Univerzita Karlova

Pedagogická fakulta

Katedra informačních technologií a technické výchovy

BAKALÁŘSKÁ PRÁCE

Vývoj mobilních aplikací v blokovém programovacím prostředí

Mobile application development in a block-based visual programming language

Jan Vavák

Vedoucí práce:	PhDr. Jiří Štípek, Ph.D.
Studijní program:	Informační technologie se zaměřením na vzdělávání
Studijní obor:	B IT-PG 20

2024

Odevzdáním této bakalářské práce na téma Vývoj mobilních aplikací v blokovém programovacím prostředí 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 10. 7. 2024

Tímto bych rád poděkoval vedoucímu mé bakalářské práce PhDr. Jiřímu Štípkovi, Ph.D. za jeho odborné vedení, užitečné rady a konzultace. Také děkuji PhDr. Tomáši Jeřábkovi, Ph.D. za možnost ověřit praktickou část mé práce.

ABSTRAKT

Práce se zabývá úvodem do problematiky vývoje mobilních aplikací u středoškolských či vysokoškolských studentů pomocí vývojového prostředí MIT App Inventor. V teoretické části popisuje architekturu Android platformy a aplikací. Následně obsahuje volbu daného IDE a jeho stručný popis. Praktická část se skládá z vytvoření adekvátních lekcí a metodiky pro učitele a jejich následného ověření ve výuce. Každá lekce obsahuje metodiku, harmonogram, zadání a řešení úlohy a domácího úkolu. Vyhodnocení probíhá na základě pozorování v hodinách a analýzy splněných domácích úkolů.

KLÍČOVÁ SLOVA

blokové programování, MIT App Inventor, mobilní aplikace, Android

ABSTRACT

The thesis deals with an introduction to the issue of mobile application development for high school or university students using the MIT App Inventor integrated development environment. The theoretical part describes the architecture of the Android platform and applications. Thesis then includes the choice of the IDE and a brief description of it. The practical part consists of the creation of adequate lessons and methodology for teachers and their subsequent validation in the classroom. Each lesson contains a methodology, a timetable, an assignment and a solution of an exercise and homework. Evaluation is based on classroom observations and analysis of completed homework.

KEYWORDS

block-based visual programming, MIT App Inventor, mobile application, Android

Obsah

Úvod

Mobilní zařízení dnes vládnou celému světu. Prostřednictvím mobilních aplikací totiž lidem nabízejí rychle a snadno spoustu užitečných funkcí pro každodenní život. Většina lidí mobilní aplikace používá na denní bázi, ať už se jedná o různé aktuality, sociální média nebo hry. Jednoduše řečeno, jsou součástí života téměř všech lidí a mají v něm zásadní význam. Už jenom z tohoto důvodu by se měli studenti informačních technologií seznámit právě s vývojem mobilních aplikací. Ve výuce je pak potřeba především zohlednit složitost vyvíjených aplikací a způsob vývoje, což bude také záviset na vstupních předpokladech samotných studentů.

Zaměření bakalářské práce jsem si zvolil na základě svých osobních zkušeností s vývojem mobilních aplikací a blokovým programováním ve školní výuce. Vývoj mobilních aplikací v klasických textových prostředích pro mě nebyl ideální. Blokové programování mě naopak velmi zaujalo svou vizualizací různých algoritmických konstruktů. Vývojové prostředí MIT App Inventor, které tyto dvě věci spojuje, pro mě tudíž představovalo velmi zajímavý nástroj.

Bakalářská práce se zabývá úvodem do problematiky vývoje mobilních aplikací u středoškolských či vysokoškolských studentů se základními vědomostmi v oblasti programování. Předkládá sadu lekcí orientovanou na seznámení s vývojem aplikací pro mobilní zařízení v blokových prostředích pomocí vývojového prostředí MIT App Inventor včetně jejich ověření ve výuce. Obsahuje také metodiku pro učitele a potenciální řešení jednotlivých úloh a domácích úkolů.

Záměrem práce je tedy vytvoření a ověření adekvátních lekcí (úloh a domácích úkolů), které budou u studentů rozvíjet základní znalosti a schopnosti spojené s vývojem mobilních aplikací v blokovém prostředí, a vytvoření metodiky pro učitele, která bude popisovat optimální průběh lekcí ve výuce.

1 Cíle práce

Hlavním cílem této bakalářské práce je vytvoření lekcí s úlohami rozvíjející u středoškolských či vysokoškolských studentů znalosti a dovednosti v oblasti vývoje mobilních aplikací pomocí vývojového prostředí s blokovým programováním a jejich následné ověření ve výuce. Úlohy jsou cíleny na studenty se znalostmi a dovednostmi v programování, avšak bez větších zkušeností s vývojem mobilních aplikací či s blokovým programováním.

K naplnění uvedeného hlavního cíle byly stanoveny následující dílčí cíle a úkoly řešení:

V úvodu práce charakterizovat architekturu mobilních aplikací pro operační systém Android. Vysvětlit z jakých komponent se skutečná aplikace skládá a jaké jsou doporučené zásady při vývoji, abychom vytvořili robustní a perspektivní aplikaci.

Vybrat vhodné vývojového prostředí. Zdůvodnit svou volbu zvoleného vývojového prostředí a stručně jej popsat.

Vytvořit řadu lekcí obsahující úlohy zaměřené na vývoj mobilních aplikací prostřednictvím zvoleného vývojového prostředí. K jednotlivým lekcím vytvořit metodiku pro učitele – optimální průběh a harmonogram lekce, včetně vysvětlení konceptu dané úlohy a ukázkového řešení. Součástí každé lekce bude zadání úlohy a povinný domácí úkol.

Lekce ověřit ve výuce na vysoké škole a vyhodnotit jejich reálný průběh a zda došlo k naplnění cílů. Vyhodnocení bude probíhat na základě pozorování v hodinách a analýzy splněných domácích úkolů.

2 Postup řešení

V teoretické části práce se nejprve zaměřuji na základní princip vývoje Android aplikací. Vycházím z obecných pravidel a doporučení přímo z oficiálních materiálů pro Android, kde lze najít vcelku podrobný popis a návod týkající se vývoje. Vybral jsem ty nejdůležitější zásady, které je možné využít v praktické části, přestože MIT AI nám stavbu aplikací a programování podstatně ulehčuje.

Na základě cílů práce a svých zkušeností jsem v další části zdůvodnil výběr vývojového prostředí MIT App Inventor jako prostředku pro část praktickou. Následně jsem popsal jeho nejdůležitější části, ve kterých je potřeba se při tvorbě aplikací orientovat.

Praktická část se skládá z vytvoření lekcí, jejichž počet a náročnost jsou dány časovým prostorem a schopnostmi cílové skupiny. Ke každé lekci jsem vytvořil povinný domácí úkol, jejichž splnění bude součástí závěrečného vyhodnocování. Vytvořil jsem úvodní prezentaci pro představení vývojového prostředí a mobilní aplikace.

Lekce byly následně ověřeny ve výuce během předmětu Edukační programovací jazyky na pedagogické fakultě Univerzity Karlovy. Jednalo se o studenty 2. ročníku jednooborového studijního programu Informační technologie.

Na závěr vyhodnocuji, zda byly lekce vytvořeny a realizovány adekvátně a došlo tak k naplnění cílů práce.

3 Teoretická východiska

3.1 Architektura Android platformy

3.1.1 Softwarová architektura

Softwarová architektura představuje zásadní rozhodnutí v návrhu systému související s jeho celkovou strukturou a chováním. Architektura pomáhá zúčastněným stranám pochopit a analyzovat, jak systém dosáhne základních kvalit, jako je modifikovatelnost, dostupnost a bezpečnost [1].

Mezi nejdůležitější přínosy softwarové architektury patří [2, s. 25-28]:

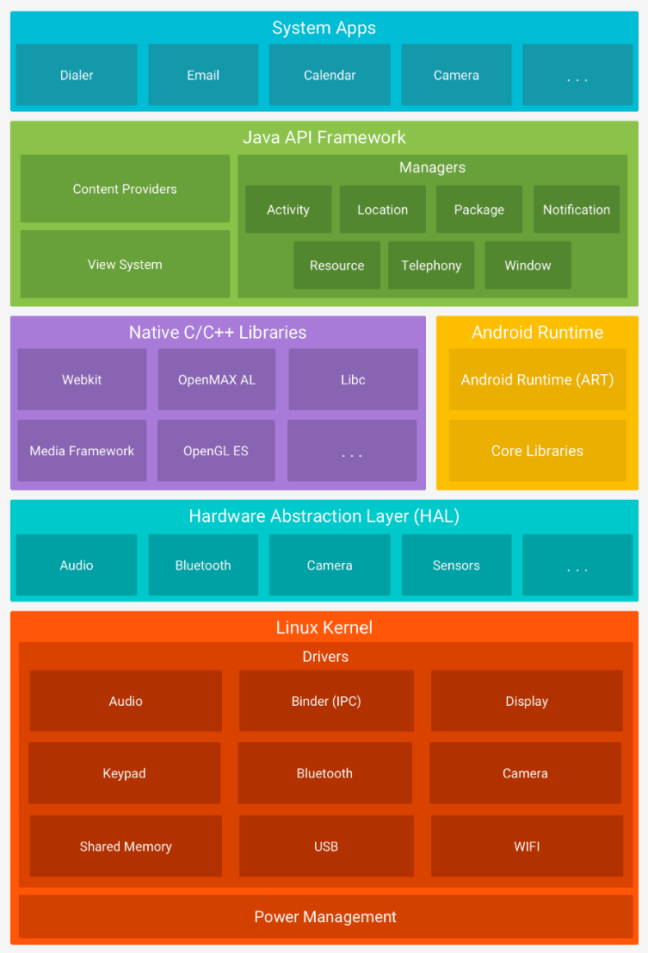
- představuje počáteční obrysy systému, umožňuje analýzu a hodnocení jeho kvality v rané fázi vývoje
- poskytuje omezení pro implementaci systému
- přispívá k opětovnému použití celé architektury či jejích komponent
- usnadňuje komunikaci mezi zúčastněnými stranami
- rozhoduje o tom, jak organizovat členy týmu a přidělovat úkoly

Architektura by měla vývojářům umožnit kvalitní návrh systému, a následně tak zajistit jeho lepší fungování, včetně snazší modifikace a správy. Všechny rozsáhlé a tím pádem složitější systémy, jsou vyvíjeny na základě určité architektury, která popisuje komponenty, ze kterých se skládají, jejich úlohu, vlastnosti a vztahy mezi nimi [3].

3.1.2 Architektura Androidu

Android je open-source platforma na bázi Linuxu určená především pro mobilní zařízení. Jde o jeden z mála operačních systémů podporujících více platforem [4, s. 65]. Jedná se o víceuživatelský systém, kde každá aplikace představuje jiného uživatele [5].

Architektura aplikace je závislá na architektuře operačního systému, na kterém bude používána – aplikace představuje nejvyšší úroveň celé architektury. Pro vývoj aplikací je potřeba znát alespoň základní informace o principech a architektuře operačního systému, ve kterém budou aplikace spouštěny [4, s. 70]. Přehled hlavních komponent Androidu ukazuje *Obrázek 1* [6].



Obrázek 1 - architektura Androidu

Pro vývojáře je pak nejdůležitější vrstvou Java API Framework. Aplikační framework obsahuje v aplikacích opakovaně použitelný software, například ovládací prvky, ikony a podobně. Framework je napsán v Javě a poskytuje aplikacím základní služby systému [4, s. 76-77]. Tyto služby mohou zpřístupňovat data v jiných aplikacích, prvky uživatelského rozhraní, upozorňovací stavový řádek, aplikace běžící na pozadí, hardware používaného zařízení a mnoho dalších služeb a funkcí [7].

3.2 Architektura Android aplikací

3.2.1 Komponenty

Běžná Android aplikace obsahuje několik základních komponent – aktivity (*activities*), fragmenty (*fragments*), služby (*services*), přijímače vysílání (*broadcast receivers*) a poskytovatele obsahu (*content providers*) [8]. Aktivity a fragmenty umožňují uživatelům přes

grafické rozhraní interakci s aplikací (přijímat či odesílat informace). Zbývající tři složky fungují „za oponou“, nemají tedy GUI [4, s. 83-84]. Všechny komponenty aplikace jsou deklarovány v manifestačním souboru nacházejícím se v kořenovém adresáři celého projektu [5].

Aktivita je hlavní třída, která se uživateli zobrazí po spuštění aplikace. Uspořádání aktivit je hierarchické, což znamená, že po spuštění aplikace se spustí nejprve hlavní aktivita, z které je následně v případě potřeby možné spouštět ostatní aktivity. Aktivita by měla být navržena tak, aby umožnila uživateli soustředit se na jednu věc, kterou momentálně potřebuje realizovat [4, s . 83-84].

Fragment představuje opakovaně použitelnou část GUI dané aplikace. Musí být součástí aktivity nebo jiného fragmentu. Rozdělení uživatelského rozhraní na fragmenty usnadňuje úpravu vzhledu aktivity za běhu [9].

Služby realizují déle trvající operace a operace na pozadí. Také umožňují spolupráci se vzdálenými procesy. Na rozdíl od aktivit služby nepotřebují uživatelské rozhraní. Typickým příkladem služby je přehrávání hudby na pozadí.

Objekty na vysílání a přijímání (*broadcast receivers*) poslouchají na pozadí a reagují na události, které se odehrávají na zařízení. Události jsou zastoupeny objekty typu *intent* (záměr). Vydavatelé vytvářejí “záměry” a následně je přes broadcast směrují do vysílání. Zachytávají je přijímače, které mají příslušné záměry objednané nebo registrované. Dobrým příkladem na ilustraci fungování broadcast receivers je přijetí SMS zprávy nebo e-mailu.

Poskytovatelé obsahu (*content providers*) umožňují ukládání a sdílení dat mezi více aplikacemi a procesy. Aplikace tak mohou přistupovat k údajům ostatních aplikací, které vystupují jako poskytovatelé obsahu. Využívá se rozhraní podobné databázovému, poskytovatelé obsahu jsou ale více než jen databáze [4, s. 84].

3.2.2 Základní architektonické principy

S růstem velikosti Android aplikací je o to důležitější definovat takovou architekturu, která aplikaci umožní škálovatelnost, zvýší její robustnost a ulehčí její testování.

Mobilní zařízení mají omezený počet zdrojů, operační systém tudíž může kdykoliv ukončit některé běžící aplikační procesy kvůli uvolnění místa pro procesy nové. Komponenty aplikace tak mohou být spouštěny jednotlivě a v různém pořadí, a operační systém nebo uživatel jsou schopni je kdykoliv ukončit. Jelikož tyto události nejsou zcela v rukách programátora, neměli bychom v komponentách uchovávat žádná data či stav aplikace a komponenty by na sobě neměly být závislé [8].

Nejdůležitější zásadou, kterou je potřeba dodržovat, je tzv. oddělení zodpovědností (*separation of concerns*) – rozdělení počítačového programu na různé části tak, aby se z hlediska funkcionality tyto části co možná nejméně překrývaly [10]. Nelze napsat celý kód pouze do jedné komponenty. Komponenty *Activity* a *Fragment* by měly obsahovat pouze logiku, která se stará o interakce uživatelského rozhraní a operačního systému.

Uživatelské rozhraní by mělo být řízeno stálými datovými modely, které reprezentují data aplikace a jsou nezávislé na prvcích UI a dalších komponentách aplikace. Uživatelé tak nepřijdou o data, když se operační systém rozhodne aplikaci ukončit.

Když v aplikaci definujeme nový datový typ, měli bychom mu přiřadit SSOT (*Single source of truth* – "jediný zdroj pravdy"). Tento zdroj pak data vlastní a jako jediný je může měnit. SSOT toho docílí tím, že data zpřístupní pomocí neměnného typu a k úpravě dat použije funkce či události, které mohou být volány jinými typy. Tato zásada centralizuje změny určitého typu dat a chrání data před manipulací od jiných typů. SSOT může být databáze, ViewModel nebo UI.

S principem SSOT spolupracuje vzor UDF (*Unidirectional Data Flow* – "jednosměrný tok dat"). UDF zajišťuje, aby stavy proudily pouze jedním směrem a události upravující data zase směrem opačným. V operačním systému Android stavy a data obvykle směřují v hierarchii z typů s vyšším rozsahem do typů s nižším rozsahem. Události jsou obvykle spouštěny z typů s nižším rozsahem, dokud nedosáhnou SSOT pro odpovídající datový typ. Aplikační data se většinou pohybují z datových zdrojů do UI a události jako stisknutí tlačítka směřují z UI do SSOT, kde jsou aplikační data upravena a vystavena v neměnném typu. UDF zaručuje větší konzistenci dat a menší náchylnost k chybám [8].

3.2.3 Strukturalizace aplikace

Pro strukturalizaci aplikací existují univerzální doporučení a praktiky, které aplikaci vybaví žádoucími vlastnostmi, avšak ke každé aplikaci je nutné přistupovat individuálně a zohlednit její specifické požadavky.

Mezi obecné praktiky mimo jiné patří:

- Reaktivní a vrstvená architektura
- Jednosměrný tok dat (UDF) ve všech vrstvách aplikace
- UI vrstva s tzv. *state holders* pro správu složitosti UI
- Koprogramy (*coroutines*) a toky (*flows*)
- Oddělení závislostí řešit návrhovým vzorem zvaným vkládání závislostí (*Dependency injection*)

Každá aplikace by měla mít alespoň 2 vrstvy. Vrstvu uživatelského rozhraní zobrazující aplikační data a datovou vrstvu obsahující tzv. *business logic* aplikace a zpřístupňující aplikační data. Pro zjednodušení a znovupoužití interakcí mezi UI a datovou vrstvou lze přidat dodatečnou doménovou vrstvu, která může posloužit jako prostředník mezi těmito dvěma vrstvami [8].

3.3 Volba vývojového prostředí

Vhodný software pro vývoj mobilních aplikací jsem vybíral především na základě svých zkušeností. Zároveň jsem musel vzít v potaz obtížnost a způsob programování, které by měly odpovídat záměrům práce a schopnostem cílové skupiny. Do dalších kritérií lze zařadit jeho vybavenost, dostupnost, cenu či multiplatformitu.

O MIT AI jsem se dozvěděl během studia na předmětu Edukační programovací jazyky a oblíbil jsem si hlavně spojení tvorby grafického uživatelského rozhraní metodou drag-and-drop a blokového programování, které je podle mého názoru jednodušší a zábavnější než klasické textové programování. Současně nabízí mnoho komponent a funkcí, díky kterým lze vytvářet zajímavé a sofistikované aplikace. MIT AI je tak vhodný jak pro začínající, tak pokročilé programátory.

Vývojové prostředí MIT App Inventor mě také zaujalo právě kvůli mým předchozím zkušenostem v oblasti programování. MIT AI je totiž založen na podobném principu

jako .NET frameworky pro tvorbu formulářových aplikací Windows Forms a modernější Windows Presentation Foundation. Oba frameworky slouží k tvorbě aplikací s uživatelským rozhraním, které se skládá z tzv. ovládacích prvků, a následného programování v programovacím jazyce C#.

Pro použití MIT AI programátorovi stačí mít přístup k internetu a Google účet pro přihlášení. Abychom pak na mobilním zařízení spustili vytvořený zdrojový kód, který se ukládá do cloudového úložiště, potřebujeme k propojení mobilní aplikaci MIT AI2 Companion. Zajištění funkčnosti tak není příliš obtížné a tento proces je zcela zdarma. Na oficiálních stránkách také nalezneme podrobnou dokumentaci, mnoho návodů, lekcí a knih.

Vývoj aplikací byl značnou dobu možný pouze pro operační systém Android. V září 2023 byla vydána verze (beta testing) podporující i operační systém iOS, tudíž by aplikace měly fungovat s menšími odchylkami na obou operačních systémech.

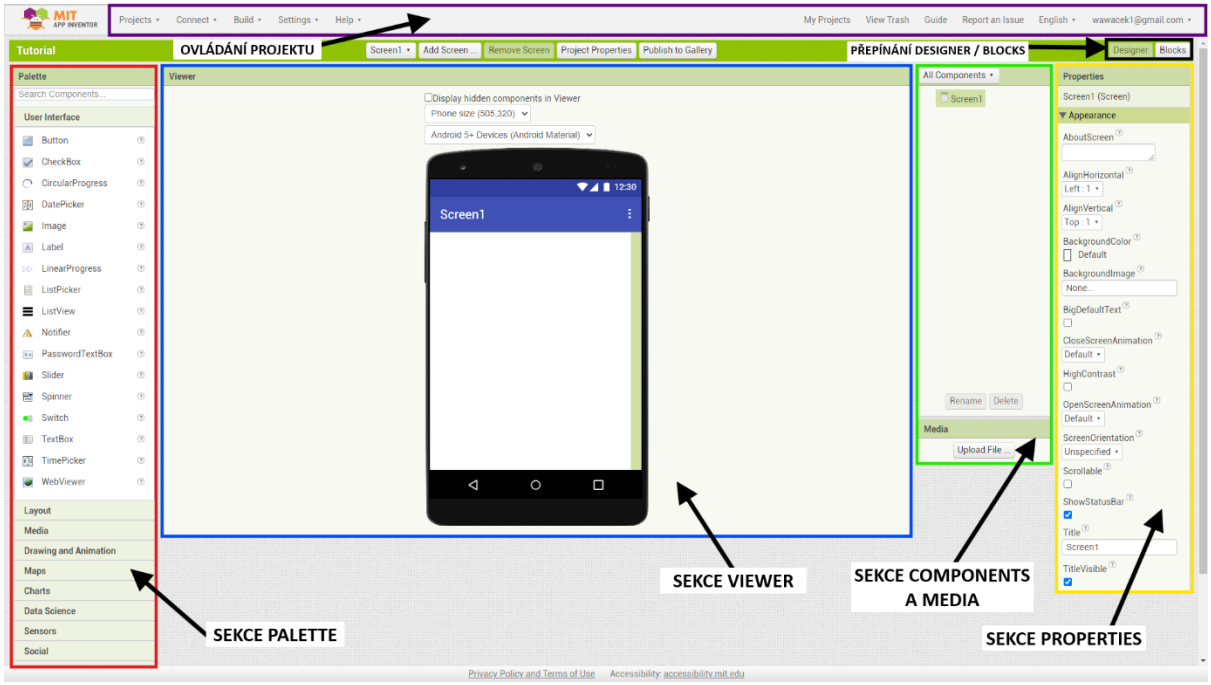
3.4 MIT App Inventor

MIT AI je webové vývojové prostředí (IDE – integrated development environment) umožňující vytvářet aplikace na mobilní zařízení s operačním systémem Android a iOS. Byl vytvořen společností Google, který ho vydal v roce 2010. Následující rok zveřejnil zdrojový kód a předal software do rukou univerzitě MIT, která jej spravuje dodnes. V roce 2013 byl vydán MIT App Inventor 2. MIT AI spolupracuje s různými online službami jako jsou Google Sheets či databáze Firebase [11].

K registraci lze využít možnost zaslání aktivačního kódu na zvolenou emailovou adresu nebo existující Google účet. Po přihlášení se uživatel dostane do svého domovského adresáře, kde lze spouštět a vytvářet projekty. Vytvoření nového projektu spustí IDE, ve kterém se defaultně nachází pouze povinná komponenta *Screen1*. IDE se skládá ze dvou hlavních částí *Designer* a *Blocks*.

Designer je část určena pro stavbu grafického uživatelského rozhraní aplikace pomocí komponent. Komponenty rozdělené do několika kategorií nalezneme v sekci *Palette* a přesouváme je do sekce *Viewer*, která reprezentuje přibližný vzhled mobilní aplikace. Každá komponenta má své specifické vlastnosti, funkce či události. Vlastnosti zvolené komponenty můžeme upravovat přímo v sekci *Properties* nebo následně blokově v části *Blocks*. Dále v

Designeru najdeme sekce *Components* zobrazující hierarchii použitých komponent a *Media* pro nahrávání souborů.



Obrázek 2 - Designer

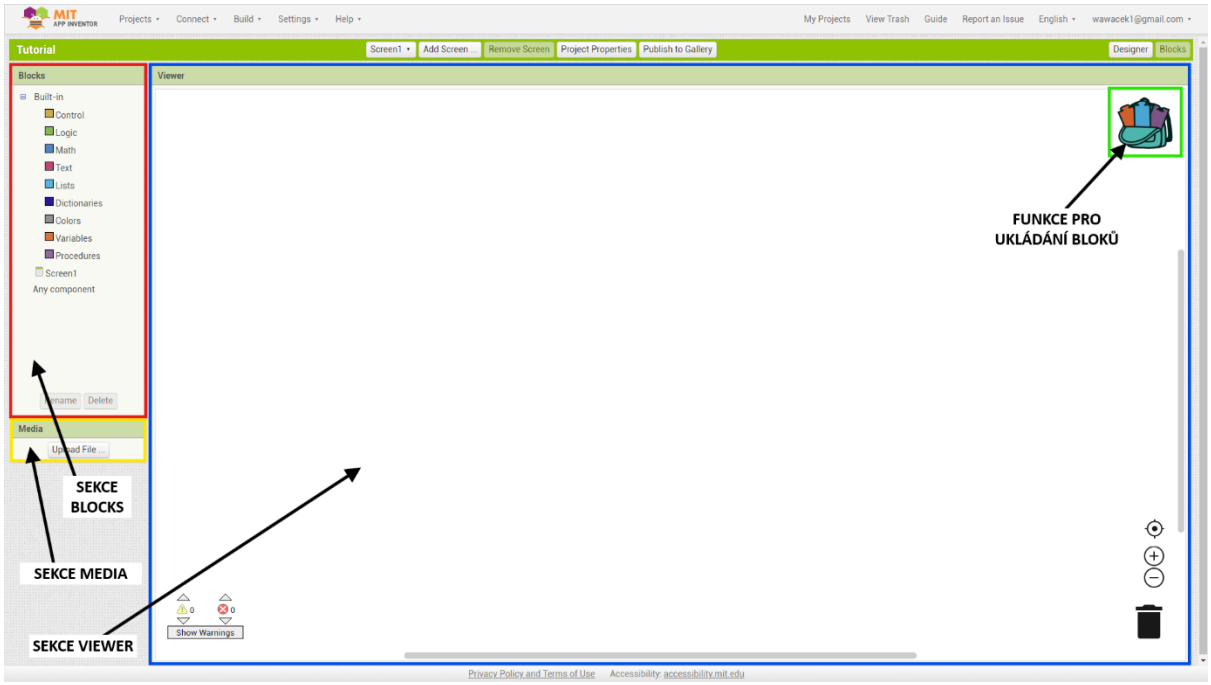
Blocks je část určena pro programování aplikace pomocí bloků. Bloky jsou rozdělené do 3 skupin – vestavěné (*built-in*), používané komponenty (např. *Screen1*) a obecné komponenty (*any component*).

Vestavěné bloky představují běžné algoritmické konstrukty, které se objevují v každém programu – proměnné, podmínky, cykly, kolekce či různé funkce.

V blocích používaných komponent přistupujeme ke konkrétním komponentám, které jsme v části *Designer* vložili do sekce *Viewer*.

V poslední skupině bloků obecných komponent lze vytvářet obecné předpisy v závislosti na používaných komponentách (např. napíšeme kód, který se provede při kliknutí na jakékoliv tlačítko v aplikaci).

Bloky vkládáme do sekce *Viewer* reprezentující zdrojový kód. I v této části najdeme sekci *Media* a nahrané soubory mají svou blokovou podobu. V pravém horním rohu se nachází užitečná funkce pro uložení vytvořených bloků a jejich přesun mezi projekty.



Obrázek 3 - Blocks

Spuštění aplikace lze realizovat celkem 3 způsoby, které najdeme na horní liště pod položkou *Connect*. Aplikaci lze spustit přímo na osobním počítači přes některý z emulátorů (např. BlueStacks X). Dále můžeme připojit mobilní zařízení a nahrát do něj aplikaci přes USB konektor. Nejjednodušší a nejrychlejší možností je AI Companion, který také jako jediný umožňuje real-time testování a ladění. Dané mobilní zařízení propojíme s projektem aplikací MIT AI2 Companion. Po vybrání volby AI Companion se vygeneruje příslušný QR kód, který mobilním zařízením naskenujeme, případně lze využít textový kód.

4 Navržené lekce

Všechny potřebné materiály pro učitele k jednotlivým lekcím jsou k dispozici v elektronické příloze této práce v SISu. V příloze lze nalézt všechna zadání úloh a domácích úkolů, materiály pro 3. a 4. lekci, úvodní prezentaci a aktualizovanou metodiku. Sekce příloha také obsahuje odkazy na konkrétní projekty v galerii MIT AI, které si lze stáhnout a spustit na vlastním profilu.

4.1 Vstupní předpoklady

Vyučující má minimálně pokročilé znalosti MIT AI.

Studenti by pro náležité porozumění lekcím a plnění úloh měli být vybaveni určitými vědomostmi a dovednostmi. Studenti by měli znát:

- základní a složené algoritmické konstrukce (posloupnost příkazů, podmínky, cykly)
- datové typy (druhy, použití, odlišnosti)
- kolekce, metody, třídy (základní přehled)
- podstatu objektově orientovaného programování

Všichni studenti budou mít ve výuce k dispozici počítač pro práci v MIT AI a vlastní či školní mobilní zařízení pro spuštění aplikace v MIT AI2 Companion.

4.2 Přehled pojmů

S vývojovým prostředím MIT AI a obecně s programováním se pojí určitá terminologie. Pro snadnější pochopení lekcí je proto důležité rozumět pojmům, které se v nich objevují. Tato kapitola obsahuje jejich stručnou definici.

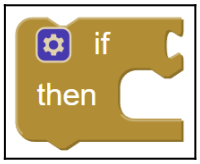
4.2.1 Komponenty

Objekty sloužící k tvorbě grafického uživatelské prostředí aplikace. Lze dělit na viditelné (*Button, TextBox, Label* apod.) a neviditelné (*Sound, Clock, Player* apod.). Komponenty mají své vlastnosti (*TextBox.Text*) a funkce (*Sound.Play*) nebo mohou vyvolávat události (*Button.Click*). V MIT AI jsou pak tyto součásti v blokové podobě barevně odlišeny: události – žlutá, funkce – fialová, vlastnosti – zelená.

Obrázek 4 - komponenta

4.2.2 Bloky

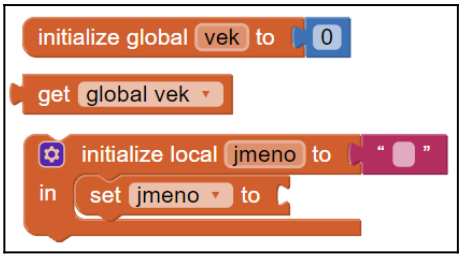
Základní stavební jednotka v programovací části. Jde v podstatě o objekty, ze kterých je složen kód aplikace. Jsou rozděleny do různých kategorií – kontrolní, logické, textové atd. Za bloky lze vlastně považovat vše, z čeho se kód aplikace skládá – základní algoritmické konstrukty, proměnné, kolekce, funkce či součásti komponent.



Obrázek 5 - blok

4.2.3 Proměnné

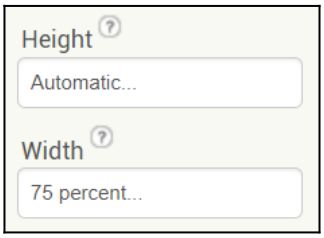
Bloky uchovávající určitá data. Blokový programovací jazyk je v MIT AI dynamicky typovaný, tudíž u proměnných nemusíme explicitně definovat datový typ – mohou nabývat různých druhů hodnot. Hodnotu proměnných můžeme číst (*get*) či nastavovat (*set*). Lze dělit na globální a lokální proměnné. Globální proměnné jsou dosažitelné v celém kódu, lokální proměnné jsou dosažitelné pouze v bloku, ve kterém byly vytvořeny.



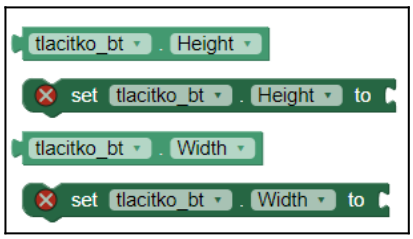
Obrázek 6 - proměnné

4.2.4 Vlastnosti

Proměnné popisující komponenty. Buď popisují jejich vzhled (*FontSize, Height, Width, Text atd.*) nebo chování (*Enabled, MultiLine, ReadOnly atd.*). K vlastnostem lze přistupovat jak v části *Designer*, tak v části *Blocks*.



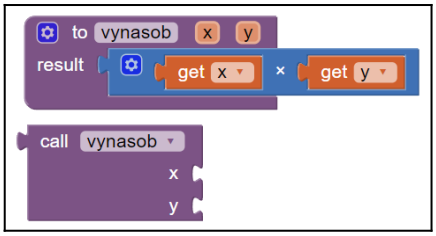
Obrázek 7 - vlastnosti 1



Obrázek 8 - vlastnosti 2

4.2.5 Funkce

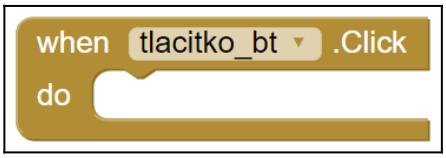
Bloky obsahující určitý kód, který se při jejich zavolání vykoná. Rozlišujeme funkce s / bez parametru nebo s / bez návratové hodnoty. Parametr je hodnota, která se funkci předá při zavolání. Návratová hodnota je hodnota, která je funkcí, zpravidla po nějakém výpočtu, předána k dalšímu použití. Lze používat funkce vlastní nebo již existující, které jsou vázány na komponenty.



Obrázek 9 - funkce

4.2.6 Události

Bloky reprezentující různé akce, které mohou nastat a po nichž většinou proběhne určitý kód. Jsou vždy vázány na nějakou komponentu, která nabízí způsob jejich uskutečnění. Například tlačítko, jehož událostí může být stisknutí.



Obrázek 10 - událost

4.3 Úvodní prezentace

Tato kapitola vysvětluje záměr úvodní prezentace a co vše by při ní měl učitel zmínit.

Úvodní prezentace (viz Prezentace1_Uvod.pptx) je součástí první lekce. Představuje software (MIT AI a MIT AI2 Companion), který bude používán k vývoji mobilních aplikací. Prezentace obsahuje stručný popis nejdůležitějších částí, ve kterých je nezbytné se při práci orientovat. Zaměřuje se především na vývojové prostředí MIT AI, jelikož mobilní aplikace MIT AI2 Companion nenabízí nijak složité uživatelské rozhraní.

MIT AI je rozdělen na 3 hlavní části – navigační lišta, *Designer* a *Blocks*. Z každé části jsou v prezentaci vybrány pouze nejpodstatnější údaje, které budou studenti při vývoji potřebovat. Učitel by měl každou část stručně popsat a zmínit jakým způsobem ji při vývoji budou používat.

Navigační lišta slouží k ovládání projektu a přesouvání mezi částmi *Designer* a *Blocks*. Mezi další významné položky patří *Projects*, *Connect* a *My Projects*.

Dále učitel popíše hlavní význam části *Designer* a jeho 4 sekce *Palette, Viewer, Components* + *Media* a *Properties*. Vysvětlí přidání komponent do sekce *Viewer* a přehled jejich vlastností v sekci *Properties*.

V části *Blocks* vysvětlí kategorizaci bloků na *Built-in, Screen1* a *Any component*. Zmíní také funkci *Backpack* pro přenos bloků mezi více projekty.

Nakonec krátce představí MIT AI2 Companion a požádá studenty, aby si tuto aplikaci stáhli na svá mobilní zařízení.

4.4 Seznam lekcí s úlohami

4.4.1 Lekce 1 – Barvy

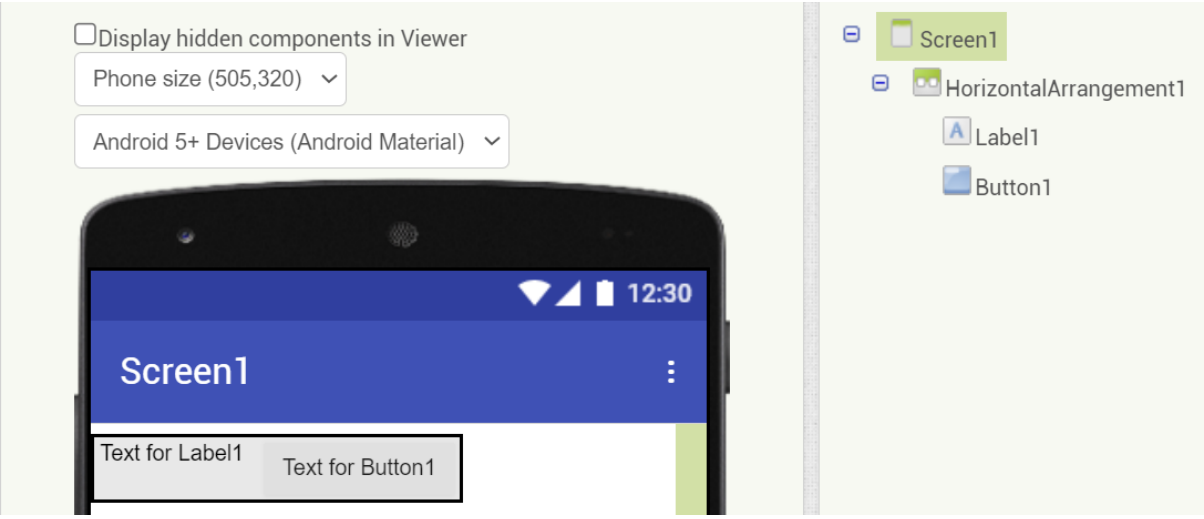
První lekce by měla být samozřejmě koncipována jako úvodní, seznamovací. Učitel studentům stručně představí vývojové prostředí MIT AI, ukáže jeho nejdůležitější části pro základní orientaci, a nakonec se s ním studenti blíže seznámí prostřednictvím první úlohy.

Smyslem lekce je naučit studenty základům MIT AI jako vytvoření nového projektu, orientace ve vývojovém prostředí, přidávání, uspořádání a práce s komponenty a pochopení principu programování v blokovém prostředí.

Průběh lekce

Učitel nejprve představí a popíše jednotlivé části vývojového prostředí za doprovodu úvodní prezentace (viz soubor Prezentace1_Uvod.pptx v příloze). Na závěr prezentace vybídne studenty k nainstalování mobilní aplikace MIT AI2 Companion, kterou si studenti stáhnou na svá mobilní zařízení. Výhodou aplikace je, že není nutné nic nastavovat. Ihned po instalaci je připravená k použití. Učitel následně studentům ukáže, jak se zaregistrovat do MIT AI a vytvořit nový projekt. Studenti jeho postup následují na svých počítačích.

Dále učitel společně se studenty vyzkouší MIT AI v praxi. Postačí jednoduchá ukázka – přidání komponent *Button* a *Label*, zkouška vlastností, přístup ke komponentám v blokové části a naprogramování změny textu či barvy po stisknutí tlačítka. Učitel by měl na úvod také zmínit a předvést komponenty z kategorie *Layout* pro uspořádání dalších komponent do logických celků především kvůli zajištění větší konzistence či úpravě vzhledu aplikace. *Layout* komponentami není potřeba se příliš zdržovat, jelikož jsou vcelku intuitivní a nikdo by neměl mít obtíže s jejich pochopením. Nakonec učitel předvede propojení MIT AI a MIT AI2 Companion.



Obrázek 11 - vzhled úvodní aplikace

Obrázek 12 - kód úvodní aplikace

Následně učitel zadá první část úlohy Barvy (viz část níže *Zadání úlohy a domácího úkolu*). Studenti se nejdříve musí zamyslet nad tím, jaké všechny komponenty bude potřeba použít. To může proběhnout v rámci společné diskuze.

Je vhodné studenty informovat, že MIT AI již obsahuje funkci pro skládání barev, jde tedy pouze o zvolení správných komponent a programovacích bloků.

Ze začátku možná někteří studenti budou potřebovat poradit. Učitel jim může objasnit koncept, na algoritmus by však měli přijít sami.

Učitel zadá studentům k první lekci domácí úkol na podobném principu jako je úloha ve výuce, díky kterému si studenti více upevní a prohloubí znalosti o dané problematice.

Pokud studenti nestihnou všechny části úlohy, lze jejich dodělání zadat také jako domácí úkol. To platí i u dalších lekcí.

Časový harmonogram

Čas	Činnost

10 minut	Učitel za doprovodu úvodní prezentace představí MIT App Inventor, demonstruje přihlášení a vytvoření nového projektu.
5-10 minut	Ukázka tlačítka + *Layout* komponent, propojení a spuštění aplikace na mobilním zařízení.
25 minut	Učitel zadá první část úlohy Barvy – generování náhodných barev. Při včasném dokončení se pokračuje další částí úlohy.
5-10 minut	Kontrola řešení studentů a vysvětlení potenciálního provedení.

Tabulka 1 - harmonogram Lekce 1

Zadání úlohy a domácího úkolu

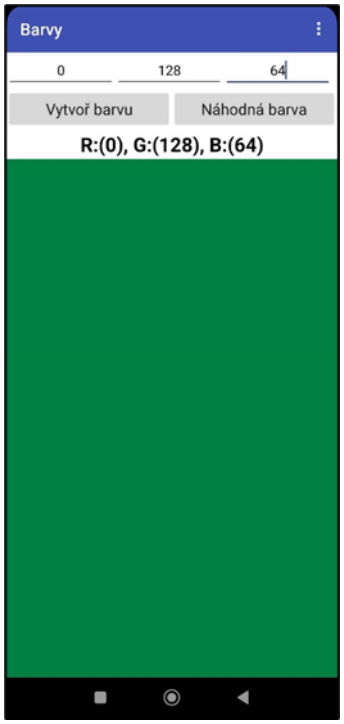
Část 1

Vytvořte aplikaci, která bude po stisknutí tlačítka generovat náhodné barvy, vygenerované hodnoty budou zobrazeny a dojde ke změně barvy pozadí.

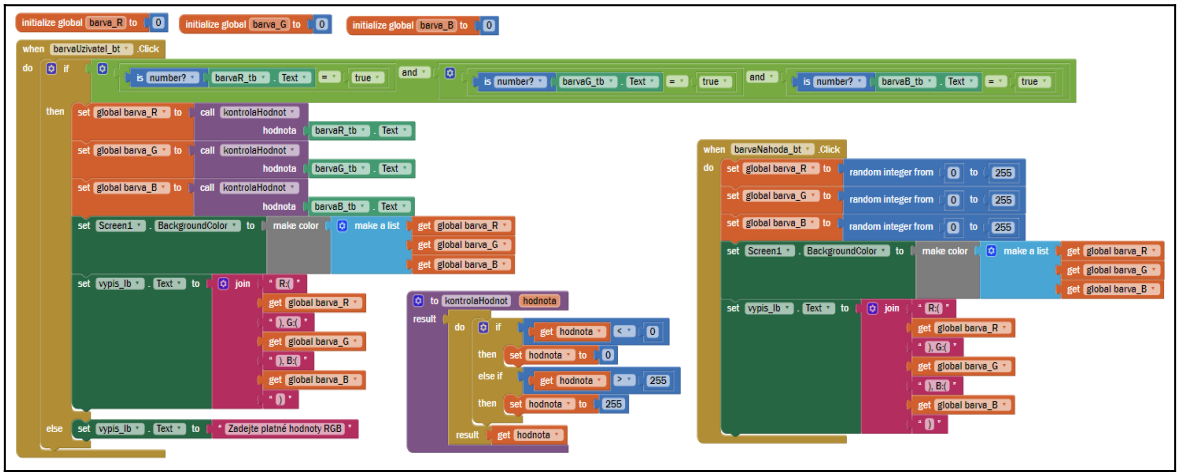
Část 2

Další tlačítko bude po stisknutí generovat barvy na základě vstupních hodnot zadaných uživatelem prostřednictvím textových polí. Konkrétně uživatel zadá tři hodnoty v rozmezí 0-255 reprezentující tři složky barevného modelu RGB a po stisknutí tlačítka se barva pozadí obrazovky změní na danou barvu.

Zabezpečte aplikaci proti nežádoucím stavům. Uživatel totiž může vyvolat chybu při zadání nesprávných hodnot – namísto čísel zadá text nebo nezadá všechny tři hodnoty.



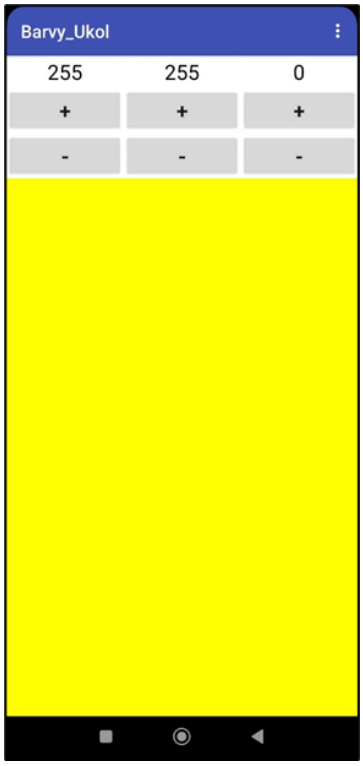
Obrázek 13 - vzhled úlohy Lekce 1 (obě části)



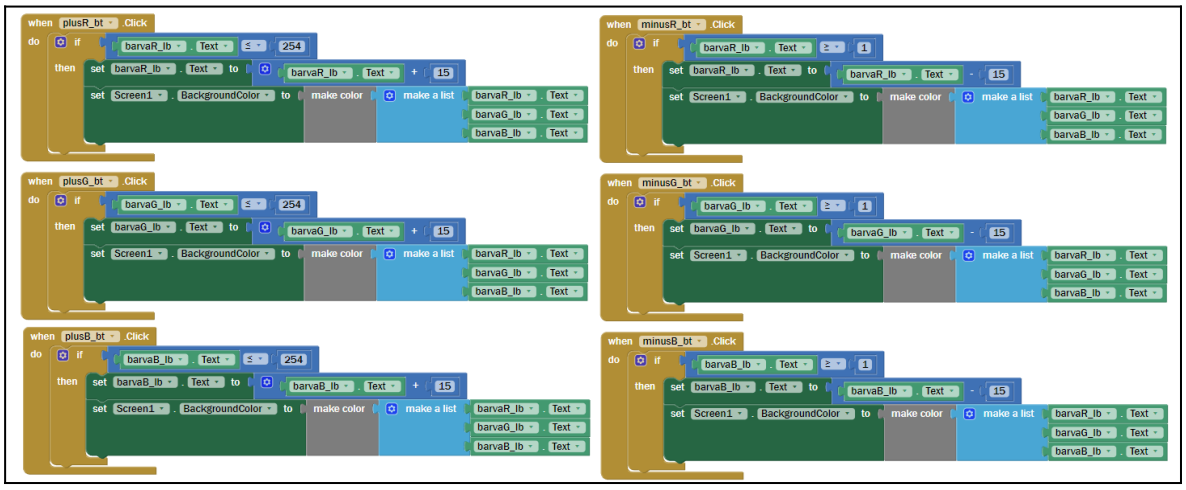
Obrázek 14 - kód úlohy Lekce 1 (obě části)

Povinný domácí úkol

Vytvořte aplikaci, která bude opět měnit barvu pozadí v závislosti na třech hodnotách. Hodnota každé barevné složky bude zobrazena a uživatel ji bude ovládat dvojicí tlačítek, která ji zvýší či sníží o 15. Barva pozadí se aktualizuje při každé změně hodnot. Uživatel bude moct pracovat pouze v rozmezí 0-255.



Obrázek 15 - vzhled úkolu Lekce 1



Obrázek 16 - kód úkolu Lekce 1

Návrh a řešení úlohy

Použité komponenty:

- 1x *Label* – výpis RGB hodnot
- 3x *TextBox* – zadání vstupních hodnot

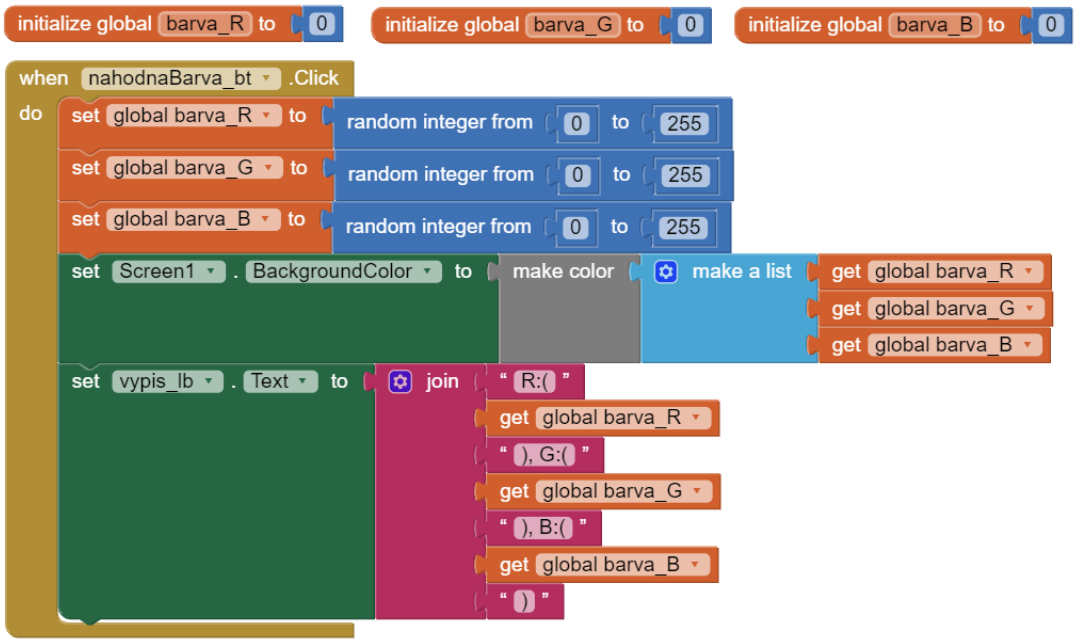
- 2x *Button* – vytvoření a zobrazení barvy

První úloha by měla studentům ukázat na jakém principu MIT AI funguje. Studenti si vyzkouší, jakým způsobem se tvoří uživatelské rozhraní pomocí přidávání základních komponent (*Button, TextBox* a *Label*) a *Layout* komponent pro jejich uspořádání. V programovací části se seznámí s bloky jako jsou základní algoritmické konstrukty, proměnné, funkce či události a sestaví kód pro změnu barvy a výpis hodnot.

Při pouhé změně barvy pozadí na náhodnou stačí vložit komponentu *Button* (tlačítko) do výchozí komponenty *Screen1*, která je obligátní pro všechny projekty, a využít typické události pro tlačítka *Click*, která se provede po stisknutí daného tlačítka, funkce *make color* a vlastnosti *Screen1.BackgroundColor.* Do funkce *make color*, která přijímá až čtyři číselné parametry (složky RGB + alfa kanál), umístíme pro každou barevnou složku funkci *random integer from* 0 *to* 255 pro generování náhodného čísla v rozmezí 0-255. Při každém stisknutí tlačítka se tedy vygenerují nové hodnoty.

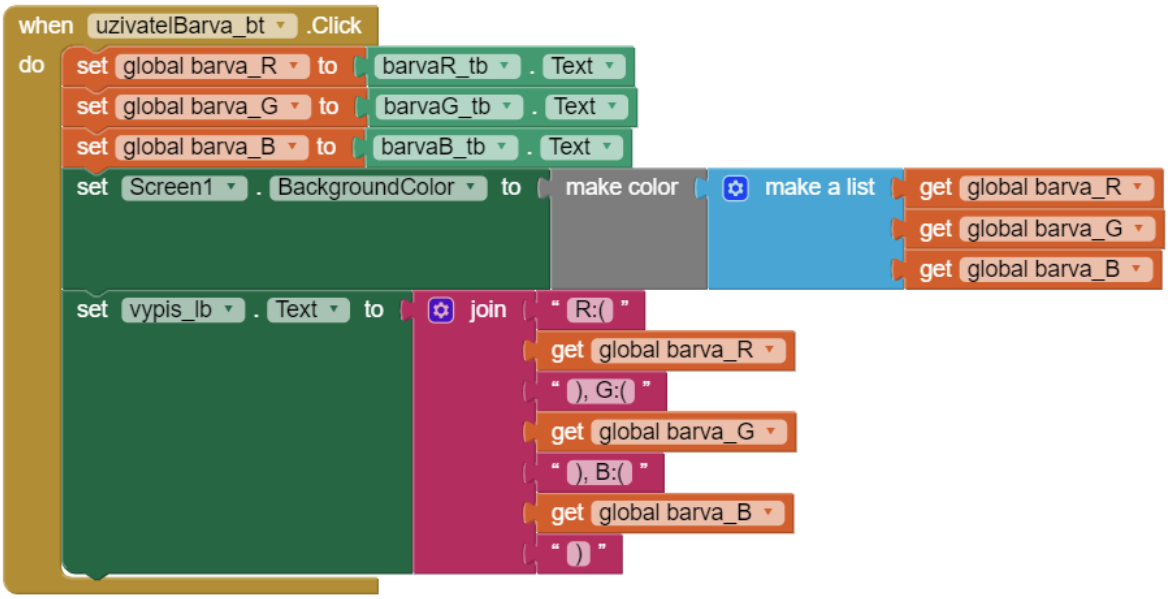
Obrázek 17 - generování náhodné barvy

Jestliže však chceme vygenerované hodnoty zároveň vypisovat na obrazovce, musíme je uložit do proměnných. Nejjednodušší způsob je deklarovat a inicializovat tři globální proměnné, které následně využijeme i ve druhé části úlohy. Při každém kliknutí uložíme do proměnných nové hodnoty a zobrazíme je výpisem proměnných. Pro výpis použijeme komponentu *Label* a její vlastnost *Text*, do které textovou funkcí *join* vložíme proměnné. Pro přehlednost lze ve funkci *join* přidat před každou proměnnou název barevné složky.



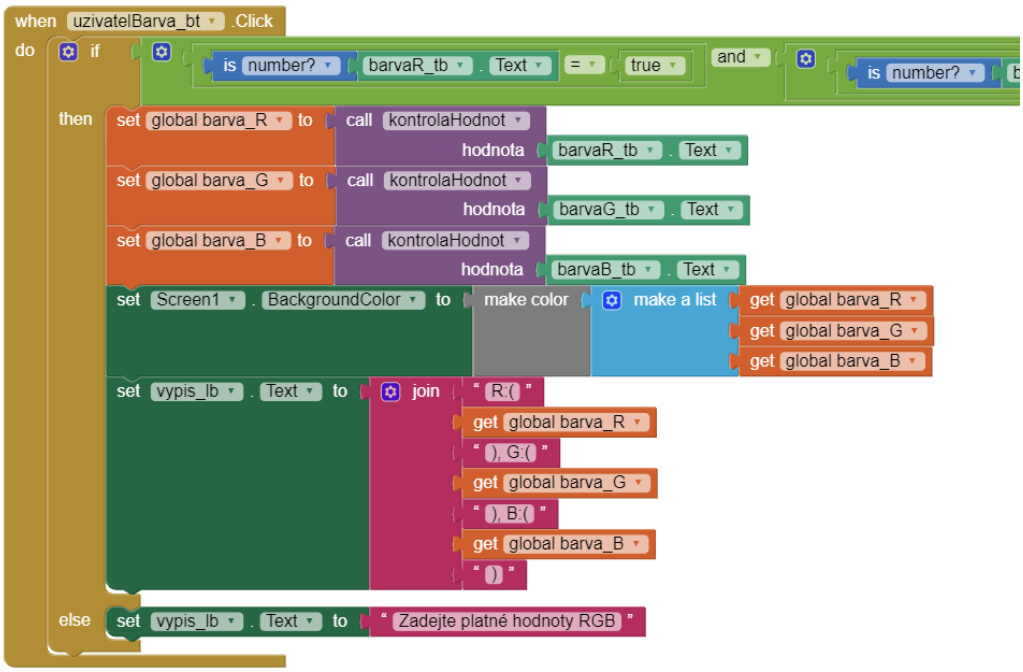
Obrázek 18 - generování a výpis náhodné barvy

Změna barvy na základě vstupních hodnot od uživatele již bude obnášet další komponentu, která umožňuje zadání hodnot prostřednictvím klávesnice. Ideální komponentou pro tuto činnost je *TextBox* (textové pole), do kterého lze na rozdíl od *Labelu* zadávat text a editovat ho. Nejjednodušší způsob řešení je použít textová pole tři – pro každou barevnou složku jedno. Přidáme další tlačítko, při jehož stisknutí se hodnoty z textových polí, vlastnosti *Text,* uloží do zmiňovaných globálních proměnných stejně jako předtím náhodné hodnoty, proměnné stejným způsobem dosadíme do funkce *make color* a poté vypíšeme.



Obrázek 19 - generování a výpis zadané barvy

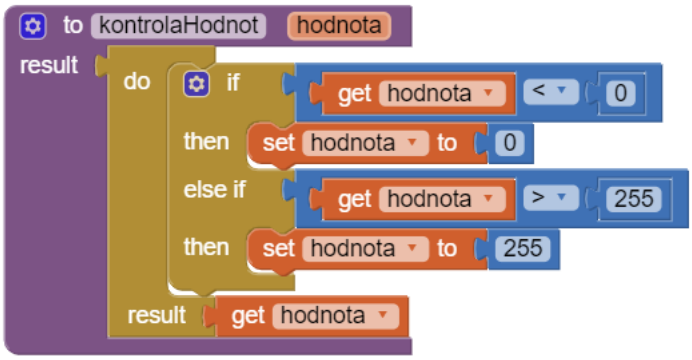
Možnost zadat hodnoty skrz textová pole se sebou ovšem také přináší způsoby, jak v naší aplikaci vyvolat různé výjimky. Proto musíme zavést patřičná opatření přímo na vstupu nebo v kódu aplikace. Zaškrtnutí vlastnosti *NumbersOnly* u textových polí zajistí, aby uživatel mohl zadávat pouze čísla, respektive se při zadávání otevře číselná klávesnice. Avšak na této klávesnici zpravidla bývají i jiné znaky než číselné (např. tečka). Tento nedostatek už musíme vyřešit přidáním podmínky do našeho kódu. Lze využít funkci *is number?*, která zkontroluje, zda je hodnota číslo. Změnu barvy pozadí a výpis hodnot tedy provedeme, pouze pokud budou všechny tři hodnoty číslo. Jinak do komponenty *Label* vypíšeme hlášku pro upozornění uživatele.



Obrázek 20 - kontrola hodnot

Někoho ze studentů může napadnout, jak bude aplikace fungovat v případě, že uživatel zadá hodnoty mimo požadovaný rozsah 0-255. Implicitně funguje funkce *make color* na principu zbytku po dělení (*výsledek* = *parametr* % 256). Když za *parametr* dosadíme čísla v rozmezí 0-255, bude výsledek odpovídat zadanému parametru. Nicméně dosadíme-li čísla menší či vyšší, výsledek přestává být tak jednoznačný (-1 % 256 = 255; 256 % 256 = 0).

Řešením může být vytvoření vlastní funkce pro kontrolu vstupních hodnot, kterou budeme volat při jejich ukládání do proměnných. V MIT AI lze tyto bloky najít v kategorii *Procedures* . Zvolíme typ s návratovou hodnotou a přidáme jeden parametr, který bude funkce vyžadovat. Samotná funkce není nijak složitá, všechny čísla menší než 0 převede na 0, a naopak všechny čísla vyšší než 255 převede na 255, čímž zabráníme, aby se do proměnných uložily nežádoucí hodnoty.



Obrázek 21 - funkce pro úpravu hodnot

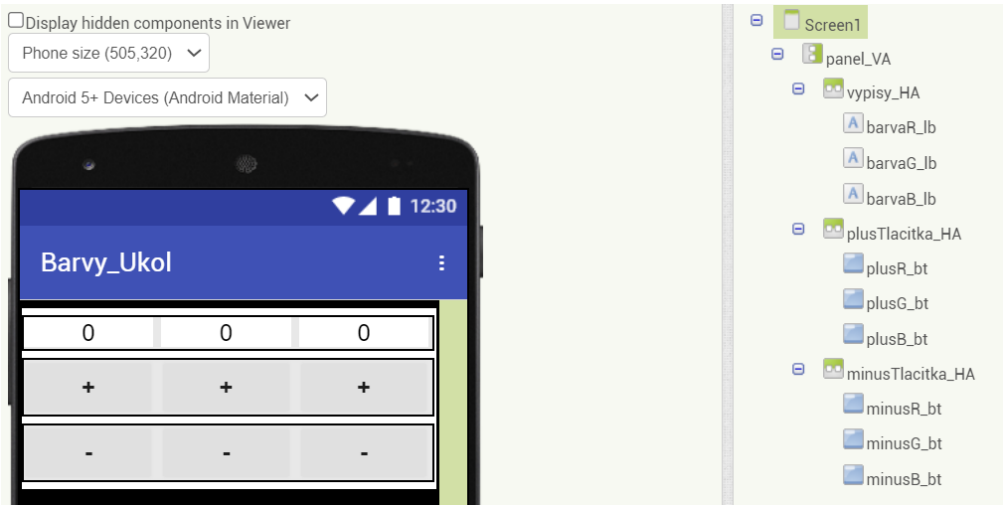
Návrh a řešení domácího úkolu

Použité komponenty:

- 3x *Label* – výpis RGB hodnot
- 6x *Button* – inkrement a dekrement hodnoty

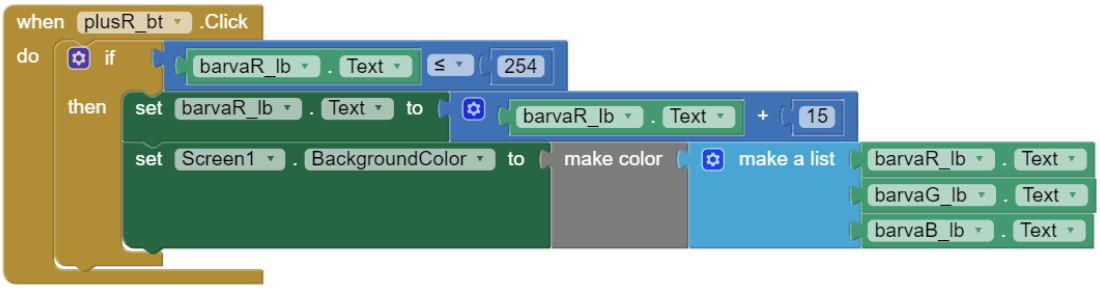
V novém projektu vytvoříme aplikaci složenou z komponent *Label* a *Button*. V tomto případě postačí *Label*, jelikož budeme hodnoty pouze vypisovat. Nemusíme vytvářet ani žádné proměnné, budeme se řídit vlastnostmi *Label.Text*, které defaultně nastavíme na nulu.

Pro každou barevnou složku přidáme dvojici tlačítek starající se o inkrement a dekrement její hodnoty. Horní tlačítka budou *Label.Text* navyšovat o 15 a spodní tlačítka naopak snižovat o 15. Každé tlačítko tedy aktualizuje hodnotu dané barevné složky a hned na to přepočítá barvu pozadí.



Obrázek 22 - vzhled GUI aplikace

Požadovaný rozsah hodnot zajistíme jednoduchou podmínkou, která kontroluje stávající hodnotu vlastnosti *Label.Text*. Tlačítka pro inkrement monitorují horní hranici a tlačítka pro dekrement spodní hranici. Když by hodnota měla opustit rozsah 0-255 kód se neprovede.



Obrázek 23 - inkrement červené barevné složky

Obrázek 24 - dekrement červené barevné složky

4.4.2 Lekce 2 – Stopky

Nyní by se studenti měli dokázat ve vývojovém prostředí orientovat a chápat podstatu vývoje aplikací prostřednictvím grafického uživatelského rozhraní a blokového programování.

Druhá lekce staví na vědomostech a dovednostech získaných v první lekci a zároveň je rozšiřuje implementací nových komponent a bloků. Úloha cílí zejména na práci s časem, přičemž se studenti také naučí vytvářet a volat vlastní funkce.

Průběh lekce

Učitel nejdříve studentům nastíní téma lekce – tvorba stopek. Ideu stopek zná pravděpodobně každý, k tomu se studenti mohou inspirovat těmi na svém mobilním telefonu. Lze také popsat vzhled hotové aplikace.

Učitel nejprve představí komponentu *Clock*, vysvětlí její princip, dostupné vlastnosti (*TimerAlwaysFires*, *TimerEnabled* a *TimerInterval*) a bloky (*Timer, Duration, DurationTo*). Následně ukáže studentům, jak vytvářet a volat vlastní funkce (bloky z kategorie *Procedures*). Postačí stručná ukázka, při které učitel stručně vysvětlí rozdíl mezi bloky deklarace a volání funkce, rozdíl mezi funkcemi bez a s návratovou hodnotou a jak přidávat parametry.

Učitel zadá první část úlohy Stopky (viz část níže *Zadání úlohy a domácího úkolu*). Studenti by měli být sami schopni vytvořit GUI. Pokud studenti nebudou vědět jakým způsobem postupovat, může učitel navrhnout rozdělení vývoje na jednotlivé části – naprogramování ovládacích tlačítek, vytvoření hlavní funkce a vytvoření kontrolních funkcí.

Časový harmonogram

Čas	Činnost
10-15 minut	Zadání a analýza úlohy Stopky. Učitel představí a vysvětlí, jak vytvářet vlastní funkce a jak zacházet s komponentou *Clock*.
25 minut	Studenti pracují na své aplikaci individuálně, případně se radí s učitelem. Stavba aplikace. Programování tlačítek, hlavní funkce a

	kontrolních funkcí.
5-10 minut	Shrnutí a reflexe lekcí *Barvy* a *Stopky*. Zadání povinných domácích úkolů k úlohám *Barvy* a *Stopky*.

Tabulka 2 - harmonogram Lekce 2

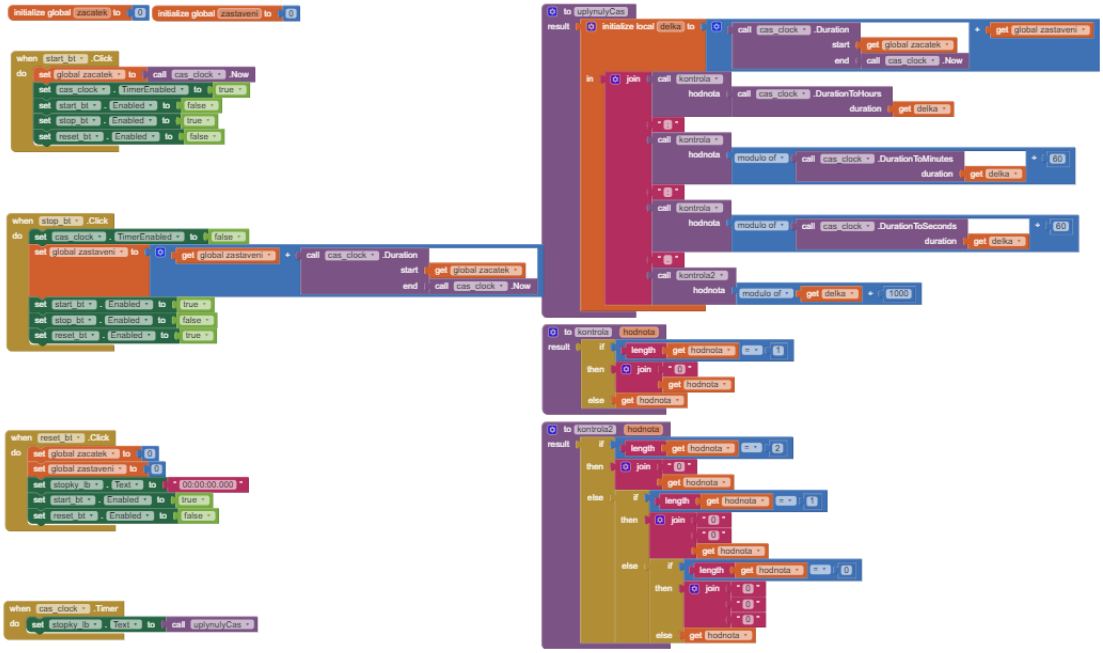
Zadání úlohy a domácího úkolu

Část 1

Vytvořte stopky. Aplikace bude umět měřit uplynulý čas, zastavit a restartovat jej. Po stisknutí prvního tlačítka začnou stopky měřit čas. Další tlačítko čas zastaví, přičemž při opětovném stisknutí prvního tlačítka bude čas pokračovat od bodu zastavení. Poslední tlačítko vše resetuje.

Vytvořte kontrolní funkce pro úpravu jednotek času do formátu "*hodiny:minuty:sekundy.milisekundy (00:00:00.000)*", které budou v závislosti na délce hodnoty doplňovat určitý počet 0. Např. 1 minuta = 01, 1 milisekunda = 001.

Obrázek 25 - vzhled úlohy Lekce 2

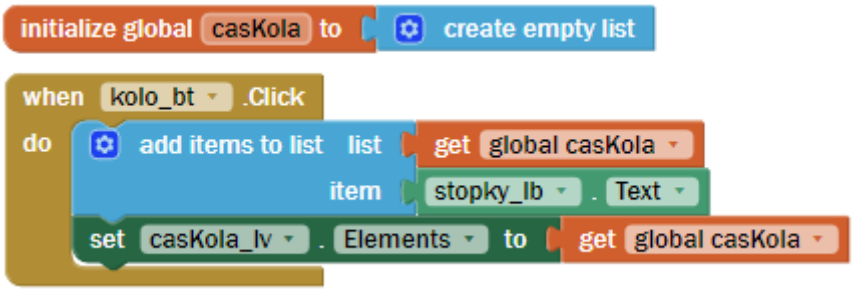


Obrázek 26 - kód úlohy Lekce 2

Povinný domácí úkol

Přidejte do aplikace možnost zaznamenání kola pomocí dalšího tlačítka a komponenty *ListView*. Po stisknutí tlačítka se zaznamená aktuální čas a vypíše se na obrazovce, zatímco stopky běží dál. Tuto funkci může uživatel použít libovolně mnohokrát. Záznamy se vymaží při stisknutí resetovacího tlačítka.

Obrázek 27 - vzhled úkolu Lekce 2

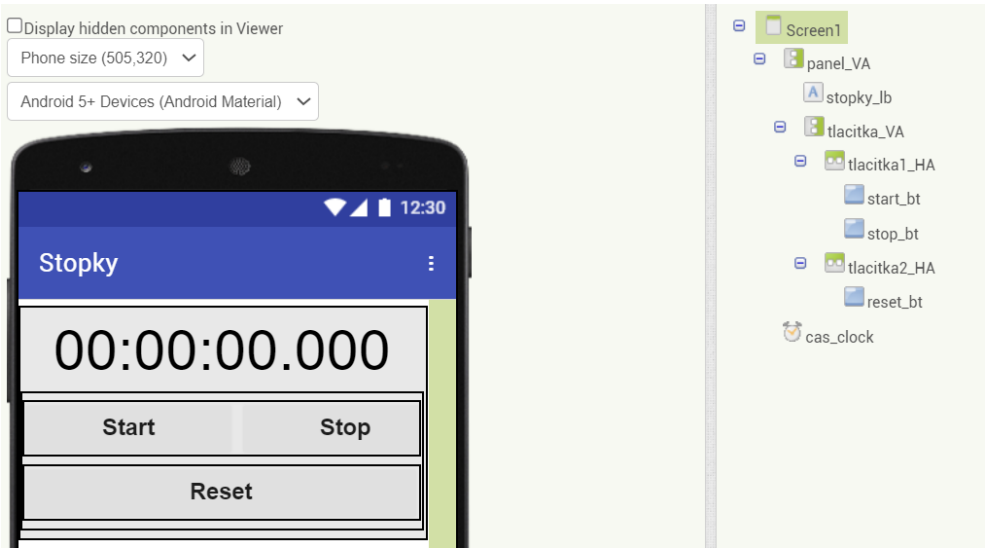


Obrázek 28 - kód úkolu Lekce 2

Návrh a řešení úlohy

Použité komponenty:

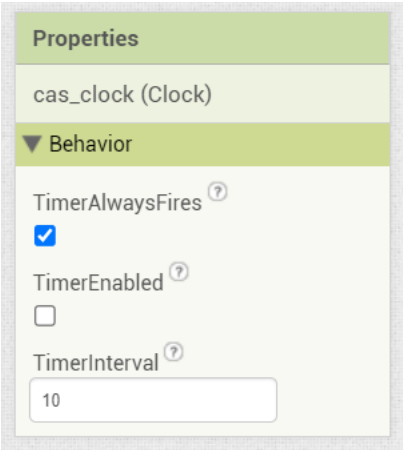
- 1x *Label* – výpis času
- 3x *Button* – start, stop a reset času
- 1x *Clock* – měření času



Obrázek 29 - vzhled GUI aplikace

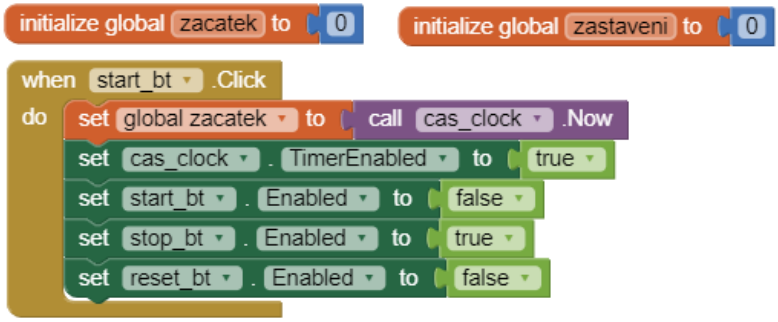
Koncept úlohy je jednoduchý – vytvořit stopky. Stěžejní roli v této úloze hraje komponenta *Clock* dostupná v kategorii *Sensors*. Jedná se o neviditelnou komponentu spolupracující s vnitřními hodinami zařízení, která dokáže periodicky spínat časovač.

U nejdůležitější komponenty *Clock* lze nastavovat pouze tři vlastnosti – *TimerAlwaysFires*, *TimerEnabled* a *TimerInterval*. *TimerEnabled* defaultně vypneme, tudíž po spuštění aplikace nebude *Clock* ihned fungovat (stopky nepoběží). *TimerInterval* udávající v milisekundách dobu, po jaké opakovaně nastane událost *Clock.Timer*, nastavíme na hodnotu 10. To znamená, že každých 10 milisekund se provede kód nacházející se ve zmiňované události. Právě tuto událost použijeme pro výpis uplynulého času.



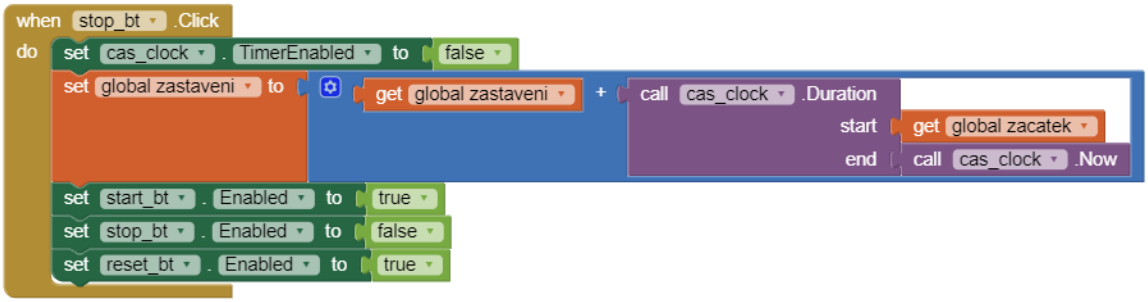
Obrázek 30 - vlastnosti komponenty Clock

Pro ovládání času využijeme celkově 3 tlačítka. Při spuštění aplikace bude k dispozici pouze startovací tlačítko, které bude spouštět stopky. Po jeho stisknutí povolíme *TimerEnabled*, tudíž uvedeme do chodu událost *cas_clock.Timer* a do globální proměnné *"zacatek"* uložíme stávající čas funkcí *Now*. Tlačítka budou zároveň upravovat dostupnost ostatních tlačítek pomocí vlastnosti *Enabled*. Startovací tlačítko samo sebe znepřístupní, aby bylo zamezeno nežádoucímu stisknutí ze strany uživatele.



Obrázek 31 - startovací tlačítko

Stisknutí startovacího tlačítka zpřístupní stopovací tlačítko, které bude stopky zastavovat, tím že se vlastnost *TimerEnabled* nastaví na *false*. Bude zpřístupněno resetovací tlačítko a startovací tlačítko, které by ale nyní mělo způsobit, že se stopky rozběhnou od místa zastavení. To zařídíme další globální proměnnou *"zastaveni"* pro ukládání celkového uplynulého času. Při každém stisknutí stopovacího tlačítka do této proměnné přičteme uplynulý čas. K tomu poslouží funkce *Duration*, která akceptuje dva parametry, počáteční a konečnou instanci času, a vrací rozdíl mezi nimi v milisekundách.



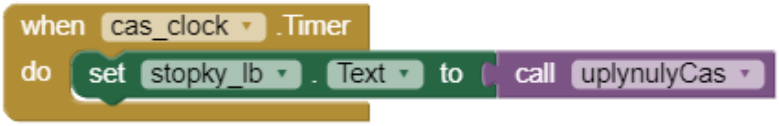
Obrázek 32 - stopovací tlačítko

Princip je tedy následovný – při startu uložíme do proměnné *"zacatek"* aktuální instanci času (*Now*), při stopnutí přičteme do proměnné *"zastaveni"* uplynulý čas v milisekundách (*Duration*) od času zapnutí (*"zacatek"*) do času stopnutí (*Now*).

Resetovací tlačítko vynuluje všechny proměnné, do komponenty *Label* vypíše defaultní text *"00:00:00.000"* a zpřístupní pouze startovací tlačítko.

Obrázek 33 - resetovací tlačítko

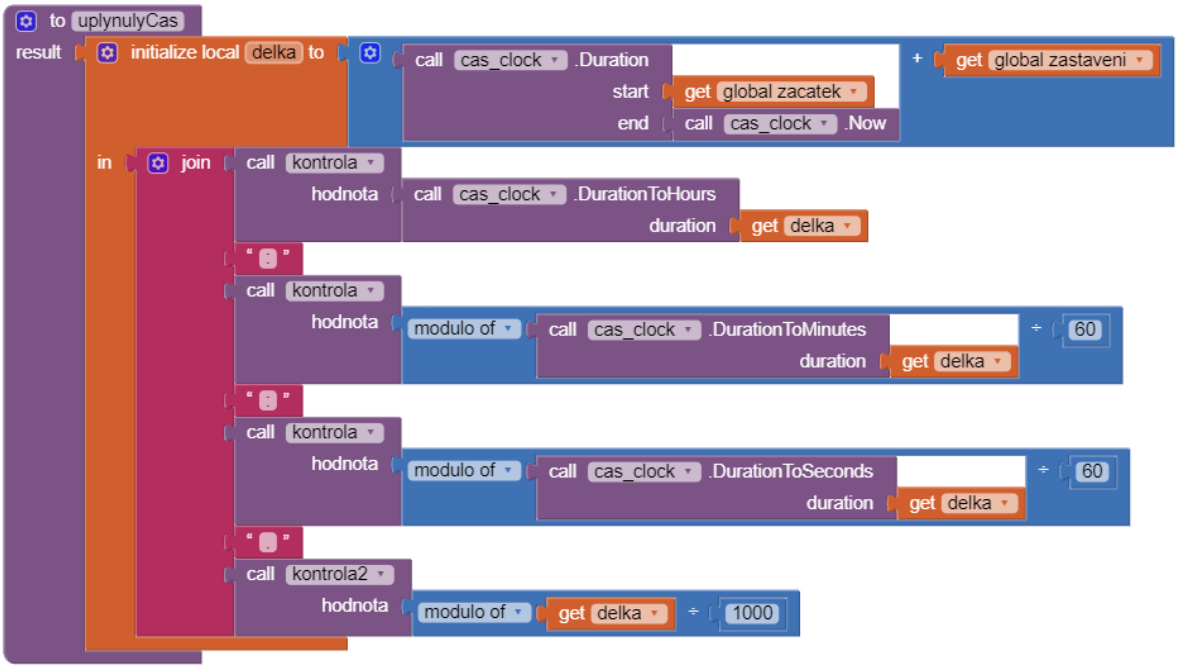
Vypisování uplynulého času zajistí událost *Clock.Timer*, která proběhne každých 10 milisekund. Obsahuje manuálně vytvořenou funkci *"uplynulyCas"*.



Obrázek 34 - výpis času

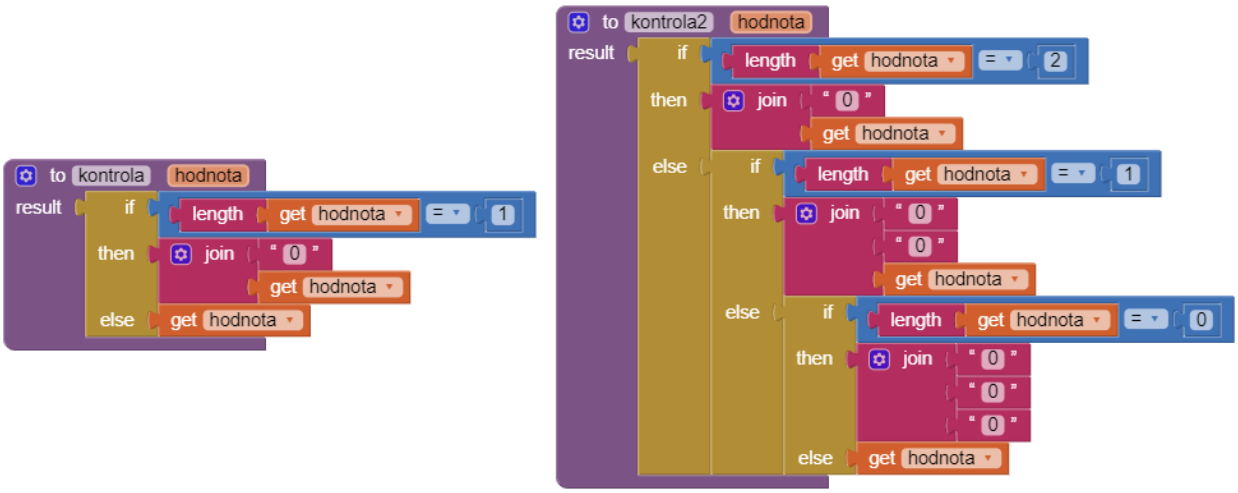
Ve funkci *"uplynulyCas"* se ukrývá lokální proměnná *"delka"*, do které se ukládá součet funkce *Duration* (první z parametrů je proměnná *"zacatek"* a druhý funkce *Now*) a hodnoty v proměnné *"zastaveni"*. Proměnná *"delka"* reprezentuje určité množství milisekund, které následně zformátujeme a výslednou hodnotu vrátíme.

Proměnnou *"delka"* musíme rozdělit na čtyři měřené jednotky času – hodiny, minuty, sekundy a milisekundy. K tomu využijeme existujících jednoparametrových funkcí *DurationToHours*, *DurationToMinutes*, *DurationToSeconds*, do každé z nich dosadíme proměnnou *"delka"*. Pro milisekundy funkci nepotřebujeme, jelikož samotná *"delka"* je v těchto jednotkách. U minut a sekund pak budeme brát modulo 60, u milisekund modulo 1000, aby se jednotky při dosažení daných mezí vrátily na hodnotu 0.



Obrázek 35 - hlavní funkce

O poslední úpravu se postará funkce *"kontrola"*, u každého výše zmiňovaného výpočtu jednotek bude kontrolovat délku hodnoty – pokud bude jednociferná, doplní před ni 0. Pro milisekundy musíme vytvořit speciální funkci *"kontrola2"*, neboť mohou nabývat délky od 0-3. Abychom udrželi konstantní trojcifernou délku, bude v závislosti na délce doplněn určitý počet 0. Tyto funkce mají především estetický význam. Všechny hodnoty nakonec sjednotíme funkcí *join*.

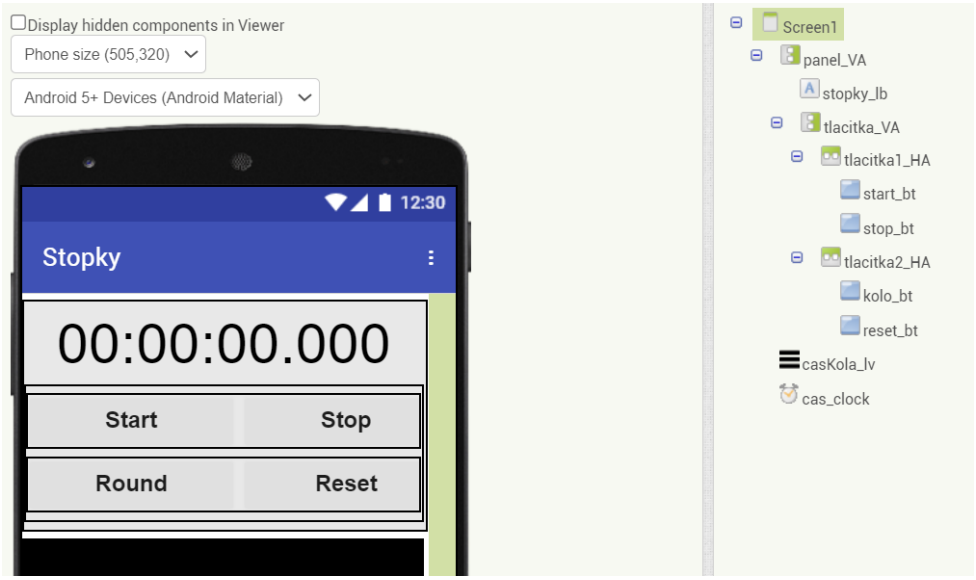


Obrázek 36 - kontrolní funkce

Návrh a řešení domácího úkolu

Potřebné komponenty:

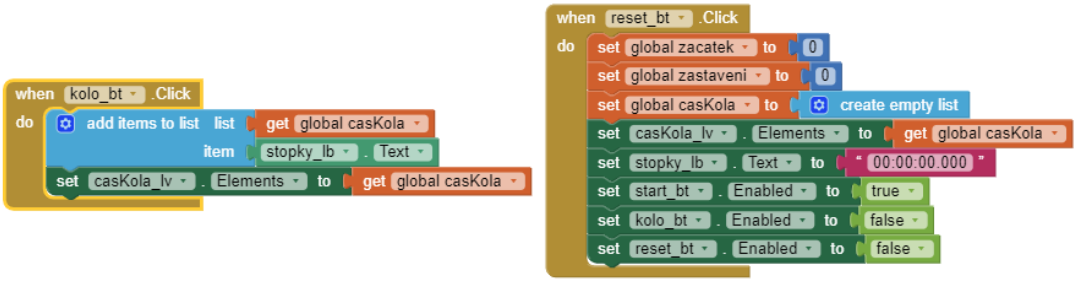
- 1x *Button* – zaznamenání kola neboli aktuálního času
- 1x *ListView* – výpis zaznamenaných kol



Obrázek 37 - vzhled GUI aplikace

Bude potřeba přidat do scény další tlačítko a novou komponentu *ListView*. Také musíme vytvořit list *"casKola"*, do kterého se budou ukládat všechny zaznamenané hodnoty.

Při každém stisknutí tlačítka pro záznam přidáme hodnotu *"stopky_lb.Text"* do listu *"casKola"* funkcí *add items to list* a aktualizujeme zobrazované položky v komponentě *ListView* použitím její vlastnosti *Elements*. Resetovací tlačítko list *"casKola"* a komponentu *ListView* vyprázdní.



Obrázek 38 - tlačítka pro záznam a reset

4.4.3 Lekce 3 – Pexeso

Tato lekce se zaměřuje především na práci s komponentou *Image*, soubory (obrázky a zvuky) a obecnými předpisy (*any component blocks*).

Výstupem této lekce je aplikace na bázi karetní hry Pexeso. Je potřeba se nad jejím principem více zamyslet, aby byl výsledný algoritmus co nejefektivnější.

Průběh lekce

Učitel by měl s předstihem zajistit, aby všichni studenti měli přístup ke složce se soubory (obrázky a zvuky), které jsou potřeba k vytvoření aplikace.

Učitel nejdříve studentům nastíní téma lekce – tvorba pexesa. Přestože se jedná o známou karetní hru a všichni studenti ji pravděpodobně budou znát, může učitel stručně připomenout pravidla a přiblížit tak, co všechno budou muset naprogramovat. Podrobnější popis aplikace se nachází v zadání Pexeso (viz část níže *Zadání úlohy a domácího úkolu*).

Učitel zadá první část úlohy Pexeso. Jelikož je tato úloha již komplexnější, studenti pravděpodobně budou potřebovat více poradit. Učitel tedy společně se studenty vytvoří rozložení aplikace, rozebere, které všechny proměnné bude potřeba vytvořit a deklaruje je.

Učitel také studentům představí kategorii bloků *any component*. Ukáže jednoduché využití události *when any Image.Click* (např. změna obrázku dané komponenty).

Na závěr učitel vysvětlí, jak lze snadno spustit novou hru využitím funkcí *open another screen* a *close screen.*

Studenti by pak měli být schopni naprogramovat zbývající funkce pro generování rozmístění karet, otočení karet, vyhodnocení tahu a dokončení tahu.

Časový harmonogram

Čas	Činnost
25 minut	Zadání a analýza úlohy Pexeso. Stavba aplikace. Deklarace potřebných proměnných. Ukázka bloků *any component*.

45-60 minut	Rozdělení programování na části podle jednotlivých funkcí aplikace. Po dokončení každé funkce proběhne ověření. Jako poslední učitel vysvětlí funkci pro restartování hry.
5 minut	Shrnutí a reflexe lekce. Zadání povinného domácího úkolu k úloze *Pexeso*.

Tabulka 3 - harmonogram Lekce 3

Zadání úlohy a domácího úkolu

Část 1

Vytvořte aplikaci na bázi karetní hry Pexeso.

Aplikace bude obsahovat hrací pole o velikosti 4x4 složené z 16 obrázků (respektive 8 dvojic) vložených do komponent Image, které budou hráči otáčet. Ze začátku všechny karty směřují obrázkem dolů, při stisknutí se zvolená karta otočí obrázkem nahoru. Hráč má ve svém tahu k dispozici 2 otočení.

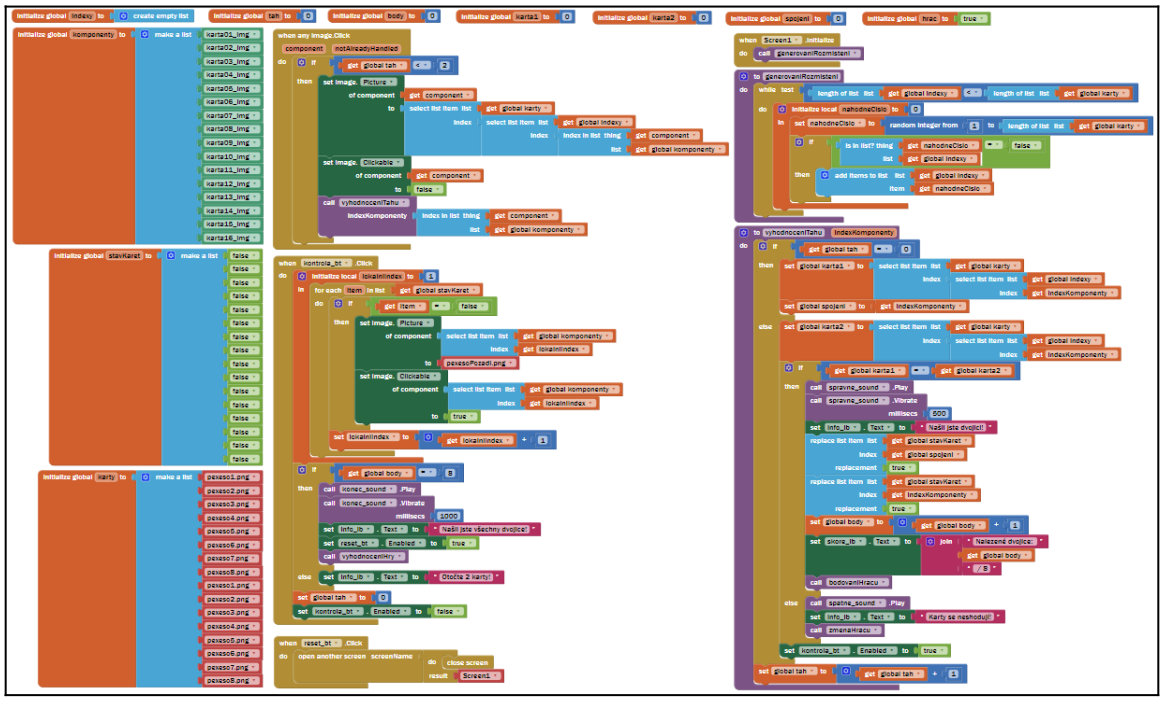
Po druhém otočení hráč dokončí svůj tah stisknutím tlačítka, které bude vyhodnocovat, zda se obrázky otočených karet shodují. Pokud se shodují, karty zůstanou otočeny, v opačném případě se karty otočí zpět dolů. Základní verze aplikace neobsahuje indikaci tahů ani počítání bodů, to je ponecháno na samotných hráčích.

Po nalezení všech dvojic, se zpřístupní resetovací tlačítko, které spustí novou hru.

Dále budeme chtít vypisovat počet nalezených dvojic a informace o průběhu hry – hráč je na tahu, karty se shodují / neshodují.

Hra bude doprovázena celkem 3 zvuky – shoda / neshoda karet a dokončení hry.

Obrázek 39 - vzhled úlohy Lekce 3

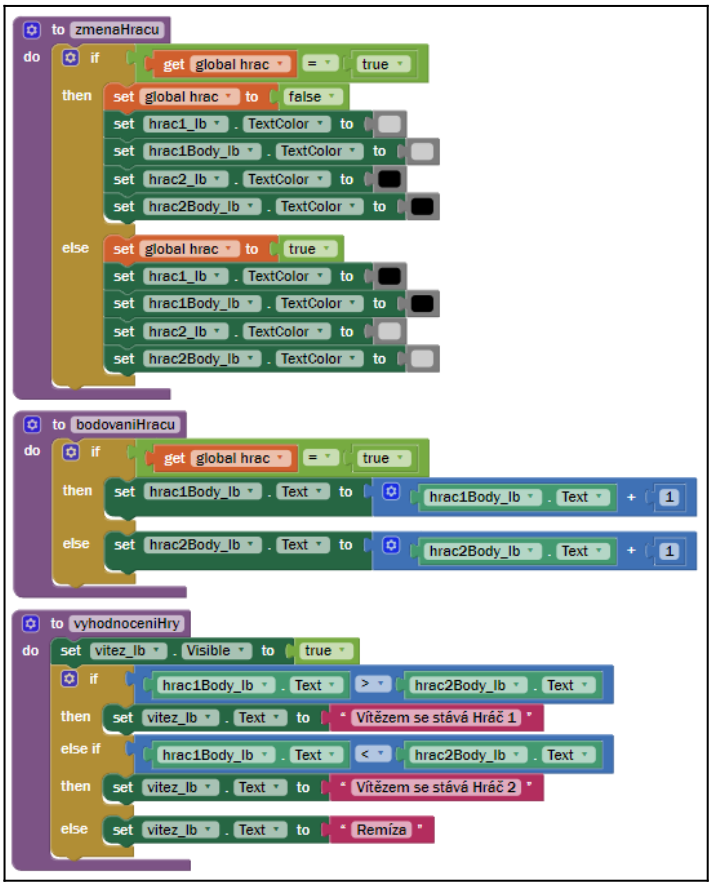


Obrázek 40 - kód úlohy Lekce 3

Povinný domácí úkol

Dodělejte u aplikace indikaci hráčů (jaký hráč je právě na tahu), počet jejich nalezených dvojic, a nakonec vybrání vítěze. Když první hráč najde dvojici pokračuje dalším tahem, pokud nenajde, hraje druhý hráč. Vyhrává ten, kdo najde více dvojic, případně může nastat remíza. Veškerou indikaci postačí udělat pomocí textových komponent *Label*.

Obrázek 41 - vzhled úkolu Lekce 3



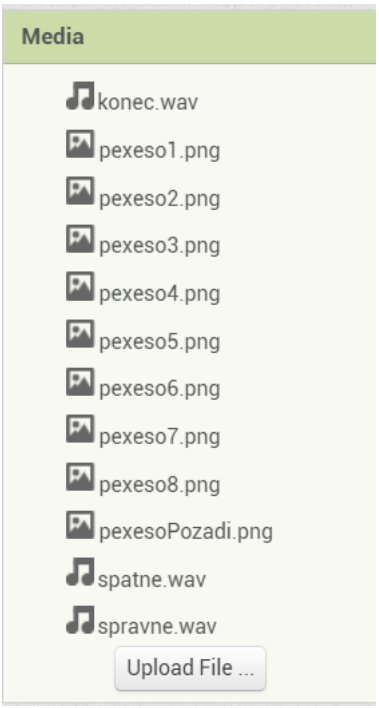
Obrázek 42 - kód úkolu Lekce 3

Návrh a řešení úlohy

Použité komponenty:

- 16x *Image* – karty
- 2x *Label* – výpis informací o průběhu hry a počtu nalezených dvojic
- 2x *Button* – dokončení tahu a resetování hry
- 3x *Sound* – zvuky

Jako první nahrajeme do projektu v sekci *Media* všechny potřebné soubory – obrázky a zvuky.

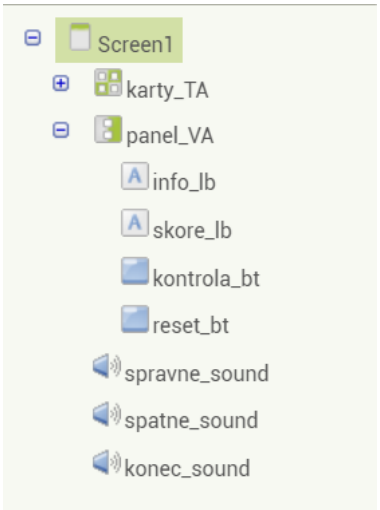


Obrázek 43 - soubory

Poté vytvoříme hrací pole. Pro uspořádání komponent do tabulky se hodí *Layout* komponenta *TableArrangement*, které nastavíme vlastnosti *Columns* na 4 (sloupce) a *Rows* na 4 (řádky). Vložíme do něj 16 komponent *Image*, u kterých povolíme vlastnost *Clickable* a nastavíme *Picture* na defaultní obrázek *"pexesoPozadi.png"*.

Do spodní části přidáme 2 komponenty *Label* pro informování hráčů a pro zobrazování počtu nalezených dvojic, *Button* pro dokončení tahu a *Button* pro resetování hry. Obě tlačítka defaultně nebudou povolena.

O zvuky a vibrace se postarají 3 neviditelné komponenty *Sound*, každé z nich přiřadíme příslušný zvuk ve vlastnosti *Source*. Komponentu *Sound* použijeme z důvodu délky zvukových souborů, neboť se jedná o krátké zvukové efekty. Komponenta *Player* je pak vhodná spíše pro písně.



Obrázek 44 - hierarchie GUI aplikace

Tato aplikace již bude vyžadovat sofistikovanější algoritmus a také podstatně více proměnných.

Základ tvoří 4 listy:

- *"karty"* – všechny obrázky karet (*"pexeso1.png"*, *"pexeso2.png"* atd.)
- *"stavKaret"* – informace o stavu jednotlivých karet, defaultně obsahuje 16x *false* (*false* – karta nebyla spárována; *true* – karta byla spárována)
- *"komponenty"* – všechny komponenty *Image* (*"karta01_img"*, *"karta02_img"* atd.)
- *"indexy"* – informace o pozici karet

Obrázek 45 - listy

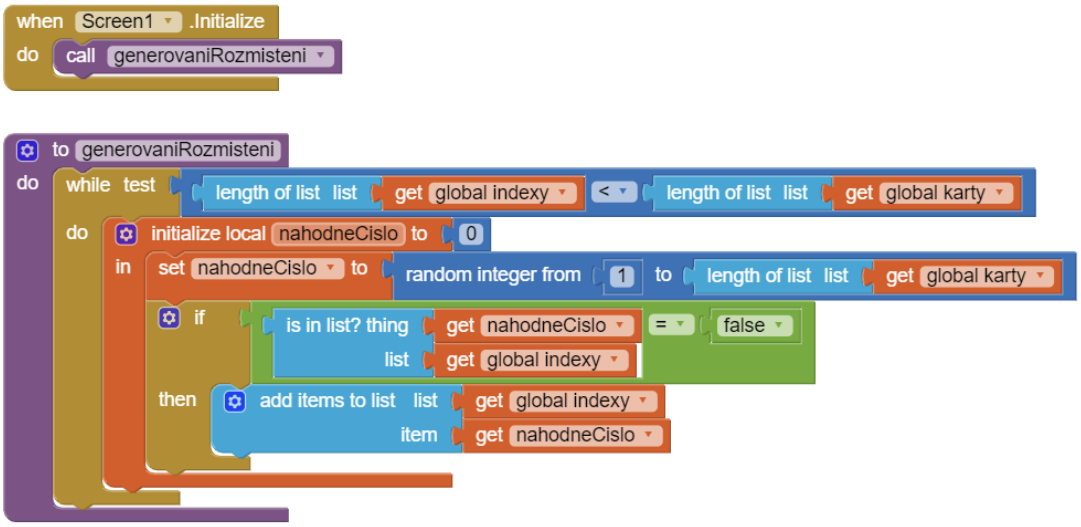
Další proměnné jsou:

- *"karta1"*– první otočená karta
- *"karta2"* – druhá otočená karta
- *"spojeni"* – index první otočené karty
- *"tah"* – počet odehraných tahů (otočených karet)
- *"body"* – počet nalezených dvojic

Obrázek 46 - proměnné

Již při inicializaci obrazovky *Screen1.Initialize* se provede první funkce *generovaniRozmisteni* , která naplní list *indexy* jedinečnými náhodnými čísly od 1 do 16. Jedinečnost zajistí funkce *is*

in list?, která rozhoduje, zda již list aktuální vygenerované číslo obsahuje (*true* – generování proběhne znovu; *false* – číslo přidáme do listu).

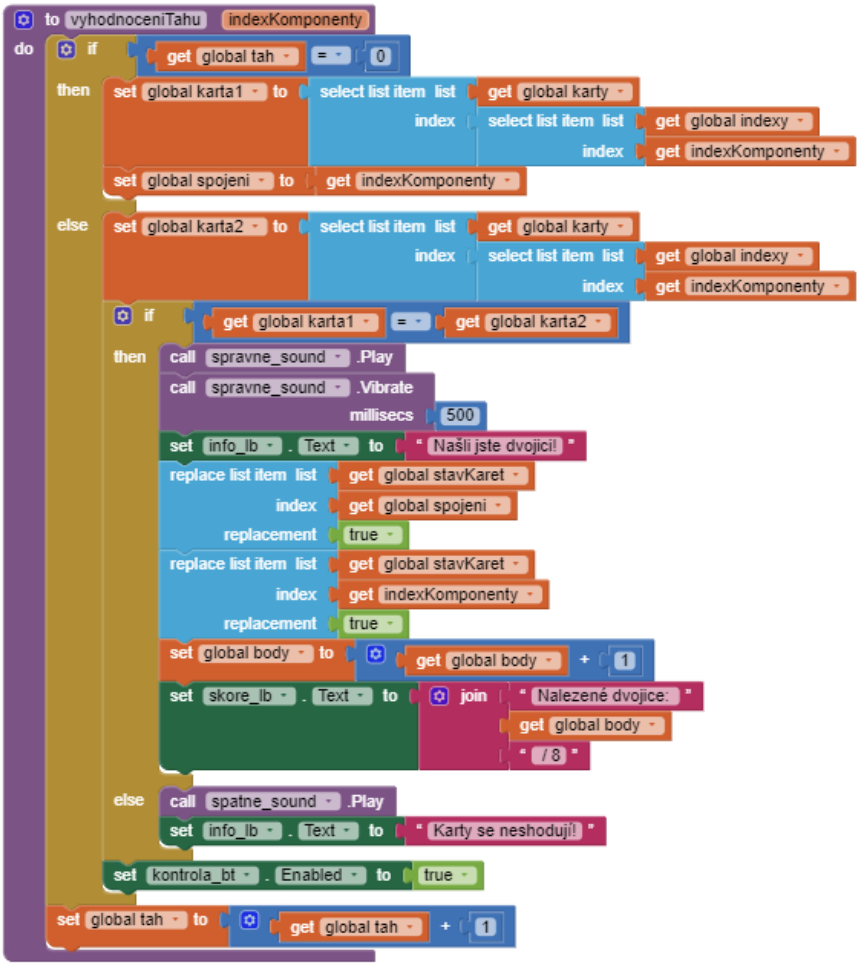


Obrázek 47 - funkce generující rozmístění

Klíčová je funkce *vyhodnoceniTahu* s parametrem *indexKomponenty*, která kontroluje, zda se otočené obrázky shodují. Vykoná se po každém otočení karty a spolupracuje s proměnnou *tah* . Vždy po provedení funkce *vyhodnoceniTahu* se navýší proměnná *tah* o 1.

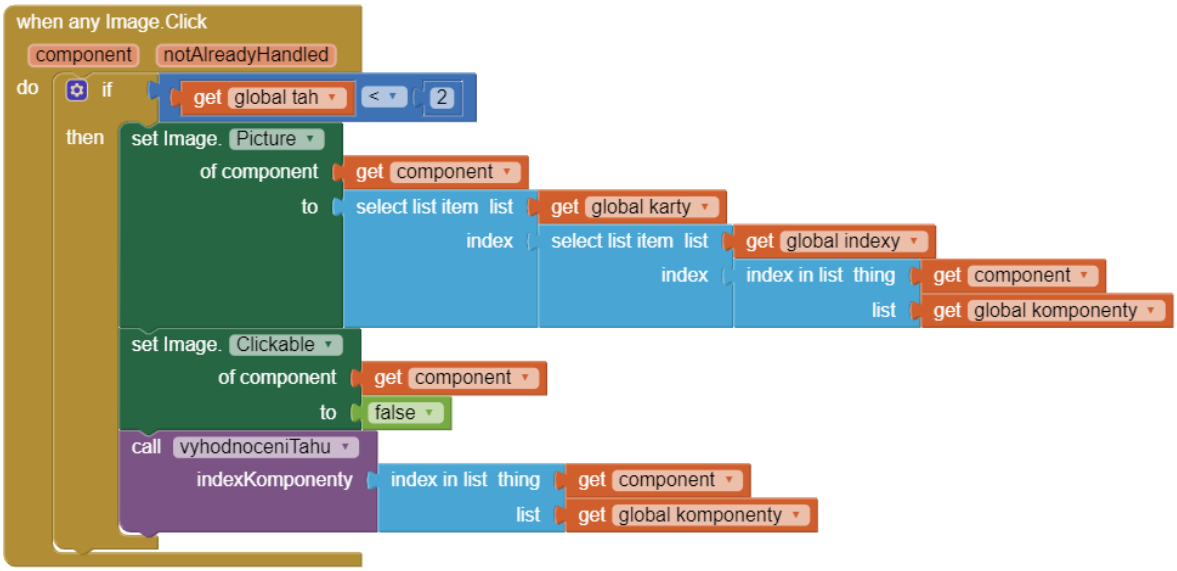
Při prvním tahu uloží do proměnné *karta1* položku z listu *karty* nacházející se na indexu, který je vybrán z listu *indexy* v závislosti na parametru *indexKomponenty* – pozice dané komponenty *Image*. Proměnnou *spojeni* použijeme pro uchování hodnoty *indexKomponenty*, která se v dalším tahu změní.

Při druhém tahu stejným způsobem uloží do proměnné *karta2* a následně se proměnné *karta1* a *karta2* porovná. Shodují-li se, provedou se příslušné akce – zvuk *spravne_sound* (funkce *Play*), krátké vibrace zařízení (funkce *Vibrate*) a oznámení o shodě (vlastnost *info_lb.Text*). V listu *stavKaret* se na indexech o hodnotě *spojeni* a *indexKomponenty* změní položky na *true* . Přičteme a vypíšeme proměnnou *body*. Pokud se karty neshodují opět se provedou příslušné akce – zvuk *spatne_sound* a oznámení o neshodě. Na závěr zpřístupníme tlačítko pro dokončení tahu a navýšíme proměnnou *tah* o 1.



Obrázek 48 - funkce vyhodnocující tahy

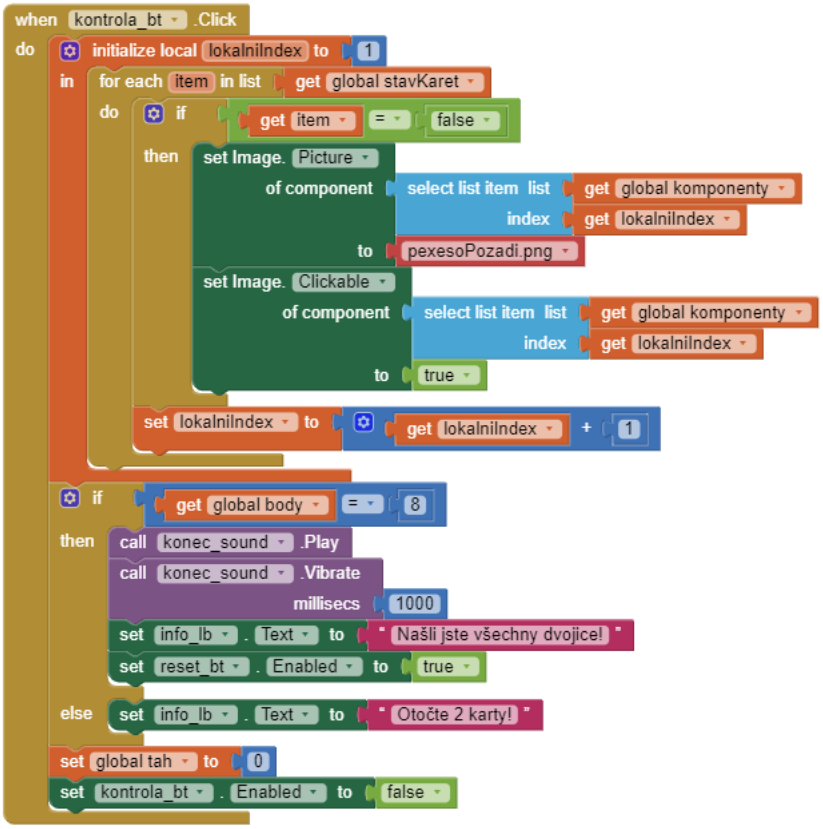
Funkci *vyhodnoceniTahu* budeme volat při stisknutí jakékoliv komponenty *Image*. Abychom tak nemuseli šestnáctkrát opakovat stejný kód, využijeme blok kategorie *any component*, konkrétně událost *when any Image.Click*. O tom, kterou přesně komponentu používáme, informuje povinný parametr *component*. Když hráč stiskne zvolenou komponentu *Image* a má k dispozici své tahy, dojde k otočení karty (změny obrázku), znepřístupnění dané komponenty a zavolání funkce *vyhodnoceniTahu*, které předáme index komponenty v listu *komponenty*.



Obrázek 49 - otáčení karet

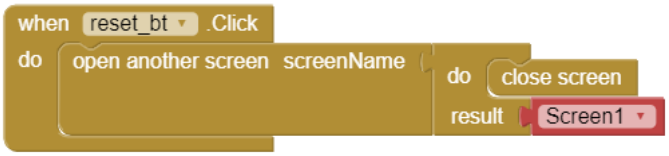
Stisknutí tlačítka *kontrola_bt* zkontroluje list *stavKaret* pomocí cyklu *for each*. Indexy položek *false* použijeme pro změnu obrázku komponent *Image* na původní obrázek *pexesoPozadi.png*, indexy *true* zůstanou ponechány ve stávajícím stavu.

Dokud hra běží, vypíše se hláška o pokračování v tahu, vynuluje se proměnná *tah* a tlačítko samo sebe znepřístupní. Jestliže se proměnná „*body*“ rovná 8 (byly nalezeny všechny dvojice), přehraje se závěrečný zvuk *konec_sound*, zařízení zavibruje, vypíše se oznámení o konci hry a zpřístupní se resetovací tlačítko.



Obrázek 50 - stisknutí tlačítka pro kontrolu

Resetovací tlačítko zavře aktuální komponentu *Screen1* a otevře novou *Screen1*. Tato akce nebude fungovat v MIT AI2 Companion. Avšak stejného výsledku dosáhneme i možností *Refresh Companion Screen* na horní liště v MIT AI.



Obrázek 51 - restart hry

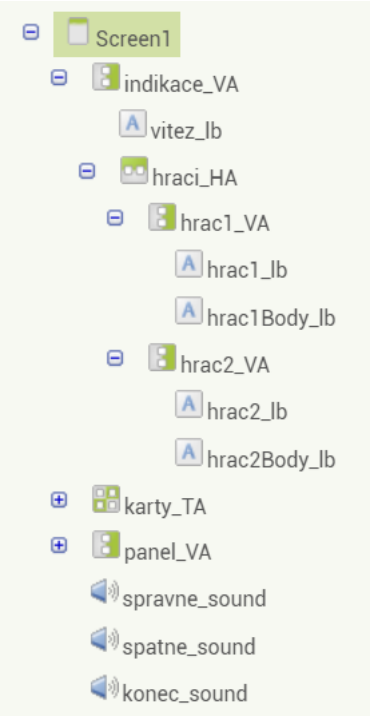
Návrh a řešení domácího úkolu

Potřebné komponenty:

- 5x *Label* – indikace hráčů, jejich nalezených dvojic, výpis vítěze

Úkol cílí zejména na orientaci ve vytvořeném kódu.

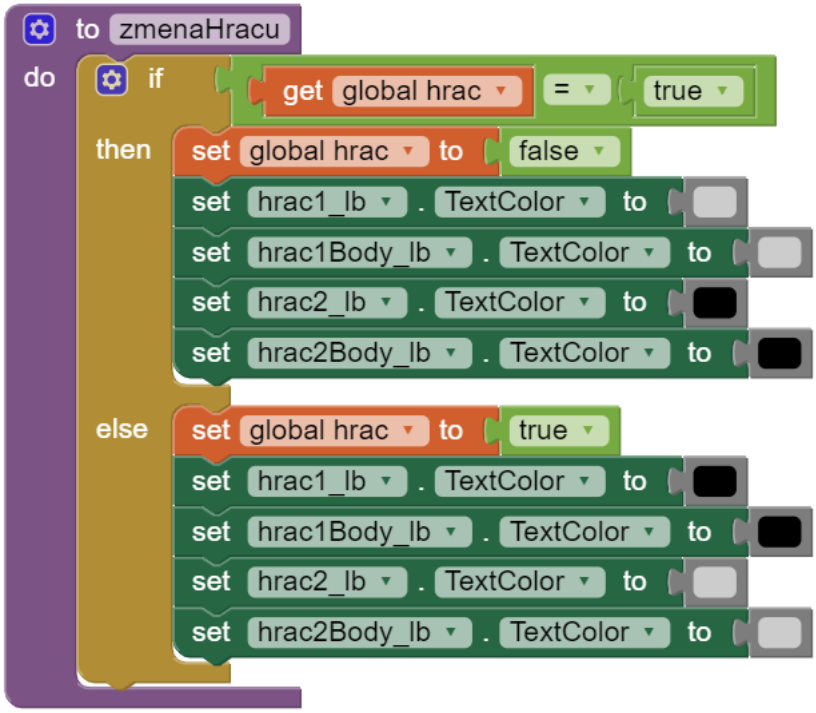
Do projektu přidáme nad hrací pole celkem 5 komponent *Label*, budeme zobrazovat 2 hráče, jejich skóre a vyhodnocení hry. Celkem tedy budou v aplikaci 3 hlavní *Layout* komponenty – *VerticalArrangement* (indikace), *TableArrangement* (hrací pole) a opět *VerticalArrangement* (hlášky a tlačítka).



Obrázek 52 - hierarchie GUI aplikace

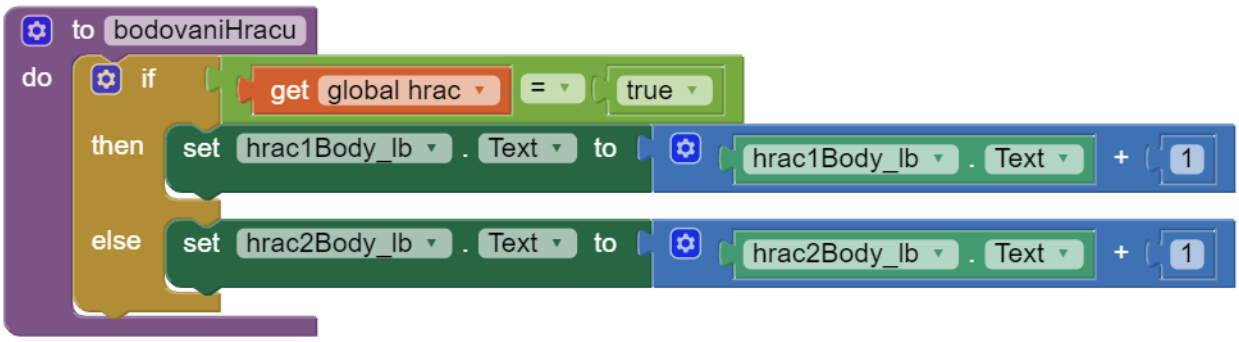
V programovací části vytvoříme novou proměnnou *hrac*, která bude informovat o tom, kdo je na tahu. Vytváříme algoritmus pro 2 hráče, tudíž tato proměnná může nabývat pouze 2 stavů (*true* – hraje hráč 1; *false* – hraje hráč 2).

Dále přidáme 3 nové funkce. Funkce *zmenaHracu* se postará o změnu hráče, který je na tahu. Implicitně bude začínat hráč 1 a proměnná *hrac* tak bude nastavena na hodnotu *true*. Funkce zkontroluje aktuální hodnotu *hrac*, pokud bude *true*, změní ji na *false* a upraví barvy daných komponent *Label*, opačný proces provedeme, pokud bude hodnota *false*.



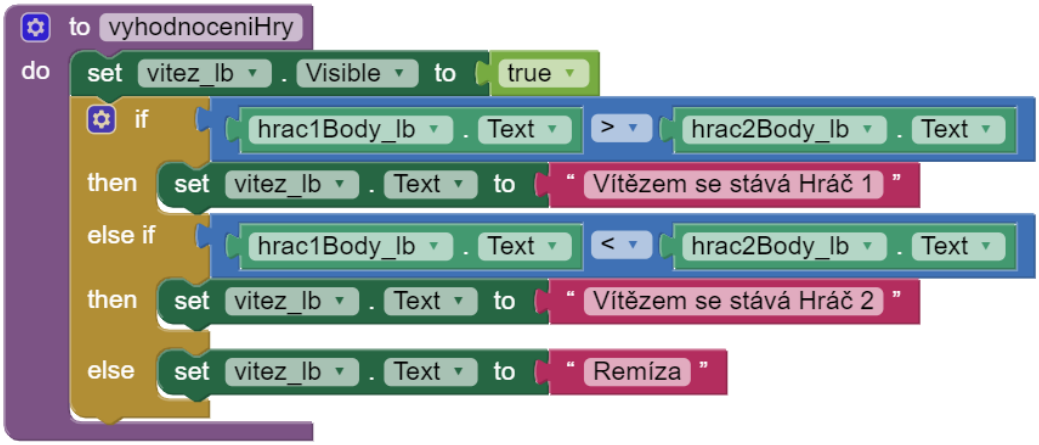
Obrázek 53 - funkce pro změnu hráčů

Funkce *bodovaniHracu* přidává body na základě toho, který hráč je momentálně na tahu.



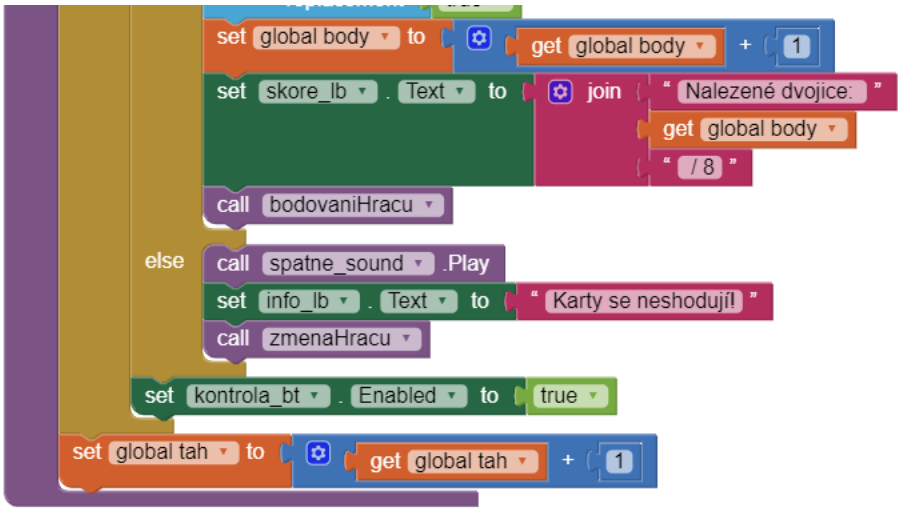
Obrázek 54 - funkce pro bodování hráčů

Funkce *vyhodnoceniHry* vypíše vítěze partie. Mohou nastat 3 situace – vyhraje hráč 1, vyhraje hráč 2 nebo remíza. Při každé z nich vypíšeme do komponenty *vitez_lb* jiný text.



Obrázek 55 - funkce pro zvolení vítěze

Nyní už zbývá jen funkce umístit na vhodná místa do kódu. Funkci *zmenaHracu* budeme volat, když hráč neotočí stejné karty, tedy ve druhé části funkce *vyhodnoceniTahu*. Naopak když se karty shodují, zavoláme funkci *bodovaniHracu*. Funkci *vyhodnoceniHry* budeme volat na konci hry, když hráči najdou všechny dvojice.



Obrázek 56 - pozice funkce 1

Obrázek 57 - pozice funkce 2

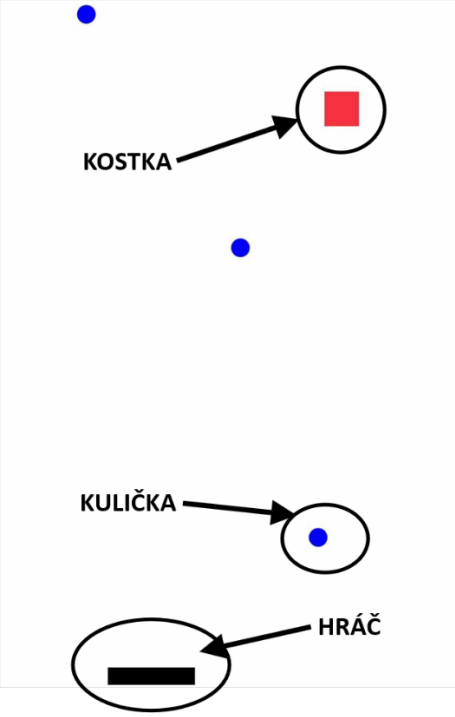
4.4.4 Lekce 4 – Pinball

Čtvrtá lekce se zaměřuje na vývoj více dynamické aplikace. Využijeme k tomu komponent z ka tegorie *Drawing and Animation – Canvas*, *ImageSprite* a *Ball*. Všechny tyto komponenty uplatníme při programování hry, která se podobá Pinballu.

Průběh lekce

Učitel nejdříve studentům nastíní téma lekce – tvorba aplikace na způsob hry Pinball. Učitel zadá první část úlohy Pinball (viz část níže *Zadání úlohy a domácího úkolu*).

Učitel společně se studenty aplikaci sestaví, přičemž představí nové komponenty, které bude potřeba použít, a vysvětlí základní principy hry. Učitel se studenty probere algoritmus pro odrážení kuličky od stěn a hráče. Bližší specifikace hry jako rozměry kostek nebo rychlost kuličky lze ponechat na studentech.



Obrázek 58 - objekty hry Pinball

Dále je potřeba vytvořit i nastavení pozic komponent při inicializaci aplikace. Studenti se však musí řídit vzhledem aplikace na svém zařízení, protože rozlišení obrazovek se pravděpodobně budou lišit. Hodnoty týkající se pozic tak mohou u každého studenta být mírně rozdílné.

Studenti by na vhodný algoritmus měli přijít sami, neboť samotné programování není příliš složité. Jde především o nastavení kolizí jednotlivých komponent a pohyb hráče.

Časový harmonogram

Čas	Činnost
25 minut	Zadání a analýza úlohy Pinball. Ukázka nových komponent *Canvas, ImageSprite, Ball*. Vysvětlení vzorce odrazu kuličky.
40-50 minut	Studenti pracují na své aplikaci individuálně, případně se radí s učitelem. Inicializace aplikace. Programování kolizí a odrazu kuličky.

	Učitel vysvětlí funkci pro restart hry.
5-10 minut	Shrnutí a reflexe lekce. Kontrola řešení studentů a vysvětlení potenciálního provedení. Zadání povinného domácího úkolu k úloze *Pinball*.

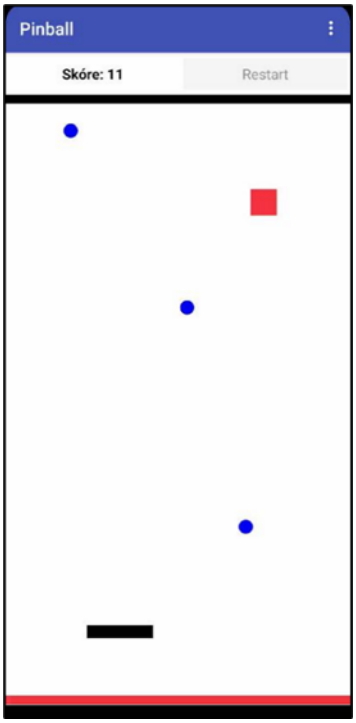
Tabulka 4 - harmonogram Lekce 4

Zadání úlohy a domácího úkolu

Část 1

Smyslem hry je udržet kuličku co nejdéle „ve vzduchu“ a trefovat se do čtverců, které budou zvyšovat její rychlost. Při zásahu čtverec změní svou polohu a navýší skóre. Kulička se bude odrážet od stěn a od objektu, který ovládá hráč tažením doleva či doprava. Hra končí, pokud kulička propadne na dolní stěnu.

Aplikace bude také zobrazovat aktuální skóre, obsahovat tlačítko pro restartování hry a při propadnutí kuličky hráče informuje o konci hry a jeho skóre prostřednictvím oznámení komponentou *Notifier*.



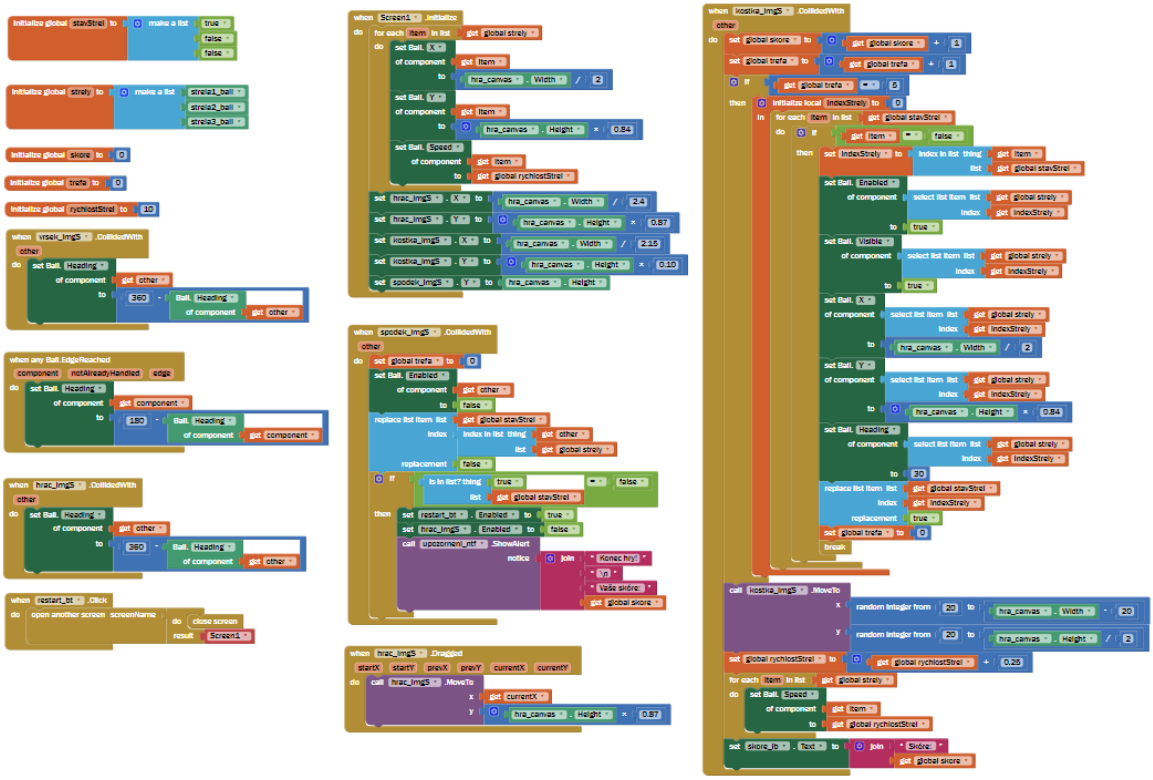
Obrázek 59 - vzhled úlohy + úkolu Lekce 4

Obrázek 60 - kód úlohy Lekce 4

Povinný domácí úkol

Hráč po trefení 5 kostek obdrží další kuličku, která bude fungovat stejně jako základní kulička. Pro zjednodušení stanovíme maximální počet kuliček na 3. Kuličky lze během hry obnovovat,

to znamená, že i když kuličky propadnou, po 5 trefených kostkách se jedna z nich vrátí do hry. Hra končí, až když nebude ve hře ani jedna kulička.

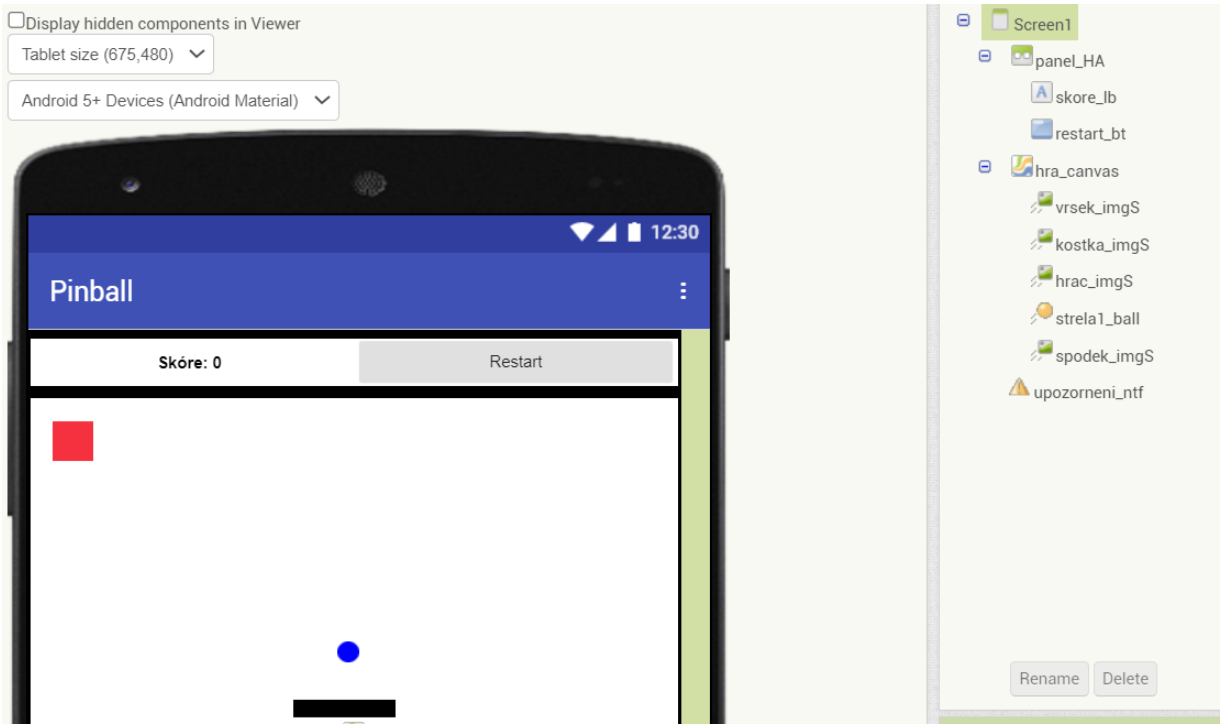


Obrázek 61 - kód úkolu Lekce 4

Návrh a řešení úlohy

Použité komponenty:

- 1x *Canvas* – hrací plocha
- 4x *ImageSprite* – horní a spodní stěna, terč, hráč
- 1x *Ball* – kulička
- 1x *Label* – výpis skóre
- 1x *Button* – restart
- 1x *Notifier* – závěrečné oznámení

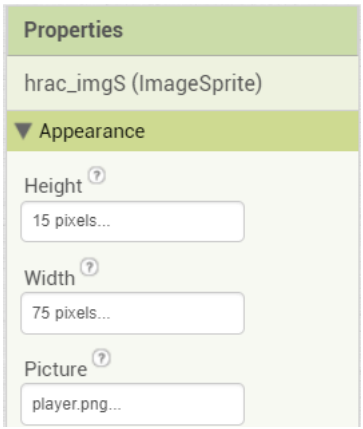


Obrázek 62 - vzhled GUI aplikace

Aplikace se inspiruje populární hrou Pinball, avšak je přizpůsobená účelům a podmínkám lekce.

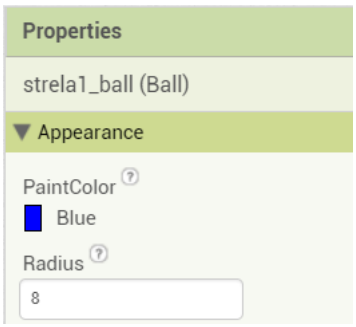
Všechno dění se bude odehrávat na komponentě *Canvas* – souřadnicová (X, Y, Z) plocha spolupracující s komponentami *Ball* a *ImageSprite*, které do ní lze vkládat.

ImageSprite funguje pouze v komponentě *Canvas* (jinak ji ani nelze přidat), kde interaguje s ost atními komponentami či reaguje na dotek uživatele. Vzhled udává obrázek specifikovaný v její vlastnosti *Picture*.



Obrázek 63 - vlastnosti komponenty ImageSprite

Ball je kulatá komponenta *ImageSprite* již předem vytvořená vývojáři MIT AI, která má své specifické vlastnosti. Její vzhled lze upravovat jen vlastnostmi *PaintColor* (barva) a *Radius* (velikost).



Obrázek 64 - vlastnosti komponenty Ball

Pro realistické odražení kuličky od horní stěny vytvoříme speciální objekt *vrskek_imgS*, který nám ulehčí programování. Jinak se bude kulička odrážet od hran komponenty *Canvas*.

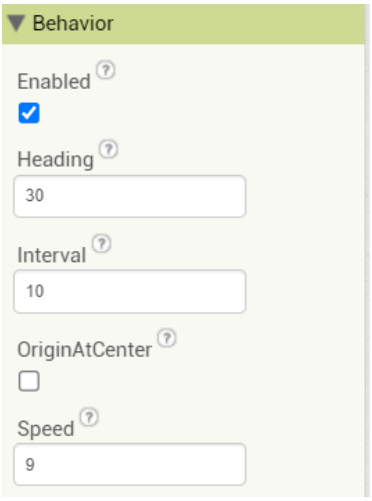
Na vzhled komponent *ImageSprite* použijeme celkem 2 obrázky (horní stěna a hráč – *player.png*; spodní stěna a kostky – *object.png*). Oba obrázky nahrajeme do sekce *Media*.

Nejdůležitější komponentu *Canvas* nastavíme tak, aby zabírala celou dostupnou plochu zařízení a vložíme do ní všechny potřebné komponenty *ImageSprite* a *Ball*. Tyto komponenty musíme také správně nastavit.

U komponent *ImageSprite* budeme řešit především jejich rozměry a pozici. Horní stěna a spodní stěna pokrývají celou šíři komponenty *Canvas*. Hráč (respektive objekt, kterým hráč pohybuje) je v podstatě také stěna, ale s menší šířkou a bude se nacházet nad spodní stěnou.

První kostku umístíme zhruba na střed horní poloviny obrazovky. Kostky by měly mít přiměřené rozměry, aby bylo možné je kuličkou zasáhnout.

U kuličky musíme nastavit výchozí hodnoty vlastností *Heading* (jakým směrem se kulička pohybuje), *Interval* (jak často se kulička pohne v milisekundách), *OriginAtCenter* (zda jsou souřadnice počítány od středu kuličky) a *Speed* (počet pixelů uražených při jednom pohybu). Velikost kuličky udává vlastnost *Radius*.



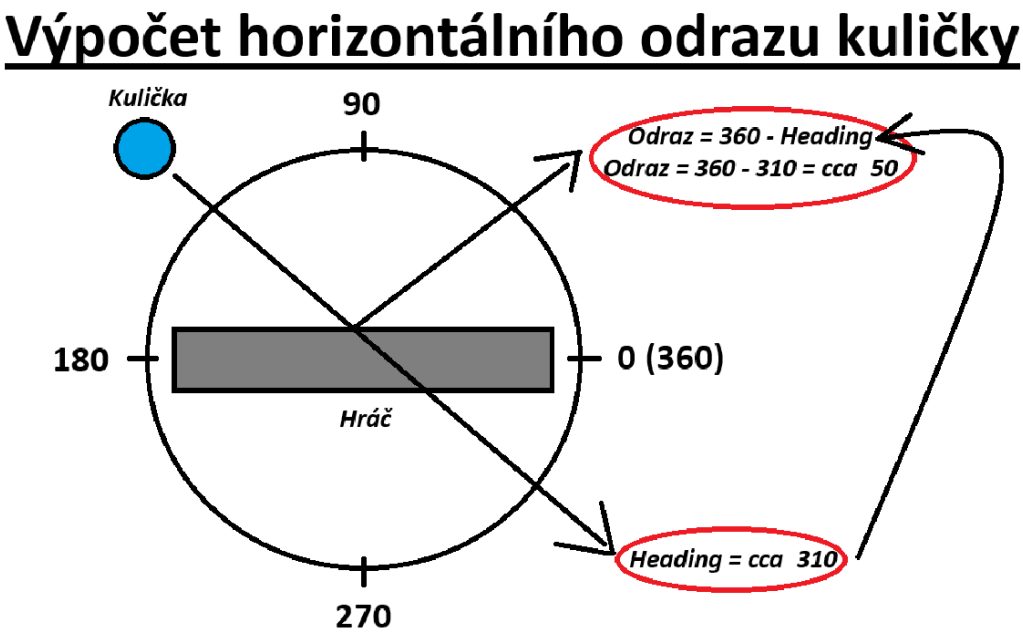
Obrázek 65 - nastavení kuličky

Nad *Canvas* přidáme komponenty *Label* a *Button*. *Label* zobrazuje aktuální skóre, tedy počet trefených kostek a *Button*, který se zpřístupní po propadnutí kuličky, hru restartuje. Poslední komponenta *Notifier* na obrazovce zobrazí oznámení o konci hry a hráčovo dosažené skóre.

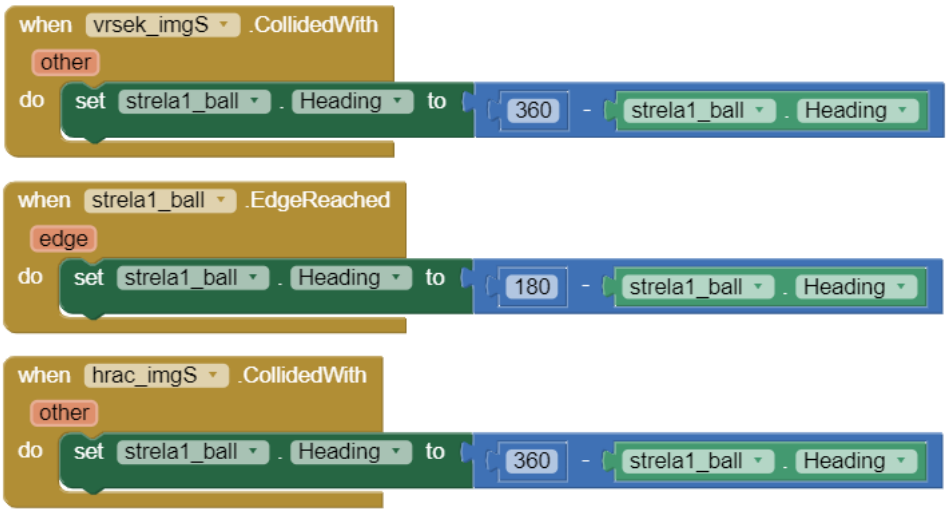
Pozici horní stěny lze nastavit v části *Designer*, protože u té jistě víme, že se bude vždy nacházet na začátku (X: 0, Y: 0). Avšak například u spodní stěny nelze nastavit souřadnici Y absolutně, jelikož je závislá na velikosti zařízení. Pozici dalších komponent tak musíme upravit v programovací části při spouštění, inicializace aplikace (*Screen1.Initialize*). Při určování hodnoty X či Y je nutné vycházet z vlastností *Height* a *Width* komponenty *Canvas*. Je potřeba zjistit vhodnou vertikální pozici hráče, který se bude pohybovat kus nad spodní stěnou. Výchozí pozici kuličky nastavíme nad hráče.

Obrázek 66 - pozice komponent

U každé komponenty také určíme, jakým způsobem bude reagovat na kolizi s jinou komponentou – události *CollidedWith* a *EdgeReached*. Stěny budou odrážet kuličku určitým směrem (nastaví její *Heading* na příslušnou hodnotu). Abychom dodrželi zákon odrazu, horní stěna a hráč odrazí kuličku podle vzorce *360 – strela1_ball.Heading*, zatímco okraje podle vzorce *180 – strela1_ball.Heading*.

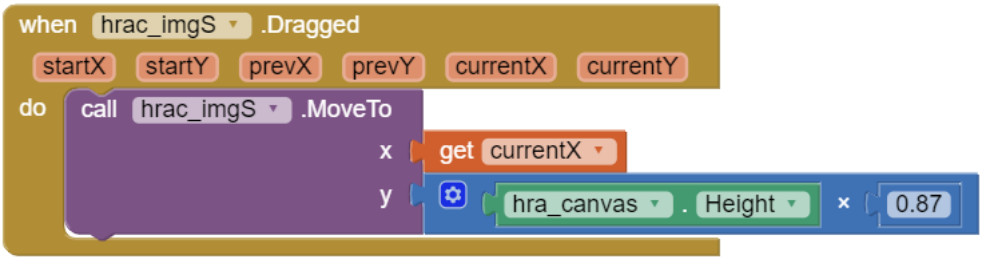


Obrázek 67 - výpočet odrazu kuličky



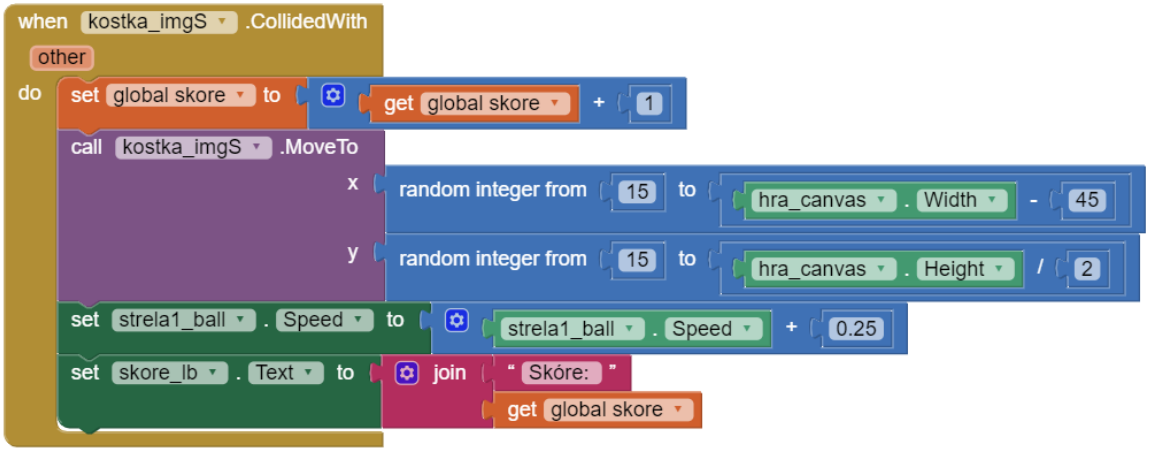
Obrázek 68 - odraz kuličky

Pro pohyb hráče použijeme událost *Dragged* a funkci *MoveTo*. Při každém tažení objektem, ho přesuneme na aktuální souřadnici X (místo dotyku). Souřadnice Y bude konstantní (vložíme do ní hráčovu výchozí hodnotu souřadnice Y), hráč se tudíž bude moct pohybovat pouze po ose X.



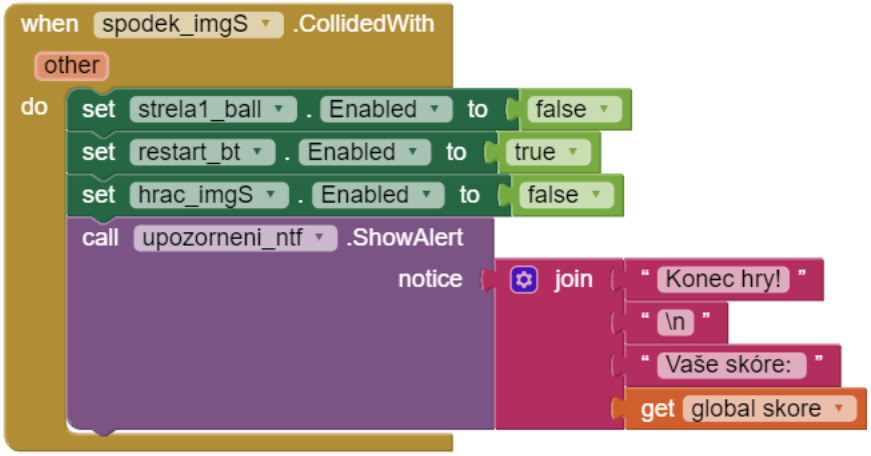
Obrázek 69 - pohyb hráče

Pokud dojde ke kolizi kostky a kuličky, navýšíme skóre a rychlost kuličky, a kostku přesuneme na náhodnou pozici v horní polovině komponenty *Canvas*.



Obrázek 70 - trefení kostky

Když kulička koliduje se spodní stěnou, nastává konec hry a provedou se náležité akce. Znemožníme kuličce a hráči dalšího pohybu (*Enabled* = *false*) a zpřístupníme restartovací tlačítko. Komponenta *Notifier* zobrazí oznámení o konci hry funkcí *ShowAlert*, jejímž parametrem je daná hláška.

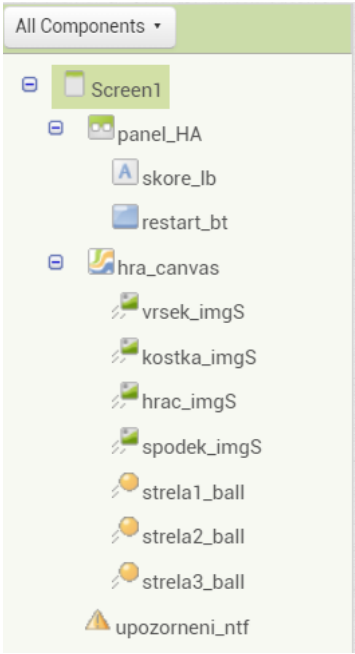


Obrázek 71 – konec hry

Návrh a řešení domácího úkolu

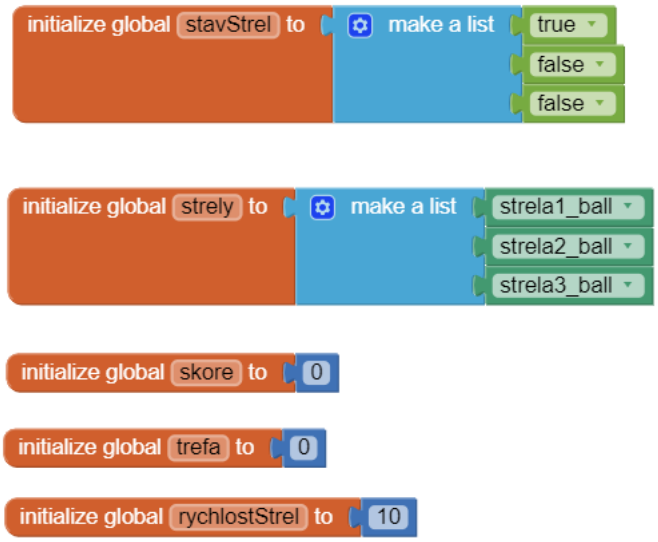
V tomto případě se nabízí více úkolů, které by aplikaci určitým způsobem obohatily a učinily hru zajímavější. Jde například o přidání různých zvukových efektů, dočasné zvětšení / zmenšení hráčova objektu, přidání kuliček, kterým by se hráč musel naopak vyhýbat nebo přidání více běžných kuliček. V této kapitole je popsána pouze poslední možnost.

Hráč po trefení 5 kostek obdrží další kuličku, která bude fungovat stejně jako základní kulička. Pro zjednodušení stanovíme maximální počet kuliček na 3, jelikož každou další kuličku musíme do hry přidat jako novou komponentu.



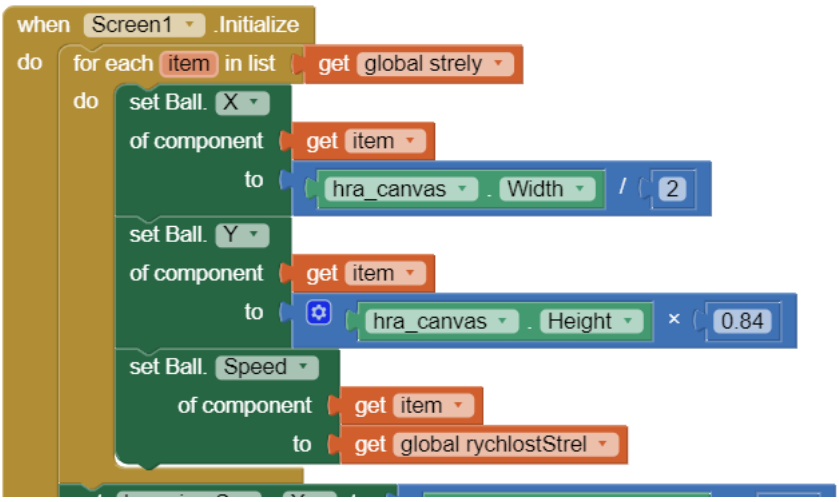
Obrázek 72 - hierarchie GUI aplikace

V komponentě *Canvas* budou celkem 3 komponenty *Ball*, ale pouze jedna bude mít na úvod hry povoleny vlastnosti *Visible* a *Enabled*. Všechny kuličky uložíme do jednoho listu *strely*. Další list *stavStrel* informuje o tom, zda jsou kuličky právě ve hře – ze začátku tedy bude obsahovat 1x *true* a 2x *false*. Proměnná *trefa* uchovává počet trefených kostek a proměnná *rychlostStrel* sjednocuje rychlost všech kuliček.



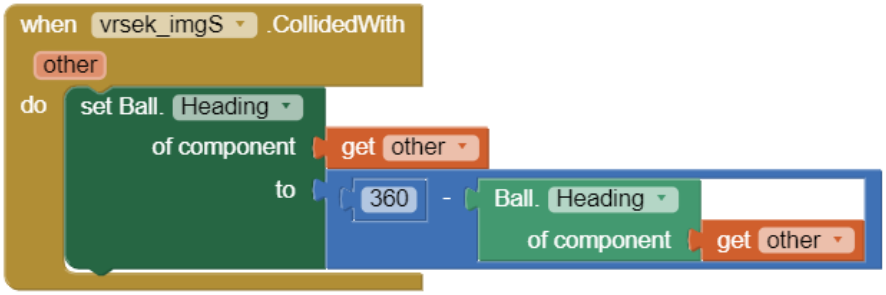
Obrázek 73 - proměnné

Jelikož nyní pracujeme s více kuličkami, musíme kód částečně pozměnit. Při inicializaci nastavíme cyklem *foreach* vlastnosti (*X, Y, Speed*) všech položek v listu *strely* neboli všech komponent *Ball*.



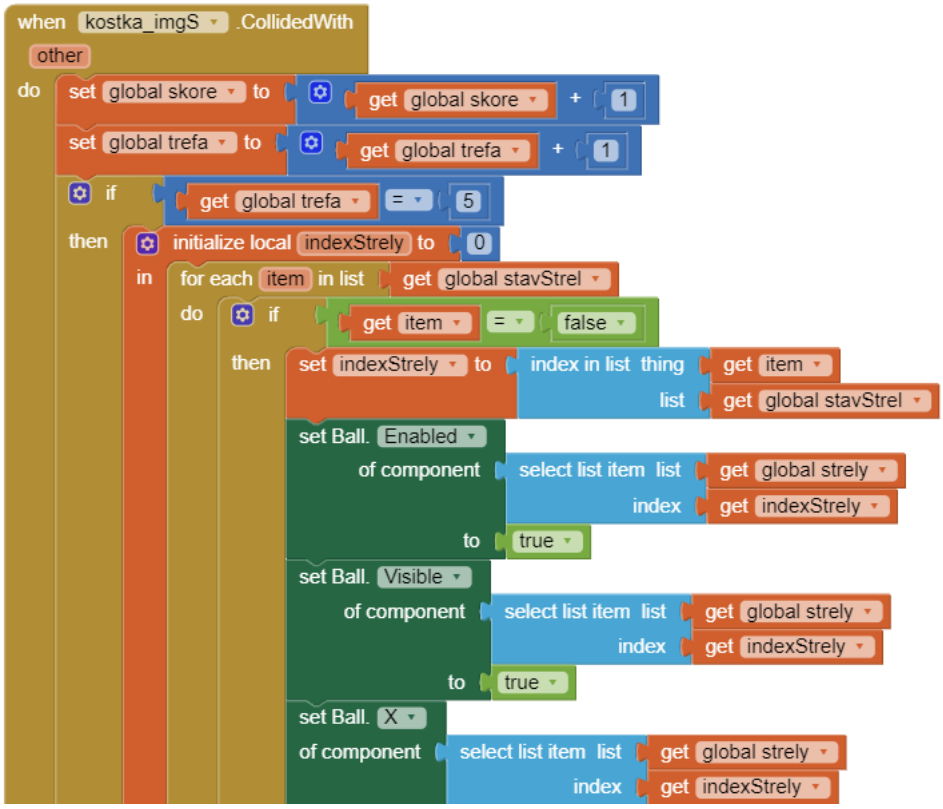
Obrázek 74 - nastavení kuliček

Všechny události kolize upravíme na obecný popis platící pro každou komponentu *Ball*. Musíme využít obecné vlastnosti *Heading*, do které dosadíme kuličku, se kterou došlo ke kolizi (proměnná *other*).

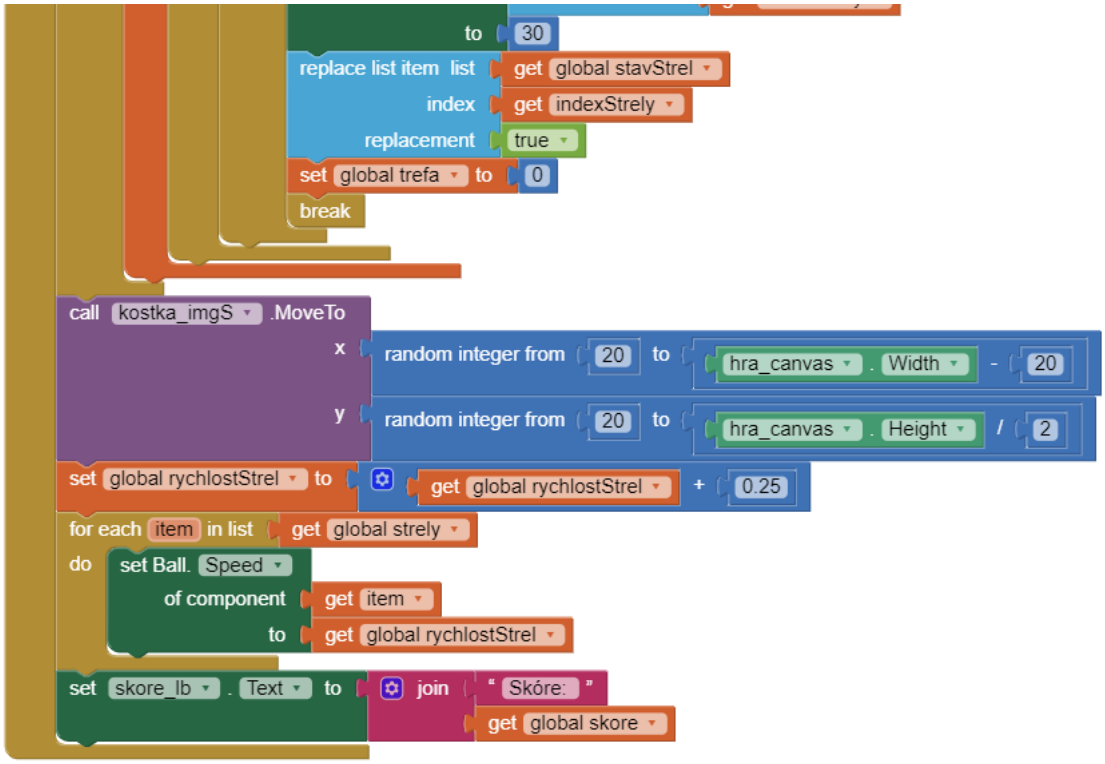


Obrázek 75 - odraz kuliček

Při trefení kostky navýšíme proměnnou *trefa*. Pokud je *trefa* rovna stanovenému počtu, zkontrolujeme položky v listu *stavStrel*. Jakmile narazíme na položku hodnoty *false* udávající, že ve hře nejsou použity všechny kuličky, změníme její hodnotu na *true* a vypůjčíme si její index. Tento index použijeme pro zpřístupnění dané komponenty *Ball* z listu *strely*. Proměnnou *trefa* vynulujeme. Také navýšíme proměnnou *rychlostStrel* a předáme ji všem kuličkám.

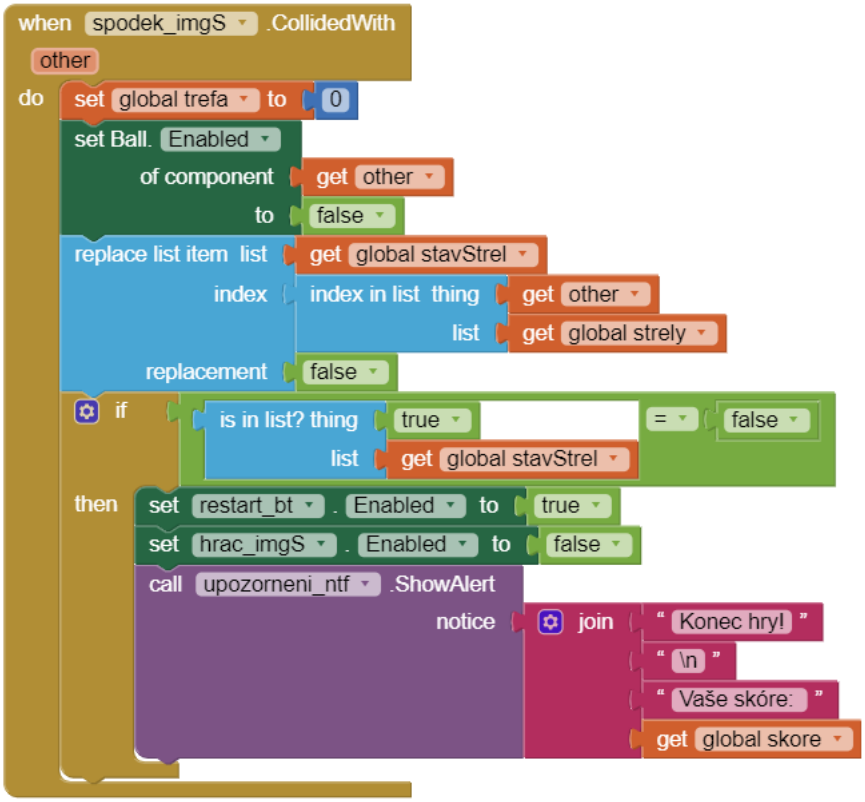


Obrázek 76 - trefení kostky 1



Obrázek 77 - trefení kostky 2

Lehce upravený bude i algoritmus pro vyhodnocování konce hry, neboť nyní konec nastane až při propadnutí všech kuliček, které jsou momentálně ve hře. Může totiž nastat situace, kdy hráč používá všechny kuličky. Pokud ztratí 1 či 2 kuličky, hra pokračuje dále a tyto kuličky je možno oživit. Při propadnutí tedy vynulujeme proměnnou *trefa* a nastavíme položku v listu *stavStrel* p ro danou kuličku na hodnotu *false* a zkontrolujeme, zda se v tomto listu nachází alespoň 1 položka s hodnotou *true*. Záporný výsledek značí, že hráči propadly všechny kuličky a hra končí.



Obrázek 78 - konec hry

4.5 Ověření lekcí ve výuce

Lekce byly ověřeny se studenty 2. ročníku jednooborového studia Informačních technologií ve výuce předmětu Edukační programovací jazyky, který se v době ověřování konal každý čtvrtek od 8:55 do 10:25. Jednalo se o studenty, kteří mají studijní plán zaměřený pouze na obor IT, a tudíž u nich lze předpokládat více teoretických znalostí a praktických dovedností.

Na základě ověření byly provedeny změny, opravy a úpravy v původním harmonogramu a metodice a byl vytvořen upravený soubor AktualizovanaMetodika&Harmonogramy.docx, který se nachází v příloze.

4.5.1 Lekce 1 – Barvy

Průběh výuky

Nejprve jsem se studentům představil a stručně jim vysvětlil své záměry. Informoval jsem je o připravené sadě lekcí, kterou budu v jejich předmětu ověřovat v rámci své bakalářské práce, a

popsal jsem zaměření těchto lekcí, tedy vývoj mobilních aplikací v blokovém vývojovém prostředí MIT App Inventor. Zmínil jsem také, že součástí ověření jsou povinné domácí úkoly, které budou odevzdávat v Moodlu tohoto předmětu.

Následně jsem přešel k úvodní prezentaci o MIT AI a MIT AI2 Companion. Po prezentaci si studenti stáhli na svá mobilní zařízení MIT AI2 Companion, přihlásili se do MIT AI a vytvořili nový projekt.

V MIT AI jsme začali společně tvořit první úvodní aplikaci složenou pouze z komponent *Button, Label* a *HorizontalArrangement.* Při této příležitosti jsem studentům vysvětlil podstatu *Layout* komponent a také dělení bloků podle barev (žluté = události, fialové = funkce a zelené = vlastnosti). Každý ze studentů si zkusil naprogramovat nějakou změnu po stisknutí tlačítka (např. změna textu komponenty *Label*) a funkčnost propojení vytvořené aplikace s mobilním zařízením. Po ověření, že se všem studentům podařilo dostat do tohoto bodu, jsme pokračovali první částí úlohy *Barvy* v novém projektu.

Z hlavní úlohy *Barvy* byla studentům nejprve zadána část pro vytvoření aplikace, která po stisknutí tlačítka změní barvu pozadí na náhodnou a hodnoty vypíše. Doporučil jsem studentům nejprve přidat pouze tlačítko a naprogramovat změnu barvy pomocí dostupných funkcí. Poté přidat komponentu *Label*, vytvořit potřebné proměnné a vygenerované hodnoty vypsat. Musel jsem alespoň stručně představit princip a práci s proměnnými – dělení na lokální a globální, *get* , *set*.

Dále jsme pokročili k druhé části úlohy – generování barev na základě hodnot od uživatele. Studenti museli do stávající aplikace přidat komponenty pro vstup od uživatele a další tlačítko. Studentům bylo také potřeba ukázat, jak upravit určité funkce, aby přijímaly více parametrů (např. funkce *join*). Průběžně jsem studenty kontroloval a dotazoval se jich, zda rozumí zadání a zda vědí jakým způsobem postupovat. Pokud student dokončil některou část úlohy dříve než ostatní, zadal jsem mu další práci (např. zařídit, aby uživatel mohl při zadávání pracovat pouze v rozmezí hodnot 0-255).

V samotném závěru hodiny jsem se vrátil k domácím úkolům. Informoval jsem studenty, že zadání prvního úkolu naleznou v Moodlu a na jeho vypracování mají čas do příští hodiny. Společně jsme se domluvili, že úkoly bude nejjednodušší odevzdávat ve formě souboru

s příponou .aia, který lze vytvořit v MIT AI přes *Projects – Export selected project (.aia) to my computer*. Tato informace byla přidána i do zadání.

Problémy a připomínky

Se samotným stažením a instalací MIT AI2 Companion žádné velké komplikace nebyly. Jeden ze studentů však neměl k dispozici chytrý mobilní telefon a druhý neměl na svém zařízení podporovaný operační systém. Oběma studentům byl zapůjčen školní tablet, na kterém vše fungovalo.

Studenti chtěli po stáhnutí mobilní aplikace MIT AI2 Companion nějakým způsobem ověřit, zda jejich mobilní zařízení dokáže úspěšně spustit aplikaci. Avšak tento problém nelze vyřešit pomocí jednoho QR kódu, který by studentům spustil stejnou aplikaci, neboť projekt v MIT AI lze vždy spárovat pouze s jedním zařízením používající MIT AI2 Companion. Nelze připojit více uživatelů k jednomu projektu. Studenti tak museli s ověřením vyčkat až do samotného programování svých aplikací.

Studentovi se podařilo vyvolat ve vývojovém prostředí MIT AI chybu po zkopírování komponenty a jejím opětovném vložení v *Layout* komponentě *TableArrangement*. Chyba byla takto vyvolána několikrát a chybová hláška zněla *"internal error"*. Bylo nutné vytvořit nový projekt.

Shrnutí výuky

Většině studentů se první úlohu podařilo splnit bez větších obtíží. Nejvíce se museli zamyslet až nad poslední částí – zamezit uživateli zadávat nežádoucí hodnoty. Všichni studenti zjistili, že je vhodné nastavit komponenty *TextBox* jako *NumbersOnly*. Uvedl jsem také, že by studenti mohli zajistit, aby uživatel směl zadávat pouze celá čísla. Jelikož nebylo v zadání přesně specifikováno, jaké bloky využít, spatřil jsem během kontroly různá řešení tohoto problému. Například:

- nahrazení znaku *"."* či *"-"* za *""* pomocí bloku *replace all text* (desetinná čísla by tak byla převedena na celá čísla)
- zaokrouhlení hodnoty na celé číslo pomocí bloku *round, ceiling* či *floor*

Ve svém návrhu na průběh první lekce jsem předpokládal, že bychom se měli během 90 minut výuky dostat i ke druhé lekci *Stopky*. Avšak při ověřování v reálné výuce jsem zjistil, že studenti se musí s novým prostředím nejprve více seznámit a zorientovat se prostřednictvím lehčí úlohy jako je právě úloha *Barvy* a poskytnout jim pro to více času. Úvodní slovo a prezentace, stažení mobilní aplikace, přihlášení a nastavení IDE a vyzkoušení úvodní aplikace zabralo také více času, než jsem předpokládal. K samotné úloze *Barvy* jsem se tak dostal až zhruba v polovině (po 45 minutách).

Přizpůsobil jsem tak harmonogram reálnému průběhu výuky a rozhodl jsem se, že druhou polovinu věnujeme plně úloze *Barvy*, aby si studenti skutečně odnesli základní znalosti o vývojovém prostředí a vyzkoušeli si práci s různými komponentami a bloky.

Několik studentů si muselo zvyknout na ovládání vývojového prostředí a práci s bloky. Někteří zase museli více přemýšlet nad samotným algoritmem. Na konci hodiny však měli všichni úspěšně dokončenou základní aplikaci *Barvy*.

Domácí úkol

Domácí úkol *Barvy* odevzdalo celkem 12 studentů a všem se podařilo jej úspěšně vypracovat. Jelikož v zadání nebylo striktně určeno, jaké bloky či komponenty při řešení použít, objevilo se hned několik variant postupu.

Někteří studenti úkol vyřešili pomocí 3 globálních proměnných pro barvy. Někteří zase úkol řešili využitím funkcí a pracovali s hodnotami přímo ve vlastnosti *Text* dané komponenty. Objevilo se také řešení pomocí pouze 1 funkce, která se starala jak o změnu barvy, tak o výpis všech barev, nebo naopak řešení pomocí 5 funkcí, kdy se každá starala o jednu činnost – inkrement, dekrement, kontrola inkrementu, kontrola dekrementu a aktualizace hodnot. Studenti použili komponentu *Label* nebo *TextBox*, který nastavili jako *ReadOnly*. Pro omezení uživatele všichni využili podmínky.

4.5.2 Lekce 2 – Stopky

Průběh výuky

Nejprve jsem studentům připomněl téma předchozí lekce *Barvy* a krátce jsem zopakoval, čím vším jsme se zabývali. To znamená:

- registrace a přihlášení do MIT AI
- vytvoření a ovládání projektu
- orientace v IDE
- princip GUI a blokového programování
- propojení a spuštění aplikace na mobilním zařízení
- základní komponenty a bloky + práce s proměnnými

Stručně jsem představil a rozebral téma lekce *Stopky*. Vysvětlil jsem studentům, že budeme v MIT AI vytvářet klasické stopky, tudíž aplikaci pro měření uplynulého času. Ukázal jsem jim také možnou podobu GUI výsledné aplikace, aby si dokázali lépe představit potřebné komponenty.

Více jsem představil hlavní komponentu *Clock*, její vlastnosti a nejdůležitější bloky jako *Timer* , *Duration* a *DurationTo*. Snažil jsem se tyto součásti popisovat v souvislosti s vytvářenou aplikací, aby studenti mohli pochopit jejich potenciální roli v této úloze.

Vysvětlil jsem podstatu vytváření vlastních funkcí pomocí bloků v kategorii *Procedures* a stručně jsem bloky demonstroval. Ukázal jsem, jak funkce vytvářet a následně volat, jaký je rozdíl mezi funkcemi bez a s návratovou hodnotou a jak přidávat a pracovat s parametry.

Promítnul jsem studentům zadání úlohy *Stopky*. Poradil jsem studentům nejprve vytvořit GUI aplikace a podotknul jsem, aby se poté především zaměřili na hlavní funkci (měření času) a aby kontrolní funkce a logiku aplikace (přístupnost tlačítek) nechali až nakonec. Nechal jsem studenty pár minut pracovat samostatně, přičemž zvolené řešení bylo zcela na nich.

Následně jsem se dotázal, zda všichni vědí, jakým způsobem postupovat a těm, kteří nevěděli, jsem se snažil podstatu mého provedení lépe vysvětlit a přiblížit jim eventuální řešení. Studenty jsem celou lekci průběžně kontroloval.

Vždy po několika minutách samostatného programování jsem studentům odhalil a vysvětlil část svého kódu. Začal jsem u proměnných a funkcí tlačítek, které do proměnných ukládají určité hodnoty. Zdůraznil jsem, že nejdůležitější je událost *Timer*, ve které aktualizuji komponentu *Label* použitím hlavní funkce. Vysvětlil jsem, avšak neukázal, že v této funkci měřím, kolik času v milisekundách uplynulo od stisknutí startovacího tlačítka (tato instance

času je uložena v proměnné pomocí funkce *Now*) a posledním voláním funkce neboli událostí *Timer* (obsahuje další proměnnou, do které při každém volání uložíme novou hodnotu opět funkcí *Now*). Hodnotu následně matematickými operacemi upravuji do požadovaného formátu. Ponechal jsem studentům dalších pár minut na předělání kódu, a nakonec jsem celý můj kód ukázal a vysvětlil.

K úloze *Stopky* jsem studentům zadal domácí úkol, jehož znění naleznou v Moodlu. Jelikož pro splnění domácího úkolu je potřeba funkční aplikace, přiložil jsem také svůj projekt pro studenty, kteří ji nedodělali během výuky.

Protože nezbývalo mnoho času, rozhodnul jsem se studentům téma příští lekce *Pexeso* pouze představit a ukázat jim nejdůležitější části aplikace. Ukázal jsem studentům finální podobu aplikace a stručně popsal s čím se budeme příště potýkat. Na závěr jsem stručně vysvětlil podstatu bloků *any component*, které budou příští lekci stěžejní. Do žádné stavby či programování aplikace jsme se již však nepouštěli.

Problémy a připomínky

V této lekci již nenastaly žádné komplikace s přihlášením do MIT AI ani s propojením s mobilním zařízením. Problémy se týkaly výhradně programování.

Studenti se například snažili v události *Timer* vkládat do *Label.Text* přímo hodnotu z funkce *Clock.Now*. Do komponenty *Label* se ovšem na první pohled vypsala nesmyslná změť dat. Tato funkce totiž vrací aktuální instanci času mobilního zařízení. Formát instance vlastně obsahuje všechny důležité informace o čase (od milisekund po časovou zónu), avšak kvůli nepřehlednosti nelze tuto hodnotu použít jako výstup pro uživatele. Hodnota funkce *Now* posloužila při výpočtu uplynulého času.

Studenti přišli na to, že by v řešení šla použít funkce *Clock.FormatTime*, která by nám zkrátila a zjednodušila kód. Funkce *FormatTime* naformátuje vstupní hodnotu v milisekundách do podoby *hh:mm:ss*. Nemuseli bychom tedy jednotky času dopočítávat z uplynulých milisekund a vytvářet kontrolní funkce. Na druhou stranu tato funkce nepracuje s milisekundami, které jsou u stopek poměrně významné. Navíc funkce při zkoušce z neznámého důvodu vypisovala 1 uplynulou hodinu ihned po spuštění aplikace. Jak nejlépe tedy funkci implementovat by chtělo více prozkoumat.

Občas se zobrazila při programování chybová hláška. Příčinou byla pravděpodobně úprava bloků, když byla zapnutá komponenta *Clock*. Chyba však na aplikaci žádný viditelný dopad neměla.

Některým studentům se také podařilo vymyslet jiné úspěšné řešení úlohy. Namísto jedné hlavní funkce s lokální proměnnou měli proměnných více (pro každou jednotku času) a tyto proměnné dále upravovali.

Shrnutí výuky

Většina lekce byla věnována úloze *Stopky*. Studenti se naučili, jak v MIT AI vytvářet a volat vlastní funkce. Byla vysvětlena a úlohou vyzkoušena podstata komponenty *Clock* sloužící k práci s časem.

Studentům byl po zadání úlohy poskytnut potřebný čas a volnost pro řešení, abych zjistil, zda dokáží po představení jednotlivých částí úlohy a vysvětlení zadání vymyslet vlastní funkční řešení. Několik studentů samo vědělo, jak úlohu vyřešit. Ostatní studenti měli největší problémy s pochopením hlavní funkce – nevěděli jakým způsobem převést měřenou hodnotu na jednotlivé jednotky času.

Bylo potřeba jim více objasnit princip – událost *Timer*, funkce *Now* a *Duration*. Nejvíce pomohla názorná ukázka vypisováním pouze uplynulých milisekund pomocí připojení funkce *Duration* v události *Timer*. Dále jsem jim vysvětlil, že milisekundy už stačí pouze upravit do žádoucího formátu a vypsat do komponenty *Label*.

Kvůli většímu množství času stráveného vysvětlováním a kontrolou nezbylo tolik prostoru pro další lekci *Pexeso*, ke které jsem se chtěl více dostat a alespoň vytvořit GUI aplikace a probrat všechny potřebné proměnné. Avšak s ohledem na zbývající čas jsem se rozhodnul, že nemá smysl pouštět se do další lekce. Úlohu *Pexeso* se tedy pokusíme dokončit poslední lekci, která se uskuteční až za 2 týdny. Předpokládám, že k lekci *Pinball* se během výuky nedostaneme.

Domácí úkol

U domácího úkolu *Stopky* již nebyli úspěšní úplně všichni studenti. Ke splnění tohoto úkolu je zapotřebí funkční aplikace *Stopky*, kterou si studenti sami vytvořili během výuky nebo měli v Moodlu k dispozici mé řešení. V zadání je specifikováno, aby byl úkol řešen pomocí

komponenty *ListView*, o což se snažili všichni studenti. Při kontrole jsem primárně dbal funkčnosti stopek a zaznamenávání kol.

Mezi nepovedenými řešeními se objevily nefunkční stopky, které běžely dále i po stopnutí, akorát se přestaly vypisovat. Dále stopky, které po stopnutí nelze opětovně spustit a musí se restartovat. Nefunkční řešení pomocí bloků *evaluate but ignore result* a funkce *CreateElement* . Nevhodný formát vypisování jednotek (např. 00:00:10.5 namísto 00:00:10.005) hodnotím spíše jako nedostatek, protože samotné zaznamenávání bylo jinak v pořádku.

V úspěšných případech jeden student přidal k vypisování kola hodnotu, o kolikátý záznam se jedná. Další řešení ku příkladu obsahovalo vypisování kola pomocí spojení jednotlivých jednotek do jednoho výstupu funkcí join. Objevilo se také funkční řešení pomocí komponenty *TinyDB* tedy databáze, do které se ukládají jednotlivé záznamy kol a poté se všechny vypisují do *ListView*.

4.5.3 Lekce 3 – Pexeso

Průběh výuky

Jelikož od poslední lekce uplynuly dva týdny, připomněl jsem na úvod, čím jsme se minule zabývali. Představil jsem téma dnešní lekce – Pexeso. Ukázal jsem zadání a vysvětlil, jak má aplikace zhruba vypadat a fungovat. Následně jsem studenty požádal, aby si stáhli z Moodlu dané materiály a nahráli všechny soubory do MIT AI. Poté začali stavět GUI aplikace, přičemž se mohli inspirovat obrázkem v zadání. Po dokončení GUI jsem podrobněji popsal nové komponenty *Image* a *Sound* – jejich nejdůležitější vlastnosti a funkce.

Společně se studenty jsme řešili, jaké všechny proměnné a listy bude potřeba vytvořit. Ještě jednou jsem vysvětlil princip aplikace, tentokrát s využitím deklarovaných proměnných. Rozdělil jsem ji na čtyři části, které bude potřeba naprogramovat (generování rozmístění, vyhodnocování tahu, otáčení karet a kontrola tahu).

Nejprve jsem studentům zadal, aby se pokusili vytvořit první funkci pro generování rozmístění karet, tudíž naplnit list 16 jedinečnými čísly 1-16. Nechal jsem studentům vždy několik minut a následně jsem ukázal a vysvětlil své provedení. Takto jsem postupoval se všemi funkcemi.

Nakonec jsem vysvětlil, jak lze aplikaci jednoduše resetovat a upozornil studenty na domácí úkol, který opět naleznou v Moodlu.

Problémy a připomínky

Je potřeba konstatovat, že největší problém se týkal samotné úlohy, respektive aplikace, která byla příliš složitá a vzhledem k omezenému času i příliš rozsáhlá. Nejlepší částí byla pravděpodobně stavba GUI, při které studenti museli přijít na to, jak komponenty správně nastavit, aby vytvořili požadovaný vzhled. Tuto část splnili všichni, někteří s menšími obtížemi.

Při nahrávání souborů se ukázaly nedostatky vývojového prostředí, protože přes tlačítko *Upload File* v sekci *Media* z nějakého důvodu nelze nahrát více souborů najednou. Repetitivní práce nastala i při vytváření proměnných typu list. Kvůli velkému počtu proměnných se pak při programování jednotlivých funkcí studenti nedokázali dobře orientovat v kódu.

Studenty jsem z důvodu časové tísně průběžně nekontroloval a ani tak se aplikaci nepodařilo dokončit celou.

Shrnutí výuky

Celkově bych tuto lekci v podobě a provedení, jež jsem navrhl a realizoval, ohodnotil jako nezdařilou. Pokud bychom chtěli zachovat téma Pexeso, bylo by nutné aplikaci nějakým způsobem zjednodušit.

Například bychom mohli snížit počet komponent. Lze použít 4 namísto 8 dvojic obrázků a zbavit se informačních komponent *Label*, což alespoň trochu zmenší výsledný kód. Nicméně k principu algoritmu mě již žádné změny nenapadají. Další možností je samozřejmě tuto úlohu do výuky vůbec nenasazovat a ponechat ji pouze jako samostudium. Náhradní náplní třetí lekce by tak mohla být úloha *Pinball*.

Domácí úkol

Pro domácí úkol *Pexeso* platily stejné podmínky jako pro úkol *Stopky*, tudíž mít funkční aplikaci *Pexeso* nebo využít mého řešení dostupného v Moodlu. Při kontrole jsem ověřoval, zda dochází k indikaci a přepínání hráčů, správnému rozdělení bodů a vyhodnocení hry.

Studenti většinou využili mou aplikaci, ke které pouze přidali domácí úkol. Většina odevzdaných úkolů byla zdařilá a všichni studenti postupovali podobným způsobem. Indikaci hráčů realizovali téměř všichni pomocí komponent *Label*, u kterých podle pravidel hry měnili vlastnost *TextColor*, značící, kdo je právě na tahu. Objevilo se však i řešení pomocí komponenty *Label*, která vypisovala číslo hráče na tahu.

Když byla nalezena dvojice, byl přičten bod hráči na tahu a v opačném případě došlo ke změně hráče. Hráč byl reprezentován proměnnou typu *boolean* (*false* = hráč 1, *true* = hráč 2) nebo *integer* (např. 1 = hráč 1, 2 = hráč 2). Pro bodování byly vytvořeny dvě proměnné. Nakonec došlo k vyhodnocení hry podmínkami na základně skóre obou hráčů.

U nepovedených řešení mezi nedostatky patřilo – chybějící indikace, který z hráčů je právě na tahu, nefunkční změna hráčů nebo neodpovídající zvukové efekty.

4.5.4 Lekce 4 – Pinball

Tato lekce nebyla ověřena ve výuce. Jak se však ukázalo u ověřených lekcí, je rozhodně lepší poskytnout studentům při vývoji aplikací více času, než jsem předpokládal. I průběh této lekce bych tedy mírně upravil, respektive bych navýšil celkový potřebný čas na dvě vyučovací hodiny. Lekci *Pinball* by také bylo pravděpodobně lepší prohodit s lekcí *Pexeso*, jelikož úloha *Pinball* není tak algoritmicky náročná. Bylo by však nutné studentům představit bloky *any component* již v této lekci, protože jejich použití vyžaduje domácí úkol.

Závěr

Bakalářská práce nejprve shrnuje nejpodstatnější informace z oblasti softwarové architektury. Pojednává zprvu obecně o softwarové architektuře a dále o architektuře platformy a aplikací Android. U Android aplikací se potom zaměřuje na jednotlivé komponenty, ze kterých se aplikace skládají, doporučené architektonické principy, které by se při vývoji měli dodržovat, a strukturalizaci aplikace na různé vrstvy.

Zdůvodnil jsem svou volbu vývojového prostředí MIT App Inventor a krátce jej popsal. Vysvětlil jsem, co vše je potřeba, abyste v tomto IDE mohli začít vyvíjet mobilní aplikace a testovat je na mobilním zařízení. Stručně jsem objasnil jeho nejdůležitější části, ve kterých je potřeba se při vývoji orientovat.

V praktické části bakalářské práce jsem pro úspěšné plnění lekcí stanovil vstupní požadavky pro učitele a studenty. Definoval jsem také určité pojmy spojené s vývojovým prostředím, které v dalších částech práce používám.

Byly vytvořeny 4 lekce zaměřené na vývoj mobilních aplikací v blokovém vývojovém prostředí MIT App Inventor, které postupně odkrývají podstatu vývoje pomocí grafického uživatelské rozhraní a blokového programování. Součástí každé lekce je úloha a domácí úkol. Každá lekce obsahuje metodiku pro učitele, časový harmonogram, zadání úlohy a domácího úkolu a jejich potenciální řešení. Pro první lekci je také připravena a popsána úvodní prezentace sloužící k představení MIT App Inventor a MIT AI2 Companion.

Tři lekce ze čtyř byly následně ověřeny ve výuce na vysoké škole. Předpokládal jsem, že se mi podaří ověřit všechny čtyři lekce, avšak neodhadl jsem správně časovou náročnost úloh. U každé ověřené lekce byl okomentován její průběh, problémy a připomínky. Třetí lekce *Pexeso* nedopadla podle mých představ a bylo by zapotřebí ji zjednodušit či celkově upravit. Čtvrtá lekce *Pinball* nebyla ověřena ve výuce a nelze tak s jistotou určit, zda je navržený průběh zcela optimální. Změny provedené v návrhu lekcí na základě ověřování ve výuce byly promítnuty do aktualizovaného návrhu lekcí AktualizovanaMetodika&Harmonogramy.docx, který je k dispozici v příloze.

Celkově bych svou bakalářskou práci zhodnotil jako úspěšnou. Hlavní cíle práce vytvořit sadu lekcí pro seznámení s vývojem mobilních aplikací pomocí IDE s blokovým prostředím a metodiku pro učitele byly splněny. Ve výuce nebyla ověřena pouze poslední lekce *Pinball.* Ověřování ukázalo, že u lekce *Pexeso* by bylo vhodné upravit rozsah a obtížnost aplikace.

Seznam použitých informačních zdrojů

1. Software Architecture. *Carnegie Mellon University Software Engineering Institute* [online]. [cit. 2024-03-28]. Dostupné z: https://www.sei.cmu.edu/our-work/software-architecture/
2. QIN, Zheng.; XING, Jian-Kuan. a ZHENG, Xiang. *Software Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN 9786612005237. Dostupné z: https://doi.org/10.1007/978-3-540-74343-9.
3. *Application Architecture: Demystifying the Concept*. Online. Radix. 2023. Dostupné z: https://radixweb.com/blog/what-is-application-architecture. [cit. 2024-03-11].
4. LACKO, Ľuboslav. *Mistrovství – Android*. Online. Albatros Media, 2017. ISBN 9788025148785. Dostupné z: https://www.google.cz/books/edition/_/WBy2DwAAQBAJ?hl=cs&gbpv=0. [cit. 2024-02-19].
5. *Application fundamentals*. Online. Android Developers. 2023. Dostupné z: https://developer.android.com/guide/components/fundamentals. [cit. 2024-06-27].
6. *Platform architecture*. Online. In: Android Developers. 2023. Dostupné z: https://developer.android.com/static/guide/platform/images/android-stack_2x.png. [cit. 2024-03-11].
7. *Android (operační systém)*. Online. Wikipedia. 2008. Dostupné z: https://cs.wikipedia.org/wiki/Android_(opera%C4%8Dn%C3%AD_syst%C3%A9m). [cit. 2024-03-28].
8. *Guide to app architecture*. Online. Android Developers. 2023. Dostupné z: https://developer.android.com/topic/architecture. [cit. 2024-03-05].
9. *Fragments*. Online. Android Developers. 2023. Dostupné z: https://developer.android.com/guide/fragments. [cit. 2024-06-27].
10. *Oddělení zodpovědností (informatika)*. Online. Wikipedia. 2012. Dostupné z: https://cs.wikipedia.org/wiki/Odd%C4%9Blen%C3%AD_zodpov%C4%9Bdnost%C3%AD_(informatika). [cit. 2024-06-27].
11. *MIT App Inventor*. Online. Wikipedia. 2010. Dostupné z: https://en.wikipedia.org/wiki/MIT_App_Inventor. [cit. 2024-03-11].

12. *Dependency injection a softwarové architektury – Online kurz*. Online. IT Network. Dostupné z: https://www.itnetwork.cz/navrh/architektury-a-dependency-injection. [cit. 2024-03-11].
13. GADDIS, Tony a HALSEY, Rebecca. *Starting Out with App Inventor for Android*. Online. Pearson Education Limited, 2015. ISBN 978-1-292-08032-1. Dostupné z: https://opac.atmaluhur.ac.id/uploaded_files/temporary/DigitalCollection/MjM5NmFiZWU5MmQwOGE3Mzk1OTg5MzExNGQ0Y2M1ZDc3YWZjYjEwNA==.pdf. [cit. 2024-03-11].
14. GARLAN, David a SHAW, Mary. *An Introduction to Software Architecture*. Online, Vědecká práce. School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213-3890: Carnegie Mellon University Pittsburgh, 2018. Dostupné z: https://kilthub.cmu.edu/articles/journal_contribution/An_Introduction_to_Software_Architecture/6603365. [cit. 2024-03-11].
15. *Software architecture*. Online. Wikipedia. 2002. Dostupné z: https://en.wikipedia.org/wiki/Software_architecture. [cit. 2024-03-11].
16. *The MIT App Inventor: Our Tutorials!*. Online. MIT App Inventor. Dostupné z: https://appinventor.mit.edu/explore/ai2/tutorials. [cit. 2024-03-11].
17. *The MIT App Inventor Library: Documentation & Support*. Online. MIT App Inventor. Dostupné z: https://appinventor.mit.edu/explore/library. [cit. 2024-03-11].
18. WOLBER, David; ABELSON, Hal; SPERTUS, Ellen a LOONEY, Liz. *App Inventor 2: Create Your Own Android Apps*. Online. 2nd. O'Reilly Media, 2014. ISBN 978-1-491-90684-2. Dostupné z: http://www.appinventor.org/book2. [cit. 2024-03-11].
19. *Lingumi - animals*. Online. Lingumi. Dostupné z: https://lingumi.com/topics/animals. [cit. 2024-07-05].
20. *PNG ALL*. Online. PNG ALL. C2016-2024. Dostupné z: https://www.pngall.com/black-rectangle-png/. [cit. 2024-07-05].
21. *Flaticon*. Online. Flaticon. C2010-2024. Dostupné z: https://www.flaticon.com/free-icon/question-mark_5726716?term=question+mark&related_id=5726716. [cit. 2024-07-05].
22. *Wikimedia Commons*. Online. Wikimedia Commons. 2018. Dostupné z: https://commons.wikimedia.org/wiki/File:Redsquare.png. [cit. 2024-07-05].

Seznam příloh

Příloha 1 - Prezentace1_Uvod.pptx

Příloha 2 - Zadani1_Barvy.docx

Příloha 3 - Zadani2_Stopky.docx

Příloha 4 - Zadani3_Pexeso.docx

Příloha 5 - Zadani4_Pinball.docx

Příloha 6 - AktualizovanaMetodika&Harmonogramy.docx

Příloha 7 - PexesoMaterialy.zip

Příloha 8 - PinballMaterialy.zip

Úloha1_Barvy (odkaz)

Úloha1_BarvyDU (odkaz)

Úloha2_Stopky (odkaz)

Úloha2_StopkyDU (odkaz)

Úloha3_Pexeso (odkaz)

Úloha3_PexesoDU (odkaz)

Úloha4_Pinball (odkaz)

Úloha4_PinballDU (odkaz)

Seznam obrázků