidit a dostat do vyhrazeného prostoru 9 náhodně rozmístěných kostek na hrací ploše. Váš celkový čas na přemístění kostek libovolným způsobem jsou 3 minuty. Za tento čas můžete absolvovat libovolný počet pokusů. Jakožto výsledný čas se Vám bude počítat nejrychlejší čas s určitým počtem kostek nebo nejrychlejší čas se všemi přemístěnými kostkami do vyhrazeného koridoru.

8.2 Program

8.2.1 Inicializace

V první části kódu si opět musíme definovat knihovny a konstanty potřebné pro náš program. Dále si inicializujeme robota a jeho komponenty jako jsou motory a potřebné senzory pro detekci kostek na hrací ploše.

```
1. #include <iostream>
2. #include <webots/Robot.hpp>
3. #include <webots/Motor.hpp>
4. #include <webots/DistanceSensor.hpp>
5. #include <webots/LightSensor.hpp>
6. #define MAX_DISTANCE 100.0
7. #define MIN_LIGHT_VALUE 1000
8. #define TIME_STEP 64
9. #define MAX_TIME 180 // Maximální doba pokusu v sekundách
10.     using namespace webots;
11.
12.     int main() {
13.         Robot *robot = new Robot();
14.         Motor *leftMotor = robot->getMotor("left wheel
   motor");
15.         Motor *rightMotor = robot->getMotor("right wheel
   motor");
16.         DistanceSensor *ultrasonicSensor = robot-
   >getDistanceSensor("ultrasonic sensor");
17.         ultrasonicSensor->enable(TIME_STEP);
18.         LightSensor *lightSensor = robot-
   >getLightSensor("light sensor");
19.         lightSensor->enable(TIME_STEP);
20.
```

8.2.2 Hlavní smyčka

V hlavní smyčce budeme neustále zpracovávat data, které sbíráme ze senzorů a podle nich se bude náš robot chovat a orientovat. Pokud bude náš hledaný objekt nalezen, robot se zastaví a bude s kostkou manipulovat. Tento krok jsem nechal otevřený a na doplnění. Je zde více možných variant konstrukce a programové řešení může být velmi různorodé na vzniklé situaci. Pokud detekovaná kostka není stále nalezena, robot vyhledává dále potencionální cíl.

```
1.     int timeElapsed = 0; // Čas uplynulý od začátku pokusu
2.     while (robot->step(TIME_STEP) != -1 && timeElapsed < 
   MAX_TIME) {
3.         double distance = ultrasonicSensor->getValue();
4.         double lightValue = lightSensor->getValue();
5.         if (distance < MAX_DISTANCE) {
6.             leftMotor->setVelocity(0);
7.             rightMotor->setVelocity(0);
8.         } else {
9.             leftMotor->setVelocity(1);
10.                 rightMotor->setVelocity(1);
11.             }
12.             if (lightValue > MIN_LIGHT_VALUE) {
13.                 // ...
14.             }
15.             timeElapsed += TIME_STEP / 1000; // Převod na 
   sekundy
```

8.2.3 Kompletmí program v C++

```
1. #include <iostream>
2. #include <webots/Robot.hpp>
3. #include <webots/Motor.hpp>
4. #include <webots/DistanceSensor.hpp>
5. #include <webots/LightSensor.hpp>
6. #define MAX_DISTANCE 100.0
7. #define MIN_LIGHT_VALUE 1000
8. #define TIME_STEP 64
9. #define MAX_TIME 180 // Maximální doba pokusu v sekundách
10.     using namespace webots;
11.     int main() {
12.     // Inicializace robota
13.     Robot *robot = new Robot();
14.     // Inicializace motorů
15.     Motor *leftMotor = robot->getMotor("left wheel motor");
16.     Motor *rightMotor = robot->getMotor("right wheel 
   motor");
17.     // Inicializace senzorů
18.     DistanceSensor *ultrasonicSensor = robot-
   >getDistanceSensor("ultrasonic sensor");
19.     ultrasonicSensor->enable(TIME_STEP);
20.     LightSensor *lightSensor = robot->getLightSensor("light 
   sensor");
21.     lightSensor->enable(TIME_STEP);
22.     // Časování
23.     int timeElapsed = 0; // Čas uplynulý od začátku pokusu
24.     // Hlavní smyčka programu
```

```
25.     while (robot->step(TIME_STEP) != -1 && timeElapsed <
   MAX_TIME) {
26.     // Čtení hodnot z senzorů
27.     double distance = ultrasonicSensor->getValue();
28.     double lightValue = lightSensor->getValue();
29.     // Detekce překážky před robotem
30.     if (distance < MAX_DISTANCE) {
31.     // Zastavení robotu
32.     leftMotor->setVelocity(0);
33.     rightMotor->setVelocity(0);
34.     } else {
35.     // Pohyb vpřed
36.     leftMotor->setVelocity(1);
37.     rightMotor->setVelocity(1);
38.     }
39.     // Detekce kostek pomocí světelného senzoru
40.     if (lightValue > MIN_LIGHT_VALUE) {
41.
42.     }
43.     // Sledování času
44.     timeElapsed += TIME_STEP / 1000; // Převod na sekundy
45.
46.     std::cout << "Čas pokusu: " << timeElapsed << " s" <<
   std::endl;
47.     delete robot;
48.     return 0;
49.     }
```

8.2.4 Kompletní program v Pythonu

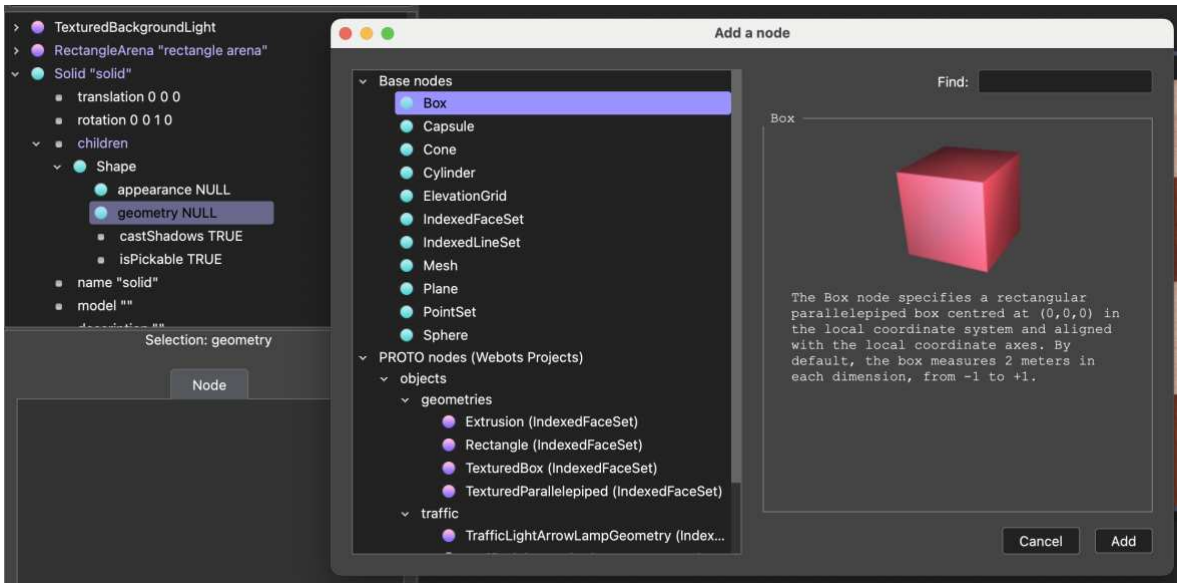
```
1. from controller import Robot, DistanceSensor, LightSensor
2. MAX_DISTANCE = 100.0
3. MIN_LIGHT_VALUE = 1000
4. TIME_STEP = 64
5. MAX_TIME = 180 # Maximální doba pokusu v sekundách
6. # Inicializace robota
7. robot = Robot()
8. # Inicializace motorů
9. left_motor = robot.getMotor("left wheel motor")
10.     right_motor = robot.getMotor("right wheel motor")
11.     # Inicializace senzorů
12.     ultrasonic_sensor = robot.getDistanceSensor("ultrasonic
   sensor")
13.     ultrasonic_sensor.enable(TIME_STEP)
14.     light_sensor = robot.getLightSensor("light sensor")
15.     light_sensor.enable(TIME_STEP)
16.     # Časování
17.     time_elapsed = 0 # Čas uplynulý od začátku pokusu
18.     # Hlavní smyčka programu
19.     while robot.step(TIME_STEP) != -1 and time_elapsed <
   MAX_TIME:
20.     # Čtení hodnot z senzorů
21.     distance = ultrasonic_sensor.getValue()
22.     light_value = light_sensor.getValue()
23.     # Detekce překážky před robotem
24.     if distance < MAX_DISTANCE:
25.     # Zastavení robotu
26.     left_motor.setVelocity(0)
27.     right_motor.setVelocity(0)
```

```
28.     # Manipulace s překážkou (např. vyzvednutí kostky)
29.     # ...
30.     else:
31.     # Pohyb vpřed
32.     left_motor.setVelocity(1)
33.     right_motor.setVelocity(1)
34.     # Detekce kostek pomocí světelného senzoru
35.     if light_value > MIN_LIGHT_VALUE:
36.     # Robot našel kostku, může ji přivézt
37.     # ...
38.     # Sledování času
39.     time_elapsed += TIME_STEP / 1000 # Převod na sekundy
40.     # Ostatní logika pokračuje zde (např. řízení podle
   čáry, reakce na signály atd.)
41.     # ...
42.     # Ukončení pokusu
43.     print("Čas pokusu:", time_elapsed, "s")
44.
```

8.3 Model

U tohoto typu úlohy budeme postupovat klasickým způsobem. Při vytváření projektu si zvolíme možnost přidání čtvercové arény. V tomto případě použijeme čtvercovou arénu jako podklad pro naši hrací plochu, a to z toho důvody, pokud by se kostky, které má robot odvézt do vyhrazeného prostoru dostali mimo hrací plochu, tak aby nespadli to nekonečna, jak je v tomto simulátoru možno, ale zůstali na hrací ploše, ovšem mimo vyhrazený prostor pro robota.

Po vytvoření projektu si přes přidáme přes tlačítko přidat v horní liště možnost solid. Následně si v možnostech solid do kolonky children přidáme možnost shape. Po přidání shape si v kolonce geometry zvolíme možnost box a ten přidáme do projektu.



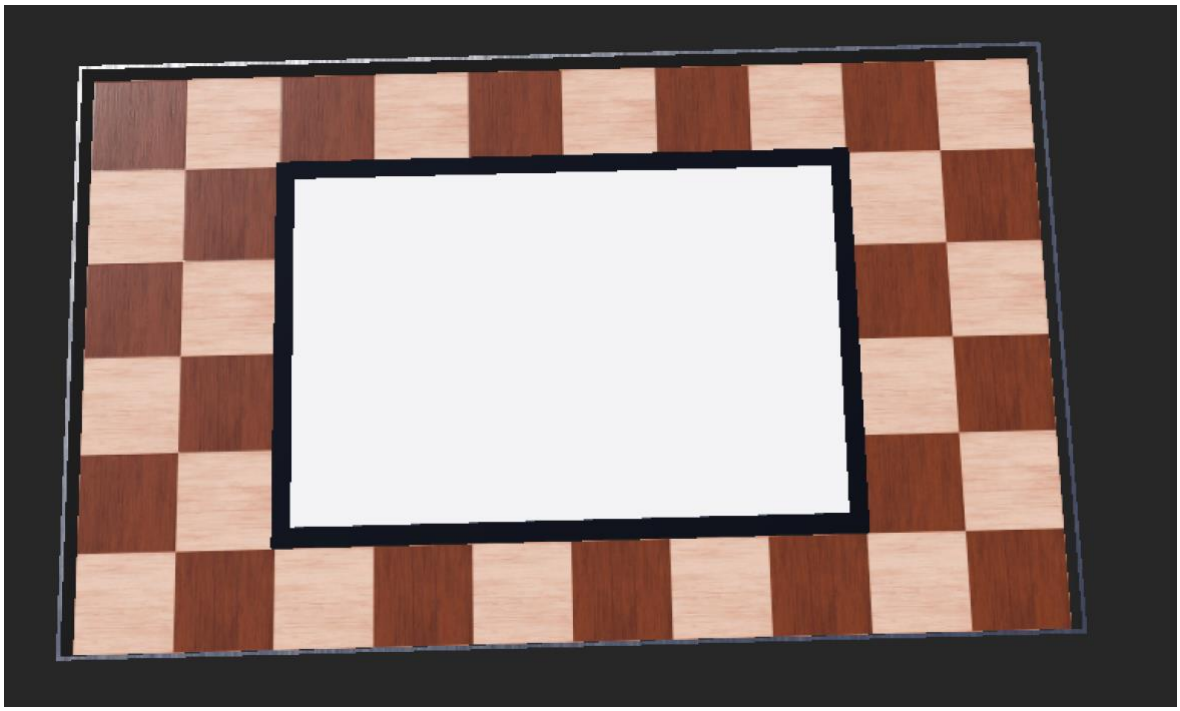
Obrázek 42 - Robot uklízeč – Hrací plocha

Po přidání čtverce do našeho projektu upravíme jeho rozměry na námi požadovanou velikost. V tomto případě to bude 1.5 m x 1 m. Tohoto docílíme tak, že si rozklikneme možnost geometry Box a zde následně kolonku size, kde upravíme rozměry.

Dále budeme potřebovat změnit barvu plochy na černou, abychom zajistili, že okraje hřiště budou zvýrazněny černým lemem. Tohoto kroku docílíme tím, že v možnosti appereance zvolíme možnost PBRAppereance. Tento krok nám umožní další editaci barev. Zde si následně rozklikneme tuto kolonku a v možnosti baseColor změníme barvu na černou. Dále pak změníme hodnotu metallnes na 0. Těmito kroky dostaneme požadovaný černý lem.

Obrázek 43 - Robot uklízeč – Tvorba plochy 2

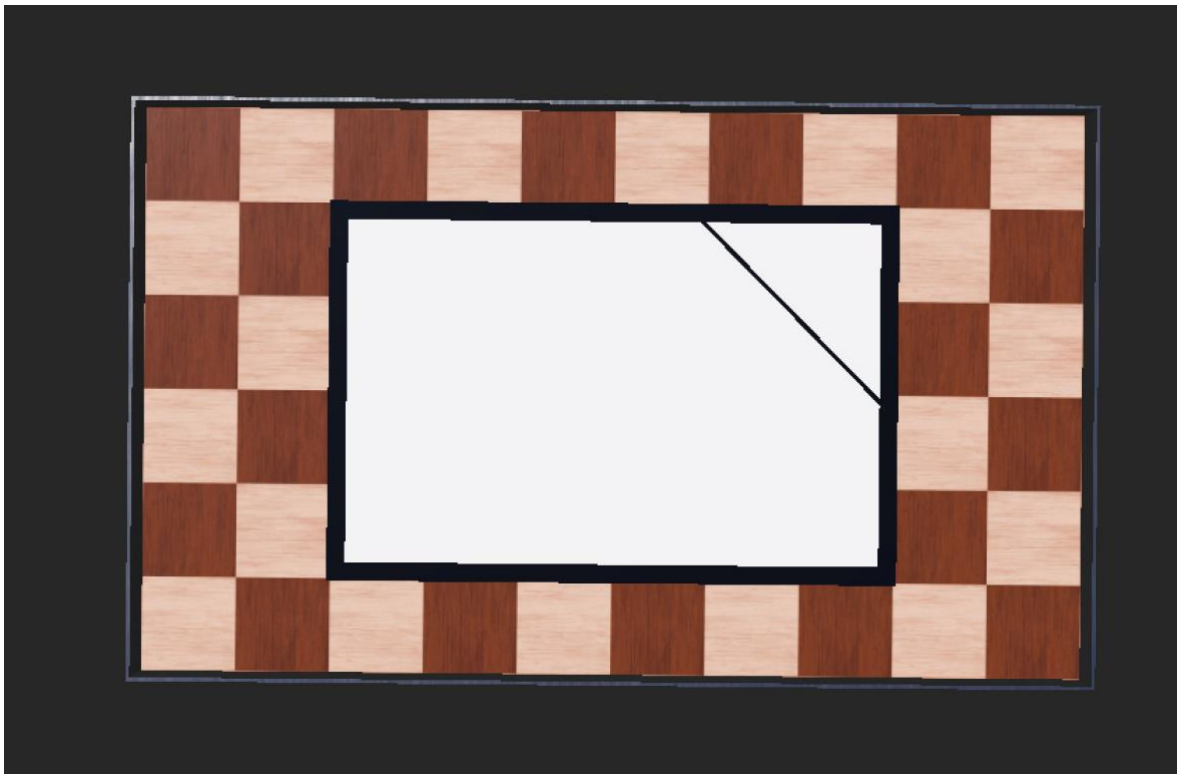
Následně postup zopakujeme a vytvoříme druhou část herní plochy, která bude mít nepatrně menší rozměr a bílou barvu. Toto uskupení bude tvořit střed hrací desky.



Obrázek 44 - Robot uklízeč – úprava plochy 2

Nyní podobnými kroky vymodelujeme poslední část hrací plochy a to čáru, která bude znázorňovat prostor, kde mají být kostky odvezeny, tedy pomyslnou cílovou rovinu. To zajistíme stejnými kroky jakožto jsme vytvořili celou hrací plochu, ale poslední krok bude

mírně odlišný. Zde upravíme rozměry čtverce tak, aby se z něj stal obdélník a následně ho vyrotujeme o potřebný úhel, aby zajistil cílovou rovinu.



Obrázek 45 - Robot Uklízeč – finální podoba

9 MICRO MOUSE

9.1 Zadání

Vymodelujte nebo použijte importovaného robota z prostředí Webots, který za co nejrychlejší čas nalezne správnou cestu v bludišti až do cíle.

Vašim úkolem je nalézt správnou a co nejkratší cestou skrze krkolomné bludiště až do cíle. Správná cesta do cíle je pouze jedna, ovšem krkolomných zatáček a pastí je zde spousta. Pokuste se optimalizovat Váš algoritmus na co nejlepší orientaci v prostoru bludiště.

9.2 Program

9.2.1 Inicializace a definovaní proměnných

V této části kódu includujeme potřebné knihovny a definujeme potřebné konstanty. Konstanty TIME_STEP, MAX_SPEED a WALL_DISTANCE_THRESHOLD určují čas, maximální rychlost motorů a vzdálenost pro detekci stěn.

```
1. #include <iostream>
2. #include <webots/Robot.hpp>
3. #include <webots/Motor.hpp>
4. #include <webots/DistanceSensor.hpp>
5. #define TIME_STEP 64
6. #define MAX_SPEED 6.28
7. #define WALL_DISTANCE_THRESHOLD 0.2
8. using namespace webots;
9. enum Direction { NORTH, EAST, SOUTH, WEST };
```

9.2.2 Inicializace robota a senzorů

Proměnné leftMotor a rightMotor jsou inicializovány pomocí metod getMotor, což umožnuje ovládání kol. Dále jsou zde pomocí frontSensor a rightSensor inicializovány a aktivovány senzory, což robotu umožnuje detekovat stěny a překážky.

```
1. Robot *robot = new Robot();
2. Motor *leftMotor = robot->getMotor("left wheel motor");
3. Motor *rightMotor = robot->getMotor("right wheel motor");
4. DistanceSensor *frontSensor = robot-
   >getDistanceSensor("front sensor");
5. frontSensor->enable(TIME_STEP);
6. DistanceSensor *rightSensor = robot-
   >getDistanceSensor("right sensor");
7. rightSensor->enable(TIME_STEP);
8. Direction direction = EAST;
9. bool wallOnRight = false;
```

9.2.3 Hlavní smyčka

V tomto kroku program přechází do nekonečné smyčky, což má za následek, že robot může neustále reagovat na změny v prostředí. Robot neustále čte hodnoty ze senzorů, aby zjistil přítomnost stěn. Na základě přítomnosti stěny na pravé straně bud zatáčí doleva, pokud je stěna příliš blízko, nebo pokračuje rovně. Pokud robot detekuje stěnu před sebou, otočí se doleva.

```
while (robot->step(TIME_STEP) != -1) {
  // Čtení hodnoty z ultrazvukových senzorů
  double frontDistance = frontSensor->getValue();
  double rightDistance = rightSensor->getValue();
  // Detekce stěny
  if (rightDistance < WALL_DISTANCE_THRESHOLD) {
    wallOnRight = true;
  } else {
    wallOnRight = false;
      }
      // Průchod bludištěm pomocí pravé ruky
      if (wallOnRight) {
        // Zatáčení doleva, protože zed je na pravém
  senzoru
        leftMotor->setVelocity(-MAX_SPEED);
        rightMotor->setVelocity(MAX_SPEED);
      } else {
        // Pokračování vpřed, protože zed není na pravém
  senzoru
        leftMotor->setVelocity(MAX_SPEED);
        rightMotor->setVelocity(MAX_SPEED);
      }
      // Kontrola, zda je cesta volná vpřed
      if (frontDistance < WALL_DISTANCE_THRESHOLD) {
        // Robot narazil na stěnu, musí provést otočku
        leftMotor->setVelocity(-MAX_SPEED);
        rightMotor->setVelocity(MAX_SPEED);
        robot->step(1000); // Otočka po dobu 1 sekundy
        direction = EAST;
      }
    }

```

9.2.4 Kompletní program v C++

```
#include <iostream>
#include <webots/Robot.hpp>
#include <webots/Motor.hpp>
#include <webots/DistanceSensor.hpp>
#define TIME_STEP 64
#define MAX_SPEED 6.28
#define WALL_DISTANCE_THRESHOLD 0.2
using namespace webots;
enum Direction { NORTH, EAST, SOUTH, WEST };
    int main() {
    // Inicializace robota
    Robot *robot = new Robot();
    // Inicializace motorů
    Motor *leftMotor = robot->getMotor("left wheel motor");
```

```
15.     Motor *rightMotor = robot->getMotor("right wheel
  motor");
16.     // Inicializace ultrazvukových senzorů
17.     DistanceSensor *frontSensor = robot-
  >getDistanceSensor("front sensor");
18.     frontSensor->enable(TIME_STEP);
19.     DistanceSensor *rightSensor = robot-
  >getDistanceSensor("right sensor");
20.     rightSensor->enable(TIME_STEP);
21.     Direction direction = EAST; // Počáteční směr robota
22.     bool wallOnRight = false; // Přítomnost zdi napravo od
  robota
23.     // Hlavní smyčka programu
24.     while (robot->step(TIME_STEP) != -1) {
25.     // Čtení hodnoty z ultrazvukových senzorů
26.     double frontDistance = frontSensor->getValue();
27.     ouble rightDistance = rightSensor->getValue();
28.     // Detekce stěny
29.     if (rightDistance < WALL_DISTANCE_THRESHOLD) {
30.     wallOnRight = true;
31.     } else {
32.     wallOnRight = false;
33.     }
34.     // Průchod bludištěm pomocí pravé ruky
35.     if (wallOnRight) {
36.     // Zatáčení doleva, protože zed je na pravém senzoru
37.     leftMotor->setVelocity(-MAX_SPEED);
38.     rightMotor->setVelocity(MAX_SPEED);
39.     } else {
40.     // Pokračování vpřed, protože zed není na pravém
  senzoru
41.     leftMotor->setVelocity(MAX_SPEED);
42.     rightMotor->setVelocity(MAX_SPEED);
43.     }
44.     // Kontrola, zda je cesta volná vpřed
45.     if (frontDistance < WALL_DISTANCE_THRESHOLD) {
46.     // Robot narazil na stěnu, musí provést otočku
47.     // Otočka doleva
48.     leftMotor->setVelocity(-MAX_SPEED);
49.     rightMotor->setVelocity(MAX_SPEED);
50.     robot->step(1000); // Otočka po dobu 1 sekundy
51.     }
52.
53.     }
54.     delete robot;
55.     return 0;
```

9.2.5 Celý program v Pythonu

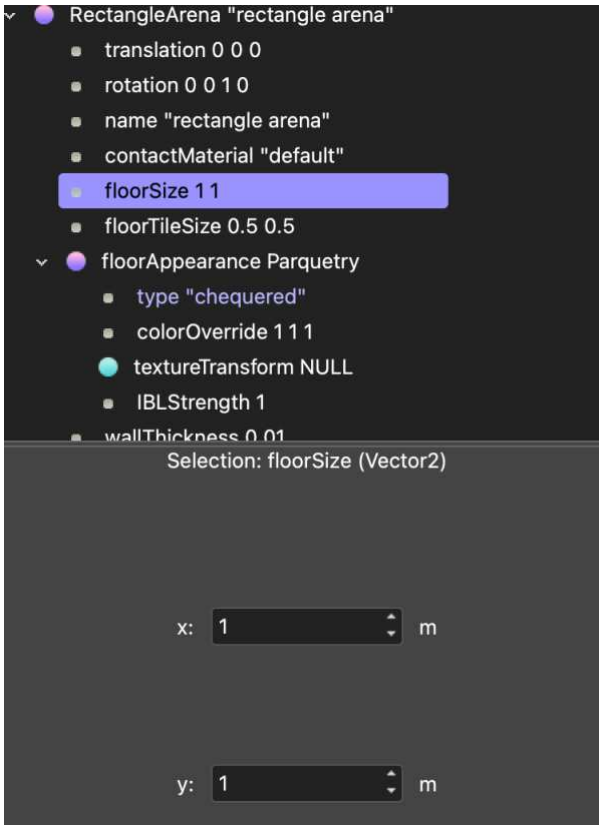
```
1. from controller import Robot, Motor, DistanceSensor
2. TIME_STEP = 64
3. MAX_SPEED = 6.28
4. WALL_DISTANCE_THRESHOLD = 0.2
5. class Direction:
6. NORTH = 0
7. EAST = 1
8. SOUTH = 2
9. WEST = 3
10.     # Inicializace robota
11.     robot = Robot()
12.     # Inicializace motorů
13.     left_motor = robot.getMotor("left wheel motor")
```

```
    right_motor = robot.getMotor("right wheel motor")
    # Inicializace ultrazvukových senzorů
    front_sensor = robot.getDistanceSensor("front sensor")
    front_sensor.enable(TIME_STEP)
    right_sensor = robot.getDistanceSensor("right sensor")
    right_sensor.enable(TIME_STEP)
    direction = Direction.EAST # Počáteční směr robota
    wall_on_right = False # Přítomnost zdi napravo od
  robota
    # Hlavní smyčka programu
    while robot.step(TIME_STEP) != -1:
    # Čtení hodnoty z ultrazvukových senzorů
    front_distance = front_sensor.getValue()
    right_distance = right_sensor.getValue()
    # Detekce stěny
    if right_distance < WALL_DISTANCE_THRESHOLD:
    wall_on_right = True
    else:
        wall_on_right = False
    # Průchod bludištěm pomocí pravé ruky
    if wall_on_right:
    # Zatáčení doleva, protože zeď je na pravém senzoru
    left_motor.setVelocity(-MAX_SPEED)
    right_motor.setVelocity(MAX_SPEED)
    else:
    # Pokračování vpřed, protože zeď není na pravém senzoru
    left_motor.setVelocity(MAX_SPEED)
    right_motor.setVelocity(MAX_SPEED)
    # Kontrola, zda je cesta volná vpřed
    if front_distance < WALL_DISTANCE_THRESHOLD:
    # Robot narazil na zeď, musí provést otočku
    # Otočka doleva
    left_motor.setVelocity(-MAX_SPEED)
    right_motor.setVelocity(MAX_SPEED)
    robot.step(1000) # Otočka po dobu 1 sekundy
```

9.3 Model

Při modelování tohoto zadání začneme stejně jako ve většině předešlých úkolů. Při konfiguraci projektu si vybereme v možnostech variantu přidání čtvercové hrací plochy. Po úspěšné konfiguraci a nastavení projektu budeme muset jako první krok zvětšit hrací pole. Předem stanovená velikost 1 x 1 metr nám pro tuto disciplínu nebude stačit. Oficiální hrací pole při robotické soutěži Robogames na FAI má 2.68 m x 2.68 m. V našem případě pro využití simulace nám bude stačit rozměr 2 m x 2 m.

Této změny dosáhneme tak, že v levém hlavním panelu s nabídkou si rozbalíme kolonku RectangleArena. Po rozbalení nalezneme možnost floorSize. Po rozkliknutí této možnosti změníme souřadnice x a y na námi požadovanou hodnotu, tudíž na 2 m.



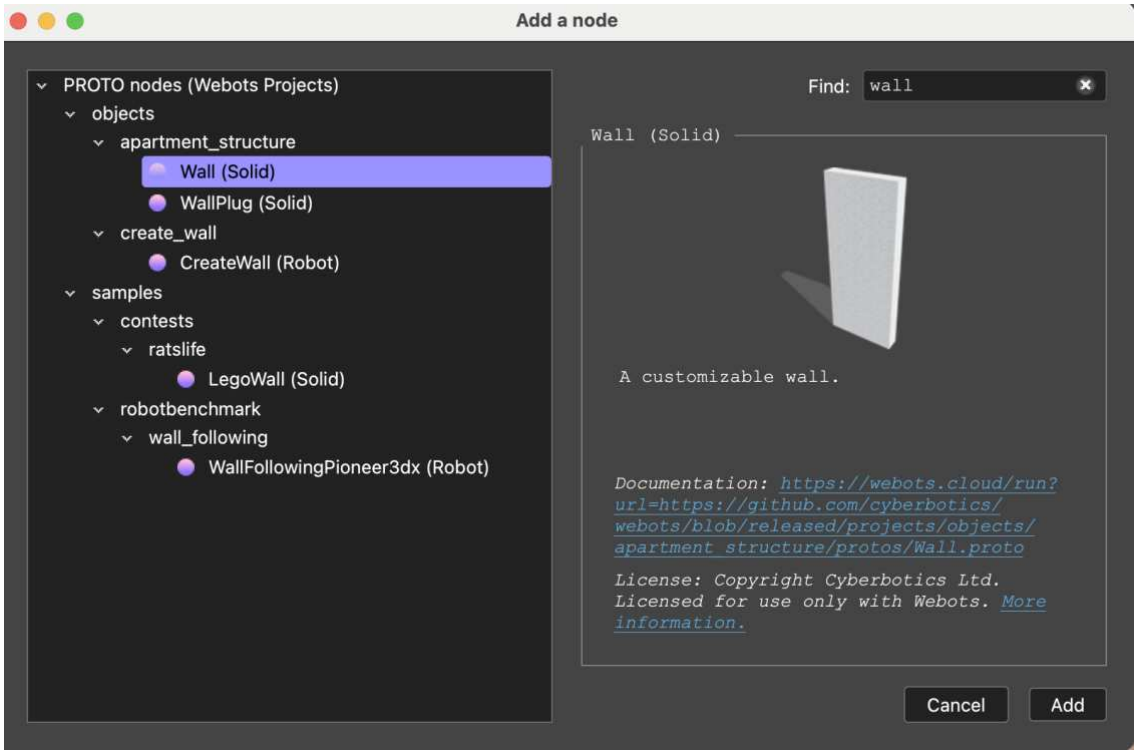
Obrázek 46 - MicroMouse – Změna velikosti hřiště

Po úspěšném změnění velikosti hrací plochy se vrhneme na vytváření bludiště, které bude robot muset projet, aby se dostal do cíle. Zde máme na výběr ze dvou variant, jak toto bludiště můžeme vytvořit. První varianta je ta, že můžeme přidat Solid a následně v geometry upravit jeho parametr na čtverec, který dále upravíme přes jeho rozměry na tvar

obdélníku. Z tohoto vytvořeného obdélníku následně můžeme tvořit zdi, které bude muset robot projíždět a louskat záludnost tohoto zadání.

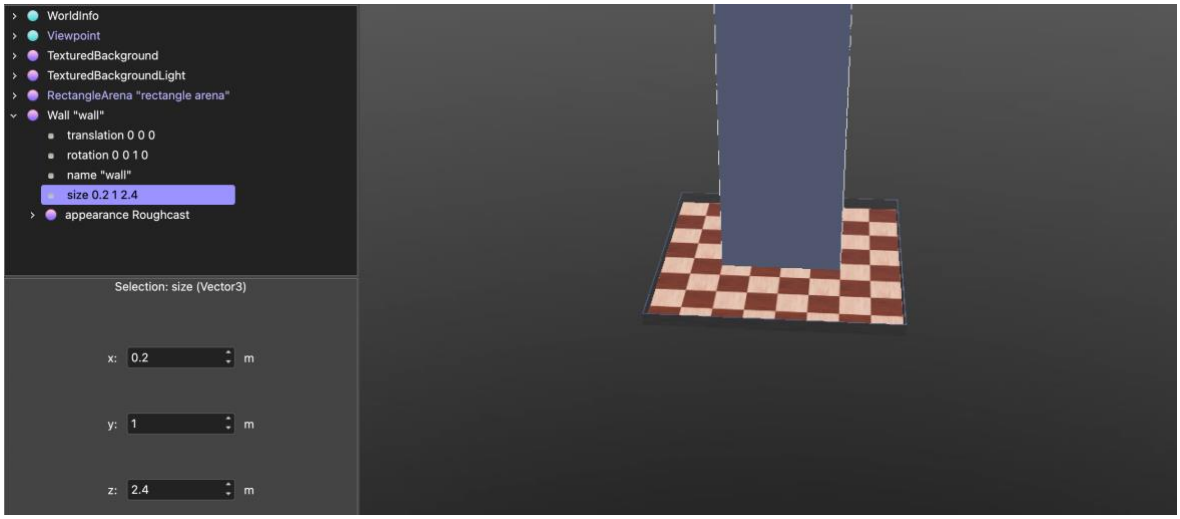
Druhou možnou variantou, jak vystavět zdi v bludišti je přímo komponenta zdi [9], která je obsažena v tomto robotickém simulátoru. Pro výstavbu našeho bludiště využijeme právě tuto druhou možnost.

Pro přidání zdi do projektu klikneme v horní liště klasicky na tlačítko přidat. Po zobrazení okna použijeme poličko pro hledání, které je umístěno v pravém horním rohu. Zde napíšeme klíčové slovíčko wall, které nám najde námi požadovanou zeď. Tu již jen nyní přidáme do projektu a můžeme pokračovat v dalších úpravách.



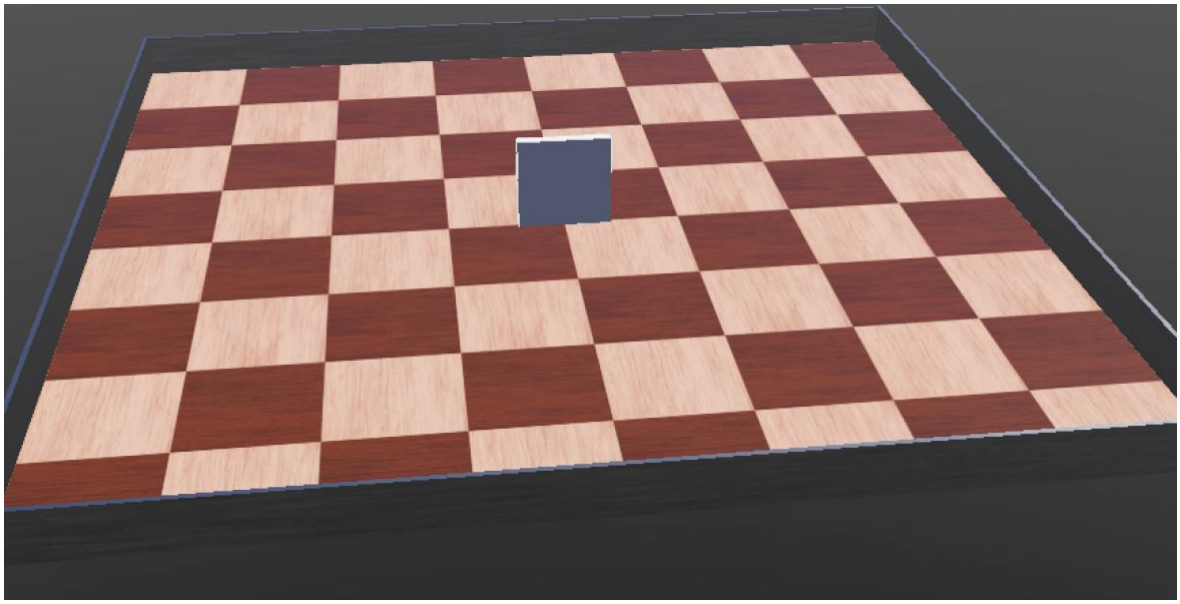
Obrázek 47 - MicroMouse – wall

Jelikož má právě přidaná zeď poněkud větší rozměr, než potřebujeme a jsou předem definované, musíme jejich velikost upravit tak, aby nám seděla do projektu. Najdeme si tedy v menu po levé straně komponentu wall, kterou rozklikneme. Po rozkliknuti si v rozšířené nabídce vybereme možnost size, zde upravíme velikosti tak, aby byly příslušné a idéalní pro naše zadání robotického bludiště.



Obrázek 48 - MicroMouse – úprava zdi

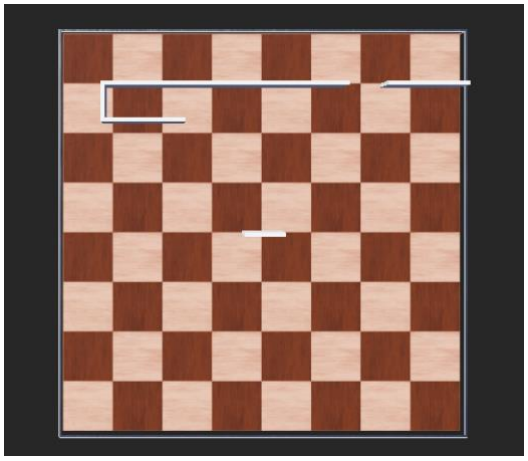
Po upravení zdi na požadované parametry nám vznikne ideální rozměr, který budeme nadále využívat.



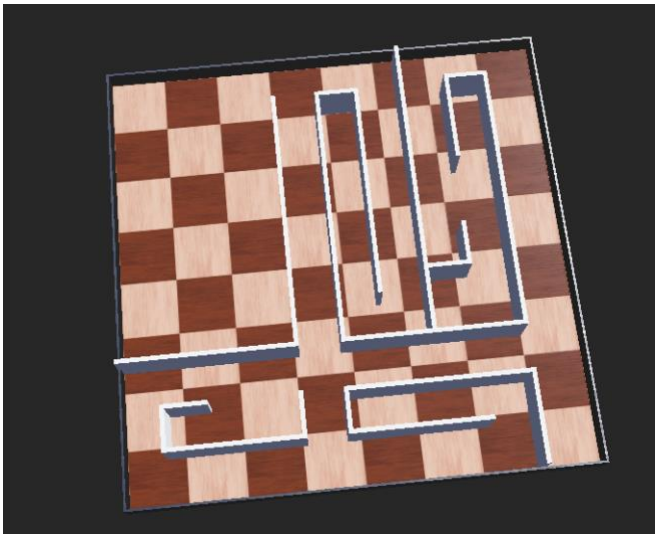
Obrázek 49 - MicroMouse – Finální podoba zdi

Nyní nám již nezbývá nic jiného než začít z těchto námi upravených bloků vyskládat celé bludiště.

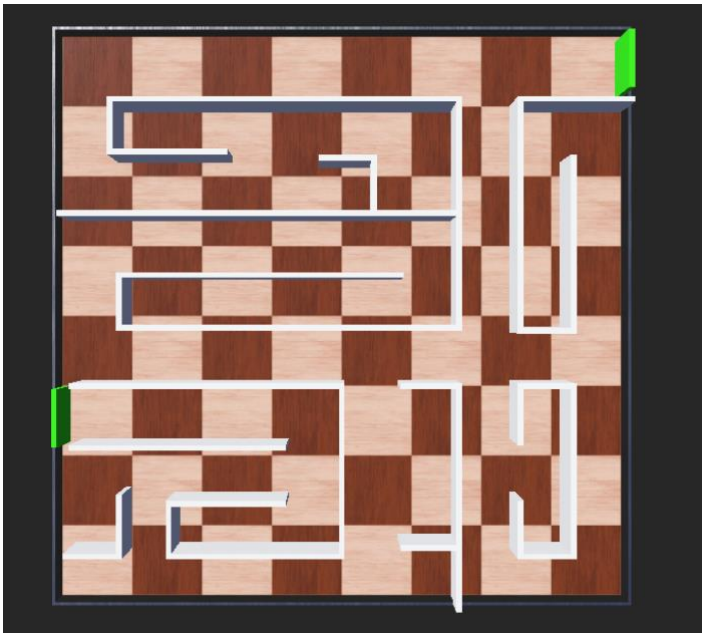
Na průběh stavby bludiště se můžete podívat v následujících krocích.



Obrázek 50 - MicroMouse – stavba bludiště 1



Obrázek 51 - MicroMouse – stavba bludiště 2



Obrázek 52 - MicroMouse – Finální podoba

10 DRAG RACE

10.1 Zadání

Vymodelujte nebo použijte importovaného robota z prostředí Webots, který ujede za co nejkratší dobu předem stanovenou dráhu.

Spálené kola, rychlé motory a doby jako za časů druhé poloviny 20. století na území Ameriky. Vžijte se díky této disciplíně do dětských snů a rychlých závodů nadupaných sporťáků. Každý z nás dozajista snil usednout do nejrychlejšího auta a zazávodit si sním. Nyní máte tu možnost v podobě simulace závodů Drag race. Vašim úkolem je za co nejkratší dobu ujet dráhu vzdálenou 10 metrů a zvítězit.

10.2 Program

Tento program je oproti ostatním jednodušší, tudíž si jej nemusíme rozdělovat na více kapitol. Jedná se pouze o to, aby robot ujel za co nejkratší čas danou vzdálenost. Musíme si zde nastavit požadovanou rychlost, oba motory však musí být nastaveny na stejnou rychlost, aby robot jel rovně. Pokud bychom nastavili hodnoty rozdílně náš robot by zatáčel, a to je nežádoucí jev v této disciplíně. Dále v hlavní smyčce vytvoříme nekonečnou smyčku pomocí while, tím zajistíme, že robot pojede do doby, než jej my fyzicky zastavíme, tímto krokem reagujeme na to, že robot může ujet jakoukoliv vzdálenost.

```
1. #include <webots/Robot.hpp>
2. #include <webots/Motor.hpp>
3.
4. #define TIME_STEP 32
5. #define MAX_SPEED 6.0
6.
7. using namespace webots;
8.
9. int main(int argc, char **argv) {
10.         Robot *robot = new Robot();
11.
12.         // Získání ukazatelů na motory
13.         Motor *leftMotor = robot->getMotor("left wheel
    motor");
14.         Motor *rightMotor = robot->getMotor("right wheel
    motor");
15.
16.         // Nastavení motorů na nekonečnou rotaci
17.         leftMotor->setPosition(INFINITY);
18.         rightMotor->setPosition(INFINITY);
19.
20.         // Nastavení stejné rychlosti pro oba motory pro
    jízdu rovně
21.         leftMotor->setVelocity(MAX_SPEED);
22.         rightMotor->setVelocity(MAX_SPEED);
```

```
23.
24.         // Hlavní smyčka programu
25.         while (robot->step(TIME_STEP) != -1) {
26.
27.         }
28.
29.         delete robot;
30.         return 0;
31.     }
```

10.2.1 Kompletní kód v Pythonu

```
1. from controller import Robot, Motor
2.
3. TIME_STEP = 32
4. MAX_SPEED = 6.0
5.
6. # Inicializace instance robota
7. robot = Robot()
8.
9. # Získání ukazatelů na motory
10.     leftMotor = robot.getDevice('left wheel motor')
11.     rightMotor = robot.getDevice('right wheel motor')
12.
13.     # Nastavení motorů na nekonečnou rotaci
14.     leftMotor.setPosition(float('inf'))
15.     rightMotor.setPosition(float('inf'))
16.
17.     # Nastavení stejné rychlosti pro oba motory pro jízdu
    rovně
18.     leftMotor.setVelocity(MAX_SPEED)
19.     rightMotor.setVelocity(MAX_SPEED)
20.
21.     # Hlavní smyčka programu
22.     while robot.step(TIME_STEP) != -1:
23.         pass  # Robot bude pokračovat v jízdě rovně, dokud
    není program zastaven
```

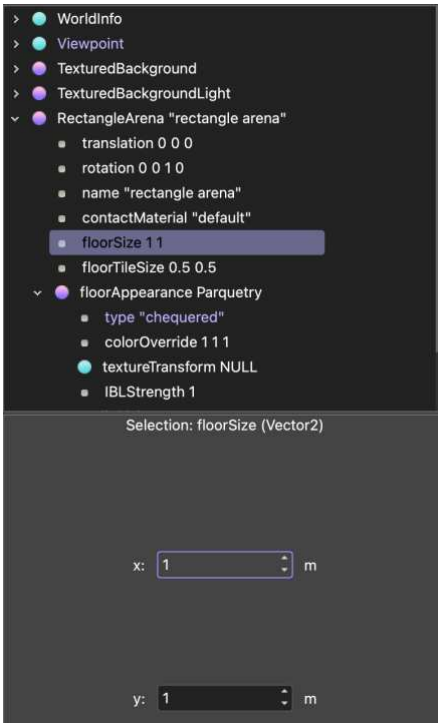
10.3 Model

Tato soutěžní disciplína je velmi oblíbená skrze její na první pohled nízkou náročnost a jednoduchost. Ovšem zdání zde může častokrát klamat. Na první pohled vypadá velmi jednoduše projet 10 metrů rovnou dráhu bez narazení na boční svodidla. Její prvopočátky se inspirovali velmi oblíbenými a rozšířenými DragRace závody ve Spojených státech Amerických, kde již od poloviny 20. století hojně bují zájem o tento sport.

Naše soutěžní dráha, kterou si vytvoříme v robotickém simulátoru Webots, bude mít 10 metrů na délku a 1 metr na šířku.

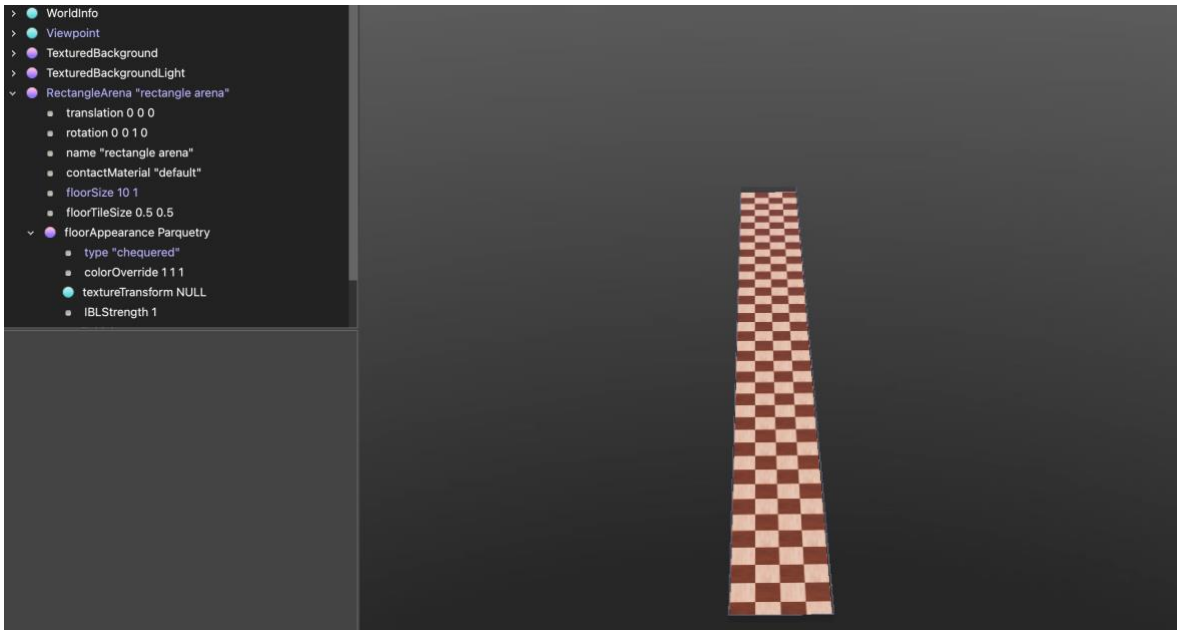
Jako první krok si klasicky otevřeme simulátor Webots, spustíme nový projekt a přidáme rectangle arena, kterou dále budeme konfigurovat a upravovat na námi zvolené požadavky.

Po spuštění programu si v pravé části nabídky vybereme možnost RectangleArena, kterou si rozbalíme. Po rozbalení další nabídky vybereme možnost floorsize. Po vybrání této nabídky nám v dolní liště vyjededou dva textboxy se souřadnicemi x a y. Právě tyto dvě hodnoty budeme měnit, abychom dostali námi požadovanou velikost závodní plochy.



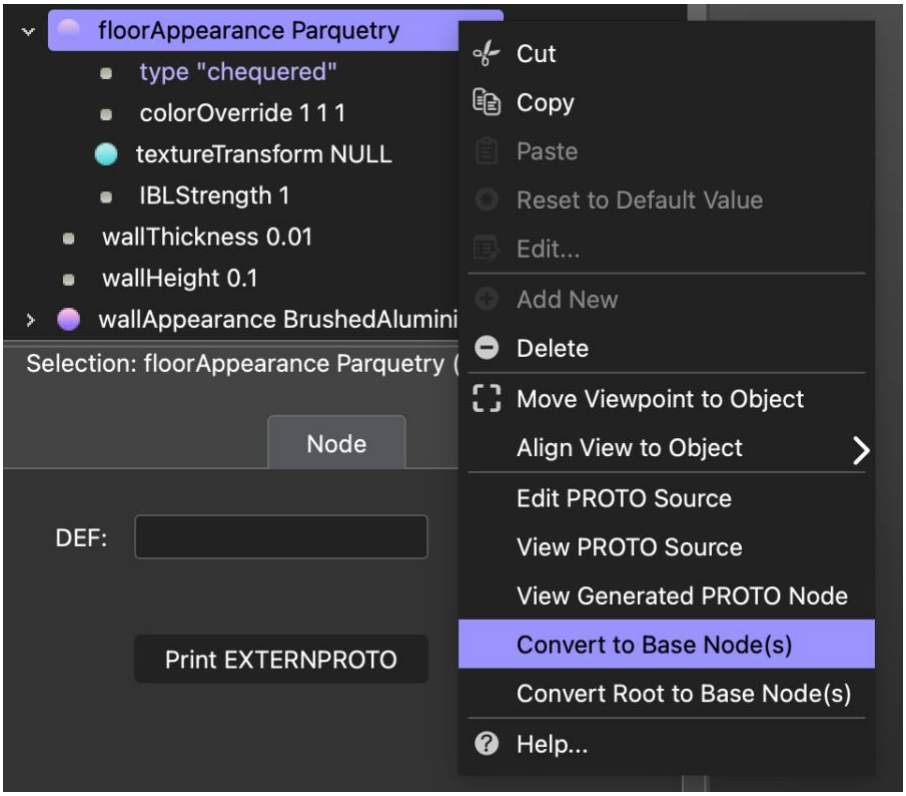
Obrázek 53 - DragRace – úprava souřadnic

Po upravení souřadnice x na velikost 10 a upravení souřadnice y na hodnotu 1 dostaneme námi požadovanou velikost hrací plochy.



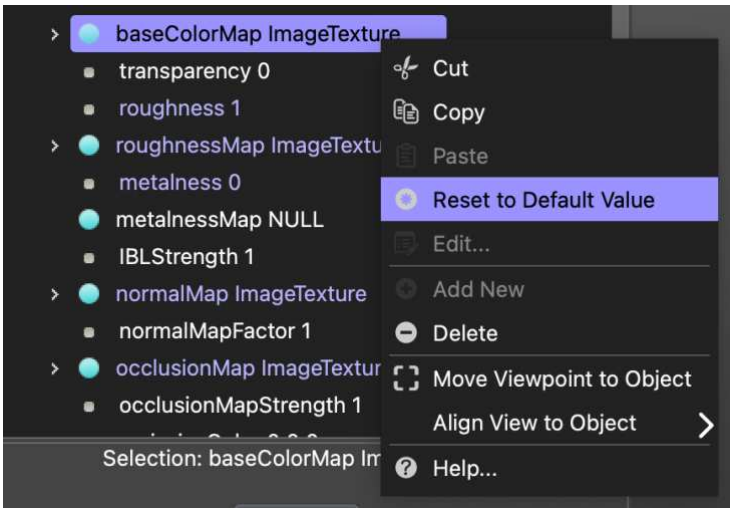
Obrázek 54 - DragRace – Upravená hrací plocha

Nyní máme rozměry dráhy přizpůsobené tak, jak potřebujeme. Stále ovšem máme defaultně vytvořené čtvercové pozadí plochy, které se nám příliš nehodí pro tuto disciplínu. Nejlépe je zvolit čistě bílé pozadí. Tuto změnu uděláme přes rozšířenou nabídku v liště na pravé straně. Najdeme si v možnostech floorAppearanceParquetry. Tuto možnost rozklikneme přes pravé tlačítko myši abychom získali další možnosti úpravy této varianty. Po dalším rozkliknutí se nám objeví rozšířené možnosti a zde vybereme možnost Convert to Base Nodes. Po vybrání této nabídky se nám změní možnosti, které jsou dostupné. Tento krok jsme potřebovali udělat skrze další nastavení a úpravy hrací plochy



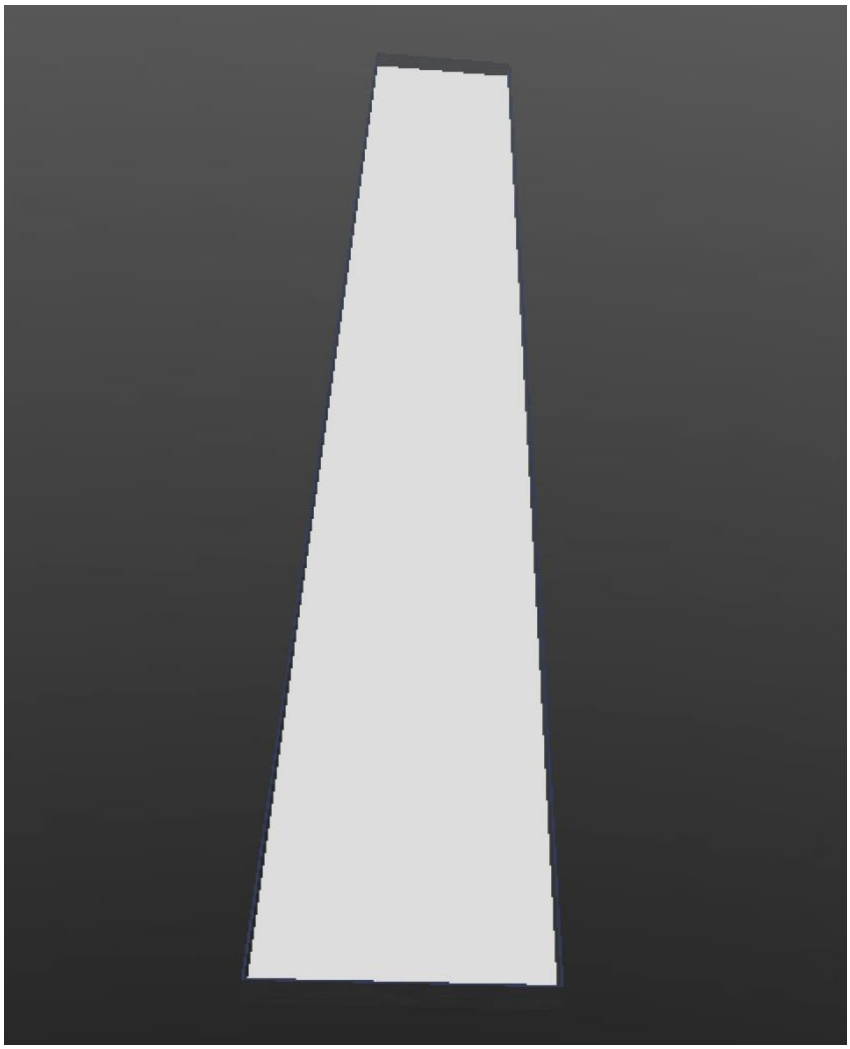
Obrázek 55 - DragRace – Rozšíření nabídky

Po provedení těchto kroků se nám v nabídce změní možnosti, se kterými můžeme pracovat. Nyní zde máme námi potřebnou komponentu baseColorMap. Na tuto komponentu opět najedeme myší a pomocí pravého tlačítka rozklikneme její rozšířené možnosti. Zde si vyhledáme možnost Reset to Default Value. Tuto možnost potvrdíme a nastavíme tímto krokem hodnotu pozadí plochy na bílou variantu.



Obrázek 56 - DragRace – Finální změna barvy plochy

Po úspěšném splnění všech kroků a podmínek, které jsme si ukázali v předešlých krocích dostaneme námi požadovanou a výslednou variantu závodní hrací plochy.



Obrázek 57 - DragRace – Finální podoba hrací plochy

ZÁVĚR

V této diplomové práci jsem se zabýval tématikou a problematikou použití robotických simulátorů ve výuce. Hlavním úkolem bylo prozkoumat a následně porovnat dva různé robotické simulátory, najít jejich výhody a nevýhody, následně je porovnat a vybrat ten, který se jeví jako vhodnější pro výukové účely. Vítězem porovnání se stal robotický simulátor Webots díky jeho větší přívětivosti pro námi hledané účely. Dále jsem v této práci navrhl a vytvořil 3D modely prostředí. Pro jednotlivé příklady z oblasti robotických soutěží jsem implementoval ukázkové řídicí kódy zadaní a následně vytvořil návod řešení.

Výstupem této práce jsou hotové 3D modely pracovních prostředí pro robotický simulátor Webots, ukázkové řídicí kódy a návod k postupu řešení.

SEZNAM POUŽITÉ LITERATURY

[1] *NXT-G*. Online. Alternativeto. 2022. Dostupné z: https://alternativeto.net/software/nxt-g/. [cit. 2024-02-10].

[2] *RobotC*. Online. Alternativeto. 2022. Dostupné z: https://alternativeto.net/software/robotc/about/. [cit. 2024-02-10].

[3] *Lego EV3*. Online. Software informer. 2022. Dostupné z: https://lego-mindstorms-ev3.software.informer.com/1.4/. [cit. 2024-02-10].

[4] *LEGO Spike Prime or LEGO MindStorms Inventor?* Online. Thecodingfun. 2022. Dostupné z: https://thecodingfun.com/2021/03/07/which-new-model-to-choose-from-lego-spike-prime-or-lego-mindstorms-inventor/. [cit. 2024-02-11].

[5] *Robogames pravidla*. Online. Robogames.utb.cz. 2024. Dostupné z: https://robogames.utb.cz/pravidla/. [cit. 2024-02-20].

[6] *Webots*. Online. Wikipedia. 2024. Dostupné z: https://en.wikipedia.org/wiki/Webots. [cit. 2024-04-20].

[7] *Webots*. Online. Cyberbotics. 2024. Dostupné z: https://cyberbotics.com. [cit. 2024-04-20].

[8] WUNDERLICH, David. *HW vývojový systém pro robotiku*. Diplomová. Zlín, 2017.

[9] *Wall*. Online. Webots.cloud. 2023. Dostupné z: https://webots.cloud/run?url=https://github.com/cyberbotics/webots/blob/released/projects/objects/apartment_structure/protos/Wall.proto. [cit. 2024-04-24].

[10] *Rectangle Arena*. Online. Webots.cloud. 2023. Dostupné z: https://webots.cloud/run?version=R2023b&url=https%3A%2F%2Fgithub.com%2Fcyberbotics%2Fwebots%2Fblob%2Freleased%2Fprojects%2Fobjects%2Ffloors%2Fprotos%2FRectangleArena.proto. [cit. 2024-04-24].

[11] *CoppeliaSim User Manual*. Online. Manual.coppeliarobotics.com. 2024. Dostupné z: https://manual.coppeliarobotics.com/. [cit. 2024-04-11].

[12] *CoppeliaSim*. Online. Wikipedia. 2023. Dostupné z: https://en.wikipedia.org/wiki/CoppeliaSim. [cit. 2024-04-24].

[13] *Getting started with CoppeliaSim Simulator*. Online. Hades.mech.northwestern.edu/. 2023. Dostupné z: https://hades.mech.northwestern.edu/index.php/Getting_Started_with_the_CoppeliaSim_Simulator. [cit. 2024-04-01].

[14] *Webots User Guide*. Online. Cyberbotics. 2024. Dostupné z: https://cyberbotics.com/doc/guide/installing-webots. [cit. 2024-04-25].

[15] *Getting Started with Webots*. Online. Cyberbotics. 2024. Dostupné z: https://cyberbotics.com/doc/guide/getting-started-with-webots. [cit. 2024-04-25].

[16] *The Scene Tree*. Online. Cyberbotics. 2024. Dostupné z: https://cyberbotics.com/doc/guide/the-scene-tree. [cit. 2024-04-25].

[17] *Assets*. Online. Cyberbotics. 2024. Dostupné z: https://cyberbotics.com/doc/guide/assets. [cit. 2024-04-25].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

PID Proporcionální Integrační Derivační

MM Micro Mouse

EXE Executable

DMG Disk Image

OS Operační systém

EDU Education

UTB Univerzita Tomáše Bati

FAI Fakulta aplikované informatiky

STL Standart triangle language

OBJ Object

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

prilohy.zip

PŘÍLOHA P I: PRILOHY.ZIP

LineFollower

SUMO

DragRace

MicroMouse

RobotUklizec

tinker