UNIVERZITA PALACKÉHO V OLOMOUCI
Pedagogická fakulta
Katedra technické a informační výchovy

DIPLOMOVÁ PRÁCE

Josef Fojtík

Hry a aktivizace informatického myšlení,
jazyk Python

Olomouc 2024 vedoucí práce: doc. RNDr. Petr Šaloun, Ph.D.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a uvedl veškerou použitou literaturu a zdroje.

V Olomouci dne 18. 6. 2024 ……..

Bc. Josef Fojtík

Poděkování

Tímto bych chtěl poděkovat panu doc. RNDr. Petru Šalounovi, Ph.D. za vedení mé diplomové práce a cenné rady a připomínky.

Dále chci poděkovat své rodině za podporu po celou dobu studia. Svým spolužákům a přátelům za jejich obětavou pomoc. V neposlední řadě děkuji Bohu za to, že jsem dokončil tuto práci.

ANOTACE

Jméno a příjmení:	Josef Fojtík
Katedra:	Katedra technické a informační výchovy
Vedoucí práce:	doc. RNDr. Petr Šaloun, Ph.D.
Rok obhajoby:	2024

Název práce:	Hry a aktivizace informatického myšlení, jazyk Python
Název v angličtině:	Games and activization of computational thinking, Python programming language
Anotace práce:	Diplomová práce se zabývá rozvojem informatického myšlení pomocí výuky algoritmizace a programování na základních školách. Ukazuje, proč je Python vhodným jazykem pro výuku programování a představuje možnosti jeho využití ve výuce.
Klíčová slova:	Informatické myšlení, Python, BBC micro:bit
Anotace v angličtině:	The thesis focuses on the development of computational thinking through the teaching of algorithmization and programming in elementary schools. It explains why Python is a suitable language for teaching programming and presents the possibilities of its use in education.
Klíčová slova v angličtině:	Computational thinking, Python, BBC micro:bit
Přílohy vázané v práci:	0
Rozsah práce:	79
Jazyk práce:	čeština

Obsah

Úvod

V současné době, kdy se svět stále více digitalizuje a technologie pronikají do všech aspektů našeho života, je klíčové, aby se mladí lidé již od útlého věku seznamovali s informatickým myšlením a základy programování. Informatické myšlení je nejen základním kamenem pro porozumění a vytváření softwaru, ale také rozvíjí kritické myšlení, logiku a schopnost řešit problémy. Tyto dovednosti jsou nezbytné nejen pro budoucí programátory, ale i pro širokou škálu profesí, které vyžadují analytický přístup a schopnost pracovat s informacemi. Tato práce se v teoretické části ukazuje postupné zařazování informačního myšlení do kurikula a také jeho současnou podobu.

Tyto dovednosti mohou žákům poskytnout rozdílovou výhodu oproti jejich vrstevníkům, kteří toto myšlení od útlého věku nerozvíjí. Vzhledem k současné globalizaci společnosti a rozvoji internetu se již o stejnou pracovní pozici nemusí ucházet pouze 2 lidé ze stejného města, ale také 2 lidé z opačných koutů světa. Jedním z úkolů našeho vzdělávacího systému je připravit žáky, aby obstáli v tomto rychle se měnícím světě a vyrovnali se ve svých schopnostech a znalostech také žákům ze zahraničních zemí. K tomu musíme zapojit také nejnovější technologie a poznatky, které máme v k dispozici a účinně je implementovat do výuky.

Tato diplomová práce volně navazuje na autorovu bakalářskou práci, která se zabývá programováním v prostředí *Scratch*, což je vizuální programovací prostředí. Zaměřujeme se zde na využití her a aktivit pro rozvoj informatického myšlení na základních školách, s důrazem na programovací jazyk Python. Python byl vybrán z několika důvodů: je relativně snadno pochopitelný pro začátečníky, má čitelnou a jednoduchou syntaxi, a přitom je dostatečně mocný a univerzální pro řešení široké škály problémů. Navíc, díky rozsáhlé komunitě a bohaté dokumentaci, nabízí mnoho zdrojů, které mohou učitelům i studentům pomoci při učení. Výběru jazyka se potom blíže věnujeme v samostatné kapitole.

Hry a interaktivní aktivity se ukázaly být efektivním nástrojem při výuce programování. Děti se prostřednictvím her učí novým konceptům přirozeně a s radostí, což je klíčové pro udržení jejich zájmu a motivace.

Cílem této práce je analýza současných řešení a přístupů k výuce programování a výběr vhodného řešení pro naprogramování hry pomocí jazyka Python.

1 Výuka informatiky v ČR

1.1 Historie

Zpočátku šedesátých let 20. století byla výuka informatiky výhradně záležitostí technických oborů vysokých škol. Počítače byly drahé a používat je uměli výhradně odborníci. Tato situace se však během dvou dekád zcela změnila. Počítače se stávaly cenově dostupnější a skladnější. Již ke konci sedmdesátých let 20. století začaly osmibitové mikropočítače zasahovat do života středních a základních škol. Společně s jejich příchodem se začala objevovat také otázka, jak tento nový obor uchopit z didaktického hlediska. Zpočátku to byla záležitost nadšených učitelů a žáků v rámci zájmových kroužků. Postupně však výpočetní technologie pronikaly do života lidí ve společnosti stále častěji. Vzhledem k jejich využití se předpokládalo, že každý žák bude muset zvládnout základy programování. Začaly se tedy vést diskuse o tom, v jakém programovacím jazyce učit žáky programovat. V té době již byly dostupné programovací jazyky vhodné pro potřeby výuky jako *BASIC, Logo, Pascal,* a od 80. let také *Robot Karel* od českých vývojářů. (Chráska, 2004, s. 223)

Diskuse o programování na školách však utlumil zpočátku 90. let masivní nástup počítačů kategorie PC (*personal computer* respektive *osobní počítač*), které přinesly přívětivější *grafické uživatelské rozhraní* (GUI) a nabídku příjemného programového vybavení. Pomocí GUI již dokázal uživatel ovládat počítač intuitivně bez nutné znalosti množství příkazů pro příkazovou řádku. Počítače už tedy přestaly být záležitostí odborníků (programátorů, operátorů), ale našly uplatnění také mezi laiky, kteří k jejich využití již nemuseli umět programovat (Chráska, 2004, s. 223). Na tuto skutečnost reagovalo také české školství, které začalo přeorientovávat výuku od programování k *práci s kancelářskými aplikacemi* a také na *využití internetu a elektronické komunikace*. (Stuchlíková a Jelínek, 2015, s. 166)

Úroveň výuky informatiky na školách však byla velice rozdílná. Vláda roku 2000 na tuto diverzitu reagovala vydáním dokumentu *Koncepce státní informační politiky ve vzdělávání* (SIPVZ), který se stal důležitým milníkem nejen pro výuku informatiky, ale pro celý vzdělávací systém ČR. Tento dokument obsahuje konkrétní cíle pro jednotlivé roky 2000 – 2005 a také dlouhodobé cíle, které mají vést k dosažení *informační společnosti*. Dokument se netýkal pouze školství, ale upravoval také celoživotní vzdělávání pracovníků, zejména ve veřejné správě a zdravotnictví. Každá škola měla nyní povinnost provozovat učebnu s multimediálními počítači připojenými k internetu. Ke zřízení těchto učeben a také jako pomoc pro učitele v počítačem podporované výuce byla zřízena na školách nová funkce *koordinátora ICT*. (MŠMT, 2000)

Po dosažení stanovených cílů byly vytvořeny vhodné podmínky pro přijetí nového školského zákona roku 2006. Do základních škol byla poprvé plošně zavedena povinná výuka v samostatné vzdělávací oblasti *Informační a komunikační technologie.* Hodinová dotace však byla pouze 1 hodina týdně v jednom ročníku daného stupně (Stuchlíková a Jelínek, 2015, s. 167). Tato minimální dotace však dle učitelů informatiky (Neumajer, 2009) neumožňovala dosažení požadovaných výstupů Rámcového vzdělávacího programu pro základní vzdělávání (dále jen RVP ZV).

1.2 Strategie digitálního vzdělávání do roku 2020

O další zásadní změně výuky informatiky bylo rozhodnuto, když vláda v listopadu 2014 představila na návrh MŠMT *Strategii digitálního vzdělávání do roku 2020* (dále jen SDV2020). Tento dokument udává směr, kterým se má ubírat výuka informatiky v následujících letech. Naznačuje také změny, které nastanou po roce 2020, kdy proběhla rozsáhlá změna RVP ZV zejména ve vzdělávací oblasti *Informační a komunikační technologie*.

V SDV2020 je *digitální vzdělávání* definováno jako *vzdělávání, které reaguje na změny ve společnosti související s rozvojem digitálních technologií a jejich využíváním v nejrůznějších oblastech lidských činností. Zahrnuje jak vzdělávání, které účinně využívá digitální technologie na podporu výuky a učení, tak vzdělávání, které rozvíjí digitální gramotnost žáků a připravuje je na uplatnění ve společnosti a na trhu práce, kde požadavky na znalosti a dovednosti v segmentu informačních technologií stále rostou.* Cílem strategie je nastavit podmínky a procesy ve vzdělávání, které toto digitální vzdělávání umožní realizovat. (MŠMT, 2014)

Digitální vzdělávání se netýká pouze výuky informatiky, ale prostupuje celým výukovým procesem. Strategie digitálního vzdělávání definuje 3 prioritní cíle:

1. Otevřít vzdělávání novým metodám a způsobům učení prostřednictvím digitálních technologií.
2. Zlepšit kompetence žáků v oblasti práce s informacemi a digitálními technologiemi.
3. *Rozvíjet informatické myšlení žáků.*

Většina učitelů a ředitelů škol projevuje zájem a vyvíjí aktivity pro začleňování ICT do výuky. Zahraniční zkušenosti a Česká školní inspekce však poukazují na celou řadu překážek, které se nacházejí na 3 úrovních:

- *učitelé* – nedostatek času, nedostatečné znalosti ICT, negativní postoj k ICT
- *školy* – neinovativní přístup, zastaralé vybavení, špatná technická podpora
- *vnější faktory* – nedostatečné finance, nedostatek příkladů dobré praxe a metodických materiálů.

Pro překonání těchto bariér a naplnění vizí SDV2020 byly vydány opatření hlavních směrů intervence:

1. Zajistit nediskriminační přístup k digitálním vzdělávacím zdrojům
2. Zajistit podmínky pro rozvoj digitální gramotnosti a informatického myšlení žáků.
3. Zajistit podmínky pro rozvoj digitální gramotnosti a informatického myšlení učitelů.
4. Zajistit budování a obnovu vzdělávací infrastruktury.
5. Podpořit inovační postupy, sledování, hodnocení a šíření jejich výsledků.
6. Zajistit systém podporující rozvoj škol v oblasti integrace digitálních technologií do výuky a do života školy.
7. Zvýšit porozumění veřejnosti cílům a procesům integrace technologií do vzdělávání.

Z obsahu tohoto dokumentu je patrné, rozvoj *informatického myšlení* by měl hrát důležitou úlohu ve výuce informatiky. Tímto směrem se v posledních letech úspěšně ubírá výuka informatiky i v západních zemích Evropy jako např. ve Spojeném království, kde je ovšem na výuku informatiky kladen ještě větší důraz a výuka je podpořena také větší hodinovou dotací. (The Royal Society, 2012)

1.3 Informatické myšlení

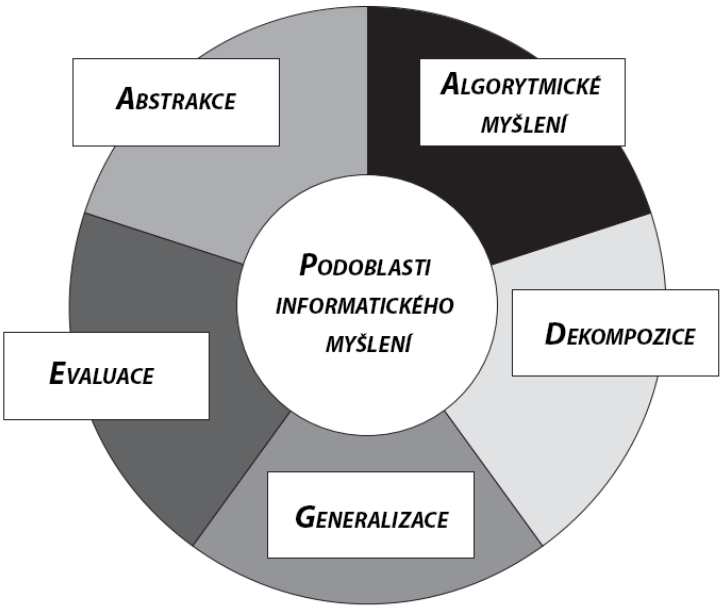
„Rozvoj a začlenění konceptu rozvoje informatického myšlení žáků do učebních osnov informatiky je v současnosti jednou z hlavních výzev, kterým český školní systém čelí.“ (Klement, 2020)

Informatické myšlení je termín, který je s výukou informatiky v posledních letech spojován stále častěji a stal se fenoménem v moderním pojetí vzdělávání. Informatické myšlení

je volným překladem z originálního názvu, který má původ v anglickém „*computational thinking*“ (dále jen CT).

Informatické myšlení je ve SDV2020 je definováno jako *„způsob uvažování, který používá informatické metody řešení problémů, a to včetně problémů komplexních či nejasně zadaných. Rozvíjí schopnost analyzovat a syntetizovat, zevšeobecňovat, hledat vhodné strategie řešení problémů a ověřovat je v praxi“*. Informatické myšlení by mělo vést k *„přesnému vyjadřování myšlenek a postupů a jejich zaznamenání ve formálních zápisech, které slouží jako všeobecný prostředek komunikace“*. V rámci informatického myšlení se můžeme setkat s univerzálními pojmy jako například *algoritmus, struktury, reprezentace informací, efektivita, modelování, informační systémy, principy fungování digitálních technologií* a podobně. (MŠMT, 2014)

Klement, Dragon a Brychová (2020) vymezují 5 základních podoblastí ze kterých se skládá informatické myšlení:



Obrázek 1: Podoblasti informatického myšlení
Zdroj: Klement, Dragon a Bryndová, 2020

- *Abstrakce* je klíčový prvek informatického myšlení. V rámci informatického myšlení chápeme abstrakci jako schopnost zjednodušit problém na jeho základy bez ztráty důležitých informací a následně pracovat se zjednodušenou

reprezentací problému. Zahrnuje práci s definováním vzorců, zobecněním, reprezentací, optimalizací a dalšími dovednostmi nezbytnými pro řešení komplexních problémů.

- *Algoritmické myšlení* je soubor schopností a dovedností najít a formulovat efektivní řešení určitého problému. V rámci informatického myšlení se názory jednotlivých autorů liší, zda algoritmické myšlení znamená také návrh řešení v určitém programovacím jazyce nebo zda je tento koncept zcela nezávislý na praktickém programování.
- *Dekompozice* je často spojovaná s modularizací. Jedná se o proces rozdělení celku na menší části, práce s těmito částmi a jejich optimalizace pomocí funkcí.
- *Evaluace* je systémové hodnocení, které obsahuje analýzu, ladění a umožňuje předpovídat výsledky a funkčnost algoritmů na základě kritického posouzení. K evaluaci je potřeba testování, analytické a logické myšlení.
- *Generalizace* neboli z*obecnění* je proces určení podobností a vazeb mezi problémy. Vede k návrhu univerzálních řešení, aplikovatelných v různých situacích.

Informatické myšlení v tomto pojetí tedy není spjaté a orientované pouze na digitální technologie, ale nese s sebou také celospolečenský přesah. Obsahuje celý soubor dovedností, předpokladů a postojů, které jsou výhodou pro uplatnění na trhu práce.

S rostoucím zájmem o zařazení informatického myšlení do výuky bylo potřeba také poskytnou oporu učitelům informatiky, což bylo jedním z důvodů vzniku projektu *„Podpora rozvoje informatického myšlení“*, zkráceně PRIM. Tento projekt je provozovatelem webové stránky *imysleni.cz*, kde jsou umístěny veřejně dostupné výukové materiály poskytované zdarma. Tyto materiály jsou orientované zejména na algoritmizaci a programování.

V lednu 2021 potom MŠMT vydalo nový RVP ZV v souladu se Strategií digitálního vzdělávání do roku 2020. Zásadní změnou prošla zejména vzdělávací oblast *Informační a komunikační technologie,* kterou lze najít nově pod kratším názvem *Informatika*.

1.4 Změna Rámcového vzdělávacího programu pro základní vzdělávání

Od 1. září 2021 vešel v platnost nový RVP ZV, který přináší do českého školství několik zásadních změn, které již školy mohou zařadit do svého ŠVP. Od 1. září 2023 je povinný pro 1. stupeň ZŠ a o rok později také pro 2. stupeň ZŠ. (Edu.cz, *n.d.*)

Tento RVP ZV přidává k původním šesti klíčovým kompetencím (kompetence k učení; kompetence k řešení problémů; kompetence komunikativní; kompetence sociální a personální; kompetence občanské; kompetence pracovní) novou kompetenci – *digitální*.

Dále nový RVP ZV upravuje jednotlivé vzdělávací oblasti, kde nejrozsáhlejší změnou prošla vzdělávací oblast *Informační a komunikační technologie*, jejíž obsah se do značné míry změnil a rozšířil. Kapitola dostala nový, kratší název – *Informatika*. Změnila se zásadně charakteristika vzdělávací oblasti, která hned v úvodu vymezuje směr, kterým se má výuka informatiky ubírat. RVP ZV uvádí, že *„vzdělávací oblast Informatika se zaměřuje především na rozvoj informatického myšlení a na porozumění základním principům digitálních technologií“* (MŠMT, 2021, 38). Předchozí RVP ZV přitom bylo zaměřeno především na získání základních dovedností s prací s moderními digitálními technologiemi a zpracování informací. Ve výsledku tedy představuje především práci s kancelářskými balíky, internetovými prohlížeči a grafickými programy.

Pro 2. stupeň ZŠ byly v RVP ZV z roku 2017 vymezeny pouze oblasti *Vyhledávání informací a komunikace* a *Zpracování a využití informací.* Nový RVP ZV vymezuje pro 1. a 2. stupeň ZŠ následující 4 vzdělávací oblasti:

- Data, informace a modelování
- *Algoritmizace a programování*
- Informační systémy
- Digitální technologie.

Jak již bylo zmíněno v přechozí kapitole, algoritmizace a programování jsou vhodným nástrojem právě pro rozvoj informatického myšlení. Jejich zařazení do RVP ZV je zárukou, že se s nimi setká opravdu každý žák základního vzdělávání, to ukazuje význam, jaký mu MŠMT přikládá.

2 Algoritmizace a programování

Algoritmizace a programování jsou dva rozdílné pojmy, které spolu úzce souvisí a v posledních letech se objevovaly ve spojení s výukou informatiky. Následující kapitola se věnuje těmto pojmům a tomu, jakým způsobem byly zařazeny do RVP ZV v oblasti Informatika.

2.1 Algoritmizace

Algoritmizace je proces vytváření algoritmů. Donald E. Knuth (1977) definuje algoritmus z hlediska počítačových věd jako *„soubor pravidel pro získání určitého výstupu z určitého vstupu. Každý krok musí být definován tak přesně, aby mohl být přeložen do počítačového jazyka a proveden strojem.“*

S algoritmy se však setkáváme i v běžném životě. Při jakékoliv činnosti, ke které je určitý návod, postup nebo předpis postupujeme podle algoritmu. Každý algoritmus zadává postup určitému procesoru. Slovo procesor pochází ještě z dob, kdy neexistovaly počítače. Procesor může být cokoliv (i člověk), co plní předepsané úkoly. (Balarinová, 2015, s. 6)

Výuka algoritmizace tedy nemusí být nutně omezena pouze na výuku s počítačem. Obzvláště učitelé prvního stupně ZŠ mohou využívat z hlediska výuky informatiky alternativní metody bez počítače. Mohou například využívat cvičení, při kterých žáci musí dodržovat přesně daný postup, který při určitých vstupech dovede žáky k přesně definovanému výstupu. Nebo naopak žáci musí definovat vstupy a postup, který má vést k určitému výstupu. Může se jednat například o recept na jídlo, specifické hračky, deskové hry nebo pracovní postup pro vytvoření výrobku.

Pšenčíková (2009, s. 7) definuje 6 podmínek, které musí každý počítačový algoritmus splňovat:

1) *Musí mít začátek a konec* – v algoritmu musí být zadáno, kdy má skončit. Pokud nemá zadaný konec, bude se opakovat stále dokola a program bychom museli vypnout ručně.
2) *Musí být věcně správný* – při zápisu algoritmu je důležité, aby každý vzorec byl zapsán správně v programovacím jazyku.
3) *Musí být jednoznačný* – v každém kroku musí být jasně definováno, jaký bude další krok.

4) *Musí být obecný* – algoritmus by měl sloužit pro co nejširší množství úloh.

5) *Musí být opakovatelný* – při zadání stejných vstupních podmínek musí dojít vždy ke stejnému výsledku.

6) *Musí být srozumitelný* – volit správné metody pro zápis algoritmů a používat komentáře pro vysvětlení. Tato podmínka je především pro další programátory, kteří budou pracovat s algoritmem a budou jej chtít případně upravovat.

Pokud má stroj vykonat vytvořený algoritmus, musí být algoritmus zapsán takovým způsobem, kterému procesor rozumí.

2.2 Programování

Programování je proces vytváření programů pomocí psaní algoritmů či instrukcí, které má počítač vykonat. Tyto zápisy lze psát v různých jazycích, jejichž zápis je počítač schopen přeložit. Takový jazyk nazýváme *programovací jazyk* a jeho zápis se nazývá *kód*, proto se lze setkat se slovem *kódování* jako synonymem ke slovu *programování*.

Kódování se však dle Feibela (1996) omezuje pouze na *„proces při kterém jsou informace v jedné formě anebo na jedné úrovni detailů reprezentované v jiné formě nebo na jiné úrovni“*, z čehož lze vyvodit, že kódování je spíše součást programování.

Programování dle Kernighana (2019, s. 93) není pouhé přepisování algoritmů, ale také hledání řešení praktických problémů jako je nedostatečná kapacita paměti, omezená rychlost procesoru, chybná, či dokonce záměrně škodlivá vstupní data, vadný hardware, přerušená síťová spojení nebo také lidská chyba.

Kód lze zapisovat v různých programovacích jazycích, je jich celá řada a stále vznikají nové. Každý z nich se vyznačuje určitými vlastnostmi a je vhodný pro jiné využití. Programovací jazyky lze rozdělit dle přístupu k procesu programování (tzv. programovací paradigmata) na *imperativní, deklarativní, funkcionální* a *objektově orientované*. Pokud jazyk podporuje více paradigmat, hovoříme o *multiparadigmatickém* jazyku. Dle míry abstrakce dělíme programovací jazyky na *nízkoúrovňové* a *vysokoúrovňové* (lze používat také kratší názvy – nižší a vyšší programovací jazyky).

Jako první vznikaly jazyky nízkoúrovňové. Tyto jazyky jsou blízko binárnímu kódu, kterým komunikuje hardware. Typickým představitelem nízkoúrovňového jazyka je *jazyk symbolických adres* označovaný také jako *assembler*. Tento jazyk je závislý na konkrétním typu

procesoru a jedna jeho instrukce obvykle odpovídá jedné instrukci procesoru, což s sebou přináší problém při přenosu programu z jednoho typu procesoru na jiný. Naučit se programovat v nižším programovacím jazyce je velmi náročné, protože tento kód je i pro zkušené programátory na první pohled nečitelný, a tak začaly vznikat vysokoúrovňové programovací jazyky. (Kernigham, 2019, s. 94-95)

Zápis operace ve vysokoúrovňovém programovacím jazyce je podobný tomu, jak ji vyjadřuje člověk v běžném jazyce. Naučit se programovat ve vyšším programovacím jazyce je oproti nižším jazykům výrazně rychlejší, stejně tak následný zápis programu je zpravidla kratší. Navíc program není závislý na žádné konkrétní procesorové architektuře a je tedy spustitelný na několika různých zařízeních. Mezi vysokoúrovňové jazyky řadíme například jazyk Java, C++, JavaScript, C#, Python a další. (Kernigham, 2019, s. 95-97)

Vedle konvenčních programovacích jazyků, které se programují pomocí psaní textu existuje specifická kategorie programovacích jazyků, a to jsou *vizuální programovací jazyky*. Tyto jazyky se nezakládají pouze na psaném zápisu, ale příkazy jsou sestaveny z určité grafické struktury, která může být doplněna textem či symboly. Tyto jazyky jsou vhodným vstupem do programování pro žáky základních škol. Vizuální programovací jazyky jsou velmi intuitivní a skládají se z již připravených příkazů, což je jejich hlavní přednost oproti standartním programovacím jazykům, kde všechny příkazy musí programátor znát nebo si je dohledat. Využití těchto vizuálních programovacích jazyků je vhodné zejména pro výuku na základních školách. Žáci se nemusí učit nazpaměť složitou syntaxi, která je odvozena od slov z anglického jazyka. Jednotlivé příkazy se navíc mohou lišit barvou či tvarem. Typickým zástupcem vizuálního programovacího jazyka je jazyk *Scratch*, který je oblíbený mezi učiteli základních škol.

Pro srovnání bude uveden zápis typického programu v různých programovacích jazycích, který vypíše na obrazovce větu *„Hello World!“*. V nízkoúrovňovém programovacím jazyce – *jazyce symbolických adres*, ve vysokoúrovňovém programovacím jazyce *Java*, a nakonec ve vizuálním programovacím jazyce *Scratch*.

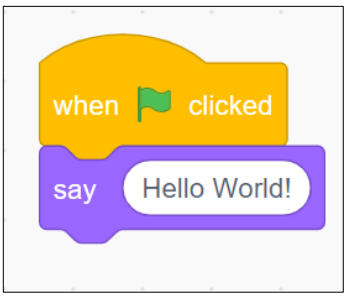
```
1.  // Program „Hello World!“ v jazyce symbolických adres
2.  bits 16
3.  org 0x0100
4.
5.  main:
6.  mov ax, cs
7.  mov ds, ax
8.
9.  mov si, zprava
10. call print_string
```

```
11.
12. ret
13.
14. print_string:
15. mov ah, 0x0e
16. xor bh, bh
17.
18. .print_char:
19. lodsb
20. or al, al
21. jz short .return
22. int 0x10
23. jmp short .print_char
24.
25. .return:
26. ret
27.
28. zprava db "Hello, World!", 0x00
```

Z uvedeného kódu je patrné, že jazyk symbolických adres je pro laika i začínajícího programátora nečitelný a u většiny řádků nemá čtenář představu o tom, co, který řádek znamená. V *jazyce symbolických adres* se najednou stává i triviální zadání poměrně složitým úkolem.

```
1.  // Program „Hello World!“ v jazyce Java
2.  class Main {
3.    public static void main(String[] args) {
4.      System.out.println("Hello world!");
5.    }
6.  }
```

Tento výše uvedený kód je napsán v objektově orientovaném jazyce *Java*. Pro samotný kód zde stačí pouze 6 řádků, které začínají úvodní poznámkou na prvním řádku (jejíž jedinou funkcí je lepší čitelnost kódu a procesor ji neprovádí) a dále definují třídu, metodu a obsah metody.



Obrázek 2: Program "Hello World!" v jazyce Scratch

Na Obrázku 1 je program *„Hello World“* vytvořen v blokovém programovacím jazyce *Scratch*. Na první pohled si můžeme všimnout barevného označení částí kódů. Velkou výhodou je, že kód obsahuje minimální množství znaků, a dokonce i laik, který nemá žádné předchozí zkušenosti s programováním dokáže pochopit průběh tohoto algoritmu. Další výhodou

prostředí Scratch je, že uživatel v podstatě nemůže udělat chybu v syntaxi, protože jediná část, do které může uživatel vepsat vlastní text jsou bílá políčka, v tomto případě obsahují text *„Hello World!“*.

Programovací jazyk je podobný tomu lidskému a stejně jako určitou větu lze říct v různých jazycích, dle toho, ke komu mluvíme, tak i stejný program můžeme naprogramovat v různých programovacích jazycích. Pro určitá zadání jsou více či méně vhodné určité programovací jazyky. Programátoři si vybírají programovací jazyk dle různých kritérií jako například na jaké platformě bude výsledný program pracovat, jaké funkce bude nabízet, kolik uživatelů jej bude používat atd.

Pro výběr programovacího jazyka při tvorbě seriózního softwaru je zásadní otázka *„Co budeme programovat?“*, ale školství je specifické prostředí, jehož cílem není dodávat software svým zákazníkům. Ve školství je také klíčová otázka *„Kdo bude programovat?“*. Stejně tak, jako používáme jiné výrazy a slovní spojení pro komunikaci s dětmi a pro komunikaci s dospělými, je výhodné tento princip použít také pro uvedení žáků do programování. Je důležité u dětí začínat s programovacími jazyky, které lépe odpovídají dětskému vnímání světa. Co taková výuka musí obsahovat nám předkládá MŠMT v dokumentu *Rámcový vzdělávací program pro základní vzdělávání* (RVP ZV), který je průběžně aktualizován.

2.4 Algoritmizace a programování jako součást RVP ZV

Jednotlivé vzdělávací oblasti RVP ZV jsou upravovány pomocí očekávaných výstupů žáka, obsahem učiva a minimální doporučenou úrovní pro úpravy očekávaných výstupů v rámci podpůrných opatření. Obsah těchto kategorií pro vzdělávací oblast algoritmizace a programování pro 2. stupeň je zpracován v tabulce 1.

Tabulka 1: RVP ZV pro 2. stupeň – algoritmizace a programování
Zdroj: http://www.nuv.cz/file/4982/.

žák	• po přečtení jednotlivých kroků algoritmu nebo programu vysvětlí celý postup; určí problém, který je daným algoritmem řešen • rozdělí problém na jednotlivě řešitelné části a navrhne a popíše kroky k jejich řešení • vybere z více možností vhodný algoritmus pro řešený problém a svůj výběr zdůvodní; upraví daný algoritmus pro jiné problémy, navrhne různé algoritmy pro řešení problému

	• v blokově orientovaném programovacím jazyce vytvoří přehledný program s ohledem na jeho možné důsledky a svou odpovědnost za ně; program vyzkouší a opraví v něm případné chyby; používá opakování, větvení programu, proměnné • ověří správnost postupu, najde a opraví v něm případnou chybu
učivo	• *algoritmizace*: dekompozice úlohy, problému; tvorba, zápis a přizpůsobení algoritmu • *programování*: nástroje programovacího prostředí, blokově orientovaný programovací jazyk, cykly, větvení, proměnné • *kontrola*: ověření algoritmu, programu (například změnou vstupů, kontrolou výstupů, opakovaným spuštěním); nalezení chyby (například krokováním); úprava algoritmu a programu • *tvorba digitálního obsahu*: tvorba programů (například příběhy, hry, simulace, roboti); potřeby uživatelů, uživatelské rozhraní programu; autorství a licence programu; etika programátora
Minimální doporučená úroveň pro úpravy očekávaných výstupů v rámci podpůrných opatření:	• *sestavuje symbolické zápisy postupů* • *popíše jednoduchý problém související s okruhem jeho zájmů a potřeb, navrhne a popíše podle předlohy jednotlivé kroky jeho řešení* • *rozpozná opakující se vzory, používá opakování známých postupů*

Z uvedené tabulky je patrné, že žákům nemusí být hned v úvodu předkládán proces vývoje programu a programování, ale *přečtení* algoritmu či programu a *vysvětlení celého postupu*. Dle tohoto konceptu mohou být žáci nejdříve postaveni před určitý problém a také před možné řešení tohoto problému. V rámci učiva by potom učitelé měli zařadit *dekompozici úlohy nebo problému*, to znamená rozklad algoritmu na jednotlivé části a podrobné vysvětlení jednotlivých částí kódu. Obzvláště po úvodních hodinách, kdy jsou již probírány složitější algoritmy, se můžou někteří žáci ztrácet v množství psaného kódu, který je také náročnější na pochopení. Pokud učitel rozdělí algoritmus na menší části, a jednotlivé části žákům podrobně

vysvětlí, bude pro žáky jednodušší poskládat jednotlivé části dohromady a pochopit fungování celého algoritmu. Po dostatečném úvodu by měli žáci zkoušet psát svoje příkazy a následně z nich tvořit jednoduché algoritmy a programy.

Mezi dalšími body očekávaných výstupů je *tvorba programu v blokově orientovaném programovacím jazyce.* Blokově orientovaný programovací jazyk je druh vizuálního programovacího jazyka. Příkazy nejsou zapisovány pouze písemným vyjádřením, ale pro tvorbu algoritmů se využívají tzv. bloky, které programátor skládá pod sebe nebo vedle sebe do výsledného programu. Bloky mají různé *tvary a barvy*. Tímto způsobem lze snadno odlišit jednotlivé bloky kódu dle své funkce. Mezi nejpoužívanější blokově orientované jazyky, jejichž součástí je také programovací prostředí, patří *Scratch* a *Lego Mindstorms* (pro Lego Mindstorms je však vyžadován další programovatelný hardware – programovatelná kostka EV3). Česká firma SGP group z Uherského Hradiště vyvinula také vizuální programovací jazyk pro děti pod jménem *Baltík.*

2.5 Další rozvoj algoritmizace a programování

S přibývajícím věkem žáků se vizuální programovací jazyky stávají poněkud infantilní a žáci začnou přirozeně toužit po seriózních programovacích nástrojích. U vizuálních programovacích jazyků také nastává problém při programování složitějších programů. Kód se může stát nepřehledným a špatně čitelným. Nepracujeme-li pečlivě, některé bloky se mohou překrývat, jak s nimi programátor manipuluje. Pokud se programátor dostane do tohoto bodu, měl by začít hledat řešení v seriózních programovacích jazycích založených na textovém programování.

Ve standartních programovacích jazycích jsou příkazy psány strukturovaně a s požadovaným odsazením dle pravidel daného jazyka. Pro zkušenější programátory je nepraktické v blokově orientovaném jazyce neustále hledat potřebné bloky a přetahovat je na obrazovku určenou pro programování. Pokud programátor zná klíčová slova, je rychlejší je jednoduše napsat a k napsanému kódu může připojit rovnou také komentář. Moderní vývojové prostředí (dále jen zkratka IDE z anglického *integrated development environment*) navíc zobrazuje programátorům nabídku klíčových slov v průběhu psaní kódu, upozorňuje na chyby v syntaxi a obsahuje další sadu nástrojů, které umožňují programátorovi příjemnější práci, a především zvyšuje produktivitu samotného programování.

Pokud tedy začne být programování v dětském programovacím jazyce limitující a žáci budou mít touhu dále rozvíjet svůj potenciál, škola by jim tuto možnost mohla poskytnout ať

už v rámci povinné výuky informatiky, volitelného předmětu nebo v rámci zájmového kroužku. Učitel by poté měl vybrat ten správný způsob výuky a také programovací jazyk, kterým bude žáky nadále rozvíjet.

RVP ZV ukládá základním školám pouze nutné minimum toho s čím by měli být žáci seznámeni během povinné školní docházky. Každá škola však dále zpracovává svůj Školní vzdělávací program (dále jen ŠVP), kde tyto minimální požadavky jednotlivých vzdělávacích oblastí mohou být rozšířeny. Díky tomuto systému mají školy určitý prostor pro úpravu probíraného učiva a náplně výuky. Vznikají tak základní školy či třídy na základních školách, které mají určité zaměření a nabízejí tak žákům či jejich rodičům možnost volby, kam chtějí směřovat své děti. Tímto způsobem vzniká mezi školami konkurenční prostředí, ze kterého benefitují především žáci. Typické jsou programy s rozšířenou výukou jazyků či matematiky, ale vznikají také programy s rozšířenou výukou informatiky. Právě třídy s rozšířenou výukou informatiky mají potenciál nabídnout žákům také programování v textovém programovacím jazyce nad rámec vizuálního blokově orientovaného jazyka.

2.6 Programovací jazyk pro výuku a stanovení kritérií

Programování je kreativní činnost, a vyžaduje plnou koncentraci programátora pro práci na stanoveném úkolu. Tato kapitola se zaměřuje na klíčové aspekty, které je třeba zvážit při rozhodování o tom, který programovací jazyk vybrat pro výuku programování na ZŠ. Výběr vhodného programovacího jazyka je kritickým krokem, který může mít značný vliv na počáteční zájem o informatiku a na rozvoj dovedností žáků v této oblasti. Programovací jazyk, který je přístupný, zábavný a vzdělávací, je také klíčovým prvkem pro úspěšnou integraci programování do školního vzdělávání.

Pecinovský (2004, s. 4) uvádí 7 kritérií, které by měl programovací jazyk využívaný ve výuce splňovat:

1. *Objektově orientovaný* – Ze čtyř hlavních programovacích paradigmat má objektově orientované programování (dále jen OOP) v současném softwarovém vývoji nejsilnější postavení (Brookshear, 2013, s. 254). Objektově orientovaný přístup je založen na vzájemné interakci mezi objekty. Objekty často reprezentují věci z reálného světa a metody připomínají způsob, jakým spolu objekty mezi sebou komunikují (Downey, 2015, s. 195). Nejde však o trend, kterým se ubírá pouze softwarové inženýrství, ale také výuka programování. Pecinovský (2004, s. 2) uvádí, že „*svět objektů, které si navzájem posílají zprávy,*

je pro žáky daleko pochopitelnější než svět posloupností a příkazů". Pokud se žáci naučí programovat a myslet objektovým způsobem, převezmou techniky a koncepty OOP, budou mít pevné základy pro využívání kterýchkoliv objektově orientovaných programovacích jazyků a přechod mezi těmito jazyky pro ně bude o to jednodušší (Kubrický a Klement, 2009, s. 137). Vizuální programovací jazyk Scratch také obsahuje prvky OOP a pro žáky, kteří v tomto jazyku programovali budou některé techniky již známé. (Vorderman, [2022])

2. *Měl by na svá bedra převzít co největší množství úkolů, které by programátor nemusel bezpodmínečně řešit* – Například správa paměti.

3. *Měl by být maximálně jednoduchý* – Tento bod v podstatě vylučuje nízkoúrovňové programovací jazyky, které mají dlouhý a složitý zápis. Zápis pro žáky by měl obsahovat co nejméně znaků a pravidel. Pokud žáci začínali s blokově orientovaným programovacím jazykem, ideálně by měl obsahovat stejná nebo alespoň podobná klíčová slova. Více slov, znaků a pravidel s sebou nese více možností pro chybný zápis, přestože algoritmus může být správný.

4. *Měl by být maximálně bezpečný a odhalit co nejvíce chyb již ve fázi překladu* – V soukromém i státním sektoru se jedná o jeden z nejdůležitějších kritérií. Při zajišťování kritické infrastruktury či práci s citlivými daty by měla být na bezpečnost kladen velký důraz.

5. *Měl by mít minimální nároky na hardware počítače* – Vzhledem k pokroku, kterým prošla výpočetní technika v posledních letech již tento bod není tak zásadní v rámci moderních stolních počítačů či laptopů. Navíc od žáků na základních školách se neočekává tvorba složitých a náročných programů. Větší význam má však u programování malých jednodeskových počítačů či mikrokontrolerů jakou je například platforma *Arduino*, které se většinou programují v jazyce C a C++.

6. *Překladače, vývojová prostředí a nástroje by měly být dostupné za minimální cenu, nejlépe zdarma* – Žáci budou procvičovat své dovednosti i mimo školní prostředí a musí mít tedy k dispozici nástroje, které budou moci používat na svých vlastních zařízeních a v domácích podmínkách. Národní program rozvoje vzdělávání (2001) se zavazuje k zajištění rovného přístupu ke vzdělávání a školy musí brát zřetel na sociální nerovnosti, které mohou mezi žáky panovat.

Momentálně je však k dispozici velké množství nástrojů, určených pro programování, které jsou dostupné zdarma v základních verzích. Zpoplatněny bývají zpravidla profesionální nástroje s pokročilými funkcemi, jejichž cílovou skupinou však nebývají žáci základních škol.

7. *Měl by být dostatečně rozšířený, aby bylo možno vyměňovat zkušenosti v rámci co nejširší komunity* – Dostatečné rozšíření jazyka může vzbudit v žácích větší zájem o tento jazyk, zejména, pokud učitel ukáže úspěšné projekty, které byly vybudovány pomocí tohoto jazyka a které žáci znají či používají. Větší rozšíření jazyka s sebou nese také širokou a aktivní komunitu programátorů. Tato skutečnost může také hrát významnou roli ve vývoji dítěte, jehož součástí je také socializace a začlenění do společnosti. Čáp a Mareš (2007, s. 54) uvádí, že *v socializaci probíhá vzájemné působení, „diskuse, konverzace" mezi jedincem a druhými lidmi i celou společností a její kulturou.* Žáci jsou v dnešním světě internetu obklopeni sociálními sítěmi a tato socializace tedy probíhá často také v online prostředí. Dle výzkumu Kopeckého a Szotkowskiho z roku 2019 používá asi 76 % dětí mezi 13–17 lety sociální sítě. Aktivní internetová komunita může pomoci žákům také v profesní socializaci. Žáci mohou v rámci internetové komunity pokládat dotazy a nacházet tak řešení svých problémů i mimo školní prostředí, což je dobrá zkušenost do profesního života. Umět správně a věcně položit dotaz je také jedna z kritických dovedností, které žáci budou využívat po celý život.

8. *Měla by k němu být k dispozici bohatá nabídka knihoven, nástrojů a volně stažitelných programů* – Knihovny mají významnou roli ve vývoji programů. Knihovna obsahuje již vytvořené procedury a funkce, které programátor může použít ve svém kódu. Práce s knihovnami je tedy velice efektivní a hraje důležitou roli při rozdělování práce u vývoje rozsáhlých projektů. Programovat v jazyce, který má k dispozici velké množství veřejně dostupných knihoven nebo knihoven vhodných pro určitý projekt je značnou výhodou.

Uvedená kritéria neobsahují striktní hranici mezi parametry, které jsou vyhovující a které již ne. Proto záleží především na osobním úsudku jednotlivých učitelů, kterým bodům budou přikládat jakou váhu.

K výše zmíněným bodům lze přiřadit ještě některé položky navíc s ohledem na současný vývoj výuky informatiky a pokrok v oblasti výpočetní techniky. Pro výběr programovacího jazyka můžou hrát významnou roli také následující kritéria:

- *Programování robotických stavebnic* – Možnost aplikovat programovací jazyk na výukovou robotickou stavebnici se jeví jako značná výhoda. Edukační robotika je jedním z nejslibnějších edukačních nástrojů ve výuce programování a často zmiňovaná v souvislosti s modernizací výuky na základních školách. Robotické stavebnice přináší značnou atraktivitu a vzbuzují u dětí zájem. Jsou účinným nástrojem pro vizualizaci probíraného učiva, kdy naprogramovaný kód je žák schopen ihned přenést do reálného světa (Klement, 2020). Dle McGill (2012) dokonce může robotika pomáhat dívkám a ženám získat sebedůvěru v oblasti programování, která je může vést k dalšímu rozvoji programovacích schopností.

- *Dostupnost studijních materiálů* – Učitelé potřebují odněkud získávat inspiraci pro výuku. Vytváření vlastních studijních materiálů vyžaduje spoustu energie a času. Proto by si měli učitelé při výběru programovacího jazyka pro své žáky udělat rešerši dostupných studijních materiálů. Pro podporu rozvoje informatického myšlení vznikl například web *imysleni.cz* na jehož obsahu se podílí všechny pedagogické fakulty ČR. Jsou zde umístěny podpůrné materiály jako učebnice či metodické listy pro učitele, které jsou určeny pro výuku algoritmizace a programování. Materiály jsou rozděleny dle jednotlivých ročníků od mateřský škol až po čtvrtý ročník středních škol.

Z vytyčených kritérií nelze vybrat jazyk, který by ve všech kritériích významně převyšoval ostatní jazyky. Můžeme však vybrat jazyky, které alespoň do určité míry splňují všechna kritéria – jedná se o jazyky Java, C#, Python a JavaScript. Tyto jazyky jsou světově velmi rozšířené a k vývoji softwaru je využívají jak malé rodinné firmy, tak velké nadnárodní korporace.

Nejlépe však vytyčená kritéria pro výuku splňuje jazyk Python. V následující kapitole bude představen programovací jazyk Python a argumenty, proč je právě Python vhodný pro výuku programování na ZŠ.

3 Python

Python začal vytvářet nizozemský programátor Guido van Rossum, který pracoval na vývoji programovacího jazyka ABC. Tento jazyk byl určen pro vzdělávací účely a měl především konkurovat jazykům BASIC a Pascal. Jazyk ABC si však mezi studenty nezískal přílišnou popularitu. Guido van Rossum se následně roku 1989 rozhodl, že svůj volný čas využije k tvorbě nového programovacího jazyka, a tak začal vytvářet jazyk Python. Název přitom nezvolil, jak se může zdát, dle anglického výrazu pro krajtu, nýbrž podle britské komediální skupiny *Monty Python*. Krajta se přesto stala oficiálním logem jazyka, respektive dvě krajty zapletené do sebe, viz následující obrázek. (Python Software Foundation, © 2001–2022)

Obrázek 3: Logo Python
Zdroj: https://www.python.org/

Držitelem autorských práv jazyka Python je nezávislá nezisková organizace *Python Software Foundation* (dále jen PSF). Kromě vývoje jazyka Python také podporuje mezinárodní komunitu uživatelů tohoto jazyka. (Python Software Foundation, © 2001-2022)

3.1 Python z hlediska stanovených kritérií

Jazyk Python odpovídá vytyčeným kritériím z předchozí kapitoly pro programovací jazyk využívaný ve výuce následovně:

- *Objektově orientovaný* – Python byl vyvíjen jako objektově-orientovaný programovací jazyk, podporuje ale také ostatní programovací paradigmata, což jej řadí mezi tzv. multiparadigmatické programovací jazyky.
- *Měl by na svá bedra převzít co největší množství úkolů, které by programátor nemusel bezpodmínečně řešit* – například správa paměti, bezpečností mechanismy či standartní knihovny. Tyto požadavky splňují rozšířené vysokoúrovňové programovací jazyky včetně jazyka Python.

- *Měl by být maximálně jednoduchý* – Tento jazyk je dostatečně jednoduchý a vhodný pro žáky základních škol. Například program *„Hello World“* z kapitoly *4.2 Programování* lze v Pythonu zapsat na jeden řádek následujícím způsobem:

```
1.  Print(„Hello World!“)
```

  Program v Pythonu lze většinou zapsat pomocí menšího množství znaků než při použití jiných objektově-orientovaných jazyků jako jsou například C# nebo Java. Díky tomu může být programátor produktivnější a trávit méně času samotným psaním kódu. Stručný kód je důležitý pro žáky, do jisté míry umožňuje lepší čitelnost, na kterou byl kladen při vývoji jazyka Python zvláštní důraz. Čitelnost je také zásadní vlastnost, kterou disponuje syntaxe jazyka Python. Python je interpretovaný programovací jazyk. Mezi výhody interpretovaných jazyků řadíme přenositelnost na jiné platformy a jednodušší debugging, což usnadňuje rychlý vývoj a testování. Běh programu je však na rozdíl od kompilovaných jazyků pomalejší, protože každá instrukce musí být zpracována interpretem. Tato interpretace také může vyžadovat více paměti a procesorového času.

- *Překladače, vývojová prostředí a nástroje by měly být dostupné za minimální cenu, nejlépe zdarma* – Základní vývojové prostředí je pro Python dostupné ihned po instalaci jazyka do počítače, protože je součástí instalačního balíčku. Jedná se o jednoduché vývojové prostředí IDLE (*Integrated Development and Learning Environment*, tedy *Integrované vývojové a výukové prostředí*), označované také jako *Python Shell*. Název IDLE vystihuje účel tohoto prostředí, který zahrnuje nejen vývoj, ale také učení a experimentování s jazykem Python. Tento editor umožňuje základní funkce jako je zvýrazňování syntaxe a odsazování kódu. Zahrnuje také interaktivní konzoli, která umožňuje okamžitě vykonávat příkazy v Pythonu a následně sledovat výsledky provedených příkazů. Dalším vhodným editorem pro začátečníky je *Mu*, který nabízí přehledné prostředí se základními funkcemi a již v základu podporu Pygame a připojení desky BBC micro:bit nebo raspberry Pi. Pro profesionální a velké projekty je zdarma k dispozici moderní komplexní IDE *PyCharm*. Další výhodou jazyka Python je velké množství online vývojových prostředí ve formě webových aplikací. Ke spuštění těchto aplikací je potřeba mít pouze

nainstalovaný webový prohlížeč a v něm mohou žáci začít okamžitě programovat v jazyce Python, aniž by museli něco instalovat, tyto vývojové prostředí ovšem většinou vyžadují registraci. Například online IDE *Repl.it* nabízí také rozšířené moduly pro jazyk Python, zmíníme například PyGame pro vývoj počítačových her. Další online IDE *EduBlocks.org* dokonce nabízí možnost vizuálního programování, kdy jednotlivé řádky kódu jsou vepsány do grafických bloků, které programátor skládá pomocí metody *drag-and-drop*.

- *Měl by být dostatečně rozšířený, aby bylo možno vyměňovat zkušenosti v rámci co nejširší komunity* – Python je jedním z nejpoužívanějších programovacích jazyků na světě. Dle statistik, které poskytuje webová stránka dotazovacích příspěvků pro vývojáře Stack Overflow, je Python druhým nejpoužívanějším jazykem na světě (pokud nepočítáme jazyky HTML/CSS, které se řadí mezi skriptovací jazyky) hned po jazyce JavaScript (Stack Overflow, 2023). Python je využíván mnohými světoznámými organizacemi pro vývoj vlastních aplikací. Mezi tyto organizace patří například Google, NASA, Disney, Mozilla, CERN a další organizace různých velikostí a všech sektorů ekonomiky (Tollervey, 2015). Python má také rozsáhlou komunitu aktivních uživatelů, o kterou pečuje především organizace *Python Software Foundation*, jež organizuje semináře a přednášky po celém světě, včetně ČR. Česká nezisková servisní organizace *Pyvec* sdružuje českou komunitu kolem programovacího jazyka Python. Pyvec zajišťuje setkávání české komunity mezi sebou i její kontakt s mezinárodní komunitou, tvorbu studijních materiálů a výuku programování, a také se podílí na celospolečenské zapojení žen do programátorské kultury skrze mezinárodní aktivitu *PyLadies* (Pyvec, 2023).

- *Měla by k němu být k dispozici bohatá nabídka knihoven, nástrojů a volně stažitelných programů* – Python nabízí spoustu standartních knihoven a další tisíce open-source knihoven třetích stran. Mimořádně rozsáhlý počet knihoven je jedním z hlavních důvodů, proč je jazyk Python tak populární nejen mezi programátory, ale také mezi vědci a studenty. (Deitel, 2020, s. 16-18).

- *Podpora jazyka na robotických stavebnicích* –Robotické stavebnice, které lze programovat jsou často programovatelné v Pythonu, jako příklad uvedeme stavebnice *LEGO MINDSTORMS EV3*. Python lze využít také na

mikrokontrolery *Arduino*, přestože jsou koncipovány pro jazyk Wiring respektive C/C++. Pro programování v jazyce Python jsou ovšem od svého vzniku koncipovány mikropočítače *Raspberry Pi* a také *BBC Micro:bit*, které vznikly v rámci projektů pro podporu výuky informatiky v UK. (Tollervey, 2017)

- *Dostupnost studijních materiálů* – Studijní materiály pro výuku jazyka Python jsou v češtině oproti ostatním programovacím jazykům na nadstandartní úrovni. Na webu *imysleni.cz* jsou k dispozici dvě učebnice ke stažení zdarma – *Programování v jazyce Python pro střední školy*, autoři jsou Blaho, Salaci a Šimandl. Další učebnicí je *Robotika pro střední školy: programujeme Micro:bit pomocí Pythonu* od autorů Pecha a Nováka. Obě učebnice navíc nabízejí metodické listy pro učitele. Mimo tyto materiále lze najít další internetové stránky zaměřené na programování pro děti, například *microla.cz, ucimesroboty.cz* a podobně.

3.2 Shrnutí

Python je objektově orientovaný programovací jazyk. Je to komplexní jazyk pro využití na robustních projektech velkých mezinárodních korporací, a přesto dostatečně jednoduchý pro využití ve výuce na základních a středních školách. Je dostupné rozsáhlé množství knihoven pro různé typy úkolů. Uživatelská komunita je rozsáhlá a aktivní jak ve světě, tak v České republice. Studijní materiály jsou v dostatečném množství v anglickém i v českém jazyce. V jazyce Python je možné programovat také populární robotické stavebnice.

Po osvojení programovacího jazyka mohou žáci tvořit vlastní software, na kterém mohou pracovat i ve skupinách. Práce na společném rozsáhlejším projektu však neznamená pouhé psaní kódu, ale žáci by měli získat představu o tom, jak vývoj softwaru probíhá, protože se nejedná pouze o psaní kódu.

4 Vývoj softwaru

Vývoj softwaru je proces, který se skládá z několika fází, které se hromadně označují jako softwarový životní cyklus z anglického *Software Development Life Cycle*, zkráceně SDLC. Každá z těchto částí hraje důležitou roli ve vývoji kvalitního, funkčního a uživatelsky přívětivého softwaru. Typické vývojové fáze moderního softwaru shrnuje Hossain (2023) následovně:

1. *Plánování*

Během plánování se definují díle projektu, rozsah práce a zdrojů. Vypracovává se časový plán a zvažují se technologické a organizační rizika.

2. *Analýza*

V této fázi se analyzují a dokumentují podrobné požadavky uživatelů a zúčastněných stran a hodnotí se jejich proveditelnost. Cílem je pochopit, které problémy má software řešit. Typicky se vypracovávají například diagramy případů užití.

3. *Návrh – Design*

Fáze designu zahrnuje tvorbu detailních plánů softwarové architektury, databází a bezpečnostních nebo integračních aspektů. Dále jsou vypracovány návrhy uživatelského rozhraní UI (user interface) a uživatelského prostředí UX (user experience). Také vyvíjí prototypy a modely, aby se ověřila funkčnost návrhu.

4. *Implementace – Programování*

V této fázi již programátoři píší a testují zdrojový kód, vychází přitom z designové dokumentace. Pokud je potřeba, může zahrnovat také integraci s jinými aplikacemi nebo systémy.

5. *Testování*

Při testování se kontroluje, zda software splňuje specifické požadavky a zda funguje bez chyb. Testování se provádí na různých úrovních například integrační testování, systémové testování nebo uživatelské akceptační testování tzv. UAT. Při testování se zjišťují chyby a následně se opravují. Cílem testování je zajistit, že software splňuje definované požadavky, že je spolehlivý a připravený na nasazení.

6. *Nasazení*

Po úspěšném testování se software nasazuje do provozního prostředí pro finální testování a ověření funkčnosti. Může zahrnovat konfiguraci serverů a databází, migraci dat, školení uživatelů a aktualizaci systémových konfigurací. Nasazení může být provedenou postupně nebo najednou, cílem je zpřístupnění softwaru pro užívání cílovými uživateli nebo zákazníky.

7. *Údržba a podpora*

Po nasazení softwaru je třeba jej nadále udržovat, opravovat chyby, zajišťovat bezpečnost a přizpůsobovat změnám dle požadavků uživatelů. Tato fáze zahrnuje také vydávání aktualizací a rozšiřování funkcí dle aktuálních požadavků.

V rámci Životního cyklu vývoje softwaru (SDLC) se setkáváme s různými přístupy a metodikami, které definují, jak by měl být software vyvíjen a spravován. SDLC poskytuje základní rámec pro organizaci a kontrolu procesů vývoje softwaru, ale samotná specifika těchto procesů mohou být řízena různými metodikami. Tyto metodiky nejenže formují specifické kroky a postupy v rámci SDLC, ale také odrážejí filozofii a kulturu organizace, která je implementuje.

Dvě nejznámější metodiky vývoje softwaru jsou *Vodopádový model* a *Agilní model.* V následujících podkapitolách se zaměříme na jejich základní charakteristiky, , jaké výhody a nevýhody přinášejí, a jak mohou být efektivně implementovány v rámci SDLC pro dosažení optimálních výsledků ve vývoji softwaru.

4.1 Vodopádový model

Vodopádový model je považován za klasickou nebo tradiční metodiku ve vývoji softwaru. Je to jeden z nejstarších modelů a je široce rozšířen zejména pro vládní zakázky. Název *„vodopádový“* odráží jeho lineární postup prací, připomínající vodopád, kde každá fáze přechází (přeteče) do fáze následující. Využívá sekvenčního přístupu, každá fáze musí být dokončena před začátkem následující fáze. Jednou dokončená fáze se již obvykle neopakuje. Vodopádový model vyžaduje rozsáhlou dokumentaci v každé fázi, což ulehčuje předání projektu mezi týmy. Tento model byl populární zejména v dobách, kdy byl vývoj softwaru méně komplexní a nepodléhal tolika změnám v krátkém časovém horizontu. (Mujumdar, 2012)

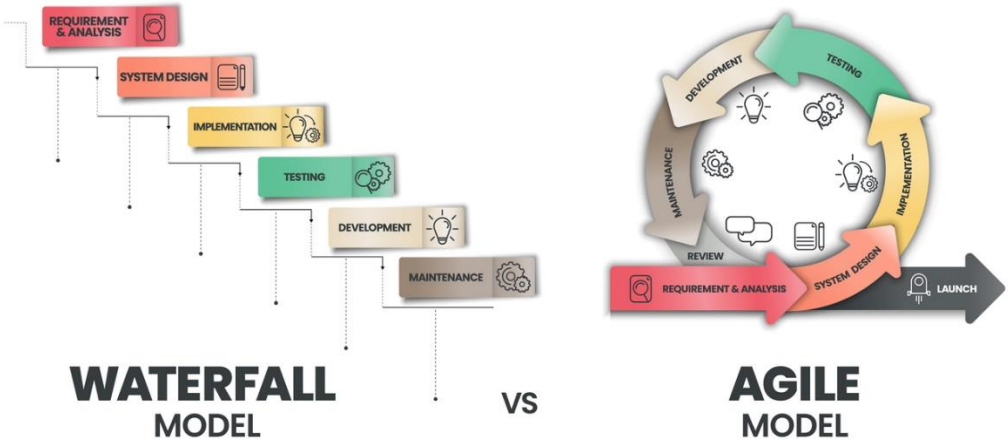
4.2 Agilní model

Agilní model představuje skupinu modernějších přístupů k vývoji softwaru, které se vyznačují společnými znaky. Agilní metodologie byla poprvé formalizována v roce 2001, kdy

skupina vývojářů vydala *Agilní manifest*, který formuluje základy agilního vývoje. Tento model nevyužívá lineárního postupu prací, ale projekty jsou rozděleny do krátkých iterací, které se označují jako *sprinty*. Sprint typicky trvá 1–4 týdny, ale může to být i déle a přináší vždy další vylepšení softwaru. V průběhu celého vývojového procesu klade agilní přístup důraz na spolupráci mezi jednotlivými vývojáři, testery a také zákazníky. Tato metodika je vhodná zejména pro projekty s rostoucími požadavky nebo pro projekty podléhající častým změnám. Týmy mohou tímto způsobem efektivně reagovat na měnící se priority. (Shore a Warden, 2007)

Průběžné zapojení zákazníků v průběhu práce na projektu je velkou výhodou této metodiky. Vývojářský tým tímto získá okamžitou zpětnou vazbu, na kterou může reagovat změnami ve vývoji. Zapojení zákazníků také zajišťuje, že konečný produkt bude lépe splňovat jejich potřeby a očekávání. Díky iterativnímu přístupu může být také funkční software dodán rychleji a také průběžně. (Mujumdar, 2012)

Nevýhodou agilních metodik je časová náročnost pro jednotlivé členy týmů z důvodu častých setkání a neustálé komunikace. Časté změny požadavků mohou vést také k neustálému přepracovávání projektu a mohou ovlivnit celkovou stabilitu a strukturu projektu. (Mujumdar, 2012)



Obrázek 4: Vodopádový a agilní přístup k vývoji softwaru
Zdroj: https://www.vecteezy.com/vector-art/8903063-agile-and-waterfall-are-two-distinctive-methodologies-of-processes-to-complete-projects-or-work-items-agile-incorporates-a-cyclic-but-the-waterfall-is-sequential-and-collaborative-process

4.3 Shrnutí

Vývoj softwaru nebo softwarové inženýrství je dynamické a komplexní odvětví, které se neustále vyvíjí a adaptuje na nové technologie a tržní požadavky a daleko přesahuje pouhé psaní kódu nebo programování. Zahrnuje celou několik fází jako jsou plánování, analýza požadavků, návrh, implementace, testování, nasazení, a ani zde ještě není vývoj ukončen, ale vstupuje do fáze údržby a podpory. Úspěšný vývoj softwaru vyžaduje rozsáhlou přípravu a pečlivé řízení. Vodopádový a agilní model představují dva rozdílné přístupy k řízení softwarových projektů, každý s vlastními silnými a slabými stránkami. Zatímco Vodopádový model poskytuje strukturovaný a fázový přístup, vhodný pro projekty s jasně definovanými požadavky, Agilní model nabízí flexibilitu a adaptibilitu vhodnou do rychle se měnícího technologického prostředí.

Jedním ze specifických odvětví softwarového vývoje je vývoj počítačových her, který je vhodný zařadit do výuky programování.

5 Počítačové hry

Hry jsou s námi už od nepaměti, dříve než první člověk. Pro děti je hra přirozenou součástí zdravého vývoje. V přírodě lze pozorovat toto chování u spousty jiných živočišných druhů, hraní si představuje pro živočichy a pro děti mimo jiné také přípravu na život a pomáhají jim rozvíjet fyzické i kognitivní schopnosti. Historie her je dlouhou cestou od primitivních aktivit, které pozorujeme také u zvířat, přes vznik strukturovaných her jako jsou například vrhcáby a šachy, až po moderní počítačové hry. (Huizinga, 1938)

Johan Huizinga dokonce ve svém díle *Homo ludens: o původu kultury ve hře* uvádí, že *Homo ludens* (z lat. hrající si člověk) by mohlo být označení lidského rodu lépe než *Homo sapiens* (z lat. Člověk rozumný). Huizinga (1938) dokonce uvádí, že *„liská civilizace nepřinesla k obecnému pojmu hry žádný podstatný znak. Zvířata si hrají právě tak jako lidé. Tím pádem fakt hry nemůže mít souvislost s racionalitou, protože rozumové založení by ji omezilo na svět lidí“*. Z uvedené definice vyplývá, že se jedná o primitivní hraní si bez pravidel, které můžeme pozorovat také u zvířat, které zejména napodobují chování dospělých jedinců.

V angličtině se setkáváme s pojmy *„play“* a *„game“*, které mají oba odlišný význam. Zatímco *„play“* zdůrazňuje proces a zážitek z činnosti, *„game“* klade důraz na strukturu, pravidla a často i soutěžní aspekt. V češtině se oba pojmy často zjednodušeně překládají jako *„hra“*, což může vést k záměně jejich specifických významů a nuancí. V této práci tedy budeme používat slovo *„hraní“* pro význam *„play“* a slovo *„hra“* pro význam *„game“*, případně anglické výrazy uvedeme explicitně.

Doklady o strukturovaných hrách (z angličtiny *games*) nalezené v archeologických vykopávkách dokazují, že touha po hře a soutěžení je hluboce zakořeněna v naší podstatě. Huizinga (1938) definuje hru jako *„dobrovolnou činnost, která je vykonávána uvnitř pevně stanovených časových a prostorových hranic, podle dobrovolně přijatých, ale bezpodmínečně závazných pravidel, která má svůj cíl v sobě samé a je doprovázena pocitem napětí a radosti a vědomím ‚jiného bytí‘, než je ‚všední život‘“*.

V českém školství mají hry dlouhou tradici, protože právě český pedagog Jan Amos Komenský ve své době prosazoval zásadu *Schola ludus* neboli *Škola hrou* (Schola ludus byl soubor divadelních her, které Komenský používal pro výuku latiny). Nebyl ovšem prvním, kdo se tématem her ve školství zabýval. Zabývali se jimi již starověcí filozofové jako například Platón, kterému je přisuzován výrok *„příteli, nezacházej s dětmi při učení násilně, nýbrž ať se učí formou hry, abys také lépe mohl pozorovat, k čemu se kdo svou přirozeností hodí“*.

Hogenová (2001) uvádí, že *„hra nás vytrhne z linie času… otevře se zde prostor významnosti, prostor porozumění, zvláštní herní světa a ten má vlastní časovost, která pohltí ty, jež si hrají“*. Dále zdůrazňuje svobodu, která je zakoušena v průběhu hry a prostor pro vlastní kreativitu. Právě tento zmíněný popis připomíná zážitek plynutí neboli *flow*. Csíkszentmihályi (2015) uvádí hru jako jednu z možností k dosažení zážitku flow.

5.1 Počítačová hra vymezení pojmu

Někteří lidé mají antipatii k počítačům, zatímco jiní je obdivují, ale vidí počítačové hry jako formu znehodnocení této technologie. Na druhé straně, existuje mnoho lidí, hlavně děti, pro které je kombinace počítačů a her přirozená a fascinující. Toto spojení je považováno za synergické, jako vzájemné nalezení dvou elementů, které dokonale spolupracují. Hravost získala prostřednictvím počítačů nevídané uplatnění a nové rozměry, počítače se také díky tomuto staly velmi hodnotnými nástroji. (Čeláková a Čelák, 1992)

Mrázek a Basler (2018, s. 11) definují počítačovou hru jako *„určitý typ počítačového programu (sofwaru), který je postaven na vzájemné interakci s uživatelem. Uživatelem rozumíme hráče počítačových her, kterému počítačové hry slouží především jako forma zábavy, odreagování, ale často mohou mít také formu vzdělávacího softwaru“*.

Význam pojmu *„počítačová hra“* se však v průběhu času měnil. Zpočátku byl totiž výrazný rozdíl v tom, pro které zařízení byla hra určená – zdali byla hra určená pro konzole a jednalo se o *„videohru“* případně *„digitální hru“* anebo se jednalo o hru určenou pro osobní počítače, tedy hru *„počítačovou“*. Technologický vývoj však tyto rozdíly téměř vymazal. Hry, které jsou dostupné na konzolích mají hráči dostupné také na osobních počítačích s menšími rozdíly, jako je například způsob ovládání. Uvedené názvy jsou dnes tedy vnímány většinou jako synonyma. (Basler a Dostál, 2016)

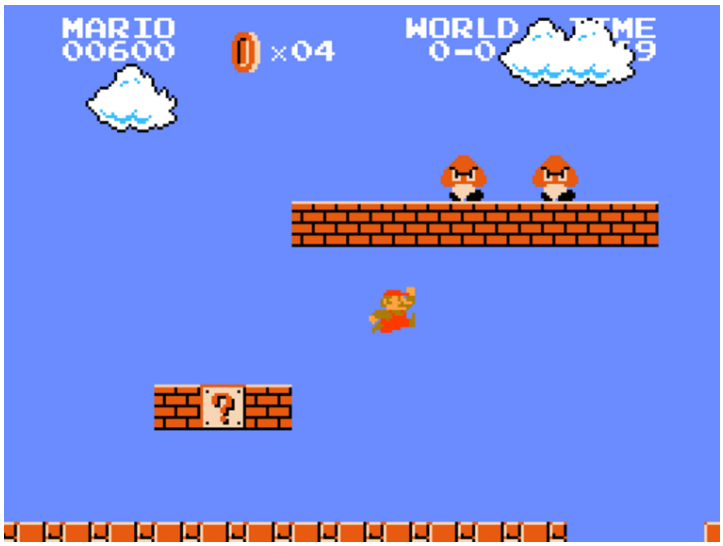
5.2 Historie počítavých her

V roce 1951 Ralph Baer předpověděl, že k pasivnímu sledování televizních pořadů jednou přibude i možnost interaktivního zapojení. Jeho myšlenky a představy o této technologii se brzy začaly realizovat. Roku 1958 William A. Higenbotham v amerických jaderných laboratořích Brookhaven National Laboratory připojil k počítači několik analogových páček a pro svou zábavu se pokusil na osciloskopu vytvořit simulaci letícího míčku. Později vytvořili programátoři na univerzitě MIT hru *Spacewar!*, která se stala populární také na ostatních univerzitách, ovšem zásadního komerčního úspěchu nedosáhla. Celosvětový úspěch slavila až

hra *PONG* od americké firmy Atari, která tuto hru uvedla roku 1972. Základ hry byl na pohybu míčku, který se odráží mezi dvěma pálkami. (Haratek, 2011)

Obrázek 5: Hra PONG na hracím automatu
Zdroj: https://commons.wikimedia.org/wiki/File:Pong_Arcade.jpg

Hra PONG byla také součástí prvního domácího videoherního zařízení *Odyssey*, jehož vývoj vedl Ralph Baer a zařízení vyráběla americká společnost Magnavox. Koncem 70. a začátkem 80. let je potom vznikaly populární tituly jako *Space Invaderes, Pac-Man* nebo *Donkey Kong*. Dalším významný milník přinesla japonská společnost Nintendo vydáním populární herní konzole NES (Nintendo Entertainment System) v roce 1985. Na této konzoli bylo možné si zahrát úspěšné herní série jako *Super Mario Bros., Metroid* nebo *Tetris*. (History.com, 2018)



Obrázek 6: Plošinová arkádová hra – Super Mario Bros.
Zdroj: https://www.nbcnews.com/tech/tech-news/program-plays-super-mario-bros-better-most-gamers-flna1c9327810

Ve druhé polovině osmdesátých let začala postupně nabývat na významu platforma osobních počítačů. Hlavním omezením pro jejich využití přímo pro videohry však byla vysoká pořizovací cena, která nedovolovala jejich použití výhradně pro tento účel. V té době byly tyto počítače často omezeny na práci s textem nebo disponovaly pouze základním grafickým režimem, což vedlo k převaze textových her. Vytvoření hry v této době bylo mnohem méně náročné než v současnosti, herní studia mohla vydat novou hru během 8 až 18 měsíců s týmy menšími než deset lidí, a to s celkovou prací méně než deset člověkoroků[1]. Naopak, moderní herní projekty mohou vyžadovat až stovky člověkoroků práce. (Haratek, 2011)

V 90. letech se videohry staly jedním z hlavních druhů zábavy, s dramaticky vyššími rozpočty než kdykoliv předtím, což bylo částečně díky rychlému technologickému pokroku. Tento pokrok umožnil hrám vypadat a znít realističtěji. V roce 1992 nastartovala hra *Dune II*, i když nebyla první ve svém žánru, vlnu popularity *real-time* strategií a předznamenala vznik titulů jako *Warcraft: Orcs & Humans* a *Command & Conquer*. Vedle vojenských strategií se rozvíjely i ekonomické strategie s hrami jako *SimCity* a *The Sims*, které otevřely cestu dalším simulátorům života a „tycoon“ hrám, včetně *Railroad Tycoon* zaměřeného na železnice. (Sláma, 2009)

[1] Člověkorok je jednotka pro měření objemu práce vykonané jednotlivcem během celého roku. Základní jednotkou je jedna člověkohodina, z kterém lze vypočítat, že jeden člověkorok je přibližně 2000 člověkohodin.

Obrázek 7: Realtime strategická hra – Warcraft: Orcs & Humans
Zdroj: https://en.wikipedia.org/wiki/Warcraft:_Orcs_%26_Humans#/media/File:Warcraft_Orcs_v_Humans_01.png

V roce 1996 byl uveden firmou *3dfx Interactive* chipset *Voodoo*, který se stal prvním široce dostupným 3D akcelerátorem pro osobní počítače, čímž umožnil prudký vzestup prvních střílečeka z pohledu první osoby (FPS – first-person shooter) na vrchol popularity mezi hráči. Po úspěchu *Doomu* z roku 1993 následovaly hity jako *Duke Nukem 3D* a zejména *Quake* v roce 1996. S nárůstem výpočetního výkonu se 3D grafika stala běžnou ve všech herních žánrech. Vznikaly nové, inovativní hry, jako *Thief: The Dark Project* z roku 1998, který přenesl stealth hry do první osoby, nebo *Grand Theft Auto III* z roku 2001, které uvedlo populární série *GTA* ve 3D. Některé klasické hry byly aktualizovány pro novou generaci s lepší grafikou, například *Prince of Persia: The Sands of Time* nebo *Broken Sword: The Sleeping Dragon*. (Sláma, 2009)

Obrázek 8: Doom - jedna z prvních 3D počítačových her
Zdroj: https://qz.com/1490069/doom-the-game-that-kicked-off-a-video-game-revolution-turns-25-today

Vývoj nových počítačových her vždy korespondoval s novými technologiemi. Jedním z přelomových momentů v přístupu k počítačům byl *rozvoj a dostupnost internetu*. Tento rozvoj v 90. letech umožnil expanzi *multiplayerových* her, jak v rámci již etablovaných žánrů, tak jako hlavní nabídku nových her. FPS hry zaměřené čistě na multiplayer, jako je *Counter-Strike* z roku 1999, modifikace *Half-Life*, nebo *Quake III Arena* z roku 2003, společně s MMOG (masivní multiplayerové online hry), jako je například *EverQuest* z roku 1999 nebo *World of Warcraft* z roku 2004, které nabídly nabídly hraní pro desítky až tisíce hráčů najednou. (Sláma, 2009)

Ve 21. století se rozšířily mobilní zařízení jako chytré telefony a tablety, které zpřístupnily hraní ještě širšímu publiku. Platformy od společností Apple a Google umožnily vývojářům snadno distribuovat hry přes obchody s aplikacemi, což s uvedením *Apple App Store* v roce 2008 znamenalo změnu v přístupu k tisícům her. Tento trend podpořil vzestup casual her (tituly pro příležitostné hráče) jako *Angry Birds* a *Candy Crush Saga*, a také otevřel dveře pro pokročilé herní zážitky s využitím rozšířené reality (AR – augmented reality) a virtuální reality (VR), což dokládají populární tituly jako *Pokémon Go* a zážitky nabízené VR brýlemi jako Oculus Rift. (Richards, 2023)

V současné době lze také očekávat rozšíření videoherního průmyslu také do smíšené reality (MR – mixed reality), která přidává ke skutečnému světu digitální vrstvu, ale nabízí také plnohodnotnou virtuální realitu. Tato technologie si získala značnou pozornost po uvedení brýlí *Apple Vision Pro* do prodeje. (Chlebek, 2024)

S postupujícím technologickým vývojem se očekává, že odvětví videoher bude nadále prosperovat. Nové cloudové herní platformy, jako je Google Stadia a Microsoft xCloud, umožňují uživatelům hrát hry na různých zařízeních bez potřeby investovat do drahého hardwarového vybavení. Tím se rozšiřuje dostupnost her pro širší okruh lidí a snižují se technologické bariéry. Další vývoj v oblasti umělé inteligence a strojového učení slibuje zásadní změny v herním průmyslu. Postavy řízené umělou inteligencí se budou schopné přizpůsobit herním strategiím hráčů, čímž výrazně zvýší úroveň ponoření do hry a vyvíjejí na hráče větší výzvy. Tyto inovace mohou znamenat průlom v oblasti realizace virtuálních světů. (Richards, 2023)

Historie videoher je příběh neustálých inovací a technického pokroku. Od jednodušších prvních her až po dnešní složité a zábavné hry. Videohry oslovují široké spektrum hráčů a mají vliv na populární kulturu. S výhledem do budoucnosti je zřejmé, že videohry budou pokračovat

v překračování stávajících limitů, nabízet zábavu a odreagování pro další generace. (Richards, 2023)

5.3 Vývoj počítačových her

Vývoj počítačových her je multidisciplinární průmysl, který spojuje umění, technologii, narativní design, a psychologii, aby vytvořil jedinečné zážitky pro hráče. Díky své komplexnosti tento sektor nabízí širokou škálu profesí, které se podílejí na tvorbě herních titulů. Každá z těchto rolí vyžaduje specifickou sadu dovedností a znalostí, společně však tvoří dynamický tým, jehož cílem je přinést na trh nové a inovativní hry.

Každý projekt je unikátní a přichází se svými vlastními výzvami, cíli, platformami a spousty dalšími vnitřními a vnějšími faktory. Role ve vývojových týmech se tedy liší (Wyman, 2011). Na většině herních projektů však můžeme najít následující role:

Game designer a level designer

Designér herního vývoje je osoba zodpovědná za vytvoření herního designu, což zahrnuje pravidla hry a všechny herní mechanismy, počínaje návrhem uživatelského rozhraní a konče fungováním AI. Obvykle se také podílí na celkovém vzhledu hry, včetně designu prostředí, scénářů pro jednotlivé úrovně a jejich map. Na základě jeho návrhů pak pracují level designéři, programátoři, animátoři a grafici. Designéři musí mít kompletní představu o hře ještě před jejím vznikem a být schopni přetvořit tuto vizi do detailně propracovaného dokumentu, který pokrývá všechny možné varianty vývoje hry. Dobrý designér by měl být také dobrým organizátorem a stratégem. Pro některé je tvorba hry přirozeným procesem, při kterém je pohání neustálá tvůrčí touha a nezbytným předpokladem je zde také kreativita. (Vávra, 2012)

Game designeři a level designeři jsou zodpovědní za bezprostřední zážitek z hraní hry. Jejich práce obvykle zahrnuje dokumentaci a někdy prototypy na začátku projektu. Jakmile jsou jednotlivé části hry vyvíjeny, přecházejí k testování hry a poskytují zpětnou vazbu a nápravná opatření výrobnímu týmu. Level designeři, kteří pracují na hře založené na úrovních, mohou programovat skripty, které řídí interaktivitu hry, například v případě střílečky z první osoby, skript úrovně může určovat počet a chování nepřátel, spouštěné události v úrovni a jakým způsobem se to děje. Tyto členy týmu přispívají k praktické realizaci hry. (Wyman, 2011)

Scénárista

Scenáristé se věnují psaní. Píšou pozadí příběhu, dialogy postav či návody. Hry mohou obsahovat velké množství textu nebo dialogů, a úkolem scenáristů je tento obsah vytvářet. S

rostoucí složitostí a detailností her se psaní stává čím dál tím důležitějším aspektem ve světě výroby videoher. Hry jsou čím dál častěji oceněny za kvalitu svého příběhu a psaní, a stále více her je adaptováno do velkofilmů, což je významné ocenění pro každého scenáristu hry. (Wyman, 2011)

Programátor

Programátoři vytváří kód, který je základem celého systému. Obvykle pracuje několik týmů programátorů, z nichž někteří se soustředí na hlavní herní engine (herní jádro), jiní na přenosy na specifické platformy, další na spojení s podkladovou technologií nebo službou a někteří se věnují konkrétním oblastem nebo funkcím ve hře. Programování je v konečném důsledku klíčové pro chování hry a pro zážitek, který hra poskytuje hráčům. (Wyman, 2011)

Grafik

Grafičtí designéři vytvářejí veškerý vizuální materiál použitý ve hře. To může zahrnovat jak 2D, tak 3D grafiku nebo jejich kombinace. Patří sem veškerá grafika ve hře, jako jsou postavy, prostředí, zbraně, lodě, auta a nábytek, stejně jako prvky prezentace včetně uživatelského rozhraní, displeje s informacemi pro hráče (HUD) a úvodní části hry. Tým grafiků rovněž vytváří úvodní obrazovky, neinteraktivní sekvence ve hře a filmové přechody. Často se od grafiků očekává, že budou vytvářet materiály nejen pro potřeby hry, ale i pro marketing. (Wyman, 2011)

UI designer

Uživatelské rozhraní (user interface – UI) slouží jako komunikační most mezi hrou a uživatelem, a tak představuje klíčovou bránu k dosažení stavu plynutí neboli „flow“. Komunikace mezi hrou a hráčem, či naopak, probíhá vždy prostřednictvím UI. Proto veškeré ovládání hry je součástí UI, včetně rozmanitých funkcí umožňujících interakci. Tyto funkce sahají od základních akcí jako je chůze, skok, dřep či zrychlení, až po jejich složitější kombinace, které hráč intuitivně vnímá jako projev svého úmyslu. Při jejich používání hráči často pociťují pohlcující chování, kdy nad nimi nepřemýšlejí a mají pocit, že ovládací prvky jsou přirozenou součástí jejich těla, čímž se jim dostává pocitu kontroly a moci. (Stefan Durmek, 2012)

Tester

Primární úlohou testerů je identifikace chyb v hře, aby je vývojáři mohli opravit v co nejkratším čase. Tyto chyby mohou být různorodé, od porušování fyzikálních zákonů ve hře,

přes postavy propadávající se do silnice, až po text přesahující okraje dialogových oken. Když tester při testování narazí na takovou chybu, vytvoří hlášení o chybě (bug report), což je dokumentace chyby určená vývojářům pro její následnou opravu. Kvalita testera se často odvíjí od kvality a přesnosti jeho hlášení o chybách. (Jirkovský, 2012)

Ostatní

K velkým herním projektům může patřit ještě další spousta profesí, které se specializují na další oblasti jako například:

- *Zvuk*: Tento tým má na starosti veškeré zvukové efekty a hudbu, která je pro hru skládána a nahrávána.
- *Marketing/PR*: Tento tým se zabývá pozicováním, propagací a prodejem hry.

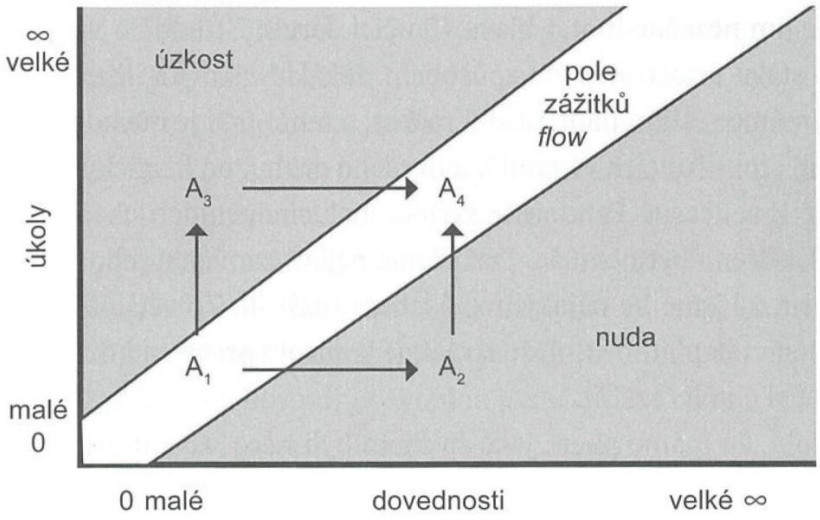
Matematika: Matematika je důležitou součástí game designu. Při navrhování her se vývojáři často setkávají s výpočty pravděpodobnosti, analýzami risku či tvorbou spravedlivých score systémů. (Wyman, 2011; Jirkovský, 2012)

V následující kapitole se budeme zabývat tím, jak by mohl vypadat vývoj počítačových her ve výuce informatiky na ZŠ s využitím jazyka Python a nástrojů, které jsou v současné době k dispozici.

6 Prostředky využitelné ve výuce pro programování hry a jazyk Python

Programování hry představuje nejen atraktivní, ale také efektivní přístup k výuce základů informatiky, neboť spojuje teorii s praktickým využitím. Studenti se nejen učí programovat, ale také rozvíjejí informatické a logické myšlení, schopnost řešit problémy a pracovat v týmu. Tato kapitola zabývá možnostmi vývoje jednoduché počítačové hry a jak podpořit informatické myšlení u žáků.

Při rozšiřování programátorských schopností roste také náročnost požadavků a učitelé mohou narazit na několik problémů. Učitelé mohou žáky postavit před příliš náročný úkol, který žáci nebudou schopni svými silami zvládnou nebo naopak představí žákům příliš triviální úkol, jehož řešení žáky nebude bavit. Abychom se vyhnuli tomuto problému, je žádoucí vyvolat v žácích tzv. zážitek *flow* který představil psycholog Michaly Csíkszentmihályi (2015). Ideálních podmínek pro dosažení zážitku *flow* dosáhneme tím, že žáky postavíme před úkol, který klade optimální nároky na dovednosti a zároveň představuje pro žáky určitou výzvu. Vztah mezi těmito proměnnými vyjadřuje graf z Obrázku 2. Z grafu je patrné, že pokud žáci stojí před úkolem, který je příliš jednoduchý v poměru k jejich schopnostem, zažívají nudu, stagnují a nerozvíjí svůj potenciál tak, jak by mohli. Pokud naopak stojí před úkolem, který je pro ně tak těžký, že přesahuje jejich schopnosti a dovednosti, mohou prožívat úzkost a ztrácet motivaci k další práci na zadání.



Obrázek 9: Podmínky pro dosažení stavu *flow*
Zdroj: Csíkszentmihályi (2015)

Jedním z požadavků, které by měli učitele klást pro svou výuku programování je tedy *dosažení zážitku flow* při práci na vývoji programu a programování.

Dále bude představeno několik řešení pro práci studentů s jazykem Python a také na vývoji počítačových her, které jsou obsaženy ve studijních materiálech.

6.1 EduBlocks

Dle RVP ZV je v rámci učiva pro 2. stupeň ZŠ zahrnut *blokově orientovaný programovací jazyk*. Na internetových stránkách *imysleni.cz* je k dispozici nejvíce výukových materiálů k blokovému programovacímu jazyku Scratch, který si získal značnou oblibu mezi učiteli i žáky. Mezi blokovým jazykem Scratch a textovým Pythonem je však stále velký rozdíl a přechod k programování v Pythonu je stále dost náročný.

Tento problém si uvědomil i mladý student Joshua Lowe, který začal již ve svých 12 letech pracovat na řešení, které by umožnilo plynulejší přechod z programování ve Scratchi k Pythonu. Začal tedy vyvíjet jednoduchou aplikaci určenou pro platformu Raspberry Pi, která umožnovala uživatelům ovládat LED diody a zvukové signály až postupně vznikl projekt *EduBlocks*. ((Hello World), 2020)

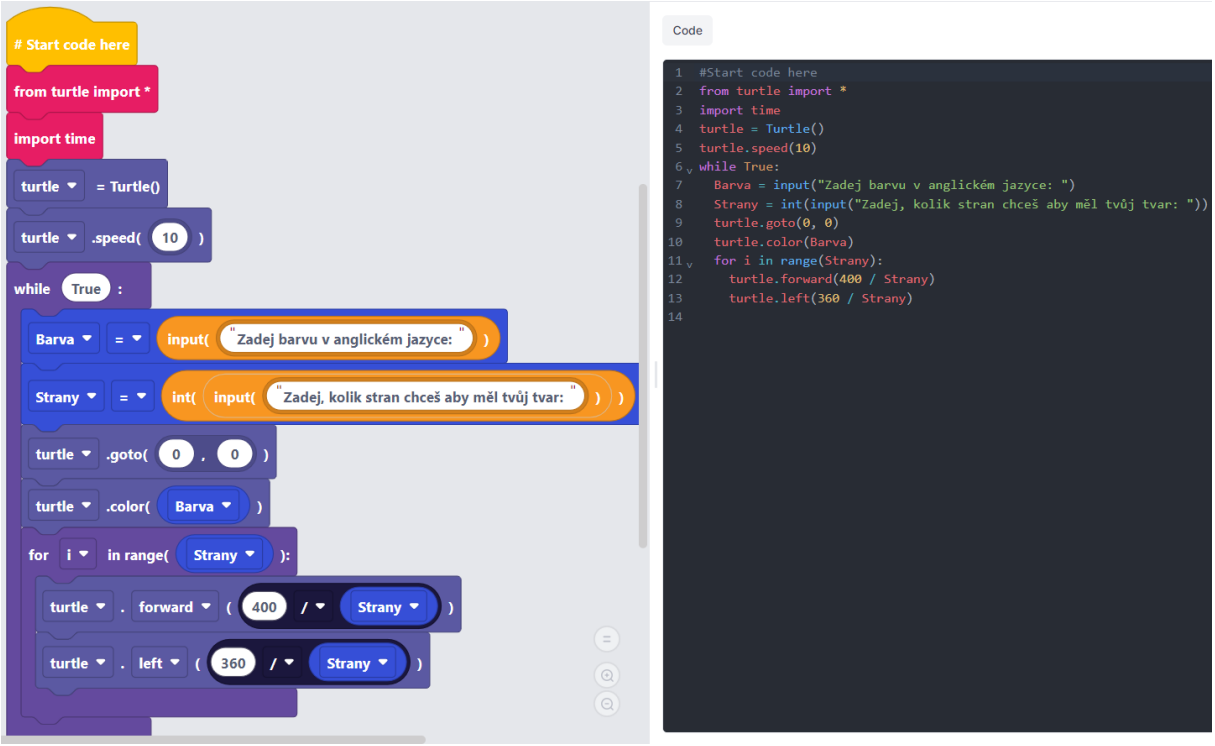
EduBlocks je online. Nabízí uživatelům drag-and-drop verzi jazyka Python. EduBlocks nabízí podobné prostředí jako Scratch a studentům tedy

EduBlocks představuje online vizuální programovací prostředí založené na Pythonu 3, které využívá metodu *drag-and-drop* a je dostupné zdarma. Tato platforma umožňuje studentům snadno se seznamovat s pravidly Pythonu při minimálním výskytu chyb, čímž zpřístupňuje Python i pro mladší začínající programátory. EduBlocks se podobá prostředí Scratch a žákům, kteří přecházejí z tohoto prostředí bude ovládání přirozené. Učitelé považují EduBlocks za užitečný nástroj pro vyplnění mezery, která studentům chybí při přechodu ze Scratche do Pythonu. Umožňuje jim jednoduše vysvětlovat základní programovací koncepty, jako jsou sekvenční zpracování, výběr a iterace, které jsou klíčové pro osvojení textového programování.

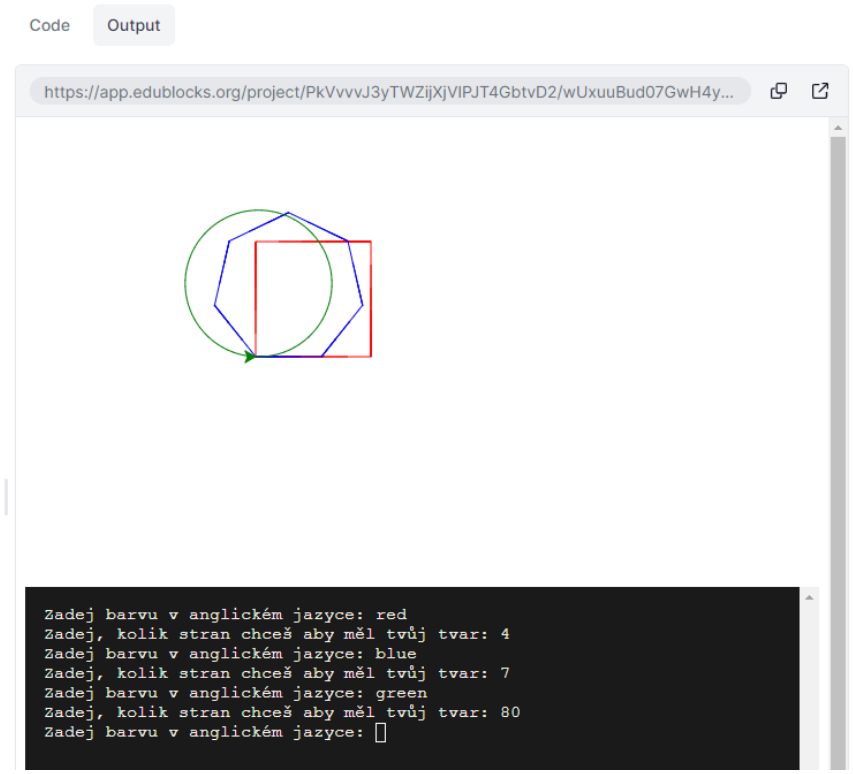
Cílem EduBlocks je usnadnit přechod od blokového programování k textovému programování pro studenty a učitele. EduBlocks toho dosahuje třemi hlavními způsoby:

1. *Zápis v Pythonu vepsaný v blocích* – jednou z unikátních vlastností EduBlocks je, že každý blok obsahuje řádek kódu v Pythonu, namísto jednoho slova nebo verbálního příkazu, jako v případě jazyka Scratch. To umožňuje studentům učit se snadno a rychle příkazy Pythonu již v prostředí blokového programování.
2. *Textový editor Pythonu* – v pravé polovině obrazovky je zobrazen textový kód Pythonu. Žáci si tedy mohou prohlížet kód sestavený v blocích také v textovém formátu Pythonu, čímž se jim znovu ukazuje spojení mezi blokovým kódem, který právě vytvořili, a tím, co by museli zadávat do textového editoru. Jakmile si studenti budou většími jistotami v používání Pythonu, mohou svůj kód v textovém pohledu upravovat. Tento princip funguje pouze jednosměrně – úpravy v blokovém editoru se v reálném čase propisují do textového editoru, ovšem z úpravy z textového editoru se neprojevují v blokovém editoru. Textový editor je tedy vhodné použít pouze v momentě, kdy již skončila práce s bloky.
3. *Zajímavé knihovny Pythonu* – Někteří studenti mohou považovat přechod z vizuálního prostředí, jako je Scratch, do textového prostředí, jako je Python, za nudný. Studenty tedy může zaujmout dostupnost knihoven Pythonu, jako jsou *Turtle* a *Processing*, aby studenti mohli vytvářet zábavné projekty podobné těm ve Scratchi, ale s použitím textově orientovaného jazyka. ((Hello World), 2020)

Želví grafika se stala populárním nástrojem ve výuce programování a je využívána také v jiných programovacích jazycích jako *Visual Basic* nebo *Scratch*. Možnost využívat tohoto nástroje v prostředí EduBlocks je dalším důležitým prvkem, který mohou učitelé se svými žáky využívat. Želví grafikou si žáci procvičují mimo programování také pojmy z matematiky, zejména práci s úhly, počítání vzdáleností, Pythagorovu větu či goniometrické funkce a mezipředmětový přesah je tedy patrný. V rámci programování si potom žáci mohou procvičit základní konstrukce jako proměnné, cykly a funkce. Další výhody želví grafiky jsou její intuitivnost – představu pohybující se želvy si lze snadno vyzkoušet na sobě, a obrázkový výstup – pro mnoho žáků může být zdrojem vnitřní motivace. (Pelánek, 2018)

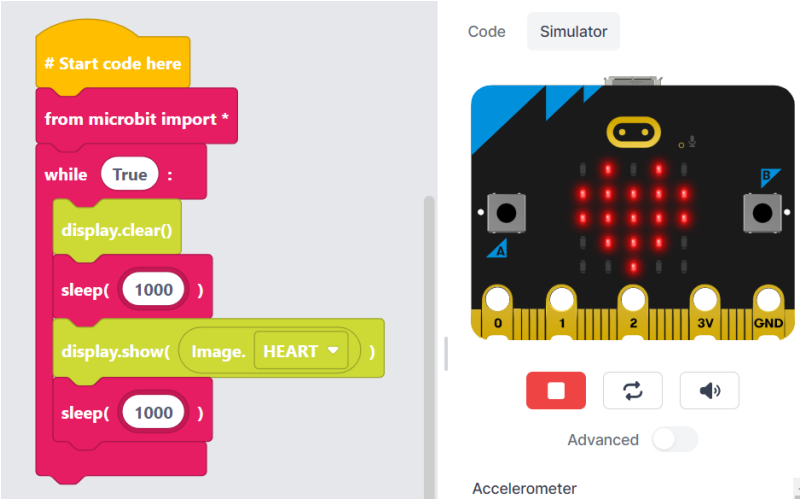


Obrázek 10: Zdrojový kód v prostředí EduBlocks využívající želví grafiku



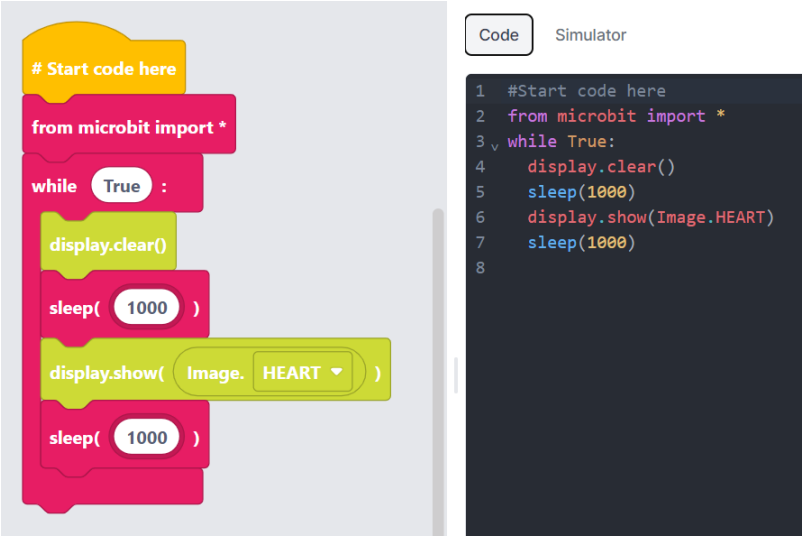
Obrázek 11: Program vytvořený s pomocí želví grafiky v prostředí EduBlocks

EduBlocks také nabízí programování externího hardweru. K dispozici jsou populární platformy *Raspberry Pi* a *Micro:bit*. Pro Micro:bit je dokonce k dispozici simulátor, takže programátor si může ověřit funkčnost svého kódu přímo v EduBlock, než kód nahraje do zařízení Micro:bit viz Obrázek 10.



Obrázek 12: Programování desky BBC Micro:bit pomocí EduBlocks

Obrázek 10 zobrazuje kód, který je napsaný pomocí blokového programování EduBlock a vedle simulátor ve tvaru desky BBC Micro:bit, který má dle kódu zobrazovat blikající srdce. Simulátor lze překliknout na textový editor viz Obrázek 13.



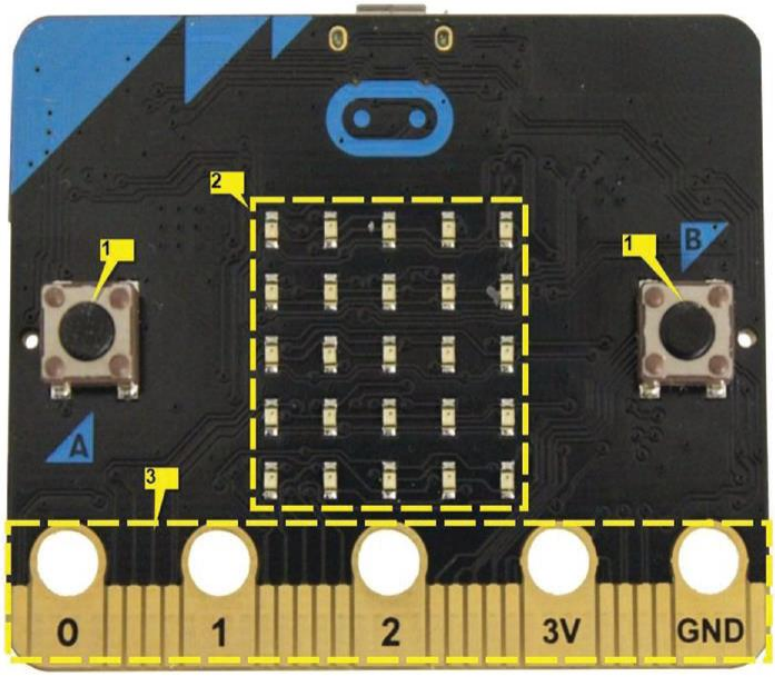
Obrázek 13: Programování desky BBC Micro:bit v jazyce Python pomocí textového editoru v EduBlocks

BBC Micro:bit je vhodná pomůcka pro výuku programování a tento mikropočítač bude blíže představen v následující kapitole spolu s programovacím prostředí *MakeCode*.

6.2 BBC micro:bit a MakeCode

V 80. letech 20. století představila britská společnost mikropočítač *BBC micro*, což byl počítač, který měl velký dopad na IT sektor ve Spojeném království, ze kterého tato země čerpá dodnes. BBC se rozhodla navázat na tento úspěch a představit nové zařízení, které bude mít podobný úspěch u dnešní generace mladých a proto se rozhodla roku 2012 založit *projekt BBC micro:bit*. Tento projekt vyvrcholil roku 2016, kdy spolu s partnerskými organizacemi (včetně ARM, Sansungu a Microsoftu) rozdala BBC milion těchto zařízení do škol ve Spojeném království. (Seneviratne, 2018)

BBC micro:bit je malé programovatelné zařízení. Jedná se o kombinaci velmi malého počítače a programovatelné desky. Micro:bit je velmi univerzální a navržený s ohledem na mladé uživatele, kteří nikdy předtím neprogramovali. (Seneviratne, 2018)

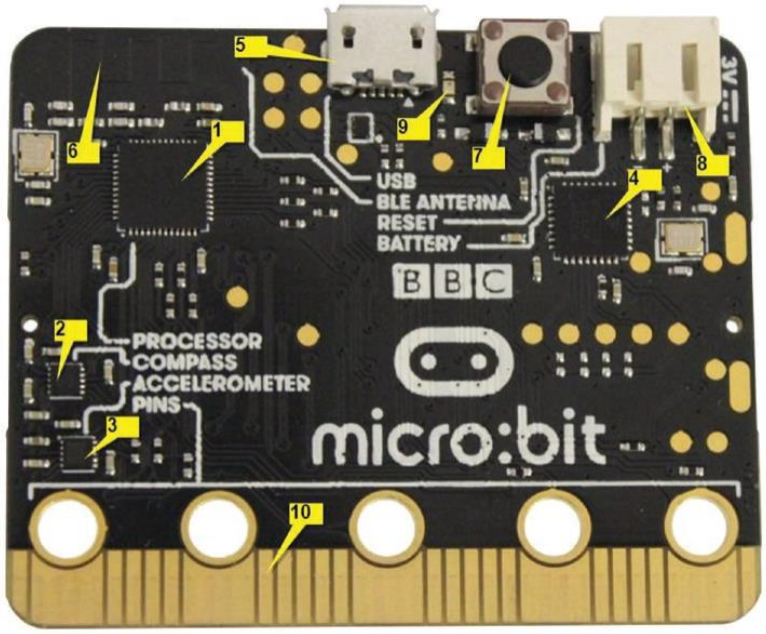


Obrázek 14: Přední strana desky BBC micro:bit
Zdroj: (Seneviratne, 2018)

Z přední strany desky se nacházejí následující prvky:

1. Tlačítka označené písmeny „A“ a „B“ na obou stranách desky
2. Displej umístění uprostřed desky o velikosti 5x5 červených LED diod

3. Konektor na spodní straně, který obsahuje celkem 25 pinů, umožňuje připojení různých senzorů, vstupně/výstupních linek, připojení napájení a uzemnění.



Obrázek 15: Zadní strana desky BBC micro:bit
Zdroj: (Seneviratne, 2018)

Ze zadní strany desky BBC micro:bit se nacházejí následující prvky:

1. Procesor ARM, který umožňuje také připojení zařízení přes Bluetooth
2. Kompas
3. Akcelerometr
4. USB kontroler
5. Micro USB konektor
6. Bluetooth anténa
7. Tlačítko RESET
8. Konektor pro připojení baterie
9. Systémová LED
10. Spodní konektor dostupný také z přední strany desky

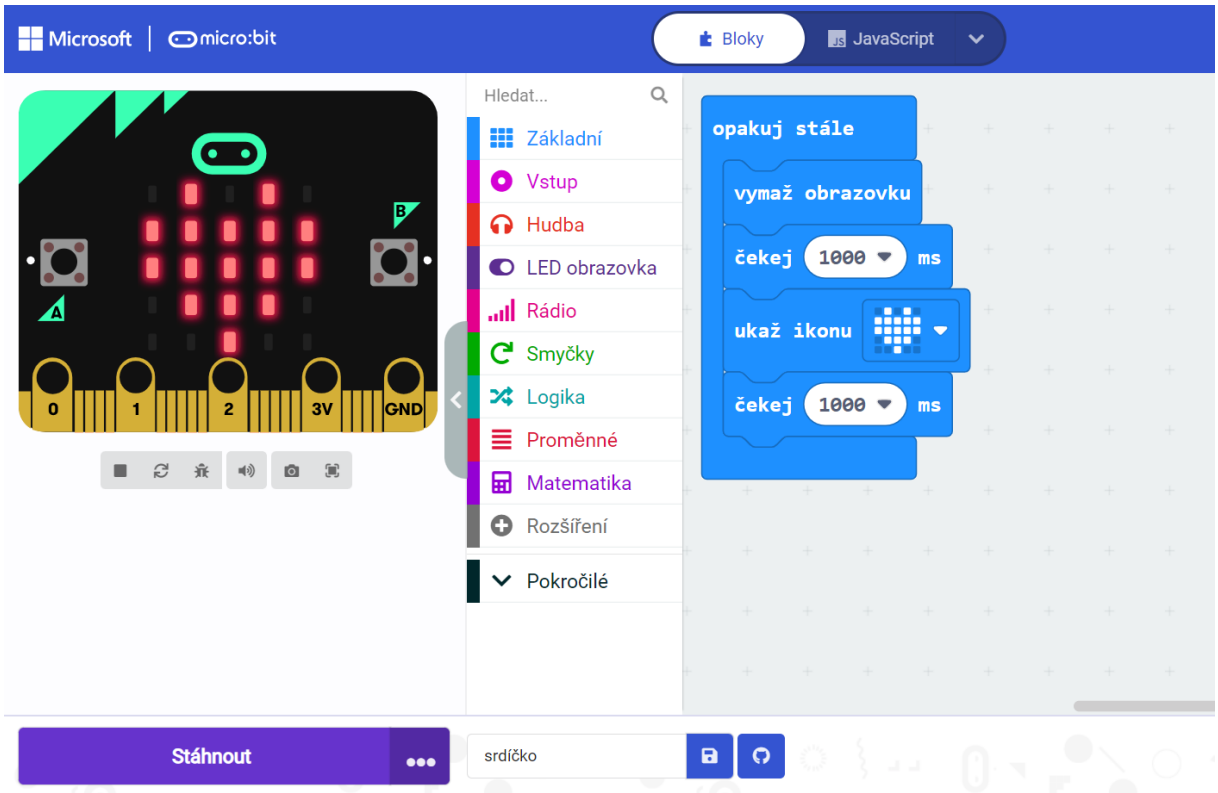
V současné době je na trhu dostupná nová verze této desky *BBC micro:bit V2*, která navíc obsahuje *reproduktor, mikrofon, vypínací tlačítko, logo citlivé na dotek, výkonnější procesor a také větší paměť.*

K BBC micro:bit je dostupné velké množství příslušenství a komponent, které rozšiřují jeho funkčnost a umožňují uživatelům experimentovat a realizovat rozmanité projekty. Mezi základní příslušenství patří rozšiřující desky (např. Kitronik nebo SparkFun), které poskytují dodatečné vstupy a výstupy, a umožňují tak snadné připojení senzorů, aktuátorů a dalších modulů. Dále jsou k dispozici speciální moduly pro rozšířené funkce, jako jsou motorové ovladače, robotické soupravy, přídavné displeje a zvukové moduly. Umožňují integraci s různými programovacími prostředími a podporují vzdělávací cíle v oblasti *STEM*. Díky modulárnímu designu a široké dostupnosti různého příslušenství lze micro:bit použít v nejrůznějších projektech, od jednoduchých učebních pomůcek po složité technologické kreace.

V roce 2020 vyšla nová verze BBC micro:bit V2, která je vzhledově podobná první desce a zachovala si všechny její funkcionality, ale nově nabízí reproduktor, mikrofon, vylepšený procesor a větší úložiště.

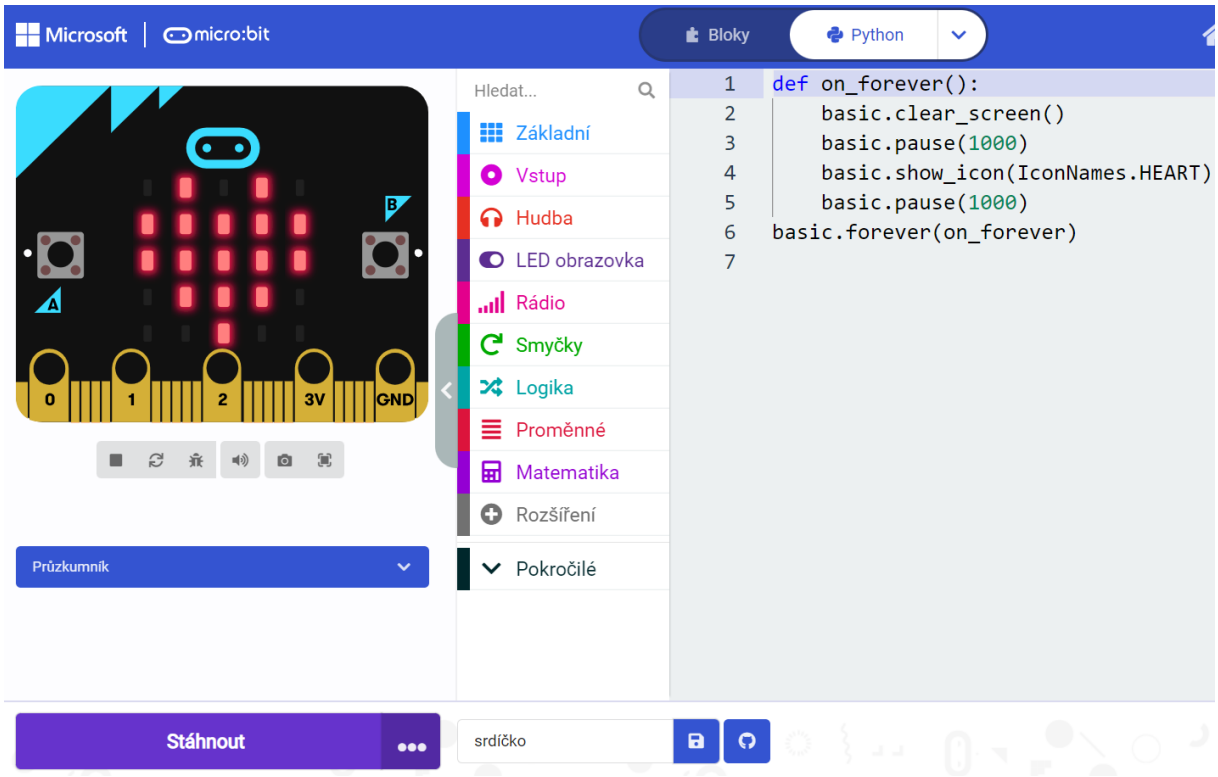
Pro programování BBC micro:bitu se nabízí několik možností. Velmi populární na základních školách je framework *MakeCode* od společnosti Microsoft. MakeCode je online editor a nabízí podobně jako Scratch nebo EduBlocks svůj grafický programovací jazyk založený na principu skládání bloků. Pro vytváření projektů si mohou uživatelé vytvořit svůj účet, ale není to podmínkou, projekty lze vytvářet i bez registrace a ukládat v paměti prohlížeče nebo stahovat přímo do vlastního úložiště jako soubor hex nebo nahrát přímo do zařízení micro:bit. (Seneviratne, 2019)

Výhodou prostředí MakeCode je jeho lokalizace do českého jazyka. Programování pomocí bloků lze tedy zvládat i bez základních znalostí anglického jazyka. Na úvodní obrazovce uživatelé naleznou množství tutoriálů, projektů, her, kurzů a pracovních listů.



Obrázek 16: Programovací prostředí Microsoft MakeCode – bloky

Programovací prostředí MakeCode viz Obrázek 16 má moderní design a plně obsahuje český jazyk. V levé části obrazovky se nachází simulátor, podobně jako tomu je v prostředí EduBlocks, kde si programátor může simulovat chod programu před nahráním na samotnou desku micro:bit. Uprostřed se nachází *paleta bloků*, které uživatel uchopí myší a přetáhne do pravé části obrazovky, kde je skládán samotný algoritmus programu. V horní části obrazovky může programátor přepínat mezi programovacími jazyky, k dispozici jsou bloky, JavaScript a Python viz Obrázek 17.



Obrázek 17: Programovací prostředí Microsoft MakeCode – Python

Na Obrázku 17 vidíme stejný program jako na Obrázku 16, ale napsaný v jazyce Python. Výhodou oproti prostředí EduBlock je, že úprava kódu v MakeCode funguje „obousměrmě“, tzn. kód, který programátor upraví v Pythonu se také propíše do prostředí s bloky a naopak. V režimu Python je v nabídce bloků pro vizuální programování zobrazena syntaxe Pythonu a krátké vysvětlení jednotlivých funkcí.

BBC micro:bit, s jeho hardwarovou flexibilitou a rozmanitým příslušenstvím ve spojení s vizuálním programovacím prostředím MakeCode doplněným o jazyky Python a JavaScript, umožňuje učitelům a žákům snadno přecházet od základních konceptů k praktickým aplikacím. Tato synergická kombinace podporuje interaktivní učení a experimentování a umožňuje uživatelům rychle převádět teoretické znalosti do praktických projektů a prototypů.

6.3 Dekompozice počítačové hry s moduly Pygame

Vývoj komplexních, seriózních počítačových her je vynikající způsob, jak si zpříjemnit programování, nicméně přináší to i své nevýhody. Hlavní nevýhodou je nutnost napsat rozsáhlé množství kódu, což vyžaduje hodně času. Vytvoření plně funkční hry úplně od základů často přesahuje rámec průvodce pro začátečníky v programování. (Watkiss, 2020)

Samotný vývoj počítačové hry je poměrně složitý proces, a proto se nabízí možnost předvést žákům funkční počítačovou hru, ukázat její zdrojový kód a následně provést tzv. *dekompozici* programu. Dekompozice je také jednou ze základních částí *informatického myšlení*, což bylo již představeno v kapitole *3.2.1 Informatické myšlení.*

Dekompozice je proces rozdělení problému na menší, snadněji zvládnutelné části. V programování to znamená rozdělení algoritmu na menší problémy, které lze řešit samostatně. U rozsáhlejších programů se problémy dále dekomponují, dokud není snadné určit, jak by každý mohl být zapsán jako samostatný podprogram v programu. To je užitečné pro vývojový tým, protože to znamená, že práci lze rozdělit mezi jednotlivými programátory. Díky rozdělení problému na menší sekce lze podprogramy znovu použít k řešení podobných problémů. (BBC, 2024)

Dekompozice je také dle RVP ZV součástí učiva, kdy v rámci *algoritmizace* by mělo učivo zahrnovat *„dekompozici úlohy, problému; tvorbu, zápis a přizpůsobení algoritmu“*. Z uvedeného jasně vyplývá, že žáci v rámci algoritmizace nemusí být schopni naprogramovat vlastní program tzv. na zelené louce, ale mohou vzít již existující program, v našem případě počítačovou hru, a upravit ji dle svých představ, přidat nové funkce a podobně.

Rich (2019) uvádí, dekompozice je společným krokem v mnoha procesech řešení problémů napříč různými obory, a přestože je dekompozice součástí také informatického myšlení, není jí věnována taková pozornost jako ostatním částem. Umět algoritmus či úlohu správně dekomponovat je přitom jedním z prvních kroků informatického myšlení a správná dekompozice může být rozhodujícím faktorem, zda rozvoj informatického myšlení bude nebo nebude efektivní. Před úspěšnou dekompozicí algoritmu zůstává problém pouze jako *„černá skříňka“*, což znamená, že vstupy a výstupy a vztah mezi nimi jsou sice známé, ale vnitřní fungování je neznámé. Dekompozice pomáhá řešiteli problému, aby lépe porozuměl vnitřnímu fungování pomocí rozložení problému na jednotlivé menší části problému.

Při dekompozici si žáci mohou osvojit jeden z důležitých aspektů programování, a to *psaní komentářů*. Anaya (2018) uvádí, že všeobecným pravidlem pro psaní čistého kódu je docílit co nejmenšího počtu komentářů. Samotný kód by měl být *„samodokumentující“* nebo *„samopopisující“*, to znamená, že už ze zápisu kódu by měl pojmenovávat věci přesně tak, aby komentáře nebyly potřeba, nebo pouze v minimálním množství.

V rámci výuky však může žákům pomoci právě větší množství komentářů pro pochopení vnitřního fungování kódu. Komentáře jsou také užitečným nástrojem při procesu dekompozice algoritmu, který si představíme na následujícím příkladu.

Pro ukázku jsme vybrali jednoduchou typickou hru pro začínající programátory, která je součástí různých učebnic nebo internetových tutoriálů. Tato hra nemá grafické prostředí a ovládání probíhá pouze pomocí příkazové řádky. Cílem hry je uhodnout náhodné číslo mezi 1 a 100. Hra po jednotlivých pokusech hráči dává nápovědu, zda je cílové číslo větší nebo menší než jeho tip. Po uhodnutí správného čísla se hráči zobrazí uhodnuté číslo a kolik pokusů potřeboval.

Kód bez poznámek a odsazení řádků by mohl vypadat následovně:

```
1. import random
2. def hadej_cislo():
3.     cislo = random.randint(1, 100)
4.     pokusy = 0
5.     print("Vítejte ve hře 'Hádej číslo'!")
6.     print("Myslím si číslo mezi 1 a 100.")
7.     while True:
8.         pokus = input("Hádejte, jaké číslo si myslím, nebo napište 'konec' pro ukončení hry:
")
9.         if pokus.lower() == 'konec':
10.            print("Hra ukončena. Skutečné číslo bylo:", cislo)
11.            break
12.        try:
13.            pokus = int(pokus)
14.        except ValueError:
15.            print("To nevypadá jako číslo. Zkuste to znovu.")
16.            continue
17.        pokusy += 1
18.        if pokus < cislo:
19.            print("Více!")
20.        elif pokus > cislo:
21.            print("Méně!")
22.        else:
23.            print(f"Gratulujeme! Uhádli jste číslo {cislo} na {pokusy} pokusů.")
24.            break
25. if __name__ == "__main__":
26.     hadej_cislo()
```

Tento kód je celkem triviální a obešel by se i bez komentářů, nicméně pro ukázku to může být pro žáky vhodné seznámení s dekompozicí a využití komentářů v kódu. Upravený kód by potom mohl vypadat následovně:

```
1. # Definice funkce hadej_cislo, která obsahuje logiku hry
2. def hadej_cislo():
3.     # Počítač si myslí náhodné číslo mezi 1 a 100
4.     cislo = random.randint(1, 100)
5.     # Nastavení proměnné pokusy na hodnotu 0
6.     pokusy = 0
7.
8.     # Uvítací zpráva pro hráče
9.     print("Vítejte ve hře 'Hádej číslo'!")
```

```
    print("Myslím si číslo mezi 1 a 100.")

    # Hlavní smyčka hry
    while True:
        # Získání vstupu od hráče
        pokus = input("Hádejte, jaké číslo si myslím, nebo napište 'konec' pro ukončení hry:
")


        # Možnost ukončení hry
        if pokus.lower() == 'konec':
            print(f"Hra ukončena. Skutečné číslo bylo: {cislo}")
            break

        # Kontrola, zda vstup je číslo
        try:
            pokus = int(pokus)
        except ValueError:
            print("To nevypadá jako číslo. Zkuste to znovu.")
            continue

        # Počítání pokusů
        pokusy += 1

        # Porovnání pokusu s tajným číslem
        if pokus < cislo:
            print("Více!")
        elif pokus > cislo:
            print("Méně!")
        else:
            # Hráč uhodl číslo, hra končí
            print(f"Gratulujeme! Uhádli jste číslo {cislo} na {pokusy} pokusů.")
            break

# Kontrola, zda je tento skript spuštěn přímo
if __name__ == "__main__":
    hadej_cislo()  # Spuštění hry

```

Větší rozdíl v čitelnosti kódu můžeme pozorovat u složitějších her, které již obsahují grafické prostředí. Tyto hry již žákům nabízejí spoustu možností pro úpravy a můžou tak aktivně pracovat nejen na dekompozici, ale také na upravování kódu. Jako příklad uvedeme jednoduchou grafickou hru, pro kterou použijeme sadu modulů *Pygame*, které umožňují tvorbu jednoduchých grafických her.

V této hře hráč pohybuje bílým čtvercem v horizontálním směru a pokouší se vyhýbat červeným čtvercům, které „padají“ z horního okraje obrazovky kolmo dolů. Po kolizi čtverců je hra ukončena. Zdrojový kód včetně komentářů vypadá následovně:

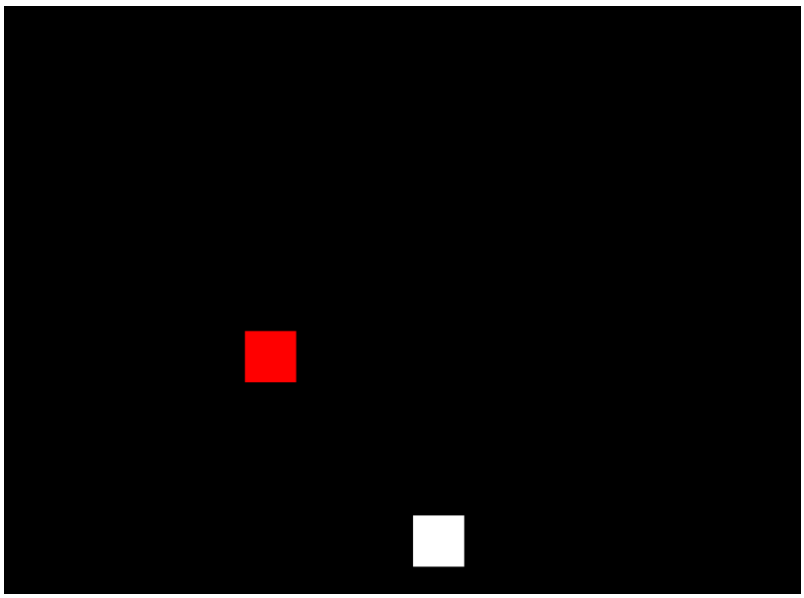
```
import pygame
import random
import sys

# Inicializace Pygame
pygame.init()

# Nastavení obrazovky
screen_width = 800
screen_height = 600
screen = pygame.display.set_mode((screen_width, screen_height))
```

```
12.
13. # Barvy
14. black = (0, 0, 0)
15. white = (255, 255, 255)
16. red = (255, 0, 0)
17.
18. # FPS (Frames per second)
19. clock = pygame.time.Clock()
20. fps = 60
21.
22. # Hráč
23. player_size = 50
24. player_x = screen_width / 2
25. player_y = screen_height - 2 * player_size
26. player_speed = 5
27.
28. # Nepřítel
29. enemy_size = 50
30. enemy_x = random.randint(0, screen_width - enemy_size)
31. enemy_y = 0
32. enemy_speed = 5
33.
34. # Funkce pro vykreslení nepřítele
35. def draw_enemy(x, y):
36.     pygame.draw.rect(screen, red, (x, y, enemy_size, enemy_size))
37.
38. # Funkce pro vykreslení hráče
39. def draw_player(x, y):
40.     pygame.draw.rect(screen, white, (x, y, player_size, player_size))
41.
42. # Hlavní smyčka hry
43. running = True
44. while running:
45.     # Kontrola událostí
46.     for event in pygame.event.get():
47.         if event.type == pygame.QUIT:
48.             running = False
49.
50.     # Ovládání hráče
51.     keys = pygame.key.get_pressed()
52.     if keys[pygame.K_LEFT] and player_x > 0:
53.         player_x -= player_speed
54.     if keys[pygame.K_RIGHT] and player_x < screen_width - player_size:
55.         player_x += player_speed
56.
57.     screen.fill(black)
58.
59.     # Pohyb nepřítele
60.     enemy_y += enemy_speed
61.     if enemy_y > screen_height:
62.         enemy_y = 0
63.         enemy_x = random.randint(0, screen_width - enemy_size)
64.
65.     draw_enemy(enemy_x, enemy_y)
66.     draw_player(player_x, player_y)
67.
68.     # Detekce kolize
69.     if enemy_y > player_y and enemy_y < player_y + player_size or player_y > enemy_y and
player_y < enemy_y + enemy_size:
70.         if enemy_x > player_x and enemy_x < player_x + player_size or player_x > enemy_x and
player_x < enemy_x + enemy_size:
71.             running = False
72.
73.     pygame.display.flip()
74.     clock.tick(fps)
75.
76. pygame.quit()
77. sys.exit()
78.
```

Grafický výstup hry je velice jednoduchý, viz Obrázek 18.



Obrázek 18: Grafický výstup hry pro vyhýbání se kolizi

Ačkoliv se úprava programu zdá jednoduchá, učitel by měl být přítomen při tomto procesu, aby mohl žákům poskytnou svůj názor a pomáhat jim s návrhy na vylepšení hry. Nabízí se například přidání úrovní obtížností, změna vizuálních parametrů, počítadlo skóre a podobně.

6.4 Projektové vyučování

Projektová výuka je běžně spojována s reformním pedagogickým hnutím na konci 19. a počátku 20. stolení. Z tohoto období jsou záznamy o využívání projektů v USA zejména v podobě domácích projektů, jejich smyslem bylo poznatky nabyté ve škole využít v domácnosti, na farmě nebo na zahradě. (Kratochvílová, 2006)

Za tvůrce teoretického rámce projektové metody je považován americký filozof a pedagog John Dewey. Ve svém pedagogickém systému spojoval psychologické a sociologické hledisko. Přizpůsoboval obsah a metody školní práce psychice dítěte a zároveň kladl důraz na uvádění žáků do zkušeností společnosti a do forem jejího života. Doweye uvádí, že *„gram zkušenosti je lepší než tuna teorie proto, že teorie má životný a ověřitelný význam jedině ve zkušenosti. Myšlení začíná tam, kde vznikají nějaké nesnáze.“* Klade důraz na výběr učiva a volbu metod použitých ve výuce, které mají zásadní vliv na přítomném duševním zájmu

a touze se učit. Uvádí, že *„žák má skutečný a pravý stav zkušenosti na základě zájmu o určitou činnost“*. (Kratochvílová, 2006)

V našich podmínkách se projektová výuka dostala do popředí teprve po změně politických a společenských poměrů v roce 1989. Nové tendence ve společnosti umožnily zásadní transformaci školství a otevřely tak cestu k novým metodám ve výuce. Díky aktivitě učitelů se v 90. letech začala projektová výuka uplatňovat také v českých školách. (Kratochvílová, 2006)

V období rozvoje internetu a globalizace došlo k rozšíření projektového vyučování, což bylo umožněno díky lepší dostupnosti rozsáhlého množství informací. Tato situace přináší nové požadavky na vzdělávací proces a poskytuje ideální podmínky pro implementaci projektového vyučování. Projektová výuka je navíc schopna pokrýt všechny čtyři základní pilíře vzdělávání, které definovala mezinárodní komise UNESCO ve své zprávě z roku 1997 nazvané "Učení je skryté bohatství". (Coufalová, 2006)

Čtyři základní pilíře vzdělávání dle UNESCO, které rozvíjí projektová výuka jsou:

- Učit se poznávat
- Učit se jednat
- Učit se žít společně
- Učit se být

Průcha (2013) definuje projektovou metodu, která je základem pro projektové vyučování jako *„vyučovací metodu, v níž jsou žáci vedeni k samostatnému zpracování určitých témat a získávání zkušenosti praktickou činností a experimentováním“*.

Samotný termín *„projekt"* se potom obvykle používá jako ekvivalent pro pojem design v kontextu technologie, techniky a konstrukce. Také se často setkáváme s pojmem „projekt“ ve významu dokumentace, ať už projektové či jiného typu. (Doležal, 2009)

Dle standartu ISO 10006 lze projekt definovat jako „*Unikátní proces koordinovaných a řízených činností s daty zahájení a ukončení, které je třeba provést k dosažení stanoveného cíle, jež vyhovuje specifickým požadavkům, včetně omezení daných časem, náklady a zdroji.“*

Tomková (2009) vyzdvihuje přínosy projektové výuky jako účinný nástroj proti stereotypu, rutině či všednosti, přestože vyžaduje velké úsilí učitele. Projektová výuka by měla vždy směřovat ke konečnému produktu. Tomková uvádí 3 základní znaky projektové výuky:

- Odpovědnost za vlastní učení
- Samostatné objevování poznatků
- Žákovo úsilí o dosažení cíle

Samotný produkt by měl motivovat žáka k činnosti a řídit její průběh, vnitřní motivace je důležitou podmínkou projektového vyučování. Učitel vybírá témata, která jsou žákům blízká. Na projektech obvykle pracují žáci delší období a projekty mohou být založeny na individuální nebo skupinové práci. Projektová výuka potom prochází následujícími fázemi:

1. Motivace, mapování, třídění
2. Řešení
3. Produkt projektu
4. Reflexe (Tomková, 2009)

Tomková zároveň upozorňuje na problémy, které s sebou nese zavádění individualizované a skupinové práce do vyučování ve školách, kde se tyto principy standartně neuplatňují. Proto uvádí následující doporučení pro učitele, kteří nově zavádějí projektovou výuku do škol:

- Změnit vztah učitele a žáka na vztah partnerský
- Učit děti potřebným dovednostem a samostatnosti
- Naučit děti učit se v souvislostech
- Vybavit třídu, školu dostatkem informačních zdrojů
- Otevírat školu a třídu dalším zdrojům učení
- Stupňovat náročnost projektů
- Vyučovací postupy volit s rozmyslem

Projektová výuka ve výuce informatiky

Projektové vyučování klade důraz na studentovo aktivní zapojení a praktické využití získaných znalostí. Vyžaduje od učitelů informatiky nejen hluboké porozumění obsahu, ale také schopnost flexibilně reagovat na dynamické technologické prostředí. Přestože implementace projektového vyučování může být náročná, zejména kvůli potřebě integrace aktuálních technologií a softwarových nástrojů, přínosy tohoto přístupu jsou neocenitelné. Studenti nejenže lépe absorbují komplexní koncepty, ale zároveň rozvíjejí klíčové kompetence jako jsou týmová spolupráce, kritické myšlení a schopnost řešit praktické problémy. Přes veškeré výzvy, které tento krok přináší, výsledky mohou výrazně obohatit výukový proces a připravit studenty na úspěšnou kariéru v digitálním věku.

Lee Sheldon ve své publikaci z roku 2020 *The Multiplayer Classroom: Designing Coursework as a Game* doporučuje vyučujícím kurz založen na projektové práci. Prostředím kurzu je imaginární herní vývojářské studio. Studenti se přihlásí na různé pozice, které mohou odpovídat profesím, které jsou zmíněny v kapitole *5.3 Vývoj počítačových her*. Poté společně pracují na tom, aby jejich hry byly zařazeny do vývojového plánu studia. Nutná je spolupráce, protože úspěch celého studia závisí na úspěchu *všech her*. Kurz klade důraz na spolupráci místo soutěžení. Hlavní cíl kurzu byl podpořit studenty, aby si navzájem pomáhali dosáhnout úspěchu, a byly zavedeny bonusové body za vůdčí schopnosti a *za úspěch celého studia* – tedy celé třídy. V této koncepci se očekává aktivní účast studentů, *pravidelné hodnocení* od učitelů i spolužáků, a neustálé zlepšování předkládaných prací. Od klasického projektového vyučování se tedy odlišuje zejména v přístupu k hodnocení, kdy Lee Sheldon představuje pravidelné hodnocení na rozdíl od klasické projektové výuky, kdy je prezentován až výsledný produkt.

7 Vývoj hry na platformě BBC micro:bit pomocí jazyka Python

Z prostředků využitelných ve výuce programování jsme se rozhodli dále zkoumat programování her pro desku BBC micro:bit ve vývojovém prostředích *MakeCode* a *Mu editor*. Toto spojení je zajímavá volba zejména pro žáky 8. a 9. ročníků ZŠ či studenty středních škol.

Před programováním samotné hry je nutné aby žáci měli znalost programování desky BBC micro:bit pomocí jazyka Python. Vycházíme z dostupných učebnic na webu *imysleni.cz*, po jejichž prostudování mají žáci základní úroveň znalostí jazyka Python, respektive jeho modifikace *MicroPython*.

První učebnice má název *Robotika pro základní školy: programujeme micro:bit pomocí Makecode* a podíleli se ní autoři Jiří Pech, Jan Pršala, Jiří Vaníček a Milan Novák. Učebnice je dostupná ve formátu *PDF*, online a zdarma. Je rozdělená na 2 části – *Pracovní listy pro žáky* a *Metodické materiály pro učitele.* Jak již název napovídá, učebnice je průvodcem programování desky mirco:bit pomocí MakeCode. Přestože prostředí MakeCode nabízí také programování pomocí jazyků Python a JavaScript, učebnice se však věnuje pouze vizuálnímu programování pomocí umisťování bloků.

Druhá učebnice s názvem *Robotika pro střední školy: programujeme Micro:bit pomocí Pythonu* navazuje na první učebnici a je určena již pro středoškoláky. Tato učebnice je rozdělena na 3 části. Opět zde najdeme *Metodické listy* pro učitele a *Pracovní listy* pro žáky, ale navíc učebnice obsahuje *Úvod a teorii.*

Úvod a teorie je PDF soubor, který lze v podstatě považovat za samostatnou a hlavní učebnici. Autoři podrobněji vysvětlují principy programování v Pythonu, respektive *MicroPythonu*, který se používá pro programování micro:bitu a podobných zařízení jako například *Raspberry Pi Pico* nebo *Lego Mindstorms EV3.* Většinou žákům bude stačit pouze samotná deska Micro:bit, ovšem některé úlohy vyžadují také připojení periferií jako kabelové vodiče, tříbarevnou diodu či teplotní čidlo. V úvodu učebnice autoři vybízí žáky k instalaci prostředí *Mu*, které je dostačující pro tvorbu programů spustitelných na platformě micro:bit a zároveň nenabízí přebytečné množství funkcí, které by odváděly žákovu pozornost. Příklady z učebnice lze ovšem vytvářet také v jiných libovolných prostředích dle potřeby. Autoři doporučují alespoň základní znalosti jazyka Python. Pokud mají žáci předchozí znalosti programování a jazyka Python, učebnice je pro ně psána tak přehledně a podrobně, že je vhodná také pro samouky, kteří nemají k dispozici pomoc učitele nebo zkušenějšího programátora. Dále nalezneme v učebnici spoustu úloh pro žáky s následným řešením – pokud si žák neví

rady, může zkopírovat zdrojový kód z učebnice a použít jej ve svém programu nebo se kódem nechat inspirovat.

Pracovní listy obsahují soubory úloh, jejichž řešením žáci postupně získávají nové znalosti a otevírají se jim nové možnosti v jejich schopnostech programovat. Úlohy často přichází s programem, který řeší určitý problém a vyzývají žáky k *„odladění a nahrání“* takového programu.

Pracovní listy však neobsahují nějakou komplexní hru. Pro účely této diplomové práce jsme tedy vytvořili hru, kterou můžou žáci následně upravovat.

7.1 Chyť padající hvězdu

Hra "Chyť padající hvězdu" je jednoduchá hra pro desku BBC micro:bit kde hráč ovládá postavu (hráče) pomocí tlačítek A a B a snaží se chytit padající hvězdy. Hra začíná odpočtem 3, 2, 1 a na konci zobrazuje počet chycených hvězd.

Funkce *countdown()* zobrazí odpočet 3, 2, 1 před začátkem hry a funkce *draw(player_x, star_x, star_y)* vykreslí aktuální stav hry na displeji micro:bitu. Hra se ovládá pomocí tlačítek, *tlačítko A* pohyb hráče doleva a *tlačítko B* pohyb hráče doprava. Hlavní smyčka hry obsahuje logiku pro pohyb hráče a hvězdy, kontrolu kolize, zobrazení výsledků a opětovné spuštění hry.

Kompletní zdrojový kód hry vypadá následovně:

```
1. from microbit import *
2. import random
3. 
4. # Funkce pro odpočet
5. def countdown():
6.     for i in range(3, 0, -1):
7.         display.show(str(i))
8.         sleep(1000)
9.     display.clear()
10. 
11. # Funkce pro vykreslení hry
12. def draw(player_x, star_x, star_y):
13.     display.clear()
14.     display.set_pixel(player_x, 4, 9)  # Hráč na spodním řádku
15.     display.set_pixel(star_x, star_y, 5)  # Padající hvězda
16. 
17. # Hlavní smyčka hry
18. while True:
19.     # Odpočet před začátkem hry
20.     countdown()
21. 
22.     # Inicializace hry
23.     player_x = 2
24.     star_x = random.randint(0, 4)
25.     star_y = 0
26.     score = 0
27. 
28.     # Smyčka hry
29.     while True:
```

```
        # Kontrola tlačítek
        if button_a.is_pressed():
            player_x = max(0, player_x - 1)  # Pohyb doleva
        if button_b.is_pressed():
            player_x = min(4, player_x + 1)  # Pohyb doprava

        # Pohyb hvězdy dolů
        star_y += 1

        # Kontrola, zda hvězda dosáhla spodního řádku
        if star_y == 4:
            if star_x == player_x:
                score += 1
                display.show(Image.HAPPY)
                sleep(500)
            else:
                display.show(Image.SAD)
                sleep(500)
                break  # Konec hry
            # Resetování pozice hvězdy
            star_x = random.randint(0, 4)
            star_y = 0

        # Vykreslení hry
        draw(player_x, star_x, star_y)
        sleep(500)

    # Zobrazení skóre
    display.scroll("Score: " + str(score))
```

Tento kód je již složitější než kódy, které se vyskytují v uvedených učebnicích a vyžaduje určité zkušenosti s programováním a s jazykem Python. Předpokládáme, že většina žáků ZŠ by sami takovýto kód nevytvořili. Pro zajímavost však můžeme žákům představit, jak takovýto kód vzniká.

7.1.1 Postup vývoje

Následující postup vývoje odpovídá jednotlivým vývojovým fázím, které jsme si představili v kapitole *4 Vývoj softwaru*. Každý vývoj začíná *plánováním a analýzou požadavků.*

1. Analýza požadavků

- Hra musí být jednoduchá a rychlá.
- Hra musí být kompatibilní s BBC micro:bit
- Kód musí být snadno čitelný a udržovatelný
- Hra začne odpočtem 3, 2, 1.
- Hráč ovládá postavu pomocí tlačítek A a B (doleva a doprava).
- Hvězda padá shora dolů.

- Hráč musí chytit hvězdu pohybem postavy.
- Pokud hráč chytí hvězdu, skóre se zvýší.
- Na konci hry se zobrazí skóre a hra začne znovu s odpočtem.

2. Návrh

- Hra bude obsahovat hlavní smyčku, která bude řídit všechny ostatní funkce.
- Funkce pro odpočet před startem hry.
- Funkce pro vykreslení herní obrazovky.
- Logika pro ovládání pohybu hráče a padání hvězdy.
- Kontrola kolize mezi hráčem a hvězdou.
- Zobrazení skóre a restart hry po skončení.
- *Diagram tříd* – není důležitý pro tento jednoduchý skript, ale pro ilustraci by mohl vypadat takto:

  |-- countdown()

  |-- draw(player_x, star_x, star_y)

  |-- main_loop()

3. Implementace

V této fázi se již vzniká zdrojový kód, který si můžeme rozdělit na 3 části:

- Import knihoven.
- Definice funkcí (`countdown`, `draw`).
- Hlavní smyčka hry (`main_loop`).

4. Testování

- Jednotkové testy**:**
  - Testování funkce `countdown` pro správné zobrazení odpočtu.

- Testování funkce `draw` pro správné vykreslení hráče a hvězdy na různých pozicích.
- Testování pohybu hráče vlevo a vpravo.
- Testování pádu hvězdy a kontroly kolize.

- Integrační testy**:**
  - Testování celé hry od odpočtu po zobrazení skóre.
  - Testování správného zvýšení skóre při chycení hvězdy.
  - Hra byla nahrána do micro:bitu a byla testována správnost chování hry při interakci s tlačítky A a B.

5. Údržba

- Doplnění dokumentace
  - Dokumentace kódu s vysvětlením jednotlivých funkcí a jejich parametrů může být v textovém souboru například takto:

```
1. Hra "Chyť padající hvězdu" pro BBC micro:bit v MicroPythonu.
2.
3. Funkce:
4. - countdown(): Zobrazí odpočet 3, 2, 1 před začátkem hry.
5. - draw(player_x, star_x, star_y): Vykreslí hráče a hvězdu na displeji micro:bitu.
6. - Hlavní smyčka hry: Řídí celou hru, včetně pohybu hráče, padání hvězdy, kontroly kolize a
zobrazení skóre.
7.
8. Ovládání:
9. - Tlačítko A: Pohyb hráče doleva
10. - Tlačítko B: Pohyb hráče doprava
```

  - Komentáře jsou obsaženy přímo v kódu pro usnadnění údržby a případné budoucí úpravy.

- Návrhy na vylepšení:
  - Přidání úrovní obtížnosti (rychlejší padání hvězd).
  - Implementace zvukových efektů pomocí buzzeru.
  - Možnosti pauzy při dlouhém stisku tlačítek

Jak již bylo řečeno, nepředpokládáme, že by žáci 2. stupně naprogramovali v jazyce Python samostatně takovou hru pouze se znalostmi z uvedených učebnic. Tento program je

určen spíše žákům k prostudování a měli by provést následně *dekompozici* programu. Následně mohou program libovolně upravovat a vylepšovat dle své libosti. Učitelé mohou upravit zdrojový kód této hry, například smazat poznámky nebo smazat nějaké funkce či vložit úmyslně nějakou chybu a žáci mohou pracovat na nápravě.

7.1.2 Vyhodnocení

Hotové řešení bylo zkoumáno na naší respondentce, žákyni 8. ročníku ZŠ. Ačkoliv neměla žádné předchozí zkušenosti s programováním v jazyce Python, setkala se již s prostředím Scratch. Naše respondentka tedy ovládá základy programování a algoritmizace. Mikropočítač BBC micro:bit však viděla poprvé v životě a okamžitě o něj projevila zájem.

Respondentka dostala krátký úvod do vizuálního programování pomocí *MakeCode* a poté vytvářela doma vlastní programy samostatně. Následně jsme přešli k editoru *Mu* a programování v Pythonu. Drželi jsme se úloh z učebnic dostupných na *imysleni.cz* a s naší pomocí úkoly zvládala. Poté, co nabyla určitou jistotu v čtení jednotlivých kódů z učebnice ji byla představena hra *Chyť padající hvězdu.*

Respondentku zaujalo a bavilo především hraní hry samotné. Dekompozice kódu se však neobešla bez naší pomoci. Co se týče úprav kódu, respondentka zvládla měnit základní atributy jako zpoždění displeje či zaměnit tlačítka mezi sebou. Přidávat samostatně nové funkce pro ni bylo zatím příliš náročné a je vhodné spíš pro starší studenty na středních školách.

Závěr

V průběhu této diplomové práce jsme se zabývali otázkou, jak efektivně rozvíjet informatické myšlení u žáků základních škol prostřednictvím her s důrazem na programovací jazyk Python. Cílem bylo ukázat, že správně zvolené pedagogické metody mohou nejen zvýšit zájem žáků o informatiku, ale také podpořit jejich schopnosti v oblasti logického a kritického myšlení, které jsou nezbytné pro řešení komplexních problémů.

V práci jsme se nejprve věnovali výuce informatiky z historie až po současnost. Dále jsme se podrobněji věnovali informatickému myšlení. Bylo ukázáno, že informatické myšlení zahrnuje dovednosti jako je dekompozice problému, rozpoznávání vzorců, abstrakce a algoritmizace. Tyto dovednosti jsou klíčové nejen pro programování, ale i pro širokou škálu dalších disciplín a profesí.

Věnovali jsme se také vývoji softwaru, současným trendům v softwarovém inženýrství a proč je důležité znát agilní metodiky vývoje softwaru. Implementace programovacího jazyka Python ve výuce byla dalším klíčovým bodem této práce. Ukázali jsme si, proč je Python dobrá volba pro výuku programování oproti jiným programovacím jazykům. Přednosti Pythonu jsou zejména jednoduchá syntaxe, široká použitelnost, velké množství dostupných materiálů, knihoven, dostupné robotické stavebnice, široká uživatelská komunita a možnost objektově orientovaného programování. Díky těmto klíčovým vlastnostem se ukázal být ideálním nástrojem pro výuku programování.

Cílem diplomové práce byla analýza současných řešení a přístupů k výuce programování a výběr vhodného řešení pro naprogramování hry pomocí jazyka Python. Ukázali jsme si různé možnosti využití jazyka Python a nástroje jako EduBlocks, MakeCode a především micro:bit. Jako dobrou platformou v kombinaci s jazykem Python se ukázala právě programovatelná deska *BBC micro:bit* na které jsme si v závěru práce ukázali postupný vývoj počítačové hry *Chyť padající hvězdu*. Tato hra je určena k dekompozici a následným úpravám dle úrovně programátorských schopností jednotlivých žáků.

Seznam použité literatury a zdrojů

ANAYA, Mariano, 2018. *Clean Code in Python: Refactor Your Legacy Code Base.* 1. vyd. Birmingham: Packt Publishing, Limited. ISBN 9781788835831;1788835832;

BALARINOVÁ, Jindra, 2015. *Úvod do algoritmizace a programování pro děti.* Ostrava: [Ostravská univerzita v Ostravě]. ISBN 978-80-7464-711-6.

BASLER, J. a DOSTÁL, J. (2016). Analysis of studies focused on research of computer games influence with an accent on education and people's psychic. In ICERI2016 Proceedings, Seville, Spain: IATED. 33–40. DOI: 10.21125/iceri.2016.1007

BROOKSHEAR, J. Glenn, David T. SMITH a Dennis BRYLOW. *Informatika.* Přeložil Jakub GONER. Brno: Computer Press, 2013, 608 s. ISBN 978-80-251-3805-2.

COUFALOVÁ, Jana, 2006. *Projektové vyučování pro první stupeň základní školy: náměty pro učitele.* Praha: Fortuna. ISBN 80-7168-958-0.

CSÍKSZENTMIHÁLYI, Mihály a Eva HAUSEROVÁ. *Flow: o štěstí a smyslu života*. Praha: Portál, 2015, 326 s. ISBN 978-80-262-0918-8.

ČÁP, Jan a Jiří MAREŠ. *Psychologie pro učitele*. Vyd. 2. Praha: Portál, 2007, 655 s. ISBN 9788073672737.

ČELÁKOVÁ, Naděžda a Jan ČELÁK, 1992. *Za tajemstvím počítačových her*. Praha: Grada. ISBN 8085623390.

DEITEL, P. J. a DEITEL, H. M. *Intro to Python: for computer science and data science: learning to program with AI, Big Data, and the Cloud*. 2020. USA: Pearson Education. ISBN 978-0-13-540467-6.

DOLEŽAL, Jan, Pavel MÁCHAL a Branislav LACKO, 2009. *Projektový management podle IPMA*. Praha: Grada. Expert (Grada). ISBN 978-80-247-2848-3.

DOWNEY, Allen B. *Think Python*. Second edition. Sebastopol: O'Reilly, 2015, s. 195, 268 s. ISBN 978-1-4919-3936-9.

DURMEK, Stefan, 2012. Úvod do game designu. In Jirkovský, J. Game industry 2. [Praha]: D.A.M.O. ISBN 978-80-904387-3-6.

Engineering – Definition, History, Functions, & Facts. Encyclopaedia Britannica. [online]. Dostupné z: https://www.britannica.com/technology/engineering [Citováno: 20. prosince 2023].

FEIBEL, Werner, 1996. *Encyklopedie počítačových sítí* [online]. Praha: Computer Press [cit. 2022-02-22]. ISBN 80-85896-67-2.

Fundamentals of algorithms - AQA: Decomposition, 2024. In: BBC. *BBC Bitesize* [online]. [cit. 2024-02-26]. Dostupné z: https://www.bbc.co.uk/bitesize/guides/zjddqhv/revision/2

HARATEK, Vít, 2011. Průřez historií videoherního průmyslu. In Jirkovský, J. *Game Industry: vývoj počítačových her a kapitoly z herního průmyslu.* 11–19. Praha: D.A.M.O.

HISTORY.COM EDITORS. Video Game History. In: *A&E TELEVISION NETWORKS. HISTORY* [online]. 2017, 2024-10-17 [cit. 2024-02-10]. Dostupné z: https://www.history.com/topics/inventions/history-of-video-games

HOGENOVÁ, Anna., Hra a filosofie. Pedagogika 2001 roč. 51 č. 4, stránky 471-487.

HOSSAIN, Mohammad Ikbal. *Software Development Life Cycle (SDLC)Methodologies for Information Systems ProjectManagement.* International Journal for Multidisciplinary Research [online]. 2023, 5(5) [cit. 2024-01-20]. ISSN 2582-2160. Dostupné z: doi:10.36948/ijfmr.2023.v05i05.6223.

HUIZINGA, Johan. *Homo ludens: o původu kultury ve hře*. 1938. Přeložil Jaroslav VÁCHA. Studie (Dauphin). Praha: Dauphin, 2000. ISBN 80-7272-020-1.

CHLEBEK, Tomáš. Apple Vision Pro v rukou a na hlavách lidí. Chodí s nimi po kavárnách, jezdí na skateboardu i řídí. In: *CzechCrunch.cz* [online]. 2024 [cit. 2024-02-17]. Dostupné z: https://cc.cz/swarovski-vybavil-dalekohled-umelou-inteligenci-diky-ni-dokaze-okamzite-identifikovat-pozorovane-zvire/

CHRÁSKA, Miroslav. *Informační technologie ve škole*. In Jiří KROPÁČ a kol. Didaktika technických předmětů. 1. vyd. Olomouc: PdF UP, 2004, 223 s. ISBN 80-244-0848-1.

IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990. [online]. Dostupné z: https://ieeexplore.ieee.org/document/159342 [Citováno: 20. prosince 2023].

JIRKOVSKÝ, Jan, 2011. *Game industry: vývoj počítačových her a kapitoly z herního průmyslu*. [Praha]: D.A.M.O. ISBN 978-80-904387-1-2.

JIRKOVSKÝ, Jan, 2012. Game industry 2. [Praha]: D.A.M.O. ISBN 978-80-904387-3-6.

KERNIGHAN, Brian W., 2019. *Jak porozumět digitálnímu světu: vše, co potřebujete vědět o internetu, bezpečnosti a soukromí.* Přeložil Petr HOLČÁK. Praha: Argo. Zip (Argo: Dokořán): Dokořán). ISBN 978-80-7363-903-7.

KLEMENT, M., BRYNDOVÁ, L., DRAGON, T., ŠALOUN, P. AND KUBÍČEK, R. SELECTED EDUCATIONAL ROBOTICS TOOLS FROM THE PERSPECTIVE OF PRIMARY SCHOOL STUDENTS. *Journal of Technology and Information Education*, 2021, vol. 13, iss. 2, p. 205-223.

KLEMENT, Milan, Tomáš DRAGON a Lucie BRYNDOVÁ, 2020. *Computational Thinking and How to Develop it in the Educational Process* [online]. Křížkovského 8, 771 47 Olomouc: Univerzita Palackého v Olomouci [cit. 2024-02-26]. ISBN 978-80-244-5796-3. Dostupné z: doi:10.5507/pdf.20.24457963

KNUTH, Donald E. Algorithms. *Scientific American* [online]. Scientific American, a division of Nature America, 1977(4), 63-66 [cit. 2022-02-22]. Dostupné z: https://www.jstor.org/stable/pdf/24953982.pdf?refreqid=excelsior%3A9d1a8d55699adf05b06e948e03a78072&ab_segments=&origin=.

Koncepce státní informační politiky ve vzdělávání. [online] Praha: MŠMT, 2000. [cit. 2021-04-01]. Dostupné z: http://www.msmt.cz/file/36102_1_1/download/.

KRATOCHVÍLOVÁ, Jana, 2006. *Teorie a praxe projektové výuky*. Brno: Masarykova univerzita. ISBN 80-210-4142-0.

KUBRICKÝ, J. AND KLEMENT, M. OBJECT ORIENTED PROGRAMMING IN TEACHING. *Journal of Technology and Information Education*, 2009, vol. 1, iss. 3, p. 136-138.

LEE SHELDON, 2020. *The Multiplayer Classroom: Designing Coursework as a Game*. 2. CRC Press. ISBN 978-0-367-24906-9.

Making the transition from Scratch to Python easier with EduBlocks. 2020. In: *(Hello World)* [online]. 13. s. 67-69 [cit. 2024-03-25]. Dostupné z: https://www.raspberrypi.org/hello-world/issues/13

MCGILL, Monica. Learning to Program with Personal Robots: Influences on Student Motivation. *ACM Transactions on Computing Education* [online]. 2012, (1) [cit. 2022-05-04]. Dostupné z: https://doi.org/10.1145/2133797.2133801

MŠMT, 2014. *Strategie digitálního vzdělávání do roku 2020* [online]. Dostupné také z: https://www.msmt.cz/uploads/DigiStrategie.pdf

MUJUMDAR, Ashwini, MASIWAL Gayatri a CHAWAN Pramila. *Analysis of various Software Process Models.* International Journal of Engineering Research and Applications (IJERA). [online]. 2012 [cit. 2024-01-24]. Dostupné z: https://www.researchgate.net/publication/316510707_Analysis_of_various_Software_Process_Models

NEUMAJER, Ondřej. *Proč a jak inovovat pojetí ICT v rámcových vzdělávacích programech*. In: Metodický portál RVP.CZ [online]. [cit. 2021-04-01]. Dostupné z: https://clanky.rvp.cz/clanek/o/z/2989/proc-a-jak-inovovat-pojeti-ict-v-ramcovych-vzdelavacich-programech.html/.

Pecinovský, Rudolf. *Proč a jak učit OOP žáky základních a středních škol*. Žilinská didaktická konferencia. [online]. 2004 [cit. 2022-04-29]. Dostupné z: http://vyuka.pecinovsky.cz/Proc_a_jak_ucit_OOP_na_ZS_a_SS.pdf

PELÁNEK, Radek, 2018. *Želví grafika: exkurze do programování, geometrie a umění*. Brno: Computer Press. ISBN 978-80-251-4905-8.

PRŮCHA, Jan, Eliška WALTEROVÁ a Jiří MAREŠ, 2013. *Pedagogický slovník*. 7., aktualiz. a rozš. vyd. Praha: Portál. ISBN 978-80-262-0403-9.

PYTHON SOFTWARE FOUNDATION, © 2001-2022. Python 3.10.6 dodumentation. *Python.org* [online]. [cit. 2022-05-04]. Dostupné z: https://docs.python.org/3/index.html

PYVEC. [online]. [cit. 2023-11-03]. Dostupné z: https://pyvec.org/

Rámcový vzdělávací program pro základní vzdělávání [online]. Praha: MŠMT, 2017. Dostupné z: https://www.msmt.cz/file/43792/.

Rámcový vzdělávací program pro základní vzdělávání [online]. Praha: MŠMT, 2021. Dostupné z: http://www.nuv.cz/file/4982/.

RICH, Peter J., Garrett EGAN a Jordan ELLSWORTH, 2019. A Framework for Decomposition in Computational Thinking. In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* [online]. New York, NY, USA: ACM, 2019-07-02, s. 416-421 [cit. 2024-02-28]. ISBN 9781450368957. Dostupné z: doi:10.1145/3304221.3319793

RICHARDS, Daniel. *The History and Evolution of Video Games* [online]. MEDIUM. 2023 [cit. 2024-02-11]. Dostupné z: https://richardsdaniel.medium.com/the-history-and-evolution-of-video-games-f96b99a2e330

RVP ZV - Rámcový vzdělávací program pro základní vzdělávání. *Edu.cz* [online]. [cit. 2024-03-13]. Dostupné z: https://www.edu.cz/rvp-ramcove-vzdelavaci-programy/ramcovy-vzdelavacici-program-pro-zakladni-vzdelavani-rvp-zv/

SENEVIRATNE, Pradeeka, 2018. *Beginning BBC micro:bit: A Practical Introduction to micro:bit Development* [online]. 1. Apress Media [cit. 2024-04-06]. ISBN 978-1-4842-3360-3. Dostupné z: doi:https://doi.org/10.1007/978-1-4842-3360-3

SENEVIRATNE, Pradeeka, 2019. *BBC micro:bit Recipes: Learn Programming with Microsoft MakeCode Blocks* [online]. 1. Apress Media [cit. 2024-04-06]. ISBN 978-1-4842-4913-0. Dostupné z: doi:https://doi.org/10.1007/978-1-4842-4913-0

SHORE, James a Shane WARDEN, 2007. *The Art of Agile Development*. 1. Sebastopol, CA: O'Reilly Media. ISBN 0-596-52767-5.

Shutdown or restart? The way forward for computing in UK schools. THE ROYAL SOCIETY. [online]. 2012 [cit. 2022-02-24]. Dostupné z: https://royalsociety.org/~/media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf

SLÁMA, David. Chléb a hry: Historie počítačových her. In: *Živě* [online]. [cit. 2024-02-10]. Dostupné z: https://www.zive.cz/clanky/chleb-a-hry-historie-pocitacovych-her/sc-3-a-147762/default.aspx

STACK OVERFLOW. *2023 Developer Survey* [online]. [cit. 2023-11-03]. Dostupné z: https://survey.stackoverflow.co/2023/#most-popular-technologies-language.

STUCHLÍKOVÁ, Iva, Tomáš JELÍNEK et al. *Oborové didaktiky: vývoj – stav – perspektivy* [online]. Brno: Masarykova Univerzita, 2015. Dostupné z: http://www.ped.muni.cz/didacticaviva/data_pdf/ knihy/oborove-didaktiky_online.pdf.

SVOBODOVÁ, Radka, Branislav LACKO a Ondřej CINGL, 2010. *Projektové řízení a projektové vyučování, aneb, Jak na výukové projekty podle zásad projektového řízení.* Choceň: PM Consulting. ISBN 978-80-254-8174-5.

TOLLERVEY, Nicholas H. *Python in Education: Teach, Learn, Program*. Sebastopol: O'Reilly Media, 2015.

TOLLERVEY, Nicholas H., 2017. *Programming with MicroPython*. 1. O'Reilly Media. ISBN 978-1-491-97273-1.

TOMKOVÁ, Anna, Jitka KAŠOVÁ a Markéta DVOŘÁKOVÁ, 2009. *Učíme v projektech*. Praha: Portál. ISBN 978-80-7367-527-1.

VÁVRA, Daniel, 2012. Práce herního designéra. In Jirkovský, J. Game industry 2. [Praha]: D.A.M.O. ISBN 978-80-904387-3-6.

VORDERMAN, Carol, 2022. *Programování pro děti: od úplných základů k programování jednoduchých her*. Přeložil Jan ANDRŠ. Praha: Slovart. ISBN 978-80-276-0325-1.

WATKISS, Stewart, 2020. *Beginning game programming with Pygame Zero: coding interactive games on Raspberry Pi using Python.* [New York]: Apress L.P. ISBN 978-1-4842-5649-7.

WYMAN, Michael Thornton, 2011. *Making great games: An Insider's Guide to Designing and Developing the World's Greatest Video Games.* 1. Elsevier. ISBN 978-0-240-81284-4.

Seznam zkratek

GUI	Graphic user interface
PC	Personal computer
SIPVZ	Koncepce státní informační politiky ve vzdělávání
SDV2020	Strategie digitálního vzdělávání do roku 2020
RVP ZV	Rámcový vzdělávací program pro základní vzdělávání
OOP	Objektově orientované programování
PSF	Python Software Foundation
VR	Virtual reality (virtuální realita)
AR	Augmented reality (rozšíření realita)
MR	Mixed reality (smíšená realita)
FPS	First-person shooter
MŠMT	Ministerstvo školství, mládeže a tělovýchovy

Seznam obrázků