VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POČÍTAČOVÁ HRA DEMONSTRUJÍCÍ 3D VIZUALIZAČNÍ MOŽNOSTI DNEŠNÍCH INTERNETOVÝCH PROHLÍŽEČŮ

A DEMONSTRATION COMPUTER GAME SHOWING 3D VISUALIZATION CAPABILITIES OF CONTEMPORARY INTERNET BROWSERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **JAKUB PAGÁČ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN PEČIVA, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Pagáč Jakub, Bc.**
Program: Informační technologie
Obor: Počítačové sítě
Název: **Počítačová hra demonstrující 3D vizualizační možnosti dnešních internetových prohlížečů**
A Demonstration Computer Game Showing 3D Visualization Capabilities of Contemporary Internet Browsers
Kategorie: Počítačová grafika
Zadání:

1. Nastudujte si postupy tvorby webových aplikací a webové technologie pro zobrazování 3D grafiky. Také si nastudujte přístupy pro tvorbu animací postav.
2. Navrhněte hru formou internetové aplikace demonstrující možnosti zobrazování 3D grafiky a animace postav.
3. Navrženou hru implementujte. Volitelně doplňte podporu pro zvuk a síťovou komunikaci.
4. Aplikaci otestujte na různých platformách. Vyhodnoťte zkušenosti a plynulost běhu aplikace na testovaných platformách.
5. Práci zveřejněte na internetu pod některou z open-source licencí.

Literatura:

- materiály k WebGL, WebGPU a animaci postav dle doporučení vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Funkční prototyp aplikace zobrazující postavu s jednoduchou animací.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Pečiva Jan, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 30. října 2020

Abstrakt

Tato práce popisuje postup tvorby hry pro webové prohlížeče za pomoci frameworku Babylon.js. Také zmiňuje témata tvorby 3D grafiky na webu, 3D animací a struktury webových aplikací. Hra samotná je psaná pomocí jazyka Typescript a je sdílená pomocí webového serveru využívající Node.js moduly. Jedná se o bojovou hru pro dva hráče, kde jejich postavy bojují za pomoci bojových umění.

Abstract

This thesis describes the process of creation of a game for web browsers using Babylon.js framework. It also touches on subjects of 3D graphics on the web, 3D animation and structure of web applications. Game itself is created using Typescript language and it is shared using a Node.js module as its web server. It is a fighting game for two players, where their characters fight using martial arts.

Klíčová slova

3D grafika, Babylon.js, grafika na webu, hra, Node.js, tvorba her, Typescript, webový prohlížeč, WebGL

Keywords

3D graphics, Babylon.js, graphics on web, game, game making, Node.js, Typescript, web browser, WebGL

Citace

PAGÁČ, Jakub. *Počítačová hra demonstrující 3D vizualizační možnosti dnešních internetových prohlížečů.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Pečiva, Ph.D.

Počítačová hra demonstrující 3D vizualizační možnosti dnešních internetových prohlížečů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Pečivy Ph.D.. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Pagáč
19. května 2021

Poděkování

Chtěl bych poděkovat svému vedoucímu diplomové práce Ing. Janu Pečivovi Ph.D. za ochotu a vstřícnost při vedení této diplomové práce.

Obsah

Kapitola 1

Úvod

Vývoj internetu a výpočetních technologií přinesl světu mnoho nových odvětví, a také zajistil rozvoj těch stávajících. Mnoho věcí, které nám nyní přijdou jako běžné, tady před několika lety vůbec nebylo. Ať už se jedná o sociální sítě, získávání a sdílení informací, sledování filmů a seriálů a mnoho dalších věcí. Je velmi důležité si uvědomit také dosah internetu a jaké zařízení jsou k němu připojena. Už se nejedná pouze o počítače, ale hlavně mobilní telefony a v poslední době je tu také trend internetu věcí. Internet je obrovská platforma na které se dají budovat různé aplikace pro různá zařízení, čímž lze oslovit velké množství uživatelů.

Když tohle dáme do kontextu herního průmyslu, který je již po dobu několika let nejvydělenějším zábavním odvětvím a mnoho nejúspěšnějších her poslední doby získalo popularitu právě využitím internetu například pro sdílení svého obsahu a budování komunity, lze si položit otázku, kdy vznikne nějaká hra, jejíž zaměření bude primárně na webové prohlížeče a tím bude dostupná na mnoha různých zařízeních. Je také důležité se ptát, zda je něco takové v dnešní době možné a pokud ne, tak zjistit z jakých důvodů tomu tak není a jak se tato situace dá do budoucna změnit.

Tato práce si dává za cíl tuto myšlenku prozkoumat. Jejím cílem je vytvořit hru, která má demonstrovat možnosti dnešních prohlížečů a určit, jaké jsou možnosti tvorby her na internetu, jak náročné je tvořit se zaměřením na web a jak je výsledek použitelný na různých zařízeních. V kapitole 2 se probírá stavba internetových aplikací, tvorba 3D animací a grafiky na webu. Pro tvorbu grafiky na webu byl zvolen framework Babylon.js, který byl použit při implementaci hry a jeho vlastnosti jsou obsahem také této kapitoly. Kapitola 3 popisuje postup návrhu a implementace zvolené hry, přičemž jsou v ní rozebrány nejpodstatnější části z daných témat vzhledem k vytvořené hře. Kapitola 4 popisuje, jak byla výsledná hra testována a jakých bylo dosažených výsledků. Poslední kapitola 5 poté zhodnocuje výsledek práce a navrhuje její rozšíření a zlepšení.

Kapitola 2

Zhodnocení současného stavu

Tato kapitola obsahují základní informace, které se týkají tvorby aplikací na internetu. Obsahuje také popis tvorby 3D animací od počátečního vytvoření postavy až po tvorbu jejich pohybů. Poslední část tvoří vysvětlení procesu vykreslování grafiky na webu a představení frameworku Babylon.js, který lze využít pro efektivní tvorbu 3D grafiky pro webové prohlížeče.

2.1 Tvorba webových aplikací

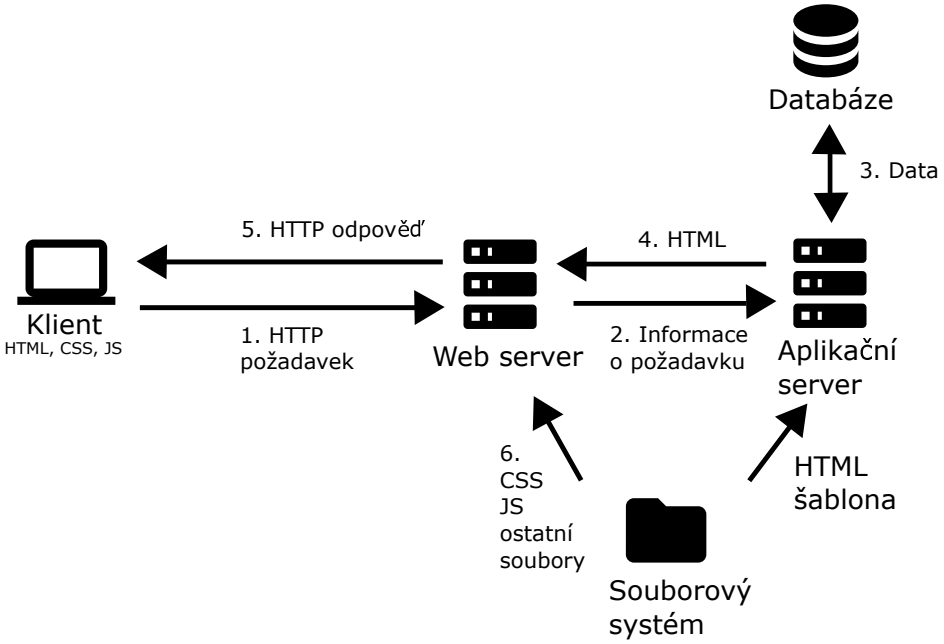
Tato podkapitola má čtenáře seznámit se strukturou webové aplikace a popsat její jednotlivé části. Nejedná se o kompletní popis, ale o výčet nejdůležitějších věcí pro pochopení fungování aplikace. Pro kompletní přehled informací k daným webovým technologiím jsou nejlepším zdrojem jejich standardy dostupné na internetu.

Webová aplikace je program typu klient–server, který běží na vzdáleném serveru a uživatel k němu přistupuje přes internet pomocí klienta – webového prohlížeče. Od běžných programů se liší v tom, že uživatel nemusí program instalovat, ale aplikace se stáhne při otevření dané stránky v prohlížeči. Webové aplikace jsou však závislé na internetovém připojení. Pokud nemá klient nebo server přístup k internetu, tak se daná aplikace stává nefunkční a pravděpodobně nebude správně fungovat.

V počátcích Internetu byly webové stránky tvořeny pouze jediným dokumentem, který byl tvořen pouze textem s jednoduchým formátováním a podporou odkazování na jiné dokumenty stejného typu. Tento koncept byl poté rozšířen tak, že bylo přidáno více možností formátování, text šlo stylizovat a přidávat do něj zvukové soubory a obrázky. Vzhled stránek se tedy zlepšil, ale stále se jednalo o statické dokumenty, s kterými uživatel nemohl manipulovat.

Další evoluce přišla s integrací podpory jazyka Javascript, který dodal statickým stránkám možnost dynamicky měnit jejich obsah. Změna obsahu tedy mohla probíhat bez komunikace se serverem nebo se nemusela měnit celá stránka, ale pouze její část. Tato změna značí počátek vývoje webových aplikací [18]. Další rozvoj přinesly přístupy tvorby, které se snaží webové aplikace přiblížit běžným aplikacím. Mezi ně patří například single page application, kdy aplikace má jednu základní stránku, kterou dynamicky mění za pomoci asynchronní komunikace se serverem, nebo progressive web application, kde se snažíme aplikaci implementovat tak, aby se funkčně přiblížila běžné aplikaci i mobilním zařízení.

Struktura běžné webové aplikace je ukázána na obrázku 2.1. Aplikace je rozdělena na klientskou a serverovou část. Klientskou část tvoří HTML, CSS a Javascript (JS) soubory,

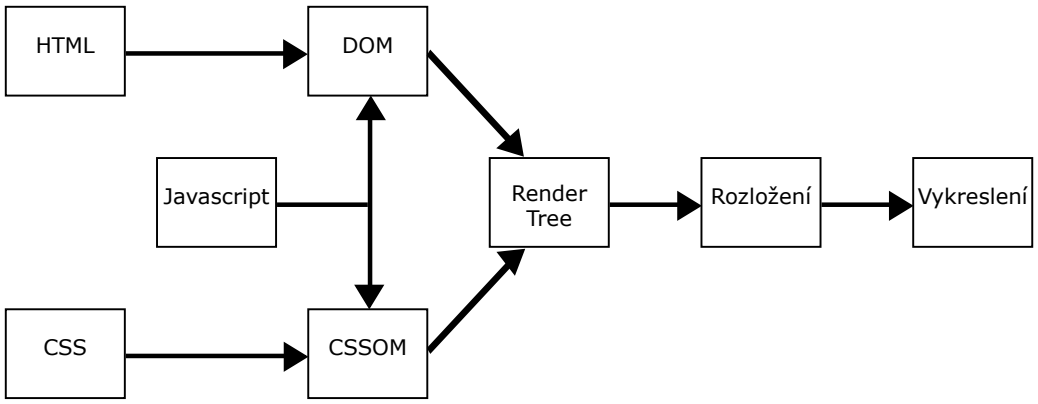


Obrázek 2.1: Struktura webové aplikace

zatímco serverovou část tvoří webový server, aplikační server a databáze. Tyto části jsou podrobněji popsány v následujících podkapitolách.

2.1.1 Client-side

Tato část aplikace je dostupná v internetovém prohlížeči u uživatele. Slouží k určení, jak se bude aplikace na klientově zařízení chovat a zobrazovat. Ovlivňuje se zde rozložení a styl prvků aplikace, navigace nebo ověřování formulářů aplikace. Hlavními částmi jsou již zmíněné HTML, CSS a JS soubory. Dále může být stránka tvořená mediálním obsahem jako jsou obrázky, videa nebo zvukové soubory. Všechny tyto soubory jsou klientovi zaslány ze serveru, který je lokálně uloží a následně je zpracuje, jak je naznačeno na obrázku 2.2. Nejdříve se zpracuje HTML soubor, ze kterého se vytvoří DOM neboli Document Object Model, což je stromová struktura, která slouží k reprezentaci objektů definovaných



Obrázek 2.2: Proces vykreslení stránky

v HTML v paměti prohlížeče, které lze pak upravovat pomocí Javascriptu a které se použijí pro vykreslení stránky. Podobná struktura se vytvoří i z CSS souborů. Ta se nazývá CSSOM neboli CSS Object Model a opět slouží k manipulaci se styly stránky za pomoci JS. Oba tyto stromy se poté složí do jediného stromu nazývaného Render Tree, který slouží k určení pořadí vykreslování celé stránky. Do tohoto stromu jsou již zavedeny změny, které mohly provést Javascript soubory. Následně prohlížeč sestaví rozložení stránky a tu vykreslí [18]. Jak se daná stránka bude zobrazovat, je ovlivněno nejen používaným zařízením, ale i zvoleným webovým prohlížečem. Každý prohlížeč používá vlastní implementaci zpracování přijatých souborů, a proto mohou některé aplikace fungovat jinak na různých prohlížečích.

HTML

Hypertext Markup Language je značkovací jazyk, který slouží k určení struktury zobrazované stránky. Aktuálně používaní verze se nazývá HTML Living Standard. Ukázka jednoduchého HTML souboru ukazuje výpis 2.1.

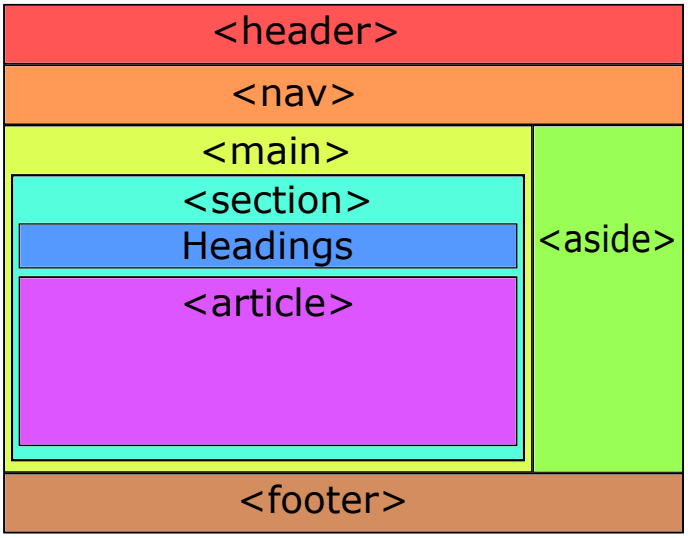
```
<!doctype html>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <p id="paragraph">Hello World</p>
    </br>
  </body>
</html>
```

Výpis 2.1: Ukázka HTML kódu

Každý dokument HTML se skládá z prvků, které jsou ohraničeny počáteční a uzavírací značkou. Uvnitř prvku je pak jeho obsah a navíc můžou být v počáteční značce uvedeny atributy daného prvku. Jednotlivé prvky lze vnořovat, pak je ale potřeba zachovávat pořadí počáteční a koncových značek. Některé prvky také nemusí mít koncovou značku, nebo jsou blokové, takže se před a za nimi zalamují řádky.

Soubor začíná úvodním prvkem <!doctype html>, který se používá z historických důvodů. Následuje počáteční značka <html>. Tento prvek tvoří kořen stromu prvků, zastřešuje veškerý obsah stránky. Do prvku <head> se přidává vše, co se nemá na stránce zobrazovat. V příkladě se v něm nachází prvek <title>, který upravuje text zobrazovaný v liště prohlížeče. Další prvky, které zde často najdeme jsou prvky meta, link nebo script. Meta nese informace o dokumentu například jeho kódování, link odkazuje na externí soubor nejčastěji soubor se styly nebo ikonu stránky a script slouží k načtení kódu ze souboru. Script není vázán pouze na prvek head, jako oba předchozí, nýbrž ho lze volat i z prvku <body>. Ten obsahuje veškerý obsah, který se má uživateli zobrazit jako text, obrázky či videa.

Jak vypadá běžná doporučená struktura prvku body ukazuje obrázek 2.3. V horní části stránky bývá umístěn prvek <header>. Tento prvek často zaplňuje celou šíři stránky, obsahuje logo či nadpis a zůstává stejný v různých částech stránky. Dalším prvek bývá <navbar> neboli lišta navigace stránky. Někdy bývá součástí předchozího prvku. Slouží k odkazování se na jednotlivé části stránky za pomoci tlačítek či odkazových prvků. Stejně jako header se při procházení stránky moc nemění, aby bylo odkazování konsistentní a nemátlo uživatele.



Obrázek 2.3: Příklad rozložení prvků stránky

Hlavní obsah stránky je poté obsažen v prvku <main>. Tento prvek by měl obsahovat obsah unikátní pro danou stránku, měl by být umístěn na střed stránky a předpokládá se, že bude zabírat nejvíce místa na stránce. Tento prvek lze dále členit pomocí prvků <section> a <article>, pro rozdělení na menší části. Vedle hlavního prvku se poté může nacházet prvek <aside>, který obsahuje informace vztahující se k hlavnímu obsahu stránky jako například odkazy na nejnovější obsah na stránce. Posledním prvek stránky by měl být <footer> neboli patička, kde se uvádí informace jako copyright, odkazy na sociální sítě nebo jiné odkazy. Tyto prvky jsou hlavně sémantické a mají spíše pomoci strukturovat psaní HTML dokumentu. Využít se však dají při následných úpravách vzhledu dokumentu [11].

CSS

Tato podkapitola vychází z [8]. Cascading Style Sheets zkráceně CSS je jazyk, kterým se upravuje vzhled stránky. Pomocí něj lze měnit rozložení jednotlivých prvků, kolik místa na obrazovce budou zabírat nebo měnit vlastnosti písma stránky. Stylizování funguje na základě pravidel definovaných v souboru se styly. Jak takový soubor může vypadat ukazuje výpis 2.2.

```
body {
    background-color: grey;
    color: #000000;
    width: 100%;
    margin: 0;
    padding: 0 0 0 0;
    font-size: 12px;
}

#main {
    color: #00FF00;
}
```

```
#main .inner-div {
    font-size: 14px;
}
```

Výpis 2.2: Ukázka CSS kódu

Pravidlo začíná selektorem, který určuje cílový prvek stylu. Následuje seznam měněných vlastností uvnitř hranatých závorek, kde upravované položky mají svůj název vlastnosti a hodnotu oddělenou dvojtečkou, přičemž jednotlivé dvojice se oddělují středníkem. CSS nemá jako HTML ucelený standard, ale je rozdělené do modulů, které jsou sdružené podle upravovaných vlastností. Souhrnně se aktuálně používané moduly označují jako CSS3. Pravidla pro stylizování se dají k HTML připojit 3 způsoby

- Externě – k HTML připojíme soubor CSS pomocí prvku <link> uvnitř prvku <head>
- Interně – CSS pravidla budou obsahem prvku <style>, tento prvek se umisťuje také to prvku <head>
- Inline – vzhled vybraného prvku se upraví pomocí atributu style, kdy pravidla píšeme jako hodnotu atributu, například <p style="color:blue;»

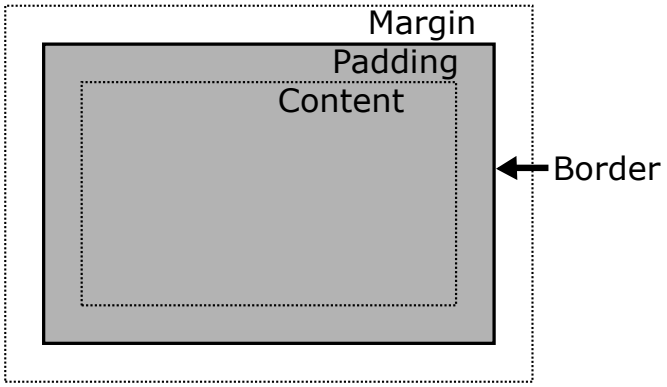
Doporučeno je používat první variantu, protože tak oddělíme úpravu vzhled od obsahu. Další důležitou částí jsou selektory. Selektor označuje HTML prvky, na které se dané pravidlo použije. Přitom je potřeba dbát na to, v jaké pořadí jsou pravidla uvedena, protože jednotlivé selektory se mohou překrývat a také jak specifický daný selektor je.

Když se vrátíme k výpisu 2.2, tak uvidíme, že obsahuje 3 pravidla. První pravidlo má selektor body, tedy vlastnosti tohoto pravidla se použijí na prvek <body> a všechny prvky uvnitř něj. To ukazuje dědičnost pravidla. Následující pravidlo má selektor #main. Toto pravidlo vybere prvek, který má ve svém atributu id uvedenu hodnotu main a nastaví jeho barvu textu na zelenou. První pravidlo nastavovalo prvkům barvu textu na černou, ale následující pravidlo ji pro vybraný prvek a jeho potomky změnilo na zelenou. Jedná se o specifičnost selektoru, kde pokud vlastnost upravuje více pravidel, tak se použije to, jehož selektor je specifičtější. Třetí pravidlo zobrazuje ještě víc specifičtější selektor a vybírá prvek, který má atribut class s hodnotou inner-div a přitom je obsažen v prvku s atributem id s hodnotou main. Jedná se o jednoduchý příklad skládání selektorů.

Kromě běžných pravidel začínajících selektorem existují ještě pravidla začínající zavináčem. Ty upravují zpracování CSS. Příkladem může být pravidlo @media, které slouží filtrování použitých CSS pravidel na základě toho, zda uživatelské zařízení splňuje požadované vlastnosti nebo má požadované parametry. Příklad použití ukazuje výpis 2.3.

```
body {
    color: blue;
}
@media screen and (max-width: 1024px) {
    body {
        color: green;
    }
}
```

Výpis 2.3: Ukázka použití @media



Obrázek 2.4: Stavba CSS boxu

Pravidlo @media zde funguje tak, že pokud danou stránku budeme zobrazovat na obrazovce a její šířka rozlišení nebude větší než 1024 pixelů, tak bude barva textu unitř prvku body zelená. Pokud dané zařízení bude mít větší obrazovku, tak dané pravidlo nepoužije a text bude modrý, jak uvádí první pravidlo. Toto pravidlo se používá pro stylizování stránek tak, aby se dobře zobrazovala na mobilních zařízeních i osobních počítačích. Dále je důležité zmínit CSS Box Model. Každý prvek má okolo sebe box. Tento box ovlivňuje, jak se bude prvek chovat v rámci rozložení stránky například vůči ostatním prvkům. Z čeho se takový box skládá se nachází na obrázku 2.4. Tvoří ho obsah prvku – content, výplň – padding, ohraničení – border a okraj – margin. Tyto části ovlivňují kolik místa bude prvek na obrazovce zaujímat. Výška a šířka prvku se vypočítá jako součet obsahu prvku, výplně a ohraničení po obou stranách obsahu. Existuje i alternativní model, který počítá rozměr prvků jinak, stále však pracuje se stejnými položkami. Jaký typ boxu prvek používá ovlivňuje vlastnost display. Tato vlastnost však také slouží k určení rozložení vnitřních prvků. Hodnota display tak může nabývat například hodnot:

1. Vnější rozložení
    - block - před a za prvkem bude konec řádku
    - inline - další prvek bude na stejném řádku, pokud bude mít dostatek místa
2. Vnitřní rozložení
    - flex - vnější chování jako block, vnitřní rozloženi podle flex modelu
    - grid - vnější chování jako block, vnitřní rozloženi podle grid modelu

Když se zaměříme na vnitřní rozložení prvku, tak jsou nejpoužívanější rozložení Flexbox a Grid. Flexbox udává potomkům, kterým směrem se mají rozložit neboli rozkládá prvky v jedné dimenzi buď horizontálně nebo vertikálně. Lze měnit směr rozložení, rozestupy mezi prvky, zda se mají prvky zvětšovat nebo zmenšovat dle potřeby a zda se mají prvky zalamovat na konci řádku. Díky tomu lze prvky jednoduše rozložit do řádků či sloupců. Pokud potřebujeme komplexnější rozklad může použít rozložení Grid, které z daný rodičovský prvek rozdělí na sloupce a řádky o zvolené velikosti, potomci jsou poté do mřížky dle vlastností naskládáni. Toto rozložení je podobné rozložení tabulkou, ale dovoluje větší volnost, protože potomci se mohou překrývat. Lze tedy upravovat kolik sloupců či řádků potomek zabírá, a tak docílit různých typů rozložení.

Javascript

Obsah podkapitoly je převzán z [14]. Javascript je skriptovací jazyk, který slouží k implementaci dynamického chování stránky. To dělá tak, že využívá rozhraní dostupných v prohlížeči díky, kterým může upravovat HTML a CSS kód stránky, získat geografické informace o zařízení nebo přístup k určitým jeho částem jako mikrofon nebo webkamera, a spoustu dalších věcí. Kód je spouštěn v takovém pořadí, v jakém je prohlížeč načte, tedy od shora dolů. Je tedy důležité pořadí jednotlivých příkazů. Javascript je interpretovaný jazyk, dnes je však spíše kompilovaný. Překlad se provádí těsně před jeho během. Tato podkapitola se věnuje klientskému Javascriptu, tedy kódu, který běží jen u klienta. Existuje i však serverový Javascript, který bude zmíněn v serverové části.

Stejně jako CSS lze JS k HTML připojit externě nebo interně. U obou se používá stejný prvek <script>, ovšem pro interní použití se kód píše dovnitř tohoto prvku, kdežto externě se zdroj uvádí jako hodnota atributu src. Tento prvek však nemusí být umístěn pouze v prvku <head>, ale může být i v prvku <body>. To je svázáno s tím, že úpravy provázené v JS jsou vázány na sestavení DOMu z HTML prvků. Může se tedy stát, že skript bude chtít upravovat část, která ještě nebyla vytvořena a jeho běh selže. Tomu lze zabránit přesunutím prvku skript na konec prvku <body> nebo použitím atributu defer v prvku <script>. Tento atribut způsobuje, že všechny skripty s tímto atributem se spustím v takovém pořadí, v jakém byly načteny a že jejich provádění započne, jakmile je DOM sestaven. Existuje ještě atribut async, který se také používá s prvek <script>, který spustí daný skript, jakmile je celý stažen a přitom neblokuje vykreslování stránky. Tento atribut se používá například pro skripty, které nemají žádnou přímou závislost na DOM.

Nyní se podíváme na některé části tohoto jazyka. Proměnné jsou v JS označeny klíčovými slovy var nebo let. Rozdíl v nich je ten, že let byl definován v novější verzi standardu a opravuje některé problémy s var jako například to, že proměnnou s var lze deklarovat několikrát se stejným jménem. Pro definování konstanty se poté používá klíčové slovo const. Javascript je dynamický jazyk, takže proměnná není vázaná na jeden datový typ. Dostupné jsou tyto datové typy:

- Primitivní typy
  - undefined - nedefinovaná hodnota, definovaná proměnná bez deklarace
  - Boolean - pravdivostní hodnota
  - Number - čísla s/bez pohyblivé čárky
  - String - řetězce
  - BigInt - velká čísla bez desetinné čárky
  - Symbol - neměnný řetězec s unikátní hodnotou
- Strukturální typy
  - Object - kolekce typu klíč–hodnota, pole a jiné komplexní typy jsou také objekty
  - Function - do proměnné lze uložit i funkce
- Speciální primitivní typ null

Datový typ proměnné nebo výrazu lze zjistit pomocí unárního operátoru typeof. JS také podporuje všechny základní operátory jako například:

- Aritmetické operátory – `+`,`-`,`*`,`/`,`%`,`**`

- Relační operátory – `<=`, `>=`, `<`, `>`, `in`, `instanceof`
- Unární operátory – `delete`, `void`, `typeof`, `+`, `-`, `~`, `!`
- Operátory rovnosti – `==`, `!=`, `===`, `!==`
- Bitové operátory - `<<`, `>>`, `>>>`, `&`, |, `^`
- Logické operátory - `&&`, ||, `??`
- Ternární operátor - `?:`
- Operátory přiřazení - `=`, `+=`, `-=`, aj.
- Operátory inkrementace/dekrementeace - `++`, `--`

Operátory, které jsou specifické pro JS jsou například:

- delete – odebírá vlastnost z objektu
- in – určuje, zda má objekt danou vlastnost
- instanceof – určuje, zda je objekt instancí jiného objektu
- void – vyhodnotí výraz a vratí undefined
- ?? – logický operátor, který vrátí pravý operand, pokud je levý operand null nebo undefined

Nakonec je ještě potřeba zmínit rozdíl mezi operátory == a ===. Oba slouží k porovnání rovnosti, ovšem první z nich kontroluje pouze rovnost operandů, kdežto druhý kontroluje i jejich typ. Je tedy doporučené používat spíše operátor ===.

JS podporuje obvyklé řídící konstrukce. Kód lze tedy možné uzavřít do bloku pomocí hranatých závorek, větvit pomocí příkazů if()...else if()...else..., cyklit příkazy while()..., do...while() nebo for().... For cyklus podporuje jak zápis for(inicializátor;podmínka;inkrement), tak i zápis for(proměnná in objekt) a for(proměnná of objekt). For..in prochází všechny vyčíslitelné vlastnosti objektu, zatím for..of funguje pouze na iterovatelné objekty jako pole. Dostupné je také příkaz switch, break a continue. Jednotlivé case větve příkazu switch se ukončují příkazem break.

Funkce v JS se deklarují klíčovým slovem function. Taková funkce má své jméno, za ním následuje závorka s případnými argumenty funkce a tělo funkce v hranatých závorkách. JS dovoluje definovat anonymní funkce, takové funkce nemají jméno a často se používají jako odpověď na nějakou událost prohlížeče. Anonymní funkce lze definovat i bez slova function pomocí konstrukce (argumenty)=>tělo funkce. Všechny funkce mají svůj rozsah, takže proměnné definované uvnitř funkce nebudou mimo ni dostupné, a také zastíní rozsah globální proměnné. I když jsou funkce vedeny jako samostatný typ, jedná se stále o objekty.

Objekt se v JS vytváří pomocí složených závorek, do kterých se zapisují jednotlivé vlastnosti a metody objektu jako dvojice klíč a hodnota oddělené dvojtečkou. Jednotlivé vlastnosti nebo metody jsou pak odděleny čárkou. Klíč může být pouze datový typ String nebo Symbol, hodnota může být cokoliv. Prázdný objekt lze také vytvořit voláním new Object(), kde Object je vestavěná třída stejného datového typu. Pokud bude objekt vytvářet na základě jiného objektu použije se také volání new nebo metoda Object.create(). K hodnotě vlastnosti objektu se přistupuje přes tečku nebo pomocí hranatých závorek,

a klíče vlastnosti. Pro změnu hodnoty vlastnosti se k vlastnosti přistoupí a přiřadí se do ní požadovaná hodnota. Tyto funkce provádí virtuální vlastnosti get a set. Ty lze pro danou vlastnost i měnit. Pokud metoda objektu potřebuje pracovat s nějakou jeho vlastností, může k ní přistoupit pomocí klíčového slova this. To v kontextu daného objektu značí právě danou instanci objektu.

Javascript je založen na prototypech. Každý objekt dědí vlastnosti z nějakého určitého objektu, kterému se říká prototyp. Prototypy zde tedy zastupují funkci tříd z třídních objektově orientovaných jazyků. Prototyp objektu se dá změnit pomocí metody Object.setPrototypeOf(). Javascript obsahuje klíčové slovo class, které však slouží pro definování šablon pro tvorbu objektů, jedná se tedy spíše o speciální druh funkce. Podporován je i systém výjimek, který funguje stejně jako v jiných jazycích. Je zde příkaz throw a blok try...catch()...finally. Za try musí být alespoň jeden blok catch nebo finally.

Poslední částí, kterou je potřeba zmínit jsou asynchronní operace v Javascriptu. Vykonávání příkazů v JS je synchronní, takže jdou postupně za sebou přičemž je vykonává pouze jedno vlákno. Při vykonávání časově náročné operace jako stahování vzdálených zdrojů by tato operace blokovala běh tohoto vlákna, dokud by neskončila. Proto se tyto operace vykonávají asynchronně, tudíž neblokují práci hlavního vlákna. K asynchronním operacím se váže objekt Promise, který představuje výsledek dané asynchronní operace a její návratovou hodnotu. Asynchronní funkci definujeme pomocí klíčových slov async function, taková funkce pak vrací právě objekt Promise. Pro zpracování výsledku z Promise se používá blok .then(). Pokud máme asynchronní funkci, ve které se vykonává nějaká Promise, tak na její výsledek můžeme počkat za pomoci klíčového slova await.

Asynchronní funkce se také používají na zpracování událostí prohlížeče. Každá taková událost má svého správce, kterému lze přiřadit nějakou funkci, která se má vykonat, jakmile tato událost nastane. Příkladem může být událost kliknutí myši na tlačítko stránky. Když tato událost nastane, tak pokud má správce daného tlačítka přidělenou nějakou funkci, která se označuje callback, která se má provést, tak ji v tom okamžiku provede. Takové funkce jsou správci předávány jako argumenty jiné funkce, v tomto případě jako argument metody addEventListener(). Vykonání této funkce je asynchronní, protože není řízeno pozicí ve zdrojovém kódu, ale pouze na tom, zda daná událost nastala.

Poslední možností, jak vykonat něco asynchronně, jsou funkce setTimeout a setInterval. Ty jako svůj argument berou funkci, která se má vykonat za určitý čas nebo opakovaně v nějakém intervale. Jelikož je jazyk JS rozsáhlý a některé problémy či úkony se řeší často, tak se často při implementaci funkcí používají knihovny a frameworky, jejichž využitím lze urychlit řešení těchto požadavků. Mezi nejpoužívanější patří React, Angular, Vue.js nebo JQuery. Další zajímavou možností je spojení Javascriptu a WebAssembly, což je kód, který vznikl přeložením z jiného programovacího jazyku například C++. Tento kód pak lze volat skrze JS a využívat jeho funkcionalitu.

2.1.2 Server-side

Tato část aplikace je stejně důležitá jako klienstká část, protože i když ji uživatel nevidí, tak slouží k mnoha důležitým věcem. Mezi hlavní funkce a vlastnosti patří:

- Efektivní ukládání a doručování informací – ukládání dat do databází, dynamická konstrukce HTML obsahu na základě aktuálních dat
- Přizpůsobení obsahu dle uživatele – poslední navštívené části aplikace, cílená reklama

- Řízení přístupu k obsahu – autentizace a autorizace uživatelů, omezování, co daný uživatel může v aplikaci vidět
- Ukládání stavových informací – udržování informací o uživateli, který právě prohlíží stránku, možnost zasílat různé odpovědi na základě tohoto stavu
- Oznámení a komunikace s uživatelem – upozornění či zaslání emailu, SMS při určité události například cizí přihlášení
- Analýza dat – sbírání a ukládání dat od uživatelů o používání aplikace pro její vylepšení

Stavba serverové části byla ukázána na obrázku 2.1. Je tvořena webovým server, aplikačním server a databází. Každá aplikace nemusí využívat databázi nebo aplikační server, tím však aplikace ztratí určitou funkcionalitu. Jednotlivé části aplikace se nemusí vyskytovat na jednom místě, můžou se nacházet na různých zařízeních, případně může jedno zařízení provozovat všechny tyto části, záleží na velikosti a funkcionalitě aplikace. Díky tomu lze aplikaci provozovat buď na vlastních zařízeních, nebo využít firem, které vám zařízení propůjčí nebo budou poskytovat přístup k vaší aplikaci na jejich zařízeních. Opět záleží, jaké funkce bude daná aplikace mít a jaké technologie bude využívat. Mezi firmy poskytující tyto služby patří Amazon, Google, Microsoft nebo IBM [13].

Web server

Podkapitola popisuje [15] a [12]. Web server je označení pro program nebo počítač na kterém daný program běží, jehož úkolem je odpovídat na požadavky od klienta a zasílat mu požadovaná data. Komunikace probíhá pomocí protokolu HTTP celým jménem Hypertext Transport Protocol. Jedná se o bezstavový aplikační protokol pro přenos souborů mezi klientem a serverem, kde klientem bývá webový prohlížeč. Protokol pracuje v režimu žádost – odpověď, kde komunikaci zahajuje klient. Využívá při tom transportní protokol TCP, kde má přiřazeno číslo portu 80 nebo TCP spojení šifruje pomocí TLS, pak se mluví o protokolu HTTPS a ten využívá port číslo 443. Klient tedy zašle požadavek, který se skládá z:

- HTTP metody – označení akce, kterou má server provést
- cesty k požadovanému zdroji – uvádí se ve formátu URL
- verze používaného HTTP protokolu – existuje několik verzí protokolu
- volitelných HTTP hlaviček pro předání dodatečných informací serveru – přidáváním hlaviček lze rozšířit funkcionalitu protokolu
- zasílaných dat – typické pro metodu POST

Nejpoužívanější HTTP metodou je metoda GET, vyjadřující požadavek na daný zdroj. Pokud klient zasílá data z formuláře, tak k tomu používá metody PUT nebo POST. Ty se liší v tom, že požadavek PUT bude mít vždy stejný efekt při jakémkoliv počtu volání, zatímco POST může na serveru vyvolat změnu. PUT tedy daný zdroj vytvoří nebo přepíše, zatímco POST pouze zašle data. K těmto metodám je ještě blízká metoda PATCH, která zdroj částečně modifikuje. Dále máme metodu DELETE, pro smazání daného zdroje, metodu HEAD, která vrací podobnou odpověď jako GET, akorát tato odpověď neobsahuje požadovaný zdroj, pouze zaslané hlavičky a metodu OPTIONS, vracející seznam metod

dostupných pro daný zdroj. Posledními metodami jsou CONNECT, ta vytváří oboustrannou komunikační cestu k danému zdroji a TRACE používaná k debugování cesty ke zdroji. Odpověď od serveru poté obsahuje:

- verzi HTTP protokolu
- stavový kód odpovědi
- stavová zpráva
- HTTP hlavičky
- zaslaná data

Existuje několik verzí protokolu HTTP, nejpoužívanější jsou verze 1.1 a verze 2.0. Verze 1.1 posílá zprávy v textovém formátu, na rozdíl od předchozí verze 1.0, dovoluje zasílat jedním TCP spojením více požadavků, zdroje lze přenášet po částech a lze požádat o více zdrojů naráz. HTTP verze 2.0 posílá zprávy v binárním formátu, může paralelně zpracovávat více požadavků, hlavičky lze komprimovat a zprávy navíc ještě formátuje do rámců. Verze 2.0 tedy není zpětně kompatibilní s předchozí verzí a obě strany tedy musí podporovat stejný protokol. Stavové kódy mají vždy tři číslice a dělí se do následujících skupin s příklady:

- 1xx Informační – 101 změna HTTP protokolu
- 2xx Úspěšné – 200 úspěch, nejčastější odpověď
- 3xx Přesměrování – 301 zdroj byl přesunut jinam
- 4xx Chyby klienta – 404 zdroj nebyl nalezen
- 5xx Chyby serveru – 503 služba je nedostupná

Poslední částí, kterou je třeba zmínit jsou HTTP hlavičky. Jak již bylo řečeno slouží k předávání dodatečných informací. Hlavička se skládá z jména a hodnoty, které jsou odděleny dvojtečkou. Hlavičky lze rozdělit do skupin dle kontextu na:

- Obecné hlavičky - jak pro požadavek, tak pro odpověď
- Hlavičky požadavku - informace ohledně požadovaného zdroje nebo informace o klientovi
- Hlavičky odpovědi - informace o zasílaném zdroji
- Hlavičky entit - nese informace o zaslaném zdroji

Systém hlaviček umožňuje přidávat do protokolu novou funkcionalitu tím, že se přidají nové typy hlaviček, bez nutnosti zasahovat do struktury protokolu. V hlavičkách se například přenáší adresa serveru, druh a verze klienta, formát přenášeného zdroje, jeho velikost, podporované kódování na straně klienta a mnoho dalších informací. Mezi nejpoužívanější implementace webového serveru patří Nginx a Apache HTTP Server.

Aplikační server a databáze

Tato serverová část se stará o zajišťování funkcí v pozadí dané aplikace. Stará se o tvorbu souborů stránky, které poté doručí webový server. Tvorba může být závislá na datech z databáze, s kterými tato část pracuje. Jde tedy o hlavní logickou část dané aplikace. Tyto funkce můžou být implementovány v různých programovacích jazycích jako například PHP, ASP.NET, Python nebo Javascript. Často se pro vývoj této části používají frameworky, které slouží k ulehčení tvorby, údržby a rozšiřování dané aplikace, protože obsahují nástroje a knihovny, které řeší potřebnou funkcionalitu jako přístup k databázi, autentizace uživatelů, formátování výstupu nebo udržování stavu spojení klienta. Některé frameworky také obsahují vlastní implementace webových serverů, tudíž je možné tyto dvě části v serverové části sloučit a používat takzvaný web aplikační server. O stavbě této části rozhoduje programátor, stejně jako o podporovaných funkcích a použitých technologiích dle smyslu dané aplikace.

Databáze slouží k efektivnímu uchovávání dat aplikace. Databázi lze pospat jako kolekci souvisejících dat. S těmito daty poté nakládá databázový systém, který slouží k manipulaci s databází. Tento systém určuje, jak budou data uložena a jaké operace s nimi bude lze provádět. Databázové systémy lze například dělit na:

- Relační
- Objektové
- Objektově-relační
- NoSQL

Nejpoužívanější jsou relační a NoSQL systémy. Relační databázový systém je postaven na koncepci relací neboli tabulek, kde tabulka je množina n-tic a n-tice je seřazený seznam atributů entity. Sloupce tedy představují klíč a řádky hodnoty pro jednotlivé klíče. Tyto tabulky mají mezi sebou vztahy na základě určitých svých sloupců. Tento koncept používají systémy založené na jazyce SQL, který slouží k manipulaci s daty v těchto systémech. Mezi implementace tohoto modelu patří MySQL, PostgreSQL nebo SQLite.

Naproti tomu máme NoSQL systémy, které data sdružují různými způsoby. Patří sem databáze typu klíč–hodnota, dokumentové databáze nebo grafové databáze. Každý s těchto systémů řeší ukládání a práci s daty jinak. Jejich používání se často váže na zvolený programovací jazyk, kvůli integraci pomocných funkcí nebo knihoven, protože nepodporují dotazování jazykem SQL. K zástupcům patří MongoDB, Redis nebo Firebase. Výběr databáze by měl být vázán na to kolik dat budeme ukládat, jaký bude jejich typ, jak často se budou data měnit nebo který programovací jazyk používáme v serverové části [19].

2.2 Tvorba 3D animací

Tato podkapitola vychází z [1]. 3D animování je odvětví 3D grafiky, ve kterém je hlavním cílem transformace objektů uvnitř trojrozměrného prostoru, která při jeho vykreslení vyvolá pocit pohybu objektu. Používá se při tom počítačů a programů, určených pro jejich tvorbu. Využití 3D animací lze nalézt v:

- Zábavním průmyslu – filmy, počítačové hry, reklama
- Vědě – medicína, architektura, forenzní vědy

- Jiná odvětví – umění, rozšířená realita, mapování projekce

Největší rozvoj 3D animace přinesl zábavní průmysl, který se snaží využívané techniky a zařízení zdokonalovat, pro vytváření reálnějších a detailnějších výstupů. Tyto pokroky jsou také svázány s celkovým vývojem počítačů, kdy v dnešní době je možné pracovat na rozsáhlejších projektech díky vývoji jejich výkonnosti. Proces tvorby 3D animace zahrnuje několik fází, které lze rozdělit na:

- Návrh
- Modelování a texturování
- Rigging a animace
- Vykreslení výstupu a postprodukce

Délka trvání jednotlivých částí se liší dle druhu projektu. Pokud jde o vědecký projekt, tak bude pravděpodobně nutné strávit nejvíce času návrhem pro ověření všech faktů daného výzkumu. Projekt ze zábavního průmyslu zase bude klást stejnou prioritu na všechny fáze tvorby. V dalších částech této kapitoly jsou jednotlivé části detailněji představeny a popsány.

2.3 Návrh

Tato část předchází počátku samotné tvorby. Zahrnuje procesy plánování, projektování a zkoumání. Vstupem do této fáze je nějaký nápad nebo myšlenka, na jejímž základě se bude daný projekt budovat. Tento nápad je dále rozveden a na jeho základě se vytvoří prvotní detaily projektu. Pokud se jedná o film nebo hru, tak se zde tvoří prvky příběhu nebo scénáře. Také je důležité určit například:

- Kdo jsou hlavní postavy příběhu?
- Jaký mají konflikt?
- Pro koho je projekt určen?
- Za jakým účelem je tvořen?
- Co bude výsledným výstupem?

Pro lepší představu se zde také vytváří náčrty a obrázky zachycující daný nápad, které slouží pro jeho lepší porozumění a které lze dále využít při samotné tvorbě. Když budeme tvořit například postavu, tak tyto nákresy můžeme použít při modelování, abychom dosáhli požadovaného výstupu. Nákresy se tvoří také proto, že je lehčí upravit a pozměnit, než provádět změny na již vymodelovaném objektu. Pro větší projekty je součástí také vytvoření plánu produkce, tedy kolik času a zdrojů bude použito na jednotlivé části projektu. Na základě tématu a typu projektu je také potřeba začít zkoumat techniky a postupy tvorby či další potřebné informace, které při samotné tvorbě využijeme.

Modelovaní a texturování

Vše, co se bude vykreslovat je nejdříve potřeba vytvořit. Tvorba těchto objektů se nazývá modelování. Model je geometrická reprezentace povrchu daného objektu. Tvorba modelu může probíhat zcela od začátku nebo je založena na nějaké formě digitalizace například 3D skenu, která se dále upravuje. Pro reprezentaci modelu se nejčastěji používají polygony.

Polygon neboli mnohoúhelník je obrazec tvořený třemi a více vrcholy, které jsou spojeny úsečkami, kterým se říká hrany. Plocha takového mnohoúhelníku se slouží poté jako část výsledného modelu. Pro rychlejší výpočet této ploch mnohoúhelníků je doporučeno nepoužívat polygony s více než pěti vrcholy, a také proto, že během procesu animace se mohou neočekávaně deformovat tak, že je nelze vykreslit. Celý model je poté tvořen velkým množstvím polygonů, které tvoří takzvanou polygonovou síť. Operace, které lze s mnohoúhelníky dělat jsou například:

- Dělení – přidáním nové hrany vznikne nový polygon, používá se pro přidání detailu geometrie
- Vyhlazování – dělení polygonu s průměrováním úhlů
- Vytlačování – tvoří novou plochu, která je spojená s plochou původní
- Zkosení – vytvoří nové vrcholy pro zaoblení hrany
- Mazání, spojování a rozdělování
- Přesun, rotace, škálování

Hustota polygonové sítě je důležitá pro rychlost vykreslování. Detailnější model bude mít hustší polygonovou síť, tedy bude potřeba delší čas pro její zpracování při vykreslení a model bude zabírat více místa, protože je potřeba uložit více informací. Pokud bude hustota moc nízká, model nebude dostatečně detailní, může ztratit svůj tvar a nebude poznat, co má představovat. Proto je lepší mít představu jak moc detailní má model být ve výsledku být a postupovat tak při jeho modelování, než po dokončení modelu odebírat nadbytečné polygony a model nedopatřením zdeformovat. Existuje několik přístupů tvorby modelu:

- Tvorba od nuly – skládání modelu pomocí jednotlivých vrcholů a hran
- Modelování pomocí primitiv – model vychází z primitivního tvaru, který je minimálně upraven do výsledné podoby nebo model je vytvořen složením několika primitiv
- Box modeling – jako předchozí, ale výchozím primitivem je kostka a je možné provádět velké úpravy primitiva
- Boolean modeling – metoda, kde se na dva a více objektům použije boolovská funkce například sloučení nebo průnik a tím vzinká nový objekt
- Skenování – reálný objekt je naskenován a je vytvořena jeho geometrická reprezentace
- Digitální sochařství – polygonální síť je možné upravovat jako při modelování z hlíny tlačením, taháním a podobně

To, který přístup použít záleží na tvůrci modelu a na tom, zda je tato metoda jeho nástroji podporována. Poté, co je model vytvořen je třeba obarvit jeho povrch. Tento proces se označuje jako texturování. Textura je obrázek, kterým se pokryje povrch modelu a dodá

mu tím určitou vlastnost. Problém spočívá v tom, že model je trojrozměrný a obrázek je pouze dvojrozměrný, takže ho nejde jednoduše na model nanést. Proto je nejdříve potřeba provést UV mapování, což je operace, která daný povrch 3D modelu rozloží do 2D plochy. Toto rozložení je poté uloženo v souboru nazývaném UV mapa, podle kterého lze vytvořit texturu pro daný model. Při samotném pokrývání modelu se používá několik vrstev textur, čímž se povrchu dodá více vlastností a efektů. Vlastnosti, které lze ovlivňovat jsou například:

- Barva
- Průhlednost
- Odraz a lom světla
- Průsvitnost
- Záření
- Hrboly
- Zrcadlení

Rigging, animace

Po aplikaci textur je potřeba model rozpohybovat. Pohyb modelu je nějak potřeba ovládat, k čemuž se používá kostra zvaná rig. Ta se skládá z jednotlivých kostí, které reprezentují jednotlivé části modelu. Tyto kosti mají svou hierarchii, kde některé kosti jsou nadřazené jiným kostem. Když se poté pohne kost nadřazená, tak se sní pohnou i její potomci. Tomuto efektu se říká dopředná kinematika. Pohyb kostí, kde se rodičovská kost pohybuje na základě pohybu potomka, se nazývá inverzní kinematika.

Jakmile máme sestavenou kostru, je potřeba ji připojit k vytvořenému modelu. Tento proces se nazývá skinning, kdy k jednotlivým kostem přiřadíme vrcholy modelu, které mají ovlivňovat. Když se poté pohne s danou kostí, tak se dle daného pohybu model transformuje a zdeformuje. Přiřazení vrcholů poté hladké, kdy jeden vrchol může ovlivňovat více kostí nebo rigidní, kdy vrchol ovlivňuje pouze jedna kost. Další možností transformace modelu je například deformace mřížkou, kdy je model obklopen sítí vrcholů, která je připojená k modelu, a poté lze jednotlivými vrcholy sítě pohybovat a tím deformovat tvar modelu. Tyto techniky slouží k přípravě pro tvorbu komplexních animací, kdy probíhá několik transformací naráz. Pro jednoduché animace, kdy není potřeba deformace model je jejich použití nadbytečné.

Jakmile je model připravený, lze s ním začít tvořit animace. Animaci je série obrázků, které jsou od sebe mírně odlišné a které jsou zobrazeny v postupném pořadí a s dostatečnou rychlostí takovou, že je lze vnímat, že jsou v pohybu. Animace se tedy skládá z jednotlivých snímků, které lze rozdělit na snímky klíčové a snímky vyplňovací. Klíčové snímky vytváří animátor neboli člověk tvořící animace tím, že manipuluje s daným modelem do zvolené pózy. Jakmile je s výsledkem snímku spokojen, tak je daný snímek uložen, animátor přejde o několik snímků dopředu a začne tvořit následující klíčový snímek.

Vyplňovací snímky jsou poté vytvořeny počítačem na základě informací ze dvou klíčových snímků tak, že pozice modelu v prvním klíčovém snímku je změněna postupnou interpolací na pozici modelu z druhého klíčového snímku. Tyto vytvořené snímky poté může animátor upravit dle potřeby tím, že změní hodnoty parametrů interpolace nebo tím,

že z nich vytvoří snímky klíčové. Rozložení snímků je zobrazeno na časové ose editoru animací, kde jsou jednotlivé snímky očíslovány. Rychlost animace je ovlivňována tím, kolik snímků trvá a jaká bude výstupní snímková frekvence. Existuje několik přístupů pro tvorbu 3D animací:

- Animace pomocí klíčových snímků – postup zmíněný výše, animátor tvoří jednotlivé snímky animace deformováním modelu
  - Pose–to–pose – nejdříve se vytvoří klíčové snímky a pak se tvoří vyplňovací snímky
  - Straight–ahead – snímky jsou tvořeny postupně jak jdou za sebou v animaci
  - Hybridní – kombinace předchozích dvou metod, nejdříve se použije pose–to–pose a poté se použije metoda straight–ahead
  - Hierarchický – používá se pro cyklické animace začínající a končící na stejný snímek, animátor nejdříve u všech snímků upraví jednu část modelu, a poté pokračuje další částí, dokud není animace hotová
- Snímání pohybu – snímací systém kamer zachytí nějaký pohyb nebo akci, ze které vytvoří množinu trojrozměrných dat, které jsou poté počítačem zpracovány a aplikovány na kostru určitého modelu, což vyvolá deformaci modelu odpovídající snímané akci
- Procedurální – animace je softwarově generována, animátor upravuje parametry generování

Jakmile jsou všechny animace dokončeny je tvorba modelu hotová. Nyní lze buď daný model vyexportovat a použít v jiném programu nebo vykreslit v nějaké scéně. V takové scéně by se měli nacházet kamera, pro zachycení vzhledu scény a zdroj světla bez kterého by model nebyl vidět. Jako výstup vykreslení může být obrázek nebo video, podle toho, co má scéna zachycovat.

2.4 Babylon.js

Pro zobrazování grafiky na webu se používá několik různých HTML značek. Pro obrázky se používá `<img>` nebo `<picture>`, pro video je vyhrazena značka `<video>`, případně lze jak obrázek, tak video zobrazit přes prvky `<embed>` nebo `<object>`. Ty se také v minulosti využívali pro vkládání plug-inů jako například Java applety nebo Flash soubory. Tyto technologie se používali k interaktivní grafice na webu, ale v dnešní době jsou již zastaralé a moderní prohlížeče je již nepodporují a nahradili je technologiemi jinými. Pro vektorovou grafiku lze použít prvek `<svg>`, obecně je však pro grafiku používán prvek `<canvas>`, který podporuje jak 2D a 3D rastrovou, tak 2D vektorovou grafiku.[10]

Princip tvorby grafiky na webu

Jak může vykreslování jednoduchého obrazce vypadat ukazuje následující úryvek kódu. Nejdříve je potřeba prvek vytvořit, abychom s ním mohli pracovat. Při vytváření můžeme nastavit, jaká bude jeho velikost. Následná manipulace obsahu se provádí přes Javascript, za pomoci dostupných rozhraní, které využívají kontext tohoto prvku. Ten lze chápat jako kreslící plátno, na které se výsledek zobrazí. Kontext je také důležitý v tom, jaké části

zařízení se pro vykreslení použijí a které funkce lze při tvorbě využívat. Existuje několik různých kontextů, přičemž nejpoužívanějšími jsou 2d a webgl nebo webgl2. 2d kontext slouží pro vykreslování 2D grafiky jako jsou tvary, čáry, gradienty, text nebo obrázky, pro které má dostupné funkce přímo v rozhraní kontextu. Díky tomu je veškerá práce prováděna buď v Javascriptu, nebo lze kombinovat Javascript s WebAssembly. V ukázce je použit právě tento postup, který vykreslí modrý obdélník. [7]

```
<html>
<head>
    ...
</head>
<body>
<canvas height="600" id="mainCanvas" width="800"></canvas>
<script>
    const canvas = document.getElementById('mainCanvas');
    const ctx = canvas.getContext('2d');

    ctx.fillStyle = 'blue';
    ctx.fillRect(10, 10, 150, 100);
</script>
</body>
</html>
```

Výpis 2.4: Ukázka použití canvas

Rozdíl mezi 2d a webgl kontextem je ten, že WebGL je nízko-úrovňový grafický API standard navržený skupinou Khronos, která sdružuje různé technologické firmy a navrhuje různé standardy pro grafiku, virtuální realitu nebo paralelní programování. WebGL vychází ze specifikace OpenGL ES pro vestavěné systémy, kde některé části byli upraveny pro použití na webu ve spolupráci s Javascriptem. Existují dvě základní specifikace WebGL verze 1.0 a 2.0. V závislosti na používané verzi se liší i používaný kontext pro vykreslování. Princip práce WebGL je založen na vykreslování bodů, přímek a trojúhelníků na základě programátorem dodaného kódu, který běží na GPU daného zařízení. Nyní následuje zjednodušený popis, jak tento proces pracuje.

Na začátku procesu je seznam vrcholů, kde každý vrchol má svoje vlastnosti, které ho reprezentují jako pozice nebo barva. Tyto vrcholy reprezentují objekty, které budou nakonec vykresleny. Jednotlivé vrcholy vstupují do funkce nazývané vertex shader, jejímž hlavním úkolem je převést pozici vrcholu z původního souřadného systému objektu do souřadného systému scény. Konkrétně jde o transformaci do prostoru nazývaného clip space, který určuje, zda se daný vrchol ve výsledné scéně vykreslí nebo ne, na základě jeho pozice v prostoru scény, pozici kamery a jejím projekčním modelu. Pozice vrcholu v clip space musí spadat do intervalu od -1 do 1, jinak se daný vrchol zahodí a nebude vykreslen. Výstupem funkce jsou tedy jednotlivé body reprezentující primitiva jako přímky a trojúhelníky, které budou v dané scéně zobrazeny. Následně tyto primitiva projdou procesem rasterizace, který je převádí na množinu fragmentů. Fragment je rastrová část primitiva tvořená atributy získanými na základě interpolace atributů vrcholů daného primitiva. Takto vytvořené fragmenty poté vstupují do funkce nazývané fragment shader, která pro každý fragment počítá jeho výslednou barvu. Ta může být ovlivněna osvětlením scény nebo použitou texturou. Takto obarvené fragmenty je potřeba převést do 2D prostoru zobrazovacího plátna. Při této finální

části jsou vynechány například fragmenty, které nejsou vidět, protože se nachází za jiným objektem a tudíž ve scéně nejsou vidět. [9]

Příklad postupu vykreslování s využitím webgl2 kontextu ukazuje výpis 2.5. Začátek je podobný jako při využití kontextu canvas. Nejdříve je třeba získat ze zvoleného prvku canvas kontext, který slouží k přístupu k GPU zařízení. Následně je potřeba dodat již zmiňované shader funkce, protože ty tvoří programovatelnou část procesu vykreslování, a proto jejich chod musí specifikovat programátor. Tyto funkce se v Javascriptu ukládají do řetězců, přičemž jsou psané v jazyce GLSL, který vychází z jazyků C a C++. Na začátku funkce je třeba specifikovat, že se jedná o verzi 3.00, jinak by se použila implicitní verze 1.00 používaná ve WebGL1. V ukázce je vidět jednoduché příklady těchto funkcí, kde vertex shader by nastavil každému vrcholu stejnou pozici a velikost a fragment shader nastavuje všem fragmentům stejnou barvu. U první funkce je implicitně stanovena výstupní proměnná gl_Position, zatímco druhá funkce může mít libovolně zvolenou výstupní proměnnou. V tomto případě je to proměnná outColor. Také lze vidět, že ve fragment shaderu je třeba definovat přesnost pro proměnné, zatímco u vertex shaderu je tato možnost volitelná.

```
const canvas = document.querySelector('canvas');
const gl = canvas.getContext('webgl2');

const vsGLSL = `#version 300 es
void main() {
    gl_Position = vec4(0, 0, 0, 1);
    gl_PointSize = 100.0;
}
`;

const fsGLSL = `#version 300 es
precision highp float;
out vec4 outColor;
void main() {
    outColor = vec4(1, 0.5, 0, 1);
}
`;

const vertexShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertexShader, vsGLSL);
gl.compileShader(vertexShader);
if (!gl.getShaderParameter(vertexShader, gl.COMPILE_STATUS)) {
  throw new Error(gl.getShaderInfoLog(vertexShader))
};

const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragmentShader, fsGLSL);
gl.compileShader(fragmentShader);
if (!gl.getShaderParameter(fragmentShader, gl.COMPILE_STATUS)) {
  throw new Error(gl.getShaderInfoLog(fragmentShader))
};

const prg = gl.createProgram();
```

```
gl.attachShader(prg, vertexShader);
gl.attachShader(prg, fragmentShader);
gl.linkProgram(prg);
if (!gl.getProgramParameter(prg, gl.LINK_STATUS)) {
  throw new Error(gl.getProgramInfoLog(prg))
};

gl.useProgram(prg);
gl.drawArrays(gl.POINTS, 0, 1);
```

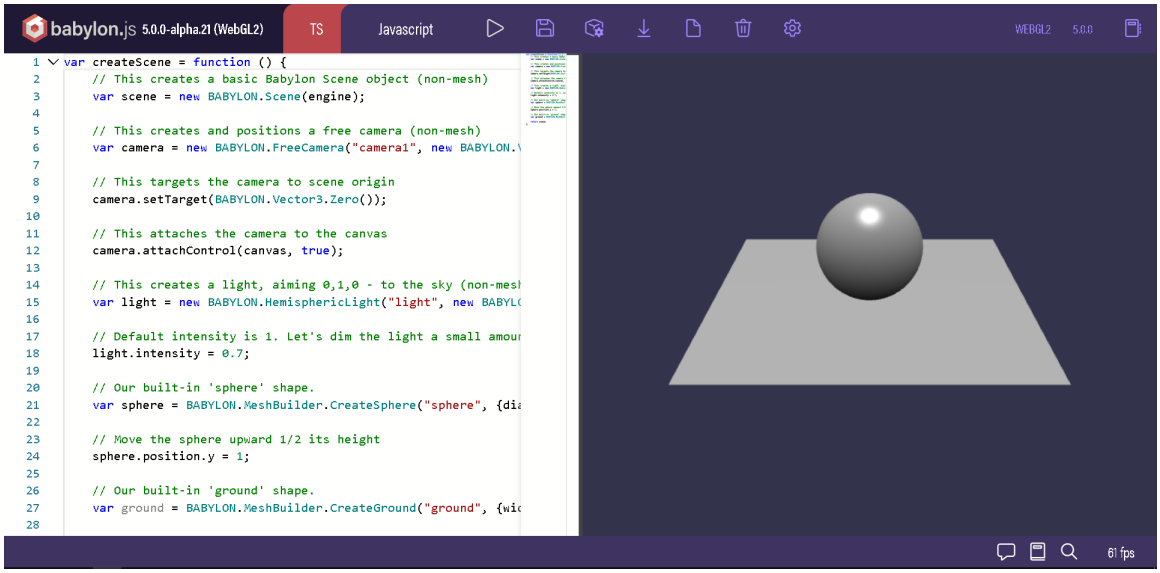
Výpis 2.5: Ukázka použití webgl kontextu

Shader funkce je dále třeba dostat na GPU. To se dělá právě za pomoci kontextu, kde nejdříve vytvoříme prázdný shader, poté specifikujeme kde se nachází jeho zdrojový kód a následně ho sestavíme. To uděláme s oběma funkcemi. Aby spolu dále mohli spolupracovat, musí se vytvořit program, který je spojí. Tento program se následně použije pro vykreslení. Kontext řekne GPU, aby použila daný program, na jehož základě vykreslí pomocí metody drawArrays jeden bod. Jelikož se jedná o jednoduchý příklad, tak zde chybí například to, jak lze do GPU dostat data potřebná pro výpočet. Ty lze předávat v podobě atributů, uniformních proměnných nebo textur. Atributy jsou vázány s buffery pomocí vertex array objektů neboli VAO, kde buffer obsahuje hodnoty daného atributu pro různé vrcholy, atribut specifikuje, jak tyto hodnoty předat shaderu a VAO udržuje informace o tomto spojení. Tímto způsobem je možné každému vrcholu přiřadit rozdílné hodnoty uvnitř shaderu. Pro sdílení stejné hodnoty mezi všemi vrcholy slouží uniformní proměnná. Tu lze použít i v rámci fragment shaderu na rozdíl od atributů, ty lze použít pouze v podobě varyings neboli proměnných atributů. Ty slouží k předávání hodnot mezi oběma shadery, kde výstupní atribut ve vertex shaderu je předán jako vstupní atribut do fragment shaderu. [20]

Struktura frameworku

Jelikož proces vykreslování za použití WebGL může být pro některé programátory příliš složitý nebo při tvorbě jejich projektu nepotřebují plnou kontrolu nad všemi jeho částmi, bylo vytvořeno mnoho knihoven a frameworků, které se o potřebné funkce starají a vývojář se tak může soustředit na tvorbu ostatních části aplikace. Jedním z těchto projektů je také Babylon.js, snažící se vytvořit výkonný engine pro vykreslování scéna na webu. Vznikl v roce 2013 jako vedlejší projekt několika pracovníků firmy Microsoft, který přerostl na plnohodnotný projekt, na němž pracují na plný úvazek. Jedná se o otevřený projekt, takže veškerý kód je volně dostupný a uživatelé do něj můžou přispívat.

Jeho hlavní části tvoří knihovna tříd pro tvorbu scén a sada podpůrných nástrojů pro ulehčení některých aspektů vývoje. Knihovna je psaná v jazyce Typescript, který se překládá do jazyka Javascript, s kterým již prohlížeč běžně pracuje. To také umožňuje využívat oba jazyky při tvorbě aplikací. Mezi podpůrné nástroje se řadí například Playground, Inspector nebo Node Material Editor. Playground je webová aplikace, která umožňuje spouštět kód napsaný s využitím knihovny bez potřeby tvorby stránek nebo souborů. Ukázku tohoto nástroje ukazuje obrázek 2.5. Lze vidět, že obrazovka je rozdělena na dvě hlavní části, a to na levou část s kódem a pravou část s výslednou scénou. Dále se v horní a spodní části nachází lišty s přepínači nebo odkazy. Horní lišta obsahuje přepínač pro změnu používaného jazyka, dále tlačítka pro běh kódu, jeho uložení a stažení nebo změnu nastavení aplikace. V pravém horním rohu jsou pak možnosti pro změnu kontextu, používané verze knihovny a složka s různými ukázkami použití knihovny. Při ukládání je uložena celá scéna

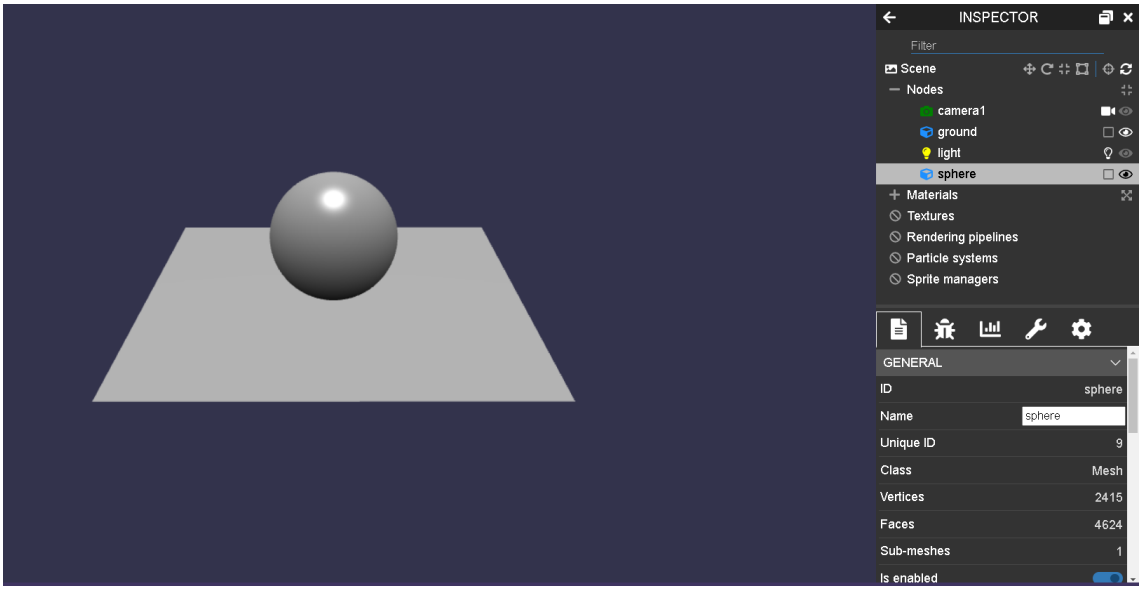


Obrázek 2.5: Nástroj Babylon.js Playground

do interní databáze aplikace, přičemž je vygenerován odkaz, díky kterému lze scénu dále sdílet. Při stažení projektu se scéna uloží do html souboru, který obsahuje vše potřebné pro její zobrazení, takže ji lze například přenést mezi různými zařízeními. Tento nástroj je užitečný v tom, že dovoluje vývoj pouze za pomoci prohlížeče a navíc dovoluje daný výsledek jednoduše sdílet. [6]

Dalším důležitým nástrojem je Inspector. Jedná se o ladící nástroj, který zobrazuje graf scény se všemi použitými zdroji společně s jejich vlastnostmi. Obrázek 2.6 ukazuje Inspector v rámci základní scény. Dělí se na dva panely, kde horní ukazuje prvky scény a spodní informace o právě vybraném prvku v horním panelu. Prvky scény jsou rozděleny do kategorií, navíc je lze filtrovat dle názvu, takže by mělo být jednoduché daný prvek najít. Spodní panel však neslouží pouze k zobrazování informací o prvku, ale také dovoluje některé vlastnosti prvku upravovat. To umožňuje například rychle měnit pozici objektu bez potřeby opakovaně spouštět celý kód programu pro zobrazení změny ve scéně. Dále lze také vypínat a zapínat jednotlivé části knihovny, což umožňuje identifikovat zdroj problému při vykreslování nebo zobrazit statistiky vykreslování scény, kde lze vidět počet snímků scény, celkové počty prvků scény nebo doba trvání jednotlivých částí procesu vykreslení. Navíc také obsahuje nástroje pro snímání a záznam obrazovky. [3]

Posledním nástrojem který je třeba vyzdvihnout je Node Material Editor. Materiály v Babylonu slouží k aplikaci barev a textur na objekty scény, jedná se tedy o shadery. Babylon obsahuje několik typů základních materiálů, pro vytvoření vlastních materiálů pak slouží Node Material. Jedná se o přizpůsobitelný materiál, jehož vlastnosti si může autor upravovat buď pomocí grafického editoru, anebo pomocí spojování bloků přímo v kódu. Node Material Editor tedy dovoluje uživateli tvořit vlastní shadery bez nutnosti umět GLSL. Jak prostředí tvorby vypadá ukazuje obrázek 2.7. Na levé straně je seznam dostupných bloků, které lze při tvorbě použít. Ty jsou rozděleny do kategorií dle jejich funkce. Uživatel může také definovat vlastní bloky. Hlavní část obrazovky tvoří plocha pro umisťování použitých bloků. Jednotlivé bloky mají svoje vstupy a výstupy, které mají svůj datový typ. Uživatel s bloky manipuluje pomocí myši. Po ploše se pohybuje tažením myši, lze ji přiblížit a oddálit. Na pravé straně se nachází panel s informacemi. Ten zobrazuje buď

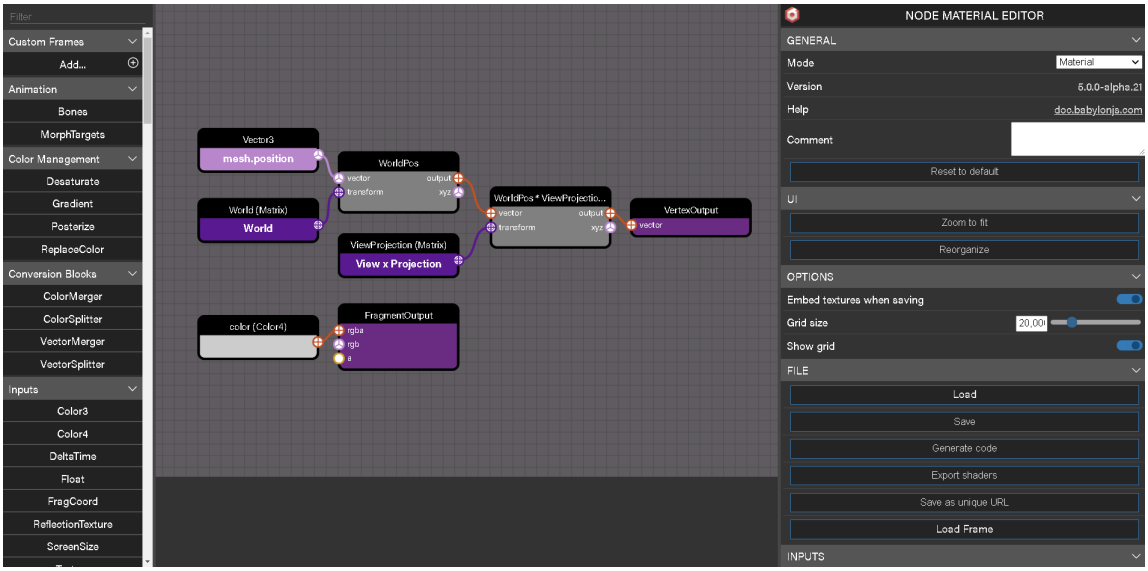


Obrázek 2.6: Nástroj Babylon.js Inspector

obecné funkce dostupné v rámci editoru, nebo informace o právě označeném bloku. Také se v tomto panelu nachází ukázka aktuálního výstupu shaderu, ten je však v ukázkovém obrázku skrytý. Výstup lze zobrazovat na různých tělesech včetně vlastních těles uživatele. Stejně jako u nástroje Playground je zde kladen důraz na přenositelnost. Je tedy možné výsledky ukládat a sdílet, případně lze na základě vytvořeného grafu bloků vygenerovat kód shaderu, a to jako v blokovém kódu pro Babylon, tak i v jazyce GLSL. Tato funkce tedy může pomoci i těm uživatelům, kteří plánují použít jiný framework nebo případně si vykreslování programovat sami. Navíc tento editor podporuje nejen tvorbu materiálů, ale i částicových systémů, procedurálních textur a post procesů. [4] S Babylonem jsou spojeny i další nástroje například vlastní editor scén. Ty jsou dostupné ze stránek projektu, případně jsou uvedeny v rámci jeho dokumentace.

Použití knihovny

Tato část popisuje základní práci s knihovnou Babylonu a podrobněji rozebírá některé její části. Podrobnější informace ke knihovně lze najít na stránkách její dokumentace, případně na GitHubu v repozitáři se zdrojovými kódy. Jak bylo zmíněno výše, tak zdrojový kód knihovny je psán v Typescriptu, což dovoluje její použití jak v tomto jazyku, tak po překladu v běžném Javascriptu. Jednotlivé funkce jsou členěné to tříd, ke kterým se v Javascriptu přistupuje přes globální proměnnou BABYLON. Jak vypadá základní postup pro vykreslení ukazuje výpis 2.6. Začátek procesu je totožný s předchozími příklady, nejdříve se získá prvek pro zobrazení výsledku. Následuje příkaz Babylonu, který vytváří instanci třídy Engine. Tato třída zajišťuje propojení s rozhraními prohlížeče jako WebGL nebo Audio, stará se tak například o kompilaci shader programu nebo vykreslovací smyčku. Zobrazované objekty jsou v rámci Babylonu sdružovány uvnitř scény, která je udržuje v kolekcích dle jejich typu. Obsahuje také metody pro získání objektů dle jejich identifikátorů. Scéna se také stará o zpracování akcí jako jsou stisky kláves či tlačítek myši nebo vyvolávání událostí na základě stavu procesu vykreslování. Při konstrukci je třeba specifikovat engine, který scénu bude



Obrázek 2.7: Nástroj Babylon.js Node Mateiral Editor

vykreslovat. Scény lze kombinovat nebo zobrazovat několik scén naráz, správu scén řídí programátor.

```
const canvas = document.getElementById("canvas");
const engine = new BABYLON.Engine(canvas, true);

const scene = new BABYLON.Scene(engine);

const camera = new BABYLON.UniversalCamera("camera",
                                    new BABYLON.Vector3(5, 1, 0));

camera.attachControl(canvas, true);
camera.setTarget(BABYLON.Vector3.Zero());

const light = new BABYLON.PointLight("light",
                                    new BABYLON.Vector3(1, 1, 0));

const box = BABYLON.MeshBuilder.CreateBox("box", {width: 2});
box.position = new BABYLON.Vector3(-2,1,2);

const ground = BABYLON.MeshBuilder.CreateGround("ground",
                                    {width:10, height:10});

const material = new BABYLON.StandardMaterial("material");
material.diffuseColor = new BABYLON.Color3(0, 1, 0);
box.material = material;

engine.runRenderLoop(function () {
    scene.render();
```

```
});
```

Výpis 2.6: Ukázka použití knihovny Babylon.js

Jakmile máme vytvořenou scénu, můžeme do ní přidávat objekty. Aby se vykreslila alespoň prázdná scéna, je třeba do ní přidat kameru. Ta slouží k označení místa, ze kterého je scéna zobrazována neboli určuje hodnoty použité pro výpočet pozice ve vertex shaderu. V Babylonu existuje několik druhů kamer, které se liší v tom, jak je lze ovládat nebo jaké mají zobrazovací vlastnosti. Nejpoužívanějšími kamerami jsou UniversalCamera a ArchRotateCamera. První z nich se používá například pro aplikace z první osoby, protože ji implicitně lze ovládat pomocí vstupů od uživatele a dokáže reagovat na kolize s objekty scény. Druhá se zase chová jako satelit a dokáže se otáčet kolem svého cíle. Další dostupné kamery jsou například FollowCamera, která následuje svůj cíl při jeho pohybu, VirtualJoystickCamera, přidávají na obrazovku 2D prvky pro ovládání jí nebo objektu scény a WebXRCamera pro VR scény. Všechny kamery nemají stejný konstruktor, takže definovat standardní postup pro přidání kamery. V ukázce přidávaná kamera přijímá v argumentech jméno kamery a její pozici v rámci scény. Následně se volají metody pro povolení přijímání uživatelského vstupu a nastavení jejího pohledu. Pro zobrazení scény však stačí pouze kameru vytvořit.

Nyní do prázdné scénu můžeme postupně umisťovat objekty a ty se v ní zobrazí. Aby objekty měli nějakou barvu, musí scéna obsahovat zdroj světla. Scéna může v implicitním nastavení obsahovat až 4 různá světla, každé světlo jde zapínat a vypínat, případně měnit jeho intenzitu. Barvu světla lze ovlivňovat pomocí dvou složek — difusní, spekulární. Difusní složka udává hlavní barvu světla, zatímco spekulární složka udává barvu při odrazu. Dostupné jsou 4 druhy světel, které se liší například v tom, jak světlo vyzařují. PointLight vyzařuje paprsky ve všech směrech od své pozice, dá se tedy přirovnat k žárovce. DirectionLight je světlo, které je udáno svým směrem a vyřazuje paprsky odkudkoli ve zvoleném směru s nekonečnou délkou. SpotLight vyzařuje paprsky jako kužel světla z určité pozice do určitého směru. Je také potřeba definovat úhel, který udává velikost paprsku a exponent útlumu světla. Poslední druhem je HemisphericLight sloužící pro simulování okolního světla. Toto světlo má svůj daný směr a lze u něj nastavit třetí složku zvanou groundColor. Jedná se o barvu používanou v opačném směru než je směr daného světla. Barvy světel se navzájem míchají, tudíž zkřížením modrého a červeného světla vznikne fialová barva. Světla v základu nevrhají stín. Pro zobrazování stínů se používá ShadowGenerator, který na základě zvoleného světla a objektů stíny vypočítá a zobrazí. Stíny však nelze tvořit vůči HemisphericLight.

Babylon poskytuje rozhraní pro tvorbu objektů na základě předdefinovaných sítí nebo uživatelských dat definovaných v kódu nebo načtených ze souboru. Třída MeshBuilder se využívá jako továrna na objekty. Zavoláním zvolené metody se ve scéně vytvoří daný objekt. Při vytváření objektu lze nastavit některé jeho vlastnosti. Ostatní vlastnosti jako například jeho poloha jdou upravovat až po jeho vytvoření. V příkladu je volána metoda CreateBox, která implicitně vytvoří krychli o rozměrech 1x1x1. Metoda také jako argumenty přijímá atributy vytvářeného objektu a scénu, ve které má být vytvořen. Všechny objekty jsou počátečně umístěny ve středu soustavy souřadnic. Pro přesun slouží atribut position. Implicitně jsou také všechny objekty obarveny stejnou základní barvou. V ukázce je také vytvářen objekt typu GroundMesh. Jedná se o vodorovnou rovinu, která v příkladu působí jako podklad a těleso se tak nevznášelo v prostoru. Pro načítání externích objektů slouží třída SceneLoader. Podporovány jsou formáty glTF, obj a stl.

Vložený objekt je třeba ještě obarvit. K tomu Babylon využívá materiály. Jak bylo zmíněno u jednoho z nástrojů, materiál dodává objektu barvu nebo na něj aplikuje texturu.

Vzhled objektu je také ovlivněn jeho nasvícením, přičemž lze ovlivňovat 4 složky reakce na světlo:

- Difusní — základní viditelná barva nebo textura
- Spekulární — barevný nádech daný osvětlením
- Emisivní — barva nebo textura tělesa, pokud samo světlo vyzařuje
- Ambientní — barva od okolního osvětlení

Několik objektů může používat stejný materiál. Jeden objekt také může tvořit více materiálů. Podporovány jsou také různé druhy textur a PBR materiálů, kde PBR znamená fyzikálně založené vykreslování, což je způsob stínovaní a vykreslování, který přesněji znázorňuje, jak světlo na daný povrch působí. Jednodušeji řešeno se jedná o model, který dovoluje simulovat chování materiálů z reálného světa ve scéně. Další způsoby stylizování objektů lze vytvořit za pomoci Node Materialů, které dovolují tvorbu vlastních shaderů. Nakonec je třeba enginu říct, kterou scénu má vykreslit. K tomu slouží metoda runRenderLoop, která očekává jako svůj argument funkci, která se má vykonat během vykreslovací smyčky. Do ní tedy umístíme volání metody render pro naši instanci scény a tím scénu zobrazíme. Přestože tento přehled pokryl pouze část toho, co Babylon může nabídnout, i tak by měl čtenář získat přehled o základních částech této knihovny a jaký mezi sebou mají vztah. Pro více informací lze navštívit stránky Babylonu, kde je dostupná dokumentace spolu s odkazy na další zdroje a projekty spojené s Babylonem [5].

Alternativy

Jelikož každý tvůrce má odlišné preference, tak je velice možné, že se najde někdo, komu Babylon v nějakém aspektu nevyhovuje a raději by zvolil jiný framework. Pro tvorbu grafiky na webu existuje mnoho různých frameworků a knihoven, je tedy na uživateli si najít to, co mu vyhovuje nejvíce. Zde jsou proto uvedeny alternativní frameworky, které stejně jako Babylon ulehčují práci s 3D grafikou na webu.

Three.js

Jedná se o nejpopulárnější knihovnu pro 3D grafiku na webu. Vznikla už v roce 2010 a její vývoj stále pokračuje. Pracuje na stejném principu jako Babylon. Je zde tedy hlavní prvek, který se stará o vykreslování, scéna se zobrazovanými objekty, světla, kamery, tělesa. Kód je psán v Javascriptu, přičemž jednotlivé části jsou členěny do tříd a funkcí, což zvyšuje přehlednost a udržovatelnost. Stejně jako u Babylonu se jedná o otevřený kód, takže do něj může kdokoliv přispět. Hlavními rozdíly je kromě jiné syntaxe to, že Three.js implicitně používá WebGL1, což ovšem není ke škodě, protože si tím drží podporu u starších prohlížečů, a také to, že mu chybí některé funkce jako má Babylon. To je způsobeno tím, že Three.js se snaží být hlavně o vykreslování, takže neimplementuje pomocné funkce pro vstup od uživatele nebo správu zdrojů. Mezi výhody zase patří to, že je populárnější, a tedy za jeho pomoci vzniklo více projektů a ukázek. I proto že nemá tolik funkcí může být snadnější ho používat, výkonově se Babylonu také vyrovná, takže jako alternativa k Babylonu jej lze určitě doporučit. [17]

PlayCanvas

Projekt PlayCanvas se prezentuje jako herní engine zaměřený na web. Soustředí se přitom na rychlost a kompatibilitu. Kód projektu je stejně jako Three.js otevřený a napsaný v Javascriptu. Jelikož se jedná o framework, jsou zde dostupné i podpůrné nástroje jako vlastní editor nebo hosting pro vytvořenou aplikaci. Uživatel se tak může soustředit na vývoj a nemusí řešit, kde na webu bude výsledek prezentovat. Princip práce má totožný s předchozími knihovnami. Přes hlavní objekt pc se přistupuje k jednotlivým třídám, opět zde engine, scény a další již zmiňované prvky. Podporovány jsou také funkce jako fyzika objektů, zpracování vstupů nebo integraci s Google Play Game Services a Facebook API. Nevýhodou může být menší popularita, ovšem dokumentace obsahuje dostatečný počet ukázek, což by pro začátek využívání frameworku mělo být dostatečné a dokumentace je také rozsáhlá. Pokud by tedy vývojář hledal alternativu k Babylonu, s podobným množstvím funkcí, PlayCanvas by určitě byla jedna z možností. [16]

A-Frame

Tento framework se od ostatních liší dvě základními aspekty. Jedním z nich je to, že se zaměřuje na 3D grafiku ve VR. Nabízí tedy podporu pro většinu dnešních VR zařízení a webových prohlížečů pro ně určených. Tím druhým je to, že používá jiný systém psaní kódu než předchozí možnosti. Místo toho, aby se kód aplikace psal v Javascriptu, používá se zde deklarativní HTML, což znamená, že se jednotlivé prvky umisťují přímo do HTML. Kód se tedy skládá z různých entit a komponent, přitom však vykonává stejný proces vykreslování jako ostatní knihovny. Fungování přitom není nějak omezeno a vývojář může používat Javascript společně s těmito prvky. To je dáno také tím, že framework je založen na knihovně Three.js a lze využívat také jejich funkcí pomocí definovaných rozhraní. Jako ostatní projekty je také otevřený. Mezi dostupné podpůrné nástroje patří Inspector pro náhled a úpravy scény nebo nástroj pro snímání pohybu, který umožňuje nahrát akce a pohyby uživatele a ty následně přehrát, což může pomoci při vývoji a testování. I díky těmto funkcím je tento framework ideální pro vývoj VR aplikací na webu a byl využit firmami jako Samsung, Toyota nebo Sony. [2]

Kapitola 3

Návrh a implementace

Tato kapitola popisuje návrh a implementaci hry, která má za cíl ukázat možnosti 3D grafiky v dnešních webových prohlížečích. Podkapitola návrhu je rozdělena na specifikaci požadavků, kde jsou popsány informace o tom, co je nutné během návrhu řešit, a také základní koncepce dané hry jako její žánr, vlastnosti nebo prostředí. Návrh poté ukazuje stavbu hry a jejich jednotlivých částí. Následuje podkapitola implementace rozebírající nejvýznamnější části a funkce vytvořené hry.

3.1 Návrh

Požadavky na výslednou hru lze rozdělit například na funkční a nefunkční. Do funkčních řadíme vše, co daná hra musí splňovat, co dělají její jednotlivé části nebo s jakými typy dat bude hra pracovat. Do nefunkčních patří volitelné požadavky na použitelnost, udržovatelnost nebo výkon. Jelikož zadání nespecifikuje, jaký žánr má mít vytvořená hra, tak jsem si jako cíl zvolil vytvořit akční bojovou hru pro více hráčů. Žánr bojových her se vyznačuje těmito vlastnostmi:

- Každý z hráčů ovládá svoji zvolenou postavu a snaží se pomocí různých útoků porazit protihráče tím, že mu udělí dostatečně velké poškození, čímž sníží jeho hodnotu zdraví pod hraniční hodnotu.
- Hry využívají 2D i 3D grafiku, přičemž pohled kamery zobrazuje scénu ze strany, čímž jsou postavy snímány z boku. Většinou je tedy jejich pohyb omezený na směry doleva, doprava, nahoru a dolů.
- Souboj je rozdělen na několik kol. Každé kolo má časový limit, po jeho vypršení vyhrává ten hráč, který udělil větší poškození. Souboj končí po vítězství hráče v určitém počtu kol.
- Komplexita těchto her bývá založena na stylu boje a sestavě hratelných postav. Každá postava se liší nejenom vzhledem, ale stylem útoků, které dokáže provést. V některých hrách postavy využívají zbraně, v jiných se například bojuje za pomoci speciálních schopností.
- Důležitou roli hraje také prostředí, ve kterém boj probíhá. Na jeho základě lze přidat jiné možnosti vítězství jako vytlačení soupeře ze zóny boje nebo přecházet mezi různými prostředími na základě akcí hráčů.

- Rozmanitost dodávají také různé herní režimy pro jednoho či více hráčů. Mezi nejdůležitější patří trénovací režim, kde hráč může zkoušet, jaké má postava útoky a vymýšlet strategie, jak je efektivně používat proti různým postavám. Dalšími režimy jsou například příběhový, turnajový nebo vlastní, kde si můžou hráči nastavit vlastní pravidla boje.

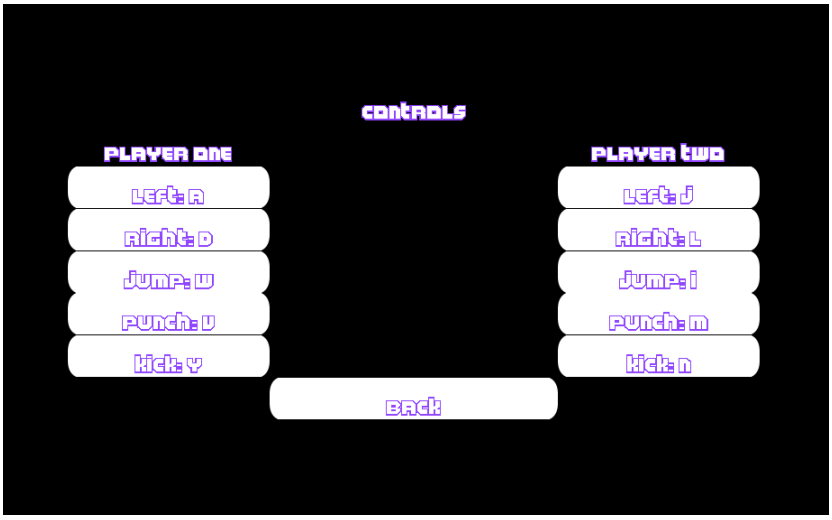
Na základě těchto vlastností jsem vybral požadavky na vytvářenou hru. Podle výše zmíněného dělení je lze rozdělit takto:

- Funkční
  - Hrají proti sobě dva hráči na stejném zařízení
  - Výběr postavy a prostředí
  - Soubojový režim s více koly a časovým limitem
  - Boj pomocí několika různých úderů nebo kopů
  - Načítání zdrojů z externích souborů
  - Možnost měnit ovládání postav
- Nefunkční
  - Podpora různých ovládacích zařízení
  - Konstantní výkon, rychlé načítání, celková velikost hry
  - Fungování na mobilních platformách
  - Možnost jednoduchého přidávání nových prostředí, postav nebo herních režimů

Zaměření na místní hru více hráčů umožní lepší počáteční vývoj, protože se tím lze vyhnout řešení problémům jako řešení komunikace a synchronizace klientů a serveru. Výběr z několika postav umožňuje hráčům si vybrat tu, která jim nejvíce vyhovuje, ať už vzhledově nebo z pohledu hratelnosti. To, že hráči musí vyhrát několik kol, dělá hru více napínavou, a také je více pravděpodobnější, že zvítězí lepší z hráčů. Časový limit zase nutí hráče k tomu, aby se snažil více útočit, pokud právě prohrává a aby souboj někdy skončil, protože se tak zamezuje nečinnosti hráčů. Více druhů útoku zase podporuje hledání vlastních metod jak hru hrát, a také hledání ideálních strategií. Rozdělení zdrojů do externích souborů přinese lepší možnost úprav bez nutnosti zásah do kódu. Základní nastavení ovládání postav nemusí všem vyhovovat, takže je dobré mít možnost provádět v něm změny.

Hráči bojových her často při hraní využívají speciální ovládače navržené právě pro tento typ her, a to i při hraní na stolních počítačích. Proto by bylo dobré podporovat také jiné druhy ovládacích zařízení než klávesnice a myš. Je zde také nutné klást důraz na to, aby se hra vykreslovala vždy se stejným počtem snímků, protože animace postav jsou tvořeny na základě cílového počtu snímků a při jejich kolísání budou animace fungovat jinak, do hry bude vneseno zpoždění a tím se zpozdí i reakce postav na příkazy hráče. Jelikož se jedná o hru v prohlížeči, tak i přes dnešní rozvoj rychlostí internetu, je třeba brát v potaz to, že hra se bude pokaždé stahovat a pro hráče s pomalým připojením může dlouhé stahování být jeden z důvodů, proč je hra nezaujme. Jelikož je většina her založena na komunitě lidí okolo, tak můžou funkce pro podporu a tvorbu tvořit zásadní prvek, který hráče zaujme. Tím, že hráči budou moci hru rozšířit vlastním obsahem, může vést k jejímu rozšíření k více lidem.

Obrázek 3.1: Návrh vzhledu hlavní nabídky



Obrázek 3.2: Návrh vzhledu nabídky nastavení

Když na základě těchto požadavků budeme hru navrhovat, tak je třeba řešit návrh audiovizuálních částí a vnitřní architektury hry. Do audiovizuálních částí patří uživatelské rozhraní, vzhled postav a prostředí nebo zvuky použité ve hře. Vnitřní architekturu hry lze dělit na data hry a řídící logiku. Při návrhu uživatelského rozhraní se řeší jednak vzhled nabídky hry, a také vzhled zobrazovacích prvků ve hře, případně dalších ovládacích prvků. Návrh vzhledu hlavní nabídky ukazuje obrázek 3.1. Ta je tvořena třemi hlavními tlačítky a pozadím. První spouští hru, druhé slouží pro přechod do nastavení a třetí pro zobrazení informací o hře. Obrázek 3.2 ukazuje jak vypadá nabídka s nastavením hry. Rozdělena je na dvě části, kde v každé z nich lze upravovat ovládání jednoho z hráčů. Rozhraní viditelné v boji je vyznačeno na obrázku 3.3. V horní částí jsou počítadla životů hráčů a časomíra. Co zde chybí je jména postav identifikující, které počítadlo patří každému z hráčů na základě zvolené postavy.

Návrh systému hry se dá popsat následovně. Hlavní část tvoří třída Game, která se stará s spojování jednotlivých částí a řídí celý cyklus hry. Třída Player reprezentuje postavu, kterou hráč ve hře ovládá. Tato třída se stará o pohyb postavy, změnu animací nebo kontrolu

Obrázek 3.3: Návrh vzhledu hry

zásahů do protivníka. Třída PlayerInput zpracovává vstupy od uživatele. Další důležitou třídou je Environment, která se stará o tvorbu prostředí bojiště. Třída ResourceManager spravuje externí zdroje a třída GUI řeší jednotlivé prvky uživatelského rozhraní. Hra bude pracovat převážně s audiovizuálními daty, součástí však budou i serializovaná data například s popisem prostředí.

3.2 Implementace

Hra je implementována v jazyce Typescript, přičemž k vykreslování využívá frameworku Babylon.js zmíněného v podkapitole 2.4. Jazyk Javascript byl použit pro implementaci serverové části, která běží za použití systému Node.js a modulu Express, který umožňuje vytvořit jednoduchý webový server, pro odesílání dat klientovi. Další použité moduly jsou:

- webpack - bundler, starající se o překlad kódu klienta v Typescriptu do Javascriptu
- webpack-dev-server - rozšíření předchozího modulu, které umožňuje tvorbu lokálního webového serveru
- ts-loader, file-loader - loadery pro webpack, umožňující zpracovávat další typy souborů
- htmlwebpackplugin - dovoluje specifikovat šablonu pro tvorbu výsledného HTML souboru
- babylonjs - knihovna frameworku pro tvorbu webové grafiky

Tyto moduly spolupracují na vytvoření výsledného souboru bundleName.js, který obsahuje veškerý kód pro spuštění hry. Proces vytvoření souboru probíhá tak, že webpack zpracuje zdrojové soubory s kódem pomocí loaderů, přitom také převezme funkce použité z knihovny Babylonu. Na svém výstupu poté vytvoří jednak Javascript soubor s daným přeloženým kódem, a také HTML soubor, který má odkaz na předchozí soubor. Tyto výsledky jsou poté klientům dostupné díky modulu Express, který daný HTML soubor použije jako svoji hlavní stránku. Pro fungování je tedy na serveru nutné mít nainstalován Node.js včetně správce

modulů npm. Poté lze v rámci kořenového adresáře s kódem hry zadat příkazy npm run build, npm run dev a npm run start. Build kód sestaví a start spustí v rámci modulu Express, zatímco dev spustí lokální webový server na port 8080. Instalace potřebných modulů proběhne automaticky na základě souboru závislostí package.json. Nyní následuje část věnující se fungování hry samotné.

3.2.1 Inicializace a hlavní nabídka hry

Počáteční operace provádí hlavní třída Game. Jelikož vytvořené HTML neobsahuje žádné prvky, tak je třeba vytvořit nový prvek canvas pro vykreslení hry. O tuto funkci se stará metoda createCanvas, která také nastaví potřebné styly, aby byl prvek zobrazen na celou velikost obrazovky. Následně se získá kontext tohoto prvku a vytvoří se prázdná scéna za pomoci příslušných tříd Babylonu. Následně se načtou všechny potřebné zdroje, které bude hra potřebovat. Tím se předejde dlouhému načítání během hry. Do těchto zdrojů patří modely postav, obrázky nebo zvuky hry. Při získávání zdrojů je zobrazena načítací obrazovka, která se odstraní, až je vše staženo a scéna je celá připravená. Do scény se totiž ještě musí umístit ovládací prvky uživatelského rozhraní. Tento problém lze řešit dvěma možnými způsoby. První možností je prvky tvořit programově a vkládat je do HTML struktury dynamicky. Tento způsob se hodí v případě, že bychom chtěli používat nějakou jinou pomocnou knihovnu, která upravuje jejich funkce nebo mění jejich vzhled. Zvolenému prvku bychom nastavili požadovaný identifikátor nebo třídu, a díky tomu by se jednoduše změnil jeho styl.

Druhou možností, která je právě ve hře použita, je využít Babylonu, jelikož poskytuje funkce pro tvorbu těchto prvků přímo ve vykreslované scéně. Ty jsou zobrazovány v rámci dynamické textury, která tvoří rodiče pro všechny prvky. Podporovány jsou všechny základní ovládací prvky jako tlačítka, zaškrtávací políčka nebo posuvníky, lze také měnit jejich vzhled nebo použitý font, takže většina očekávaných funkcí je dostupná. Po načtení zdrojů se tedy vytvoří hlavní nabídka a scéna se zobrazí. Z hlavní nabídky se dá hra spustit, upravit její nastavení nebo zobrazit informace o hře. Hlavní nabídka využívá jako kontejner na prvky třídu StackPanel, zatímco nabídka nastavení používá třídu Grid, která využívá mřížku pro rozložení prvků. Pozice prvku je tedy identifikována zvoleným řádkem a sloupcem. Rozhraní se také přizpůsobuje různým velikostem obrazovky.

Změna základního ovládání postav probíhá tak, že rozhraní obsahuje tlačítka se štítky, které identifikují daný pohyb postavy. Když bude chtít hráč změnit nastavení vybraného pohybu, klikne na příslušné tlačítko, čímž se vyvolá příslušná funkce a na obrazovce se objeví informační okno s tím, že má hráč zmáčknout cílovou klávesu, aby změnu provedl. Během toho je zablokovaná manipulace se všemi prvky. K odblokování po stisknutí zvolené klávesy. Systém ovládání je nastaven tak, že žádné dva pohyby nemůžou sdílet stejnou klávesu. Pokud tedy již nově zvolená klávesa obsluhuje nějaký pohyb, tak je vyměněna s klávesou, která zvolený pohyb před změnou řídila. Díky tomu je vždy jednoznačné, jaký příkaz má postava vykonat. Změny se ukládají ukládají ihned po provedení, takže je není třeba nijak potvrzovat a zůstávají uložené i po opuštění nabídky. Když se vrátíme do hlavní nabídky, tak kliknutím na příslušné tlačítko hru spustíme.

3.2.2 Příprava úrovně a postav

Nejdříve se zobrazí načítací obrazovka a následně se vytváří příslušné prostředí, kde se bude souboj odehrávat. Prostředí je tvořeno na základě popisného json souboru, který je tvořen objektem udržujícím informace o použitých primitivech a zdrojích. Prostředí je tak tvořeno

pomocí tříd dostupných v Babylonu pro tvorbu objektů a jejich obarvení. Babylon také podporuje serializaci scén, akorát k tomu využívá svůj vlastní formát s názvem babylon, který má však také formát typu json. Tento způsob konstrukce prostředí tak umožňuje zavést do hry libovolnou scénu vytvořenou v Babylonu, avšak s omezenými možnostmi použitých objektů nebo textur. Pokud by konstruktor využíval interní formát Babylonu, tak by byla podpora nejspíše úplná.

Po vytvoření prostředí je třeba vytvořit postavy ovládané hráči. Pro tvorbu postav jsem použil nástroje Blender, kde jsem vycházel z volně dostupných modelů, které jsem dle potřeby upravil. Modely jsou uloženy ve formát glb, což je binární verze formátu gltf pro ukládání scén a modelů. Ten jsem využil, protože výsledek pak měl nejmenší velikost, takže se zkrátí doba stahování. Model obsahuje kromě popisu vrcholů také textury pro jeho obarvení a kolekci svých animací. Všechny tyto informace jsou importovány do scény, takže s nimi lze dále manipulovat. Přidaný model je transformován na svou počáteční pozici ve scéně a je připraven na spojení s herní logikou. Ještě před propojením se do scény přidají světla a kamera. Využívány jsou dvě světla, přičemž jedno slouží ke generování stínů ve scéně a druhé zlepšuje osvětlení scény. Kamera je nastavena tak, aby ze své pozice zabírala scénu ze strany společně s postavami ovládanými hráči.

O ovládanou postavu se stará třída Player, která manipuluje s daným importovaným modelem. Při vytvoření se primárně připraví rozhraní pro ovládání animací modelu a aktivuje se animace postavy, kterou postava zaujímá, když nepřijala žádný příkaz. Další animace, která postava má jsou animace pohybu a útoků. Přepínání animací je řízeno stavovým automatem, který mění stavy na základě vstupu uživatelů a vnějších vlivů na postavu jako například obdržení zásahu. Po inicializaci postav se ještě zajistí, aby spolu mohli interagovat a propojí se zpracování vstupů od uživatele s danou postavou. Prohlížeč při každém stisknutí klávesy vyvolá událost, kterou lze zpracovat a podle zmáčknuté klávesy, pak vyvolat příslušnou akci. Jako poslední se do scény přidávají prvky pro zobrazení stavu hry jako počítadlo životů a časomíra. Jakmile je vše připraveno, tak se skryje načítací obrazovka a začíná samotný průběh hry.

3.2.3 Průběh boje

Na začátku jsou oba hráči postaveni proti sobě v určité vzdálenosti. Aby na sebe mohli útočit, tak se k sobě nejdříve musí přiblížit. Když se k sobě přiblíží příliš blízko, tak se jejich pohyb na danou stranu zastaví, protože dojde k jejich kolizi. Kolize postav jsou řešeny za pomoci systému dostupného v Babylonu, kdy je třeba povolit ve scéně detekci kolizí, a pak nastavit, u kterých objektů se má kontrola provádět. Tento princip však funguje korektně pouze s objekty vytvořenými pomocí funkcí pro tvorbu primitiv dostupných v Babylonu. Pro objekty přidané externě tento proces funguje pouze částečně. Objekty jsou při kolizích reprezentovány zástupným objektem, který má tvar elipsoidu. Tímto způsobem není výpočet kolizí tak náročný, jak při použití komplexních tvarů. Hlavním problémem u přidaných objektů je fakt, že dva přidané objekty mezi sebou nekolidují. Kolize mezi objektem přidaným a vytvořeným v Babylonu funguje v pořádku. Postava díky tomu nemůže procházet zdmi prostředí, avšak interakci postav bylo třeba řešit jiným způsobem.

Pro simulaci kolize postav byl využit jiný systém Babylonu konkrétně systém paprsků, který lze také použít pro kontrolu kolizí a průniku objektů. Z určitého místa ve scéně je vyslán paprsek, který projde scénou a zastaví při kontaktu s první překážkou nebo pokračuje dál dle typu použitého paprsku. Paprsek má svůj směr a délku. Poté lze definovat podmínku, která se vyhodnotí po kolizi paprsku a díky tomu provést nějakou akci. Když tedy bude

postava vysílat paprsky ve směru svého pohybu a paprsek narazí na druhou postavu, tak můžeme zamezit pohybu postavy v daném směru, a tím vyvolat dojem, že postavy spolu kolidují.

Dalším důležitým prvkem je reakce postav na zásah. Když postava provede nějakou útočnou animaci při níž zasáhne protivníka, tak je třeba reagovat tím, že se protivníkovi sníží počet životů. Řešit se při tomu musí jednak to, o jaký útok se jednalo a také, zda šlo o opravdový zásah. V bojových hrách se k tomu používají takzvané hitboxy a hurtboxy. Hitbox je neviditelný jednoduchý objekt reprezentující část modelu dané postavy, který se používá jako kontaktní bod mezi postavami. Hurtbox je také neviditelný objekt, avšak jeho hlavní funkcí je to, že představuje oblast nutnou k zasažení, aby postava obdržela poškození. Počítaní interakcí složitých modelu je výpočetně náročné, a proto se používají tyto zástupné objekty. Jednotlivé části modelu postavy jsou tedy při detekci zásahů nahrazeny tímto zjednodušeným modelem, a tím je výpočet možné provádět rychleji.

Ve hře je tento princip proveden tak, že součástí importovaného modelu jsou i hitboxy, které byli přidány při jeho tvorbě a které jsou s modelem spojeny tak, aby kopírovali pohyb celého modelu při provádění animací. Díky tomu lze tak jednoduše určit příslušný hitbox pro daný útok. Hurtbox je pak tvořen samotným importovaným modelem, protože je tím dosaženo větší přesnosti detekce. Důležité je také zmínit to, že animace útoku se skládá z několik částí a toho se musí řídit i samotná reakce. Řešení bývá takové, že během útoku je pouze několik snímků animace, kdy se kontroluje zásah vůči protivníkovi, protože nedává smysl, aby protivník obdržel poškození v době, kdy postava dokončila útok a vrací se klidové pozice.

Cyklus útoku je tedy následující. Nejdříve je přijat vstup od uživatele, ten se vyhodnotí a určí se, jestli je postava ve stavu ve kterém může zaútočit. Pokud ano, tak se spustí daná animace útoku. Ta probíhá do momentu, dokud útok nedovrší finálního bodu, kde by mělo dojít k zásahu oponenta. Během daného snímku se tedy provede kontrola toho, zda došlo k průniku hitboxu daného útoku s hurtboxem protivníka, což následně způsobí odebrání určitého počtu životů protivníka dle druhu provedeného útoku. Poté animace pokračuje, dokud útok neskončí a po jejím konci se postava vrací do klidového stavu a může útočit znovu. Když jedna z postava obdrží dostatečné poškození, tak hra končí a je ji třeba spustit od začátku kliknutí na zobrazené tlačítko.

Kapitola 4

Testování

Tato kapitola popisuje, jak byla vytvořená hra testována a jakých výsledků bylo při testování dosaženo. Testování lze rozdělit na statické a dynamické. Do statického můžeme zařadit kontrola kódu programu a externích souborů. Do dynamického testování patří kontrola sestavení a běhu programu, testování jednotlivých částí hry a měření výkonosti a funkčnosti na různých platformách. Jelikož je kód psán v Typescriptu, který podporuje na rozdíl od Javascriptu specifikaci typů pro atributy, funkce nebo proměnné, tak lze kód analyzovat během jeho psaní pomocí funkcích dostupných ve vývojovém prostředí, a tím zamezit například syntaktickým chybám. Díky tomu se při hledání chyb dá zaměřit přímo na chyby v algoritmu namísto hledání problému při volání funkce. Je třeba také zkontrolovat to, že všechny zdroje používané v kódu jsou na daném místě dostupné a že se jedná o jejich nejnovější verze. Čitelnost a strukturovanost kódu by mohla být lepší, protože zde nejsou dodržovány všechny doporučené zásady při psaní kódu. Členění do tříd rozděluje kód na funkční celky, avšak třídy většinou zastávají více funkcí, což zase zhoršuje jeho srozumitelnost. Použité textury by měli používat jednotný formát, nejlépe bezztrátový pro jejich lepší škálování. Modely postav nemají tolik detailů, což není až tak viditelné ze zobrazované vzdálenosti ve hře, ale bylo by dobré, aby přibyla možnost volby modelu dle požadované volby detailu zobrazení.

Když se zaměříme na překlad kódu, tak ten probíhá v pořádku, díky modulům zmíněným v předchozí kapitole. I díky tomu, že je použit Typescript a ne Javascript, se kód musí překládat a při tomto procesu lze některé chyby objevit a opravit, protože zabrání úspěšnému překladu. Fungování hry bylo vyzkoušeno v několika dnešních prohlížečích konkrétně tedy:

- Google Chrome verze 90.0.4430.212
- Microsoft Edge verze 90.0.818.62
- Mozilla Firefox verze 88.0.1
- Opera GX verze 75.0.3969.267
- Google Chrome pro mobily verze 90.0.4430.210

Ve všech těchto prohlížečích byla hra spustitelná, avšak na mobilních zařízení nebyla hratelná, protože nemá dodělanou podporu pro jejich ovládání. Bylo by dobré hru otestovat také v prohlížeči Safari, protože se jeho funkce a specifikace mírně liší od ostatních. Rozbor fungování hry poukázal na její nedostatky. Jednak jsou zde problémy se škálováním některých prvků při vysokých a nízkých rozlišeních, dále jsou zde problémy s animací postav,

GPU	Rozlišení	Nejnižší hodnota FPS	Stabilní hodnota FPS
intel hd graphics 510	800 X 480	58	60
intel hd graphics 510	1280 X 720	43	50
intel hd graphics 510	1920 X 1080	30	35
intel hd graphics 510	2560 X 1440	21	25
nvidia gtx 850m	800 X 480	60	60
nvidia gtx 850m	1280 X 720	60	60
nvidia gtx 850m	1920 X 1080	59	60
nvidia gtx 850m	2560 X 1440	58	60

Tabulka 4.1: Výsledky měření výkonnosti hry dle GPU a rozlišení

kdy některé animace na sebe plynule nenavazují, což způsobuje viditelné posuvy částí modelu a nepřirozenost pohybů. Mezi další nedostatky patří limitovaný počet možných útoků postavy, která je omezená pouze na několik základních pohybů a úderů. Posledním problémem implementace je nulová možnost volby postavy nebo prostředí, čímž je limitovaná znovuhratelnost hry.

Testování výkonnosti probíhalo za pomocí nástroje Babylonu nazývaného Inspector, který dovoluje monitorovat některé výkonnostní charakteristiky scény. Konkrétně se sledovala statistika snímků za sekundu na základě zvoleného rozlišení a použité GPU. Použit byl přitom prohlížeč Google Chrome, který dovoluje škálovat rozlišení zobrazované stránky. Výsledky ukazuje následující tabulka 4.1. Babylon má maximální počet vykreslených snímků za sekundu omezen na hodnotu 60.

Při pohledu na výsledky je patrné, že obě položky mají vliv na výkon procesu. Největší vliv na plynulost hry bude mít používaný GPU adaptér, který provádí vykreslování scény. Bylo by dobré také otestovat výkonnost na mobilních zařízení a výsledky porovnat s výsledky integrované grafické karty. Tomu zatím brání nedostatečná podpora mobilní platformy. Další možnou platformou pro testování můžou být herní konzole, ale pokus spustit hru na zařízení Xbox One v prohlížeči Microsoft Edge selhal, hra se nenačetla ani do úvodní obrazovky a zjišťování, kde nastala chyba je složité, protože v daném prostředí nejsou dostupné vývojářské nástroje.

Kapitola 5

Závěr

Cílem práce bylo vytvořit hru využívající 3D grafiku, která bude dostupná na webu a hratelná skrze webový prohlížeč. Nejdříve se bylo třeba seznámit s tím, jak se tvoří aplikace na internetu, z jakých částí se skládají a jaká je jejich funkce. Následně bylo také třeba nastudovat principy 3D animace a tvorby 3D grafiky na webu, jelikož bez toho by hra nešla vytvořit. Pro tvorbu hry byl použit framework Babylon.js, který mimo hlavní knihovnu pro tvorbu grafiky také nabízí velkou řadu podpůrných nástrojů pro zvýšení efektivity práce. Následně byla výsledná hra navržena, implementována a otestována na několika různých zařízeních.

Výsledná hra je funkční, avšak obsahem spíše minimalistická. Postrádá hlavně větší možnosti volby pro hráče ať už ze strany hratelnosti nebo uživatelské přívětivosti. Slouží tedy jako spíše ukázka některých technik při tvorbě webových her a základních herních principů. Hraje je momentálně dostupná na těchto webových stránkách[1] a její zdrojové kódy jsou dostupné na githubu[2]. Dále by měla být umístěna do galerie studentských prací, kde se uchovávají zajímavé ukázky ze studentské tvorby.

Jako další rozšíření projektu bych navrhoval přidání nového obsahu v podobě nových postav, prostředí a herních módů. Dále by bylo dobré, kdy se vylepšily základní mechaniky hry, protože v tuto chvíli fungují velice jednoduše. Také by bylo dobré zapracovat na vzhledu hry, hlavně v oblasti uživatelských prvků a přidat více možností ovládání hry. Posledním rozšířením by byla integrace hry po síti, což by umožnilo hru rozšířit mezi více hráčů. To je však limitováno tím, že hra byla převážně zamýšlena pro lokální hraní a tudíž by se museli řešit problémy spojené s konektivitou jednotlivých klientů nebo jejich synchronizace. Je také vhodné řešit otázky ohledně toho, jak by například narostla velikost hry, pokud by také podporovala hru po síti a jak by byla zaručena bezpečnost komunikace proti narušení ze strany hráčů.

[1] `https://fighter-dp.herokuapp.com/`
[2] `https://github.com/zionedge/fighter-dp`

Literatura

[1] Beane, A. *3D Animation Essentials*. 1. vyd. Indianapolis: Wiley, 2012. ISBN 978-1-118-14748-1.

[2] Contributors, A.-F. *A-Frame – Make WebVR* [online]. 2021. Revidováno 27.3.2021 [cit. 2021-04-24]. Dostupné z: https://aframe.io/.

[3] Contributors, B. *The Inspector | Babylon.js Documentation* [online]. 2021. Revidováno 21.3.2021 [cit. 2021-04-24]. Dostupné z: https://doc.babylonjs.com/toolsAndResources/tools/inspector.

[4] Contributors, B. *Node Material | Babylon.js Documentation* [online]. 2021. Revidováno 28.4.2021 [cit. 2021-04-24]. Dostupné z: https://doc.babylonjs.com/divingDeeper/materials/node_material/nodeMaterial.

[5] Contributors, B. *Node Material | Babylon.js Documentation* [online]. 2021. Revidováno 2.4.2021 [cit. 2021-04-24]. Dostupné z: https://doc.babylonjs.com/.

[6] Contributors, B. *Playground | Babylon.js Documentation* [online]. 2021. Revidováno 13.11.2020 [cit. 2021-04-24]. Dostupné z: https://doc.babylonjs.com/toolsAndResources/tools/playground.

[7] Contributors, M. *Canvas API* [online]. 2021. Revidováno 19.2.2021 [cit. 2021-04-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API.

[8] Contributors, M. *CSS: Cascading Style Sheets* [online]. 2021. Revidováno 11.1.2021 [cit. 2021-01-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS.

[9] Contributors, M. *Explaining basic 3D theory* [online]. 2021. Revidováno 27.1.2021 [cit. 2021-04-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/Basic_theory.

[10] Contributors, M. *Graphics on the Web* [online]. 2021. Revidováno 3.2.2021 [cit. 2021-04-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Guide/Graphics.

[11] Contributors, M. *HTML: HyperText Markup Language* [online]. 2021. Revidováno 14.1.2021 [cit. 2021-01-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML.

[12] Contributors, M. *HTTP* [online]. 2021. Revidováno 23.12.2020 [cit. 2021-01-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP.

[13] Contributors, M. *Introduction to the server side* [online]. 2021. Revidováno 21.12.2020 [cit. 2021-01-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction.

[14] Contributors, M. *JavaScript* [online]. 2021. Revidováno 14.1.2021 [cit. 2021-01-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[15] Contributors, M. *What is a web server?* [online]. 2021. Revidováno 2.1.2021 [cit. 2021-01-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server.

[16] Contributors, P. *PlayCanvas WebGL Game Engine* [online]. 2021. Revidováno 6.3.2021 [cit. 2021-04-24]. Dostupné z: https://playcanvas.com/.

[17] Contributors, T. *Three.js – JavaScript 3D Library* [online]. 2021. Revidováno 24.4.2021 [cit. 2021-04-24]. Dostupné z: https://threejs.org/.

[18] Grigorik, I. *High-performance browser networking*. 1. vyd. Sebastopol: O'Reilly, září 2013. ISBN 978-1-449-34476-4.

[19] Lemahieu, W., Broucke, S. vanden a Baesens, B. *Principles of database management: the practical guide to storing, managing and analyzing big and small data*. 1. vyd. New York: Cambridge University Press, 2018. ISBN 978-1-10718612-5.

[20] Tavares, G. *WebGL2 Fundamentals* [online]. 2021. Revidováno 25.4.2020 [cit. 2021-04-24]. Dostupné z: https://webgl2fundamentals.org/webgl/lessons/webgl-fundamentals.html.