Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Implementace pozornostních a paměťových zkoušek v Unity u pacientů s obtížemi v oblasti paměti a pozornosti

Plzeň 2023 Bc. Jan Bartošek

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jan BARTOŠEK**
Osobní číslo:	**A21N0039P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Téma práce:	**Implementace pozornostních a paměťových zkoušek v Unity u pacientů s obtížemi v oblasti paměti a pozornosti**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s existující implementací parametrizovaných úloh v rámci systému BrainIn.
2. Prostudujte problematiku testování paměti a pozornosti u lidí s problémy v oblasti pozornosti a krátkodobé paměti.
3. Dle požadavků klinických psychologů navrhněte obsah 3 testovacích parametrizovaných úloh a navrhněte vylepšení existující implementace a komunikace úloh se systémem BrainIn.
4. Dle návrhu v bodě 3 implementujte 3 úlohy v Unity a začleňte je do webové aplikace BrainIn.
5. Ověřte kvalitu implementovaného řešení, otestujte úlohy na reprezentativním vzorku testovacích subjektů a zhodnoťte dosažené výsledky práce.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Petr Brůha**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **9. září 2022**
Termín odevzdání diplomové práce: **18. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2022

Poděkování

Touto cestou bych chtěl poděkovat svému vedoucímu diplomové práce Ing. Petru Brůhovi za odborné vedení a cenné rady. Dále bych chtěl poděkovat PhDr. Pavlu Králi, Ph.D. za odbornou pomoc v oblasti klinické psychologie a PhDr. Karolíně Malé za odborné rady v oblasti psychoterapie.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 10. května 2023

Bc. Jan Bartošek

Abstract

Master thesis aims at creating games that can be used in the treatment of people suffering from attention and short-term memory disorders, and at implementing one template that could be used in the further development of these games. The theoretical part of the thesis explores the existing implementation of games in the BrainIn web application. Next, the theory of cognitive functions, attention and memory is briefly described. The practical part describes the design and implementation of a simple extensible template and 3 games that could help patients in their treatment.

Abstrakt

Tato diplomová práce je zaměřena na vytvoření her, které lze využít při terapiích osob trpících poruchami pozornosti a krátkodobé paměti, a na implementaci jedné kostry (šablony), jež by mohla být použita při dalším vývoji těchto her. V teoretické části práce je prozkoumána existující implementace her v systému BrainIn. Dále je krátce popsána teorie kognitivních funkcí, pozornosti a paměti. V praktické části je popsán návrh a implementace jednoduše rozšiřitelné kostry a 3 her, které napomáhají pacientům při jejich léčbě.

Obsah

1 Úvod

Moderní technologie poskytují nové příležitosti pro léčbu pacientů s neurologickými potížemi (poruchami řeči, motoriky, paměti, pozornosti). Jednou z nich je webový systém BrainIn, jehož účelem je umožnit pacientům léčit se z pohodlí domova, a to bez nutnosti přítomnosti terapeuta nebo zdravotní sestry. Terapeut obvykle zadá svému přiřazenému pacientovi balík předem definovaných her. Ten je ve svém volném čase hraje, čímž vytváří výstupní hodnoty, které slouží k sledování jeho výkonu a potenciálního zlepšení.

Cílem této práce je nejen vytvoření 3 zkoušek (her), které pacientům pomohou s trénováním pozornosti a krátkodobé paměti, ale také vylepšení existující implementace a zjednodušení předávání hodnot z (do) webové aplikace. Toho by mohlo být dosaženo navržením a implementováním jednoho vzoru (šablony) tak, aby budoucí rozšíření těchto her, nebo implementace jiných, byla co nejjednodušší a nejrychlejší. Šablona by měla být vytvořena na základě doporučených postupů v softwarovém inženýrství, měla by být jednoduše rozšiřitelná a udržovatelná. Navržené hry budou disponovat svým flexibilním průběhem, který bude konfigurovatelný podle aktuálních požadavků, a to hlavně z důvodu jejich přizpůsobení jednotlivým pacientům trpícím různě vážnými problémy.

Kvalita navržených a implementovaných her by měla být dostatečně ověřena běžně používanými metodami pro ověřování kvality software. Hry by měly být otestovány na reprezentativním vzorku skutečných osob. Dosažené výsledky z odehraných her budou zaznamenávány pro měření potenciálního zlepšení pacientů odborníky z řad klinických psychologů.

Diplomová práce se skládá z teoretické a praktické části. Teoretická část zkoumá implementaci již existující šablony a her, včetně webové aplikace BrainIn. Stručně zkoumá kognitivní funkce, zejména funkce paměti a pozornosti, společně s existujícími testy pro měření pacientova výkonu. Dále popisuje návrh jedné ucelené šablony her včetně vylepšení komunikace mezi hrami a hostitelským webovým portálem. Praktická část je věnována implementaci výše zmíněné šablony a jejímu rozšíření do 3 navržených her, detailnějšímu popisu vývoje a testování.

2 Webová aplikace BrainIn

V rámci projektu „Využití moderních informačních technologií pro neurorehabilitace pacientů se získaným poškozením mozku“, realizovaným v rámci programu *Interreg V-A*, byl na Západočeské univerzitě v Plzni vytvořen interaktivní webový portál BrainIn, jenž slouží pro neurorehabilitaci pacientům, kteří se potýkají s vážnými neurologickými problémy a napomáhá při jejich léčbě či tréninku. Vytvořením účtu získá terapeut přístup k možnostem tvorby úloh, které mohou být přizpůsobeny individuálním potřebám pacientů. Terapeut vytvořené úlohy pro své pacienty zařazuje do tzv. balíků (obsahujících i několik takových úloh), což umožňuje pacientům samostatně rehabilitovat klidně i z pohodlí domova. Tento způsob terapie může být pro pacienty velmi užitečný, zejména pokud podstupují dlouhodobé a opakující se rehabilitace. Výstupem z her jsou různé hodnoty, pomocí nichž mohou terapeuti sledovat a analyzovat pokrok pacienta a upravovat úlohy pacientovu aktuálnímu stavu. [2, 9, 24]

Obrázek 2.1: Logo webové aplikace BrainIn (zdroj: [8])

2.1 Prostředí aplikace

Webová aplikace se skládá z více sekcí. Níže je popis z nich těch nejdůležitějších.

2.1.1 Uživatelé

Každý člověk (terapeut, pacient, vývojář), který si ve webové aplikaci BrainIn vytvoří účet má přidělenou určitou roli, jež ovlivňuje v rámci aplikace

jeho možnosti a chování. Celkem je k dispozici 6 různých rolí (upravující možnosti, které jsou uživatelům dostupné) s následujícími specifikacemi:

- **Super-Administrátor** je nejmocnější rolí v systému. Byla vytvořena hlavně pro vývojáře, kteří se na vývoji webové aplikace a her podílí. Super-Administrátor zajišťuje nastavení prostředí aplikace, kontroluje chybové logy apod.
- **Administrátor** má na starosti správu a chod aplikace. Mezi jeho pravomoce patří správa uživatelů (např. přidávání práv nově registrovaným terapeutům na základě jejich žádosti) a vytváření nových šablon (her).
- **Terapeut** je rolí vytvořenou všem terapeutům a rehabilitačním sestrám, kteří mají za úkol vést léčbu svých pacientů. K této roli je možné libovolně (v rámci pracovního prostředí) přiřazovat pacienty. Těmto pacientům terapeut následně přiděluje balíky úloh a analyzuje jejich výsledky.
- **Super-Terapeut** je role připravená pro nadřízené ostatních terapeutů. Svými pravomocemi se od klasického terapeuta příliš neliší, je schopen jen několika dalších funkcionalit. Například může vytvořit pracovní prostředí, do kterého následně přidá všechny své podřízené terapeuty.
- **Pacient** může po přihlášení odehrát jemu přidělené hry. Systém disponuje i možností konverzace mezi terapeutem a pacientem. Teoreticky tuto funkcionalitu pro komunikaci s terapeutem mohou využívat rodinní příslušníci v případě, že pacient toho není ze zdravotních důvodů schopný (např. nedokáže psát souvislý text).

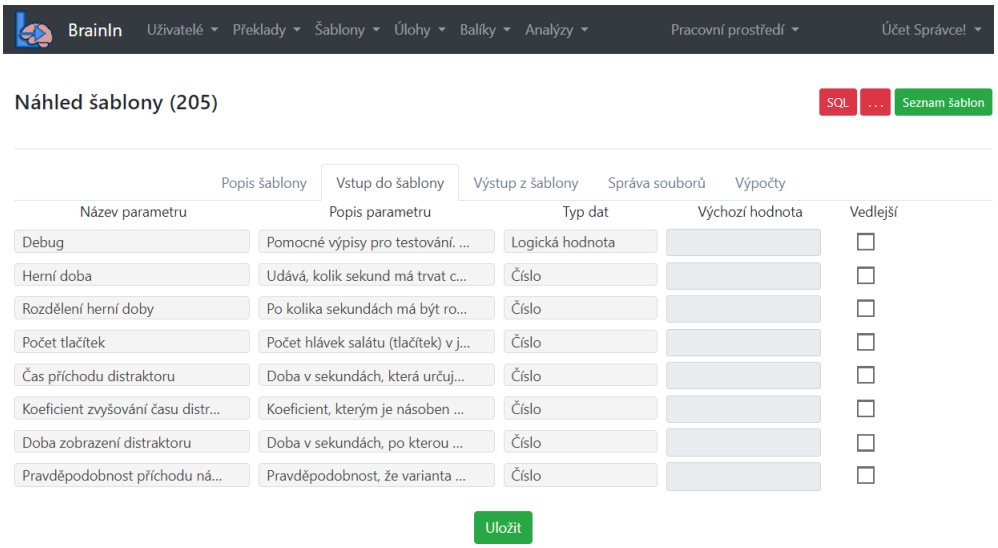
2.1.2 Pracovní prostředí

V systému BrainIn může vystupovat nespočet terapeutů a pacientů z několika nemocnic či neurologických léčeben. Je nežádoucí, aby kdokoliv věděl o zaměstnancích nebo dokonce o pacientech jiných lékařských zařízení. Z tohoto důvodu webová aplikace zavádí tzv. workspace (pracovní prostředí). Každý uživatel s rolí *Super-Terapeut* může vytvořit pro své zdravotnické zařízení jedno pracovní prostřední. Do něj potom může jednoduše přiřazovat terapeuty a své pacienty. Terapeuti si mohou přiřadit do péče pouze pacienty ze svého pracovního prostředí. Pacienti a lidé z ostatních pracovních prostředí jsou jim zcela skrytí. Vytvořené úlohy mohou být také přepnuty do

stavu, kdy jsou přístupné pouze kolegům z pracovního prostředí. Terapeut si tak může k vytvořené úloze bez obav zapisovat citlivé informace o daném pacientovi.

2.1.3 Šablona

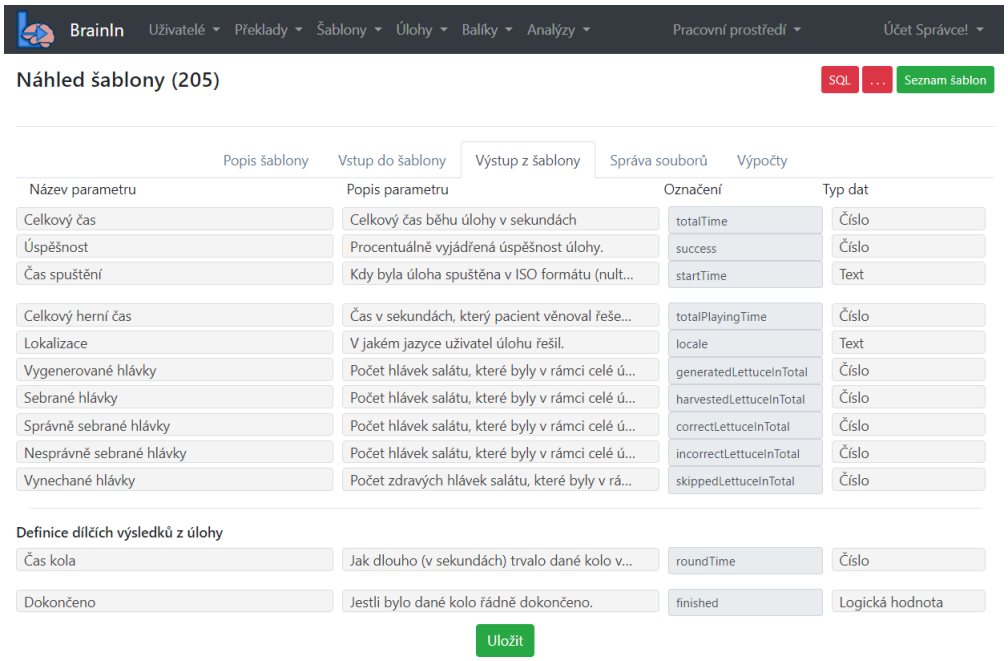
V této sekci je možné přidávat do systému nově implementované hry a definovat tak kontrakt mezi webovým systémem a konkrétní implementací jednotlivých her. Administrátor a vývojář her tedy mohou být dva různí lidé. Musí se však společně domluvit na formátu vstupních a výstupních parametrů dané hry. Hodnoty vstupních parametrů mohou ovlivňovat vzhled a průběh her. Hodnoty výstupních parametrů střádaných v průběhu pacientova hraní jsou po dokončení hry odesílány zpět do webové aplikace, kde jsou zpracovány. Formát a pořadí jednotlivých hodnot musí být totožný s definovaným formátem a pořadím výstupních parametrů v definici šablony (stejně jako u vstupních parametrů). Ukázky uživatelského rozhraní výše zmíněných částí jsou na obrázcích 2.2 a 2.3.



Obrázek 2.2: Ukázka části vstupních parametrů v sekci *Šablona*

2.1.4 Úloha

Tvořit nové úlohy pacientům na míru je proveditelné po definování šablony a nahrání sestavené hry do webové aplikace. Tvorbu úloh už má v zásadě na starosti terapeut, který definované parametry nahrazuje takovými hodnotami, jež mají vliv na vzhled či průběh hry (viz obr. 2.4). Hlavní výhodou



Obrázek 2.3: Ukázka části výstupních parametrů v sekci *Šablona*

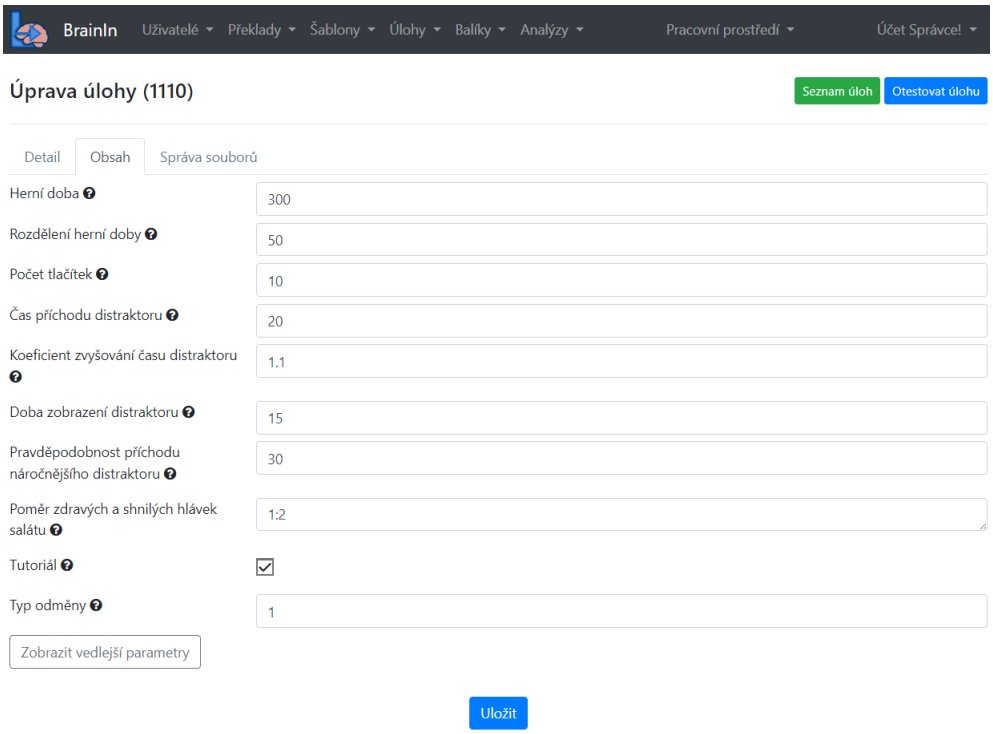
parametrizace je možnost takové úpravy úlohy, že může být připravena pro každého přímo na míru. Tvůrce úlohy vyplní hlavní parametry, ale může si po zobrazení spodní části upravit i parametry vedlejší. Ty jsou ve většině případů správně nastaveny již výchozími hodnotami. Mezi vedlejší parametry spadají informace o pomocných výpisech pro ladění, doba zobrazování výsledků apod.

2.1.5 Balík

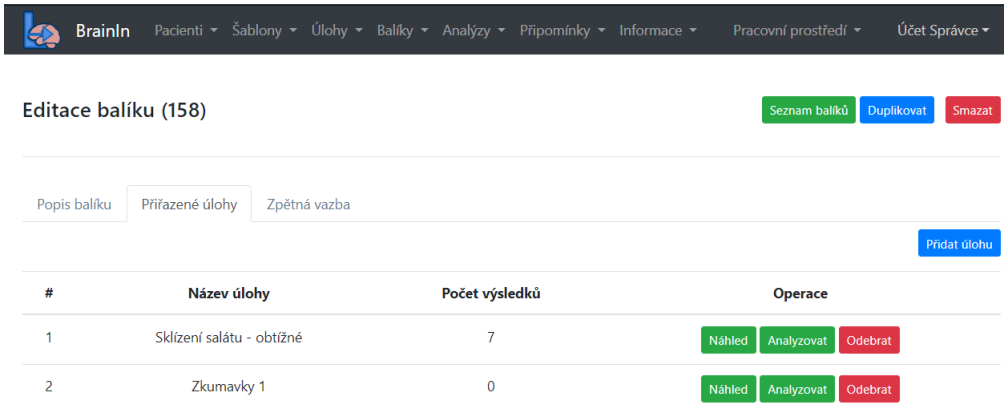
Terapeut přiřazuje úlohy svým pacientům přidáním jich do tzv. balíků. K balíkům je možné připojit libovolné množství úloh (viz obr. 2.5). Pacient si vybere konkrétní balík, načež jsou mu v řadě spouštěny úlohy do balíku přidané. Terapeut má přehled o všech úlohách v balících, jejich stavech (nové, probíhající, dokončené apod.) a o počtu výsledků (resp. počtu dokončených úloh a tedy počtu existujících analýz).

2.1.6 Analýza

Výstupní hodnoty vyprodukované jakoukoliv dokončenou úlohou jsou k prohlédnutí a dalšímu zkoumání v sekci *Analýzy*. Terapeut může filtrovat zobrazované výsledky na základě konkrétního pacienta, úlohy či šablony. Výstupní

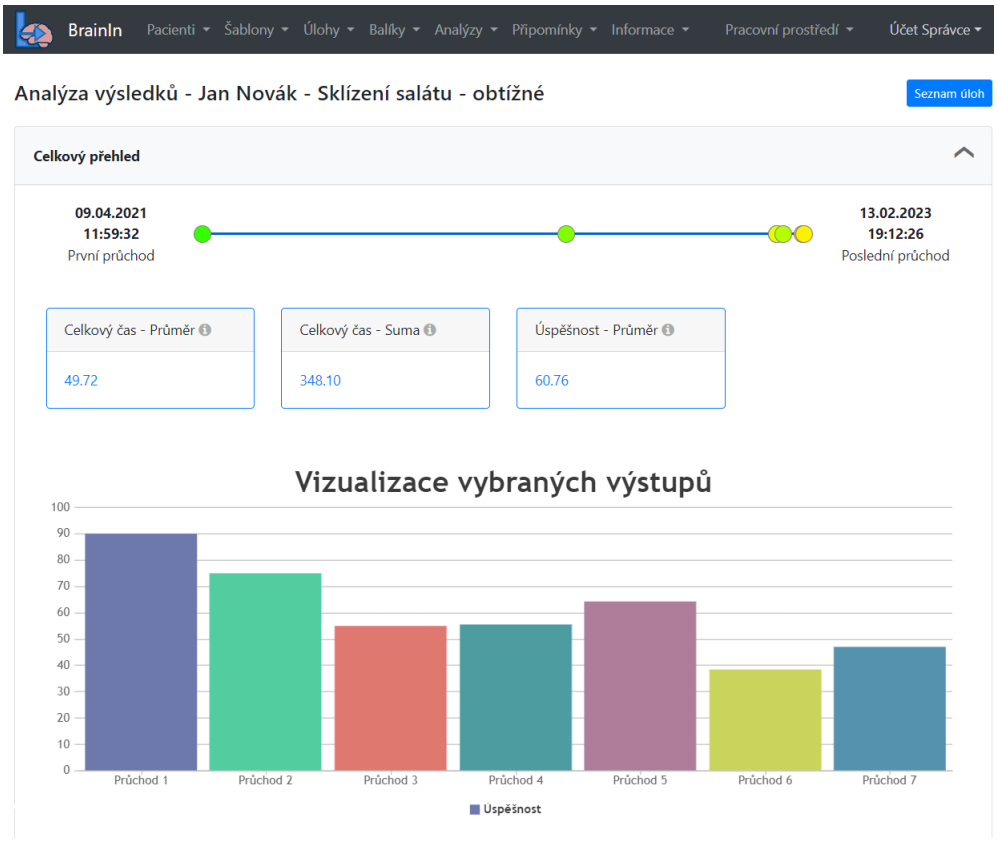


Obrázek 2.4: Ukázka definice hodnot vstupních parametrů konkrétní úlohy



Obrázek 2.5: Ukázka sekce *Balík* a přiřazených úloh

hodnoty z úloh se mohou vztahovat k odehranému času, informaci o dokončení úlohy, procentuální úspěšnosti nebo pozicím prvků v herní scéně apod. Terapeutům jsou prezentovány jak výsledky jednotlivých spuštění, tak i celkové výsledky (viz obr. 2.6) obsahující informace o průměrech herních dob nebo třeba celkové procentuální úspěšnosti.



Obrázek 2.6: Ukázka sekce *Analýzy* a obecných výsledků jednoho pacienta

2.1.7 Překlad

Aplikace BrainIn je vytvářena v rámci mezinárodního projektu [9]. Z tohoto důvodu jsou veškeré texty v uživatelském rozhraní prezentovány v třech jazycích – češtině, angličtině a němčině. Do těchto jazyků jsou přeloženy také texty všech integrovaných her. Webová aplikace zasílá informaci ohledně aktuální lokalizace do her společně se vstupními parametry. Implementace her by tedy měly obsahovat soubor se všemi lokalizovanými texty, které se zobrazují na základě získaného jazykového kódu z webového systému.

3 Existující implementace

V rámci projektu BrainIn bylo implementováno již několik her, které byly do webového systému integrovány.

3.1 Vlastnosti implementace

Systém BrainIn je připraven pro integraci her vyvíjených především v multiplatformním herním enginu[1] Unity. Hry jsou spouštěny pomocí skriptovacího jazyka JavaScript a WebGL[2]. Díky těmto technologiím dokáží obě aplikace komunikovat i za běhu Unity programu, a to prostřednictvím svých veřejných funkcí, které navzájem volají [29]. K souborům se sestavenou hrou je potřeba při nasazení přidat několik skriptů, jež danou hru spouštějí, řídí komunikaci a po dokončení hry předávají řízení zpět skriptům webové aplikace (viz obr. 3.1).

3.2 Herní engine Unity

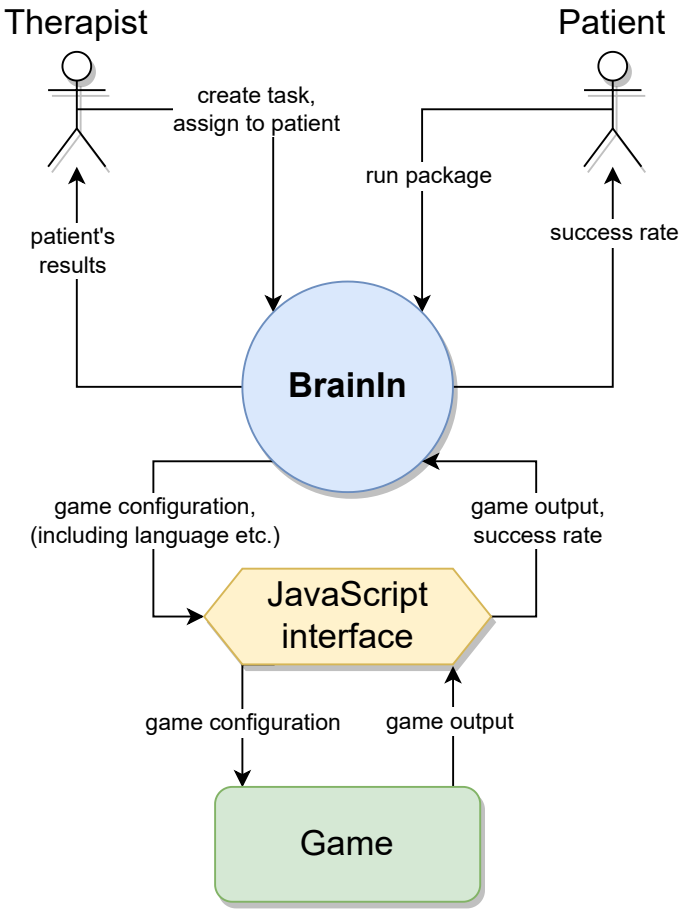
Unity je herní engine vytvořen společností Unity Technologies. V tomto nástroji je možné vyvíjet video hry s podporou více než 19 platforem. Ve vývojovém prostředí tohoto frameworku se vytváří především front-end aplikací, logiku her a chování jednotlivých herních objektu řídí C# skripty, jež se k herním objektům mohou připojovat. [12]

V herním enginu Unity je vytvořena například oblíbená česká hra ve virtuální realitě – *Beat Saber*.

Většina již vytvořených her pro webovou aplikaci BrainIn (včetně existující šablony všech her) byla vyvíjena v Unity ve verzi 2018.4.7f1.

[1] Herní engine je software, jenž poskytuje základní funkcionality, které jsou běžně používané při vývoji počítačových (video) her. [19]

[2] WebGL je JavaScriptové API pro nativní zobrazování grafiky. WebGL programy se skládají z obslužného kódu napsaného v JavaScriptu a tzv. shading language součástí OpenGL. [20]



Obrázek 3.1: Diagram propojení webové aplikace a Unity hry

3.2.1 Prvky vývojového prostředí

Vývojové prostředí Unity nabízí obrovské množství možností tvorby aplikací. Některé stavební kameny vývoje jsou popsány níže:

- Všechny herní objekty, uživatelské rozhraní a grafika se vkládají do tzv. scény (**Scene**). Aplikace může disponovat i několika různými scénami (např. pro hlavní menu, samotnou hru apod.).
- **Asset** je jakýkoliv soubor, jenž je možné vložit do Unity projektu, nebo ho v něm přímo vytvořit. Obvykle se do projektů přidávají např. 3D modely, které jsou vytvořeny v úplně v jiném nástroji. Dalšími příklady assetu vytvořeného v Unity mohou být skript nebo třeba prefab.
- Do každé scény jsou vkládány tzv. herní objekty (**GameObject**). Ke každému objektu je možné přidružovat všelijaké komponenty od základního skriptu ovládající objekt, až po mechanismus, jenž simuluje

působení gravitace. Tyto komponenty ovlivňují vlastnosti a chování herních objektů. Všechny prvky ve scéně disponují hierarchickou strukturou. Z tohoto důvodu mohou být některé objekty složeny z jiných herních objektů (např. obrazovka hlavního menu je složena z obrázku a 3 tlačítek – hrát, nastavení, ukončit).

- **Component** je volitelná součást prvků v scéně. Ke každému hernímu objektu je možné přidat komponentu, jež definuje jeho aspekty a chování. Komponentou mohou být vývojářem definované skripty, které ovlivňují např. pohyb objektu po herní scéně.
- **Prefab** je dříve vytvořený herní objekt, který je při vytváření herního prostředí vlastně znovupoužitelnou komponentou. Ve stavebnictví je k prefabu analogií např. okno nebo dveře. Stavbyvedoucí nemusí při stavbě domů vytvářet nová okna, stačí použít ta, která už někdo vymyslel a vytvořil. Výhodou prefabů je jednoduchá instanciace herních objektů i za běhu programu.

Obrázek 3.2: Logo herního frameworku Unity

3.2.2 Průběh hry

Průběh programu je řízen skripty jednotlivých herních objektů. Tyto skripty, jež dědí od `MonoBehaviour`, mohou obsahovat několik speciálních funkcí ovlivňujících běh aplikace. Hlavními funkcemi jsou `Awake`, `Start` a `Update`. Funkce `Awake` je něco jako konstruktor třídy. Volá se při inicializaci herního objektu, a to vždy jen jednou. Funkce `Start` se volá v chvíli, kdy se herní objekt stane poprvé aktivním. To může být při načtení celé herní scény nebo i v průběhu hry (např. při vytváření objektu v kódu, nikoli ve vývojovém prostředí Unity). Volání funkce `Update` probíhá několikrát za sekundu, z tohoto důvodu je funkce ideální např. pro aktualizaci vlastností objektů, které se mohou v průběhu hry měnit. [12]

3.3 Vzor

Všechny již existující hry sdílejí některé své vlastnosti. Tyto vlastnosti se týkají nejen funkcionality, ale i uživatelského rozhraní nebo vstupních a výstupních parametrů. Z tohoto důvodu byl vytvořen jakýsi vzor pojmenovaný jako „šablona šablon“. Idea tohoto vzoru je taková, že všechny nově implementované hry budou využívat totožné uživatelské rozhraní, budou sdílet společné vlastnosti a jejich vývoj bude realizován jen doplněním vytvořené šablony. Bohužel tato šablona nebyla navržena tak, aby byla jednoduše rozšiřitelná. Hlavní skript `Main` ovládá několik různých věcí včetně hlavního řízení běhu aplikace, logiku dané hry nebo odpočítávání před každým kolem. Není tak dosaženo správného oddělení zodpovědností a většina implementovaného řešení je vázána těsnými závislostmi. Z tohoto důvodu není úplně jednoduché šablonu rozšířit nebo upravit pro aktuální potřeby vytvářené hry. Další nevýhodou je nemožnost jednotkového testování. Kvůli těsným závislostem je nemožné otestovat program po malých částech. Projekt není zcela vhodně strukturován, takže zorientovat se v něm trvá delší dobu. Po zběžném průchodu implementací bylo zjištěno, že i některé části projektu jsou nevhodně pojmenovány, a jsou tedy zavádějící, některé jsou realizovány zbytečně složitě a jejich zjednodušení by zvýšilo čitelnost a porozumění implementovaného řešení.

3.4 Komunikace

V sekci 3.1 stojí, že hry v Unity a webová aplikace BrainIn spolu komunikují voláním svých veřejných funkcí. Kontaktovat Unity hru z webového prohlížeče je možné zavoláním funkce `SendMessage` nad objektem představujícím běžící instanci hry. Při volání této funkce je nutné specifikovat název aktivního objektu v herní scéně a název metody nacházející se ve skriptu přidanému k tomuto objektu.

Zdrojový kód 3.1: Příklad komunikace z prohlížeče do Unity aplikace

```
1 unityInstance.SendMessage("MyGameObject", "MyFunction");
```

Komunikace z instance hry do prohlížeče je realizována podobnou metodou. Ve funkci `Application.ExternalCall` je nutné specifikovat název funkce definované v JavaScriptu společně se všemi argumenty, jež je potřeba do dané funkce předat.

Zdrojový kód 3.2: Příklad komunikace z Unity aplikace do prohlížeče

```
1 Application.ExternalCall("MyFunction", true, 5);
```

3.4.1 Vstupní hodnoty

Při spuštění hry je na základě vyplněných vstupních parametrů v definici dané úlohy vygenerován textový řetězec. Jednotlivé části tohoto vstupního řetězce jsou odděleny speciálními textovými znaky (např. několika po sobě jdoucími znaky #). Vstupní parametry jsou zastoupeny v řetězci v totožném pořadí, ve kterém byly definovány ve webové aplikaci. Přidáním nového vstupního parametru je extrémně složité, protože implementovaná hra musí počítat s přesným pořadím vstupních parametrů, které se po jejich získání snaží rozdělit do příslušných proměnných.

```
cs###2022-12-12T10:10:10.000Z###500##3##1##1##1##6
```

3.4.2 Výstupní hodnoty

Výstupní parametry jsou zasílány do webové aplikace opět v podobném textovém řetězci jako v sekci 3.4.1. Stejně jako u vstupních parametrů musí být výstupní parametry v textovém řetězci ve stejném pořadí jako jejich definice na webu. Hra musí zasílat do webové aplikace přesně to, co webová aplikace vyžaduje. Příjem výstupních hodnot mohou narušit i doplňující informace, které však v šabloně nejsou definovány. Úprava výstupních parametrů je znovu extrémně složitá. V případě, že by se ukázalo, že jeden či více výstupních parametrů je u hry nežádoucí, je nutné upravit i implementovanou hru, znovu ji sestavit a nasadit. Webová aplikace totiž očekává výstupní hodnoty v přesném pořadí. Odstraněním definice jedné z nich vznikne kolize mezi tím, co aplikace očekává a tím, co hra odesílá.

4 Teorie paměti a pozornosti

4.1 Poznávací funkce

Kognitivní funkce zahrnují nejen schopnosti přijímat nové podněty či informace a zpracovávat je, ale i interagovat s okolím pomocí signálů, řečí nebo chováním. Tyto procesy se skládají z pozornosti a vnímání. Využívají ukládání a vybavování informací z paměti, zpracovávají vnímání pomocí myšlení, vedou k plánování a rozhodnutí atp. Tyto procesy spolupracují a umožňují člověku úspěšně interagovat s prostředím a řešit určité úkoly. Kognitivní funkce zahrnují koncentraci, paměť, pozornost, rychlost myšlení a dovednost pochopit nové informace. Mezi poznávací funkce spadají i tzv. exekutivní funkce. Mezi ně se řadí takové aktivity, jejichž účelem je dosažení konkrétního cíle: zahájení, plánování a organizace. [1]

Výše popsané funkce se vyskytují v různých oblastech mozku. Úraz hlavy nemusí vždy způsobit narušení úplně všech funkcí, ale třeba jen těch, které se vyskytují v porušené oblasti. Porušení jakékoliv z výše zmíněných funkcí se projevuje komplikací běžného života. Tyto poruchy jsou běžné a často nebývají na první pohled snadno odhalitelné. Poruchy kognitivních funkcí mohou být způsobeny různými faktory, jako jsou poranění hlavy, cévní mozkové příhody, neurodegenerativní onemocnění nebo psychiatrická onemocnění. [7, 32]

4.1.1 Pozornost

Pozornost je psychický stav jedince, jenž se projevuje soustředěností a zaměřeností vědomí. Soustředění pozornosti umožňuje průběh různých kognitivních procesů, jako je například vnímání, paměť nebo myšlení, a má vliv i na chování. Závisí na ní i to, na co se daný jedinec zaměřuje, nad čím přemýšlí a co si zapamatuje. Pozornost je klíčová pro zpracování aktuálních informací, uchování těchto informací v krátkodobé paměti a následné použití informací, které jsou nutné pro nějaký konkrétní účel. Například pro doplnění představy nebo pro řešení úkolu, který vyžaduje kombinaci různých znalostí. Pozornost působí na různých úrovních zpracování podnětů, zvyšuje zaměření na vybrané podněty a snižuje zaměření na ty, které nejsou důležité nebo nejsou dostatečně intenzivní, aby si pozornost samy upoutaly. Obecně se projevuje vliv pozornosti tím, že umožňuje zaměřit se na to, co je důležité a potlačovat to, co je nepodstatné nebo rušivé. [31]

Pozornost reguluje různé psychické procesy, jako jsou prožívání, poznávání a chování:

- Díky pozornosti je jedinec schopen orientovat se v konkrétních okolnostech a uvědomit si jejich význam a důsledky. Pozornost může ovlivnit směr jeho úvah, na co se bude koncentrovat a na co naopak reagovat nebude. [30]
- Přispívá k propojení minulých zkušeností s nově získanými poznatky. Pokud se jedinec soustředí na čerstvé podněty, obvykle si také vybavuje další poznatky, které s touto oblastí souvisí. [30]
- Pozornost je důležitou součástí kontroly a plánování příštích činností, které vychází z hodnocení situace a zkušeností. Tím může ovlivnit budoucí zaměření a následné jednání. [30]

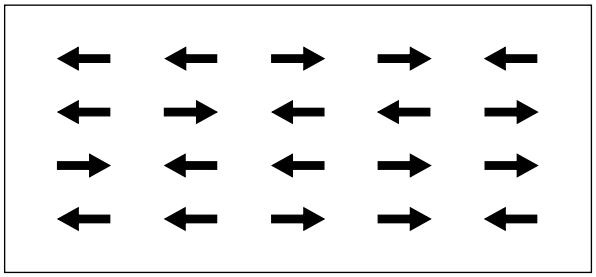
Koncentrace

Jedná se o vyjádření míry, s jakou se jedinec zaměřuje na určité impulsy nebo aktivity. Úroveň soustředění se mění v závislosti na aktuálních okolnostech a jednání. Nejnáročnější jsou zejména nové činnosti, které jedinec dosud neovládl, neboť vyžadují větší míru koncentrace na jednotlivé kroky. Jakmile si člověk konkrétní dovednosti osvojí, nemusí jim dále věnovat tolik pozornosti. Zautomatizované a zafixované činnosti a dovednosti obvykle probíhají nevědomě, člověk je vykonává automaticky a zbývající pozornost může být využita pro důležitější aktivity. Pokud však dojde ke změně situace, může se pozornost aktivovat a umožní jedinci účinně reagovat na nové podmínky. Zkušený řidič automobilu nemusí věnovat příliš pozornosti plynovému, spojkovému a brzdovému pedálu. Řízení motorového vozidla zvládá automaticky, dokud na silnici nenastane nečekaná situace, kterou musí řešit (např. uzavírka). [30, 31]

Soustředění na různé impulsy mohou být různě těžká, a to zejména kvůli způsobu jejich prezentace. Vizuální podněty mají často výhodu delšího trvání, kdy je jedinec může vnímat dlouho, a dokáže se na ně snáze soustředit. Soustředit se na zvukové podněty, zejména na mluvenou řeč, je náročnější kvůli omezené délce jejího trvání. Lidé se totiž nemohou k těmto informacím vracet, a nemohou je tak vnímat déle jako vizuální informace. Musí se na ni soustředit v době, kdy mluvčí, například přednášející, zrovna mluví. [30, 31]

Stabilita

Vytrvalost pozornosti je schopnost udržení soustředění na určitou aktivitu nebo činnost po dostatečně dlouhou dobu. Zaměření se na jednu činnost po delší dobu může mít za následek zhoršení soustředění, únavu, ztrátu motivace a podobně. Stabilita pozornosti je závislá na úrovni jejího řízení. Exekutivní funkce mohou vytrvalost pozornosti přímo ovlivňovat. Jedná se např. o pracovní paměť, o schopnost potlačovat tendenci k nežádoucím reakcím nebo o přepínání mezi různými činnostmi a podněty. Stabilitu pozornosti je možné hodnotit pomocí tzv. flank testu, kde si dotyčný má všímat jednoho směru šipky, jež je umístěna mezi dalšími šipkami opačného směru (viz obr. 4.1). [30, 31]



Obrázek 4.1: Ukázka flank testu (zdroj: [31])

Selektivita

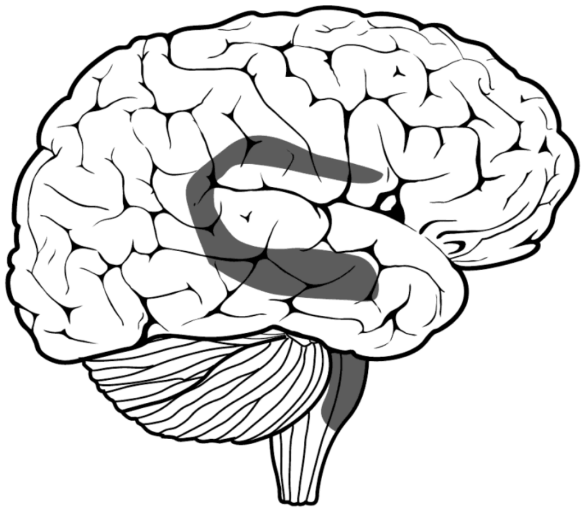
Selektivita pozornosti je schopnost zaměřit se jen na podstatné podněty a přehlížet ty nepodstatné. Není možné aktivně vnímat vše v našem okolí nebo se věnovat více aktivitám současně. Selektivita poskytuje lidem dovednost reagovat pouze na určitý typ podnětů a ostatní přehlížet. Tato schopnost volby je velmi důležitá, bez ní by si musel jedinec všímat např. rozhovoru svých spolupracovníků a nemohl by se plně soustředit na svou práci. [30, 31]

Flexibilita

Flexibilita pozornosti popisuje tzv. přepínání a rozdělování pozornosti mezi různé činnosti nebo více informačních zdrojů. Distribuce pozornosti je nezbytná pro zvládnutí určitých úkolů, ale děje se tak na úkor její koncentrace. Čím více je každá z vykonávaných činností více zautomatizovaná a nevyžaduje tedy takovou míru koncentrace, tím je přenášení pozornosti z jednoho podnětu na druhý jednodušší. [30, 31]

4.1.2 Paměť

Paměť je nejspíše centrálním prvkem pro všechny kognitivní funkce a pravděpodobně pro vše, co je charakteristické pro lidské chování. Paměť umožňuje uložení informací, jejich nalezení a opětovné použití. Paměť je úzce spojena s myšlením, vnímáním a prožíváním. Usnadňuje orientaci, umožňuje poznávat známé tváře, situace a správně na ně reagovat. Závažné poškození paměti oddělí pacienty od prakticky smysluplného kontaktu s okolním světem a zbavuje je smyslu života. [15, 31]



Obrázek 4.2: Mozkové oblasti, které umožňují zapamatování a uchování získaných poznatků (zdroj: [31])

Krátkodobá paměť

Krátkodobá paměť uchovává získané informace po určitou nepříliš dlouhou dobu. Alternativou k ní je pracovní paměť, jež bývá někdy odlišována od krátkodobé paměti na základě faktu, že neslouží pouze k uchování získaných informací, ale i jejich dalšímu uspořádání a zpracování (funguje jako tzv. operativní komponenta manipulující s aktuálními poznatky). [31]

Pracovní paměť umožňuje dočasně uchovat informace, které jsou v daném okamžiku potřebné, bez ohledu na to, zda je jedinec právě vnímá, nebo si je vybavil. Pokud informace nejsou zpracovány, rychle z pracovní paměti mizí, obvykle do 30 sekund. Obsah pracovní paměti není příliš stabilní a může být snadno ovlivněn dalšími podněty. Kapacita pracovní paměti je omezená a závisí na počtu jednotek, které může člověk v paměti udržet. Uchování informací v pracovní paměti závisí na jejich množství, délce a složitosti, ale i na způsobu, jakým jsou prezentovány. Bylo vypozorováno, že když si

člověk řadu čísel nebo slov přečte nahlas, dosáhne lepšího výsledku, než když ji pouze slyší nebo čte v duchu. [31]

4.2 Testové metody

Testové metody reprezentují standardizovaný postup vyšetření dodržující specifická pravidla. Tento postup používá jednoduché nástroje a k zpracování nabytých poznatků přistupuje jednotným způsobem. Tyto metody jsou v skutečnosti jakýmsi pokusem, jelikož navozují určitá jednání vyšetřované osoby v kontrolovaných podmínkách. [16]

4.2.1 Výkonové testy

Testy schopností (výkonové testy) se řadí k nejstarším diagnostickým postupům v oblasti psychologie. Název naznačuje, že jsou zaměřeny na měření výkonu a úspěšnosti vyšetřované osoby. Tyto testy nabízejí lepší příležitosti pro měření, řazení a vyhodnocování výsledků odpovědí testovaných osob. Tyto testy jsou založeny na objektivních kritériích a výsledky nejsou ovlivněny subjektivními motivy testovaných jednotlivců, jako je úmyslné klamání nebo ovlivňování výsledků. Mezi výkonové testy patří např. testy inteligence, testy speciálních schopností a jednotlivých psychických funkcí, testy paměti, testy kreativity nebo testy vědomostí. [16]

4.2.2 Testy paměti

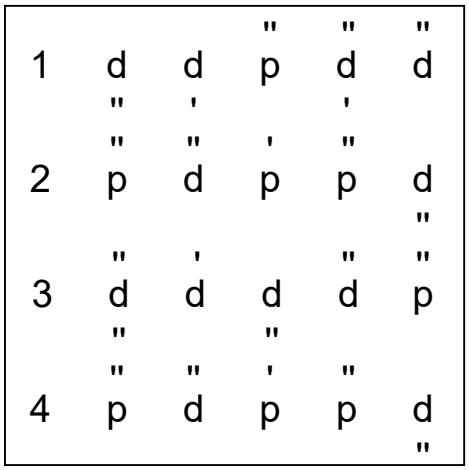
Obvykle jsou vytvářeny testy, které ověřují a trénují více schopností a psychických funkcí. V menší míře existují i metody zaměřené primárně na zjišťování paměti. Velké množství těchto testů je však tajných a je zakázána distribuce jich civilním osobám. [16]

- New Word Learning and Retention Test je orientován na odhalení poruch paměti. Obsahuje slovníkový test verbálního učení a zapamatování si nových slov orientovaných na běžné paměťové situace. [16]
- Wechsler Memory Scale je test k dosažení tzv. paměťového kvocientu. Test zkoumá zrakovou a sluchovou paměť vyobrazováním např. obrázku rodiny na zahradě a následným doptáváním se na jejich činnosti. [16]
- LGT-3 je test obsahující několik subtestů, ve kterých si má proband např. zapamatovat plány imaginárního města a má odpovědět na dotaz, jak se dostane k muzeu atp. [16]

4.2.3 Testy pozornosti

Testy pozornosti obvykle souvisí s testy paměti [16]. Níže je popis několika z nich.

- Při testu koncentrace pozornosti, který vytvořil v roce 1980 Miloslav Kučera, má proband porovnávat levý a pravý sloupec, v nichž jsou různé znaky. Každý řádek pravého sloupce se mírně liší od řádku v levém sloupci a cílem testu je jej podle levého sloupce opravovat. [16]
- NQ-S zátěžový test je připraven výlučně pro počítačové použití. Při realizaci testu je na monitoru zobrazen číselný čtverec, ve kterém má proband nalézt jedno číslo, které je mu prezentováno v levém horním rohu obrazovky. [16]
- Test pozornosti d2 tvoří formulář obsahující pouze písmena „d“ a „p“, která mají nad sebou či pod sebou nakresleny jednu, dvě nebo žádnou čárku. Úkolem probanda je označit všechna písmena „d“ mající právě dvě čárky nehledě na jejich umístění (viz obr. 4.3). [16]



Obrázek 4.3: Ukázka d2 testu (zdroj: [16])

5 Návrh

Pro pacienty s potížemi v oblasti krátkodobé paměti a pozornosti je nutné vytvořit 3 hry, které poslouží k jejich rehabilitaci a tréninku příslušných kognitivních funkcí. Navržené hry budou využívány PhDr. Pavlem Králem, Ph.D., vedoucím Katedry klinické psychologie při Institutu postgraduálního vzdělávání ve zdravotnictví. Navržené a implementované hry bude nutné integrovat do systému BrainIn. Pan doktor Král definoval několik základních požadavků na vytvářené hry:

- Průběh každé hry by měl reflektovat herní průběh dohodnutý při iniciační schůzce.
- Vzhled herních prvků her by měl odpovídat náčrtu v návrhu herní scény (konkrétní detaily je však možné zpracovat libovolně).
- Hry bude možné konfigurovat pomocí dohodnutých vstupních parametrů (také jedna z podmínek integrace her do webové aplikace BrainIn).
- Projekt bude v průběhu vypracování schopen reagovat na změny požadavků.

Hry by měly být postavené na rozšiřitelné, jednoduše udržovatelné a otestované kostře, která bude následovat „best practices“ v softwarovém inženýrství. Navržená kostra (a potom i každá hra z ní vytvořená) bude reagovat na konkrétní lokalizaci přicházející společně s vstupními parametry z webové aplikace.

Komunikace mezi hrou a webovým systémem by měla být jednoduchá a rozšiřitelná, aby případné změny v definici vstupních a výstupních parametrů byly co nejvíce jednoduché a rychlé, nebo nevyžadovali opětovné sestavení implementovaných her. Reakce na změnu parametrů bez nutnosti opětovného sestavení Unity her se týká nejspíše jen odebírání výstupních parametrů ve webové aplikaci. V případě jejich přidávání je nutné, aby je hra posílala (pokud to dosud nedělá a webová aplikace je úspěšně neignoruje). V případě vstupních parametrů je může webová aplikace do hry posílat, ale když na ně hra nereaguje, tak postrádají smyslu. Odebrání vstupního parametru z definice úlohy bez sestavení nové verze hry je nemožné, protože hra nejspíše počítá s vstupním parametrem, který by byl odebrán.

5.1 Šablona

Tato sekce se věnuje návrhu znovupoužitelné kostry při vývoji nových her pro systém BrainIn.

5.1.1 Unity

V sekci 3.2 stojí, že pro již existující hry byl použit editor Unity ve verzi 2018.4.7f1. S každou novou verzí je v tomto herním frameworku opraveno spoustu chyb, přidána nová rozšíření usnadňující vývoj apod. Z tohoto důvodu by bylo příhodné zvolit pro implementaci novější verzi Unity. Vhodnou verzí by mohla být jedna z verzí LTS.

Unity LTS (Long-Term Support) je verze, která je určena pro vývojáře, kteří potřebují stabilní a spolehlivou verzi enginu pro své projekty. LTS verze jsou obvykle podporovány po dobu 2 let, což znamená, že během této doby jsou opravovány chyby a vydávány bezpečnostní aktualizace. Hlavním cílem Unity LTS je poskytnout vývojářům zajištění, že pokud se rozhodnou použít tuto verzi enginu, budou mít dlouhodobou podporu a záruku, že jejich projekt bude fungovat správně a bude moci být aktualizován bez nebezpečí, že se něco pokazí. Na rozdíl od verze Unity s krátkodobou podporou, jež je určena pro vývojáře, kteří chtějí vyzkoušet nejnovější funkce a vylepšení, Unity LTS se zaměřuje na stabilitu, spolehlivost a konzistenci vývojářského procesu.

Nejnovější doporučovanou LTS verzí Unity je dle Unity Hubu[1] verze 2021.3.20f1. Tyto 2021.3.XX verze mají však, dle oficiálních fór a portálu pro ohlašování chyb v editorech Unity, občas problémy se sestavením programů do WebGL (konkrétně někde v IL2CPP). Z tohoto důvodu by mohlo být upuštěno od použití LTS verze, místo ní by mohla být použita jedna z nejnovějších verzí obohacená o nové funkcionality např. 2022.1.23f1, v níž se výše zmíněné chyby nevyskytují.

5.1.2 Architektura

Jednoduché projekty v Unity pravděpodobně nevyžadují využití nějakého známého vzoru softwarové architektury. Veškerý kód je většinou obsažen přímo v skriptech, které jsou k herním objektům přidruženy jako komponenty. Průběh hry je poté řízen přímo těmito skripty.

V tomto případě by měl být výsledný produkt snadno rozšiřitelný a srozumitelný, proto je vhodné nad architektonickými vzory a styly uvažovat.

[1] Oficiální instalátor verzí vývojových prostředí Unity

Entity Component System

Entity-Component-System (ECS) je softwarová architektura primárně používaná v oblasti vývoje videoher, která využívá datově orientovaný přístup. ECS se skládá ze tří hlavních konceptů - entit, komponent a systémů. Entity představují herní objekty, ale neobsahují žádná data ani metody. Komponenty obsahují data popisující vlastnosti entity, jako jsou barva, velikost a rychlost. Systémy definují herní logiku a přistupují ke komponentám entit pro jejich zpracování a aktualizaci. ECS odděluje data a logiku, což usnadňuje přidávání nových herních mechanismů nebo interakčních modalit s omezeným dopadem na existující kód. Dále optimalizuje přístup ke komponentám a umožňuje lepší kontrolu organizace dat v paměti. Tento vzor je užitečný pro vytváření různých reprezentací herních objektů, které lze měnit v reálném čase bez vlivu na herní logiku, což z něj dělá vhodnou volbu pro vývoj videoher. [17]

Právě v této architektuře je vidět ten klasický přístup vývoje her v Unity, kde entitou jsou herní objekty, komponentou právě komponenty těchto herních objektů a systémem jsou herní skripty, které jsou k objektům připojovány.

Model-View-Controller

Hlavní myšlenkou za architektonickým vzorem MVC je, že každá část kódu má jiný účel. Některé části kódu uchovávají data, jiné se starají o vzhled aplikace, zatímco jiné řídí, jak aplikace funguje. Tento vzor odděluje datovou logiku, business logiku a uživatelské rozhraní, takže rozděluje frontendový a backendový kód do samostatných komponent a poskytuje modularitu, čitelnost a snadnost testování. Poskytuje volné propojení mezi jednotlivými elementy a skládá se ze tří částí: Model, View a Controller. Model implementuje logiku pro datovou doménu aplikace. View je komponentou, která zobrazuje uživatelské rozhraní (UI) aplikace. Controller funguje jako prostředník mezi view a modelem a umožňuje propojení těchto dvou komponent. [14]

V případě Unity her by View bylo reprezentováno herními objekty v herní scéně vytvářenými přímo ve vývojovém prostředí Unity nebo za běhu programu instanciací. Ke všem (nebo alespoň k většině) herním objektům by byl přiřazen skript (Controller), který by představoval jeho logiku. Modelem by byli všichni backendoví manažeři, služby a ostatní backendové třídy, jejichž úkolem by byla správa dat a dalšího vnitřního chování programu.

5.1.3 Vstupní bod

Programy vytvořené v Unity nemají předem definovaný vstupní bod (entry point) jako programy implementované v jiných programovacích jazycích (např. funkce `main` v C++). Po spuštění se začne vykonávat kód v metodách `Awake` a `Start` u aktivních objektů ve scéně.

Jeden vstupní bod (single-entry point) však napomůže v lepším vytváření a konfiguraci globálních objektů. Dále je pak možné mít pod kontrolou pořadí inicializace různých backendových manažerů apod. [4]

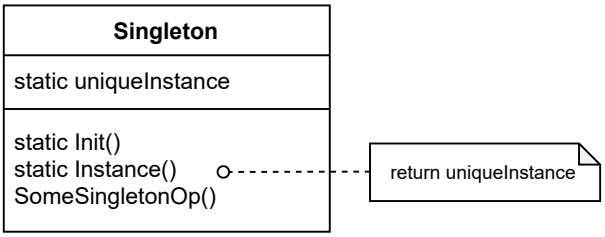
5.1.4 Vkládání závislostí

K dosažení rozšiřitelnosti a dobré testovatelnosti výsledného řešení je potřeba vyřešit tzv. těsné závislosti. Těsné závislosti snižují obecnou flexibilitu aplikace a zvyšují náročnost na údržbu. [18]

Těsné závislosti je možné vyřešit pomocí vkládání závislostí (Dependency Injection), a to např. použitím externí entity (označované jako kontejner nebo injektor), jež vkládá potřebné služby do tříd na nich závislých. Dependency Injection lze realizovat pomocí konstruktoru, nastavením metodou setteru nebo pomocí rozhraní třídy. [18]

Jelikož by i frontendové třídy měly možnost využívat služeb backendových tříd, získání konkrétních závislostí by mohlo být realizováno pomocí návrhového vzoru Service-Locator. Service-Locator je možné si představit jako kontejner všech vytvořených instancí, jenž na tyto instance poskytuje reference pomocí definovaných globálních přístupových bodů. Ačkoliv je tento návrhový vzor občas považován za anti-pattern, v tomto případě najde své uplatnění při vkládání závislostí do tříd herních objektů a všech ostatních UI tříd. Ostatní manažeři a třídy by mohli využívat obou principů, jelikož většina z nich bude vytvářena při samotné inicializaci hry a reference na vytvořené instance si mohou předat přímo v konstruktoru (budou-li vytvářeny ve správném pořadí). [13]

Service-Locator je typicky vytvářen pomocí návrhového vzoru Singleton. Návrhový vzor Jedináček zajišťuje, že daná třída má pouze jednu instanci, a k ní poskytuje globální přístupový bod (což je přesně to, co je od kontejneru všech instancí vyžadováno). Singleton definuje metodu `Instance` (typicky se jedná o statickou metodu nebo veřejný atribut třídy) poskytující klientům přístup ke své jediné instanci (viz obr. 5.1). Zároveň může být zodpovědný za vytváření své vlastní jediné instance. Taková třída má totiž většinou privátní konstruktor, čímž zabraňuje vytváření instance z jiných částí programu. [11, 21]



Obrázek 5.1: UML diagram třídy návrhového vzoru Singleton - Jedináček

5.1.5 Lokalizace

Součástí vstupních parametrů je také aktuální nastavení jazyka ve webové aplikaci BrainIn. Každá hra by měla reagovat na konkrétní jazykový kód a po jeho zpracování by měl být každý text v aplikaci převeden do jedné z předem definovaných jazykových mutací. Mechanismus, který bude správné nastavování lokalizace řídit, by měl umět pracovat s importem lokalizačních souborů. To zajistí, že může být kdykoliv upraven kterýkoliv překlad bez nutnosti opětovného sestavení aplikace.

Herní engine Unity nabízí použití balíčku *Localization*, který požadované chování nabízí. Umožňuje přidávat lokalizované texty, které následně poskytuje podle konkrétních klíčů a aktuálně nastaveného jazyka. Nabízí také import a export lokalizačních souborů ve formě XLIFF, CSV a Google Sheets. [28]

5.1.6 Komunikace

Zasílané zprávy od webové aplikace může přijímat pouze aktivní herní objekt v scéně, tedy konkrétní metoda skriptu připojeného k aktivnímu hernímu objektu. Odesílat zprávy je potom možné z jakékoliv třídy. Na backendu by mohla být vytvořena třída, která se bude starat o komunikaci mezi hrou a webovou aplikací. Tato třída by reagovala (nikoliv přijímala) na všechny možné přijaté zprávy a zároveň by odesílala zprávy do webové aplikace. Na scéně vytvořený objekt (pojmenovaný např. „Listener“), jehož skript by disponoval referencí na instanci výše zmíněné třídy, by pouze naslouchal příchozím zprávám a řízení poté předával instanci na níž by měl referenci.

Výhodou tohoto přístupu bude i zjednodušení implementace a ladění hry ve vývojovém prostředí, kde obvykle žádné zprávy z webové aplikace hra nepřijímá. Vývojář bude schopen jednoduše nasimulovat potenciální přijaté zprávy (a reakce na ty odeslané). Vše bude schopen řídit z jednoho místa (z metod výše zmíněné třídy) a aplikace se bude i ve vývojovém prostředí chovat jako v prostředí reálném, tj. nasazená ve webové aplikaci BrainIn.

Vstupní a výstupní parametry

Implementovaná kostra by se měla vyhnout již existujícímu způsobu předávání vstupních/výstupních parametrů do/z hry. Tento způsob je absolutně neintuitivní a hlavně velmi náročný na rozšiřitelnost a potenciální změny. Místo toho by mohla tato forma komunikace probíhat předáváním si dat např. ve formátu *JSON*[2]. V jazyce C# existuje velké množství knihoven, které jsou schopné jednoduše převést (serializovat) datové třídy do textového řetězce ve formátu *JSON* a naopak deserializovat textový řetězec do instance třídy. V případě náročnějších konverzí je možné si vytvořit i vlastní konvertor, ve kterém lze specifikovat přesný postup (de)serializace. Jedním z nejrozšířenějších frameworků pro práci s *JSON* v jazyce C# je Json.NET od společnosti Newtonsoft.

Příkladem předávaných parametrů mezi webovou aplikací a hrou může být následující ukázka:

```
{
  "maxTime": 300,
  "rounds": 3,
  "lang": "cs"
}
```

Tento přístup bude vyžadovat úpravu odesílání vstupních parametrů a přijímání výstupních parametrů také ve webové aplikaci BrainIn. S těmito úpravami by měl být kladen důraz na zachování zpětné kompatibility. Nová varianta předávání parametrů by měla být implementována jako nová funkcionalita, nikoliv jako nahrazení starší varianty. S tímto přístupem by mohla být realizována i implementace samotné kostry her, kde by mohla být taktéž zachována starší varianta pro případ, že by to bylo někdy přímo vyžadováno. Aplikace by tedy měla především pracovat s rozhraním, aby bylo možné kdykoliv nahradit konkrétní implementace (např. „parser“ parametrů), jež si poradí s konkrétním formátem dat přijatým na vstupu. Webová aplikace BrainIn je implementována také v jazyce C# a využívá výše zmíněný framework pro práci s datovým formátem *JSON*, takže by potřebné změny neměly být příliš složité.

[2]JavaScript Object Notation je datový formát založený na datových typech v programovacím jazyce JavaScript. V posledních několika letech získal tento datový formát obrovskou popularitu nejen mezi webovými vývojáři. V dnešní době se jedná o hlavní formát pro výměnu informací a dat přes web. [22]

5.1.7 Event Aggregator

Za běhu programu může nastat situace, kdy bude nutné předat řízení programu z backendových manažerů přímo do jedné z frontendových tříd. Jednou z takových situací může být například ta, kdy herní objekt přijme zprávu z webového rozhraní s obsahem vstupních parametrů, jež bude třeba přesměrovat do příslušného manažera. Není však vhodné, aby v backendových třídách existovala přímá reference na třídy ovládající herní objekty. Z tohoto důvodu by mohl být implementován návrhový vzor Event Aggregator.

Návrhový vzor Event Aggregator je jakýsi prostředník a používá se v softwarovém inženýrství k řešení problémů spojených s komunikací mezi různými komponentami systému, které o sobě nemusí vůbec vědět. Tento návrhový vzor funguje na principu centrálního prostředníka, který přijímá zprávy (nebo události) od různých komponent aplikace a následně je distribuuje (publish) dalším komponentám, které se na tyto události předem přihlásily (subscribe). Tito příjemci se mohou přihlásit k odběru určitého typu události, a přijímat tak pouze relevantní informace. Hlavní výhodou tohoto vzoru je, že umožňuje oddělit zdroj události od cílových komponent, což výrazně snižuje vzájemnou závislost mezi těmito částmi aplikace. Komponenty, které produkují události, nemusí mít žádné znalosti o tom, jaké další komponenty je budou přijímat. Naopak komponenty, které na události reagují, nemusí mít žádné znalosti o publikujících komponentách těchto událostí. Vzor umožňuje jednoduchou a centralizovanou správu komunikace mezi různými komponentami aplikace. [33]

Event Aggregator je ve skutečnosti kombinací návrhových vzorů Observer a Mediator. Pozorovatel (Observer) je vzor definující vztah mezi různými komponentami tak, že když jeden změní svůj stav, tak jsou o této změně obeznámeni i ostatní. Obvykle existuje jeden pozorovaný objekt na více pozorovatelů. Prostředník (Mediator) umožňuje komunikaci mezi objekty, aniž by se navzájem znaly. Veškerou komunikaci mezi těmito objekty řídí jeden prostředník. Používáním návrhového vzoru Event Aggregator je možné přispět k větší rozšiřitelnosti a znovupoužitelnosti kódu. [3]

5.1.8 Logování

Součástí řešení by měl být také systém logování, a to jak informačních výpisů, tak i chyb nebo varování. Systém by měl být připraven na případné změny a rozšíření, aby bylo možné logy zapisovat i do souborů atp. Tuto vlastnost by mělo zajistit vytvoření rozhraní loggeru, který se bude používat napříč aplikací. Každá třída poté využije pro své logování konkrétní implementaci rozhraní. Pro aktuální účely kostry však bude stačit jednodu-

chý logger, který bude výpisy tisknout přímo do konzole (u Unity WebGL aplikací jsou logy zapisovány do konzole prohlížeče). Všechny logy by měly být zapisovány v nějakém konkrétním formátu, kde bude zřejmé, kdy byl log vytvořen a jaká třída jej vyprodukovala. Tímto chováním výchozí výpisy do konzole (`Debug.Log`) bohužel nedisponují.

5.1.9 Oddělení zodpovědností

Oddělení zodpovědností (Separation of Concerns) je osvědčenou praktikou rozdělení softwarového programu na jednotlivé části tak, aby se z hlediska funkcionality co nejméně překrývaly. To znamená, že každá část programu by měla být vytvořena s cílem plnit pouze určitou funkcionalitu a měla by být schopna ji vykonávat bez zásahu do jiných částí. Tohoto přístupu k návrhu a implementaci by mělo být využíváno nejen na backendu programu, ale i na frontendu.

Při návrhu a vývoji kostry (a poté konkrétních her) by měl být vytvořen pro každou obrazovku samostatný controller. Hlavní výhodou použití samostatných oddělených skriptů a tříd je, že kód se stává přehlednějším. Při použití jen jednoho skriptu pro více herních objektů je náročné pochopit, které úseky kódu odpovídají kterým herním prvkům, a jak mohou ovlivňovat ostatní. Dalšími výhodami jsou např. usnadnění změn funkčnosti aplikací nebo zlepšení modularity. Je možné provádět změny v konkrétních částech programu a díky tomu je snazší experimentovat a vytvářet prototypy nových požadavků a funkcionalit. Tento přístup zároveň umožňuje znovupoužití určitých částí i na jiných místech aplikace nebo v jiných projektech.

5.2 Návrh parametrů

Průběh navrhovaných her bude ovlivňován vstupními hodnotami, výstupní statistiky z pacientova snažení budou předávány do webové aplikace parametry výstupními.

5.2.1 Vstupní hodnoty

Vstupní hodnoty obvykle záleží na charakteru úlohy. Parametry pro hru specifické přímo ovlivňují konkrétní průběh hry, tj. u her pozornostních např. určují, na co má proband dávat pozor, u her paměťových potom definují, co si má proband pamatovat apod. Některé vstupní parametry však mohou být u her totožné. Může se jednat především o délku úlohy, délku jednoho kola, možnosti přeskočení atp.

Integrace her do webové aplikace BrainIn vyžaduje, aby všechny hry působily podobným dojmem a nabízely totožné možnosti (přeskočení kola, předčasné ukončení hry, nápovědu apod). Samozřejmě záleží také na charakteru úlohy a požadavcích terapeutů. Obvykle ne všechny požadované funkcionality dávají u všech her vždy smysl.

5.2.2 Výstupní hodnoty

Podobně jako vstupní hodnoty, tak i výstupní hodnoty se budou měnit s ohledem na specifika úlohy. Mezi tyto výstupní hodnoty by se mohli řadit např. procentuální úspěšnost, doba dokončení úlohy, vygenerované herní prvky, počty správných a špatných herních kroků, informace o dokončení či přeskočení části hry apod.

Již existující hry integrované do systému BrainIn obsahují také různé parametry zapsané ve formátu *JSON*, které obsahují informace o pozicích herních prvků a jejich změnách v průběhu hry (jakýsi monitoring hráčovy manipulace s herními objekty). Ačkoliv tato informace není přímým požadavkem pana doktora Krále, bylo by vhodné tyto hodnoty v implementované kostře zachovat pro budoucí implementace, kde tyto parametry vyžadovány budou.

5.2.3 Požadavky na parametry

Během vývoje by měl být brán zřetel na příchozí požadavky ze strany pana doktora Pavla Krále. Požadavky se mohou týkat nejen vstupních a výstupních parametrů her, ale i herních průběhů. Zapracované požadavky by měly být se zadavatelem průběžně diskutovány, a to nejlépe nad vytvořenými prototypy a demonstracemi hotových částí aplikací.

5.2.4 Šablona

Všechny hry mohou mít některé vstupní a výstupní parametry společné. Níže jsou uvedeny společní zástupci vstupních parametrů:

- **Herní doba** (přirozené číslo) je čas, který má pacient na dokončení celé úlohy.
- **Počet kol** (přirozené číslo) reprezentuje počet kol v dané úloze.
- **Přeskočit kolo** (logická hodnota) udává, má-li pacient možnost přeskočit kolo.

- **Přeskočit zbylá kola** (logická hodnota) udává, zda má pacient možnost předčasně ukončit hru.
- **Trvání přechodu mezi koly** (kladné desetinné číslo) udává v sekundách dobu trvání přechodu mezi jednotlivými koly.
- **Doba zobrazování hodnocení** (přirozené číslo) definuje dobu, po kterou bude hráči zobrazována procentuální úspěšnost z odehrané hry.
- **Seed** (textový řetězec) reprezentuje počáteční hodnotu, která se používá pro generování náhodných průběhů jednotlivých kol.
- **Debug** (logická hodnota) zobrazuje a skrývá pomocné výpisy pro ladění programů.

Níže jsou uvedeny společné výsledky výstupních parametrů:

- **Celkový čas** (přirozené číslo) informuje o celé délce hry (včetně čtení nápověd, trvání přechodů mezi koly atd.).
- **Úspěšnost** (přirozené číslo) udává výsledek pacientova snažení v procentech.
- **Datum a čas spuštění** (textový řetězec) informuje o okamžiku spuštění úlohy.
- **Lokalizace** (textový řetězec) nabývá hodnot *cs*, *en* nebo *de*.
- **Celkový herní čas** (přirozené číslo) udává v sekundách délku plnění cílů hry tj. celková doba bez všech pauz, přechodů mezi koly apod.
- **Přeskočena zbylá kola** (logická hodnota) udává, zda došlo k předčasnému ukončení úlohy.
- **Seed** (textový řetězec) reprezentuje hodnotu s jakou byly vygenerovány obsahy všech kol v úloze.
- **Nastavený počet kol** (přirozené číslo) udává, kolik kol bylo původně nastaveno k odehrání.

Níže jsou uvedeny společné dílčí výsledky výstupních parametrů:

- **Čas kola** (přirozené číslo) informuje o délce celého kola (včetně čtení nápověd, trvání přechodů mezi koly atd.).
- **Herní čas kola** (přirozené číslo) udává v sekundách délku plnění cílů hry v daném kole.

- **Dokončeno** (logická hodnota) informuje o tom, jestli bylo dané kolo dokončeno (tj. nebylo přeskočeno, předčasně ukončeno apod.).
- **Správně vyřešeno** (logická hodnota) udává, bylo-li kolo po jejím kompletním dohrání i správně vyřešeno.
- **Kolo přeskočeno** (logická hodnota) udává, jestliže došlo k přeskočení daného kola.

5.3 Hry

Vedoucí Katedry klinické psychologie IPVZ[3] PhDr. Pavel Král, Ph.D. vymyslel 3 hry, které by měly sloužit k léčbě a trénování pozornosti a krátkodobé paměti.

5.3.1 Signals

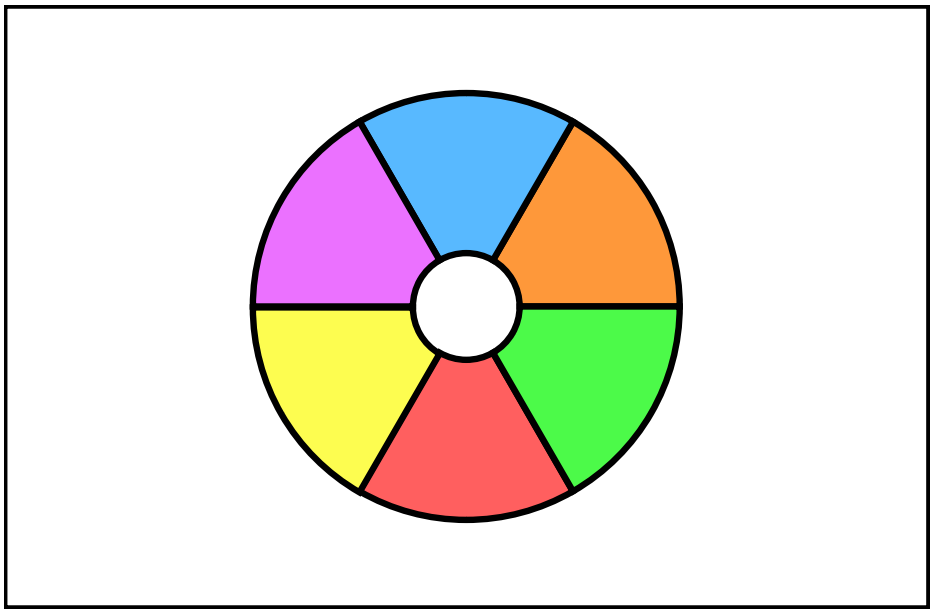
Tato hra je zaměřená především na trénování pozornosti a krátkodobé paměti. Princip hry spočívá v tom, že je pacientovi zobrazena sekvence barev, kterou se následně snaží zopakovat. Zobrazované barvy budou reprezentovány barevnými tlačítky ve tvaru kruhové výseče (viz obr. 5.2), které dohromady tvoří celý kruh. Zvýraznění tlačítka by mělo být snadno postřehnutelné a výrazné, aby proband stihl zaznamenat kompletní sekvenci zvýrazňovaných tlačítek.

Vstupní hodnoty

Níže jsou uvedeny vstupní hodnoty této hry:

- **Počet rozsvícení** (formátovaný textový řetězec) hodnoty jsou odděleny pro jednotlivá kola čárkou a udávají počet barevných tlačítek, které se v průběhu kola zvýrazní.
- **Délka rozsvícení** (formátovaný textový řetězec) hodnoty jsou odděleny pro jednotlivá kola čárkou a v sekundách udávají trvání zvýraznění jednoho barevného tlačítka.
- **Délka mezi zvýrazněními** (formátovaný textový řetězec) hodnoty jsou odděleny pro jednotlivá kola čárkou a v sekundách udávají dobu mezi jednotlivými zvýrazněními.

[3]Institut postgraduálního vzdělávání ve zdravotnictví



Obrázek 5.2: Návrh herní scény hry Signals

Výstupní hodnoty

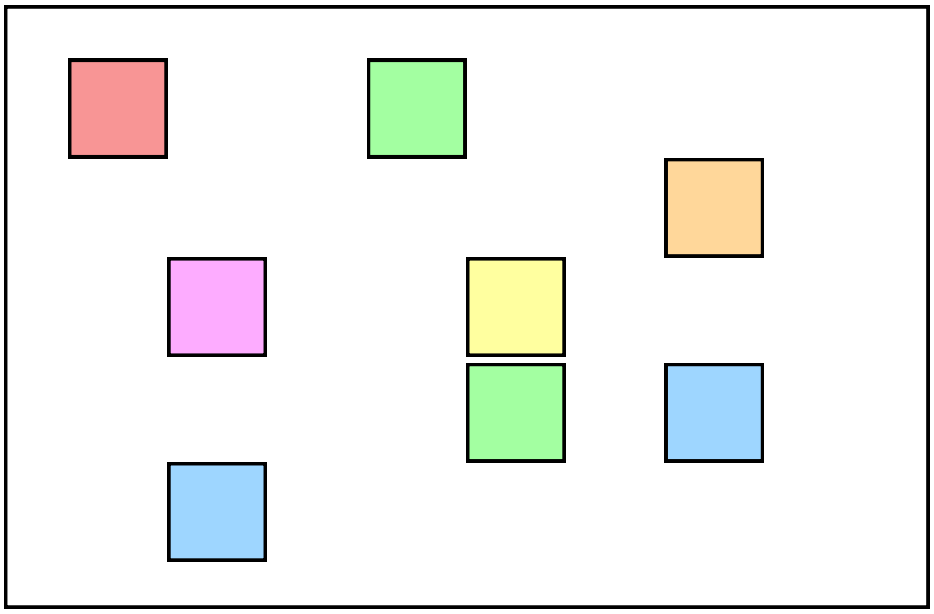
Níže jsou uvedeny dílčí výsledky výstupních hodnot této hry:

- **Úspěšnost zopakování sekvence** (desetinné číslo) udává procentuální úspěšnost zopakování sekvence v daném kole.
- **Sekvence k opakování** (*JSON*) reprezentuje seznam tlačítek tak, jak byla zobrazena pacientovi k zopakování.
- **Sekvence stisknutí** (*JSON*) reprezentuje seznam tlačítek tak, jak byla pacientem skutečně stisknuta.

5.3.2 GlowingSquares

Účel této hry je totožný jako u té předchozí, neboť pomáhá léčit a trénovat stejné neurologické obtíže.

V průběhu kola se na herní scéně postupně zvýrazňují barevné čtverce (viz obr. 5.3). Tuto sekvenci by si měl daný hráč zapamatovat a poté zopakovat. Na scéně může být několik čtverců a zobrazované sekvence se v jednom kole prodlužují v případě, že předchozí sekvenci dokázal hráč zopakovat správně.



Obrázek 5.3: Návrh herní scény hry GlowingSquares

Vstupní hodnoty

Níže jsou uvedeny vstupní hodnoty této hry:

- **Počet tlačítek** (formátovaný textový řetězec) hodnoty jsou odděleny pro jednotlivá kola čárkou a udávají počet zobrazených tlačítek (čtverců).
- **Délka rozsvícení** (formátovaný textový řetězec) hodnoty jsou odděleny pro jednotlivá kola čárkou a v sekundách udávají trvání zvýraznění jednoho barevného tlačítka.
- **Délka mezi zvýrazněními** (formátovaný textový řetězec) hodnoty jsou odděleny pro jednotlivá kola čárkou a v sekundách udávají dobu mezi jednotlivými zvýrazněními.
- **Výchozí počet zvýrazněných čtverců** (formátovaný textový řetězec) hodnoty jsou odděleny pro jednotlivá kola čárkou a udávají délku první zvýrazněné sekvence. V případě úspěšného zopakování bude hodnota v daném kole inkrementována o 1.
- **Barevné čtverce** (logická hodnota) rozhoduje o tom, zda budou na scéně čtverce barevné nebo ne (v tom případě budou šedé).

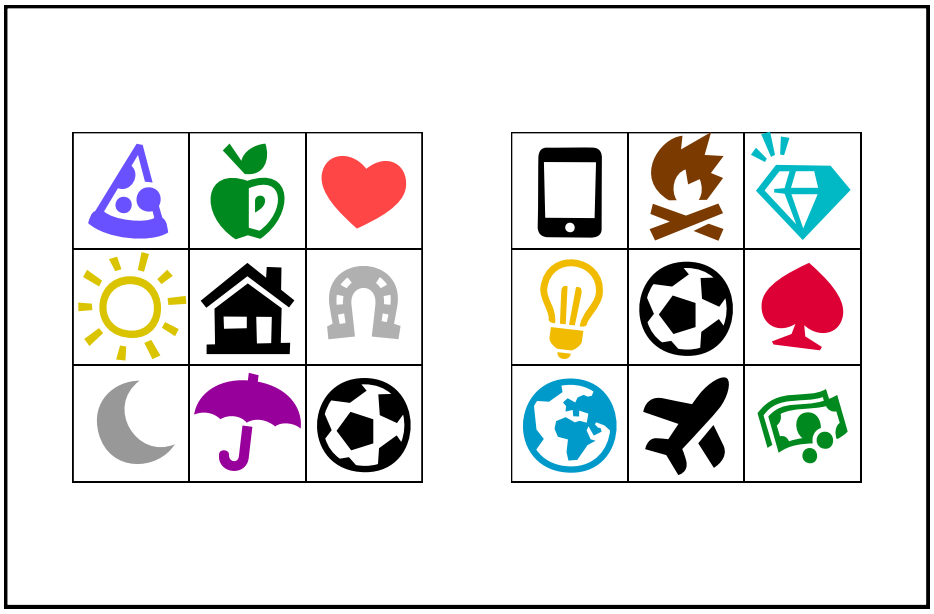
Výstupní hodnoty

Níže jsou uvedeny dílčí výsledky výstupních hodnot této hry:

- **Úspěšnost zopakování sekvence** (desetinné číslo) udává procentuální úspěšnost zopakování sekvence v daném kole.
- **Sekvence tlačítek** (*JSON*) reprezentuje seznamy tlačítek v pořadí, ve kterém byla pacientovi zobrazena k zapamatování a v pořadí, ve kterém byla pacientem skutečně stisknuta.

5.3.3 Match'em

Hra *Match'em* je zaměřena především na trénink pozornosti. Na herní scéně budou zobrazeny dvě 3x3 mřížky (viz obr. 5.4). Každé okénko této mřížky obsahuje jeden tvar, obrázek, kombinaci několika alfanumerických znaků apod. V každé mřížce je však jedno okénko, jehož obsah je totožný s obsahem jednoho okénka v mřížce druhé. Cílem hráče je najít shodný tvar, znak nebo obrázek a označit jej kliknutím na něj.



Obrázek 5.4: Návrh herní scény hry Match'em (zdroj obrázků: [6])

Vstupní hodnoty

Níže jsou uvedeny vstupní hodnoty této hry:

- **Počet mřížek** (přirozené číslo) je analogií pro počet kol. Udává, kolikrát se bude hráč snažit najít stejné části ve dvou mřížkách.
- **Varianta s obrázky** (logická hodnota) udává, má-li být nějaká z mřížek vytvořena z obrázků.
- **Varianta s písmeny** (logická hodnota) udává, může-li být některá z mřížek vytvořena z písmen.
- **Varianta s čísly** (logická hodnota) udává, jestli může být některá z mřížek vytvořena z číslic.
- **Vzhůru nohama** (logická hodnota) vyjadřuje informaci o tom, zda může být jedna z mřížek otočena vzhůru nohama.
- **Počet znaků** (přirozené číslo) reprezentuje počet znaků v jednom okénku v případě, že se vytváří mřížka z písmen nebo číslic.
- **Chybné volby** (přirozené číslo) určují, kolik chybných určení může hráč provést před tím, než bude kolo označeno jako chybné.
- **Obrázky** (seznam souborů) obsahují všechny obrázky, z kterých je možné při vytváření obsahu kola vybírat a poté vkládat do jednotlivých okének herních mřížek.

Výstupní hodnoty

Níže jsou uvedeny dílčí výsledky výstupních hodnot této hry:

- **Vygenerované mřížky** (*JSON*) informuje o vzhledu vygenerovaných herních mřížek.
- **Sekvence stisknutých tlačítek** (*JSON*) je seznam stisknutí na jednotlivá okénka v herních mřížkách.
- **Chybné volby** (přirozené číslo) sdělují, kolikrát se hráč při hledání zdvojeného obsahu okénka v herních mřížkách spletl.

6 Implementace

V následujících sekcích je podrobně popsána implementace navržené kostry a her, jež právě z této kostry vychází. Implementované řešení bylo realizováno v herním frameworku Unity ve verzi 2022.1.23f1. Společně s implementací her byla přepracována také komunikace s webovou aplikací, včetně úprav přímo v jejím projektu. Dále byly přepracovány všechny skripty, jež správné spuštění hry ve webovém prostředí řídí.

6.1 PoC

Na začátku vývoje byl realizován proof of concept (PoC) některých částí návrhu. Součástí PoC byla nejen jednoduchá aplikace v Unity (2022.1.23f1), jež obsahovala některé z navržených architektonických prvků v sekci 5.1, ale také implementace potřebných změn na straně webové aplikace. Tento prototyp měl ověřit proveditelnost navržených řešení. Cíle PoC byly následující:

1. ověřit, že implementace návrhového vzoru Event Aggregator bude využitelná pro komunikaci mezi backendovými a frontendovými třídami,
2. prověřit, zda návrhový vzor Service-Locator bude vhodným nástrojem pro sdílení závislostí,
3. zkontrolovat a otestovat, že sestavená aplikace v této verzi Unity bude spustitelná přes uživatelské rozhraní webové aplikace,
4. ověřit, že webová aplikace bude schopna odesílat vstupní hodnoty ve formátu *JSON* a bude schopna ve stejném formátu přijímat výstupní parametry.

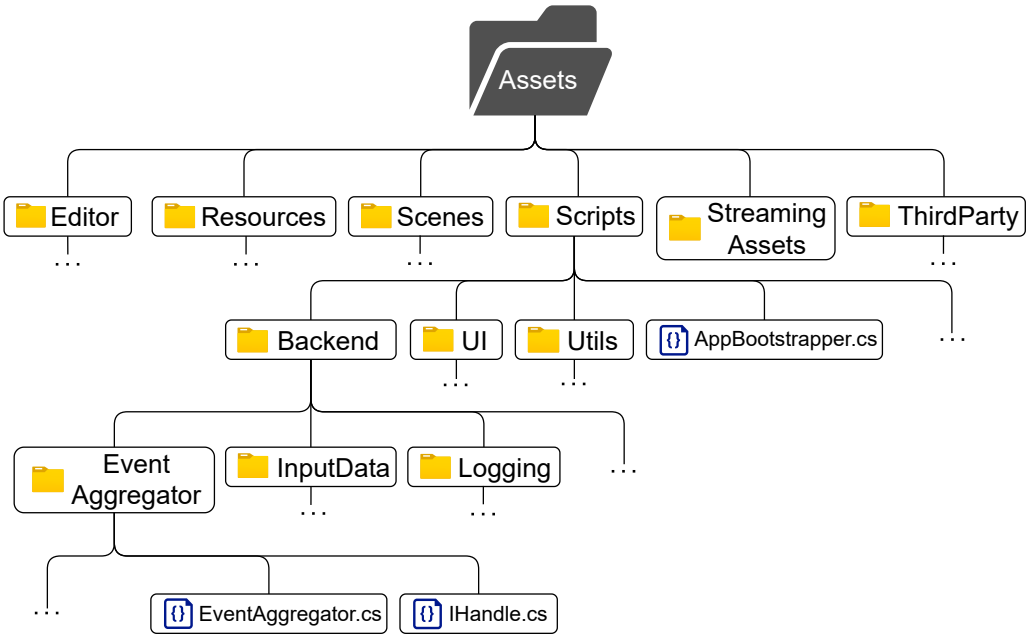
Tento přístup (splňující filozofii „fail fast“) minimalizoval rizika spojená s návrhem a budoucí implementací. PoC potvrdil, že navržené postupy budou vhodné pro vytvoření kvalitního softwarového produktu.

6.2 Šablona

Tato sekce je věnována vysvětlení nejdůležitějších informací o znovupoužitelné kostře pro její pochopení při vývoji nových her pro systém BrainIn. Následující sekce popisují konkrétní implementace jednotlivých částí.

6.2.1 Struktura projektu

Struktura projektu je organizována ve stylu „package by feature“. Tedy každý balíček (adresář) obsahuje všechny třídy, které se přímo vztahují k určité funkcionalitě (viz obr. 6.1).



Obrázek 6.1: Nástin organizace adresářové struktury projektu

Editor

Adresář *Editor* obsahuje všechny zdrojové kódy a soubory, které se využívají při běhu editoru (vývojové prostředí Unity). Jedná se především o testy, uměle vytvořená data používaná pro správný průběh jednotkových testů nebo soubory nástroje `DeployHelper` pro jednodušší sestavení a nasazení aplikace (viz sekce 6.6.1).

Resources

Adresář *Resources* slouží k ukládání souborů, jež jsou používány za běhu aplikace. V tomto adresáři mohou být uloženy různé typy souborů, jako jsou obrázky, zvukové stopy, fonty, animace a další soubory potřebné pro správnou funkčnost aplikace. Obrázky uložené v tomto adresáři mohou být využity pro vizuální součásti aplikace, jako například pozadí obrazovky, ikony, tlačítka a další prvky uživatelského rozhraní.

Scenes

Všechny scény, vytvořené v rámci Unity projektů, jsou obvykle uloženy v adresáři *Scenes*. Všechny projekty implementovaných her (včetně kostry) obsahují pouze jednu scénu, a to tu hlavní.

Scripts

Všechny zdrojové kódy jsou uloženy v adresáři *Scripts*. Ten je rozdělen do tří částí – *Backend*, *UI*, *Utils*. Na obrázku 6.1 je na obsahu adresáře *Backend* znázorněno členění projektu dle „package by feature“. V něm se nachází několik dalších balíčků, kde každý reprezentuje právě jednu funkcionalitu. Třídy a soubory, na sobě těsně závislé, se tedy vždy vyskytují pohromadě na jednom místě.

Streaming Assets

Tento adresář slouží pro ukládání dat, jež jsou nezávislá na sestavení aplikace. Při nasazování aplikace se adresář umístí přímo do souborového systému webové aplikace. Lokalizační soubory a další podobné soubory se tak mohou měnit bez nutnosti opětovného sestavení programu. Například chybu v přeloženém textu je možné snadno opravit jen úpravou lokalizačního souboru.

ThirdParty

ThirdParty představuje úložiště pro všechny potřebné knihovny a balíčky, u nichž je nutné, aby byly součástí zdrojových souborů (v tomto případě např. *Newtonsoft JSON*).

6.2.2 Namespace

V projektu byl vytvořen jeden hlavní výchozí jmenný prostor `BrainIn`, k němuž se přidávají další klíčová slova a vytvářejí se nové jmenné prostory. Tento přístup vylepšuje logickou organizaci kódu. Všechny ostatní jmenné prostory v projektu jsou definovány na základě výše zmíněné struktury. Tedy například skript `AudioPlayer` v jmenném prostoru `BrainIn.UI.Audio` je umístěn v adresáři Scripts/UI/Audio.

6.2.3 Herní scéna

Většina herních prvků v implementovaných hrách je vytvořena staticky. Dynamicky se poté mohou vytvářet herní objekty, které jsou spojené s určitou hrou. Scéna obsahuje 3 základní herní objekty – `Main Camera`, `Main Canvas` a `Services`.

`Main Camera` je objekt, který se stará o zachycení herního světa a zobrazení jej uživateli. Jedná se o velmi jednoduchou kameru zobrazující 2D svět bez komplexnějšího ovládání světel, stínů apod.

`Main Canvas` reprezentuje hlavní část herní scény, ve které se vytváří všechny grafické a interaktivní prvky hry. Obsahuje různé pohledy (views):

- `Background` reprezentuje, jak již z názvu plyne, barevné pozadí.
- `Game View` je část scény, jež je zobrazena ve chvíli spuštění určitého kola, kde by uživatel měl začít plnit cíle hry. Tento herní objekt v sobě obsahuje např. prvky vztahující se k určité hře nebo třeba speciální zobrazení výpisů při ladění.
- `PreRound View` je zobrazení fáze mezi jednotlivými koly.
- `Pause View` má na starosti zastavení hry a času (např. při snaze přeskočit kolo).
- `Tooltip View` představuje zobrazení textových nápověd po najetí kurzoru na tlačítka v uživatelském rozhraní.

Objekt `Services` sdružuje všechny ostatní části herní scény, které poskytují určité služby. Patří mezi ně objekt spravující přehrávání zvukových stop, systém zaznamenávání pohybu kurzoru po herní scéně nebo např. přijímač zpráv z webové aplikace. Všechny tyto prvky ve scéně jsou vždy při startu aplikace neaktivní s výjimkou `UIBootstrapper`, jehož úkolem je zajistit korektní spuštění hry.

6.2.4 Spuštění hry

V sekci 6.2.3 stojí, že o správné spuštění hry se stará jediný aktivní herní objekt `UIBootstrapper`. Jelikož je jediným aktivním objektem ve scéně, je na začátku hry spuštěn pouze jeho obslužný kód. Tento skript poté předává řízení backendové třídě `AppBootstrapper` invokací její statické metody `Initialize`.

`AppBootstrapper` jakožto jediný backendový inicializátor se pokusí o korektní nastavení jedináčka třídy `ServiceLocator` (pouze o inicializaci požádá, samotný akt má na starosti třída sama) a naplnění jej vytvořenými

backendovými manažery a dalšími třídami, jež jsou ve všech částech aplikace hojně využívány.

Po úspěšné inicializaci backendu provede `UIBootstrapper` také nastavení frontendové části aplikace, a to aktivací všech skriptů umístěných v herním objektu `Services`. Mezi ně patří skript `MainController` starající se následně o průběh a správné zobrazování jednotlivých částí hry (více v sekci 6.2.5).

6.2.5 Herní průběh

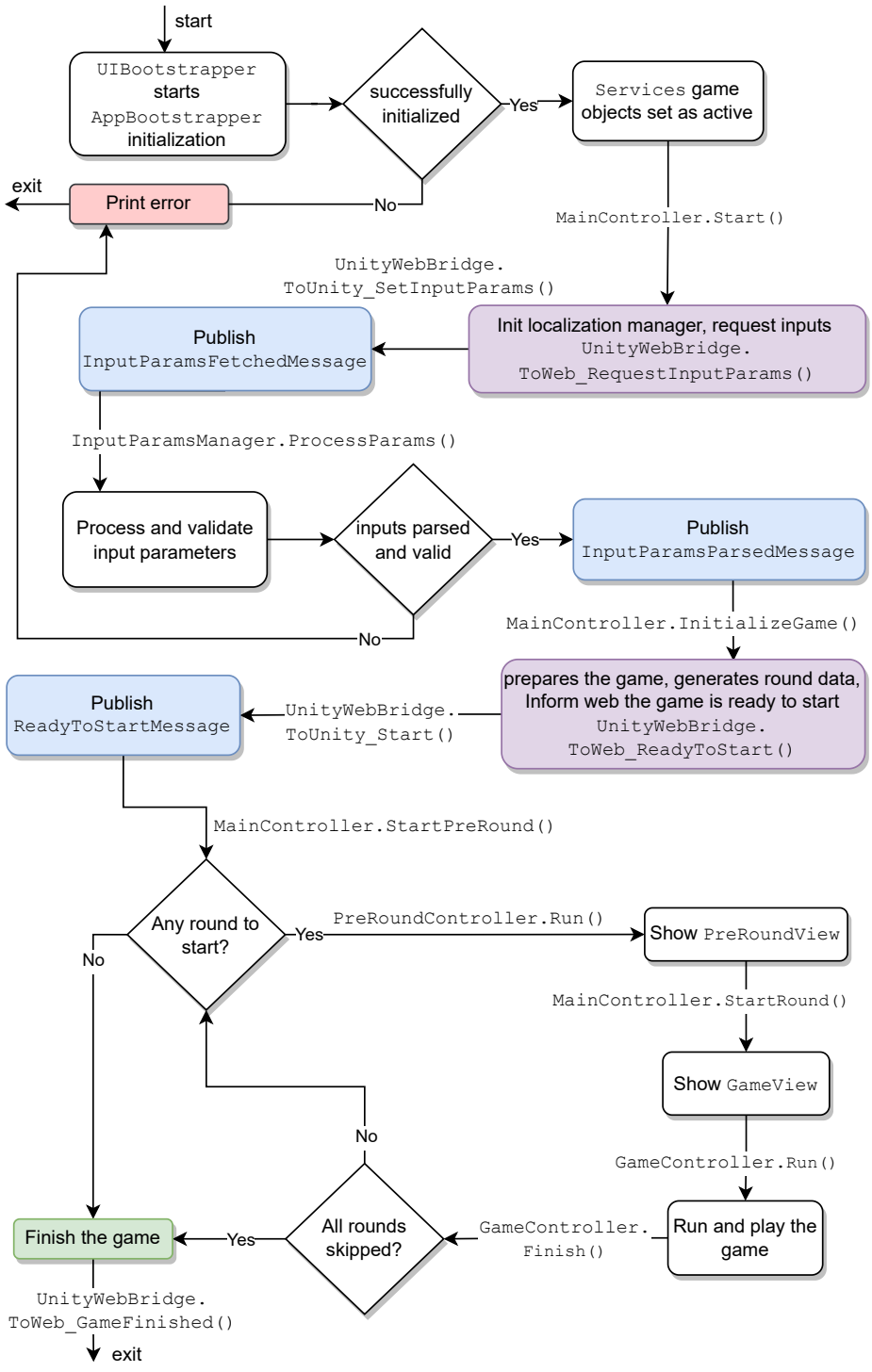
Diagram 6.2 popisuje (mírně zjednodušený) průběh aplikace od spuštění až po její ukončení. Po spuštění programu a úspěšné inicializaci je aktivován `MainController`, který vykonává další důležité kroky a řídí průběh celé aplikace.

Nejprve je načten lokalizační soubor a inicializován lokalizační manažer. Na základě konkrétní podoby absolutní cesty k souboru je soubor buď stahován z internetu, nebo načítán ze souborového systému. Rozšiřující metoda (extension method) datového typu string `IsLocalPath` určuje, zda-li textový řetězec obsahuje lokální cestu (v případě spuštění aplikace v Unity editoru), nebo internetový odkaz.

Po inicializaci backendové třídy pro správu lokalizací textů je Unity hra připravena přijmout vstupní parametry dané úlohy. Hra webovou aplikaci o vstupní parametry požádá zasláním příslušné zprávy (zavoláním JavaScriptové metody `setInputParams`). Komunikaci s webovou aplikací zajišťuje implementace rozhraní `IUnityWebBridge`. Příjem zpráv z webové aplikace probíhá na frontendu (vlastnost Unity), takže po přijetí textového řetězce se vstupními parametry je backend informován publikováním zprávy `InputParamsFetchedMessage`.

Na výše zmíněnou zprávu reaguje třída `InputParamsManager` starající se o zpracování vstupních parametrů a jejich validaci. Součástí vstupních parametrů je také jazyk zvolený na webu při spuštění úlohy. Je tedy možné rovnou nastavit příslušný jazyk i pro Unity aplikaci (`I18nManager.SetLang`). Jazyk je nutné nastavit ještě před validací vstupních parametrů, neboť jsou přeložené i chybové zprávy napomáhající opravit formát nebo obsah vstupních hodnot. Po těchto krocích je nutné opět upozornit hlavní controller, a to publikováním zprávy `InputParamsParsedMessage`.

`MainController` přijme deserializované vstupní parametry a spustí inicializaci hry. Inicializační metoda ukládá některé informace do výstupních parametrů (např. datum a čas spuštění nebo počet nastavených kol) a pro každé kolo generuje herní data. Následně je informována webová aplikace, že je Unity hra připravena k spuštění. V tuto chvíli se totiž na webu v di-



Obrázek 6.2: Mírně zjednodušený diagram herního průběhu. Fialové akce reprezentují předání řízení webové aplikaci a čekání na její odpověď. Modré akce představují způsob komunikace přes EventAggregator. Červené stavy představují chybu a zelené korektní ukončení hry

alogovém okně uživateli zobrazí přeložené instrukce k dané hře. Po skrytí dialogového okna je řízení předáno zpět Unity aplikaci, jež konečně spustí hru.

Skript `MainController` využívá stavovou proměnnou definovanou hodnotami třídy výčtového typu `GameProgressState`. Metoda `Update` na základě hodnoty výše zmíněné proměnné (v aktuální chvíli popisu průběhu je nastavena na hodnotu `PreRound`) aktivuje třídu `PreRoundController`, jež zobrazí přechodovou fázi mezi jednotlivými koly.

Ihned poté je spuštěno dané kolo invokací metody `StartRound` a aktivací skriptu `GameController` (zavoláním jemu náležející metody `Run`). Po odehrání kola (nebo jeho přeskočení) je řízení navráceno zpět hlavnímu controlleru, který ověří, nedošlo-li k předčasnému ukončení hry. Pokud ano, je hra ukončena, pokud ne, je spuštěno další kolo (je-li k dispozici, v opačném případě je hra ukončena).

6.3 Backend

V následujících sekcích je uveden popis nejdůležitějších částí backendu aplikace.

6.3.1 Vkládání závislostí

Na základě návrhu v sekci 5.1.4 byl návrhový vzor ServiceLocator realizován do stejnojmenné třídy (jmenný prostor `BrainIn.Backend.IoC`). Třída disponuje několika veřejnými statickými metodami a několika soukromými metodami. k vytvoření instance jedináčka dochází v metodě `Init`. Ostatní veřejné statické metody (`Register`, `Get`, `Unregister`, `UnregisterAll`) usnadňují komunikaci s vytvořenou instancí, obvykle jen volají soukromé instanční metody. Jelikož je možné, že by frontendové třídy mohli přistupovat k zaregistrovaným instancím současně, byl ServiceLocator implementován jako thread-safe. Pomocí konstrukce `lock` je zajištěn výhradní přístup k slovníku všech registrovaných instancí. Při každé manipulaci (přidávání, odebírání apod.) se tedy slovník pro ostatní uzamkne.

Backendové třídy si obvykle předávají závislosti v konstruktorech, protože většina těchto tříd je instanciována společně v `AppBootstrapper` (viz zdrojový kód 6.1). Veřejné metody pro získání závislostí tedy využívají především frontedové třídy. Ukázka 6.1 demonstruje předávání závislosti v konstruktoru třídy, registraci vytvořené instance a její následné získání.

Zdrojový kód 6.1: Ukázka předávání závislostí

```
var eventAggregator = new EventAggregator();
// UnityWebBridge gets the reference to EventAggregator in ctor.
var bridge = new UnityWebBridge(eventAggregator);

// Register the implementation as its interface.
ServiceLocator.Register<IUnityWebBridge>(bridge);
// Get previously registered IUnityWebBridge implementation.
var bridgeReference = ServiceLocator.Get<IUnityWebBridge>();
```

6.3.2 Event Aggregator

Pro jednodušší komunikaci mezi backendovými a frontendovými třídami byl v rámci jmenného prostoru `BrainIn.Backend.EventAggregator` (dle návrhu v sekci 5.1.7) implementován návrhový vzor EventAggregator. Celkem ho tvoří 4 soubory – `IEventAggregator`, `IHandle`, `EventAggregator` a `EventAggregatorObserver`.

Účelem třídy `EventAggregator`, implementující `IEventAggregator`, je nejen správa všech pozorovatelů, ale také publikování zpráv. S využitím funkce `Publish` lze odeslat zprávu všem přihlášeným třídám, které zprávu sledují. Přihlásit nového pozorovatele je možné metodou `Subscribe` (případně odhlásit pomocí `Unsubscribe`). Tato metoda vlastně vytvoří novou instanci třídy `EventAggregatorObserver` a uloží ji do seznamu všech pozorovatelů.

Instance třídy `EventAggregatorObserver` reprezentuje přihlášeného pozorovatele. Konkrétní pozorovatel se může přihlásit ke sledování libovolného množství zpráv, a to implementací rozhraní `IHandle<T>`. Generický typ je samozřejmě při reálném použití nahrazen typem zprávy, na kterou pozorovatel reaguje. Reagovat na zaslanou zprávu je možné implementací jediné metody tohoto rozhraní `Handle(T value)`. Při vytváření nového pozorovatele se v konstruktoru třídy `EventAggregatorObserver` (s pomocí reflexe) získají všechna pozorovatelem definovaná rozhraní `IHandle` a existující metody `Handle` jsou uloženy do slovníku pro budoucí použití (klíčem je typ zprávy). `EventAggregator` při publikování zprávy projde všechny své registrované pozorovatele a pokud má nějaký pozorovatel ve svém slovníku uloženou metodu s klíčem dané zprávy, provolá ji.

6.3.3 Logování

Pro jednotný přístup k chybovým (a ostatním) výpisům bylo vytvořeno rozhraní `IBraininLogger` s abstraktními metodami – `Info`, `Warn` a `Error`. Třída

`SimpleLogger`, jež je implementací výše zmíněného rozhraní, využívá pro tisk do konzole metody z třídy `UnityEngine.Debug`. Každý obsah výpisu je upraven do uceleného formátu ve tvaru:

```
yyyy/MM/dd HH:mm:ss - [namespace.class] -> content of the log
```

Při vytváření instance loggeru se do konstruktoru vkládá typ třídy nebo konkrétní objekt klíčovým slovem `this` (viz ukázka zdrojového kódu 6.2).

Zdrojový kód 6.2: Ukázka vytváření a používání loggeru

```
1 IBraininLogger Log = new SimpleLogger(typeof(ServiceLocator));
2 Log.Error("Some error!");
```

Tento přístup umožňuje libovolnou záměnu implementací logovacího nástroje. Je tedy možné kdykoliv připravit logger, jenž zaznamenává chybové výpisy do souborů nebo je přímo odesílá do nějakého nástroje pro jejich správu a monitoring (např. *Backtrace*).

6.3.4 Lokalizace

Pro úspěšnou integraci her do webové aplikace BrainIn je nutné umožnit jejich spuštění v různých jazycích. Texty v nápovědě k ovládání, popisky tlačítek apod. byly přeloženy do angličtiny a němčiny.

Na základě návrhu v sekci 5.1.5 byl nejprve použit balíček *Localization* od Unity. Bohužel nebylo v tomto případě použití balíčku příliš vhodné. Hlavní problém byl v nastavování konkrétního jazyka za běhu aplikace. Hra totiž nejprve přijme vstupní parametry společně s jazykem nastaveným ve webové aplikaci. Až poté je možné tento jazyk zvolit také pro Unity aplikaci. Lokalizační systém však není ihned připraven, je nutné po startu aplikace (nebo po změně jazyka) počkat na dokončení tzv. `InitializationOperation`. Z charakteru existující implementace je nutné počkat na dokončení této operace na backendu pomocí `WaitForCompletion` nebo `InitializeSynchronously`. To však není dle oficiální dokumentace u WebGL projektů možné. [28]

Proto byl implementován zcela nový lokalizační systém (jmenný prostor `BrainIn.Backend.Localization`) bez použití dalších balíčků či knihoven. Lokalizační soubor (ve formátu *XML*) se po spuštění aplikace stahuje ze souborového systému webové aplikace. Obsah souboru je hned poté nastaven do privátního atributu třídy `I18nManager` jako textový řetězec. Po zjištění jazyka je možné sestavit novou tabulku všech přeložených textů. Do tabulky překladů se ukládá záznam s názvem lokalizace (klíč) a konkrétním přeloženým textem v daném jazyce (hodnota). Vytváření slovníku má na starosti metoda `SetLang`. Metoda projde všechny konstanty definované

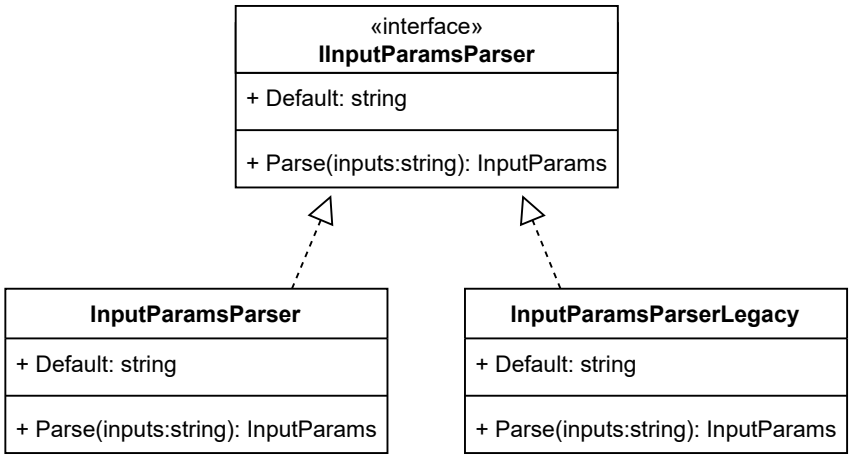
v třídě `I18n`, jež slouží jako databáze všech existujících překladů (zároveň k nim poskytuje jednodušší a intuitivní přístup), najde k nim přeložený text v *XML* souboru (využívajíc třídy z jmenného prostoru `System.Xml.Linq`) a uloží ho do slovníku. Získat přeložený text je pak možné pomocí metody `Get` (viz ukázka 6.3).

Zdrojový kód 6.3: Ukázka nastavení jazyka a získání přeloženého textu

```
1 i18nManager.SetLang("cs");
2 var helpText = i18nManager.Get(I18n.Help)
```

6.3.5 Vstupní hodnoty

Zpracování vstupních parametrů mají na starosti třídy z jmenného prostoru `BrainIn.Backend.InputData`. Proces zpracování začíná v backendovém manažeru `InputParamsManager` v metodě `ProcessParams`, jež přijme na vstupu textový řetězec se vstupními hodnotami (ať už v novém, nebo v starém formátu). Při inicializaci backendu, v třídě `AppBootstrapper`, se vytváří instance implementací rozhraní `IInputParamsParser` (viz obr. 6.3) a `IInputParamsValidator`. Pro rozdělení parametrů a jejich následnou validaci je tedy možné použít libovolnou implementaci, což mimo jiné zajišťuje i zpětnou kompatibilitu (viz `InputParamsParserLegacy`).



Obrázek 6.3: Diagram rozhraní `IInputParamsParser`

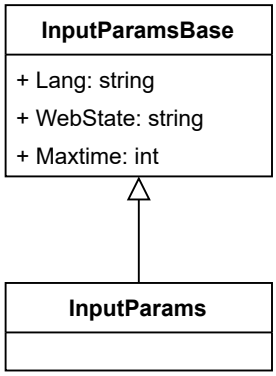
Manažer předá v metodě `TryParseParams` řízení svému „parseru“ a počká na dokončení rozdělení vstupních dat do datové třídy `InputParams`. U nového formátu vstupních dat (předávaných ve formátu *JSON*) je celé rozdělení obsaženo v deserializaci předaného textového řetězce do výše zmíněné třídy. Občas je nutné ke konkrétním parametrům přidat speciální konvertor.

Webová aplikace zasílá ve vstupních hodnotách datový typ `bool` jako textový řetězec s hodnotou „1“ nebo „0“. Z tohoto důvodu se musí napsat speciální převodník pro převod textové hodnoty „1“ na `true` apod. Všechny převodníky jsou vytvářeny v jmenném prostoru `BrainIn.Backend.Utils.Json`.

U starého formátu je implementace náročná, náchylná k chybám a matoucí (cca 50 řádků zdrojového kódu oproti 3 u nového formátu). V případě spuštění aplikace v Unity Editoru jsou zpracovávány předem definované výchozí parametry `IInputParamsParser.Default`.

Po úspěšném rozdělení vstupních parametrů je spuštěna kontrola správnosti vstupních dat (např. je-li počet kol skutečně přirozené číslo). Při jakékoliv chybě dojde k vyhození výjimky `InputParamsException` a ukončení aplikace. Autor chybné konfigurace úlohy je o této skutečnosti informován společně s označením špatného parametru a příslušným chybovým hlášením. Ve vytvořeném validátoru je možné každý parametr ověřit pomocí metody `Validate`. Metoda přijímá na vstupu konkrétní hodnotu, označení daného parametru, hodnotu vůči které bude validace provedena, speciální validační funkci (byla připravena série několika takových funkcí – `GT, LTE, EQ` apod.) a vzor chybového hlášení, jenž bude v případě chyby doplněn o další informace.

Datová třída vstupních parametrů `InputParams` dědí od svého předka `InputParamsBase` (viz obr. 6.4). V „base“ třídě se totiž definují společné parametry pro všechny hry (jazyk, počet kol apod.). Parametry specifické pro konkrétní hru jsou definovány v třídě podřazené. Tím se vymezují místa, která není při nových implementacích třeba upravovat a celkově se tak zrychluje a usnadňuje nový vývoj.



Obrázek 6.4: Diagram dědičnosti `InputParams`

6.3.6 Měření času

Jmenný prostor `BrainIn.Backend.TimeManagement` obsahuje několik tříd, jejichž úkolem je měření času v průběhu hry. Různé hry mohou měřit různé časy, ale původní kostra měří časy dva – herní a celkový. Měření celkového času je spuštěno ihned při inicializaci třídy `TimeManager`. Měření herního času se spouští při začátku kola v metodě `Start`. Metoda `Pause` jej zastavuje při konci kola nebo v průběhu dialogů pro přeskočení kola atp.

Jeden čas je reprezentován instancí třídy `Time`. Metoda `Start` jej inicializuje zaznamenáním aktuálního data a času (`DateTime.Now`). V metodě `Stop` je uplynulá doba uložena do atributu `elapsedTime`. Uplynulá doba se měří rozdílem aktuálního času a uložené hodnoty při spuštění v instančním atributu `startTime`.

6.3.7 Generování herních dat

V metodě `InitializeGame` v třídě `MainController` se po rozdělení a validaci vstupních parametrů spustí i generování dat do každého kola. Po úspěšném vygenerování herních dat je o této skutečnosti informována webová aplikace (viz sekce 6.2.5).

Generování těchto dat má na starosti třída `RoundDataGenerator` (implementace rozhraní `IRoundDataGenerator`) nacházející se v jmenném prostoru `BrainIn.Backend.Game.Generator`. Instance této třídy disponuje jednou veřejnou metodou `Generate`, jež přijímá na vstupu datovou třídu vstupních parametrů a vrací seznam instancí třídy `RoundData`. Logika generování dat je pro každou hru jiná, takže ve vytvořené kostře her není generátor implementován.

Seed

Herní data jsou generována pomocí generátoru pseudonáhodných čísel. Jejich výsledná podoba tedy závisí na konkrétním nastaveném seedu. Každý terapeut by měl mít možnost spustit úlohu se stejným seedem a vytvořit tak totožný průběh hry.

Ve většině případů však tvůrce konfigurace úlohy zadávat seed nebude. Ten je obvykle vytvořen z aktuálního systémového času. Bohužel není poté jednoduché získat hodnotu, se kterou byl generátor pseudonáhodných čísel inicializován. Seed by nemohl být odeslán společně s výsledky a terapeut by přišel o možnost zopakovat totožný průběh hry.

Řešením je použití předaného nebo vygenerovaného seedu. Pokud autor konfigurace úlohy specifikuje nějaký svůj seed, bude použit k inicializaci

generátoru. V případě, že ho nespecifikuje (v tu chvíli nabývá hodnot „0“, „-“ nebo hodnoty prázdného řetězce), vytvoří algoritmus náhodný textový řetězec, jenž bude reprezentovat seed pro skutečný generátor herních dat. V každém případě je hodnota využitá pro inicializaci generátoru odeslána do webové aplikace společně se vstupními parametry.

Vstupem při inicializaci systémového generátoru pseudonáhodných čísel je hodnota datového typu `integer`. Seed je však možné zadávat jako textový řetězec. Textové hodnoty jsou pro terapeuty intuitivnější než ty číselné. Z tohoto důvodu je tedy nutné převést textovou hodnotu na číselný seed. Pro převod je nutné využít deterministický algoritmus, aby v každém případě byl textový řetězec převeden na totožnou číselnou hodnotu. Převod na číselný seed je realizován generováním hashe a jeho následnou konverzí na 32bitový integer. Vznik kolizí není úplně vyloučen, ale šance je velmi malá a v tomto konkrétním případě jsou kolize zanedbány.

6.3.8 Výstupní hodnoty

Jmenný prostor `BrainIn.Backend.OutputData` obsahuje třídy, jež se starají o sběr výstupních hodnot. Hodnoty z celé hry jsou sbírány do datové třídy `GameResultsData` a hodnoty z jednotlivých kol do `RoundResultsData`. Obě tyto třídy mají svého „base“ rodiče definujícího opakující se výstupní hodnoty (podobně jako u vstupních parametrů v sekci 6.3.5).

Zaznamenávat různé hodnoty je možné voláním příslušných metod rozhraní `IResultsCollector`. Všechny metody jsou implementovány v abstraktní třídě `ResultsCollectorBase`. Logika záznamu výstupních statistik se nemění, avšak mění se způsob jejich formátování do výstupního textového řetězce, který bude odeslán do webové aplikace. Nový způsob realizace výstupních parametrů pouze sesbírá nastřádané informace o hráčově manipulaci s objekty a poté serializuje výše zmíněné datové třídy to textového řetězce ve formátu *JSON*. Starší „legacy“ způsob obtížně kombinuje jednotlivé atributy datových tříd do textového řetězce (oddělováním jich různými kombinacemi speciálních znaků).

Již existující implementace her v systému BrainIn sbírají v průběhu her informace o manipulaci s objekty, pohyb kurzoru, stisknutí kláves apod. Tento systém byl v nové implementaci kostry zachován (ačkoliv mírně přepracován) a v kombinaci s novým systémem výstupních parametrů může hra tyto informace produkovat a odesílat, přestože je šablona na webu přímo nevyžaduje. Tento systém sběru hráčova chování je implementován v jmenném prostoru `BrainIn.Backend.OutputData.UserBehaviourMonitoring`.

6.4 Frontend

V této sekci je uveden popis implementace různých částí uživatelského rozhraní včetně skriptů a tříd ovládajících herní objekty a průběh hry.

6.4.1 Pravidla

Ve všech skriptech kontrolujících uživatelské rozhraní a frontend aplikace byla nastavena určitá pravidla:

- Metoda `Awake` typicky slouží pouze pro získání všech potřebných referencí na backendové instance, načtení různých souborů z adresáře *Resources* apod. Neměla by vykonávat žádný kód vztahující se ke konkrétní hře nebo danému prvku uživatelského rozhraní.
- V metodě `Start` jsou vykonány všechny nutné inicializační akce prvků uživatelského rozhraní.
- `Update` může vykonávat periodicky opakující se operace.
- Každý controller spravující nějaký herní objekt by měl dědit od třídy `ControllerBase`. Ten poskytuje možnosti pro skrývání a zobrazování daného objektu, metody pro čekání na dokončení „coroutine“ apod.

6.4.2 Průběh kola

Jak již bylo zmíněno výše, průběh jednotlivých kol řídí `GameController`. V metodě `Start` se registrují obsluhy všech tlačítek uživatelského rozhraní zobrazených během kola (např. tlačítko *Exit* po svém stisknutí vyvolá metodu `TryQuitGame` a uloží do výstupních dat informaci o pokusu přeskočit všechna kola). Metoda `Run` inicializuje všechny potřebné atributy (včetně spuštění měření herního času) a předá řízení metodě vytvořené v rámci konkrétní hry (v navržené kostře žádná herní logika není). Obvykle průběh kola končí v metodě `Finish`, kde je zastaveno měření herního času, jsou skryty všechny příslušné herní objekty a zavoláním tzv. „callbacku“ je předáno řízení zpět do hlavního controlleru.

Skripty objektů vytvářených her by měly být zařazovány do stejného jmenného prostoru (a tedy i adresáře) jako výše zmíněný `GameController`, tj. `BrainIn.UI.Game`.

6.4.3 Úsek mezi koly

Zobrazení před každým kolem řídí `PreRoundController`. V metodě `Update` postupně odčítá uplynulý čas od předem nastaveného odpočtu až do doby, než ho vynuluje. V té chvíli předává řízení zpět přes nastavený „callback“ `onFinishAction`. Tento mechanismus je navržen tak, že je možné jednoduše nahradit zobrazení prezentovaného obsahu. Jednotlivá zobrazení řídí implementace rozhraní `IPreRoundView`.

Rozhraní `IPreRoundView` obsahuje následující veřejné metody a vlastnosti:

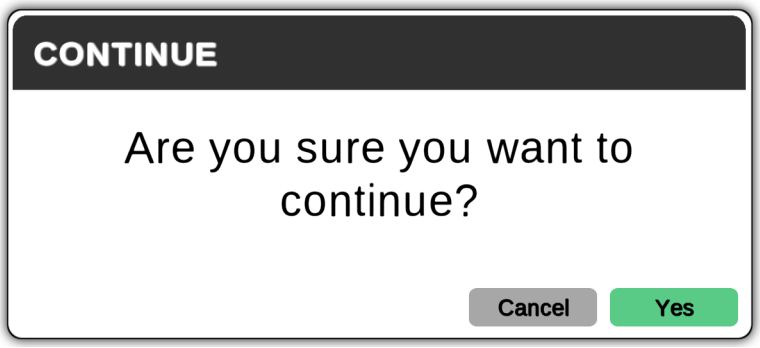
- `SoundPath` obsahuje cestu k zvukové stopě v adresáři *Resources*, jež je přehrávána při změně stavu.
- `CountdownTime` udává v sekundách délku prezentace zobrazení mezi koly.
- Metoda `Initialize` slouží pro jakékoliv nastavení zobrazení před jeho spuštěním. Přijímá požadovanou délku zobrazení, kterou může nastavit do výše zmíněné vlastnosti. Například konkrétní implementace `TrafficLightController` v této metodě také nastavuje svůj počáteční stav.
- Do metody `CountdownChanged` vstupuje aktuální stav odpočtu. Controller daného zobrazení na základě předané hodnoty zkontroluje, nemá-li změnit svůj stav. Pokud k změně stavu dochází, vrací metoda hodnotu `true`.

V případě potřeby je tedy nahrazení vzhledu tohoto úseku velmi jednoduché. Stačí jej vytvořit společně s controllerem implementujícím rozhraní `IPreRoundView` a nastavit vytvořený herní objekt v Unity Editoru do atributu třídy `PreRoundController`.

6.4.4 Pauza

V některých případech je nutné hru zastavit. Z tohoto důvodu bylo vytvořeno zobrazení `Pause View` společně s `PauseController`. Manipulace s tímto nástrojem je velmi jednoduchá a intuitivní, a to skrze veřejné metody `Pause` a `Resume`. Při pozastavení hry je zastaveno měření herního času a na obrazovku se umístí šedý překryv.

Pozastavení hry se využívá hlavně u dialogů pro přeskočení kola nebo předčasné ukončení hry. `PauseController` nabízí možnost pozastavení hry společně se zobrazením dialogu (tzv. modal) invokací metody `ShowModal`.



Obrázek 6.5: Ukázka dialogu

Metoda očekává na vstupu nadpis, obsah a reakce na stisknutí obou tlačítek. Ačkoliv v aktuálním řešení existuje jen tzv. potvrzovací dialog (viz obr. 6.5), další rozšíření by neměla být náročná.

Zdrojový kód 6.4: Příklad použití a zobrazení dialogu (viz obr. 6.5)

```
var title = "Continue";
var content = "Are you sure you want to continue?";
pauseController.ShowModal(title, content, onYes: () =>
{
    Log.Info("The Yes button has been pressed!");
});
```

6.4.5 Hlavní tlačítka

Uživatelské rozhraní má v průběhu kola v pravé části obrazovky umístěna 4 tlačítka – *ukončit hru*, *vypnout/zapnout zvuk*, *ovládání hry* a *nápověda*. Všechny třídy obsahující další funkcionality těchto tlačítek se nachází ve jmenném prostoru `BrainIn.UI.UIButtons` (např. skript hlídající správnost ikony tlačítka pro ztlumení hlasitosti `SoundsButtonController`). Mezi ně se řadí i `TooltipController` a `Tooltip`. Každé z výše zmíněných tlačítek k sobě připojuje „vyskakovací“ popisek (grafické zobrazení popisku a jeho chování reprezentuje třída `Tooltip`). Přidat popisek k tlačítku je možné připojením komponenty `TooltipController` a zavoláním metody `Initialize`, do které je nutné předat obsah popisku (viz zdrojový kód 6.5). Volitelnými argumenty metody jsou nadpis, zarovnání nadpisu, zarovnání obsahu a logická hodnota, jež určuje, má-li se popisek objevovat po stisknutí tlačítka nebo při přejetí kurzoru nad ním.

Zdrojový kód 6.5: Příklad připojení popisku k tlačítku

```
var tc = button.AddComponent<TooltipController>();
tc.Initialize("content", "title", TextAlignmentOptions.Left);
```

Všechna ve hrách vyskytující se tlačítka používají grafické styly z balíčku *6000+ Flat Buttons Icons*, které byly zakoupeny z oficiálního Unity obchodu s assety. [23]

6.4.6 Obsluha stisknutí tlačítka

Obsluhy stisknutí tlačítka se registrují u třídy `MousePointerController`, jež je k tlačítku připojována jako komponenta. Tento skript, mimo jiné, mění i vzhled kurzoru nad tlačítky. Nastavit konkrétní obsluhu tlačítka je možné pomocí metody `Handle` (viz zdrojový kód 6.6). Jejím zavoláním se předaná událost zařadí mezi ostatní obsluhy daného prvku. Při registraci obslužné rutiny se zaznamenává i tlačítko myši, na které má reagovat (`All`, `Left`, `Right` nebo `Middle`). Je tedy možné hernímu objektu nastavit několik reakcí na různá tlačítka myši. Nicméně skutečná stisknutí tlačítek odchytává třída `ClickController` společně s monitorováním všech stisknutí a ukládání těchto akcí do výstupních parametrů (a samozřejmě předává tuto informaci do výše zmíněného skriptu).

Zdrojový kód 6.6: Příklad nastavení obsluhy při stisknutí tlačítka myši

```
skipButton.GetComponent<MousePointerController>().Handle(s =>
{
    Log.Info("Skip button pressed!");
}, ClickType.Left);
```

6.4.7 Přehrávání zvuku

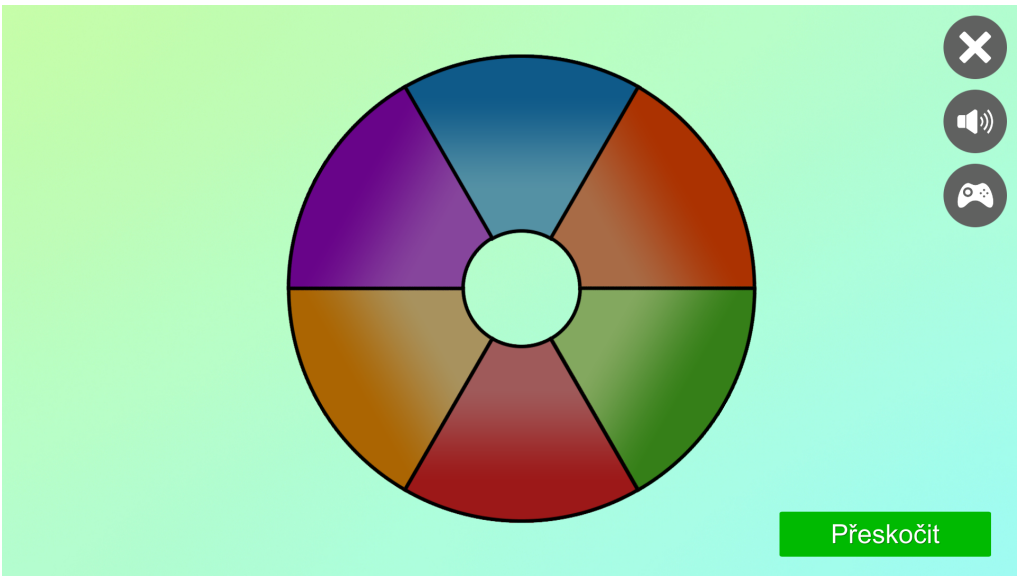
Pro jednoduché přehrávání audio stop je možné využít herní objekt `Audio Player`, jenž má k sobě připojený stejnojmenný skript. Přehrát zvuk lze zavoláním metody `Play`, jež přijímá na vstupu audio stopu (`AudioClip`) a volitelně také hlasitost (hodnoty mezi 0 a 1). Skript reaguje na uživatelova stisknutí tlačítka pro povolení/zakázání zvuku (nastavuje atribut `muted`). V případě, že je tedy zrovna zvuk zakázán, nepřehraje metoda `Play` žádnou audio stopu.

6.5 Hry

Tato sekce popisuje implementovanou herní logiku a generování herních dat každé z navržených úloh.

6.5.1 Signals

Výsečová tlačítka v herní scéně byla vytvořena v grafickém programu. Vzhled výsledné hry je na obrázku 6.6.



Obrázek 6.6: Ukázka herní scény hry Signals

Generování dat

Data pro jednu úroveň tvoří pole členů třídy výčtového typu reprezentující barvy `SignalColor`. Na základě těchto hodnot se budou v průběhu kola zvýrazňovat tlačítka s daným zabarvením.

Do pole každého kola se jednoduše vloží předem daný počet barev. Ty jsou reprezentovány celočíselnými náhodnými hodnotami v rozsahu od 0 do 5, jež jsou následně přetypovány na členy třídy `SignalColor` (celočíselné hodnoty odpovídají hodnotám definovaných členů).

Zdrojový kód 6.7: Ukázka generování herních dat ve hře Signals

```
1 var signals = new SignalColor[signalsNum];
2
3 for (var i = 0; i < signalsNum; i++)
4     signals[i] = (SignalColor) rand.Next(6);
```

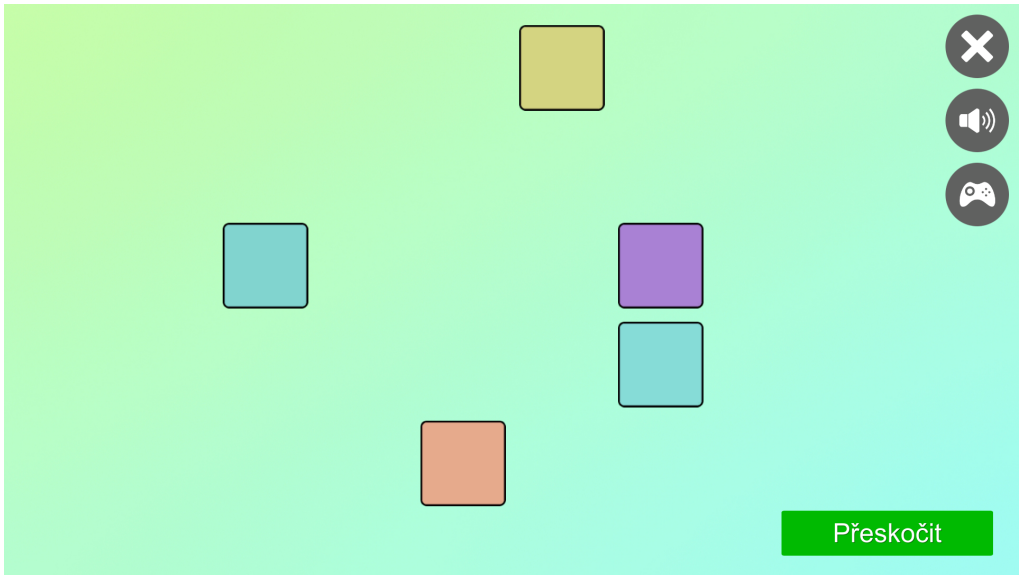
Herní logika

Průběh hry se dělí na dvě části – zapamatování a napodobení. Ihned na začátku kola je spuštěna první fáze zapamatování, v níž je postupně procházeno vygenerované pole barev. V každé iteraci je na základě aktuální barvy získána instance třídy `SliceButtonController`, jež je přidána do fronty představující sekvenci zvýrazněných tlačítek. Následně je zavolána metoda `LightUp` zobrazující animaci zvýraznění tlačítka.

Po dokončení fáze zapamatování přichází na řadu fáze napodobení. Na každé stisknutí herního tlačítka reaguje metoda `OnSignalClick` volající metodu `CheckSelectionCorrectness`. Barva stisknutého tlačítka je přidána do seznamu `signalsActuallyPressed` a je porovnána s barvou, která skutečně měla být stisknuta. V případě shody je tlačítko (resp. controller) odebráno ze začátku fronty. Kolo končí vyprázdněním výše zmíněné fronty nebo pokud je velikost seznamu stisknutých tlačítek větší nebo rovna počtu vygenerovaných barev pro dané kolo.

6.5.2 GlowingSquares

Herní objekty ve scéně jsou vytvářeny z prefabu obsahující grafiku bílého čtverce. Tlačítka jsou vytvářena před spuštěním kola dynamicky. Ukázka herní scény je na obrázku 6.7.



Obrázek 6.7: Ukázka herní scény hry GlowingSquares

Generování dat

Rozsah části scény pro zobrazování herních tlačítek byl rozdělen (v závislosti na velikosti čtverce) na mřížku o 5 řádcích a 9 sloupcích. Informace o jednom čtverci (id, řádek, sloupec a barva) spravuje datová třída `SquareTileData`.

Algoritmus generující obsah kola si připraví seznam instancí výše zmíněné datové třídy, do kterého přidá největší možný počet čtverců ($5 \cdot 9 = 45$). Obsah jednoho kola tvoří několik sekvencí postupně zvýrazňovaných barevných čtverců. Vždy následující sekvence je o jeden čtverec delší než sekvence předchozí. V zdrojovém kódu je tato skutečnost reprezentována frontou polí. Do fronty je v cyklu přidáváno pole o n prvcích vybíraných z předem připraveného a zamíchaného seznamu (viz zdrojový kód 6.8). Náhodu ve vytváření herních dat zařizuje právě zamíchání připraveného seznamu (prováděno Fisher-Yatesovým algoritmem).

Zdrojový kód 6.8: Ukázka generování herních dat ve hře GlowingSquares

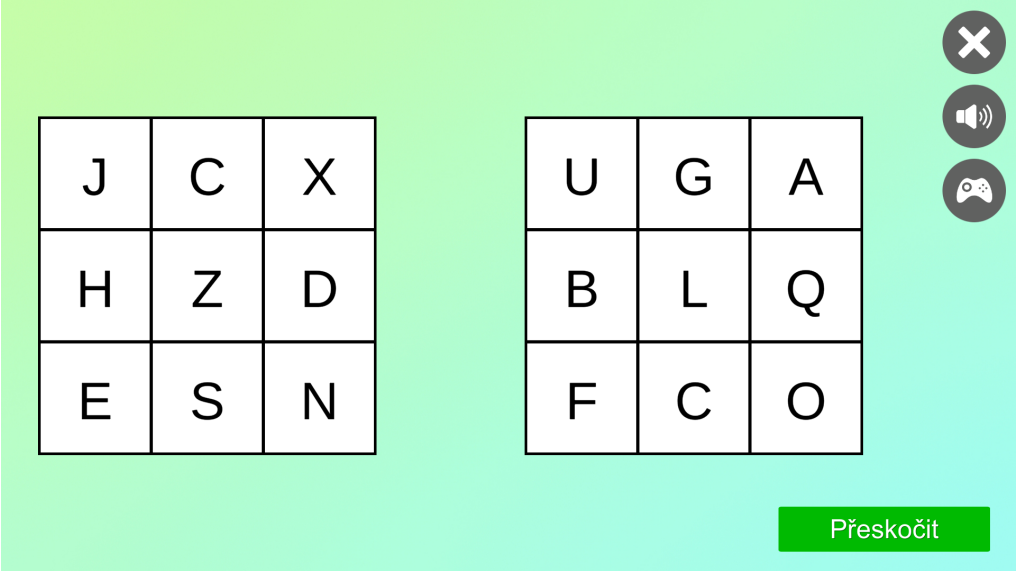
```
var buttonsOrders = new Queue<SquareTileData[]>();

for (var i = initButtonsCount; i <= squares.Length; i++)
{
    Shuffle(squares);
    buttonsOrders.Enqueue(squares.Take(i).ToArray());
}
```

Herní logika

Průběh hry se velmi podobá průběhu hry Signals s několika málo rozdíly. Na začátku kola je nutné instanciovat nové herní objekty tlačítek s jim nastavenými souřadnicemi při generování herních dat. Dalším rozdílem je kontrola správnosti stisknutí tlačítka. Pokud pacient dokáže správně zopakovat celou sekvenci, je znovu spuštěna fáze zapamatování s novou (delší) sekvencí. Kolo končí chybnou volbou nebo dokončením všech připravených sekvencí.

6.5.3 Match'em

Pro tuto hru nebylo nutné vytvářet žádné speciální grafické prvky. Textový nebo grafický obsah (přidaný terapeutem) se vkládá jednoduše do dlaždic každé z mřížek (viz obr 6.8). Obě mřížky jsou vytvořeny staticky v Unity Editoru.



Obrázek 6.8: Ukázka herní scény hry Match'em

Generování dat

Herní data použitá v průběhu každého kola spravuje třída `RoundData`. Každá instance této třídy obsahuje informace o typu mřížky a totožném prvku. Dále pak zahrnuje seznam prvků v levé mřížce, seznam prvků v pravé mřížce a informaci, má-li být mřížka vykreslena vzhůru nohama. Typ mřížky reprezentuje třída výčtového typu `GridType`.

Vygenerovaný obsah kola velmi závisí na hodnotách vstupních parametrů `ShouldCreateImages`, `ShouldCreateLetters` a `ShouldCreateNumbers`. Parametry určují, zda mají být generována data obsahující text nebo terapeutem vložené obrázky. Herní data však vytváří dvě implementované generující funkce – `GenerateImagesGrid` a `GenerateTextGrid`. Druhá zmíněná přijímá na vstupu kromě vstupních parametrů ještě referenci na funkci, jež vrací náhodně vybranou číslici nebo písmeno. Tímto způsobem je možné použít totožný kód k generování více typů obsahu. Potenciální rozšíření o generování mřížek obsahující další textové hodnoty by bylo triviální (viz jedna z variant, která při více nastavených znacích kombinuje písmena a číslice). Možnosti pro mřížky s písmeny jsou vybírány z 26 písmenné anglické abecedy. Číselné mřížky se konstruují z číslic 0 až 9. Na začátku generování se do seznamu generujících funkcí přidají takové, které odpovídají hodnotám výše zmíněných vstupních parametrů. Jedna varianta mřížek je pak vygenerována náhodným výběrem z těchto generujících funkcí (viz zdrojový kód 6.9). Pokud to autor konfigurace vyžaduje, má každá varianta 50% šanci, že její pravá mřížka bude vzhůru nohama.

Zdrojový kód 6.9: Ukázka generování herních dat ve hře Match'em

```
for (var i = 0; i < inputs.Grids; i++)
{
    var genMethod = generations[rand.Next(generations.Count)];
    var grid = genMethod();

    if (inputs.UpsideDown && rand.NextDouble() < 0.5)
        grid.UpsideDown = true;

    grids.Add(grid);
}
```

Herní logika

Herní logika je velmi jednoduchá. Na začátku kola se nastaví všechny dlaždice levé a pravé mřížky (`GridController.Set`). Každý `GridController` obsahuje seznam všech instancí třídy `GridTileController`, jež ovládá jedno políčko v mřížce. Metoda `GridTileController.Set` přijímá na vstupu nastavovaný obsah, typ mřížky a informaci, má-li být políčko označeno jako správné. Metoda tedy podle typu mřížky aktivuje buď komponentu umožňující vkládat text nebo komponentu zobrazující obrázky. Zároveň každé políčko ví, zda je správné, či nikoliv. V průběhu hry je pak možné se ptát konkrétního stisknutého tlačítka, zda mělo být stisknuto. Kolo končí výběrem správného obsahu, který se objevuje v levé i pravé mřížce, nebo překročením maximálního stanoveného počtu chybných voleb.

6.6 JavaScriptové rozhraní

Existující implementace her probíhala v Unity verzi 2018.4.7f1. Novější verze Unity však sestavují aplikace pro WebGL jinak, než tomu bylo dříve. Postupy jejich spuštění na webu se změnily a JavaScriptové soubory, jež se starají o korektní spuštění her tedy musely být přepracovány.

Úpravy skriptů se však netýkaly pouze integrace Unity aplikace do internetové aplikace BrainIn. Nově implementované hry využívají méně metod pro komunikaci s webovou aplikací (např. dialogy byly implementovány přímo v Unity). Velká část tohoto kódu byla odstraněna, zjednodušena a upravena pro jeho lepší pochopení.

Dále byl také upraven přístup k přeloženým textům. Kód nyní vyhledává přeložené texty na základě názvu atributu, a nikoliv podle pořadí. Webová aplikace potřebuje znát dobu pro zobrazování výsledků, která je uložena ve

vstupních parametrech. Kód byl proto upraven, aby jednoduše četl konkrétní hodnotu z datového formátu *JSON*.

Dosud bylo nutné při nasazování aplikace složitě procházet několik souborů a nahrazovat různé textové konstanty konkrétním názvem hry (název hry se musí shodovat s název šablony na webu). Všechna tato místa byla zredukována na pouze jednu konstantu, která se vyskytuje na začátku hlavního skriptu. Nyní je tedy při nasazení potřeba pouze přejmenovat hlavní skript a změnit v něm výše zmíněnou konstantu.

6.6.1 DeployHelper

I po výše zmíněných úpravách byla integrace her do systému BrainIn stále příliš zdlouhavá a otravná. Z tohoto důvodu byl vytvořen primitivní skript v jazyce Python. Uživatel skriptu `DeployHelper` zadá do textového vstupu název hry, skript následně najde adresář se stejným názvem a sestavenou Unity hrou. Do tohoto adresáře překopíruje všechny JavaScript soubory, přejmenuje výše zmíněnou konstantu společně i s hlavním skriptem, odstraní všechny zbytečné soubory a připraví adresář tak, že je poté možné už bez dalších kroků nahrávat soubory přímo do webové aplikace.

Tento skript byl však poté přepsán do jazyka C#, byl vytvořen tzv. Editor skript, který přidává do lišty hlavní nabídky vývojového prostředí vlastní menu a možnosti automatického sestavení včetně přípravy souborů k nasazení. Proces sestavení a přípravy souborů k nasazení byl tak zredukován na jeden krok (název hry je automaticky doplňován z vlastností projektu). Skript byl vytvořen pouze pro automatizaci procesu a zjednodušení práce. Otestován byl jen jeho používáním, žádné cílené testování skriptu neproběhlo.

7 Ověřování kvality

Ověřování kvality software (vytvořené kostry a všech her) bylo rozděleno do tří fází. V první fázi byla ověřena správnost implementace na úrovni jednotkových testů ve vývojovém prostředí Unity. V druhé fázi byl testován průběh her, včetně provedení kontroly správnosti výstupních hodnot. V poslední fázi byly hry testovány na skutečných klientech pana doktora Pavla Krále, kteří trpí poruchami pozornosti a krátkodobé paměti.

7.1 Jednotkové testování

Implementovaná kostra byla před svým rozšířením do dalších her pokryta jednotkovými testy. Testovací skripty se nachází v jmenných prostorech `BrainIn.Tests.Backend`. Testování aplikace se zaměřuje nejen na backendovou část, ale částečně i na tu frontendovou. Správná funkčnost herní logiky rozhraní byla zkontrolována v dalších fázích testování.

Unity nabízí použití testovacího frameworku *TestRunner*, který dovoluje testování kódu přímo ve vývojovém prostředí. Tento framework je integrací open-source knihovny *NUnit* pro jednotkové testování .NET jazyků do vývojového prostředí Unity. [26]

TestRunner nabízí dva módy testování – *EditorMode* a *PlayMode*. Testy v *EditorMode* jsou spouštěny přímo v Editoru, proto je tento režim ideální pro otestování všech tříd, které přímo nezávisí na Unity skriptech ovládajících herní objekty (`MonoBehaviour` atd.). *PlayMode* testy slouží pro simulaci běhu aplikace a otestování různých částí herní scény apod.

Bylo vytvořeno cca 240 testů. Většina z nich ověřuje přímo implementovanou kostru a zbytek ověřuje specifické funkcionality jednotlivých her. Vytvořené testy jsou zastoupeny v obou výše zmíněných režimech. Každá testová metoda je implementována podle AAA vzoru a její jméno vždy popisuje metodu či funkcionalitu, jež testuje.

7.1.1 NSubstitute

Při testování backendových tříd je nutné v některých případech nahradit reference na jiné objekty jejich tzv. *mock* variantami. Knihovna *NSubstitute* umožňuje jednoduše vytvářet mock objekty a s nimi otestovat třídy, které na ně v tu chvíli mají referenci. [25]

Například validátor vstupních parametrů vlastní referenci na manažera spravující jazykové mutace. Pro otestování průběhu validace však není třeba mít chybové hlášení v konkrétním jazyce. Validátoru se tedy předá reference na *mock* objekt a nastaví se mu odpověď na zavolání metody `Get` (viz ukázka zdrojového kódu 7.1).

Zdrojový kód 7.1: Ukázka vytvoření mock objektu

```
var mock = Substitute.For<II18nManager>();
mock.Get(Arg.Any<string>()).Returns("{0}");
```

7.1.2 FluentAssertions

Pro lepší porozumění vytvořeným testům (ale i pro jejich jednodušší vytváření) byla využita knihovna *FluentAssertions*. Knihovna umožňuje psát srozumitelnější testy, umožňuje kombinovat více testů dohromady a produkuje lepší chybová hlášení (viz zdrojový kód 7.2). [5, 10]

Zdrojový kód 7.2: Ukázka aserce v testové metodě

```
var numbers = new int[] { 1, 2, 3 };

numbers.Should()
    .HaveCount(3).And
    .OnlyContain(n => n > 0);
```

7.1.3 Pokrytí testy

V rámci ověřování kvality implementovaného řešení bylo dosaženo 70% pokrytí testy (procenta udávají pokrytí řádků kódu, v případě pokrytí metod se jedná o hodnoty kolem 85 %). Backend aplikace je testy pokryt z 99 %. Pokrytí frontendové části činí cca 40 %, kde testy ověřují hlavně jednotlivé komponenty uživatelského rozhraní (přehrávač zvuků, skripty ovládající dialogy apod.). Ostatní části aplikace jsou následně testovány při manuálních testech.

Metrika pokrytí zdrojového kódu testy byla realizována knihovnou *Code Coverage*, jež nabízí generování výsledných reportů s užitečnými statistikami, historií a vizualizací (ne)pokrytých částí zdrojového kódu. [27]

7.2 Beta testování

Účelem této fáze testování bylo ověřit správnou funkčnost vytvořené šablony a průběh jednotlivých her. Dále byly kontrolovány vygenerované výstupní

hodnoty z odehraných her. Testování bylo realizováno za účelem nalezení chyb ještě před předáním aplikací panu doktoru Pavlu Královi k testování na jeho skutečných klientech.

Beta testování probíhalo dvoukolově na ZŠ a MŠ Vacov v rámci zdravotnického kroužku. Vytvořené konfigurace her (18 variant) si v průběhu několika hodin vyzkoušelo zahrát cca 50 dětí (v prvním kole cca 30, v druhém 21) ve věku od 8 do 10 let. Výsledky se úspěšně ukládaly do webové aplikace. Na základě výsledků beta testování bylo možné opravit nalezené chyby a potvrdit tak funkčnost nového systému komunikace, korektnost výstupních dat a bezproblémový průběh každé z her (včetně pochopení herních instrukcí dětmi).

7.3 Testování na Katedře klinické psychologie

PhDr. Pavel Král, Ph.D. vytvořil několik konfigurací úloh, které postupně ukazoval a přiřazoval svým klientům. Během terapií má každý proband vytvořené balíky úloh několikrát odehrát. Výstupní hodnoty byly panem doktorem v průběhu terapie analyzovány.

Během několika týdnů byly úlohy představeny několika klientům pana doktora Krále. U dvou osob se hry staly také součástí skutečné terapie. Bohužel průběh terapií není zcela banální proces a vyžaduje několikanásobně delší dobu trvání. Z tohoto důvodu nemohl být počet těchto osob vyšší.

Implementované hry byly panem doktorem akceptovány a navrhované úpravy, které vyplynuly z testování, byly ještě před předáním finálního produktu zapracovány.

8 Zhodnocení dosažených výsledků

V rámci této diplomové práce byla navržena a poté vytvořena funkční a rozšiřitelná kostra (šablona) her s využitím ověřených postupů v softwarovém inženýrství. Implementovaná kostra přichází s architektonickým návrhem podporujícím rozšiřitelnost či implementaci zcela nových her a novým intuitivním způsobem komunikace s webovou aplikací BrainIn. Nový systém komunikace a předávání parametrů s internetovou aplikací vyžadoval změny přímo v jejím projektu. Všechny změny byly realizovány bez narušení kompatibility s již existujícími hrami. Podobně se chovají i různé části implementované šablony dovolující vytvořit hru, jež bude zpětně kompatibilní. Způsob odesílání a získávání informací umožňuje jednoduché úpravy v již existujících šablonách v uživatelském rozhraní webové aplikace. Je tedy možné například odebírat nechtěné výstupní parametry bez nutnosti sestavení nové verze hry v Unity (internetová aplikace nežádoucí přijaté hodnoty zkrátka ignoruje). Zároveň ale významně zjednodušuje rozdělení řetězce s vstupními parametry do příslušných proměnných uvnitř Unity aplikace. Současně bylo upraveno a zjednodušeno JavaScriptové rozhraní tak, aby byla integrace budoucích her do internetové aplikace jednodušší.

Rozšířením výše zmíněné kostry byly vytvořeny celkem 3 parametrizovatelné funkční hry, jež mají pomáhat lidem při terapiích nebo kognitivním tréninku s jejich obtížemi v oblasti krátkodobé paměti a pozornosti. Kvalita vytvořených her (současně s původní kostrou) byla ověřena nejen jednotkovými testy, pokrývajícími přes 70 % zdrojového kódu, ale i tzv. beta testováním v ZŠ a MŠ Vacov. Výše zmíněná tvrzení, že je implementovaná kostra velmi jednoduše rozšiřitelná podporuje skutečnost, že po otestování vytvořené kostry a všech tří požadovaných her byly v relativně krátkém časovém období (10 hodin) implementovány ještě dvě další hry. Všechny vytvořené hry byly úspěšně integrovány do systému BrainIn nasazením na jeho produkční prostředí.

Dne 8. 3. 2023 byl uskutečněn online workshop s klinickými psychology z Katedry klinické psychologie při Institutu postgraduálního vzdělávání ve zdravotnictví. V rámci workshopu byly prezentovány vytvořené hry společně s popisem webové aplikace BrainIn. Obsahem školení byl popis základních postupů pro vytvoření účtu, vytvoření úlohy, přidání úlohy do balíku a přiřazení jej pacientům. Vytvořené hry byly představeny lidem, kteří

s nimi dosud nebyli obeznámeni, včetně podrobnějšího vysvětlení vstupních a výstupních hodnot, pravidel pro jejich zadávání a významu.

PhDr. Pavel Král, Ph.D. provedl test her na svých klientech a skutečných osobách. Hry byly součástí skutečných terapií, tudíž lze s jistotou očekávat, že budou účinným nástrojem pro léčbu pacientů s poruchami pozornosti a krátkodobé paměti, a to nejen v rámci terapeutických sezení, ale i jako součást individuálního tréninku v domácím prostředí. Během návrhu a testování her byla diskutována možnost převedení vytvořených testovacích zkoušek na normalizované testy. Tento proces však obvykle trvá delší dobu a vyžaduje zapojení několikanásobně větší testovací populace. Ačkoliv proces standardizace přesahuje rozsah této diplomové práce, shodl se PhDr. Pavel Král, Ph.D. s PhDr. Karolínou Malou, že by do budoucna bylo teoreticky možné pokusit se z implementovaných testů vytvořit standardizovanou normu.

Vytvořená šablona se stala intuitivním nástrojem pro rychlou tvorbu her pro kognitivní trénink. Implementace je rozšiřitelná nejen v rámci převodu na novou hru, ale také při přidávání nových funkcionalit, jako je například přidání nového typu vstupního parametru nebo úplně nové obrazovky s nápovědou k hře (tutorial) před začátkem hry. Toto řešení by tak mohlo být označeno jako výchozí bod pro implementaci parametrizovatelných her do aplikace BrainIn. Současně by do této kostry mohly být převedeny všechny již existující hry, čímž by se zajistil jednotný vzhled a jejich snadná modifikovatelnost. Nicméně tato skutečnost by vyžadovala určité úpravy, neboť některé z existujících her obsahují funkcionality, které nebyly panem doktorem Králem přímo vyžadovány, jako například tutorial.

9 Závěr

V rámci této diplomové práce byla nejdříve prozkoumána webová aplikace BrainIn společně s implementací již existujících her. Teoretická část práce se věnuje také stručnému studiu kognitivních funkcí a teorii v oblasti pozornosti a krátkodobé paměti (společně s existujícími testy těchto kognitivních funkcí). Na základě znalostí o existující implementaci (práce v jednotlivých kapitolách popisuje různé obskurnosti existující šablony znesnadňující potenciální vývoj) byla navržena zcela nová šablona, která by měla sloužit pro budoucí implementace her do systému BrainIn. Součástí návrhu byly po diskuzích s PhDr. Pavlem Králem, Ph.D. navrženy 3 hry (vzhled, průběh, vstupní parametry, výstupní hodnoty), jež by měly sloužit při budoucích terapiích.

Na základě návrhu byla implementována nová šablona her. Implementované jednoduše rozšiřitelné řešení využívá několik návrhových vzorů a obecně známých doporučených postupů v softwarovém inženýrství, kód je dobře strukturován a okomentován. Z šablony vychází 5 vytvořených her (3 původní, 2 bonusové) splňující všechny požadavky stanovené v návrhu. Hry by měly být schopné pomáhat při terapiích u lidí s problémy v oblasti pozornosti a krátkodobé paměti. Hry přijímají vstupní hodnoty, které mění jejich průběh a generují hodnoty výstupní, které mohou autoři a uživatelé konfigurací her analyzovat. Kvalita implementovaného řešení byla ověřena pomocí testovacích nástrojů (*NUnit*, *FluentAssertions*, *NSubstitute*). V rámci beta testování byly hry podrobeny zkouškám na několika skutečných osobách.

Všechny požadavky pana doktora Krále byly splněny. Průběh vývoje her byl s panem doktorem několikrát konzultován nad skutečnými prototypy a ukázkami. Hry byly otestovány také na skutečných klientech Katedry klinické psychologie IPVZ.

Vytvořená kostra a všechny hry byly po nutných úpravách webové aplikace (nový systém komunikace) a JavaScriptového rozhraní (novější verze vývojového prostředí Unity) úspěšně integrovány do systému BrainIn. Začleněné hry jsou užívány pro terapie a trénink na Katedře klinické psychologie IPVZ a v ambulantní praxi pana doktora Krále.

Literatura

[1] Bartoš, A. Kognitivní funkce, soběstačnost a kognitivní syndromy. *Psychiatrie pro praxi*. 2022, s. 91–97.

[2] Bartošek, J. Vývoj parametrizovaných her v Unity pro pacienty trpící poruchou pozornosti s hyperaktivitou. Bachelor's Thesis, Faculty of Applied Sciences, University of West Bohemia, 2021.

[3] Bishop, J. *C# - návrhové vzory*. Zoner Press, 2010. ISBN 978-80-7413-076-2.

[4] Bonet, R. T. *A better architecture for Unity projects* [online]. Game Developer, 2018. [cit. 20.03.2023]. Dostupné z: https://www.gamedeveloper.com/disciplines/a-better-architecture-for-unity-projects.

[5] Boundfox Studios. *Fluent Assertions for Unity* [online]. 2022. [cit. 17.04.2023]. Dostupné z: https://github.com/BoundfoxStudios/fluentassertions-unity.

[6] Chikin Studio. Chikin Icons For Everything, 2021. Dostupné z: https://www.figma.com/community/plugin/910824263334396226/Chikin-Icons-For-Everything.

[7] Chmelařová, D. Rehabilitace kognitivních funkcí. *Neurology for practice*. 2018, s. 62–69. ISSN 12131814. Dostupné z: https://www.neurologiepropraxi.cz/artkey/neu-201691-0012.php.

[8] Department of computer science and engineering, University of West Bohemia. *BrainIn* [online]. 2020. Dostupné z: https://brainin.kiv.zcu.cz.

[9] Department of computer science and engineering, University of West Bohemia. *BrainIn - Terms* [online]. 2020. [cit. 01.03.2023]. Dostupné z: https://brainin.kiv.zcu.cz/home/Terms.

[10] Doomen, D. *Fluent Assertions* [online]. 2022. [cit. 17.04.2023]. Dostupné z: https://fluentassertions.com.

[11] Gamma, E. et al. *Návrh programů pomocí vzorů*. Grada publishing, 2003.

[12] Hocking, J. *Unity in action: multiplatform game development in C#*. Simon and Schuster, 2022.

[13] Kovko, E. *Dependency Injection vs. Service Locator* [online]. Baeldung, 2023. [cit. 21.03.2023]. Dostupné z: https://www.baeldung.com/cs/dependency-injection-vs-service-locator.

[14] Kuzmenko, E. *Model-View-Controller Architecture Pattern: Usage, Advantages, Examples* [online]. Hackernoon, 2022. [cit. 20.03.2023]. Dostupné z: https://hackernoon.com/model-view-controller-architecture-pattern-usage-advantages-examples.

[15] Lezak, M. D. et al. *Neuropsychological assessment.* Oxford University Press, USA, 2004.

[16] Mojmír, S. et al. *Psychodiagnostika dospělých.* Portál, 2013. ISBN 978-80-262-0363-6.

[17] Muratet, M. et al. *Accessibility and serious games: What about Entity-Component-System software architecture?* Springer, 2020.

[18] Nageshkumar, M. et al. Handling Unhandled Exceptions in Python 3 using Dependency Injection (DI) or Inversion of Control (IoC). *International Journal of Engineering Research & Technology (IJERT).* 2021.

[19] Okita, A. *Learning C# programming with Unity 3D.* CRC press, 2014.

[20] Parisi, T. *WebGL: up and running.* O'Reilly Media, Inc., 2012.

[21] Pecinovský, R. *Návrhové vzory.* Computer Press, 2013. ISBN 978-80-251-1582-4.

[22] Pezoa, F. et al. Foundations of JSON schema. In *Proceedings of the 25th international conference on World Wide Web*, s. 263–273, 2016.

[23] RainbowArt. 6000+ Flat Buttons Icons Pack. Unity Asset Store, 2020. Dostupné z: https://assetstore.unity.com/packages/2d/gui/icons/6000-flat-buttons-icons-pack-64223.

[24] Skala, P. Webová aplikace pro správu neurorehabilitačních programů. Master's thesis, Faculty of Applied Sciences, University of West Bohemia, 2019.

[25] Tchepak, D. et al. *NSubstitute* [online]. 2020. [cit. 17.04.2023]. Dostupné z: https://nsubstitute.github.io.

[26] Unity Technologies. *About Unity Test Framework* [online]. 2022. [cit. 17.04.2023]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html.

[27] Unity Technologies. *About Code Coverage* [online]. 2021. [cit. 17.04.2023]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.testtools.codecoverage@1.1/manual/index.html.

[28] Unity Technologies. *About Localization* [online]. 2023. [cit. 22.03.2023]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.localization@1.4/manual/index.html.

[29] Unity Technologies. *Interaction with browser scripting* [online]. 2022. [cit. 17.03.2023]. Dostupné z: https://docs.unity3d.com/2022.2/Documentation/Manual/webgl-interactingwithbrowserscripting.html.

[30] Vágnerová, M. *Vývoj pozornosti a exekutivních funkcí*. Dr. Josef Raabe s.r.o., 2020. ISBN 978-80-7496-441-1.

[31] Vágnerová, M. *Obecná psychologie: dílčí aspekty lidské psychiky a jejich orgánový základ*. Charles University in Prague, Karolinum Press, 2016. ISBN 978-80-246-3268-1.

[32] Válková, L. *Rehabilitace kognitivních funkcí v ošetřovatelské praxi*. Grada Publishing, a.s., 2015. ISBN 978-80-247-5982-1.

[33] Viswan, A. *Event Aggregator* [online]. ByteLanguage.Net, 2021. [cit. 23.03.2023]. Dostupné z: https://bytelanguage.com/2021/07/19/event-aggregator/.

A Přehled zkratek

ECS Entity-Component-System

IPVZ Institut postgraduálního vzdělávání ve zdravotnictví

JSON JavaScript Object Notation

KIV Katedra informatiky a výpočetní techniky

PoC Proof of Concept

UI User interface - uživatelské rozhraní

B Seznam obrázků

C Uživatelská dokumentace

C.1 Funkce tlačítek

Implementovaná kostra nabízí v každém kole interakci s několika tlačítky. Obrazovka s tlačítky je na obrázku C.1 a jejich funkce jsou následující:

- Tlačítko 1 zobrazí dialog pro předčasné ukončení úlohy.
- Tlačítko 2 umožňuje zapnout/vypnout zvuk.
- Tlačítko 3 zobrazí nápovědu k ovládání aktuální hry.
- Tlačítko 4 zobrazí nápovědu ke konkrétní situaci v daném kole.
- Tlačítko 5 slouží k přeskočení kola.

Do středu obrazovky se obvykle vkládají herní prvky.

Obrázek C.1: Herní scéna v průběhu kola

C.2 Návod k implementaci

Níže je doporučený návod k vytvoření nové hry do implementované šablony. Jedná se vlastně o postup, jak byly implementovány výše zmíněné hry.

1. V otevřeném Unity Editoru z horní lišty vyberte **Edit** a následně **Project Settings**. V nově otevřeném okně změňte **Product Name** z `BrainIn - Template` na `BrainIn - <nazev_hry>`.

2. Výchozí parametry v třídě `InputParamsParser` upravte tak, aby konfigurace vedla ke korektnímu spuštění hry.

3. Upravte datové třídy vstupních parametrů (`Backend/InputData`) tak, aby parser dokázal úspěšně přeměnit textový řetězec na instanci třídy (lze ověřit laděním po deserializaci parametrů nebo jednotkovým testem).

4. Napište validaci hodnot vstupních parametrů (je nutné znát rozsahy apod.).

5. V skriptu `GameController` implementujte herní logiku, vytvořte herní objekty do `Game View/Game Objects` v herní scéně a implementujte příslušné skripty ovládající vytvořené objekty.

6. Doplňte datové třídy v `Backend/OutputData`, `IResultsCollector` a `ResultsCollectorBase` tak, aby se korektně zaznamenávaly požadované výstupní hodnoty.

7. Pokryjte vámi implementované třídy jednotkovými testy.

8. Sestavte aplikaci, nasaďte do webové aplikace a ověřte její funkčnost.

C.3 Návod k sestavení a nasazení

Návod na sestavení hry pro operační systém Windows.

1. Přes instalátor verzí vývojového prostředí Unity instalujte Unity ve verzi 2022.1.23f1.

2. Přímo v Unity Hubu zvolte možnost **Open** a vyberte cestu ke kořenové složce projektu aplikačních souborů `Development/<nazev_hry>/`.

3. Po spuštění projektu je občas potřeba najít v prohlížeči souborů projektu adresář `Scenes` a dvojím kliknutím na scénu *Main* ji vybrat jako aktivní.

4. V otevřeném vývojovém prostředí se zvoleným projektem v hlavní (horní) liště vyberte **BrainIn → Build + deploy prep**.

5. Po úspěšném sestavení a přípravě ostatních souborů budou v adresáři `Deployment/<nazev_hry>/` všechny nezbytné soubory k integraci hry do webové aplikace BrainIn a její správné funkčnosti.

D Elektronické přílohy

Obsah souboru `A21N0039P_prilohy.zip`, jenž byl přiložen k diplomové práci, tvoří několik adresářů a souborů:

- Obsah adresáře `Aplikace_a_knihovny` tvoří další dva podadresáře – `Deployment` a `Development`.
  - V `Development` jsou Unity projekty všech her společně s adresářem `Template`, jenž obsahuje Unity projekt implementované šablony, nad kterou je možné vytvářet další hry (viz kapitola C.2). V každém z nich je adresář `Assets`, ve kterém jsou všechny soubory a zdrojové kódy, ze kterých lze sestavit WebGL aplikace. Git repository diplomové práce je k dispozici na internetové adrese `https://bitbucket.org/bartosej/diplomova-prace` (větev *master* reprezentuje šablonu, každá z her má pak svou vlastní větev, což zároveň usnadňuje aplikaci obecných změn a vylepšení do každé z her). Návod k sestavení je v textu diplomové práce v kapitole C.3.
  - `Deployment` obsahuje také podadresáře pro každou z her, tentokrát jsou však jejich obsahem sestavené hry společně se všemi soubory potřebnými k její integraci do internetové aplikace BrainIn. Sestavené hry je možné si spustit a prohlédnout na následujících adresách (nebo přímo v Unity):

    Signals:
    `https://brainin.kiv.zcu.cz/Test/Demo?exerciseId=1105`

    GlowingSquares:
    `https://brainin.kiv.zcu.cz/Test/Demo?exerciseId=1134`

    Matchem:
    `https://brainin.kiv.zcu.cz/Test/Demo?exerciseId=1128`

    CircleDot:
    `https://brainin.kiv.zcu.cz/Test/Demo?exerciseId=1144`

    CupsAndBall:
    `https://brainin.kiv.zcu.cz/Test/Demo?exerciseId=1146`

- Adresář `Poster` obsahuje všechny soubory k vytvořenému posteru.
- Obsahem adresáře `Text_prace` jsou všechny zdrojové soubory pro překlad diplomové práce do PDF a výsledek překladu s textem diplomové práce.
- Adresář `Vysledky` obsahuje dva podadresáře:
  - `beta` obsahuje soubor výsledků soutěže mezi testery při druhém kole beta testování, snímky obrazovky vytvořených balíků a konfigurací úloh při testování.
  - `ipvz` obsahuje snímky obrazovky výsledků některých klientů pana doktora Krále.