Univerzita Karlova
Pedagogická fakulta
Katedra informačních technologií a technické výchovy

BAKALÁŘSKÁ PRÁCE

Rozvoj algoritmického myšlení žáků prostřednictvím herních projektů v prostředí Scratch

Development of Pupils' Algorithmic Thinking Through Game Projects in Scratch

Vojtěch Benýšek

Vedoucí práce:	PhDr. Jiří Štípek, Ph.D.
Studijní program:	Informační technologie se zaměřením na vzdělávání (B0114A140004)
Studijní obor:	Informační technologie se zaměřením na vzdělávání (0114RA140004)

2024

Odevzdáním této bakalářské práce na téma Rozvoj algoritmického myšlení žáků prostřednictvím herních projektů v prostředí Scratch potvrzuji, že jsem ji vypracoval pod vedením vedoucího práce samostatně za použití v práci uvedených pramenů a literatury. Dále potvrzuji, že tato práce nebyla využita k získání jiného nebo stejného titulu.

V Praze dne 2. 4. 2024

…….

podpis

Poděkování

Chtěl bych velmi poděkovat vedoucímu bakalářské práce PhDr. Jiřímu Štípkovi, Ph.D., za jeho cenné rady, ochotu, pečlivost, snahu neustále mě posouvat dále a také za trpělivost při vedení této práce. Děkuji i pracovišti DDM Praha 9 na Proseku za poskytnutí možnosti vést kroužek programování, v rámci něhož byl sestavený kurz ověřen. V neposlední řadě patří můj dík i celé mé rodině za podporu ve studiu.

ABSTRAKT

Bakalářská práce se zabývá tématem rozvoje algoritmického myšlení žáků prostřednictvím herních projektů ve vývojovém prostředí Scratch. Hlavním cílem práce bylo vytvořit soubor na sebe navazujících lekcí (kurz) ve Scratch, který by bylo možno využít jako metodiku v hodinách informatiky pro rozvoj algoritmického myšlení žáků ve věku 8-10 let.

Teoretická část se věnuje zejména pojmu informatické, resp. algoritmické myšlení, přičemž se opírá o studium odborných zdrojů. Zaměřuje se na vysvětlení a definování těchto termínů, u algoritmického myšlení jsou identifikovány možnosti jeho rozvoje s důrazem na programování. Za účelem nalezení dalších přístupů k rozvoji algoritmického myšlení byl zanalyzován jak Rámcový vzdělávací program České republiky, tak i příslušné dokumenty vybraných evropských států.

Poznatky získané v teoretické části byly následně zužitkovány v části praktické, tedy při vytváření samotného kurzu. Ten sestává z deseti lekcí, jejichž hlavní náplní je tvorba projektu. Pro motivování žáků bylo rozhodnuto, že projekty budou vždy hrami. Lekce kromě postupu možného řešení obsahují i metodické poznámky, které vyučující upozorňují na případná úskalí, či informují, jak vést výklad dané problematiky.

Soubor lekcí byl následně ověřen v hodinách kroužku programování na dvou skupinách po devíti žácích v cílové věkové kategorii. V průběhu kurzu se ukázalo, že žáci jsou schopni využívat osvojené algoritmické koncepty z předchozích lekcí, tudíž lze usuzovat, že u nich došlo k rozvoji algoritmického myšlení. Jednotlivé lekce byly navíc na základě zkušeností získaných při ověřování upraveny a vylepšeny.

KLÍČOVÁ SLOVA

algoritmické myšlení, informatické myšlení, rozvoj algoritmického myšlení, dětské programovací jazyky, blokové programování, Scratch, herní projekty, učební materiál

ABSTRACT

The bachelor thesis focuses on the topic of developing pupils' algorithmic thinking through gaming projects in the Scratch development environment. The main goal of the work was to create a set of interconnected lessons (a course) in Scratch, which could be used as educational material in computer science classes to develop algorithmic thinking of students aged 8-10 years.

The theoretical part mainly addresses the concept of computational, or rather, algorithmic thinking, relying on the study of professional sources. It focuses on explaining and defining these terms, identifying the possibilities of developing algorithmic thinking with an emphasis on programming. To find further approaches to the development of algorithmic thinking, both the Czech Republic's educational plan and those of selected European countries were analyzed.

The insights gained in the theoretical part were subsequently utilized in the practical part, i.e., in creating the course itself. This course consists of ten lessons, the main content of which is project creation. To motivate students, it was decided that the projects would always be games. The lessons, besides the possible solution procedure, also contain methodological notes, which alert teachers to potential pitfalls or inform them how to present the given topic.

The set of lessons was then tested in programming club sessions with two groups of nine pupils each in the target age category. During the course, it was observed that the pupils were able to use the algorithmic concepts learned in previous lessons, suggesting that their algorithmic thinking had developed. Moreover, based on the experience gained during testing, the individual lessons were adjusted and improved.

KEYWORDS

algorithmic thinking, computational thinking, development of algorithmic thinking, educational programming languages for children, block-structured programming, Scratch, game projects, educational material

Obsah

Část I
Teoretická východiska práce

1. Úvod

Na digitální kompetence jednotlivce jsou v posledních letech kladeny čím dál větší nároky. V souladu s tím dostává v oblasti vzdělávání stále více prostoru i informatika. Na základě revize RVP došlo k vytvoření nové vzdělávací oblasti Informatika, která se má dle znění dokumentu zaměřit zejména na rozvoj informatického myšlení. Pojem informatické myšlení je poměrně široký a zahrnuje řadu konceptů včetně algoritmického myšlení, jehož rozvoji je v rámcovém vzdělávacím programu věnována značná pozornost. Jednou z metod rozvoje algoritmického myšlení je programování, které někteří považují za všeobecnou součást vzdělání člověka 21. století.[1] Cílem vzdělávacího systému není vytvořit ze všech žáků profesionální programátory, ale spíše rozvíjet a kultivovat jejich myšlení. Výuka programování by měla žáky naučit popsat a zanalyzovat problém, rozdělit si ho na dílčí části, se kterými si dokáží poradit a následně navrhnout efektivní řešení. Tyto kompetence nacházejí využití nejen v oblasti informatiky, vyzdvihováno je jejich všeobecné uplatnění v každodenním životě.

Od 1. září 2023 navíc začala platit povinnost základních škol učit na prvním stupni dle nově revidovaného rámcového vzdělávacího programu. Přestože se o plánované úpravě výuky informatiky vědělo od roku 2021 a školy tak teoreticky měly čas se na změnu připravit, některé z nich měly se zaváděním výuky „nové informatiky“ problémy. Komplikace byly obvykle spojeny s nedostatečně proškolenými učiteli a absencí výukových materiálů. Část učitelů na 1. stupni kupříkladu nikdy programování neučila a společně s ostatními novými náležitostmi Informatiky se tak učitelé dostali pod tlak, jenž plyne zejména z nedostatku času na přípravu lekcí či případné dostudování všech částí vzdělávací oblasti. Z těchto důvodů proto v posledních měsících pramení velká poptávka po připravených, názorných a ověřených učebních materiálech, prostřednictvím kterých by učitelé mohli vést výuku nové informatiky. Posláním této práce je tak pomoci tento „hlad po materiálech“ alespoň částečně uspokojit.

Cílem bakalářské práce je proto vytvořit metodiku pro rozvoj algoritmického myšlení, konkrétně pro výuku programování, která bude pro učitele informatiky volně dostupná. Materiál by měl obsahovat metodické poznámky a názorná vysvětlení, aby ho mohl používat k výuce i učitel, který nemá s výukou programování žádné zkušenosti. Práce bude sestávat z teoretické části, jež bude zaměřena na algoritmické myšlení a možnosti jeho rozvoje, a z praktické části obsahující samotnou metodiku.

[1] Vaníček, J., Šimandl, V., & Dobiáš, V. (2022). *Situational algorithmic tasks with the assembling of program code as a tool for developing computational thinking*. Journal of Technology and Information, 14(2), 101–119, konkrétně s. 102. https://doi.org/10.5507/jtie.2022.014

2. Vymezení cílů a úkolů práce

Hlavním cílem bakalářské práce je vytvořit soubor lekcí (kurz) pro rozvoj algoritmického myšlení žáků na prvním stupni základních škol, konkrétně pro věkovou kategorii 8-10 let. Pro naplnění vytyčeného cíle byly stanoveny následující úkoly, které současně představují i postup řešení.

Nejprve je nutné pokusit se definovat pojem algoritmické myšlení a vymezit ho v rámci nadřazeného informatického myšlení. Seznámení s těmito pojmy bude probíhat prostřednictvím studia odborných zdrojů.

Následně je třeba se zaměřit na to, jak algoritmické myšlení u žáků základních škol rozvíjet, důraz bude kladen zejména na roli programování. Cílem je identifikovat vhodné typy aktivit k rozvoji algoritmického myšlení.

Dalším dílčím cílem je zanalyzovat, jak s pojmem algoritmické myšlení pracuje RVP a jak ho vymezuje. Úkolem bude prostudovat nejen RVP České republiky, ale i národní kurikula vybraných evropských zemí, což ve výsledku umožní z jednotlivých vzdělávacích systémů a jejich odlišných přístupů načerpat inspiraci pro rozvoj algoritmického myšlení.

Posledním cílem teoretické části práce bude zjistit, v jakém pořadí jsou jednotlivé algoritmické koncepty osvojovány. Zde bude východiskem zmapování příslušných zdrojů, tedy analýza odborných studií a výukových materiálů týkajících se dětských programovacích jazyků.

V praktické části bude ucelený soubor lekcí pro zvolenou věkovou skupinu sestrojen tak, aby na sebe jednotlivé lekce navazovaly. Žáci se v kurzu postupně seznámí s algoritmickými strukturami, přičemž záměrem je, aby během tvoření projektů došlo k rozvoji jejich algoritmického myšlení. U každé lekce bude výpis všeho nového, co se žáci v dané lekci naučí. Následovat bude deskripce samotné náplně lekce, která bude doplněna též o metodické poznámky, které vyučujícím pomohou v tom, jak k dané problematice přistoupit a jak ji žákům vysvětlit.

Posléze bude cílem ověřit navržený kurz v praxi. Připravený soubor lekcí bude ověřen v hodinách kroužku programování v DDM Praha 9 Prosek. Na základě získaných zjištění při výuce a zpětné vazby od účastníků kurzu budou provedeny úpravy jednotlivých lekcí. Bude vysvětleno, k jakým změnám došlo, proč k nim došlo a bude popsán průběh samotného ověřování s přihlédnutím k tomu, zda žáci dokázali využít osvojené algoritmické koncepty z předchozích lekcí.

3. Algoritmické myšlení a možnosti jeho rozvoje

Hlavní náplní této kapitoly je vymezení pojmů algoritmické, resp. informatické myšlení tak, jak budou chápány pro účely této práce. Algoritmické myšlení je součástí nadřazeného a šířeji definovaného tzv. informatického myšlení.

3.1 Vymezení pojmu informatické myšlení

Vymezení informatického myšlení (angl. *computational thinking*, zkr. CT) a jeho implementace do výuky je předmětem akademických diskuzí po více než deset let.[2] Pro pojem totiž neexistuje jednotná definice a každý z autorů, jenž se pojmem zabýval, navíc kladl důraz na něco jiného.

Ačkoliv je autorství pojmu *computational thinking* obvykle připisováno Seymouru Papertovi,[3] v kontextu vzdělávání ho poprvé zavedla Jeanette Wingová[4]. Ve svém článku neposkytla definici CT, ale popsala ho jako základní schopnost a přirovnala důležitost této schopnosti na roveň čtení, psaní a počítání. Wingová se snažila ukázat, že porozumění informatice se hodí nejen profesionálům napříč obory, ale nachází využití i při řešení běžných každodenních problémů. Za hlavní součást CT považuje schopnost rozložit složitý problém na menší části, se kterými si umíme poradit (dekompozice), nebo převedení problému na jiný. Právě díky informatice si podle ní můžeme tento způsob přemýšlení osvojit. Později Wingová definovala CT jako myšlenkové postupy, které jsou zapojeny při formulování problémů a jejich řešení. Tyto postupy umožní řešení efektivně provést agentem (stroj, člověk) zpracovávajícím informace.

Konkrétnější definice informatického myšlení pochází z dokumentu organizací ISTE (International Society for Technology in Education) a CSTA (Computer Science Teachers Association). Dokument původně vymezoval šest hlavních schopností a dovedností, z nichž sestává CT. Mezi ně patří schopnost logicky uspořádávat a zkoumat data, reprezentovat data prostřednictvím abstrakcí jako jsou modely a simulace, formulovat strojové řešení, hledat optimální řešení, zobecňovat a používat postup řešení na podobné problémy a schopnost automatizovat řešení pomocí algoritmického myšlení.[5] Z této definice v současnosti vychází i většina evropských školních kurikul.[6]

---

[2] Vědeckou diskuzi z pohledu českého školství se pokouší ve svém příspěvku shrnout Bryndová, L. (2021). *The approach of computer science teachers to the concepts of computational thinking and the implementation of its development in primary schools*. Journal of Technology and Information Education, 13(2), 151-163, shrnutí na 152-154. doi: 10.5507/jtie.2021.015

[3] Významem pojmu „*computational thinking*“ se v době, kdy ještě pro tento pojem neexistoval český překlad, zabýval Lessner, D. (2014). *Analysis of term meaning "Computational Thinking"*. Journal of Technology and Information Education, 6(1), 71-88, původ pojmu na s. 73. doi: 10.5507/jtie.2014.006.

[4] Wing, J. M. (2006). *Computational thinking*. Communications of the ACM, 49(3), 33-35. https://doi.org/10.1145/1118178.1118215

[5] K rozboru definice CT z dokumentu ISTE a CSTA srov. Bryndová, L. (2021). *The approach of computer science teachers*. s. 154., Lessner, D. (2014). *Analysis of term*, s. 75.

[6] Bryndová, L. (2021). *The approach of computer science teachers*, s. 154.

Daniel Lessner ve své studii CT definuje jednoduše jako schopnost „*myslet jako informatik při řešení problémů*“. Všímá si širokého využití informatického myšlení v běžném životě, což dokazuje na příkladu maminky, která „*myslí jako informatik, když zorganizuje jogurty v chladničce do prioritní fronty podle data spotřeby*“. Rozvoj tohoto druhu myšlení by tak dle něj měl být jedním z cílů výuky informatiky. Lessner hovoří o tom, že cílem výuky informatiky není jen ovládnout programování pro tvorbu programů, ale kultivace myšlení obecně, podobně jako k němu dochází při počítání v matematice. Díky informatice si tedy osvojíme způsob přemýšlení, ne použití technologií.[7]

Projekt iMyšlení.cz propagující výuku zaměřenou na rozvoj informatického myšlení v ČR definuje informatické myšlení jako „*způsob myšlení, který se zaměřuje na popis problému, jeho analýzu a hledání efektivních řešení“*. Na stránkách projektu jsou navíc popsány i nástroje a postupy CT, kupříkladu rozdělení problému na menší a snáze řešitelné celky, systematické posouzení různých řešení a vybrání nejvhodnějšího pro danou situaci, či tvorba postupů vedoucí spolehlivě k cíli, i když je vykonává někdo jiný.[8]

Na základě uvedených zdrojů se přikláníme spíše k definici informatického myšlení dle projektu iMyšlení.cz, protože je jasná, konkrétní, a navíc systematicky popisuje všechny kompetence související s tímto způsobem myšlení. Informatické myšlení tak budeme v rámci této práce chápat jako takové myšlení, které zahrnuje následující schopnosti:

- rozpoznání a dekompozice problému (rozklad na menší části), či převedení na jiný druh
- abstrakce (zaměření pouze na detaily nezbytné pro nalezení řešení, ignorování irelevantních informací)
- zobecnění řešení pro možnost jeho opakovaného použití při setkání s podobnými problémy
- algoritmizace (vytvoření postupu a sepsání jednotlivých kroků postupu vedoucích k řešení)

3.2 Vymezení pojmu algoritmické myšlení

Algoritmické myšlení je považováno za důležitou schopnost, kterou by měl v současném světě každý ovládat.[9] Aby bylo možné vymezit tento pojem, je nejprve nutné říci, co je algoritmus.

Algoritmus je postup k vyřešení problému, který se skládá z přesně definovaných kroků.[10] Algoritmické myšlení úzce souvisí s konceptem vytváření a zpracovávání algoritmů,[11] algoritmus je konečným

---

[7] Lessner, D. (2014). *Analysis of term*, s. 72.
[8] Stránka imysleni.cz vznikla v rámci projektu „Podpora rozvíjení informatického myšlení“. Dostupné z: https://imysleni.cz/informaticke-mysleni/co-je-informaticke-mysleni [cit. 28. 12. 2023]
[9] Katai, Z. (2015). *Promoting algorithmic thinking*. Journal of Computer Assisted Learning, 31, 287-299, tvrzení na s. 287. https://doi.org/10.1111/jcal.12070
[10] V literatuře bývá algoritmus definován různě, pro účely práce postačí tato definice od Futschek, G. (2006). *Algorithmic thinking: The key for understanding computer science*. Lecture Notes in Computer Science, 4226, 159–168, definice algoritmu na s. 160. https://doi.org/10.1007/11915355_15
[11] Srov. TAMTÉŽ. Katai, Z. (2015). *Promoting algorithmic thinking*, s. 287.

produktem takového myšlení.[12] Zjednodušeně řečeno, člověk musí umět algoritmicky myslet, aby mohl vůbec algoritmus vytvořit. Samotné myšlení zahrnuje hned několik schopností:[13]

- analyzovat problém
- formulovat problém, tedy přesněji ho specifikovat (rozklad problému na části, odlišení podstatných prvků od nedůležitých)
- vytvoření algoritmu prostřednictvím adekvátních kroků, které povedou ke správnému řešení
- vylepšení efektivity algoritmu (optimalizace postupů, popř. ošetření výjimek a speciálních situací)
- hledání a opravování chyb algoritmu

Z výše vypsaných vlastností algoritmického myšlení je jasné, že je nezbytnou součástí širšího informatického myšlení. Podobně jako informatické myšlení má i to algoritmické samo o sobě využití v běžném životě, typickými příklady jednoduchých algoritmů jsou recepty na vaření nebo navigace pro cestu z jednoho místa na druhé.

3.3 Rozvoj algoritmického myšlení ve výuce

Přestože je algoritmické myšlení úzce propojeno s informatikou a programováním, je vhodné klást důraz na jeho všeobecné využití. Při výuce je třeba žákům nejprve představit algoritmy na příkladech z jejich každodenního života, u žáků prvního stupně ZŠ např. na postupu čištění zubů a ideálně slovo algoritmus vůbec nepoužívat.[14]

Futschek radí učit algoritmické myšlení tak, že si žáci vyzkouší vyřešit co nejvíce problémů. Zpočátku by se mělo jednat o řešení bez použití specifického programovacího jazyka kupříkladu prostřednictvím pseudokódu.[15] Přestože je možné rozvíjet algoritmické myšlení mnoha různými způsoby (unplugged metodami atd.),[16] rozbor všech z nich by značně přesahoval rozsah této práce. Proto se na následujících řádcích vzhledem k tématu práce krátce zaměřím na roli programování při rozvoji algoritmického myšlení a na tzv. dětské programovací jazyky.

---

[12] Olkhova, N. (2022). *Development of algorithmic thinking in primary school students when studying computer science*. Scientific Bulletin of Mukachevo State University. Series "Pedagogy and Psychology", 8(2), 25-32, zde s. 26. https://doi.org/10.52534/msu-pp.8(2).2022.25-32

[13] Srov. Futschek, G. (2006). *Algorithmic thinking,* s. 160. Jiang, B., & Li, Z. (2021). *Effect of Scratch on computational thinking skills of Chinese primary school students.* J. Comput. Educ. 8, 505–525, konkrétně s. 507. https://doi.org/10.1007/s40692-021-00190-z

[14] Mezak, J., & Papak, P. P. (2018). *Learning scenarios and encouraging algorithmic thinking*. 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 760-765. https://doi.org/10.23919/mipro.2018.8400141

[15] Futschek, G. (2006). *Algorithmic thinking,* s. 160.

[16] Způsoby rozvoje algoritmického myšlení zhodnotil Vais, J. (2021). *Rozvoj algoritmického myšlení žáků základních škol*. Diplomová práce, vedoucí Štípek, Jiří. Kapitola 4 viz s. 23-42. http://hdl.handle.net/20.500.11956/125706

Jednou z nejlepších metod pro rozvoj algoritmického myšlení je programování.[17] Kromě toho, že Vaníček považuje programování za součást všeobecného vzdělání člověka 21. století, vidí v něm *„tréninkové hřiště pro rozvíjejí schopností a kompetencí jedince"*. K rozvoji algoritmického myšlení pomocí programování je dle něj nutné mít soubor propojených situací poskytující takové problémy, které žákovi umožňují objevovat důležité myšlenky. Výuka programování by měla obsahovat prvky jako aktivní učení, učení se děláním a konstruování znalosti jako výsledku tvořivé práce.[18] Pro rozvoj algoritmického myšlení klasifikuje Vaníček dva základní typy programovacích úloh:

- „etudy" – krátké, několikaminutové aktivity zaměřené na dovednost či koncept, cílem je konkrétní znalost
- „projekty" – větší a delší aktivity, často formou komplexnějších her, cílem je produkt[19]

Produkt jako takový ovšem není sám o sobě z hlediska algoritmického myšlení a jeho rozvoje přínosem. Dle mého názoru tkví přínos produktu pro žáky v tom, že při jeho tvorbě si zopakují získané znalosti (např. z „etud"), vzájemně si je propojí a výsledný produkt navíc slouží jako motivátor do budoucna, tedy pro příští aktivity.

Vaníček dodává, že při zvolení úlohy typu „projekt" lze díky délce této aktivity během procesu tvorby či při analýze programu napsaného žákem zjistit, jak žák zadání chápe, jaké přístupy při řešení problémů používá či jakou má úroveň algoritmického myšlení.[20] Domnívám se, že tuto úroveň je možné sledovat zejména v dlouhodobém horizontu na základě toho, zda žáci dokážou využívat již osvojené algoritmické koncepty v dalších navazujících projektech.

Na prvním stupni základní školy se na podporu rozvoje algoritmického myšlení žáků používají při výuce tzv. dětské programovací jazyky. Jsou to zjednodušené programovací jazyky cílící na nejmladší věkovou skupinu, čemuž odpovídá i rozvržení a design. Prostředí bývají většinou atraktivní pro děti a programuje se v nich obvykle pomocí tzv. bloků, což jsou „puzzle" s určitou funkcí, které je možné skládat dohromady. Výhodou blokového zápisu programu je, že zamezuje chybám syntaxe.[21]

Podle nejnovějších studií mají dětské programovací jazyky pozitivní vliv na rozvoj informatického myšlení dětí obecně a na rozvoj jejich dovednosti kódování (*coding skills)*.[22] Odpověď na otázku „Od

---

[17] Olkhova, N. (2022). Development of algorithmic thinking in primary school, s. 31.
[18] Vaníček, J., Šimandl, V., & Dobiáš, V. (2022). *Situational algorithmic tasks with the assembling of program code as a tool for developing computational thinking*. Journal of Technology and Information, 14(2), 101–119, konkrétně s. 102-103. https://doi.org/10.5507/jtie.2022.014
[19] TAMTÉŽ, s. 103.
[20] TAMTÉŽ, s. 103-104.
[21] Srov. TAMTÉŽ, s. 106. Jiang, B., & Li, Z. (2021). *Effect of Scratch on computational thinking skills*, s. 506.
[22] O pozitivním vlivu aplikace ScratchJr na informatické myšlení dětí ve věku 5-7 let píše např. Stamatios, P. (2024). *Can Preschoolers Learn Computational Thinking and Coding Skills with ScratchJr? A Systematic Literature Review*. International Journal of Educational Reform, 33(1), 28-61. https://doi.org/10.1177/10567879221076077 Rozsáhlý průzkum o tom, jaký má Scratch vliv na rozvoj informatického myšlení žáků základních škol v Číně, provedli Jiang, B., & Li, Z. (2021). *Effect of Scratch on computational thinking skills*, s. 505–525.

kdy je vhodné začít s dětmi programovat?“ přinesl výzkum dvojice Atamtzidou-Demetriadis, jehož výsledkem bylo, že začít s programováním již v raném věku má pozitivní efekty na kognitivní vývoj dětí.[23]

Aby bylo možné vést úspěšně výuku cílenou na rozvoj algoritmického myšlení, je třeba žáky motivovat. To je první krok ke každému zdárnému edukačnímu procesu. Motivace je nutnou prerekvizitou a korekvizitou k učení se.[24] To znamená, že je motivace potřebná jak při inicializaci samotného procesu učení, tak i během něj, dokud není znalost patřičně osvojena.[25]

Jedním z častých způsobů, jak motivovat žáky, je spojit hraní her a samotný edukační proces. O motivování žáků prostřednictvím propojení hraní a programování se pokouší zejména tzv. program-to-play hry, jimiž se zabývali například Weintrop a Wilensky.[26] Ti dospěli k tomu, že programováním herních mechanik se žáci seznámí se základními algoritmickými koncepty a rozvinou tak své algoritmické myšlení zábavnou a smysluplnou formou, jež je pro žáky i silně motivační.[27] Učit se prostřednictvím vytváření her je považováno za jeden z nejefektivnějších způsobů výuky programování, který vede k lepší motivaci žáků a jejich většímu zapojení do výuky.[28]

Motivátorem při učení může být i přiměřená výzva (*moderate challenge*), tedy postavení žáků před takovou výzvu, jejíž obtížnost by měla takříkajíc balancovat mezi tím, aby zadání nebylo příliš těžké (a tím pádem i demotivující), ale zároveň aby se nejednalo o velmi jednoduchou úlohu, jejíž motivační efekt také ztrácí na významu. Složitost výzev by tak měla odpovídat aktuálním dovednostem žáků a obtížnost jako taková by se měla v ideálním případě s každou další výzvou postupně zvyšovat (tzv. *moderate-progressive challenge*).[29]

V praxi to znamená, že v případě tvorby souboru na sebe navazujících lekcí je vhodné jednotlivé lekce koncipovat tak, aby se jejich složitost postupně zvyšovala a pro žáky se tak jednalo vždy o určitou výzvu. Zároveň by žáci měli být schopni si na základě znalostí získaných z předchozích lekcí poradit alespoň s částí řešeného problému. Z výše zmíněného také vyplývá, že pro udržení motivace žáků ve vývojovém

---

[23] Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant diferences. Robotics and Autonomous Systems, 75, 661–670. https://doi.org/10.1016/j.robot.2015.10.008

[24] Palmer, D. (2005). *A Motivational View of Constructivist-Informed Teaching*. International Journal of Science Education. 27, 1853-1881. https://doi.org/10.1080/09500690500339654

[25] Pintrich, P. R., Marx, R. W., & Boyle, R. A. (1993). Beyond Cold Conceptual Change: The Role of Motivational Beliefs and Classroom Contextual Factors in the Process of Conceptual Change. Review of Educational Research, 63(2), 167-199. https://doi.org/10.3102/00346543063002167

[26] Program-to-play jsou ty hry, které dělají z programování centrální součást samotného hraní, více viz Weintrop, D., & Wilensky, U. (2016). *Playing by Programming: Making Gameplay a Programming Activity. Educational Technology*, 56(3), 36–41. http://www.jstor.org/stable/44430491

[27] Weintrop, Wilensky (2016). *Playing by Programming*, s. 41.

[28] Více viz Chiu, C.-F. (2015). *Introducing pre-service teachers to programming concepts with game creation approach.* International Journal of Learning Teaching and Educational Research, 13(1), 85-93. Dostupné z: https://www.ijlter.org/index.php/ijlter/article/view/414 [cit. 29. 12. 2023]

[29] Katai, Z. (2015). Promoting algorithmic thinking, s. 290.

prostředí Scratch bude vhodné tuto snahu navíc podpořit i motivátorem v podobě pro žáky smysluplného a zábavného cílového produktu, kdy hlavní projekty, jejichž tvorba bude esenciální náplní jednotlivých lekcí kurzu, budou hrami.

4. Algoritmické myšlení ve školském systému

Rozvoj algoritmického myšlení se spolu s rozvojem informatického myšlení dostává postupně alespoň v nějaké formě do výuky prakticky všech států. V této kapitole bude rozebráno, jak a zda vůbec definuje algoritmické myšlení (resp. informatické myšlení) Rámcový vzdělávací program pro základní vzdělávání (RVP ZV) MŠMT pro 1. stupeň ZŠ. Jelikož národní kurikula představují ucelený a komplexní dokument zabývající se rozvojem algoritmického myšlení u žáků cílové věkové kategorie, budou v další části kapitoly stručně rozebrány zahraniční vzdělávací plány a následně bude jejich obsah základně porovnán s příslušnou částí českého vzdělávacího programu, abychom zjistili, zda ostatní plány nabízejí jiné pohledy na rozvoj algoritmického (resp. informatického) myšlení, z nichž by bylo možné načerpat inspiraci, či se s touto problematikou vypořádávají podobně, a tudíž pro nás nebudou přínosné.

4.1 Algoritmické myšlení a jeho role v RVP

Vzdělávací obsah RVP ZV ve vzdělávací oblasti Informační a komunikační technologie nebyl aktualizován od roku 2005, v lednu 2021 byla schválena změna nového revidovaného rámcového vzdělávacího programu. V září téhož roku zahájily první školy dobrovolně výuku podle inovovaného plánu a od 1. září 2023 mají povinnost učit dle tohoto plánu i všechny školy na prvním stupni.[30]

Hlavní změnou v RVP ZV z ledna 2021 bylo zavedení nové vzdělávací oblasti Informatika, která nahradila původní oblast Informační a komunikační technologie. Cílem revize bylo „*modernizovat obsah vzdělávání tak, aby odpovídalo dynamice a potřebám 21. století*“.[31] Nová oblast má v rámcovém učebním plánu minimální časovou dotaci 2 hodiny na prvním stupni a 4 na druhém.[32]

Vzdělávací oblast Informatika se dle programu zaměřuje „*především na rozvoj informatického myšlení a na porozumění základním principům digitálních technologií*“. K pojmu informatické myšlení ovšem MŠMT nepřikládá žádnou přesnou definici, nenalézá se ani ve slovníčku použitých výrazů na konci dokumentu. Možností je tak pouze pokusit se idenfitikovat prvky algoritmického, resp. informatického myšlení, jež se ve znění plánu nachází. Oblast má být založena na aktivních činnostech, při kterých žáci využívají informatické postupy, což by odpovídalo definici Lessnera.[33] O oblasti se dále dočteme, že

---

[30] Všechny změny přehledně spolu s novým zněním RVP a ŠVP jsou k dispozici na webu revize.edu.cz spravovaném Ministerstvem školství, mládeže a tělovýchovy ve spolupráci s Národním pedagogickým institutem ČR. Dostupné z: https://revize.edu.cz/ [cit. 29. 12. 2023]

[31] Viz jednotný metodický portál MŠMT edu.cz, kde jsou dostupná i všechna znění RVP ZV. Dostupné z: https://www.edu.cz/rvp-ramcove-vzdelavaci-programy/ramcovy-vzdelavacici-program-pro-zakladni-vzdelavani-rvp-zv/ [cit. 29. 12. 2023]

[32] V rámci zkoumání RVP ZV budu vycházet z aktuálně poslední verze z června 2023, která již žádné další úpravy pro Informatiku nezavedla. Viz MŠMT. (2023). *RVP ZV z června 2023*. Rámcový učební plán na s. 140. Dostupné z: https://www.edu.cz/wp-content/uploads/2023/07/RVP_ZV_2023_cista_verze.pdf [cit. 29. 12. 2023]

[33] Srov. MŠMT. (2023). *RVP ZV z června 2023*. s. 39. a zjednodušenou definici Lessnera „*myslet jako informatik*“ viz Lessner, D. (2014). *Analysis of term*, s. 72.

poskytuje prostředky a metody ke zkoumání řešitelnosti problémů i hledání a nalézání jejich optimálních řešení a na základě řešení praktických úkolů poznatky, kdy je lepší práci přenechat stroji.[34]

Výuka Informatiky na prvním stupni by měla probíhat prostřednictvím her, experimentů a diskusí. Žáci si během ní rozvíjejí schopnost popsat problém, analyzovat ho, hledat jeho řešení a ve vhodném programovacím prostředí si mají ověřovat algoritmické postupy.[35]

V obsahu vzdělávacího oboru v kategorii „Algoritmizace a programování" mezi očekávanými výstupy nalezneme, že žák by na konci 5. třídy ZŠ měl umět:

- sestavit a testovat symbolické zápisy postupů
- popsat jednoduchý problém a navrhnout jednotlivé kroky jeho řešení (tedy tvorba algoritmu)
- sestavit program v blokově orientovaném programovacím jazyce; rozpoznat opakující se vzory, používat opakování a připravené podprogramy
- ověřit správnost navrženého postupu a opravit případnou chybu

Minimální doporučená úroveň ovšem podstatnou část těchto kompetencí vynechává.[36]

Z aktuálního RVP vidíme, že algoritmické a informatické myšlení, ač přímo nezmíněno či nedefinováno, své místo v RVP má a ve výuce se s rozvojem těchto myšlení počítá, což je možné vidět jak v úvodním odstavci o celé vzdělávací oblasti, tak i konkrétně mezi očekávanými výstupy.

4.2 Algoritmické myšlení v zahraničních vzdělávacích systémech

Nyní se zaměříme na to, jak k rozvoji algoritmického (resp. informatického) myšlení přistupují ostatní státy v Evropě. Budeme sledovat, jakou roli mají ve výuce inkriminovaná myšlení, na jaké kompetence jednotlivé národní plány cílí, a nakonec se pokusíme porovnat plán výuky algoritmického myšlení v ČR a zahraničí, abychom zjistili, zda zahraniční plány přistupují k problematice rozvoje algoritmického myšlení jiným způsobem, než jak je obsaženo v RVP. Seznámení s odlišnými přístupy by mohlo posloužit k získání inspirace pro tvorbu metodiky.

Na Slovensku je aktuální verze ŠVP (Štátny vzdelávací program pre základné vzdelávanie) z března 2023.[37] Slovenští žáci mají v prvním a druhém cyklu (1.-5. třída) povinně celkově tři hodiny informatiky týdně pro toto období.[38]

V programu je možné se v charakteristice předmětu informatika dočíst, že má kultivovat informatické činnosti, jakými jsou například objevování a zobecňování vztahů a postupů či abstraktní uvažování.

---

[34] MŠMT. (2023). *RVP ZV z června 2023*. s. 39.
[35] TAMTÉŽ.
[36] MŠMT. (2023). *RVP ZV z června 2023*. s. 40.
[37] ŠVP je dostupný z oficiálních stránek slovenského ministerstva školství. Dostupné z: https://www.minedu.sk/data/files/11808_statny-vzdelavaci-program-pre-zakladne-vzdelavanie-cely.pdf [cit. 30. 12. 2023]
[38] MŠVVŠ. (2023). Štátny vzdelávací program pre základné vzdelávanie. Rámcový učební plán na s. 13.

Důraz je kladen na rozvoj informatické gramotnosti, což podle programu znamená osvojit si informatické postupy, které umožňují řešit problémy jako informatik. V linii informatického myšlení se žáci mají seznámit s řešením algoritmických úloh s důrazem na rozpoznání správnosti řešení. Žák na konci 5. třídy by měl být dle dokumentu schopen:[39]

- identifikovat vzory, u kterých se dá použít opakování
- vytvořit program skládáním příkazů (i příkazů s parametry)
- vytvořit program, který vyžaduje známý počet opakování
- sestavit a upravit (doplnit, dokončit, modifikovat) program v jazyku vykonavatele
- rozpoznat, že program pracuje nesprávně, najít v něm chybu a opravit ji

V Polsku došlo k poslední změně informatiky v kurikulu v roce 2017.[40] Během prvních tří let na základní škole se žáci učí analyzovat a řešit problémy, řeší též úkoly a hlavolamy vedoucí k objevu algoritmů. Žák programuje vizuálně, používá jednoduché příkazy a sekvence pro tvorbu vlastních příběhů a situací.[41] Od 4. třídy se žák dle kurikula učí algoritmicky řešit problémy, je schopen analyzovat problém a definovat postup k jeho řešení. Žák ve vizuálním programovacím jazyku používá kromě jednoduchých příkazů i cykly a podmínky, dokáže sestavit a zapsat program ovládající postavu na počítači, nebo robota. Mezi další cílové kompetence se řadí i rozpoznání funkce programu a případná optimalizace algoritmů.[42]

V národním kurikulu Anglie je předmět „*Computing*“ povinný již od první třídy.[43] Předmět by měl žáky vybavit schopností používat *computational thinking*, důraz je kladen i na praktické využití získaných znalostí prostřednictvím programování. Za první dva roky výuky by se žák měl naučit, co je to algoritmus a jak se používá v programech, tvořit jednoduché programy, rozpoznat funkci programů a najít případnou chybu v nich. Po dalších třech letech výuky, tedy přibližně v 11 letech, by žák měl být schopen:

- navrhovat, psát a ladit programy, které plní určitou funkci
- řešit problémy tím, že je rozloží na menší části (dekompozice)
- používat sekvence, opakování, proměnné a různé typy vstupu a výstupu
- logicky vysvětlit, jak fungují jednoduché algoritmy

---

[39] Charakteristika předmětu a jeho obsahové výstupy týkající se programování ve vzdělávacích standardech pro oblast Matematika a Informatika na s. 45-54, v digitalizované verzi celého ŠVP (viz MŠVVŠ. (2023). *Štátny vzdelávací program pre základné vzdelávanie*.) se jedná o stránky 446-455.
[40] Obsah předmětu „Informatyka“ viz MEN. (2017). *Podstawa programowa kształcenia ogólnego. Szkoła podstawowa – Informatyka*. Dostupné z: https://www.ore.edu.pl/wp-content/uploads/2017/05/informatyka.-pp-z-komentarzem.-szkola-podstawowa-1.pdf [cit. 30. 12. 2023]
[41] TAMTÉŽ. s. 11.
[42] TAMTÉŽ. s. 12-13.
[43] Národní kurikulum pro výuku dětí od 5 do 16 let. DfE. (2013). *The national curriculum in England*. s. 7. Dostupné z: https://www.gov.uk/government/publications/national-curriculum-in-england-primary-curriculum [cit. 30. 12. 2023]

- detekovat a opravit případné chyby v algoritmech a programech

Ve francouzských *programmes d'enseignement* naopak žádné hlubší zaměření na informatické myšlení nenalezneme.[44] Ve vzdělávacím systému neexistuje přímo předmět informatika, první tři roky je výuka informatiky součástí předmětu matematika. Algoritmizace je v obsahu předmětu zmíněna pouze v souvislosti s pokroky žáků, kdy se píše, že od 2. třídy se žáci mohou pokoušet kódovat pohyby pomocí vhodné programovací aplikace a ve 3. třídě by pak měli být schopni vytvořit jednoduché algoritmy. Posléze je výuce informatiky nejblíže vcelku široce zaměřený předmět *Sciences et technologie*. Výuka tohoto předmětu začíná až v třetím cyklu, tedy od *cours moyen 1* (CM1, žáci přibližně 9 let), což odpovídá české 4. třídě. Celá charakteristika předmětu je poměrně vágní, v úvodu se pouze dočteme, že žák bude schopen formulovat problém a pracovat s informacemi k dosažení výsledku.[45] V rozsáhlých výstupech předmětu pak nalezneme jedinou zmínku týkající se algoritmizace, konkrétně že „*žáci objevují algoritmy používáním vizuálních a zábavných aplikací*".[46] Rozvoj algoritmického myšlení řešením problémů pomocí algoritmů je vytyčen mezi cíli předmětu *Technologie*, jenž se ovšem vyučuje až od 7. třídy (5ème), tedy přibližně od 12 let.[47]

Po porovnání jednotlivých státních výukových plánů můžeme deklarovat, že ČR se svými aktuálními vzdělávacími programy není nikterak pozadu, ba naopak se revizí podařilo takříkajíc „vyplnit" mezeru, kterou český školský systém ve výuce algoritmického (resp. informatického) myšlení měl. Výuka těchto myšlení je ve sledovaných státních kurikulech pojímána prakticky stejně, výjimkou zůstává Francie, které schází samostatný předmět, jenž by se cíleně věnoval informatice a rozvoji informatického myšlení. S předmětem cíleně zaměřeným na rozvoj algoritmického myšlení se tam žáci setkají až ve 12 letech, což je oproti ostatním státům poměrně pozdě.[48] Zahraniční kurikula tak ve srovnání s českým RVP přímo žádné nové či inovativní pohledy na rozvoj algoritmického myšlení neobsahují, a tudíž nepřinášejí ani inspiraci pro tvorbu metodiky.

[44] MENJ. (2015). Programmes d'enseignement du cycle des apprentissages fondamentaux (cycle 2), du cycle de consolidation (cycle 3) et du cycle des approfondissements (cycle 4). Dostupné z: https://www.education.gouv.fr/media/48461/download [cit. 30. 12. 2023]
[45] TAMTÉŽ. s. 183.
[46] TAMTÉŽ. s. 194.
[47] TAMTÉŽ. s. 353-354.
[48] Z vzdělávacích programů vychází, že žáci v České republice mají výuku informatiky nejpozději od 4. třídy, na Slovensku od 3. třídy a Polsko v nějaké formě už od první třídy. Premiantem je v tomto směru Anglie, kde předmět *Computing* mohou mít děti už od 5 let. Francie se svojí 7. třídou tak v této kategorii ztrácí.

5. Pořadí osvojování algoritmických konceptů

Tato kapitola se na základě zmapování zdrojů v podobě výukových materiálů k dětským programovacím jazykům a lekcí ve frekventovaně užívaných aplikacích pro rozvoj algoritmického myšlení pokusí shrnout, v jakém pořadí jsou algoritmické koncepty osvojovány. Při výběru zdrojů jsem vycházel z vlastních osobních zkušeností a z interního seznamu zdrojů zaměřených na tyto aktivity Katedry informačních technologií a technické výchovy PedF UK.[49]

Prakticky všechny zdroje, které byly pro tuto kapitolu podrobeny zkoumání, se shodují v tom, že jako první učí žáka pohybovat s postavou. Obvykle se jedná o postavu hráče, ať už jde o raketu, panáčka, či kočičku. Prvním krokem tedy bývá seznámení uživatele s pohybovými bloky.

Následně přichází na řadu vcelku složitá otázka, zda dříve žáky naučit cykly, nebo podmínky. Přestože u vyšších programovacích jazyků bývá jednodušší představit jako první podmínku, ze zmapovaných zdrojů vychází, že v případě dětských programovacích jazyků a aplikací pro rozvoj algoritmického myšlení je tomu naopak.

Po seznámení s pohybovými bloky je pro vyřešení dalších úloh v Robomisi nutné využít cykly s daným počtem opakování, posléze cykly s podmínkou a až poté podmíněné příkazy.[50] V aplikaci GalaxyCodr uživatel také nejprve používá bloky pohybu, následně se naučí pohyb opakovat prostřednictvím cyklů. Podmínky se děti naučí až později, přesněji při seznámení s cykly s neurčitým počtem opakování (opakuj dokud nenastane).[51] BlocklyGames obsahuje hned několik her, z nichž pouze některé jsou vhodné pro rozvoj algoritmického myšlení dětí. Takovou je kupříkladu hra s názvem Bludiště, ve které se jako první používá cyklus (3. úroveň). Na podmínky žáci narazí až od 6. úrovně.[52] Podobná je situace i u Run Marco!, kde se žáci po prvních deseti úrovních naučí pracovat s cyklem s určitým počtem opakování.[53]

Při výuce dětských programovacích jazyků se spíše také dává přednost cyklům před podmínkami. V připravených úlohách zjednodušené verze jazyka Logo (tzv. EasyLogo)[54] se žáci nejprve seznámí s cykly. I v sérii aktivit pro ScratchJr[55] nalezneme jako první prvek opakování, konkrétně se jedná o cyklus s určitým počtem opakování, který je integrován do bloků pohybu. Zadáním čísla pod blok se

[49] Seznam prozkoumaných zdrojů pro účely této kapitoly se nachází v příloze č. 6 - Zmapované zdroje.
[50] Oficiální stránky Robomise. Dostupné z: https://robomise.cz/ [cit. 4. 1. 2024]
[51] Oficiální stránky GalaxyCodr. Dostupné z: https://galaxycodr.com/sk [cit. 4. 1. 2024]
[52] Oficiální stránky BlocklyGames. Dostupné z: https://blockly.games/?lang=cs [cit. 4. 1. 2024]
[53] Oficiální stránky Run Marco!. Dostupné z: https://runmarco.allcancode.com/ [cit. 4. 1. 2024]
[54] Verze EasyLogo s českou lokalizací dostupná zdarma z: https://edu.fmph.uniba.sk/~salanci/EasyLogo/ [cit. 4. 1. 2024]
[55] Seznam aktivit se nachází na oficiálních stránkách ScratchJr. Dostupné z: https://www.scratchjr.org/teach/activities [cit. 4. 1. 2024]

vybraný blok zopakuje přesně tolikrát, kolik je číslo. V učebnici Základy programování ve Scratch pro 5. ročník základní školy[56], která byla roku 2021 schválena MŠMT k zařazení do seznamu učebnic pro základní vzdělávání pro obor Informatika, nalezneme opakování a cykly hned v prvním modulu. Seznámení s podmínkami je obsahem až třetího (tedy posledního) modulu. Podmínky naprosto chybějí v oficiální metodické příručce k úvodním aktivitám v prostředí Scratch. V aktivitách se můžeme setkat pouze s cykly „opakuj stále“ a „opakuj x krát“.[57]

V neposlední řadě jsme mohli vidět upřednostňování cyklu ve výuce i v kurikulech jednotlivých států, které schopnost rozpoznání vzorů a aplikování opakování často vytyčují jako jeden z cílů předmětu informatika.[58] Nabízí se tak otázka, proč ve většině zmíněných zdrojů dochází k preferování cyklů.

Důvodem toho může být fakt, že po osvojení pohybových bloků se využití opakování přímo samo vybízí. Stručně řečeno, když žák kupříkladu nastaví, aby se postavička pohnula o jeden krok doprava, je pro něho vcelku jednoduché pochopit, že pro pohyb postavy o pět kroků doprava musí dát buď pět stejných příkazů po sobě, nebo jeden stejný příkaz pětkrát zopakovat. Ve vizuálních programovacích jazycích, které se pro výuku programování mladších žáků používají, je navíc výsledek opakování vidět ihned. Podmínka s sebou obvykle nese nutnost použití dalších prvků, ať už příkazů, postav či proměnných (např. když se dotkneš červené barvy, udělej krok vpravo atd.), čímž se stává celá situace podstatně složitější. Při použití podmínky se navíc žák také poprvé setká s tím, že se příkaz někdy vykoná a jindy k jeho spuštění naopak vůbec nedojde.

---

[56] Celá učebnice je ke stažení zdarma ze stránek projektu iMyšlení.cz. Dostupné z: https://imysleni.cz/ucebnice/zaklady-programovani-ve-scratchi-pro-5-rocnik-zakladni-skoly [cit. 4. 1. 2024]

[57] Tyto materiály pro učitele spolu s doporučenými úvodními aktivitami jsou volně k dispozici na oficiálních stránkách Scratch, metodická příručka dostupná z: https://resources.scratch.mit.edu/www/guides/en/EducatorGuidesAll.pdf [cit. 4. 1. 2024]

[58] Příkladem budiž slovenský ŠVP, ve kterém se dočteme, že žák po dokončení druhého cyklu (5. třídy) dokáže „*vytvoriť program, ktorý vyžaduje známy počet opakovaní*“ viz MŠVVŠ. (2023). *Štátny vzdelávací program pre základné vzdelávanie.* Ve vzdělávacích standardech s. 51, v digitalizovaném dokumentu s. 452.

Část II
Praktická část práce

6. Úvod praktické části

Při tvorbě souboru lekcí jsem vycházel z poznatků získaných v teoretické části práce. Po přemýšlení nad koncepcí kurzu jsem se rozhodl držet teze J. Vaníčka, který říká, že výuka programování by měla obsahovat prvky jako aktivní učení, učení se děláním a konstruování znalosti jako výsledku tvořivé práce (viz kapitola 3.3, s. 12).[59]

Vaníček také pro rozvoj algoritmického myšlení klasifikuje dva základní typy programovacích úloh („etudy“ a „projekty“, viz kapitola 3.3, s. 12),[60] přičemž jsem dospěl k závěru, že kurz bude sestávat zejména z „projektů“. „Etudy“ formou krátké propedeutické aktivity budou součástí jen těch lekcí, kde to bude situace vyžadovat (seznámení s komplexnějšími koncepty apod.). Tvorbu větších „projektů“, jejichž cílem je produkt, jsem za hlavní náplň lekcí zvolil proto, že se jedná o delší aktivitu vhodnou pro výukové bloky, díky jejíž komplexnosti si navíc žáci mohou zopakovat hned několik získaných znalostí a navzájem si je propojit. Cílový produkt bude pro podpoření motivace žáků vždy hrou, která by měla být pro žáky zábavná a smysluplná, čímž se výsledný produkt stane i motivátorem do budoucna. Náplň lekcí bude strukturována tak, aby odpovídala definici informatického myšlení jako způsobu myšlení zaměřeného na popis problému, jeho analýzu a hledání efektivních řešení (viz kapitola 3.1, s. 10).[61]

Jako motivátor bude využit i prvek přiměřené výzvy (*moderate challenge*)[62], kdy hlavní projekty v lekcích budou koncipovány tak, aby se složitost jednotlivých projektů v průběhu kurzu postupně zvyšovala a zároveň aby si žáci vždy dokázali poradit alespoň s dílčími částmi projektu na základě znalostí získaných z předchozích lekcí (viz kapitola 3.3, s. 13).

[59] Vaníček, J., Šimandl, V., & Dobiáš, V. (2022). *Situational algorithmic tasks*, s. 103.
[60] TAMTÉŽ.
[61] To odpovídá definici představené v projektu iMyšlení.cz. Dostupné z: https://imysleni.cz/informaticke-mysleni/co-je-informaticke-mysleni [cit. 18. 1. 2024]
[62] Katai, Z. (2015). *Promoting algorithmic thinking*, s. 290.

7. Obecné informace ke kurzu

V této části bakalářské práce se nachází samotný kurz programování ve vývojovém prostředí Scratch. Kurz sestává z deseti lekcí, v každé z nich je hlavním cílem vytvořit funkční projekt. Tři lekce navíc obsahují i propedeutickou aktivitu, jejíž cílem je představit jednoduchou a hravou formou složitější koncepty (souřadnicová síť, klonování, seznamy), které žáci využijí při následné tvorbě projektu. Pro lepší motivování žáků bylo přistoupeno k tomu, že vytvářený projekt je vždy hrou. Komplexnost her se postupně s každou lekcí zvyšuje spolu s lepšími znalostmi žáků. V každé lekci se žáci naučí hned několika novým blokům ve Scratchi, přičemž lekce jsou za sebou řazeny tak, aby na sebe navazovaly. Snahou bylo co nejvíce se přiblížit ideálnímu stavu, kdy by se žáci v jedné lekci naučili používat nový blok a v té následující by tuto nově získanou znalost využili. Samozřejmě ne všechny bloky se využívají tak frekventovaně, aby na nich pro dosažení popsaného ideálního stavu mohly stát hned dva projekty po sobě (a nejlépe aby byly tyto bloky obsaženy i ve všech následujících projektech), proto kvůli praktičnosti došlo ke kompromisu. S některými nově naučenými bloky se tak žáci setkají znovu až přespříští lekci atp. Celý kurz byl ověřen ve výuce na dvou skupinách po devíti žácích. V první skupině byli žáci ve věku 7-9 let, v druhé 8-11 let, přičemž naprostá většina žáků obou skupin patřila do věkového rozmezí 8-9 let. Vyučovací blok probíhal jednou týdně a pokaždé trval 80-90 minut, tomu byla uzpůsobena i délka všech lekcí, aby bylo možné je vždy stihnout v časovém limitu jednoho výukového bloku.

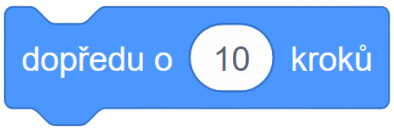
Kurz lze využít jako metodickou příručku pro výuku programování ve vývojovém prostředí Scratch. Každá lekce vždy obsahuje název projektu (hry), který by měl každý žák během výukového bloku vytvořit. Délka lekcí se různí na základě několika faktorů (komplexnost projektu, počet nových bloků atd.), délky se pohybují v rozmezí od 45 minut do 90 minut, přičemž průměrně jedna lekce zabere 80 výukových minut. Princip hry je krátce popsán v části „O projektu“, pod ním je pak možné nalézt ukázku ze hry v podobě obrázku. Následně jsou vždy popsány cíle lekce, zpravidla se jedná o výčet znalostí, které by si žáci měli projitím danou lekcí alespoň základně osvojit. Všechny bloky, se kterými se žáci v lekci setkají poprvé, jsou vypsány v části „Nově naučené bloky“. Poté následuje náplň lekce, ve které je krok po kroku popsána možná cesta k řešení. Popis je doplněn o ukázky kódu a různé vysvětlující obrázky. Obrázky považované za klíčové (návody, názorná vysvětlení) jsou navíc k dispozici v příloze s názvem Obrázky, pokud by je chtěl vyučující promítat. Obrázky jsou pojmenovány podle řadového čísla obsaženého v popisku pod obrázkem a rozřazeny podle lekcí. V krocích se též nacházejí poznámky doplňující výklad, popřípadě upozorňující na různá úskalí spojená s tvorbou kódu či s vysvětlováním vyučujícího. Nedílnou součástí některých lekcí je i kreslení (vlastní postavy, pozadí…) v editoru. To bývá poměrně časově náročné, proto je vždy možné v případě nedostatku času využít předpřipravené projekty (obsahují v názvu slovo Start, např. BIT_8_Dálnice_Start), které již obsahují všechny postavy a pozadí. Je žádoucí si proto ještě před začátkem kurzu rozmyslet, zda budete s žáky kreslit, či nikoliv,

a podle toho se řídit ve všech lekcích. Kreslení v pozdějších lekcích totiž počítá se základní znalostí editoru, v němž se žáci naučili pracovat v předchozích lekcích, není proto možné práci v editoru např. v lekci 2 přeskočit a v lekci 5 následně s žáky kreslit. V kroku „Úvodní diskuze“, který je součástí náplně každé lekce, lze vždy nalézt hypertextový odkaz na dokončený projekt. Projekt je nutné žákům ukázat a vysvětlit, jak daná hra funguje, aby všichni pochopili, co je cílem lekce. Některé lekce obsahují i výše zmíněné propedeutické aktivity, které je vhodné splnit ještě před představením projektu. Všechny tři aktivity a pár lekcí přímo vyžadují poskytnutí nějakého projektu žákům, držte se proto příslušných instrukcí (u aktivit je obvykle potřeba poskytnout připravený projekt, jehož kód je třeba dokončit). Kurz si neklade za cíl odhalit všechna možná řešení projektů, lekce tak vždy obsahuje pouze jedno z několika možných řešení. Případné další možnosti řešení jsou zpravidla popsány v poznámkách a doplněny o ukázky kódu s vysvětlením. Pokud k tomu hra přímo vybízí, obsahuje metodika i návrhy na možná rozšíření hry v případě, že by zbyl čas, návrhy lze využít i jako zadání úkolů pro rychlejší žáky. Na konci každé lekce je popsán i vývoj projektu spolu se zdůvodněním, proč u nich došlo v průběhu tvorby k některým změnám. Následuje detailní popis průběhu ověření lekce ve výuce a s ním i příslušný výčet úprav, k nimž došlo na základě reflexe ověření. Celý kurz včetně všech materiálů byl navíc zpřístupněn ke stažení na stránce https://kraken.pedf.cuni.cz/~benysekv/bit/.

8. Vymezení pojmů

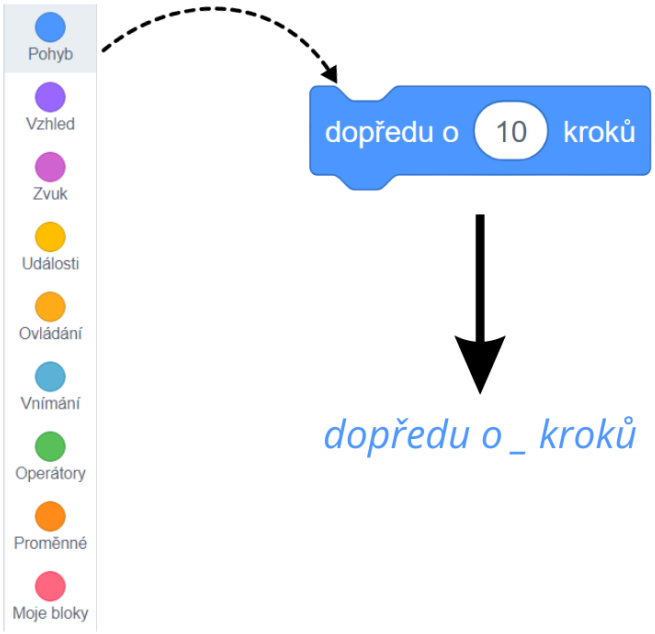
V kurzu se vyskytují některé pojmy a termíny, které se používají jako ustálená spojení pro konkrétní věci. Jelikož nemusí být na první pohled patrné, co přesně je jakým termínem myšleno, jsou níže základní a často se opakující pojmy vysvětleny.

- blok – základní jednotka blokového programování, prostřednictvím bloků se nastavuje chování postav, z bloků se skládáním dohromady sestavují sekvence; bloky jsou v textu kurzu odlišeny od zbytku tím, že jsou zapsány kurzívou a obarveny dle barvy příslušné kategorie bloků (viz definice kategorie níže)



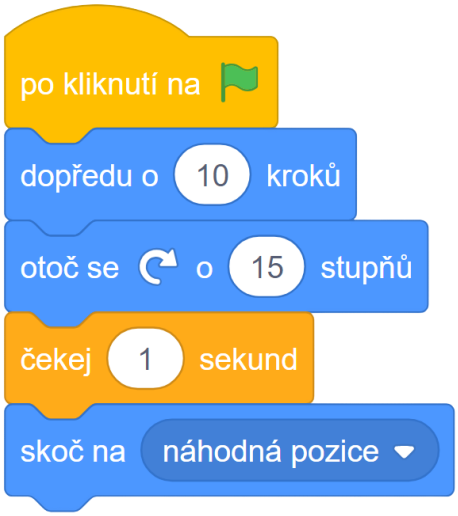
Obrázek 1 - Ukázka bloku

- kategorie – bloky jsou ve vývojovém prostředí Scratch rozděleny podle funkcí do kategorií bloků, každá kategorie má kromě názvu i svoji specifickou barvu, tou jsou bloky z dané kategorie obarveny v textu kurzu



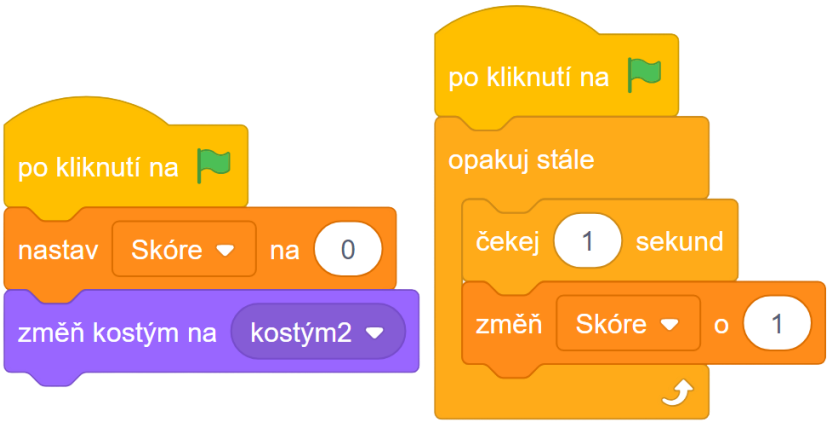
Obrázek 2 – Ukázka zapsání bloku v kurzu dle barvy příslušné kategorie

- sekvence – spojením více bloků dohromady vznikne sekvence, ta po spuštění vykoná jednotlivé bloky v takovém pořadí, jak jdou za sebou



Obrázek 3 - Ukázka sekvence

- kód – kódem se v kurzu myslí všechny sekvence a bloky, o kterých je zrovna hovořeno



Obrázek 4 - Ukázka kódu

- scénář – všechen kód (bloky a sekvence) jedné postavy

Obrázek 5 - Ukázka scénáře postavy

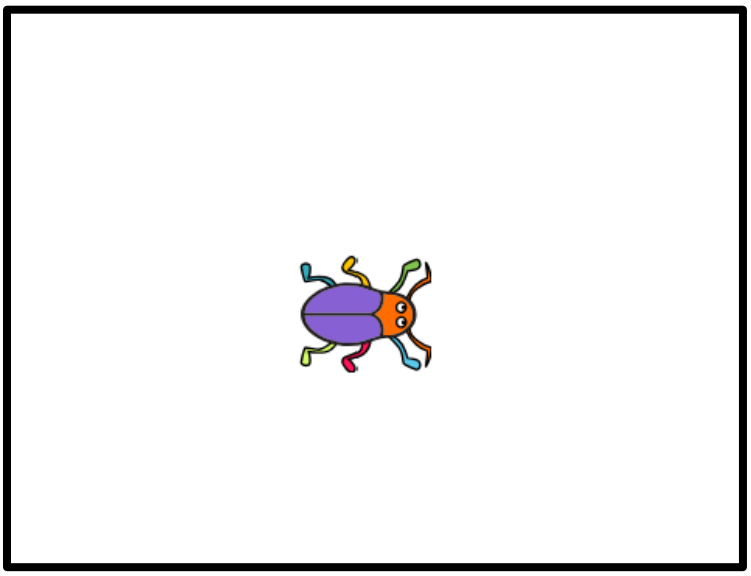
9. Soubor lekcí

Níže se nachází samotný soubor lekcí (kurz) obsahující základní informace (název, délka, obsah, cíl) všech 10 lekcí. Dále je součástí popisu i seznam nově naučených bloků a posléze formou návodu doplněného o metodické poznámky detailně popsána i náplň lekce.

9.1 Lekce 1 – Brouček

Délka: 45 minut

O projektu: Hráč hraje za broučka, kterým dokáže pomocí šipek pohybovat všemi směry.



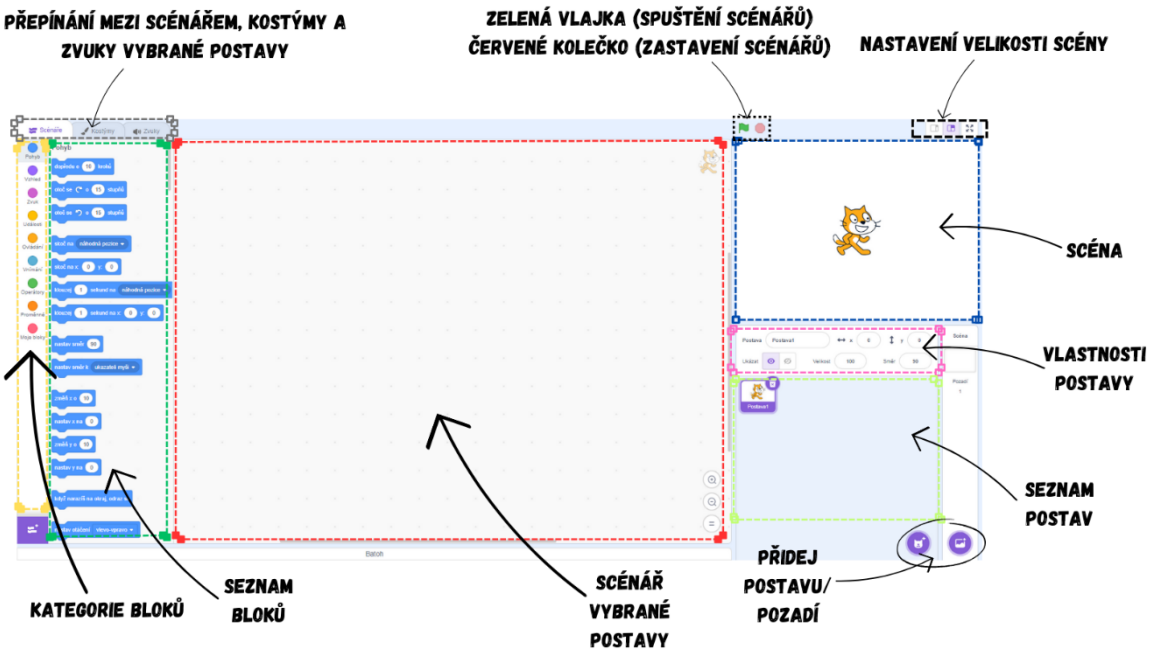
Obrázek 6 - Ukázka ze hry

Cíl lekce: Žáci se seznámí s vývojovým prostředím Scratch, začnou se seznamovat také s principy a podstatou programování a následně získané znalosti uplatní ve svém prvním projektu ve Scratchi. Osvojí si základní znalosti o blokovém programování a naučí se, jak fungují bloky pohybu a události. Vyzkoušejí si též opakovaný pohyb pomocí cyklu.

Nově naučené bloky: *dopředu o _ kroků*; *nastav směr _*; *po kliknutí na zelenou vlajku*; *po stisku klávesy*; *opakuj stále*

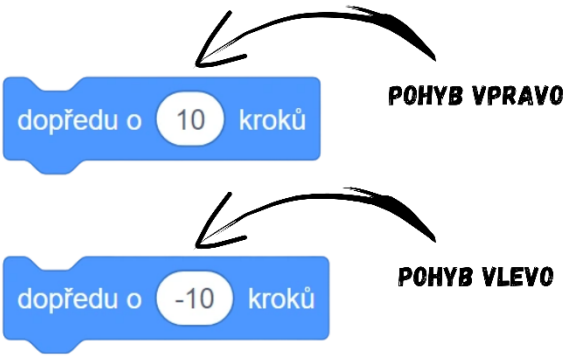
Náplň lekce:

1. Úvodní diskuze:
    - Představte žákům vývojové prostředí Scratch a osvětlete, proč ho používáme (prostředník mezi námi a počítačem). Projděte se žáky základní rozložení vývojového prostředí (scénář, kategorie bloků, seznam bloků, seznam postav, scéna, pozadí…)



Obrázek 7 - Ukázka vývojového prostředí Scratch

2. Vytvoření postavičky:
    - Použijte předpřipravenou postavu v knihovně Scratch (ideální pro začátek je *Beetle*).
3. První pohyb:
    - Přesuňte blok *dopředu o _ kroků* (nachází se v kategorii *Pohyb*) ze seznamu bloků do scénáře podržením levého tlačítka myši. Prozatím k vykonání bloku používejte myš. Klikněte na něj levým tlačítkem.
    - Ukažte žákům princip fungování bloku *dopředu o _ kroků* (větší číslo = větší krok apod.).
    - Opakováním klikáním na blok dojde postava na kraj a zarazí se. Takto zaraženou postavu lze přesunout, např. pomocí myši. Další možností je využít získané znalosti o bloku *dopředu o _ kroků*. Zkuste, zda žáci vymyslí, jak udělat couvání.



Obrázek 8 - Ukázka kódu

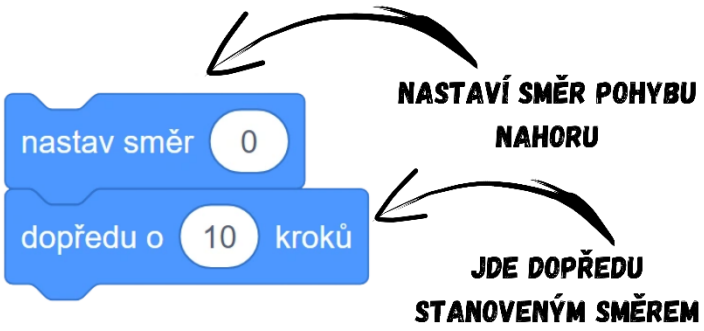
Poznámka: Je možné, že žáci bloky omylem spojí do sekvence. Ukažte jim proto, jak se spojují a rozpojují bloky ve Scratchi. Je třeba je upozornit, že táhnutí pomocí levého tlačítka myši za vrchní blok vyústí v přesun celé sekvence, táhnutí za bloky pod ním povede k rozdělení sekvence. Vhodné také může být již nyní ukázat, jak se bloky mažou, tzn. přetáhnout blok kamkoliv do seznamu bloků, více viz krok 7.

4. Otáčení:

   - Pro jednoduché otočení postavy použijte blok *nastav směr _* z kategorie *Pohyb* a ukažte žákům, jak funguje. Nejlepším způsobem je využít kružnici se šipkou po kliknutí na číslo v bloku.

Obrázek 9 - Ukázka kódu

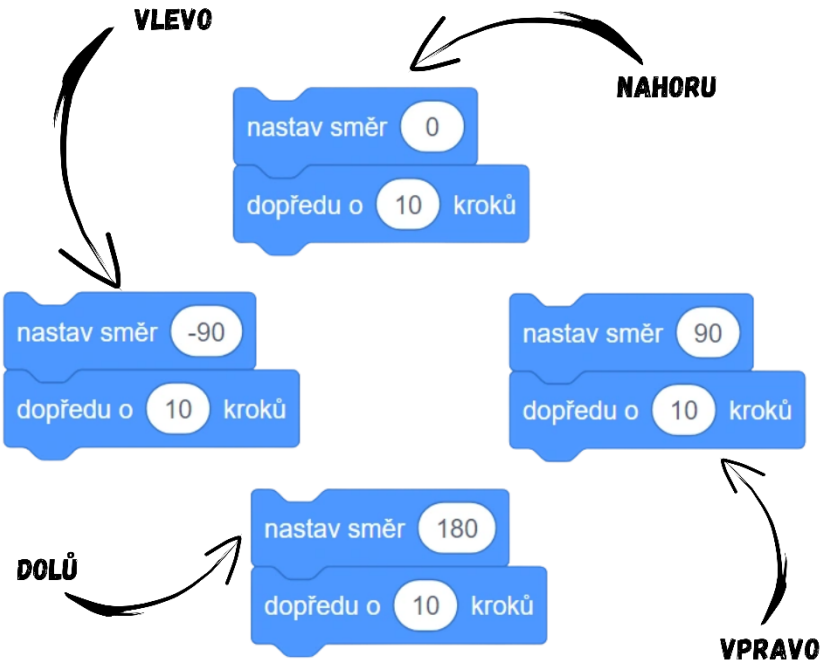
   - Spojte bloky *nastav směr _* a *dopředu o _ kroků* do jedné sekvence bloků. Jednotlivé bloky kódu sekvence se po kliknutí vykonají postupně, jak jdou za sebou. Pro pohyb nahoru by tedy blok měl vypadat následovně:



Obrázek 10 - Ukázka kódu

Poznámka: Je třeba poskládat bloky do sekvence v tomto pořadí. V opačném případě (první *dopředu o _ kroků* a druhé *nastav směr _*) by pohyb nefungoval tak, jak chceme. Postava by vždy nejdřív vykonala pohyb a pak by teprve nastavila nový směr. Při vysvětlování tohoto problému můžete udělat ukázku v reálném životě, kdy z jednoho žáka uděláte postavu **`Broučka`** a budete mu zadávat příkazy z bloků. Žák i všichni ostatní uvidí, že záleží na pořadí bloků v sekvenci a uvědomí si, proč je tedy jedna ze sekvencí chybná.

5. Pohyb všemi směry:
    - Zkuste nechat žáky samostatně vytvořit čtyři sekvence kódu skládající se ze dvou bloků pohybu pro všechny čtyři směry.

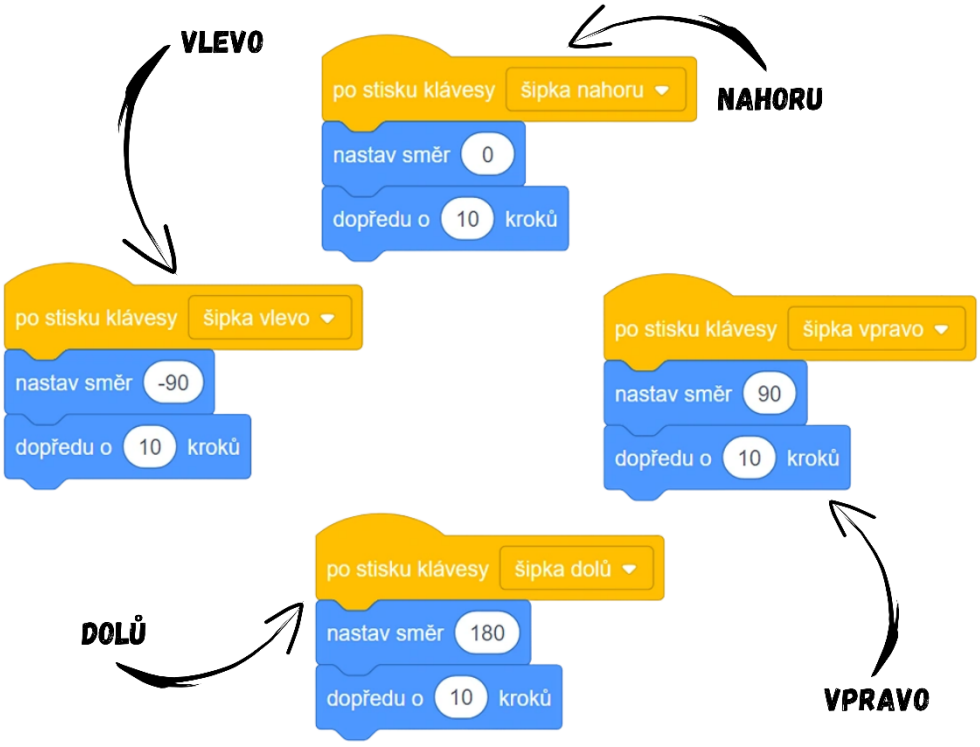


Obrázek 11 - Ukázka kódu

6. Vytvoření události:
    - Vysvětlete žákům princip fungování bloků z kategorie *Události*.
    - Použijte blok *po stisku klávesy _* a připojte k němu již vytvořenou sekvenci pro pohyb.

Obrázek 12 - Ukázka kódu

- Nechte žáky samostatně dokončit pohyb postavy. Výsledek by měl vypadat následovně:



Obrázek 13 - Ukázka kódu

Poznámka: Toto řešení lze nalézt v projektu BIT_1_Brouček_šipky.

7. Pohyb pomocí cyklu

    - Vysvětlete žákům, že existují další druhy pohybu a různé možnosti, jak postavu rozpohybovat. Jednou z možností je pohyb pomocí cyklů.

- Pokud bychom chtěli, aby se postava hýbala sama a hráč pouze určoval směr pohybu, budeme potřebovat cykly. Cykly najdeme v kategorii *Ovládání*, jsou to všechny bloky, které v sobě obsahují slovo „opakuj“, což je v podstatě hlavní účel cyklů. Cyklus dokáže opakovat bloky, které do něj vložíme, a to kolikrát budeme chtít, klidně i donekonečna. Nemusíme tedy dávat sekvenci stejných bloků za sebou, abychom zopakovali nějakou činnost. Stačí vzít jeden a vložit ho do cyklu.

Obrázek 14 - Ukázka kódu

- Pro samostatný pohyb využijeme cyklus *opakuj stále*, do kterého vložíme *blok dopředu o _ kroků*. Cyklus jednoduše zapneme kliknutím na něj, vhodnější by však bylo navázat jeho spuštění na nějakou událost. Pokud chceme, aby se nějaký blok spustil po zapnutí hry, je ideální navázat ho na událost *po kliknutí na zelenou vlajku*.

Obrázek 15 - Návod

- Celá sekvence by měla vypadat následovně:

Obrázek 16 - Ukázka kódu

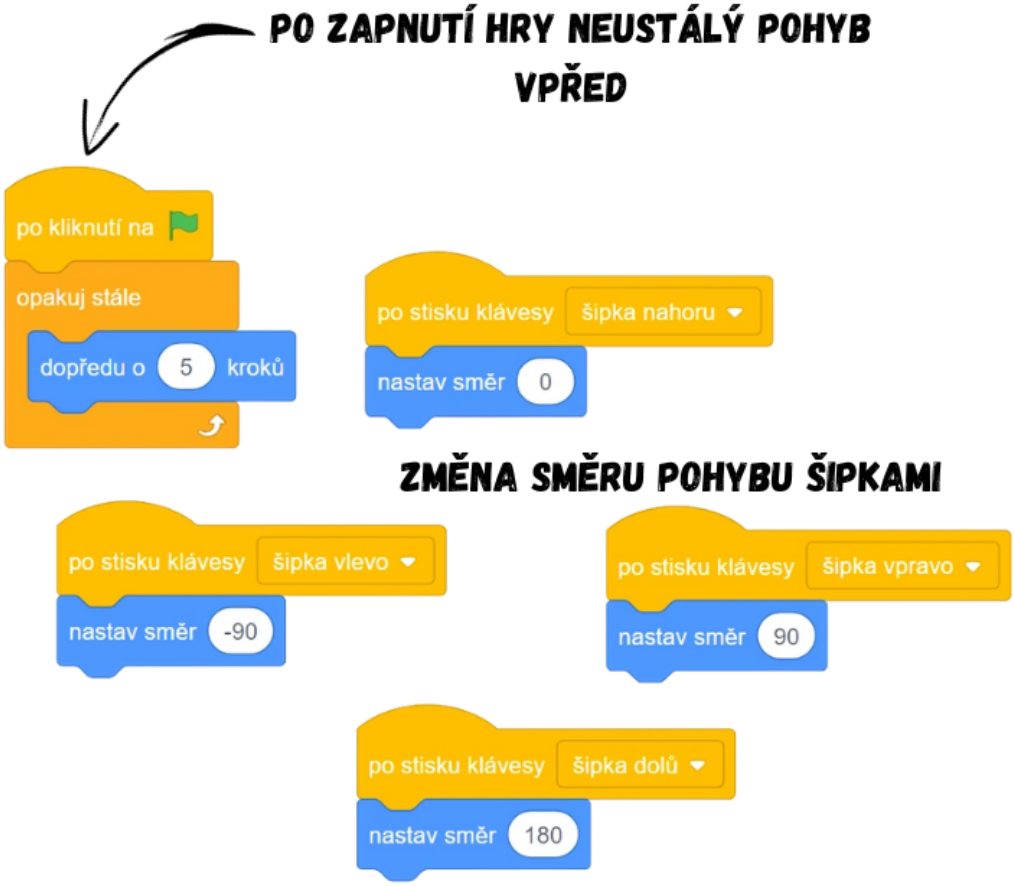
Poznámka: Někteří žáci po vyzkoušení hry zjistí, že postava se nyní pohybuje velmi rychle. Ukažte žákům, že rychlost postavy je ovlivněna počtem kroků. Nechte žáky vyzkoušet si, kolik kroků se jim zdá ideálních pro pohyb. Brzy dojdou k závěru, že blok *dopředu o 10 kroků* kvůli rychlému opakování cyklu dodává postavě až příliš velkou rychlost.

- Nyní už jen zbývá upravit sekvence následující po událostech *po stisku klávesy _*. Blok *dopředu o _ kroků* se zde stal nepotřebným, protože pohyb vpřed nám nyní zajišťuje blok vložený v cyklu *opakuj stále*. Smažeme proto nepotřebné bloky ze sekvencí přetažením zpět do seznamu bloků.

Poznámka: Některé žáky může napadnout vložit cyklus *opakuj stále* do všech čtyř sekvencí následujících po stisku šipek pro každý směr, popř. jim nemusí být jasné, proč stačí jeden cyklus *opakuj stále* a bloky *dopředu o 10 kroků* můžeme z ostatních sekvencí smazat. Pro vysvětlení celé situace můžete použít přirovnání s autem. Řekněte žákům, ať si představí, že **`Brouček`** je automobil. Cyklus *opakuj stále* s blokem *dopředu o 5 kroků* uvnitř představují motor vozidla, který zapneme otočením klíčků (zelenou vlajkou). Auto (**`Brouček`**) je poháněno dopředu zapnutým motorem (cyklus) a volantem (sekvencemi *po stisku klávesy _*) již pouze určujeme směr, kam se má auto (brouček) vydat. Jelikož nám pohyb zajišťuje motor (*opakuj stále*), není nutné, aby volant (*po stisku klávesy _*) také přidával plyn (*dopředu o 10 kroků*).

Obrázek 17 - Návod

- Konečný výsledek by měl vypadat takto:



Obrázek 18 - Ukázka kódu

Poznámka: Řešení pohybu pomocí cyklu lze nalézt v projektu BIT_1_Brouček_cyklus.

Vývoj: Zpočátku Brouček neobsahoval takto detailní postup řešení (krok po kroku), ale pro zlepšení přístupnosti, zejména z důvodu, že se s materiálem může setkat i vyučující, jenž má se Scratchem malé zkušenosti, bylo přistoupeno k názorným grafickým ukázkám a rozšíření textu. Původně projekt vůbec neobsahoval krok č. 7. Seznámení s cykly mělo původně proběhnout až v druhé a třetí úloze, ale kvůli velkému skoku, který by pro žáky nastal (druhá úloha by měla 9 nových příkazů), jsem se po konzultaci s vedoucím práce a kolegy rozhodl rozšířit projekt Brouček a přidat první použití cyklu už do něj, což se pak velmi hodí v druhé úloze, kde znalost cyklů velmi ušetří čas. Dalším důvodem pro toto rozhodnutí byla též krátkost prvního projektu, která nestačila na zaplnění celé výukové hodiny (Brouček vystačil na 25-30 minut), proto byl krokem 7 prodloužen. Díky úpravám tak Broučka lze využít jako samostatně stojící aktivitu pro celou první hodinu Scratche.

Průběh ověření: Žáky se podařilo úvodními otázkami a diskuzí (co je to programování) namotivovat a takříkajíc „naladit“ na správnou vlnu, což vyústilo v získání jejich pozornosti. Obě skupiny byly po vysvětlující prezentaci natěšené na Scratch. Aktivitou bylo procházeno krok po kroku, přičemž jsem

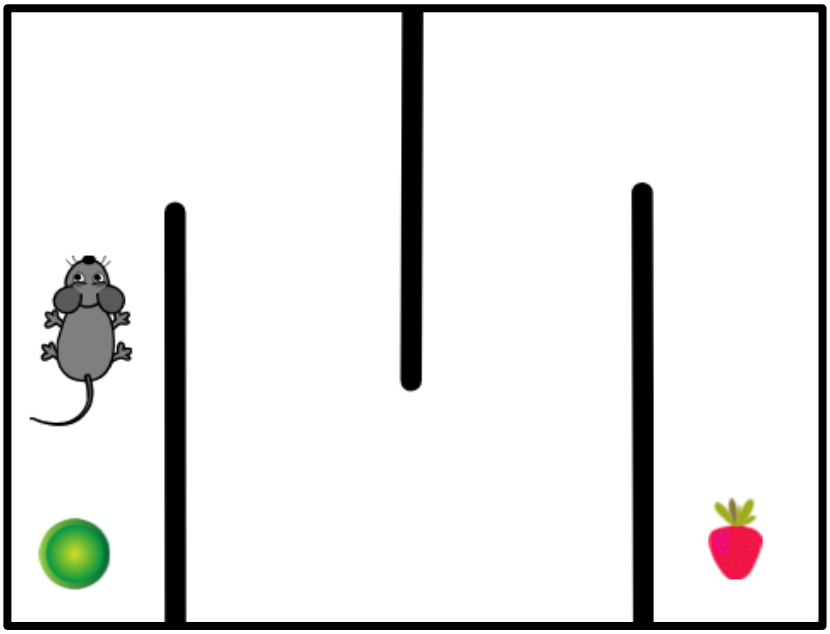
vždy ústně vysvětlil zadání dalšího cíle, kterého chceme dosáhnout a po vysvětlení věcí nezbytných pro řešení následovala diskuze o možném postupu k vyřešení problému. Žákům byly pokyny k úloze jasné, pokládali validní dotazy a navrhovali téměř vždy velmi konstruktivní a logická řešení. Přestože některý žák nalezl ideální řešení, vyslechl jsem i zbytek návrhů od dalších žáků a případně diskutoval o tom, v čem je jiné řešení lepší. Obě skupiny dokázaly samy vymyslet všechna řešení dílčích problémů, některé věci pochopili žáci až neočekávaně rychle (fungování cyklu). Jejich pozornosti neunikly ani některé nedostatky, na které bez jakéhokoli zásahu sami upozornili (pohyb pomocí cyklu je příliš rychlý, je třeba ho zpomalit zadáním nižšího počtu kroků do bloku). V průběhu celé aktivity zůstala pozornost obou skupin na vysoké úrovni, nicméně někteří jednotlivci se nechali strhnout nepřebernou řadou možností, které Scratch nabízí (žáci zkoušeli funkce různých bloků), proto bylo občas třeba některé věci zopakovat vícekrát, aby potřebné znalosti získali i tito jedinci. Žáky aktivita bavila a bylo vidět, že je blokové programování ve Scratchi oslovilo.

Úpravy po ověření ve výuce: Některým žákům se podařilo omylem spojit hned první dva bloky do sekvence, bylo proto přistoupeno k dřívějšímu vysvětlení principu spojování a rozpojování sekvencí, jež bylo přidáno do poznámky. Tato problematika se pak ještě spolu se smazáním bloku rozebírá více v kroku 7. Původně v metodice chyběly příklady, jak vysvětlit některé problémy, byly proto připsány ukázky možného vysvětlení (rozdíl mezi pořadím bloků v sekvenci na ukázce v reálném životě, pohyb pomocí cyklu opakuj stále jako motor automobilu).

9.2 Lekce 2 – Hladová myška

Délka: 70-90 minut

O projektu: Hráč hraje za myšku, která netrpělivě jde za jahůdkou. Myška vždy vychází ze zeleného kolečka. Hráč musí nasměrovat myšku při její chůzi správným směrem tak, aby nenarazila do překážky.



Obrázek 19 - Ukázka ze hry

Cíl lekce: Žáci si zopakují bloky pohybu a rozšíří si znalosti o tom, jak cykly fungují a v čem nám ušetří práci. Vyzkoušejí si též nový druh pohybu postavy (otáčení s využitím *otoč se o _ stupňů*) a naučí se základní práci v editoru pozadí.

Nově naučené bloky: *otoč se o _ stupňů*; *skoč na _*; *bublina _ _ sekund*; *čekej dokud nenastane _*; *zastav všechno*; *dotýkáš se _*; *dotýkáš se barvy*

Náplň lekce:

1. Úvodní diskuze:
    - Představte žákům zamýšlený projekt a ukažte jim, jak by měl výsledek vypadat, viz BIT_2_Hladová myška.
    - Diskutujte s dětmi o tom, s čím už si jsou schopny poradit díky svým znalostem z předchozí lekce a s čím si rady neví, co je tedy naopak nové.
2. Vytvoření postavičky:
    - Použijte předpřipravenou postavu v knihovně Scratch (např. *Mouse*).
3. Pohyb postavy hráče:
    - Obtížnost hry tkví v tom, že postava se automaticky neustále posouvá dopředu a hráč může pouze určovat směr otáčení, aby se vyhnul překážkám.

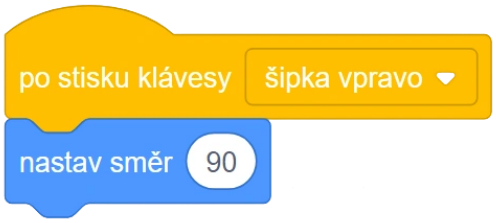
- Postava hráče se má od zapnutí hry hýbat neustále vpřed. Připomeňte žákům, k čemu jsou dobré cykly (nemusíme pořád klikat nebo něco spouštět ručně, cyklus bude automaticky opakovat bloky, které do něj vložíme).
- Naprogramujte pomocí cyklu *opakuj stále* nekonečný pohyb hráče. Dovnitř vložte blok *dopředu o _ kroků*. Počet kroků (tzn. jak moc má být postava rychlá) záleží na zamýšlené obtížnosti hry.

Poznámka: Žáci sami pravděpodobně upozorní na to, že je postava hráče příliš rychlá, často to zjistí až při výsledném zkoušení hry. Nechte žáky, ať na problém a řešení k němu přijdou sami, do té doby hodnotu klidně nechte na 10 krocích.

- Jelikož se má postava hráče začít hýbat od zapnutí hry, navážeme celý blok kódu k *po kliknutí na zelenou vlajku*.

Obrázek 20 - Ukázka kódu

- Pro otáčení využijte již známý blok *po stisku klávesy _*, nechte žáky, aby pomocí bloku *nastav směr _* naprogramovali otáčení všemi směry.



Obrázek 21 - Ukázka kódu

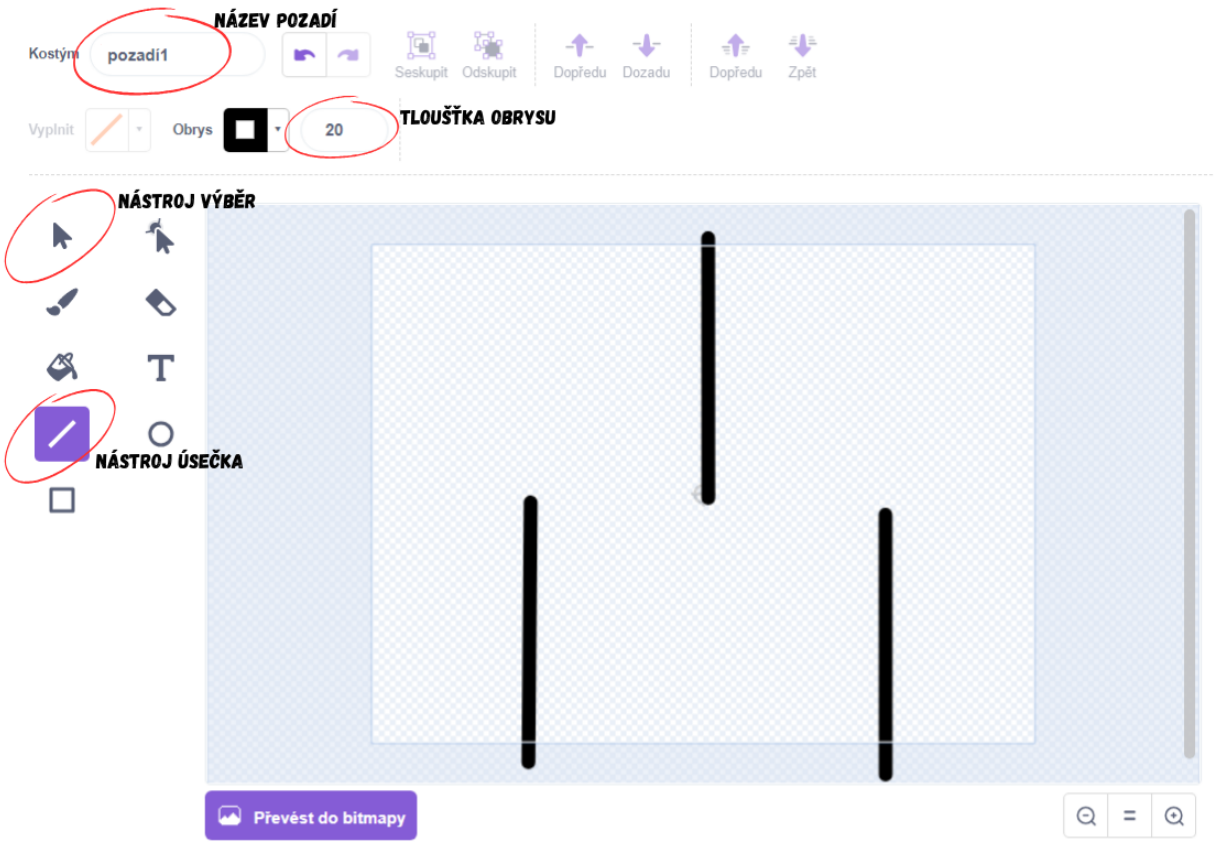
- Postava hráče se nyní otáčí buď vlevo, vpravo, nahoru, nebo dolů, ale nic mezi tím. Ve výsledném projektu se ovšem postava hráče otáčela postupně po malých kouscích. Pro postupné otáčení s každým stisknutím klávesy je ideální blok *otoč se o _ stupňů*. Před použitím bloku bude třeba žáky seznámit s úhly a stupni, výklad můžete opřít o prezentaci Úhly a stupně (příloha č. 5).
- Po vysvětlení stupňů začněte sestavovat s žáky kód. Programovat budeme už pouze dvě šipky, protože postavou budeme otáčet buď vlevo, nebo vpravo, tedy pouze dva směry. Jaký vhodný počet stupňů zadat do bloku *otoč se o _ stupňů* lze zjistit

postupným zkoušením, nechte proto žákům prostor. Zeptejte se žáků, zda potřebujeme sekvenci po stisku klávesy nahoru a dolu.

Obrázek 22 - Ukázka kódu

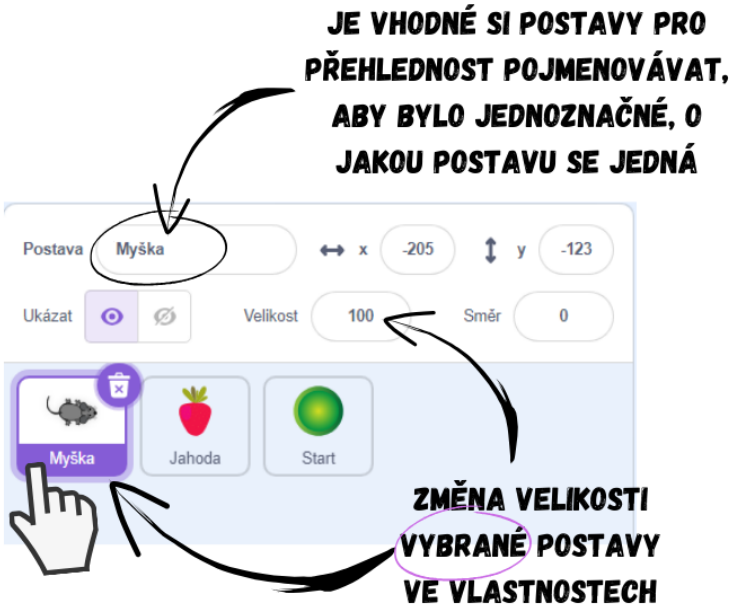
4. Nakreslení vlastního pozadí:

   - Pro vytvoření překážek použijeme editor obrázků. Je vhodné vytvořit nové pozadí a vkreslit překážky do něj.
   - Seznamte žáky s editorem. Ukažte jeho přednosti a slabiny. Vhodné může být představit i jednotlivé nástroje, zejména nástroj Úsečka, který hned využijete.
   - Nakreslete jednoduché vlastní pozadí pomocí nástroje Úsečka. Stačit by měly dvě nebo tři čáry pro vytvoření trasy (záleží na zamýšlené obtížnosti, důležité je, aby hra byla hratelná a ne nemožná, tzn. čáry nesmí být blízko, nesmí jich být mnoho, prostor mezi okrajem a koncem čáry musí být dostatečný pro průchod myši atd.), doporučuji nastavit vlastnost Tloušťka napravo od Obrysu na vyšší hodnotu, např. 20. Všechny nakreslené objekty je možné ještě následně upravovat pomocí nástroje Výběr.



Obrázek 23 - Návod

- Pokud je postava příliš velká pro projití trasy, je možné ji zmenšit upravením Velikosti ve vlastnostech postavy. Ideální hodnotu získáme zkoušením.

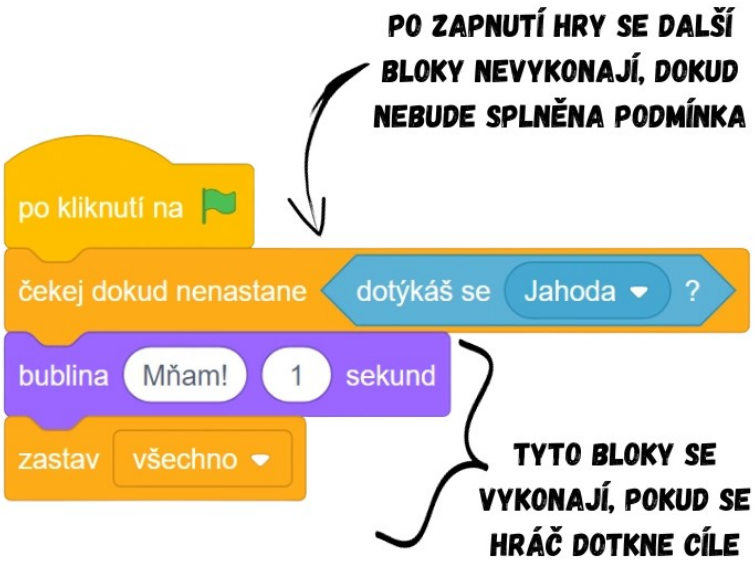


Obrázek 24 - Návod

- Možností je i v případě nedostatku času přeskočit tento krok (tvorbu pozadí) využitím projektu BIT_2_Hladová myška_Start, který je tak třeba žákům na začátku hodiny nasdílet, popř. dát ke stažení.

5. Výhra a porážka:

   - Pro výhru je třeba přidat do hry cíl prostřednictvím nové postavy jakožto odměny (ideální je *Strawberry* apod.). Při dotyku postavy hráče s cílem postava hráče řekne pomocí bloku *bublina _ _ sekund*, např. když se myška dotkne jahody, tak *bublina Mňam! 1 sekund*. Nejdříve ale musí dojít k samotnému dotyku, na který budeme po zapnutí hry (*po kliknutí na zelenou vlajku*) čekat. Vhodný je k tomuto např. blok *čekej dokud nenastane _*, do kterého lze vložit podmínka, v našem případě použijte *dotýkáš se _*. Podmínka *dotýkáš se _* se splní, pokud se daná postava dotkne postavy vybrané v bloku. Pro ukončení hry využijte blok *zastav všechno*, jehož funkce je obsažena již v jeho názvu.



Obrázek 25 - Ukázka kódu

Poznámka: Je třeba žákům zdůraznit, že blok *čekej dokud nenastane _* funguje jako každý jiný a je třeba ho nejdříve zapnout, aby fungoval (kliknutím/událostí). Někteří žáci si totiž mohou myslet, že sekvence pod podmínkou se spustí sama od sebe po jejím splnění.

   - Ukažte žákům, jak funguje blok *dotýkáš se barvy*, jenž využijeme pro naprogramování porážky, zejména je seznamte s nástrojem kapátko. *Dotýkáš se barvy* funguje stejně jako *dotýkáš se _*, akorát místo konkrétní postavy vybíráme barvu. Podmínka se vyplní, když se daná postava dotkne zvolené barvy.

Obrázek 26 - Návod

- Blok kódu porážky by měl být obdobný s výhrou, nechte proto žáky samostatně pracovat.

Obrázek 27 - Ukázka kódu

6. Restart hry

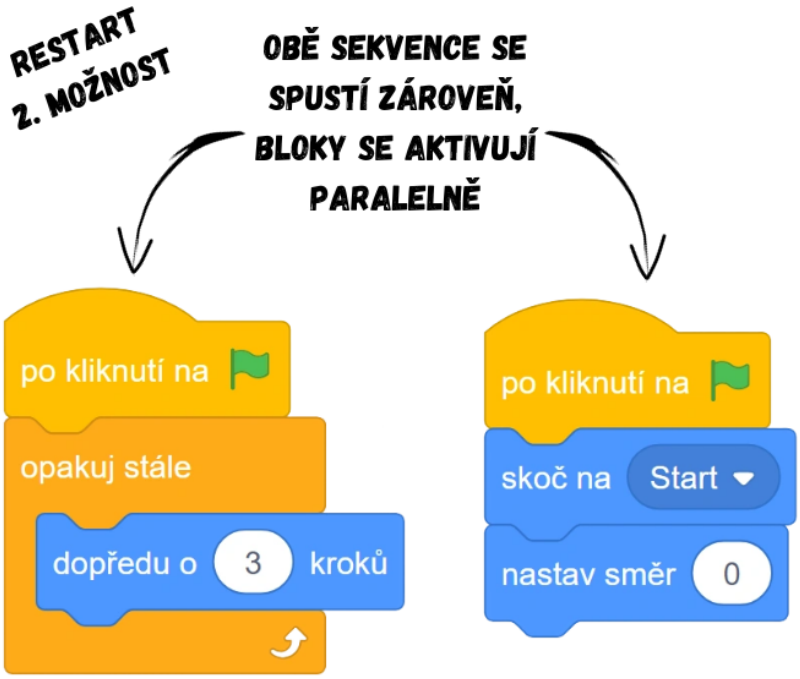
- Po dokončení hry (ať už výhrou, či porážkou) a opětovném zapnutí zjistíme, že postava hráče je stále na pozici, ve které byla hra ukončena. My ovšem chceme, aby při každém zapnutí hry postava hráče začínala na stejné startovní pozici. Pro funkčně správný restart je proto třeba přidat do hry novou postavu (např. *Button1*), která bude plnit funkci startovací pozice, na kterou se postava hráče po restartu vrátí. Postavu nazvěte třeba **`Start`** a přesuňte ji na vybranou pozici. Ve scénáři hráče pak pomocí

bloku *skoč na _*, ve kterém vyberete postavu **`Start`** označující startovací pozici, hráč skočí na vybranou postavu. Blok připojte k *po kliknutí na zelenou vlajku*.

- Musíme též nastavit počáteční směr postavy pomocí *nastav směr _*, aby se postava začala pohybovat vždy stejným směrem. Blok vložte k výše zmíněnému *skoč na _*.

Poznámka: Bloky vykonávající restart lze připojit k již existujícímu bloku *po kliknutí na zelenou vlajku* ve scénáři hráče, např. je možné vložit tyto bloky před cyklus *opakuj stále*. Je třeba žákům vysvětlit, že sekvence se ve Scratchi spouští paralelně a je tudíž jedno, zda vytvoříme kompletně novou sekvenci začínající blokem *po kliknutí na zelenou vlajku*, nebo bloky vložíme do již existující sekvence. Viz níže obě možnosti:

Obrázek 28 - Ukázka kódu



Obrázek 29 - Ukázka kódu

Vývoj: Projekt Hladová myška byl původně zamýšlen až jako třetí lekce, ve které se měla zopakovat práce s cykly, pohybovými bloky a naučit se kreslit vlastní pozadí. Nakonec byl po konzultaci s kolegy kvůli své jednoduchosti pro lepší plynulost kurzu vyměněn s projektem Hvězdička, který je nyní zařazen jako třetí v pořadí. I přes větší přístupnost projektu by se jednalo pro žáky o velký skok, lekce obsahovala 9 nových bloků, proto byly cykly přesunuty do prvního projektu. V kódu byly původně bloky obsahující souřadnice, které měly být lehce vysvětleny, ale díky vymyšlení jednoduššího řešení pro restart hry (skok na postavu Start) mohla být nakonec celá složitá problematika souřadnic odsunuta až na šestý projekt. Lekce byla rozšířena o detailnější návody, např. jak jednoduše nakreslit vlastní pozadí, či jak změnit velikost postav. Přidána byla též témata týkající se spíše technické stránky, kupříkladu otázka vhodnosti pojmenovávání postav (a proč je to důležité) či vysvětlení paralelního spouštění sekvencí ve Scratchi. Restart hry byl dříve součástí čtvrtého kroku, ale jelikož nutnost restartovat polohu postavy se plně prokáže až po naprogramování výhry a porážky (krok 5), byla tak sekvence restartu vydělena a dána jako samostatný krok následující až po pátém kroku.

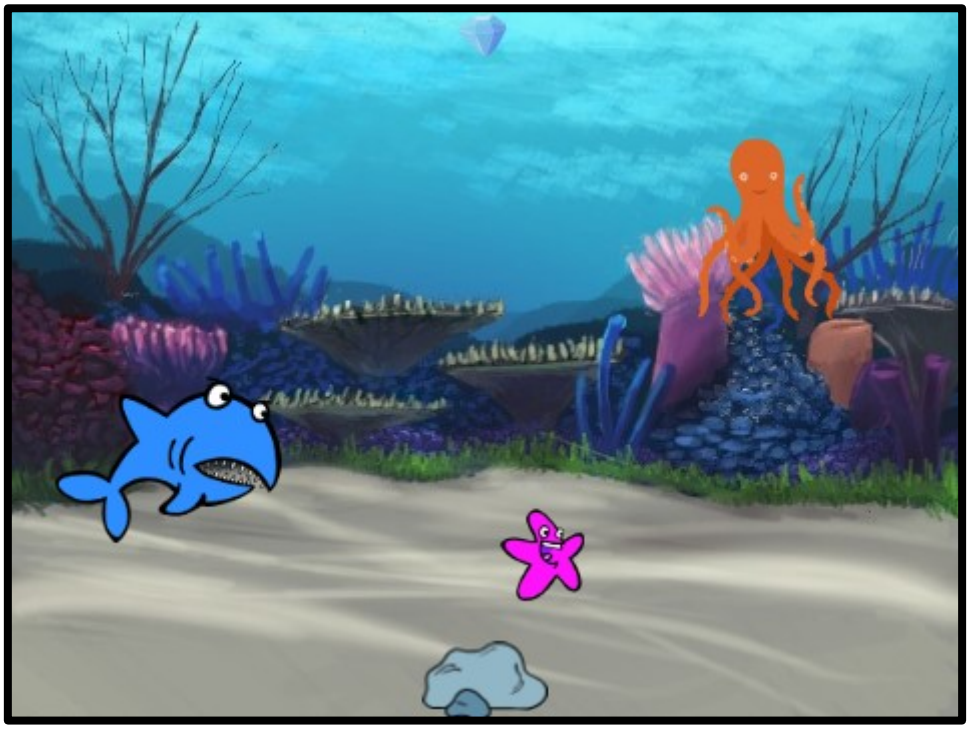
Průběh ověření: Obě skupiny pracovaly s úctyhodným nasazením, aktivita bez větších zádrhelů postupovala podle plánu. Jako velmi výhodné se ukázalo začít s cykly již v první lekci, poněvadž toto rozhodnutí vedlo k velkému ušetření času v této aktivitě. V obou skupinách si téměř všichni pamatovali přesný postup k vytvoření pohybu pomocí cyklu (uchytilo se označení „styl pohybu auto“), přičemž s programováním začali všichni téměř sami od sebe a pracovali samostatně, což vedlo k odložení vytváření pozadí na později. Jeden žák zopakoval chybu z předchozí hodiny a do každé sekvence vložil cyklus s *dopředu o 10 kroků*, tudíž jsem na problém znovu upozornil a vysvětlil ho. Posléze však již výuka obou skupin pokračovala bez přerušení, žáci velmi dobře pracovali v editoru, ve kterém jsem jim ukázal základní možnosti kreslení ve Scratchi a použitelné nástroje. Obě skupiny zvládly až překvapivě dobře práci s úhly, které jsem jim vysvětlil na kouscích pizzy viz prezentace Úhly a stupně. Dokázaly dopočítávat zadávané směry, odpovídat na cvičné otázky a uvědomit si využití stupňů ve Scratchi (*nastav směr _*, *otoč se o _ stupňů*). Stejně obstojně si žáci poradili i s bloky *dotýkáš se* a *čekej dokud nenastane _*. U podmínky bylo však třeba zdůraznit, že se nezkontroluje sama od sebe (což si někteří ve skupině mysleli), ale je třeba spustit sekvenci s podmínkou pomocí kliknutí či události stejně jako každou jinou sekvenci. Velmi se osvědčilo použít pro restart postavy místo skoku na souřadnice raději skok na novou postavu s názvem Start. Při výsledném zkoušení hry žáci zjistili, že pět kroků je příliš velká rychlost a hra je až moc obtížná, sami na problém upozornili a nalezli řešení (přestože v té době bylo v postavě hráče již několik sekvencí, věděli přesně, která z nich ovlivňuje pohyb a jakou hodnotu upravit).

Úpravy po ověření ve výuce: Po ověření bylo přistoupeno k prohození kroků 3 a 4. Žáci v obou skupinách po zadání projektu a vytvoření postavy hráče začali téměř automaticky programovat pohyb pomocí cyklu, tudíž tak nejdříve došlo k bývalému kroku 4 a až poté přišlo na kreslení vlastního pozadí. Z tohoto důvodu byly tyto dva kroky prohozeny, nejdříve se tedy vytvoří postava a naprogramuje její pohyb (nynější krok 3) a až posléze se tvoří pozadí (nynější krok 4). Byla též přidána poznámka ke zdůraznění nutnosti spouštět sekvence s podmínkami stejně jako jiné bloky. U malé části žáků došlo k přesvědčení, že podmínka se hlídá sama od sebe a její splnění vyústí ve spuštění celé sekvence. U kroku tři byla po zadání počtu kroků postavy hráče přidána poznámka obsahující doporučení využít tuto problematiku jako samostatnou činnost pro žáky a řešení jim neprozrazovat.

9.3 Lekce 3 – Hvězdička

Délka: 60-70 minut

O projektu: Cílem hry je dostat se k pokladu bez toho, aniž by se postava hráče dotkla nepřítele. Hráč hraje za mořskou hvězdici, která musí proplavat mořem k pokladu a vyhýbat se žralokům a olihním.



Obrázek 30 - Ukázka ze hry

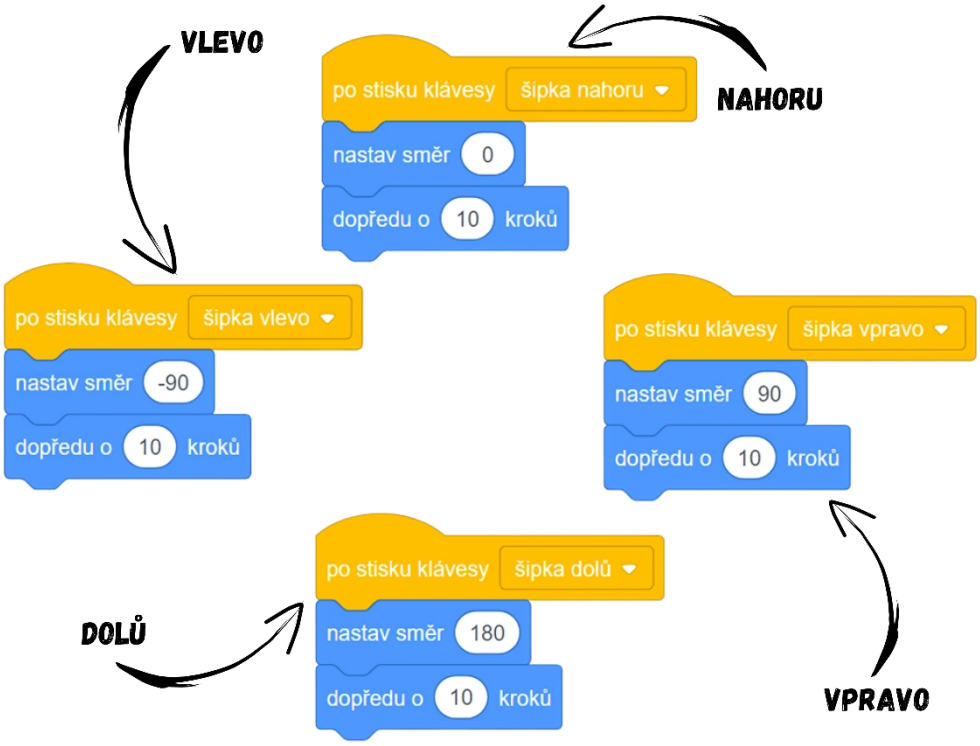
Cíl lekce: Cílem této lekce je zejména zopakovat s žáky získané znalosti z předchozího projektu. Žáci opakovanou praxí lépe pochopí, jak daný blok funguje, k čemu je dobrý a kdy je vhodné ho použít. Žáci se též naučí pracovat s více postavami najednou a naprogramovat interakci mezi nimi.

Nově naučené bloky: *když narazíš na okraj, odraz se*; *nastav otáčení _*

Náplň lekce:

1. Úvodní diskuze:
    - Představte žákům zamýšlený projekt a ukažte jim, jak by měl výsledek vypadat, viz projekt BIT_3_Hvězdička.
    - Diskutujte s dětmi o tom, s čím už si jsou schopny poradit díky svým znalostem z předchozích lekcí a s čím si rady neví, co je tedy naopak nové.
2. Vybrání pozadí:
    - Použijte předpřipravené pozadí podmořského světa z knihovny Scratch (např. *Underwater 1*/*Underwater 2*).
3. Vytvoření postavičky:

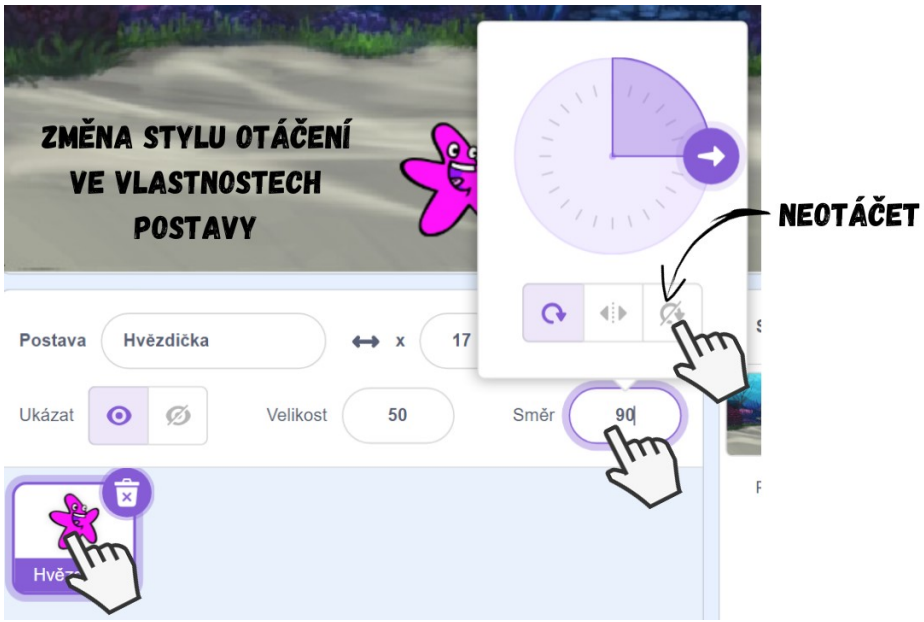
- Použijte předpřipravenou postavu vhodného mořského živočicha v knihovně Scratch (např. mořská hvězdice *Starfish*).
- Nechte žáky, ať zkusí na základě získaných zkušeností z předchozích projektů samostatně naprogramovat pohyb hráče všemi směry (bez cyklu).



Obrázek 31 - Ukázka kódu

- Po vyzkoušení pohybu postavy hráče zjistíme, že se postava nepřirozeně otáčí vzhůru nohama. Ukažte žákům nastavení otáčení (vlastnost Směr v seznamu postav, popř. blok *nastav otáčení _*) a změňte styl otáčení na vlevo/vpravo.

Obrázek 32 - Ukázka postavy ze hry



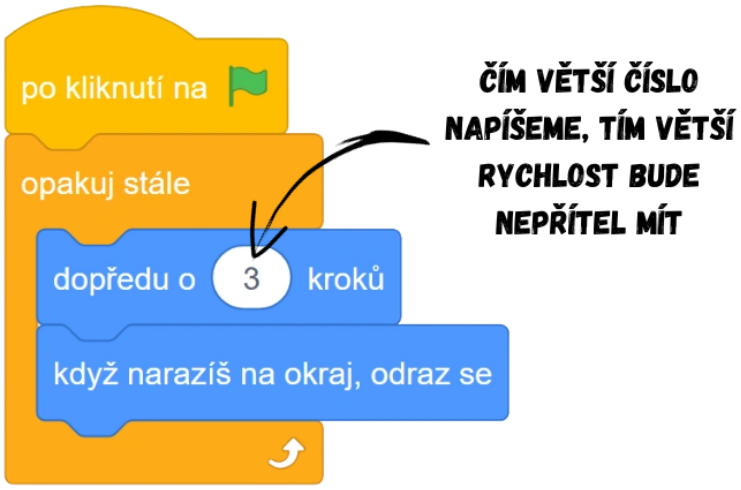
Obrázek 33 - Návod

4. Vytvoření nepřátel:

    - Přidejte do hry další postavu z knihovny Scratch (např. *Crab, Shark*, *Jellyfish…*). Začněte tím, že vytvoříte pouze jednoho nepřítele, teprve po jeho dokončení případně přidejte dalšího, ať se nerozmělňuje pozornost žáků do mnoha postav najednou.

Poznámka: Žáci se mohou leknout, že jim všechen dosavadní kód po vytvoření nové postavy zmizel. Zopakujte, že každá postava má svůj vlastní scénář, a tudíž je logické, že nová postava je prázdná a že dosavadní kód najdou v postavě hráče.

    - Pro neustálý pohyb nepřátel zleva doprava využijeme znalosti z předchozí lekce, zkuste žáky nechat vymyslet řešení. Do scénáře nepřítele přidejte cyklus *opakuj stále* s vnořeným *dopředu o _ kroků* a bloky připojte k *po kliknutí na zelenou vlajku*. Počet kroků (tzn. jak moc má být nepřítel rychlý) záleží na zamýšlené obtížnosti hry.
    - Pro odražení se od okraje použijte blok *když narazíš na okraj, odraz se*, jehož funkce je obsažena již v jeho názvu.



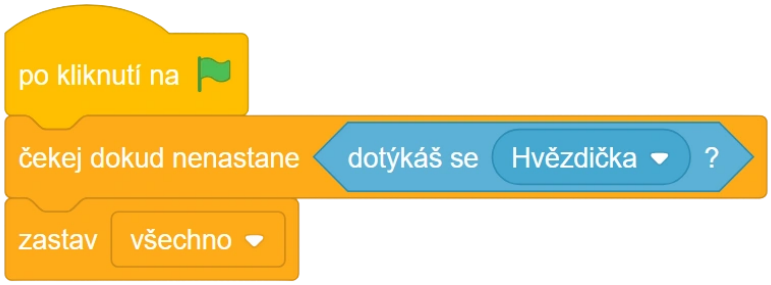
Obrázek 34 - Ukázka kódu

- Po vyzkoušení zjistíme, že se nepřítel nepřirozeně otáčí po odrazu vzhůru nohama. Nechte žáky využít získané vědomosti o nastavení a změnit styl otáčení na vlevo/vpravo (žáci by měli sami zjistit, že v případě nastavení stylu na neotáčet pohyb vypadá nereálně, protože nepřítel „couvá“ zpět, místo toho, aby se otočil).

Obrázek 35 - Návod

5. Výhra a porážka:

- V případě vzájemné kolize hvězdice s jakýmkoliv nepřítelem musí dojít k ukončení hry, což naprogramujeme v nepříteli. Po zapnutí hry (*po kliknutí na zelenou vlajku*) chceme, aby program hlídal, zda se hvězdice dotkla nepřítele. Použijeme blok *čekej dokud nenastane _*, do kterého lze vložit podmínka, v našem případě použijte *dotýkáš se _*. Pro ukončení hry využijte již známý blok *zastav všechno*.

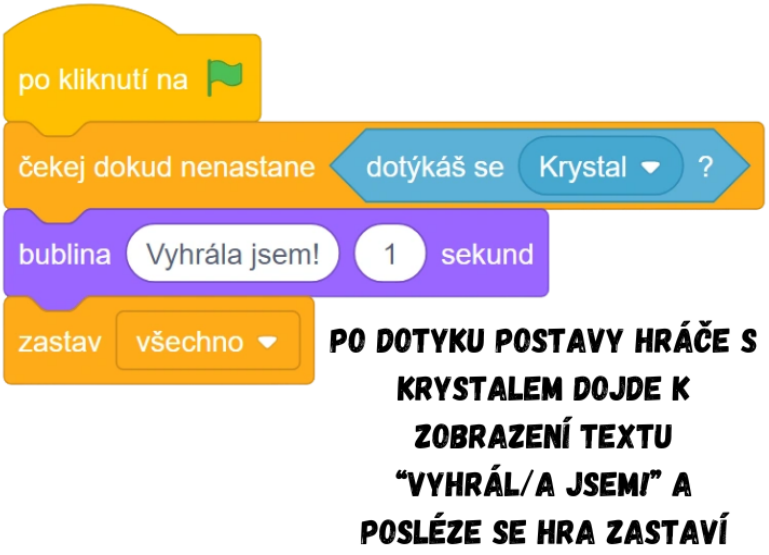


Obrázek 36 - Ukázka kódu

Poznámka: Prodiskutujte s žáky, do jakých postav je možné umístit sekvenci porážky. Položte jim otázku, zda je rozdíl mezi tím, jestli se bude hlídat v nepříteli dotyk s postavou hráče (viz obrázek výše), nebo v postavě hráče dotyk s nepřítelem. Pro vysvětlení názorně ukažte obě možnosti. Funkčnost je naprosto stejná, jediný rozdíl bude v příkazu *dotýkáš se _*.

- Pro případ výhry přidejte do hry novou postavu, např. *Crystal* z knihovny Scratch. Při dotyku hvězdice s touto novou postavou hvězdice „promluví“ pomocí bloku např. *bublina Vyhrál jsi! 2 sekund*. Nejdříve ale musí dojít k samotnému dotyku, na který budeme čekat. Postupujte obdobně jako u nepřítele, přidejte tedy do scénáře postavy hráče blok *čekej dokud nenastane _* a do něj vloženou podmínku *dotýkáš se _*. Dotyk si budeme zase hlídat po zapnutí hry, přidáme tedy celý blok kódu k *po kliknutí na zelenou vlajku*, lze již využít existující blok zařizující restart pozice. I v případě výhry nakonec dojde k ukončení hry pomocí bloku *zastav všechno*.

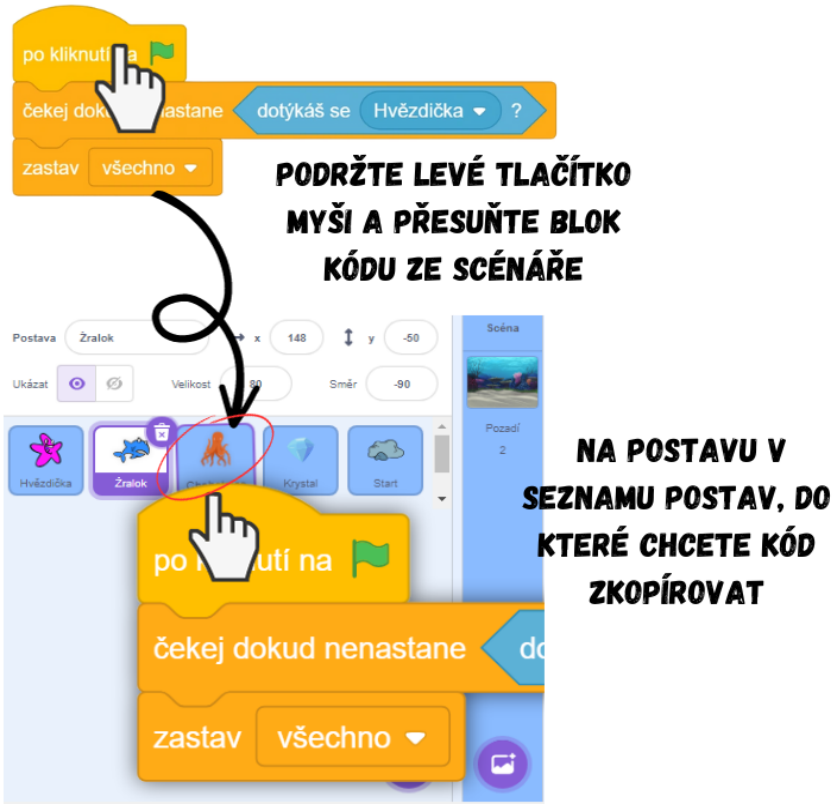
SEKVENCE VÝHRY:



Obrázek 37 - Ukázka kódu

Poznámka: Na rozdíl od porážky, jejíž sekvence se může vyskytovat jak v postavě hráče, tak v nepříteli, se sekvence výhry může vyskytovat pouze v postavě hráče. Je tomu tak z důvodu výherní promluvy „Vyhrála jsem!“, který říká mořská hvězdice, nikoliv třeba krystal, proto kód patří do postavy hráče.

Poznámka: Máme-li nepřátel více, je třeba mít kód pro porážku v každém z nich (v případě sekvence porážky v postavě hráče je třeba vytvořit sekvenci pro každého nepřítele). Žákům tak můžete ukázat funkci kopírovat kód. Podržením levého tlačítka myši na svrchním bloku kódu hýbete s vrchním blokem a všemi k němu připojenými, přesuňte celý kód na postavu v pravé části prostředí (seznam postav). Inkriminovaný kód se zkopíruje do vybrané postavy.

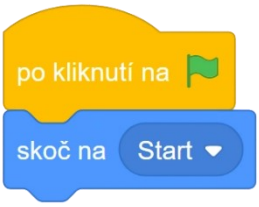


Obrázek 38 - Návod

Poznámka: Někteří žáci mohou namítat, že i po porážce stále mohou hýbat s postavou hráče. Vysvětlete, že tomu skutečně tak je, protože pohyb je navázán nikoli na zelenou vlajku, ale na stisk klávesy, na který je reagováno i po zastavení hry. Nicméně funkčnost hry není nikterak narušena, protože po zastavení hry nemůže dojít k výhře, takže je hráč stejně donucen k restartu.

6. Restart hry

   - Při každém zapnutí hry chceme, aby se postava hráče vrátila na startovní pozici. Situaci lze vyřešit podobně jako v předchozí lekci. Přidejte do hry novou postavu (např. *Rocks*), kterou pojmenujte **`Start`** a přesuňte ji na vybranou pozici. Do scénáře hráče poté vložte bloky *po kliknutí na zelenou vlajku* a *skoč na _*.



Obrázek 39 - Ukázka kódu

Poznámka: Sekvenci restartu lze vytvořit jako samostatnou a oddělenou od ostatních (viz obrázek výše), ale možností je využít již existující sekvence, např. vložit blok *skoč na Start* i na začátek sekvence výhry. Zeptejte se žáků, jestli je mezi těmito možnostmi nějaký funkční rozdíl a posléze vysvětlete, proč ne (paralelní spouštění sekvencí).

Vývoj: Projekt Hvězdička se původně nalézal v pořadí lekcí na pozici Hladové myšky, mělo se jednat o první projekt s cykly, kvůli své složitosti byl však odsunut na třetí pozici. Díky tomuto prohození byl počet nově naučených kroků zredukován a Hvězdička se tak stala zejména projektem vhodným pro zopakování a rozšíření předchozích znalostí. Hra původně obsahovala mechanismus restartu postavy pomocí skoku na souřadnice, podobně jako u Hladové myšky bylo po konzultaci s kolegy přistoupeno na jednodušší verzi prostřednictvím nové postavy sloužící jako startovací pozice. Restart byl dříve součástí kroku 3, pro jasnější uvědomění si nutnosti postavu hráče po každém zapnutí hry vrátit zpět na start byl restart přeměněn na samostatný krok a přidán na závěr. K úpravě došlo i v rovině stylistické, např. Hvězdička nyní říká "Vyhrála jsem!" místo "Vyhrál jsi!" pro lepší propojení žáků s postavou.

Průběh ověření: Obě skupiny dorazily značně motivované a natěšené, aktivita tak postupovala podle plánu. Žáci si dokázaly po celou dobu výuky udržet vysokou úroveň pozornosti, zejména v první skupině. Jelikož má projekt Hvězdička především za cíl zopakovat a prohloubit dosavadní znalosti žáků, celá aktivita byla pojímána spíše formou samostatné práce, kdy jsem kladl dotazy, co mám dělat dál a žáci mi radili. Společně tak tvořili postup k vytvoření celé hry, já byl pouze v pozici jakéhosi mentora dohlížejícího na skupinu. Práce s pohybovými bloky i cykly probíhala bez problému (žákům pomáhá rozlišení druhů pohybu na „manuální" a „auto", tedy bez využití cyklu a s využitím), obě skupiny samy nalezly nový blok *když narazíš na okraj, odraz se*, když ho potřebovaly. Žáci si věděli rady i se zrychlením/zpomalením pohybu nepřátel. Při vysvětlování stylů otáčení nedávaly ve druhé skupině dva žáci pozor, proto se v následujících minutách nezávisle po sobě zeptali, jak zabránit otáčení vzhůru nohama. Dal jsem prostor ostatním žákům, aby řešení znovu vysvětlili, posléze tak už všichni věděli, co v tomto případě dělat.

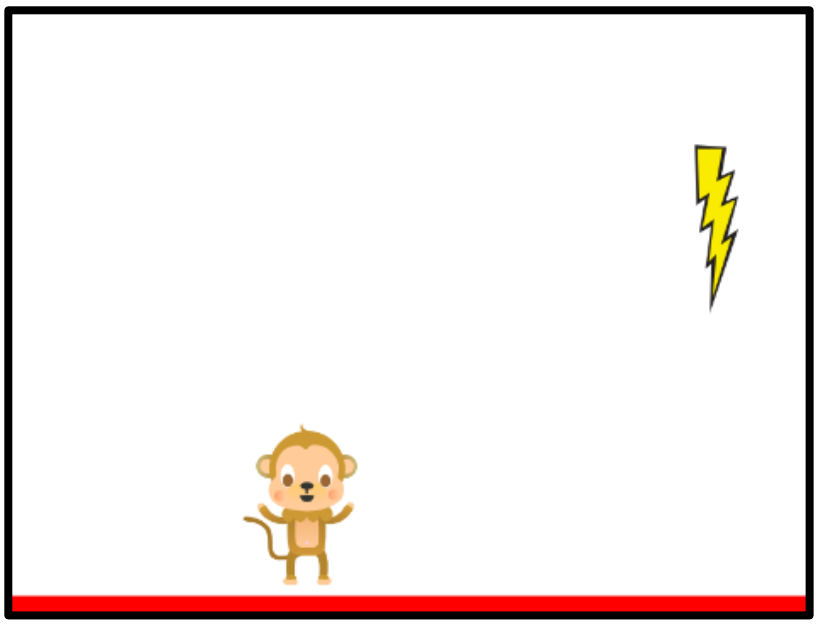
Na základě postupu žáků a jejich schopností samostatně řešit některé problémy lze usuzovat, že u nich došlo v průběhu lekcí k osvojení základních algoritmických konceptů (pohybové bloky, cyklus, podmíněný příkaz *čekej dokud nenastane _*) a k rozvoji jejich algoritmického myšlení.

Úpravy po ověření ve výuce: Žáci si všimli nevhodného otáčení postavy již při vytváření pohybu pro postavu hráče. Začali se dotazovat, jak situaci řešit, tudíž došlo k přidání vysvětlení stylů otáčení již u kroku 3. Rozšířen byl popis vytváření nepřítele, do více kroků byly přidány poznámky poukazující na možné reakce žáků na určité problémy, které navíc obsahují návod, jak k nim přistoupit a jak je žákům vysvětlit.

9.4 Lekce 4 – Opička

Délka: 80-90 minut

O projektu: Hráč hraje za Opičku. Cílem hry je, aby se Opička vyhnula všem bleskům, které na ní padají. V případě, že opička nestačí padajícímu blesku uhnout a je zasažena, hra končí.



Obrázek 40 - Ukázka ze hry

Cíl lekce: Žáci si procvičí práci s cykly a programování kolizí (dotyku) postav. Naučí se používat podmínky a osvojí si, jak udělat pohyb postavičky plynulejší než v předchozích úlohách.

Nově naučené bloky: *když _ tak*; *klávesa _ stisknuta?*

Náplň lekce:

1. Úvodní diskuze:
    - Představte žákům zamýšlený projekt a ukažte jim, jak by měl výsledek vypadat, viz BIT_4_Opička.
    - Diskutujte s dětmi o tom, s čím už si jsou schopny poradit díky svým znalostem z předchozích lekcí a s čím si rady neví, co je tedy naopak nové.
2. Vytvoření hráče:
    - Pro postavu hráče použijte předpřipravenou postavu v knihovně Scratch (např. *Monkey*).
3. Cyklus plynulého pohybu:
    - Nechte žáky, aby naprogramovaly pohyb hráče vlevo/vpravo tak, jak umí. Diskutujte nad nevýhodami pohybu pomocí bloku *po stisku klávesy _*. Žáci by měli dospět k závěru, že pohyb je neobratný a reakce na stisk klávesy pomalá.

Poznámka: Pohyb postav prostřednictvím událostních bloků *po stisku klávesy _* není příliš plynulý. Při držení klávesy dojde místo plynulého pohybu ke krátkému záseku a až po něm k pohybu daným směrem. Jedná se o tzv.

keyboard repeat delay. Pro ukázku můžete otevřít například textový editor, ve kterém po podržení klávesy dojde k napsání jednoho písmena, ke krátké pauze, a pak teprve k rychlému psaní písmen za sebou. Obdobně funguje i pohyb ve Scratchi při použití událostního bloku *po stisku klávesy _*.

- Pro vytvoření plynulého pohybu využijeme jiných bloků. Představte žákům podmínky, jak fungují a k čemu slouží. Výklad můžete založit na principu již známého bloku *čekej dokud nenastane _*.
- Začněte tvořit kód s *když _ tak*, do kterého vložíte podmínku z kategorie *Vnímání*, konkrétně *klávesa _ stisknuta?*. Poukažte na podobnost tohoto kódu s blokem *po stisku klávesy _*. Výhodou *když _ tak* je, že ho můžeme ještě vnořit do částí jiného kódu, např. do cyklů.

Obrázek 41 - Ukázka kódu

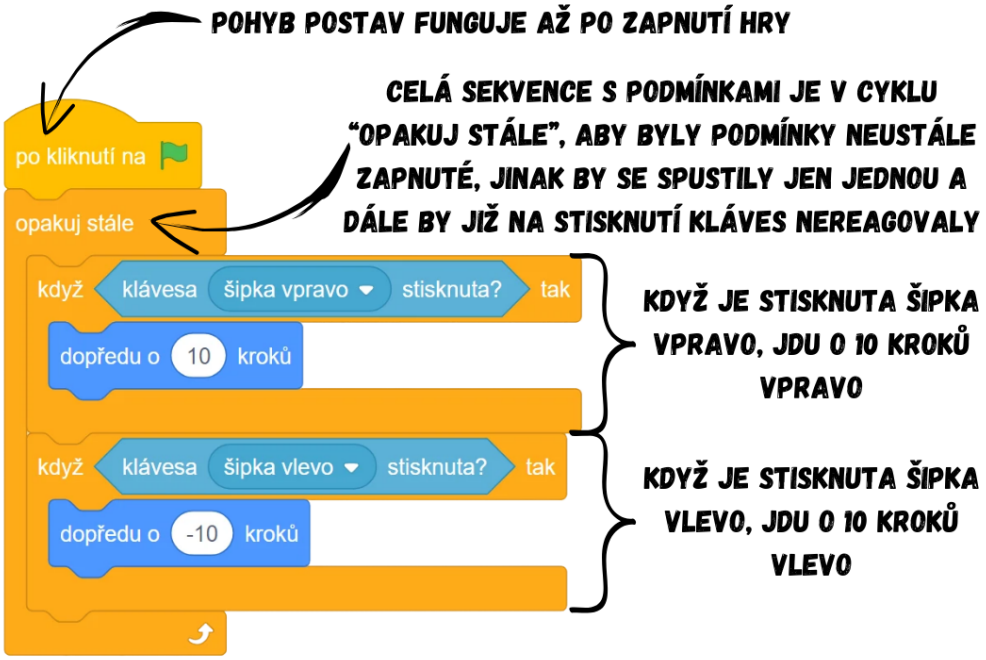
- Do podmínky vložte bloky *dopředu o _ kroků*. Při směru postavy 90 platí, že o 10 kroků = doprava, o -10 kroků = doleva. Po přidání postavy do hry má standardně nastavený směr 90 každá postava.

Poznámka: Žáci mohou být z předchozích úloh zvyklí na použití *nastav směr* před *dopředu o _ kroků*. Přestože se jedná o jedno z možných řešení, je dobré jim ukázat, že pokud vytváříme hru, ve které se postava hýbe jen dvěma směry (v tomto případě vlevo/vpravo), mohou využít znalosti o pohybu vpřed a o couvání (záporné hodnoty). Zásadní je nastavit směr postavy na 90, jinak hodnoty +10 a -10 nebudou fungovat tak, jak bylo zamýšleno. Dejte pozor na to, že někteří žáci mohou mít nastavený směr postavy na -90, celá sekvence pak funguje obráceně.

Obrázek 42 - Ukázka kódu

- Vysvětlete žákům, že sekvence s podmínkou se spustí po námi zvolené události, ideální je např. *po kliknutí na zelenou vlajku*. Nezapomeňte celou podmínku vnořit do *opakuj stále*.

Poznámka: Častým úskalím bývá, že žáci zapomenou na cyklus *opakuj stále*. Ukažte jim, co se stane bez cyklu a vysvětlete, proč se tak děje. Je důležité si uvědomit, že standardně se podmínka zkontroluje jenom jednou a posléze už se nehlídá (není spuštěná, sekvence končí), proto ji musíme nechat hlídat po celou dobu hry pomocí nekonečného cyklu.



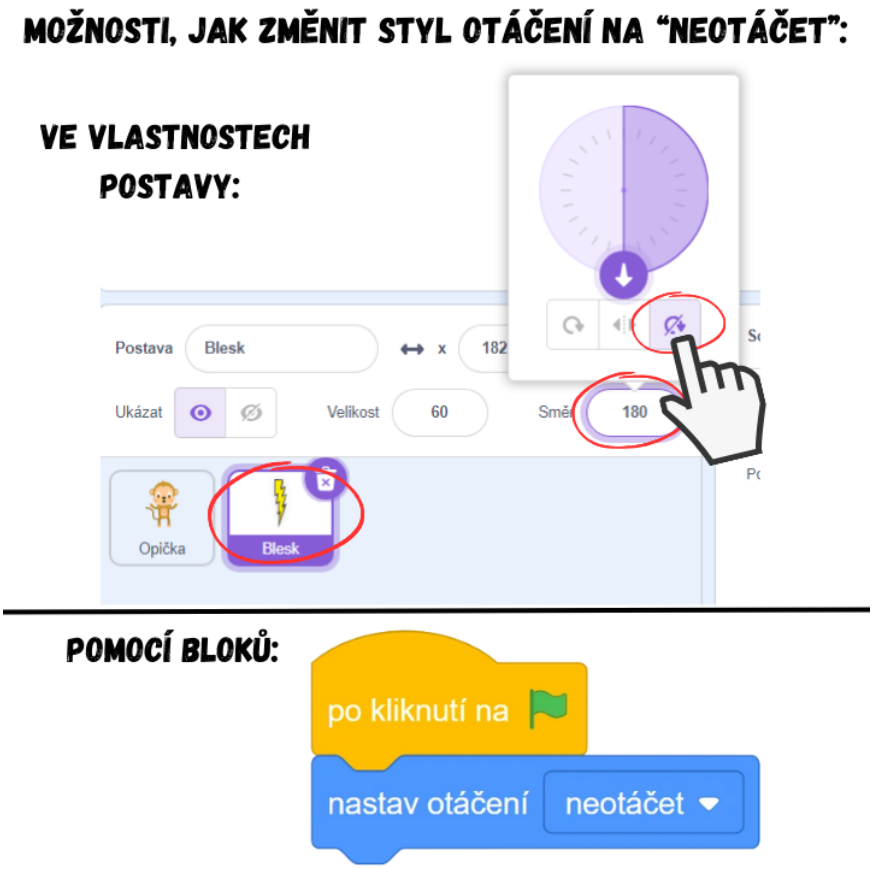
Obrázek 43 - Ukázka kódu

Poznámka: Spuštěnou sekvenci lze poznat tak, že kolem ní svítí žlutá barva. V případě, že by v sekvenci chyběl cyklus *opakuj stále*, po zapnutí hry by sekvence jen žlutě problikla. To znamená, že se spustila, rychle proběhla a pak skončila. V případě, že podmínka hlídá např. stisknutí klávesy, je nutné, aby tak činila po celou dobu hry, ne pouze jednou a dost.

Poznámka: Někteří žáci mohou celou situaci řešit tak, že si vytvoří pro každý směr jednu sekvenci. Jedná se o naprosto validní možnost řešení, při které dojde k využití paralelního spouštění sekvencí. Po zapnutí hry se obě sekvence spustí a pohyb tak bude fungovat stejně, jako v případě jedné sekvence obsahující dvě podmínky.

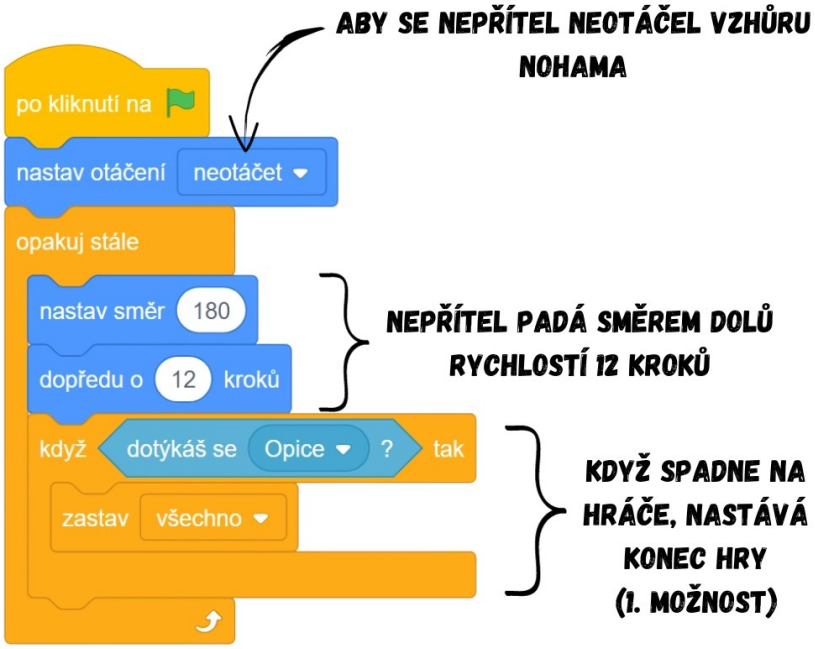
4. Vytvoření nepřítele:
   - Pro postavu nepřítele použijte předpřipravenou postavu v knihovně Scratch (např. *Lightning*).
   - Nepřítel se má náhodně objevit někde nahoře a spadnout dolů. Celá situace se opakuje až do konce hry, tj. do zasažení hráče.
   - Pro pohyb dolů využijte již známou kombinaci bloků *nastav směr _* a *dopředu o _ kroků*. Aby se nepřítel neotáčel vzhůru nohama, nastavte jeho styl otáčení na neotáčet

(buď ve vlastnosti postavy Směr, popřípadě pomocí bloku viz obrázek níže). To celé patří do cyklu *opakuj stále* napojeného na *po kliknutí na zelenou vlajku*.



Obrázek 44 - Návod

- Když se postava hráče dotkne nepřítele, dojde k ukončení hry. Pro naprogramování této kolize můžeme využít buď již známé *čekej dokud nenastane _*, nebo využít již existujícího cyklu v nepříteli a přidat do něj podmínku *když _ tak* s *dotýkáš se _*. V obou případech hra skončí blokem *zastav všechno*. Scénář nepřítele (i s oběma možnými variantami bloku pro porážku) by mohl vypadat následovně:

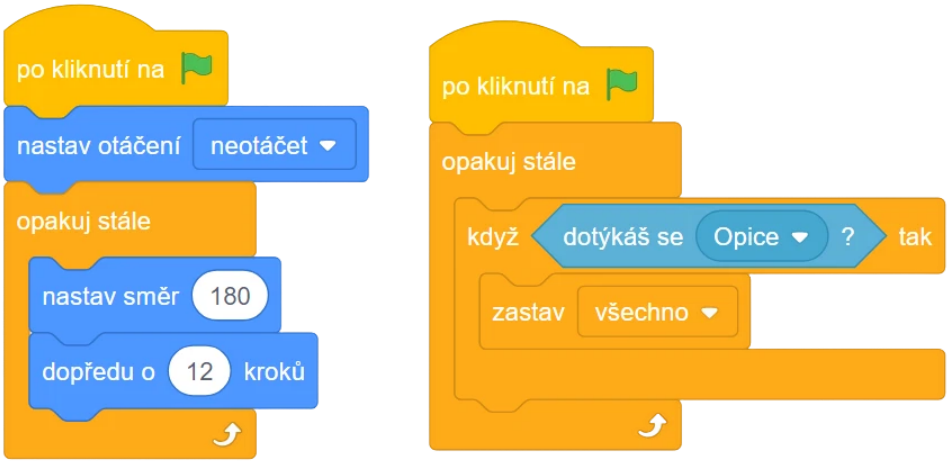


Obrázek 45 - Ukázka kódu

Poznámka: Z hlediska efektivnosti kódu by bylo lepší blok *nastav směr 180* vložit před cyklus opakuj stále, ovšem žáci jsou z předchozích projektů zvyklí, že pro pohyb určitým směrem používají bloky *nastav směr _* a *dopředu o _ kroků* vždy pospolu, proto je vhodnější nechat žáky vložit *nastav směr 180* dovnitř cyklu. V případě, že bychom vložili blok *nastav směr 180* před cyklus, museli bychom ho později přidat ještě jednou do podmínky resetu pozice **`Nepřítele`**, viz krok 5.

Obrázek 46 - Ukázka kódu

Poznámka: Díky paralelnímu spouštění sekvencí by mohla být i první možnost pro dotyk s postavou hráče vytvořena separátně jako samostatná sekvence. Je vhodné ukázat žákům, že nezáleží na tom, zda vložíme sekvenci s podmínkou do jednoho cyklu s pádem nepřítele, či budeme mít sekvence dvě (jedna na pád, druhá na porážku).



Obrázek 47 - Ukázka kódu

5. Dokončení hry:

   - Nyní po zapnutí hry ovšem nepřítel spadne na zem jen jednou. Je třeba proto určit hranici (spodní okraj), po jejímž dosažení nepřítel vyskočí zpět nahoru a začne znovu padat, tentokrát však z jiného místa.
   - Existuje mnoho způsobů, jak si určit tuto hranici. Možností je např. přidat do hry novou postavu jakožto podlahu („zem“), jejíž kostým bude sestávat jen z čáry, popřípadě využít již předpřipravenou postavu *Line*. Postavu vhodně pojmenujte, třeba **`Linie`**.

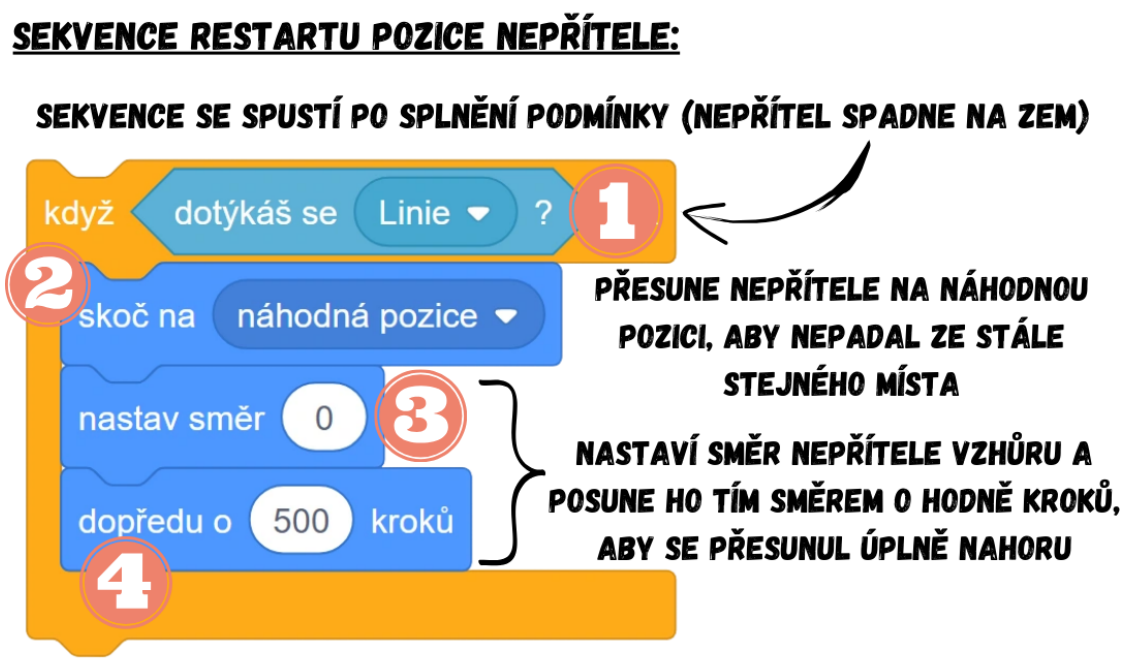
Poznámka: Celá situace by šla řešit prostřednictvím souřadnic, s těmi se však žáci seznámí později, proto nyní preferujeme řešení s další postavou.

   - Pro restart pohybu nepřítele (tzn. pro restart pádu) přidejte do nepřítele další podmínku hlídající si dotyk s bodem dopadu. To lze řešit buď blokem *dotýkáš se _*, případně *dotýkáš se barvy _* a pomocí nástroje Kapátko vybral barvu **`Linie`**.

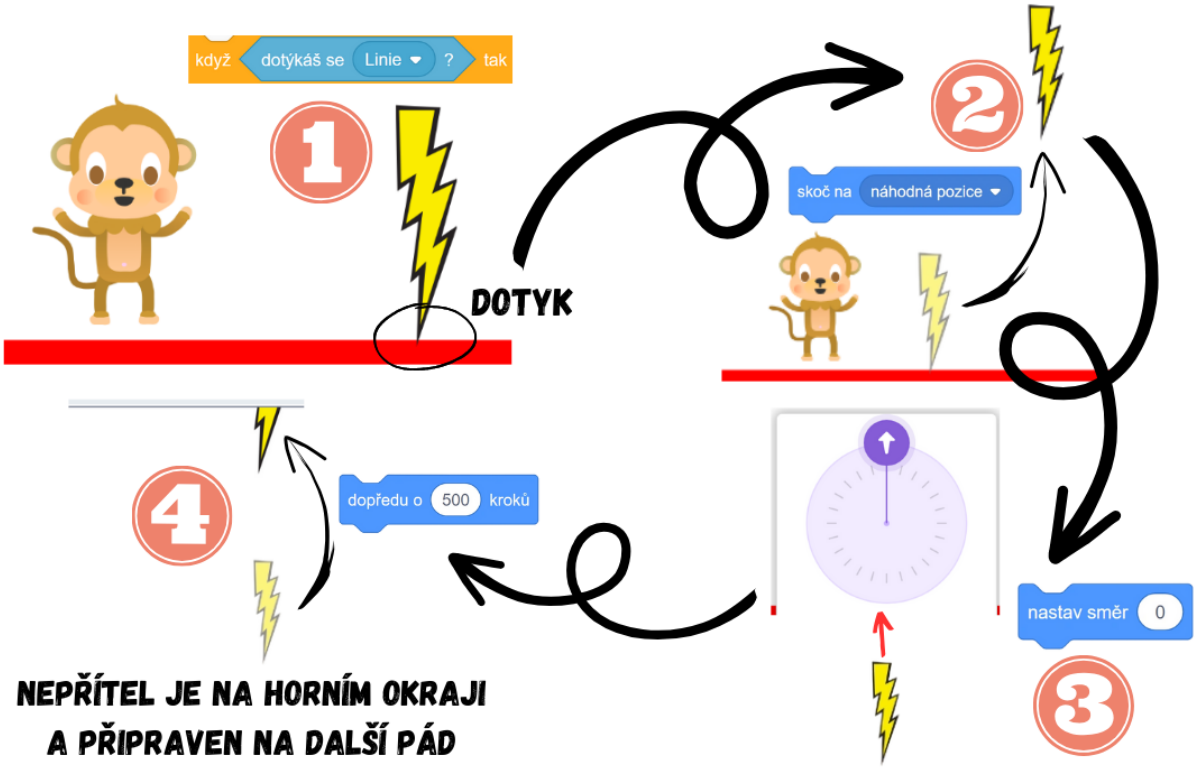
Poznámka: V případě použití bloku *dotýkáš se barvy _* pro hlídání dotyku nepřítele a **`Linie`** je nutné si dávat pozor na barvy dalších postav a pozadí při případném rozšiřování hry. Použitím bloku hlídající dotyk s konkrétní barvou se tak totiž vystavujeme možnému riziku, že se stejná barva vyskytne i jinde na scéně, což by mohlo vyústit v předčasné vyplnění podmínky atp.

   - Po splnění podmínky skočí nepřítel na náhodnou pozici pomocí bloku *skoč na náhodná pozice*. Tento blok nám sice zajistí, že nepřítel bude někde náhodně umístěn, ovšem to může být např. přímo na hráči a nepřítel má vždy padat shora. Proto po náhodném umístění ještě nepřítele zvedneme nahoru pomocí bloků *nastav směr _* a *dopředu o _ kroků* (kroků je potřeba hodně, aby nepřítel vyskočil až nahoru, kupříkladu 500 by vzhledem k rozměrům scény mělo stačit, nechte žáky zkoušením najít dostatečně velké číslo).

Poznámka: Rozměry scény jsou přibližně 540 kroků na šířku a 400 kroků na výšku. Pokud zadáme příliš vysokou hodnotu do bloku dopředu o _ kroků, postava se vždy přesune pouze na nejvyšší možnou, tzn. na okraj. Je třeba si uvědomit, že i když postavu posuneme o 500 kroků, její souřadnice y bude přibližně 200. Soustava souřadnic a práce s ní je však až součástí šestého projektu, proto zadání velmi vysokého čísla pro řešení tohoto problému nyní postačí.



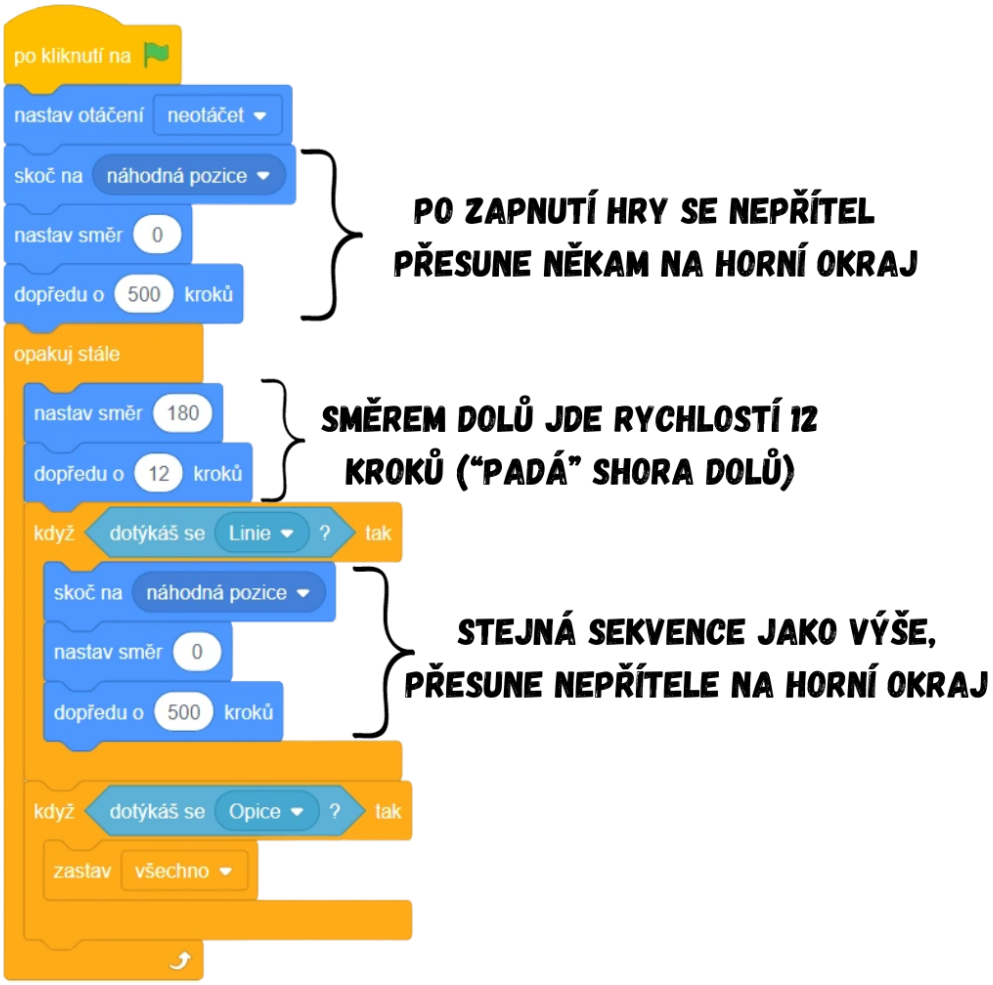
Obrázek 48 - Ukázka kódu



Obrázek 49 - Princip fungování sekvence ve hře

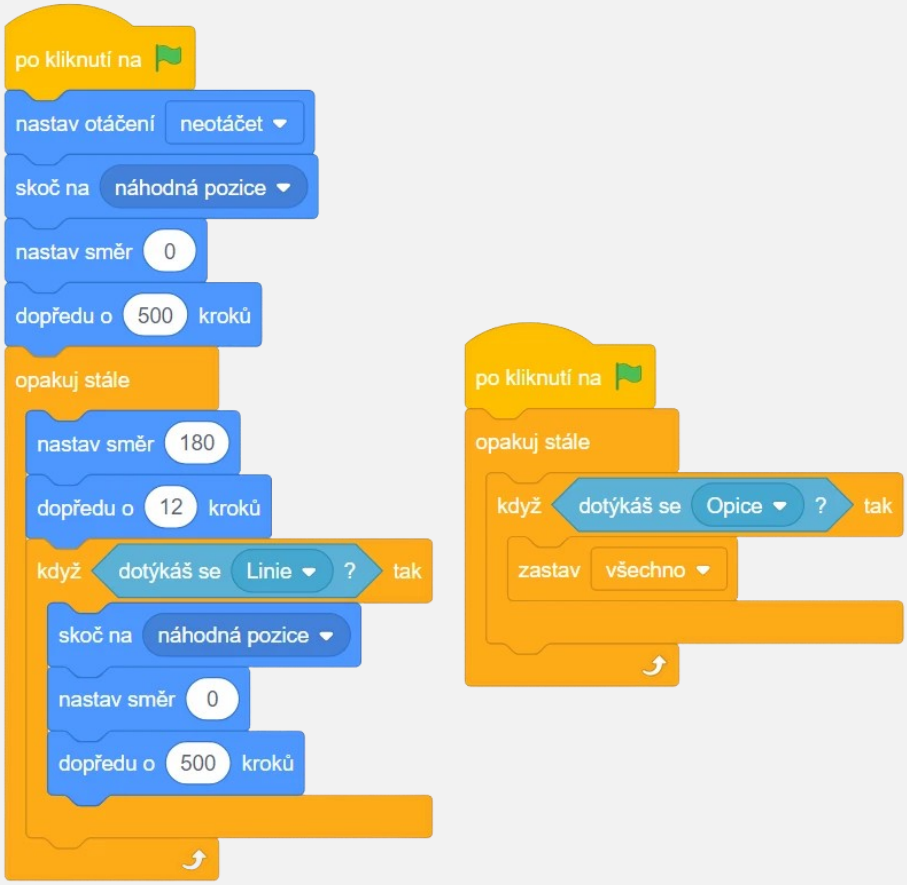
- Pokud postavu hráče zasáhl nepřítel a hra je ukončena, po jejím opětovném zapnutí dojde k chybě. Nepřítel se totiž stále dotýká postavy hráče a hra tak automaticky ihned opět skončí kvůli jejich dotyku. K zabránění vzniku této chyby je potřeba nepřítele po zapnutí hry poslat zpět nahoru. Proto musíme přidat sekvenci restartu pozice nepřítele

i po *po kliknutí na zelenou vlajku*, aby se tak stalo při každém zapnutí hry. Celý scénář nepřítele by mohl v jedné dlouhé sekvenci vypadat takto:



Obrázek 50 - Ukázka kódu

Poznámka: Je silně doporučeno pracovat v této fázi projektu již pouze s jednou dlouhou sekvencí. V případě, že by některý z žáků měl podmínku hlídající dotyk **`Nepřítele`** s postavou hráče v samostatné sekvenci (viz kód níže v poznámce), mohlo by dojít k chybě. Problémem je blok *skoč na náhodná pozice*, který by mohl náhodně posunout postavu **`Nepřítele`** přímo na postavu hráče. Při použití jedné dlouhé sekvence je vše v pořádku, protože po *skoč na náhodná pozice* se vykoná pohyb vzhůru a pak se teprve spustí (zkontroluje) podmínka hlídající dotyk s postavou hráče. Pokud ale žák má podmínku dotyku s postavou hráče v samostatné sekvenci, je pak podmínka spuštěna neustále, takže by se teoreticky po skoku na náhodnou pozici, který by vedl k dotyku **`Nepřítele`** s postavou hráče, mohla podmínka vyplnit a celá hra zastavit dříve, než by se dokončila sekvence resetu pozice Nepřítele následující po *skoč na náhodná pozice*. Přestože ve verzi Scratche 3.0 z nějakého důvodu k takové chybě ani při použití dvou samostatných sekvencí nedochází, je i tak doporučeno těmto možným chybám předcházet a snažit se jim vyvarovat, v tomto případě vytvořením jedné dlouhé sekvence.



Obrázek 51 - Ukázka kódu

Poznámka: V případě dostatku zbývajícího času můžete s žáky otevřít diskuzi nad efektivitou kódu. Jelikož se v tomto kódu vyskytují dvě identické sekvence, měli bychom se tak snažit jednu ze sekvencí smazat a spustit ji odkazem na tu druhou. V praxi by to poté vypadalo tak, že abychom nemuseli duplikovat sekvenci restartu pozice nepřítele, lze skok na horní okraj při každém zapnutí hry vyřešit jedním blokem. Je třeba si uvědomit, že restart pozice se provede vždy, když se nepřítel dotkne země (bod dopadu, **`Linie`**). Proto když namísto horní sekvence restartu pozice přidáme blok *skoč na _* a místo *náhodná pozice* vybereme název postavy představující bod dopadu (**`Linie`**), dosáhneme stejného výsledku. Po zapnutí hry (*po kliknutí na zelenou vlajku*) totiž nepřítel skočí na zem (**`Linie`**), v cyklu se vyplní podmínka a hned se tak provede přesun nepřítele na horní okraj. Je nutno dodat, že celé toto řešení je pro žáky velmi náročné k pochopení, a proto je celý tento postup označen za volitelný. Upravený kód by vypadal takto:

Obrázek 52 - Ukázka kódu

Vývoj: Projekt Opička původně obsahoval bloky pracující se souřadnicemi, podobně jako u předchozích projektů bylo od souřadnic upuštěno ve prospěch sice na první pohled složitějšího (více bloků), ale v jádru jednoduššího způsobu řešení, a to prostřednictvím známých bloků. Některé fráze byly po konzultaci s kolegy a vedoucím práce upraveny a doprovodný text k projektu celkově rozšířen. V poznámce byl přidán možný postup při vysvětlování problému plynulosti pohybu pomocí *po stisku klávesy* _. Detailněji popsáno bylo i řešení restartu nepřítele, zejména pomocí nových návodných obrázků. Bylo též přidáno formátování textu pro odlišení postav.

Průběh ověření: Skupiny se při této aktivitě velmi lišily, a to nejen v návrzích řešení, ale i celkovým postupem. První skupina dorazila na hodinu plná entuziasmu a po celou dobu výuky si udržovala vysokou úroveň pozornosti. Ve druhé skupině byl patrný pokles koncentrace, což několikrát vyústilo v nutnost výklad jistých kroků opakovat. Obě skupiny si po mém dotazu vzpomněly na možnost pohybu pomocí couvání prostřednictvím zadání záporné hodnoty, u některých žáků však došlo k nefunkčnosti kódu kvůli tomu, že předtím použili sekvenci pro pohyb i s blokem *nastav směr* _, čímž si postavu nastavili na směr -90. První skupina porozuměla principu plynulého pohybu pomocí podmínky *když* _

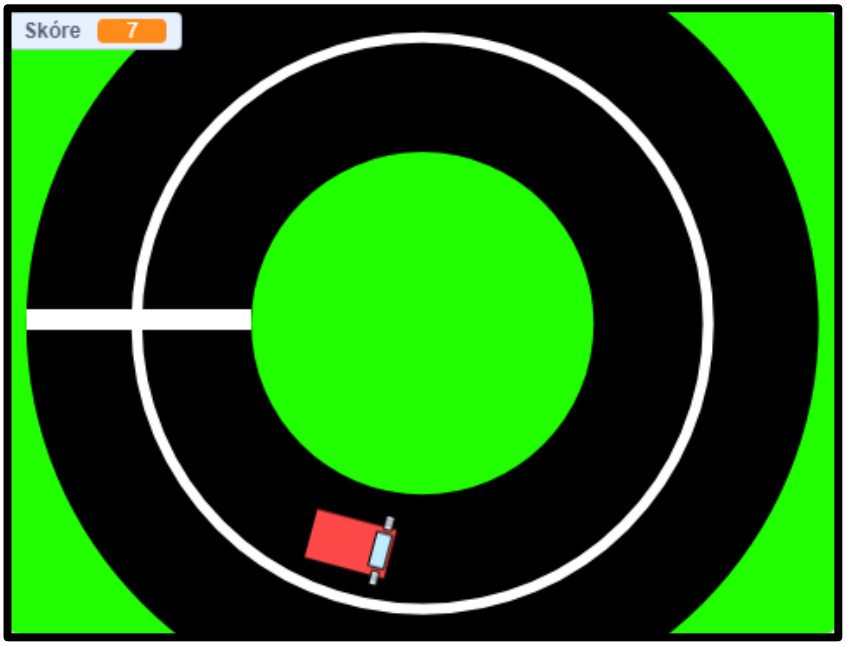
tak a cyklu *opakuj stále*, postřehla i rozdíl v plynulosti pohybu. Ve druhé skupině jsem zvolil jiný přístup, zkusil jsem cyklus plynulého pohybu nechat až jako poslední krok, aby byl jasnější rozdíl přímo v hotové hře. Jako vhodnější cesta se ovšem ukázala býti ta, podle které se postupovalo v první skupině, proto tvorba plynulého pohybu zůstala v metodice pod krokem 3. Ukázalo se, že je lepší řešit plynulost pohybu rovnou v okamžiku, kdy se tvoří pohyb postavy hráče, než se k této problematice zpětně vracet až na konci, kdy se již skupina zabývala jiným problémem. V obou skupinách žáci často zapomínali na cyklus *opakuj stále* při vytváření sekvence s podmínkou *když _ tak* (jak při řešení pohybu, tak při řešení kontaktu nepřítele s **`Linií`**), po upozornění si však dokázali chybu v kódu najít sami. V průběhu se vyskytly ještě drobnější chyby (skok na náhodnou pozici až po přesunu nepřítele nahoru), které jsem vždy řešil ukázkou na plátně spolu s vysvětlením, co je špatně, aby celou situaci viděli i ostatní žáci a dali si na chybu pozor. Většina žáků podle mého názoru příliš nepochopila myšlenkový proces za řešením návratu nepřítele do výchozí pozice prostřednictvím *skoč na Linie*, což byl případ zejména druhé skupiny. Projekt se podařilo dokončit s oběma skupinami ve stanoveném čase 90 minut.

Úpravy po ověření ve výuce: Jelikož někteří žáci měli špatně nastavený směr postavy (-90) při programování pohybu, což vedlo k nesprávnému fungování scénáře, bylo přistoupeno k rozšíření textu a přidání poznámek v kroku 3. Dále je v metodice více kladen důraz na paralelní spouštění sekvencí, byly přidány vysvětlující poznámky a obrázky dalších možných řešení uspořádání kódu. Přidána byla též poznámka zdůrazňující a vysvětlující nutnost vložit sekvence s podmínkami do *opakuj stále*. Pár žáků navrhlo do podmínky dotyku s **`Linií`** dát blok *dotýkáš se barvy _*, byla proto do metodiky přidána poznámka varující před možnými nedostatky tohoto postupu. Z důvodu příliš složitého myšlenkového procesu za blokem *skoč na Linie*, který měl původně vést ke zjednodušení sekvence resetu pozice **`Nepřítele`**, bylo v metodice přistoupeno k upřednostnění původního řešení s více bloky.

9.5 Lekce 5 – Auto

Délka: 80-90 minut (bez kreslení 45-60 minut)

O projektu: Úkolem hráče je zůstat co nejdéle na závodní dráze. Automobil je však rychlý a je tak těžké se na vyhrazené dráze udržet dlouho.



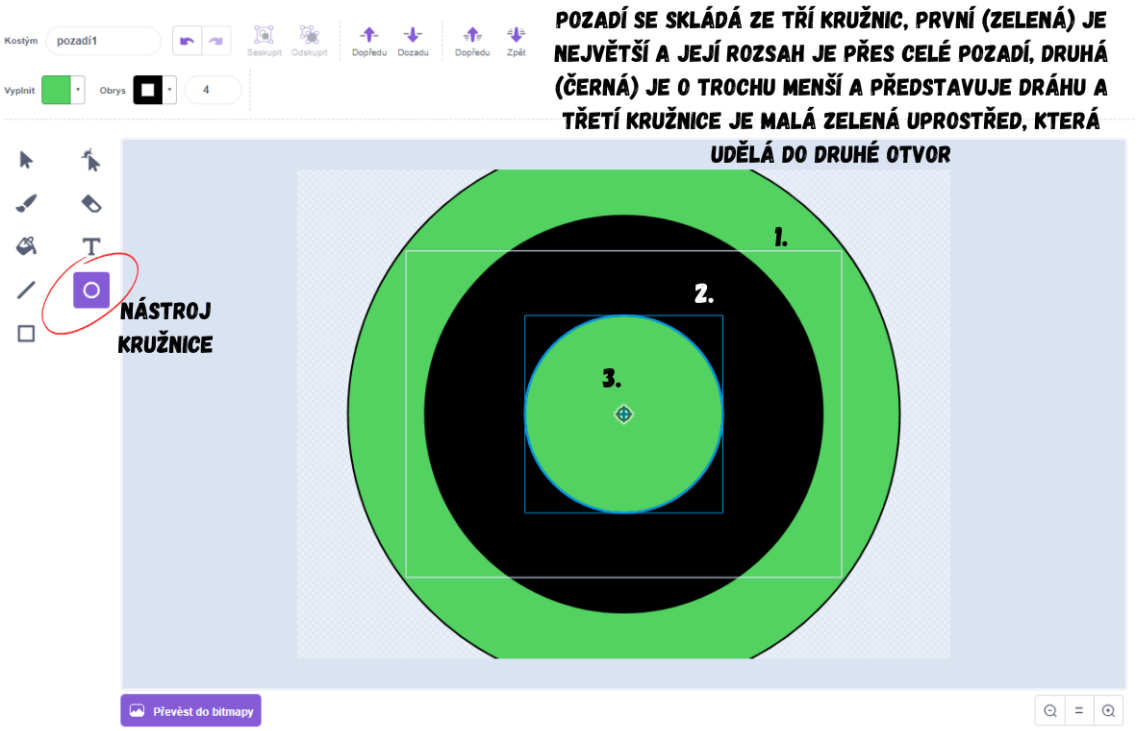
Obrázek 53 - Ukázka ze hry

Cíl lekce: Žáci si zopakují práci v editoru při tvorbě jednoduchého pozadí a vytvoří si vlastní postavu. Naučí se, co jsou to proměnné a k čemu jsou dobré, dokážou je nastavovat, měnit a používat.

Nově naučené bloky: *čekej _ sekund*; *vytvoř proměnnou*; *nastav _ na _*; *změň _ o _*

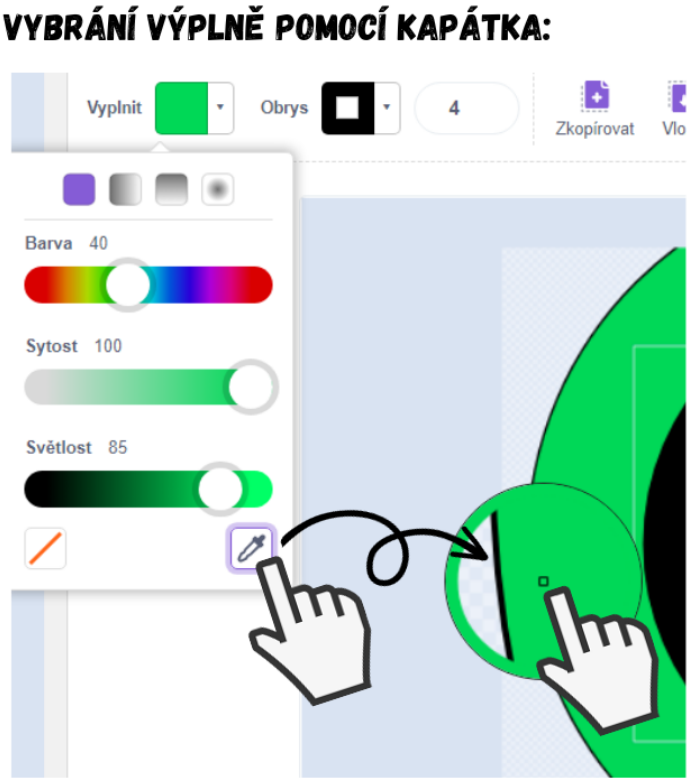
Náplň lekce:

1. Úvodní diskuze:
    - Představte žákům zamýšlený projekt a ukažte jim, jak by měl výsledek vypadat, viz BIT_5_Auto.
    - Diskutujte s dětmi o tom, s čím už si jsou schopny poradit díky svým znalostem z předchozích lekcí a s čím si rady neví, co je tedy naopak nové.
2. Vytvoření pozadí:
    - Nechte žáky nakreslit jednoduchou dráhu pro auto.
    - Nejjednodušší je vytvořit pomocí nástroje Kružnice závodní okruh. Začněte s největší kružnicí přes celé pozadí, která bude např. zelená jako trávník. Pro utvoření kružnic prostřednictvím nástroje Kružnice stačí držet klávesu Shift, jinak může dojít zploštěním k vytvoření elipsy. Poté udělejte menší kružnici, např. černou, jako silnici. Poslední kružnice uprostřed bude stejné barvy jako první k vymezení vnitřního okraje závodního okruhu. K vybrání stejné výplně použijte nástroj Kapátko, který žáci znají např. z bloku *dotýkáš se barvy _*.



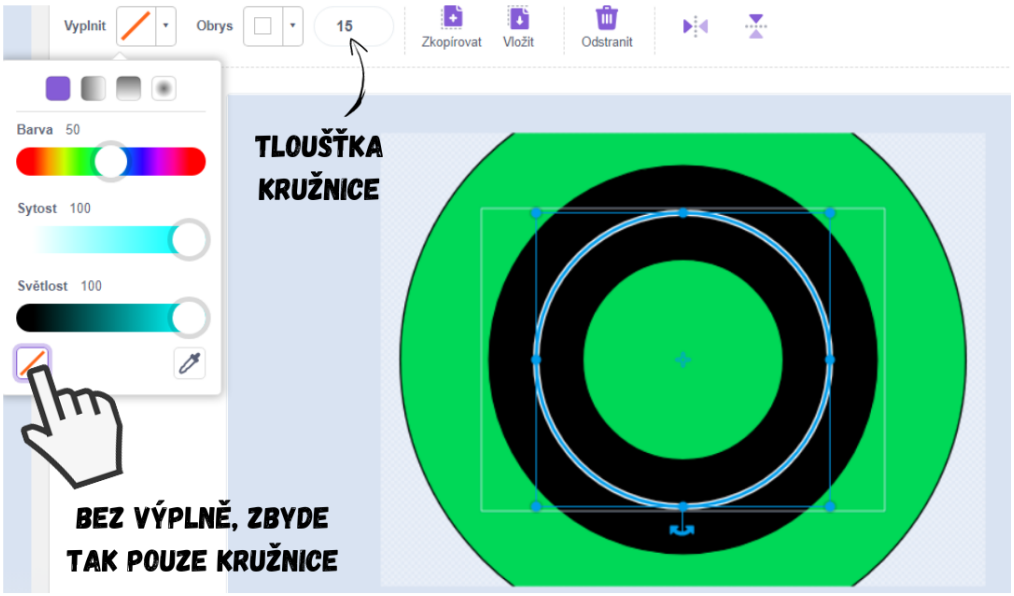
Obrázek 54 - Návod

Poznámka: Může se stát, že někteří žáci nepoužijí nástroj kapátko a budou odstín zelené barvy odhadovat a zadávat ručně. To pak může vyústit v nefunkčnost hry, protože porážka se má spustit po dotyku s barvou zelených kružnic (kružnice 1 a 3 na obrázku výše), pokud ale mají kružnice různý odstín zelené, dojde k porážce pouze po dotyku s jednou z nich.



Obrázek 55 - Návod

Poznámka: V případě dostatku času můžete přidat i bílou kružnici představující silniční čáru. Nakreslete pouze kružnici bez výplně, to uděláte tak, že vybere ve výběru výplně možnost vlevo dole . Tento nástroj způsobí, že kruh bude bez výplně a zbyde tak pouze kružnice.

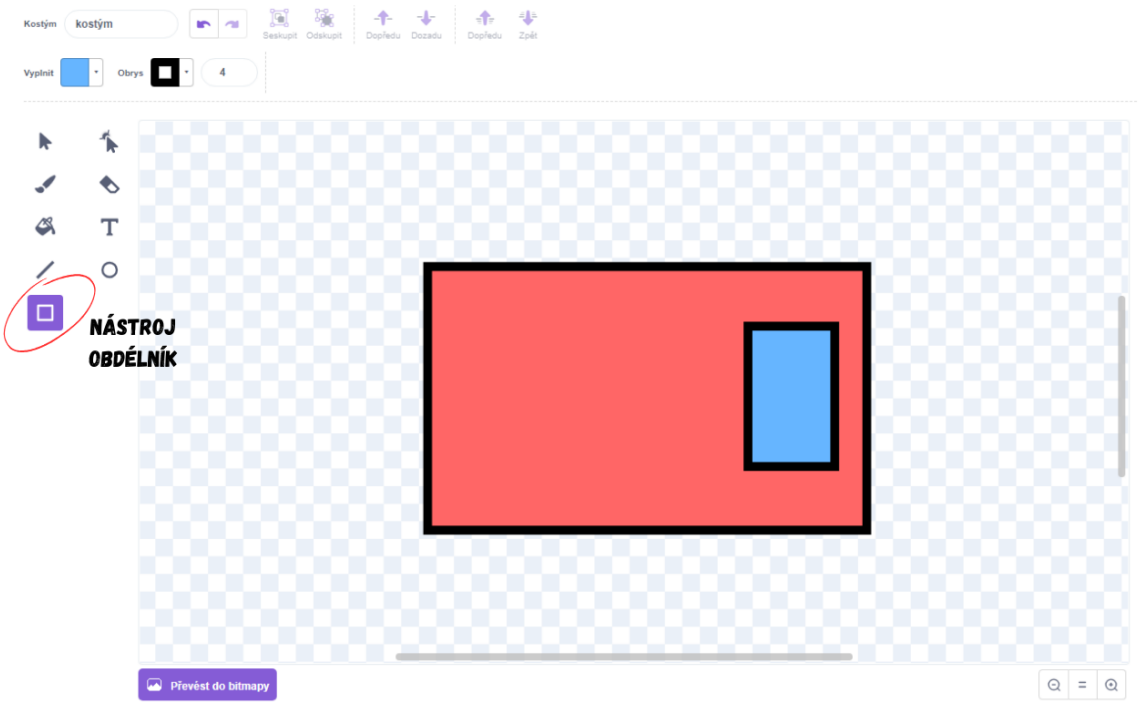


Obrázek 56 - Návod

Poznámka: Zdůrazněte, že startovní čára není součástí pozadí, ale později ji vytvoříme zvlášť jako postavu **`Start`**.

3. Vytvoření hráče:

   - Hráč v tomto projektu hraje za auto. Ukažte žákům, jaký postup zvolit při kreslení automobilu. Nejjednodušší je vytvořit automobil pomocí nástroje Obdélník. Udělejte jeden větší obdélník a druhý menší obdélník použijte jako přední sklo. Autu můžete též dodělat např. zrcátka či pneumatiky pomocí elipsy (nástroj Kružnice bez držení klávesy Shift).
   - Nechte žáky nakreslit jednoduchý závodní automobil.



Obrázek 57 - Návod

   - Důležité je hlídat si při kreslení polohu tzv. středového bodu. Jedná se o „těžiště“ postavy, které funguje jako střed otáčení, kolem kterého se postava točí. Pokud není potřeba z důvodu specifičnosti projektu jinak, střed postavy a středový bod by se měli překrývat.

Obrázek 58 - Návod

Obrázek 59 - Návod

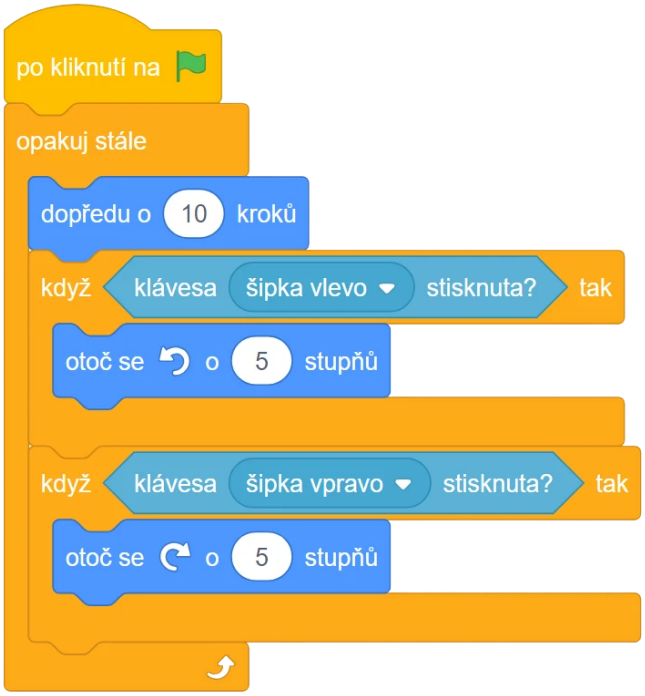
Poznámka: Ukažte žákům fungování středového bodu na otáčení postavy. Posuňte kostým naschvál mimo středový bod a nechte žáky, ať sledují, jaká změna proběhla v otáčení postavy.

Obrázek 60 - Návod

Poznámka: V případě nedostatku času použijte předpřipravený projekt s pozadím i postavami BIT_5_Auto_Start.

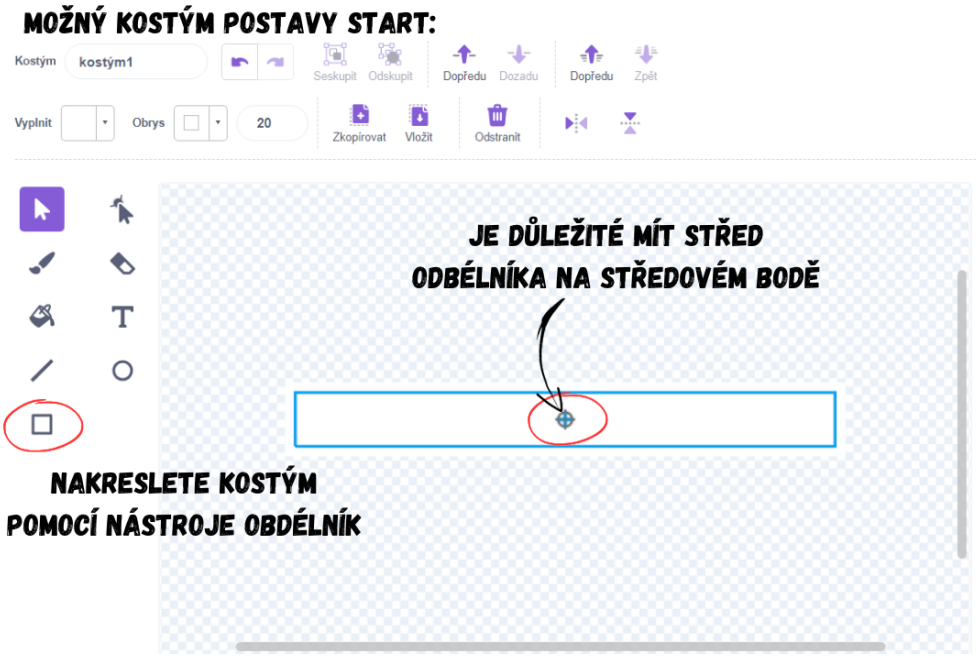
4. Naprogramování hráče:

   - Pro neustálý pohyb vpřed použijeme již známou kombinaci bloků *po kliknutí na zelenou vlajku*, *opakuj stále* a *dopředu o _ kroků*. Plynulé otáčení auta lze zařídit podmínkou *když _ tak* s *klávesa _ stisknuta?*, které vložíme do cyklu. Uvnitř bloku *když _ tak* je prosté *otoč se o _ stupňů*. Vyzkoušejte, kolik stupňů je ideální pro hru.



Obrázek 61 - Ukázka kódu

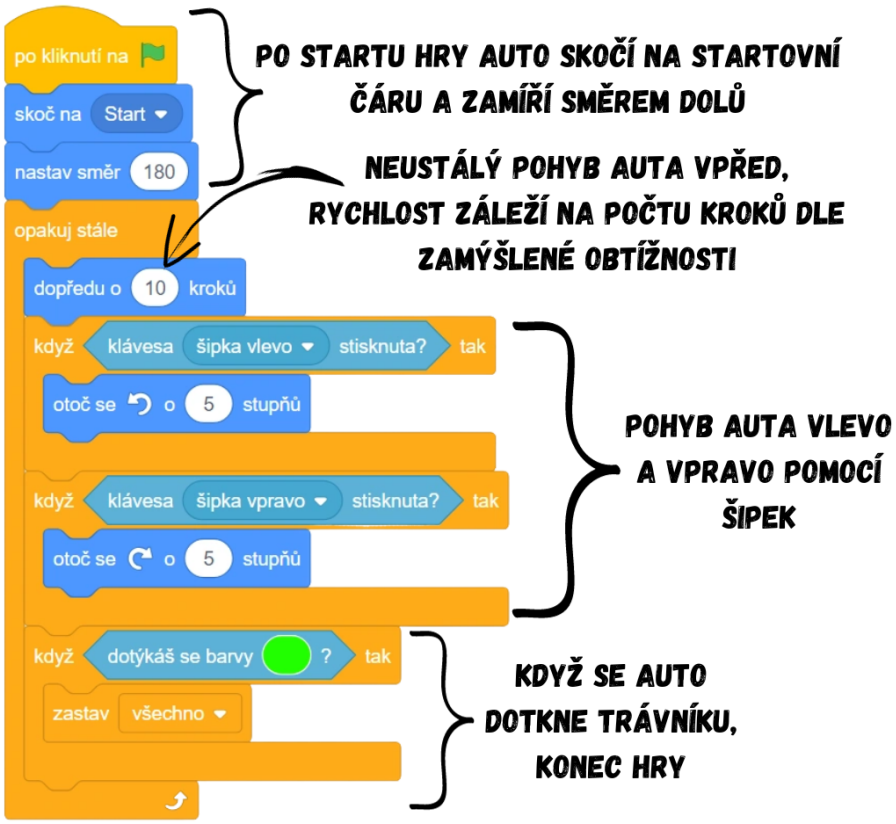
- Auto jede dopředu, dokud nevyjede mimo dráhu. V případě jasně barevně odlišených pozadí (jedna barva pro prostor mimo dráhu) lze pro ukončení hry využít podmínky *když _ tak* s *dotýkáš se barvy*, po jejímž vyplnění dojde k zastavení hry prostřednictvím *zastav všechno*.
- Správný restart hry je vhodné vyřešit nastavením počáteční pozice a počátečního směru **`Auta`**. Řešením by mohlo být vytvořit novou postavu, která bude představovat počáteční pozici, nazvěte ji třeba **`Start`**. Její kostým bude bílá čára. Následně naprogramujte ve scénáři **`Auta`** restart pomocí bloků *skoč na _* a *nastav směr _*.



Obrázek 62 - Návod

Poznámka: Na středový bod je vázán například i blok *skoč na _*, který přesune postavu právě na středový bod vybrané postavy (nikoliv na střed dané postavy!). Někteří žáci mohou nakreslit obdélník mimo středový bod, což ve hře vyústí v to, že **`Auto`** skočí mimo kostým postavy **`Start`**, konkrétně na její středový bod.

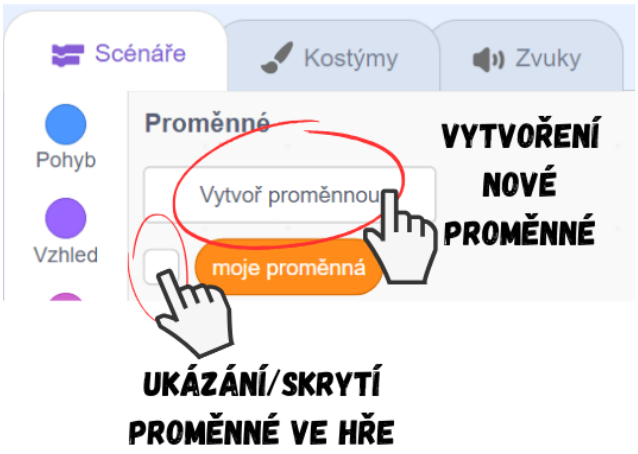
- Kód hráče by mohl vypadat například takto:



Obrázek 63 - Ukázka kódu

5. Vytvoření skóre:

   - Vysvětlete žákům, co jsou to proměnné a proč se tak jmenují (šuplík s měnící se hodnotou uvnitř). Zeptejte se žáků, k čemu by se daly proměnné využít, co chceme počítat a mění se to (čas, skóre, životy atd.).
   - Použijte tlačítko *Vytvoř proměnnou* v kategorii *Proměnné*. Proměnnou nazvěte např. `Skóre` a dejte OK. Proměnné lze ve hře ukazovat nebo skrýt pomocí checkboxu vedle jejich jména, popřípadě pomocí bloků *ukaž proměnnou _* a *skryj proměnnou _*.



Obrázek 64 - Návod

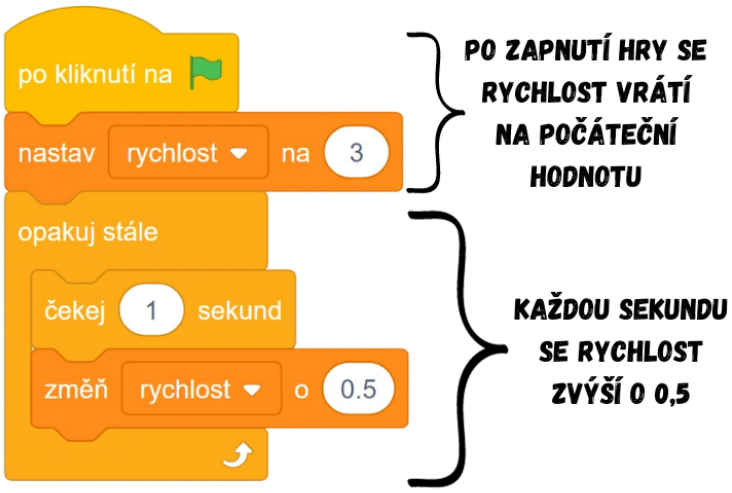
- Aby `Skóre` fungovalo tak, jak chceme, je třeba ho naprogramovat. `Skóre` se má zvyšovat každou sekundu, co auto jede po silnici. Tedy po zapnutí hry (*po kliknutí na zelenou vlajku*) se spustí nekonečný cyklus (*opakuj stále*), který za jednu sekundu zvýší skóre o 1. K tomu využijte bloky *čekej 1 sekund* a *změn _ o 1*. Proměnné programujte vždy tam, kde to dává logicky smysl, sekvenci počítající skóre tak skládejte např. ve scénáři scény (pozadí).
- Po restartu hry chceme, aby se `Skóre` vyresetovalo a bylo opět 0. Přidejte tedy blok *nastav Skóre na 0* k *po kliknutí na zelenou vlajku*.

Obrázek 65 - Ukázka kódu

Poznámka: Celý kód nastavující proměnnou `Skóre` musí být v samostatné sekvenci. Pokud by žáci vložili kód do jedné sekvence spolu s pohybem, blok *čekej 1 sekund* by celý cyklus přerušil na daný časový úsek a pohyb by tak během tohoto „zamrznutí“ nefungoval, proto je nutné `Skóre` nastavit zvlášť v samostatné sekvenci.

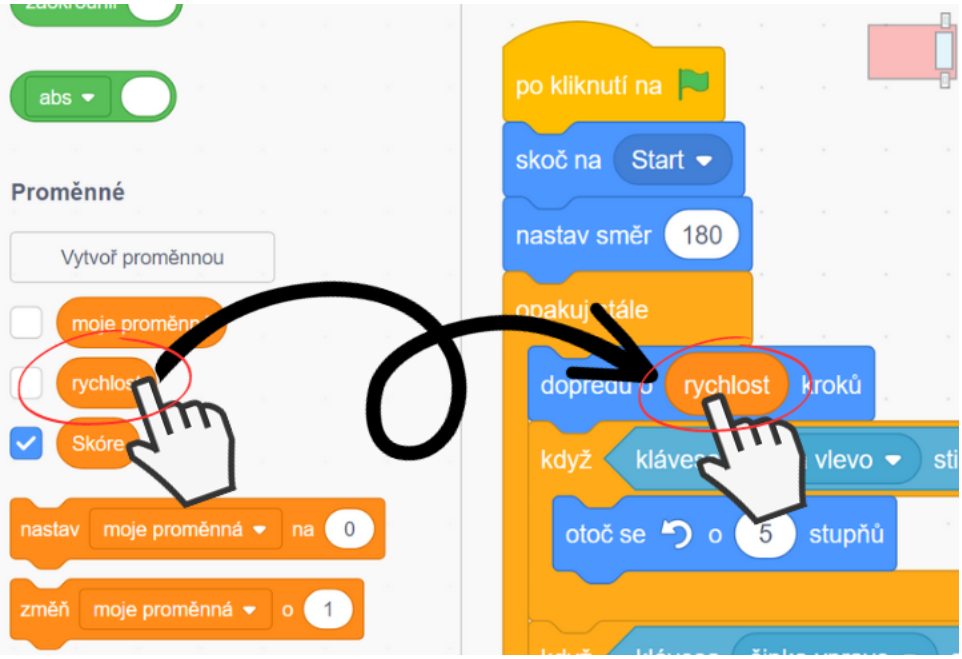
6. Volitelné: Postupně rostoucí obtížnost

- Zbývající čas lze využít pro rozšíření hry, kupříkladu postupně hru ztěžovat. To lze udělat jednoduše postupným zvyšováním rychlosti automobilu. Vytvořte novou proměnnou a nazvěte ji např. `rychlost`. Jelikož se `rychlost` týká **`Auta`**, naprogramujte tuto proměnnou ve scénáři této postavy. Obdobně jako u `Skóre`, po startu hry nastavíme `rychlost` na počáteční hodnotu (např. 3) a postupně ji budeme každou sekundu zvyšovat (za 1 sekundu např. o 0,5).



Obrázek 66 - Ukázka kódu

- Aby proměnná fungovala tak, jak chceme, je nutné do bloku *dopředu o _ kroků* v nekonečném cyklu pohybu **`Auta`** vložit proměnnou rychlost z kategorie *Proměnné*.



Obrázek 67 - Návod

Vývoj: Podobně jako tomu bylo u předchozích projektů, projekt Auto původně obsahoval blok využívající souřadnice, který byl po konzultaci s kolegy nahrazen jednodušší variantou řešení. Rozšířen

byl zejména popis postupu při práci v editoru (vlastní pozadí, vlastní postava). Kromě detailnějšího postupu bylo přidáno i vysvětlení středového bodu. V metodice došlo k vizuálnímu odlišení proměnných a postav v textu pomocí formátování.

Průběh ověření: Obě skupiny dorazily na výuku vcelku soustředěné, tudíž postup v projektu by se dal označit za plynulý. Projekt Auto se obecně setkal s vřelým přijetím od žáků, zdálo se, že je hra nadchla a bavila. Zadání obě skupiny pochopily bez problémů, většina žáků dokonce projevila velmi dobré analytické schopnosti při rozboru úlohy, kdy už byli schopni vidět za jednotlivými funkcionalitami hry bloky a sekvence, které použijeme pro řešení projektu. Skupiny tak byly schopny dát na základě dosavadních získaných znalostí dohromady řešení částí hry, a to zejména těch částí, jež znaly z předchozích projektů. Hodně času (30-45 minut) zabralo kreslení pozadí a postav, přestože žáci instrukce chápali a dokázali si s úkolem dobře poradit. Nejsem si jistý, zda všichni žáci naplno pochopily význam středového bodu a pochopily rozdíly v otáčení postavy při umístění mimo středový bod. Při programování pohybu si obě skupiny správně vzpomněly na blok *otoč se o _ stupňů*, ale první skupina už nedokázala dát sama kompletně dohromady plynulé otáčení na základě znalostí o plynulém pohybu z předchozí lekce. Oproti předchozí lekci žáci nyní už lépe chápali spouštění bloků *když _ tak* v *opakuj stále* a už se nebáli programovat vše v jedné dlouhé sekvenci. Žáci dobře reagovali na mé doplňující dotazy (co mám změnit, aby se auto zrychlilo/aby se zpomalilo otáčení atd.) a odpovídali správně. Vypadalo to, že novou problematiku – proměnné – pochopily obě skupiny, dokázali popsat rozdíl mezi bloky *nastav proměnnou na _* a *změn proměnnou o _*. Sami žáci poté vyžadovali i postupné zrychlování **`Auta`** (volitelný krok 6), obě skupiny dokázaly samy proměnnou naprogramovat. Celkově projekt Auto hodnotím jako úspěšný, zdálo se, že vytváření hry žáky bavilo a naučili se nově práci s proměnnými.

V průběhu této lekce velmi názorně vyplynulo na povrch, které algoritmické koncepty si zvládli žáci již plně osvojit (podmíněné příkazy, cykly), a které si ještě stále osvojovaly (plynulý pohyb pomocí cyklu). Obecně však lze u žáků spatřovat vzestupný trend co se týče komplexnosti jejich znalostí, s každým dalším projektem jsou v řešení problémů zběhlejší, z čehož je možné usuzovat, že se jejich algoritmické myšlení rozvíjí.

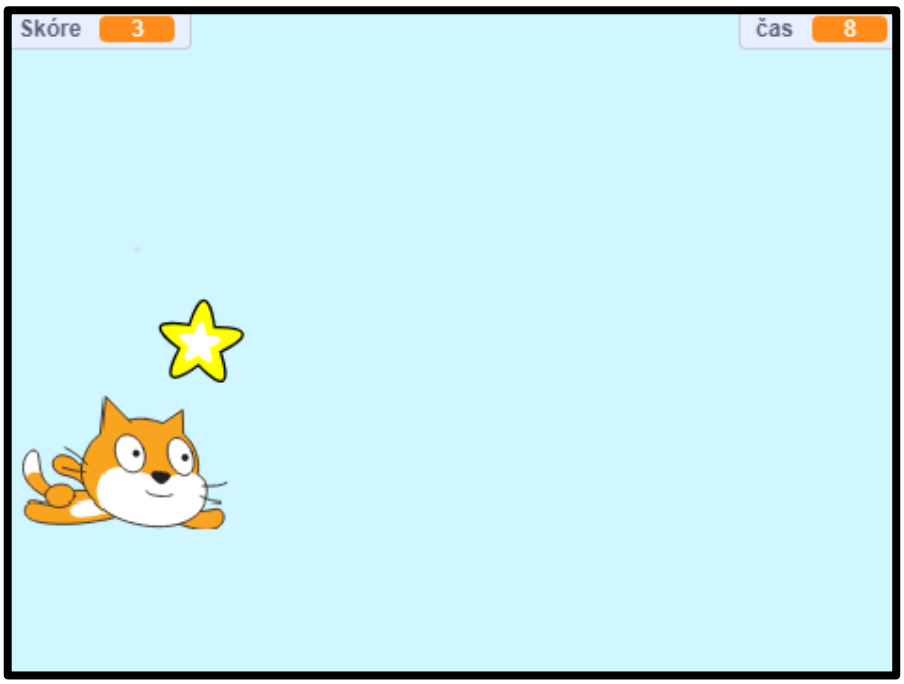
Úpravy po ověření ve výuce: Protože někteří žáci nepoužili při tvorbě kružnic ukázaný nástroj kapátko, měli pak kružnice s různými barvami, což vedlo k nesprávnému fungování porážky pomocí *dotýkáš se barvy _*. Byla proto do metodiky přidána poznámka upozorňující na tuto možnou chybu a varující před jejími důsledky. Přidána byla též poznámka a názorný obrázek pro tvorbu bílého silničního pruhu, na který je vhodné použít kružnici bez výplně. Rozšířen byl popis středového bodu a u tvorby postavy **`Start`** připsána poznámka o možném nesprávném fungování bloku *skoč na _* v případě, že je střed

kostýmu mimo středový bod. Metodika nyní obsahuje instrukce o tom, v jakém scénáři se mají programovat jednotlivé proměnné.

9.6 Lekce 6 – Létání

Délka: 80-90 minut

O projektu: Hráč hraje za létající postavu a jeho cílem je sbírat hvězdičky. Na tento úkol má však omezený čas, pokud nestihne nasbírat dostatečný počet, prohrál.



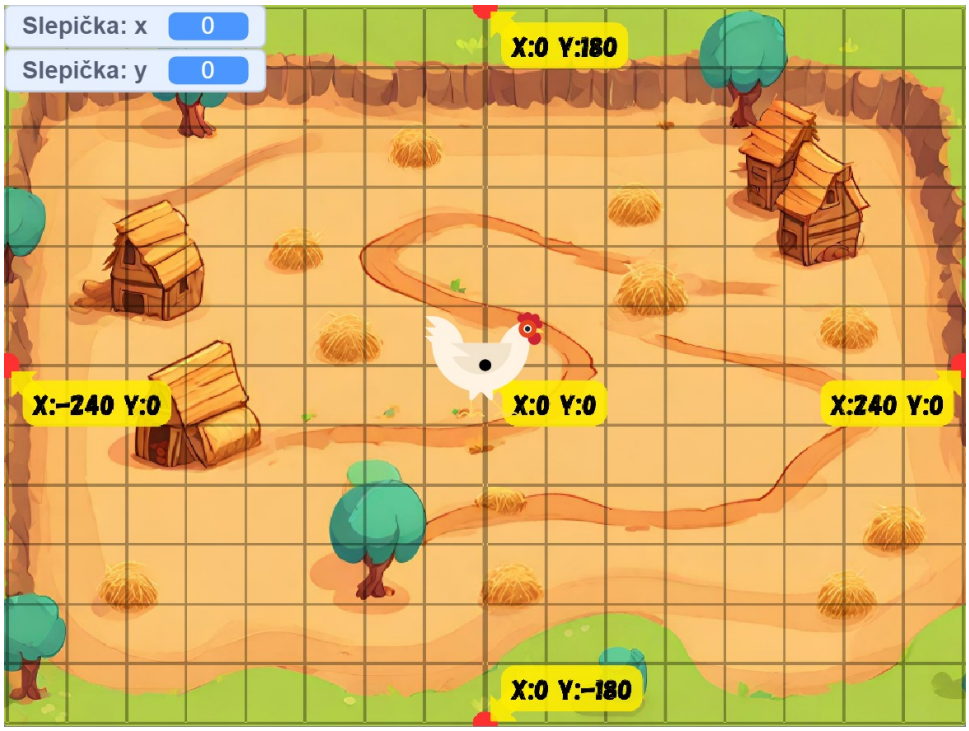
Obrázek 68 - Ukázka ze hry

Cíl lekce: Žáci prohloubí své znalosti o cyklech a proměnných, dokáží pracovat s hodnotami proměnných a využívat je. Získají základní přehled o tom, jak fungují souřadnice a kdy změnit x a kdy y. Součástí lekce je i otevření otázky vhodného umístění jednotlivých sekvencí do té či oné postavy.

Nově naučené bloky: *skoč na x: _ y: _*; *klouzej _ sekund na x: _ y: _*; *ukaž se*; *skryj se*; *opakuj dokud nenastane _*; *náhodné číslo od _ do _*; *_ = _*

Náplň lekce:

1. Propedeutická aktivita na soustavu souřadnic
    - V tomto projektu budou žáci pracovat se souřadnicemi, proto je nutné je nejdříve s jejich principem obeznámit.



Obrázek 69 - Ukázka z propedeutické aktivity

- Otevřete připravenou hru BIT_Aktivita_souřadnice, která obsahuje soustavu souřadnic s popisy základních bodů. Je vhodné, aby vyučující projekt promítal a aby si hru otevřel i každý žák sám. Po spuštění hry žákům vysvětlete, jak funguje soustava souřadnic. Vlevo nahoře se zobrazují proměnné obsahující aktuální souřadnice **`Slepičky`**. Postavou lze pohybovat pomocí šipek, jeden krok daným směrem posune slepici o 10, čemuž odpovídá i změna souřadnic. Souřadnice x se mění v případě pohybu vlevo/vpravo, y zase při pohybu nahoru/dolu. Nechte žáky, ať si vyzkouší, jak souřadnice fungují.

- Samotný příběh hry se zapne až po stisknutí klávesy mezerník, pokud jsou tedy žáci dostatečně obeznámeni se soustavou souřadnic, spusťte stisknutím klávesy příběh. **`Slepičce`** se ztratilo její kuřátko a cílem hráče je pomoci jí ho najít. **`Slepička`** dává hráči příkazy obsahující souřadnice, kam by měl hráč jít. Doporučením je řešit hru společně, tzn. vyučující promítá a žáci mu radí, jaké klávesy má stisknout. V takovém případě je výuková činnost organizovaná, vyučující může volit tempo vhodné pro celou skupinu a jednotlivé kroky doplňovat o krátký komentář nebo nechat o dalších krocích žáky hlasovat (x je 150, kde se to přibližně nachází? vpravo? vlevo? nahoře? dole?). V případě, že by žákům aktivita příliš nešla, je možné je po společném dokončení nechat projít hru ještě jednou samostatně.

2. Úvodní diskuze:

   - Představte žákům zamýšlený projekt a ukažte jim, jak by měl výsledek vypadat, viz BIT_6_Létání.

   - Diskutujte s žáky o tom, s čím už si jsou schopny poradit díky svým znalostem z předchozích lekcí a s čím si rady neví, co je tedy naopak nové.

3. Vybrání pozadí:

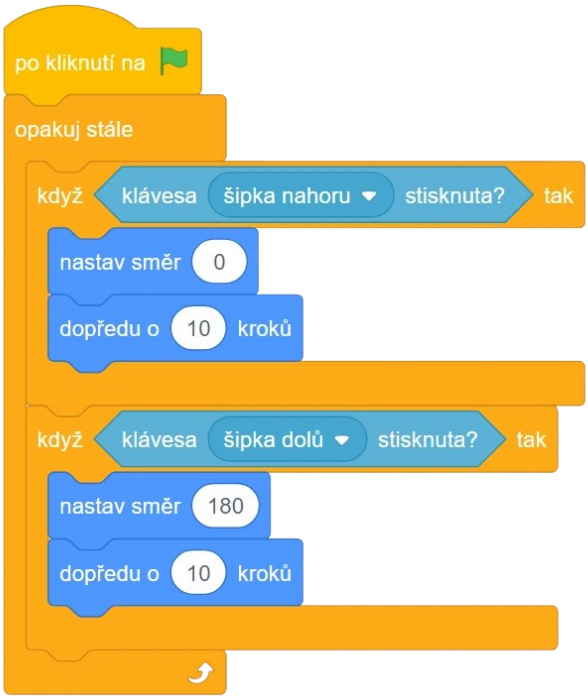
- Použijte předpřipravené pozadí z knihovny Scratch tematicky vhodné pro hru (např. *Blue Sky 2*).

4. Vytvoření postavy hráče:

    - Použijte předpřipravenou postavu v knihovně Scratch (ideální je postava dívající se vpravo, jenž může létat, aby hra dávala smysl, tedy např. *Butterfly 2*, *Griffin*, *Cat Flying*).

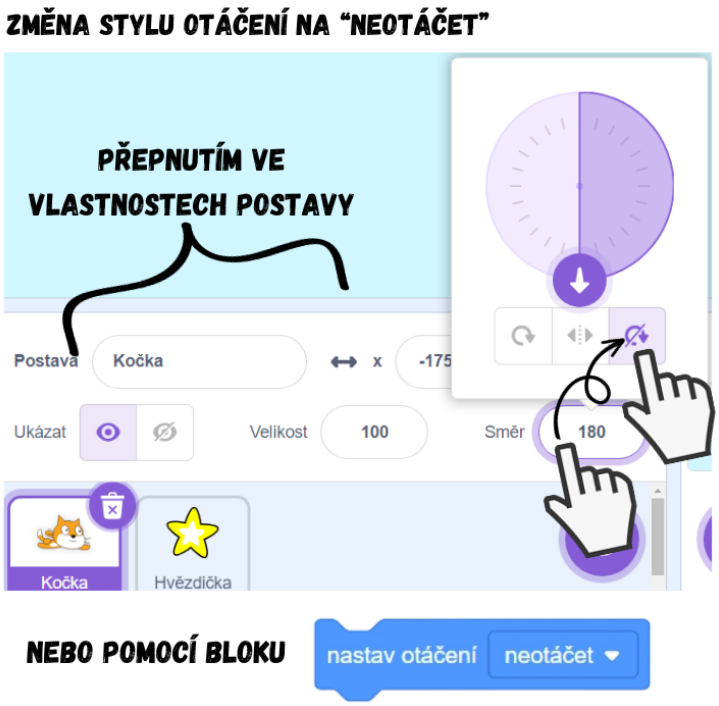
Obrázek 70 - Ukázka vhodných postav

- Hráč bude moci měnit výšku letu postavy, tedy pouze pohyb nahoru a dolů, přesuňte proto postavu k levému kraji a naprogramujte plynulý pohyb pro dva směry prostřednictvím cyklu *opakuj stále*, podmínky *když _ tak* a *klávesa _ stisknuta?*. Zprvu nechte žáky pracovat samostatně na základě jejich zkušeností z předchozích lekcí.
- Pro pohyb nahoru a dolů využijte bloky *nastav směr _* a *dopředu o _ kroků*.



Obrázek 71 - Ukázka kódu

- Problémem je, že postava se poté otáčí vzhůru nohama. K zabránění otáčení stačí ve vlastnosti Směr postavy nastavit styl otáčení na neotáčet, popř. využít bloku *nastav otáčení _*.



Obrázek 72 - Návod

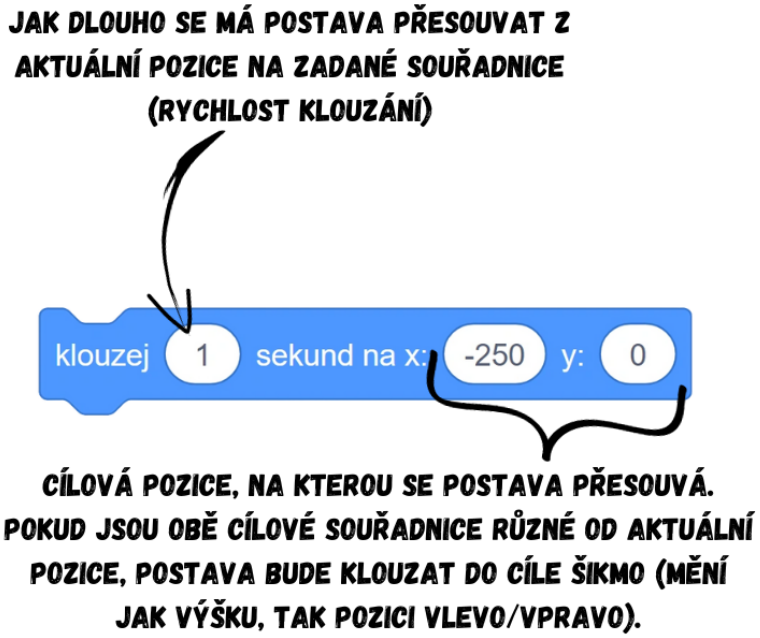
Obrázek 73 - Ukázka otáčení postavy dle stylu otáčení

5. Vytvoření pokladu:

    - Pro vytvoření pokladu, který má hráč chytat, použijte předpřipravenou postavu v knihovně Scratch (např. *Star*).

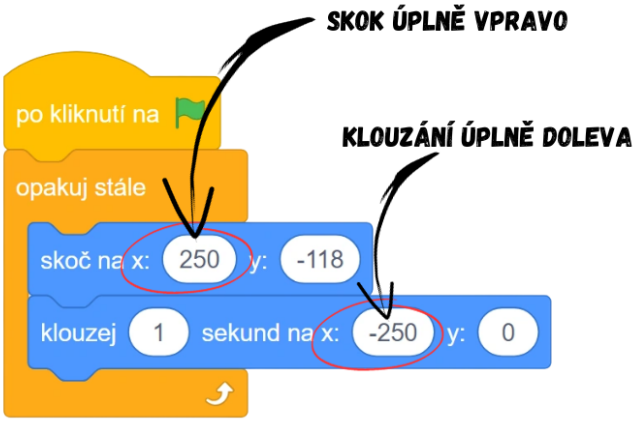
Poznámka: Ve hře se bude zdát, že postava hráče letí vpřed a míjí poklady. Jedná se o tzv. iluzi letu, kterou právě tvoříme tím, že hráč ve skutečnosti pouze mění výšku a poklady jsou tím, co se hýbe (létají zprava doleva). Diskutujte s žáky na toto téma, zda se například s iluzí letu již někde setkali.

- Poklad létá zprava doleva a pokaždé má vyletět odjinud. Pokud se postava hráče dotkne pokladu, získá bod.
- Pro pohyb pokladu zprava doleva využijeme nový blok *klouzej _ sekund na x: _ y: _*. Vysvětlete žákům, jak funguje a nechte je vyzkoušet si ho v praxi (spouštějte klikáním). Využijte při tom znalosti z úvodní aktivity. Důležité je pochopit, že hodnoty x a y v bloku odpovídají souřadnicím, na které bude postava klouzat ze své výchozí pozice. Souřadnice x rozhoduje o tom, jestli má být cílová destinace, kam má postava „doklouzat“, vlevo či vpravo. Písmenko y zase rozhoduje o výšce destinace, tedy zda má být nahoře či dole. Je třeba, aby si žák uvědomil, že velké kladné x nastaví cílovou destinaci k pravému okraji a velké záporné x zase k levému okraji. S těmito znalostmi by měl žák dojít k tomu, že v daném bloku je třeba nastavit x na velmi nízké číslo (tedy vysoké záporné), například -250, tedy takové, aby se cílová destinace nacházela při levém okraji a poklad tak dojel až úplně vlevo.



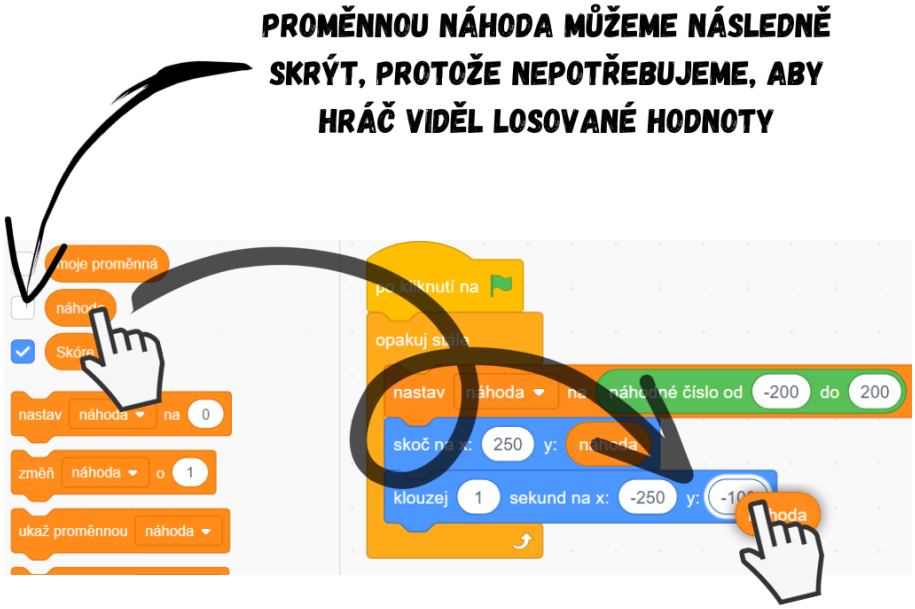
Obrázek 74 - Ukázka kódu

- Poklad nyní klouže zprava doleva, ovšem po dokončení pohybu zůstává na souřadnicích cílové destinace. Abychom vyvolali iluzi toho, že postava hráče během letu nalézá různé poklady, ke kterým se přibližuje, musíme poklad po dosažení cíle (levého okraje) vrátit zase zpět na pravý okraj. Pro restart pohybu pokladu je tedy třeba před samotným klouzáním přesunout poklad úplně vpravo, tedy na počáteční pozici, ze které bude klouzat doleva. K tomu využijeme blok *skoč na x: _ y: _*, který po svém spuštění instantně přesune postavu na vložené souřadnice. Defaultně jsou v bloku vepsány hodnoty souřadnic aktuální pozice postavy. Důležitá je pro nás nyní souřadnice x, kterou musíme upravit, aby poklad skočil co nejvíce vpravo. Nechte žáky, ať vymyslí, zda se bude jednat o velmi nízké číslo, či velmi vysoké. Vhodná hodnota je např. 250, ale záleží na velikosti zvolené postavy pro poklad. Připojte k bloku i dříve zmíněný *klouzej _ sekund na x: _ y: _* a celou sekvenci vložte do cyklu *opakuj stále* spuštěného po zapnutí hry (*po kliknutí na zelenou vlajku*).



Obrázek 75 - Ukázka kódu

- Poklad nyní létá zprava doleva, ovšem v případě, že máme různé hodnoty y, tak létá nakřivo. Problém je v nastavené výšce. Souřadnice y se nám změní skokem vpravo, ovšem v bloku *klouzej _ sekund na x: _ y: _* se nachází jiná hodnota výšky y nezávislá na skoku. Pokud hodnotu přepíšeme na stejné číslo, zjistíme však, že poklad létá pořád stejnou trasou. Proto budeme muset přidat do bloku prvek náhody.
- K tomu je ideální blok z kategorie *Operátory* s názvem *náhodné číslo od _ do _*. Tento blok si jednoduše při spuštění vylosuje číslo v zadaném rozsahu. Této funkce využijeme pro vylosování náhodné výšky (souřadnice y), ve které poklad poletí. Existuje více možností, jak postupovat, přehledným řešení je např. vytvořit si novou proměnnou `náhoda`, do níž si uložíme vylosovanou hodnotu z *náhodné číslo od _ do _*. Velikost mezních hodnot záleží na výběru, v případě, že chceme, aby poklad létal v jakékoliv výšce, musíme zadat vysoká čísla, kupříkladu -200 (dolní mez) a 200 (horní mez). Hodnotu proměnné `náhoda` zatím nikde nevyužíváme, proto musíme z kategorie *Proměnné* přetáhnout levým tlačítkem myši `náhodu` do bloků *skoč na x: _ y: _* a *klouzej _ sekund na x: _ y: _*, konkrétně do souřadnice y.



Obrázek 76 - Návod

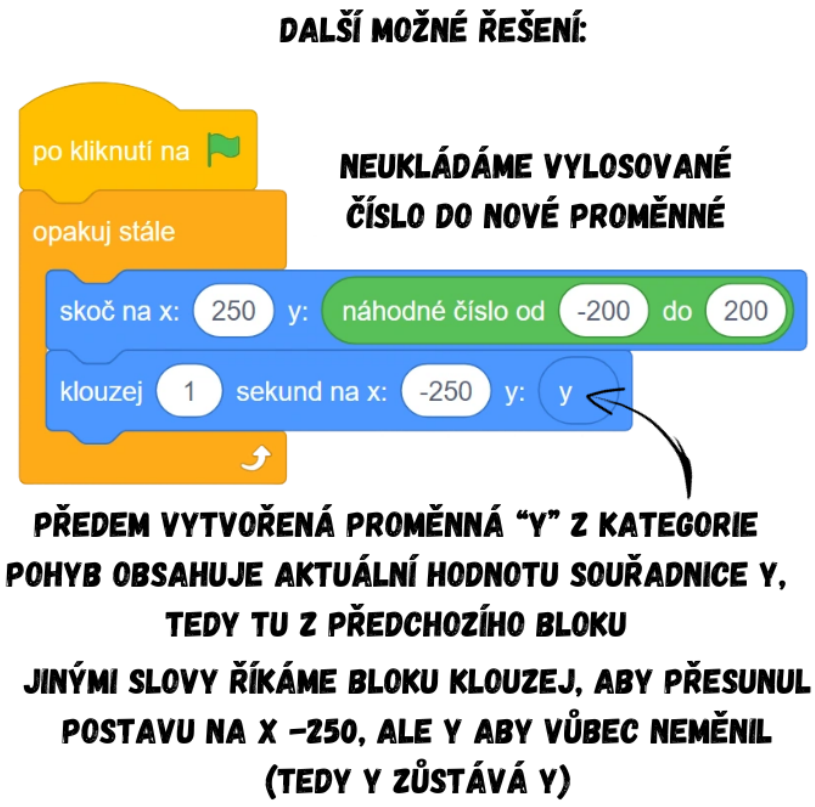
- V případě, že bychom proměnnou vložili pouze do prvního bloku, poklad by sice započal klouzání jinde, ale pokaždé by směřoval do stejných konečných souřadnic. Proto je nutné proměnnou vložit do obou bloků. Poklad tak bude nadále klouzat pouze vodorovně, protože souřadnice y bude v obou blocích vždy přepsaná na aktuální vylosované číslo nacházející se v `náhodě`. V praxi to znamená, že poklad bude létat pouze zprava doleva a již nebude měnit svoji výšku.

Obrázek 77 - Ukázka kódu

- Jedno z možných řešení by mohlo vypadat následovně:

Obrázek 78 - Ukázka kódu

Poznámka: Existuje spoustu možných způsobů, jak úlohu řešit. Vybrané řešení s novou proměnnou považuji za vcelku jednoduché na pochopení a poměrně jasně čitelné v kódu. Možností je kupříkladu celou situaci řešit bez souřadnic podobně jako pohyb Nepřítele v projektu č. 4 Opička. Další z možných řešení využívajících souřadnice může vypadat například takto:



Obrázek 79 - Ukázka kódu

- Pokud nám přijde, že poklady létají příliš velkou rychlostí, můžeme zvýšit počet sekund v *klouzej _ sekund na x: _ y: _*, aby do cíle létaly pomaleji. V případě, že nám vadí, že poklad létá podél horního či dolního okraje, stačí upravit meze v bloku *náhodné číslo od _ do _* na menší hodnoty (např. -175 a 175), aby poklad nelétal úplně při krajích. Jestliže chceme zvýšit rozestupy mezi poklady a létají příliš krátce po sobě, stačí přidat na konec cyklu blok *čekej _ sekund*. Po přidání bloku a vyzkoušení hry ale zjistíme, že poklad nyní takříkajíc „čeká“ na levé straně, dokud nemá letět znovu. Abychom ho v tento okamžik před hráčem skryli, využijeme jednoduchého bloku *skryj se*, jenž vložíme ještě před samotné čekání. Poklad je v tuto chvíli neviditelný, pro jeho opětovné zviditelnění je třeba přidat před klouzání blok *ukaž se*.

Obrázek 80 - Ukázka kódu

6. Vytvoření skóre:

   - Postupujte obdobně jako v projektu Auto. Použijte tlačítko *Vytvoř proměnnou* v kategorii *Proměnné*. Proměnnou nazvěte např. `Skóre` a dejte OK. V pokladu nastavte `Skóre` po zapnutí hry na 0 (*po kliknutí na zelenou vlajku*, *nastav _ na 0*). Dotyk s hráčem si budeme hlídat pomocí cyklu *opakuj stále* a vnořené podmínky *když _ tak* s *dotýkáš se _*. Při splnění podmínky (hráč chytil poklad) se zvýší `Skóre` o 1.
   - Pokud nyní hru vyzkoušíme, zjistíme, že poklad po dotyku s postavou hráče letí dál až do svého cíle, což způsobí, že `Skóre` se nám zvýší hned několikrát. Řešením může být nechat poklad po chycení hráčem zmizet (*skryj se*).

Poznámka: Již nemusíme nikam dávat blok *ukaž se*, protože ten už máme obsažený v cyklu pohybu pokladu. Jelikož oba cykly běží paralelně, poklad se stejně vždy po dokončení klouzání ukáže, přestože bude po dotyku s postavou hráče skrytý.

Obrázek 81 - Ukázka kódu

Poznámka: Může se stát, že někteří žáci navrhnou sloučit obě sekvence do jedné dlouhé. To by však vedlo k nefunkčnosti hry, protože v cyklu pohybu pokladu je např. blok *čekej _ sekund*. Tento blok způsobí, že se sekvence zastaví a kód v podstatě „zamrzne". V takovém případě by nebyla spuštěná podmínka hlídající dotyk s postavou hráče, a proto by na něj nereagovala a hráč by tak nemohl získat bod za chycení pokladu. Podobné trable působí i blok *klouzej _ sekund na x: _ y: _*, který prakticky pozastaví sekvenci, dokud není dokončen. Podmínku, kterou potřebujeme mít neustále spuštěnou, je tak nutné nechat v samostatné sekvenci s cyklem.

7. Výhra:

- K výhře dojde, pokud hráč nasbírá např. 5 bodů, tzn. hra čeká, dokud není `Skóre` 5. K tomu využijte blok *čekej dokud nenastane _*, který přidejte do scénáře postavy hráče a do něhož vložte složenou podmínku. Tu najdeme v kategorii Operátory, kde se nachází blok _ = _, jenž přetáhněte ze seznamu bloků do *čekej dokud nenastane _*. Na prázdná místa lze napsat číslo, text, nebo vložit proměnnou.

Obrázek 82 - Ukázka kódu

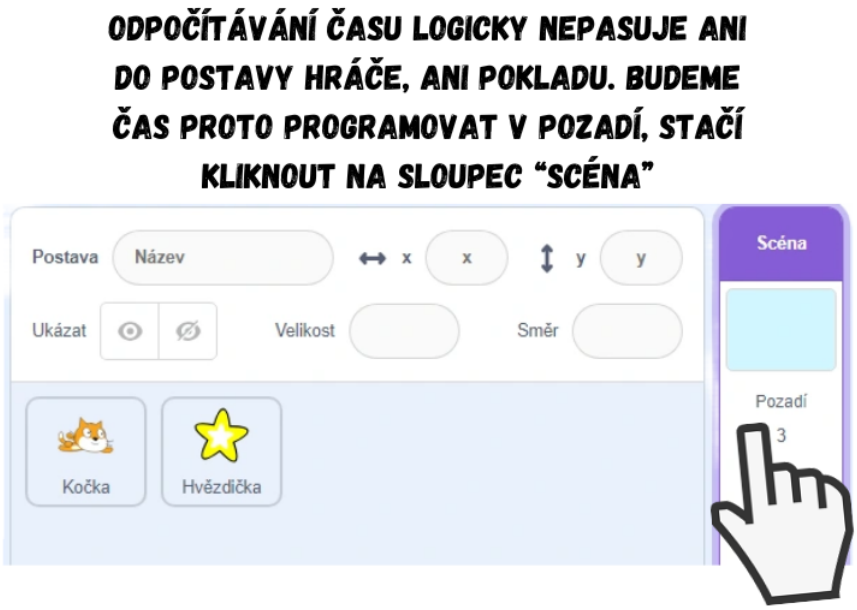
- V našem případě tedy vložíme na první pozici proměnnou, kterou používáme pro počítání skóre, na druhou počet bodů nutný k výhře. Celá podmínka by tedy mohla vypadat následovně:

Obrázek 83 - Ukázka kódu

- Po splnění podmínky napište výherní zprávu a hru ukončete blokem *zastav všechno*.

Obrázek 84 - Ukázka kódu

8. Přidání času & otázka umístění sekvencí:
    - Pro ztížení hry dejte hráči na sesbírání dostatečného počtu bodů časový limit. Vytvořte novou proměnnou, nazvěte ji např. `Čas`.
    - Nyní se nabízí otázka, v jaké postavě začít proměnnou `Čas` programovat. Vždy je vhodné mít sekvence logicky uspořádané a vkládat je tam, kde dávají smysl. Diskutujte s žáky o této problematice.
    - Kvůli logickému uspořádání jsou sekvence pohybu v postavě hráče, respektive pokladu. Dotyk postavy hráče a pokladu jsme mohli vložit do jednoho z aktérů události, výše jsme zvolili poklad, ale sekvence v postavě hráče by byla téměř identická, jenom by se změnila podmínka. Podobně sekvence výhry patří do postavy hráče, protože v ní voláme bublinu „vyhrál jsem!“, což asi nebude křičet poklad. Simulace času ovšem logicky nepatří ani do postavy hráče, ani do pokladu (ani jeden nejsou budík), ideální je tak programovat sekvenci porážky (čas vypršel) v pozadí.

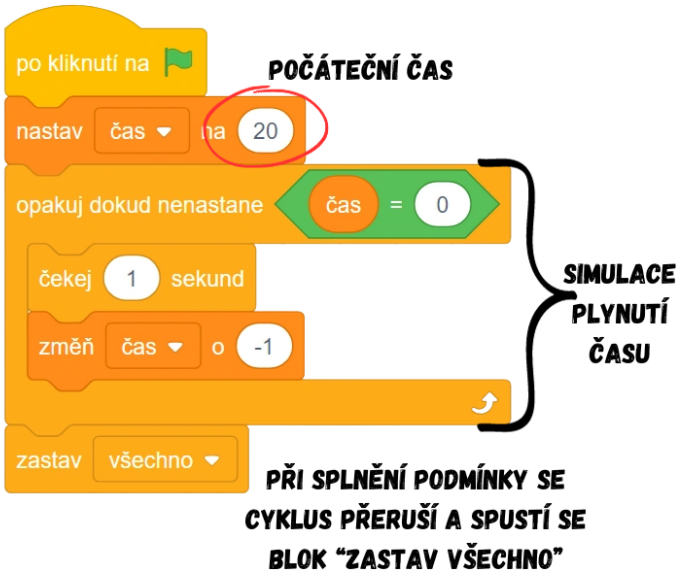


Obrázek 85 - Návod

- `Čas` nastavte na vybraný počet sekund (kupříkladu 20). Posléze chceme každou sekundu snížit čas o 1, abychom simulovali odpočet (budík). Při použití cyklu *opakuj stále* by `Čas` ovšem šel do mínusu a dál. Proto musíme použít jiný cyklus. Pro určení konce odečtu času je vhodný blok *opakuj dokud nenastane _*, který přestane při splnění určité podmínky opakovat svoji činnost. Když tedy čas vyprší (dosáhne nuly), hra se zastaví. Celý blok by tedy mohl vypadat například takto:

Obrázek 86 - Ukázka kódu

- Uvnitř cyklu odečet `Casu` zařídí bloky *čekej 1 sekund* a *změň _ o -1*, celá sekvence je zakončena blokem *zastav všechno*.



Obrázek 87 - Ukázka kódu

Vývoj: Po konzultaci s vedoucím práce a kolegy byla metodika projektu Létání podstatně rozšířena. Pro jasnější představu o tom, jaké jsou vhodné postavy hráče k výběru pro tuto hru, byly do textu přidány obrázky takových postav. Kompletně předělána byla práce se souřadnicemi v Létání, součástí je nyní i aktivita „Slepička“ na seznámení se se soustavou souřadnic, která by vyučujícímu měla ulehčit výklad, jenž tak byl podle toho v metodice pozměněn. Celkový kód ve scénářích postav byl upraven, počítá s lepší znalostí žáků díky aktivitě „Slepička“, původně se se souřadnicemi měli žáci seznámit jen povrchně. Pro živější zpracování hry pro žáky a pro jejich lepší ponoření do hry byl výherní text upraven z „vyhrál jsi“ na „vyhrál jsem“, poněvadž žáci hrají za postavu a ta se raduje, že vyhrála. Kromě stylistických úprav popisu projektu došlo i k přidání dalších možných řešení dílčích problémů. Součástí metodiky je nyní i otázka umisťování sekvencí do postav dle logického uspořádání (sekvence budíku vložena do pozadí apod.).

Průběh ověření: Tato lekce je jednou z těch, která obsahuje více nových informací, proto jsem při výuce nasadil rychlé tempo výkladu, aby se stihla celá hra do 90 minut vytvořit. Zdálo se, že se všichni žáci dokázali bez větších komplikací přizpůsobit nastavené rychlosti výkladu. Obě skupiny zvládly úvodní aktivitu „Slepička“ velmi dobře, většina žáků souřadnice pochopila a na dotazy odpovídali správně. Podle doplňujících dotazů jsem zjistil, že dva žáci v první skupině si princip fungování souřadnic plně

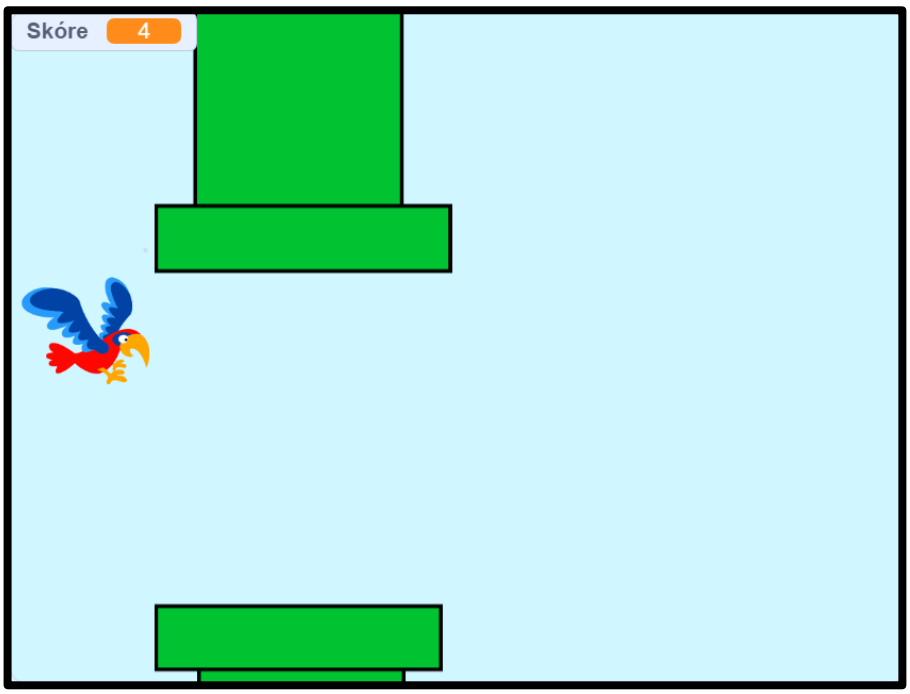
neosvojili, bylo proto nutné celou aktivitu zopakovat ještě jednou. Naprosto bezproblémové bylo programování plynulého pohybu postavy hráče, všichni žáci dokázali samostatně sestavit příslušný kód. Žáci si dokázali poradit i s novými bloky bez větších obtíží, sami přišli na případné chyby v návrzích (v obou skupinách kupříkladu padl návrh vložit blok *náhodné číslo od _ do_* do *skoč na x: _ y: _* i *klouzej _ sekund na x: _ y: _*, vždy po upozornění žáci sami přišli na to, v čem je chyba). Obě skupiny výborně pracovaly s proměnnými, zvládly bez problému programování budíku (odečet `Času`). Žáci správně odpovídaly na mé doplňující dotazy (např. proč nemusím přidávat *ukaž se* do sekvence s podmínkou hlídající dotyk pokladu a postavy hráče, přestože po dotyku se poklad skryje) a dokázali své odpovědi správně odůvodnit a vysvětlit. Projekt se podařilo s oběma skupinami dokončit ve stanoveném čase 90 minut bez jakýchkoli větších komplikací v průběhu tvorby hry.

Úpravy po ověření ve výuce: Při ověření došlo v aktivitě „Slepička“ k chybě, kdy jsem **`Slepičkou`** hýbal vpravo a došel jsem až na konec scény u pravého okraje (tedy za x 240). Souřadnice x posléze byla 246 a aktivitu tak nebylo možné dohrát, protože všechny lokality jsou na souřadnicích zaokrouhlené na celé desítky a **`Slepička`** by po zbytek hry měla souřadnici x končící 6 (v případě záporných čísel 4). Aktivita byla proto opravena a **`Slepička`** nedojde žádným směrem dále, než jaký rozsah ji vymezuje souřadnicová mřížka a vyznačené mezní body. Na základě dotazu žáka, jak zařídit, aby poklad nelétal podél okrajů, byl krok 5 v metodice rozšířen o příslušný návod (menší rozsah v *náhodné číslo od _ do _*). Jelikož padl návrh sloučit obě sekvence (sekvence nastavení `Skóre` a sekvence pohybu pokladu) ve scénáři pokladu do jedné dlouhé sekvence, byla přidána poznámka obsahující detailní výklad, proč by takový návrh vedl k nefunkčnosti celé hry.

9.7 Lekce 7 – Flappy

Délka: 60-70 minut (bez kreslení 45 minut)

O projektu: Hra je variací na známou a populární hru Flappy Bird. Cílem je vyhýbat se jako papoušek překážkám a doletět co nejdále.



Obrázek 88 - Ukázka ze hry

Cíl lekce: Žáci si zopakují a prohloubí své znalosti o souřadnicích a plynulém pohybu. Nově se naučí využívat bloky měnící souřadnice i při základním pohybu, obeznámí se s kostýmy a pozadími a osvojí si práci s nimi.

Nově naučené bloky: *změň x o _*; *změň y o _*; *změň kostým na _*; *další kostým*; *přepni pozadí na _*; *další pozadí*; *když _ tak jinak*

Náplň lekce:

Poznámka: Nedílnou součástí této lekce je i práce s kostýmy v editoru postav. V této lekci je silně doporučeno kreslení nepřeskakovat a pracovat s žáky v editoru. Žáci při kreslení lépe pochopí vysvětlení některých problémů, na které mohou narazit. Pokud žáci chápou princip tvoření kostýmů v editoru, je pro ně posléze mnohem jednodušší pracovat s přepínáním kostýmů a pozadí. V případě nedostatku času můžete příslušné části kroků týkající se editoru postav a otázek s ním spojených přeskočit a využít předpřipravený projekt BIT_7_Flappy_Start se všemi postavami a pozadími. Při zvolení této cesty je třeba dát žákům startovní projekt k dispozici.

1. Úvodní diskuze:
    - Představte žákům zamýšlený projekt a ukažte jim, jak by měl výsledek vypadat, viz BIT_7_Flappy.
    - Diskutujte s žáky o tom, s čím už si jsou schopny poradit díky svým znalostem z předchozích lekcí a s čím si rady neví, co je tedy naopak nové.

Poznámka: Při diskuzi můžete narazit na to, že žáci získali z ukázky přesvědčení, že se překážka skládá ze dvou postav. Vysvětlete jim, že ačkoliv se to tak může zdát, skutečnost bude poněkud jiná. Je vhodné, aby si žáci uvědomili, že kostým jedné postavy se může skládat ze dvou částí oddělených mezerou.

2. Vybrání pozadí:
    - Použijte předpřipravené pozadí oblohy z knihovny Scratch (např. *Blue Sky 2*).
3. Vytvoření postavy hráče:
    - Použijte předpřipravenou postavu v knihovně Scratch vhodnou pro hru (ideálně papoušek *Parrot*, ten má připraveno několik kostýmů, které jsou navíc vhodné pro animaci letu, resp. mávání křídly).
    - Podobně jako u projektu Létání bude moci hráč měnit výšku letu postavy, tedy pouze pohyb nahoru a dolů. Tentokrát však pohyb dolů bude simulovat klesání prostřednictvím gravitace (pád). Postava hráče tedy začne sama klesat vždy, když hráč přestane hýbat postavou směrem nahoru.

Poznámka: Princip fungování pohybu můžete demonstrovat na projektu BIT_7_Flappy_Ovládání, jedná se o upravenou verzi finálního projektu vhodnou k popisu pohybu. V této verzi projektu chybí překážka (místo ní jsou tam pouze dva čtverce, na které postava hráče nijak nereaguje), tudíž nic nebude přerušovat výklad. Ukažte žákům, jak přesně funguje pohyb postavy hráče.

- Přesuňte postavu k levému kraji a naprogramujte plynulý pohyb pro směr vzhůru prostřednictvím cyklu *opakuj stále*, podmínky *když _ tak* a *klávesa _ stisknuta?*. Posléze je nutné přidat ještě směr dolů, který se bude vykonávat vždy. Zprvu nechte žáky pracovat samostatně na základě jejich zkušeností z předchozích lekcí. Sekvence by mohla vypadat například takto:

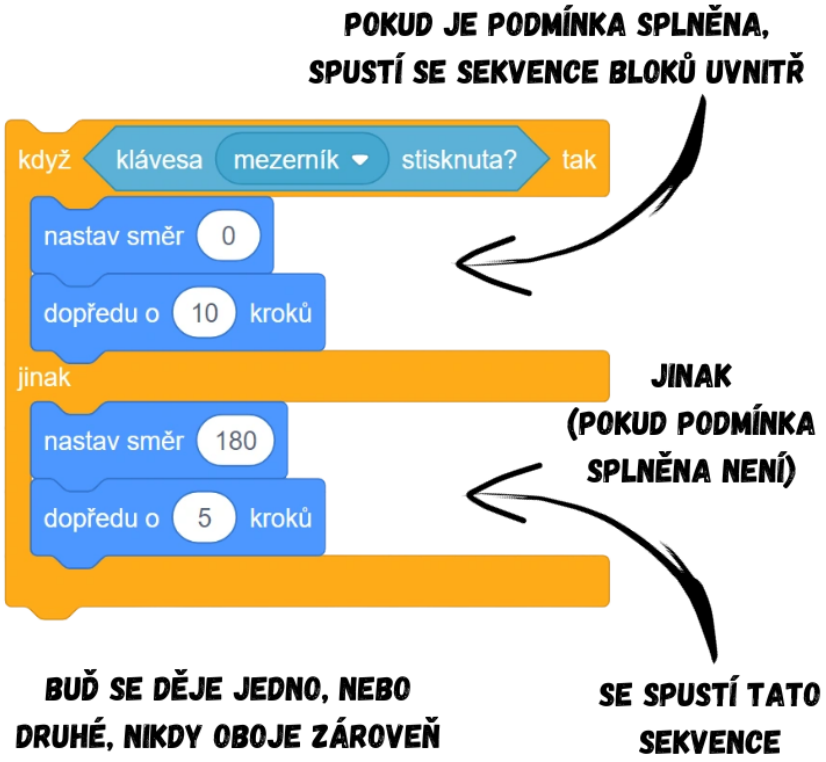
Obrázek 89 - Ukázka kódu

Poznámka: Postava hráče se bude po nastavení směru otáčet vzhůru nohama. Nechte žáky problém vyřešit na základě jejich zkušeností z předchozích projektů. Jednoduše stačí nastavit styl otáčení na „neotáčet“ ve vlastnostech postavy, popř. blokem *nastav otáčení neotáčet*.

- Ukažte žákům, že pokud bychom zadali stejný počet kroků do obou bloků *dopředu o _ kroků*, pouze bychom stiskem klávesy pád pozastavili. Zeptejte se žáků, proč se tak děje.

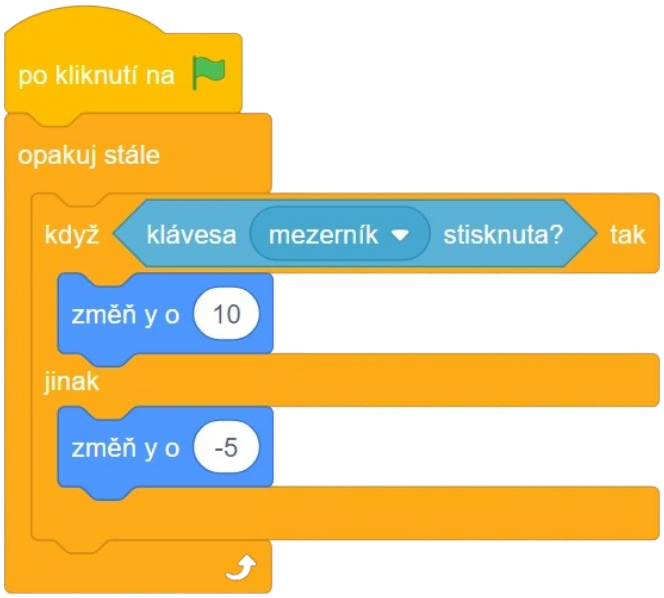
Obrázek 90 – Princip fungování sekvence

- Abychom předešli tomuto „přetlačování“ dvou protichůdných pohybů, můžeme využít blok *když _ tak jinak*. V něm se provádí buď bloky po splnění podmínky, nebo se plní druhá sekvence bloků (příkladem z reálného života budiž „když bude venku hezky, půjdeme ven, jinak zůstaneme doma“, tzn. děje se buď jedno, nebo druhé, ne oboje zároveň).



Obrázek 91 - Ukázka kódu

- Místo využívání kombinace bloků *nastav směr _* a *dopředu o _ kroků* pro pohyb nahoru a dolu lze využít získané znalosti o soustavě souřadnic z předchozího projektu. Zopakujte s žáky, jaké písmeno používáme pro jaký směr (x pro vlevo/vpravo, y pro nahoru/dolu). Následně seznamte žáky s bloky *změň x o _* a *změň y o _*. Zeptejte se žáků, který z těchto dvou bloků využijeme. V této hře nám bude stačit pouze druhý jmenovaný. Zvyšováním y zvyšujeme i výšku, ve které se postava nachází. Nechte žáky, ať přijdou na hodnotu nutnou pro klesání (záporné číslo). Celá sekvence pohybu by mohla vypadat následovně:



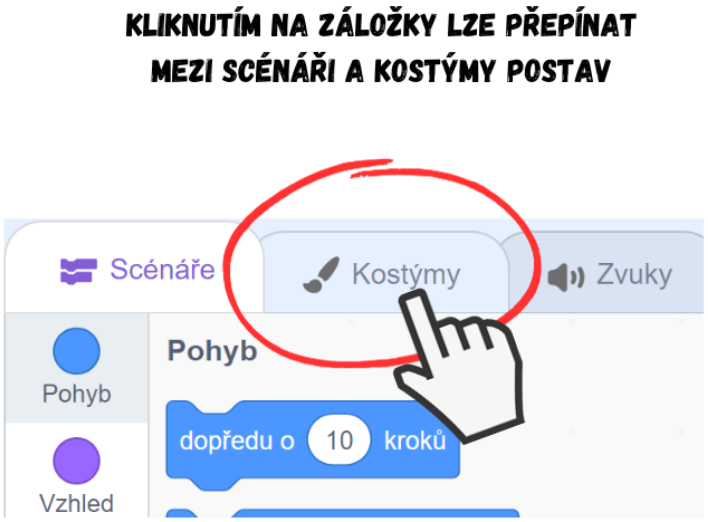
Obrázek 92 - Ukázka kódu

Poznámka: Čím větší zápornou hodnotu zvolíme pro klesání (po jinak), tím rychlejší pád bude. Najděte s žáky ideální hodnotu postupným zkoušením, záleží na zamýšlené obtížnosti hry a možné hratelnosti.

- Nechte žáky, ať naleznou výhody pohybu pomocí bloků *změň x o _* a *změň y o _* a diskutujte s nimi o nich (méně bloků v sekvenci pro pohyb, přehlednější kód, nemusíme řešit nevhodné otáčení vzhůru nohama a upravovat styl otáčení).

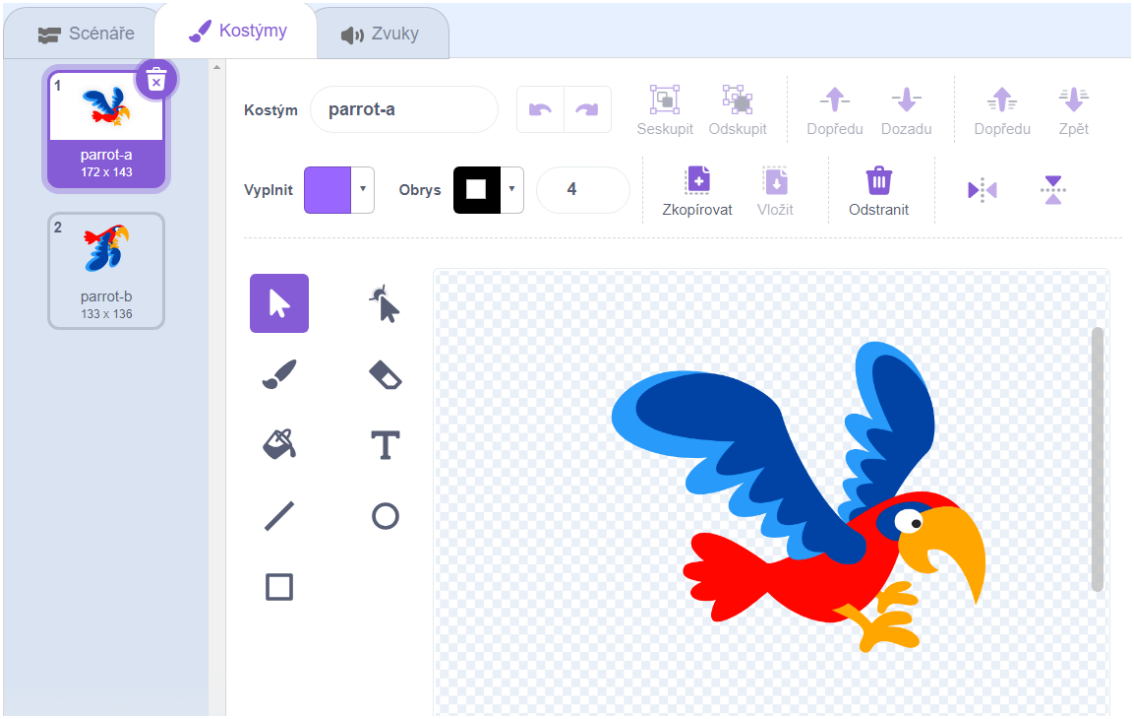
4. Animace postavy hráče:

- Pro vytvoření animace postavy hráče bude třeba, abyste žáky seznámili s kostýmy. Každá postava může mít hned několik kostýmů (vzhledů), mezi kterými může během hry přepínat. Kostýmy naleznete pod záložkou „Kostýmy“.



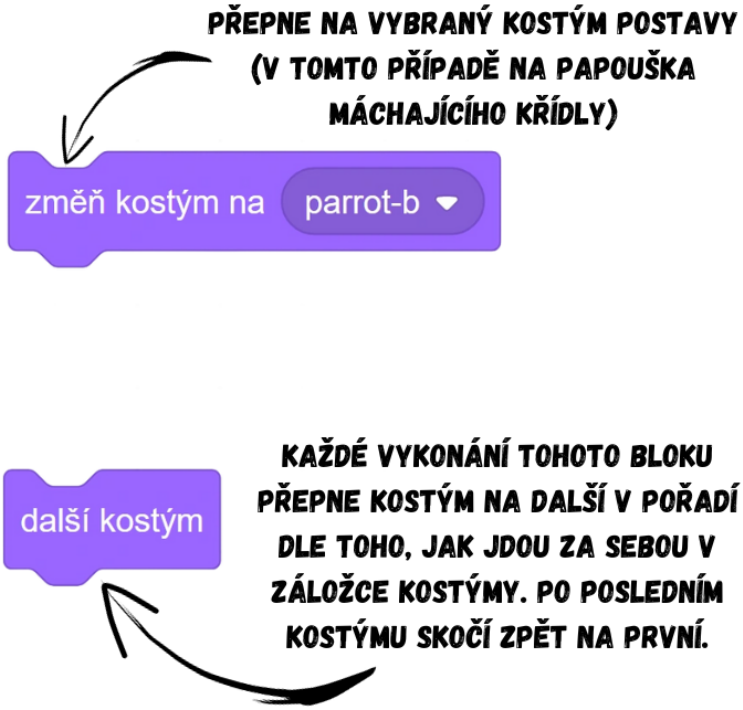
Obrázek 93 - Návod

- Například *Parrot* má předpřipravené dva kostýmy, které se nám budou hodit pro vytvoření animace, konkrétně plachtícího papouška (parrot-a) a papouška máchajícího křídly (parrot-b):



Obrázek 94 – Ukázka obsahu záložky Kostýmy

- Mezi kostýmy lze libovolně přepínat kliknutím myši. Diskutujte s žáky o tom, na jakém principu funguje animace (video, film atd.). Následně měňte rychle kostýmy, aby žáci viděli, že pohyb postavy bude díky naší zrakové nedokonalosti vypadat plynule.
- Přepínat kostýmy můžeme i pomocí bloků z kategorie *Vzhled*, např. *změň kostým na _* či *další kostým*. Vraťte se proto zpět do záložky „Scénáře“.



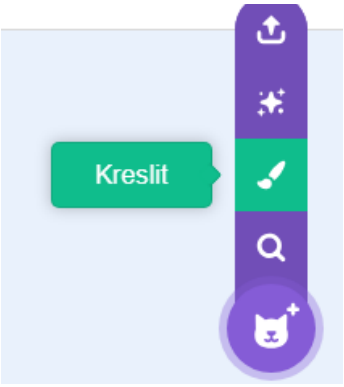
Obrázek 95 - Ukázka kódu

- V našem případě budeme chtít přepínat na konkrétní kostýmy. Když stiskneme danou klávesu, postava hráče změní kostým na máchnutí křídly (parrot-b). Jinak plachtí a klesá pomalu dolů (parrot-b). Celý pohyb i s animací by tedy v případě použití postavy *Parrot* vypadal následovně:

Obrázek 96 - Ukázka kódu

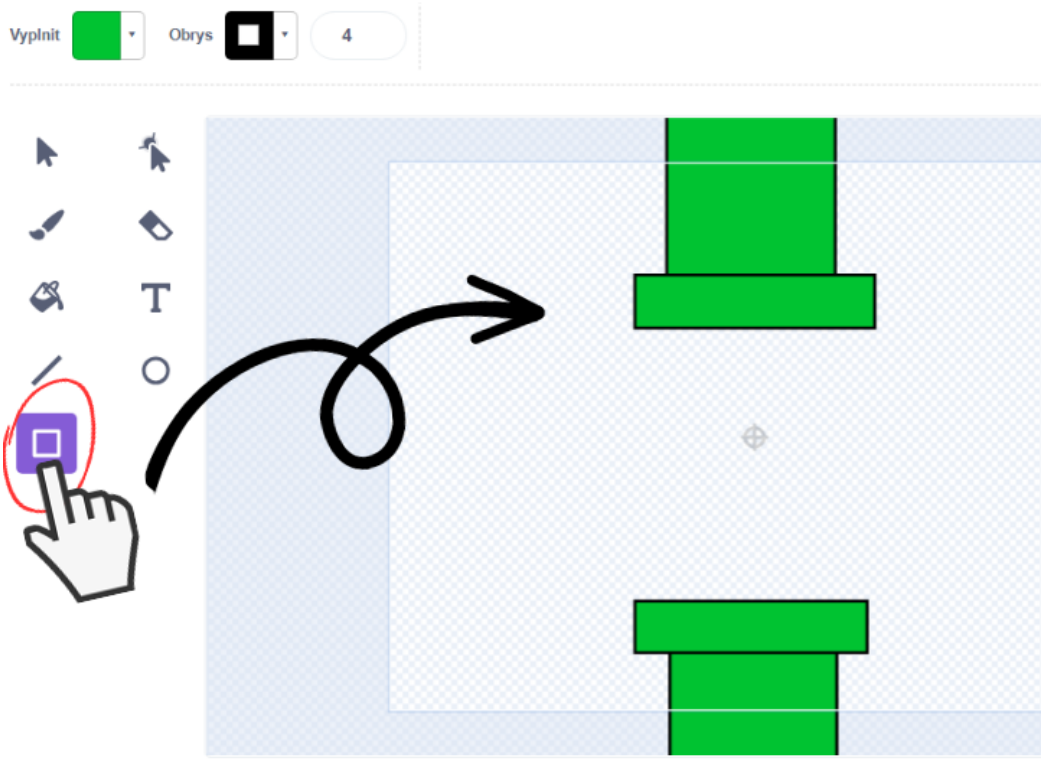
5. Vytvoření překážky:

   - Vytvořte novou postavu v editoru postav, nazvěte ji **`Překážka`**.



Obrázek 97 - Návod

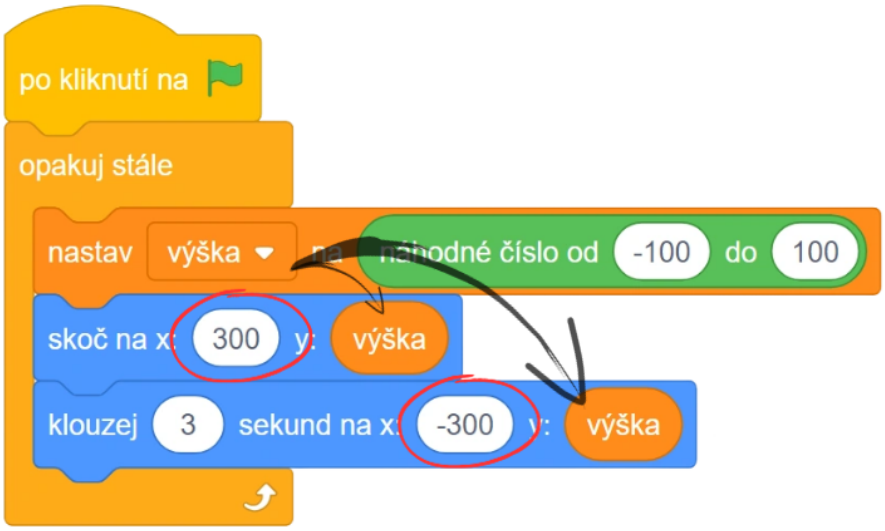
   - Nakreslete jednoduchou překážku (měla by se táhnout shora i zdola, mezera uprostřed). Použijte nástroj Obdélník, **`Překážka`** by se měla skládat pouze ze 4 obdélníků, viz ukázka:



Obrázek 98 - Návod

   - Ve scénáři naprogramujte jednoduchý pohyb **`Překážky`** zprava doleva na základě znalostí získaných v projektu Létání. Využijte bloků *skoč na x: _ y: _* a *klouzej _ sekund na x: _ y: _* ke skoku **`Překážky`** úplně vpravo a následnému klouzání úplně vlevo. Zkuste nechat žáky pracovat samostatně, cílem je vytvořit pohyb **`Překážky`** zprava doleva a pokaždé v náhodné výšce. Vytvořte novou proměnnou `výška`, kterou nastavíte na náhodné číslo ve vhodném rozmezí dle velikosti postavy (např. -100 a

100). Hodnotu `výšky` poté využívejte v souřadnicích y v obou pohybových blocích. Řešením by mohlo být např. toto:

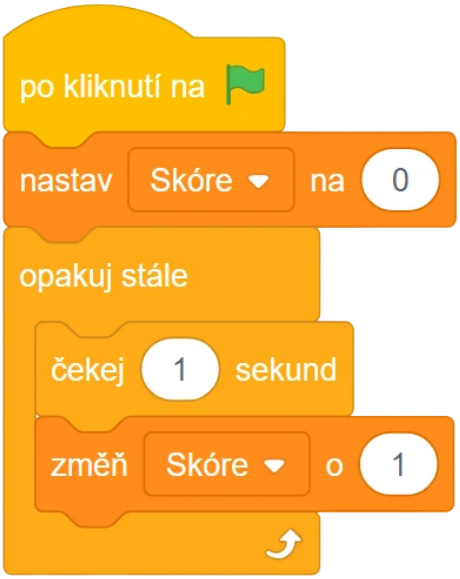


Obrázek 99 - Ukázka kódu

Poznámka: Pokud žáci nakreslili kostým **`Překážky`** široký, zadané hodnoty pro x (300 a -300) nemusí být dostatečné. Postava tak může začínat svůj pohyb dále od okraje, a naopak končit svůj pohyb ještě před druhým okrajem, tedy dříve, než je záhodno. Řešením je dát postavě větší prostor na pohyb, tzn. zvětšit rozmezí pohybu zadáním většího x a následným zkoušením nalézt ideální hodnotu pro danou postavu.

Poznámka: Podobně jako u projektu Létání i zde by šlo situaci řešit jinak než s novou proměnnou, např. využitím již předpřipravené proměnné obsahující hodnotu souřadnice y. Pro detailnější rozbor celé problematiky viz krok č. 4 v projektu Létání.

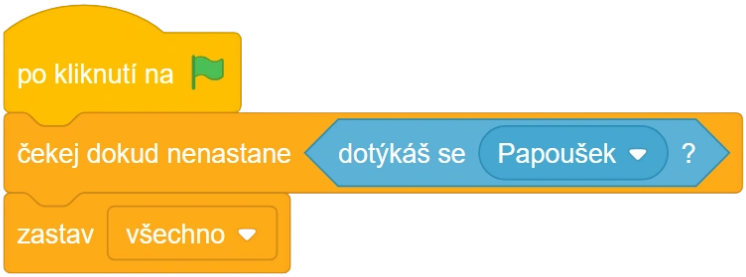
6. Vytvoření skóre:
    - Nechte žáky pracovat samostatně. Vytvořte novou proměnnou `Skóre` a naprogramujte jí tak, aby zvýšila svou hodnotu o 1 za každou sekundu. Diskutujte s žáky, kam kód pro `Skóre` umístit (vhodný je scénář pozadí, protože se `Skóre` zvyšuje nezávisle na postavách).



Obrázek 100 - Ukázka kódu

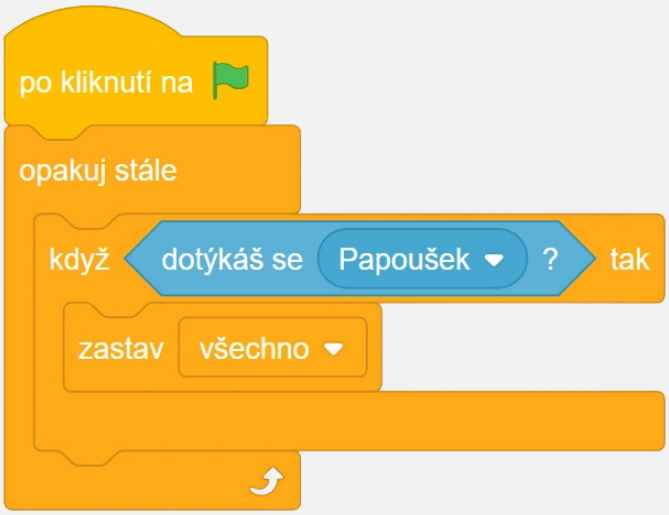
7. Porážka:

   - Pro naprogramování porážky se přesuňte do scénáře **`Překážky`**, popř. postavy hráče. Nechte žáky, ať vytvoří jednoduchou sekvenci, která bude čekat na dotyk překážky s postavou hráče a posléze hru zastaví (ukončí).



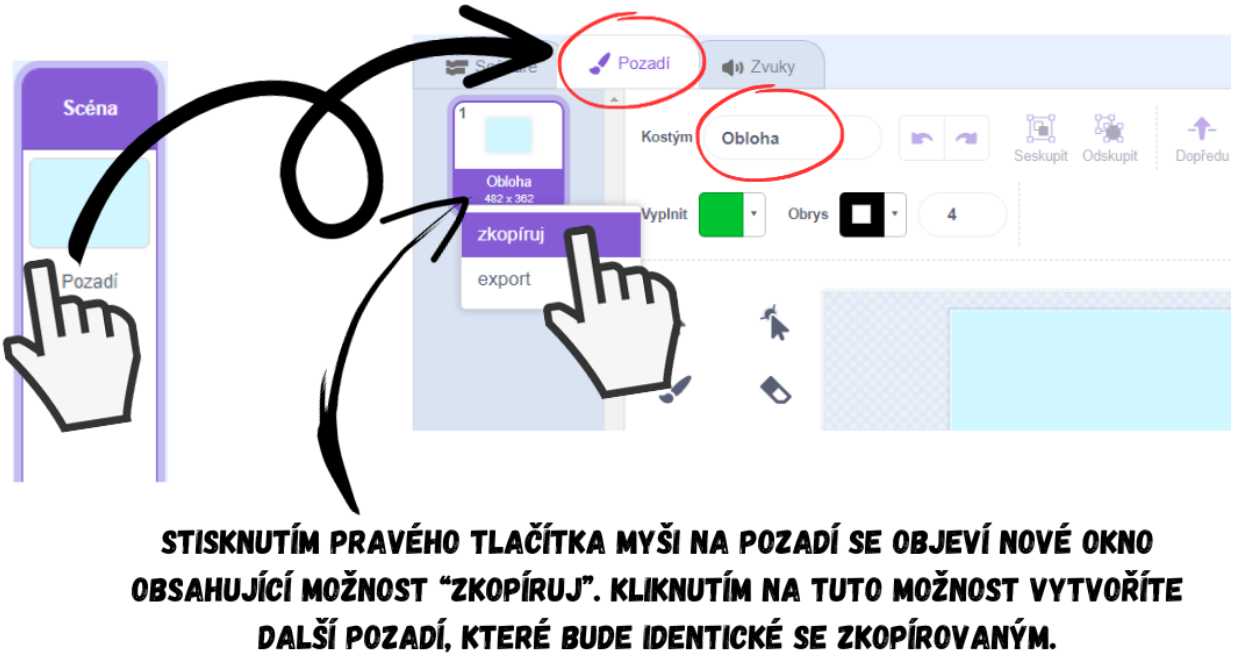
Obrázek 101 - Ukázka kódu

Poznámka: Toto samozřejmě není jediné řešení, možností je například i následující kód:



Obrázek 102 - Ukázka kódu

- Vhodné je hru zakončit spolu se zobrazením textu „Prohrál jsi“, či „Game over“. Existuje několik cest, jak tohoto docílit, jednou z možností je přepnout pozadí. Ukažte žákům, jaký je postup. Vybráním pozadí a kliknutím na záložku „Kostýmy“ si zobrazíte všechna pozadí hry. Pozadí si náležitě pojmenujte, v případě *Blue Sky 2* kupříkladu **`Obloha`**. Zkopírujte toto pozadí.

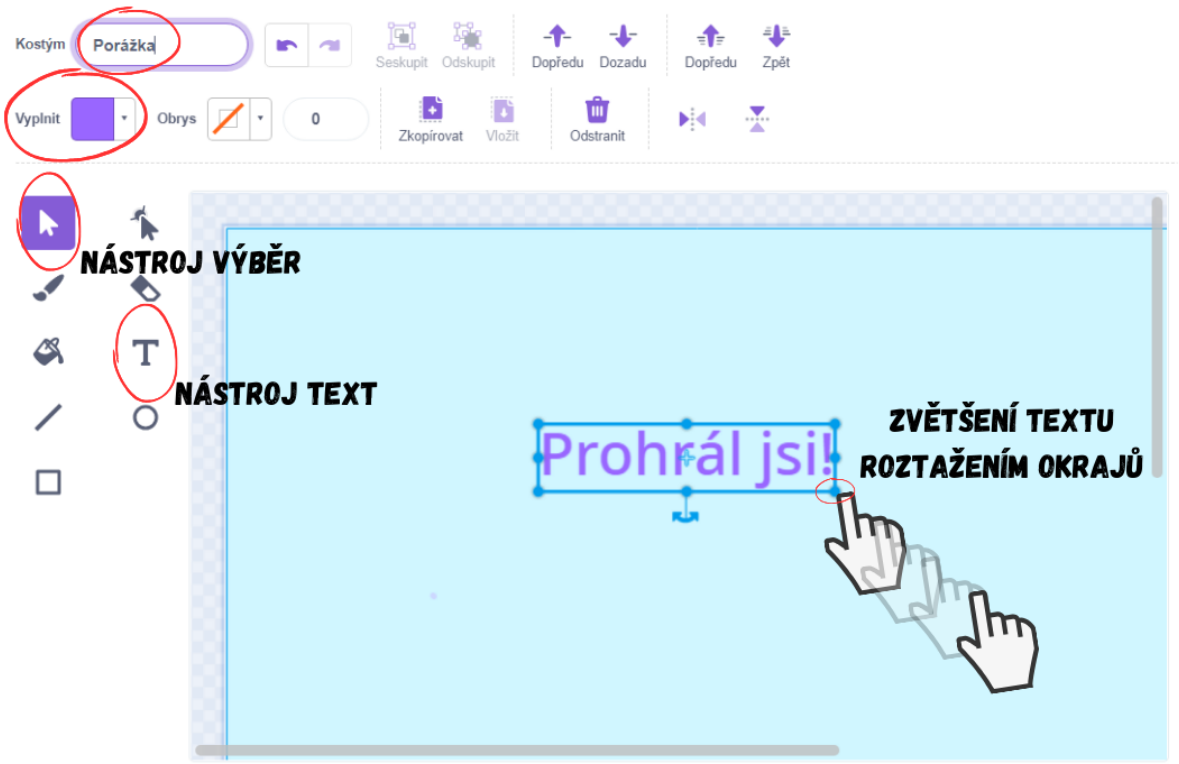


Obrázek 103 - Návod

Poznámka: Jak bylo zmíněno výše, existuje několik možností, jak zobrazit text ve scéně při porážce. Někteří žáci mohou navrhnout místo nového pozadí vytvořit novou postavu, jejíž kostým se bude skládat pouze z textu, jenž budeme zobrazovat. Zobrazení této postavy při porážce by se ovšem řešilo pomocí vysílání zpráv, které jsou předmětem až 8. projektu Dálnice. Na podobné návrhy tedy reagujte vysvětlením, že má žák pravdu, ale že řešení

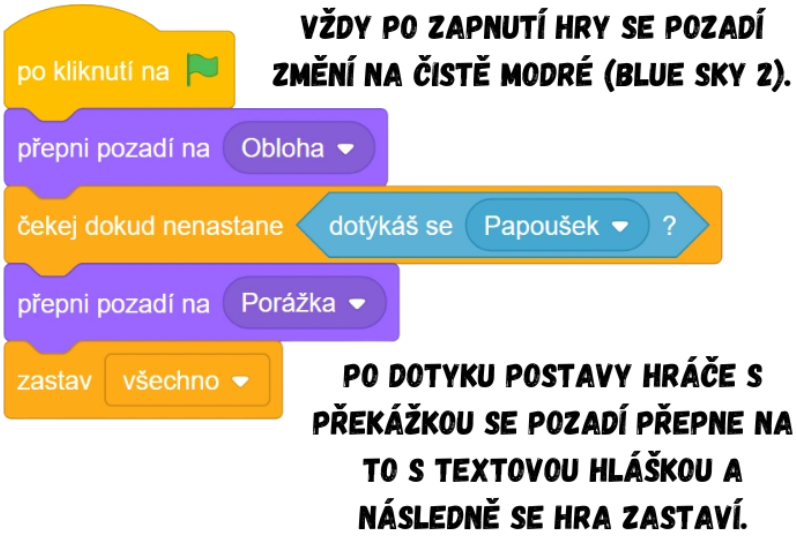
s postavou vyžaduje další znalosti, které žáci získají později. Proto volíme řešení s pozadím, ke kterému stačí pouze pochopit blok *přepni pozadí na _*.

- Zkopírované pozadí pojmenujte např. **`Porážka`** k odlišení s předchozím. Přidejte do pozadí text „Prohrál jsi“ pomocí nástroje Text. Po výběru nástroje klikněte kamkoliv do obrázku a začněte psát. Následně pomocí nástroje Výběr klikněte na text a vyberte barvu výplně, popř. upravte velikost nápisu roztažením okrajů.



Obrázek 104 - Návod

- Zbývá už jen nastavit, kdy se má zobrazit jaké pozadí. K tomu slouží bloky *přepni pozadí na _* a *další pozadí*, které fungují stejně jako jejich obdoba *změň kostým na _* a *další kostým*. Blok *přepni pozadí na _* změní pozadí na konkrétní vybrané, *další pozadí* pouze přepne na další v pořadí, v případě neexistence dalších pozadí začne zase od prvního v pořadí. Využijte sekvence porážky hlídající dotyk postavy hráče s **`Překážkou`** a přidejte do ní bloky *přepni pozadí na _*. Po dotyku **`Překážky`** a postavy hráče přepněte pozadí na **`Porážka`**. Po zapnutí hry nyní zjistíme, že textová hláška prohrál jsi zůstává po prohře neustále zobrazena, proto musíme pozadí restartovat a při každém zapnutí hry zobrazit čisté nebe, tj. pozadí **`Obloha`**.



Obrázek 105 - Ukázka kódu

Vývoj: Pohyb postavy hráče se původně měl vysvětlovat na dokončeném projektu Flappy Bird. Jelikož se v dokončené verzi ale pohybují překážky a postava hráče na dotyk s nimi reagovala porážkou, byl tak výklad neustále přerušován. Po konzultaci s vedoucím práce proto byla vytvořena upravená verze BIT_7_Flappy_Ovládání, která je vhodná pro vysvětlení fungování pohybu postavy hráče, protože neobsahuje žádné překážky, na které by bylo reagováno. Metodika reflektuje změny v předchozích projektech, Flappy Bird tak nyní díky předpokladu hlubší znalosti souřadnic z předchozího projektu obsahuje programování pohybu pomocí bloků *změň x o _* a *změň y o _*. Zobrazení textu spojeného s porážkou (nápis „Game Over“ apod.) bylo původně řešeno prostřednictvím nové postavy, jejíž kostým se skládal pouze z textu. To ovšem vyžadovalo vyslání zprávy, se kterým se žáci zatím nesetkali. Proto bylo nakonec zvoleno jednodušší přepínání mezi pozadím pro hru a pro porážku a vysílání zpráv bylo odsunuto až do projektu 8 Dálnice. Nastavování proměnné `Skóre` je nyní umístěno ve scénáři scény (pozadí) na základě otázky logického umisťování sekvencí.

Průběh ověření: Na rozdíl od projektu č. 6 Létání je tato lekce spíše opakovacího charakteru, většinu věcí žáci znají a své znalosti tak pouze rozšiřují. Tomu odpovídal i styl výkladu, spíše jsem pokládal otázky a dělal to, co mi žáci poradili. Řešení si tak obě skupiny prakticky vytvořily samy, já postup pouze korigoval. Žáci v obou skupinách dokázali samostatně vytvořit pohyb postavy hráče, dokázali i přijít na to, proč postava neletí nahoru, když zadali stejný počet kroků pro oba směry (např. pohyb nahoru i pohyb dolu byl 10 kroků). Všichni také již samostatně dokážou vyřešit problém s nechtěným otáčením postavy. Zdálo se, že žáci chápou rozdíl mezi *když _ tak* a *když _ tak jinak*, v druhé skupině nový blok nalezli sami ještě před mým upozorněním. Obě skupiny během tvorby převzaly iniciativu a

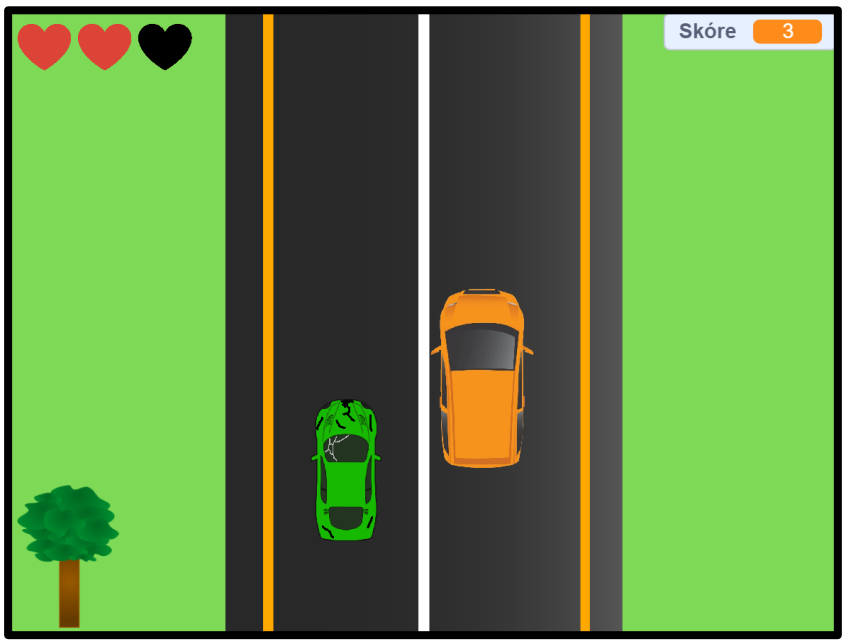
začaly navrhovat různá vylepšení či rozšíření do hry (aby došlo k porážce, když se postava dotkne dolního okraje), k jejichž realizaci bohužel nakonec kvůli nedostatku času nedošlo. Žáci si bez větších problémů osvojili práci s kostýmy a pozadími. Obě skupiny dokázaly hru dokončit za přibližně 70 minut a vypadalo to, že je projekt Flappy Bird velmi bavil.

Úpravy po ověření ve výuce: Po ukázce cílového projektu z následné diskuze vyplynulo, že si žáci mysleli, že **`Překážka`** se skládá ze dvou postav, byla proto do metodiky přidána příslušná poznámka. Některým žákům nebylo zpočátku úplně jasné, jak funguje přetlačování dvou protichůdných pohybů, nakreslil jsem proto na tabuli obrázek, na kterém jsem vysvětlil, co se děje s postavou krok po kroku (postava hráče o 10 kroků klesne, poté o 10 kroků vzletí, proto zůstává na místě). Tento obrázek byl nyní přidán i do metodiky v rámci tvorby pohybu. Na základě návrhu jednoho z žáků zobrazit nápis „Game Over“ jako novou postavu, jejíž kostým by obsahoval pouze text, byla do metodiky přidána poznámka obsahující rozbor tohoto řešení a zdůvodnění, proč v tomto projektu použijeme řešení jiné – přepínání pozadí.

9.8 Lekce 8 – Dálnice

Délka: 90 minut

O projektu: Ve hře hráč hraje za závodní auto, které předjíždí ostatní řidiče na dálnici. Úkolem je vyhýbat se ostatním a nenabourat do nich. Čím déle hráč vydrží ve hře, tím větší je jeho konečné skóre.



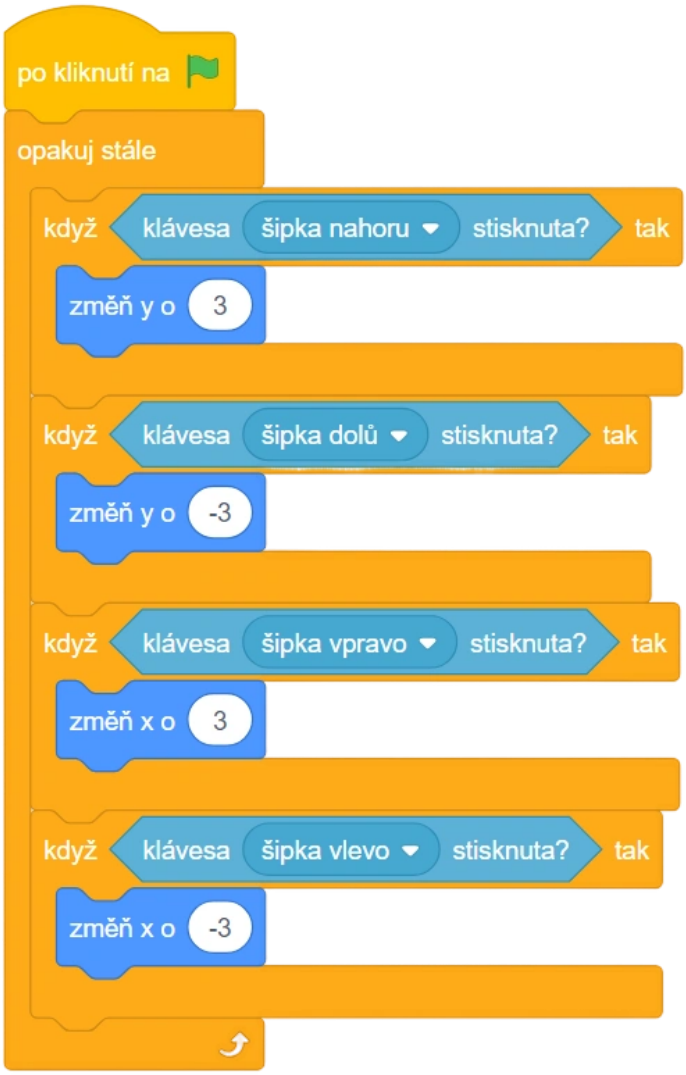
Obrázek 106 - Ukázka ze hry

Cíl lekce: Žáci si zopakují své znalosti o kostýmech a zdokonalí se v používání souřadnic a proměnných. Naučí se používat zprávy a navázat na ně reakci hned několika postav najednou.

Nově naučené bloky: *po obdržení zprávy _*; *vyšli zprávu _*; _ < _

Náplň lekce:

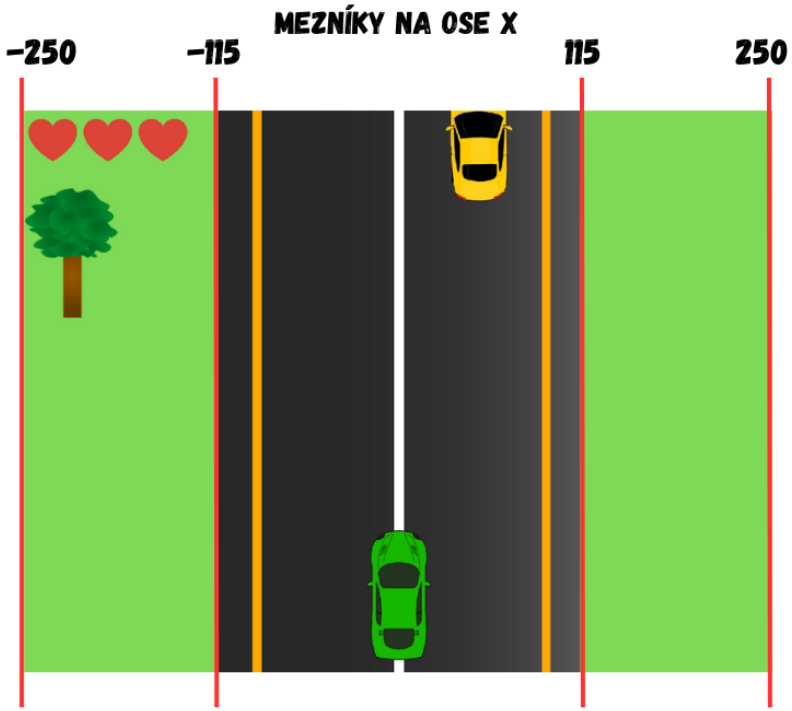
1. Úvodní diskuze:
    - Představte žákům zamýšlený projekt a ukažte jim, jak by měl výsledek vypadat, viz BIT_8_Dálnice.
    - Diskutujte s žáky o tom, s čím už si jsou schopny poradit díky svým znalostem z předchozích lekcí a s čím si rady neví, co je tedy naopak nové.
2. Stáhněte si počáteční projekt:
    - Použijte připravený projekt BIT_8_Dálnice_Start, který obsahuje všechny postavy a pozadí potřebné pro hru. Žákům na začátku hodiny projekt nasdílejte, popř. dejte ke stažení.
3. Pohyb postavy hráče:
    - Nechte žáky naprogramovat pohyb automobilu všemi směry pomocí bloků *změň x o _* a *změň y o _*. Důležité je připomenout žákům, že x měníme pro pohyb vlevo/vpravo, y pro pohyb nahoru/dolu. Sekvence by mohla vypadat například takto:



Obrázek 107 - Ukázka kódu

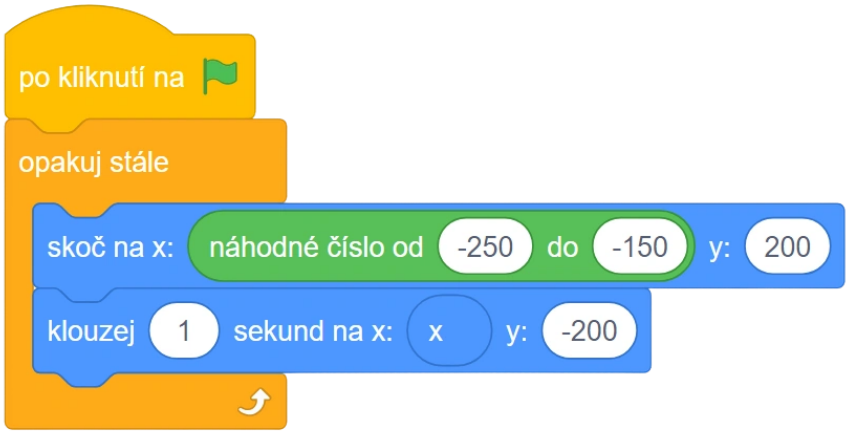
4. Iluze pohybu:

    - Abychom docílili iluze pohybu, tzn. aby to vypadalo, že auto jede po dálnici a míjí ostatní vozidla, bude třeba rozhýbat okolí. V projektu se nachází dvě postavy nacházející se mimo silnici – **`Strom1`** a **`Strom2`**. Pro jejich pohyb shora dolů využijte již známé kombinace *skoč na x: _ y: _* a *klouzej _ sekund na x: _ y: _*. Zadání správných souřadnic může být složitější, je proto dobré si vyzkoušet ve vlastnostech postav, kde jsou přibližně důležité mezníky na pozadí, viz níže:



Obrázek 108 - Pomůcka pro orientaci ve scéně

- Pokud chceme, aby se **`Strom1`** pohyboval na levé straně od vozovky, tzn. v rozmezí -250 až -115, je vhodné hodnotu u silnice trochu zmenšit, aby koruna stromu nezasahovala do vozovky. Pro skok hodně nahoru využijeme vysoké kladné y, pro pohyb úplně dolů vysoké záporné y. Možností by mohlo být např.:



Obrázek 109 - Ukázka kódu

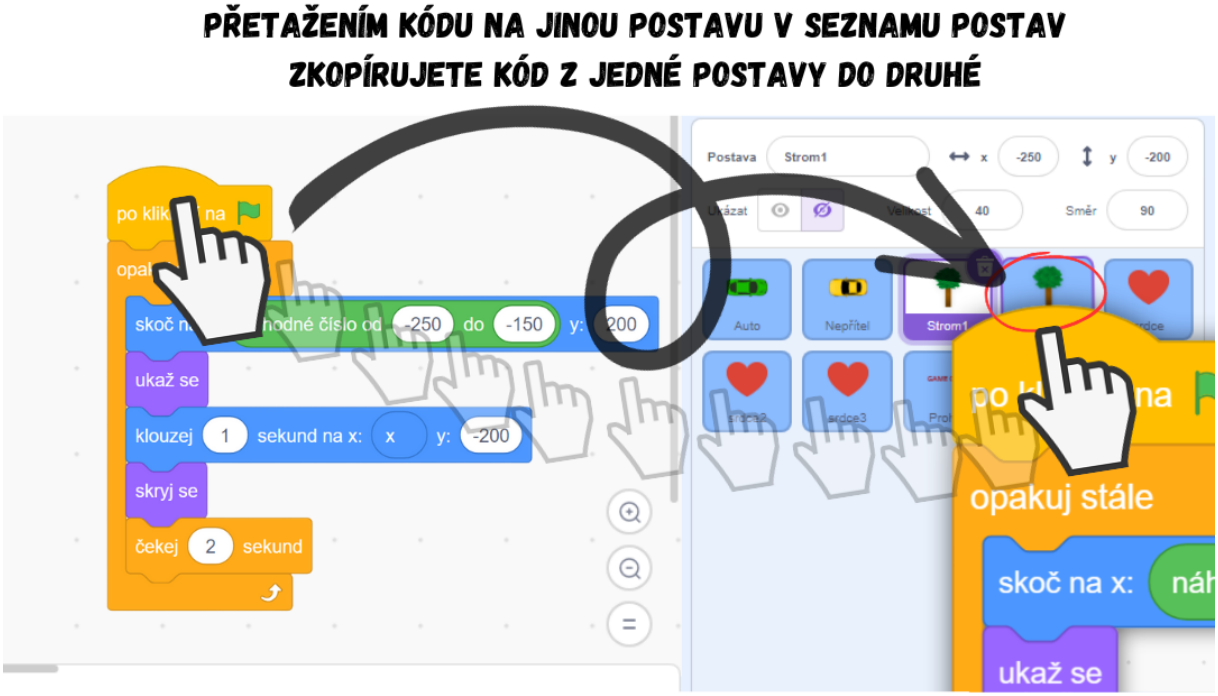
Poznámka: Výše v kódu využíváme předpřipravené proměnné `x`, celou situaci ale lze řešit i vytvořením nové pomocné proměnné, do které se uloží vylosovaná hodnota a následně ji vložíme do obou bloků na pozici souřadnice x.

Obrázek 110 - Návod

- **`Strom1`** se nyní pohybuje shora dolů a díky náhodnému číslu pokaždé jinde. Pokud žákům přijde, že se strom pohybuje příliš rychle, stačí prodloužit blok klouzej na více sekund. Než rychlost pohybu spíše nerealisticky vypadá to, že **`Strom1`** dokončí svůj pohyb a hned vyráží další. Diskutujte s žáky o možném řešení. Nejjednodušší je nastavit mezi dokončeným a dalším pohybem menší časový odstup. Toho lze docílit pomocí bloku *čekej _ sekund*, jenž ovšem nyní způsobí, že strom „čeká“ na konci své trasy na další pohyb, proto je třeba ho po dokončení pohybu skrýt a logicky na začátku pohybu zase ukázat. Sekvence by mohla být následovná:

Obrázek 111 - Ukázka kódu

- Podobný kód budeme potřebovat i pro rozpohybování dalšího stromu, zkopírujte proto celou sekvenci a vložte ji do postavy **`Strom2`**.

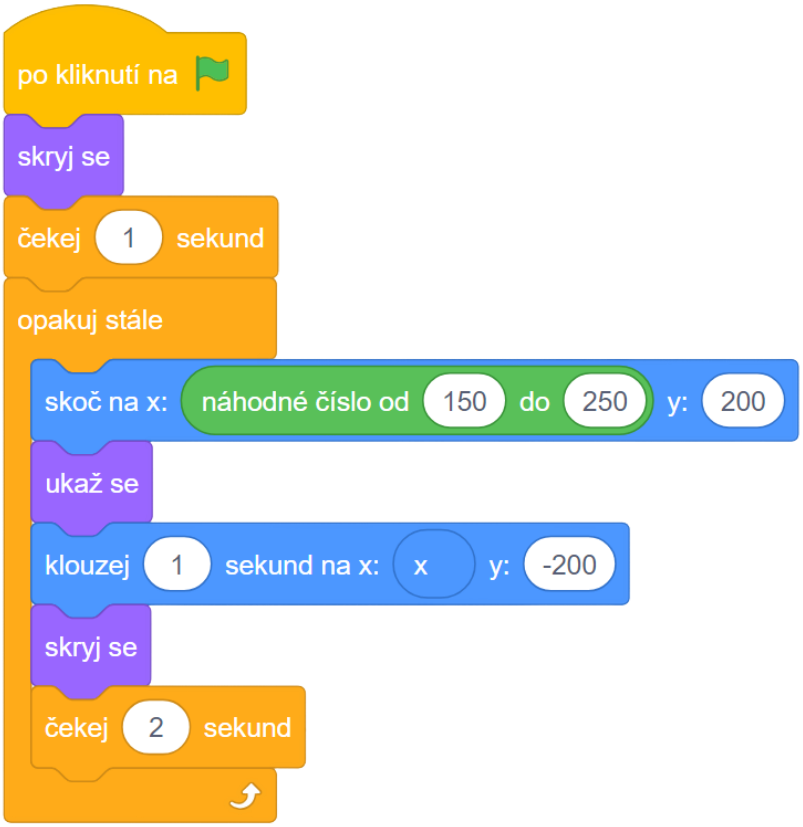


Obrázek 112 - Návod

- Nechte žáky pohyb stromů vyzkoušet. Žáci zjistí, že nyní oba stromy putují na levé straně. Zeptejte se žáků, jak problém vyřešit a co je nutné v kódu upravit. Jelikož se **`Strom2`** nachází v pravé části pozadí, musíme upravit hodnotu x.

Obrázek 113 - Ukázka kódu

- Nerealisticky vypadá nyní to, že oba stromy putují zároveň. Nechte žáky, ať problém vyřeší samostatně. Možným řešením je start jednoho stromu o chvíli opozdit, aby se pohybovaly střídavě. Čekající strom ale nesmí zůstat stát na planině, je třeba ho skrýt, než se začne hýbat.

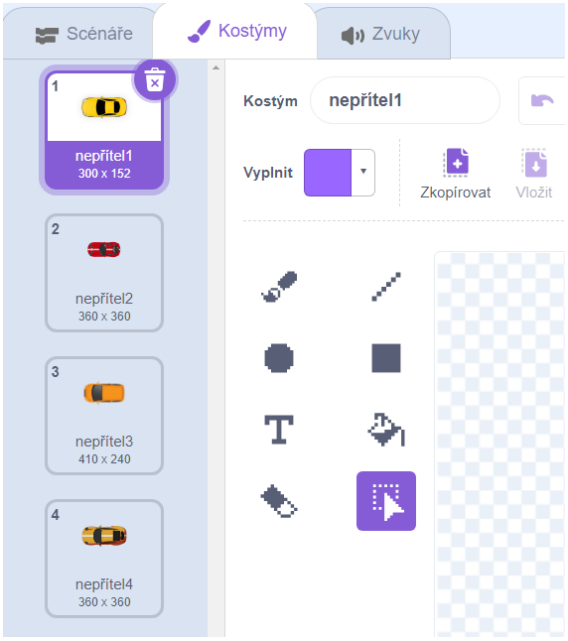


Obrázek 114 - Ukázka kódu

Poznámka: Žáci mohou přijít s různými řešeními problému dvou stejně putujících stromů, popř. i s různými vylepšeními. Možností je kupříkladu využití bloku *náhodné číslo od _ do _*, který žáci mohou vložit do *čekej _ sekund*. Diskutuje s žáky nad jejich řešením.

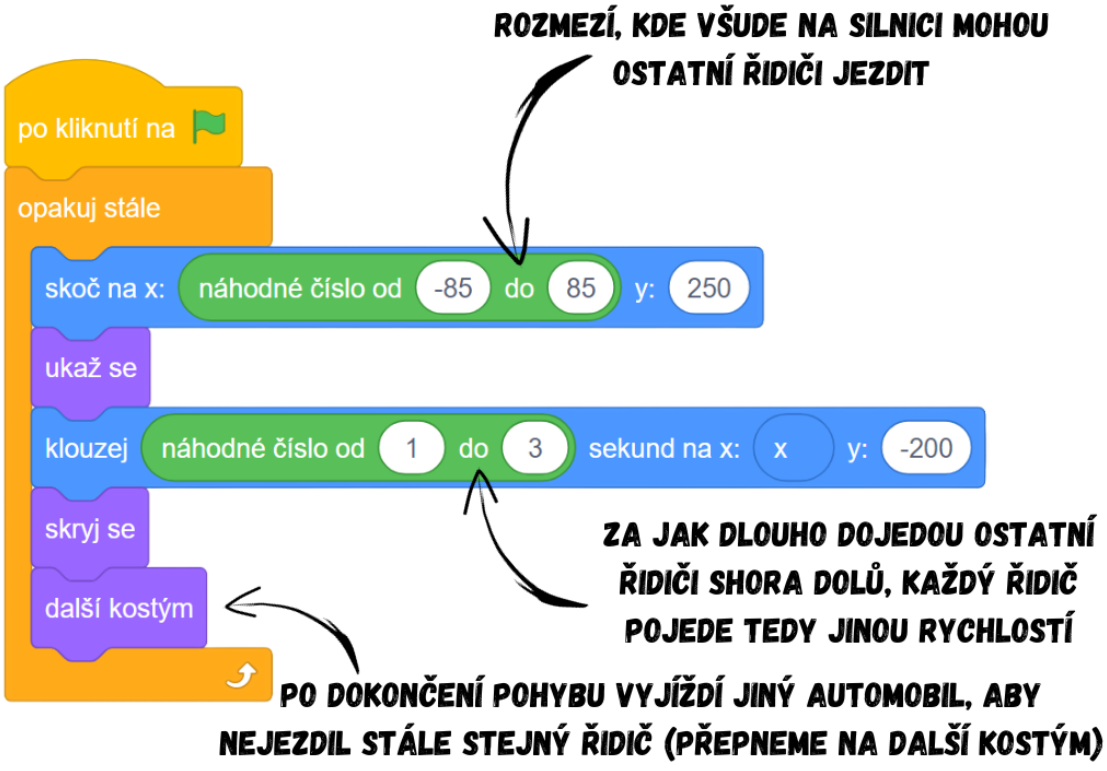
5. Pohyb nepřítele

   - Podporovat iluzi pohybu budou i okolní řidiči, kteří ve skutečnosti pojedou „pozpátku“, tzn. budou pomalu couvat ke stojícímu automobilu hráče. Ve hře to ale bude vypadat tak, že automobil pomalu dojíždí okolní řidiče, až je předjede.
   - Postava **`Nepřítel`** obsahuje hned několik připravených kostýmů, jejich přepínáním (*další kostým*) tak zajistíme iluzi toho, že na dálnici jezdí různá auta a různí řidiči, ve skutečnosti všechno bude jedna postava.



Obrázek 115 - Ukázka obsahu záložky Kostýmy

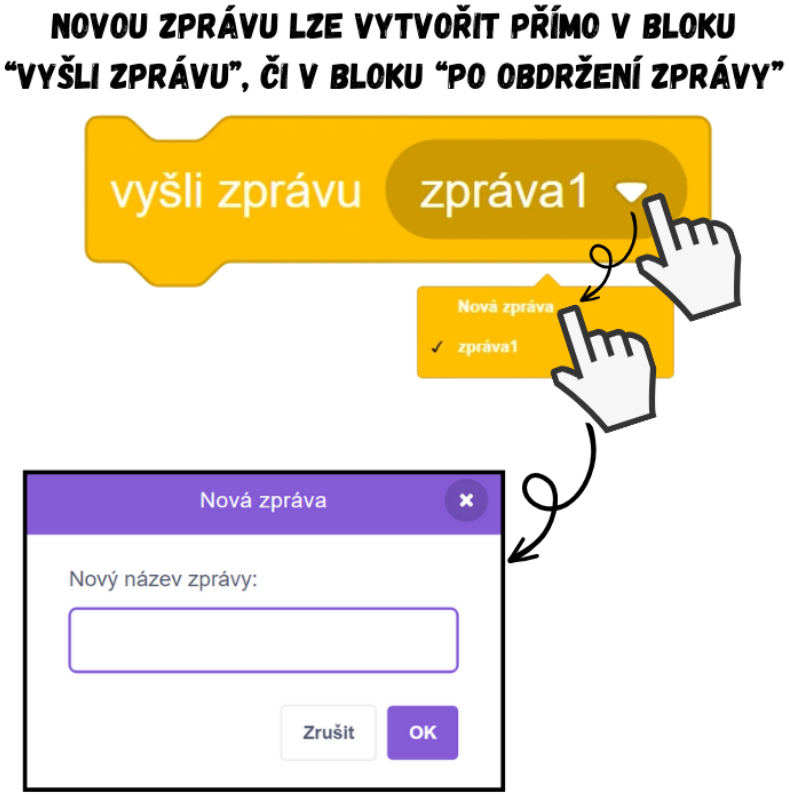
- Pohyb nepřítele se dá naprogramovat hned několika cestami. Variantou může být např. použití bloku *klouzej _ sekund na x: _ y: _*, kód pak bude podobný tomu v postavách **`Strom1`** a **`Strom2`**:



Obrázek 116 - Ukázka kódu

6. Náraz

- Jakýmsi středobodem celého projektu je samotná kolize vozidla hráče s ostatními auty či stromy. Když dojde k nárazu, ovlivněno tím bude hned několik postav, a navíc tato událost může spustit i sekvenci porážky a ukončit hru:
    a. **`Auto`** (postava hráče) – po nárazu má změnit kostým na více poškozený automobil
    b. **`Nepřítel`** (ostatní auta), případně náraz do **`Strom1`** či **`Strom2`** – po nárazu se mají skrýt
    c. **`srdce`**, **`srdce2`**, **`srdce3`** – po nárazu mají změnit kostým (ubírání životů), pokud i poslední srdce zčerná, hra končí
- V situacích, kdy má jedna událost spustit sekvence hned v několika postavách, je vhodné využít ve Scratchi zpráv. Pro práci se zprávami ve Scratchi existují pouze dva bloky – *vyšli zprávu _* a *po obdržení zprávy _*. Spuštěním bloku vyšli zprávu se vybraná zpráva vyšle do všech postav (i do scény). Na zprávu mohou ostatní postavy (či scéna) reagovat pomocí bloku *po obdržení zprávy _*. Na tento blok lze navázat bloky, které se mají po obdržení dané zprávy spustit. S žáky si toto můžete jednoduše vyzkoušet i v reálném životě pomocí jednoduché hry: Vyučující poklepe na rameno náhodně vybranému žákovi, ten po poklepání zavolá „Potlesk“ (vyšle zprávu). Všichni ostatní žáci jsou „naprogramováni“ tak, aby poté, co uslyší „Potlesk“ (po obdržení zprávy), zatleskali. Z této hry by žáci měli být schopni říci, v čem je výhoda vysílání zpráv. Diskutujte o možných výhodách používání zpráv ve Scratchi (na jednu událost může reagovat hned několik postav najednou).



Obrázek 117 - Návod

- Výhodou je, že blok *vyšli zprávu _* vyšle danou zprávu do všech postav, tzn. na jednu událost tak může reagovat hned několik postav najednou. Toho využijeme i v této hře

– když **Auto** narazí do **Nepřítele**, tak **Nepřítel** zmizí, **Auto** bude více porouchané a **Srdce** zčerná.

- Vytvořte novou zprávu „náraz“, která se vyšle po dotyku **Nepřítele** s **Autem**. Kód by mohl vypadat ve scénáři **Nepřítele** kupříkladu takto:

Obrázek 118 - Ukázka kódu

Poznámka: Nabízí se otázka, proč mít tento kód v **Nepříteli**, když by teoreticky mohl „náraz“ vycházet i z **Auta**. Vysílat zprávu „náraz“ z **Auta** by se ale později ukázalo jako nevhodné řešení, protože v **Nepříteli** i ve stromech (**Strom1**, **Strom2**) bychom vytvořili sekvenci, která by po obdržení zprávy „náraz“ postavu skryla, což by vyústilo v komplikaci, jelikož při nárazu by se skryl **Nepřítel** i oba stromy, přestože došlo k nárazu pouze do **Nepřítele**. Řešením by bylo přidat do **Auta** podmínku pro každou postavu zvlášť a pro každou postavu vyslat jinou zprávu, aby došlo k jejich odlišení.

- Zeptejte se žáků, zda je nutné někam vkládat blok *ukaž se*, když se po nárazu **Nepřítel** skryje. Žáci by měli dojít k tomu, že to není třeba, protože blok *ukaž se* je již v sekvenci pohybu **Nepřítele**, takže **Nepřítel** se ukáže vždy poté, co skočí na svou startovní pozici (ať už předtím narazil, či projel).
- Jelikož hráč může ztratit život, pokud narazí do **Nepřítele**, **Stromu1**, nebo **Stromu2**, vložte tutéž sekvenci, která vyšle po dotyku s **Autem** zprávu „náraz“, i do postav **Strom1** a **Strom2**.
- Nyní je třeba naprogramovat samotný proces porážky. Po nárazu do jedné z překážek má **Auto** změnit svůj kostým na více rozbité vozidlo. V postavě hráče (**Auto**) se nachází čtyři kostýmy (auto, auto2, auto3, auto4), přičemž první kostým odpovídá vozidlu v normálním stavu a čtvrtý obsahuje úplně rozbitý automobil.

Obrázek 119 - Ukázka kostýmů postavy Auto

- Přidejte do **Auta** bloky *po obdržení zprávy náraz* a *další kostým*. Při každém nárazu **Auta** do překážky tak dojde ke změně jeho kostýmu a bude vypadat více rozbité. Vždy po zapnutí hry by ale **Auto** mělo být znovu v normálním stavu, vhodné by bylo **Auto** pokaždé vrátit na stejnou počáteční pozici. Nechte žáky samostatně pracovat. Po *po kliknutí na zelenou vlajku* přidejte blok *změň kostým na auto*. Pro návrat **Auta** na stejnou počáteční pozici použijte *skoč na x: _ y: _*, např. na pozici dole uprostřed (x: 0 y: -130).
- Dále je po nárazu třeba, aby se hráči ubral jeden život, tzn. aby srdce postupně zčernávala. Hráč začíná se třemi životy, při prvním nárazu má zčernat **Srdce**, při druhém **Srdce2** a jako poslední **Srdce3**, které svým zčernáním ukončí celou hru. Jedním z možných a jednoduchých řešení je vytvořit si novou proměnnou `Životy`, kterou nastavíme po zapnutí hry na hodnotu 3. S každým nárazem by se pak měla proměnná snížit o 1, což naprogramujeme v sekvenci, která se spustí po *po obdržení zprávy náraz*. Jelikož se nám počet aktuálních životů bude zobrazovat pomocí srdíček, proměnnou `Životy` nepotřebujeme zobrazovat na scéně, a proto ji skryjte.
- Celý kód přidaný do **Auta** by mohl vypadat následovně:

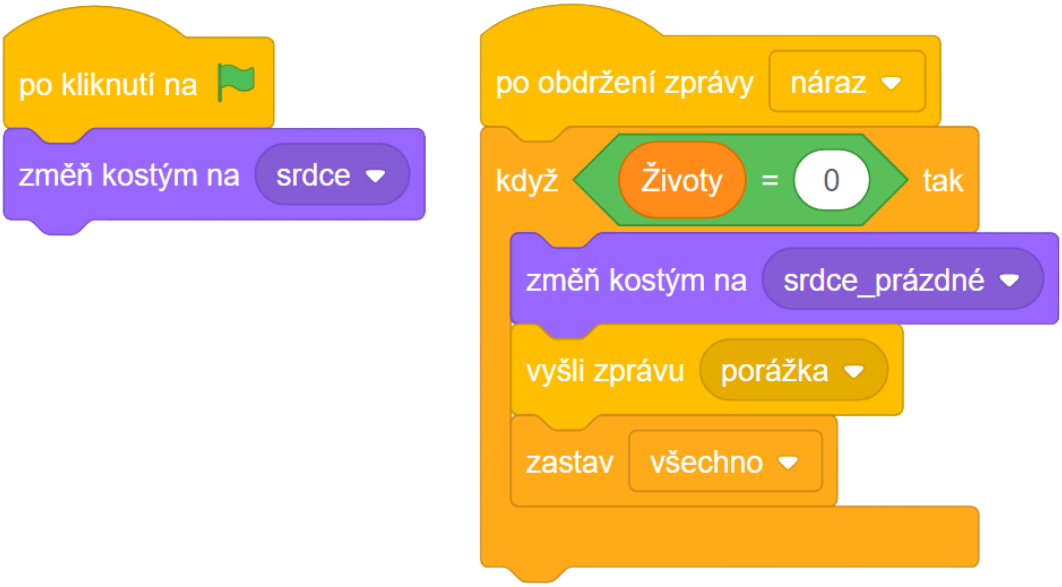
Obrázek 120 - Ukázka kódu

- V srdcích (postavy **`Srdce`**, **`Srdce2`**, **`Srdce3`**) bude až na menší detaily pokaždé stejný scénář, proto vytvořte jeden pro **`Srdce`**, který následně zkopírujete do zbylých srdcích a v nich již jen upravíte rozdíly.
- Srdce obsahují vždy dva kostýmy – červené srdce (kostým srdce) a černé srdce (kostým srdce_prázdné). Po zapnutí hry by všechna srdce měla být resetována a mít kostým srdce, tzn. červené. Posléze při obdržení zprávy „náraz“ si každé srdce zkontroluje, zda proměnná `Životy` klesla na požadovaný počet, kdy má dané srdce zčernat. Kupříkladu **`Srdce`**, které má zčernat po prvním nárazu, změní svůj kostým na srdce_prázdné při hodnotě proměnné `Životy` 2, viz níže:

Obrázek 121 - Ukázka kódu

- Scénář ze **`Srdce`** zkopírujte i do **`Srdce2`** a **`Srdce3`**. **`Srdce2`** má zčernat po druhém nárazu, tzn. když `Životy` budou 1, je proto nutné dle toho upravit podmínku. Poslední **`Srdce3`** zčerná při 0 `Životech,` a navíc ukončí hru pomocí *zastav všechno*.
- Porážku hráči oznámíme kromě zastavení hry i nápisem „Game Over“ přes celou scénu. Tentokrát však nenapíšeme text do pozadí, ale využijeme postavu **`Prohra`**, jejíž kostým se skládá pouze z textu oznamující konec hry. Jelikož se má text zobrazit jen při porážce, je třeba si tento okamžik hlídat. Možností je například vytvořit si

novou zprávu „porážka“, kterou nám před ukončením hry vyšle **`Srdce3`**. Celý kód v **`Srdci3`** by pak mohl vypadat takto:



Obrázek 122 - Ukázka kódu

- Scénář postavy **`Prohra`** se bude skládat ze dvou sekvencí. Ukáže se jen po *po obdržení zprávy porážka*, jinak po každém zapnutí hry se zase skryje, aby se text nezobrazoval stále.

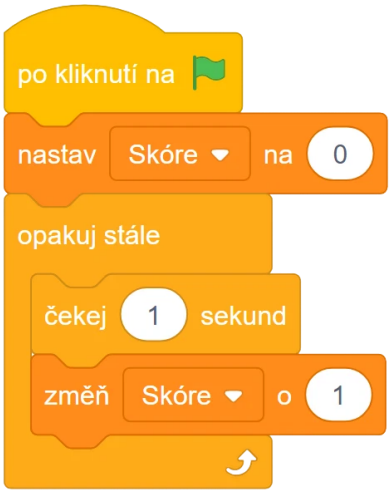
Obrázek 123 - Ukázka kódu

Poznámka: Okamžik porážky lze v **`Prohře`** samozřejmě řešit i bez použití zpráv, např. můžeme čekat, dokud `Životy` nedosáhnou 0, viz kód níže:

Obrázek 124 - Ukázka kódu

7. Vytvoření skóre

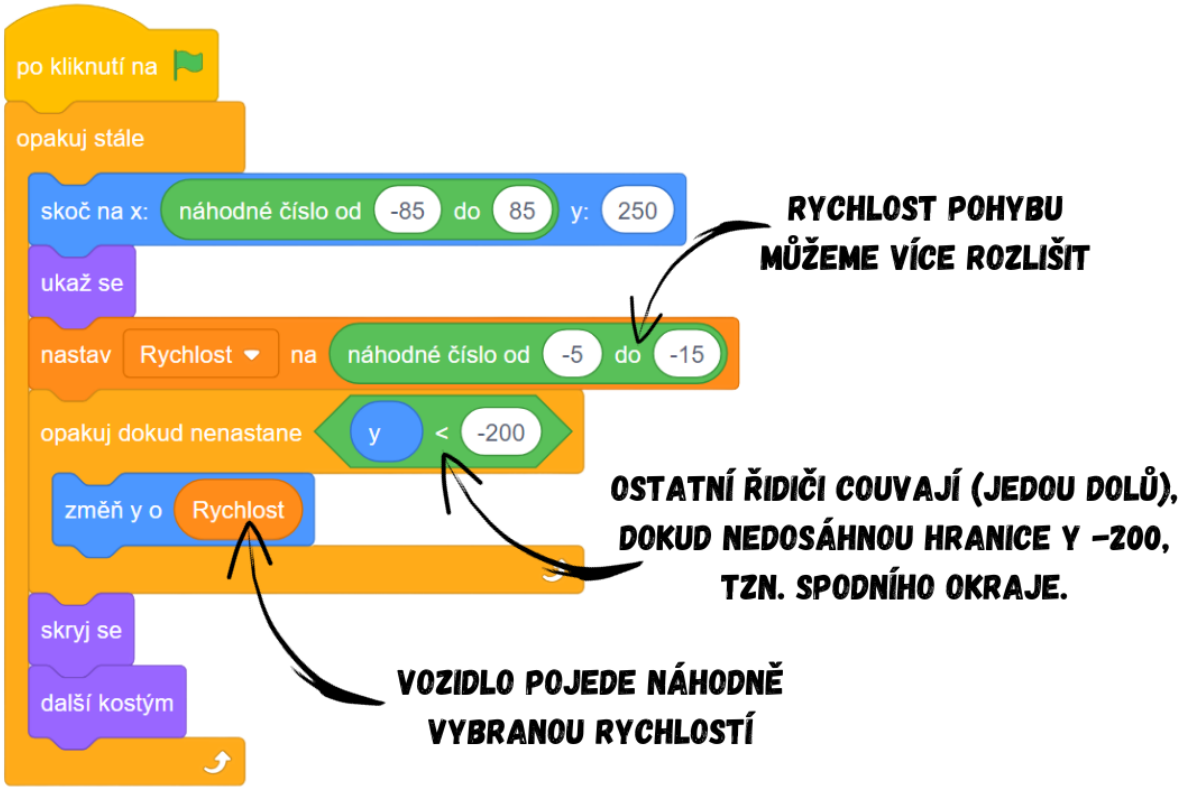
   - Během hry získá hráč za každou sekundu jeden bod.
   - Vytvořte novou proměnnou `Skóre` a ve scénáři pozadí ji naprogramujte. Hodnotu `Skóre` je při zapnutí hry nutné resetovat na 0.



Obrázek 125 - Ukázka kódu

8. Volitelné: Vylepšené nastavení rychlosti ostatních aut

   - Nevýhodou sekvence pohybu v **`Nepřiteli`** je nemožnost více rozlišit rychlost řidičů. V tomto případě nám padne „rychlost“ buď 1 sekunda, 2, nebo 3. Můžeme rozmezí rozšířit na další sekundy (4, 5…), ale po vyzkoušení zjistíme, že posléze už jsou ostatní auta příliš pomalá (respektive v rámci iluze jsou ta auta naopak hodně rychlá a my je proto dojíždíme velmi pomalu). Pro získání lepší kontroly nad rychlostí aut můžeme kód upravit a použít místo klouzání pohyb pomocí cyklu *opakuj dokud nenastane _* a *změň y o _*. Rychlost můžeme určit losem a uložit si ji do nové proměnné (např. `Rychlost`), kterou vložíme do *změň y o _*. Každý řidič tedy pojede náhodně určenou rychlostí:



Obrázek 126 - Ukázka kódu

Poznámka: Pro porovnání hodnoty předpřipravené proměnné y použijte operátor _ < _, v našem případě *y < -200*, což odpovídá spodnímu okraji. Když **`Nepřítel`** tedy odcouvá až úplně dolů, cyklus se přeruší, **`Nepřítel`** se skryje, změní kostým na další automobil v pořadí a celá situace se opakuje díky vnějšímu cyklu *opakuj stále*.

Vývoj: Pozadí projektu Dálnice původně obsahovalo přerušovanou čáru, ta se však nehýbala, a tak narušovala iluzi pohybujícího se pozadí, bylo proto přistoupeno na plnou čáru. Po konzultaci s kolegy a vedoucím práce byl rozšířen popis iluze pohybu při vytváření **`Nepřítele`** pro lepší pochopení toho, co **`Nepřítel`** skutečně dělá a jak to vypadá v iluzi (**`Nepřítel`** couvající pomaleji v iluzi odpovídá tomu, že naopak jede hodně rychle a my ho pouze pomalu dojíždíme atp.). Přidány byly výukové poznámky do některých kroků, aby bylo jasné, kdy mají žáci pracovat samostatně (resp. na co se mají pokusit přijít sami) a kdy se jedná o výklad vyučujícího. Lekce před úpravami po ověření obsahovala i rozbor technického nedostatku Scratche, kdy jsou různé sekvence spouštěny v různou dobu, přestože by měly být spuštěny ve stejném okamžiku. Tento nedostatek mohl vést k nefunkčnosti hry, na což bylo spolu s možným řešením upozorněno v poznámce. Nakonec však bylo přistoupeno na jiné řešení, které funguje nezávisle na tomto technickém nedostatku, tudíž ho není nutné řešit, viz úpravy po ověření ve výuce.

Průběh ověření: Projekt Dálnice je již vcelku komplexní hrou, se kterou se žáci mohou setkat i běžně ve svém životě (jeden žák ukazoval ostatním, že má podobnou hru na mobilním telefonu). S touto komplexností souvisí i dlouhá délka tvorby tohoto projektu, proto jsem zvolil rychlé tempo výkladu. Obě skupiny dokázaly samostatně naprogramovat pohyb postavy hráče (**Auto**), velmi dobře též reagovaly na mé dotazy, jaké bloky použít pro pohyb stromů. Prostřednictvím doplňujících otázek jsem zjistil, že si všichni žáci již plně osvojili znalost souřadnic a jsou schopni s nimi samostatně pracovat. Zdálo se, že žákům nedělala problém ani práce s kostýmy. Vypadalo to, že obě skupiny plně pochopily princip fungování zpráv a jejich přínos. Někteří žáci navrhovali během tvorby i různé úpravy do hry (**Auto** nesmí vyjet mimo silnici), k jejich realizaci ovšem kvůli nedostatku času nakonec nedošlo. Nedostatek času se projevil zejména ve druhé skupině, kde se projekt nestihlo dokončit v 90 minutách a bylo nutné vymezený časový blok o chvíli přetáhnout. Žáci z lekce odcházeli nadšení, zdálo se tedy, že tvorba hry je bavila.

Úpravy po ověření ve výuce: Během ověření byla v sekvenci pohybu **Nepřítele** nalezena menší chyba, kdy byla losovaná rychlost (*náhodné číslo od -5 do -15*) vložena přímo do bloku *změň y o _*, což způsobilo, že s každým projitím cyklu byla vylosovaná jiná rychlost a vozidlo se kvůli neustále se měnící rychlosti pohybovalo lehce sekaně. Situace byla jednoduše vyřešena pomocí nové proměnné `Rychlost`, do níž se uloží vylosovaná hodnota ještě před spuštěním cyklu. V metodice byl na základě zkušeností z ověření rozšířen výklad pro vysvětlení fungování zpráv. Sekvence měnící kostým srdcí se nakonec spouští po obdržení zprávy „náraz“ z důvodu lepšího pochopení ze strany žáků (pro žáky je jasnější, že srdce reaguje na zprávu „náraz“, než čekáním na počet `Životů`) a z důvodu eliminace možné nefunkčnosti hry kvůli technickému nedostatku Scratche v podobě spouštění paralelních sekvencí v různý čas. Hlavním problémem celé lekce byla dlouhá délka tvorby projektu, která i při rychlém tempu dosahovala přes 90 minut. Při takové rychlosti postupu ovšem byla mnohem větší šance, že se nějaký žák začne v kódu ztrácet. Uvažovalo se proto nad mnoha možnostmi, jak projekt zkrátit. Z návrhu vložit hotovou sekvenci pro nastavení `Skóre`, či připravený scénář **Prohry** již do startovní verze BIT_8_Dálnice_Start nakonec sešlo, protože ušetřený čas by byl marginální. Další možností bylo odsunout zobrazování životů pomocí srdcí na závěr projektu jako volitelný krok, hráč by se pak musel řídit pouze kostýmem **Auta**. Jelikož si ale žáci zobrazování srdcí hodně žádali, a navíc se jednalo o integrální součást celého projektu (postavy **Srdce**, **Srdce2** a **Srdce3** se nacházejí ve startovní verzi projektu a jsou to postavy vhodné k vyzkoušení si, jak funguje vysílání zpráv a reakce na ně v praxi), byl i tento návrh odmítnut. Nakonec bylo na základě konzultace s vedoucím práce dohodnuto, že pro ušetření času bude pohyb **Nepřítele** bude fungovat na stejném principu jako pohyb stromů, tedy použitím bloku *klouzej na x: _ y: _*. Vylepšená varianta s použitím proměnné `Rychlost` a *změň y o _* byla pro žáky poměrně těžká na pochopení, byla proto vyjmuta a přidána na závěr jako separátní poslední krok, který je pouze volitelný.

9.9 Lekce 9 – Střílečka

Délka: 80-90 minut

O projektu: Hráč hraje za modré kolečko uprostřed scény, které se otáčí za kurzorem myši. Kolečko vystřeluje náboje, pomocí nichž ničí nepřátele, kteří se na kolečko sbíhají ze všech stran. Cílem je držet nepřátele co nejdéle od kolečka a získat tak nejvyšší skóre.

Obrázek 127 - Ukázka ze hry

Cíl lekce: Hlavním cílem lekce je seznámit žáky s principem klonování postav, tzn. naučit je, jak klonování funguje a jaké jsou jeho výhody, jak klonovat postavy a jak klony následně programovat k dalšímu využití. Žáci si v projektu též zopakují zejména dosavadní získané znalosti ohledně cyklů a podmínek, rozšíří si povědomí o operátorech.

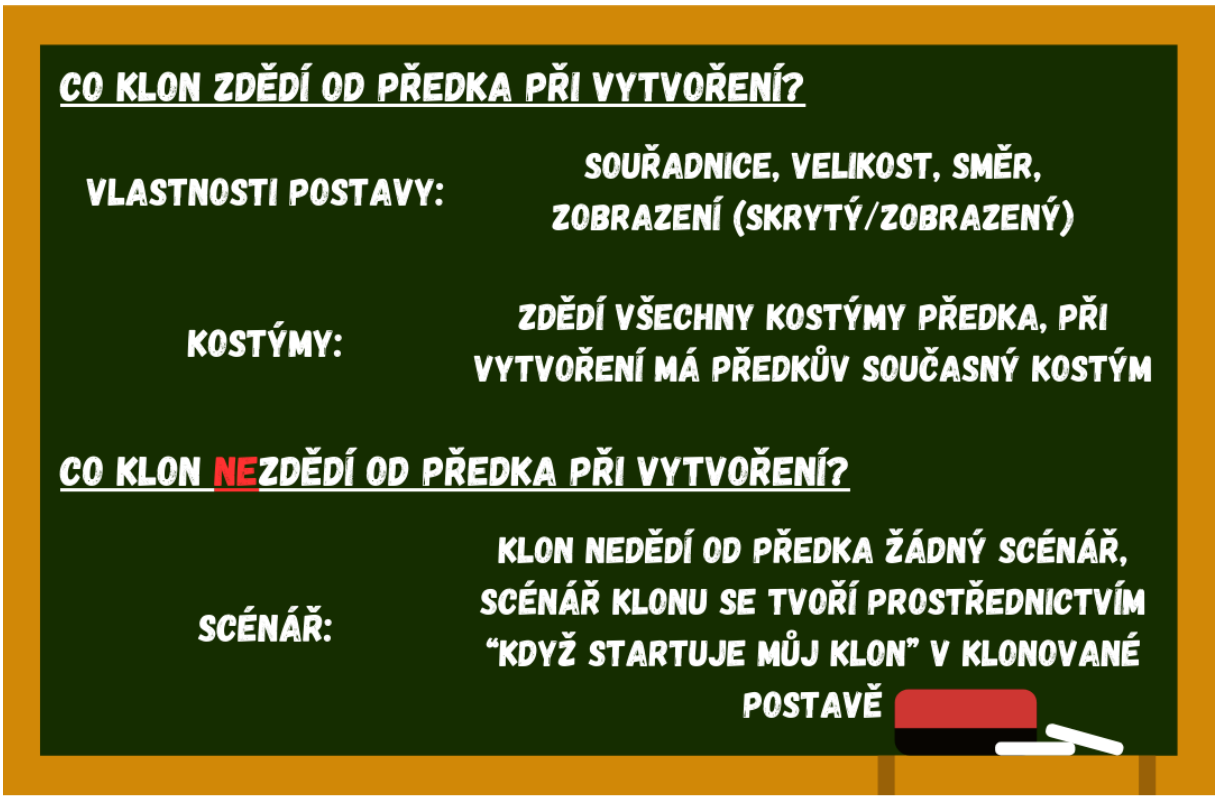
Nově naučené bloky: *nastav směr k _*; *když startuje můj klon*; *klonuj _*; *zruš tento klon*; *_ a _*; *_ nebo _*

Náplň lekce:

1. Propedeutická aktivita na klonování
    - V tomto projektu budou žáci pracovat s klony, proto je nutné je nejdříve s principem klonování obeznámit.
    - Poskytněte žákům projekt BIT_Aktivita_Klonování_Start, který si otevřou. Nechte žáky, ať zjistí, co scénář v postavě **`Hvězda`** dělá. Po spuštění projektu postava **`Hvězda`** pouze nastaví svůj kostým, a poté každou sekundu skáče na náhodnou pozici ve „výšce“ 100. Projekt je totiž nedodělaný a úkolem je splnit zadání, které je napsané v poznámce ve scénáři. Ukažte žákům, jak by měl vypadat výsledek, viz projekt BIT_Aktivita_Klonování. Žáci si pravděpodobně při prozkoumávání kódu všimli neznámého bloku *když startuje můj klon*, přičemž sekvence pod ním se zatím

nespouští. Vysvětlete žákům, že bloky navázané na *když startuje můj klon* se spustí poté, co bude vytvořen klon dané postavy.

- Klon je ve Sratchi vlastně kopie klonované postavy, ovšem přebírá od ní pouze některé věci (např. kostýmy, lokální proměnné, aktuální pozici, velikost, směr, skrytí…). Nejedná se tedy o kopii 1:1, **klon kupříkladu vůbec nepřebírá od svého předka (parent, rodič) scénář**. Co mají klony dělat tak programujeme právě v sekvenci začínající blokem *když startuje můj klon*. Pokud se podíváme na sekvenci v BIT_Aktivita_Klonování_Start, zjistíme, že po vytvoření klonu má klon změnit svůj kostým, aby byl odlišitelný od předka.



Obrázek 128 - Model školní tabule s možným výkladem

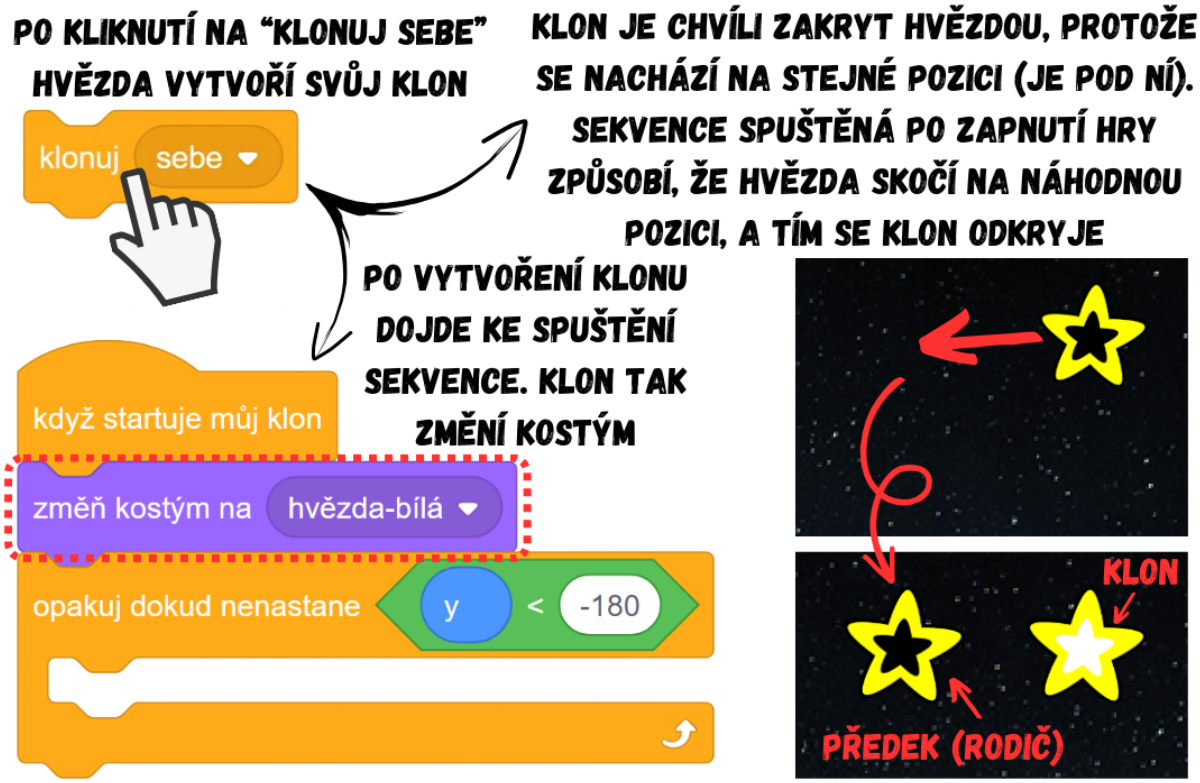
Poznámka: S žáky můžete postupně na tabuli ve třídě doplňovat, co klon zdědí a co ne, a to na základě jejich zjištění v průběhu tvorby. Diskutujte pouze nad zásadními prvky (viz obrázek výše), které by si žáci měli zapamatovat. Zabývat se kupříkladu tím, že klon dědí i zvuky z postavy předka, je v tuto chvíli nadbytečné.

- Pro vytvoření klonu použijte blok *klonuj* _z kategorie *Ovládání*, přičemž v našem případě bude postava **`Hvězda`** klonovat sama sebe (*klonuj sebe*).

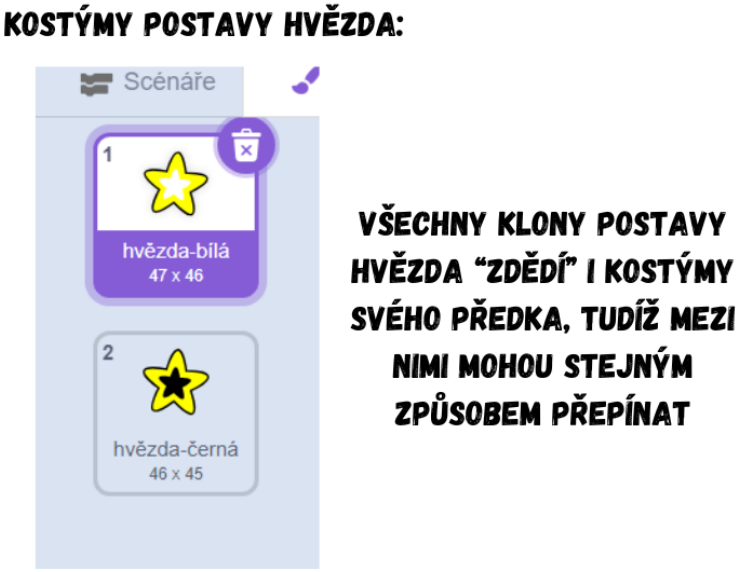
Obrázek 129 - Ukázka kódu

Poznámka: Pokud bychom měli ve hře více postav, výběr v bloku *klonuj _* by obsahoval názvy všech z nich. Postava **Hvězda** může tvořit i klony jiných postav a naopak. Chování klonů je však třeba vždy programovat v klonované postavě prostřednictvím *když startuje můj klon*., tzn. postava Hvězda může tvořit klony jiných postav, ale chování těchto klonů je nutné nastavit v klonované postavě.

- Nechte žáky vyzkoušet princip fungování bloku tím, že kliknutím na zelenou vlajku spustí připravený scénář a následně budou spouštět i blok *klonuj sebe* klikáním myši. Klon se vždy vytvoří na souřadnicích předka, je tak zakryt, dokud předek (hvězda s černým středem) neskočí na další náhodnou pozici. Na původní pozici by nyní měl být klon **Hvězdy**, akorát na rozdíl od svého předka má jiný kostým, protože došlo k vytvoření klonu a tím i ke spuštění sekvence následující po *když startuje můj klon*, která obsahuje blok *změň kostým na hvězda-bílá* (hvězda s bílým středem, tedy kostým klonu).

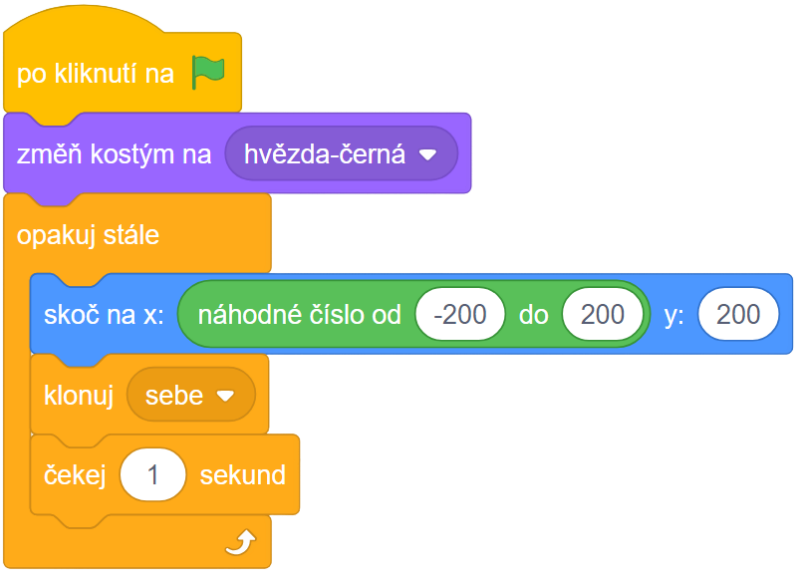


Obrázek 130 - Princip fungování sekvence



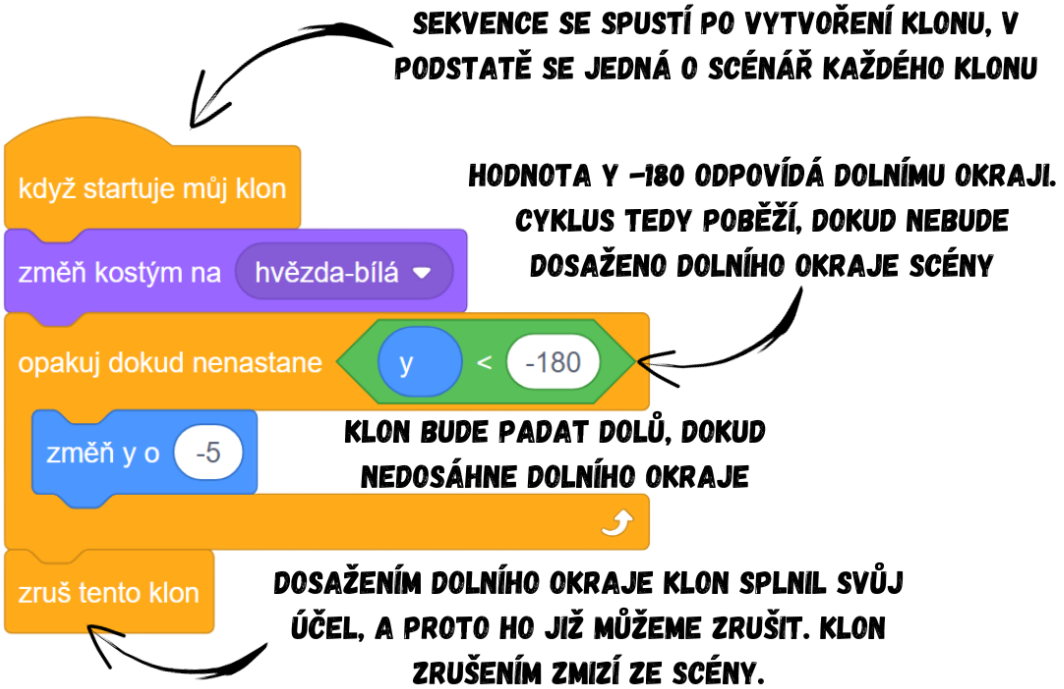
Obrázek 131 - Ukázka kostýmů postavy Hvězda

- Pokud vložíme blok *klonuj sebe* do cyklu *opakuj stále*, jenž se spouští po zapnutí hry, Hvězda každou sekundu skočí na náhodnou pozici ve „výšce" 100 a vytvoří svůj klon.



Obrázek 132 - Ukázka kódu

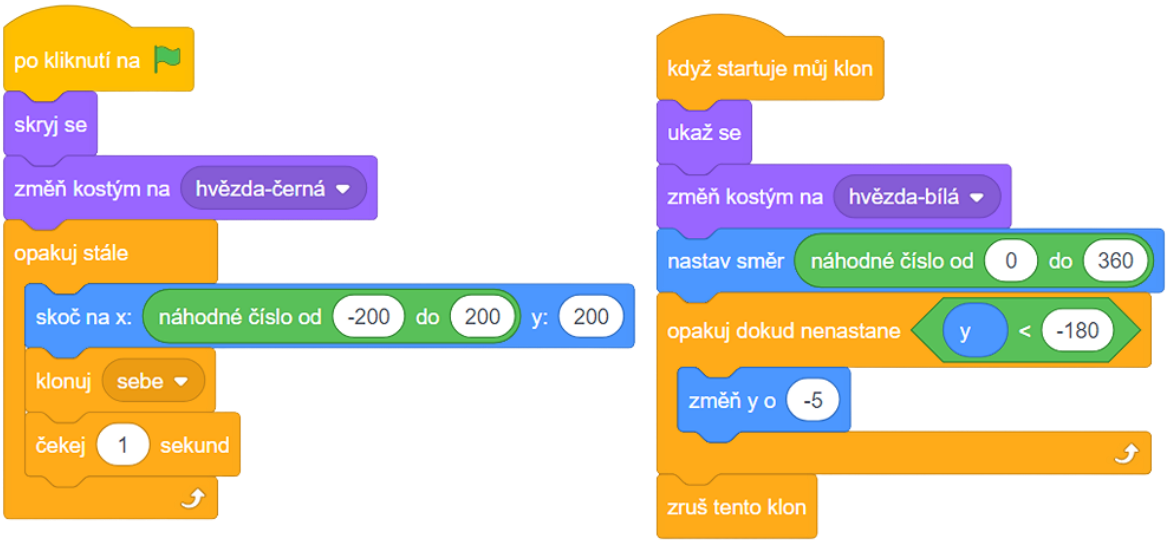
- Problémem ale je, že klony nyní kromě změny kostýmu nic nedělají, cílem je však naprogramovat je tak, aby padaly dolů a po dotyku dolního okraje zmizely. Pád dolů jednoduše vytvoříte pomocí *změň y o _* (např. -5). Předpřipravená podmínka v cyklu *opakuj dokud nenastane _* hlídá, zda bylo dosaženo dolního okraje, který odpovídá hodnotě y -180. Po jeho dosažení dojde k ukončení cyklu, klon se dále nebude hýbat. Nechte žáky, aby přišli na to, že v tomto případě by za chvíli při dolním okraji měli nashromážděno velmi mnoho hvězd. Klon po dokončení pohybu splnil svůj účel a nedává tak smysl ho dále uchovávat ve hře. Proto na závěr sekvence přidejte blok *zruš tento klon*, jenž způsobí, že klon přestane existovat a zmizí ze scény.



Obrázek 133 - Ukázka kódu

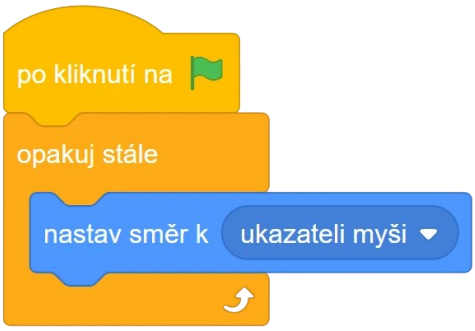
Poznámka: Blok *zruš tento klon* je skutečně zásadní a ideálně by jím měla končit každá sekvence po *když startuje můj klon*. Klonování je poměrně náročný proces, který zatěžuje procesor počítače. V případě, že bychom vytvořili hodně klonů a nikdy je nezrušili, zbytečně bychom mrhali zdroji, a navíc by se časem hra začala sekat. Zdůrazněte proto žákům důležitost rušení klonů a odůvodněte nutnost jejich zrušení.

- Každou sekundu nyní **Hvězda** skočí na náhodnou pozici a „vypustí“ jeden klon. Abychom dosáhli iluze padajících hvězd, bude třeba hvězdy vypouštět z horního okraje. Zeptejte se žáků, na kolik přibližně je potřeba přepsat hodnotu y v bloku *skoč na x: _ y: _* (cca ze 100 na 200). Hvězdy nyní padají úplně shora, ovšem stále je vidět předek, který klony vypouští. Je proto třeba předka (**Hvězdu**) skrýt. Přidejte blok *skryj se* po *po kliknutí na zelenou vlajku*. Jelikož klony přebírají od předka i vlastnosti (ukázat/skrýt se), budou nyní všechny klony také skryté. Přidáním bloku *ukaž se* na začátek sekvence následující po *když startuje klon* problém vyřešíte. Tím je zadání splněno a aktivita dokončena, hvězdy padají na nočním nebi. Celý scénář ve **Hvězdě** by mohl vypadat takto:



Obrázek 134 - Ukázka scénáře postavy Hvězda

2. Úvodní diskuze:
    - Představte žákům zamýšlený projekt a ukažte jim, jak by měl výsledek vypadat, viz BIT_9_Střílečka.
    - Diskutujte s žáky o tom, s čím už si jsou schopny poradit díky svým znalostem z předchozích lekcí či aktivity a s čím si rady neví, co je tedy naopak nové. Zeptejte se žáků, zda v ukázce viděli klony (střely a nepřátelé).
3. Stáhněte si počáteční projekt:
    - Použijte připravený projekt BIT_9_Střílečka_Start, který obsahuje všechny postavy a pozadí potřebné pro hru. Žákům na začátku hodiny projekt nasdílejte, popř. dejte ke stažení.
4. Naprogramování postavy hráče
    - Ve hře hrajete za postavu **`Hráč`**, která se má otáčet za kurzorem myši a vystřelovat náboje.
    - Vysvětlete žákům, že se postava otočí za kurzorem myši, pokud ji nastavíme daný směr. K tomu je vhodný blok *nastav směr k _*, který nastaví směr postavy směrem k jiné postavě, případně ukazateli myši. V tomto případě nastavte směr k ukazateli myši. Aby se **`Hráč`** neustále otáčel za kurzorem, je třeba blok vložit do cyklu *opakuj stále*.

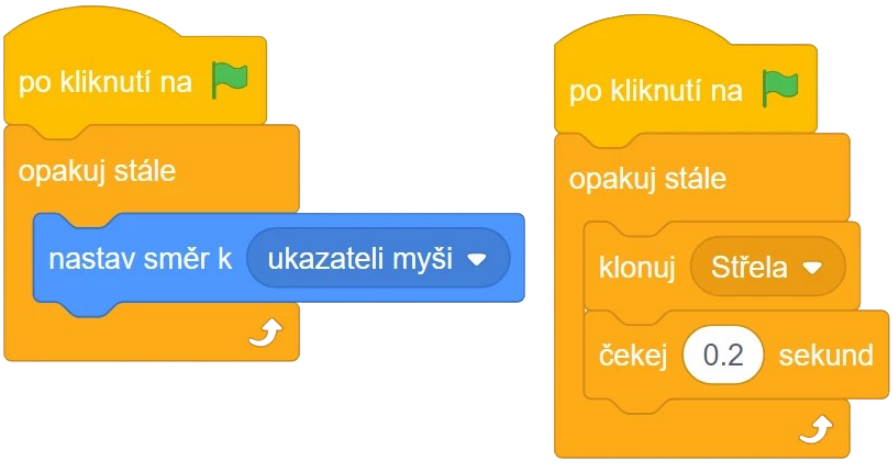


Obrázek 135 - Ukázka kódu

5. Naprogramování střelby

- **Hráč** má vystřelovat **Střelu** každou chvíli, řekněme třeba každé 0,2 sekundy. Při takhle krátkém intervalu ale **Střela** nestačí dokončit svoji trajektorii před tím, než má znovu vyletět z **Hráče**. Vysvětlete žákům, že tedy není možné použít jednu postavu, která by doletěla na okraj, posléze skočila na **Hráče** a opět vyletěla. Samotná **Střela** by to jednoduše nestíhala. Jelikož potřebujeme v podstatě jednu postavu na dvou (či více) místech najednou, je třeba použít klonování. **Hráč** bude **Střelu** klonovat každých 0,2 sekund a klon **Střely** vyletí směrem k ukazateli myši. Na scéně tak najednou může být hned několik **Střel**, a to v podobě jejích klonů.

- Do scénáře **Hráče** přidejte blok *klonuj _* a vyberte postavu **Střela**. Pokládejte žákům otázky, co má vlastně **Střela** dělat a co od ní očekáváme. Společně s žáky převádějte odpovědi na dotazy do kódu. Klony **Střely** mají být vytvářeny neustále, vložte proto *klonuj Střela* do *opakuj stále*. Do cyklu ještě přidejte blok *čekej 0,2 sekund* pro stanovení intervalu, za který se má vytvořit další klon **Střely**. V případě, že bychom interval nestanovili, ve hře by pak náboje lítaly nepřetržitě jeden za druhým, což by značně zatížilo procesor počítače. V **Hráči** by tak měly být dvě sekvence:



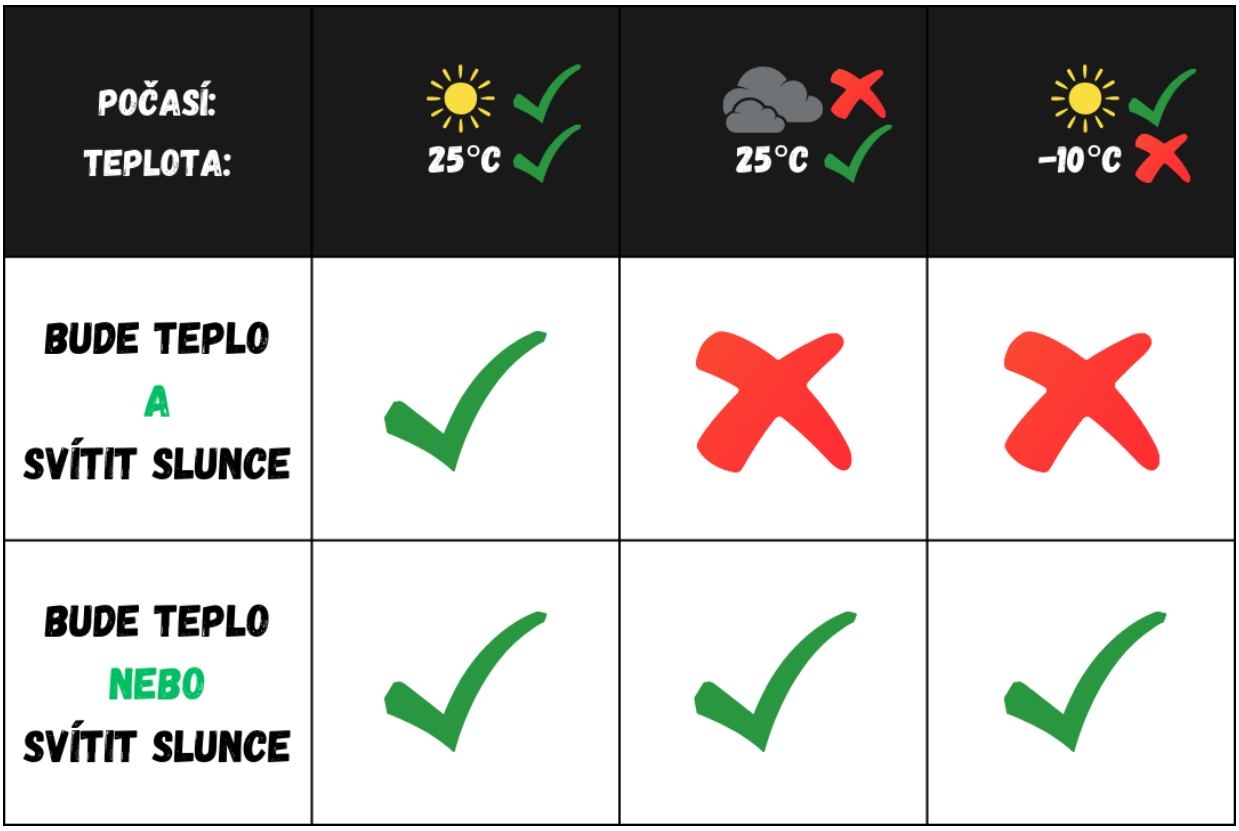
Obrázek 136 - Ukázka kódu

- Nyní je potřeba vytvořit scénář pro klony **Střely**. V jejím scénáři přidejte blok *když startuje můj klon*, jenž pomocí sekvence na něj navázané určuje, co mají klony dělat.

Nechte žáky zformulovat, co potřebujeme do sekvence přidat. Každý náboj má vycházet z **`Hráče`** a letět směrem k ukazateli myši. Přidejte proto pod *když startuje můj klon* bloky *skoč na Hráč* a *nastav směr k ukazateli myši*.

Obrázek 137 - Ukázka kódu

- Pro rozpohybování klonů je nutné přidat ještě cyklus s *dopředu o _ kroků*. Diskutujte s žáky, jaký cyklus je vhodný pro pohyb střel. Střely mají letět vpřed, dokud buď nezasáhnou **`Nepřítele`**, nebo dokud nedoletí na konec scény (hráč se netrefí do **`Nepřítele`**), tzn. dotknou se nějakého okraje. Ideálním pro řešení této situace je tedy cyklus *opakuj dokud nenastane _*, přičemž chceme hlídat hned dvě podmínky:
    1. klon narazil do **`Nepřítele`**
    2. klon doletěl na okraj
- Ukažte žákům, že pro hlídání více podmínek najednou lze použít operátory *_ a _* či *_ nebo _*. Operátor *_ a _* se používá, pokud chceme, aby se něco stalo až po splnění obou dílčích podmínek. Obě podmínky tedy musí být splněny, pokud je splněna jen jedna ze dvou, celkově se pak složená podmínka považuje za nesplněnou. Naproti tomu operátor *_ nebo _* hlídá také dvě dílčí podmínky, ale pro celkové splnění stačí vyplnit jen jednu z nich. Žákům rozdíl mezi operátory můžete vysvětlit na příkladu z reálného života: KDYŽ bude venku teplo A bude svítit slunce TAK půjdeme ven / KDYŽ bude venku teplo NEBO bude svítit slunce TAK půjdeme ven. V prvním případě půjdeme ven pouze za slunného a zároveň teplého dne. V druhém případě bychom šli ven kupříkladu za slunného dne, i kdyby venku bylo -10 stupňů. Rozdíly v tom, kdy se podmínky splní, jsou patrné v následující tabulce:



Obrázek 138 - Tabulka vysvětlující rozdíly mezi operátory A a NEBO

Poznámka: Tuto tabulku můžete žákům promítnout, popř. nakreslit podobnou tabulku na tabuli a společně s žáky do ní doplnit fajfky a křížky dle podmínek.

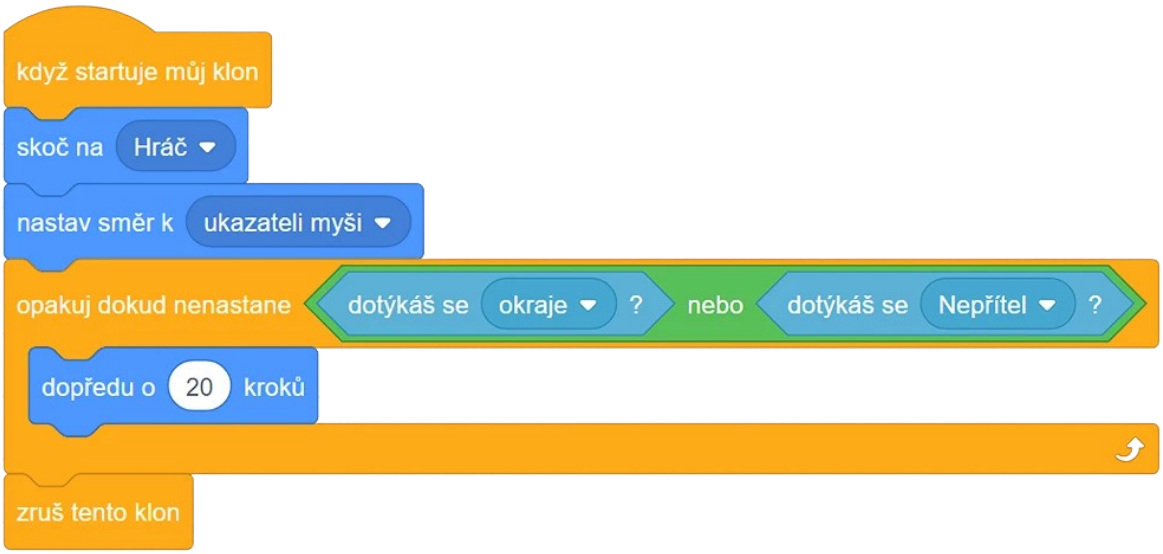
- Z tabulky je jasně patrné, že ke splnění složené podmínky stačí s operátorem _ *nebo* _ splnění alespoň jedné z dílčích podmínek.
- Vraťte se zpět k pohybu střel. Na základě výkladu se žáků zeptejte, zda do cyklu *opakuj dokud nenastane* _ vložíme _ *a* _, či _ *nebo* _. Klony **`Střely`** se mají pohybovat vpřed, dokud se nedotknou **`Nepřítele`** NEBO dokud nenarazí na okraj. Rozdíly mezi možnostmi vysvětlete žákům, např. pomocí obrázku níže:

Obrázek 139 - Vysvětlení rozdílů mezi operátory A a NEBO

- V tomto případě je pro řešení správná druhá varianta s operátorem _ *nebo* _. Pohyb by tedy mohl být vyřešen například takto:

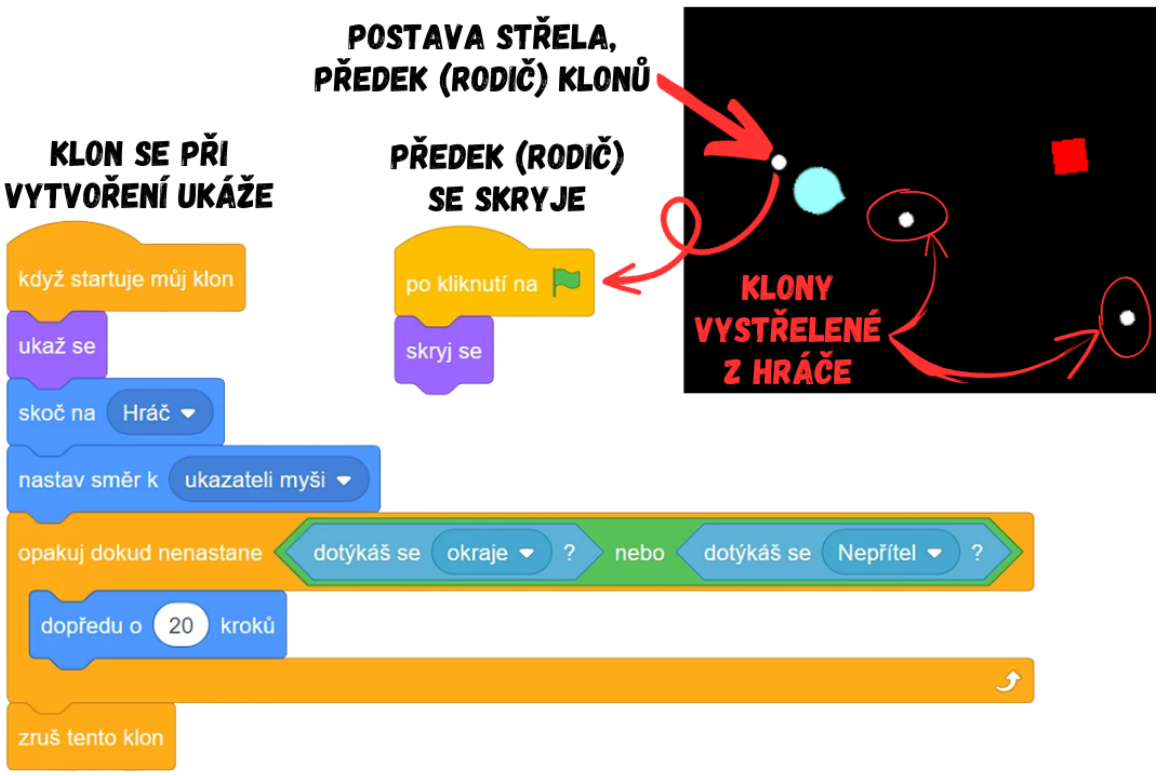
Obrázek 140 - Ukázka kódu

- Poté, co klon dokončí pohyb (narazil do **`Nepřítele`**, nebo doletěl na okraj), je jeho úloha ve hře splněna a je třeba ho zrušit, jinak by na scéně bylo po chvíli příliš mnoho střel a procesor by byl zbytečně zatížen. Přidejte na konec sekvence blok *zruš tento klon*. Zkuste nechat žáky, ať nutnost tohoto kroku sami vysvětlí.



Obrázek 141 - Ukázka kódu

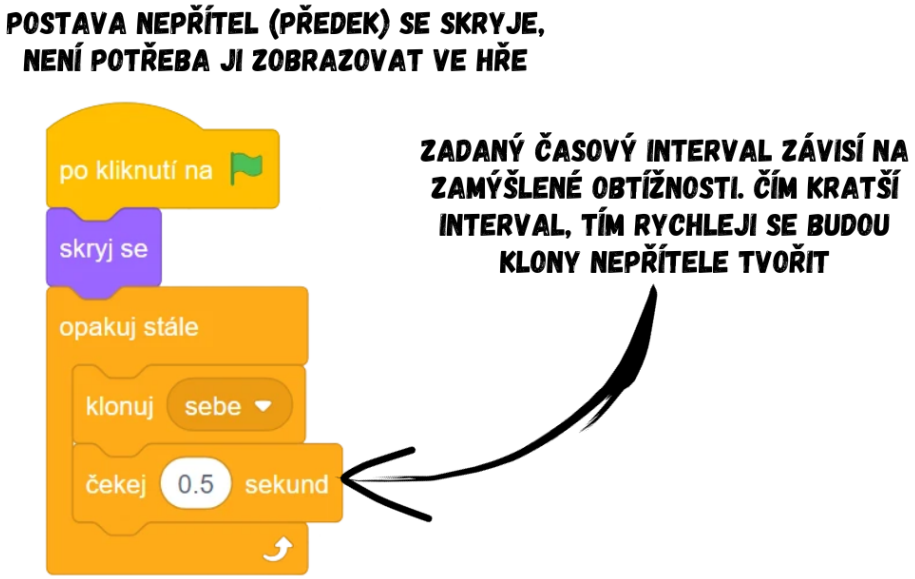
- Po zapnutí hry nyní z **`Hráče`** vystřelují klony postavy **`Střela`** a po nárazu na okraj zmizí, ovšem stále je ve scéně vidět předek (rodič) klonů, totiž samotná postava **`Střela`**, kterou používáme jen jako předka pro tvorbu klonů. Jelikož **`Střela`** nemá žádný jiný užitek (prakticky nyní jen stojí na místě a tvoří klony), schovejte ji pomocí bloku *skryj se*. Nechte žáky samostatně dotvořit postavu **`Střela`** na základě jejich znalostí získaných v propedeutické aktivitě. Pokud by si žáci nevěděli rady, připomeňte jim, že klony od předka vlastnost zobrazení (skrýt/ukázat se) dědí, proto klony po skrytí předka nejsou vidět. Zeptejte se žáků, kam je nutné přidat blok *ukaž se*. Přidejte ho na začátek sekvence po *když startuje můj klon*.



Obrázek 142 - Princip fungování sekvence

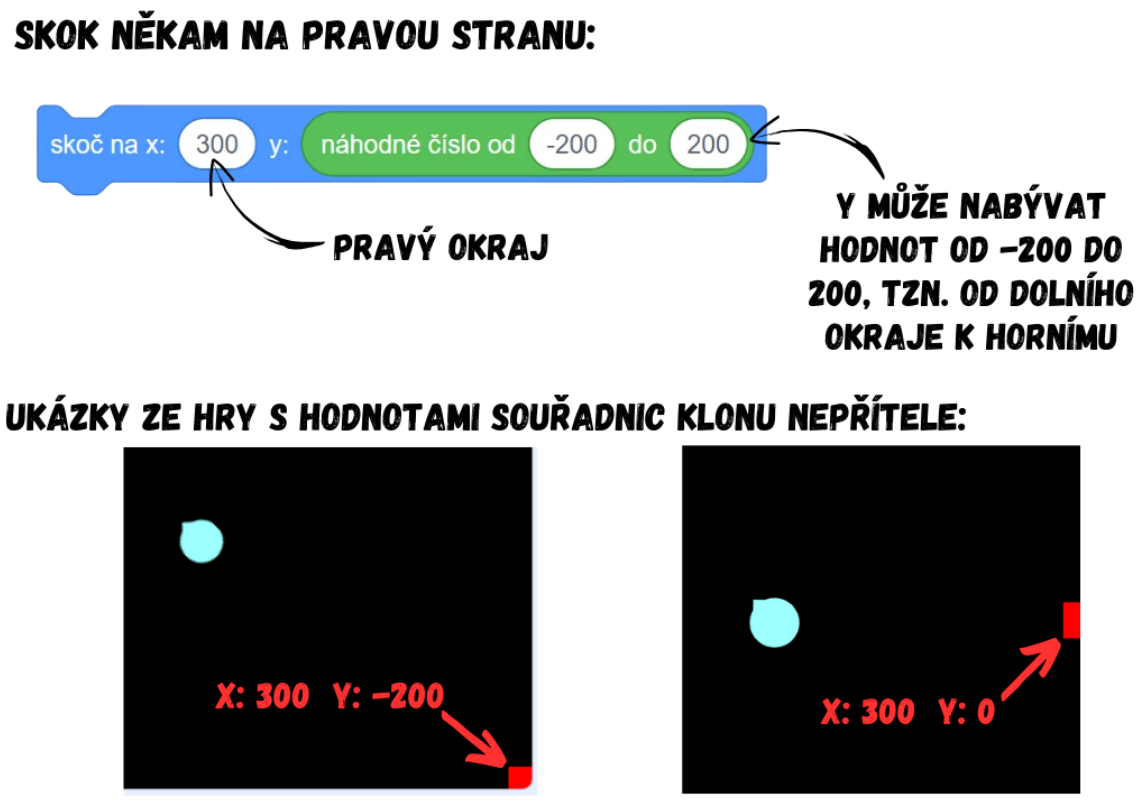
6. Vytvoření nepřátel

- Nechte žáky zformulovat, co má **`Nepřítel`** dělat. Nepřátelé mají pomalu jít směrem k **`Hráči`**, pokud se ho dotknou, tak nastane konec hry. Zeptejte se, zda bude třeba použít klonování. Jelikož se v jednu chvíli ve hře nachází hned několik pohybujících se nepřátel najednou, je nutné ho využít.
- Při programování klonování budeme postupovat podobně jako u střel. Pokládejte proto žákům návodné otázky a nechte je skládat kód. Nejdříve ve scénáři **`Nepřítele`** přidejte cyklus, který za určitý časový interval (1 sekunda, 0,5 sekundy, záleží na zamýšlené obtížnosti hry) vytvoří nový klon **`Nepřítele`**. Podobně jako u **`Střely`**, i zde předka klonů skryjeme, protože ho ve hře vidět nepotřebujeme, všichni pohybující se nepřátelé budou klony.



Obrázek 143 - Ukázka kódu

- Nyní je třeba nastavit, co mají dělat vytvořené klony. Nejdříve je vhodné naprogramovat počáteční pozici klonů. Zeptejte se žáků, odkud mají nepřátelé chodit. Klony **`Nepřítele`** mají vycházet náhodně ze všech čtyř stran, a to vždy od kraje. Vysvětlete žákům, že chceme vlastně vytvořit *skoč na náhodná pozice*, akorát jen při okrajích. Jelikož *skoč na náhodná pozice* využívá celé scény (**`Nepřítel`** by tak mohl skočit i doprostřed scény), je nutné udělat náhodný skok pro každý okraj (pro každou stranu) zvlášť. Pro určení pozice použijte blok *skoč na x: _ y: _* a místo jedné souřadnice vložte dovnitř *náhodné číslo od _ do _*, který zajistí, že klon pokaždé vyjde odjinud (pro pravý okraj bychom např. nastavili x na velmi vysoké číslo a y by bylo náhodné).



Obrázek 144 - Princip fungování bloku

- Nechte žáky dotvořit bloky *skoč na x: _ y: _* tak, aby měli celkově čtyři bloky *skoč na x: _ y: _*, tzn. pro každou stranu jeden blok.

Obrázek 145 - Princip fungování bloků

Poznámka: V případě, že žáci stále plně neovládají souřadnice, je důležité připomínat, že souřadnice x ovlivňuje umístění vlevo/vpravo a y nahoru/dolu. Pomůckou pro žáky může být zapamatování si, že: pravá strana = vysoké x, levá strana = malé x (myšleno vysoké záporné), horní strana = vysoké y, dolní strana = malé y (vysoké záporné).

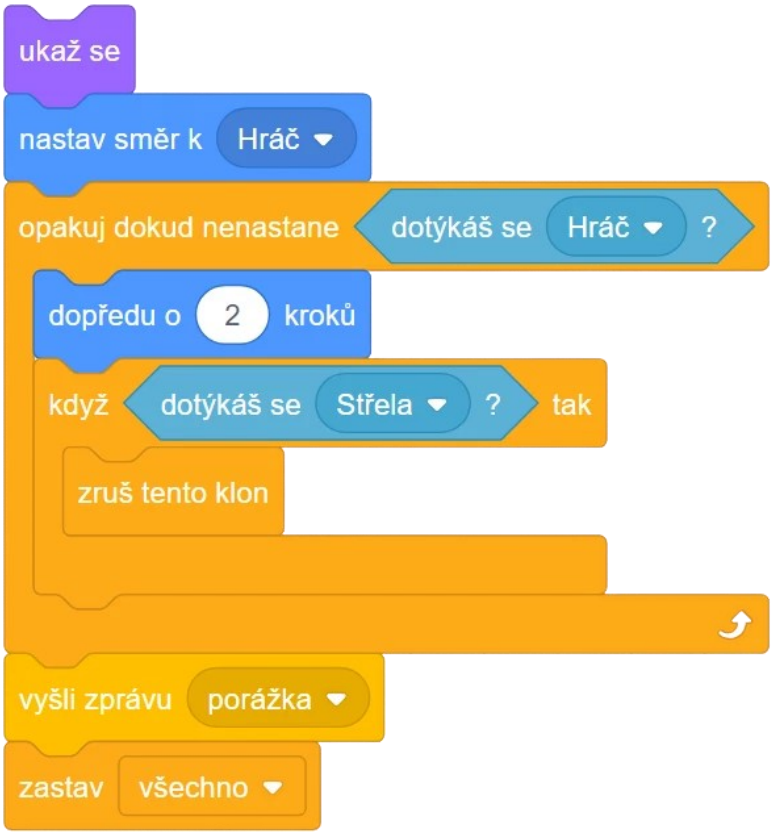
- Když se vytvoří nový klon, má skočit náhodně na jednu z těchto čtyř možností. Vysvětlete žákům, že klon **`Nepřítele`** si v podstatě po svém vytvoření vylosuje, odkud půjde. Diskutuje s žáky, jaké je možné provedení takového návrhu. Situaci lze vyřešit tak, že vytvoříte novou proměnnou `Strana`, kterou nastavíte na *náhodné číslo od 1 do 4*. Podle toho, jaké číslo bylo vylosováno jako hodnota `Strany`, se rozhodne, na kterou stranu klon skočí. Číslo 1 bude např. pravá strana, 2 bude odpovídat levé straně, 3 horní straně a číslo 4 té dolní. Realizace by mohla vypadat následovně:

Obrázek 146 - Ukázka kódu

Poznámka: Žáci mohou být zvyklí dávat blok *když _ tak* do cyklu *opakuj stále*. Vysvětlete žákům, že bloky *když _ tak* zde nemusejí být v cyklu, protože podmínku v tomto případě stačí zkontrolovat jen jednou. Vytvoří se klon, nastaví se mu náhodně `Strana` a posléze se jednou zkontroluje, která ze stran to je, a jen jednou se pak provede příslušný skok na počáteční pozici klonu, odkud se má začít pohybovat k **`Hráči`**. Podmínky zde tak vlastně pouze

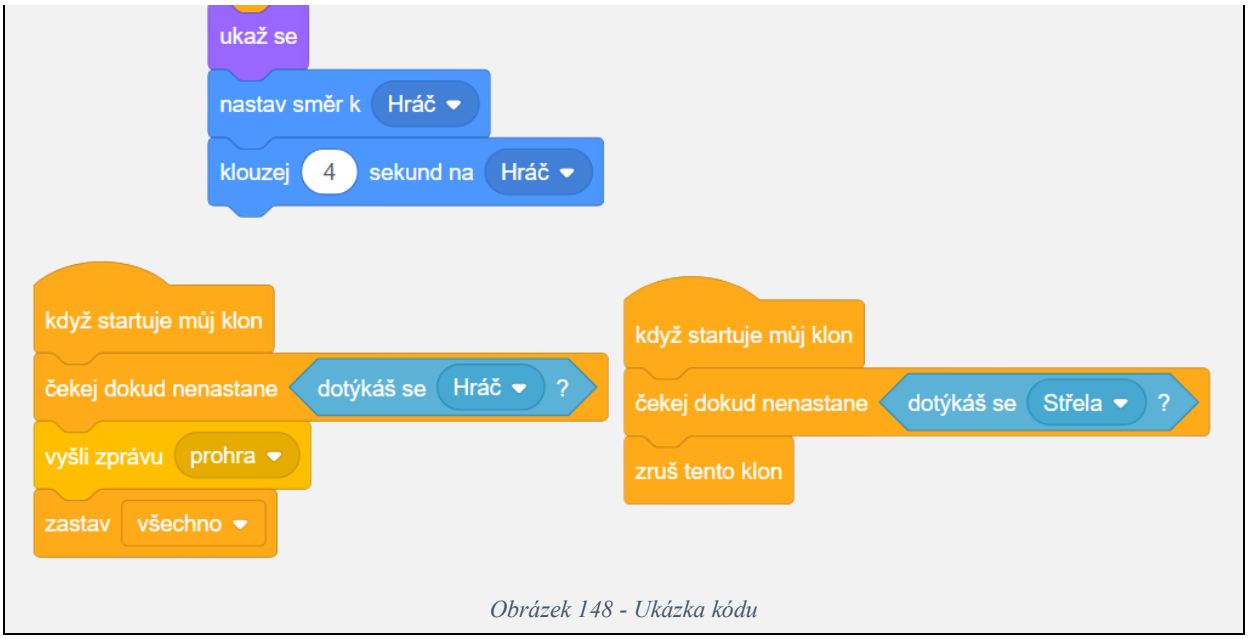
zkontrolují, jak dopadlo losování strany, což každému klonu při jeho startu stačí pouze jednou, není nutné podmínky spouštět neustále.

- Samotné chování klonů **Nepřítele** bude tvořit druhou část sekvence spouštějící se po *když startuje můj klon*. Zeptejte se žáků, jak konkrétně má vypadat pohyb klonu **Nepřítele**. Abychom viděli klony od skrytého předka, musíme je odhalit pomocí *ukaž se*. Následně má klon vyrazit ze své počáteční pozice, kterou řeší první část sekvence, směrem k **Hráči**. Pohyb klon opakuje, dokud se nedotkne **Hráče**. V případě, že je cestou zasažen **Střelou** (respektive jejími klony), klon **Nepřítele** se zruší. Pokud klon dojde až k **Hráči**, cyklus pohybu se přeruší, dojde k zobrazení textu Game Over. Nechte žáky přijít na možné řešení, jak text zobrazit (vyšleme zprávu „porážka“, na kterou následně zareaguje postava **Prohra**). Po dotyku s **Hráčem** se hra zastaví. Druhá část sekvence by tak mohla vypadat takto:



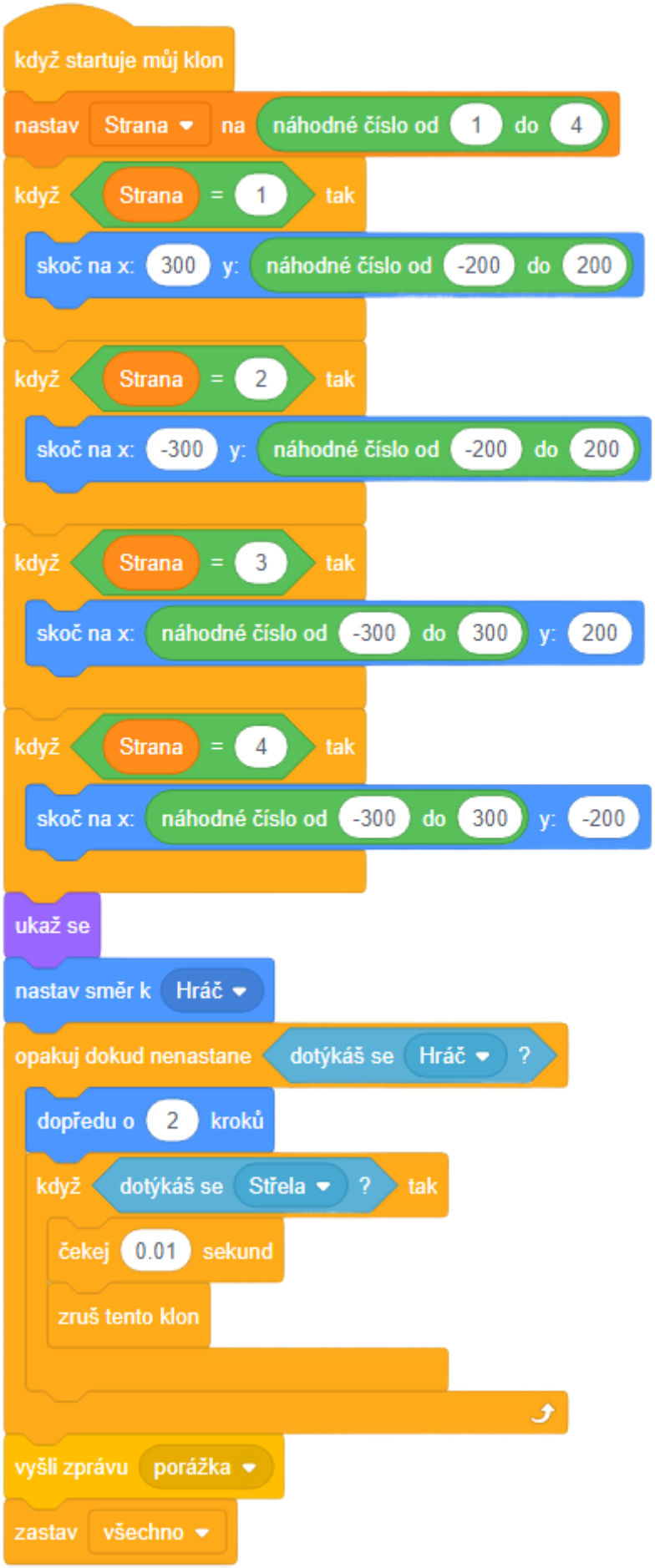
Obrázek 147 - Ukázka kódu

Poznámka: Žáci mohou navrhnout použít pro pohyb **Nepřítele** blok *klouzej _ sekund na Hráč*. Tento blok sice plní požadovanou funkci (pohyb **Nepřítele** směrem k **Hráči**), ale je nevhodný k použití v sekvenci. Po jeho spuštění totiž průběh sekvence „zamrzne“ (podobně jako při *čekej _ sekund*) a čeká se, dokud se blok nevykoná. Použití klouzání by tedy způsobilo, že by **Nepřítel** kvůli „zamrznutí“ kódu vůbec nereagoval na dotyky s **Hráčem** či **Střelou**. V případě použití klouzání pro pohyb by se tak musely vytvořit tři paralelní sekvence *když startuje můj klon*, řešení by mohlo být následovné:



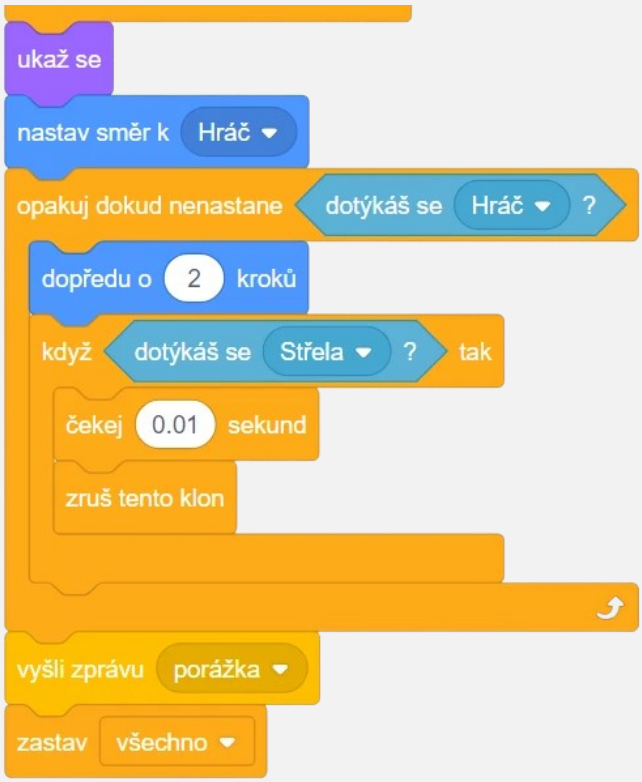
Obrázek 148 - Ukázka kódu

- Celá sekvence ve scénáři **`Nepřítele`** po *když startuje můj klon* bude poměrně dlouhá:


když startuje můj klon
nastav Strana na náhodné číslo od 1 do 4
když Strana = 1 tak
skoč na x: 300 y: náhodné číslo od -200 do 200
když Strana = 2 tak
skoč na x: -300 y: náhodné číslo od -200 do 200
když Strana = 3 tak
skoč na x: náhodné číslo od -300 do 300 y: 200
když Strana = 4 tak
skoč na x: náhodné číslo od -300 do 300 y: -200
ukaž se
nastav směr k Hráč
opakuj dokud nenastane dotýkáš se Hráč ?
dopředu o 2 kroků
když dotýkáš se Střela ? tak
čekej 0.01 sekund
zruš tento klon
vyšli zprávu porážka
zastav všechno

Obrázek 149 - Ukázka kódu

Poznámka: Může se stát, že při zapnutí hry nyní klony **Střely** prolítávají skrz nepřátele a nezruší se. Problémem je, že přestože se scénář všech postav spouští ve Scratchi paralelně, může dojít k menším časovým odchylkám, tzn. některá sekvence se spustí o trochu dříve než jiná. To se děje i v tomto případě, kdy se při nárazu **Střely** do **Nepřítele** (respektive jeho klonů) nejdřív vyplní podmínka v klonu **Nepřítele**, který se tak zruší, a pak teprve se zkontroluje ve scénáři **Střely**, zda se má zrušit její klon po dotyku s **Nepřítelem**. Jelikož je ale v té době již klon **Nepřítele** zrušen, podmínka ve **Střele** se tak nevyplní a klon **Střely** proletí skrz klon **Nepřítele** dál až na okraj, přičemž právě až po dotyku s ním se zruší. Řešením může být pozdržet malou chvíli zrušení klonu **Nepřítele**, aby na dotyk stačil zareagovat i klon **Střely**. Stačí proto přidat v **Nepříteli** před *zruš tento klon* blok *čekej _ sekund* a dovnitř zadejte velmi malou hodnotu (např. 0,01 sekundy). I tento malý okamžik navíc postačí k tomu, aby klon **Střely** stačil na situaci zareagovat a skryl se.



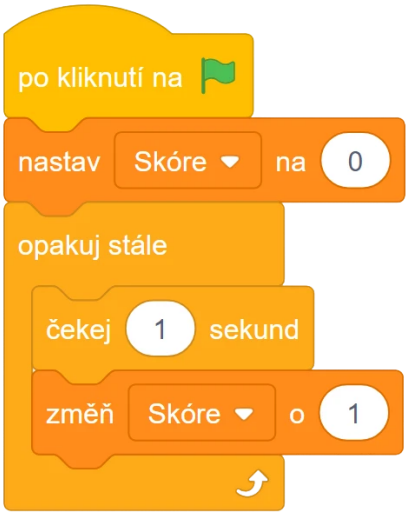
Obrázek 150 - Ukázka kódu

7. Dokončení hry – porážka a skóre

   - Pro zobrazení textu „GAME OVER“ je potřeba přidat scénář do postavy **Prohra**. Nechte žáky samostatně pracovat. V postavě **Prohra** stačí přidat dvě krátké sekvence – *po kliknutí na zelenou vlajku* se skryj a *po obdržení zprávy porážka* se ukaž. Text „GAME OVER“ se tak zobrazí pouze poté, co hráč prohraje, jinak zůstává skryt.

Obrázek 151 - Ukázka kódu

- Aby bylo možné porovnat, kdo z hráčů vydržel déle, je vhodné do hry přidat ukazatel v podobě skóre. Žáci by měli být schopni si s realizací poradit. Vytvořte novou proměnnou `Skóre`, která se bude zvyšovat každou sekundu o 1. Diskutujte s žáky, kam kód pro nastavení proměnné umístit a proč (vhodné je např. pozadí).



Obrázek 152 - Ukázka kódu

8. Volitelné: Rozšíření hry
    - Hra přímo vybízí k různým vylepšením a rozšířením, můžete případně ve zbývajícím času žákům nechat prostor pro kreativitu.
    - Možností je i zeptat se žáků, jak by změnili kód, aby:
        1. se rychlost pohybu klonů **`Nepřítele`** postupně zvyšovala
        2. se klony **`Nepřítele`** objevovaly v kratším intervalu
        3. **`Hráč`** střílel jen po stisku klávesy/myši

Vývoj: Cílem projektu Střílečka bylo původně pouze představení klonování, až v průběhu tvorby kódu bylo do lekce přidáno i vysvětlení operátorů k vytváření složených podmínek. Uvažovalo se nad tím, že by součástí projektu bylo i představení funkce Moje bloky, zejména z časových důvodů (lekce by byla

příliš obsáhlá na 90 minut) ale z myšlenky sešlo. Pro první seznámí s klonováním byla následně vytvořena aktivita BIT_Aktivita_Klonování. Po konzultaci s kolegy byly v aktivitě k projektu přejmenovány kostýmy na hvězda-černá a hvězda-bílá, protože původní názvy (hvězda-rodič a hvězda-klon) by mohly u žáků vést ke zmatení a k nesprávnému pochopení látky. Na základě konzultace s vedoucím práce byla podstatně rozšířena metodika o více pedagogických poznámek, aby bylo jasné, jaká by měla být role vyučujícího a jak by měl být veden výklad při jednotlivých krocích.

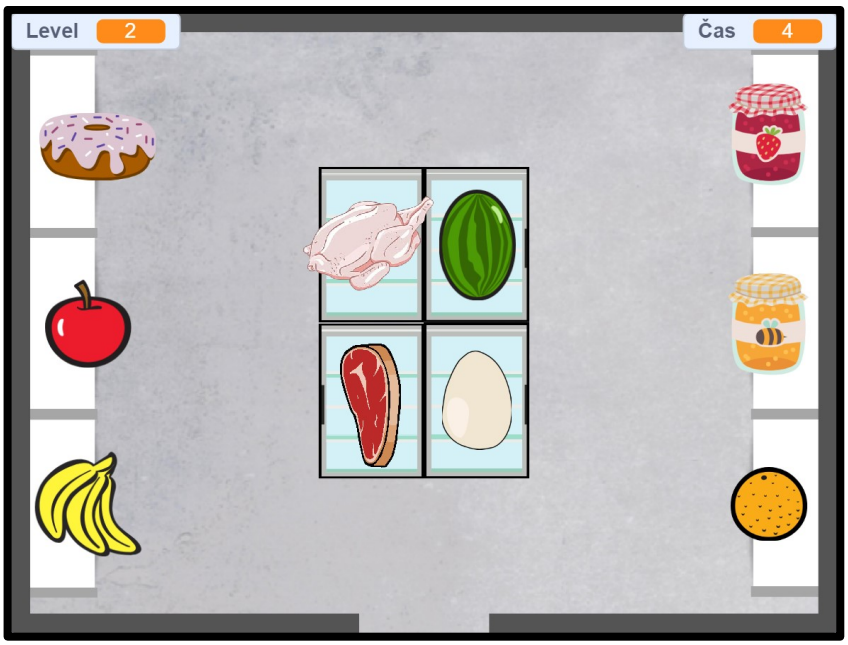
Průběh ověření: Obě skupiny zvládly úvodní aktivitu bez větších komplikací, dle odpovědí žáků na dotazy lze usuzovat, že všichni pochopily základní princip klonování a možnosti jeho využití. Aktivita nakonec zabrala delší čas, než se předpokládalo, s oběma skupinami trvala přibližně 30 minut. Žáci dokázali na základě dotazů správně určit, kde se využije klonování v projektu Střílečka. Jako bezproblémový pro pochopení se ukázal blok *nastav směr k _*, kteří někteří žáci sami nalezli a navrhovali k použití. Tvorba **Hráče** i **Střely** probíhala bez jakýchkoliv zádrhelů, v obou skupinách pochopili rozdíl mezi *_ a _* a *_ nebo _* a dokázali následně určit, který operátor je vhodný k použití v projektu. Pro některé žáky bylo při vytváření scénáře **Nepřítele** příliš obtížné zpočátku pochopit, proč pro každou stranu vytváříme jednu podmínku, smysl sekvence pochopili až posléze při vyzkoušení. Obě skupiny si udržely vysokou míru koncentrace po celou dobu výukového bloku, dle jejich reakcí se zdálo, že je projekt bavil. Celý projekt se i s aktivitou podařilo dokončit do 90 minut.

Úpravy po ověření ve výuce: Při vytváření pohybu **Nepřítele** někteří žáci navrhovali použít blok *klouzej _ sekund na Hráč*, který by sice měl požadovanou funkci (pohyb směrem k **Hráči**) a navíc by se v kódu použil jen jeden blok místo dvou, ale v sekvenci by způsobil komplikace, které by nakonec vedly k nefunkčnosti celého kódu. Byla proto do metodiky přidána poznámka upozorňující na tento problém, jež kromě ukázky možného alternativního řešení obsahuje i detailní rozbor a odůvodnění, proč je lepší pro pohyb použít kombinaci bloků *nastav směr _* a *dopředu o _ kroků*.

9.10 Lekce 10 – Nakupování

Délka: 80-90 minut

O projektu: Obchod zavírá za malou chvíli, hráč tak musí za deset sekund stihnout nakoupit vše, co má napsáno na nákupním seznamu. Problémem je, že nákupní seznam si nechal doma, tudíž musí spoléhat na svou paměť a koupit pouze ty potraviny, které měl na svém nákupním seznamu.



Obrázek 153 - Ukázka ze hry

Cíl lekce: Hlavním cílem lekce je seznámit žáky se seznamy. Žáci se naučí, jak seznamy fungují a jak s nimi pracovat. Lekce si klade za cíl též zopakovat získané znalosti z předešlých lekcí a procvičit schopnost informatického myšlení žáků při vytváření hry. Základní oporou fungování celé hry je vysílání a obdržení zpráv, žáci si tak rozšíří znalosti i v této oblasti.

Nově naučené bloky: *po kliknutí na mě*; *opakuj _ krát*; *vytvoř seznam*; *přidej _ k _*; *smaž _ z _*; *smaž všechno z _*; *pořadí _ ve _*; *délka _*; *_ obsahuje _*; *ukaž seznam _*; *skryj seznam _*

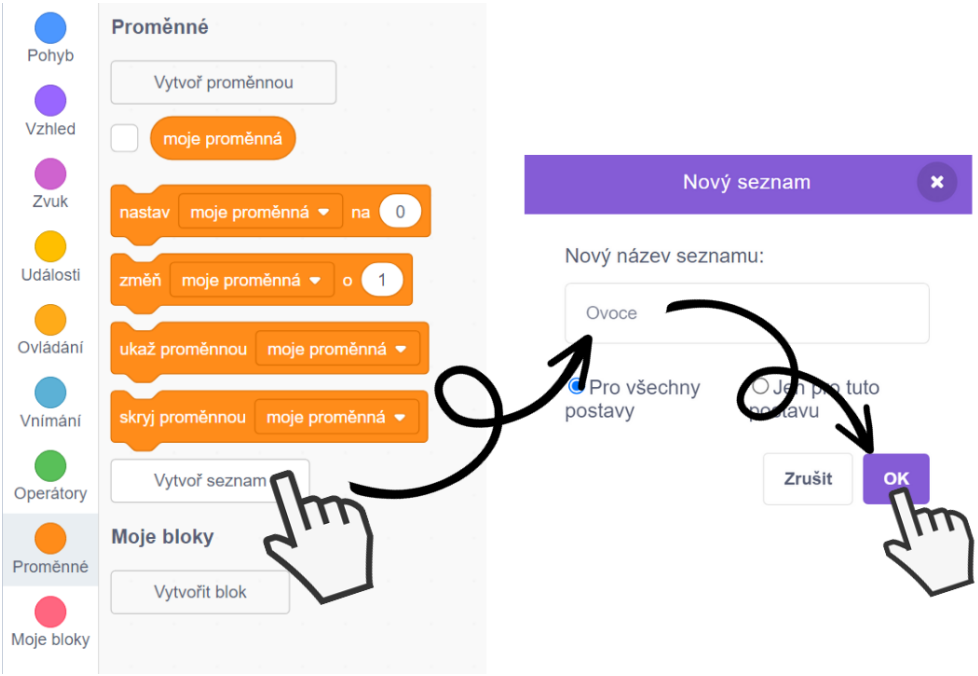
Náplň lekce:

1. Propedeutická aktivita na seznamy:
    - V tomto projektu budou žáci pracovat se seznamy, proto je nutné je nejdříve s principem fungování seznamů obeznámit.
    - Poskytněte žákům projekt BIT_Aktivita_Seznam_Start, který si otevřou. Nechte žáky, ať prozkoumají scénáře všech postav. Zjistí, že projekt není hotový a bude třeba ho dokončit. V postavě **`Jablko`** se nachází zadání, které je napsané v poznámce ve scénáři postavy. Cílem je vytvořit projekt, který si zapisuje, na jaké ovoce hráč klikl. V případě, že hráč klikne na ovoce v pořadí **`Jablko-Banány-Jahoda`**, zobrazí se nápis „Správně!“ (tzn. zobrazí se postava **`Výhra`**). Ukažte žákům, jak by měl vypadat výsledek, viz projekt BIT_Aktivita_Seznam.

- Diskutujte s žáky nad možným postupem k vytvoření cílového projektu. Možností je vytvořit si tři proměnné, přiradit každou k jedné postavě a nastavit, že po kliknutí na postavu se daná proměnná zobrazí, jinak bude skrytá. Posléze by bylo nutné hlídat pořadí, v jakém bylo na postavy kliknuto, což by bylo možné řešit další proměnnou a bloky *čekej dokud nenastane _*. Takovéto řešení je ale příliš komplikované a zdlouhavé, proto žákům představte seznamy.
- Seznam je vhodné použít, pokud potřebujeme uchovávat větší množství hodnot. Délka seznamu je proměnlivá, můžeme hodnoty do seznamu přidávat i je odebírat. Pokud bychom například chtěli pracovat se jmény žáků, místo toho abychom vytvářeli např. 15 proměnných pro každého žáka (`Žák1`, `Žák2`, `Žák3`…), stačí vytvořit jeden seznam, do kterého můžeme přidat prakticky nekonečno hodnot (ve Scratchi 3.0 je délka seznamu omezena na 200 000).

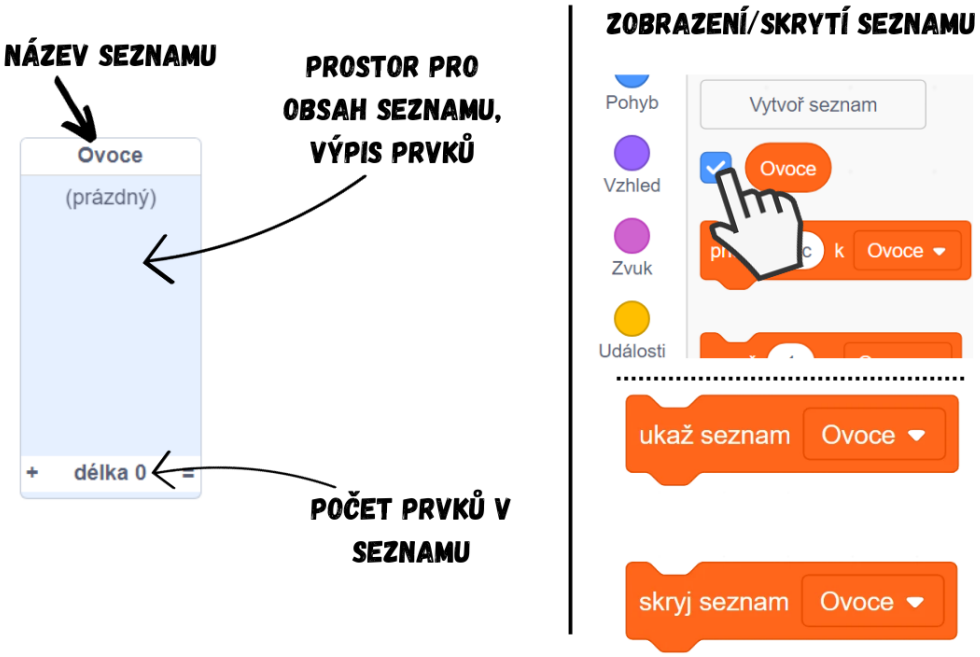
Poznámka: Během ověřování ve výuce se ukázalo jako velmi vhodné vysvětlovat funkcionality seznamu na příkladech z reálného života, např. na třídě. Pokládejte žákům otázky typu „Jaká by byla délka seznamu `Třída`, který by obsahoval všechny lidi v této třídě?“, „Obsahuje seznam `Třída` prvek Matěj?“ atp.

- Seznam vytvoříte kliknutím na *Vytvoř seznam* v kategorii *Proměnné*. Objeví se okno, do kterého zadejte název seznamu (kupříkladu Ovoce) a potvrďte stisknutím tlačítka OK.



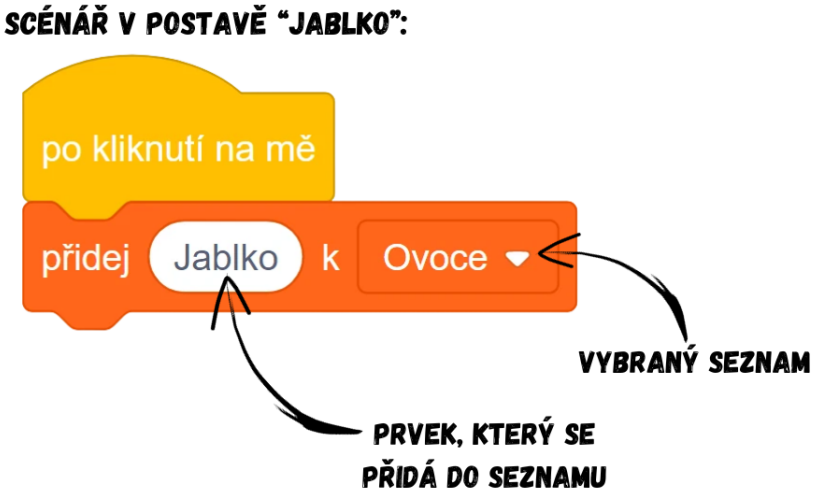
Obrázek 154 - Návod

- Seznam se po vytvoření standardně zobrazí ve scéně, avšak nyní neobsahuje žádné prvky, má tudíž délku 0. Seznamy lze ukazovat a skrývat podobně jako proměnné, tedy buď prostřednictvím checkboxu vedle názvu seznamu v kategorii *Proměnné*, popř. je možné využít bloků *ukaž seznam _* a *skryj seznam _*.



Obrázek 155 - Návod

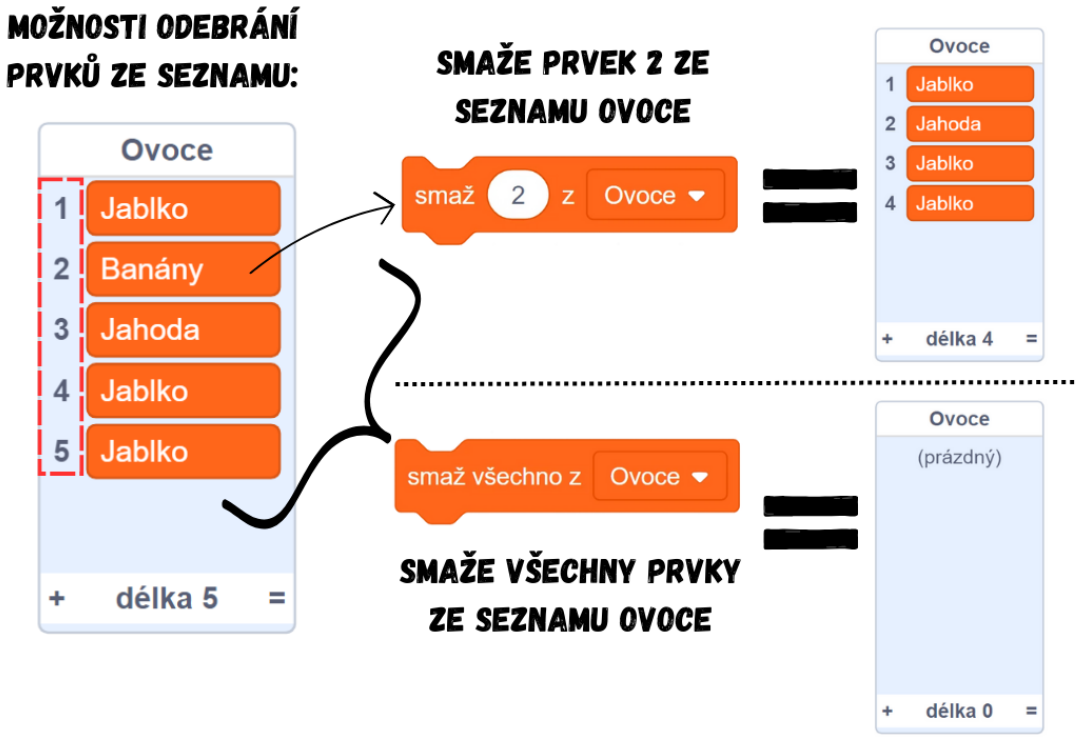
- Pro přidání prvku do seznamu použijte blok *přidej _ k _*. Vložte ho do scénáře **`Jablka`** a připojte k *po kliknutí na mě*. Napište místo „věc“ slovo „Jablko“ a vyberte seznam, do kterého se má prvek „Jablko“ přidat, tedy např. *přidej Jablko k Ovoce*. Po kliknutí na **`Jablko`** se nyní seznam rozšíří o nový prvek „Jablko“. Pokud klikneme na postavu **`Jablko`** vícekrát, budeme mít v seznamu vícekrát stejný prvek. Všimněte si, že každý prvek v seznamu má své pořadové číslo, tzv. index. Na základě získaných znalostí nechte žáky přidat bloky *přidej _ k _* i do postav **`Banány`** a **`Jahoda`** tak, aby se po kliknutí na **`Banány`** přidal do seznamu `Ovoce` prvek „Banány“ a po kliknutí na **`Jahodu`** zase prvek „Jahoda“.



Obrázek 156 - Princip fungování bloku

Obrázek 157 - Princip fungování seznamu

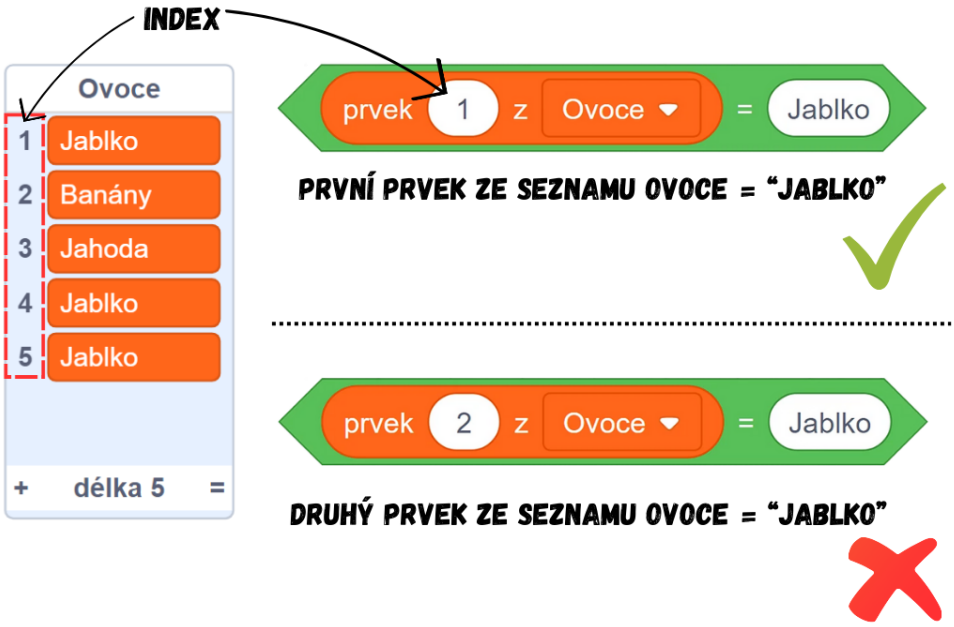
- V této chvíli pravděpodobně žáci mají velmi dlouhý seznam plný prvků. Pro odebrání prvků můžete použít *smaž _ prvek z _*, který smaže jeden prvek nacházející se na daném indexu, např. *smaž 3 prvek z Ovoce* by smazal v pořadí třetí prvek v seznamu `Ovoce` (prvek s indexem 3) a všechny prvky s vyšším indexem by posunul o jednu pozici níže (prvek s indexem 4 klesne na index 3, pátý klesne na čtvrtý atd.). Pro kompletní „vyčištění" seznamu a odebrání všech prvků je vhodnější použít blok *smaž všechno z _*, který smaže všechny prvky z vybraného seznamu, jehož délka tedy posléze bude 0. Připojte tento blok k *po kliknutí na zelenou vlajku* a celou sekvenci vložte např. do scénáře scény (pozadí).



Obrázek 158 - Návod

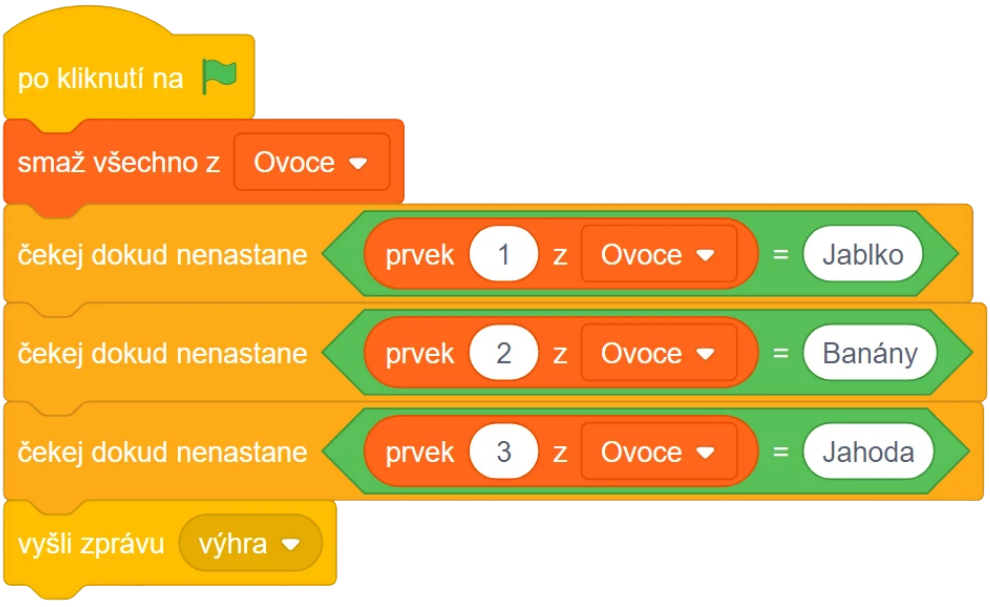
Obrázek 159 - Ukázka kódu

- Seznam je nyní po zapnutí hry vždy prázdný, klikáním na postavy do něj přidáte příslušné prvky. Je třeba naprogramovat, aby si hra hlídala, zda byl zadán správný „kód“ (tzn. zda bylo kliknuto na postavy ve správném pořadí, v tomto případě **`Jablko-Banány-Jahoda`**), který povede k zobrazení nápisu „Správně!“ (čili zobrazení postavy **`Výhra`**). Nejvhodnější řešení je použít ve scénáři scény (pozadí) blok *prvek _ z _*, který v sobě obsahuje hodnotu prvku nacházejícího se na zadané pozici ve vybraném seznamu. Pomocí tohoto bloku a operátoru _ = _ vytvořte podmínku, která se splní, pokud bude na zadané pozici požadovaný prvek. Ukažte žákům, kdy bude podmínka splněna a kdy ne (viz obrázek níže).



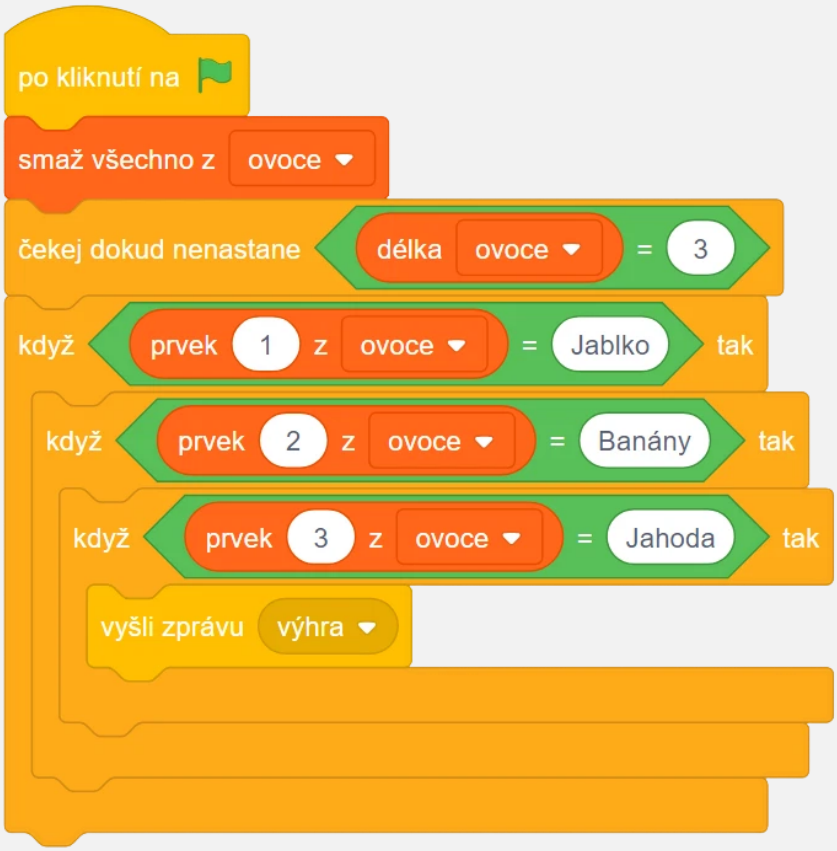
Obrázek 160 - Princip fungování bloků

- Vložte operátor *prvek 1 z Ovoce = Jablko* jako podmínku do bloku *čekej dokud nenastane _*. Postupujte obdobně pro zbytek správného „kódu“, tedy přidejte další blok *čekej dokud nenastane _*, tentokrát však s podmínkou *prvek 2 z Ovoce = Banány* a pro třetí blok vytvořte podmínku *prvek 3 z Ovoce = Jahoda*. Vytvořená sekvence se tak skládá ze tří bloků *čekej dokud nenastane _*, které se budou v případě správného postupu postupně vyplňovat. Představit si to můžete jako západy v zámku, které se postupně otevírají. Zde se postupně plní dílčí podmínky, až se dojde na konec, kde vyšleme zprávu „výhra“ prostřednictvím *vyšli zprávu výhra*, po jejímž obdržení se zobrazí postava **`Výhra`**, viz scénář postavy **`Výhra`**. Řešení ve scénáři scény (pozadí) by vypadalo následovně:



Obrázek 161 - Ukázka kódu

Poznámka: Výše uvedené řešení není z hlediska efektivity kódu příliš vhodné v případě, že hráč nakliká špatné pořadí. Sekvence totiž stále běží, respektive stále čeká na správné pořadí v seznamu, přestože už seznam obsahuje kupříkladu 10 prvků a k „vítězství“ tak už dojít logicky nemůže. Vhodnější by bylo proto určit moment, ve kterém se seznam zkontroluje a pokud obsahuje správnou kombinaci, vyšle zprávu „výhra“. V opačném případě by sekvence skončila bez vyslání zprávy. Onen moment vhodný ke zkontrolování by mohl nastat, když seznam obsahuje 3 prvky. K takovému řešení použijte blok *délka _*, který v sobě uchovává hodnotu délky vybraného seznamu. Sekvence by tak čekala, dokud délka seznamu není 3, posléze by se zkontrolovala, zda je zadán správný „kód“ a podle toho by sekvence skončila buď s vysláním, či bez vyslání zprávy „výhra“. Řešení by mohlo vypadat takto:



Obrázek 162 - Ukázka kódu

Možností je samozřejmě místo tří podmínek utvořit jednu složenou pomocí operátorů *_ a _*.

2. Úvodní diskuze:
    - Představte žákům zamýšlený projekt a ukažte jim, jak by měl výsledek vypadat, viz BIT_10_Nakupování.
    - Diskutujte s žáky o tom, s čím už si jsou schopny poradit díky svým znalostem z předchozích lekcí či aktivity a s čím si rady neví, co je tedy naopak nové.
3. Stáhněte si počáteční projekt
    - Použijte připravený projekt BIT_10_Nakupování_Start, který obsahuje všechny postavy a pozadí potřebné pro hru. Žákům na začátku hodiny projekt nasdílejte, popř. dejte ke stažení.

- Kromě postavy **`Start`** jsou všechny ostatní postavy skryté. Ve scénáři scény (pozadí) se nachází připravená sekvence spouštějící se po zapnutí hry.

4. Vymyšlení koncepce řešení

   - Na základě ukázky výsledné hry diskutujte s žáky nad možným postupem řešení. Hra se skládá v podstatě ze dvou stavů. Prvním je stav, ve kterém všechny potraviny zmizí, pozadí ztmavne a zobrazí se nákupní seznam spolu s tlačítkem **`Start`**. Po kliknutí na **`Start`** se hra přepne do druhého stavu – tlačítko spolu s nákupním se seznamem skryje, změní se pozadí na světlejší, ukážou se všechny potraviny a začne běžet čas. Základním stavebním kamenem hry budou zprávy, jejichž prostřednictvím budeme rozlišovat dva stavy hry, nazvěme je například „pauza“ a „hra“.

Obrázek 163 - Ukázka rozdělení hry na dva stavy

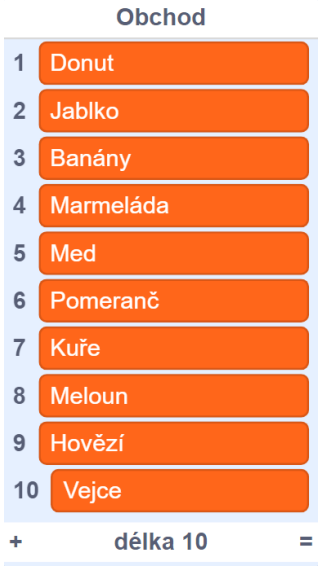
   - Pro odlišení těchto dvou stavů budeme proto v každé postavě sestavovat dvě sekvence – jednu, která se spustí po *po obdržení zprávy hra*, druhou po *po obdržení zprávy pauza*.

Poznámka: Je doporučeno se žáky o dvou stavech hry široce diskutovat a nechat je sepsat kupříkladu na tabuli, jak se budou postavy chovat v odlišných stavech. Je důležité, aby si žáci uvědomili, jak postavy na změnu stavu reagují a co je tedy vlastně cílem při tvorbě jednotlivých scénářů.

Obrázek 164 - Model školní tabule s možným výkladem

5. Vytvoření stavu „pauza“

   - Připomeňte žákům, že ve scénáři scény (pozadí) se nachází připravená sekvence, která vytváří pomocný seznam `Obchod`, jenž obsahuje názvy všech potravin. Obchod je skrytý, nechte žáky, ať si ho zobrazí a podívají se na jeho obsah (buď zaškrtnutím checkboxu v kategorii *Proměnné*, či pomocí bloku *ukaž seznam _*). Po prozkoumání nezapomeňte `Obchod` zase skrýt, jedná se pouze o pomocný seznam, který využijeme k sestavení seznamu nákupního.

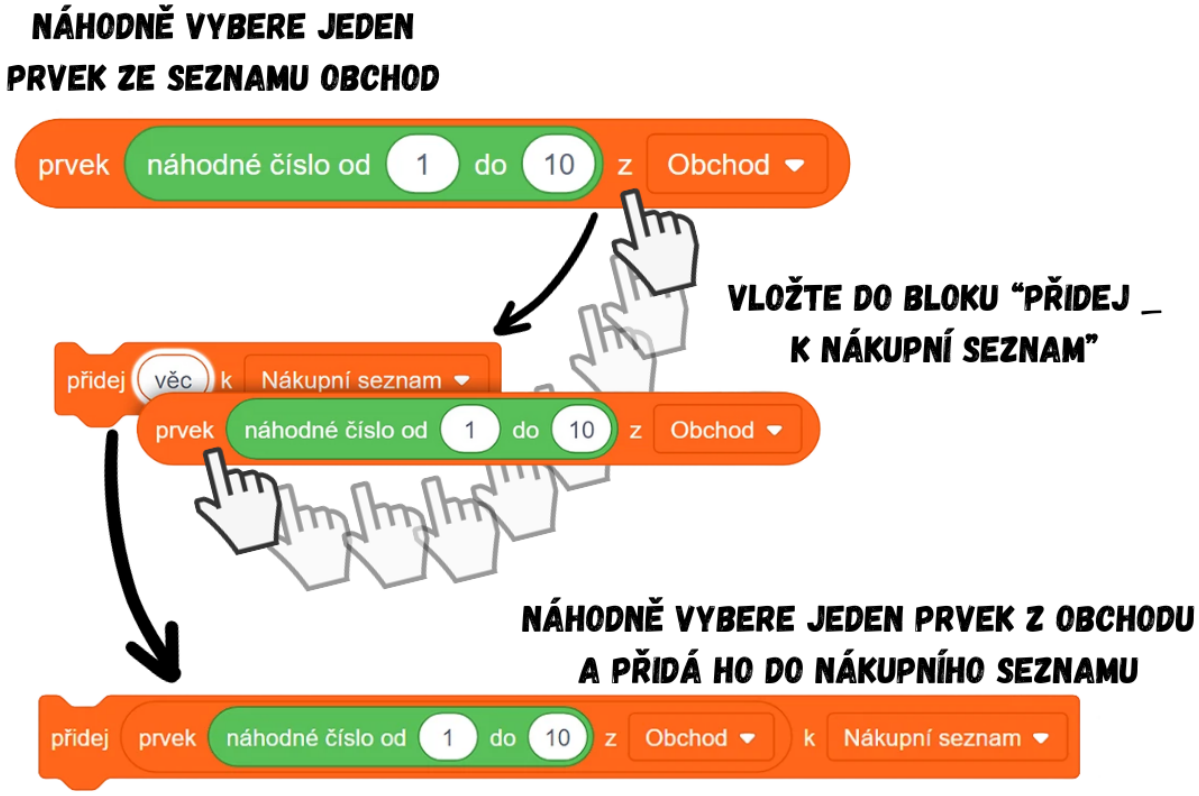


Obrázek 165 - Ukázka seznamu Obchod

- Z tohoto seznamu tak můžeme jednoduše sestavit nákupní seznam tím, že budeme náhodně vybírat daný počet prvků ze seznamu `Obchod` do nového seznamu. Vytvořte nový seznam a nazvěte ho např. `Nákupní seznam`.

Obrázek 166 - Návod

- Do `Nákupního seznamu` budeme chtít přidávat určitý počet prvků z `Obchodu`. Nejprve musíme náhodně vybrat jeden z prvků seznamu `Obchod`, zeptejte se žáků, jaký blok využijeme (použijte k tomu blok *prvek _ z Obchod*). Místo toho, abychom zadávali pevně index, vložte dovnitř bloku *náhodné číslo od 1 do 10* (`Obchod` má délku 10, obsahuje tedy deset prvků s indexy 1-10). Správnost sestaveného bloku můžete ověřit opakovaným kliknutím na něj, vybrané prvky by se měly měnit. Prvky chceme přidat do `Nákupního seznamu`, nechte žáky dokončit přidání (tzn. sestavený blok vložíme do *přidej _ k Nákupní seznam*).

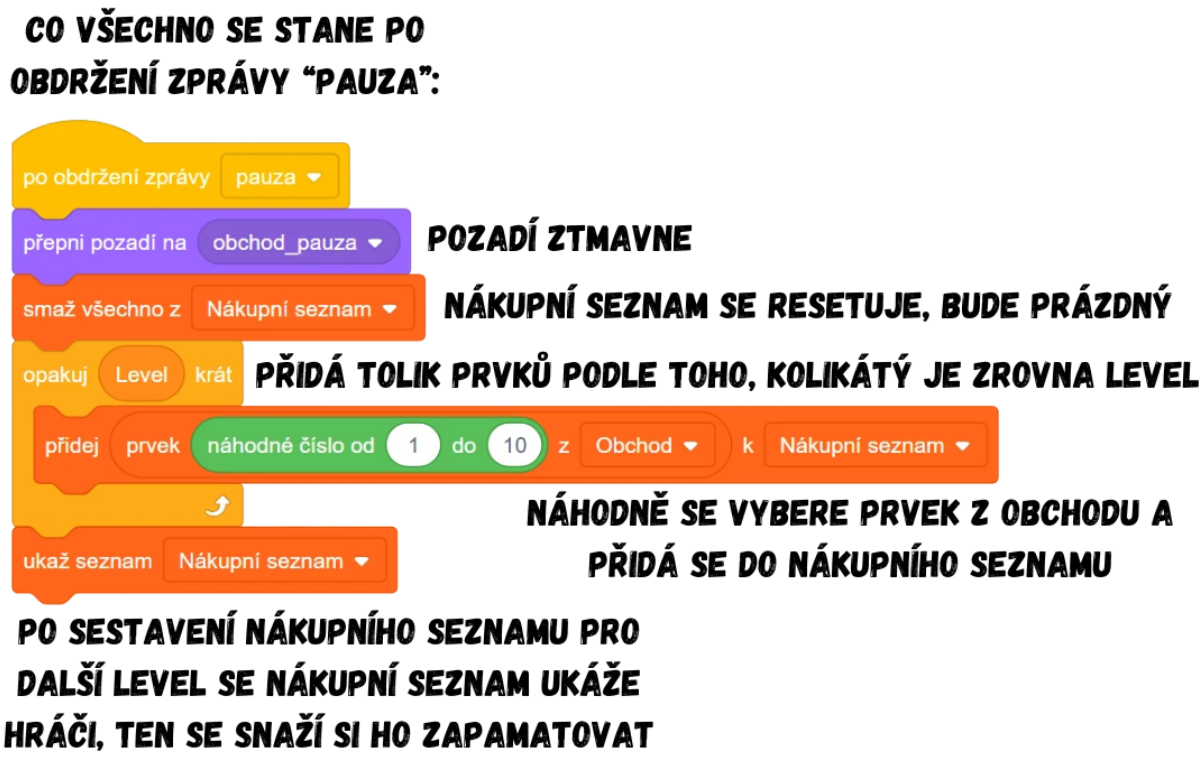


Obrázek 167 - Ukázka kódu s návodem

- Nyní už jen zbývá určit, kolik prvků má `Nákupní seznam` mít. Hra funguje na principu levelů, tzn. s každým levelem se obtížnost ztíží v podobě rozšíření `Nákupního seznamu` o jeden prvek. Po zapnutí hry se proměnná `Level` nastaví na 1. Použijeme hodnotu proměnné `Level` k určení toho, kolik prvků se má přidat do `Nákupního seznamu`. Nechte žáky, ať podle toho určí, kolik prvků má být v jakém levelu. V prvním levelu bude mít `Nákupní seznam` jednu položku, v druhém dvě, ve třetím tři atd. Přidání prvku budeme opakovat tolikrát, kolik je zrovna hodnota `Levelu`. K tomu použijte cyklus *opakuj _ krát*. Vysvětlete žákům, že na rozdíl od *opakuj stále* zopakuje tento cyklus kód do něj vložený pouze tolikrát, kolik mu zadáme. V našem případě má přidání prvku do `Nákupního seznamu` zopakovat `Level`-krát (když bude `Level` 1, cyklus se provede jednou a přidá tak pouze jeden prvek, když bude `Level` 5, v seznamu poté nalezneme pět prvků).

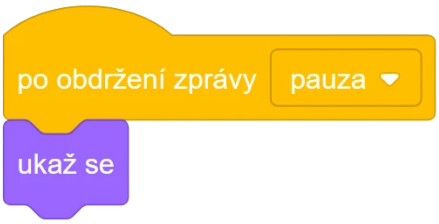
Obrázek 168 - Ukázka kódu s návodem

- Připojte cyklus k již předpřipravené sekvenci *po obdržení zprávy pauza*. Vysvětlete žákům, že jelikož se po dokončení každého levelu opět vyšle zpráva „pauza“, měl by se `Nákupní seznam` s každým obdržením této zprávy ještě resetovat, aby byl pokaždé před sestavením prázdný. Podobně ve stavu „hra“ bude `Nákupní seznam` skrytý, je proto nutné ho během „pauzy“ po sestavení zase ukázat, aby hráč měl šanci zjistit, jaké potraviny má nakoupit. Celá sekvence spouštějící se po *po obdržení zprávy pauza* by mohla vypadat následovně:



Obrázek 169 - Ukázka kódu

- Zeptejte se žáků, jaké všechny postavy mají reagovat na zprávu „pauza“. Na „pauzu“ reagují také všechny postavy potravin. Po jejím obdržení se skryjí, aby uvolnili prostor na scéně pro tlačítko **`Start`** a `Nákupní seznam`. Nechte žáky využít znalosti o kopírování kódu a přidejte do scénáře každé potraviny následující kód:



Obrázek 170 - Ukázka kódu

- Stejný kód vložte i do tlačítka **`Start`**, které se musí po každém dokončeném levelu zobrazit spolu s `Nákupním seznamem`.

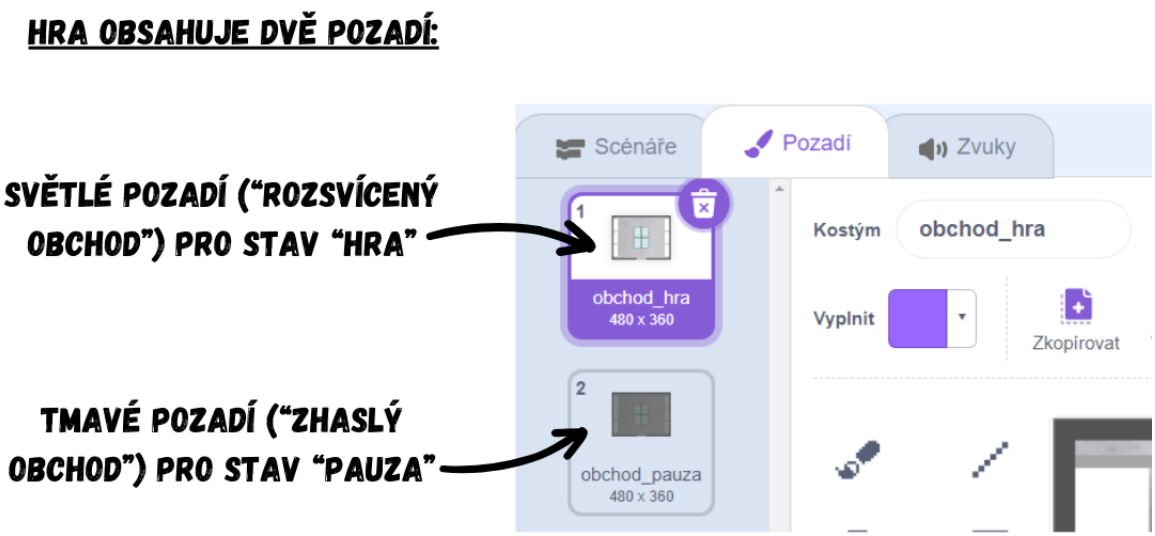
6. Vytvoření stavu „hra“

- Diskutujte s žáky o stavu „hra“. Připomeňte, že k vyslání zprávy „hra“ a tím i k přepnutí celé hry do tohoto stavu dojde po kliknutí na tlačítko **`Start`**. Ve scénáři tlačítka proto přidejte *po kliknutí na mě*. Po stisknutí se **`Start`** skryje, skryje se i `Nákupní seznam` a vyšle se zpráva „hra“, kterou je třeba vytvořit pomocí nástroje „Nová zpráva“ v bloku *vyšli zprávu _*. Po vyslání „hry“ se má změnit pozadí, má začít běžet čas a mají se zobrazit všechny postavy potravin. Řekněte žákům, že je vhodné, aby měl hráč po kliknutí na tlačítko **`Start`** malou chvíli na přípravu a nezačal mu

hned běžet čas, řešením může být do sekvence spouštějící se po kliknutí na **`Start`** přidat před vyslání zprávy blok *čekej 1 sekund*. Celý scénář tlačítka **`Start`**:

Obrázek 171 - Ukázka kódu

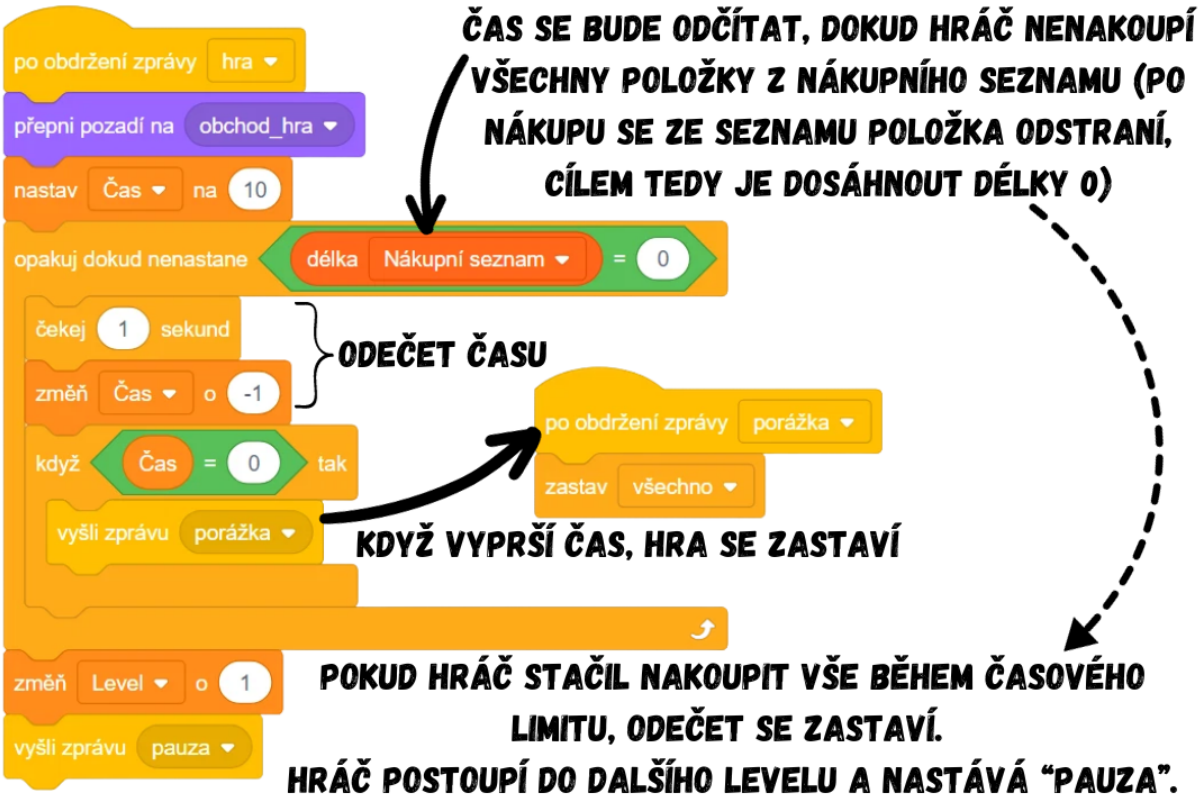
- Ve scénáři scény (pozadí) přidejte blok *po obdržení zprávy hra*, na který navážeme bloky související se změnou stavu. Zeptejte se žáků, jak reaguje pozadí na stav „hra“. Po obdržení této zprávy se má pozadí přepnout na světlé pozadí (obchod_hra).



Obrázek 172 - Ukázka obsahu záložky Pozadí

- Připomeňte žákům, že se dále má s každým vysláním „hry“ také spustit odpočet času. Vytvořte novou proměnnou `Čas` a nastavte ji např. na 10 (záleží na zamýšlené obtížnosti hry, tedy kolik chceme dát hráči času na nakoupení všech položek). Nechte žáky naprogramovat cyklus, který za každou sekundu sníží `Čas` o 1. Vhodný je k tomu cyklus *opakuj dokud nenastane _*, ovšem tentokrát budeme chtít, aby odpočet `Času` přestal, když hráč stihne nakoupit všechny požadované položky, nebo když bude `Čas` 0. Zeptejte se žáků, jak se změní proměnná Level v prvním případě

(změníme `Level` o 1 pro signalizaci toho, že hráč postoupil do další úrovně, což se odrazí i v počtu vylosovaných položek na `Nákupním seznamu`). Po dokončení levelu nastává stav „pauza“. V případě, že `Čas` vyprší a dosáhne hodnoty 0, je třeba zastavit odpočet a i celou hru (došlo k porážce) pomocí *zastav všechno*. Možné řešení celé situace:



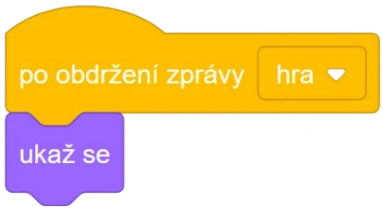
Obrázek 173 - Ukázka kódu

Poznámka: V případě použití tohoto řešení je třeba s žáky předem prodiskutovat, na jakém principu bude fungovat „nákup“ potravin. Zde se vychází z konceptu, že při nákupu správné potraviny (kliknutí na postavu potraviny, která se nachází na `Nákupním seznamu`) se příslušná položka z `Nákupního seznamu` odstraní. Proto odečet `Času` opakujeme, dokud není vše nakoupeno (délka `Nákupního seznamu` je 0). Když vyprší `Čas` (hodnota `Času` 0), vyšle se nová zpráva „porážka“. Všimněte si, že blok *zastav všechno* se nachází mimo sekvenci a spouští se vysláním zprávy „porážka“. Když vyjdeme z poznatku, že hru budeme chtít ukončit hned v několika situacích (když vyprší `Čas`, když hráč klikne na potravinu, která se nenachází na `Nákupním seznamu`), je vhodnější mít *zastav všechno* navázaný na obdržení zprávy, protože tak tento blok můžeme spouštět i z jiných scénářů (např. vysláním zprávy „porážka“ po kliknutí na postavu potraviny, jež se nenachází na `Nákupním seznamu`) a nemusíme mít *zastav všechno* ve scénářích hned několika postav.

Poznámka: Pokud by došlo v průběhu lekce ke zdržení, je možné z časových důvodů odpočet času ve hře vynechat a vytvořit jednodušší verzi bez časového limitu. Přidání mechaniky času lze pak nechat na žácích jako volitelné rozšíření hry. Zjednodušený kód bez odpočtu času by mohl vypadat takto:

Obrázek 174 - Ukázka kódu

- Během stavu „hra“ se mají zobrazit všechny postavy potravin. Vložte proto příslušnou sekvenci do každé z nich. Nechte žáky pracovat samostatně.



Obrázek 175 - Ukázka kódu

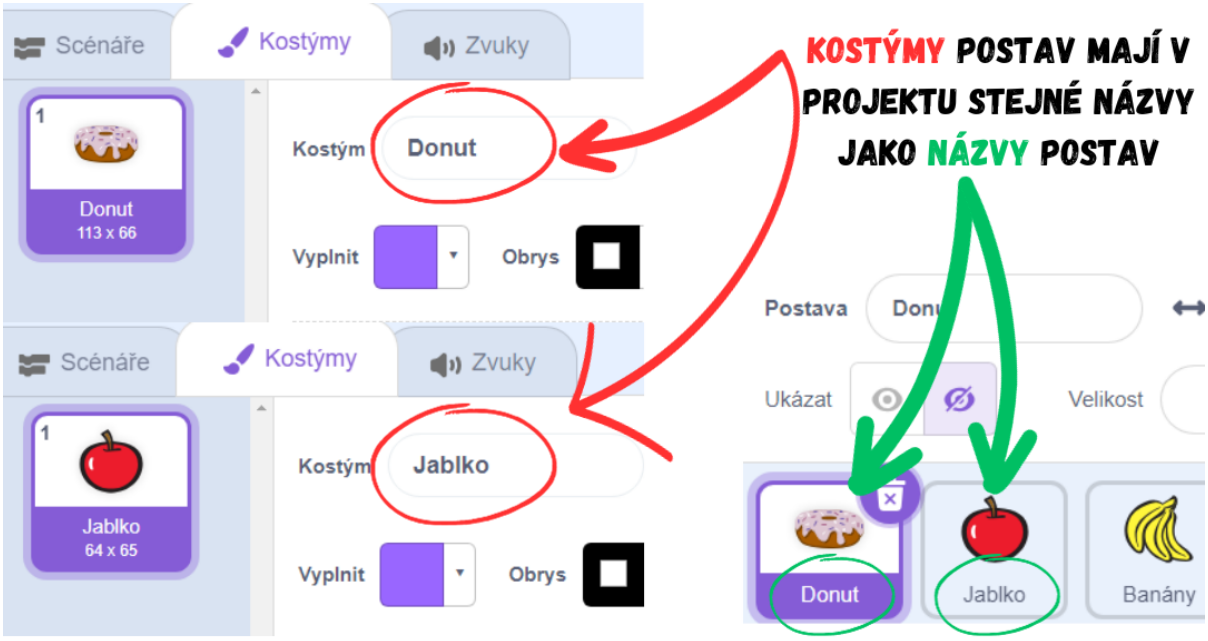
7. Nákup potravin a ověření správnosti

   - Zeptejte se žáků, co zbývá naprogramovat ohledně potravin (v postavách potravin zbývá naprogramovat jejich reakce na kliknutí myši). Po kliknutí na potravinu se musí zkontrolovat, zda se daná potravina nachází na `Nákupním seznamu`. Pokud ano, tato položka se z `Nákupního seznamu` smaže, v opačném případě dojde k porážce (tzn. po kliknutí na potravinu, která se nenachází na `Nákupním seznamu`, se vyšle zpráva „porážka“, po jejímž obdržení se hra zastaví prostřednictvím bloku *zastav všechno*, jenž se v případě použití výše ukázaného řešení v kroku 6 nachází ve scénáři scény). Připomeňte žákům, že blok *smaž _ z _* pracuje pouze s indexem (nelze napsat *smaž Donut z Nákupní seznam*), je proto nutné do něj napsat číslo indexu, na kterém se nachází postava, kterou chceme smazat z `Nákupního seznamu`. Ke zjištění indexu podle položky využijte blok *pořadí _ ve _*. Vysvětlete, že pokud by se např. **`Donut`** nacházel v `Nákupním seznamu` na prvním místě, blok *pořadí Donut ve Nákupní seznam* by vrátil hodnotu 1. Pro postavu **`Donut`** by kód mohl vypadat takto:

Obrázek 176 - Ukázka kódu

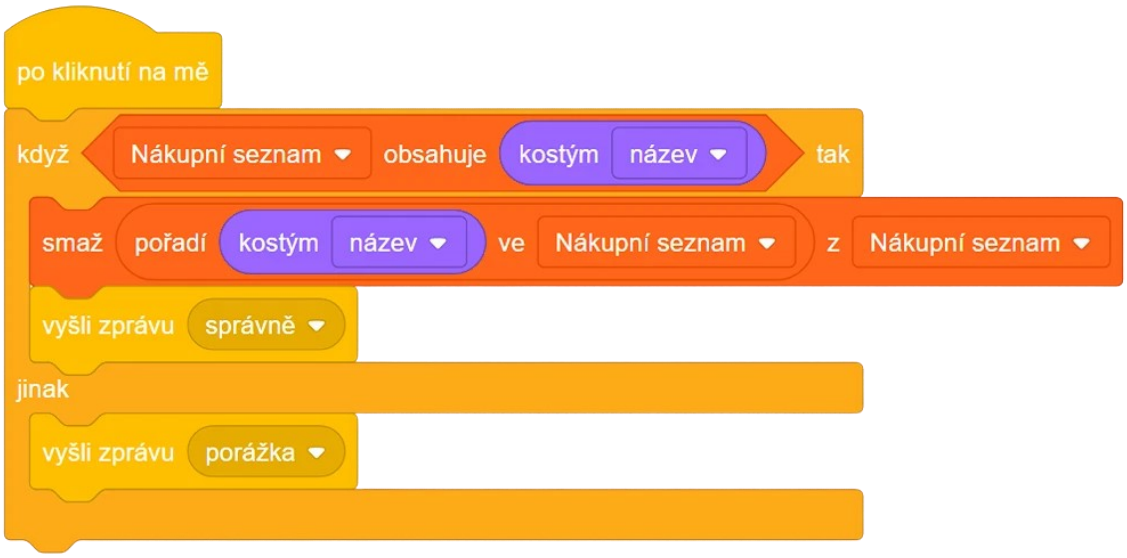
Poznámka: V sekvenci se v případě kliknutí na správnou potravinu kromě smazání příslušné položky v `Nákupním seznamu` vyšle také zpráva „správně“. Je vhodné hráči nějakým způsobem dát vědět, že se „trefil“ a klikl na správnou potravinu. To je možné řešit tím, že zobrazíme například nějakou postavu (její kostým může být buď text, nebo třeba fajfka). Tuto postavu budeme zobrazovat právě tehdy, když hráč klikne na správnou potravinu, tedy když je vyslána zpráva „správně“. V této sekvenci si tak „připravujeme půdu“ pro to, co je detailněji rozebíráno a řešeno v kroku 8. Pokud bychom však nyní *vyšli zprávu správně* do sekvence nevložili a sekvenci následně zkopírovali do všech postav potravin, museli bychom pak při kroku 8 pracně upravit i scénář všech těchto postav.

- Abychom nemuseli psát v sekvenci ručně názvy jednotlivých potravin, bylo by ideální vložit místo názvů jednu proměnnou, jež by v sobě obsahovala název postavy, na kterou bylo kliknuto. Vysvětlete žákům, že ve Scratchi neexistuje způsob, jak vzít název postavy a vložit ho do proměnné. Lze se však pomocí předpřipravené proměnné *kostým _* v kategorii *Vzhled* dostat alespoň k názvu aktuálního kostýmu pomocí *kostým název*.



Obrázek 177 - Návod

- Připravený projekt BIT_10_Nakupování_Start byl navržen tak, že se názvy postav shodují s názvy kostýmů, což celou situaci velmi ulehčuje. Stačí tak v sekvenci pouze vložit *kostým název* na místa, kde se ručně psal název postavy.



Obrázek 178 - Ukázka kódu

- Celá sekvence nyní funguje obecně, stačí ji tedy vložit do scénářů všech postav potravin bez jakýchkoliv úprav. Nechte žáky pracovat samostatně. Scénář všech postav potravin by tedy měl vypadat identicky, kupříkladu následovně:

Obrázek 179 - Ukázka kódu

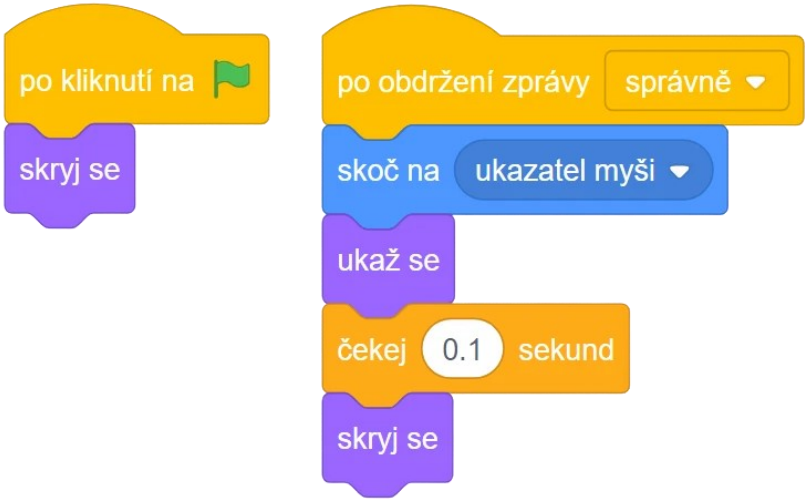
8. Dokončení hry – postavy **`Správně`** a **`Prohra`**

   - V současném stavu by hra měla být funkční, zbývá už pouze dodělat poslední detaily. Postava **`Prohra`** by se měla zobrazit pouze až po obdržená zprávy „porážka“, aby ale nezůstal nápis „GAME OVER“ svítit i po restartu hry, je třeba ho s každým zapnutím hry skrýt. Nechte žáky pracovat samostatně. Scénář postavy **`Prohra`**:

Obrázek 180 - Ukázka kódu

   - Další detail se týká zvýraznění správné odpovědi. Pokud hráč nyní klikne na správnou potravinu nalézající se na `Nákupním seznamu`, nedostane od hry v ten moment žádnou zpětnou vazbu. To může u hráče vyvolat dojem, že kupříkladu neklikl správně, což může vést k tomu, že kliknutí zopakuje a opakovaný nákup jedné potraviny povede k porážce. Bylo by proto vhodné dát hráči po kliknutí na správnou položku signál, že jeho volba byla správná, např. pomocí malé zelené fajfky.
   - Ukažte žákům, že k tomu využijeme předpřipravenou postavu **`Správně`**, jejíž kostým sestává právě ze zelené fajfky. Chceme, aby se fajfka krátce zobrazila na kurzoru myši, pokud došlo ke kliknutí na správnou potravinu. To se hlídá ve scénáři všech potravin, kdy se po kliknutí na správnou postavu vyšle zpráva „správně“, kterou nyní využijeme. Pokaždé, když postava **`Správně`** obdrží tuto zprávu, na chvíli se zobrazí a poté opět skryje. Tato problikávající fajfka bude sloužit jako dobrá zpětná

vazba pro hráče, který nyní bude vědět, že klikl správně a hra jeho kliknutí započítala (tzn. příslušná položka byla odebrána z `Nákupního seznamu`). Nechte žáky pracovat samostatně. Celý scénář postavy **`Start`** by mohl vypadat následovně:



Obrázek 181 - Ukázka kódu

Vývoj: Při koncipování této lekce se uvažovalo nad několika návrhy samotného projektu, mezi možnostmi byla hra, která by po „naklikání“ tajného kódu prostřednictvím barevných tlačítek otevřela dveře. Po konzultaci s kolegy byl nakonec zvolen návrh nakupovací hry spočívající v zapamatování si nákupního seznamu. Původní návrh projektu byl pozměněn a zjednodušen a jeho upravená verze posloužila jako základ pro propedeutickou aktivitu.

Průběh ověření: Obě skupiny na lekci dorazily s vysokou mírou koncentrace, tudíž celý počáteční výklad o seznamech probíhal bez komplikací a žáci na všechny doplňující dotazy správně odpovídali. Úvodní aktivita nedělala žákům větší problémy, sami v některých částech navrhovali řešení (použít tři podmínky *když _ tak*) a různá vylepšení („tajný kód“ složen ze čtyř kliknutí na ovoce). Žáci během diskuze nad postupem při tvorbě samotného projektu Nakupování projevili dobré analytické schopnosti a sami dokázali popsat, jak přesně se bude ve hře využívat seznam. Z toho lze usuzovat, že úvodní aktivita splnila svůj účel a žákům vhodně posloužila jako první seznámení se se seznamy, což poté ulehčilo práci při tvorbě projektu. Obě skupiny dokázaly sepsat na tabuli vše, co se má stát ve stavu „pauza“ a „hra“. Samy navrhly použití cyklu *opakuj _ krát* pro přidávání prvků na `Nákupní seznam`, stejně tak zvládly využít v něm proměnnou `Level`. Tvorba scénáře ve všech postavách byla bezproblémová, žáci mi radili, jaký má být další krok a pracovali tak po většinu času samostatně. Žáci sami požadovali různá vylepšení, s většinou se v lekci počítalo (zobrazení fajfky při správné odpovědi, odpočet času), k realizaci některých z časových důvodů nedošlo (náhodné umístění potravin v obchodě). Úvodní aktivita zabrala necelou půlhodinu, ve zbývajícím čase se celý projekt podařilo dokončit.

Žáci dokázali i přes komplexnost projektu samostatně provést rozbor problému a vypracovat návrh řešení. V průběhu kurzu si postupně osvojili všechny základní algoritmické koncepty, které byli schopni následně využít, z čehož lze usuzovat, že u žáků došlo k rozvoji algoritmického myšlení.

Úpravy po ověření ve výuce: Na základě ověření byla metodika podstatně rozšířena o komentáře k tomu, jak vést daný výklad. Byly přidány poznámky obsahující doporučení, o jaké příklady výklad opřít, mimo jiné i doplněné o nové obrázky (tabule s rozdíly postav podle stavů). Jelikož propedeutická aktivita zabrala v obou skupinách necelých třicet minut, je pravděpodobné, že jakékoliv případné zdržení by způsobilo, že vytvoření celého projektu v rámci bloku 90 minut by bylo nemožné. Z tohoto důvodu byla přidána poznámka navrhující v případě nedostatku času úpravu kódu pro vytvoření jednodušší verze projektu (vynechání mechanismu odpočtu času).

9.11 Souhrn výsledků ověřování

Po ověření všech lekcí ve výuce, konkrétně na dvou skupinách po devíti žácích s průměrným věkem 8-9 let, se přímo vybízí pokusit se zhodnotit výsledky a shrnout zkušenosti získané z ověřování deseti lekcí.

Obecně lze říci, že lekce žáky bavily, z výuky odcházeli většinou nadšeni s tím, že je mrzí, že již lekce skončila. Rozhodnutí rozvíjet algoritmické myšlení prostřednictvím tvorby her se ukázalo jako správné, pro žáky se jednalo o zábavnou a motivující formu učení se. Za velkou výhodu lze považovat i diverzifikaci jednotlivých projektů, kdy se nejedná o stále stejné hry, ale pokaždé je hra v něčem jiná, ač obvykle obsahuje jisté podobnosti s hrou předchozí. To se děje z prostého důvodu snahy o vytvoření návaznosti lekcí v kurzu, tedy aby žák využil nově získané znalosti nejlépe hned v další lekci. Různorodostí her se tak snižuje riziko, že by některého žáka nebavily všechny hry, zjednodušeně řečeno každého oslovila některá hra více a jiná zase méně. U chlapců bylo možné pozorovat zvýšenou motivaci a zapojení při tvorbě střílečky či her s automobilem, u dívek zase u her s postavami zvířat či her logických (Nakupování).

Osvojování algoritmických konceptů probíhalo u žáků až nad očekávání rychle, žáci až na výjimky neměli problém princip nově naučených bloků za krátkou dobu pochopit. Tyto získané zkušenosti byly proto promítnuty i do struktury celého kurzu prostřednictvím úprav jednotlivých lekcí po ověření. Společně s časovými důvody se jednalo o nejčastější příčinu změny struktury lekce. Pokud se při výuce ukázalo, že tvorba projektu trvá déle, než bylo plánováno, přistoupilo se k redukci některých kroků v lekci. V průběhu lekcí bylo možné při ověřování sledovat, jak se schopnosti žáků rozvíjejí a jak si postupně jednotlivé algoritmické koncepty osvojují.

Ověřování ve výuce přineslo i poznatky užitečné pro vhodnou skladbu metodiky. Podle reakcí žáků a jejich následných dotazů bylo jasnější, jak žáci přemýšlejí a co tedy ve výkladu obsáhnout a zdůraznit, popřípadě jak danou problematiku žákům vysvětlit, aby to pro ně bylo srozumitelné. Metodika byla proto upravena tak, aby tyto nabyté poznatky z výuky reflektovala.

Výsledným stanoviskem ověřování je, že soubor lekcí jako celek obstál a společně s provedenými úpravami má předpoklad k tomu být použit jako materiál pro výuku programování na 1. stupni základního vzdělávání, konkrétně pro věkovou kategorii žáků 8-10 let.

Závěr

Hlavním cílem práce bylo vytvoření souboru lekcí (kurzu) pro rozvoj algoritmického myšlení žáků na prvním stupni základních škol, konkrétně pro věkovou kategorii 8-10 let. Pro naplnění tohoto cíle bylo nutné zpracovat i jednotlivé úkoly.

Nejprve byly prostudovány odborné zdroje k pochopení toho, co vlastně algoritmické (resp. informatické) myšlení je a jaké všechny kompetence zahrnuje. Pro účely práce byla nakonec vybrána definice informatického myšlení jako způsobu myšlení zaměřeného na popis problému, jeho analýzu a hledání efektivních řešení, z níž bylo v práci nadále vycházeno.

Dalším krokem bylo nalezení vhodných typů programovacích aktivit k rozvoji algoritmického myšlení, přičemž se výzkum soustřeďoval i na otázku motivace žáků, bez níž by nebylo možné rozvíjet u žáků požadované schopnosti.

Posléze byly prostudovány příslušné pasáže v novém znění RVP pro základní vzdělávání s cílem zjistit, jak se k algoritmickému (resp. informatickému) myšlení a jeho rozvoji staví tento dokument MŠMT. Pro inspiraci byly navíc zanalyzovány i inkriminované části národních kurikul několika evropských států a při porovnání těchto částí s českým RVP se dospělo k závěru, že pojetí výuky informatiky a přístup k rozvoji algoritmického myšlení je ve všech vzdělávacích plánech prakticky stejný. Studium zahraničních kurikul tedy získání informací o případných jiných či inovativních přístupech nepřineslo.

Posledním úkolem teoretické práce bylo zmapovat zdroje zaměřené na rozvoj algoritmického myšlení a zjistit, v jakém pořadí jsou obvykle algoritmické koncepty osvojovány. Výsledkem mapování bylo, že ve výukových aplikacích a dětských programovacích jazycích se výhradně přistupuje k tomu, že po příkazech ovládajících pohyb postavy se žáci seznamují nejdříve s cykly a až poté s podmíněnými příkazy. Důvodem může být jak lepší názornost fungování cyklů na příkladu opakování pohybu, tak i nutnost použití dalších prvků u podmíněných příkazů, u kterých je pro žáky mj. nový i koncept toho, že se příkaz někdy vykoná a jindy ne.

Na základě výsledků teoretické části bylo přistoupeno k té praktické, jejíž hlavní náplní byla tvorba samotného kurzu. Lekce byly strukturovány tak, aby se v každé z nich žáci naučili něco nového, ale zároveň aby na sebe jednotlivé lekce navazovaly a žáci mohli využít osvojené algoritmické koncepty z předchozích lekcí. Obtížnost a komplexnost lekcí se v průběhu kurzu postupně zvyšuje takovým způsobem, aby se pro žáky vždy jednalo o přiměřenou výzvu, která v edukačním procesu slouží jako vhodný motivátor. Také bylo rozhodnuto, že cílem každé lekce bude tvorba hry, čímž byla zajištěna motivace žáků do budoucna pro další aktivity. Vytvořený soubor lekcí byl následně ověřen v praxi v hodinách kroužku programování. Poznatky získané při ověřování byly následně reflektovány v metodice kurzu a využity pro úpravu či vylepšení lekcí. Celý kurz byl následně zpřístupněn ke stažení na stránce https://kraken.pedf.cuni.cz/~benysekv/bit/. V průběhu jednotlivých lekcí bylo vidět, jak si žáci dokážou poradit s dílčími problémy, s jejichž obdobami se již setkali, přičemž jejich schopnost

samostatně pracovat se s každou lekcí zlepšovala. Výsledkem ověřování bylo, že žáci dokázali využívat osvojené algoritmické koncepty z předchozích lekcích v těch následujících, z čehož lze usuzovat, že u nich došlo k pochopení základních algoritmických konstrukcí a k rozvoji algoritmického myšlení.

Seznam použitých informačních zdrojů

Atmatzidou, S., & Demetriadis, S. (2016). *Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant diferences*. Robotics and Autonomous Systems, 75, 661–670. https://doi.org/10.1016/j.robot.2015.10.008

Bryndová, L. (2021). *The approach of computer science teachers to the concepts of computational thinking and the implementation of its development in primary schools*. Journal of Technology and Information Education, 13(2), 151-163. doi: 10.5507/jtie.2021.015

DfE. (2013). *The national curriculum in England*. Dostupné z: https://www.gov.uk/government/publications/national-curriculum-in-england-primary-curriculum [cit. 30. 12. 2023]

Futschek, G. (2006). *Algorithmic thinking: The key for understanding computer science*. Lecture Notes in Computer Science, 4226, 159–168. https://doi.org/10.1007/11915355_15

Chiu, C.-F. (2015). *Introducing pre-service teachers to programming concepts with game creation approach.* International Journal of Learning Teaching and Educational Research, 13(1), 85-93. Dostupné z: https://www.ijlter.org/index.php/ijlter/article/view/414 [cit. 29. 12. 2023]

Jiang, B., & Li, Z. (2021). *Effect of Scratch on computational thinking skills of Chinese primary school students.* J. Comput. Educ. 8, 505–525. https://doi.org/10.1007/s40692-021-00190-z

Katai, Z. (2015). *Promoting algorithmic thinking*. Journal of Computer Assisted Learning, 31, 287-299. https://doi.org/10.1111/jcal.12070

Lessner, D. (2014). *Analysis of term meaning "Computational Thinking"*. Journal of Technology and Information Education, 6(1), 71-88. doi: 10.5507/jtie.2014.006.

MEN. (2017). *Podstawa programowa kształcenia ogólnego. Szkoła podstawowa – Informatyka*. Dostupné z: https://www.ore.edu.pl/wp-content/uploads/2017/05/informatyka.-pp-z-komentarzem.-szkola-podstawowa-1.pdf [cit. 30. 12. 2023]

MENJ. (2015). Programmes d'enseignement du cycle des apprentissages fondamentaux (cycle 2), du cycle de consolidation (cycle 3) et du cycle des approfondissements (cycle 4). Dostupné z: https://www.education.gouv.fr/media/48461/download [cit. 30. 12. 2023]

Mezak, J., & Papak, P. P. (2018). *Learning scenarios and encouraging algorithmic thinking*. 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 760-765. https://doi.org/10.23919/mipro.2018.8400141

MŠMT. (2023). *RVP ZV z června 2023*. Dostupné z: https://www.edu.cz/wp-content/uploads/2023/07/RVP_ZV_2023_cista_verze.pdf [cit. 29. 12. 2023]

MŠVVŠ. (2023). *Štátny vzdelávací program pre základné vzdelávanie*. Dostupné z: https://www.minedu.sk/data/files/11808_statny-vzdelavaci-program-pre-zakladne-vzdelavanie-cely.pdf [cit. 30. 12. 2023]

Oficiální stránky projektu iMyšlení. Dostupné z: https://imysleni.cz/ [cit. 28. 12. 2023]

Olkhova, N. (2022). *Development of algorithmic thinking in primary school students when studying computer science*. Scientific Bulletin of Mukachevo State University. Series "Pedagogy and Psychology", 8(2), 25-32. https://doi.org/10.52534/msu-pp.8(2).2022.25-32

Palmer, D. (2005). *A Motivational View of Constructivist-Informed Teaching*. International Journal of Science Education. 27, 1853-1881. https://doi.org/10.1080/09500690500339654

Pintrich, P. R., Marx, R. W., & Boyle, R. A. (1993). Beyond Cold Conceptual Change: The Role of Motivational Beliefs and Classroom Contextual Factors in the Process of Conceptual Change. Review of Educational Research, 63(2), 167-199. https://doi.org/10.3102/00346543063002167

Stamatios, P. (2024). *Can Preschoolers Learn Computational Thinking and Coding Skills with ScratchJr? A Systematic Literature Review*. International Journal of Educational Reform, 33(1), 28-61. https://doi.org/10.1177/10567879221076077

Vais, J. (2021). *Rozvoj algoritmického myšlení žáků základních škol*. Diplomová práce, vedoucí Štípek, Jiří. Univerzita Karlova v Praze, Pedagogická fakulta, Katedra informačních technologií a technické výchovy. http://hdl.handle.net/20.500.11956/125706

Vaníček, J., Šimandl, V., & Dobiáš, V. (2022). *Situational algorithmic tasks with the assembling of program code as a tool for developing computational thinking*. Journal of Technology and Information, 14(2), 101–119. https://doi.org/10.5507/jtie.2022.014

Weintrop, D., & Wilensky, U. (2016). *Playing by Programming: Making Gameplay a Programming Activity. Educational Technology*, 56(3), 36–41. http://www.jstor.org/stable/44430491

Wing, J. M. (2006). *Computational thinking*. Communications of the ACM, 49(3), 33. https://doi.org/10.1145/1118178.1118215

Seznam příloh

Příloha č. 1 – Složka Obrázky (k promítání ve výuce)

Příloha č. 2 – Scratch soubory

Příloha č. 3 – Pracovní verze lekcí (vývoj lekcí)

Příloha č. 4 – Tabulka pořadí osvojování algoritmických konceptů

Příloha č. 5 – Prezentace Úhly a stupně

Příloha č. 6 – Zmapované zdroje