FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Myšák - Editor pracovních listů
Student:	Kateřina Kasalická
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Myšák je týmový projekt zaměřený na vývoj výukové a diagnostické aplikace pro děti předškolního věku v prostředí WWW a OS Android. Tento projekt zahrnuje implementované pracovní listy, které jsou založeny na pracovních listech navržených psychology. Cílem této práce je vytvořit prototyp editoru pracovních listů.
1. Analyzujte pracovní listy psychologů zaměřené na diagnózu vývoje dovedností dětí předškolního věku.
2. Podrobte uživatelskému testování čtyři pracovní listy.
3. Analyzujte výsledky testování a navrhněte úpravy testovaných pracovních listů.
4. Pomocí metod softwarového inženýrství navrhněte editor pracovních listů pro pedagogy.
5. Pro prostředí WWW implementujte prototyp editoru pracovních listů. Co nejvíce využijte již implementované pracovní listy z týmového projektu.
6. Aplikaci podrobte vhodným testům (uživatelské, integrační).

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 2. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Myšák – Editor pracovních listů

Kateřina Kasalická

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Chludil

12. května 2019

Poděkování

V první řadě chci poděkovat svému vedoucímu práce Jiřímu Chludilovi za odborné rady a vedení celým procesem tvorby této práce. Dále chci poděkovat Ondřeji Brémovi za odborné rady k uživatelskému testování, všem účastníkům uživatelského testování a rodičům, kteří mi dovolili provést uživatelské testování s jejich dětmi. V poslední řadě děkuji velmi speciální skupině Čtvrtkohraní za psychickou a morální podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. května 2019 ……

Odkaz na tuto práci

Kasalická, Kateřina. *Myšák – Editor pracovních listů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce se zabývá návrhem a implementací editoru pro vytváření nových úrovní pro výukovou aplikaci Myšák, která se zaměřuje na děti předškolního věku. Tyto úrovně jsou založeny na pracovních listech od psychologů. Práce se nejdříve zabývá analýzou podkladových pracovních listů a na základě uživatelského testování s cílovou skupinou navrhuje změny v implementovaných úrovních.

Výuková aplikace je implementována v editoru Unity 3D. Proto se práce dále zabývá analýzou Unity scén. Dále analyzuje možné způsoby, jak vytvářet nové úrovně pro Unity aplikaci. V návrhové a implementační části práce podrobněji popisuje, jak vytvářet nové scény pro Unity aplikaci, a věnuje se tvorbě intuitivního uživatelského rozhraní editoru.

Klíčová slova editor, Unity, výuková aplikace, diagnostika, pracovní list, děti, uživatelské testování

Abstract

This thesis is concerned with the design and implementation of the editor for creation of new levels for educational application The Mouse for preschool children. These levels are based on worksheets from psychologists. First task of this thesis is to analyze the worksheets and perform user testing of the implemented levels with the target audience. Based on the user testing, the thesis suggests changes in the implemented levels.

The educational application The Mouse is implemented in editor Unity 3D. Therefore this thesis deals with the analysis of Unity scenes. The thesis analyzes possible ways to create new levels for the Unity application. In the design and implementation chapters thesis describes creation of new scenes for an Unity application and is concerned with the creation of intuitive user interface of the editor.

Keywords editor, Unity, educational application, diagnostics, worksheet, children, user testing

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Úvod

V dnešní době se moderní technologie dostávají do více a více částí našeho života. To se týká i životů těch nejmenších. Vznikají spousty aplikací, které jsou určeny pro zábavu dětí, které ještě neumí ani číst. Ne všechny tyto hry ale mají rozumnou náplň.

Projekt Myšák, na kterém jsem se podílela od jeho počátku, se zabývá právě vývojem aplikace pro děti předškolního věku. Tato aplikace obsahuje úrovně, které jsou založeny na pracovních listech od psycholožky. Díky tomu má tato aplikace pro děti vhodný obsah a zároveň (zatím nevyužitý) diagnostický potenciál. Aplikace navíc obsahuje možnost tvorby takzvaných scénářů. Scénář je libovolná posloupnost úrovní, kterou lze přiřadit dítěti. Díky tomu je možné kontrolovat obsah, který dítě hraje. Kromě aplikace projekt Myšák obsahuje i podpůrné systémy, konkrétně webovou aplikaci pro správu scénářů.

Díky scénářům a webové aplikaci pro správu je aplikace vhodná k použití i ve školkách nebo u psychologů. Potenciál tohoto projektu jsem se rozhodla rozšířit tím, že vytvořím editor pro tvorbu nových úrovní pro aplikaci Myšák. Ten potom umožní nejen vytvářet nové úrovně rychleji, ale zároveň poskytne možnost vytvářet nové úrovně prakticky komukoliv, například rodičům dětí, vychovatelkám ve školkách nebo psychologům.

Pracovní listy

Pracovní listy, které jsou podkladem pro úrovně v aplikaci Myšák, jsou původně vytvořeny pro diagnostiku vývoje schopností dětí předškolního věku. Tyto listy se zaměřují na různé oblasti vývoje dítěte, jako je například řeč, sluch nebo myšlení. Na základě toho, jak pracovní listy dítě vyplní, pak může psycholog zjistit, jak se u dítěte dané schopnosti rozvíjí a například i to, jestli je dítě připravené nastoupit do školy.

Vyplňování pracovních listů u psychologa v neznámém prostředí může být ale pro dítě stresující. Proto původně vznikla aplikace Myšák, díky které může

dítě plnit stejné úkoly jako v pracovních listech v pohodlí domova, kde se cítí jistěji a kde podá více vypovídající výsledky.

Tvorba nových úrovní

Náplní výsledné aplikace bude tvorba nových úrovní do výukové aplikace Myšák. Aplikace Myšák je vytvořena v editoru Unity 3D. Úkolem této práce bude tudíž přijít na to, jak vytvářet nové úrovně do aplikace vytvořené v editoru Unity, ale téměř bez Unity. To považuji za poměrně neprobádanou oblast a je to jeden z hlavních důvodů, proč jsem se rozhodla pro tuto práci. V důsledku toho totiž tato práce vyžaduje detailnější studium editoru Unity a následně vymyšlení vlastního způsobu, jak nové úrovně vytvářet.

Struktura práce

V analytické části práce je nejdříve vysvětlen kontext této práce, kterým je projekt Myšák. Dále jsou analyzovány pracovní listy, které sloužily jako podklad pro úrovně aplikace Myšák a na základě provedeného uživatelského testování jsou navrženy změny v implementovaných úrovních. Následující sekce jsou věnovány analýze požadavků, které jsou kladeny na editor úrovní a jeho případům užití. V poslední sekci analytické části je popsána struktura Unity scén.

Návrhová část je v první řadě zaměřena na zvolení vhodné metody pro generování nových Unity scén. Dále se zabývá návrhem úrovní aplikace Myšák, které bude možné tvořit ve vlastním editoru. Další sekce jsou věnovány návrhu backendové a frontendové části editoru. V poslední sekci je popsáno provedené uživatelské testování návrhů obrazovek frontendu.

V implementační části je popsán proces generování Unity scény. To zahrnuje Unity skripty, které dynamicky nastavují danou scénu, a backend, který generuje soubor Unity scény. Dále je ukázán výsledný frontend editoru a je popsáno jeho chování. V poslední řadě tato část zahrnuje instalační příručku a uvádí možnosti budoucího rozvoje editoru.

Poslední kapitola je věnována testování vytvořeného editoru a popisu testů, kterým byl implementovaný prototyp podroben.

Kapitola 1

Cíl práce

Cílem této práce je vytvořit editor pro tvorbu pracovních listu, který bude mít intuitivní uživatelské rozhraní vhodné i pro uživatele bez větších informatických znalostí. To zahrnuje tvorbu vlastního generátoru Unity scén, které budou kompatibilní s již vytvořenou OS Android aplikací Myšák. Tento generátor bude umožňovat nastavit atributy scény, to zahrnuje obrázek pozadí, audio instrukce, obrázky jednotlivých objektů a vztahy mezi nimi v závislosti na úkolu pracovního listu.

Cílem je také prozkoumat zaměření jednotlivých pracovních listů a optimalizovat je před implementací editoru, který bude umožňovat tvorbu jejich již optimalizovaných verzí. Cílem je této optimalizace dosáhnout s pomocí analýzy původních materiálů od psycholožky a provedení uživatelského testování již implementovaných pracovních listů a následnou analýzou jeho výsledků.

Tato práce si klade za cíl implementovat pro editor intuitivní uživatelské rozhraní, které bude srozumitelné pro běžné uživatele. Bude tudíž umožňovat i běžným uživatelům vytvářet si vlastní pracovní listy. Díky tomu bude možné editor využít i k tvorbě nového obsahu pro aplikaci Myšák, která bude časově efektivnější než tvorba v editoru Unity 3D.

Kapitola 2

Analýza

2.1 Současný stav projektu Myšák

Projekt Myšák se zaměřuje na děti předškolního věku. V rámci předmětů BI-SP1 a BI-SP2 jsme s týmem vytvořili několik modulů, které jsou součástí projektu Myšák. V této sekci nejdříve budou uvedeny pojmy, které budou požívány v této práci a jejich synonyma používaná napříč projektem Myšák.

Pracovní list – odpovídá jedné minihře. Je to nějaký konkrétní úkol, například třídění objektů mezi auto a loď.

Synonyma: worksheet, úroveň, task setting, minihra, konfigurace úkolu, level

Typ úlohy – určuje nějaké obecné zadání pro pracovní listy, příkladem typu úlohy může být třídění objektů, kde pod tento typ bude spadat pracovní list s úkolem roztřídit objekty mezi loď a auto.

Synonyma: task, typ, úkol

Kategorie – je oblast dovedností, která je u dítěte sledována. Pod kategorii potom spadají různé typy úloh, například pod kategorii *Myšlení a řeč* spadá typ úlohy třídění nebo spojování dvojic.

Synonyma: category, task category

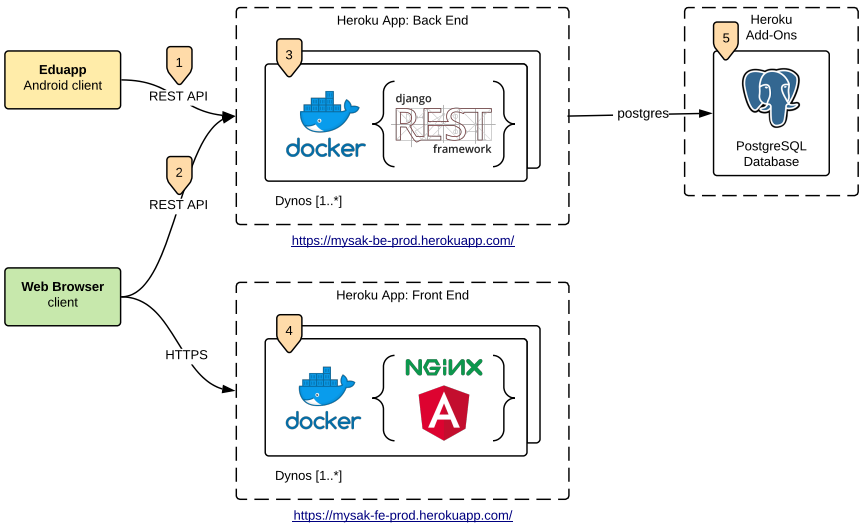
Scénář – je specialita projektu Myšák. Scénář je posloupnost pracovních listů. Například lze sestavit scénář pro základní otestování schopností dítěte, do kterého bude zařazen jeden pracovní list od každého typu úlohy. Nebo se naopak lze zaměřit na jednu oblast a sestavit scénář jen z pracovních listů zaměřených na třídění objektů.

Synonyma: scenario

Fronta scénářů – je posloupnost scénářů, které byly dítěti přiřazeny. Dítě potom hraje jednotlivé scénáře v pořadí, v jakém mu byly přiřazeny.

Synonyma: scenario queue

V současnosti se projekt skládá z výukové aplikace a serveru, který se skládá z backendu a frontendu, jak lze vidět na obrázku 2.1. Z backendu jsou zatím implementovány pouze endpointy pro frontend, endpointy pro výukovou aplikaci jsou zatím pouze navrženy, ale nejsou implementovány. Frontend vzniká současně s touto prací v bakalářské práci Radka Ježka [4].



Obrázek 2.1: Diagram modulů projektu Myšák [5]

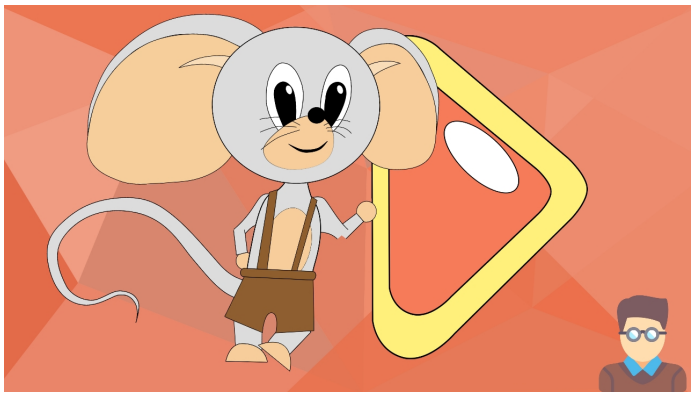
2.1.1 Výuková aplikace Myšák

Výuková aplikace Myšák pro OS Android byla vytvořena ve spolupráci s psycholožkou pro děti předškolního věku. Aplikace obsahuje implementované pracovní listy, které jsou založeny na pracovních listech od psycholožky. Tyto pracovní listy slouží k analýze rozvoje schopností dítěte.

Jelikož se jedná o aplikaci pro děti předškolního věku, které ještě neumí číst, místo textových vysvětlivek aplikace využívá k předání instrukcí a nápověd mluvených nahrávek. Děti celou hrou provádí postava Myšáka (na obrázku 2.2), který dětem na začátku každého pracovního listu sděluje jejich úkol a průběžně je chválí za správné kroky a upozorňuje na chybné tahy.

V aplikaci jsou dva možné módy. Prvním je volná hra, kde si dítě v menu vybírá z nabídky pracovních listů. Druhým módem je hraní scénáře z fronty

scénářů dítěte. Pokud má dítě ve frontě scénářů přiřazený alespoň jeden scénář, tak scénář, který je na řadě, se dítěti automaticky spustí po přihlášení do aplikace a odstraní se z fronty scénářů až po jeho dohrání. Dítě má možnost si znovu zahrát volnou hru, až když vyprázdní svoji frontu scénářů.



Obrázek 2.2: Úvodní obrazovka aplikace Myšák home [3] (snímek pořídila autorka práce)

V obou módech se z odehraných pracovních listů sbírá statistika dítěte. Tato statistika obsahuje procentuální úspěšnost dítěte a čas, po který dítě pracovní list plnilo. Tato statistika je zatím v aplikaci pouze ukládána a aplikace ji nijak nezobrazuje.

Aplikace byla vyvinuta v editoru Unity 3D [6], který pracuje s takzvanými scénami (podrobněji viz [7]). V aplikaci Myšák jeden pracovní list odpovídá právě jedné Unity scéně.

Ve skutečnosti existují dvě verze aplikace, a to verze home a education. Verze home obsahuje vlastní menu pro management uživatelů a scénářů, pro verzi education se o tento management stará server, který s aplikací komunikuje přes REST API. Obě tyto verze ale obsahují stejné pracovní listy a pracují se stejným formátem pracovních listů, proto dále nebudou rozlišovány.

2.2 Pracovní listy

Výuková aplikace Myšák obsahuje implementované pracovní listy, které se zakládají na pracovních listech navržených psycholožkou Simonou Pekárkovou v [8]. Tyto pracovní listy se zaměřují na různé oblasti schopností, které se vyvíjejí u dětí předškolního věku. Jinými slovy pomáhají s diagnostikou jejich schopností a mimo jiné i se zjištěním, jestli jsou již připravené na nástup do školy.

Jednou z oblastí, na kterou se aplikace Myšák zaměřuje, je *Myšlení a řeč*. K nejzásadnějšímu vývoji v této oblasti u dětí dochází během prvních tří let života. Pokud dítě v tomto období není vystaveno stálému kontaktu s druhými

a jejich řeči, v pozdějším věku může mít nižší jazykové schopnosti. Pracovní listy zaměřené na tuto oblast nám mohou pomoct tyto problémy odhalit již v raném věku. Například předškolák by měl zvládnout:

- popsat příběh podle obrázků;
- převyprávět příběh, který zná (například pohádku);
- roztřídit předměty do kategorií (rozlišit nadřazené a podřazené pojmy);
- přiřadit věci, které k sobě patří;
- poznat protiklady a pojmenovat je;
- rozpoznat nesmyslné detaily či obrázky a vysvětlit je. [8]

Obrázek 2.3: Pracovní list *Loď a auto* [8]

Obrázek 2.4: Implementace pracovního listu *Loď a auto* [3]

Například na obrázku 2.3 je pracovní list *Loď a auto*, který se zaměřuje na zjištění toho, jestli dítě dokáže předměty roztřídit do kategorií. Dítě má za úkol v něm pospojovat předměty buď s autem, nebo lodí, podle toho, kam patří. Ve výukové aplikaci Myšák jsme spojování nahradili přetahováním předmětů na jejich kategorie. Implementaci pracovního listu *Loď a auto* vidíme na obrázku 2.4. Tento implementovaný pracovní list řadíme do kategorie *Myšlení a řeč* a do typu úlohy *Třídění*. Jak název napovídá, typ úlohy *Třídění* spočívá v třídění předmětů mezi kategorie, kde konkrétní pracovní listy tohoto typu úlohy se liší jenom v předmětech a kategoriích, konkrétně v jejich počtu a obrázcích.

Dalším příkladem na obrázku 2.5 je pracovní list *Kdo co dává?* který zjišťuje, jestli dítě dokáže přiřadit věci, které k sobě patří. Dítě zde má za úkol spojit dvojice podle toho, které zvíře co dává. Analogicky jsme u implementace tohoto pracovního listu nahradili spojování přetahováním dvojic na sebe, které funguje na obě strany. Tato implementace je na obrázku 2.6. Tento pracovní list řadíme do kategorie *Myšlení a řeč* do typu úlohy *Dvojice*. U tohoto

Obrázek 2.5: Pracovní list *Kdo co dává?* [8]

Obrázek 2.6: Implementace pracovního listu *Kdo co dává?* [3]

typu úlohy se u jednotlivých pracovních listů opět mění pouze počet objektů a jejich vzhled.

Dalšími implementovanými typy úloh založených na základě předlohy od psycholožky ve výukové aplikaci jsou *Poznej objekt* z kategorie *Myšlení a řeč* a *Počáteční písmeno* z kategorie *Sluchové vnímání*. U typu *Poznej objekt* má dítě za úkol na základě mluveného popisu z množiny předmětů vybrat ten, který popisu nejlépe odpovídá. Typ *Počáteční písmeno* spočívá v rozpoznání toho, jestli dvojice slov začíná na stejné písmeno.

2.3 Uživatelské testování

Uživatelské testování pomáhá určit, jestli je design produktu navržen srozumitelně pro cílovou skupinu produktu. Pomocí zpětné vazby potenciálních uživatelů lze zjistit, jestli předpoklady při návrhu produktu byly správné a případně, jak slabá místa produktu zlepšit. [9]

Princip uživatelského testování spočívá v pozorování uživatele při používání testovaného produktu. Potom na základě získaných dat, jak už získaným pozorováním či vyzpovídáním uživatele, lze identifikovat, které části produktu jsou pro uživatele neintuitivní, a případně i zjistit úspěšnost uživatelů při používání produktu. Uživatelské testování lze rozdělit do několika fází:

1. **Rozhodnout, jaká část produktu se bude testovat.** Není nutné testovat celý produkt najednou, stačí se zaměřit na část, kterou je považována za problémovou. Příliš dlouhé testování neposkytuje tak realistické výsledky, protože může být pro účastníka testování únavné. A proto je vhodnější produkt testovat postupně po menších částech. [10]

2. **Určit úkol**, který má účastník splnit. Měl by to být obvyklý úkon uživatelů produktu.

3. **Vytvořit očekávání**, jak bude účastník postupovat. Je důležité si určit, jak by měl podle našeho očekávání uživatel postupovat při plnění úkolu.

Potom lze toto očekávání porovnat s výslednými daty a na základě počtu úspěchů a selhání určit, jestli je rozhraní produktu pro uživatele intuitivní.

4. **Vytvořit scénář**, na základě kterého bude testování probíhat. Na začátku by měl být účastníkům sdělen kontext k produktu a účel testování. Pro detailnější následnou analýzu může být vhodné se účastníků optat na jejich znalost testovaného produktu, případně produktů se stejným zaměřením. Je důležité účastníkům vysvětlit, že předmětem testovaní je produkt a nikoliv oni, tudíž neexistují žádné špatné odpovědi. Poté seznámíme účastníky s jejich úkolem. Pokud chceme konzistentní výsledky, všichni moderátoři by se měli držet scénáře.

5. **Určit místo a čas**, kde bude testování probíhat. Jednou z možností může být specializovaná laboratoř, kterou ovšem nemusí mít každý k dispozici. Důležité je, aby to bylo poklidné místo, kde se můžou účastníci uvolnit, a kam se vejde moderátor a případní pozorovatelé a nahrávací vybavení. Poznámky přímých pozorovatelů a záznam nahrávání poskytne přesnější data, ovšem může to být na úkor pohodlí účastníků. Pokud se rozhodneme testování jakkoliv nahrávat, je nutné získat od účastníků svolení.

6. **Určit role** členů týmu. Moderátor musí zůstat neutrální, držet se striktně scénáře a zároveň být příjemný k účastníkům. Na pozici pozorovatele se hodí dobrý posluchač a ideálně někdo, kdo rozumí řeči těla.

7. **Nalézt účastníky**, kteří odpovídají cílové skupině produktu. Testování bude efektivnější, pokud bude testování provedeno s menší skupinou účastníků (okolo pěti), kteří alespoň zhruba odpovídají cílové skupině produktu, než když bude produkt testován s velkým množstvím náhodných účastníků. Také není vhodné, aby účastníci znali členy testovacího týmu, jelikož účastníci pak nemusí být schopni produkt posoudit objektivně a otevřeně ho kritizovat.

8. **Provést testování**, přičemž moderátor se striktně drží navrženého scénáře. Je důležité neposkytovat účastníkům žádné vedení, neodpovídat na jejich případné dotazy a ani výrazy obličeje a řečí těla nedávat najevo svoje emoce. Pokud se například zeptají „Co se stane, když kliknu na tohle?“ je dobré například odpovědět „Co myslíte, že se stane?“. Lze požádat účastníka, aby přemýšlel nahlas a okamžitě sděloval svoje úvahy a dojmy, případně ho po testování požádat o zpětnou vazbu nebo mu dát k vyplnění dotazník. Ovšem okamžitá zpětná vazba podá nejčerstvější dojmy účastníků.

9. **Analyzovat data** získané z poznámek a případných záznamů a dotazníků. Při analyzování dat je nutné věnovat pozornost nejen úspěšnosti

účastníků, ale i jejich pocitům a tomu, jak na ně produkt působí. Určíme problémová místa a jejich důležitost, a na základě toho navrhneme úpravy produktu.

10. **Vytvořit zprávu**, která bude obsahovat další kroky ke zlepšení produktu a očekávání toho, jak by tyto kroky měly testovaný produkt vylepšit. [10, 11]

Na úplném začátku plánování bychom měli rozhodnout, jaká data chceme pomocí testování získat. Pokud chceme zjistit především kvalitativní data, neboli odhalit slabá místa produktu, stačí testovat s menším počet účastníků, které ovšem musíme podrobit detailnějšímu pozorování, abychom zachytili co nejvíce informací. [11]

Naopak pro získání kvantitativních dat, neboli průměrné úspěšnosti uživatelů při plnění jednotlivých úkolů potřebujeme větší počet účastníků. Abychom získali co nejpřesnější data, účastníci by měli být pod co nejmenším stresem, tudíž nejspíš zvolíme méně monitorované prostředí. Kvantitativní data se mohou hodit v případě, pokud chceme například náš produkt porovnat s konkurencí nebo se rozhodnout pro jeden z více možných designů. [11]

2.3.1 Testování pracovních listů v rámci dne dětí na FITu

Dne 29. května 2018 se v SAGElabu na FIT ČVUT konal den dětí pro zaměstnance fakulty. Tuto příležitost jsme se Zuzanou Václavikovou využily k tomu, abychom otestovaly prototyp aplikace Myšák, která v té době obsahovala pouze dva implementované pracovní listy, a to *Loď a auto* a *Počáteční písmeno*.

2.3.1.1 Závěry testování

Na základě našich poznámek jsme u jednotlivých pracovních listů zaznamenaly tyto problémy:

Loď a auto

- „Roztřiď“ není úplně jasný pokyn. Děti nevědí, co přesně mají dělat.
- Děti nerozumí označení „objekty“.

Počáteční písmeno

- Instrukce „podej mi kartičky“ je nejasná. Děti neví, že je mají podat Myšákovi.
- Čekají, že když se dotknou levé strany kartičky, přečte se pouze název na levém obrázku a dotekem pravého obrázku se přehraje název pravého obrázku.

Potom u obou pracovních listů jsme zaznamenaly, že dabing je pro děti příliš zdlouhavý a často nemají trpělivost si poslechnout celé zadání. A pokud spustí nějakou jinou hlášku, například pochvalu, nebo naopak upozornění na chybu, přeruší to instrukce a děti se nedozví celé zadání.

2.3.1.2 Hodnocení

Zpětně vidíme, že došlo při testování k několika chybám:

- neměly jsme dopředu připravený jasný scénář, proto testování s každým dítětem probíhalo jinak;
- naše stanoviště se nacházelo v místnosti s třemi dalšími stanovišti pro děti, takže hluk v místnosti často přehlušil zadání jednotlivých úkolů a děti jim v důsledku toho nemohly rozumět;
- některé děti byly mladší, nebo i starší než naše cílová skupina;
- občas do testování zasahovali rodiče a ostatní děti.

Testování sice proběhlo v prostředí, na které jsou děti zvyklé a ve kterém se mohly uvolnit, ale ostatní negativní faktory mohly nepředvídatelně zkreslit naše data. Nicméně získaná data lze využít v dalším testování, kde se můžu zaměřit na tyto potencionální slabá místa.

2.3.2 Testování pracovních listů v mateřské škole

Dne 27. 1. 2019 jsem moderovala uživatelské testování pracovních listů v mateřské škole Lvíčata. Testování jsem připravila a provedla podle předchozí kapitoly o uživatelském testování a zaměřila jsem se na získání kvalitativních dat. Testování se zúčastnili celkem 4 děti ve věku 5 až 6 let, které měly menší zkušenosti s tabletem nebo mobilem. Testování jsme natočili na dvě kamery a zaznamenali postup dětí nahráváním obrazovky.

Jelikož se jednalo o testování s dětmi, obvyklý postup uživatelského testování jsem musela značně přizpůsobit. Testování jsem se rozhodla provést ve školce, aby děti byly ve známém prostředí a cítily se pohodlněji. Taky jsem se snažila minimalizovat počet dospělých v místnosti, aby se děti necítily tolik v menšině. Na začátku jsem se dětí vždy optala na jejich zkušenosti s tabletem nebo mobilem a snažila se vést krátkou konverzaci, aby se děti mohly nejdříve uvolnit a zvyknout si na svoje prostředí. Po testování jsem dětem dala k vyplnění krátký dotazník, ve kterém zhodnotily jednotlivé pracovní listy. Dotazník jsem přizpůsobila tomu, že děti neuměly číst, a tak každý pracovní list hodnotily tím, že vybarvily jeden ze škály smajlíků, od smějícího se po plačícího.

Obrázek 2.7: Uživatelské testování pracovních listů ve školce (snímek pořídila autorka práce)

2.3.2.1 Závěry

Všechny děti nakonec splnily zadání, některé ovšem potřebovaly v průběhu menší radu. Po analýze záznamů a poznámek z testování jsem nalezla tyto problémy:

- Ze začátku dětem není jasné, že mají předměty přetahovat prstem.
- Když průvodce Myšák dítě požádá, aby mu něco podalo, tak dítě neví, co má dělat.
- U složitějších úkolů, jako je například *Počáteční písmeno*, mají problém najednou pochopit celé zadání.
- Děti si nejsou jisté, jestli postupují správně, pokud aplikace na jejich tah nijak nereaguje.

Lze vidět, že se potvrdila některá podezření z předchozího testování. Naopak například tentokrát děti neměly vůbec problém s tím, že by si nedoposlechly celé zadání a ukvapeně jednaly. To si vysvětluji rozdílným prostředím, konkretně v předchozím testováním byly děti roztěkané a občas neslyšely zadání přes hluk v místnosti. Celkově i přes to, že děti tentokrát byly pod větším tlakem, které bylo poznat v jejich nejistotě, věřím, že toto testování nám poskytlo pravdivější a přesnější data.

Co se týče zpětného hodnocení postupu uživatelského testování, děti byly při testování zjevně pod velkým tlakem a působily velmi nejistě. Z toho důvodu bych příště zvažovala možnost testovat aplikaci spíše s dvou či tři člennými skupinkami dětí. Děti by se pak mohly cítit jistěji a navíc by spolu mohly diskutovat, díky čemu bychom mohli lépe zaznamenat to, jak uvažují při řešení jednotlivých pracovních listů a zjistit příčiny největších problémů.

2.3.3 Navrhované změny

Na základě prvních třech poznatků z uživatelského testování ve školce a několika dalších bodů z předchozího testování, které naznačují potíže s pochopením zadání, pokud děti hrají pracovní listy poprvé, bych navrhla do aplikace přidat tutoriál. Tento tutoriál by se automaticky spustil každému novému uživateli. Nejdříve by se v něm představil Myšák, aby si děti jednoznačně spojily postavu a hlas Myšáka. Dále by Myšák děti krok po kroku provedl jednotlivými typy úloh a vizuálně jim ukázal, že mají objekty přetahovat prstem, kdy mají předměty podávat jemu atd.

Co se týče čtvrtého bodu, při tvorbě aplikace Myšák jsme původně chválení dítěte po každém tahu zamítli, protože nám připadalo po čase příliš otravné. Nicméně na základě testování bych navrhovala do aplikace chválení po každém tahu vrátit, aby děti měly po každém tahu zpětnou vazbu aplikace.

V závěrech prvního testování jsou také zmíněny drobnější problémy s dabingem. Proto navrhuji se v budoucnosti více zaměřit na terminologii, kterou používá cílová skupina pracovních listů. Například místo pojmu „objekt“ by pro děti mohl být přívětivější pojem „předmět“ nebo „věc“.

2.4 Požadavky

Na editor pro tvorbu pracovních listů jsou kladeny následující **funkční** požadavky:

F1 Zvolení typu úlohy – každý pracovní list spadá pod nějaký typ úlohy. Uživatel si bude moci zvolit, jakého typu úlohy tvořený pracovní list bude.

F2 Nastavení pozadí pracovního listu – uživatel by měl mít možnost měnit pozadí pracovního listu. Editor by měl umožňovat nahrát vlastní obrázek a nastavit ho jako pozadí vytvářeného pracovního listu.

F3 Vložení objektu do pracovního listu – pracovní list každého typu úlohy obsahuje nějaké typy objektů, například *Třídění* obsahuje tříděné objekty a objekty reprezentující kategorie, do kterých jsou objekty tříděny třídíme. Editor by měl umožňovat vytvářet nové objekty všech typů, které aktuální typ úlohy obsahuje.

F4 Nastavení atributů objektu – objekty pracovního listu mohou mít různé atributy v závislosti na typu úlohy. Například u *Třídění* u tříděných objektů je potřeba nastavit, do jakých kategorií jednotlivé tříděné objekty patří.

F5 Nastavení obrázku objektu – každý objekt musí mít nějaký obrázek, který ho reprezentuje. Uživatel by měl mít možnost nahrát vlastní obrázek a přiřadit ho vytvořenému objektu.

F6 Nastavení pozice objektu – uživatel by měl mít možnost měnit rozložení objektů v pracovním listů tím, že bude měnit pozici jednotlivých objektů.

F7 Vložení audio instrukcí – každý pracovní list musí obsahovat audio instrukce, které na začátku dítěti sdělí, co je jeho úkolem. Uživatel by měl mít možnost nahrát vlastní audio instrukce, které se přehrají na začátku pracovního listu. Pokud daný typ úlohy vyžaduje i další audio nahrávky, editor by měl umožňovat jejich přidání.

F8 Vložení výchozích audio instrukcí – pokud uživatel nenahraje vlastní audio instrukce, editor by měl do pracovní listu vložit výchozí audio instrukce, které budou univerzální pro daný typ úlohy. To se ovšem týká jen případů, kde je to možné. Například pokud pracovní list bude vyžadovat přečtení názvu obrázku, editor nemá jak toto univerzálně vyřešit.

F9 Spuštění pracovního listu v editoru – uživatel si bude moci pracovní list po jeho vytvoření zahrát přímo v editoru, aby si vyzkoušel jeho funkčnost.

Nefunkční požadavky:

N1 Dostupnost přes web – editor bude dostupný jako webová aplikace zobrazitelná v internetovém prohlížeči.

N2 Kompatibilita s výukovou aplikací Myšák – pracovní listy vytvořené editorem budou kompatibilní s výukovou aplikací Myšák. To znamená, že vygenerované pracovní listy budou spustitelné po přidání do aplikace. Budou zaznamenávat statistiku ve formátu, v jaké jsou v současnosti zaznamenávány a budou obsahovat nápovědu a možnost opustit pracovní list tak jako v již implementovaných pracovních listech.

N3 Kompatibilita se serverem Myšák – pracovní listy vytvořené editorem budou kompatibilní se serverem projektu Myšák. Konkrétně s jeho databází pracovních listů.

Kromě editoru jsou kladeny požadavky i na ostatní moduly projektu Myšák. To z toho důvodu, že po vytvoření pracovního listu v editoru bude potřeba tento pracovní list dále distribuovat do výukové aplikace pomocí serveru, aby si ho děti mohly zahrát. Proto následují funkční požadavky na projekt Myšák:

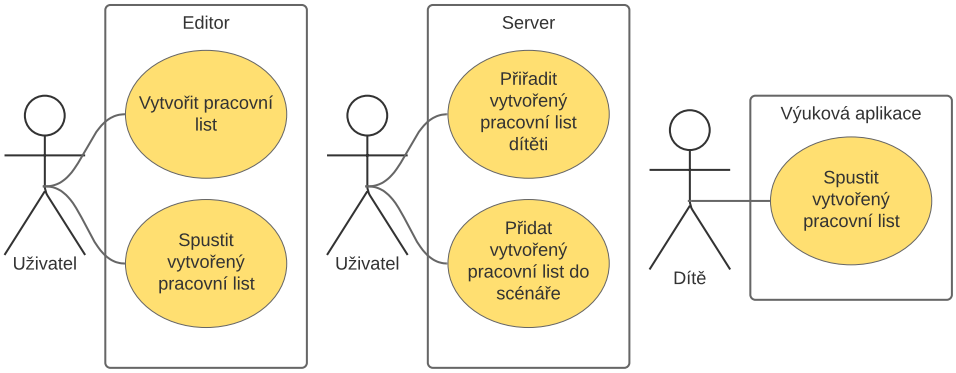
F10 Přiřazení pracovního listu profilu dítěte – uživatel může vytvořený pracovní list přiřadit dítěti. Po přihlášení dítěte do aplikace se vybraný pracovní list stáhne ze serveru a dítě si bude moci vytvořený pracovní list zahrát.

F11 Přidání pracovního listu do scénáře – vytvořený pracovní list bude možné přidávat do scénářů. Pokud bude scénář s daným pracovním listem přiřazen dítěti, vytvořený pracovní list se automaticky stáhne výukovou aplikací ze serveru po přihlášení daného dítěte.

Tyto požadavky ovšem vyžadují změny v modulech projektu Myšák, které nejsou součástí této práce, proto není náplní této práce je splnit.

2.5 Případy užití

Mezi potenciální uživatele editoru pracovních listů patří hlavně učitelky mateřských škol, příbuzní dětí, kteří hrají výukovou aplikaci Myšák, a případně i psychologové, kteří mohou využít diagnostického potenciálu pracovních listů.



Obrázek 2.8: Model případů užití

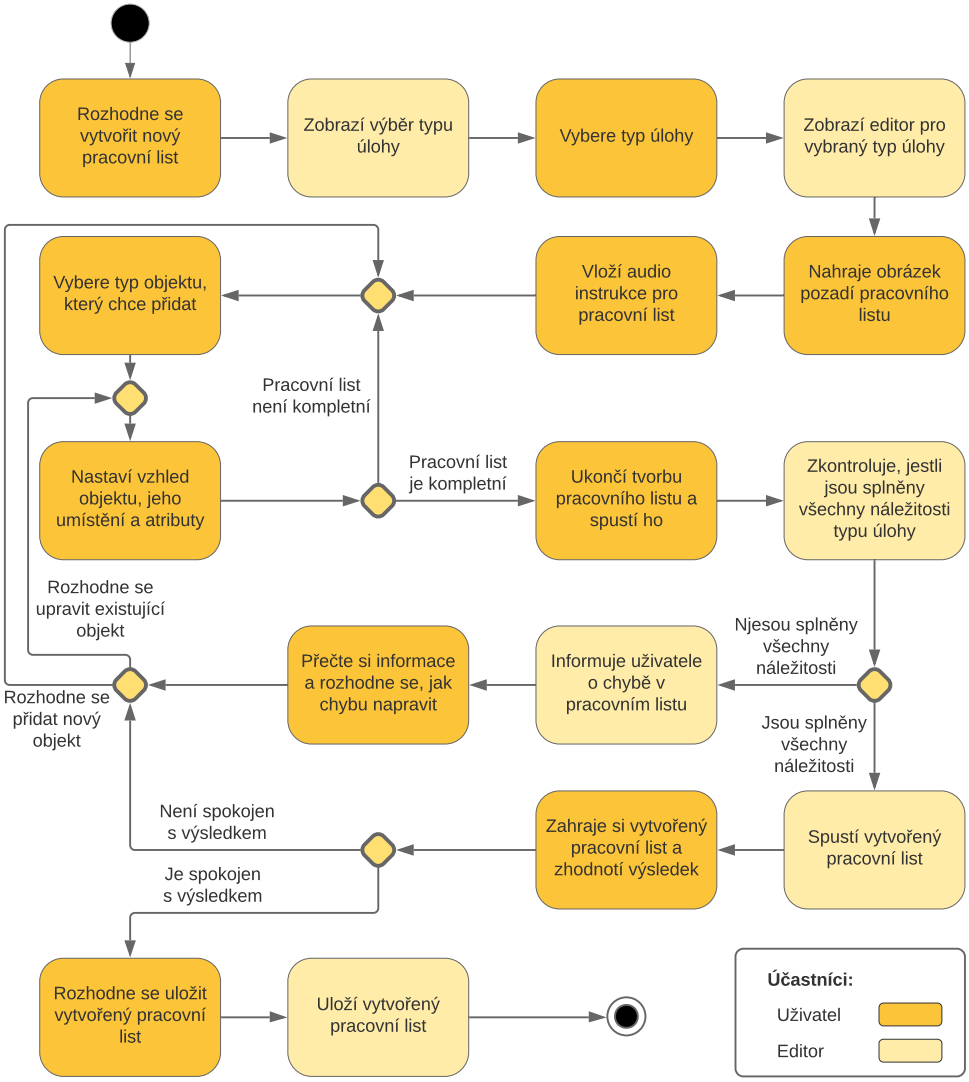
Uživatel bude editor využívat k vytvoření pracovního listu pro nějaké konkrétní dítě nebo skupinu dětí, nejspíš aby vytvořil pracovní list tématicky na míru dítěte. Pomocí editoru může uživatel vytvořit pracovní list, který dítě zaujme tématem, o které se dítě zajímá. Toho docílí právě tím, že zvolí vhodný vzhled pro jednotlivé objekty pracovního listu.

Při vytvoření nového pracovního listu si uživatel nejdříve zvolí z nabídky typů úloh, který chce implementovat. Potom v editoru nastaví vzhled jednotlivých objektů, vztahy mezi nimi, pozadí pracovního listu a volitelně může vložit vlastní audio instrukce.

Po vytvoření pracovního listu by uživatel měl mít možnost si ověřit jeho funkčnost přímo v editoru. Scénář obou případů užití je zachycen v diagramu aktivit 2.9.

Na základě požadavků jsou zvažovány tři další případy užití ostatních modulů projektů myšák (výuková aplikace a server), a to přiřazení pracovního listu dítěti, po kterém si dítě může pracovní list zahrát ve volné hře. Nebo může

uživatel vytvořený pracovní list přidat do scénáře, se kterým se pak může zacházet jako s obvyklým scénářem. Ovšem implementace těchto případů užití by zasahovala do celého projektu Myšák a do modulů, které v současnosti nejsou plně funkční, a proto by jejich implementace byla příliš náročná. Proto se jimi tato práce nebude dále zabývat, ale je důležité je zmínit k uvědomění si celého potenciálu editoru.



Obrázek 2.9: Diagram aktivit případů užití editoru

2.6 Unity scéna

V aplikaci Myšák odpovídá jeden pracovní list právě jedné scéně v Unity 3D a tato scéna je uložena v právě jednom souboru. Kromě binárního formátu Unity 3D umožňuje tyto soubory ukládat v textové podobě, která je implementována pomocí formátu YAML. V této sekci se budu věnovat formátu, ve kterém se Unity scény ukládají, protože jeho pochopení je klíčové pro implementaci editoru. [2]

2.6.1 YAML

YAML (rýmuje se s anglickým „camel“) je serializační jazyk navržený tak, aby byl kompatibilní s moderními programovacími jazyky. YAML se využívá hlavně v konfiguračních souborech. Hlavními rysy je jeho jednoduchost a čitelnost. Nejnovější verze YAMLu je 1.2, pro tuto práci je ale relevantní verze 1.1, kterou se budu nadále zabývat. [12]

Znakem YAMLu je, že pro určení struktury a hierarchii používá indentaci (předsazení), konkrétně indentace o jednu úroveň odpovídá dvěma mezerám. K oddělení klíčových slov a jejich hodnoty využívá dvojtečky. Pomlčky se používají pro jednotlivé položky seznamů (neboli neasociativních polí). Tyto znaky lze vidět ve výpisu kódu 2.1.

```
martin:
  name: Martin D'vloper
  job: Developer
  skills:
    - python
    - perl
    - pascal
```

Výpis kódu 2.1: Ukázka YAML dokumentu [1]

Neasociovaná pole (seznamy) lze zapsat i do hranatých závorek jako ve výpisu kódu 2.2.

```
[python, perl, pascal]
```

Výpis kódu 2.2: Zápis neasociovaného pole v YAMLu

Takzvaná asociovaná pole (dvojice klíče a hodnoty) lze zapsat i dovnitř složených závorek jako ve výpisu kódu 2.3. Jednotlivé položky jsou opět zadány

jako dvojice klíče a hodnoty, položky jsou oddělené čárkou. Asociovaná pole jsou způsobem, jak zapsat více dat na jednu řádku.

```
martin: {name: Martin D'vloper, job: Developer}
```

Výpis kódu 2.3: Zápis asociovaného pole v YAMLu

Každá jednotlivá položka celého YAML souboru je považována za dokument. Pomocí sekvence `---` se oddělují jednotlivé dokumenty, jak lze vidět ve výpisu kódu 2.4. [12]

```
---
fruits:
  - Apple
  - Banana
  - Raspberry
---
vegetables:
  - Lettuce
  - Carrot
---
```

Výpis kódu 2.4: Ukázka oddělovací sekvence YAMLu

2.6.2 Formát scény

Unity soubory scén v textové formě ukládá ve formátu YAML. V tomto souboru je uloženo celé nastavení dané scény a všechny její objekty. Každý objekt scény v tomto souboru odpovídá jednomu YAML dokumentu, který začíná `---` sekvencí. Příklad uložení objektu typu *GameObject* je zobrazen ve výpisu kódu 2.5. [2]

Lze vidět, že oproti obvyklému YAML dokumentu, soubor Unity scény na začátku dokumentu obsahuje navíc sekvenci `!u!1 &6`. Sekvence `!u!1 &6` obsahuje `!u!`, což indikuje třídu, v tomto případě třídu *GameObject*, která odpovídá číslu 1. Číslo následující za & je ID objektu, které je unikátní pouze v souboru konkrétní scény. To znamená, že není potřeba řešit jejich unikátnost napříč scénami. Jednotlivé objekty scény, neboli dokumenty YAML souboru, na sebe mohou přes tyto ID odkazovat. V tomto případě se ve výpisu kódu 2.5 dokument na řádcích 8 až 11 odkazuje na 4 další dokumenty. Právě hodnotou pod klíčem `fileID` se odkazuje na další dokumenty souboru. [2]

```
--- !u!1 &6
GameObject:
  m_ObjectHideFlags: 0
  m_PrefabParentObject: {fileID: 0}
  m_PrefabInternal: {fileID: 0}
  importerVersion: 3
  m_Component:
  - component: {fileID: 8}
  - component: {fileID: 12}
  - component: {fileID: 13}
  - component: {fileID: 11}
  m_Layer: 0
  m_Name: Cube
  m_TagString: Untagged
  m_Icon: {fileID: 0}
  m_NavMeshLayer: 0
  m_StaticEditorFlags: 0
  m_IsActive: 1
```

Výpis kódu 2.5: Uložení objektu GameObject v souboru Unity scény [2]

Jednotlivé dokumenty se mohou odkazovat i na soubory, které nejsou jinak uvedeny v daném souboru scény. Typicky se jedná o obrázky, audio klipy a skripty. Ty jsou uloženy v databázi projektu a Unity jim automaticky přiřazuje ID, přes které se na ně lze odkazovat ze souboru scény. Na tyto soubory jednotlivé dokumenty vždy odkazují klíčovým slovem `guid`, jak lze vidět ve výpisu kódu 2.6. Tato `guid` ale Unity editor těmto souborům přiřazuje pouze za běhu Unity editoru, nikoliv za běhu Unity aplikace. Proto se nelze tímto způsobem na tyto soubory odkazovat, pokud jsou přidávány za běhu Unity aplikace. [13]

```
m_EditorHideFlags: 0
  m_Script: {fileID: 11500000, guid: dcdc07e38a52168419f4f866
    662a4141, type: 3}
  m_Name:
```

Výpis kódu 2.6: Ukázka odkazu na externí soubor v souboru scény [3]

2.6.3 Komponenty Unity scény

Unity scéna obsahuje různé komponenty a objekty, které poté nalezneme i ve výsledném souboru dané scény. V textovém souboru scény jsou komponenty a objekty uloženy na stejné úrovni a bez znalosti Unity je nelze rozlišit. V editoru Unity jsou ovšem jednoznačně rozlišeny. Zatímco objekty mohou v Unity scéně existovat samostatně, komponenty jsou vždy připnuty k nějakému objektu a většinou nějak definují chování daného objektu. Tato sekce se bude věnovat těm komponentám a objektům, se kterými se lze setkat v pracovních listech aplikace Myšák, a se kterými bude tudíž muset editor pracovních listů pracovat.

MonoBehaviour je základní třída, od které dědí všechny skripty (skripty jsou vždy komponenty, ale komponenty nejsou nutně skripty dědící od monobehaviour). Musí od ní dědit všechny skripty, které uživatel v editoru Unity vytvoří. Tyto skripty pak lze připínat k objektům Unity scény, pak tento skript může modifikovat chování daného objektu. V souboru scény tuto třídu indikuje číslo 114, které je potom stejné pro všechny potomky (dědící skripty). [14]

Camera zachycuje obrazovku tak, jak ji vidí uživatel výsledné aplikace. Zpravidla musí být přítomna v každé scéně. Proto je vhodné k ní připínat skripty, které mají za úkol ovlivňovat chování celé scény. Tuto třídu indikuje číslo 20. [15]

GameObject je generický objekt scény. Používá se pro téměř všechny objekty, které ve většině scén najdeme (výjimkou je například kamera). Sám o sobě tento objekt nemá žádné chování, ani zobrazení. Tuto třídu indikuje číslo 1. [16]

Transform je komponenta, která je automaticky připnuta ke každému objektu třídy `GameObject`. Určuje pozici, rotaci a škálování daného objektu. Tuto třídu indikuje číslo 4. [17]

SpriteRenderer je komponenta, která určuje vzhled objektu. Může obsahovat obrázek, který pak definuje vzhled objektu. Velikost výsledného obrázku se vypočte na základě původní velikosti obrázku a škálování, které je uvedeno v komponentě *Transform*. Tuto třídu indikuje číslo 212. [18]

BoxCollider2D je komponenta, která se využívá pro simulaci fyziky u objektů. Konkrétně lze s její pomocí například vyhodnotit, jestli spolu dva objekty s komponentou `BoxCollider2D` kolidují. Tuto třídu indikuje číslo 61. [19]

Unity scény pracovních listů obsahují i další komponenty, které nebyly v této kapitole zmíněny. Určují totiž základní nastavení scény, které bude

pro všechny vygenerované pracovní listy stejné. Tudíž editor pracovních listů s nimi nebude nijak manipulovat. Dále editor Unity nabízí širokou škálu dalších komponent, které ale nejsou použity v implementovaných pracovních listech.

2.6.4 Prefaby

Editor Unity obsahuje systém prefabů, který umožňuje ukládat objekty scény se všemi jejich komponentami a nastavením, a vkládat je pak do dalších scén. Jedná se o takové šablony, od kterých lze vytvářet neomezené množství instancí. Pokud jsou poté provedeny úpravy na šabloně, změny se projeví i ve všech jejích instancích. Pokud je upravena instance šablony, nijak se to na šabloně a jejích dalších instancích neprojeví.

V souboru Unity se to projeví tak, že místo obvyklého výpisu všech komponent a jejich atributů, je u instance prefabu uveden jen odkaz na její prefab, který se nachází mimo tento soubor. Poté následují jen atributy objektu, které jsou různé od původního prefabu. Ve výpisu kódu 2.7 lze vidět zkrácenou verzi uložení instance prefabu tlačítka nápovědy. Tento `GameObject` obsahuje komponenty `Transform`, `BoxCollider2D`, `SpriteRenderer` a skript pro přehrávání instrukcí po stisknutí jeho ikonky. Místo toho je zde uveden jen odkaz na prefab a modifikace instance. Jako poslední modifikaci lze vidět změnu audio klipu, který obsahuje instrukce specifické pro daný pracovní list. Také lze vidět, že třídu `Prefab` indikuje číslo 1001.

```
--- !u!1001 &1728184092
Prefab:
  m_ObjectHideFlags: 0
  serializedVersion: 2
  m_Modification:
    ...
    - target: {fileID: 114789009658023456, guid: 714a74faf135
        c9349b3dc25fbc2ab3ec, type: 2}
      propertyPath: instructions
      value:
      objectReference: {fileID: 8300000, guid: 4907de1cf8dad2
        44cada022189b6cab1, type: 3}
    m_RemovedComponents: []
  m_ParentPrefab: {fileID: 100100000, guid: 714a74faf135c9349
    b3dc25fbc2ab3ec, type: 2}
  m_IsPrefabParent: 0
```

Výpis kódu 2.7: Ukázka zápisu prefabu v souboru scény [3]

Prefaby bude vhodné použít při generování nových pracovních listů. Jelikož jednotlivé pracovní listy obsahují několik společným objektů, jako jsou například instrukce nebo kamera. V případě, že tyto objekty budou změněny, změny se projeví i v již vytvořených pracovních listech. [20]

Kapitola 3

Návrh

3.1 Srovnání metod generování pracovních listů

Zamýšlený editor bude generovat nové pracovní listy pro aplikaci, tudíž bude muset nějakým způsobem vytvářet Unity scény. V této sekci budou porovnány dva způsoby jak nové pracovní listy generovat, které vyšly z detailnější studia formátu scén. Prvním způsobem je generování celého souboru scény pro každý nový pracovní list a druhým způsobem je vytvoření po jedné scéně ve výukových aplikacích pro každý typ úlohy, který bude konfigurovatelný pomocí konfiguračního souboru vygenerovaným editorem.

3.1.1 Srovnávací metodika

Jelikož je Myšák rozsáhlý projekt zahrnující dvě android aplikace, server a současně vznikající frontend k serveru a je možné, že se tento projekt bude dále rozvíjet, bude zváženo nejen to, jak bude náročná implementace samotného editoru, ale také to, jak to případě ovlivní celý projekt. Proto budou hodnoceny oba přístupy podle následujících kritérií:

Pracnost editoru určuje, jak náročný bude návrh a implementace samotného editoru, který je předmětem této práce.

Pracnost změn ve výukových aplikacích určuje, jak daná metoda ovlivní již implementované výukové aplikace. Jelikož obě aplikace používají stejný formát pracovních listů, obě aplikace budou vybranou metodou ovlivněny stejně. Do tohoto kritéria není zahrnuta pracnost změn, které jsou společné pro obě metody (tvorba nových Unity skriptů pro jednotlivé typy úloh).

Pracnost změn na serveru zahrnuje to, jak vybraná metoda ovlivní již implementovaný server a jeho v současnosti vznikající frontend.

Rozšiřitelnost určuje, jak pracné bude v budoucnu projekt Myšák dále rozšiřovat. Týká se to především přidávání nových typů úloh, dále i změn v již existujících typech.

Jednotlivá kritéria se budou hodnotit na stupnici od 0 do 5 bodů, kde vyšší ohodnocení znamená větší náročnost. Všechna kritéria mají stejnou váhu, tudíž lepší ohodnocení bude mít metoda s menším součtem počtu bodů z jednotlivých kritérií.

3.1.2 Generování celých scén

Tento způsob využívá toho, že jeden pracovní list odpovídá jedné Unity scéně, která odpovídá právě jednomu souboru. Tudíž editor by měl za úkol vytvořit kompletní soubor scény, který by se pak mohl přidat k ostatním scénám pracovních listů.

Pracnost editoru – v tomto případě bude editor muset generovat celou scénu, tudíž bude mít na starost i nastavení celé scény, což zahrnuje například nastavení kamery (zachycuje to, co vidí uživatel), zdroje zvuku, nastavení renderování atd. Navíc bude muset nastavovat i detailnější atributy herních objektů, což za nás doteď dělalo Unity. Celkově bude tudíž zapotřebí hlubší porozumění formátu souborů scén Unity, aby editor mohl namapovat jednotlivé objekty scén. (5b)

Pracnost změn ve výukových aplikacích – v současnosti jsou ve výukových aplikacích pracovní listy ve stejném formátu, který by editor generoval. Takže u výukových aplikacích by nebyly zapotřebí žádné změny. (0b)

Pracnost změn na serveru – analogicky jako u výukových aplikací, na serveru by nebylo potřeba dělat úpravy. (0b)

Rozšiřitelnost – při vytváření nových typů úloh bude zapotřebí přidat tyto nové typy do editoru, ale dále to nijak neovlivní další části projektu. Přidání nového typu do editoru ovšem nebude triviální, protože každý typ úlohy nejspíše bude vyžadovat lehce jiné funkce editoru.

Co se týče změn v již existujících typech úloh, obtížnost úprav v editoru se bude odvíjet od toho, o jak rozsáhlé změny se bude jednat. Ale dá se očekávat, že pracnost těchto úprav nebude příliš vysoká. Do již vytvořených pracovních listů se tyto změny propagovat nebudou, jelikož by to mohlo narušit jejich funkčnost. (3b)

3.1.3 Generování konfiguračních souborů

Princip tohoto způsobu spočívá v tom, že by se pro každý typ úlohy vytvořila ve výukové aplikaci scéna, která by byla konfigurovatelná pomocí konfigurač-

ního souboru, který by generoval zamýšlený editor. Tato scéna by již byla nastavená podle daného typu úlohy a při spuštění by jenom na základě konfiguračního souboru vygenerovala herní objekty, které uživatel přidal v editoru.

Pracnost editoru – v tomto případě editor nebude muset řešit nastavení scény, ani detailnější nastavení jednotlivých objektů. Jediné, co editor bude vlastně řešit, je typ úlohy, vzhled přidaných herních objektů, jejich poloha a vztah mezi nimi. (3b)

Pracnost změn ve výukových aplikacích – v současnosti by formát pracovních listů neodpovídal formátu, který by editor generoval. Výukové aplikace by bylo zapotřebí přizpůsobit tomu, že pracovní listy nebudou jen ve formátu samotných scén Unity, ale i ve formátu konfiguračních souborů, které by bylo potřeba předat Unity scéně daného typu úlohy. Proto by musela být do aplikací přidána nová logiku, která by rozlišovala tyto dva typy pracovních listů a podle nich spustila buď přímo scénu pracovního listu, nebo scénu typu úlohy pracovního listu, které by předala konfigurační soubor daného pracovního listu. (3b)

Pracnost změn na serveru – obdobně jako výukové aplikace, server podporuje pouze formát, kde jeden pracovní list odpovídá jedné Unity scéně. Nicméně stačilo by jen pracovní list rozdělit na dva typy a to na pracovní listy reprezentované pomocí Unity scén a pracovní listy reprezentované pomocí konfiguračních souborů. Identifikace pracovních listů by ale zůstala stejná a toto rozdělení by nijak neovlivnilo dosavadní logiku serveru. (1b)

Rozšiřitelnost – ekvivalentně jako u předchozí metody, při vytváření nových typů úloh bude zapotřebí přidat tyto nové typy do editoru. Navíc ale bude potřeba vytvořit pro každý nový typ úlohy novou konfigurovatelnou scénu ve výukových aplikacích.

Co se týče změn v již existujících typech úloh, úpravy editoru budou zhruba stejně složité jako v předchozí metodě. Ovšem situace s konfigurovatelnými scénami ve výukových aplikacích se značně zkomplikuje, protože by v projektu muselo být zavedeno verzování typů úloh tak, aby výukové aplikace rozpoznávaly, které scény jsou kompatibilní s kterými konfiguračními soubory. (5b)

3.1.4 Vyhodnocení

Jak lze vidět v tabulce 3.1, pro projekt Myšák se více hodí metoda tvorby pracovních listů pomocí generování celých scén. Jde vidět, že tato metoda má vyšší pracnost editoru, ale nijak neovlivní již existující nebo vznikající moduly projektu a navíc bude jednodušší projekt v budoucnu rozšiřovat o nové typy úloh.

Kritérium	Generování celých scén	Generování konfiguračních souborů
Pracnost editoru	5	3
Pracnost změn ve výukových aplikacích	0	3
Pracnost změn na serveru	0	1
Rozšiřitelnost	3	5
Celkem	8	12

Tabulka 3.1: Výsledek srovnání metod generování pracovních listů

3.2 Zvolené technologie

Editor bude webová aplikace, která se zpravidla skládá z backendové a frontendové části. V projektu Myšák už existuje modul webové aplikace sloužící k managementu uživatelů a scénářů. Její backend je vyvinut v jazyce Python ve frameworku Django [21] a komunikuje přes REST API s frontendem, který je vyvíjen v jazyce TypeScript s použitím frameworku Angular [22]. Abych co nejvíce zachovala homogennost projektu Myšák, rozhodla jsem se zvolit stejné technologie.

3.3 Pracovní list

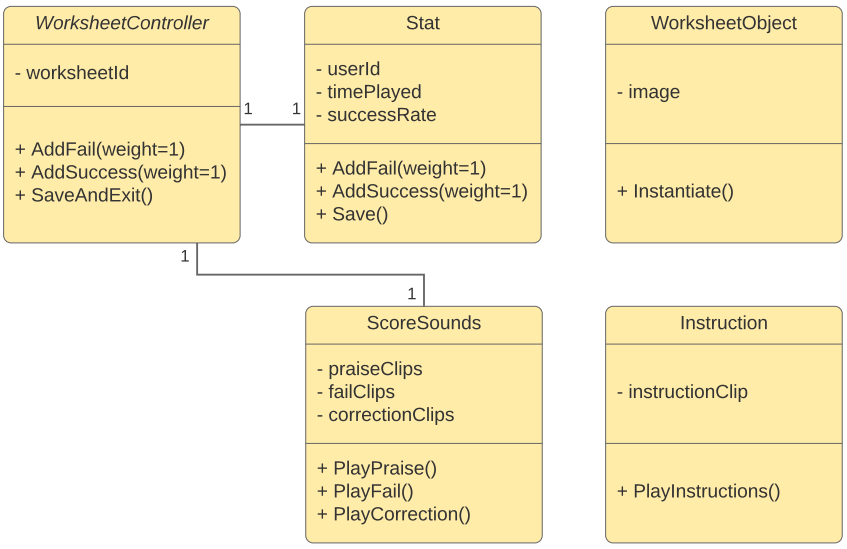
Částečně na základě již implementovaných pracovních listů je navržen základ pracovního listu, který bude společný pro pracovní listy všech typů úloh. Na obrázku 3.1 lze vidět stěžejní třídy pracovního listu, které budou společné pro všechny typy úloh. Nejsou zde uvedeny implementační detaily specifické pro Unity. Následuje stručný popis jednotlivých tříd:

Stat obsahuje statistiku hraného pracovního listu pro aktuálně přihlášené dítě.

ScoreSounds se stará o přehrávání zpětné vazby uživateli. Vždy náhodně vybírá z několika možných zvukových stop, aby nedocházelo k přílišnému opakování hlášek.

WorksheetController zodpovídá za management celého pracovního listu. Pokud dítě provede správný tah, zaznamená ho do statistiky a podá dítěti zpětnou vazbu, analogicky tomu je u nesprávných tahů. Po ukončení pracovního listu uloží statistiku a načte následující scénu.

WorksheetObject je nějaký objekt, se kterým dítě nějakým způsobem interaguje (například tříděné objekty v typu úlohy *Třídění*). Při spuštění



Obrázek 3.1: Doménový model základu pracovního listu

pracovního listu musí nejdříve každý objekt nastavit svůj obrázek dynamicky pomocí zadané cesty k souboru obrázku. To z toho důvodu, že v době tvorby pracovního listu není známo `guid` zvoleného obrázku, které je vyžadováno v souboru scény.

Instruction automaticky přehraje instrukce při spuštění scény. Dále skript této třídy bude přidán k ikonce s nápovědou. Takže po kliknutí na ikonku se přehraje zadání znovu.

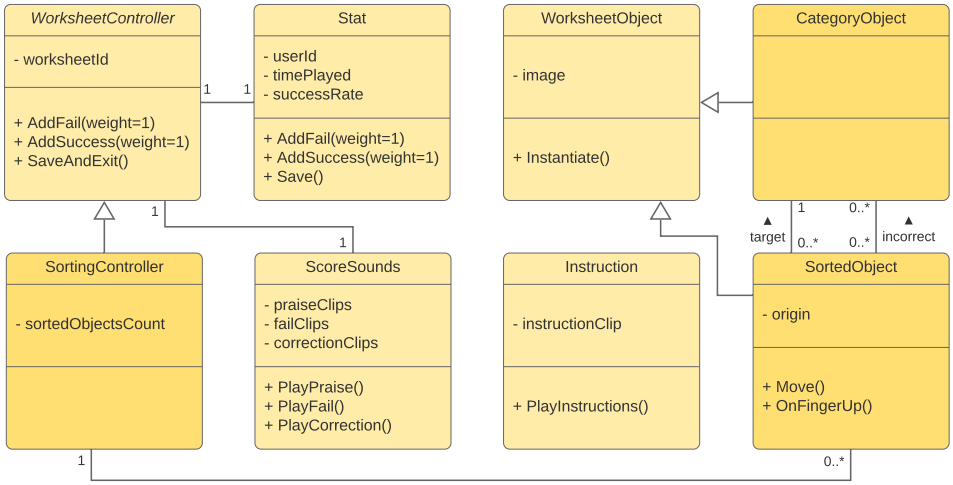
U jednotlivých typů úloh bude zapotřebí vytvořit třídu dědící od abstraktní třídy `WorksheetController` a jednu nebo více tříd dědící od třídy reprezentující objekt `WorksheetObject`.

3.4 Typ úlohy třídění

Tato sekce se bude věnovat návrhu obecného pracovního listu pro typ úlohy *Třídění*, jejíž princip spočívá v třídění libovolného počtu objektů do libovolného počtu kategorií. Kromě zobrazení návrhu tohoto konkrétního typu úlohy je účelem i ukázat, jak mohou být nadále vytvářeny další typy úloh na základě návrhu obecného pracovního listu.

V tomto typu úlohy jsou identifikovány dva typy herních objektů, a to tříděné objekty a kategorie, do kterých jsou objekty tříděny. S objekty reprezentující kategorii dítě nemůže pohybovat a jedná se spíše o takové pasivní objekty. Tříděné objekty má dítě za úkol přetahovat na jejich kategorii. Pracovní list musí zaznamenávat nejen případ, kdy dítě objekt správně přetáhne

na jeho kategorii, ale i případy, kdy dítě objekt přetáhne na jinou kategorii. V obou případech musí aplikace podat dítěti odpovídající zpětnou vazbu a zanést tah do statistiky.



Obrázek 3.2: Doménový model typu úlohy třídění

Na obrázku 3.2 jsou zvýrazněny třídy, které budou přidány k základu obecného pracovního listu. Třída `SortedObject`, která reprezentuje tříděné objekty, má dvě vazby na třídu `CategoryObject`, která reprezentuje kategorie, do kterých jsou objekty tříděny. První vazba je s kategorií, do které má být tříděný objekt přiřazen a druhá vazba je se všemi ostatními kategoriemi. Tyto vazby jsou zde z toho důvodu, že potom, co dítě objekt pustí, objekt musí zjistit, na jakou kategorii byl umístěn a podat dítěti odpovídající zpětnou vazbu. Pokud objekt nebyl umístěn na žádnou kategorii, vrátí se na svou původní pozici.

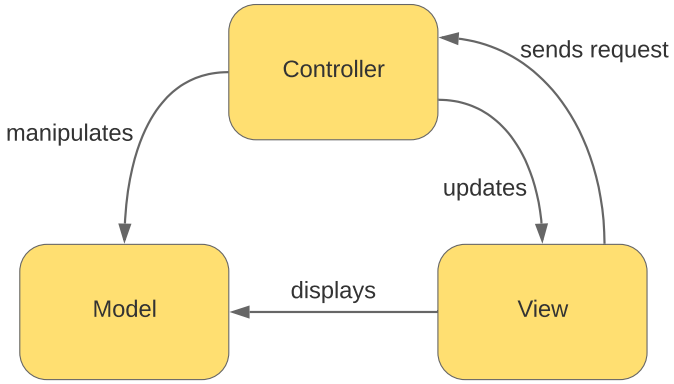
3.5 Backend

Backendová část bude implementována ve webovém frameworku Django, který implementuje architekturu MVC. Diagram MVC architektury lze vidět na obrázku 3.3. V MVC indetifikujeme následující tři části:

Model – slouží k ukládání dat.

View – zobrazuje některá data z modelu a interaguje s uživatelem. Určuje to, co uživatel vidí.

Controller – obsahuje kontrolující logiku a zprostředkovává některá data mezi modelem a view. Neboli na základě dat od uživatele, které dostává od view, manipuluje s daty v modelu. [23, 24]



Obrázek 3.3: MVC architektura

3.5.1 Model

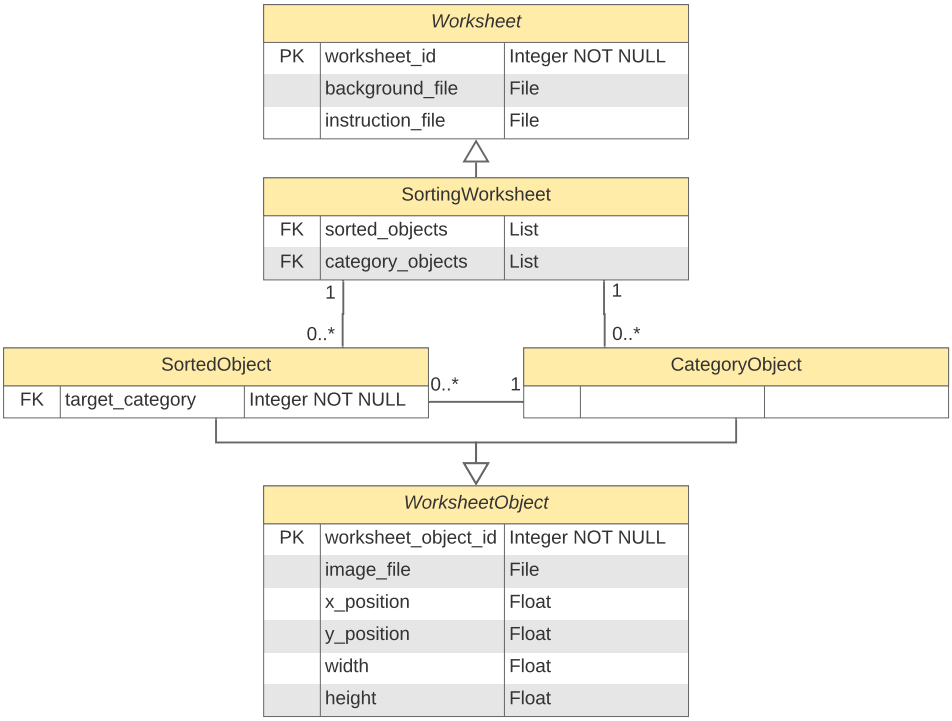
Jak již bylo řečeno, model slouží k ukládání dat. Na rozdíl od existujícího serveru projektu Myšák budou data ukládány v databázi SQLite. Z toho důvodu, že jeho konfigurace je jednoduchá, rychlá, a poskytuje vše potřebné pro prototyp editoru. Základní strukturu databáze lze vidět na obrázku 3.4. Tento návrh obsahuje pouze data potřebná pro pracovní list typu *Třídění*, protože charakter dat ostatních typů úloh bude tomuto velmi podobný a bylo by příliš vyčerpávající tu popisovat i uložení ostatních typů úloh, které bude téměř identické. Tento návrh vychází z požadavků a návrhu typu úlohy třídění. Následuje popis jednotlivých tabulek databáze:

Worksheet – odpovídá jednomu obecnému pracovnímu listu. Jak je zmíněno v požadavcích (sekce 2.4), uživatel bude mít možnost nahrát vlastní obrázek a nastavit ho jako pozadí pracovního listu. Obdobně bude mít možnost nastavit pracovnímu listu audio instrukce, které se spustí na začátku pracovního listu. Od této tabulky budou dědit tabulky všech typů úloh.

WorksheetObject – jedná se o obecný objekt obecného pracovního listu, od kterého budou dědit tabulky všech herních objektů v pracovních listech. U každého objektu musíme zaznamenávat jeho obrázek a pozici.

SortingWorksheet – odpovídá pracovnímu listu typu *Třídění*. V tomto případě se liší od obecného pracovního listu vazbami na objekty, které jsou specifické pro tento typ úlohy.

CategoryObject – reprezentuje kategorii, do které jsou následně tříděny objekty.



Obrázek 3.4: Databázový model

SortedObject – odpovídá tříděnému objektu. U tříděných objektů je potřeba navíc zaznamenávat, do jaké kategorie mají být přiřazeny, což je uloženo pomocí vazby na daný *CategoryObject*.

3.6 REST API

Webové aplikace je zvykem dělit na frontend a backend. Backend, neboli server, se stará o ukládání dat a hlavní logiku aplikace. Frontend je klientská část, která zobrazuje data cílovému uživateli a zprostředkovává jeho komunikaci se serverem. Frontend a backend spolu komunikují přes API. V tomto případě bude využito REST API, které je již v projektu použito.

Opět bude uveden jen stručný návrh API pro tvorbu pracovního listu *Třídění*, který může sloužit ilustračně pro ostatní typy úloh.

POST /sorting – vytvoření nového pracovního listu typu *Třídění* a odeslání jeho id

GET /sorting/{wsid} – detail pracovního listu *Třídění* s identifikátorem {wsid}

POST /sorting/{wsid}/sortedobject – vytvoření nového tříděného objektu v pracovním listu s identifikátorem {wsid}

POST /sorting/{wsid}/categoryobject – vytvoření nového objektu kategorie v pracovním listu s identifikátorem {wsid}

PUT /sorting/{wsid}/sortedobject/{id} – úprava již existujícího tříděného objektu s identifikátorem {id} v pracovním listu s identifikátorem {wsid}

PUT /sorting/{wsid}/categoryobject/{id} – úprava objektu kategorie s indetifikátorem {id} v pracovním listu s identifikátorem {wsid}

PUT /sorting/{wsid}/background – úprava obrázku pozadí v pracovním listu s identifikátorem {wsid}

PUT /sorting/{wsid}/instruction – úprava souboru audio instrukcí v pracovním listu s identifikátorem {wsid}

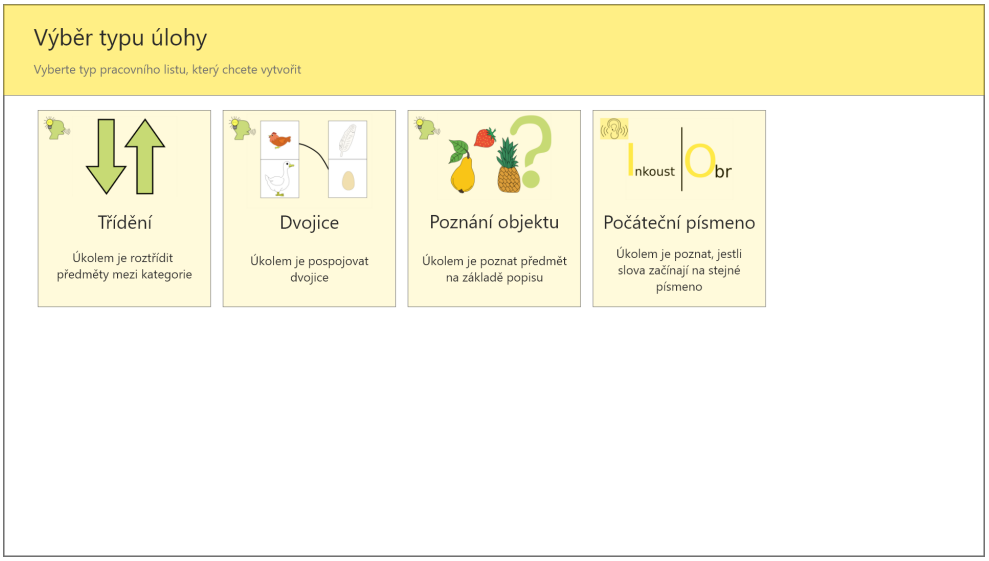
GET /sorting/{wsid}/scene – vygenerovaný textový soubor Unity scény vytvořeného pracovního listu s identifikátorem {wsid}

3.7 Návrh obrazovek

Frontend webové aplikace bude obsahovat několik obrazovek, a to úvodní obrazovku pro výběr typu úlohy, specifické obrazovky editoru pro každý typ úlohy a obrazovku s emulátorem pro spuštění vytvořeného pracovního listu. Obrazovka s emulátorem bude obsahovat pouze vytvořený pracovní list a tlačítko pro návrat, jedná se tedy o triviální obrazovku z pohledu návrhu vzhledu a tato sekce se jí nebude nadále věnovat.

3.7.1 Obrazovka výběru typu úlohy

Na první obrazovce si nejdříve uživatel musí vybrat z nabídky typů úloh, podle jeho volby se následně načte odpovídající editor. Při prvním setkání s editorem uživatel nejspíše nebude vědět, co je náplní jednotlivých typů úloh, proto by u každého typu mělo být k dispozici jeho vysvětlení. Na obrázku 3.5 je návrh zmíněné obrazovky. V horní části jsou instrukce pro uživatele. Potom lze vidět dlaždice reprezentující jednotlivé typy úloh, ze kterých může uživatel vybírat. Na jednotlivých dlaždicích je vždy název daného typu, jeho popis a ikonka, která je pro daný typ použita ve výukové aplikaci Myšák.



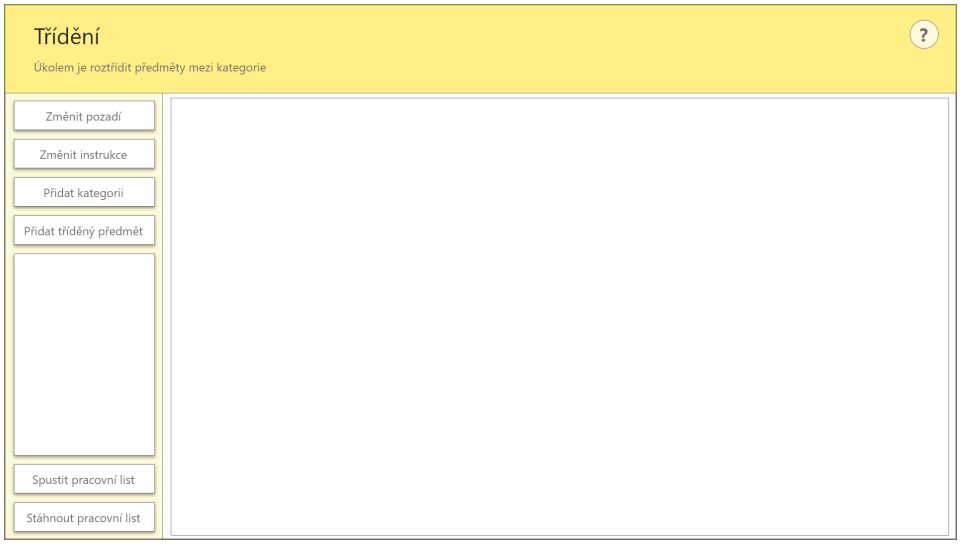
Obrázek 3.5: Návrh obrazovky výběru typu úlohy (ikonky typů úloh z [3])

3.7.2 Obrazovka editoru Třídění

Vzhled editoru bude jen lehce odlišný pro jednotlivé typy úloh. Na obrázku 3.6 je návrh obrazovky editoru pro typ úlohy *Třídění*, kde lze vidět i rysy, které budou společné pro všechny typy úloh. Všechny typy mají společné následující komponenty a jejich rozložení:

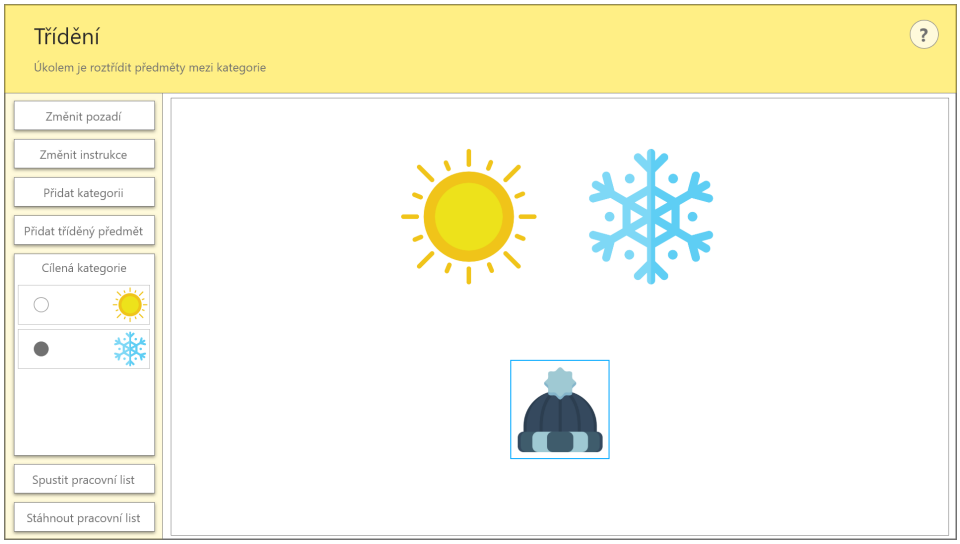
- Horní lišta, která obsahuje název a popis aktuálně tvořeného typu úlohy, a nápovědu s přesným návodem, jak vytvořit pracovní list daného typu.
- Tlačítko pro změnu pozadí, po jehož stisknutí se uživateli otevře dialog pro nahrání obrázku pozadí.
- Tlačítko pro změnu instrukcí, po jehož stisknutí se uživateli otevře dialog pro nahrání audio souboru s instrukcemi.
- Inspektor objektů, který slouží k úpravě atributů jednotlivých objektů pracovního listu. Na obrázku 3.6 je to prázdný obdélník v levé části obrazovky.
- Tlačítko pro spuštění pracovního listu, které vytvořený pracovní list spustí v emulátoru.
- Tlačítko pro stažení vytvořeného pracovního listu, po jehož stisknutí se vygeneruje Unity scéna pracovního listu, která se následně stáhne.

Dále jsou na obrázku 3.6 komponenty typické pro typ úlohy *Třídění*. Jejich počet, funkce a vzhled se bude měnit v závislosti na typu úlohy. Nicméně pro typ *Třídění* jsou to tyto komponenty:



Obrázek 3.6: Návrh obrazovky editoru typu úlohy *Třídění*

- Tlačítko pro přidání kategorie, po jehož stisknutí se uživateli otevře dialog pro nahrání obrázku dané kategorie. Jelikož uživatel bude vnímat jednotlivé objekty hlavně jako obrázky, které jim přiřadí, je tomu přizpůsoben i návrh editoru.
- Tlačítko pro přidání tříděného předmětu, po jehož stisknutí se uživateli otevře dialog pro nahrání obrázku daného předmětu.

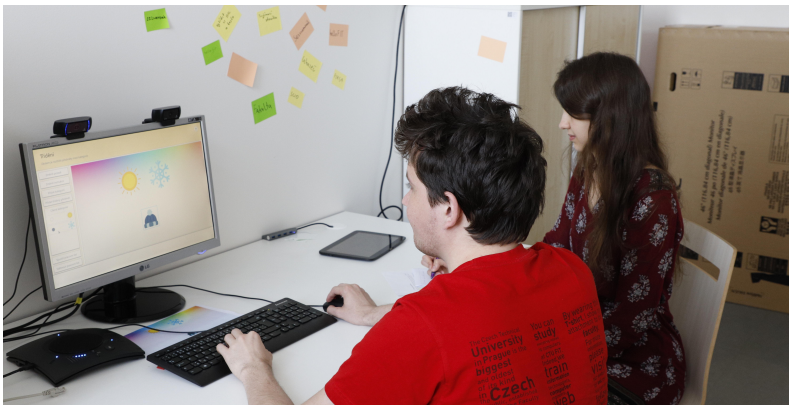


Obrázek 3.7: Návrh obrazovky editoru typu úlohy třídění s aktivovaným inspektorem (ikonky objektů z [25, 26, 27])

Pro lepší představu je na obrázku 3.7 zobrazena práce s inspektorem objektů pro tříděný předmět v typu úlohy *Třídění*. V tvořeném pracovním listu jsou dvě kategorie, a to slunce/léto a zima, a zimní čepice, která představuje tříděný objekt. Jelikož je čepice označená, v inspektoru objektů se zobrazuje výběr pro její cílenou kategorii, kde je momentálně vybraná kategorie reprezentující zimu.

3.8 Uživatelské testování návrhů obrazovek

Dne 9. 4. 2019 jsem moderovala uživatelské testování wireframů v usability labu Fakulty informačních technologií ČVUT. Testování proběhlo pouze s wireframy, které jsou přiloženy v příloze. Testování s wireframy bylo vybráno z toho důvodu, že umožňuje zjistit hlavní chyby návrhu ještě před jeho implementací. Potom je možné návrh upravit ještě před implementací, což bývá méně časově náročné než úprava již implementovaných obrazovek po jejich testování.



Obrázek 3.8: Uživatelské testování wireframů (autor nímku Jiří Melnikov)

Zvolila jsem způsob testování, který má blíže k takzvanému hallway (v překladu chodbovému) testování. Princip tohoto testování spočívá v tom, že místo testování přímo s potenciálními uživateli, se testování provádí s účastníky, které moderátor v podstatě odchytí na chodbě [11]. To ovšem neznamená, že by se měly ostatní části testování zanedbat, stále je zapotřebí mít připravený scénář moderátora a další záležitosti. Tento způsob testování byl vybrán z toho důvodu, že umožňoval testování uspořádat poměrně rychle a testování s wireframy stejně mělo jistá omezení, která by bránila využití celého potenciálu testování s cílovými uživateli.

Testování ze zúčastnilo celkem 5 účastníků. Mezi nimi byly 3 sekretářky, 1 učitelka, jejíž zaměření neobsahuje programování, a 1 student. Tudíž i přesto, že se jednalo o účastníky získané na fakultě, většina z nich měla velmi blízko

k cílové skupině editoru. Při testování měli za úkol vytvořit pracovní list na základě předloženého obrázku a slovního popisu.

3.8.1 Závěry

Všichni účastníci splnili zadaný úkol, vyskytlo se ale několik problémů. Po analýze poznámek z testování a následných rozhovorů s účastníky jsem byly nalezeny tyto problémy:

1. Pojem kategorie není příliš jasný. Na první pohled účastníci většinou nevěděli, co označuje.

2. Pokud nepřidali tříděný předmět jako první, bylo později zapotřebí ho označit, aby se v inspektoru zobrazila možnost pro určení cílové kategorie. V takovém případě účastníkům chvíli trvalo si uvědomit, že musí ještě nastavit cílovou kategorii předmětu.

3. Jedna účastnice očekávala tlačítko, které bude umožňovat vytvořit pozadí, místo jeho změny.

4. Účastníkům nebylo jasné, co si mají představit pod slovem instrukce. Někteří očekávali, že tím nastaví spíše chování pracovního listu, neboli že tím určí, jaká je cílová kategorie tříděného předmětu.

5. Potom, co přidali obrázek objektu, nebylo účastníkům jasné, které obrázky reprezentují kategorie, a které reprezentují tříděné předměty.

6. Účastníci většinou předpokládali, že v horní části obrazovky budou umístěny pouze kategorie a ve spodní části pouze tříděné předměty.

7. Někteří účastníci očekávali, že se obrázky budou automaticky umisťovat na správné místo do pracovního listu a nebudou s nimi nijak manipulovat. V jednom případě účastnice odpověděla, že místo přetahování objektu myší by se pokusila obrázek nahrát znovu, aby se nahrál na správné místo. Ovšem ostatní účastníci by se obrázek pokusili přetáhnout.

V mnoha bodech lze vidět, že problémem je vybraný slovník. V návrzích jsou použity pojmy, které by účastníci nepoužili. Zajímavým poznatkem je také to, že účastníci metodicky procházely tlačítka shora dolů, a místo použití nápovědy raději zkoušeli na postup přijít samostatně.

3.8.2 Navrhované změny a doporučení pro implementaci

Na základě předchozí kapitoly následuje seznam navrhovaných změn, kde čísla jednotlivých změn korespondují s čísly problémů z předchozí kapitoly. Některé tyto návrhy byly rovnou zkonzultovány s účastníky testování a případně byly na základě jejich vazby upraveny.

1. Změnit pojem kategorie na skupinu.
2. Ve wireframech toto nešlo řádně implementovat, ale v takovém případě by měl editor uživatele minimálně při spuštění pracovního listu upozornit, že daný předmět nemá určenou kategorii.
3. Změnit popisek tlačítka pro změnu pozadí na „Nastavit pozadí“.
4. Změnit popisek tlačítka na „Nastavit audio instrukce“, nebo „Nastavit slovní instrukce“.
5. V inspektoru objektů by se měla zobrazovat informace, jakého typu je právě označený objekt.
6. Toto očekávání bych doporučovala dodržet, to znamená umisťovat kategorie pouze do horní části obrazovky a tříděné předměty pouze do spodní části. Pomůže to zachovat podobný formát u pracovních listů stejného typu a zároveň bude tvorba pracovních listů pro uživatele přehlednější.
7. V tomto případě 3 z 5 účastníků očekávali, že se budou objekty umisťovat automaticky. Ideální by bylo, aby se editor pokoušel objekty umisťovat sám, ale zároveň dával uživatelům možnost objekty přemisťovat.

Kapitola 4

Implementace

Tato kapitola se bude podrobněji zabývat hlavně těmi částmi implementace, které jsou klíčové pro generování nových Unity scén. Nebude se zabývat implementačními detaily, které jsou společné pro obecnou tvorbu serveru a jeho frontendu, jelikož ty již byly podrobně popsány v mnoha pracích. Bude se věnovat především těm částem, které se zaměřují na generování nových Unity scén, protože věřím, že ty jsou hlavním přínosem této práce.

4.1 Unity skripty

Před implementací samotného editoru bylo zapotřebí vytvořit nové skripty, které budou kompatibilní s editorem. Tvorba nových skriptů vycházela z již hotových skriptů z [3] tak, aby byla zachována zpětná kompatibilitu.

V aplikaci Myšák již bylo úspěšně implementováno několik typů pracovních listů a byly pro ně vytvořeny potřebné skripty. Existuje ale několik důvodů, proč bylo zapotřebí implementovat nové skripty:

- Bylo zapotřebí zapracovat navržené změny z uživatelského testování. Konkrétně zavést, aby dítě dostávalo zpětnou vazbu po každém tahu.
- Zavést dynamické načítání obrázků. Jak již bylo uvedeno v sekci 2.6.2, Unity v souboru scény používá při odkazování na obrázky `guid`, které obrázku přiřadí samo Unity. Toto `guid` ale v době tvorby pracovního listu v editoru není známo, proto bylo zapotřebí obrázky načítat dynamicky pomocí zadané cesty k obrázku. To se týče pozadí a vzhledu jednotlivých objektů.
- Zavést dynamické načítání audio klipů. Obdobně jako u obrázků, audio instrukce bylo zapotřebí načítat pomocí zadané cesty.
- Zavést automatické škálování obrázků. Místo toho, aby Unity v scéně ukládalo cílovou velikost obrázku, ukládá pouze data pro škálování cílo-

vého obrázku. Respektive si ukládá násobitele jednotlivých rozměrů obrázku. Po vynásobení těchto násobitelů a rozměrů získá cílenou velikost obrázku přímo za běhu. Takže bylo zapotřebí po dynamickém načtení obrázku vypočítat jeho škálování také dynamicky.

Ve výpisu kódu 4.1 je ukázka ze skriptu `WorksheetObject` (potomek třídy `MonoBehaviour`), od kterého dědí všechny skripty herních objektů scény. Tato metoda na základě cesty obrázku a cílových rozměrů nastaví škálování daného obrázku. Potom tento obrázek nastaví jako vzhled daného objektu.

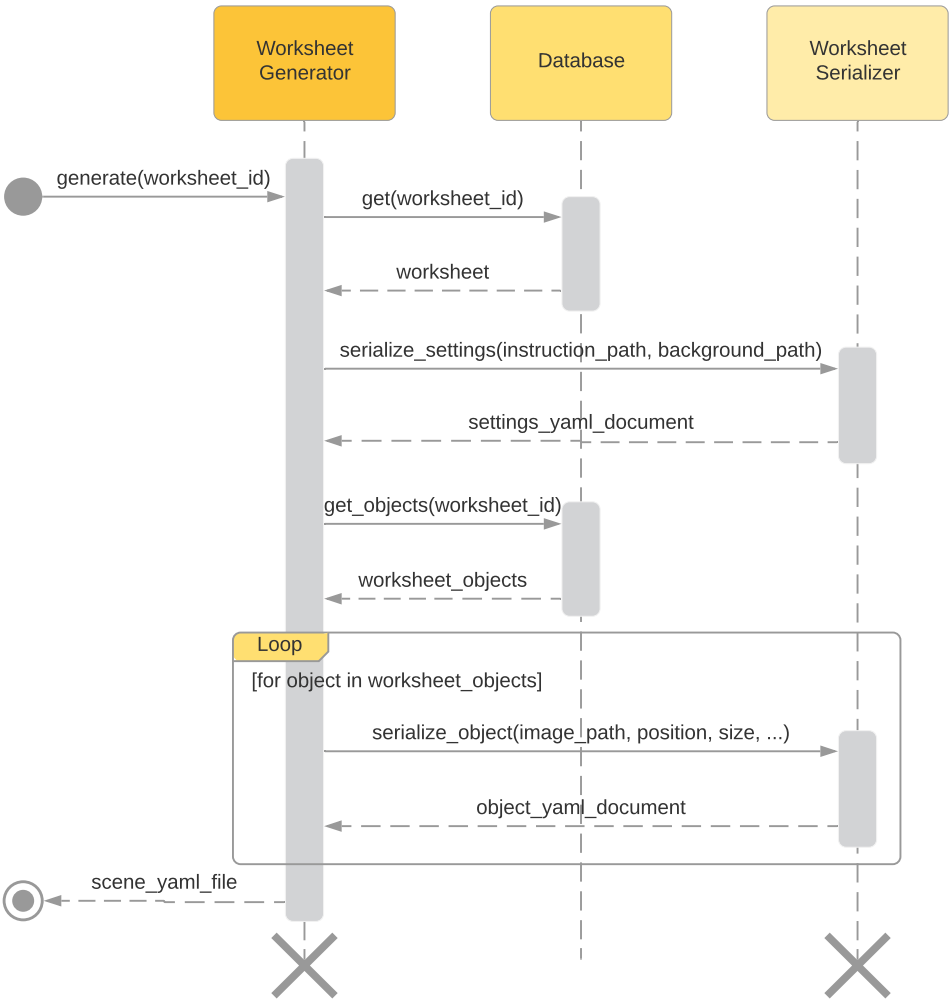
```
protected virtual void Init()
{
    var sprite = Resources.Load<Sprite>(imagePath);
    if (sprite == null) {
        throw new System.ArgumentNullException(
                        "Invalid path for image file.");
    }
    Vector3 localScale = new Vector3 {
        x = width / (sprite.bounds.size.x *
            sprite.pixelsPerUnit),
        y = height / (sprite.bounds.size.y *
            sprite.pixelsPerUnit),
        z = 1
    };
    gameObject.GetComponent<Transform>().localScale
                                        = localScale;
    gameObject.GetComponent<SpriteRenderer>().sprite = sprite;
}
```

Výpis kódu 4.1: Inicializační metoda ze skriptu *WorksheetObject*

4.2 Backend

Prvním úkolem backendu je umožnit uživateli měnit atributy jednotlivých pracovních listů. To zahrnuje ukládání dat v databázi a zprostředkování komunikace mezi databází a klientem. To je implementováno pomocí REST API podle návrhu v sekci 3.6. K implementaci byl použit Django REST framework [28].

Zajímavějším úkolem backendu je generování nových pracovních listů. K tomu byly vytvořeny dva druhy tříd – serializéry a generátory. Serializéry na základě dat jednotlivých objektů vytvoří jejich YAML dokument. Mají v sobě



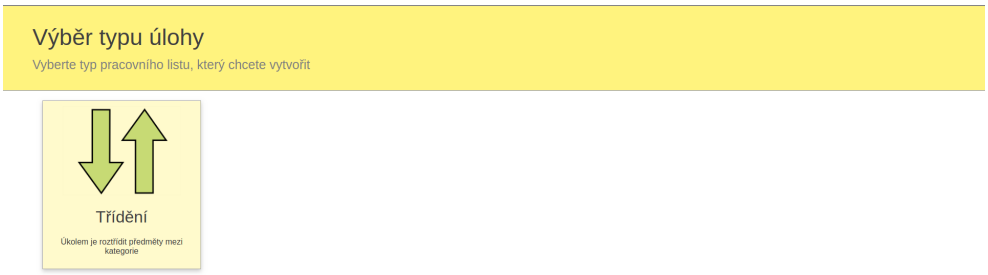
Obrázek 4.1: Zjednodušený sekvenčí diagram generování pracovního listu

uloženou strukturu jejich YAML dokumentu, ve kterém mění jen nastavitelné atributy. Generátory jsou prostředníkem mezi databází a serializéry. Získávají data z databáze a předávají potřebná data ve správném formátu serializérům. Také mají například za úkol generování identifikátorů komponent a objektů výsledného YAML souboru. Generátor celého pracovního listu má za úkol projít všechny jeho objekty a zavolat pro ně vhodné generátory. Zjednodušená spolupráce mezi generátory a serializéry při tvorbě obecného pracovního listu je zachycena na obrázku 4.1. V příloze je přiložen sekvenční diagram, který kompletně zachycuje generování pracovního listu typu *Třídění*.

4.3 Frontend

V této sekci bude zobrazeno výsledné provedení frontendu a stručně popsáno jeho chování. Budou vysvětlena rozhodnutí, která bylo potřeba učinit na základě původního návrhu a návrhů z uživatelského testování wireframů.

4.3.1 Výběr typu úlohy



Obrázek 4.2: Obrazovka výběru typu úlohy

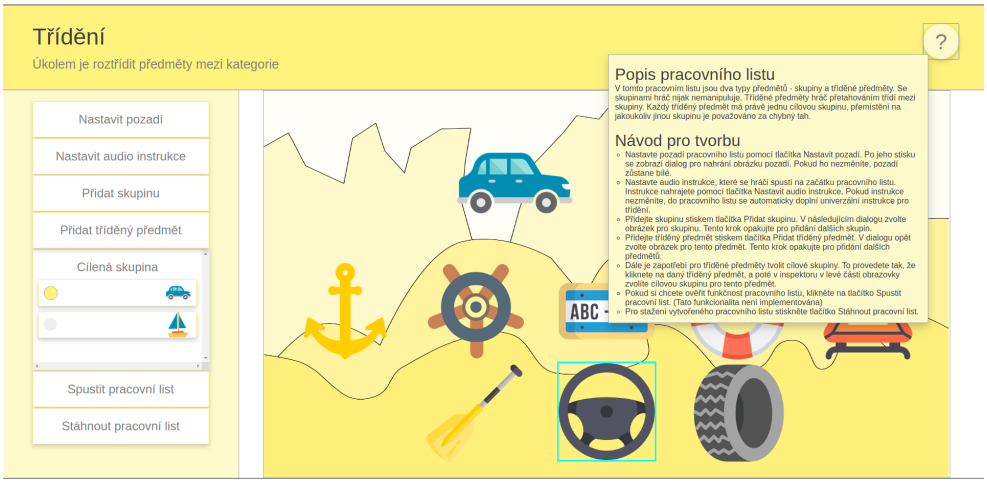
Na obrázku 4.2 je výsledná obrazovka pro výběr typu úlohy. Jelikož při uživatelském testování s touto obrazovkou účastníci neměli problém, implementovaná obrazovka odpovídá návrhu. Jediným rozdílem je absence ikonky kategorie v levém horním rohu kartičky typu úlohy. Ta je vynechána, protože ve výukové aplikaci se téměř nevyskytuje (vyskytuje se pouze v rozhraní pro tvorbu scénářů), a proto pro uživatele není příliš vypovídající.

Jak lze vidět na obrázku 4.2, ve výsledné aplikaci byl implementován editor pro tvorbu pracovních listů typu *Třídění*.

4.3.2 Editor Třídění

Na obrázku 4.3 je implementovaná obrazovka editoru Třídění, ve kterém je i ukázka vytvořeného pracovního listu. Pro ilustraci je zobrazena i nápověda. Na základě uživatelského testování se od návrhu liší popisky některých tlačítek a v závislosti na tom i text nápovědy.

Z uživatelského testování vyplynuly dvě možnosti, jak implementovat umisťování objektů do pracovního listu. Většina účastníků totiž uvedla, že očekávali, že editor bude automaticky určovat pozici vkládaných objektů. Zároveň ale uvedli, že v případě, že by je editor automaticky neumisťoval, objekty by



Obrázek 4.3: Obrazovka editoru *Třídění* s ukázkovým pracovním listem (ikonky objektů z [29, 30, 31, 32, 33, 34, 35, 36, 37], pozadí z [3])

sami umístili přetáhnutím myší. Z toho vychází dvě možnosti, jak umisťování implementovat:

- editor umístí nový objekt na libovolné místo a uživatel ho sám umístí přetáhnutím,
- editor bude automaticky určovat rozmístění objektů.

Rozhodla jsem se implementovat umisťování objektů podle druhé možnosti z následujících důvodů:

- Pokud budou objekty umisťovány automaticky, bude možné u všech vytvořených pracovních listů zajistit, aby rozložení objektů bylo vždy stejné. Například u *Třídění* budou kategorie vždy umístěny v horní části a tříděné předměty v dolní části. To zajistí, že pro děti bude plnění pracovních listů intuitivnější.
- Tato možnost bude pro uživatele editoru méně náročná.
- Implementace automatického umisťování je jednodušší než implementace přetahování objektů.

4.4 Splnění požadavků

V sekci 2.4 jsou vyjmenovány požadavky, které byly na editor kladeny. V tabulce 4.1 lze vidět, jak byly tyto požadavky splněny. Jak vychází z předchozích kapitol, většina požadavků byla splněna. V této sekci bude vysvětleno, proč některé požadavky splněny nebyly.

Požadavek	splněn/nesplněn
F1 Zvolení typu úlohy	splněn
F2 Nastavení pozadí pracovního listu	splněn
F3 Vložení objektu do pracovního listu	splněn
F4 Nastavení atributů objektu	splněn
F5 Nastavení obrázku objektu	splněn
F6 Nastavení pozice objektu	částečně splněn
F7 Vložení audio instrukcí	splněn
F8 Vložení výchozích audio instrukcí	splněn
F9 Spuštění pracovního listu v editoru	nesplněn
N1 Dostupnost přes web	splněn
N2 Kompatibilita s výukovou aplikací Myšák	splněn
N3 Kompatibilita se serverem Myšák	splněn

Tabulka 4.1: Tabulka splnění požadavků

Požadavek *F6 Nastavení pozice objektu* byl splněn pouze částečně. Tento požadavek vyžaduje, aby uživatel mohl přidané objekty přetahovat a tím měnit jejich pozici. Ovšem z uživatelského testování v sekci 3.8 vyplývá, že většina uživatelů očekávala, že se objekty budou umisťovat automaticky. Z tohoto a dalších důvodů uvedených v sekci 4.3.2 jsou objekty umisťovány automaticky. Uživatel ovšem má možnost určit pořadí objektů, které je dané pořadím přidání objektů.

Požadavek *F9 Spuštění pracovního listu v editoru* vyvinutý editor nesplňuje. Byly zvažovány dvě možnosti, jak spouštění pracovních listů v editoru implementovat:

- využít Unity podpory pro web,
- implementovat vlastní emulátor.

Unity dříve ke spouštění aplikací ve webovém prostředí používalo Unity Webplayer, ten už ale v současnosti není podporován [38]. Proto Unity doporučuje v současnosti používat WebGL [39], což je JavaScript API pro zobrazení interaktivní 2D a 3D grafiky ve webových prohlížečích. WebGL ovšem přijímá pouze skripty napsané v JavaScriptu, zatímco celá aplikace Myšák je napsána v jazyce C#. [40] Spuštění vytvořených pracovních listů pomocí WebGL by tedy vyžadovalo přepsání části aplikace Myšák do jazyka JavaScript. Tato možnost byla zamítnuta z důvodu, že tato možnost se příliš neshoduje s cílem této práce a je implementačně neúměrně náročná.

Druhou možností byla implementace vlastního emulátoru, který by na základě dat pracovního listu simuloval fungování aplikace Myšák. To by prakticky znamenalo nejen přepsání Unity skriptů daného typu úlohy do jazyka

TypeScript, ale také implementování funkcionalit, které u aplikace Myšák poskytovalo Unity. Mezi tyto funkcionality patří například posouvání objektu s kurzorem myši, detekce kolizí objektů a další. To by ovšem bylo také příliš implementačně náročné.

4.5 Omezení vyvinutého editoru

Vyvinutý editor je pouze prototyp, z čehož vyplývají některé nedostatky. Prvním nedostatkem je absence kontroly uživatelských vstupů. Frontend, backend ani výuková aplikace nekontrolují vstupy, mezi které patří například obrázky, audio nahrávky instrukcí a nastavené atributy u objektů pracovních listů (například editor u tříděných nekontroluje, jestli má zvolenou cílovou kategorii). Pokud tedy uživatel dodá editoru neplatný vstup, chování není definované.

Druhý nedostatek se týče pouze frontendové části. Frontend není responzivní, v důsledku toho se může na menších obrazovkách zobrazovat nepřehledně.

4.6 Instalační příručka

4.6.1 Backend

Pro spuštění backendu je potřeba mít nainstalovaný python verze 3.5.2 nebo vyšší. Následuje seznam kroků, které jsou nutné ke spuštění backendové části aplikace:

- Otevřete terminál v hlavní složce backendové části (složka se souborem `manage.py`).
- Pokud nemáte nainstalovaný pip, naistalujete ho příkazem

  ```
  $ python3 get-pip.py
  ```

- Pro nainstalování požadavků spusťte příkaz

  ```
  $ pip install -r requirements.txt
  ```

- Pro vytvoření databázových tabulek zadejte příkazy

  ```
  $ python3 manage.py makemigrations
  $ python3 manage.py migrate
  ```

- Pokud chcete využívat superuser účet, vytvoříte ho příkazem

  ```
  $ python3 manage.py createsuperuser
  ```

- Nyní můžete aplikaci spustit příkazem

  ```
  $ python3 manage.py runserver
  ```

Po zadání posledního příkazu se server spustí na adrese
`http://127.0.0.1:8000/`

4.6.2 Frontend

Následuje seznam kroků, které jsou nutné pro spuštění frontendové části aplikace:

- Otevřete terminál v hlavní složce frontendové části.
- Nejdříve nainstaluje Node.js [41] z internetu
- Nainstalujte Angular CLI příkazem

  ```
  $ npm install -g @angular/cli
  ```

- Spusťte aplikaci příkazem

  ```
  $ ng serve
  ```

Po zadání posledního příkazu se klientská aplikace spustí na adrese
`http://localhost:4200/`

4.7 Budoucí rozvoj

Jak již bylo zmíněno v sekcích 2.4 a 2.5, pro plnou integraci je potřeba propojit vytvořený editor s výukovou aplikací a serverem pro management profilů dětí a scénářů. Konkrétně po vytvoření pracovního listu v editoru by uživatel měl mít možnost vytvořený pracovní list přiřadit profilu dítěte. Po přihlášení dítěte do výukové aplikace by pak aplikace přiřazený pracovní list automaticky stáhla a dala dítěti možnost si stažený pracovní list zahrát.

Tohoto lze docílit pomocí Unity AssetBundles. AssetBundle je archiv assetů (obrázky, audio klipy, prefaby, scény atd.), který může být načten za běhu Unity aplikace. Aplikace si takový archiv může stáhnout, a pak přistupovat k jednotlivým položkám archivu. Takto by bylo možné stahovat vytvořené pracovní listy, které se skládají ze souboru Unity scény a použitých obrázků a audio klipů. [42]

Kapitola 5

Testování

Tato kapitola se bude věnovat testování, kterému byl podroben vyvinutý editor. Jednotlivé komponenty editoru byly testovány průběžně, protože průběžné testování umožňuje nalézt chyby dříve, díky čemuž je možné chyby dříve opravit.

Kromě testů zmíněných v této kapitole byla uživatelská rozhraní editoru a výukové aplikace Myšák podrobena uživatelským testování, která jsou uvedena v sekcích 2.3 a 3.8.

5.1 Continous integration

Continous integration (CI), neboli průběžná integrace, je soubor principů a praktik, které umožňují vývojovému týmu rychleji a spolehlivěji vyvíjet software. [43] Princip CI spočívá v integrování kódu ve sdíleném repozitáři a pravidelném testování každé jeho změny. Mezi výhody využívání CI patří:

- rychlá detekce chyb,
- předchází integračním problémům,
- vyhnutí se slučovacím problémům. [44]

Byl vybrán Gitlab CI, protože umožňuje jednoduchou integraci s Gitlab repozitářem, a jeho konfigurace je jednoduchá. Mezi jeho další výhody patří podpora široké škály jazyků, podpora široké škály platforem, stabilita, a další vlastnosti. Pro CI Gitlab poskytuje také sdílené runnery, takže pro jednoduchou konfiguraci stačí jenom do repozitáře nahrát jednoduchý konfigurační soubor. [44]

5.2 Unit testy

Unit testy, neboli testování jednotek, se zaměřuje na testování samostatných jednotek systému. Za jednotku je považována část systému, kterou lze samostatně testovat. Tyto jednotky jsou vždy testovány odděleně tak, aby byly nezávislé na zbytku systému. Za jednotku může být považována například funkce, třída nebo její konkrétní metoda. [45]

Framework Django má zabudouvanou podporu pro unit testy. Django umožňuje jednoduše definovat jednotlivé testovací případy a následně všechny spustit jediným příkazem nebo je spouštět i jednotlivě. Ve výpisu kódu 5.1 se testuje správné fungování metody pro získání id třídy generátoru id, který je určen pro generování nových Unity scén. [46]

```
class IDGeneratorTestCase(TestCase):

    def test_get_next0(self):
        generator = IDGenerator()
        self.assertEqual(generator.get_next(), 1000)
        self.assertEqual(generator.get_next(), 1001)

    def test_get_next1(self):
        generator = IDGenerator(init_id=42)
        self.assertEqual(generator.get_next(), 43)
        self.assertEqual(generator.get_next(), 44)
```

Výpis kódu 5.1: Unit test generátoru id

Angular používá ke spouštění unit testů nástroj Karma [47]. U složitějších komponent může být ale psaní unit testů poměrně obtížné. Jedním z důvodů je nutnost mockovat vnořené komponenty. To znamená, že pokud testovaná komponenta obsahuje vnořené komponenty, tyto komponenty je nutné při testování mockovat. Z tohoto důvodu nejsou především složitější komponenty plně pokryty unit testy. [48]

5.3 End to End

E2E testování se na rozdíl od většiny testování nezaměřuje na přímé testování kódu, ale na testování produktu v produkčním prostředí. E2E testy simulují kroky uživatele ve spuštěném produktu. Tyto testy lze spouštět v různých prostředích, čímž lze zjistit případnou nekompatibilitu. [49]

Tyto testy jsou vhodné pro testování frontendu. Angular k E2E testům defaultně využívá nástroj Protractor [50]. Protractor umožňuje automaticky

spustit zvolený prohlížeč, ve kterém přejde na zadanou adresu. Jednotlivé elementy webové stránky umožňuje vyhledávat například pomocí css tříd nebo identifikátorů. S těmito elementy potom interaguje podobně jako uživatel – kliká na tlačítka, zadává vstup. Chování stránky poté opět kontroluje vyhledáním zadaných elementů a jejich porovnáním se vzorovými daty. [51]

5.4 Integrační testy

Integrační testování je stupněm testování softwaru, při kterém jsou jednotlivé části systému spojeny a testovány jako celek. Účelem tohoto testování je odhalit chyby v komunikaci mezi jednotlivými částmi. [52]

Předmětem tohoto testování byla hlavně komunikace mezi backendem a frontendem. Vzhledem k menší velikosti systému byly testy prováděny manuálně. Testy byly prováděny průběžně s implementací frontendu a nalezené chyby byly ihned opraveny. V současnosti nejsou nalezeny žádné integrační chyby mezi backendem a frontendem.

5.5 Testování výstupů

Již v průběhu vývoje bylo nutné otestovat funkčnost vygenerovaných scén. Z toho důvodu, že jednotlivé vygenerované scény obsahují stovky, až tisíce řádek a ruční tvorba referenčních scén by proto byla velmi časově náročná, vygenerované scény byly testovány manuálně. Princip takového testování spočíval v tom, že vygenerované scény byly v editoru Unity přidány do výukové aplikace Myšák mezi ostatní pracovní listy. Potom stačilo aplikaci spustit a zkontrolovat správné fungování pracovního listu.

Vygenerované scény byly testovány průběžně již od implementace generátoru backendu (v sekci 4.2), aby byla ověřena jeho správná funkčnost. Po implementování frontendu (v sekci 4.3) bylo nutné scény otestovat, aby byl ověřen správný převod dat (například převod pozice objektu, protože její formát se u Angularu a Unity liší).

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a implementovat editor pracovních listů pro aplikaci Myšák vyvíjené v editoru Unity 3D. Před návrhem editoru byly analyzovány již implementované pracovní listy, které také byly podrobeny uživatelskému testování ve školce. Na základě tohoto testování byly navrženy změny testovaných pracovních listů, které byly později zahrnuty do vytvořeného editoru.

Následně byla analyzována struktura Unity scén a způsoby, kterými je možné nové scény generovat. Možnými způsoby bylo generování celých Unity scén a generování konfiguračních souborů pro připravené Unity scény. Jako výsledný způsob bylo vybráno generování celých Unity scén především z toho důvodu, že druhý způsob by příliš zasáhl do již implementovaných modulů projektu Myšák.

Výsledkem práce je editor, který umožňuje tvorbu pracovních listů typu *Třídění*. Tento editor se skládá z backendové části, která ukládá data tvořeného pracovního listu, na základě kterých generuje scénu daného pracovního listu. Druhou část editoru tvoří frontend, který poskytuje přívětivé uživatelské rozhraní. Toto rozhraní bylo podrobeno uživatelskému testování a navržené změny byly zapracovány do implementace. Editor generuje scény, které využívají upravených Unity skriptů. Původní Unity skripty aplikace Myšák byly upraveny tak, aby umožňovaly některé části pracovního listu nastavovat dynamicky (například načítat a nastavovat obrázky objektů).

Výsledný editor umožňuje uživateli vytvořit nový pracovní list, nastavit jeho pozadí a audio instrukce, vkládat do něj nové objekty, a nastavovat vzhled a chování vložených objektů.

Bibliografie

1. *YAML Syntax* [online]. Dostupné také z: `https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html`. [cit. 2019-03-24].
2. *Description of the Format* [online]. 2018. Dostupné také z: `https://docs.unity3d.com/Manual/FormatDescription.html`. [cit. 2018-12-03].
3. KASALICKÁ, Kateřina; JEŽEK, Radek; TOPIČ, Jakub; TISLICKÝ, Jan; VÁCLAVIKOVÁ, Zuzana. *Myšák home*. [cit 2019-01-23].
4. JEŽEK, Radek. *Myšák – Webový klient pro vzdělávací aplikaci*. 2019. České vysoké učení technické v Praze, Fakulta informačních technologií.
5. KASALICKÁ, Kateřina; JEŽEK, Radek; TOPIČ, Jakub; TISLICKÝ, Jan; ŠLEJTR, Ondřej; VÁCLAVIKOVÁ, Zuzana. *Návrh serverového modulu - 2. iterace SP2*. 2018. Technická zpráva. České vysoké učení technické v Praze, Fakulta informačních technologií. [cit 2019-01-23].
6. *Unity 3D*. Dostupné také z: `https://unity.com/`.
7. *Scenes* [online]. 2018. Dostupné také z: `https://docs.unity3d.com/Manual/CreatingScenes.html`. [cit. 2019-03-04].
8. PEKÁRKOVÁ, Simona. *Jdu do školy: Chytrý pomocník pro děti a rodiče*. Englewood Cliffs: Fragment, 2017. Č. 9788025331118.
9. CHI, Clifford. The Beginner's Guide to Usability Testing. *HubSpot* [online]. 2018. Dostupné také z: `https://blog.hubspot.com/marketing/usability-testing`. [cit. 2018-10-20].
10. MURPHY, Christopher. A Comprehensive Guide To User Testing. *Smashing magazine* [online]. 2018. Dostupné také z: `https://www.smashingmagazine.com/2018/03/guide-user-testing/`. [cit. 2018-10-20].

11. BANK, Chris; CAO, Jerry. *The Guide to Usability Testing*. Dostupné také z: https://www.uxpin.com/studio/ebooks/guide-to-usability-testing/. [cit. 2018-10-20].

12. *YAML Ain't Markup Language (YAML™) Version 1.1* [online]. Dostupné také z: https://yaml.org/spec/1.1/. [cit. 2019-03-24].

13. *Behind the scenes* [online]. Dostupné také z: https://docs.unity3d.com/Manual/BehindtheScenes.html. [cit. 2019-03-22].

14. *MonoBehaviour* [online]. Dostupné také z: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html. [cit. 2019-03-22].

15. *Camera* [online]. Dostupné také z: https://docs.unity3d.com/ScriptReference/Camera.html. [cit. 2019-03-22].

16. *GameObject* [online]. Dostupné také z: https://docs.unity3d.com/ScriptReference/GameObject.html. [cit. 2019-03-22].

17. *Transform* [online]. Dostupné také z: https://docs.unity3d.com/ScriptReference/Transform.html. [cit. 2019-03-22].

18. *SpriteRenderer* [online]. Dostupné také z: https://docs.unity3d.com/ScriptReference/SpriteRenderer.html. [cit. 2019-03-22].

19. *BoxCollider2D* [online]. Dostupné také z: https://docs.unity3d.com/ScriptReference/BoxCollider2D.html. [cit. 2019-03-22].

20. *Prefabs* [online]. Dostupné také z: https://docs.unity3d.com/Manual/Prefabs.html. [cit. 2019-03-22].

21. *Django*. Dostupné také z: https://www.djangoproject.com/.

22. *Angular CLI*. Dostupné také z: https://cli.angular.io/.

23. *Basic MVC Architecture* [online]. Dostupné také z: https://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm. [cit. 2019-03-12].

24. *MVC Architecture* [online]. Dostupné také z: https://www.tutorialsteacher.com/mvc/mvc-architecture. [cit. 2019-03-12].

28. *Django REST framework*. Dostupné také z: https://www.django-rest-framework.org/.

38. *Unity Webplayer* [online]. Dostupné také z: https://unity3d.com/webplayer. [cit. 2019-05-03].

39. *WebGL* [online]. Dostupné také z: https://www.khronos.org/webgl/. [cit. 2019-05-03].

40. *Getting started with WebGL development* [online]. Dostupné také z: https://docs.unity3d.com/Manual/webgl-gettingstarted.html. [cit. 2019-05-03].

41. *Node.js* [online]. Dostupné také z: https://nodejs.org/. [cit. 2019-05-01].

42. *AssetBundles* [online]. Dostupné také z: https://docs.unity3d.com/Manual/AssetBundlesIntro.html. [cit. 2019-05-01].

43. SACOLICK, Isaac. What is CI/CD? Continuous integration and continuous delivery explained. *InfoWorld* [online]. 2018. Dostupné také z: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html. [cit. 2019-04-02].

44. *GitLab Continuous Integration & Delivery* [online]. Dostupné také z: https://about.gitlab.com/product/continuous-integration/. []cit. 2019-04-02].

45. *Unit Testing* [online]. Dostupné také z: http://softwaretestingfundamentals.com/unit-testing/. [cit. 2019-03-27].

46. *Writing and running tests* [online]. Dostupné také z: https://docs.djangoproject.com/en/2.2/topics/testing/overview/. [cit. 2019-03-27].

47. *Karma*. Dostupné také z: http://karma-runner.github.io/latest/index.html.

48. *Unit Testing* [online]. Dostupné také z: https://docs.angularjs.org/guide/unit-testing. [cit. 2019-05-01].

49. *End to End Testing Angular 2, A Complete Guide (Part 1)* [online]. 2016. Dostupné také z: https://www.gistia.com/end-to-end-testing-angular-2-p2/. [cit. 2019-04-29].

50. *Protractor*. Dostupné také z: https://www.protractortest.org.

51. DOUBEK, Martin. *HTML 5 a JavaScript pro vývoj a simulaci uživatelského rozhraní automobilu*. 2015. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií.

52. *Integration Testing* [online]. Dostupné také z: http://softwaretestingfundamentals.com/integration-testing/. [cit. 2019-04-30].

Převzaté obrázky

25. SMASHICONS. Sunny free icon. In: dostupné také z: https://www.flaticon.com/free-icon/sunny_136723#term=sun&page=1&position=7.
26. FREEPIK. Snowflake free icon. In: dostupné také z: https://www.flaticon.com/free-icon/snowflake_1200430#term=snowflake&page=1&position=1.
27. SMASHICONS. Beanie free icon. In: dostupné také z: https://www.flaticon.com/free-icon/beanie_1470650#term=winter%5C%20hat&page=1&position=39.
29. FREEPIK. Car free icon. In: dostupné také z: https://www.flaticon.com/free-icon/car_171239#term=car&page=1&position=14.
30. FREEPIK. Big Anchor free icon. In: dostupné také z: https://www.flaticon.com/free-icon/big-anchor_85117#term=anchor&page=1&position=2.
31. SMASHICONS. Helm free icon. In: dostupné také z: https://www.flaticon.com/free-icon/helm_139927#term=helm&page=1&position=8.
32. FREEPIK. License plate free icon. In: dostupné také z: https://www.flaticon.com/free-icon/license-plate_290122#term=car%5C%20plate&page=1&position=8.
33. BUDDHA, Pixel. Help free icon. In: dostupné také z: https://www.flaticon.com/free-icon/help_179540#term=lifesaver&page=1&position=75.
34. FREEPIK. Reflective triangle free icon. In: dostupné také z: https://www.flaticon.com/free-icon/reflective-triangle_290117#term=accident%5C%20triangle&page=1&position=2.
35. FREEPIK. Rowing free icon. In: dostupné také z: https://www.flaticon.com/free-icon/rowing_434031#term=paddle&page=1&position=74.

36. FREEPIK. Steering wheel free icon. In: dostupné také z: https://www.flaticon.com/free-icon/steering-wheel_391050.

37. SMASHICONS. Tire free icon. In: dostupné také z: https://www.flaticon.com/free-icon/tire_272026#term=tire&page=1&position=86.

Příloha A

Seznam použitých zkratek

API Application programming interface

CI Continous integration

E2E End to End

ID Identification data

MVC ModelViewController

OS Operační systém

REST Representational State Transfer

YAML YAML Ain't Markup Language

Příloha B

Uživatelské testování pracovních listů ve školce

B.1 Scénář

Ahoj, já jsem (moderátor:) Kačka a tohle je (asistent:) Radek. Jsme studenti z FIT ČVUT a byli bychom rádi, kdyby jste si zahráli hru, kterou jsme přinesli. Je to taková krátká hra na tabletu a vaším úkolem bude si ji zahrát na stanovišti, které jsme připravili ve vedlejší místnosti.

Bude to probíhat tak, že si vždy jednoho z vás vezmu stranou, kde si nejdřív trošku popovídáme a já vám dám přesnější instrukce. Potom si zahrajete tu hru, která vám sama napoví, co máte dělat. A pak se vás trochu optám, jak se vám ta hra hrála a jak se vám líbila. Po celou dobu tam bude přítomna vaše učitelka a pokud si nebudete vědět s něčím rady, můžete se mě na cokoliv zeptat.

- Zeptej se pedagoga, kde se nachází první dítě z tvého seznamu.
- Přejdi k prvnímu dítěti ze seznamu.

Ahoj, ty jsi .....? Pojď se mnou prosím vedle.

- Odveď dítě na připravené stanoviště.
- Vyptej se dítěte na pár otázek a snaž se vést krátkou nenucenou konverzaci. Možné otázky:
  - Hraješ doma nějaký hry na mobilu nebo tabletu?
  - Ukážeš mi prosím, s čím si tady nejraději hraješ?
  - Chtěl/a by jsi radši umět létat, nebo být neviditelný/a?
  - Jaká je tvoje oblíbená barva?

Tak teď už se můžeme přesunout k té hře. Za chvíli ti předám tablet s tou hrou. Postupně se ti tam spustí čtyři mini hry. Ta hra ti bude slovně říkat instrukce, bude na tebe mluvit, takže ti řekne, co máš v jednotlivých mini hrách dělat. Pokud bys nerozuměl/a zadání, můžeš si zadání spustit znovu pomocí ikonky v pravém horním rohu. Máš nějaké otázky?

- Odpověz na případné dotazy.
- Podej dítěti tablet.

Tak jo, tak můžeme začít. Teď stačí, když ťukneš na bránu a spustí se ti první mini hra.

- Vzdal se na takovou vzdálenost, aby se dítě cítilo pohodlněji, ale aby tě zároveň v případě nouze mohlo požádat o pomoc.
- Pozoruj dítě tak, aby mu to nebylo nepříjemné a zapamatuj si poznatky z jeho chování.
- Počkej, až dítě dohraje všechny čtyři mini hry.

Tak to už si si zahrál/a všechny úkoly, které tu pro tebe máme. Teď ti tu ještě spustím některou z těch úloh, kterou už si hrál/a a budu se tě trošku vyptávat. Pokud v tom úkolu něco nechápeš, nebo se ti nezdá, budu ráda, když mi to řekneš.

- Zvol některou z úloh v menu, nejlépe mini hru, který byla pro dítě neproblematičtější.
- Ptej se na jednotlivé tahy dítěte.
  - Proč si vybral/a právě tohle?
  - Proč myslíš, že není tohle správná odpověď?
  - Rozumíš, proč se to vrátilo na svou původní pozici?
  - Víš, proč se to přesunulo jinam, než si to dal/a?
- Po skončení mini hry dej tablet stranou a dej před dítě výstupní dotazník a pastelky.

Výborně, teď bych tě ještě chtěla požádat, aby si tady na tom papíru ohodnotil/a, jak se ti líbily jednotlivé mini hry. Máš tady vždycky obrázek některé z mini her a pod ním smajlíky. Je tu celá škála smajlíků od smějícího se po nešťastného smajlíka. A ty si máš pod každým obrázkem vybrat jeden smajlík podle toho, jak tě ta mini hra bavila. Pokud se ti hodně líbila a hodně tě bavila, vybarvíš ten nejšťastnější smajlík, a pokud se ti vůbec nelíbila, vybarvíš ten nejsmutnější smajlík. A pokud se ti líbila nějak mezi tím, vybereš odpovídající smajlík mezi. Rozumíš tomu, co se po tobě chce?

- Ujisti se, že dítě správně pochopilo zadání.
- Nech dítěti soukromí, ať může nerušeně vyplnit dotazník.
- Zaznamenej si poznatky z testování do formuláře.
- Počkej, až dítě vyplní dotazník.

Super, tak to je všechno. Teď tu pro tebe mám menší odměnu za to, že jsi tak skvěle spolupracoval/a.

- Předej dítěti odměnu.

A teď už můžeš jít zpátky k ostatním.

- Odveď dítě zpět k ostatním, nebo o to požádej učitelku.
- Seber vyplněný dotazník a připrav stanoviště pro další dítě.
- Testuj s dalšími dětmi podle předchozího postupu, dokud neotestuješ aplikaci se všemi dětmi.
- Po ukončení testování sbalte stanoviště a případně se rozlučte s dětmi.

B.2 Očekávání

Loď a auto

1. Dítě si poslechne zadání.
2. Pokud dítě nerozumí napoprvé zadání, dotkne se podle instrukcí otazníku v pravém horním rohu.
3. Dítě vybere jeden z objektů a umístí ho tam, kam si myslí, že patří.
4. Vyslechne si zhodnocení jeho tahu.
5. Dále přetahuje objekty na jejich cílové destinace.
6. Po roztřídění všech objektů vyčká na zhodnocení jeho posledního tahu a přepnutí do hlavního menu.

Počáteční písmeno

1. Dítě si poslechne zadání.
2. Pokud dítě nerozumí zadání, dotkne se otazníku v pravém horním rohu.
3. Dotkne se kartičky, díky čemuž se přehraje název věcí na ní.

4. Přesune, nebo nepřesune kartičku na myšáka podle toho, jestli ji považuje za jednu ze správných odpovědí.
5. Stejným způsobem projde ostatní kartičky.
6. Zazvoní na zvonek.
7. Poslechne si vyhodnocení a počká na přepnutí do hlavního menu.

Ovoce

1. Dítě si poslechne zadání.
2. Pokud dítě nerozumí napoprvé zadání, dotkne se podle instrukcí otazníku v pravém horním rohu.
3. Dítě vybere objekt, který je podle něho správnou odpovědí a přemístí ho na myšáka.
4. Vyslechne si zhodnocení jeho tahu.
5. Dokud dítě není úspěšné, opakuje body 3 a 4.
6. Po správném přiřazení počká na přepnutí do hlavního menu.

Kdo co dává

1. Dítě si poslechne zadání.
2. Pokud dítě nerozumí napoprvé zadání, dotkne se podle instrukcí otazníku v pravém horním rohu.
3. Dítě vybere jeden z objektů a umístí ho na objekt, o kterém se domnívá, že je jeho dvojice.
4. Vyslechne si zhodnocení jeho tahu.
5. Dále přetahuje objekty na jejich cílové dvojice.
6. Po spárování všech dvojic vyčká na zhodnocení jeho posledního tahu a přepnutí do hlavního menu.

Příloha C

Uživatelské testování wireframů frontendu

C.1 Scénář

Dobrý den, jmenuji se Kateřina Kasalická a provedu vás dnešním testováním. Později vám pustím jednu aplikaci a řeknu vám, co chci, aby jste v něm udělal/a. Účelem tohoto testování je zjistit, jestli se vám s tou aplikací dobře pracuje a případně zjistit, jak by ho šlo vylepšit.

Představte si, že máte dítě ve věku 5 let a občas mu dáváte zahrát vzdělávací aplikaci Myšák, kterou hraje na tabletu. Tu hru tady mám a chtěla bych, aby jste si ji nejdříve zahrál/a, aby jste se s ní seznámil/a. Ta aplikace není předmětem tohoto testování. Jen je potřeba, aby jste se seznámil/a s tím, jak vypadá.

- Předej tablet s aplikací účastníkovi.
- Stručně testerovi vysvětli, jak funguje hlavní menu a spusť mu pracovní list třídění.
- Nech ho/ji si hru v kliku zahrát, nijak ho/ji nesměruj a ani nepozoruj postup. Jen se ujisti, že si zahraje pracovní list zaměřený na třídění.

Výborně, myslíte, že jste se se hrou dostatečně seznámil/a? Tak teď jste se dozvěděl/a, že existuje editor, ve kterém můžete vytvořit nové levely pro tuto aplikaci a rád/a by jste ho vyzkoušel/a. Rozhodla jste se, že pro začátek vytvoříte jen jednoduchý level, aby jste si vyzkoušela, jak editor funguje. Vaše představa vypadá takto.

- Dej před účastníka obrázek s cílovým pracovním listem.

Úkolem tohoto levelu má být roztřídit čepici podle toho, kam patří, kam se víc hodí – k slunci, neboli létu, nebo k vločce, neboli zimě. Takže tam chcete

vložit objekty, které vidíte tady na obrázku a nastavit pozadí tak, jak vidíte. Žádné vlastní audio instrukce jste nenahrál/a.

Editor, který budete testovat, je pouze prototyp. Takže tam nejsou implementovány všechny funkcionality. Bude ovšem fungovat vše, co budete potřebovat. Jen vám to nedovolí dělat něco, co není vaším úkolem. Všechny obrázky, které budete potřebovat, už máte staženy a při nahrávání obrázku se vám vždy automaticky vybere správný obrázek. Máte nějaké dotazy?

Takže jen zopakuji, že vašim úkolem je vytvořit tento level, jehož úkolem je určit, jestli čepice patří pod sluníčko, nebo pod vločku. Level by měl vypadat naprosto stejně jako na obrázku, audio instrukce jste žádné nenahrál/a a ani nemáte v úmyslu je nahrávat. Potom, co level vytvoříte podle předlohy, si chcete nějak ověřit jeho funkčnost. Je vše jasné? Dobře, tak se do toho pustíme. Byla bych ráda, kdyby jste přemýšlel/a nahlas a popisoval/a, co děláte.

- Přepni obrazovku na wireframy a přejdi k testování.

Výborně, tak jste splnil/a celé zadání. Máte nějaké dotazy? Nebylo vám něco jasné? Já bych se ještě chtěla zeptat, co by jste dělal/a, kdyby se vám přidané objekty automaticky neumístily na správné místo? Například kdyby se čepice vložila do levého horního rohu, co by jste dělal/a?

Výborně, tak to je ode mne vše. Děkuji vám za spolupráci.

Příloha D

Uživatelská příručka

Nejdříve spusťte backend a frontend editoru podle instalační příručky v sekci 4.6.1. V editoru pracovních listů postupujte podle jeho instrukcí. Přímo v editoru najdete návod pro tvorbu pracovního listu po rozkliknutí otazníku v pravém horním rohu.

Jelikož editor není propojen s aplikací Myšák, vytvořené pracovní listy je potřeba do projektu aplikace vkládat manuálně podle následujících instrukcí.

- Otevřete složku projektu aplikace Myšák (na přiloženém médiu ve složce `./src/impl/eduapp/Mysak_home-editor_adaptation`).
- Do složky `./Assets/Scenes/Worsheets/CustomEditor` vložte vygenerovaný soubor scény (soubor s koncovkou `unity`).
- Otevřete složku projektu backendu, který jste použili pro vytvoření daného pracovního listu.
- Složku s obrázky a audio klipy `media/CustomEditor` nakopírujte do složky `./Assets/Resources` projektu aplikace.
- Poté musíte přidat pracovní list do seznamu pracovních listů aplikace. To znamená, že do souboru `./Assets/Resources/worksheets.json` přidáte informace o vytvořeném pracovním listu podle následujících instrukcí:
  - `Name` – název pracovního listu, můžete zvolit libovolný string.
  - `Image` – jméno souboru obrázku daného pracovního listu ve složce `./Assets/Resources/Sprites/WorksheetImages`. Tento obrázek se bude zobrazovat pouze v menu pro dozor. Pro jednoduchost lze zvolit obrázek nějakého již existujícího pracovního listu.
  - `Id` – název vygenerovaného souboru Unity scény bez přípony.

  - `Difficulty` – obtížnost vytvořeného pracovního listu od 1 do 5, kde 1 je nejjednodušší.

- Informaci o pracovním listu musíte vložit i do odpovídajícího typu pracovního listu. Takže v souboru `./Assets/Resources/types.json` do seznamu pracovních listů `worksheetIds` přidáte `Id` (název souboru scény bez přípony) vytvořeného pracovního listu.

- Poté spustíte projekt aplikace v editoru Unity a pod lištou `File` v nastavení `Build settings` přidáte vytvořenou scénu do `Scenes In Build`.

- Nyní lze aplikaci v editoru spustit, nebo projekt zkompilovat a nainstalovat aplikaci na vašem Android zařízení. V hlavním menu pro spuštění pracovního listu zvolte ikonku `Třídění` (rozklikněte první kategorii a zvolte první ikonku). Po kliknutí na typ `Třídění` se náhodně spustí jeden z pracovních listů tohoto typu. Proto může chvíli trvat, než se spustí pracovní list, který jste vložili.

Příloha E

Obsah přiloženého paměťového média

- `readme.txt` ..... stručný popis obsahu paměťového média
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `eduapp` ..... zdrojové kódy aplikace Myšák
    - `backend` ..... zdrojové kódy backendu
    - `frontend` ..... zdrojové kódy frontendu
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `attachment` ..... přílohy práce