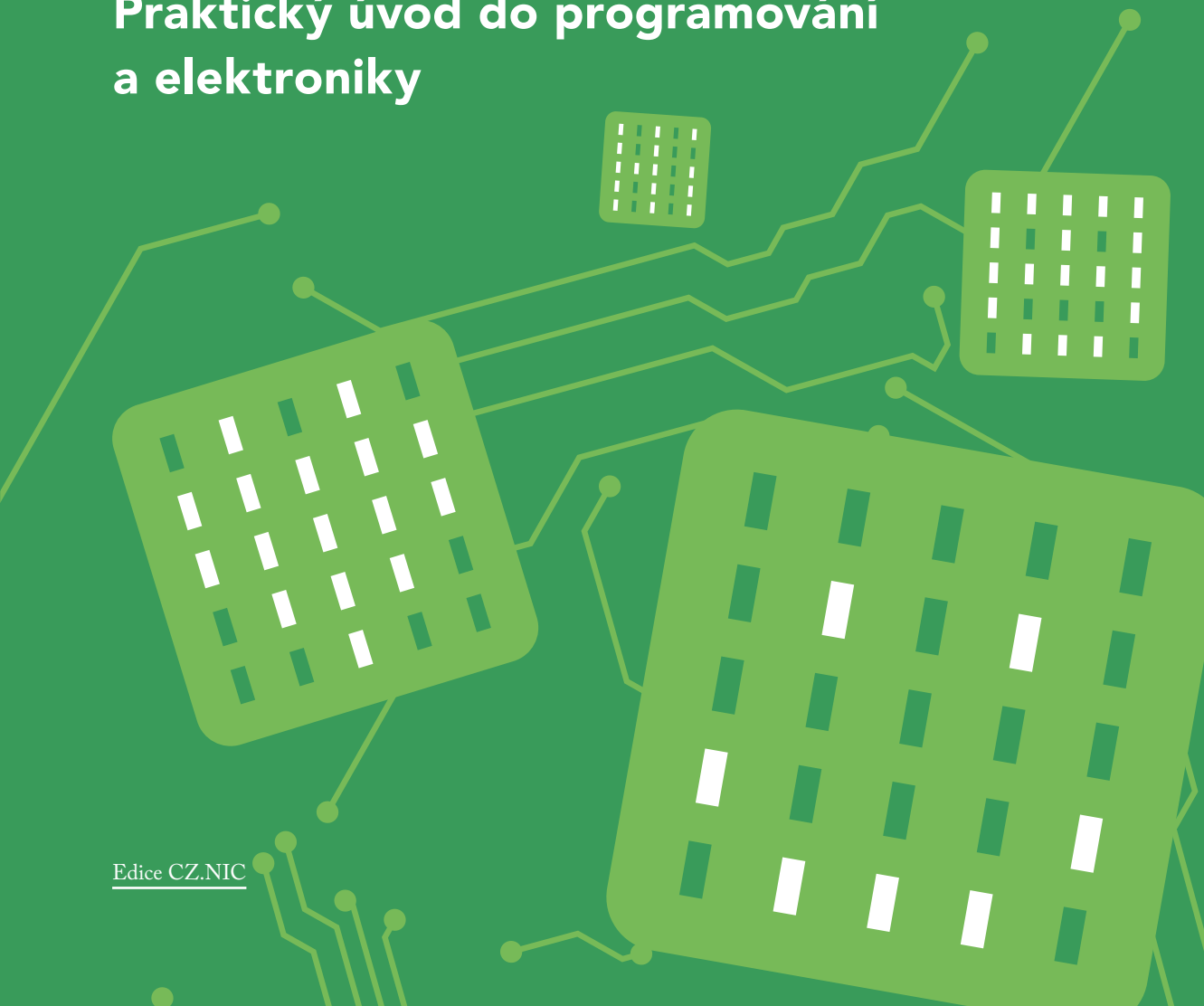
Martin Malý

Micro:bit krok za krokem

Praktický úvod do programování a elektroniky



Edice CZ.NIC

MICRO:BIT KROK ZA KROKEM
Praktický úvod do programování a elektroniky

Martin Malý

Vydavatel:
CZ.NIC, z. s. p. o.
Milešovská 5, 130 00 Praha 3
Edice CZ.NIC
www.nic.cz

1. vydání, Praha 2023
Kniha vyšla jako 29. publikace v Edici CZ.NIC.
ISBN 978-80-88168-70-6

CZ.NIC | SPRÁVCE DOMÉNY CZ

ISBN 978-80-88168-70-6

— Martin Malý

Micro:bit krok za krokem

Praktický úvod do programování a elektroniky

— Edice CZ.NIC

Poděkování

Poděkování

Před téměř dvěma lety mi napsal Oldřich Horáček, provozovatel „bastlířského“ e-shopu HWKITCHEN, že by možná nebylo od věci napsat knihu o microbitu. Že si to hodně kupují učitelé do škol, a ptají se po materiálech v češtině. V té době jsem dopsal poslední knihu své „elektronické trilogie“ a myslel jsem si, že tím moje osvětová činnost končí. Ale ten microbit mi stále vrtal v hlavě… Největší dík proto patří právě Oldřichu Horáčkovi, který mě na toto téma přivedl, stále mi ho jemně předhazoval a hodně mi pomohl se sběrem materiálu, s testováním a sháněním modulů, stavebnic a všeho okolo, a nakonec knihu laskavě přečetl a opřipomínkoval.

Děkuju všem lidem, kteří mi při psaní fandili. Všem čtenářům předchozích knih, co mi psali, že s nimi je začala elektronika zase bavit. Všem lidem, kteří mi psali, že se těší na pokračování. Všem těm, kteří četli rukopis a přispěli radami a názory. Všem těm, kteří mi dali ve chvílích pochybností podporou najevo, že to není úplně zbytečná práce, že to někoho zajímá a že nepíšu do zdi.

Děkuju Barboře Havířové za souhlas s využitím některých jejích materiálů, které připravila pro svou výuku a zveřejnila na webu microbiti.cz, a za cenné poznámky k rukopisu.

Velké poděkování patří vydavateli a všem z Edice CZ.NIC, kteří z mého strojopisu udělali už počtvrté knihu.

Díky!

— Poděkování

Předmluva vydavatele

Předmluva vydavatele

Vážení čtenáři,

právě držíte v ruce čtvrtou knihu Martina Malého, tentokrát na téma programování micro:bitu. Pokud s programováním teprve začínáte, může vám pomoci s prvními krůčky a zvládnete s ní svou první hru či animaci. S micro:bitem si ale vyhrajete, i pokud už jste ostřílenější programátoři.

Když jsem na střední škole, tedy před více jak 20 lety, začínal s programováním hardwaru já, měli jsme k dispozici pouze nepájivé pole a logické obvody. Největší vymožeností tehdy byly mikroprocesory řady 8051 a k nim připojený programátor běžící na obstarožní 486 s DOSem. A studenti, ukažte, co umíte. Pro naučení základů a proniknutí do tajů digitální techniky to stačilo. Ovšem jen hrstce z nás to připadalo úžasné a skvělé. Assembler, ať je jakkoli užitečný a vlastně jednoduchý, totiž prostě není sexy.

Později na vysoké škole jsme se setkali s již sofistikovanějším přístupem programování mikroprocesorů přes programovací jazyk C, který byl uživatelsky mnohem příjemnější. Tehdy jsem si říkal, že tím už bych se snad i živit mohl, ale vzpomínky na těžké začátky mě od toho stejně nakonec odradily.

Při pročítání knihy Micro:bit krok za krokem mě ihned napadlo, že mít podobnou k dispozici v dobách středoškolského studia, možná bych měl k programování přívětivější vztah. Jednoduché připojení různých periferií, práce s interními bloky a další funkce jsou totiž přesně to, co (nejen) mladého člověka může nadchnout a vzbudit v něm potřebný zápal pro věc. To samé přeji i všem čtenářům této knihy. A ať je pro vás tato kniha průvodcem, který ve vás vzbudí další zvědavost pro budoucí práci s hardwarem.

Petr Bílek, CZ.NIC
Praha, 9. srpna 2023

Předmluva

Předmluva

Vždy, když začnu psát novou knihu, představím si její čtenáře. Tedy vás. Kdo jste, proč jste po této knize sáhli, co v ní hledáte, co od ní čekáte? A mohu vám to splnit?

U této knihy se musím přiznat: představuju si několik různých skupin čtenářů, které se od sebe velmi liší. Jediné, co je spojuje, je zájem o elektroniku a programování.

Jedna skupina to o sobě ví, nebo alespoň tuší. To jsou lidé, kteří už přičichli třeba k programování, možná se tím i živí, ale rádi by vedle abstraktních programů zkusili programovat taky něco, na co si mohou sáhnout. Ovšem různých Arduin se bojí, protože to je příliš „elektronické". Microbit je pro ně jako stvořený – je jednoduchý, z určitého úhlu na použití jednodušší než třeba to Arduino, ale zase výkonnější a méně omezený. Křivka učení pro ně bude velmi příznivá. Stejně tak ti, co rozumí elektronice, nebo třeba mechanice, rádi by stavěli třeba roboty, ale nevěří si u programování.

Druhá skupina to o sobě třeba ještě ani neví. Jsou to mladí lidé, žáci a studenti, kteří jsou sice školou a jejími postupy už dostatečně otrávení, ale zároveň v nich ještě nezemřela radost z poznávání. Ironicky řečeno: *Kluci a holky, kteří mají radost, když se něco naučí, i přesto, že chodí do školy.* Věřím, že tihle mladí lidé ocení, že s microbitem mohou udělat spoustu zajímavých věcí, od legrácek a hříček po zajímavé aplikace, hry, nebo třeba řiditelné roboty. Pro starší může být microbit vstupenkou třeba do světa umělé inteligence, strojového učení nebo internetu věcí (IoT).

A třetí skupina souvisí s tou druhou. Jsou to nadšení učitelé, kteří mají chuť opravdu něco naučit, baví je to, co učí, chtějí, aby to bavilo i děti, ale hodně často naráží na limity školství a systému. Oni o microbitu vědí, jako vědí o Arduinech, Raspberry, o 3D tiskárnách a dalších věcech. Často takové sady nakoupí do školy, někdy i ze svého platu, prostě proto, že věří, že to má smysl a že dokážou svým žákům a studentům předat něco ze svého nadšení. Ale pak bohužel zjistí, že nejsou vhodné materiály, podle nichž by mohli postupovat.

Rád bych touto knihou oslovil všechny tři skupiny, jak nadšence, tak studenty i jejich učitele. Proto jsem do knihy zařadil nejen popis experimentů a konstrukcí s microbitem, ale k nim i pár pracovních a metodických listů. Kniha se kromě samotného microbitu věnuje i nejčastějším kitům, periferiím a stavebnicím, s nimiž se můžete na českém trhu a v českých školách potkat.

Věřím, že se mi podaří alespoň trochu zaplnit mezeru, která na českém trhu dlouho byla, a že kniha pomůže všem zájemcům o práci s microbitem v jejich prvních krocích.

Metodická poznámka

Metodická poznámka

Tuto knihu píšu na jaře roku 2023. Prostředí, v němž se microbity programují nejčastěji, je MakeCode. Najdete jej na adrese https://makecode.microbit.org/. Toto prostředí je z větší části přeloženo do češtiny. Někdy mám ke zvolenému překladu výhrady, ale většinou je překlad srozumitelný a pochopitelný. Proto se jej přidržím i v této knize. Co je přeloženo, budu zapisovat v přeložené podobě tak, jak je to přeloženo v době psaní knihy. Je možné, dokonce pravděpodobné, že po nějakém čase bude překlad vylepšený, změněný, nebo že části, které jsou zatím nepřeložené, budou dopřeloženy. Pro tyto případy poznamenávám i originální význam, aby byla kniha použitelná i do budoucna.

Výklad je sice zaměřen na začátečníky, ale předpokládám, že zájemce už ví, co je *počítač, programovací jazyk* nebo *program*, popřípadě že zná způsoby algoritmizace problému a podobně. Nevysvětluju, co je číslo a co je znak; u základních programových struktur vysvětluju jejich funkci jen obecně. Mým cílem není udělat z této publikace učebnici programování, k tomu sloužit nemá.

Při výkladu se soustředím především na prostředí blokového editoru MakeCode. Jsem si vědom toho, že pro pokročilejšího čtenáře není skládání bloků ideální způsob programování, přesto doporučuju, aby si i čtenář, znalý např. Pythonu nebo JavaScriptu / TypeScriptu, jednotlivé úvodní kapitoly prošel. Seznámí se tak s mnoha specifiky programování pro microbit. Pro snazší orientaci zařazuju do knihy výpisy některých příkladů v jazycích Python i JavaScript. Předpokládám, že zkušenější čtenář ví, co je např. podmínka nebo cyklus, a tak tyto věci osvětluju pouze pomocí bloků.

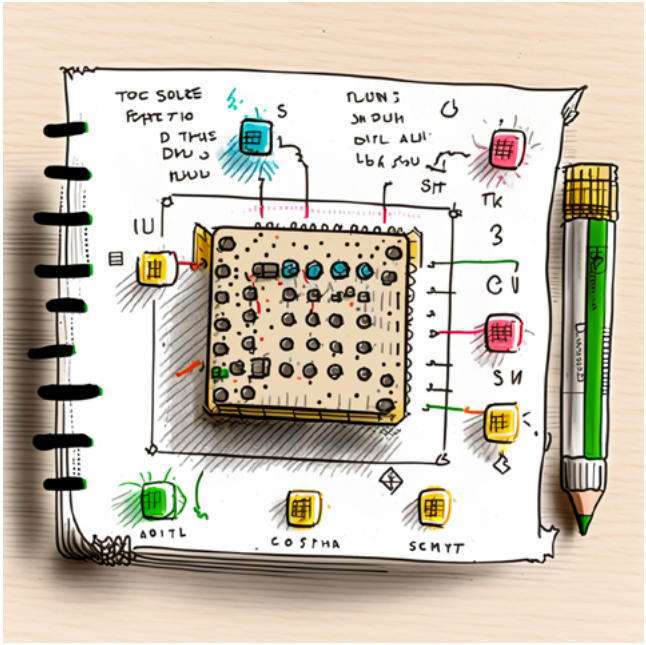
Knihu jsem rozdělil na několik částí. V první části probírám naprosté základy, včetně toho, co je proměnná a cyklus. Druhá část probírá pokročilejší techniky – rádio, kompas, magnet, práce s řetězci, seznamy atd. Ve třetí části se věnuju propojení microbitu a vnějšího světa. Čtvrtá část probírá pokročilá témata: mechanismus událostí a zpráv, vytváření vlastních rozšíření nebo některé alternativy k microbitu. Pátá část je věnována pokročilejším programátorům, kteří ovládají JavaScript a hodlají v něm programovat microbity. V dodatcích pak najdete schéma, praktické rady, jak například aktualizovat firmware, a v neposlední řadě pak odkazy na online materiály.

Jazyková poznámka

Jazyková poznámka

Originální název je „BBC micro:bit“ – se zjevným odkazem na legendární počítač BBC Micro z 80. let. V češtině se používá hned několik tvarů: s dvojtečkou i bez dvojtečky, s velkým M i s malým, někde i ve variantě „mikro…“ V této knize se budu držet varianty „microbit“ – tedy bez dvojtečky a s malým „m“. Varianta s dvojtečkou je ideální v titulcích nebo nadpisech, ale v běžném textu není její použití moc praktické, proto se budu držet tvarů microbit, microbitu, microbitem, microbity, …

U názvů funkcí, bloků a dalších prvků se řídím tou podobou, jakou mají v době psaní knihy. Některé jsou přeložené, pak použiju český aktuální překlad, i když není ideální a bude pravděpodobně změněn; na tuto možnost upozorňuju. Pokud nějaká část přeložena není, použiju anglický název a možný český překlad.



Obsah

1 Jak číst tuto knihu?

1 Jak číst tuto knihu?

Nemyslím si, že čtenáři nevědí, jak číst knihu. *Stačí přeci číst slova a na konci pravé stránky vpravo dole obrátit list.*

Přesto dovolte jedno doporučení:

Pokud začínáte s microbitem a s programováním, začněte hned následující kapitolou. Čtěte postupně, zkoušejte si popsané příklady, zkoušejte si jednotlivá cvičení a postupujte tak, jak jde výklad.

Pokud umíte programovat, ale s microbitem nemáte zkušenosti, tak následující kapitoly berte spíš jako rozcvičku a ukázku toho, co se s microbitem dá dělat. Nemusíte se zabývat „obrázkovým programováním", můžete si přepnout na Python nebo JavaScript dle své libosti a zkusit rovnou programovat „psaním kódu". Pravděpodobně nebudete vědět, jakou knihovnou a kde použít – editor MakeCode naštěstí vhodně napovídá a doplňuje.

Pokud si troufáte na microbit, ale rádi byste zkusili něco s hardware, tak můžete začít kapitolami o zabudovaných senzorech (akcelerometr, rádio) a pak plynule přejít na část o „ovládání světa pomocí microbitu".

Pokud jste už s microbitem něco dělali, tak oceníte možná kapitoly „pro pokročilé" – najdete v nich popis pokročilejších technik, popis mechanismu „událostí" (Message Bus) a obsáhlejší kapitolu o tvorbě rozšíření pro MakeCode a jejich lokalizaci. Pokud je vaším jazykem JavaScript, tak na konci knihy najdete obsáhlou kapitolu s výpisem dostupných zabudovaných funkcí, jejich popisem a stručným úvodem, kde je vysvětlen rozdíl mezi standardním JavaScriptem a TypeScriptem, použitým v MakeCode.

Pokud jste učitel a hodláte zapojit microbit do výuky, tak si přečtěte celou knihu, včetně příkladů, metodických listů a návodů, všechno si na vlastní kůži vyzkoušejte, všechny úkoly vypracujte a na všechny řečnické otázky odpovězte, *máte na to 30 minut, pak nekompromisně vyberu testy! Kdo bude opisovat, má automaticky nedostatečnou!* Ne, promiňte, nebudu vás zkoušet, ale ten zbytek myslím vážně. Opravdu si zkuste co nejvíc věcí a buďte připraveni na co nejvíc situací, které mohou při výuce nastat. Od problémů s připojeným hardware po „záhadné" chyby. Trochu jsem se snažil napomoci „metodickými listy" na koncích kapitol. Vyšel jsem hlavně ze skvělých materiálů Barbory Havířové (https://microbiti.cz), která laskavě souhlasila s použitím několika materiálů v této knize, za což jí ještě jednou děkuju. Navštivte i web https://m-bit.cz, kde je v Odkazech speciální sekce s metodickými a pracovními listy. *Držím vám palce!*

2 Microbit – úplně první kroky

2 Microbit - úplně první kroky

2.1 První program

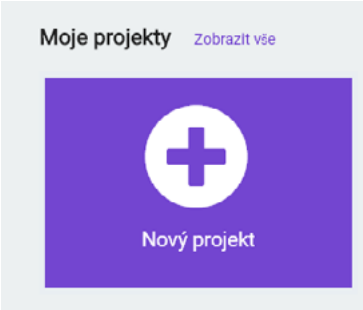
Vezměte microbit, připojte ho k USB kabelu (typ MicroUSB) a zapojte k počítači. Pokud je váš microbit úplně nový, spustí se ukázkový program, který předvede, co všechno microbit dokáže.

Zkuste jej chvíli pozorovat, zkuste si stisknout tlačítka A, B, zkuste dotykovou plošku a sledujte, co se děje. A až se vynadíváte, pustíme se do práce.

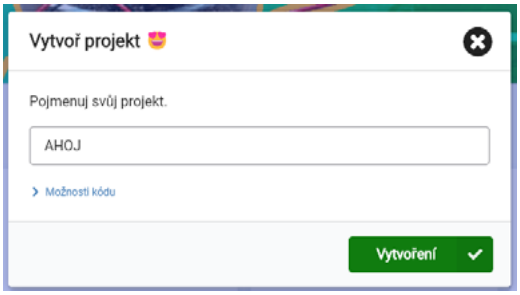
Každý programátor ví, že se začíná nápisem AHOJ SVĚTE. Nebo alespoň HELLO WORLD. U elektroniky je obdobou bliknutí LEDkou.

Na microbitu je ale celý displej, poskládaný z pětadvaceti LED v matici 5×5. Kterou rozsvítíme? No, uvidíme…

Otevřete prohlížeč a jděte na adresu https://makecode.microbit.org/ Najdete zde spoustu materiálů, ukázek, kurzů a námětů. Zůstaneme ale v sekci *Moje projekty* – tam klikněte na **Nový projekt**.

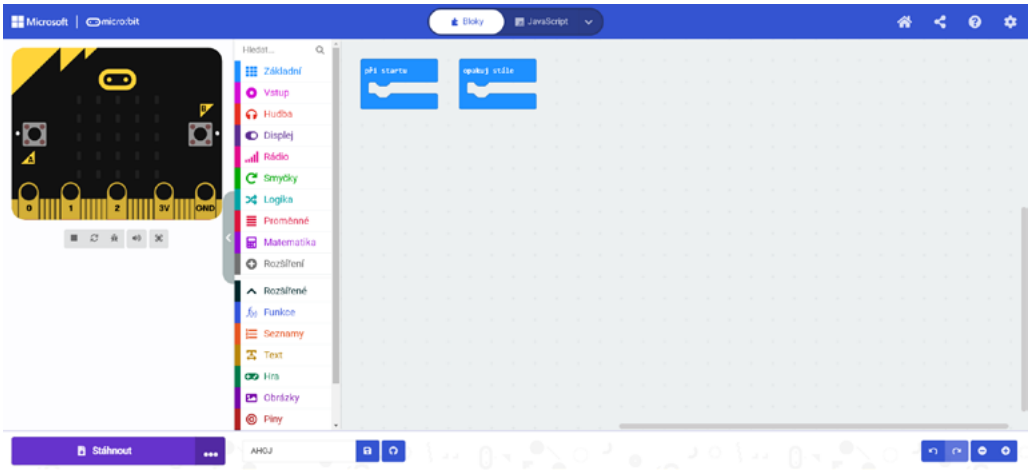


Náš nový projekt pojmenujeme třeba AHOJ a klikneme na tlačítko Vytvoření:



Po vytvoření nového projektu se ocitneme ve vývojovém prostředí microbitu, nazvaném „MakeCode“. Vlevo je simulátor microbitu, vedle něj sloupec se základními bloky, z nichž budeme program skládat, a vpravo prostor pro tvorbu programu. Nahoře je, jak je dobrým zvykem, menu, a dole pak několik ovládacích prvků, ke kterým se ještě dostaneme.

Pokud vám MakeCode připomíná jiný vizuální programovací nástroj, totiž Scratch, tak se nemýlíte. Blokový editor pro „skládání programů“ je totiž prostředím Scratch velmi silně inspirován. Není naprosto shodný, ale je tak podobný, že člověk, který zná Scratch, bude v MakeCode intuitivně chápat, co a jak dělat.



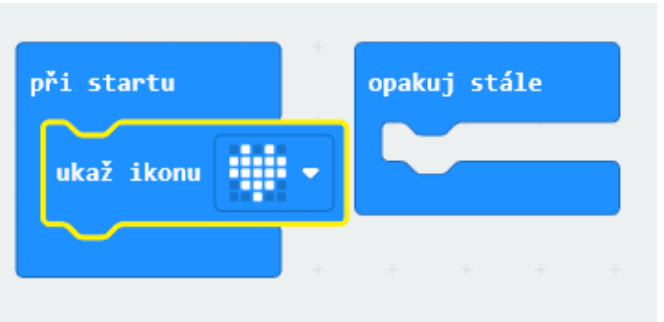
Na tomto místě musím uklidnit všechny odpůrce vizuálního programování: Nebojte, můžete použít i klasické programovací jazyky. Prostředí MakeCode přímo podporuje Python a JavaScript (respektive jejich „micro" dialekty), a lze použít i C/C++. Ale začnu s vizuálním editorem.

Klikněte na seznam bloků s názvem Základní (v originále Basic). Rozevře se nabídka bloků jako na následujícím obrázku:

Vezměte blok „Ukaž ikonu" a přetáhněte jej doprava na plochu editoru.

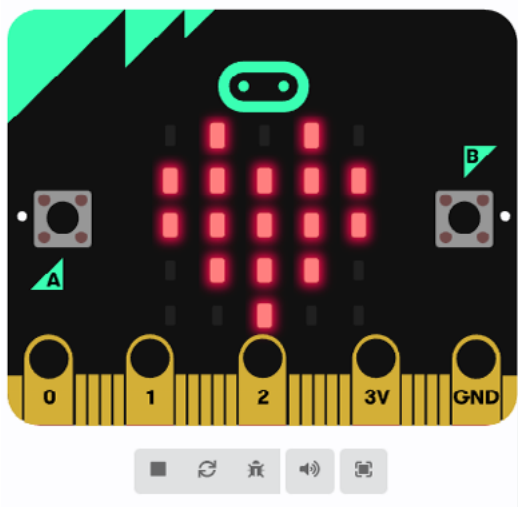
Všimněte si, že je zašrafovaná a vůbec vypadá velmi neaktivně. Myší ji přetáhněte do volného prostoru v modrém bloku „při startu“. Všimněte si, že tento blok má ve volném prostoru takové dva výřezy, do kterých blok „ukaž ikonu“ přesně zapadne. Když blok přetáhnete na vhodné místo, naznačí vám to editor tím, že ukáže, kam se blok připojí. Udělejte to.

Všimněte si, že jakmile na takovém místě pustíte blok, tak s jemným zacvaknutím zapadne přesně do vybraného místa a zmodrá.



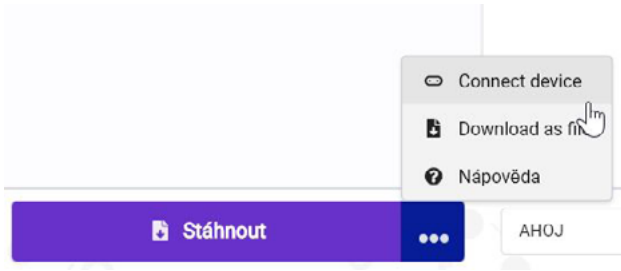
Tím jste vytvořili první program. Ano, je to tak. A tento program udělá přesně to, co je napsáno na obrazovce: **Při startu ukáže ikonu srdce.**

Po několika sekundách se program spustí i v simulátoru vlevo, a vy uvidíte výsledek:

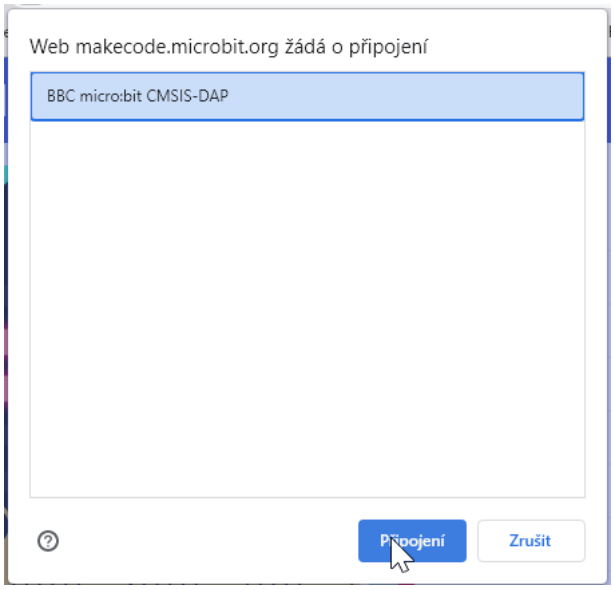


Takže program funguje, ukazuje to, co ukazovat má, ale pořád není v microbitu. Přišel čas, kdy jej tam nahrajeme.

Předpokládám, že máte microbit stále připojený kabelem k počítači. Klikněte na ikonu tří teček vedle tlačítka Stáhnout v levém dolním rohu a vyberte možnost „Connect device“ (v době psaní knihy ještě nebyla tato možnost přeložena do češtiny, pozn. aut.)



Průvodce připojením vám oznámí, že musíte v okně, které se objeví, vybrat možnost „BBC microbit CMSIS-DAP“ nebo „DAPLink CMSIS-DAP“ a potvrdit.



Všimněte si, že se změnila ikonka u tlačítka Stáhnout:

Z ikony „stažení dokumentu“ se stala ikonka microbitu. Teď klikněte na Stáhnout a nechte zázraky, ať se dějí. Po malé chvíli se rozsvítí nápis „Staženo“ (Downloaded) a na vašem microbitu se ukáže krásné červené srdce, úplně stejné jako v tom simulátoru.

Pokud se stane, že se microbit nepřipojí, např. máte starší prohlížeč, který nepodporuje standard WebUSB, tak můžete soubor stáhnout do počítače. Pak se například Průzkumníkem připojíte k microbitu, jako kdybyste se připojovali třeba k FLASH disku. Uvidíte ho mezi ostatními úložišti. Stažený soubor (má příponu .hex) jednoduše zkopírujte do microbitu. Jakmile se dokopíruje, microbit jej spustí.

Pokud máte podporovaný prohlížeč (Chrome, Edge atd.) a přesto se nelze připojit, jako se to stalo mně, můžete zkusit:

1. *Zkontrolovat, zda funguje ovladač winUSB, popřípadě jej přeinstalovat (utilitou Zadig)*

2. *Ve Správci zařízení odinstalovat zařízení mbed (pokud jste s ním někdy experimentovali, jako já)*

Po restartu už by mělo vše fungovat k plné spokojenosti.

Gratuluji. Právě jste v seznamu úkolů splnili ten úplně první: **Vytvořili jste program pro microbit a spustili ho!**

A teď si všichni dáme zmrzlinu…

Podívejte se znovu na svůj program. Vedle ikony srdce vidíte malou šipku, která naznačuje, že je víc možností, stačí si jen vybrat. Zkuste to a místo srdce vyberte smajlík. Program nahrajte do microbitu.

Metodický list

Co si připravit?

- Microbit, kabel USB – microUSB, propojení s PC
- Ověřit, zda PC s microbitem komunikuje
- Ověřit, zda je dostupné prostředí MakeCode
- Spárovat s microbitem přes WebUSB

Co jsme se naučili?

- Že programovat je fajn a zábava

Jak postupovat?

- Ideálně připravit microbity tak, aby byly připojené k PC a aby byly spárované s MakeCode
- Začít motivací – proč se učit programovat a že to může být i zábava
- Pomalu projít bod po bodu celý postup až do okamžiku, kdy se na displeji objeví ikona
- Podporovat je, ať se podívají, co tam jde vykreslit dál. Vysvětlit, že změnu v programu je potřeba „nahrát" do microbitu.

Úkoly a výzvy

- Změnit srdíčko na jinou ikonu
- Zvídavé nechat, ať se podívají například na blok „vypiš text", nebo ať zkusí použít dva bloky s výpisem ikony po sobě.

2.2 Než začneme doopravdy

Než se pustíme do všech těch skvělých a zajímavých věcí, co s microbitem lze dělat, položím vám jednu prostou otázku: **Máte microbit? A víte, co to vlastně je?**

Pokud ano, můžete klidně skočit na další kapitolu, protože v této kapitole se budeme věnovat právě tomu, kde microbit sehnat, jaké modely jsou k dostání, čím se od sebe liší a tak dál. Ale přesto bych vám doporučil tuto kapitolu nepřeskakovat, protože si řekneme i o doplňcích, které si můžete pořídit.

BBC microbit se představuje

Kdybych měl nějak popsat, co je microbit, tak použiju slova „miniaturní mikropočítač, zaměřený na výuku". Co tam tedy dělá to BBC? Je to opravdu zkratka britské veřejnoprávní televize?

Pokud znáte historii počítačů, víte, že britská BBC měla už v jednom počítači prsty. Šlo o známý počítač BBC Micro na začátku 80. let. Tehdy BBC začala s výukovým projektem počítačové gramotnosti, čímž reagovala na předpovědi mnohých odborníků, že počítače čeká obrovský boom. Při přípravě tohoto vzdělávacího programu došlo jeho autorům, že potřebují, aby v Británii byl dostupný počítač, ke kterému se budou moci lidé dostat, buď ve škole, nebo doma, a který bude v projektu použit jako referenční. Proto ve veřejné soutěži oslovili několik britských výrobců s poptávkou po takovém počítači. Nakonec vyhrála společnost Acorn se svým počítačem Proton, který byl přejmenován na BBC Micro, a úspěšně jej mnoho let dodávala na trh. Výukový projekt nakonec velmi výrazně nastartoval a podpořil to, co dnes známe jako boom domácích počítačů.

V desátých letech tohoto století si BBC opět vzpomněla na to, že jejím posláním je i vzdělávání, a rozhodla se přijít s podobnou iniciativou „Make It Digital". A během ní představila právě microbit – malý mikropočítač, vhodný pro výuku.

Proč raději nepředstavila obyčejné PC, nebo něco takového? Odborníci a učitelé z technických škol v té době upozorňovali na zajímavý jev. Zatímco v osmdesátých letech a na začátku devadesátých let nastupovali na technické školy studenti, kteří počítač uměli používat, programovat a chápali, jak pracuje, tak po přelomu století tahle znalost pomalu vymizela. Na školy se hlásili lidé, kteří uměli počítač používat. Programovat uměla jen velmi malá část z nich, a ještě menší část chápala, jak počítač vůbec pracuje. A důvodem bylo mimo jiné právě to, že u domácího počítače v osmdesátých letech jste museli umět aspoň trošičku programovat, abyste jej mohli použít, a abyste jej mohli programovat lépe, museli jste chápat, jak pracuje. Na druhou stranu, když jste jej chtěli programovat, stačilo ho zapnout – a už na vás blikal nejčastěji nějaký druh BASICu. Vy jste napsali PRINT 1+1, a byli jste programátor. Jenže s moderními PC jste uživatel. Což je samozřejmě dobře, jen to úplně neprobouzí ten druh zvídavosti, který vyučující na *technikách* očekávali.

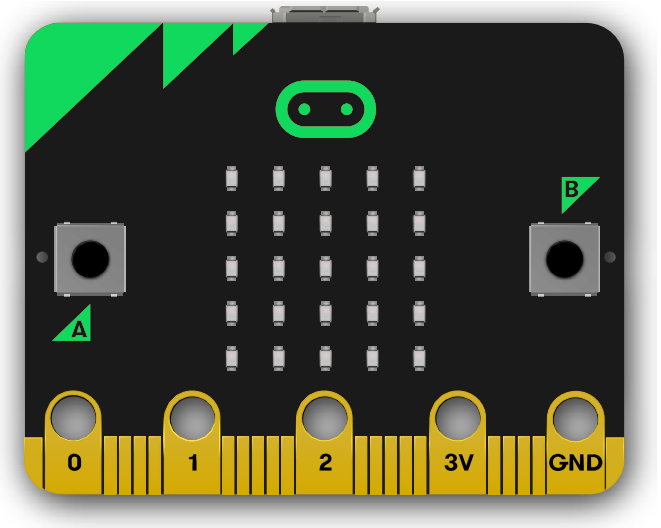
Proto v BBC chtěli počítač, který bude odolný, který bude mít periferie, který bude levný a z kterého budou, zjednodušeně řečeno, *viset dráty*. Počítač, který bude natolik jednoduchý, aby bylo možné ho pochopit. Počítač, kde bude programování základním způsobem, jak s ním komunikovat. A pokud možno bez složitých IDE a překladačů a knihoven.

A tak vznikl BBC microbit (též BBC Micro:Bit nebo Micro:Bit).

Verze 1? Verze 2?

Zařízení je vybaveno procesorem ARM Cortex-M0, senzory akcelerometru a magnetometru, připojením Bluetooth a USB, displejem tvořeným 25 LED diodami, dvěma programovatelnými tlačítky a může být napájeno z USB nebo externí baterie. Vstupy a výstupy zařízení jsou prostřednictvím pěti kruhových konektorů, které jsou součástí většího 25pinového konektoru na hraně. V říjnu 2020 byla vydána fyzicky téměř identická deska v2, která je vybavena mikrokontrolérem Cortex-M4F, větší pamětí a dalšími novými funkcemi.

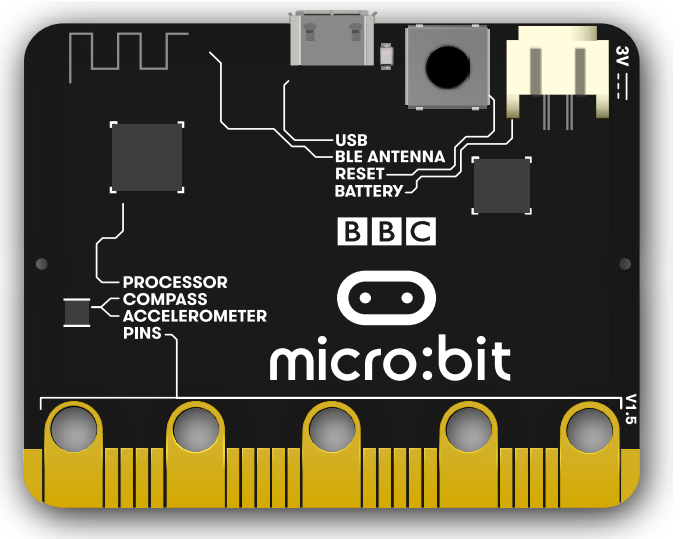
Verze 1



Fyzické rozměry desky jsou 43 × 52 mm a v první výrobní sérii zahrnovala:

- Nordic nRF51822 - 16 MHz 32bitový mikrokontrolér ARM Cortex-M0, 256 KB paměti flash, 16 KB statické paměti RAM, bezdrátovou síť Bluetooth low energy 2,4 GHz. Jádro ARM má možnost přepínat mezi frekvencí 16 MHz nebo 32,768 kHz.
- NXP/Freescale KL26Z - 48MHz mikrokontrolér s jádrem ARM Cortex-M0+, který obsahuje plnorychlostní řadič USB 2.0 On-The-Go (OTG), používaný jako komunikační rozhraní mezi USB a hlavním mikrokontrolérem Nordic. Toto zařízení také provádí regulaci napětí z napájení USB (4,5-5,25 V) na nominálních 3,3 V, které používá zbytek desky plošných spojů. Při provozu na baterie se tento regulátor nepoužívá.
- NXP/Freescale MMA8652 – tříosý akcelerometrický senzor, připojený na sběrnici I^2C.

- NXP/Freescale MAG3110 – tříosý magnetometrický snímač, připojený na sběrnici I²C (funguje jako kompas a detektor kovů).
- Konektor MicroUSB.
- Konektor pro připojení baterie.
- 25pinový hranový konektor.
- Displej tvořený 25 LED diodami v matici 5×5.
- Tři tlačítka (dvě pro aplikace, jedno pro reset).



Hranový konektor obsahuje i pět kruhových konektorů (jeden napájecí 3 V, jeden zemnící GND a tři datové vstupy/výstupy 0, 1, 2), do nichž lze zastrčit 4mm konektory „banánky“, nebo tzv. krokosvorky. Pro základní zapojení těchto pět pinů stačí.

Kompletní 25pinový hranový konektor nabízí podle konfigurace 2-3 výstupy PWM, 6 až 17 pinů GPIO, až šest analogových vstupů, sériové I/O, SPI a I²C.

Princip sběrnic SPI a I²C i s příklady využití najdete v knize Hradla, volty, jednočipy od stejného nakladatelství (dostupná i jako e-book zdarma).

Na rozdíl od prvních prototypů, které měly integrovanou baterii, lze k napájení zařízení jako samostatného nebo nositelného výrobku použít externí baterii (baterie AAA).

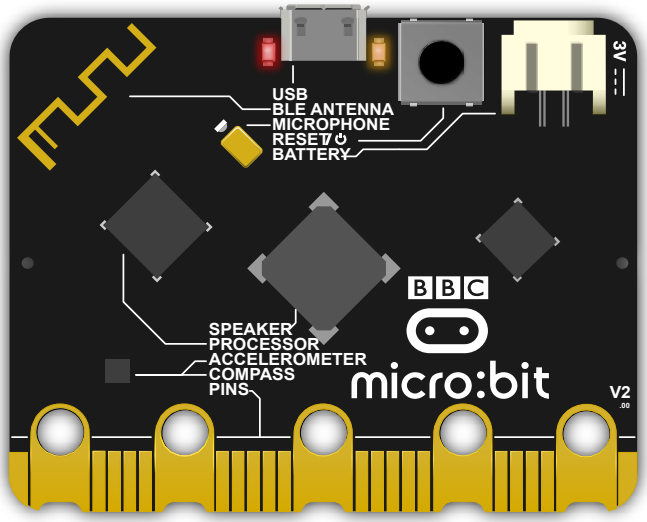
Dostupná dokumentace návrhu hardwaru se skládá pouze ze schématu a kusovníku šířeného pod licencí CC-BY, rozvržení desky plošných spojů není k dispozici. Existuje však plně dokumentovaný referenční návrh od Microbit Educational Foundation.

Verze 2

Druhá verze, vydaná 13. října 2020, obsahuje:

- Nordic nRF52833 - 64 MHz 32bitový mikrokontrolér ARM Cortex-M4, 512 KB paměti flash, 128 KB statické paměti RAM, bezdrátová síť Bluetooth low energy 2,4 GHz zajišťovaná zařízením Nordic S113 SoftDevice, integrovaný teplotní senzor.
- NXP/Freescale KL27Z - 48 MHz mikrokontrolér s jádrem ARM Cortex-M0+, předprogramovaný jako plnorychlostní řadič USB 2.0, používaný jako komunikační rozhraní mezi USB a CPU.
- ST LSM303 nebo NXP FXOS8700 – tříosý kombinovaný akcelerometr a magnetometr přes sběrnici I^2C.
- Mikrofon MEMS Knowles s vestavěným indikátorem LED.
- Reproduktor Jiangsu Huaneng MLT-8530.
- Konektor MicroUSB.
- Konektor pro připojení baterie JST PH.
- 25pinový hranový konektor.
- Displej tvořený 25 LED diodami v matici 5×5.
- Tři tlačítka (dvě pro aplikace, jedno pro reset) a dotykové tlačítko (logo).

U této verze microbit lze tlačítko reset použít k vypnutí desky jeho podržením po dobu 3 sekund.



Kde koupit?

Microbity nabízí mnoho obchodů, kamenných i online. Nemusí jít jen o specializované obchody s elektronikou, ale doporučuji začít právě tam, protože tam k Microbitu seženete většinou i nějaké základní příslušenství, nebo rovnou celé výukové sady se součástkami a moduly, které můžete připojit.

V České republice v době psaní knihy nabízely microbit například následující e-shopy:

- HWKITCHEN
- RPiShop
- Megarobot
- Rasel
- Farnell
- emo

Výčet není úplný ani autoritativní.

Co koupit dál?

Kromě samotného microbitu potřebujete už jen **kabel USB – MicroUSB**, kterým microbit připojíte k počítači a budete jej programovat. Většina z nás ho má doma, například od staršího telefonu nebo od jiného zařízení, ale doporučuju koupit nový, ideálně kratší, kolem jednoho metru. Díky tomu si ušetříte případné hledání problémů s „nefunkčním" microbitem (vadný kabel způsobuje „záhadné" výpadky, poruchy a problémy).

Pro první seznámení a vytvoření nějakého zajímavého zapojení toto bohatě stačí. Ale doporučuju po prvním seznámení pořídit i nějaký kit s dalšími součástkami. Kitům a stavebnicím se budu věnovat v příslušné části knihy, ale zde doporučím:

- Basic kit – jednoduchý kit s několika málo moduly, které rozšíří možnosti microbitu a demonstrují jeho propojení s vnějším světem
- Starter kit – vylepšený kit, který obsahuje kromě potřebných modulů i nepájivé kontaktní pole, sadu součástek a návod, takže nejste omezeni na předpřipravené moduly, ale můžete připojit *téměř cokoli*, co se vejde do nepájivého kontaktního pole.
- Ring:bit car – stavebnice robotického vozítka, které je řízené microbitem. Namísto „abstraktního" zapojování součástek zde stavíte konkrétní věc, která funguje autonomně, a kterou lze dle libosti rozvíjet.

- Nezha Inventor Kit – kit, který jde v tvořivosti asi nejdál. Microbit a sada senzorů je zde doplněna sadou základních kostek, kompatibilních s LEGO Technic, takže fantazii se meze nekladou. Problémem však může být vyšší pořizovací cena, která je na druhou stranu vyvážena vysokou variabilitou a možnostmi. Při stavbě konstrukcí jste omezeni jen dostupností součástek a vlastní fantazií.

Kde hledat inspiraci?

Největším problémem při zkoušení podobných věcí, jako je microbit, není ani tak zjistit, jak věci udělat, ale vymyslet, co udělat. Buď vás napadá všechno, nebo nic. Dobré je začít tím, co vymyslel už někdo jiný.

V této knize popisuju několik různých projektů, nápadů a návodů, včetně námětů k dalšímu rozvoji, ale kniha je omezená svým rozsahem. Další místa, kde hledat inspiraci, najdete buď v odkazech na konci knihy, nebo na webu knihy https://m-bit.cz/.

Na tomto místě mi dovolte uvést dva zdroje:

- Microbiti – stránky Barbory Havířové, která se specializuje na využití microbitů při výuce. Na webu má mnoho materiálů, které mohou být inspirací pro učitele, co s žáky či studenty probírat. https://www.microbiti.cz/
- Bastlírna HWKITCHEN – stránky Oldřicha Horáčka, kde najdete mnoho návodů a námětů k vlastní tvorbě. Nejsou zaměřeny didakticky, spíš techničtěji, ale jsou vhodné i pro začátečníky. https://bastlirna.hwkitchen.cz/category/novinky/tutorialy/microbit/

2.3 Začínáme

V této kapitole si projdeme úplné základy programování. Dozvíte se, co je proměnná, funkce, co je cyklus, co je podmínka, zkusíte si pracovat s blokovým editorem a naučíte se programovat microbit bez toho, abyste museli znát jediný příkaz v nějakém programovacím jazyce.

Předesílám to rovnou na začátku, abych předešel nedorozumění. Tato část je zaměřena na naprosté začátečníky, proto bude výklad velmi pomalý, plný příkladů, úloh a téměř výhradně orientovaný na blokový editor. Pokud jste si jisti, že toto všechno znáte, můžete tuto část přeskočit, nebo jen rychle prolistovat.

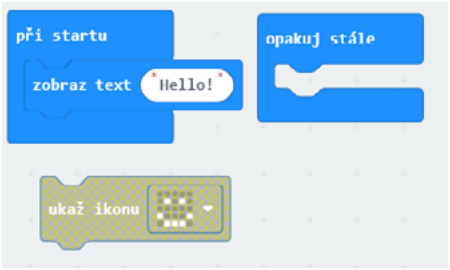
Základní nabídka

Text

Podívejte se do nabídky základních bloků, tam, odkud jste brali blok „ukaž ikonu“. Nad ním jsou bloky Zobraz číslo a Ukaž tvar, pod ním Zobraz text. Zkusme si teď vypsat to AHOJ.

Klikněte na „Zobraz text“. Blok se přesune do pracovního prostoru, ale nikam se nepřipojí. Nechte jej teď být – nejprve si musíme uklidit blok „Při startu“ – v něm zůstalo stále zobrazení ikony.

Vezměte blok „Ukaž ikonu“ v bloku „Při startu“ myší a odtáhněte jej kousek stranou, mimo blok. Když jej pustíte, zůstane ležet na ploše. Teď místo něj do bloku „Při startu“ odtáhněte blok „Zobraz text“.

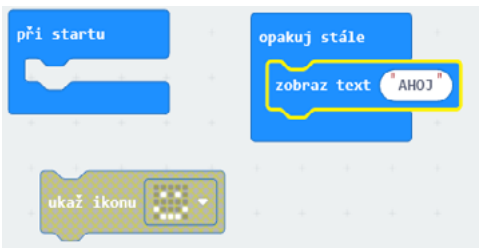


Jistě, můžeme být světoví a zobrazit Hello, ale řekli jsme si, že zobrazíme AHOJ, tak to taky dodržíme. Stačí kliknout na text Hello v bílé bublině a přepsat ho na AHOJ.

Jakmile se tak stane, uvidíte v simulátoru, že přes displej přejedou písmena A, H, O a J. A pak nic. Tma, černo. *To nebyla moc velká show, že?*

Problém je v tom, že microbit nemá moc možností, jak zobrazit delší text než ukázat písmeno po písmenu. A tak ukáže písmeno po písmenu a na konci – no, na konci je konec.

Udělejte takový trik. Vezměte blok „Zobraz text AHOJ“ a přesuňte jej z bloku „Při startu“ do bloku „Opakuj stále“.



A už teď v simulátoru uvidíte, že projede nápis AHOJ, a po něm projede nápis AHOJ, a po něm projede nápis AHOJ, a… No, bude se to dít stále a stále.

! *A to je rozdíl mezi těmi dvěma bloky. Jakmile nahrajete program do microbitu, nebo zapnete napájení, nebo stisknete tlačítko RESET, tak microbit udělá to, co má napsáno v bloku „Při startu". Udělá to přesně jednou. A pak udělá to, co je v bloku „Opakuj stále", a to, co tam je, opakuje stále.*

Zůstal nám na ploše jeden nepoužitý blok. Co s ním?

- Můžete jej odtáhnout myší doleva, do oblasti nabídky bloků. Zobrazí se ikona odpadkového koše. Když teď blok pustíte, smaže se.
- Můžete na něj kliknout a zmáčknout klávesu Delete. Smažete ho tím.
- Můžete kliknout pravým tlačítkem myši a vybrat „Odstranit blok".
- Nebo ho můžete znovu použít…

⚇ *Zkuste nepoužitý blok s ikonou použít znovu v bloku **Při startu.** Co se stane? A zkuste jej použít v bloku **Opakuj stále.** Zkuste jej dát před blok zobrazení textu, zkuste jej dát za něj. Co se stane?*

⚇ *Zkuste zadat v textu různé znaky, jako velká a malá písmena. Dokáže je microbit zobrazit? A co číslice? Dokáže zobrazit řadu číslic „0123456789"? A co třeba písmena s diakritikou? Dokáže microbit zobrazit slovo „Příšerný"? Umí tečku, dvojtečku, vykřičník, otazník…?*

Další základní funkce

Nabídka *Základní* obsahuje víc bloků pro práci s displejem.

Zobraz číslo zobrazí číslo

Ukaž tvar umí zobrazit jakoukoli kombinaci rozsvícených a zhasnutých bodů, kterou nakreslíte do mřížky 5×5 bodů

Zhasni displej udělá přesně to, co byste čekali: zhasne displej

Opakuj stále můžete použít, pokud chcete, aby probíhalo souběžně několik úloh, které se mají stále opakovat.

Při startu může být v programu jen jednou. Pokud si přihodíte na plochu další, tak se předchozí blok zneaktivní. Hodí se, pokud si jej omylem smažete.

Čekej se hodí k tomu, když potřebujete, aby nastala nějaká pauza. Hned si ukážeme jeho praktické využití.

Ukaž šipku funguje podobně jako Ukaž ikonu, ale místo nadefinovaných ikon pracuje s osmi světovými stranami (4 hlavní a 4 vedlejší). Hodit se bude při práci s kompasem, jak si ukážeme později.

- *Použijte blok „ukaž tvar" k zobrazení obrysu srdce, k zobrazení mřížky a k zobrazení vlnovky*
- *Ukažte na displeji šipku, která se bude točit stále dokola.*

Metodický list

Co si připravit?

- Microbit, kabel micro-USB, propojení s PC
- Ověřit, zda PC s microbitem komunikuje
- Spárovat přes WebUSB

Co jsme se naučili?

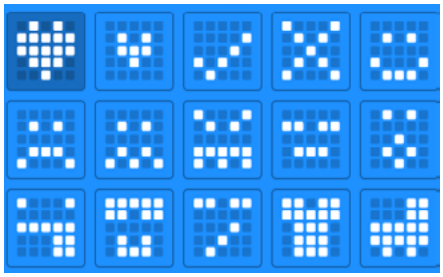
- Procvičili jsme práci s editorem
- Využití displeje pro výpis textů
- Základní bloky „po startu" a „opakuj stále"

Úkoly a výzvy

- Kombinace dvou různých výpisů – text a ikona
- Vlastní výzkum na téma „co všechno půjde vypsat? Speciální znaky? Čeština? Emotikony?"

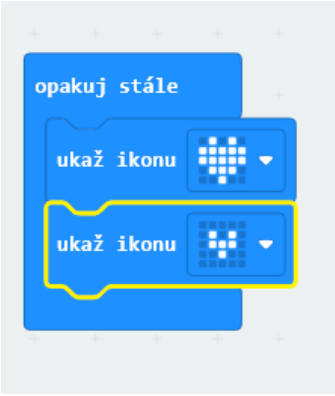
Blikající srdce

Pojďme si zkusit něco dalšího. Už víte, jak zobrazit srdíčko. Všimněte si, že v nabídce ikon jsou dvě srdíčka vedle sebe:



Pojďme si zkusit naše srdce rozpohybovat. Zobrazíme malé, a pak velké, a pak zase malé, a tak donekonečna.

Předpokládám, že vaše řešení vypadá nějak takto:



U microbitu zobrazení ikony chvilku počká, takže naše srdce hezky pravidelně tepe. Ale pojďme jej udělat tak, aby to menší bylo zobrazeno delší dobu než to větší.

Jestli nevíte, jak na to, zkusím napovědět: microbit by měl zobrazit velké srdce, pak menší srdce a pak chvíli **počkat**.

Už víte? Podívejte se do nabídky bloků v sekci Základní… Už vidíte?

(Jen dodám, že čas se zde udává v milisekundách, takže 500 ms je půl sekundy. A to je úplně ideální čas pro naše blikání.)

Zkuste přepracovat program „tepající srdce“ tak, že použijete funkci na ukázání šipky a budete šipkou točit kolem dokola…

Vyzkoušejte funkci „Ukaž číslo" a vypište postupně čísla 1, 2, 3, 4, 5. Co se stane, když má číslo víc číslic?

! **TIP:** *Pokud si chcete program stáhnout, můžete k tomu použít ikonku diskety dole vedle názvu programu. Uloží se na disk jako soubor typu .hex – tedy ve stejné podobě, v jaké ho můžete nahrát do microbitu. Takový program ale můžete přetáhnout zpátky do prostředí MakeCode, a on se opět otevře v původní zdrojové podobě.*

Krátká odbočka pro pokročilé

Už jsem předesílal, že prostředí MakeCode nabízí kromě programování pomocí skládání funkčních bloků i dva „dospělé" programovací jazyky, totiž Python a JavaScript (přesněji TypeScript). Pomocí přepínače v horní části můžete přepínat mezi blokovým editorem a řádkovými editory obou jazyků. Program, který vzniká, je stále tentýž. Když použijete blok „ukaž číslo 0" a podíváte se do zdrojového kódu pro Python, uvidíte:

```
basic.show_number(0)
```

Jak správně tušíte, jde o použití metody show_number (ukaž číslo) z balíčku basic (základní). Zkuste přepsat číslo 0 na 123 a podívejte se zpátky do blokového editoru– hodnota se změnila i zde.

Po přepnutí na JavaScript uvidíte:

```
basic.showNumber(123)
```

Tedy podobně jako u Pythonu. Python se drží konvence s podtržítkem, JavaScript konvence camelCase.

Všechny zatím použité bloky (tedy ze sekce Základní) mají své odpovídající metody v obou jazycích. Příklad s blikajícím srdcem vypadá takto:

```
JS  basic.forever(function () {
JS      basic.showIcon(IconNames.Heart)
JS      basic.showIcon(IconNames.SmallHeart)
JS      basic.pause(500)
JS  })
```

V Pythonu velmi podobně:

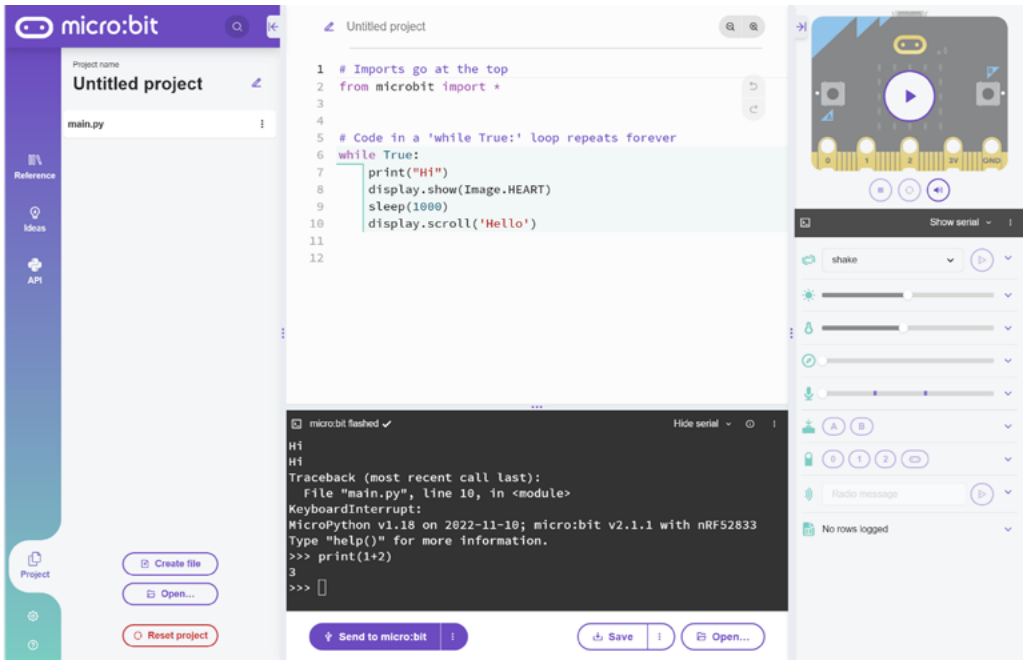
```
def on_forever():
    basic.show_icon(IconNames.HEART)
    basic.show_icon(IconNames.SMALL_HEART)
    basic.pause(500)

basic.forever(on_forever)
```

Můžete si svobodně vybrat, jestli použijete skládání bloků, Python nebo JavaScript.

Ve skutečnosti ale MakeCode Python není úplně *plnohodnotný Python*, je to „skript v jazyce pythonského typu“, tedy se stejnými syntaktickými pravidly, odsazením apod. Pro práci s microbitem ale můžete použít „skutečný (micro)Python“ – najdete jej na https://python.microbit.org/

Práce s MicroPythonem probíhá tak, že při prvním spárování a nahrání se v microbitu rozběhne interpret jazyka MicroPython, který můžete používat tak, jako se používá „velký“ Python, tj. včetně knihoven, souborů, importů, dokonce můžete využít i interaktivní REPL:



2.4 Kreslení

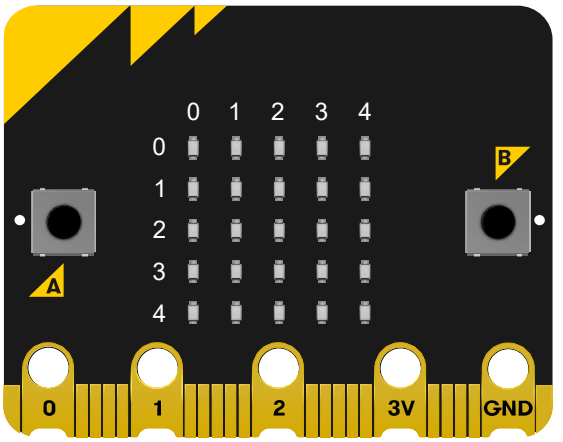
Naučili jste se, jak zobrazit na displeji microbitu tvary, znaky, písmena, číslice a ikony. Celý displej se přitom přepíše. Jak ale rozsvítit jenom jednu LEDku? A kterou?

Vytvořte si nový projekt a použijte blok, který najdete v nabídce Displej a jmenuje se „Rozsvit".

Tento blok má dva číselné parametry, X a Y.

Těmito parametry říkáte programu, která že LEDka má svítit. A jaká čísla tam mají být?

Podívejte se znovu na displej microbitu a všimněte si, že je uspořádán do mřížky 5 řádků × 5 sloupců LED. A teď si představte, že si sloupce i řádky očíslujeme od 0 do 4 z levého horního rohu. Takto:



Každá jednotlivá LED může být určena kombinací dvou hodnot: čísla sloupce a čísla řádku. Číslo sloupce je X (stejně jako v matematice se u grafů vodorovná osa označuje X), číslo řádku je Y (opět stejně jako v matematice, ale tentokrát je nula nahoře a hodnota roste směrem dolů).

Pokud tedy řekneme microbitu, aby rozsvítil LED na pozici X=0 a Y=0 (zkráceně můžeme zapsat „na pozici [X, Y]“, tedy [0,0]), měla by se rozsvítit LED v levém horním rohu.

Vyzkoušejte to.

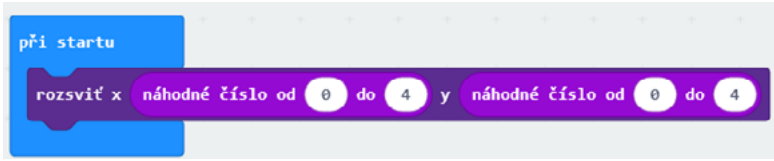
- *Zkuste si rozsvítit všechny LED ve všech rozích. Zkuste rozsvítit jen prostřední. Zkuste pomocí funkce „rozsvit“ nakreslit vodorovnou čáru uprostřed.*
- *Co se stane, když zadáte hodnotu větší než 4, nebo menší než 0?*

Druhý blok, který si teď ukážeme, je blok „Zhasni“. Jak už název napovídá, měl by rozsvícenou LEDku opět zhasnout. Se znalostí těchto dvou bloků už dokážete rozblikat třeba prostřední LED. Zkuste si to.

A teď uděláme trochu divočejší blikání, ano? Místo toho, abychom říkali microbitu, kterou LED má rozsvítit, necháme to na něm, ať si rozsvítí nějakou náhodně.

Použijeme zase blok „rozsvit“ a jako hodnoty X a Y nezadáme číslo, ale přetáhneme tam blok, který najdeme v nabídce Matematika a který se jmenuje „Náhodné číslo od-do“.

Všimněte si, že tento blok má oválný tvar a zapadá tak do bílého prostoru pro číslo. Nemůžeme ho tedy použít jako funkční blok, a stejně tak nemůžeme například blok „čekej“ použít jako hodnotu.



Všimněte si, že jsem změnil hodnoty „od“ a „do“. Blok teď můžeme číst tak, že se má rozsvítit bod na pozici [X, Y], kde X i Y jsou náhodná čísla od 0 do 4.

Když kód zkusíte spustit, rozsvítí se nějaký bod. Pokud jej spustíte znovu, rozsvítí se pravděpodobně jiný. Hezké, ale poněkud nudné. Zkuste proto celý blok „rozsvit“ přesunout do „Opakuj stále“.

To už je jiná podívaná, že? Displej se během krátké chvíle zaplní rozsvícenými body, a pak…? A pak nic! Program sice rozsvěcí stále a stále nové LED na náhodných pozicích, ale ty už svítí, takže se změna nijak neprojevuje.

Pro přesnost je potřeba dodat, že čísla nejsou náhodná, ale pseudonáhodná. Je to proto, že jsou počítána matematicky z nějaké úvodní hodnoty, a nejsou tedy úplně náhodná. Získat v počítači opravdu náhodné číslo je problém, a tak se používají převážně tato pseudonáhodná, která jsou pro naprostou většinu běžných aplikací dostatečně „nepředvídatelná“.

Zkuste funkci „rozsvit“ nahradit funkcí „přepni“ (opět v nabídce Displej). Jako X a Y dejte opět náhodná čísla a spusťte.

Funkce „přepni“ pracuje tak, že pokud LED na zadané pozici svítí, zhasne ji. Pokud je zhasnutá, rozsvítí ji. Proto se při každém zavolání vizuálně obrazec změní.


```
JS  function led.plot(x,y)
JS  function led.unplot(x,y)
JS  function led.toggle(x,y)
JS  function randint(min,max):number
```

```
    def led.plot(x,y)
    def led.unplot(x,y)
    def led.toggle(x,y)
    def randint(min,max):number
```

Metodický list

Co si připravit?

- Microbit, kabel micro-USB, propojení s PC
- Ověřit, zda PC s microbitem komunikuje
- Spárovat přes WebUSB

Co jsme se naučili?

- Ovládání displeje po jednotlivých bodech
- Adresace pozic na displeji pomocí dvojice souřadnic
- Náhodná čísla

Úkoly a výzvy

- V nabídce Displej – Více je blok pro rozsvícení LED s určitým jasem. Co jej použít u náhodného zaplňování displeje? Jak bude fungovat?

2.5 Vstup

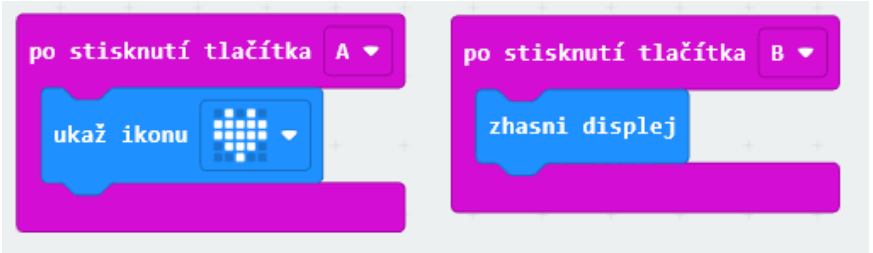
Určitě jste si všimli, že microbit má na přední straně dvě tlačítka, označená A a B. A teď nastala chvíle, kdy je zapojíme do našich experimentů.

A nejen tlačítka! Microbit obsahuje mnoho senzorů a vstupů, které si postupně představíme. Pokud chcete mít představu, co všechno naleznete na tak malé destičce, podívejte se do úvodní kapitoly.

Podívejte se do nabídky Vstup. Hned první funkční blok se jmenuje „Po stisknutí tlačítka…“ Přidejte ho na plochu.

Všimněte si, že se vzhledem ke svému tvaru nedá zapojit ani do „Po startu“, ani do „Opakuj stále“. Je samostatný a stojí mimo. Funguje vlastně nezávisle na vašem programu (programátoři jej nazvou třeba „obsluha asynchronní události“).

Dejme do něj naši oblíbenou funkci „ukaž ikonu srdce“. Přidejte ještě jeden tento blok, ale změňte tlačítko z A na B, a do něj dejte funkci „zhasni displej“.



Funkce tohoto programu je tak jednoduchá, že snad ani není potřeba popisovat, co se stane. Proto si tento program nahrajte do microbitu a vyzkoušejte, že dělá opravdu to, co má.

> *Psal jsem, že blok „Po stisknutí tlačítka" funguje nezávisle na hlavním programu. Co se stane, když hlavní program bude stále dokola kreslit jinou ikonu, třeba smajlík, a přitom někdo zmáčkne tlačítko A?*

Pojďme si udělat náročnější program, který bude počítat, kolikrát jsme stisknuli tlačítko.

Počítadlo

Tak, jak na to?

Je jasné, že základ bude blok „Po stisknutí tlačítka". A v něm bude blok „Zobraz číslo". A logicky, že tam bude jednička… Nebo ne?

To neřeší náš problém. My potřebujeme, aby se někde počítalo každé stisknutí tlačítka, a pak se zobrazil celkový součet. Potřebujeme nějaké místo, kam si dáme na začátku nulu, a po každém stisknutí přičteme jedničku…

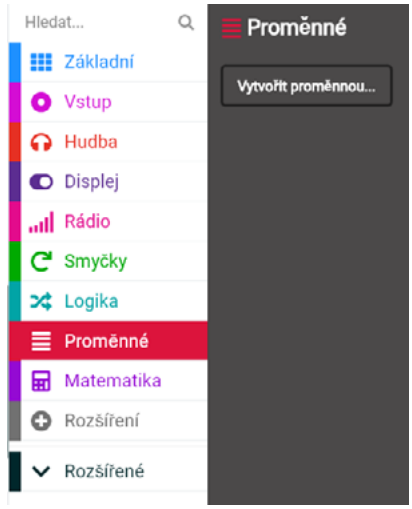
Přesně takové místo v programování existuje, a říká se mu „proměnná". To proto, že hodnota se může měnit. Proměnná má nějaké své jméno, třeba „A" nebo „xfh734", nebo třeba „pocitadlo".

Můžeme si proměnnou představit jako pojmenovaný kousek papíru, na kterém je napsáno obyčejnou tužkou číslo. Když je potřeba, můžeme ho vygumovat a napsat nové. Nebo jako tabuli, na kterou píšeme číslo křídou, smažeme a přepíšeme. Nebo jako pojmenovanou přihrádku, kam si

vložíme číslo. Zkrátka jako nějaké pojmenované místo, kde může být nějaká hodnota (nejen číslo), která se může změnit.

Dobrým zvykem je pojmenovat proměnné tak, aby bylo jasné, co v nich je, takže například ne Bflm, ale „pocitadlo". Taky je dobrým zvykem nepoužívat háčky a čárky. Některé jazyky to ani nedovolí. A důležité je nezapomenout na to, že většina jazyků má ještě nějaká další pravidla pro jména proměnných, takže ideální je držet se malých a velkých písmen, číslic a znaku podtržítko.

Takže si vytvoříme svoji první proměnnou. V hlavním menu je nabídka „Proměnné". Klikněte a zadejte „Vytvořit proměnnou".



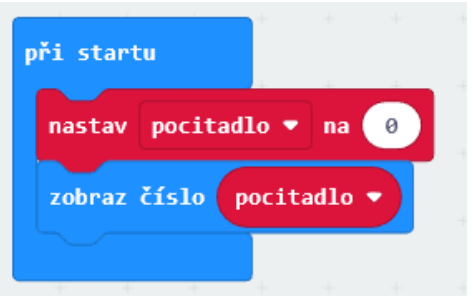
Proměnnou pojmenujeme „pocitadlo". V sekci „proměnné" se objeví dva nové funkční bloky „nastav proměnnou" a „změň proměnnou" a jeden blok, který reprezentuje hodnotu proměnné (ano, to je ten oválný). Rovnou je zkusíme použít.

Do bloku „Při startu" vložte dva bloky:

- Nastavení proměnné pocitadlo na hodnotu 0
- Zobrazení čísla

Jako hodnotu pro zobrazené číslo použijte hodnotu proměnné „pocitadlo".

Výsledek by měl vypadat takto:



Po spuštění se na displeji ukáže nula. Jupí. Zvládli jsme vytvořit proměnnou a použít ji. A teď použijeme ještě blok „Změň proměnnou“. Přetáhněte ho do bloku „Po stisknutí tlačítka A“ a – máme vlastně hotovo. Po stisknutí tlačítka A se stav proměnné „pocitadlo“ zvedne o 1.

Když program přeložíte a nahrajete, zjistíte jednu zradu. Tlačítko sice mačkáte, ale na displeji je stále „0“. Důvod je jednoduchý: Sice jsme změnili hodnotu, ale zapomněli jsme ji vypsat na displej!

Klikněte pravým tlačítkem na blok „Zobraz číslo ‚pocitadlo‘“ a vyberte možnost Klonovat. Celý blok i s vloženou hodnotou proměnné „pocitadlo“ se zkopíruje na plochu. Vezměte jej a vložte za zvýšení stavu proměnné (tedy do bloku „Po stisknutí tlačítka A“).

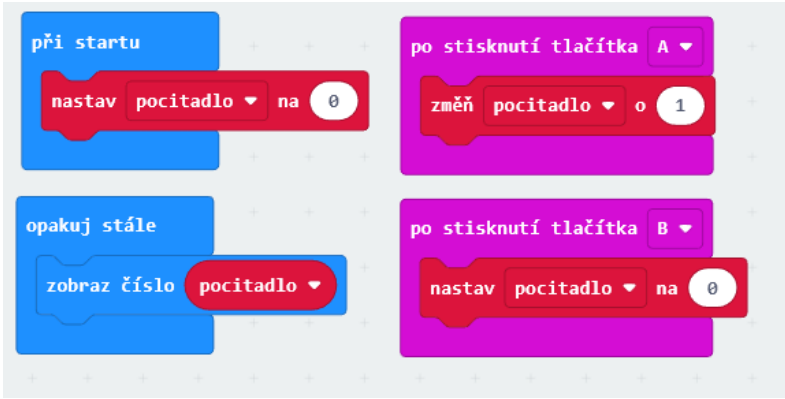
Teď už program krásně funguje. Počítá dál a dál…

🔗 *Přidejte funkci, která po stisknutí tlačítka B počítadlo opět vynuluje.*

Předpokládám, že tento úkol byl pro vás hračkou. Vlastně stačilo jen zopakovat to, co se děje v bloku Při startu.

🔗 *Co se stane, když stisknete tlačítko A několikrát rychle za sebou?*

A teď se podívejte na tento program. Co se změnilo? Jak se změní chování? A bude to vůbec fungovat?



Hlavní rozdíl je ten, že čísla jsou zobrazována neustále. Což u čísel 0 až 9 vidět není, ale od 10 výš už ano. Druhá změna je, že když stisknete tlačítko několikrát rychle po sobě, tak počítadlo skočí z 0 třeba rovnou na 4. Předchozí verze počítala 1… 2… 3… 4.

Důvod je ten, že microbit po zobrazení čísla čeká zhruba půl sekundy. Proto se může stát, že zobrazí hodnotu 0, půl sekundy čeká, vy během té doby dvakrát stisknete tlačítko, a pak, když se znovu zobrazuje hodnota proměnné, tam je už 2. Předchozí verze na dvojitý stisk zobrazila 1 a čekala půl sekundy. A po celou tu dobu čekala i informace o druhém stisku tlačítka…

Čekání se u funkce Zobraz číslo dá potlačit. Přepněte se např. do JavaScriptu, najděte řádek, kde je příkaz basic.showNumber(pocitadlo) a změňte jej na basic.showNumber(pocitadlo,0). Stisknutí budou započítána a zobrazena ihned. Bohužel, nulové čekání způsobí, že vícemístná čísla budou po displeji létat tak rychle, že nebudou k přečtení…

Metodický list

Co si připravit?

- Microbit, kabel micro-USB, propojení s PC
- Ověřit, zda PC s microbitem komunikuje
- Spárovat přes WebUSB

Co jsme se naučili?

- Koncept proměnné
- Ovládání pomocí tlačítek

Jak postupovat?

- Koncept proměnné vysvětlovat adekvátně věku či probíranému učivu v matematice. Abstrakce: „Krabička“

Úkoly a výzvy

- nulovat stiskem obou tlačítek najednou (A+B) a tlačítkem B snižovat o 1

Skokoměr

Microbit kromě tlačítek obsahuje i další zajímavé senzory. Jedním z nich je akcelerometr.

Akcelerometr je zařízení, které snímá zrychlení. Za normálního stavu, tedy v klidu, snímá směr zemského zrychlení a dokáže tak určit polohu zařízení vůči Zemi (tedy jestli leží na zadní straně, na přední, jestli stojí na boku atd.) Pokud se zařízení pohybuje, tak akcelerometr měří směr a velikost zrychlení. Lze tak měřit například otřesy, pád, švihnutí a podobné jevy.

V nabídce Vstup je hned pod blokem „po stisknutí tlačítka“ blok „při zatřesení“. Přidejte ho do programu počítadla a přesuňte do něj to, co je teď jako obsluha stisknutí tlačítka A. Tedy „změň pocitadlo o 1“.

Od této chvíle microbit nebude počítat stisknutí tlačítka A, ale počet, kolikrát s ním zatřesete. Zkuste si to.

⚯ *Jak silné zatřesení je potřeba? Stačí třeba otřesy od toho, jak chodíte? A co když budete microbit držet v ruce a budete s ním skákat na místě? Bude počítat poskoky? Změřte si, kolik poskoků uděláte za deset sekund…*

Blok „Při…“ dokáže zareagovat nejen na zatřesení, ale i na další takzvaná „gesta“. Gesto může být, že microbit otočíte displejem nahoru, displejem dolů, logem nahoru, logem dolů, že jej nakloníte doleva nebo doprava, popřípadě že zaznamená volný pád (opatrně!), případně nějaké přetížení.

⚯ *Upravte skokoměr tak, aby ho bylo možné vynulovat tím, že otočíte microbit displejem dolů.*

Hlukoměr

Microbit verze 2 obsahuje integrovaný mikrofon, který dokáže snímat hlasitost okolního prostředí. Můžeme tedy použít zvuk jako další způsob, jak microbit ovládat, například na písknutí nebo tlesknutí, anebo můžeme měřit úroveň hluku v okolí a nějak na něj reagovat.

My si z microbitu uděláme jednoduchý hlukoměr. Budeme měřit hluk a zobrazovat jeho velikost. Což je velmi snadný úkol, stačí jen v nekonečné smyčce („opakuj stále“) použít blok „Zobraz číslo“, a jako parametr zadat „úroveň hluku“ z nabídky Vstup (v době psaní knihy jako „sound level“).

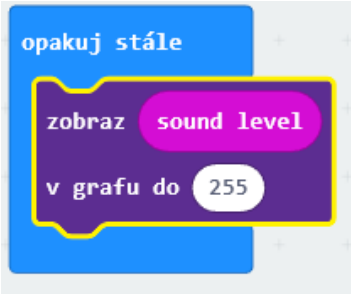
Když si to zkusíte, tak zjistíte dvě věcí:

1. Funkce „zobraz číslo“ je na toto velmi nepraktická, protože dlouho trvá, než číslo zobrazí

2. Zkoušet, jaké největší číslo dokážete „vykřičet“, je docela zábavné.

K druhému bodu: Já jsem se opravdu poctivě snažil, ale víc než 134 se mi nepodařilo. A to podle dokumentace je nejvyšší hodnota 255, takže jsem byl opravdu jen kousek za půlkou.

První bod je mnohem zásadnější. Naštěstí v nabídce Displej existuje blok, který se jmenuje „Zobraz N v grafu do M“. Použijte tento blok a zadejte ho jako „zobraz sound level v grafu do 255“.



Zobrazení v grafu funguje tak, že čím vyšší je hodnota, tím víc LED svítí. Je to taková opravdu jednoduchá, ale vizuálně účinná metoda, jak rychle ukázat nějakou hodnotu. Namísto čekání na nějaké číslo vidíme na první pohled, jestli je sledované hodnoty „hodně“, nebo „málo“.

> *Použijte měření zvuku v předchozím příkladu. Jen místo reagování na otřesy bude počítadlo reagovat na silný zvuk (loud). Počítejte tak tlesknutí.*

Teploměr, světloměr…

Předchozí příklad fungoval jen na microbitech verze 2. Lehce ho upravíme, a bude fungovat i na verzi 1.

Místo úrovně zvuku použijte nyní jinou funkci z bloku Vstup, a to hodnotu nazvanou „intenzita světla“. Další úpravy už nejsou potřeba, vše ostatní ponechte v původní podobě, a program nahrajte do microbitu.

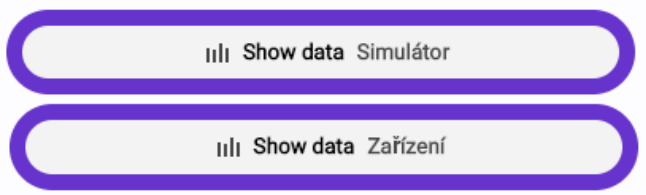
Nyní by graf měl ukazovat intenzitu osvětlení přední části microbitu. Zkuste si na něj posvítit silným světlem (baterka v mobilním telefonu), nebo jej naopak zastínit…

Jak microbit měří světlo, když v popisu nebylo nic jako „fotorezistor“, „fototranzistor“ ani nic podobného? Používá jednoduchý trik: světelná dioda (LED) dokáže fungovat nejen tak, že světlo generuje, ale i obráceně: že jej detekuje. Microbit měří v takovém případě světlo pomocí devíti diod v displeji a z jejich průměru vypočítává míru osvětlení, opět na škále od 0 do 255.

Obdobně můžeme místo hodnoty „intenzita světla“ použít hodnotu „teplota“. Jen ten graf bude poněkud nezajímavý, protože teplota se nezmění na 255 stupňů (a pokud ano, máme problém).

Zkuste si upravit graf tak, aby maximum bylo třeba 40 stupňů. Zkuste microbit zahřát – jakým způsobem by to šlo?

Když budete experimentovat s těmito snímači a budete mít připojený a spárovaný microbit k počítači, tak zjistíte, že se pod simulátorem objevily dvě tlačítka, nazvaná „Show data“ (v době psaní knihy ještě nebyla přeložená).



Pokud nemáte microbit spárovaný, druhé tlačítko se neobjeví. Ale pokud ho vidíte, tak si na něj klikněte a zjistíte, že váš microbit po USB stále posílá naměřenou hodnotu. Vidíte ji jednak v grafu nahoře, a jednak jako sloupec stále běžících čísel dole.

- *Změňte vstupní hodnotu z teploty opět na světlo. Prohlédněte si data v grafu a v číslech.*

- *Změňte vstupní hodnotu na zvuk. Pusťte si vizualizaci dat v grafu a zkuste sledovat, jak jsou různé aktivity různě hlučné. Zkuste experiment: položte microbit na stůl, vedle něj přidržte pravítko tak, aby přesahovalo hranu stolu zhruba polovinou svojí délky, a drnkněte na jeho volný konec. Sledujte, jak se v grafu původně silný zvuk plynule vytrácí. (Vyžaduje microbit V2)*

```
JS  let val = 0
JS  basic.forever(function () {
JS      val = input.temperature()
JS      led.plotBarGraph(
JS      val,
JS      40
JS      )
JS  })
```

Metodický list

Co si připravit?

- Microbit, kabel micro-USB, propojení s PC
- Ověřit, zda PC s microbitem komunikuje
- Spárovat přes WebUSB

Co jsme se naučili?

- Práci se snímači
- Fungování mikrofonu, akcelerometru, teploměru a čidla světla
- Na začátku každé práce s nějakým čidlem je dobré udělat si jednoduchý „testovací program“

Úkoly a výzvy

- „Screaming contest“ – měření hluku a zobrazení nejvyššího výsledku / průměru za sekundu. Kdo zakřičí nejhlasitěji?

2.6 Zvuk

Už ve verzi 1 byl zvukový výstup microbitu populární aplikací. Připojovalo se sluchátko mezi pin 0 a zem. A protože bylo využívání zvuku velmi časté, tak ve verzi 2 přibyl malý reproduktor přímo na desce a není proto potřeba nic připojovat.

Pokud máte verzi 2, a to pravděpodobně máte, můžete si následující experiment vyzkoušet hned. S verzí 1 si nejprve připravte zvukový výstup podle návodu na konci knihy.

Křeček

Možná se pamatujete na vlnu digitálního zvířátka jménem Tamagoči. My si uděláme mnohem jednodušší zvířátko. Takového digitálního křečka.

Digitální křeček bude fungovat podobně jako ten reálný: bude většinu času spát. To znázorníme ikonkou spícího obličeje. Když ho pohladíme (u verze 2 k tomu použijeme dotykové logo, u verze 1 třeba tlačítko A), tak se křeček probudí, bude mít radost (veselý smajlík) a udělá radostný zvuk. A když s ním zatřeseme, nebude mít radost, a patřičně to zvukově dokreslí.

Většinu potřebných funkcí už známe – použijeme blok „při startu“, v němž zobrazíme spícího křečka, a pak dva bloky, které reagují na události: „při zatřesení“ a „on logo pressed“ (při stisknutí loga – s microbitem V1 použijte např. „po stisknutí tlačítka“).

V bloku obsluhy událostí nejprve zobrazte ikonu veselého nebo smutného obličeje. Na konci zobrazte zase spícího křečka.

Můžete si teď vyzkoušet funkčnost aplikace. Měla by dělat to, co od ní vyžadujeme, jen nebude zatím vydávat zvuk.

Otevřete nabídku Hudba. Najdete v ní sekce Melodie, Tón, Hlasitost, Tempo, Melodie-rozšíření a microbit (V2). V té poslední sekci najděte blok „play sound (giggle) until done“ (v době psaní knihy ještě nebylo přeloženo).

Tento blok vložte do bloku „při zatřesení“ mezi zobrazení emotikonu a spícího křečka. Po umístění změňte „giggle“ na „mysterious“.

Vezměte znovu nový blok „play sound“ a přeneste ho do obsluhy události „stisknutí loga“, opět mezi zobrazení ikony šťastného křečka a spícího křečka. Po umístění změňte zvuk „giggle“ na zvuk „hello“.

Přeložte, nahrajte do microbitu a vyzkoušejte si, jak váš nový digitální křeček reaguje. Na zatřesení vydává nepříjemné zvuky, při pohlazení dotykového loga má radost.

- *Zkuste si změnit zvuky, jaké křeček vydává při jednotlivých podnětech.*
- *Doplňte funkci, která při zapnutí microbitu zahraje krátkou znělku. Nápověda: Použijte funkci „hraj melodii“ a zvolte melodii „zapnutí“.*

```
JS  input.onGesture(Gesture.Shake, function () {
JS      basic.showIcon(IconNames.Sad)
JS      soundExpression.mysterious.playUntilDone()
JS      basic.showIcon(IconNames.Asleep)
JS  })
JS  input.onLogoEvent(TouchButtonEvent.Pressed, function () {
JS      basic.showIcon(IconNames.Happy)
JS      soundExpression.hello.playUntilDone()
JS      basic.showIcon(IconNames.Asleep)
JS  })
JS  music.startMelody(music.builtInMelody(Melodies.PowerUp), MelodyOptions.Once)
JS  basic.showIcon(IconNames.Asleep)
JS  basic.forever(function () {
JS
JS  })
```

```
    def on_gesture_shake():
        basic.show_icon(IconNames.SAD)
        soundExpression.mysterious.play_until_done()
        basic.show_icon(IconNames.ASLEEP)
    input.on_gesture(Gesture.SHAKE, on_gesture_shake)

    def on_logo_pressed():
        basic.show_icon(IconNames.HAPPY)
        soundExpression.hello.play_until_done()
        basic.show_icon(IconNames.ASLEEP)
    input.on_logo_event(TouchButtonEvent.PRESSED, on_logo_pressed)
```

```
music.start_melody(music.built_in_melody(Melodies.POWER_UP), MelodyOptions.ONCE)
basic.show_icon(IconNames.ASLEEP)

def on_forever():
    pass
basic.forever(on_forever)
```

Metodický list

Co si připravit?

- Microbit, kabel micro-USB, propojení s PC
- Ověřit, zda PC s microbitem komunikuje
- Spárovat přes WebUSB

Co jsme se naučili?

- Přednastavené zvukové efekty

Úkoly a výzvy

- Další interakce s křečkem – může reagovat na hluk, na světlo atd.

2.7 Kreslení 2

Vraťme se ještě na chvíli ke kreslení. Skončili jsme s tím, že umíme rozsvítit bod, zhasnout bod, nebo přepnout jeho stav. Pojďme udělat něco většího…

Zkusme nakreslit čáru! A začněme třeba vodorovnou, hezky uprostřed.

Vodorovná čára, to je pět bodů vedle sebe. Pokud má být uprostřed, tak je to řádek číslo 2. Musíme tedy rozsvítit tyto LEDky:

```
[0,2] [1,2] [2,2] [3,2] [4,2]
```

Takže použijeme pět příkazů „rozsvit".

Je to nejjednodušší a velmi přímočaré řešení. Prostě je jeden po druhém rozsvítíme.

Máte to? Skvěle!

A teď něco jiného. Namalujte dvě vodorovné čáry, jednu nahoře (řádek 0), jednu dole (řádek 4).

Jak jste na to šli?

Způsobů je několik. Můžete přidat dalších pět příkazů „rozsvit", přepsat souřadnice, a zadání je splněno. I když poněkud nepřehledně.

V programování platí základní pravidlo: Dělejte věci co nejjednodušší a co nejkratší. Nejde o stroj, tomu je jedno, jak to bude dělat, jde o nás, programátory. Věci by měly být přehledné a jasné, abychom věděli druhý den, co náš vlastní kód dělá.

U kreslení čáry je důležité si uvědomit, že probíhá pět velmi podobných akcí. Akce „rozsvit", při které se mění jen jedna hodnota, totiž X, a druhá zůstává stále stejná.

Pokaždé, když se má něco opakovat, buď úplně stejně, nebo podobně, tak programátoři použijí konstrukci, které se říká **cyklus**.

Cyklus

Cyklů je několik druhů. My teď použijeme cyklus s řídící proměnnou a daným počtem opakování. Což zní velmi učeně, ale všichni programátoři ho beztak znají pod krycím názvem **cyklus FOR**.

V běžných programovacích jazycích má tvar např. takovýto:

JS
```
for (let x = 0; x <= 4; x++) {
        led.plot(x, 2)
    }
```

```
for x in range(5):
        led.plot (x, 2)
```

V blokovém editoru jej najdete v nabídce „Smyčky". Je přeložen jako „pro (pořadí) od 0 do (N)" – tedy doslovný překlad z anglického FOR x FROM 0 TO N…

Co tento cyklus (smyčka) dělá?

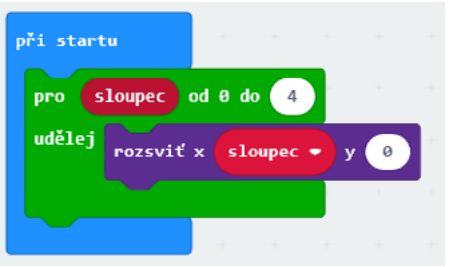
1. Nejprve nastaví řídící proměnnou na hodnotu 0.
2. Zkontroluje, jestli řídicí proměnná není vyšší než hodnota N („horní mez“)
3. Pokud má vyšší hodnotu než N, cyklus se neprovede a pokračuje se v provádění příkazů za ním.
4. Pokud je hodnota řídicí proměnné nižší nebo rovna N, pokračuje se následujícím krokem.
5. Provede příkazy, které jsou do něj vnořené („tělo cyklu“).
6. Po vykonání všech příkazů v těle cyklu zvýší hodnotu řídicí proměnné o 1 a vrátí se na krok 2

Vložte si blok „pro“ do programu a do něj vložte příkaz „rozsvit“. Řídící proměnnou („pořadí“) si pravým tlačítkem myši přejmenujte – zde se nabízí například „sloupec“. (Pamatujete na radu *pojmenovávejte si proměnné tak, aby bylo jasné, co v nich je*?)

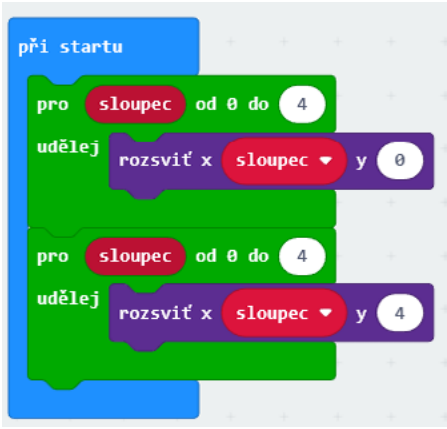
I poměrně striktní seznamy doporučení pro psaní programů většinou dovolují řídící proměnnou cyklu pojmenovat „i“ (případné vnořené pak „j“ a „k“), protože to tak dělá naprostá většina programátorů, kteří jsou za svou dobu praxe zvyklí, že vídají „FOR i“, a je to taková nepsaná dohoda, že to „i“ nic neznamená, že to je jen počítadlo průchodů cyklem.

V našem konkrétním případě by teoreticky šlo použít i „x“, protože proměnná opravdu obsahuje souřadnici X pro rozsvěcený bod.

V příkazu „rozsvit“ nastavte hodnotu X na proměnnou sloupec (najdete ji buď v bloku Proměnné, nebo ji můžete přetáhnout z bloku „pro…“), hodnotu Y na 0 (tedy horní řádek).

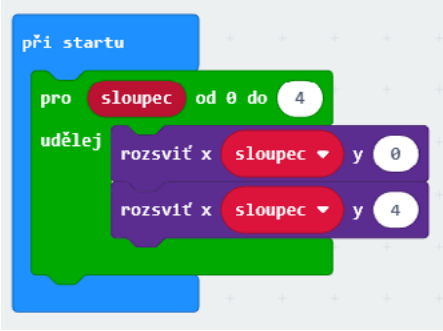


Když se tento kód spustí, namaluje vodorovnou čáru nahoře. A my chceme ještě jednu dole:



Je to jasné a přehledné. Tedy – pro nás, protože víme, co jsme dělali a proč.

Můžeme kód trochu zkrátit a oba příkazy „rozsviť" provést v jednom cyklu? Samozřejmě můžeme, a výsledek bude stejný:



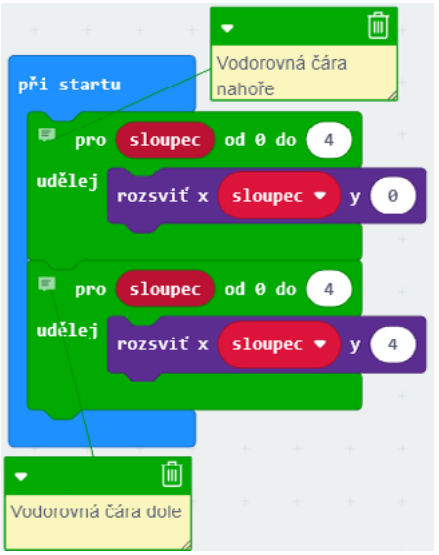
V těle cyklu může být libovolný počet dalších příkazů. Dokonce i dalších cyklů, jak si ukážeme za chvíli. Ale teď se ještě na chvilku vraťme k předchozímu tvaru.

Vidíme, že jsou si podobné (jediný rozdíl je 0-4). Druhé, co vidíme, je to, že na první pohled nemusí být jasné, co se děje. *Probíhá cyklus, kreslí to bod, aha, aha, to asi bude nějaká čára, asi teda vodorovná…*

Dobrý programátor proto u kódu, který nemusí být zcela zřejmý, použije takzvaný *komentář*.

Komentář

Klikněte na blok pravým tlačítkem a zvolte možnost Vložit komentář. Objeví se malá ikonka poznámky, a když na ni kliknete, ukáže se blok s komentářem. A tam můžeme napsat, co daný blok dělá. Ne jak to dělá, to programátor vidí, ale co, popřípadě proč.



Ale komentář je jen jeden způsob zpřehledňování programů. Druhý, mnohem důležitější princip, se jmenuje DRY.

Funkce

DRY je akronym (zkratka) z anglického Don't Repeat Yourself, tedy „Neopakuj se!"

Už jsme tento princip jednou použili, totiž když jsme zavrhli opakované přidávání dalších a dalších bloků „rozsviť", a místo toho jsme použili cyklus.

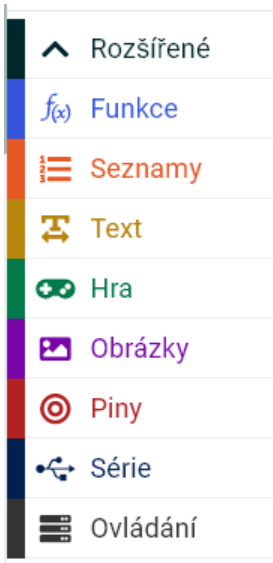
Teď máme v programu dva cykly, které jsou naprosto stejné, až na to, že jeden kreslí na řádek 0, druhý na řádek 4. Hodilo by se, kdybychom ten cyklus měli někde jen jednou a akorát bychom mu říkali, na který řádek má kreslit.

Téhle programátorské konstrukci se říká různě: podprogram, procedura, metoda, nebo třeba funkce. Jsou mezi tím drobné významové nuance, ale těmi se nemusíme teď vůbec zabývat. Budeme používat slovo „funkce“.

Funkce v programování nemá tentýž význam, jako třeba v matematice. Pojetí funkce je mnohem širší. Programátorská funkce je blok příkazů, který je nějak pojmenován, a kterému můžeme (nebo nemusíme) při jejím provádění („volání funkce“) předat nějaké hodnoty („parametry funkce“). Funkce po provedení bloku příkazů vrátí nějakou hodnotu („návratová hodnota“) tomu, kdo ji volal.

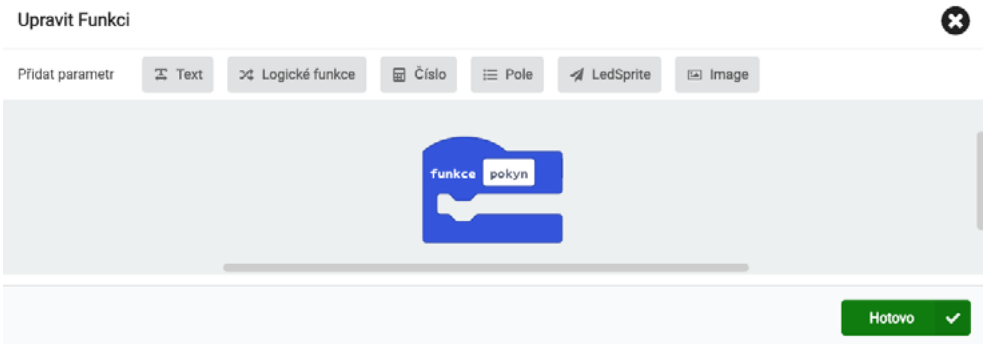
Příkladem funkce by byla například funkce „mocnina“, která by měla jeden parametr (x) a návratovou hodnotou by byl výsledek součinu (x*x). V programování se dnes běžně říká *funkce* i takovým případům, kdy se nevrací žádná hodnota (v jazyce Pascal se jim říkalo „procedury“). A jednu takovou si napíšeme.

Rozbalte si záložku „Rozšířené“ v menu:



Hned první nabídka je to, co hledáme. Podobně jako u proměnných tu vytvoříme funkci, která bude kreslit vodorovnou čáru. Klikněte tedy na Vytvořit funkci.

Objeví se editor funkcí:

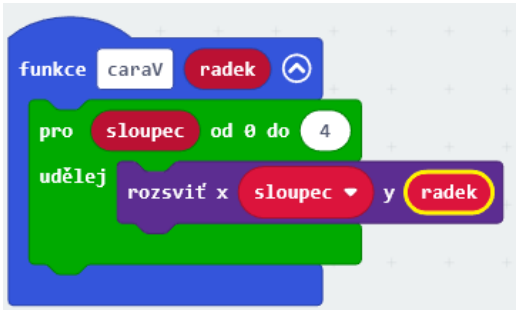


Nejprve přepište její jméno. Slovo „pokyn“ přepište na „caraV“ – jako že čára vodorovná.

Této funkci budeme předávat jeden parametr, totiž řádek, na kterém má tu čáru nakreslit. Proto klikněte v řádku „Přidat parametr“ na možnost „číslo“.

Objeví se prostor pro parametr, nám známý oblý blok pro zadání čísla. Místo slova „číslo“ napište „radek“ – tedy „řádek“, ale diakritiku v názvech proměnných je lépe nepoužívat.

A teď klikněte na „Hotovo“. Na ploše se objeví nová prázdná funkce. Přetáhněte do ní jeden z cyklů, třeba ten pro dolní čáru. Jako hodnotu Y v příkazu „rozsviť“ nastavte hodnotu parametru radek (tedy přetáhněte z hlavičky funkce prvek „radek“ na místo hodnoty Y). Výsledek by měl vypadat takto:



A to je celé. Máme funkci, která nakreslí vodorovnou čáru na zadaném řádku. Pojďme si ji zkusit!

V nabídce Funkce se objevil nový blok – „vykonej caraV“. Dejme dva tyto bloky do sekce „při startu“, jednomu nastavte parametr radek na 0, druhému na 4 a podívejte se na výsledek.

Když přijde nějaký jiný programátor po nás, tak koukne a vidí: „Aha, tady se namalují dvě vodorovné čáry, na řádku 0 a na řádku 4“. Pro jistotu ještě můžete přidat k funkci „caraV“ komentář, že se jedná o kreslení vodorovné čáry, a je to!

Následující úkol pro vás bude určitě naprosto triviální:

- *Zaplňte celou obrazovku rozsvícenými body.*

Ale trochu si to upřesníme a rozepíšeme:

- *Použijte k zaplnění obrazovky funkci pro kreslení vodorovné čáry*
- *Vytvořte funkci pro kreslení svislé čáry a zaplňte obrazovku pomocí těchto čar.*
- *Použijte funkci pro kreslení vodorovné a svislé čáry a namalujte uprostřed displeje kříž.*

U posledního úkolu si vzpomeňte na letmou zmínku o tom, co všechno může být v cyklu…

- *Rozsviťte všechny LED na displeji, ale 1. nesmíte použít žádnou definovanou funkci a 2. smíte použít přesně jeden příkaz „rozsvit“.*

Metodický list

Co si připravit?

- Microbity, prostředí MakeCode, propojení s PC

Co jsme se naučili?

- Princip cyklu FOR (ve zjednodušené podobě)
- Komentování kódu
- Princip a základní použití funkcí

2.8 Odpočet

Zní to jednoduše: **udělejte z microbitu displej, který bude odpočítávat sekundy od 9 do 1, a pak zobrazí smajlík. Po stisknutí tlačítka A začne odpočítávání znovu.**

S tím, co už známe, to nebude problém. Uděláme si funkci, která bude zobrazovat odpočet. Zobrazí číslo 9, 8, 7, … počká sekundu, a když dojde k nule, tak místo nuly ukáže smajlík. Tuto funkci zavoláme z bloku „Při startu“ a z bloku „po stisknutí tlačítka A“.

Ano, bude to dělat to, co bylo v zadání. A pokud k tomu zobrazování čísel použijete smyčku, bude to naprosto ideální.

Pojďme se chvíli soustředit jen na to vypisování číslic 9 až 1. Jak je budeme vypisovat?

Smyčka PRO

Známe už smyčku PRO. Bohužel pro nás: počítá od 0 výš. Můžeme si ale pomoci matematikou.

Jak z čísel 0, 1, 2, 3, … získáme čísla 9, 8, 7, 6, …?

Potřebujeme vymyslet vzoreček, funkci, která z nuly udělá devítku, z jedničky osmičku… Když se nad tím zamyslíme, je to vlastně jednoduché odečítání: **9-X**

- 9 - 0 = 9
- 9 - 1 = 8
- 9 - 2 = 7
- …

Použijeme k tomu nabídku Matematika a vybereme odčítání:

Tím jsme dosáhli toho, že proběhne počítání od 9 do 1.

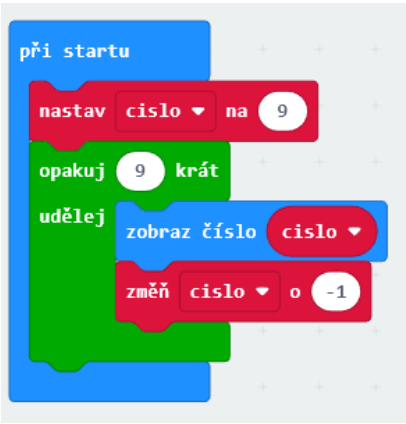
> *Proč je smyčka „od 0 do 8“ a ne do 9? Co se stane, když bude do devíti?*

Samotná smyčka počítá nahoru, ale zobrazují se čísla opačně.

Smyčka OPAKUJ

Místo smyčky PRO můžeme použít variantu smyčky s pevně daným opakováním. Vytvoříme si proměnnou, která bude udržovat aktuální hodnotu čísla. Na začátku ji nastavíme na 9, a pak zopakujeme 9x kroky

- Zobrazit číslo z proměnné
- Snížit hodnotu proměnné o 1



Smyčka DOKUD

Třetí smyčka, jakou můžeme použít, se jmenuje DOKUD. Dokud něco platí, tak smyčka běží, jakmile to neplatí, smyčka se ukončí a pokračuje se programem za ní.

Můžeme použít stejný postup jako u předchozí smyčky, tedy nastavit proměnnou na 9 a v těle zobrazit číslo a zmenšit proměnnou o 1. Ale tentokrát neřekneme, kolikrát se má smyčka provést, ale **do kdy se má provádět**. Má se provádět do té doby, dokud je proměnná větší než 0.

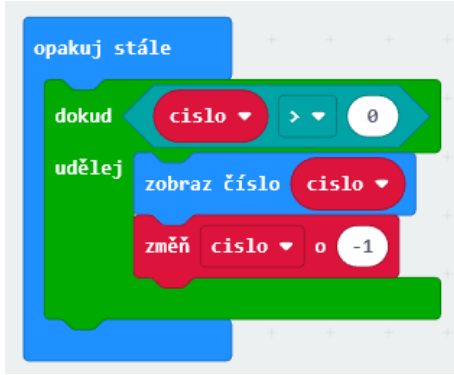
Všimněte si zvláštního tvaru bloku s podmínkou. Není ani hranatý, ani oválný – má tvar špičatého šestiúhelníku:

Tím nám editor naznačuje, že na tom místě nemůžeme použít ani číslo, ani příkaz (nepasují tam). Zde musíme použít takzvaný **logický výraz**.

Logické výrazy jsou takové, u kterých můžeme říct, jestli jsou pravda, nebo nepravda. Třeba: Je 9 > 11? – nepravda. Je 3 < 5? – pravda. Nebo: Je hodnota proměnné větší než 0?

Pro vytváření logických výrazů máme k dispozici nabídku, nazvanou Logika. Zde jsou dva bloky *podmínek*, k nimž se dostaneme za chvíli, pak nabídka porovnání, logických funkcí a speciálních hodnot „pravda“ a „nepravda“.

Použijeme blok porovnání a vybereme jeden z prvních dvou. Třetí slouží k porovnávání textů a znaků, první dva porovnávají čísla. Je jedno, který z těch prvních dvou vyberete, protože jsou oba stejné, jen mají jinou předvybranou podmínku. Vyberte jej, vložte na místo podmínky u smyčky „dokud“, a pak nastavte vlevo proměnnou, uprostřed symbol „>“ (je větší) a vpravo nechte nulu.



Smyčka bude probíhat, dokud je hodnota proměnné „číslo“ větší než nula. Poté skončí a probíhat nebude.

Celý program

Zkuste si tedy se znalostí těchto tří postupů zobrazit odpočet, na jeho konci ukázat smajlík a při stisknutí tlačítka A spustit celé znovu.

Možná si říkáte, že je jedno, jestli použijete OPAKUJ, nebo DOKUD. Zatím ano, ale teď si lehce upravíme zadání:

Tlačítkem A se znovuspustí odpočítávání, i když předchozí stále probíhá.

Tedy počítá se 9 – 8 – 7 – 6 … a vy v tu chvíli stisknete tlačítko A. Teď by měla na displeji zase naskočit devítka, osmička…

Se smyčkou OPAKUJ nastane drobný problém: je pevně nastaveno, kolikrát se musí zopakovat, a nemáte moc šancí ji spustit znovu.

Samozřejmě můžete použít příkaz „přeruš“ (neboli „break“ z programovacích jazyků Python nebo JS), testovat ve smyčce stav tlačítka a ošetřit opakované spuštění smyčky, ale není to v tomto případě moc elegantní řešení.

Se smyčkou DOKUD můžeme udělat jeden pěkný trik. Nechat ji běžet stále dokola. Pokud bude proměnná rovna nule, vnitřek smyčky se nebude provádět, a tak se nic nestane. Pokud bude stisknuté tlačítko, nastavíme proměnnou opět na hodnotu 9. Tím se při dalším průběhu smyčka znovu aktivuje (podmínka bude pravdivá).

Za samotnou smyčku pak vložíme jen zobrazení smajlíku, a úkol je splněn. Nebo ne?

Zkuste si to naprogramovat a nahrát do microbitu. Když odpočet doběhne do konce a stisknete tlačítko A, poběží hezky od devítky. Když ale tlačítko stisknete třeba někde u hodnoty 5, tak začne nový odpočet od osmičky. Jak to?

Problém je v časování. Když se podíváte na tělo smyčky, tak se dějí tři úkony:

- Porovnání
- Zobrazení čísla
- Změna proměnné

Porovnání i změna jsou velmi rychlé. Zobrazení čísla ale obsahuje cca půlsekundovou pauzu. Takže je velmi pravděpodobné, že se stisk tlačítka „trefí“ právě do této pauzy. Co se stane? Proměnná je nastavena na hodnotu 9, a pak se pokračuje za zobrazením čísla: zmenší se o 1 (tedy na 8), porovná se, zobrazí, čeká…

Jak to vyřešit?

Jedno z možných řešení je změnit program tak, že budeme počítat s tím, že se trefujeme právě do té pauzy. Prohodíme odčítání a zobrazení a nebudeme na začátku nastavovat číslo na 9, ale na 10. Podmínku ve smyčce musíme změnit na „cislo > 1“

Proč musíme změnit podmínku ve smyčce?

```
JS  let cislo = 9
JS  input.onButtonPressed(Button.A, function () {
JS      cislo = 10
JS  })
JS  basic.forever(function () {
JS      while (cislo > 1)  {
JS          basic.showNumber(cislo--) }
JS      basic.showIcon(IconNames.Happy)
JS  })
```

```
fcislo = 9
def on_button_pressed_a():
    global cislo; cislo = 10
input.on_button_pressed(Button.A, on_button_pressed_a)
def on_forever():
    global cislo
    while cislo > 1:
        basic.show_number(cislo--)
    basic.show_icon(IconNames.HAPPY)
basic.forever(on_forever)
```

Metodický list

Co si připravit?

- Microbity, prostředí MakeCode, propojení s PC

Co jsme se naučili?

- Princip cyklu FOR
- Princip cyklu s daným počtem opakování
- Princip cyklu s podmínkou
- Podmínky a logické výrazy

Odbočka pro pokročilé

Jenže toto řešení je polovičaté. Jen jsme využili pravděpodobnost toho, že se stisk tlačítka trefí právě do té pauzy po zobrazení čísla. Ale existuje určitá nenulová pravděpodobnost, že se *trefí* před zobrazení a odpočet začne desítkou. Chcete-li si tento problém nasimulovat, vložte mezi „změň číslo“ a „zobraz číslo“ blok čekání, třeba 300 milisekund. Pak začne odpočet někdy od 9, někdy od 10.

Tento problém nastává vždy, když do hry vstupují události, které nastávají asynchronně, tedy kdykoli, nejen tehdy, kdy programátor očekává, že nastanou a počítá s nimi. Pokud takové asynchronní události nějak ovlivňují výpočet, je potřeba zajistit, aby nevstoupily do děje v nevhodný okamžik, například mezi změnou hodnoty a zobrazením čísla…

Problém asynchronní události se řeší tak, že buď odstraníme asynchronní obsluhu události, nebo událost synchronizujeme. Ukážeme si oba postupy.

Synchronizace události spočívá v tom, že odstraníme obsluhu události (tedy celý blok „při stisknutí…"), a místo toho na bezpečné místo vložíme kontrolu: „Když je stisknuto tlačítko A, tak nastav proměnnou na 9".

Podmínky („když") jsme si zatím nepředstavili, to až v následující kapitole, ale pokud jste pokročilí, jistě o jejich existenci víte… Už jsme je letmo zahlédli. Byly v nabídce Logika.

Kde je bezpečné místo? No, jednak na konci cyklu (tam se obslouží znovustisknutí během odpočítávání), a pak na konci cyklu „opakuj stále" (tam se obslouží stisknutí po doběhnutí). A protože obě kontroly budou naprosto totožné, naprogramujeme je jako funkci (pamatujete se na princip „neopakuj se"?)

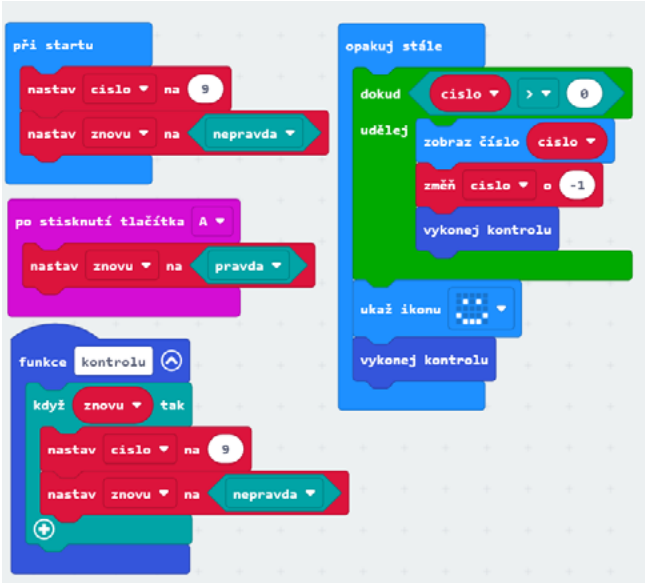
Problém, na který brzo přijdete, dělají krátká stisknutí. Pokud stisknete tlačítko jen na chvilku, tak se jeho stisknutí i puštění může odehrát během čekání. Takže následná kontrola stisk nezachytí. Proto použijeme druhou metodu:

Odstranění asynchronní obsluhy události spočívá v tom, že událost obsloužíme co nejrychleji, ale jen „jakoby". V našem případě si jen poznamenáme, že se má nastavit hodnota proměnné na 9, ale neuděláme to hned (nevíme, ve kterém místě je zrovna hlavní program). Místo toho se v hlavním programu podíváme, jestli je takový požadavek, a pokud ano, provedeme ho. Výhoda je, že můžeme zcela přesně určit, kdy je bezpečné se tomuto požadavku věnovat.

Ono se řekne – „poznamenáme si", ale jak a kam? No, do proměnné. Vytvoříme si proměnnou, kterou si nazveme „znovu", a využijeme toho, že do proměnné lze zapsat nejen číslo, ale i logickou hodnotu (pravda, nebo nepravda). Pokud bude tato proměnná obsahovat hodnotu pravda, tak na bezpečném místě nastavíme počítadlo opět na 9.

Nesmíme zapomenout nastavit „znovu" na nepravda, protože jinak by se neustále dokola opakovalo nastavování devítky! Stejně tak musíme proměnnou „znovu" nastavit na nepravdu při startu.

Tato verze už bude fungovat přesně tak, jak očekáváme.



Proč jsem funkci nepojmenoval „kontrola“, ale „kontrolu“? Správně podle konvencí bych ji měl pojmenovávat tak, jak je zvykem, tj. v prvním pádu (kontrola). Ale mně se líbí, když se ve smyčce její volání objevuje jako „vykonej kontrolu“, ne „vykonej kontrola“. Angličtina takové problémy nemá…

Zkuste si tuto poslední verzi přepsat tak, že kontrola bude volána jen na jednom místě. Napovím: budete muset využít toho, že blok „opakuj stále“ je sám o sobě smyčka.

```
JS  function kontrolu () {
JS      if (znovu) {
JS          cislo = 9
JS          znovu = false
JS      }
JS  }
JS  input.onButtonPressed(Button.A, function () {
JS      znovu = true
JS  })
```

```
JS  let znovu = false
JS  let cislo = 0
JS  cislo = 9
JS  znovu = false
JS  basic.forever(function () {
JS      while (cislo > 0) {
JS          basic.showNumber(cislo)
JS          cislo += -1
JS          kontrolu()
JS      }
JS      basic.showIcon(IconNames.Happy)
JS      kontrolu()
JS  })

    def kontrolu():
        global cislo, znovu
        if znovu:
            cislo = 9
            znovu = False

    def on_button_pressed_a():
        global znovu
        znovu = True
    input.on_button_pressed(Button.A, on_button_pressed_a)
    znovu = False
    cislo = 0
    cislo = 9
    znovu = False

    def on_forever():
        global cislo
        while cislo > 0:
            basic.show_number(cislo)
            cislo += -1
            kontrolu()
        basic.show_icon(IconNames.HAPPY)
        kontrolu()
    basic.forever(on_forever)
```

Metodický list

Co si připravit?

- Microbity, prostředí MakeCode, propojení s PC

Co jsme se naučili?

- Problém synchronizace událostí
- „Race condition“ chyby

2.9 Hod mincí

Je spousta situací, kdy se člověk má rozhodnout, ale nemá dostatek informací, takže nechá rozhodnutí na náhodě. Nejjednodušší způsob, jak takové náhodné rozhodnutí udělat, je hodit si mincí.

Mám jít lesem, nebo přes louku? Obě cesty jsou stejně dlouhé a je mi jedno, kudy půjdu, tak co teď? Vytáhnu minci, hodím s ní, a když padne panna, půjdu lesem, když orel, tak loukou.

Pojďme si udělat takovou „elektronickou minci“ s microbitem. V klidu bude zobrazovat otazník, a když s ním zatřeseme, tak na chvíli ukáže buď veselý, nebo smutný smajlík.

Při startu zobrazíme otazník. Funkce „Ukaž ikonu“ nám nepomůže – musíme použít funkci „ukaž tvar“ a nakreslit si otazník sami. Třeba takto:

Teď je potřeba zareagovat na to zatřesení. Je jasné, že jako základ použijeme blok „při zatřesení“ z nabídky Vstup. Ale co do něj?

V předchozí kapitole jsme si u smyčky DOKUD říkali o **podmínkách**. Podmínky jsou základním prvkem při rozhodování, jak se má program zachovat. Podmínka je buď splněná, a pak se zachová jedním způsobem, nebo splněná není, a pak se zachová jiným způsobem.

Podmínky používáme v normálním životě naprosto běžně. Říkáme si: „Když bude venku pršet, pojedu do školy tramvají, jinak půjdu pěšky“. Podmínka je právě to „bude pršet“. Pokud bude pravdivá, tak udělám něco (pojedu tramvají), pokud bude nepravdivá, udělám něco jiného (půjdu pěšky).

Microbit má úplně stejný mechanismus rozhodování. Najdete ho v nabídce Logika, a jmenuje se přesně tak, jak jsme si právě popsali: **KDYŽ něco, TAK …, JINAK …**

V nabídce je i zkrácený tvar, kde je pouze **KDYŽ něco, TAK …** Můžeme si domyslet ono známé „jinak nic…“ – a přesně tak to je.

„Když budeš mít ještě hlad, tak si přidej!“ je příklad takového podmíněného rozhodování. Když je podmínka (mám hlad) splněná, tak něco udělám (přidám si). Když není, tak nic nedělám.

V angličtině se tyto podmínkové konstrukce označují jako IF-THEN-ELSE a IF-THEN.

My použijeme do našeho „mincového“ programu podmínku úplnou, tedy KDYŽ-TAK-JINAK, a budeme se rozhodovat náhodně.

Na náhodné rozhodování naštěstí existuje přímo samostatný blok – v nabídce Matematika je položka „náhodně pravda-nepravda“. Kromě této položky je tu možnost i „náhodné číslo od-do“, ale tu nebudeme potřebovat, nám stačí jen pravda/nepravda. Panna, nebo orel. Smajlík veselý, nebo smutný.

V bloku „při zatřesení“ tedy bude náš podmínkový blok. Popišme si ho slovy:

KDYŽ <náhodně pravda/nepravda>, TAK zobraz smajlík, JINAK zobraz smutník.

Pak chvilku počkáme, a nakonec samozřejmě zase znovu ukážeme otazník.

Dokážete program z takového popisu už sestavit sami?

- *Zmínil jsem se o funkci, která generuje náhodné číslo od-do. Upravte náš Hod mincí na hrací kostku. Funkce bude následující:*
- *Při zatřesení ukáže na sekundu ikonu šachovnice, pak vygeneruje náhodné číslo od 1 do 6 a zobrazí ho na displeji.*
- *Máte hrací kostku? Upravte ji tak, že vygenerované číslo neukáže na displeji jako číslici, ale tak, jak je zvykem na hrací kostce, tedy počtem teček.*
- *Úloha pro pokročilejší: Použili jste u kostky funkci „ukaž tvar"? Zkuste použít funkci „rozsvit" a využít toho, že tečky se nerozsvěcí náhodně, ale podle určitých pravidel: prostřední pouze u lichých čísel, dvě rohové u všech čísel větších než 1, protilehlé rohy pouze u čísel větších než 3, poslední dvě pak pouze u šestky.*

```
JS  let hodnota = 0
JS  input.onGesture(Gesture.Shake, function () {
JS      basic.showIcon(IconNames.Chessboard)
JS      basic.pause(1000)
JS      hodnota = randint(1, 6)
JS      basic.clearScreen()
JS      if (hodnota % 2 == 1) {
JS          led.plot(2, 2)
JS      }
JS      if (hodnota > 1) {
JS          led.plot(0, 0)
JS          led.plot(4, 4)
JS      }
JS      if (hodnota > 3) {
JS          led.plot(4, 0)
JS          led.plot(0, 4)
JS      }
JS      if (hodnota == 6) {
JS          led.plot(0, 2)
JS          led.plot(4, 2)
JS      }
JS  })
JS  basic.forever(function () {
JS
JS  })
```

```
hodnota = 0

def on_gesture_shake():
    global hodnota
    basic.show_icon(IconNames.CHESSBOARD)
    basic.pause(1000)
    hodnota = randint(1, 6)
    basic.clear_screen()
    if hodnota % 2 == 1:
        led.plot(2, 2)
    if hodnota > 1:
        led.plot(0, 0)
        led.plot(4, 4)
    if hodnota > 3:
        led.plot(4, 0)
        led.plot(0, 4)
    if hodnota == 6:
        led.plot(0, 2)
        led.plot(4, 2)
input.on_gesture(Gesture.SHAKE, on_gesture_shake)

def on_forever():
    pass
basic.forever(on_forever)
```

Metodický list

Co si připravit?

- Microbity, prostředí MakeCode, propojení s PC

Co jsme se naučili?

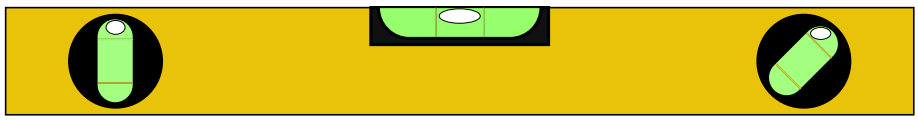
- Další použití podmínek, KDYŽ a JINAK, řetězení podmínek

2.10 Vodováha

V tomto experimentu využijeme faktu, že microbit má zabudovaný akcelerometr, a vytvoříme si z něj jednoduchou vodováhu.

Akcelerometr je součástka, která měří zrychlení. Nejčastěji se používají tříosé akcelerometry, které měří zrychlení ve všech směrech, tj. dopředu/dozadu, doleva/doprava a nahoru/dolů. Pokud součástka leží v klidu, tak měří zrychlení rovné gravitačnímu zrychlení, tedy 1 g směrem dolů. Je nutné na to pamatovat! Pokud součástka měří ve všech třech směrech 0, znamená to, že právě padá volným pádem (anebo je ve stavu beztíže, což je technicky totéž).

Hlavní součást vodováhy je skleněná trubička, mírně zakřivená, ve které je kapalina (většinou obarvená, aby byla lépe vidět), a v ní malá vzduchová bublina. Když je vodováha přesně rovně, je bublina uprostřed. Pokud ji mírně nakloníme, posune se bublina vpravo nebo vlevo, podle toho, která strana vodováhy je výš.



My bublinu nahradíme svítícím bodem na displeji. Ten budeme posouvat podle toho, jaký náklon nám ukáže akcelerometr. Ale než začneme programovat, musíme si zjistit, co vlastně akcelerometr změří. Naprogramujeme si proto takový malý tester, který bude jen číst údaje a zobrazovat je.

Test snímačů

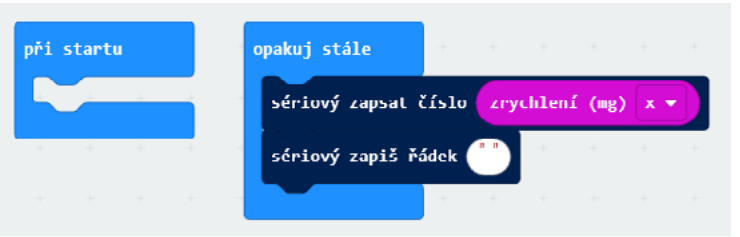
Protože vůbec netušíme, jaké údaje z akcelerometru vystupují, tak nemůžeme použít displej microbitu, bylo by to velmi nepraktické, kdyby třeba posílal dlouhá čísla. Proto si naměřené údaje pošleme do počítače, kde čísla uvidíme mnohem líp.

Samozřejmě se můžeme podívat do dokumentace, ale víte, jak to s dokumentací chodí: člověk něco vyčte, myslí si, že to pochopil, a pak ve skutečnosti… Je lepší si tyto věci ověřit právě podobnými testovacími programy.

Použijeme k tomu funkci, skrytou v nabídce Rozšířené, sekce Série.

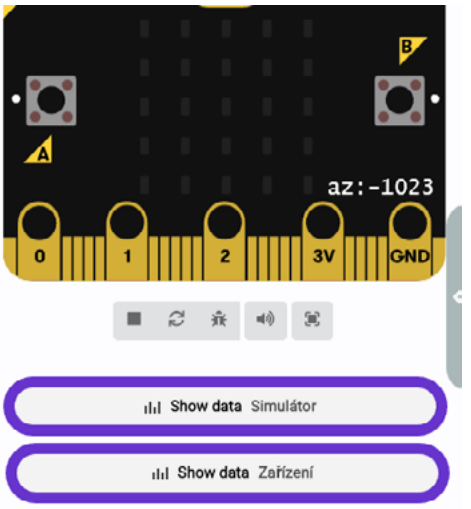
Český překlad, který nabízí prostředí MakeCode, je v době psaní knihy poněkud kostrbatý. Anglické „serial" by bylo lépe přeložit jako „sériová komunikace" a bloky Serial Write Line a Serial Write Number nějak jinak než „Sériový zapiš řádek" a „sériový zapsat číslo". Doufejme, že v době, kdy tuto knihu budete držet v rukou, bude už překlad lepší a konzistentní.

Sériová komunikace je jeden ze způsobů, jakým může microbit komunikovat s počítačem. Používá k tomu USB kabel, kterým je připojen při programování. Uděláme si jednoduchý projekt, který bude dělat stále dokola jeden úkol: bude číst údaj z akcelerometru a posílat ho do počítače.



Údaj z akcelerometru (zrychlení) najdete v nabídce Vstup, funkční bloky pro zápis na sériový port v nabídce „Série". Použijeme zápis čísla pro zrychlení a po něm zápis prázdného řádku, abychom mohli čísla líp číst.

Nahrajte program do microbitu a všimněte si, že se v levé části objevily dvě tlačítka s animovanou ikonou: Ukázat data ze simulátoru a Ukázat data ze zařízení. Použijte možnost Ukázat data ze zařízení.



Všimněte si, že v pravé části se nahoře kreslí graf a dole vypisují čísla.

Když microbit nakloníte, uvidíte, jak se čísla mění. Všimněte si několika věcí:

1. Čísla jsou v rozmezí zhruba od -1030 do +1030. Měří se v tisícinách normálního gravitačního zrychlení (mili-g).

2. Pokud microbit leží „na zádech“, je hodnota zhruba nulová. Při náklonu doleva jsou hodnoty záporné, při náklonu doprava kladné.

3. Naklánění dopředu ani dozadu naměřenou hodnotu moc neovlivňuje (jen chyby z nepřesného naklánění).

4. Když microbit nenakláníme, ale prudce ho otočíme, nebo s ním švihneme ze strany na stranu, čísla vyskočí třeba až na 2000 (nebo -2000).

Všechno to souvisí s tím, že jsme zjišťovali zrychlení v ose X (levá/pravá). Zrychlení v této ose je v klidu nulové. Pokud microbit nakloníme doleva nebo doprava, znamená to, že přitažlivá síla Země působí na levou nebo pravou stranu a vyvolá zrychlení o hodnotě zhruba 1 g (tedy 1000 mili-g). Pokud postavíme microbit na horní hranu, přesune se tíhové zrychlení v ose Y, takže v ose X by mělo zůstat stále nulové.

Jen pro jistotu: normální tíhové zrychlení g je rovno 9,81 m/s2.

No a konečně při prudkém pohybu doleva nebo doprava překonává zrychlení hodnotu 2 g, a proto se ukazují čísla přes 2000.

Neznamená to, že jsme zrychlili právě a přesně zrychlením 19,62 m/s², ale je to proto, že akcelerometr je nastaven na maximální citlivost 2 g. Když půjdete do nabídky Vstup a kliknete na podnabídku „…více", najdete zde funkční blok „nastav rozsah akcelerometru". Použijte ho v bloku Při startu, nastavte rozsah 8 g a zkuste si znovu švihnout.

Pokud jste opravdu rychlí, snadno naměříte maximální hodnotu přes 8000.

Je důležité si znovu připomenout, že akcelerometr neměří rychlost, ale *zrychlení*. Tedy nikoli to, jak rychle se microbit pohybuje, ale *jak rychle se mění jeho rychlost*. Proto při rychlém cuknutí vidíte v grafu dva vrcholy: jeden v jednom směru, to když microbit přecházel z klidu do pohybu, a druhý v opačném směru, když se pohyb zpomaloval. Pokud dokážete zrychlit, pak chvíli udržet stejnou rychlost a pak zpomalit, uvidíte v grafu dva zřetelně oddělené vrcholy. (Spíš se vám to nepodaří, nebo si jako já vytrhnete v zápalu švihání USB kabel z počítače...)

Spíš si zkuste položit microbit na stůl, nechat ho jen tak ležet, a pak opodál lehce bouchnout do stolu. Uvidíte, jak je akcelerometr citlivý i na malé otřesy.

- *Zkuste si ověřit, jak akcelerometr měří v osách Y a Z. Která je předo-zadní a která svislá?*
- *Prozkoumejte poslední možnost (síla). Co dělá, co měří?*
- *Zkuste pokusně ověřit, jestli je gravitační zrychlení při volném pádu opravdu nulové. Ale buďte opatrní a pouštějte microbit z malé výšky a vždy raději na něco měkkého.*

JS
```
input.setAccelerometerRange(AcceleratorRange.OneG)
basic.forever(function on_forever() {
    serial.writeNumber(input.acceleration(Dimension.Strength))
    serial.writeLine("")
})
```

```
input.set_accelerometer_range(AcceleratorRange.ONE_G)

def on_forever():
    serial.write_number(input.acceleration(Dimension.STRENGTH))
    serial.write_line("")
basic.forever(on_forever)
```

Metodický list

Co si připravit?

- Microbity, prostředí MakeCode, propojení s PC

Co jsme se naučili?

- Testování snímačů
- Ověření funkce
- Základní sériovou komunikaci

Aplikace

Podobný test snímačů doporučuju udělat pokaždé, než nějaký senzor začnete používat. Díky tomu zjistíte, jak s ním pracovat, jaké údaje měří, kdy je měří, v jakém rozsahu jsou naměřená čísla atd.

Díky testu teď víme, že na vodováhu můžeme použít osu X, tedy pravo-levou, že v klidu bude hodnota zhruba 0 a že při náklonu doleva jsou čísla záporná, doprava kladná.

Budeme tedy číst hodnotu akcelerometru v ose X. Když bude 0, zobrazíme bod uprostřed, tedy na pozici [2,2], když bude hodnota záporná, zobrazíme ho vpravo (protože microbit je nakloněn doleva, tak bude bublinka vpravo), když kladná, tak vlevo.

Nejprve si vytvořte proměnnou, řekněme „naklon“ – tam si uložíme naměřenou hodnotu. Tu pak budeme různě testovat, tak abychom ji neměřili pořád dokola.

Začneme tedy přiřazením zrychlení v ose X do této proměnné. To je jednoduché. A teď budeme zobrazovat bod podle pravidel, co jsme si řekli.

Bod bude vždy v prostředním řádku, tedy [?,2]. Hodnota sloupce by měla být:

- 0 pro naklon >> 0
- 1 pro naklon > 0
- 2 pro naklon = 0
- 3 pro naklon < 0
- 4 pro naklon << 0

Symboly >> a << nemyslím bitové posuny, jak je znáte možná z programovacích jazyků; používám je v „matematickém“ smyslu „výrazně větší, výrazně menší“.

V programovacích jazycích máme k dispozici jen „je větší“ a „je menší“, nic jako „výrazně větší“ nemáme. Musíme určit, co je to „výrazně“.

Víme, že microbit otočený na bok ukazoval čísla kolem tisíce (plus či mínus). Vodováha, která by ukazovala až takový náklon, je docela na nic, to vidíme prostým okem. Řekněme tedy, že „výrazně“ bude třeba sto. Tedy:

- 0 pro naklon > 100
- 1 pro naklon > 0
- 2 pro naklon = 0
- 3 pro naklon < 0
- 4 pro naklon < -100

Což zvládneme pomocí sady podmínek „když“. *Když naklon>100, tak rozsvíť [0,2]* a tak dále.

Pokud podmínky dáte za sebe, nebude to fungovat. Představte si, že hodnota bude 200 – taková hodnota splní jak podmínku *větší než 100*, tak podmínku *větší než nula*. Musíme je uspořádat tak, aby se taková věc nestávala.

Buď můžeme pět podmínek uspořádat tak, že každá další bude součástí bloku „jinak“ u té předchozí:

Což je poměrně nepraktické a nepřehledné, protože se brzo ztratíte v tom, co je vnořeno kam.

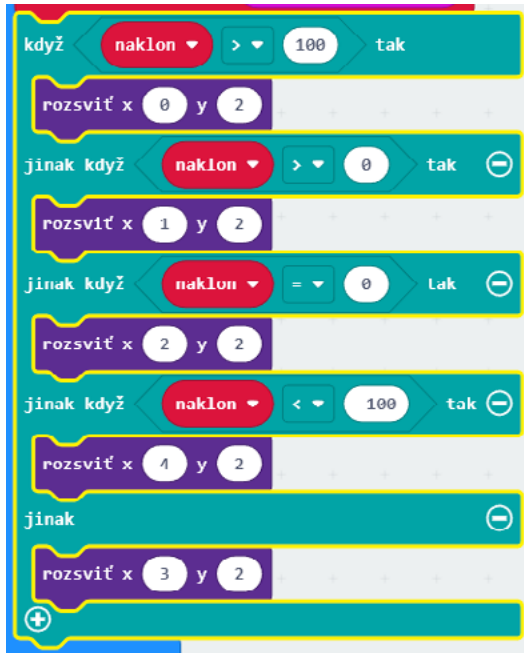
Jinak si všimněte dvou věcí:

1. testujeme nejprve popořadě >100, >0, =0 – ale pak testuji nejprve <-100. Kdybych otestoval <0, tak vyhoví každé záporné číslo. Proto nejdřív testuju ta „větší“.

2. Poslední podmínka chybí. Není potřeba, protože v tu chvíli už je jasné, že číslo je menší než nula (nebylo zachyceno dřív), a zároveň není menší než -100.

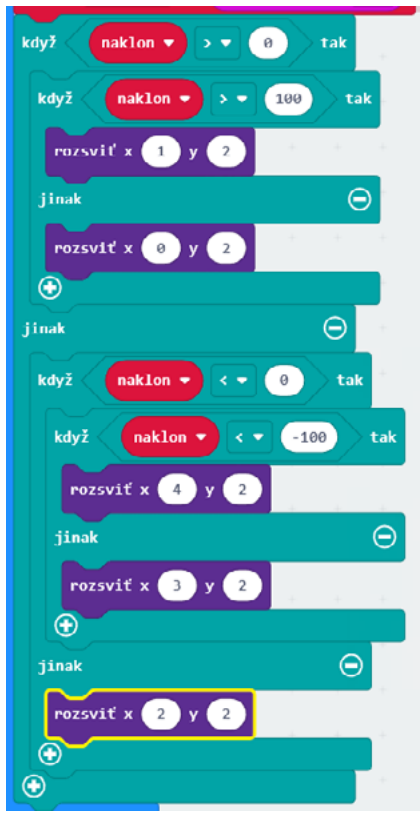
Proti nepěknému vnořování existuje jednoduchá zkratka, nazvaná „jinak když…“ (v programovacích jazycích „else if“).

Ukáže se při klikání na spodní „plus“ u bloku „když“. Po jednom kliknutí vznikne blok „jinak“, při dalším „jinak když“. Můžeme tedy podmínky přeuspořádat takto:



Vnořené bloky jsou hezky pod sebou a neodsouvají se moc doprava, zároveň je celá konstrukce kompaktnější a snáze pochopitelná.

Ještě si ukážeme jinou konstrukci podmínek, která naopak využije toho, že čísla >100 jsou zároveň >0 atd.



V našem případě se taková konstrukce moc nehodí, ale jsou situace, v nichž potřebujete např. výrazně jiným způsobem pracovat se zápornými čísly a s kladnými, a pak taková či podobná vznikne a má svůj smysl.

Pojďme ale otestovat funkci. Nahrajte program do microbitu a zkuste naklánět… Funguje?

```
JS  let naklon = 0
JS  basic.forever(function () {
JS      naklon = input.acceleration(Dimension.X)
JS      basic.clearScreen()
JS      if (naklon > 100) {
JS          led.plot(0, 2)
```

```
JS      } else if (naklon > 10) {
JS          led.plot(1, 2)
JS      } else if (naklon == 0) {
JS          led.plot(2, 2)
JS      } else if (naklon < -100) {
JS          led.plot(4, 2)
JS      } else {
JS          led.plot(3, 2)
JS      }
JS  })

    naklon = 0

    def on_forever():
        global naklon
        naklon = input.acceleration(Dimension.X)
        basic.clear_screen()
        if naklon > 100:
            led.plot(0, 2)
        elif naklon > 10:
            led.plot(1, 2)
        elif naklon == 0:
            led.plot(2, 2)
        elif naklon < -100:
            led.plot(4, 2)
        else:
            led.plot(3, 2)
    basic.forever(on_forever)
```

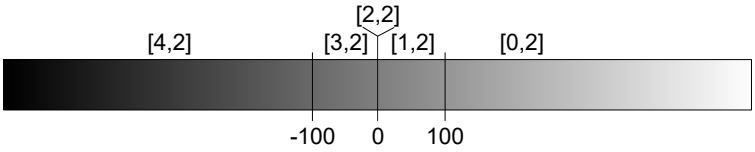
Aplikace, verze 2

Moc ne. Zapomínáme body zhasínat. Takže před zobrazení bodu vložíme blok „zhasni displej".

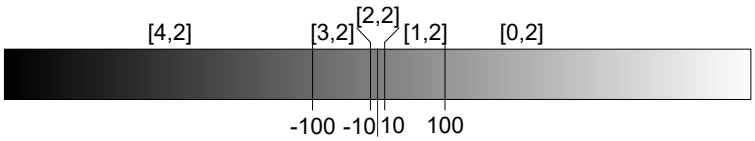
Nahrajte, naklánějte – bublina se hezky posouvá, ale jako by přeskakovala ten prostřední bod, že? Čím to je?

Hodnota náklonu se pohybuje od -1000 do +1000 (zhruba, ono je to víc). Pro hodnoty mezi -1000 a -101 je bublina na pozici [4,2]. Na pozici [3,2] je pro hodnoty v rozsahu -100 až -1. Na pozici [2,2] je jen a pouze ve chvíli, kdy je hodnota přesně 0… A to je ten problém. Na pozici vedle

středu je bublina pro sto hodnot, uprostřed jen pro jednu jedinou. A při citlivosti akcelerometru je téměř vyloučené tu nulu trefit.



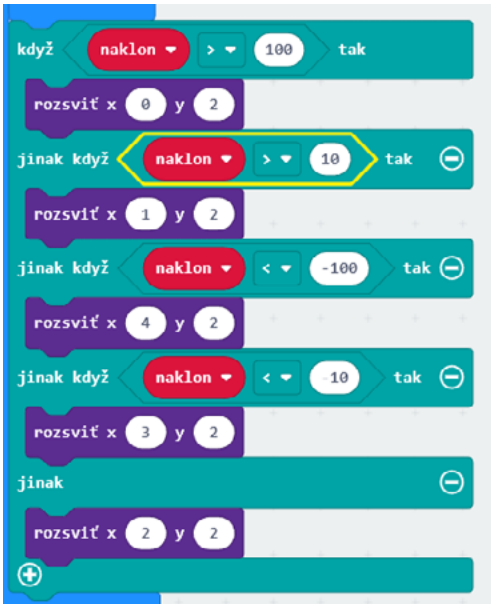
Musíme tedy upravit náš algoritmus a místo nuly dát nějaké rozmezí hodnot „okolo nuly“, kdy pro nás bude odchylka „dost malá“. Řekněme plus mínus 10. Výsledná škála se tak změní:



Na „trefení středu“ máme tak zhruba 20 hodnot. Zkuste si to naprogramovat.

Ale kde je podmínka „něco je v rozmezí mezi…“? No, není. Buď musíme použít konstrukci, které se říká „logický výraz“ a dokáže spojit víc podmínek dohromady, nebo tam, kde to jde, přeuspořádat podmínky jinak.

Tady to naštěstí jde, takže můžeme naši konstrukci „když… jinak když…“ přeuspořádat tak, že zkontrolujeme záporná čísla, pak kladná, a na konci zůstanou ty „okolo nuly“.



Nahrajte si a otestujte.

Logický výraz

Jak by to vypadalo, pokud bychom použili původní verzi, kde se porovnávalo s nulou? Museli bychom použít dvě podmínky: X je větší nebo rovno -10 **a zároveň** je X menší nebo rovno 10.

Logický výraz spojuje dva jiné logické výrazy (označme si je L1 a L2) do jednoho. Můžeme je spojit dvěma způsoby:

1. výsledek je *pravda* pouze tehdy, když *L1 a L2 jsou pravda*

2. výsledek je *pravda* pouze tehdy, když *L1 nebo L2 je pravda*

Prvnímu spojení („platí L1 A platí L2“) se říká logické násobení, nebo anglickým slůvkem AND. Druhé spojení („platí L1 nebo L2“) se říká logické sčítání, popřípadě anglicky OR. Pozor, v druhém případě nejde o „buď – anebo“, klidně může platit obojí.

Čeština používá slovo „nebo“ pro oba tyto způsoby. V běžné řeči to nepoznáme, v psaném jazyce se odlišují významy tím, zda je před „nebo“ čárka. Pokud není, jde o slučovací NEBO, pokud čárka je, jde o vylučovací NEBO.

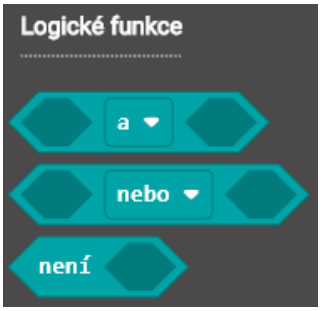
Dáš si kečup nebo hořčici? – Dám si obojí (slučovací)

Dáš si kečup, nebo hořčici? – Dám si kečup (jedna možnost vylučuje druhou)

Když je potřeba přesně vyjádřit, o jaké NEBO se jedná, používá se například tvar „a/nebo“ – tedy ve výše uvedeném příkladu by otázka zněla: „Dáš si kečup a/nebo hořčici?“

Druhému způsobu „nebo“, tomu, kde jedna možnost vylučuje druhou, se říká „exkluzivní NEBO“, anglicky „exclusive OR“, nebo zkráceně „XOR“.

Při programování microbitu máme na výběr z možností „A“ a „NEBO“ (kde jde o slučovací NEBO, tedy OR).

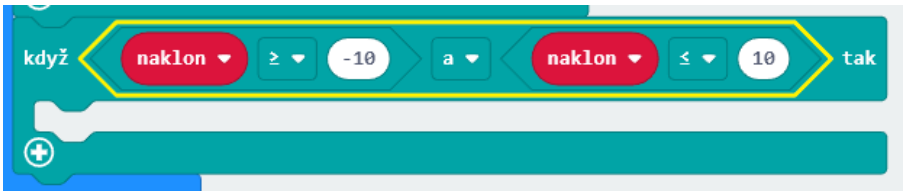


Poslední možnost otáčí výsledek. Je-li logický výraz „pravda“, jeho opakem je „nepravda“.

Výsledky ukazuje následující tabulka:

Výraz 1 (L1)	Výraz 2 (L2)	=>	L1 A L2	L1 NEBO L2	NENÍ L1
Nepravda	Nepravda		Nepravda	Nepravda	Pravda
Pravda	Nepravda		Nepravda	Pravda	Nepravda
Nepravda	Pravda		Nepravda	Pravda	Pravda
Pravda	Pravda		Pravda	Pravda	Nepravda

V našem případě tedy musíme nahradit podmínku „naklon = 0“ logickým výrazem:



Ten bude *pravda* pouze tehdy, když oba výrazy uvnitř budou taky *pravda*.

Metodický list

Co si připravit?

- Microbity, prostředí MakeCode, propojení s PC

Co jsme se naučili?

- Složitější logické výrazy

Aplikace, verze 3

S touto vodováhou jsem byl docela spokojen, až na jeden detail. Když jsem ji položil tak, aby ležela zcela vodorovně na stole, ukazovala drobnou výchylku doleva. Inu, mám křivý stůl.

Pomohlo by, kdybych zvětšil rozmezí „středu" na rozsah -20 až +20?

Ale přivedlo mě to na myšlenku: *co když chci, aby věci nebyly vodorovné vůči Zemi, ale vůči, řekněme, stolu?!*

Podobnou funkci mají kuchyňské váhy. Pokud potřebujete navážit například mouku, dáte si na kuchyňskou váhu nádobu a váze řeknete: Toto je nula. Je to pohodlnější, než si vážit nádobu a pamatovat si, že má 117 gramů a že čtvrt kila mouky bude tedy 367 gramů…

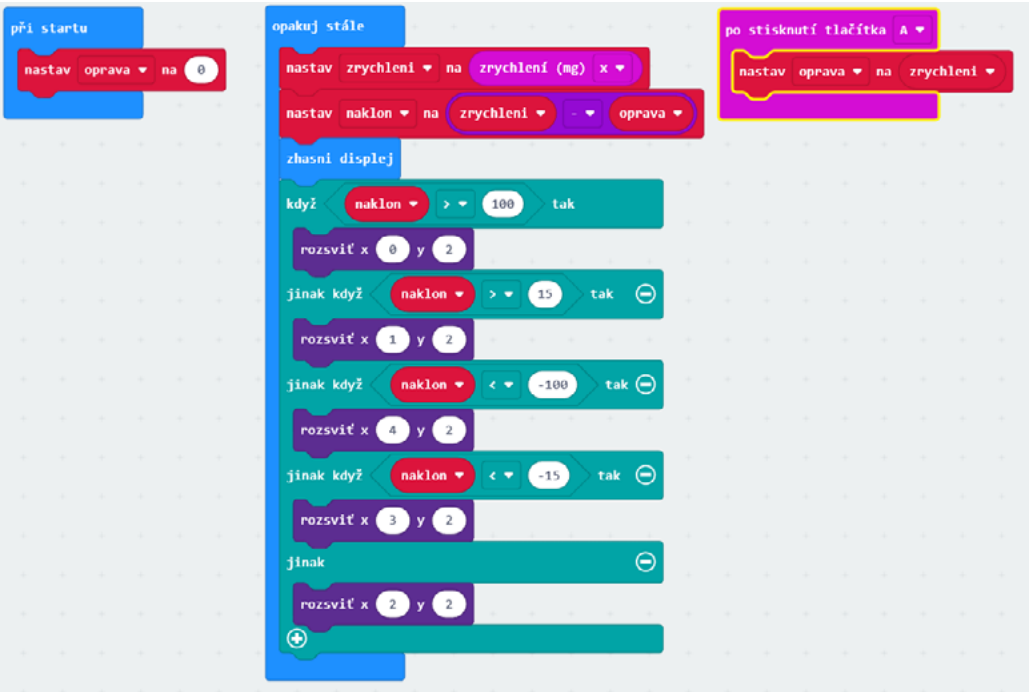
Takže jsem si zabudoval do vodováhy stejnou funkci. Jak jsem na to šel?

Požadovaná funkce: Chci, aby se po stisknutí tlačítka A nastavil aktuální náklon jako 0 a dokud jej nezměním, tak aby všechna další měření s touto opravou pracovala.

Řešení: Když stisknu tlačítko A, tak se aktuálně naměřený náklon uloží do proměnné „oprava". Při zobrazování bodu nebudu brát hodnotu „naklon", ale „naklon mínus oprava".

Praktický postup:

- Hodnota proměnné „naklon" se používá na čtyřech místech. Šlo by ji všude nahradit výpočtem „naklon-oprava", ale bylo by to jednak pracné, ale hlavně: neustále by se počítal výsledek naprosto stejného odčítání. A protože základní pravidlo programování zní „neopakuj se!", tak se nebudu opakovat a udělám to jinak. Naměřenou hodnotu si uložím do jiné proměnné, řekněme „zrychleni", a z ní si vypočítám náklon jako „zrychleni-oprava".
- V obsluze tlačítka A budu do proměnné oprava ukládat nikoli hodnotu proměnné „naklon", ale proměnné „zrychleni".
- Nesmím zapomenout vynulovat na počátku hodnotu proměnné oprava.



S touto verzí jsem už byl spokojenější. Přesto jsem ale ještě musel upravit rozmezí středu na hodnotu <-15;+15> - a pak už vodováha fungovala téměř k mé plné spokojenosti.

Jak upravit vodováhu, aby neměřila „vleže“, ale „vestoje“, tedy postavena na spodní hranu?

```
JS  input.onButtonPressed(Button.A, function () {
JS      oprava = zrychleni
JS  })
JS  let naklon = 0
JS  let zrychleni = 0
JS  let oprava = 0
JS  oprava = 0
JS  basic.forever(function () {
JS      zrychleni = input.acceleration(Dimension.X)
JS      naklon = zrychleni - oprava
```

```
JS      basic.clearScreen()
JS      if (naklon > 100) {
JS          led.plot(0, 2)
JS      } else if (naklon > 15) {
JS          led.plot(1, 2)
JS      } else if (naklon < -100) {
JS          led.plot(4, 2)
JS      } else if (naklon < -15) {
JS          led.plot(3, 2)
JS      } else {
JS          led.plot(2, 2)
JS      }
JS  })
```

```
    def on_button_pressed_a():
        global oprava
        oprava = zrychleni
    input.on_button_pressed(Button.A, on_button_pressed_a)

    naklon = 0
    zrychleni = 0
    oprava = 0
    oprava = 0

    def on_forever():
        global zrychleni, naklon
        zrychleni = input.acceleration(Dimension.X)
        naklon = zrychleni - oprava
        basic.clear_screen()
        if naklon > 100:
            led.plot(0, 2)
        elif naklon > 15:
            led.plot(1, 2)
        elif naklon < -100:
            led.plot(4, 2)
        elif naklon < -15:
            led.plot(3, 2)
        else:
            led.plot(2, 2)
    basic.forever(on_forever)
```

Metodický list

Co si připravit?

- Microbity, prostředí MakeCode, propojení s PC

Co jsme se naučili?

- Jak refaktorovat kód
- Postupné iterace funkcionality a vylepšování aplikací

Aplikace, verze 4

Ano, „téměř spokojený". Nespokojenost je pro programátora požehnáním i prokletím. Nespokojenost jej pohání k tomu, aby výsledek byl co nejlepší, ale taky mu brání v tom, aby byl zcela spokojen se svým dílem.

Dobrý programátor zlepšuje. Nejlepší programátor pozná ten moment, kdy další zlepšování zabere mnoho času a nepřinese výrazně lepší výsledek. Je dobré se takový přístup naučit. Díky tomu budete moci věci nejen programovat, ale taky naprogramovat a dokončit.

Já se rozhodl, že mi vadí mazání celého displeje po každém měření. A to ze dvou důvodů.

Zaprvé: když se bod nehýbe, tak se zcela zbytečně maže a zase kreslí.

Zadruhé: co když budu mít na zbytku displeje něco nakresleného? Třeba nějakou škálu?

A tak jsem se rozhodl, že nejdřív odstraním neustálé překreslování.

Řešení: Budu si pamatovat, kde se nakreslil bod, tj. jeho souřadnici X. Pokud bude nová souřadnice stejná, nebudu dělat nic. Pokud jiná, smažu displej a nakreslím bod na novém místě.

Praktický postup:

- Potřebuju proměnnou, do níž si budu ukládat původní („starou") pozici. Nazvu si ji třeba „stara".
- V té konstrukci, kde teď kreslím bod, nebudu kreslit bod, ale budu si nastavovat druhou proměnnou, která se bude jmenovat „nova". Nastavovat ji budu na stejnou hodnotu, jaká je teď použitá ve funkci „rozsvit" pro parametr „x".

- Na konci porovnám, jestli se „stara“ a „nova“ liší. Pokud ano, tak smažu displej a nakreslím bod na nové souřadnici. Nesmím zapomenout nastavit hodnotu proměnné „stara“ na tu novou hodnotu!
- V bloku „při startu“ musím nastavit hodnotu „stara“ – před prvním měřením by neměla žádnou hodnotu. Nabízí se nastavit ji na nulu, ale co když po prvním měření bude hodnota taky 0? Třeba bude microbit postavený na bok? Pak na konci bude nova obsahovat totéž co stara a žádný bod se nenakreslí.
- Řešením je nastavit do proměnné „stara“ hodnotu, která bude zaručeně jiná. A protože hodnoty jsou 0 až 4, tak klidně 5. Ale lepší je použít třeba 99 nebo -1, aby bylo na první pohled patrné, že to je hodnota, která je „mimo“.

A touto úpravou jsem si rovnou připravil prostor pro řešení druhého problému. Místo „smazat displej“ teď dám jen „zhasni bod na pozici [stara,2]“. Takže si můžu na začátku zobrazit nějaký obrázek, mřížku nebo něco takového.

Tím mi ale blok „Opakuj stále“ nehezky nakynul. Proto udělám něco, čemu se v programování říká *refaktoring kódu* – tedy jeho přepracování tak, aby funkce zůstala stejná.

Refaktoring

Cílem přepracování je nejčastěji zkrácení kódu, odstranění opakujících se částí nebo rozdělení dlouhého kódu do více logických celků.

Já udělám to poslední, a udělám to tak, že z hlavní smyčky „opakuj stále“ vytáhnu stranou tu část, která měří hodnotu akcelerometru a převádí ji na pozici bodu 0 až 4. A když říkám „vytáhnu stranou“, znamená to, že si udělám funkci.

S funkcemi jsme se už setkali. Je to pojmenovaný kus kódu, který dělá nějaký úkol. Říkali jsme si, že mohou mít parametry (tedy hodnoty, které funkce zpracovává) a že sama funkce může nějakou hodnotu vracet.

Teď nastal čas na ono „vracení hodnoty“.

Vytvořím si funkci (je v rozšířené nabídce) a nazvu ji „mereni“.

Poznámka pro pokročilejší: Pomocí bloků se nedá jednoduše vytvořit lokální proměnná. Proměnné jsou globální. Je to, bohužel, nešikovné a nešťastné jak z hlediska programátorského, tak z hlediska didaktického. Pro teď se s tím budeme muset smířit. I když neradi.

Do této funkce zkopíruju vše od začátku měření po konec podmínkového bloku.

Teď nastal problém: ve funkci nastavuju proměnnou „nova“, a s toutéž proměnnou stále pracuju v bloku „opakuj stále“. To není moc dobré – nechci, aby se měnily proměnné, které používám jinde. Co když budu chtít použít tuto funkci jinde, kde už bude proměnná „nova“ použitá – tímto bych si ji přepsal.

Udělám tedy takové trošku krkolomné řešení: proměnnou „nova“ přejmenuju na „hodnota_mereni“. Tedy použiju za podtržítkem stejné jméno jako má funkce a budu si pamatovat, že tuto proměnnou nemám používat nikde jinde.

Jenže se mi přejmenovala i proměnná „nova“ v bloku „opakuj stále“. Takže si vytvořím novou proměnnou „nova“ a dám ji do „opakuj stále“ na všechna místa, kde byla původně (a kde je teď „hodnota_mereni“).

Ve funkci „mereni“ tedy proběhne celé měření a nastaví se požadovaná pozice body do proměnné „hodnota_mereni“. A na konci se zapomene…

To není moc dobré. Vložím tedy na konec blok „návratová hodnota“ z nabídky Funkce. A jako hodnotu, která se vrací, dám proměnnou „hodnota_mereni“.

Když teď proběhne funkce „mereni“, tak udělá, co má, a vrátí číslo z rozsahu 0 až 4, které říká, kde se má nakreslit bod.

Základní otázky jsou: Jak proběhne? A komu to číslo vrátí?

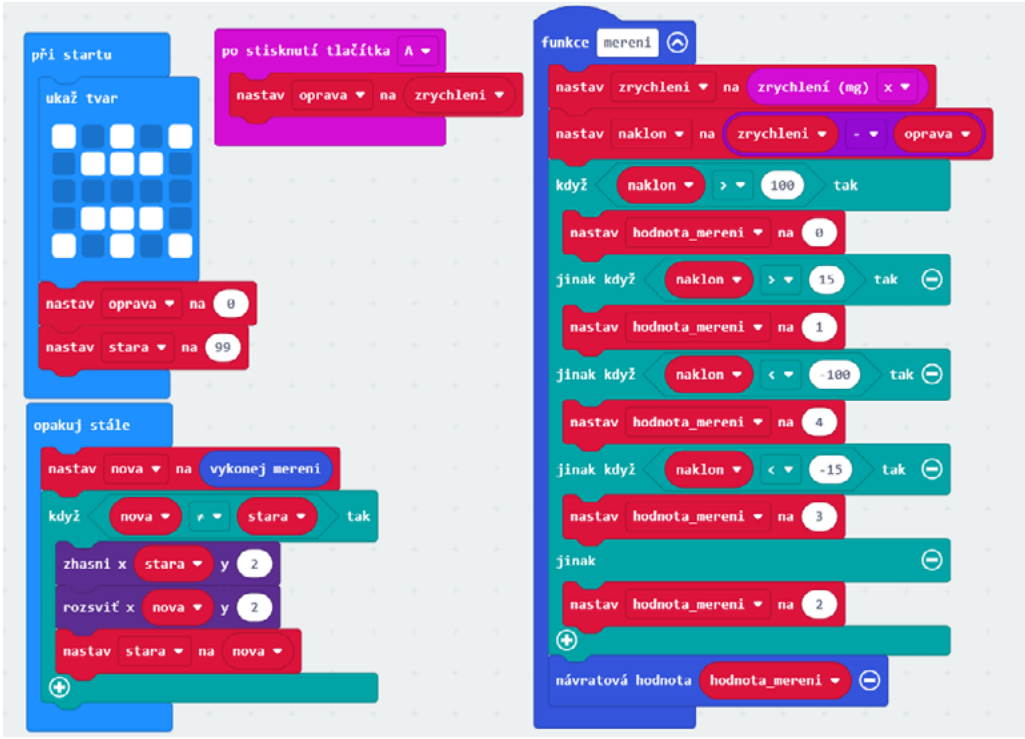
Všimněte si, že v nabídce Funkce jsou dva možné způsoby vyvolání („vykonej“). Jeden je klasický funkční blok (obdélník se „západkami“), druhý je ovál, který používáme tam, kde se používá hodnota. A to je ten, který potřebujeme!

Do bloku „opakuj stále“ si vložíme blok pro nastavení proměnné: „nastav ‚nova‘ na 0“. A na místo té nuly přetáhneme obdélníkový blok „vykonej mereni“.

Význam této konstrukce je následující: nejprve zavolá funkci „mereni“. Ta proběhne tak, jak jsme si popsali výše, a *vrátí návratovou hodnotu*. Tato hodnota se pak použije místo onoho oválného „vykonej…“ – tedy pro nastavení nové hodnoty do proměnné „nova“.

Celý blok tedy můžeme číst jako „nastav ‚nova‘ na návratovou hodnotu funkce ‚mereni‘.“

Všimněte si, že jsem nepřejmenovával proměnnou „zrychleni“. To proto, že ji využívá obsluha stisknutí tlačítka mimo funkci. Není to úplně čisté řešení. V tuto chvíli to nechme tak, hned v následující kapitole to opravíme.




```
JS  input.onButtonPressed(Button.A, function () {
JS      oprava = zrychleni
JS  })
JS  function mereni () {
JS      zrychleni = input.acceleration(Dimension.X)
JS      naklon = zrychleni - oprava
JS      if (naklon > 100) {
JS          hodnota_mereni = 0
```

```
JS      } else if (naklon > 15) {
JS          hodnota_mereni = 1
JS      } else if (naklon < -100) {
JS          hodnota_mereni = 4
JS      } else if (naklon < -15) {
JS          hodnota_mereni = 3
JS      } else {
JS          hodnota_mereni = 2
JS      }
JS      return hodnota_mereni
JS  }
JS  let nova = 0
JS  let hodnota_mereni = 0
JS  let naklon = 0
JS  let zrychleni = 0
JS  let oprava = 0
JS  basic.showLeds(`
JS      # . # . #
JS      . # # # .
JS      . . . . .
JS      . # # # .
JS      # . # . #
JS      `)
JS  oprava = 0
JS  let stara = 99
JS  basic.forever(function () {
JS      nova = mereni()
JS      if (nova != stara) {
JS          led.unplot(stara, 2)
JS          led.plot(nova, 2)
JS          stara = nova
JS      }
JS  })

    def on_button_pressed_a():
        global oprava
        oprava = zrychleni
    input.on_button_pressed(Button.A, on_button_pressed_a)

    def mereni():
        global zrychleni, naklon, hodnota_mereni
```

```
    zrychleni = input.acceleration(Dimension.X)
    naklon = zrychleni - oprava
    if naklon > 100:
        hodnota_mereni = 0
    elif naklon > 15:
        hodnota_mereni = 1
    elif naklon < -100:
        hodnota_mereni = 4
    elif naklon < -15:
        hodnota_mereni = 3
    else:
        hodnota_mereni = 2
    return hodnota_mereni
nova = 0
hodnota_mereni = 0
naklon = 0
zrychleni = 0
oprava = 0
basic.show_leds("""
    # . # . #
        . # # # .
        . . . . .
        . # # # .
        # . # . #
""")
oprava = 0
stara = 99

def on_forever():
    global nova, stara
    nova = mereni()
    if nova != stara:
        led.unplot(stara, 2)
        led.plot(nova, 2)
        stara = nova
basic.forever(on_forever)
```

Metodický list

Co si připravit?

- Microbity, prostředí MakeCode, propojení s PC

Co jsme se naučili?

- Refaktoring a postupné vylepšování

2.11 Ovládání bodu náklonem

Uff, to bylo, co? Spousta verzí jednoho programu. Když už jsem si s ním tak hezky hrál, tak mě napadlo: Co takhle udělat totéž, ale ve dvou osách! Nejen doleva a doprava, ale i dopředu a dozadu.

Displej to dokáže zobrazit. Akcelerometr to dokáže změřit. Tak to pojďme upravit.

Požadovaná funkce: Upravit Vodováhu tak, aby fungovala ve dvou osách.

Řešení: Použijeme poslední verzi Vodováhy, tu s funkcí a opravou, a upravíme ji tak, aby fungovala pro osy X i Y.

Praktický postup:

- Všechny proměnné, co teď používáme a souvisejí s měřením, přejmenujeme tak, že na konec přidáme velké písmeno X. Tedy z nova se stane novaX, stara bude staraX atd. Tedy všechny s výjimkou proměnné hodnota_mereni, kterou používáme pouze ve funkci „mereni".
- Použijeme funkci „mereni", ale jen tu část, kde se z náklonu (po opravě) počítá pozice bodu. Samotné zjištění hodnoty z akcelerometru vrátíme na začátek bloku „opakuj stále".
- Funkci přejmenujeme – místo „mereni" se bude jmenovat „pozice".
- Funkci upravíme tak, aby místo měření převzala nějakou zadanou hodnotu, kterou jí pošleme. Jednou to bude zrychlení v ose X, podruhé v ose Y.
    - Klikněte pravým tlačítkem na funkci, zvolte „Upravit funkci"
    - V editoru funkcí klikněte na „přidat parametr – číslo"

- Přejmenujte parametr na „naklon“ a dejte „Hotovo“
- V hlavičce funkce se objevil parametr „naklon“. Nenajdete ho v nabídce „proměnné“. Pouze zde. Pomocí myši ho vezměte a přetáhněte na všechna místa ve funkci, kde se pracuje s hodnotou náklonu.

- Přejmenujeme „hodnota_mereni“ na „hodnota_pozice“. Jen kosmetická úprava bez vlivu na funkci, ale ať máme pořádek…

Výsledná podoba funkce „pozice“:

- Adekvátně tomu je potřeba upravit volání této funkce. Všimněte si, že volání v bloku „nastav novaX…“ se změnilo – je tam „vykonej pozice (1)“ Místo této jedničky dáme hodnotu naklonX.

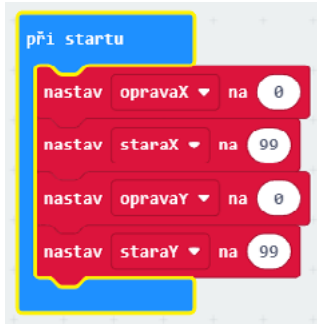
- Proměnná „naklonX“ se vlastně stala úplně zbytečnou. Nastavíme ji – a pak ji použijeme přesně jednou. Pokud nějakou proměnnou použijeme jen jednou, můžeme ji nahradit její hodnotou. Tedy v tomto případě výrazem „zrychleniX-opravaX“. A pak můžeme ze seznamu proměnných proměnnou „naklonX“ klidně smazat.

Šlo by se stejným způsobem zbavit proměnné zrychleniX?

V tento okamžik by měl program fungovat stále tak, jako poslední funkce Vodováhy. Neudělali jsme zatím nic víc, než že jsme přejmenovali proměnné a popřesouvali bloky tak, aniž by to změnilo funkci.

Nastal čas přidat druhou osu.

- Ke všem proměnným, kde je teď X, vytvoříme jejich sourozence s písmenem Y. Tedy novaY, staraY, zrychleniY a opravaY.
- Naklonujeme bloky z „při startu“, kde se nastavují hodnoty opravy a staré pozice:

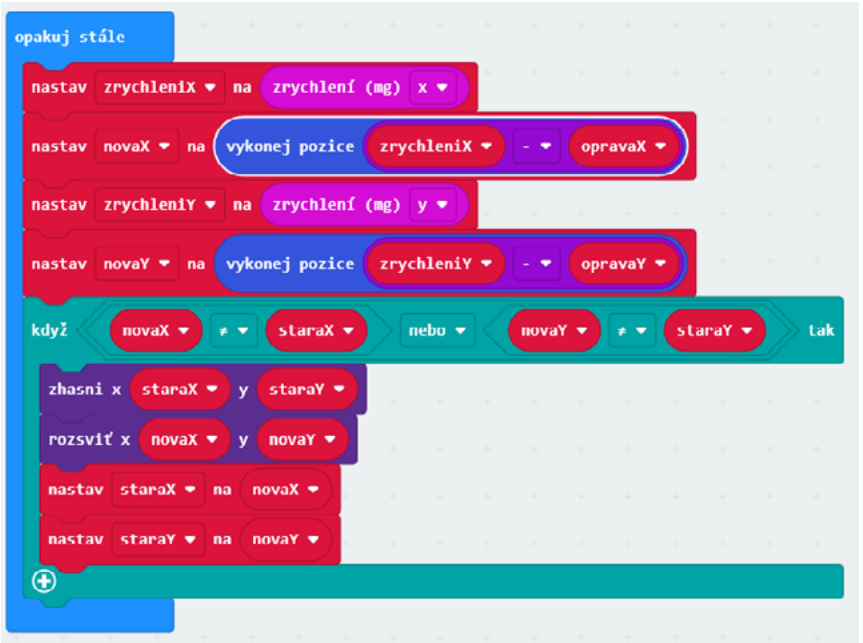


- Totéž uděláme v obsluze stisku tlačítka:

- V bloku „Opakuj stále“ musíme naklonovat měření. Stejně jak měříme zrychleniX a z něj počítáme novaX, tak uděláme totéž pro osu Y.
- Podmínka se musí rozšířit. Budeme kontrolovat nejen to, zda staraX je jiná než novaX, ale i to, zda staraY je jiná než novaY. Použijeme logický výraz NEBO.

Proč zrovna NEBO?

- Přepíšeme dvojky (tedy řádek) na hodnoty staraY a novaY.
- Duplikujeme přiřazení nové hodnoty do staré i pro osu Y.




```
JS  input.onButtonPressed(Button.A, function () {
JS      opravaX = zrychleniX
JS      opravaY = zrychleniY
JS  })
JS  function pozice (naklon: number) {
JS      if (naklon > 100) {
JS          hodnota_pozice = 0
JS      } else if (naklon > 15) {
JS          hodnota_pozice = 1
JS      } else if (naklon < -100) {
JS          hodnota_pozice = 4
JS      } else if (naklon < -15) {
JS          hodnota_pozice = 3
JS      } else {
JS          hodnota_pozice = 2
JS      }
JS      return hodnota_pozice
JS  }
JS  let novaY = 0
```

```
JS  let novaX = 0
JS  let hodnota_pozice = 0
JS  let zrychleniY = 0
JS  let zrychleniX = 0
JS  let opravaY = 0
JS  let opravaX = 0
JS  opravaX = 0
JS  let staraX = 99
JS  opravaY = 0
JS  let staraY = 99
JS  basic.forever(function () {
JS      zrychleniX = input.acceleration(Dimension.X)
JS      novaX = pozice(zrychleniX - opravaX)
JS      zrychleniY = input.acceleration(Dimension.Y)
JS      novaY = pozice(zrychleniY - opravaY)
JS      if (novaX != staraX || novaY != staraY) {
JS          led.unplot(staraX, staraY)
JS          led.plot(novaX, novaY)
JS          staraX = novaX
JS          staraY = novaY
JS      }
JS  })

    def on_button_pressed_a():
        global opravaX, opravaY
        opravaX = zrychleniX
        opravaY = zrychleniY
    input.on_button_pressed(Button.A, on_button_pressed_a)

    def pozice(naklon: number):
        global hodnota_pozice
        if naklon > 100:
            hodnota_pozice = 0
        elif naklon > 15:
            hodnota_pozice = 1
        elif naklon < -100:
            hodnota_pozice = 4
        elif naklon < -15:
            hodnota_pozice = 3
        else:
            hodnota_pozice = 2
```

```
    return hodnota_pozice
novaY = 0
novaX = 0
hodnota_pozice = 0
zrychleniY = 0
zrychleniX = 0
opravaY = 0
opravaX = 0
opravaX = 0
staraX = 99
opravaY = 0
staraY = 99

def on_forever():
    global zrychleniX, novaX, zrychleniY, novaY, staraX, staraY
    zrychleniX = input.acceleration(Dimension.X)
    novaX = pozice(zrychleniX - opravaX)
    zrychleniY = input.acceleration(Dimension.Y)
    novaY = pozice(zrychleniY - opravaY)
    if novaX != staraX or novaY != staraY:
        led.unplot(staraX, staraY)
        led.plot(novaX, novaY)
        staraX = novaX
        staraY = novaY
basic.forever(on_forever)
```

Metodický list

Co si připravit?

- Microbity, prostředí MakeCode, propojení s PC

Co jsme se naučili?

- Použít k řešení komplexnější úlohy předchozí úlohu a rozšířit ji

2.12 Hra Chyť mě!

Učení má být zábava a hra, to říkal už Jan Ámos Komenský. A nejlépe si člověk u programování hraje, když tvoří hru. To si hraje dvakrát: jednak programuje, a pak si hraje tu hru.

Naše hra bude úplně jednoduchá: Na displeji se bude pohybovat zločinec a policajt. Hráč bude ovládat policajta a jeho úkolem bude chytit zločince, který náhodně pobíhá. Hráč bude svou postavu ovládat náklonem microbitu – k tomu využijeme předchozí zkušenosti z vodováhy. Můžeme dokonce zkopírovat funkci „pozice" a použít ji.

Tentokrát využijeme nabídku „Hra", kterou najdete v rozšířeném menu. Nabídka Hra je plná výrazů jako „sprite" a „životy" a „skóre", ale nebojte, za chvilku je budete znát.

Sprite je v terminologii počítačových her nějaký objekt, který se může pohybovat. Třeba raketa, střela, postava, protivník, cokoli. Toto „cokoli" se nazývá u microbitu „postava". Postava není přímo figurka, je to programátorsky řečeno objekt, který má nějaké *vlastnosti* a *metody*.

Sprite znamená v angličtině „skřítek". Proto se právě toto slovo začalo používat ve hrách pro označení různých příšerek. Souvislost s citronovou limonádou značky Sprite tam není…

Vlastnosti objektu jsou podobné proměnným, až na to, že jich objekt může mít víc. Třeba u objektu typu „sprite" (programátorsky řečeno „objekt třídy sprite") jsou to souřadnice X a Y, směr pohybu, jeho jas a blikání.

Výhoda tohoto přístupu je, že v proměnné máme celý sprite. Nemusíme si proto dělat spoustu proměnných a pojmenovávat je „policajtX, policajtY, zlodejX, zlodejY, …"

Metody objektu jsou zase podobné funkcím, s tím rozdílem, že jsou svázané s daným objektem. Objekt třídy „sprite" tak má metody jako „posuň o N", „otoč o N", „nastav pozici v souřadnici X na…"

V tuto chvíli zjednodušuju, jak jen to je možné. Samotné objektové programování je disciplína, o níž jsou napsány celé stohy knih, ale mým cílem není přidávat další knihu o objektovém programování. Zájemce o větší podrobnosti proto odkážu na příslušnou literaturu. Tamtéž s dovolením odkážu i všechny rozhořčené čtenáře – programátory, kteří shledávají můj popis nedostatečně komplexní a přesný…

U microbitu to zas tak úžasné nebude, naším spritem bude rozsvícený bod na displeji. Ale proti obyčejnému bodu má sprite několik výhod:

- Nemusíme se starat o jeho kreslení a mazání, o to se stará „něco uvnitř“ (ve skutečnosti to „uvnitř“ funguje úplně stejně jako v naší Vodováze s proměnnými staraX, novaY atd.) My jen říkáme, kde se bod-sprite nachází a kam se má posunout.
- Sprite můžeme přiřadit do proměnné.
- Sprite nabízí užitečné metody, jako třeba zjištění, jestli se sprite dotýká okraje displeje, nebo jestli se dotýká jiného. Tyto předpřipravené metody nám ušetří spoustu práce a programování.

Místo dlouhých řečí začneme rovnou ukázkou. Vytvoříme si sprite („policajta“), a budeme s ním pohybovat stejně jako jsme pohybovali bublinkou ve vodováze. Pojďme na to.

Hráč

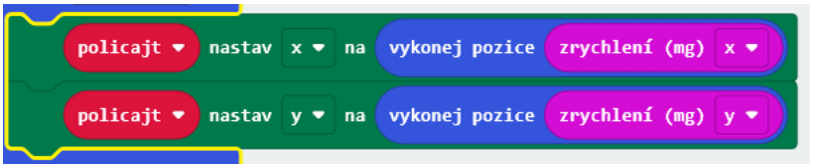
Začneme tím, že si vytvoříme proměnnou „policajt“ a v bloku Při startu do ní přiřadíme novou postavu. K tomu slouží blok „vytvoř postavu“ z nabídky Hra. Má oválný tvar, takže už tím napovídá, že jde o nějakou hodnotu, vhodnou k přiřazení *někam*.

První blok tedy bude „nastav ‚policajt‘ na ‚vytvoř postavu na x:2, y:2‘“. Tím jsme si vytvořili objekt třídy „sprite“ (proč se to jmenuje „vytvoř postavu“ a ne „vytvoř sprite“, to nevím, možná se to změní) a přiřadili jsme jej do proměnné „policajt“.

V cyklu „opakuj stále“ budeme měnit policajtovi jeho pozici X a Y podle náklonu microbitu. K tomu použijeme funkci „pozice“ z Vodováhy.

Pro změnu pozice nemůžeme použít blok na změnu proměnné, ale *metodu spritu*. Najdete ji opět v nabídce Hra a jmenuje se „sprite nastav X na …“

Vložte ji do bloku a místo „sprite“ vyberte „policajt“, vlastnost ponechte na X a na místo hodnoty dejte volání funkce „pozice“, tedy „vykonej pozice“. Parametrem bude vstup z akcelerometru, tedy zrychlení v ose X. Totéž pro osu Y.



A to je vše. Teď to funguje stejně jako ovládání bodu náklonem.

Slyším vás, jak voláte: Moment – to je celé? To je opravdu celé? Tak proč jsme se předtím patlali s tím rozsvěcením bodu a pamatováním toho všeho?

Odpověď je prostá: chci, abyste rozuměli tomu, jak ta věc funguje uvnitř. Že si pak práci zjednodušíte, to je v pořádku. Tak to má být. Ale je dobré vědět, jak to funguje. Kdybychom začali rovnou spritem, tak by všechno bylo „black box", magická černá skříňka, která „něco dělá". Někdy to je dobře, ale zrovna v tomto případě bylo na místě vysvětlit základní principy na jednodušším příkladu.

Vyzkoušejte si, jak to funguje. Docela dobře, ne? Jednu jedinou věc bych tomu vytkl, totiž že ovládání je neintuitivní. Nakláníme do opačného směru, než jde bod. Ale to se dá snadno upravit. Ve funkci „pozice" přepíšeme výsledné hodnoty „hodnota_pozice" tak, že místo 4 dáme 0, místo 3 dáme 1, místo 1 dáme 3 a místo 0 dáme 4. Tím se vše obrátí a bod začne po displeji putovat tak, jak jsme zvyklí.

Refaktorování 1

Kód si upravíme. Nastavování pozice policajta podle náklonu destičky přesuneme do samostatné funkce „pohybPolicajta" a do smyčky „opakuj stále" dáme jen volání: „vykonej pohybPolicajta".

Nemá to žádný význam pro samotný běh programu. Jen jsme si dali jeden logický celek stranou, dali jsme mu jasný název a můžeme na něj zapomenout.

Soupeř

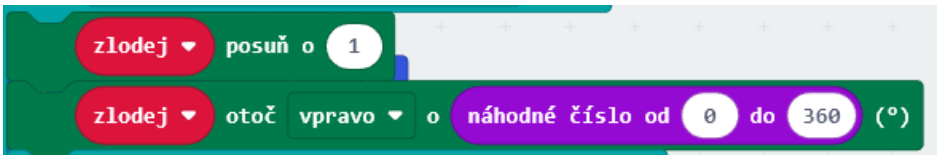
Soupeřem, oním „zlodějem", kterého budeme chytat, je taky sprite. Takže začneme stejnými přípravnými kroky: vytvoříme proměnnou „zlodej" a přiřadíme do ní vytvořenou postavu na pozici [0,0].

Teď musíme vyřešit zlodějův pohyb. Podíváme se, co nám třída „sprite" nabízí za metody:

- Posuň o N
- Otoč o N °
- Změň pozici v dané ose o N
- Nastav pozici v dané ose na N

Samozřejmě můžeme náhodně posouvat v obou osách, ale to by zloděj před policajtem neutíkal, spíš by prováděl náhodnou teleportaci. Takový hon by byl na houby. Proto řekneme, že zloděj se pokaždé posune o 1 políčko a náhodně změní směr.

Zkusíme si to přidat do bloku „opakuj stále“:



Po nahrání a spuštění vidíme hned první problém: zloděj běhá po displeji jak bodnutý vosou a chytíte ho jen náhodou. Musíme ho zpomalit. Jak na to?

Můžeme přidat známý blok „čekej“. Ale když to zkusíme, zpomalí se nejen zloděj, ale i policajt. A nejen to: policajt začne provádět ony „teleportační skoky“ – ve skutečnosti jsme nezpomalili jeho pohyb, jen zobrazování. Tudy cesta nepovede.

Ale kudy tedy vede cesta?

Ukážeme si nejprve tradiční cestu.

Tradiční cesta spočívá v tom, že použijeme pomocnou proměnnou, třeba „pohyb“, a do ní si nastavíme aktuální čas. Ne ten reálný, jako že je úterý, půl páté, ale ten vnitřní.

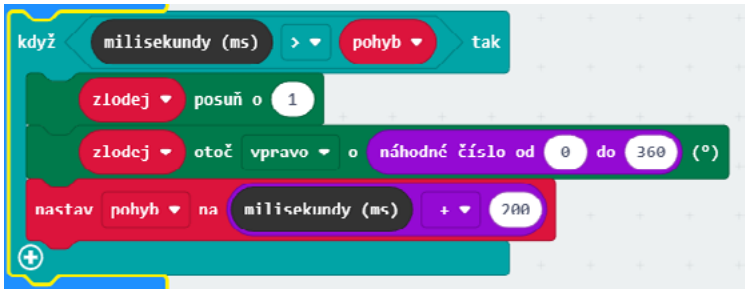
Vnitřní čas se počítá v milisekundách od startu programu.

Na konci rozšířeného menu najdete nabídku „Ovládání“, v originále Control. Překlad není zcela přesný, přesnější by bylo „Řízení“. V této nabídce je položka, která se jmenuje „milisekundy (ms)“ – a to je to, co hledáme.

Pak se při každém průchodu smyčkou „opakuj stále“ podíváme, jestli je aktuální čas už víc než třeba 200 milisekund po tom, který je zaznamenaný v proměnné „pohyb“. A pokud ano, tak pohneme zlodějem a nastavíme si proměnnou „pohyb“ na novou hodnotu.

V praxi se nepostupuje takto, ale trošku jinak. V praxi se nenastavuje do proměnné aktuální čas, ale čas, kdy má dojít k akci, tedy „aktuální čas + 200“ (200 je hodnota, jak dlouho chceme čekat). A pak podmínka zní jednoduše: „když je aktuální čas vyšší než nastavený čas, tak…“

Vlastně si děláme obdobu kuchyňské minutky: až bude čas vyšší o N, tak se něco stane.



Nastavení proměnné „pohyb“ si jen zkopírujeme do bloku „při startu“, ať máme vhodnou počáteční hodnotu.

> *Je toto nastavení nutné? A mohlo by místo něj být třeba nastavení na 0?*

Druhá cesta je o něco jednodušší, ale nebudu ji ukazovat. Dostanete ji jako samostatný úkol.

> *Nahraďte počítání milisekund jiným způsobem. Nápověda se skrývá v nabídce „Smyčky“... Máte ji? Ještě napovím: „každých 200 ms“...*

Refaktorování 2

Pohyb zloděje si opět dáme do samostatné funkce. Netřeba vymýšlet složitosti – pojmenujeme si ji „pohybZlodeje“ a v bloku „opakuj stále“ ji jen vykonáme.

Spustíme, otestujeme…

Situace je o něco lepší, ale mám ještě jeden zlepšovák, který není tak úplně refaktorování, protože mění funkci, ale to nevadí: zloději nastavíme menší jas, aby byl rozeznatelný. Za jeho *inicializaci* (tedy za tím místem, kde je nastavení proměnné „zlodej“) přidejte volání metody „sprite nastav x na…“ a změňte sprite na „zlodej“, místo vlastnosti „x“ vyberte vlastnost „jas“ a hodnotu nastavte na 32.

Hodnota jasu je číslo mezi 0 až 255 (maximum). Možná budete muset na svém microbitu použít hodnotu jinou, než je 32. Já jsem u svého použil 128, tedy poloviční jas, a zjistil jsem, že simulátor sice ukazuje krásné málo jasné body, ale reálný microbit stále svítil docela dost. Postupným snižováním jsem dospěl až k hodnotě 32, která už mi připadá dostatečně viditelně ztlumená. Ale možná mi jen microbit moc svítí.

Chycen při činu

Zloděj běhá, policajt za ním, a dál se nic neděje. Musíme přidat kontrolu toho, jestli náhodou policajt zloděje nelapnul.

V nabídce Hra na to máme logický výraz „sprite se dotýká …“ Je to špičatý obdélník, tedy vhodný k použití v bloku „Když…“ Dejte si tedy do bloku „opakuj stále“ blok „když…“ a jako podmínku zadejte tu metodu „sprite se dotýká“. Nastavte ji na „policajt dotýká se zloděj“ (policajt a zloděj jsou proměnné, samozřejmě).

Tak, a co teď? Policajt chytil zloděje, tak to musíme nějak oslavit. Ideálně tím, že si připočítáme bod.

Vytvoříme si proto proměnnou „skore“ a… Ne, počkejte!

Využijeme toho, že v nabídce Hra jsou pomocné funkce pro počítání životů a skóre. Do bloku „při startu“ dáme blok „nastav skóre na 0“ a do podmíněného bloku „když se policista dotýká zloděje“ dáme blok „změň skóre o 1“.

Zkuste si, jak to bude fungovat teď.

Skvělé, funkce asi počítá body, ale co je hlavní: zobrazí animaci na displeji. To je hezké. Až hra skončí, tak se skóre vypíše…

Kdy vlastně hra skončí? Dobrá otázka, ale to si necháme na později.

Teď by měl policista chytit dalšího padoucha. Můžeme tedy zadat blok „smaž postavu“ a pak znovu vytvořit postavu a přiřadit ji do proměnné „zloděj“. Anebo můžeme ekologicky zloděje recyklovat a použít toho už chyceného. My sice víme, že to je tentýž, ale co – policajtovi to říkat nebudeme.

Kdybychom jen přičetli bod, tak co se stane? Pokud by hráč udržel destičku ve stejné pozici, tak by po animaci znovu zloděje chytil a znovu a znovu a znovu. To by bylo příliš snadné. Proto musíme dát zloději nějaký náskok.

Takže po započtení bodu se spritem-zlodějem uskočíme někam stranou. Nejprve ho otočíme o nějaký náhodný úhel, a pak ho posuneme o náhodný počet pozic. Použijeme k tomu náhodná čísla. Úhel bude od 0 do 180, počet pozic od 1 do 4. Minimum je jedna pozice, aby se alespoň o něco málo posunul.

A když už tam máme tu animaci, co takhle přidat ještě zvukový efekt? Z nabídky „hudba" vyberte možnost „hraj melodii dadadum jednou", dejte ji před změnu skóre a trochu změňte parametry: melodie bude „skok nahoru" a místo „jednou" zvolte „jednou na pozadí".

Zkuste si: co by se stalo, kdybyste ponechali možnost „jednou"?

Melodie je trochu pomalá, takže občas chytnete dalšího zloděje dřív, než melodie dozní. Tomu snadno odpomůžeme tím, že nastavíme tempo třeba na 300 (nabídka Hudba, blok „nastav tempo").

A klasické refaktorování: kontrolu přesuňte do nové funkce, kterou pojmenujete „kontrola". Do bloku „opakuj stále" dejte blok vykonání funkce „kontrola", a dejte jej hned za vykonávání funkce „pohybPolicajta".

Game over...

Hra je to dobrá, jen nějak nekončí. Honíte další a další zloděje a nic. Každá hra by ale měla mít i konec. Jak jinak bychom věděli, kdo má nejvyšší skóre?!

Musíme vymyslet mechanismus, který zajistí, že hra skončí. V hrách se obvykle používají „životy". Máte třeba tři, a postupně vám ubývají. Uděláme to stejně. Ale kdy má hráč o svůj život přijít?

Možností se nabízí spousta, mě napadla jedna úplně primitivní: Budeme počítat čas od chycení posledního zloděje (tedy od „zadržení pachatele"). Pokud během pěti sekund hráč nechytí zloděje, přijde o život. Když chytí, počítá se mu znovu 5 sekund.

Opět využijeme stejné techniky jako u zpomalování zlodějů. Vytvoříme si proměnnou „chyt", kterou si nastavíme na aktuální čas + 5 sekund (tedy 5000 milisekund). A v hlavní smyčce bude kontrola, zda od chycení posledního zloděje už uběhl požadovaný čas. Tedy „Když milisekundy > chyt, tak..."

Využijeme další pomocnou službu z nabídky Hra, totiž „uber životy (1)". Zkuste to. Pět sekund nehýbejte a uvidíte, že ubrání životů přehraje video efekt.

Před ubrání životů dejte „hraj melodii“, vyberte „wawawawaa“ a nechte ji přehrát jednou na pozadí. A za animaci dejte opět nastavení proměnné „chyt“ na aktuální čas plus 5000.

Nastavení proměnné „chyt“ zkopírujte do bloku, v němž se ošetřuje stav, kdy policajt chytil zloděje. Dejte to až na konec, za „změň skóre o 1“.

Nastavení proměnné „chyt“ zkopírujte ještě jednou, na konec bloku „při startu“. Před něj nezapomeňte dát blok „nastav životy (3)“.

Spusťte, hrajte...

Všimněte si, že když počet životů klesne na 0, hra sama skončí, nemusíme používat blok „konec hry“. Microbit vypíše hlášku „game over“ a „score: XX“. Novou hru spustíte buď resetem, nebo stiskem tlačítek A a B najednou (to taky zařizuje knihovna Hra).

Refaktorování 3

Refaktorujte kód. Přesuňte kontrolu zbývajících životů do samostatné funkce.

Během refaktorování kódu si všimněte, že v něm máme několik „magických konstant“. Třeba počet milisekund, kterým brzdíme pohyb zloděje – je to 200, ale když jej budeme chtít upravit, musíme to změnit na více místech. To samé pro čas, který má policista na chycení zloděje. Je to 5000, ale na třech místech.

Vytvořte si proměnné „RychlostZlodeje“ a „CasovyLimit“ a při startu si do nich uložte ony hodnoty 200 a 5000.

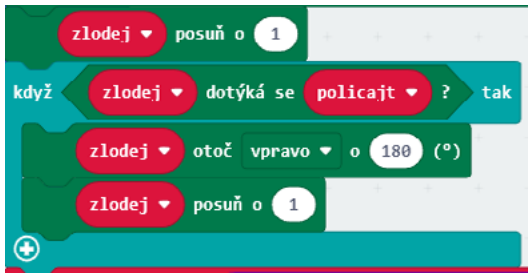
Všimněte jsi, že jsem použil první písmena velká. Důvod je jednoduchý: blokový editor nemá možnost vytvořit takzvanou konstantu. Konstanta je podobná proměnné, ale její hodnota se v programu po prvním nastavení už nedá změnit. A protože blokový editor nezná konstanty, tak si proměnné pojmenuju s velkým písmenem, abych je odlišil od normálních proměnných.

A teď výskyty oněch konstant nahraďte proměnnými RychlostZlodeje a CasovyLimit.

Play harder

V téhle verzi stačí někdy počkat, a on do policajta narazí zloděj sám. Tomu se dá snadno předejít.

Ve funkci Pohyb zloděje přidáme jednu kontrolu. Kdyby zloděj během svého pohybu vrazí do policajta, tak ho otočíme o 180 stupňů a uděláme krok zpátky. Takže se nestane, že když policista bude v klidu, tak by mu zloděj sám skočil do náruče.



Další úpravy

Tentokrát vám neprozradím, jak je udělat, ale zkuste si to vymyslet sami.

> *Všimněte si v nabídce Hra podnabídky „...více". V ní jsou dva bloky. Pauza a Obnov. Zařiďte, aby po stisknutí tlačítka A byla hra zapauzována. Po dalším stisknutí musí být odpauzována.*

Pravděpodobně jste udělali řešení správně, ale zapomněli jste na jednu věc: Musíte pozastavit i počítadlo onoho pětisekundového intervalu. Jenže bohužel, hodnota milisekund systémového času se zastavit nedá. Takže kontrolu ve funkci „zivoty" zapouzdříme do podmínkového bloku „když je spuštěno" a bude probíhat jen tehdy, když je spuštěná hra.

> *Toto řešení má ale ještě jednu chybu. Pokud odpauzujete, tak okamžitě přijdete o život. Sice se čas nekontroloval, ale jakmile se začne opět kontrolovat, tak je dávno přes limit. Co s tím?*

```
JS  function kontrola () {
JS      if (policajt.isTouching(zlodej)) {
JS          music.startMelody(music.builtInMelody(Melodies.JumpUp), MelodyOptions.
    OnceInBackground)
```

```
JS            zlodej.turn(Direction.Right, randint(0, 180))
JS            zlodej.move(randint(1, 4))
JS            game.addScore(1)
JS            basic.pause(1000)
JS            chyt = control.millis() + CasovyLimit
JS        }
JS    }
JS    function pohybZlodeje () {
JS        if (control.millis() > pohyb) {
JS            zlodej.turn(Direction.Right, randint(0, 360))
JS            zlodej.move(1)
JS            if (zlodej.isTouching(policajt)) {
JS                zlodej.turn(Direction.Right, 180)
JS                zlodej.move(1)
JS            }
JS            pohyb = control.millis() + RychlostZlodeje
JS        }
JS    }
JS    input.onButtonPressed(Button.A, function () {
JS        if (game.isPaused()) {
JS            chyt = control.millis() + (CasovyLimit - pauza)
JS            game.resume()
JS        } else {
JS            pauza = chyt - control.millis()
JS            game.pause()
JS        }
JS    })
JS    function pozice (naklon: number) {
JS        if (naklon > 100) {
JS            hodnota_pozice = 4
JS        } else if (naklon > 15) {
JS            hodnota_pozice = 3
JS        } else if (naklon < -100) {
JS            hodnota_pozice = 0
JS        } else if (naklon < -15) {
JS            hodnota_pozice = 1
JS        } else {
JS            hodnota_pozice = 2
JS        }
JS        return hodnota_pozice
JS    }
```

```
JS  function zivoty () {
JS      if (game.isRunning()) {
JS          if (control.millis() > chyt) {
JS              music.startMelody(music.builtInMelody(Melodies.Wawawawaa), MelodyOptions.
    OnceInBackground)
JS              game.removeLife(1)
JS              basic.pause(1000)
JS              chyt = control.millis() + CasovyLimit
JS          }
JS      }
JS  }
JS  function pohybPolicajta () {
JS      policajt.set(LedSpriteProperty.X, pozice(input.acceleration(Dimension.X)))
JS      policajt.set(LedSpriteProperty.Y, pozice(input.acceleration(Dimension.Y)))
JS  }
JS  let hodnota_pozice = 0
JS  let pauza = 0
JS  let chyt = 0
JS  let CasovyLimit = 0
JS  let pohyb = 0
JS  let RychlostZlodeje = 0
JS  let zlodej: game.LedSprite = null
JS  let policajt: game.LedSprite = null
JS  policajt = game.createSprite(2, 2)
JS  zlodej = game.createSprite(0, 0)
JS  zlodej.set(LedSpriteProperty.Brightness, 32)
JS  RychlostZlodeje = 200
JS  pohyb = RychlostZlodeje
JS  game.setScore(0)
JS  game.setLife(3)
JS  music.setTempo(300)
JS  CasovyLimit = 5000
JS  chyt = control.millis() + CasovyLimit
JS  basic.forever(function () {
JS      if (game.isRunning()) {
JS          pohybPolicajta()
JS          kontrola()
JS          pohybZlodeje()
JS          zivoty()
JS      }
JS  })
```

```
def kontrola():
    global chyt
    if policajt.is_touching(zlodej):
        music.start_melody(music.built_in_melody(Melodies.JUMP_UP),
            MelodyOptions.ONCE_IN_BACKGROUND)
        zlodej.turn(Direction.RIGHT, randint(0, 180))
        zlodej.move(randint(1, 4))
        game.add_score(1)
        basic.pause(1000)
        chyt = control.millis() + CasovyLimit
def pohybZlodeje():
    global pohyb
    if control.millis() > pohyb:
        zlodej.turn(Direction.RIGHT, randint(0, 360))
        zlodej.move(1)
        if zlodej.is_touching(policajt):
            zlodej.turn(Direction.RIGHT, 180)
            zlodej.move(1)
        pohyb = control.millis() + RychlostZlodeje

def on_button_pressed_a():
    global chyt, pauza
    if game.is_paused():
        chyt = control.millis() + (CasovyLimit - pauza)
        game.resume()
    else:
        pauza = chyt - control.millis()
        game.pause()
input.on_button_pressed(Button.A, on_button_pressed_a)

def pozice(naklon: number):
    global hodnota_pozice
    if naklon > 100:
        hodnota_pozice = 4
    elif naklon > 15:
        hodnota_pozice = 3
    elif naklon < -100:
        hodnota_pozice = 0
    elif naklon < -15:
        hodnota_pozice = 1
    else:
```

```
            hodnota_pozice = 2
        return hodnota_pozice
    def zivoty():
        global chyt
        if game.is_running():
            if control.millis() > chyt:
                music.start_melody(music.built_in_melody(Melodies.WAWAWAWAA),
                    MelodyOptions.ONCE_IN_BACKGROUND)
                game.remove_life(1)
                basic.pause(1000)
                chyt = control.millis() + CasovyLimit
    def pohybPolicajta():
        policajt.set(LedSpriteProperty.X, pozice(input.acceleration(Dimension.X)))
        policajt.set(LedSpriteProperty.Y, pozice(input.acceleration(Dimension.Y)))
    hodnota_pozice = 0
    pauza = 0
    chyt = 0
    CasovyLimit = 0
    pohyb = 0
    RychlostZlodeje = 0
    zlodej: game.LedSprite = None
    policajt: game.LedSprite = None
    policajt = game.create_sprite(2, 2)
    zlodej = game.create_sprite(0, 0)
    zlodej.set(LedSpriteProperty.BRIGHTNESS, 32)
    RychlostZlodeje = 200
    pohyb = RychlostZlodeje
    game.set_score(0)
    game.set_life(3)
    music.set_tempo(300)
    CasovyLimit = 5000
    chyt = control.millis() + CasovyLimit

    def on_forever():
        if game.is_running():
            pohybPolicajta()
            kontrola()
            pohybZlodeje()
            zivoty()
    basic.forever(on_forever)
```

Metodický list

Co si připravit?

- Microbity, prostředí MakeCode, propojení s PC

Co jsme se naučili?

- Vyvinout aplikaci od úvodního zadání po hotový výsledek
- Použít „herní“ knihovnu
- Vývoj metodou postupných iterací, které lze rovnou zkoušet.