Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Sada úloh pro středoškolskou demonstrační laboratoř techniky

Plzeň 2022 Václav Šíma

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Václav ŠÍMA**
Osobní číslo:	**A19B0203P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Sada úloh pro středoškolskou demonstrační laboratoř techniky**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s hardwarovým vybavením demonstrační učebny techniky na Gymnáziu Sokolov, zejména s vybavením zaměřeným na robotiku. Prostudujte jejich parametry a způsob ovládání (API).
2. Navrhněte způsob, jak demonstrovat vlastnosti a schopnosti dostupných zařízení.
3. Zvolte dvě konkrétní zařízení (popř. kombinace zařízení) a vytvořte pro každé z nich zadání pro sadu úloh seznamujících návštěvníky laboratoře s funkcí daných zařízení. Sada se bude skládat ze základní úlohy, realizovatelné bez dohledu podle návodu, která návštěvníkovi laboratoře bude demonstrovat smysl zařízení, dále navazující úlohy, při jejímž řešení návštěvník získá kontrolu nad konkrétním zařízením, a konečně pokročilé úlohy (zhruba na úrovni SOČ).
4. Vytvořte referenční řešení navržených úloh.
5. Vytvořené metodické materiály a referenční řešení důkladně otestujte v demonstrační učebně techniky a popište vlastnosti vytvořených řešení a možnosti pro další rozvoj.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Doc. Ing. Libor Váša, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **4. října 2021**
Termín odevzdání bakalářské práce: **5. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 14. října 2021

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 4. května 2022

Václav Šíma

Abstract

This bachelor thesis describes how methodical materials for the technology laboratories at the Sokolov grammar school were made. These task assignments will serve as exercises for external grammar school visitors. They will also assist in teaching information technology subjects. From the set of available robots, the two most suitable candidates will be selected, for which tasks of different difficulties will be constructed. For each task instructions on how to solve the given problem will be created.

Abstrakt

Cílem práce je vytvořit metodické materiály do laboratoře robotiky na Gymnáziu Sokolov, které budou sloužit jako metodické pomůcky v rámci výuky a jako referenční úlohy pro externí návštěvníky gymnázia. Z množiny dostupných robotů budou zvoleni dva nejvhodnější kandidáti, ke kterým budou sestrojeny úlohy o různých obtížnostech a návody, jak dané úlohy vyřešit.

Poděkování

Rád bych poděkoval Doc. Ing. Liboru Vášovi, Ph.D. za cenné rady a věcné připomínky, které mi velmi pomohly při vypracování bakalářské práce.

Obsah

1 Úvod

1.1 Motivace

Zařazení práce s roboty do výuky umožňuje studentům získat praktickou zkušenost v odvětví robotiky, která má stále širší pole působnosti. Výhody využití robotů v praxi tkví především v eliminaci negativních lidských faktorů z hlediska pracovní efektivity. Konkrétní schopnosti, kterými roboti oplývají a lidé je postrádají jsou například nízká chybovost, snášenlivost zdraví ohrožujících podmínek na pracovišti nebo plnění rutinních úkolů v konzistentní rychlosti a kvalitě. Uplatnění robotiky je na aktuálním trhu práce rozsáhlé. Student, jenž disponuje znalostmi umožňující konfiguraci a ovládání robotů, se může angažovat do širokého spektra atraktivních pracovních pozic v různých sektorech trhu. Vytvoření vhodných metodických materiálů pak může pomoci studentům rozvinout zájem o samostudium této problematiky.

1.2 Cíle

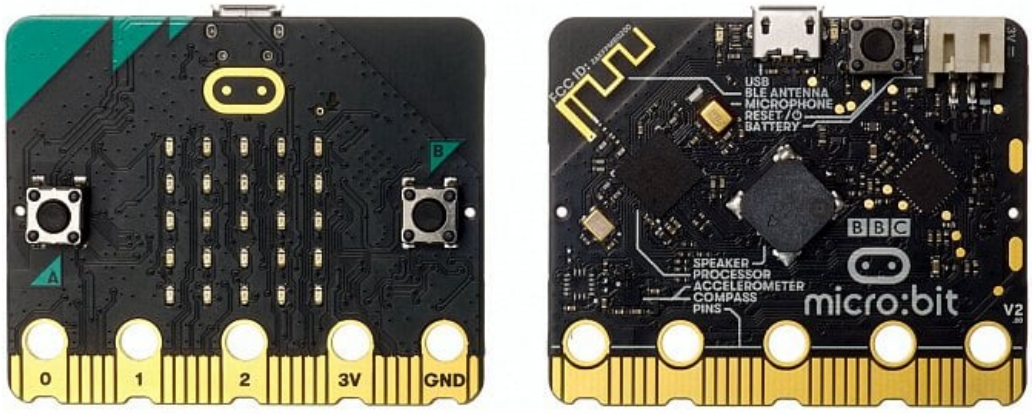
Cílem této práce je prozkoumat dostupné roboty na Gymnáziu Sokolov a vybrat dva konkrétní modely. Pro tyto dva roboty budou dále vytvořeny, zdokumentovány a otestovány sady úloh o různých obtížnostech. Konkrétně budou pro každého robota zhotoveny tři různě obtížné úlohy, které budou obsahovat přesně definované zadání úlohy, postup, jak danou úlohu zdárně dokončit a další možnosti rozšíření. Tyto výukové materiály se poté dále budou využívat ke vzdělávání návštěvníků a studentů Gymnázia Sokolov v rámci školních akcí a výuky. Možnosti využití těchto materiálů mohou být potenciálně i mimo Gymnázium Sokolov.

2 Analýza robotů

Po komunikaci se zástupci z Gymnázia Sokolov a získání seznamu všech dostupných robotických zařízení začal analytický rozbor robotů a jejich vlastností. K dispozici byl nejnovější *LEGO* robot ze série *LEGO Mindstorms*, *IoT* sada *Micro:bit* a dva menší a jednodušší roboti: *Ozobot* a *Edison*. V rámci analýzy každého z robotů budu zkoumat jejich hardwarové a softwarové vybavení, podporu pro programovací jazyky, potenciál pro škálování obtížnosti úloh a celkovou atraktivitu v rámci zapojení do výuky, kterou budu hodnotit zcela subjektivně. Z dostupných kandidátů pak vyberu dva nejvhodnější reprezentanty, pro které budu vymýšlet zadání konkrétních úloh.

2.1 BBC Micro:bit

Micro:bit je vestavěný systém s otevřeným zdrojovým kódem vyvinutý Britskou firmou *British Broadcasting Corporation*. Primárně se využívá jako pomůcka během výuky předmětů spojených s informačními technologiemi[2].



Obrázek 2.1: Ukázka *BBC Micro:bit*

Zdoj:`https://www.root.cz/zpravicky/novy-bbc-micro-bit-dostal-mikrofon-a-reproduktor/`

Hardwarová specifikace

Tento jednočipový počítač je od roku 2020 dostupný ve dvou verzích, lišících se zejména výkonem jednotlivých periferií. V laboratořích Gymnázia Sokolov se nachází starší a méně výkonná verze označená jako *"v1"*. Tato verze obsahuje následující komponenty:

- 32 bitový mikroprocesor *ARM Cortex-M0*
- tříosý akcelerometr
- tříosý magnetometr
- tři programovatelná tlačítka
- display složený z 5x5 matice LED diod
- Micro USB konektor
- Vstupní a výstupní konektory pro krokosvorky a jednopólové zástrčky

Novější verze rozšiřuje čip o výkonnější mikroprocesor, mikrofon, reproduktor a umožňuje přechod do úsporného režimu.

Softwarová specifikace

Počítač je programovatelný pomocí širokého spektra programovacích jazyků. Webové rozhraní umožňuje programovat pomocí vizuálního programovacího nástroje *Microsoft MakeCode* a skriptovacích jazyků *Python* a *JavaScript*, respektive *TypeScript*. Celý produkt disponuje poměrně širokou komunitou lidí, kteří vytváří další blokové editory, které pomáhají rozvíjet logické myšlení a návyky spojené s programováním, zejména u mladších žáků, kteří mají minimální zkušenosti s programováním. Pro zařízení s *iOS* je také možnost programovat v jazyce *Swift*.

Micro:bit classroom

Pro práci s *Micro:bity* v rámci výuky existuje dedikovaný nástroj *Micro:bit classroom*, který umožňuje efektivní správu studentských prací a prezentaci programů v rámci třídy. Nástroj využívá webové rozhraní a je dostupný z oficiálních webových stránek *Micro:bitu*. Funguje na principu místností, do kterých se žáci mohou připojit. Pro připojení do místnosti není vyžadováno přihlašování pomocí jména a hesla a odpadá tak nutnost registrace. Identifikace místnosti je realizována pouze pomocí přihlašovacího kódu. Po připojení má

každý žák svůj vlastní program, který je automaticky sdílený vyučujícímu, který tak má přehled o aktuálním stavu implementovaných programů celé třídy. Dále je zde také možnost pro vyučujícího poslat svůj kód konkrétnímu žákovi nebo všem žákům, popřípadě sdílet program některého ze studentů. V rámci nástroje jsou také připraveny jednoduché úlohy na různá témata, jako jsou dokončování rozpracovaných úloh, řešení primitivních problémů nebo odhalování chyb v programech. Celý nástroj je velice intuitivní, nevyžaduje žádnou instalaci a je zdarma dostupný. Díky dostupnému emulátoru není pro realizaci některých úloh nutné vlastnit ani fyzickou verzi *Micro:bitu*.

2.2 Ozobot

Ozobot je robot ve tvaru polokoule programovatelný pomocí sekvencí barevných kódů. Slouží především jako edukační pomůcka během výuky předmětů spojených s informačními technologiemi, zejména s programováním.

Hardwarová specifikace

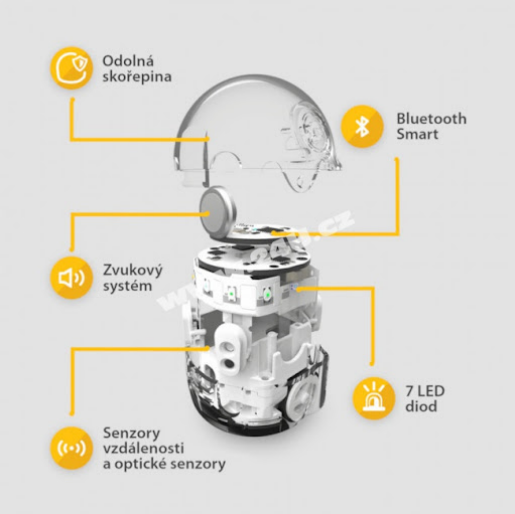
Zespodu robota jsou umístěny optické senzory, které umožňují robotovi sledoval nakreslenou linii a rozpoznat její barvu. Jeho hlavní funkčnost je pohybovat se po nakreslené trajektorii pomocí koleček poháněných elektromotorem a reagovat na barevné sekvence a křižovatky. Některé barevné kombinace mají speciální význam a umožňují robotovi zrychlit, zpomalit, rozhodnout o směru, kterým se vydá na další křižovatce, a mnoho dalších funkcí. Dostupné jsou dvě generace *Ozobota* a na Gymnáziu Sokolov je k dispozici novější z nich - *Ozobot BIT 2.0*. Tato novější generace je pak dále dělena do dvou verzí:

Ozobot EVO

Ozobot EVO je pokročilejší a dražší varianta *Ozobota*. Tato verze má oproti omezené variantě přidané senzory vzdálenosti, které dokáží zabránit kolizi mezi dvěma nebo více *Ozoboty*, Bluetooth konektivitu, která umožňuje ovládat robota z mobilní aplikace a obsahuje také reproduktor umožňující vydávání zvuků. Dále má dedikovanou aplikaci pro *Android* i *iOS*, která obsahuje úlohy a možnost kreslení vlastních virtuálních trajektorií pro robota. Jeho cena se ovšem pohybuje okolo čtyř tisíc korun. V rámci fanouškovské komunity má tato verze také podporu pro vytváření vlastních oblečků modifikujících vzhled *Ozobota*.

Ozobot BIT

Levnější a méně výkonná varianta *Ozobota*. Neobsahuje výše zmíněná rozšíření a tento typ je navíc před použitím nutné zkalibrovat pro použitý povrch. Podle něj se pak i odvíjí výdrž baterie. Po zbylé hardwarové stránce se zásadně neliší od první verze. Oproti pokročilejší verzi je téměř o polovinu levnější, jeho cena je kolem 2500 Kč.



Obrázek 2.2: *Schéma Ozobota*

Zdoj:`https://www.4kids.cz/ozobot-evo-programovatelny-robot-bily`

Ozoblockly

Do obou z výše zmíněných robotů lze nahrát vlastní program, vytvořený ve vizuálním editoru *Ozoblockly*. Ten umožňuje slučováním grafických elementů vytvořit plnohodnotný program bez syntaktických znalostí jakéhokoliv programovacího jazyka. *Ozoblockly* podporuje podmíněné příkazy, cykly, práci s proměnnými a mnoho dalších funkcí známých z klasických programovacích jazyků. Pro kreslení trajektorií je dostupná druhá aplikace *Ozobot Draw*, kde je možné vytvořit virtuální dráhu.

2.3 LEGO Mindstorms - robotí vynálezce

LEGO Mindstorms

LEGO Mindstorms je série stavebnic *LEGO*, specializovaná na programovatelné roboty. První model této stavebnice byl vydán v roce 1998[3]. Všichni roboti této série obsahují řídící počítač, dále zmiňovaný jako *hub*, který ovládá servomotory, senzory a další ovladatelné periferie. Série *LEGO Mindstorms* obsahuje 5 generací a v rámci vybavení na Gymnáziu Sokolov je k dispozici nejnovější verze robota, vydaná v roce 2020 - *Robotí Vynálezce* (v angličtině *Robot Inventor*).

Robotí vynálezce

Tento model je koncipován jako 5v1, tudíž v rámci celého *LEGO* setu lze zkonstruovat celkem 5 různých robotů s různým zaměřením. V následující sekci budou popsány vlastnosti každého z dostupných modelů.

Model robota - Blast

Nekomplexnější varianta robota a vlajková loď celého setu má přízvisko *Blast*. Jde o robota, který svou konstrukcí připomíná člověka. Pohybuje se pomocí dvou koleček poháněných elektromotory a pomocí diferenciálu, umístěného na hrudi, dokáže pohybovat rukama a hlavou současně. Konce jeho paží jsou modifikovatelné a lze na ně umístit různá rozšíření jako střelnou zbraň, kladivo nebo aparát umožňující robotovi uchopit předměty.

Model robota - Tricky

Jednoduchý robot pohybující se pomocí dvou kol umístěných po stranách robota. Zespodu robota je umístěný optický snímač pro detekci barev, umožňující robotovi pohyb po předkreslené trajektorii. Horní část robota je variabilní a lze na ni připevnit několik různých rozšíření. Nejkomplexnější z nich je aparát pro uchopení míčku, který poté lze pomocí dalšího motoru vyhodit do vzduchu, případně do košíku, který lze v rámci setu taktéž zkonstruovat.

Model robota - Gelo

Čtyřnohý robot, který svým vzhledem připomíná medvěda. Dokáže se pohybovat neobvyklým stylem kopírujícím pohyb čtyřnohých tvorů. Jeho konstrukce také umožňuje vyhýbání se překážkám a provádění triků, jako je například kotoul. Každá z jeho noh je řízena vlastním motorem a v přední

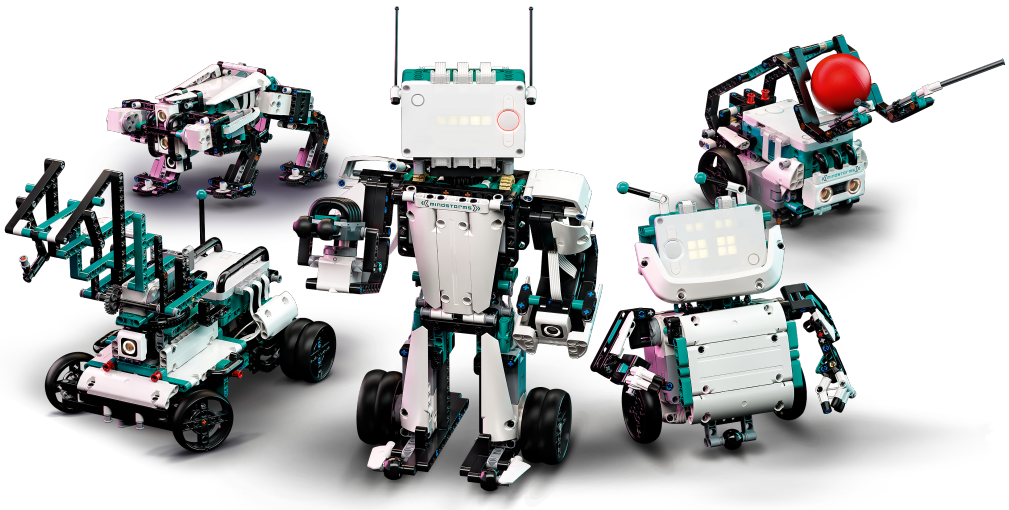
části robota je umístěn distanční a barevný senzor, které po vizuální připomínají hlavu. Po praktické stránce pak umožňují získat robotovi informace o objektech před ním.

Model robota - M.V.P.

Nejmodifikovatelnější model robota získal označení *M.V.P (Modular Vehicle Platform)*. Svým vzhledem připomíná automobil a na tomto konceptu staví i všechny jeho variace. Základ zůstává pro všechny varianty robota stejný, čtyři kola, každé poháněné jedním motorem. Šasi robota se ale mění, dle činnosti, ke které je postaven. Robota lze přestavit například na vysoko zdvižný vozík, buginu nebo tank.

Model robota - Charlie

Druhý humanoidní model robota, který je z mého pohledu nejméně komplexní. Jedná se o menší a jednodušší variantu prvního zmíněného modelu - *Blast*. Díky ušetřeným kostkám lze pak k robotovi zkonstruovat více přidružených objektů, se kterými může interagovat. Jedním z těchto objektů je například souprava bicích nástrojů, na které robot může po vytvoření příslušného programu zahrát.



Obrázek 2.3: *Všechny dostupné konfigurace LEGO robota*

Zdoj:`https://www.LEGO.com/cs-cz/product/robot-inventor-51515`

Ovládací blok

Hlavní komponenta robota je chytrý ovládací blok, obsahující baterii, šestiosý gyroskop, akcelerometr a podporuje *Bluetooth* konektivitu. V dolní části bloku je umístěn mikro *USB* konektor pro dobíjení baterie nebo připojení k počítači. Po stranách bloku jsou umístěny 3 páry portů pro připojení periferních zařízení pomocí kabelů se speciální *LEGO* koncovkou podobnou *RJ12* konektoru.

Konstrukce a nahraní programu

Podrobný návod k sestavení robota, kde je krok po kroku znázorněno, jak jednotlivé *LEGO* dílky pospojovat, lze najít v digitální podobě, v rámci *LEGO* aplikace. Sestavení robota v konfiguraci *Blast* trvalo okolo tří hodin. V rámci aplikace taktéž najdeme návod, jak si založit vlastní projekt a jak následně nahrát implementovaný program do robota. Dále jsou zde dostupné jednodušší úlohy, na kterých si lze procvičit základní programování robota.

Aplikace LEGO MINDSTORMS

Se stavebnicí je sdružená aplikace, ve které jsou stavební návody, zdokumentované implementované úlohy a komunitní stránka pro sdílení vlastních modifikací a programů. Mimo jiné je zde možné robota programovat v jazyce *Scratch* a v beta verzi je zde možná i implementace v jazyce *Python*. V rámci aplikace je možné se k robotovi připojit vzdáleně pomocí technologie *Bluetooth*.

2.4 Edison

Edison je robot ve tvaru kvádru o rozměrech 7,5 x 4 x 8,5 cm. Byl vytvořený hlavně jako pomůcka pro výuku informatiky. Obsahuje řadu senzorů, motory a diody, které jsou programovatelné pomocí různých nástrojů a jazyků. Robot je napájený pomocí čtyř *AAA* baterií a celé tělo robota, i jeho motory, jsou kompatibilní se stavebnicí *LEGO*, což umožňuje robota zakomponovat do komplexnějších projektů.

Specifikace vstupních zařízení

Vstupní periferie robota *Edison* jsou senzory a tlačítka. Senzory jsou dvojího typu. První z nich je světelný senzor a druhý infračervený světelný senzor. Oba se skládají z emitoru ve formě *LED* diody. Ta vyzařuje světlo, jež se odráží zpět a je snímáno senzorem vyhodnocujícím, v případě distančního senzoru, vzdálenost a v případě senzoru světelného, barvu podkladu. Infračervený senzor také umožňuje vzájemnou komunikaci mezi roboty nebo dálkové ovládaní robota pomocí speciálního ovladače. Druhým vstupem jsou pak klasická mechanická tlačítka, na jejichž stisk je možné reagovat. Vstupy jsou následně interpretovány pomocí tří druhů výstupů.

Specifikace výstupních zařízení

Mezi výstupy robota nalezneme *LED* diody, bzučák a elektromotor. Diody jsou umístěné v přední části robota, jedna na levé a jedna na pravé straně. Lze je nezávisle rozsvítit a mohou sloužit například jako indikátor chyby. Bzučák umožňuje robotovi vydávat tóny, což může být jeden ze způsobů, jak zakomponovat základy hudební výchovy v kolaboraci s výukou informatiky. Poslední možnost, jak interpretovat data, jsou dva elektromotory, opět jeden na levé a druhý na pravé straně robota. Motory lze ovládat opět nezávisle, což umožňuje robotovi otáčení a lze k nim připojit různé druhy rozšíření, jako například kola.

Softwarová specifikace

Edison je programovatelný pomocí tří různých jazyků, které jsou všechny zdarma dostupné z internetového prohlížeče. První dva z nich, *EdBlocks* a *EdScratch* jsou grafické blokové jazyky, určené především pro mladší programátory. *EdBlocks* je primitivnější verze s omezenějšími možnostmi a *EdScratch* je nadstavba nad programovacím jazykem *Scratch*, tudíž jeho případ užití je takřka totožný s tímto jazykem. Další programovací jazyk, který lze

pro programování robota použít je *EdPy*. Tento jazyk je založený na jazyku *Python* a umožňuje psát textové příkazy jako každý moderní programovací jazyk.

Obrázek 2.4: *Robot Edison*

Zdoj:`https://meetedison.com/product/edpack1/`

2.5 Závěr analýzy

V rámci výběru dvou nejvhodnějších kandidátů pro realizaci úloh jsem narazil na velké rozdíly v komplexnosti jednotlivých robotů. První faktor, který jsem chtěl zohlednit, byla podpora pro programování v textovém a současně i grafickém programovacím jazyce. V případě dostupnosti této funkcionality lze uplatnit tuto vlastnost jako jednu z možností jak škálovat obtížnost úloh. Tento požadavek však splňují všichni ze zmíněných robotů, tudíž ve finálním rozhodování nehrál roli.

LEGO robot má z mého úhlu pohledu nejširší možnosti pro vytvoření zajímavých úloh, tudíž se jeví jako ideální kandidát číslo jedna. Je to nejkomplexnější, nejdražší a vizuálně nejatraktivnější robot ze všech zmíněných zařízení, tudíž mi přišlo nesmyslné ho do projektu nezařadit. Naopak nejprimitivnějšího z robotů jsem shledal *Ozobota*. Tento robot má ze všech zmí-

něných nejméně funkcí a celý jeho koncept staví pouze na pohybu po předkreslené trajektorii. Bylo by tedy relativně obtížné vymyslet úlohy o třech úrovních obtížnosti.

Poslední rozhodování bylo mezi robotem *Edison* a mikročipem *BBC Micro:bit*. Hlavním důvod, proč jsem zvolil *Micro:bit* před *Edisonem*, bylo perfektně zpracované webové vývojové prostředí. Kombinace on-line emulátoru, aplikace *Micro:bit classroom* a široká škála možností užití mi přišly jako perfektní prerekvizity pro zařízení, na kterém budou realizovány demonstrační úlohy. Robot *Edison* mě naopak odradil vizuálně nepřívětivým grafickým rozhraním vývojové aplikace a menší škálou možností, které nabízí. Z výše zmíněných robotů jsem se tedy ve finále rozhodl úlohy realizovat na dvojici *LEGO Mindstorms* a *BBC Micro:bit*.

3 Pedagogický pohled

Tvorba plánů

Vzhled a kvalita metodických plánů může mít zásadní vliv na pozornost a vstřebávání informací studentů. Materiály by měly mít přímou souvislost s probíraným tématem, měly by být lehce pochopitelné a neměly by studenty přehltit nadbytečnými informacemi. Farrow doporučuje používat jednodušší jazykovou formu, poskládanou do krátkých vět a komplexnější témata popsat pomocí obrázků a diagramů[7].

Jeden z pohledů jak ohodnotit kvalitu metodických materiálů je pomocí motivace studentů. Keller tvrdí, že motivovaní žáci mají lepší studijní výsledky než žáci nemotivovaní[10]. Pro vytvoření efektivních materiálů je tedy třeba brát tento fakt v potaz a vytvořit materiály, které udrží studenty motivované. Model zabývající se tímto problémem se nazývá *ARCS* model. Zkratka modelu vznikla jako kombinace prvních písem anglických slov *attention* (A), *relevance* (R), *confidence* (C), a *satisfaction* (S), kde každé z písmen popisuje jeden pilíř, na kterém je systém založen[10].

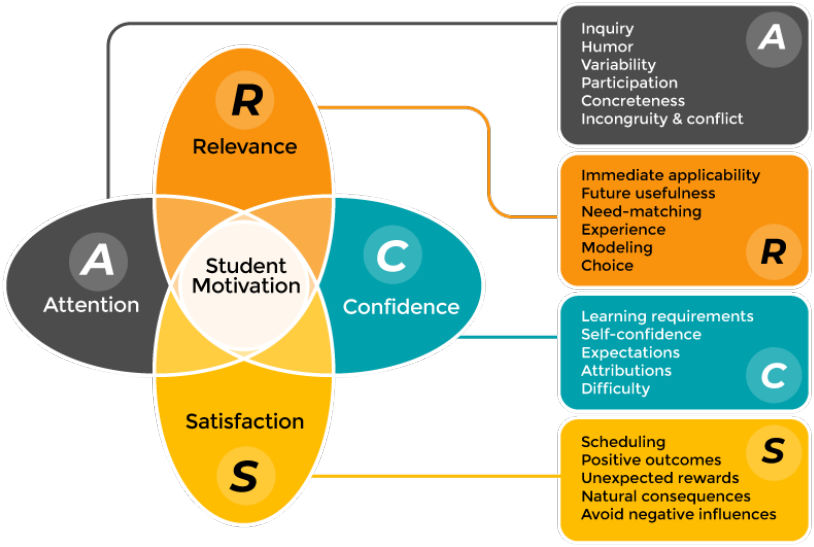
3.1 ARCS model

ARCS model staví na principu motivace. Motivace je klíčový faktor pro hlubší prohloubení znalostí studentů. Motivovaní studenti mají tendenci se vzdělávat nad rámec studijních požadavků, díky pozitivnímu emocionálnímu zážitku, vytvořeném při seznámení s danou problematikou[12].

Cílem je tedy studenty navnadit ke studiu vybraného problému tak, aby se samostatně vzdělávali. Toho lze dosáhnout díky interní motivaci, která vzniká při zájmu o dané téma. Aby model správně fungoval, je nutné, aby studenti pochopili, proč je informace, kterou se mají naučit důležitá, a jak ji dokáží v praxi reálně uplatnit.

Pozornost

Získání pozornosti je první krok pro vtažení studentů do studia. Autor metody *ARCS*, Keller tvrdí, že pozornost studentů lze získat pomocí netradičních výukových metod, které v studentech evokují pocit dobrodružství[10].



Obrázek 3.1: *Schéma modelu ARCS*

Zdoj:`https://www.flarelearning.com/post/22-leveraging-learning-theories-in-elearning`

Relevance

Pojem relevance v tomto konceptu definuje především spojení teoretické výuky s reálnou využitelností. Studenti mají vyšší motivaci získávat znalosti, které jsou v praxi užitečné, a proto je vhodné jejich aplikaci během výuky rovnou demonstrovat.

Sebejistota

Sebejistotu studenta lze podpořit přizpůsobením náročnosti dané látky studentovým aktuálním znalostem. Dále také může pomoci dopředné informování o náročnosti probírané látky, potřebných prerekvizitách k absolvování dané části výuky a postupu, jakým bude výsledek ohodnocen.

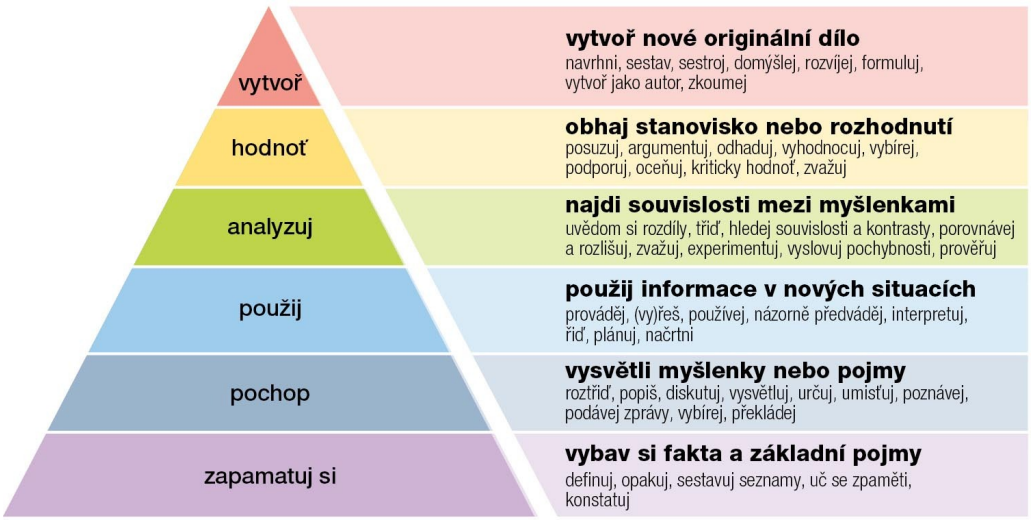
Uspokojení

Vytvoření uspokojení z nabytých znalostí je poslední část metodiky *ARCS*. Je to přirozený důsledek uplatnění osvojené schopnosti na externí úloze mimo výuku, kde student uvidí zhodnocení investovaného času do studia dané problematiky. Následky uspokojení jsou pak: vyšší pravděpodobnost samostudia, či získávání většího všeobecného rozhledu v rámci podobných

studijních odvětví. Pokud například student využije v rámci výuky informatiky některý matematický aparát, který ho do té doby příliš neoslovil, či ho vůbec nechápal, může se k němu po praktickém uplatnění vrátit a znovu prostudovat. Jelikož má teoretickou informaci, například matematický vzorec, asociovanou s reálným praktickým využitím, bude mít vyšší motivaci si teoretické téma prostudovat podrobněji.

3.2 Bloomova taxonomie

Další schéma rozdělující učební cíle do jednotlivých kategorií, dle komplexnosti, je Bloomova taxonomie. Struktura pyramidového schématu byla navržena Benjaminem Bloomem v roce 1956 a následně byla upravena do aktuálnějšího formátu jedním z jeho studentů v roce 1990[1]. Taxonomie je hierarchicky členěna do šesti úrovní dle kognitivní náročnosti a abstraktnosti. Každá z vyšších úrovní zahrnuje i principy z úrovně nižší, tudíž pro postoupení vyšších pater struktury je nutné ovládnout principy všech pater nižších[8].



Obrázek 3.2: *Bloomova taxonomie vzdělávacích cílů*

Zdoj:`https://vesmir.cz/cz/casopis/archiv-casopisu/2021/cislo-9/jak-muzeme-dosahnout-vynikajici-urovne-vyuky.html`

3.3 Aplikace poznatků do výuky informatiky

Dle dostupných zdrojů se jako efektivní postupy při vysvětlování komplexnějších pojmů jeví připodobnění k procesům či objektům z reálného života[16]. Dalším z principů je práce v týmech, kde si studenti mohou navzájem vysvětlit problémy, kterým nerozumí, bez přítomnosti vyučujícího, kterého by se normálně z různých důvodů nezeptali. Poslední koncept, který pomáhá ve výuce, je nechat studenta po dokončení úlohy vysvětlit jeho postup a ověřit si tak, že danému problému rozumí. Po dokončení je také možné mu postup zrevidovat a ukázat, kde udělal chybu, případně které části by mohly být realizovány lépe[6]. Tento postup se využívá i v rámci většiny firem. Lidé na seniorštějších pozicích často revidují práci nováčkům na pozicích junioršťějších, aby předešli chybám a současně poskytli začátečníkům zpětnou vazbu k odvedené práci.

4 Principy periferií LEGO robota

V následující kapitole jsou popsány periferie *LEGO* robota. Pochopení jejich principů bude důležité pro realizaci jednotlivých úloh. Funkce periferií budou popsány ve fyzikálním smyslu pomocí jevů, ze kterých vychází.

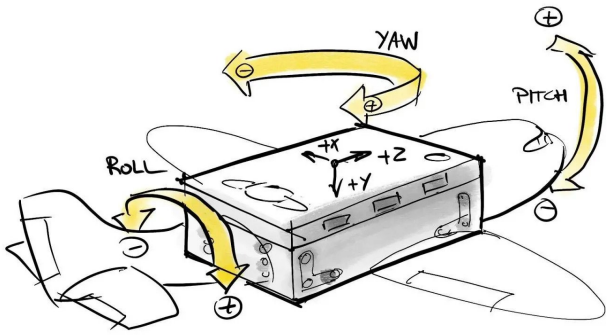
4.1 Gyroskopický senzor

Gyroskopický senzor je integrován v hubu robota a umožňuje mu realizovat úlohy spojené s otáčením a zrychlením. Senzor je založen na principu gyroskopu. Obecně je gyroskop mechanické zařízení, jehož neoddělitelnou součástí je rychle rotující setrvačníkové kolo s těžkým obvodem, smontované tak, že jeho osa rotace se může natáčet kolem pevného bodu ležícího na této ose[11].

Pro přesné výsledky senzoru je nutné ho po spuštění nechat zkalibrovat. Kalibrace se děje automaticky, ale je důležité, aby se během ní senzor vyhnul jakémukoliv pohybu. Během kalibrace se kontroluje teplota hubu a jelikož planeta Země je rotující neinerciální vztažná soustava je třeba odfiltrovat hodnoty spojené s externím zrychlením. Následky špatné kalibrace mohou způsobit nepřesné hodnoty gyroskopu, které se v případě repetetivních operací mohou sčítat a způsobit signifikantní rozdíly. Další hodnota, kterou je nutné vzít v potaz, je maximální úhlová rychlost, kterou dokáže senzor zaznamenat. V případě překročení této hodnoty robot interpretuje jakoukoliv vyšší rychlost jako jeho maximální, což opět může vyústit v nepřesně naměřené hodnoty. Ideální je tedy při úlohách, kde budeme pracovat s gyroskopickým senzorem, snížit rychlost pohybu robota na nižší hodnoty.

Senzor měří hodnoty úhlové rychlosti v jednotlivých osách. Jinak řečeno, měří jak rychle se mění úhel na jedné z jeho os. Z těchto hodnot se následně spočítá tzv *Heading*, což je aktuální směr pohybu vyjádřený ve stupních. V případě modelu *Robot Inventor* došlo k vylepšení senzoru od předchozí verze. Předešlá verze robota *EV3* měla gyroskopický senzor, který dokázal měřit hodnoty pouze v jedné ose. Aktuální model dokáže měřit hodnoty už ve 3 osách. V rámci aplikace jsou tyto hodnoty definovány následovně:

- Yaw angle - otáčení podél svislé osy
- Pitch angle - otáčení podél příčné osy
- Roll angle - otáčení podél podélné osy



Obrázek 4.1: *Vizualizace os gyroskopu*

Zdoj:`https://antonsmindstorms.com/2021/01/14/advanced-undocumented-python-in-spike-prime-and-mindstorms-hubs/`

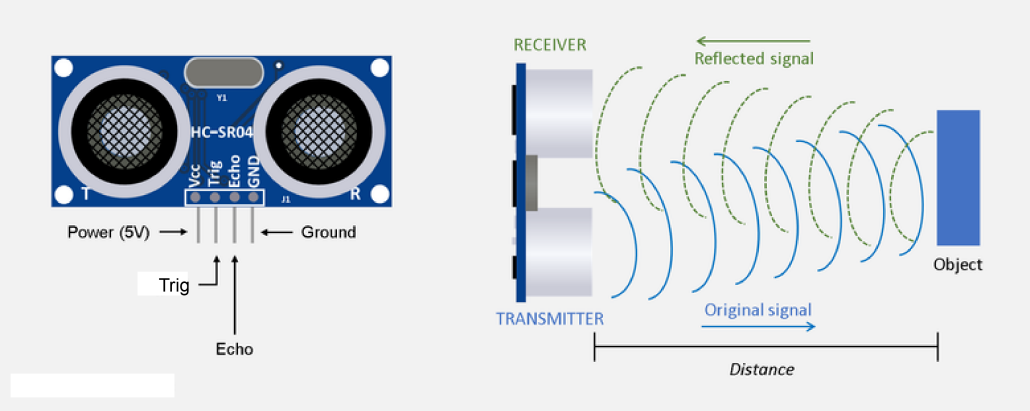
4.2 Senzor vzdálenosti

Senzor vzdálenosti nebo také ultra sonický senzor je zařízení, které je určeno k měření vzdáleností a detekci překážek. V rámci terminologie, kterou prezentuje firma *LEGO*, se používá pojem distanční senzor nebo senzor vzdálenosti, ale jedná se specificky o senzor na principu ultrazvuku, proto je pojem ultra sonický senzor nebo ultrazvukový snímač přesnější.

Snímač emituje vysokofrekvenční zvukové vlny, mimo spektrum slyšitelné lidským uchem. Většina lidské populace dokáže vnímat zvuk o maximální frekvenci 20kHz a za ultrazvuk se považuje vše nad touto hranicí. Po vyslání zvukové vlny snímač čeká na následný odraz této vlny od překážky a převádí ho na elektrický signál. Prodleva mezi odesláním a přijmutím pak signalizuje, jak je objekt vzdálený od senzoru. Výpočet vzdálenosti získáme pomocí následujícího vztahu

$$vzdálenost = \frac{rychlost\ zvuku \cdot doba\ šíření}{2} \tag{4.1}$$

Tento proces je velice podobný echo lokaci, kterou v přírodě využívají například netopýři nebo delfíni. Výhoda tohoto procesu tkví především ve funkčnosti bez jakéhokoliv osvětlení, které vyžadují jiné typy senzorů, založené na odrazu světla. Stejně jako u ostatních senzorů využívající principu odrazu od překážky bude i tento fungovat nejlépe, pokud bude povrch objektu paralelní k vysílači. To způsobí přímý odraz zpět. V opačném případě se může odraz vychýlit a naměřené hodnoty mohou být zkreslené.



Obrázek 4.2: *Ultra sonický senzor*

Zdoj:`https://osoyoo.com/2018/09/18/micro-bit-lesson-using-the-ultrasonic-module/`

4.3 Snímač pro detekci barvy

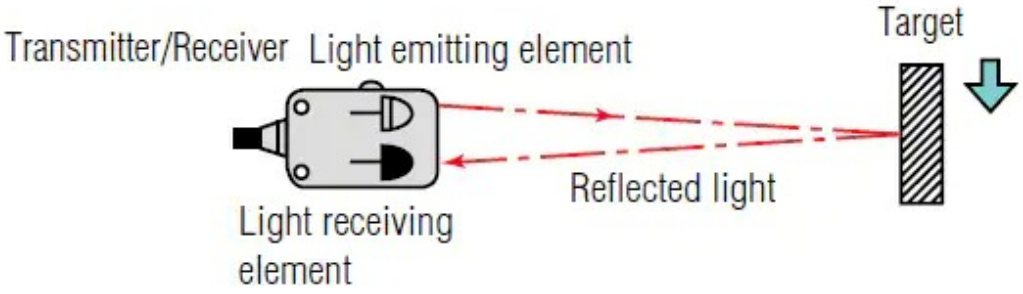
Hlavní funkcí tohoto senzoru je detekce intenzity okolního světla. Oproti senzoru z minulého modelu robota se jeho rozměry zmenšily na polovinu. V rámci světelného senzoru se využívá principu odrazu světla. Podle zákonu odrazu je úhel odrazu vždy roven úhlu dopadu. Snímač dokonce detekuje pouze paprsky vyzářené vlastním emitorem, respektive paprsky o speciální frekvenci, které vyzařuje pouze jeho vlastní dioda. Tato frekvence je však zvolená tak, aby docházelo k co nejmenším interferencím s okolními světelnými paprsky a předcházelo se případným chybám. Senzor dokáže pracovat v pěti režimech:

- Snímání barvy
- Snímání odrazu

- Snímání intenzity červené barvy
- Snímání intenzity modré barvy
- Snímání intenzity zelené barvy

Druhý z módů je zaměřený na snímání odrazu vyzářeného pomocí *LED* diody uvnitř senzoru. Po vyzáření světelného paprsku se snímá jeho odraz, a v rámci senzoru, se vyhodnotí, jaká procentuální část se odrazila zpět. Tato hodnota nám pak může určit, jak tmavá je plocha, od které se paprsek odrazil. Tmavší povrch bude odrážet méně světla a získáme tedy nižší procentuální hodnotu intenzity odraženého světla. Naopak u světlejších povrchů se bude odrážet světla více a hodnota intenzity odraženého světla bude vyšší. Tohoto mechanismu lze využít například při konstrukci robota, který snímá intenzitu odraženého světla pod sebou a snaží se pomocí výběru vyšších z naměřených hodnot pohybovat po trajektorii nakreslené tmavou barvou po světlem povrchu.

Poslední tři z vyjmenovaných režimů snímají pouze surové hodnoty jednotlivých složek, ze kterých se následné skládají barvy pomocí aditivního barevného modelu *RGB*. V rámci těchto režimů můžeme získat číselné hodnoty v intervalu 0 až 255. Tyto hodnoty pak reprezentují zastoupení dané barvy ve sloučené finální hodnotě.



Obrázek 4.3: *Princip funkce snímače pro detekci barvy*

Zdoj:`https://www.logicbus.com.mx/Sensores-Fotoelectricos-logicbus.php`

5 Rozdělení úloh

5.1 Jednoduché úlohy

Nejjednodušší úlohy budou určené pro řešitele s malou či nulovou zkušeností s programováním. Primárním cílem bude demonstrace schopností robotů a cílem sekundárním bude seznámit uživatele s rozhraním robotů. Další úkol bude s pomocí implementačních návodů vytvořit jednoduchý algoritmus, který mohou fyzicky nahrát do zařízení a vidět tak, co jejich program dělá v praxi. Programy budou tvořeny v jednodušších blokových editorech, pro lehčí prvotní pochopení programovacích konceptů a odstranění nutné znalosti syntaxe textových programovacích jazyků.

5.2 Mírně pokročilé úlohy

Realizace mírně pokročilých úloh bude primárně spočívat v přechodu z blokového editoru do textových programovacích jazyků. Cílem bude si osvojit syntaktické konstrukce daného jazyka a pochopit vzájemný vztah mezi blokovým editorem a vysokoúrovňovým textovým jazykem. Dále zde budou představeny řídící datové struktury jako proměnné, cykly, podmínky, pole a další. Výsledná úloha bude demonstrovat kombinaci jednoduchých úkonů, spojených v jeden celistvý program.

5.3 Pokročilé úlohy

Pokročilé úlohy budou určeny pro studenty, kteří již mají zkušenosti s programováním. Nutná prerekvizita pro realizaci těchto úloh bude ovládat nějaký programovací jazyk na základní úrovni. V rámci tutoriálu již nebudou vysvětlovány základní koncepty programování a v případě neznalosti je řešitel nucen si tyto pojmy samostatně dostudovat. Cílem bude se seznámit s rozhraním zařízení a v případě neznalosti jazyka, ve kterém bude úloha implementována i s ním. Úlohy budou vyžadovat pokročilé porozumění daného zařízení a pochopení realizovaného problému po algoritmické stránce. Rozsah úlohy bude zhruba v úrovni práce středoškolské odborné činnosti.

6 Využité programovací jazyky

6.1 Scratch

Vizuální programovací jazyk *Scratch* byl vyvinut na *MIT* s účelem oslovit mladé lidi a pomocí snadné a zábavné hry je naučit základní koncepty programování. Princip jazyka je skládání grafických elementů, které reprezentují základní programovací konstrukce. Skládání grafických elementů má však určité podmínky a pouze některé lze společně zkombinovat, podobně jako je tomu například u stavebnice *LEGO*[18]. Díky fundamentálně jednoduššímu konceptu dokáží studenti rychle implementovat programy bez znalosti syntaxe, jako je tomu třeba u jakéhokoliv textového programovacího jazyka. Využití tohoto prostřední ve výuce informačních technologií taktéž motivuje studenty dále prohlubovat svoje znalosti v oblasti programování[14].

6.2 Python

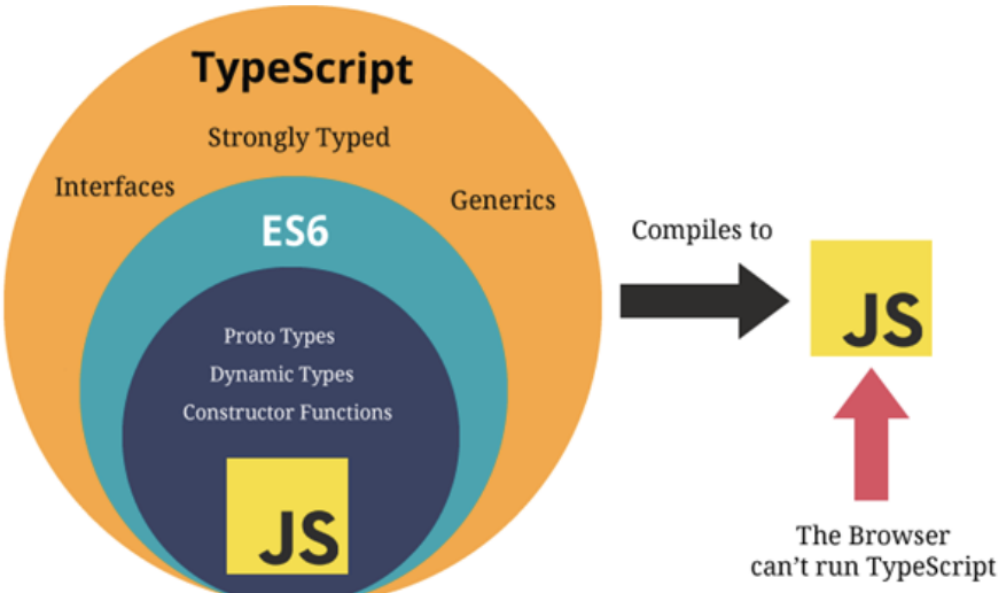
Python je vysokoúrovňový dynamicky typovaný programovací jazyk se širokou škálou možností užití. Lze jej použít na jednoduché skripty i na rozsáhlé aplikace obsahující miliony řádek kódu. Ve srovnání s ostatními programovacími jazyky má *Python* jednu z nejpřívětivějších učících křivek, což způsobuje progresivnější vývoj začínajících programátorů[15].

MicroPython

V rámci bakalářské práce budu pracovat se zařízeními, která pro svůj provoz nebudou potřebovat všechny funkcionality, které jazyk *Python* nabízí. Pro realizaci programů bude efektivnější využít jeho podmnožinu *MicroPython*, určenou přímo pro programování *IoT* zařízení, vestavěných systémů a mnoho dalších zařízení, jejichž řadič tento jazyk podporuje. Oproti původnímu jazyku *Pyton* obsahuje *MicroPython* pouze omezenou podmnožinu původních knihoven, ale má navíc modul pro podporu nízkoúrovňové komunikace se zařízením na hardwarovém rozhraní[4].

6.3 TypeScript

TypeScript je nadstavba nad jazykem *JavaScript*. Hlavní rozšíření je především statické typování, které umožňuje vyšší kontrolu a menší chybovost u velkých projektů. Statické typování umožní odhalit mnoho chyb ještě před spuštěním programu, čímž zvyšuje rychlost vývoje. Další rozšíření, která *TypeScript* přináší, jsou genericita, moduly, rozhraní a anotace datových typů. *TypeScript* je vytvořen nad standardem *EcmaScript* 5, tudíž každý *JavaScriptový* program lze současně přeložit pomocí *TypeScriptového* překladače[5]. Pro realizaci úloh v rámci bakalářské práce budou v tomto jazyce programovány komplexnější úlohy pro *BBC Micro:bit*. Podpora pro překlad *TypeScriptu* do blokového editoru a zpět je zajištěna v rámci vývojové platformy *Microsoft MakeCode*.



Obrázek 6.1: *Struktura jazyka TypeScript*

Zdoj:`https://www.techlearn.live/blog/typescript-vs-javascript-what-is-the-difference/`

7 Struktura metodických materiálů

V rámci rozhodování o typu realizace metodických materiálů jsem zvažoval dvě varianty. První z nich byla textová varianta ve formátu prezentace s popisem řešení doprovázeným obrázky a krátkými video ukázkami. Tato varianta se jevila jako neefektivní z hlediska získání pozornosti studentů, kde většina moderních výukových materiálů je realizována formou videa či jinou interaktivní alternativou. Na druhou stranu je tato varianta vhodná pro rozsáhlejší úlohy, v mém případě pro pokročilé úlohy, v rozsahu střední odborné činnosti, které jsou obsahově náročnější a konzumace interaktivního obsahu by byla zdlouhavá a vedla ke ztrátám pozornosti. V rámci těchto úloh je v interaktivních materiálech také problematičtější najít si konkrétní část implementace, kterou je potřeba vysvětlit či jinak detailně prozkoumat.

Druhá varianta je již zmíněná forma video tutoriálu. Studenti zde nemusí pročítat dlouhé odstavce textu, které mohou hned na první pohled odradit a demotivovat. Stačí pouze pasivně přijímat edukační obsah distribuovaný formou videa, kde se v každém kroku tutoriálu dozvídají, co přesně mají udělat a jakým způsobem daný problém řešit. Stejně jako u textové formy je možnost, v případě uvíznutí na specifickém problému, se vrátit zpátky na dané téma a pustit si problematickou část znovu. Nevýhoda je méně podrobný obsah z důvodů časové úspory v rámci vytvoření videa o optimální délce.

Metodické materiály pro Gymnázium Sokolov jsem se nakonec rozhodl pojmout formou kombinace video tutoriálu s přiloženými zdrojovými kódy a stručným popisem implementace v dokumentu. Video tutoriál zajistí vysvětlení problematiky poutavější formou a umožní formulovat zadání úlohy v kratším čase. Tutoriály budou obsahovat konkrétní zadání úlohy, s demonstrací finálního programu, pro lepší představu řešitelů, krok po kroku popsanou implementaci programu, s hlasovým doprovodem, a možná rozšíření úlohy, která mohou studenti implementovat v rámci samostatné činnosti. V audio záznamu se pokusím objasnit základní programovací koncepty, řídící struktury počítačových programů a další pojmy, které mohou působit složitě pro začínající programátory. Zdrojové kódy pak reprezentují finální řešení, do kterého studenti mohou nahlédnout v případě uvíznutí na pro-

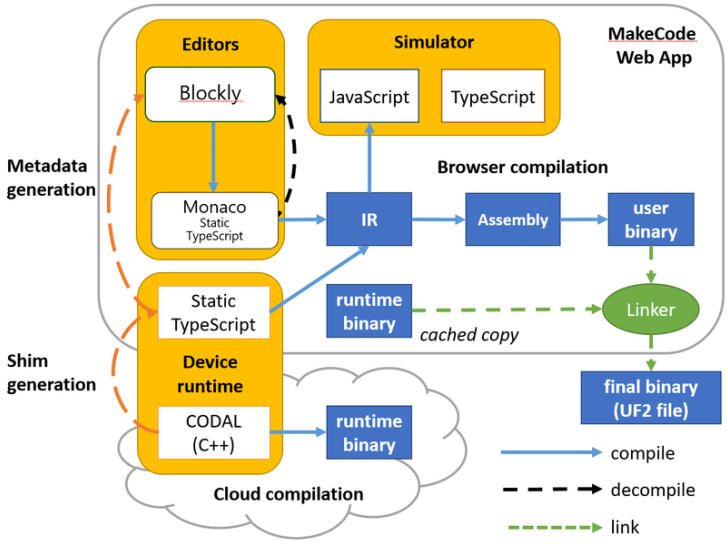
blému bez nutnosti hledání konkrétní pasáže ve videu. V rámci textového dokumentu bude shrnuta implementace do bodů. Dokument bude složit jako statický materiál při realizaci úloh, který budou mít řešitelé k ruce během videa. Kroky z videa zde budou stručně popsané do jedné až dvou stránek formátu A4. Tím se částečně eliminuje nutnost přeskakovat v rámci videa kroky, které řešitel nestihl zaregistrovat.

8 BBC Micro:bit

První zařízení, pro které budou konstruována zadání úloh je *BBC Micro:bit*. Detailní popis tohoto zařízení je sepsán v rámci analýzy v kapitole 2.1.

8.1 Microsoft MakeCode

Microsoft MakeCode je platforma pro vývoj programů pro vestavěná zařízení, určená primárně pro začínající programátory. Platforma je realizována webovou aplikací obsahující vývojové prostředí využívající dvou paradigmat. Prvním z nich je vizuální přístup. Zde se využívá jazyka *Scratch* popsaného v kapitole 6.1. Druhým z nich je klasické textové programování využívajících jazyků *Python* a *TypeScript*. Nespornou výhodou se ukazuje eliminace instalace vývojového prostředí a celková jednoduchost celé platformy, což jsou v profesionálně využívaných vývojových prostředích faktory, které začínající programátory mohou odradit. Vývojové prostředí dokonce dokáže program z blokového editoru přeložit přímo do jazyků *Python* a *TypeScript*. Tento proces funguje i opačně.



Obrázek 8.1: *Architektura platformy MakeCode*

Zdoj:`https://www.researchgate.net/figure/The-MakeCode-and-CODAL-Architecture_fig1_325633895`

8.2 Dostupné API

API pro mikročip *Micro:bit* je popsané jak pro *MicroPython*, tak pro jazyk *TypeScript*. Dokumentace celého *API* je dostupná z oficiálních webových stránek. Oproti dokumentaci robota *LEGO Mindstorms* je přehledně tato dokumentace uspořádána a nenašel jsem žádné rozhraní, které by zde nebylo zdokumentováno.

8.3 Jednoduchá úloha - čítač kroků

V rámci první úlohy bude implementován program, sloužící jako krokoměr. Pro detekci kroků bude použita vestavěná funkce, která reaguje na pohyb čipu. Po spuštění programu se inicializuje proměnná reprezentující počet kroků na nulu a ve chvíli, kdy je zaznamenán pohyb, se tento čítač inkrementuje. Jedním z tlačítek se pak počet kroků zobrazí a druhým se nastaví čítač zpět na hodnotu 0. Zadání úlohy, návod pro implementaci a referenční řešení programu je k dispozici v příloze *Zadání úlohy - krokoměr*.

Rozšíření úlohy

Rozšíření úlohy bude spočívat v naprogramování logiky, která dokáže spočítat celkovou nachozenou vzdálenost. To bude pouze jednoduchý výpočet, který vynásobí délku kroku s počtem kroků a zobrazí výsledek. Délka kroku bude konstanta zadaná přímo v programu a počet kroků se bude dynamicky měnit. Druhé možné rozšíření je upravit snímání kroku tak, aby bylo citlivější k nárazům. Referenční verze využívá funkci *Shake*, která v rámci testování nefungovala zcela přesně. Některé kroky vůbec nezaznamenala a pro delší trasy by krokoměr měl signifikantní rozdíly oproti skutečným hodnotám. *Micro:bit* ale obsahuje akcelerometr, pomocí kterého by tato funkce šla ekvivalentně nahradit. Numerická data, která akcelerometr poskytuje, udávají informace o zrychlení v jednotlivých osách, a dokonce i sílu zrychlení. Právě hodnota velikosti zrychlení se může využít jako práh pro určení, jestli se jedná o krok či nikoliv. Korektní volba tohoto prahu pak může výsledek zpřesnit.

8.4 Mírně pokročilá úloha - Kámen, nůžky papír

Středně pokročilá úloha pro mikročip *Micro:bit* bude jednoduchá hra, založená na náhodě. Hra bude určena pro dva hráče a každý z nich bude využívat mikročip pro výběr jednoho ze tří předmětů. Tlačítky, umístěnými po stranách mikročipu, budou hráči měnit volbu předmětu a stisknutím obou tlačítek svoji volbu potvrdí. Potvrzením se zablokuje možnost změnit volbu a současně se odešle hráčův výběr na protivníkův čip. Po potvrzení mohou nastat dva případy. První z nich je, že oponent svůj výběr potvrdil a odeslal, ještě před potvrzením hráče. V tu chvíli se pouze vyhodnotí výsledek podle následující logiky.

- Kámen poráží nůžky
- Papír poráží kámen
- Nůžky poráží papír

V případě, že hráč odeslal výsledek před oponentem, se displej zablokuje, a čeká se na zaslání výběru protihráče. V rámci referenčního řešení nejsou řešeny časové limity pro získání výsledků. Pro nižší komplexnost úlohy se předpokládá, že výsledek vždy dorazí. V obou případech se po vyhodnocení výsledku na displej zobrazí verdikt a hra se spustí znovu. Zadání úlohy, návod pro implementaci a referenční řešení programu je k dispozici v příloze *Zadání úlohy - kámen, nůžky, papír*.

Cíle úlohy

Úloha má řešitele seznámit se syntaxí jazyka *TypeScript* a rozhraním, které *Micro:bit* v tomto jazyku poskytuje. Z hlediska řídících datových struktur bude vysvětlen koncept konstrukce podmíněného větvení a generování náhodných čísel. Spolu s tím se řešitel seznámí s vizuální interpretací výsledků do *LED* displaye a funkcemi pro přijímaní a odesílání zpráv pomocí rádiového spojení.

Rozšíření úlohy

Řešitel si dále může nabyté znalosti procvičit přidáním zvukových efektů při interakci hráče s tlačítky a po vyhodnocení výsledků hry. Druhým možným rozšířením je do hry implementovat čítač skóre, který vychází z počtu výher,

proher a remíz. Komplexnější řešení toho problému je do hry zavést systém *Elo*, který se využívá například v šachu.

8.5 Pokročilá úloha - Herní konzole

Výsledným produktem pokročilé úlohy bude program, který promění mikropočítač v herní konzoli. V rámci své demonstrační úlohy jsem se rozhodl realizovat hry *Snake* a *Flappy Bird*. Jelikož čip obsahuje pouze dvě tlačítka, je výběr her radikálně limitován. Po spuštění programu se na LED displeji zobrazí číslo reprezentující index hry. Číslo, a s ním i vybranou hru, lze měnit pomocí tlačítek A a B, které hodnotu inkrementují, případně dekrementují. Při stisknutí obou tlačítek současně se hra spustí a mikropočítač se přepne do herního módu. Zadání úlohy, návod pro implementaci a referenční řešení programu je k dispozici v příloze *Zadání úlohy - herní konzole*.

8.5.1 Snake

První z implementovaných her bude *Snake*. Jedná se starou arkádovou hru, kde hráč ovládá hada po herní ploše a snaží se nasbírat co nejvíce náhodně generovaných jablek, která reprezentují body. Po každém sebraném jablku se však had o velikost jednoho pole zvětší a hra se zrychlí, čímž se stává obtížnější. Hada lze ovládat pouze pohyby vlevo a vpravo a cíl je nasbírat co nejvyšší počet jablek bez kolize hada s okrajem herního pole. Kolize hada sama se sebou taktéž způsobí konec hry.

Implementace hry Snake

V rámci platformy *MakeCode* lze definovat hráče a jiné interaktivní objekty pomocí objektu *LedSprite*. Ta ve výchozím stavu rozsvítí *LED* diodu, kterou lze poté posouvat, otáčet nebo kontrolovat, zda koliduje s jiným objektem stejného typu. V rámci hry *Snake* tuto konstrukci využijeme pro definici hada a jablka. Využití bude výhodné pro oba objekty, jelikož u hada musíme zajistit jeho pohyb v každé iteraci cyklu a u jablka jeho přemístění při kolizi s hlavou hada. Had bude realizován jako pole objektů *LedSprite*. Jeho aktuální pohyb se bude ukládat v proměnné, která se bude měnit stisknutím tlačítek. Tím umožníme změnu směru pohybu hada vlevo a vpravo. Vizuálního pohybu po *LED* displeji docílíme rozsvícením další diody ve směru aktuálního pohybu hada a současně zhasnutím diody, která bude uložena jako poslední prvek ve výše zmíněném poli. Při kolizi hlavy hada s jablkem zhasnutí diody vynecháme, čímž hada prodloužíme o jeden bod. Maximální

délka hada bude 10 bloků, což jsou přesně dvě pětiny displeje. Při takto malém displeji by vyšší délka v kombinaci se zvyšující se rychlostí činila hru velice obtížnou. Před každým posunem hada je také nejprve nutné ověřit, zda se jedná o validní pozici. Pokud se nejedná o validní pozici, ukončíme hru a zobrazíme skóre. Validní pozice je taková, která leží v mezích, definovaných rozměrem displeje, a současně na této pozici v daný moment není žádná z částí hada. S narůstajícím skórem pak zvyšujeme koeficient definující rychlost hry.

8.5.2 Flappy Bird

Druhá implementovaná úloha bude upravená verze hry *Flappy Bird*. V této hře je hráč reprezentován jako letící pták snažící se vyhýbat překážkám. Překážky se generují náhodně a s přibývajícím skórem se hra zrychluje, což ji činí obtížnější. V originální verzi hry na ptáka, reprezentujícího hráče, působí gravitace a automaticky ho přitahuje k dolnímu okraji obrazovky. To na malém *LED* displayi není realizovatelné, respektive by přidání této funkčnosti znehodnotilo herní zážitek, protože pohyb směrem dolů na takto malém rozlišení by v kombinací s malou obnovovací frekvencí hry vyústil v neintuitivní chování hráčovy postavy. Ovládání bude tedy realizováno pomocí tlačítek, kde jedním z nich bude hráč moci posunout svou postavu směrem vzhůru a druhým z nich směrem dolu.

Implementace hry Flappy Bird

Stejně jako ve hře *Snake* se i zde bude využívat modulu *game* a objektu *LedSprite*. Pomocí těchto modulů budou konstruovány překážky i samotný hráč. Hráč bude umístěn vždy v levém okraji obrazovky a každou iteraci ve hlavní smyčce hry se k němu budou přibližovat vygenerované překážky. Generování struktury překážek bude náhodné a bude probíhat každé 3 iterace cyklu. Překážka bude pokrývat celý jeden sloupec displeje, kromě jednoho náhodně vybraného bodu, který bude sloužit jako úniková cesta pro hráče. Přiblížení překážek k hráči bude realizováno přičtením záporné hodnoty k aktuální souřadnici překážky na ose x. Konkrétně se bude překážka posouvat každou iteraci cyklu o jeden sloupec vlevo, k souřadnici se tedy bude přičítat hodnota -1. Pokud se hráči podaří překážce vyhnout, zvýší se mu skóre čímž se hra zrychlí.

Rozšíření úlohy

Program lze rozšířit přidáním dalších jednoduchých her do výběru zobrazeném v úvodní obrazovce. Další hry, které lze ovládat pouze dvěma tlačítly jsou například *Pong*, *Snakes* & *Ladders*, *Space Invaders* nebo hra *kámen, nůžky, papír*, která byla implementována v druhé úloze.

9 Robot LEGO Mindstorms

Druhé zařízení, pro které byly realizovány úlohy je robot ze série *LEGO Mindstorms* popsaný v kapitole 2.3.

9.1 Výběr konfigurace

Jelikož je stavebnice koncipovaná jako *"5 v jednom"*, je možné z jednoho setu kostek vytvořit 5 různých robotů. Před sestavením robota je tedy nutné předem určit, která sestava se bude využívat pro realizaci úloh. Pro všechny úlohy jsem se rozhodl využít robota v sestavě *Blast*, což je vlajková loď této stavebnice, která se jeví jako nejatraktivnější varianta poskytující nejvíce možností.

9.2 Podpora pro jazyk Python

Řídící kostka robota podporuje programování pomocí jazyka *Python*, respektive *MicroPython*. Nízkoúrovňová hardwarová zařízení často využívají syntakticky složitější programovací jazyky. Hlavní výhoda *MicroPythonu* je použití syntakticky lehce naučitelného jazyka na programování těchto nízkoúrovňových periferií[17].

9.3 Dostupné API

Oficiální *API* robota od firmy *LEGO* je bohužel nekompletní a neobsahuje tedy informace o rozhraních některých z periferií robota (například senzor vzdálenosti a senzor pro snímání barev). Během implementace bylo tedy nutné čerpat informace i z dalších zdrojů, zejména z neoficiální dokumentace vytvořené fanoušky *LEGO* produktů.

9.3.1 Možnosti hubu

Hub je modul reprezentující ovládací kostku robota. Jeho rozhraní umožňuje například:

- Konfiguraci *Bluetooth* adaptéru
- Ovládání mikrofonu a *LED* diod

- Ovládání šestiosého gyroskopu
- Získání informací o stavu baterie

Dále také umožňuje získat informace o nahraném firmwaru a o aktuálních datech z periferních komponent na jednotlivých portech.

9.3.2 Možnosti periferních zařízení

Periferní zařízení robota jsou oficiální dokumentací zdokumentována velmi stroze. Dostupné je zde pouze *API* pro generickou periferii a aplikační rozhraní pro ovládání motorů. Motory lze spustit ve třech módech.

- Pulzně šířková modulace
- Limitace časem běhu
- Limitace počtem otáček

Změna směru je ve všech módech realizována pomocí opačného znaménka. Pro distanční senzor a senzor barev jsem čerpal z dokumentace dostupné na *GitHubu* vytvořené fanouškovskou základnou.

9.4 Jednoduchá úloha - pohyb robota s trajektorií čtverce

Zadání a cíle úlohy

Cílem nejjednodušší úlohy pro *LEGO* robota bude vytvořit program umožňující uvedení robota do pohybu po předem stanovené trajektorii. Pro referenční úlohu jsem zvolil trajektorii čtverce. Úloha bude implementována v programovacím jazyce *Scratch* v aplikaci *LEGO MINDSTORMS* a v rámci video tutoriálu bude vysvětleno i zakládání projektu a nahrání programu do hubu robota přes bezdrátové spojení. Po shlédnutí tutoriálu by uživatel zpracovávající danou úlohu měl být schopen samostatně založit svůj vlastní projekt a nahrát implementovaný program do robota. Po pochopení základních operací s motory robota a jejich vztahů s vizuálními bloky by také měl být schopen nakonfigurovat pohyb robota dle libosti. Zadání úlohy, návod pro implementaci a referenční řešení programu je k dispozici v příloze *Zadání úlohy - pohyb ve čtvercové trajektorii.*

Implementace

Po založení projektu a představení vývojového prostředí se za audio doprovodu popisujícího detailní popis provedených úkonů implementuje první verze programu. V rámci dalších úprav programu pak objasním koncept cyklu. Jelikož čtvercový pohyb se skládá ze dvou operací, pohybu vpřed a otočky o 90 stupňů, budu demonstrovat užití cyklu na této problematice. Čtyři iterace těchto dvou pohybů pak dohromady vytvoří požadovanou trajektorii. Dále také vysvětlím dva způsoby, jak zajistit otočku robota. Jednodušší z nich je odhad potřebné vzdálenosti k otočení a následné zpřesňování konstanty. Komplexnější pohled na tento problém je s využitím integrovaného gyroskopu, který obsahuje informace o polohových vlastnostech robota. Po překročení hranice 90 stupňů na směrové ose máme tedy taktéž přibližně zaručeno, že se robot otočil o požadovanou hodnotu bez nutnosti odhadování konstanty. Tento způsob je ovšem taktéž zatížen chybou způsobenou nepřesnostmi, popsanými v kapitole 4.1, zaměřené na popis principu gyroskopu. Při více iteracích se mohou chyby načítat a vytvořit signifikantní odchylku od přesného řešení. Pro 4 iterace je však řešení přesné dostatečně. Princip algoritmu je popsán pseudokódem níže.

Algoritmus 1: Algoritmus pro základní verzi programu

```
distance = 10;
for i = 0 to 4 by 1 do
    move_forward(distance);
    turn_by_90_deg();
end
```

Rozšíření úlohy

Jedním z možných rozšíření úlohy je přidat další trajektorie, po kterých by se robot pohyboval. Mezi jednodušší z nich by patřil například rovnostranný trojúhelník nebo obdélník, mezi složitější kružnice. Dalším z možných rozšíření ověřující pochopení práce s motory je implementace inkrementálního přírůstku rychlosti pohybu robota v závislosti na uražené vzdálenosti a ukončení jeho pohybu po dosažení rychlosti maximální.

9.5 Mírně pokročilá úloha - Režim hlídání

Mírně pokročilá úloha bude realizována v jazyce *Python* s využitím modulu *mindstorms*. Úloha by měla řešitele seznámit s jazykovými konstrukcemi tohoto jazyka a objasnit mu aplikační rozhraní robota, zejména pro motory a distanční senzor. Cílem úlohy bude naprogramovat robota tak, aby hlídal úsek ve tvaru kruhové výseče a v případě zaznamenání objektu reprezentujícího nepřítele po něm vystřelil z jeho zbraně. Zadání úlohy, návod pro implementaci a referenční řešení programu je k dispozici v příloze *Referenční řešení*.

Popis implementace

K získaní informace o blízkých objektech bude využit distanční senzor umístěný na robotovi paži. K následné střelbě pak bude využita zbraň na paži druhé. Aby střelba i detekce vzdálenosti dávaly smysl, je nejprve nutné, aby robot zvedl jednu končetinu do pozice rovnoběžné se zemí. Obě končetiny robota, společně s hlavou, jsou řízeny pomocí jednoho diferenciálu, tudíž robot zvládne předpažit vždy jen jednu končetinu pomocí motoru umístěného na jeho zádech. V rámci přechodu z režimu detekce do režimu střelby bude tedy nejprve nutné napozicovat periferie na končetinách tak, aby mohly správně realizovat své funkce.

Snímaní oblasti bude realizováno ve smyčce, která detekuje aktuální vychýlení robota a hodnotu distančního senzoru. Po překročení limitní hodnoty vychýlení, která bude nastavena na 30 stupňů, se změní směr otáčení robota. Výsledný úhel snímaný robotem bude tedy zhruba 60 stupňů. Hodnotu vychýlení robota lze získat z gyroskopu integrovaného v *hubu*.

Po zaznamenání objektu v blízkosti se pohybová smyčka přeruší a motor, ovládající pohyb končetin, se uvede do pohybu opačným směrem. To vyústí ve zvednutí končetiny se zbraní do pozice pro střelbu. Zbraň je ovládána čtvrtým motorem, který je stejně jako ostatní ovládaný *hubem* robota. Její princip spočívá v mechanickém stisknutí tlačítka, které uvolní stlačenou pružinu, což uvede projektil umístěný v přední části zbraně do pohybu. Pro dva výstřely je nutné uvést motor do pohybu nejprve jedním (stisknutí prvního tlačítka střelného mechanismu umístěného vlevo) a poté druhým směrem (stisknutí druhého tlačítka umístěného vpravo). Zdrojový kód programu je k dispozici v příloze *Referenční řešení*.

Algoritmus 2: Algoritmus pro hlídací režim

```
while found == false do
    distance = get_distance();
    if distance < 30 then
        found = true;
        stop_movement();
        shoot();
    angle = get_angle() ;
    if scan_right == false then
        turn_left();
    else
        turn_right();
    if |angle| > 30 then
        scan_right = !scan_right;
end
```

Rozšíření úlohy

Rozšíření úlohy bude spočívat v hlubším seznámením se s aplikačním rozhraním robota a doplněním zvukových a vizuálních efektů k jednotlivým

úkonům. Tyto efekty lze doplnit pomocí integrovaného mikrofonu a konfigurovatelného *LED* displeje. Jejich *API* je popsáno v oficiální *LEGO* dokumentaci. Úkony, kterým tyto efekty budou přidány jsou následující:

- Začátek programu
- Zaznamenání nepřítele
- Střelba
- Ukončení programu
- Změna směru skenování

9.6 Pokročilá úloha - Bludiště

Pokročilá úloha bude zaměřena na problematiku úniku z bludiště. Algoritmy pro únik z bludiště lze klasifikovat podle dvou kritérií. První klasifikace je podle struktury, kde rozlišujeme bludiště se známou a neznámou počáteční strukturou. Při známé počáteční struktuře máme k dispozici schéma nebo jiný popis překážek v bludišti, které můžeme před průchodem analyzovat a využít k nalezení řešení. Druhá varianta je pak neznámá počáteční struktura, kde začínáme s nulovou informací o schématu bludiště a jeho strukturu vytváříme až za běhu programu, podle získaných informací ze vstupů. Druhá klasifikace bludišť je podle cyklů. Bludiště, které neobsahuje cykly, lze považovat za ekvivalent k pojmu strom z teorie grafů a mezi libovolnými dvěma body vede vždy jen jedna cesta[13]. Opakem jsou pak bludiště, kde je nutné vzít v potaz existenci cyklů. Zde mohou cykly znemožnit konvergenci algoritmu ke konečnému řešení.

Předpoklady pro bludiště, které bude řešeno v rámci této úlohy, jsou počáteční neznalost struktury bludiště a absence cyklů. Rozšíření úlohy pak bude spočívat v úpravě programu tak, aby dokázal řešit i bludiště, kde se smyčky vyskytují. K řešení úloh spojených s nalezením cesty k východu lze použít širokou škálu algoritmů. Před implementací úlohy je nejprve nutné rozhodnout o tom, který algoritmus pro řešení úniku z bludiště využijeme. Zadání úlohy, návod pro implementaci a referenční řešení programu je k dispozici v příloze *Zadání úlohy - únik z bludiště*.

9.6.1 Algoritmus random mouse

Nejjednodušší z algoritmů založený na principu náhody je algoritmus Random mouse. Princip je triviální, ve chvíli, kdy se robot dostane do situace, kde má na výběr 2 a více možností, náhodně vybere jednu z nich a tou se vydá. Tento algoritmus je velice neefektivní z časového hlediska a současně nemusí nutně nalézt optimální řešení, a dokonce řešení nemusí nalézt vůbec. Naopak je velice efektivní po paměťové stránce, jelikož neukládá žádné informace o předchozích pohybech v bludišti. Bez žádných dalších úprav lze tedy tento algoritmus považovat za bezstavový. Jedna z možností vylepšení tohoto algoritmu je ořezávání hran, které vedou do slepých uliček a ukládání těchto cest pro eliminaci nevalidních tras a rychlejší konvergenci ke správnému řešení.

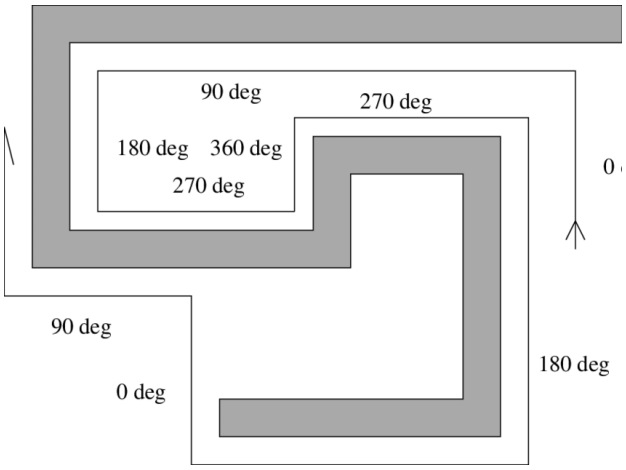
9.6.2 Algoritmus na principu sledování zdí

Další možný přístup pro řešení východu z bludiště je sledování zdi. Zjednodušený princip toho algoritmu lze popsat analogií z reálného života, kdy při vstupu do bludiště vybereme jednu ruku a po celou dobu průchodu se budeme snažit udržovat tuto roku podél zdi. Pro popis algoritmu definujeme jednu stranu jako primární. V případě, že na primární straně bude zeď a nelze tedy touto cestou pokračovat, se pokračuje směrem rovně (pokračuje se podél této zdi) a až v případě, že nelze pokračovat ani primární stranou, ani rovně, se vydáme stranou opačnou k primární (levostranná nebo pravostranná zatáčka). Pokud se dostaneme do slepé uličky (nemůžeme pokračovat rovně, vlevo ani vpravo) provedeme otočku o 180 stupňů a pokračujme dále. Tím se nám prohodí zeď, kterou následujeme a máme možnost se dostat do jiné části bludiště. Základní verze tohoto algoritmu si však nedokáže poradit se smyčkami v bludišti. Naopak v případě perfektních bludišť (mezi libovolnými dvěma body vede vždy jen jedna cesta) nalezneme řešení vždy[13].

9.6.3 Algoritmus Pledge

Princip toho algoritmu spočívá v obcházení překážek. Před řešením je nutné vybrat směr pohybu, kterým se robot bude vydávat, v případě absence překážky. Tento směr bude dále označován jako preferovaný a ve většině případů je reprezentován pohybem rovně. V případě výskytu překážky se pak robot otáčí vlevo nebo vpravo (závisí na konkrétní implementaci) dokud nejsou splněny konkrétní podmínky popsané níže. Současně s otáčením si ukládá informace o kolik stupňů a jakým směrem se otočil a informaci o aktuálním

směru. V případě, že aktuální směr i součet úhlů, o které se robot otočil jsou rovny nule, se robot vyhnul překážce a může pokračovat dále. Pokud by robot nebral v potaz součet úhlů, reprezentující celkový úhel otočení, měl by problém s bludištěm obsahující smyčky[9]. Princip algoritmu je znázorněn na následujícím obrázku.



Obrázek 9.1: Algoritmus Pledge

Zdoj:`https://www.researchgate.net/figure/Execution-of-Pledge-algorithm_fig3_2377958`

Implementace algoritmu

V rámci implementace je nutné inicializovat dvě proměnné na hodnotu 0. První z nich je čítač reprezentující celkový počet otoček robota. Pro pravoúhlé bludiště, ve kterém se robot otáčí pouze o 90 stupňů pak může být otočka vlevo reprezentována inkrementací čítače a otočka vpravo dekrementací čítače. Druhá proměnná pak bude sloužit k uchování informace o aktuálním směru robota. Bude tedy nabývat čtyř různých hodnot, kde každá reprezentuje jeden směr. První z čítačů je nutná prerekvizita pro řešení bludišť obsahujících smyčky, jelikož otočení robota o 360 stupňů způsobí nastavení druhé proměnné na hodnotu 0 (robot je otočený požadovaným směrem), ale nezaručuje, že se robot kompletně vyhnul překážce před ním. Celý tento proces pak vložíme do smyčky, kde zvolíme vhodný způsob jak zareagujeme na nalezení cíle bludiště. To může být například delší úsek, kde robot nenarazí na překážku nebo reprezentace konce pomocí specifické barvy, která se vyhodnotí pomocí barevného senzoru.

9.6.4 Závěr analýzy algoritmů

V rámci analýzy dostupných algoritmů jsem bral v potaz pouze varianty zaměřené na předem nedefinovanou strukturu bludiště. Pro triviální typy bludišť by bylo možné použít všechny tři vybrané algoritmy, ale v rámci komplexnosti úlohy jsme se rozhodl pro demonstrační úlohu využít algoritmus na principu sledování zdí. Pro implementaci rozšíření, které dokáže řešit i úlohy s cykly lze tento algoritmus rozšířit o logiku, která umožňuje uchovávání navštívených sekcí, které při opětovném průchodu vyhodnotí jako překážku a zamezí tak zacyklení. Druhá možnost je implementovat algoritmus Pledge, který zacyklení eliminuje bez nutnosti rozšíření.

9.6.5 Implementace

Začátek programu je stejný jako u úlohy pro hlídací režim. Je třeba zkalibrovat končetiny robota a zvednout distanční senzor do pozice, kde bude moci snímat objekty před robotem. Dále je nutné zvolit primární stranu, kterou robot bude volit. V případě této implementace to byla strana levá. Poté se zahájí hlavní smyčka programu. V každé iteraci programu robot jako první vyhodnotí, kterým směrem se může vydat. Tímto směrem se otočí a zároveň zkontroluje, jestli není v cíli. Kontrola cíle je realizována pomocí barevného senzoru, umístěného u kol robota. Konkrétně je cíl reprezentován bílou barvou pod robotem. Pokud je robot v cíli, za doprovodu zvukového efektu a rozsvícení zelené barvy na tlačítku se program ukončí. Pokud není, pokračuje směrem rovně do té doby, než se před ním objeví překážka nebo dokud neujede vzdálenost jedné buňky bludiště. Podmínka pro ujetí jedné buňky bludiště umožňuje robotovi v každé části bludiště zkontrolovat, zda-li se nemůže vydat vlevo. Tím se redukuje možnost zacyklení. Tento proces se opakuje, dokud se robot nedostane do cíle.

9.6.6 Testování programu

Program jsem testoval na referenčních bludištích s velikostí jedné buňky 30x30 cm. Obě bludiště měly čtvercovou strukturu. První z nich bylo o rozměrech 3x3 buňky a druhé o rozměrech 4x4 buňky. Stěny bludiště jsem vytvořil z kartonu a cíl bludiště jsem reprezentoval bílým papírem. Pro obě bludiště dokázal robot nalézt cestu ven, ale u většího z bludišť byl již patrný problém spojený s otáčením robota. Během otáčení robota se nakumulovala odchylka, která by při větším rozměru bludiště způsobovala komplikace a činila tak algoritmus nefunkční. Videa z testování obou bludišť jsou dostupná v elektronické příloze.

10 Testování a zhodnocení úloh

Metodické materiály byly otestovány v rámci výuky na Gymnáziu Sokolov, mnou a jednoduché úlohy jsem také nechal otestovat dva participanty. Tito dva testeři měli nulovou zkušenost s programováním a pouze základní počítačovou gramotnost. Oba dokázali úkoly v blokovém editoru dokončit během 30 minut čistého času. Jednoduché úlohy nedělaly problém ani studentům tercie a kvarty na Gymnáziu Sokolov. Vyučující dokonce zmiňovali, že by je po seznámení s vývojovým prostředím dokázali dokončit i studenti primy.

Pokročilejší úlohy činily těmto studentům problémy. Prerekvizita pro řešení úloh je základní znalost programovacího jazyka *Python* nebo *JavaScript*. Studenti ve zmíněných ročnících však s těmito jazyky ještě nemají zkušenost a realizace úloh pro ně byla velmi obtížná. Další nedostatky, které byly zmíněny v rámci zpětné vazby, byly příliš vysoká rychlost videa a nedostatečně detailní popis implementace referenčního řešení. Plný text zpětné vazby je dostupný v příloze *Plný text zpětné vazby z Gymnázia Sokolov*

Tyto nedostatky jsou dle mého názoru způsobeny kombinací příliš vysoké obtížností pokročilých a mírně pokročilých úloh pro studenty daných ročníků a současně příliš vysokou rychlostí videa. Rychlost videa je pro pokročilejší úlohy rozhodně příliš vysoká v případě, že chceme současně sledovat video i programovat. Většina programovacích pasáží je zrychlená a simultánní programovaní a sledování videa je tak téměř nemožné bez zastavování. Pro tyto účely by bylo lepší vytvořil detailnější video tutoriál, který by vysvětlovat problematiku více do hloubky. Slovní projev by mohl být pomalejší a video by ideálně mohlo mít pasáže, kde by si studenti mohli porovnat jejich aktuální implementaci s referenčním řešením. Další nedostatek, který byl zmíněn v rámci zpětné vazby je špatná kvalita zvuku v některých pasážích videa. Tento problém by šel částečně vyřešit doplněním titulků pro tyto úseky nebo vytvořením nového zvukového záznamu.

11 Závěr

Cíl této práce bylo analyzovat dostupné hardwarové vybavení učebny techniky na Gymnáziu Sokolov a pro dva nejvhodnější kandidáty vytvořit sadu úloh demonstrující jejich schopnosti. V rámci teoretického části byl proveden podrobný rozbor čtyř zařízení: robot *LEGO Mindstorms - Robotí vynálezce*, mikropočítač *BBC Micro:bit*, robot *Edison* a *Ozobot*. Na základě analýzy byla pak z dostupných kandidátů vybrána dvojce robot *LEGO Mindstorms* a mikropočítač *BBC Micro:bit*. Dále se v teoretické části rozebírají způsoby a principy pro korektní tvorbu metodických materiálu, jejichž sumarizace je následně využita v části praktické.

V praktické části byla pro každé zařízení vytvořena zadání tří úloh o různých obtížnostech. Pro všechna zadání se podařilo zdárně implementovat referenční řešení společně s video tutoriálem, obsahujícím detailní postup jak každou z úloh naprogramovat. Pro všechny úlohy bylo taktéž navrženo možné rozšíření, kterým lze na každou z úloh navázat a rozšířit tak její funkčnost.

Jedno z možných rozšíření této práce je vytvoření tutoriálů pro doplňující úlohy zmíněné v závěru každého videa. Dále by bylo vhodné stávající materiály upravit dle požadavků Gymnázia Sokolov a odstranit nedostatky zmíněné v kapitole Testování a zhodnocení úloh.

Seznam zkratek

- IoT - Internet of Things
- BBC - British Broadcasting Corporation
- ARM - Advanced RISC Machines
- USB - Universal Serial Bus
- LED - Light-Emitting Diode
- MIT - Massachusetts Institute of Technology
- API - Application Programming Interface

Seznam obrázků

Literatura

[1] Anderson, L. W. – Sosniak, L. A. *Bloom's taxonomy*. Univ. Chicago Press Chicago, IL, USA, 1994.

[2] Austin, J. et al. The BBC micro: bit: from the UK to the world. *Communications of the ACM*. 2020, 63, 3, s. 62–69.

[3] Baum, D. *Extreme MINDSTORMS: an advanced guide to LEGO MINDSTORMS*. Apress, 2000.

[4] Bell, C. *MicroPython for the Internet of Things*. Springer, 2017.

[5] Bierman, G. – Abadi, M. – Torgersen, M. Understanding typescript. In *European Conference on Object-Oriented Programming*, s. 257–281. Springer, 2014.

[6] Denning, P. J. A debate on teaching computing science. *Communications of the ACM*. 1989, 32, 12, s. 1397–1414.

[7] Farrow, R. Creating teaching materials. *BMJ*. 2003, 326, 7395, s. 921–923.

[8] Forehand, M. Bloom's taxonomy. *Emerging perspectives on learning, teaching, and technology*. 2010, 41, 4, s. 47–56.

[9] Kamphans, T. – Langetepe, E. The pledge algorithm reconsidered under errors in sensors and motion. In *International Workshop on Approximation and Online Algorithms*, s. 165–178. Springer, 2003.

[10] Keller, J. M. How to integrate learner motivation planning into lesson planning: The ARCS model approach. *VII Semanario, Santiago, Cuba*. 2000, s. 1–13.

[11] Lembard, T. Tříosý senzor úhlové rychlosti pro letecké použití. Master's thesis, Vysoké učení technické v Brně, 2008.

[12] Li, K. – Keller, J. M. Use of the ARCS model in education: A literature review. *Computers & Education*. 2018, 122, s. 54–62.

[13] Matějka, P. Algoritmy pro generování a řešení bludišť. Master's thesis, Masarykova Univerzita, 2011.

[14] Ouahbi, I. et al. Learning basic programming concepts by creating games with scratch programming environment. *Procedia-Social and Behavioral Sciences*. 2015, 191, s. 1479–1482.

[15] Python, R. Python. *Python Releases for Windows*. 2019, 24.

[16] Sentance, S. – Csizmadia, A. Teachers' perspectives on successful strategies for teaching Computing in school. In *IFIP TC3 Working Conference 2015: A New Culture of Learning: Computing and Next Generations*, 2015.

[17] Tollervey, N. H. Programming with MicroPython: embedded programming with microcontrollers and Python. 2017.

[18] Wilson, A. – Moffat, D. C. Evaluating Scratch to Introduce Younger Schoolchildren to Programming. In *PPIG*, 1, s. 1–12, 2010.

Přílohy

Popis adresářové struktury

- Přílohy jsou z důvodu nadstandardní velikosti rozděleny do dvou archivů:
  - Adresáře Aplikace_a_knihovny a Text_prace jsou pro oba archivy stejné
  - Archiv *priloha_1* obsahuje v archivu *Vysledky* video tutoriály a textové návody zaměřené na *LEGO* robota
  - Archiv *priloha_2* obsahuje v archivu *Vysledky* video tutoriály a textové návody zaměřené na mikročip *BBC Micro:bit*
- **Text_prace** - obsahuje zdrojové soubory, obrázky a PDF soubor s plným textem práce.
- **Aplikace_a_knihovny** - obsahuje exportované zdrojové kódy programů z dedikovaných vývojových prostředí
  - Adresář *Programy_LEGO_robot* obsahuje importovatelné .lms soubory pro aplikaci *LEGO MINDSTORMS*
  - Adresář *Programy_MicroBit* obsahuje importovatelné .hex soubory pro vývojové prostředí *Microsoft MakeCode*
- Adresář **Vysledky** obsahuje textové návody úloh a video tutoriály a jejich zdrojové soubory. Liší se pro archivy *priloha_1* a *priloha_2*.
- Adresář je dále dělen na 3 podadresáře pro každou úlohu. Podadresář obsahuje:
  - Video tutoriál ve formátu mp4
  - Importovatelný Camtasia projekt
  - Textový návod ve formátu PDF
  - Zdrojové soubory a obrázky pro textové návody

Jednoduchá úloha pro LEGO robota

Zadání úlohy - pohyb ve čtvercové trajektorii

Výstup této úlohy bude program, který rozpohybuje robota v trajektorii ve tvaru čtverce.

Stručný návod pro implementaci

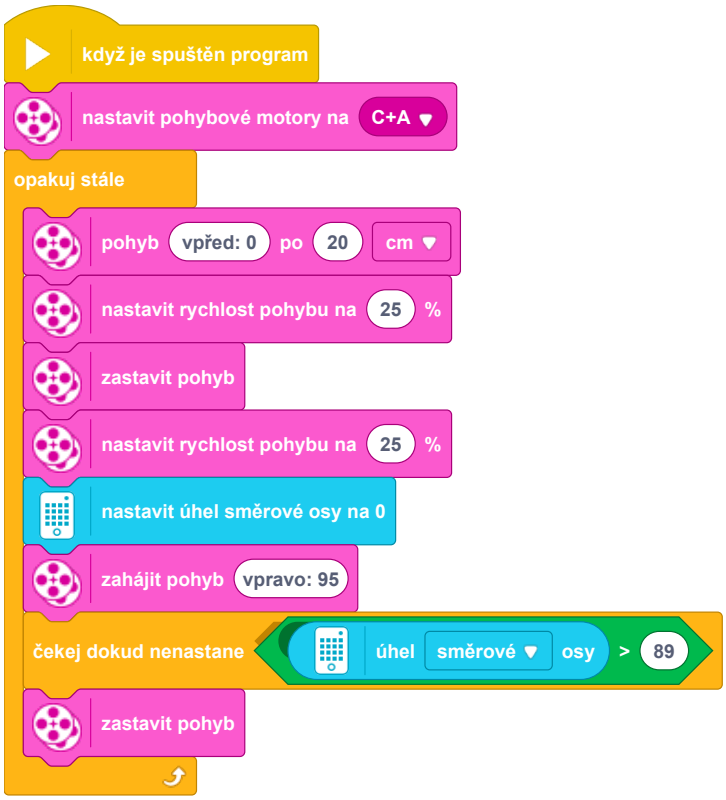
1. V aplikaci *LEGO MINDSTORMS* založte nový projekt.
2. Ze sekce *Pohybové bloky* vyberte blok "`nastavit pohybové motory na`"a vložte do hlavní smyčky programu (blok s názvem "`když je spuštěn program`")
3. Motory nastavte dle Vašeho zapojení. V rámci řídicí kostky je u každého portu napsaný jeho název.
4. Pod tento blok ze sekce *Pohybové bloky* vložte blok "`nastavit rychlost pohybu na`"a nastavte rychlost na 25 %.
5. Ze sekce *Ovládání* vyberte blok "`opakuj 10 krát`"a místo čísla 10 napište číslo 4. Do tohoto bloku budeme vkládat zbytek programu. Blok opět vložte do hlavní smyčky.
6. Do bloku "`opakuj 4 krát`" vložte ze sekce *Pohybové bloky* blok "`pohyb vpřed:0 po 10 cm`". První parametr ponechte a druhý upravte dle prostoru ve Vaší místnosti.
7. Pod něj vložte ze sekce *Senzory* blok "`nastavit úhel směrové osy na 0`".
8. Dále vložte opět blok "`pohyb vpřed:0`" a změňte hodnotu "`vpřed:0`" na "`vpravo:100`".
9. Ze sekce *Události* vložte blok "`čekej dokud nenastane`". Do tohoto bloku vložte podmínku pro zastavení otáčení.
10. Podmínku je v sekci *Operátory*. Bude to zelený blok se symbolem `>`.
11. V levé části podmínky bude hodnota směrové osy gyroskopu, vložená ze sekce *Senzory*. V pravé části pak bude číselná hodnota "`89`".

12. Jako poslední blok vložte ze sekce *Pohybové bloky*, blok `"zastavit pohyb"`.

13. Nahrajte program do robota a otestujte Vaše řešení.

14. V případě chyby porovnejte s referenčním řešením na druhé straně listu a opravte.

Referenční řešení úlohy



Obrázek 11.1: Referenční řešení jednoduché úlohy pro LEGO robota

Mírně pokročilá úloha pro LEGO robota

Zadání úlohy - hlídací režim

Cílem této úlohy bude naprogramovat robota tak, aby snímal území ve tvaru kulové výseče, a v případě detekce nepřítele, po něm vystřelil. Pro detekci nepřítele použijte distanční senzor. Pro střelbu použijte zbraň ovládanou motorem. V rámci rozšíření dodělejte zvukové a vizuální efekty.

Stručný návod pro implementaci

1. V aplikaci *LEGO MINDSTORMS* založte nový projekt.
2. Vytvořte proměnné spojené s periferiemi robota a pomocnou proměnnou pro změnu směru skenování..
3. Vytvořte funkci s hlavní smyčkou programu. První proveďte kalibraci končetin a nastavte distanční senzor do správné polohy pro snímání okolí.
4. S využitím gyroskopu začněte otáčet robota jedním směrem. Po překročení určité hranice (např. 30 stupňů) změňte směr otáčení. Změnu směru zařídíte například pomocí funkce `motors.start()` s opačným znaménkem prvního parametru.
5. Nezapomeňte před změnou směru resetovat hodnotu směrové osy pomocí funkce `hub.motion_sensor.reset_yaw_angle()`
6. Do hlavní smyčky programu vložte detekci nepřítele před robotem pomocí funkce `dist_sensor.get_distance_cm()`
7. V případě, že je vzdálenost menší než Vámi vybraný práh (např. 50 cm), zastavte otáčení a zahajte střelbu.
8. V rámci střelby první zvedněte končetinu se zbraní. Potom pomocí motoru ovládající zbraň proveďte dva výstřely.
9. Po střelbě vraťte obě končetiny do výchozí polohy.
10. Nahrajte program do robota a otestujte Vaše řešení.
11. V případě chyby porovnejte s referenčním zdrojovým kódem a opravte.

Referenční řešení

```
"""
Sensors and motors initialization
"""
hub = MSHub()
body_motor = Motor("D")
gun_motor = Motor("B")
movement_motors = MotorPair("C", "A")
dist_sensor = DistanceSensor("F")

"""
Calibrates robot arms to correct starting positions
"""
def calibrate():
    body_motor.run_for_seconds(3,5)
    body_motor.run_for_rotations(2.3, -50)


"""
Shoots two projectiles with gun motor
and returns to default position
"""
def shoot():
    body_motor.run_for_rotations(1,-50)
    gun_motor.run_for_degrees(70,75)
    gun_motor.run_for_degrees(-140,75)
    gun_motor.run_for_degrees(70,75)
    body_motor.run_for_rotations(1,50)

"""
Main loop of the program
- Robot scans the environment until he identifies
the target in front of him
- Then raises his other arm and shoots
"""
def start_program():
    target_found = False
    scan_right = False
    body_motor.run_for_rotations(2,50)
```

```
    hub.motion_sensor.reset_yaw_angle()
    while target_found == False:
        distance = dist_sensor.get_distance_cm()
        if distance == None:
            distance = 0

        if scan_right == True:
            movement_motors.start(100,10)
        else:
            movement_motors.start(-100,10)

        if abs(hub.motion_sensor.get_yaw_angle()) > 30:
            scan_right = not scan_right
            hub.motion_sensor.reset_yaw_angle()

        if distance > 1 and distance < 50:
            movement_motors.stop()
            body_motor.run_for_rotations(2,-50)
            shoot()
            target_found = True

# Main input of the program
start_program()
```

Pokročilá úloha pro LEGO robota

Zadání úlohy - únik z bludiště

Naprogramujte algoritmus `Wall follower` a nahrajte ho do robota. Ke snímání překážek využijte distanční senzor. Cíl reprezentujte barvou odlišnou od podlahy bludiště. K detekci cíle použijte snímač barev. Vytvořte testovací bludiště a otestujte zda-li algoritmus funguje. V rámci rozšíření úlohy opravte nedostatky algoritmu spojené s cykly v bludišti nebo naprogramujte jiný algoritmus, který tyto nedostatky řeší.

Stručný návod pro implementaci

1. V aplikaci *LEGO MINDSTORMS* založte nový projekt.
2. Vytvořte proměnné spojené s periferiemi robota a konstantu pro zastavení robota před překážkou zvolenou podle velikosti bludiště.
3. Vytvořte funkci s hlavní smyčkou programu. Zkalibrujte končetiny robota a nastavte distanční senzor do správné polohy.
4. Vytvořte dvě funkce. Jedna otočí robota o 90 stupňů vpravo a druhá o 90 stupňů vlevo.
5. Implementujte otáčení pro algoritmus *Wall Follower*:
   (a) Robot se otočí vlevo. Pokud zde není překážka zůstane otočený. Pokud je, otočí se zpět vpravo.
   (b) Pokud není překážka před robotem, neděje se nic, pokud je, otočí se vpravo.
   (c) Robot je nyní otočený vpravo, pokud před ním není překážka zůstane otočený. Pokud je, otočí se znovu vpravo.
   (d) Nyní je robot otočený o 180 stupňů a algoritmus končí.
6. Do hlavní smyčky programu vložte volání tohoto algoritmu, za kterým bude následovat uvedení robota do pohybu.
7. Přidejte do smyčky kontrolu cíle pomocí snímače barev. Pokud je robot v cíli, ukončete program a signalizujte pomocí vizuálních a zvukových efektů nalezení správného řešení.

8. Přidejte do programu logiku, která zastaví robota po uražení vzdálenosti jedné buňky bludiště. Můžete využít časový čítač nebo funkci motorů, která robota zastaví po zdolání této vzdálenosti. Tato logika umožní robotovi vždy vybrat dostupnou cestu vlevo i pouze s jedním senzorem umístěným vepředu.

9. Vytvořte bludiště a otestujte algoritmus. V případě problému porovnejte řešení s referenčním zdrojovým kódem.

Referenční řešení

```
"""
Sensors and motors initialization
"""
hub = MSHub()
body_motor = Motor("D")
movement_motors = MotorPair("C", "A")
dist_sensor = DistanceSensor("F")
color_sensor = ColorSensor("E")

movement_speed = 30
obstacle_distance_limit = 10
timer = Timer()

"""
Wrapper for distance function, converts value to number
"""
def get_distance():
    distance = dist_sensor.get_distance_cm()
    if distance == None:
        distance = 0
    return distance


"""
Decides whether robot is close enough to the obstacle
"""
def find_obstacle():
    distance = get_distance()
    if distance > 1 and distance < obstacle_distance_limit:
```

```
            return True
        return False

"""
Turns robot 90 degrees to the left side
"""
def turn_left():
    movement_motors.start(-100,10)
    while True:
        if abs(hub.motion_sensor.get_yaw_angle()) > 89:
            movement_motors.stop()
            hub.motion_sensor.reset_yaw_angle()
            return


"""
Turns robot 90 degrees to the right side
"""
def turn_right():
    movement_motors.start(100,10)
    while True:
        if abs(hub.motion_sensor.get_yaw_angle()) > 89:
            movement_motors.stop()
            hub.motion_sensor.reset_yaw_angle()
            return

"""
Calibrates robot arms to correct starting positions
"""
def calibrate():
    body_motor.run_for_seconds(3,5)
    body_motor.run_for_rotations(2.3, -50)


"""
Wall follower algorithm implementation
"""
def scan_environment():
    left_obstacle_found = False
    right_obstacle_found = False
```

```
    front_obstacle_found = False

    turn_left()
    left_obstacle_found = find_obstacle()
    if not left_obstacle_found:
        return

    turn_right()
    front_obstacle_found = find_obstacle()
    if not front_obstacle_found:
        return


    turn_right()
    right_obstacle_found = find_obstacle()
    if not right_obstacle_found:
        return

    turn_right()


"""
Main loop of the program and response for finding an exit
"""
def start_program():
    calibrate()
    body_motor.run_for_rotations(2, 50)
    hub.status_light.on("red")
    hub.motion_sensor.reset_yaw_angle()
    timer.reset()
    while True:
        distance = get_distance()
        color = color_sensor.get_color()
        movement_motors.start(0,50)
        if (distance > 1
        and distance < obstacle_distance_limit)
        or timer.now() == 2:
            movement_motors.stop()
            if color == "white":
                break;
```

```
            scan_environment()
            timer.reset()
            movement_motors.start(0, 50)
    hub.status_light.on("green")
    hub.light_matrix.show_image("HAPPY")
    hub.speaker.play_sound("Celebrate")

# Main input of the program
start_program()
```

Jednoduchá úloha pro BBC Micro:bit

Zadání úlohy - krokoměr

Naprogramujte jednoduchý krokoměr. Použijte integrovanou funkci `Shake` pro detekci kroku. Jedním z tlačítek počet kroků vypište a druhým počet kroků nastavte na hodnotu 0.

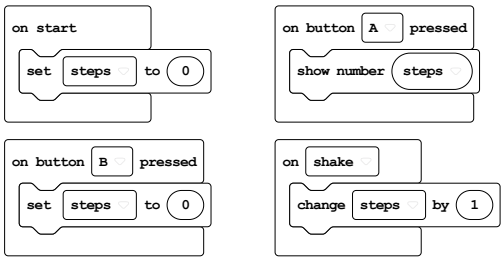
Stručný návod pro implementaci

1. V aplikaci *Microsoft Make Code* založte nový projekt.
2. Ze sekce `Input` vyberte blok zodpovědný za vyvolání interakce na stisk tlačítka (`on button A pressed`) a vložte ho do prostoru, ve kterém je tvořen program.
3. Postup z minulého bodu zopakujte, ale v bloku změňte název tlačítka na *B*. Název tlačítka lze měnit kliknutím na písmeno A nebo šipku dolu.
4. V sekci `Variables` vytvořte novou proměnnou a pojmenujte ji `steps`.
5. V sekci `Variables` vyberte blok `set variable` a umístěte do bloku `on start`. Hodnotu této proměnné ponechte na 0.
6. Postup z minulého bodu zopakujte i pro stisk tlačítka B
7. Ze sekce `Input` vyberte blok zodpovědný za vyvolání interakce na zatřesení (`on shake`) a vložte ho do programu.

8. Do bloku `on shake` vložte ze sekce `Variables` blok `change by`, sdružený s proměnou `steps`, a inkrement ponechte na hodnotě 1.

9. Do bloku `on button A pressed` vložte ze sekce `Basic` blok `show number`.

10. Do bloku `show number` pak následně vložte ze sekce `Variables` blok `steps`. Program by v tuto chvíli měl vypadat následovně:

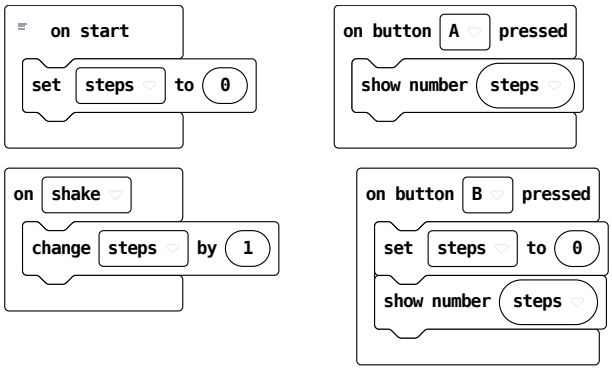


11. Blok `show number` můžete vložit i do bloku pro stisknutí tlačítka B, pro aktualizaci displeje po resetování.

12. Otestujte program pomocí emulátoru umístěného v levém rohu aplikace nebo nahrajte do mikročipu a otestujte na fyzickém zařízení.

13. V případě chyby porovnejte s referenčním zdrojovým kódem a opravte.

Referenční řešení



Obrázek 11.2: Referenční řešení jednoduché úlohy pro Micro:bit

Mírně pokročilá úloha pro BBC Micro:bit

Zadání úlohy - kámen, nůžky, papír

Vytvořte jednoduchou hru pro dva hráče - *kámen, nůžky, papír*, realizovanou po rádiovém spojení. Pro komunikaci mezi dvěma zařízeními využijte modul `radio`. Po odehrání partie program restartujte. Výběr předmětu bude volen pomocí tlačítek A a B. Jeho potvrzení bude realizováno pomocí stisknutí obou tlačítek současně.

Stručný návod pro implementaci

1. V aplikaci *Microsoft Make Code* založte nový projekt.
2. Vytvořte si celočíselnou proměnnou `counter`, ve které se bude uchovávat aktuální výběr hráče.
3. Tuto proměnnou pomocí jednoho z tlačítek inkrementujte a pomocí druhého z tlačítek dekrementujte.
4. Vytvořte si funkci, která pomocí operace modulo převede proměnnou `counter` na hodnotu 0, 1 nebo 2. Každá hodnota bude reprezentovat jeden objekt.
5. Ve stejné funkci pro každou hodnotu vymodelujte její grafickou reprezentaci. Siluety objektů mohou vypadat například takto:

```
Nůžky - 0          Kámen  - 1          Papír - 2

# . . . #          . . . . .          # # # # #
# # . # .          . # # # .          # . . . #
# # # . .          . # # # .          # . . . #
# # . # .          . # # # .          # . . . #
# . . . #          . . . . .
```

```
# # # # #
```

6. Do displeje zobrazte siluetu dle proměnné `counter`.
7. Na začátku programu nastavte výchozí rádiovou skupinu pomocí funkce `radio.setGroup()` a zobrazte do displeje výchozí obrazec.
8. Vytvořte funkci, která bude na stisknutí obou tlačítek odesílat hráčův aktuální výběr. Použijte k tomu funkci `radio.sendNumber()`.
9. Vytvořte si proměnnou `result_sent`, ve které bude uložena informace o tom, jestli již hráč výsledek odeslal.
10. Pokud hráč výsledek odeslal zablokujte možnost přepínání objektů pomocí tlačítek.
11. Vytvořte proměnnou `opponent_result`, ve které bude uložen protihráčův výsledek.
12. Výsledek protihráče získáte pomocí funkce `radio.onReceivedNumber()`.
13. Vytvořte funkci, která porovná hráčův výběr s protihráčovým a rozhodne o výsledku.
14. Výsledek pak nějakým způsobem zobrazte do displeje (text, emotikony...) a resetujte aplikaci pomocí `control.reset()`. Vyhodnocení výsledku může vypadat například takto:

```
function showResult(opponentChoice: Number) {
    const myChoice = Math.abs(counter % 3)
    if (myChoice == 0 && opponentChoice == 2 ||
        myChoice == 1 && opponentChoice == 0 ||
        myChoice == 2 && opponentChoice == 1) {
        basic.showString("Winner!")
    } else if (myChoice == 0 && opponentChoice == 1 ||
        myChoice == 1 && opponentChoice == 2 ||
        myChoice == 2 && opponentChoice == 0) {
        basic.showString("Loser!")
    } else {
        basic.showString("Tie!")
    }
    control.reset()
}
```

15. Funkce rozhodující o výsledku se bude volat ve dvou místech:

    (a) po odeslání Vaší volby v případě, že protihráč již svou odeslal

    (b) po přijmutí zprávy v případě, že jste již odeslali svou volbu

16. Pomocí proměnných `opponent_result` a `result_sent` zajistěte, aby byl výsledek zobrazen oběma hráčům současně.

17. Otestujte program pomocí emulátoru umístěného v levém rohu aplikace nebo nahrajte do mikročipu a otestujte na fyzickém zařízení.

18. V případě chyby porovnejte s referenčním zdrojovým kódem a opravte.

Referenční řešení

```
// Saves opponents choice or evaluates results
radio.onReceivedNumber(function (receivedNumber) {
    if (result_sent) {
        showResult(receivedNumber);
    } else {
        opponent_result = receivedNumber
    }
})



// Changes LED display based on counter value
function changeLEDs () {
    if (counter % 3 == 0) {
        basic.showLeds(`
            # . . . #
            # # . # .
            # # # . .
            # # . # .
            # . . . #
            `)
    } else if (Math.abs(counter % 3) == 1) {
        basic.showLeds(`
            . . . . .
            . # # # .
            . # # # .
            . # # # .
            . . . . .
            `)
    } else {
```

```
        basic.showLeds(`
            # # # # #
            # . . . #
            # . . . #
            # . . . #
            # # # # #
            `)
    }
}

// Incremets counter representing players choice
input.onButtonPressed(Button.A, function () {
    if (!(result_sent)) {
        counter += 1
        changeLEDs()
    }
})

// Sends players choice to the opponent
input.onButtonPressed(Button.AB, function () {
    radio.sendNumber(Math.abs(counter % 3))
    result_sent = true
    if (opponent_result || opponent_result == 0) {
        showResult(opponent_result);
    }
})

// Decrements counter representing players choice
input.onButtonPressed(Button.B, function () {
    if (!(result_sent)) {
        counter += -1
        changeLEDs()
    }
})

// Compares players and opponent choices and evaluates result,
function showResult(opponentChoice: Number) {
    const myChoice = Math.abs(counter % 3)
    if (myChoice == 0 && opponentChoice == 2 ||
    myChoice == 1 && opponentChoice == 0 ||
    myChoice == 2 && opponentChoice == 1) {
        basic.showString("Winner!")
    } else if (myChoice == 0 && opponentChoice == 1 ||
    myChoice == 1 && opponentChoice == 2 ||
    myChoice == 2 && opponentChoice == 0) {
        basic.showString("Loser!")
    } else {
        basic.showString("Tie!")
```

```
    }
    control.reset()
}

/**
 * Declaration and initialization of variables
 */
let result_sent = false
let counter = 0
let opponent_result: number;
basic.showLeds(`
    # . . . #
    # # . # .
    # # # . .
    # # . # .
    # . . . #
    `)
radio.setGroup(1)
```

Pokročilá úloha pro BBC Micro:bit

Zadání úlohy - herní konzole

Vytvořte program, který přemění *BBC Micro:bit* na herní konzoli. Ve výchozím stavu se na displeji zobrazí číslo reprezentující hru. Umožněte tlačítky A a B číslo zvyšovat a snižovat. Stisknutím obou tlačítek potvrdíte výběr. Po potvrzení budou tlačítka sloužit k ovládání hry. Návrat do nabídky her bude realizován pomocí gesta *Shake*. V rámci návodu bude popsána implementace hry *Flappy Bird* a *Snake*. Dále můžete zkusit *Snakes & Ladders*, *Pong*, *Space Invaders* nebo jinou hru dle vlastního výběru.

Stručný návod pro implementaci

1. V aplikaci *Microsoft Make Code* založte nový projekt.
2. Vytvořte proměnnou, kterou budete inkrementovat stisknutím jednoho z tlačítek a dekrementovat stisknutím druhého. Současně se změnou hodnoty proměnnou zobrazte do displeje.
3. V levém dolním rohu klikněte na plus (vedle nápisu *Explorer*) a založte nové TypeScriptové soubor pro hry *FlappyBird* a *Snake*. V každém souboru definujte `namespace` do kterého budete vkládat funkce spojené s hrou.

Flappy Bird

1. Založte si dvě proměnné typu `LedSprite`, jednu pro hráče a druhou pro překážky.
2. Vytvořte funkci `startFlappyBird`, která se bude volat z hlavního modulu programu (`main.ts`). Uvnitř funkce vytvořte hlavní smyčku programu.
3. Nastavte hráčovi výchozí pozici a vytvořte čítač, který se bude inkremetnovat v každé iteraci smyčky.
4. Každé 3 iterace vygenerujte sloupcovou překážku s jedním volným blokem. Index volného bloku náhodně vygenerujte.
5. Vytvořte proměnnou pro skóre hráče. Na základě skóre nastavte rychlost hry. Rychlost hry lze nastavit parametrem funkce `basic.pause`.

6. Překážky v každé iteraci posuňte směrem vlevo. Využijte funkci `change`.
7. Pokud se překážka dostane na začátek LED displeje, odstraňte ji.
8. V případě, že překážka koliduje s hráčem (shodují se jejich souřadnice X i Y), ukončete hru a zobrazte skóre.
9. Vytvořte ovládání hry, stisknutím jednoho z tlačítek polohu hráče zvyšte a druhým snižte.
10. Před funkci obsahující hlavní smyčku programu a ovládání doplňte klíčové slovo `export` a upravte hlavní modul programu tak, aby s nimi dokázal pracovat.

Snake

1. Založte si dvě proměnné typu `LedSprite`, jednu pro hráče (reprezentace pomocí pole) a druhou pro jablko.
2. Dále si založte dvourozměrné pole, ve kterém budete evidovat všechny směry, kterými se může hráč vydat. Reprezentace směrů může vypadat například takto:
   `let smery: number[][] = [[1, 0], [0, 1], [-1, 0], [0, -1]]`
3. Dále bude třeba uložit aktuální směr hada, skóre a maximální přípustnou délku hada.
4. Vytvořte funkci, která bude obsahovat hlavní smyčku programu. Smyčka se bude pozastavovat pomocí `basic.pause`. Parametr této funkce by měl být závislý na aktuálním skóre.
5. Pomocí funkce `game.createSprite` vytvořte objekt hráče (hada) a jablka.
6. Vytvořte funkci, která posune hada o jedno pole aktuálním směrem. V této funkci proveďte také validaci tohoto pole. Pole nesmí být mimo displej a současně na něm nesmí být žádná část hada. Validaci doporučuji zapsat opět do vlastní funkce.
7. Vytvořte ovládání hada. Stisknutím tlačítek změňte směr hada vlevo a vpravo. Může se Vám hodit operace modulo.
8. V hlavní smyčce programu zavolejte v každé iteraci cyklu funkci, která posuna hada vpřed. Dále ověřte, jestli had nekoliduje s jablkem. Pro kolizi můžete využít funkci `isTouching`.

9. Pokud had koliduje s jablkem, přesuňte jablko na náhodnou pozici (POZOR - vynechte pozice na kterých je aktuálně had) a prodlužte ho o jedno pole. Nezapomeňte také zvýšit skóre.

10. Doplňte klíčové slovo `export` před funkce, které se budou využívat v souboru `main.ts` (hlavní smyčka programu a ovládání hada).

Dokončení

1. Vytvořte proměnnou, která se bude měnit v závislosti na tom, jestli hráč hraje nebo ne. Podle proměnné a čísla hry pak přiřaďte tlačítkům správnou funkci.

2. Stisknutím obou tlačítek spusťte právě vybranou hru.

3. Otestujte program pomocí emulátoru umístěného v levém rohu aplikace nebo nahrajte do mikročipu a otestujte na fyzickém zařízení.

4. V případě chyby porovnejte s referenčním zdrojovým kódem a opravte.

Referenční řešení

```
// Button A handler for game change / in game control
input.onButtonPressed(Button.A, function on_button_pressed_a() {
    if (!game_started) {
        if (game_counter > 1) {
            game_counter += -1
            basic.showNumber(game_counter)
        }

    } else {
        if (game_counter == 1) {
            FlappyBird.buttonPressedA()
        }

        if (game_counter == 2) {
            Snake.turnRight()
        }

    }

})
```

```
// Button AB handler for game selection
input.onButtonPressed(Button.AB, function on_button_pressed_ab() {
    if (!game_started) {
        if (game_counter == 1) {
            game_started = true
            FlappyBird.startFlapyBird()
        } else if (game_counter == 2) {
            game_started = true
            Snake.startSnakeLoop()
        }

    }

})

// Button B handler for game change / in game control
input.onButtonPressed(Button.B, function on_button_pressed_b() {
    if (!game_started) {
        if (game_counter < 2) {
            game_counter += 1
            basic.showNumber(game_counter)
        }

    } else {
        if (game_counter == 1) {
            FlappyBird.buttonPressedB()
        }

        if (game_counter == 2) {
            Snake.turnLeft()
        }

    }

})

// Resets game with shake gesture
input.onGesture(Gesture.Shake, function on_gesture_shake() {
    control.reset()
})

/**
 * Declaration and initialization of variables
 */
let game_started = false
let game_counter = 1
basic.showNumber(game_counter)
```

```
/**
 * Namespace including function and variables
 connected with FlappyBird
 */
namespace FlappyBird {

    // Variable initialization
    let obstacles: game.LedSprite[] = []
    let player: game.LedSprite = null
    let score : number = 0;
    let ticks : number = 0;

    /**
    * Main loop of the program
    *   - Obstacles are generated at the end of LED display
    *   - Every tick they get one cell closer to player
    *   - If obstacle collides with player game ends
    */
    export function startFlapyBird() {
        player = game.createSprite(0, 2)
        player.set(LedSpriteProperty.Blink, 300)
        while (true) {
            while (obstacles.length > 0 &&
            obstacles[0].get(LedSpriteProperty.X) == 0) {
                obstacles.removeAt(0).delete()
            }

            for (let obstacle of obstacles) {
                obstacle.change(LedSpriteProperty.X, -1)
            }

            if (ticks % 3 === 0) {
                let empty_y_coord = randint(0, 4)
                for (let i = 0; i <= 4; i++) {
                    if (i != empty_y_coord) {
                        obstacles.push(game.createSprite(4, i))
                    }
                }

            }

            if ((ticks - 2) % 3 === 0 && ticks !== 2){
                score++;
            }

            for (let obst of obstacles) {
                if (obst.get(LedSpriteProperty.X) ==
```

```
                player.get(LedSpriteProperty.X) &&
                obst.get(LedSpriteProperty.Y) ==
                player.get(LedSpriteProperty.Y)) {
                    game.setScore(score)
                    game.pause()
                    basic.pause(1000)
                    game.gameOver()
                }
            }

            ticks += 1
            basic.pause(Math.max(100, 1000 - score * 50))
        }
    }

    // Changes Y coordinate of player
    export function buttonPressedA() {
        player.change(LedSpriteProperty.Y, -1)
    }

    // Changes Y coordinate of player
    export function buttonPressedB() {
        player.change(LedSpriteProperty.Y, 1)
    }
}


/**
 * Namespace including function and variables
 * connected with Snake
 */
namespace Snake {
    /**
    * LedSprite based variable initialization
    * for interactive objects
    */
    let snake: game.LedSprite[] = [];
    let apple: game.LedSprite = null;
    let last_snake_part: game.LedSprite = null

    // Movement variables
    let directions: number[][]  = [[1, 0], [0, 1], [-1, 0], [0, -1]]
    let direction : number = 1
    let pos_y : number = 0
    let pos_x : number = 0

    let score: number = 0
    let maxLength : number = 10
```

```
// Turns snake left
export function turnLeft() {
    direction = (direction + 3) % 4
}

// Turns snake right
export function turnRight() {
    direction = (direction + 1) % 4
}

// Creates sprite array object representing snake
function createSnake(arr: Array<number>) {
    let result = [];
    result.push(game.createSprite(arr[0], arr[0]));
    result.push(game.createSprite(arr[0], arr[1]));
    return result;
}

// Checks whether coordinate colides with snake
function isSnake(x: number, y: number) {
    for (let part of snake) {
        if (part.x() == x && part.y() == y) {
            return true
        }
    }
    return false
}

// Shifts snake one cell forward
function moveForward() {
    pos_x += directions[direction][0]
    pos_y += directions[direction][1]
    if (!(validateCoords(pos_x, pos_y))) {
        gameOver()
    }
    snake.unshift(game.createSprite(pos_x, pos_y))
    last_snake_part = snake.pop()
    last_snake_part.delete()
}

// Reaction on the end of the game
function gameOver() {
    game.setScore(score)
    game.pause()
    basic.pause(1000)
    game.gameOver()
}
```

```
    // Moves apple to random generated position
    function moveApple() {
        let x = Math.randomRange(0, 4);
        let y = Math.randomRange(0, 4);
        while (isSnake(x, y)){
            x = Math.randomRange(0, 4);
            y = Math.randomRange(0, 4);
        }
        apple.goTo(x, y);
        apple.setBrightness(100);
    }
    /*
    * Validates whether coordinate is inside of the LED display
    * and not snake itself at the same time
    */
    function validateCoords(new_x: number, new_y: number) {
        return new_x >= 0 && new_x <= 4 && new_y >= 0
        && new_y <= 4 && !(isSnake(new_x, new_y))
    }


    /**
     * Main loop of the program
     *  - every tick moves snake one cell forward
     *    in selected direction
     * */
    export function startSnakeLoop(){
        snake = createSnake([pos_x, pos_y, pos_x + 1, pos_y]);
        apple = game.createSprite(2, 2)
        moveApple()
        while(true){
            if (game.isGameOver()) {
                return;
            }
            basic.pause(Math.max(100, 1000 - score * 50))
            moveForward()
            if (snake[0].isTouching(apple)) {
                if (snake.length < maxLength) {
                    snake.push(snake[snake.length - 1])
                }
                score += 1
                moveApple()
            }
        }
    }
}
```

Plný text zpětné vazby z Gymnázia Sokolov

Micro:bit sada úloh 1 - Den 1

- Vyzkoušeno ve třídě 3.A na Gymnáziu Sokolov, tercie
- 11:50 začátek hodiny
- 12:20 konec videa
- 12:25 měření kroků pravítkem, vytvořit program a kontrola programu u všech studentů se nestihlo.
- 12:35 konec hodiny

Micro:bit sada úloh 2 - Den 1

- Vyzkoušeno ve třídě 4.A na Gymnáziu Sokolov, kvarta
- 12:45 začátek hodiny
- 13:30 konec hodiny

Za celou hodinu jsme nestihli shlédnout video a současně vytvářet program v JavaScriptu, návod byl velmi rychlý a jelikož studenti neměli s JavaScriptem žádnou zkušenost, tak přestože se snažili, nestihli to. Příští hodinu dokoukáme a doděláme, vyzkoušíme a pokusíme se doplnit program o přidané úlohy.

Vlastní postřehy z 1. hodiny by Zdeňka Bábíčková

Datum: 20. 4. 2022
Obě třídy měli zkušenosti s blokovým programováním, kvarta i s micro:bity. Tercii nedělaly micro:biti problém, již znali scratch, code.org i programování ozobotů v blokovém prostředí. Tercie zvládla základní úlohu dobře, i s přehráním videa vše zabralo cca 30 minut. Kvarta má dostatečné znalosti s blokovým programováním, měli ho již v tercii. S JavaScriptem bohužel zkušenosti žádné neměli, proto jim dělalo problém vše rychle psát a program vytvářet. Ve kvartě byl navíc jiný problém - velmi rychlá práce s tímto scriptovacím jazykem, rychlé kopírování + vkládání příkazů, tvarů kamene, nůžek i papíru. Musela jsem každou chvilku (cca vždy po 20 - 30 vteřinách) video pauznout a čekat, až si studenti vše napíší. Lepší by bylo mít dvouhodinovku, pak bychom stihli vše. Určitě doděláme úlohu příští hodinu (opět další středu, 27. 4.) a dopíši zpětnou vazbu i z druhé hodiny.

Postřehy k lego tutoriálům by Pavel Švácha

Video 1

Vhodné i pro primu (ne hned jako první hodina, chybí bližší seznámení s vývojovám prostředím) Je špatně rozumnět, dobré by bylo přidat titulky. Konec programu nebyl dostatečně dovysvětlený (záběr na kód končí hned po umístění posledního bloku - není vysvětleno, proč tam je)

Video 2

Velmi rychlé, žáci by ocenili mnohem detailnější vysvětlování. Není vhodné pro ZŠ, nejdříve první ročník se zkušeností v obecném Pythonu. Pokud by měli kód sami opisovat, chce to pokaždé zastavit a vysvětlit každý řádek (i souvislosti).