Stellar Odyssey 1.0

Tadeáš František Lednický

Creative Hill College

Klauzurní práce 1. pololetí 2. ročníku 2023 - 2.C

https://the-random-unknown.github.io/

Úvod

Vítejte v dokumentaci mé klauzurní práce 1. pololetí 2. ročníku na téma “Projekt v programovacím jazyce Javascript”. Tato dokumentace slouží jako kompletní průvodce mým projektem jménem “Stellar Odyssey”.

Ve Stellar Odyssey se hráč vydává na nebezpečnou cestu vesmírem jako kapitán malého vesmírného plavidla. Toto dobrodružství, inspirované mou oblíbenou hrou Faster Than Light (FTL), kombinuje strategické rozhodování i intenzivní bitvy a to vše v náhodně generovaném systému eventů.

Stellar Odyssey bylo původně vyvíjeno pro potřeby mých přijímacích zkoušek na CHC stejně jako můj druhý projekt The Project Titan. Na rozdíl od TPT jsem Stellar Odyssey nedotáhl do stavu, aby se dal opravdu plnohodnotně hrát. To bylo především kvůli mým omezeným zkušenostem v programování, leč nápad přetrvával, a tak jsem se rozhodnul ho použít pro účel těchto klauzurních prací. Hra je určena pro PC a ke hraní vyžaduje pouze myš a prohlížečové okno o minimální velikosti 1000x640.

Technologie

Při vývoji jsem použil následující technologie:

1. Kódování
    a. Vim / Neovim
    b. Visual Studio Code
2. Testování
    a. Brave Browser
    b. Firefox
    c. Google Chrome
    d. Vivaldi
3. Grafika
    a. Inkscape
    b. Google Fonts
4. Prezentace a dokumentace
    a. Google Dokumenty
    b. Google Prezentace

5. Jazyky
    a. Javascript
    b. HTML
    c. CSS

6. Ostatní
    a. Chat GPT (místo lorem ipsum - aby to přibližně pasovalo do příběhu)

Popis

Hra je udělaná ve vanilla Javascriptu bez použití jakéhokoliv frameworku nebo knihovny. Opět stejně jako u mého webového portfolia jsem se zaměřil na jednoduchou expandibilitu a práci se zdrojovým kódem. Elementy jsou dynamicky vytvářené, takže je jednoduché nějaký přidat nebo ubrat. To stejné platí například o eventech v event systému.

Struktura

Web je strukturovaný hned do několika částí

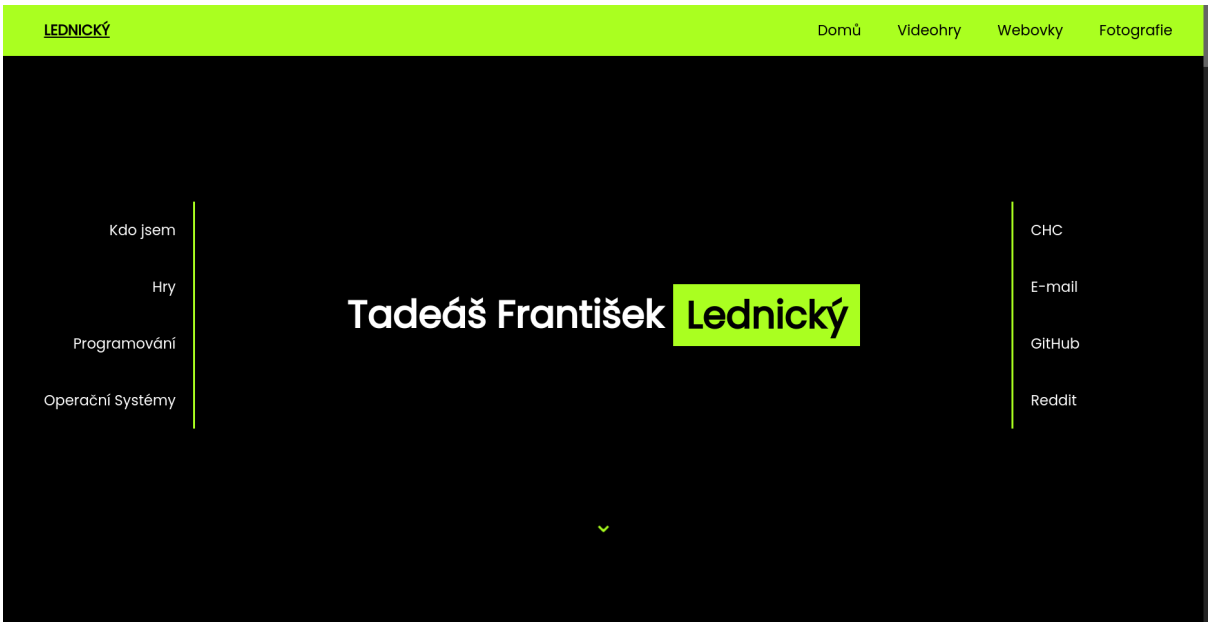
1. index.html
2. game.html
3. game_over.html
4. složka css
    a. master.css
5. složka js
    a. battle.js
    b. destination.js
    c. events.js
    d. factions.js
    e. game_over.js
    f. inventory.js
    g. main.js
    h. menu.js
    i. output.js
    j. utils.js

6. složka src
    a. background.mp4
    b. planet.png
    c. složka ships (obsahuje spoustu obrázků vesmírných lodí)

Kořenová složka společně se složkou css obsahují soubory HTML a CSS, které tvoří základní konstrukci uživatelského rozhraní. Složka src slouží k uchování obrázků a videí potřebných ke správnému vykreslování UI. Poslední složkou je složka js. Složka js obsahuje veškeré skripty, které hru pohání. Celkově se ve složce nachází 10 javascriptových souborů. Každý z nich je pojmenovaný podle účelu nebo funkce, kterou plní.

Uživatelské rozhraní

Rozložení mého uživatelského rozhraní vám může připadat poněkud povědomé. Jedná se totiž o stejné rozložení jako u mého webového portfolia. Porota mi zhodnotila tento design jako velmi šikovný, ale pro webové portfolio trochu moc složitý. Ale pro hru mi to připadá jako ideální rozložení. Hráč má dostatečný přehled nad všemi informacemi, ke kterým má přístup, ale zároveň není přehlcen. Také to neubírá příliš na prostoru pro hlavní kontent.



Sekce

UI je celkově rozděleno na 4 sekce s různým počtem panelů (pod-sekcí)

1. Top section
2. Left section
    a. Obrázek hráčovy lodě
    b. Panel inventáře
    c. Panel Frakcí
3. Right section
    a. Obrázek momentální destinace
    b. Panel destinace
4. Center section
    a. Hlavní output panel
    b. Prostor (container) pro tlačítka

Je důležité mít na mysli, že panely se dají velmi jednoduše přidat, ubrat nebo modifikovat. Tím pádem jejich počet ani pořadí a pozice nemusí být stálé. Panely se mohou vytvářet i za běhu hry (například v tutoriálu). Statické jsou pouze sekce, které jsou definovány v .html souborech. Jedinou výjimkou je output panel, který je jediným panelem, bez kterého hra nemůže fungovat, tudíž je statický a nemůže být odstraněn.

Top Section

Horní sekce je statická a přítomná na všech stránkách projektu. Slouží pouze k navigaci a ukazuje čas. Tato sekce není určena k vytváření vlastních panelů.

Left & Right Section

Levá a pravá sekce jsou výhradně určené k použití dynamických panelů. V momentální verzi 1.0 (potažmo 0.14.56) levá sekce disponuje třemi panely a pravá sekce disponuje dvěma panely.

Center Section

Tato sekce je nejdůležitější sekcí. Stejně jako horní sekce je přítomna na všech stránkách projektů a také není určena k vytváření vlastních panelů. Přesto se jedná o dynamickou sekci. Sekce totiž v sobě má tři containery - Output Container, Buttons Container a battleContainer. Output Container slouží k vypisování hlavního příběhového textu na obrazovku hráče. Jeho obsah se tak neustále mění. To však podle dosavadní terminologie tohoto projektu nedělá z Output Containeru dynamickou sekci. Tou důležitou věcí je buttonsContainer, který, jak už název napovídá, slouží pro vykreslování tlačítek. Tlačítka se dynamicky vytváří a mažou podle potřeby příběhu. A tlačítko jako takové je svým způsobem předem definovaný panel s vlastním obsahem. Pod outputContainerem je ukrytý battleContainer. Ten je víceméně statický. Slouží k informování hráče o průběhu souboje. K ovládání souboje jsou používány tlačítka v buttonsContaineru.

Javascript

Jak už bylo zmíněno, projekt obsahuje celkem 10 JS souborů. Každý ze souborů má na starost nějaký aspekt ze hry a drží v sobě funkce, classy, objekty a metody potřebné pro danou úlohu (výjimkou je soubor utils.js). Celkově lze soubory roztřídit do čtyř kategorií:

1. Kořenové - output.js, battle.js
2. Panelové - inventory.js, destination.js, factions.js
3. Pomocné - events.js, utils.js
4. Inicializační - main.js, menu.js, game_over.js

Kořenové

Tyto soubory zajišťují nejzákladnější chod hry. Veškeré ostatní funkce hry staví nebo závisí na funkcích těchto souborů.

output.js

Jedná se nejdůležitější soubor celého projektu. Jeho prací je obsluhovat výstup hry. To znamená ovládat většinu Center Section. Celý soubor obsahuje pouze jednu třídu jménem outputController. Ta začíná načtením potřebných elementů z HTML.

```
class outputController {
    output = document.getElementById("outputContainer");
    buttons = document.getElementById("buttonsContainer");
    battle = document.getElementById("battleContainer");
```

Vzhledem k tomu, že všechny containery jsou statickými elementy, stačí je pouze pomocí funkce getElementById() načíst jako referenci.

Třída dále obsahuje definice funkcí:

1. write(string) - vypisuje do outputContaineru text, který byl podán jako parametr při volání
2. skipWrite() - zvýší rychlost vypisování do outputContaineru
3. createButton(string, callback) - vytvoří tlačítko s textem, který byl podán jako první ze dvou parametrů (string) a automaticky přiřadí onclick event s odkazem na funkci, která byla podána jako druhý parametr (callback)
4. clearOutput() - vymaže všechen obsah outputContaineru
5. clearButtons() - vymaže všechna tlačítka nacházející se v buttonsContaineru
6. setBattleVisible() - nastaví přesun battleContaineru do popředí a přepne tak hru do soubojového módu
7. setOutputVisible() - vrátí do popředí outputContainer, a tím vrátí hru do story módu

Instance této třídy je vytvořena jednou při načtení stránky z inicializačního skriptu pod názvem “output”. Ostatní funkce a objekty následně volají její funkce jako output.foo(bar).

battle.js

Stejně jako output.js obsahuje pouze jednu třídu a pracuje také pouze se statickými elementy. Jeho prací je řídit průběh soubojů. Obsahuje také constructor, který se stará o vybrání náhodného obrázku z databáze lodí a jeho dosazení na místo nepřátelské lodě. Použití této třídy se liší od ostatních tříd v tomto projektu. Instance této třídy není vytvořena jednou na začátku a poté pouze ovládána zvenčí. Tato třída si řídí svůj běh sama. Pro vytvoření souboje nám stačí pouze vytvořit novou instanci této třídy a předat jí parametry o počtu životů nepřítele, jeho síle a callback funkci, která bude zavolána po ukončení souboje.

```
new Battle(20, 5, callback());
```

Panelové

Panelové soubory mají na starost jednotlivé panely v bočních sekcích. Obsahují vždy jednu třídu. Ta vždy má svůj constructor, který vytvoří tělo panelu a nastaví jeho parametry. Instance panelu pak může být vytvořena v inicializačním skriptu společně s instancí outputControlleru. To však nemusí být pravidlem a instance panelů mohou být vytvořeny kdykoli za běhu hry. Pokud se tak například v rámci příběhu dostane hráč do bodu, kde získá novou technologii, může být panel vytvořen až v momentě, kdy tuto technologii nainstaluje na své vesmírné plavidlo. Panely klidně mohou být specifické pouze pro jeden specifický event a po jeho dokončení se samy zničí.

příklad z factions.js:

```
constructor() {
    const container = document.getElementById("leftContainer");

    this.factions = document.createElement("div");
    this.factions.className = "panel statusContainer";
    container.appendChild(this.factions);
}
```

Panely samy o sobě většinou neřídí svůj běh. Je možnost vytvořit asynchronní funkce, které se budou spouštět samy pomocí funkcí setTimeout() a setInterval(). Za normálních okolností

jsou ovládány zvenčí za pomocí setFoo(bar) a getFoo(bar). Některé panely disponují ještě funkcemi addFoo(bar), které slouží pro dynamické upravování namísto setFoo(bar), které hodnotu přepíše.

Pomocné

events.js

Z názvu jde poměrně snadno usoudit účel tohoto souboru. Vzhledem k tomu, že Stellar Odyssey je hra založená na náhodných eventech, které vás mohou potkat během cesty, je potřeba tyto eventy odněkud brát. K tomu slouží právě tento soubor. Soubor je rozdělen do několika částí, přičemž každá z částí představuje jeden event. Eventy se větví do stromové struktury. Této struktury je dosáhnuto pomocí dvou vnořených arrayů. Důvod, proč jsem nepoužil id a parent id strukturu, která se standartně používá pro stromové struktury, je zaprvé fakt, že už předem vím přesný počet položek, které se ve stromě nacházejí. Druhým faktem je, že scénář pro příběh se velmi podobá arrayi jako takovému, tím pádem je jednodušší pro práci než id a parent id.

Soubor začíná deklarací arraye eventů

```
let events = [];
```

dále vytvořím druhý array pro určitý event a přidám ho do arraye events[]

```
//======== Ukazkovy event ========
let ukazkovy_event = [];
events.push(ukazkovy_event);
```

Nyní do nultého indexu ukázkového eventu dosadím funkci, která obsahuje event jako takový

```
ukazkovy_event[0] = async function () {
    destination.setName("Planeta " + getRadnomCode());
    await output.write("Ukázka eventu");
    output.createButton("Tlačítko", ukazkovy_event[1]);
}
```

Na poslední řádce této funkce předávám funkci output.createButton další opět

ukazkovy_event ale s indexem 1. Mohu takhle vytvořit více tlačítek, která mohou ukazovat na různé možnosti v rámci celého eventu, stejně jako více tlačítek z různých míst mohou ukazovat na jednu a tu samou možnost. Zbytek volaných funkcí jsem popsal v samostatných kapitolách jejich třídy.

utils.js

Souhrn všech funkcí, které se nehodily do žádného z jiných souborů. Většinou se jedná o pomocné funkce, jako je například funkce sleep(ms). Některé z nich jsou na druhou stranu klíčovou součástí herní smyčky, jako je například funkce generateRandomEvent().

Inicializační

Tyto soubory slouží jako hlavní entry point pro jednotlivé stránky. Mají na starost vytvořit potřebné objekty a spustit funkce, které zařídí běh hry. Každý ze souborů obsahuje funkci main(). Každá ze tří stránek má svůj inicializační soubor s upravenou funkcí main() pro její potřeby. Funkce se poté spouští jako onload na HTML elementu “body”.

Příklad ze souboru main.js linknutém v souboru game.html

```
const inventory = new inventoryPanel;
const factions = new factionsPanel;
const destination = new destinationPanel;
const output = new outputController;
let score = 0;
let currect_event;

async function main() {
    updateClock();
    inventory.update();
    destination.update();
    factions.update();
    generateRandomEvent();
}
```

Závěr

Během dělání tohoto projektu jsem získal spoustu nových zkušeností. Se skriptovacím jazykem Javascript jsem nikdy předtím nepracoval, a proto to pro mě byl velký nezvyk. Javascript funguje velmi odlišně, než všechny programovací jazyky, se kterými jsem se kdy setkal. Struktury, které jsem byl naučen používat z jazyků jako jsou C a C++, v Javascriptu prostě nefungují. Tím největším nezvykem byl execution flow celého projektu. Z ostatních programovacích jazyků jsem byl zvyklý, že program má jeden vstupní bod (entry point) a od toho body se odvíjely všechny následují kroky programu. Poté se postupným zanořováním call-stacku vyvíjela logika programu. Pokud někde bylo potřeba zpracovávat eventy, vytvořil jsem si vlastní event loop. Asynchronní akce jsem řešil na jednotlivých jádrech. Bohužel tím, že Javascript není programovací jazyk, se tento flow razantně mění. Vstupní bod jsem si byl schopen vytvořit sám pomocí inicializačních skriptů, ale největší problém byl s asynchronní exekucí. Z main() funkce zavolám generateRandomEvent() a ten spustí asynchronní funkci eventu a ukončí sám sebe. Díky tomu se call-stack zredukuje z main() > generateRandomEvent() > event() jen na event(). Vím, že toto se dá řešit pomocí klíčového slova await, ale to ne vždy fungovalo opravdu tak, jak jsem potřeboval. Dalším bodem je event loop, který v případě Javascriptu řídí webový prohlížeč. Po nastavení eventu listeneru se event ukončí, a tím zastaví exekuci veškerého javascriptového kódu. V tuto chvíli nemám jako programátor jakoukoliv kontrolu na chováním mojí aplikace. To je velmi nepříjemné, protože pokud nastane situace, na kterou jsem nebyl připraven, nemohu jí předem nijak zabránit bez toho, abych na každý možný element přidal event listener, ve kterém budu zvlášť vyhodnocovat každý jeho stav. Tato moje frustrace je na kódu zřetelně viditelná a ve spojení s tím, že při začátku projektu mi ani nebylo jasné, co chci vytvořit, to způsobilo, že kód je velmi nekonzistentní, zmatečný, náchylný na bugy a celkově velmi křehký. Jednotlivé elementy jsou sice robustní, ale pokud by se například nezkonstruoval jeden z důležitých objektů v inicializačním skriptu, celý projekt se sesype jako domeček z karet. Dát podmínku, která přesměruje uživatele na chybovou stránku je jedno z řešení, ale není opravdu robustní. V ideálním případě by měl být schopen skript zjistit, jaké jeho části nefungují, a buď je alternativními cestami spravit a nebo spustit projekt v omezeném módu tak, aby žádná jiná z částí neodkazovala na tu, která nemusí fungovat. Celý projekt by se měl chovat více jako rozvětvené domino. Pokud se někde zasekne, ostatní větve budou pokračovat nezávisle na té zaseklé. Poukazováním na tyto chyby chci ukázat, že jsem se opravdu něco naučil a že tento projekt opravdu splnil svůj účel, protože kdybych ho psal nyní, psal bych ho úplně jinak.