Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Webová aplikace pro hledání cest na herní mapě

Bakalářská práce

Autor: Matyáš Peremský
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Ph.D., Vojtěch, Vorel

Hradec Králové Leden 2024

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 24.4.2024

Matyáš Peremský

Poděkování:

Děkuji vedoucímu bakalářské práce Mgr. Ph.D., Vojtěch, Vorel za metodické vedení práce a podporu.

Abstrakt

Cílem bakalářské práce je vytvořit aplikaci pro hledání nejkratší cesty na libovolné herní mapě Trackmania. Práce se zaměřuje zejména na verzi z roku 2008, TrackMania Nations Forever. Jako řešení byla zvolená webová aplikace s grafickým editorem a vizualizací cest. Editor používá snímek obrazovky ze hry, na který uživatel obkreslí důležité části mapy. Aplikace použije vlastní algoritmy na výpočet nejkratších cest a nalezené cesty následně vizualizuje animací.

Testy na reálných mapách prokázali úspěšné vyhledání kratších cest než původně zamýšlených autorem mapy, a to v průměru o více než 20 %. Zároveň potvrdilo přijatelnou časovou náročnost alespoň pro jeden z algoritmů.

Výsledkem této práce je užitečný nástroj, který pomůže hráčům ve světě kompetitivní hry Trackmania.

Abstract

Title: Web application for finding the shortest route on a game map

The goal of this bachelor thesis is to create an application for finding the shortest path on any map of the game Trackmania. More specifically this work is focused on the 2008 version, TrackMania Nations Forever. A web application with a graphical editor and path visualization was chosen as the solution. The editor utilizes a screenshot from the game onto which the user draws important parts of the map. The application uses custom algorithms to compute the shortest paths, which are then visualized through animation.

Tests on real maps demonstrated successful discovery of shorter paths than originally intended by the map author, averaging over 20% shorter. Additionally, the tests confirmed acceptable time complexity for at least one of the algorithms.

The outcome of this work is a valuable tool that assists players in the competitive world of Trackmania gaming.

Klíčová slova:

Web; aplikace; graf; hledání; algoritmus; hra

Key words:

Web; application; graph; search; algorithm; game

Obsah

1 Úvod

Trackmania je série arkádových závodních her vydaných v letech 2003-2020. Tato práce se bude soustředit na verzi z roku 2008: „TrackMania Nations Forever“. Hlavním cílem hry je za co nejkratší čas se dostat ze startu do cíle. Jako součástí hry je editor, kde si hráči mohou postavit vlastní trať. Za desítky let, co hra vyšla, hráči postavili nespočet tratí lišící se stylem a složitostí. Některé z tratí jsou na tolik komplikované, že je téměř nemožné vyhledat nejkratší cestu pouhým instinktem. Proto jsem se rozhodl vytvořit nástroj, který má za úkol pomoct hráči s tímto problémem.

Rád bych svojí prací pomohl budoucím hráčům rozluštit tento problém, ušetřit jim spoustu času, navést je na správnou cestu nebo umožnit efektivněji sdílet jejich strategie.

Obor informatiky a vývoj webových stránek/aplikací je mi velmi blízký, jelikož je to záležitost, které jsem se jako student, při studii a ve volném čase věnoval. Další záležitost, které jsem se nepochybně věnoval jsou počítačové hry. Proto jsem se rozhodl spojit moje odborné dovednosti (vývoj webových stránek) a koníčky (počítačové hry) a vytvořit nástroj, který pomůže hráčům.

2 Cíl a metodika práce

Hlavním cílem práce je vytvořit nástroj (webová aplikace), který je schopný najít optimální cestu na jakékoliv trase ve hře Trackmania. Jelikož se ukáže, že je tento problém NP-těžký, je nutné, aby hledání proběhlo v praktickém časovém rámci.
Takový nástroj může mít několik podob: Webová aplikace, desktopová aplikace, modifikace hry. Z důvodu přístupnosti a bezpečnosti jsem zvolil webovou aplikaci.
Aplikace bude napsaná v jazyce *JavaScript* pomocí knihovny *SvelteKit*. Uživatelské rozhraní se bude skládat ze dvou částí: Editor, Generátor. Pro reprezentaci trati, jsem si zvolil 2d pohled ze shora. Výhodou je rychlá a jednoduchá tvorba tratě v editoru.

Bakalářská práce se skládá ze dvou částí, teoretická a praktická.
Úvodní teoretická část se zaměřuje na vysvětlení základních pravidel a prvků hry Trackmania.
Praktická část se zaměřuje na aplikaci jako takovou.

3 Teoretická část

3.1 Trackmania

Trackmania (2003-2020) je série závodních arkádových her od společnosti *Ubisoft Nadeo*, dříve jen *Nadeo*.

Trackmania United Forever (dále jako TMUF) a *Trackmania Nations Forever* (dále jako TMNF) jsou verze vydány v roce 2008. TMNF je s TMUF plně kompatibilní, tj. hráči mohou spolu hrát na stejné trati a připojit se na stejný server pro multiplayer. TMNF je oproti TMUF neplacená verze, ale s limitovaným obsahem hry.

Rozdíl mezi verzí není pro tuto práci relevantní, a proto nadále výraz „Trackmania“ bude odkazovat na oboje verze TMUF a TMNF.

Typicky se Trackmania hraje ve formátu „závod na čas“, kde se skóre měří v milisekundách časovaných od spuštění závodu do úspěšného ukončení závodu. Jelikož je fyzika hry deterministická, tak se místo času uloží seznam vstupů s přesným časem aktivace jako záznam. Hra pak může záznam pomocí vstupů perfektně replikovat v pozadí s použitím svého herního enginu. Díky této funkci lze jakýkoliv záznam ověřit, zda byl hrán bez jakékoli modifikace hry. Hra také umožňuje záznam přehrát jako „duch“ živě při novém pokusu. Lze tedy závodit se svým nejlepším pokusem a dokola se zdokonalovat. Díky službě *tmnf.exchange*, lze záznamy sdílet online na žebříček a stahovat záznamy ostatních, tedy s nimi lokálně soutěžit.

Trackmania nabízí oficiální kampaň se seznamem tratí se vzrůstající obtížností postavenými přímo vývojáři Nadeo. Nové tratě lze postavit v editoru přímo ve hře. Opět, služba tmnf.exchange umožňuje sdílet postavené tratě. Tratě lze tedy kdykoliv stáhnout a zahrát si. Dále hra nabízí mód „Multiplayer“, kde hráči spolu živě soupeří na jedné trati.

3.1.1 Prvky

Každá trať je postavená z bloků nacházející se na mřížce ve 3D prostoru.

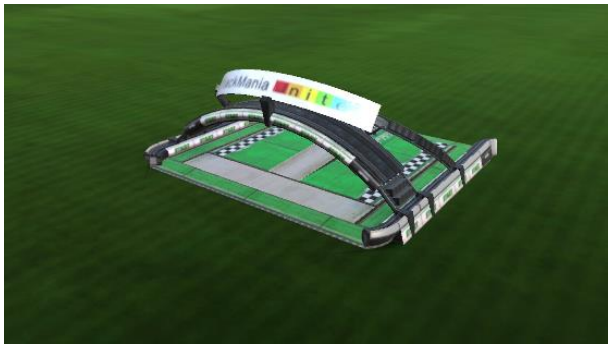
Tato část popisuje jednotlivé prvky trati, které jsou relevantní pro tuto práci.

Formule

Obrázek 1 – Formule s lakem reprezentující český národ.

Na rozdíl od ostatních závodních her, ve hře TMNF je na výběr jen jedno vozidlo. Tím je *formule*. Formule nelze nijak vylepšovat nebo modifikovat. Díky tomuto faktu má každý hráč stejné podmínky při závodu a jediné co rozhoduje o lepším výsledku jsou jeho dovednosti.

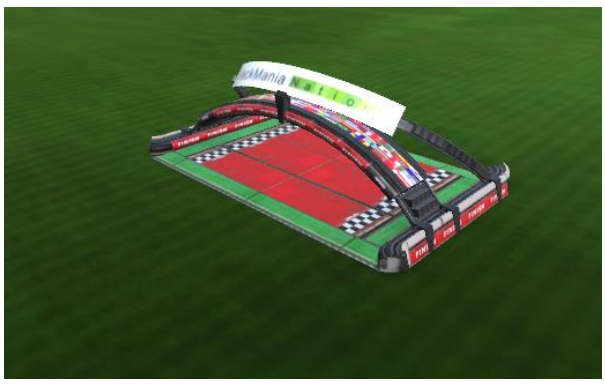
Start



Obrázek 2 – Startovní blok postavený na trávě stadionu.

Startovní blok existuje na každé trati jen jednou. Značí pozici, na které se formule ocitne na začátku závodu.

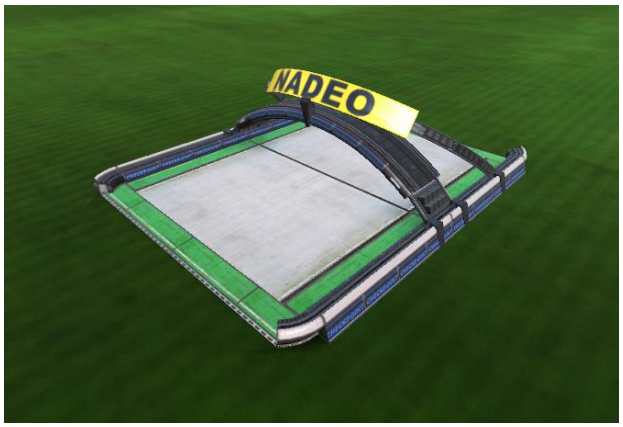
Cíl



Obrázek 3 - Cílový blok postavený na trávě stadionu.

Každá trať má alespoň jeden *cílový blok*. Může ale obsahovat více než jeden. Pro aktivaci *cílového bloku* se musí formule dotknout čáry nakreslené uprostřed. Závod je dokončen, pokud formule projela skrz všechny checkpointy alespoň jednou a dotkla se cílové čáry.

Checkpoint



Obrázek 4 – Checkpoint blok postavený na trávě stadionu.

Checkpoint, česky záchytný bod, je nepovinný prvek. Trať může obsahovat jakýkoliv počet checkpointů. Slouží jako bod na trati, který musí hráč navštívit a kam může hráč obnovit svojí pozici.

Ring Checkpoint

Obrázek 5 – Ring Checkpoint blok ve vzduchu.

Ring checkpoint, česky kruhový záchytný bod, je varianta *checkpointu*. Většinou se nachází ve vzduchu a formule musí skrz něj proletět. Dalším rozdílem je, že hráč nemůže obnovit svojí pozici na pozici ring checkpointu.

3.1.2 Pravidla

Cílem každého závodu je se dostat za co nejkratší čas do *cíle*. Po projetí skrz *checkpoint* se checkpoint zapíše jako splněný. Pokud je na trati jakýkoliv počet checkpointů, je povinné před dosažení cíle každý splnit. Na pořadí nezáleží.

Závod se ukončí právě v tu chvíli, když formule překročí cílovou čáru a zároveň formule projela skrz všechny checkpointy.

Na rozdíl od ostatních her pravidla více hráče nelimitují. Je proto možné např. jet opačným směrem, vyjet ze zamýšlené dráhy a pokračovat jízdu po trávě nebo vystoupat na střechu stadionu, ve kterém se všechny mapy nachází.

Reset

Reset je funkce, která závod ukončí a spustí nový pokus.

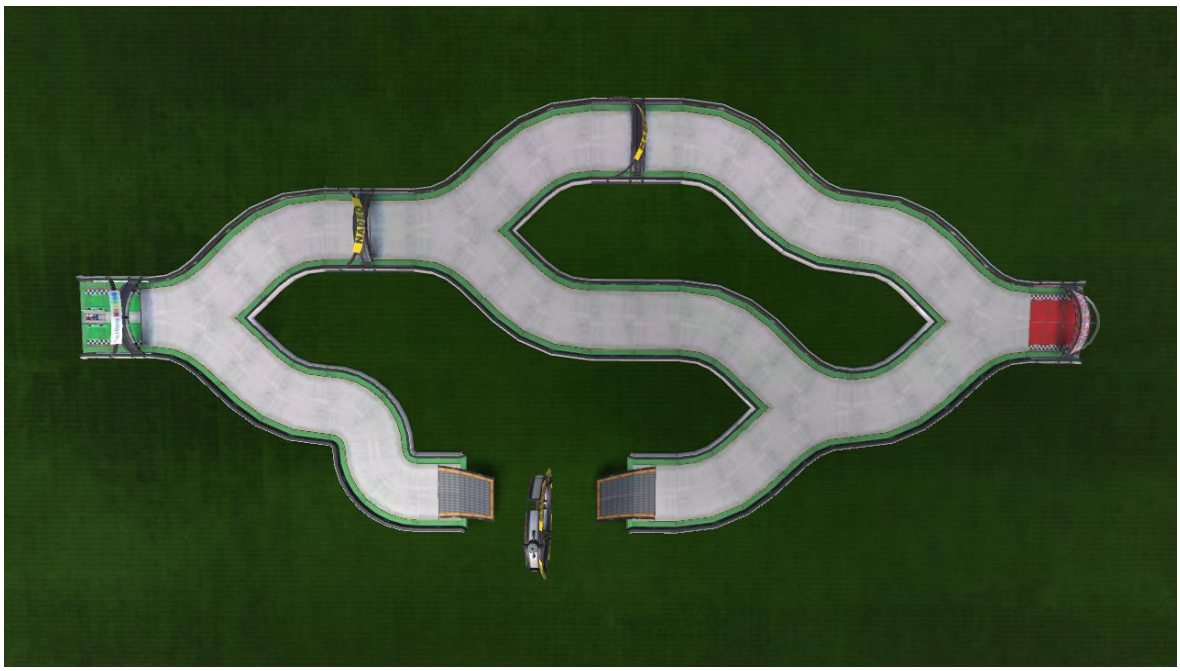
Respawn

Respawn je funkce kterou má možnost hráč kdykoliv v průběhu závodu aktivovat. Po aktivování přesune/teleportuje formuli na pozici posledního splněného checkpointu, který není kruhový. Zároveň nastaví rychlost formule na 0. Pokud hráč doposud nesplnil žádný checkpoint, tak se namísto aktivuje reset.

Tato funkce byla zamýšlena jako nouzová pomoc hráči, pokud omylem vyjede z dráhy. Hráči ale objevili, že je možné funkci využít pro časovou výhodu.

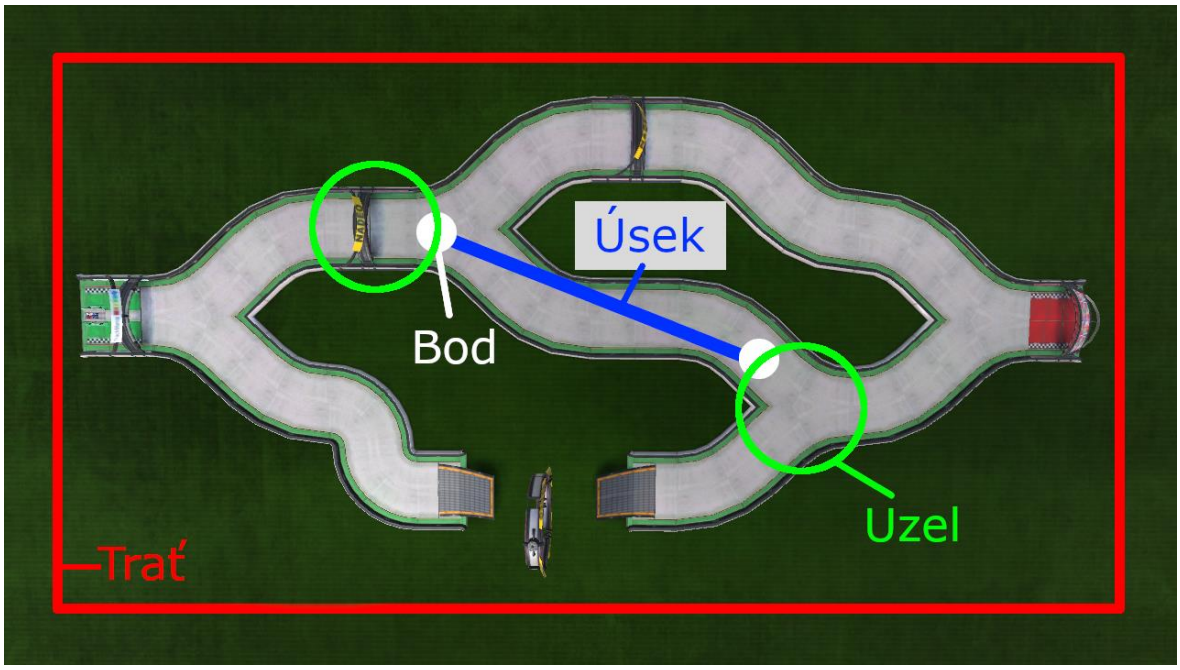
Existuje výjimka, kdy je závod spuštěný v režimu „official mode“. Pokud hráč aktivoval funkci respawn a zároveň nesplnil žádný checkpoint nekruhového typu, je pro přesun použita pozice startovního bloku.

3.2 Vymezení problému



Obrázek 6 – ukázka jednoduché mapy, kterou budeme dále reprezentovat grafem. Trať zleva počíná startovním blokem, spojené cestou s klasickými checkpointy nahoře a kruhovým checkpointem dole a ukončená cílovým blokem napravo.

Vysvětlení pojmů



Obrázek 7 – Příklad použitých pojmů na ukázkové mapě.

Vrchol / Uzel – jeden z aktivních (vysvětleno níže) prvků nebo křižovatka na trati. V oboru teorii grafu **vrchol**.

Úsek – Spojení mezi dvěma vrcholy. V oboru teorii grafu **hrana**.

Bod / Konec – Každý úsek obsahuje dva konce. Každý konec je napojený na vrchol.

Cesta / Trasa – *Trať* obsahuje mnoho *cest* a cesta se skládá z jednoho nebo více úseků. V našem případě se snažíme hledat nejkratší cestu od startu do cíle. V teorii grafu **sled**.

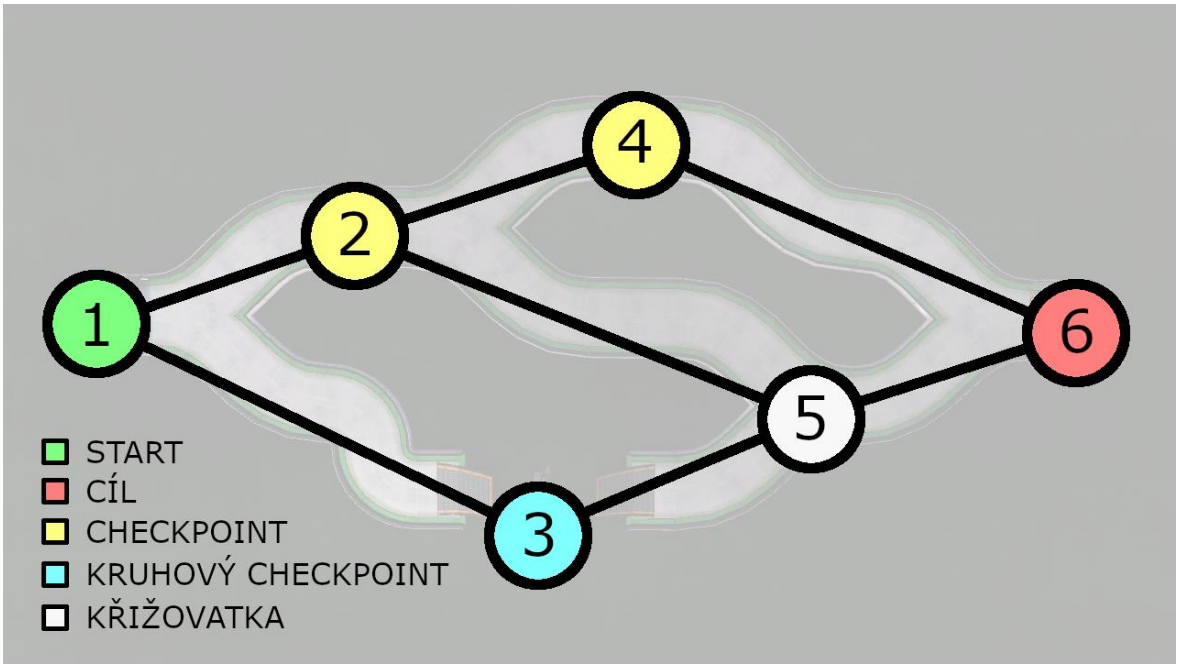
Mapa / trať – Celé prostředí, kde se koná závod. Obsahuje všechny postavené aktivní a pasivní bloky. V oboru teorii grafu je to **graf**.

Převedení obrázku na graf

Každá mapa se skládá z bloků. Tyto bloky lze zařadit do dvou kategorií: *aktivní* a *pasivní*. Aktivní jsou bloky, které ovlivňují podmínky nebo stav závodu. Např. startovní blok určuje počáteční pozici formule a checkpoint určuje pozici návratu a při prvním projetí navýší počet splněných checkpointů. Patří mezi ně: startovní blok, checkpoint, kruhový checkpoint a cílový blok. Mezi pasivní bloky pak patří ostatní.

Typická mapa je postavena z těchto aktivních bloků, které jsou často spojeny cestou. Pokud k těmto aktivním blokům a místům, kde se cesty kříží přiřadíme vrcholy a

propojení mezi nimi cestou označíme jako hrany, tak každou mapu můžeme reprezentovat grafem. Např. *Obrázek 8* je grafová reprezentace *Obrázek 6*.



Obrázek 8 – Grafová reprezentace ukázkové mapy.

Související matematické problémy

Shortest Path Problem

Problém nejkratší cesty spočívá v nalezení nejkratší cesty mezi dvěma vrcholy v grafu *G = (V, E)*, kde každá hrana e je spojena s cenou *c(e)*. S daným počátečním vrcholem *s* a cílovým vrcholem *t* je cílem najít cestu z *s* do *t* s minimální celkovou cenou. [1].

Traveling Salesman Problem

Problému obchodního cestujícího, známém také jako "TSP", se zabývá hledání nejkratší *Hamiltonovy kružnice* v grafu.

Za předpokladu, že máme neorientovaný úplný graf *G = (V, E)*, kde každá hrana *e* je spojena s cenou *c(e)*. Celková cena cesty je součet cen všech jeho hran. Cílem je najít cestu s nejmenší celkovou cenou, která obsahuje každý vrchol právě jednou a končí počátečním vrcholem. Jinými slovy Hamiltonovu kružnici.[2] Tento problém má složitost *NP-těžký*[3]. Tím pádem jakýkoliv algoritmus, který se tímto problémem zabývá má alespoň polynomiální časovou složitost.

Steiner Traveling Salesman Problem

Steinerův problém obchodního cestujícího (STSP) je rozšíření klasického TSP.

STSP je definovaný na neorientovaném grafu *G = (V, E)*, ne nutně úplném, kde každá hrana *e* je spojena s cenou *c(e)*. Máme množinu povinných vrcholů *VR*, která je podmnožinou *V*. Celková cena sledu je součet cen všech jeho hran, kde každá hrana je započítána tolikrát kolikrát je navštívena. Cílem je najít sled s nejmenší celkovou cenou, který navštíví každý povinný vrchol alespoň jednou a končí v počátečním vrcholu. Jelikož je STSP rozšíření TSP tak má tento problém rovněž složitost NP-těžký.

3.2.1 Hledání nejkratší cesty

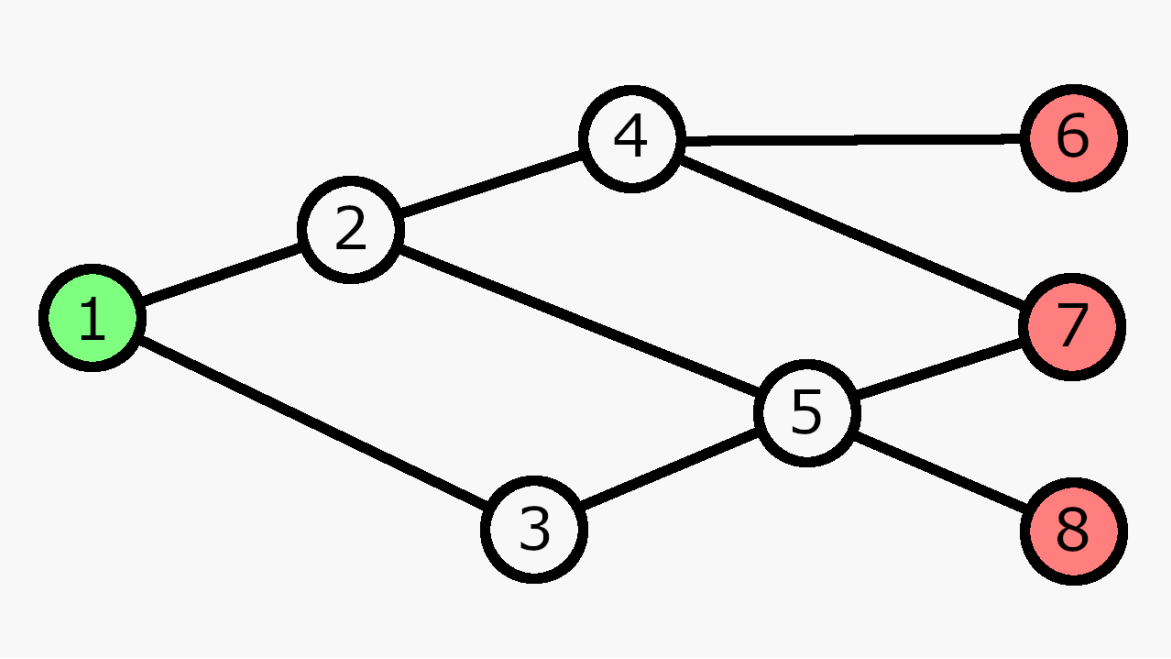
Níže je popsaný postup při hledání nejkratší cesty v různých případech.

Triviální případ

Obrázek 9 – Graf s jedním startem a jedním cílem.

Minimální požadavek pro validní mapu je jeden startovní blok a alespoň jeden cílový blok. V tomto případě se jedná o graf s dvěma vrcholy a jednou hranou, která je spojuje. Graf obsahuje jen jednu cestu, která je tím pádem nejkratší.

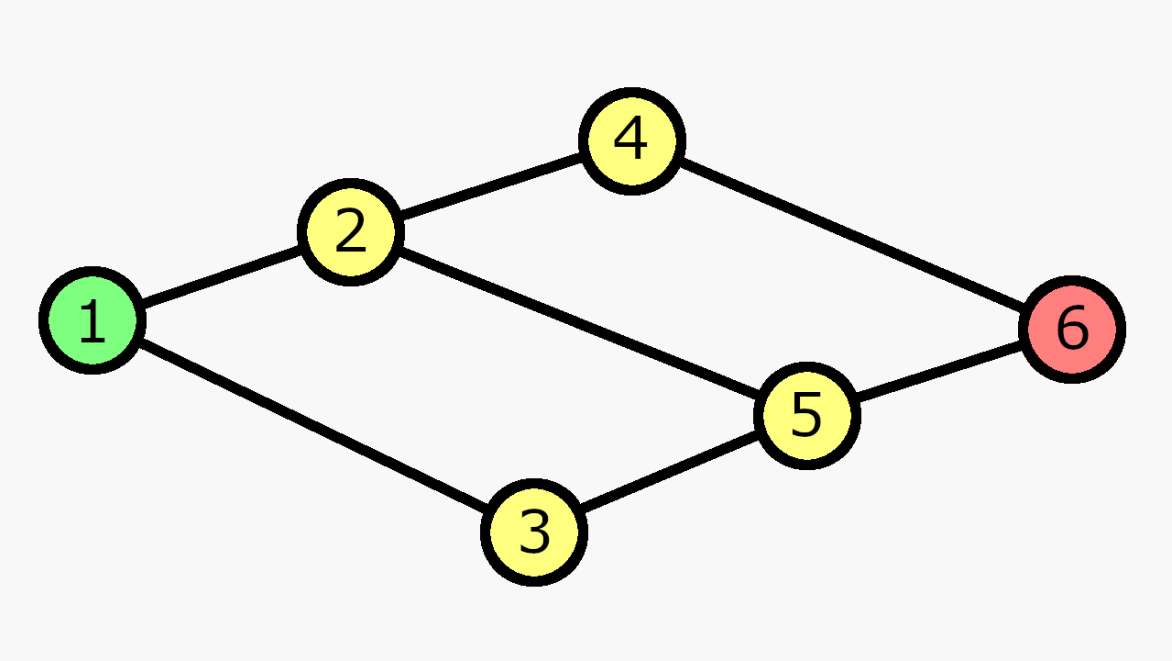
Jen křižovatky + jeden nebo více cílových bloků



Obrázek 10 – Graf s mnoha křižovatky a mnoha cíli.

Pokud bychom do grafu přidali vrcholy reprezentující křižovatky, tak problém můžeme klasifikovat jako „Problém nejkratší cesty“. Jelikož všechny hrany nabývají délky nezáporné hodnoty, lze úlohu vyřešit v polynomiálním čase za použití Dijkstrův algoritmu.[1]

Jen checkpointy



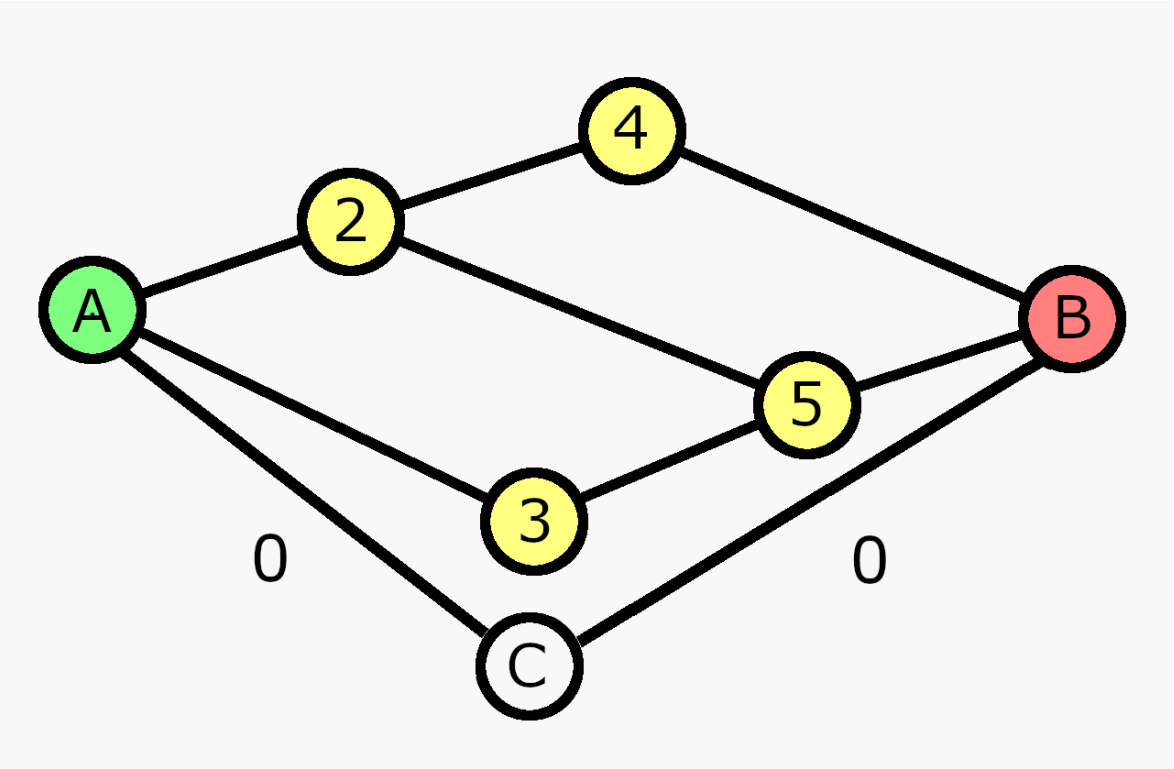
Obrázek 11 – Graf s mnoha checkpointy a jedním cílem.

Pokud graf obsahuje jen jeden startovní blok, jeden cílový blok a *n* checkpointů (*Obrázek 11*), tak je možné graf modifikovat, aby odpovídal problému „Problém obchodního cestujícího“.

Cílem problému obchodního cestujícího je najít nejkratší Hamiltonovu kružnici. My chceme namísto kružnice najít nejkratší cestu od bodu *A* do bodu *B* (start a cíl). Když zajistíme, že na výsledné Hamiltonové kružnici vrchol *A* a *B* sdílí totožný sousední vrchol *C*, tak nám pro získání požadované cesty stačí z kružnice odstranit vrchol *C*.

Pro modifikaci postupujeme takto:

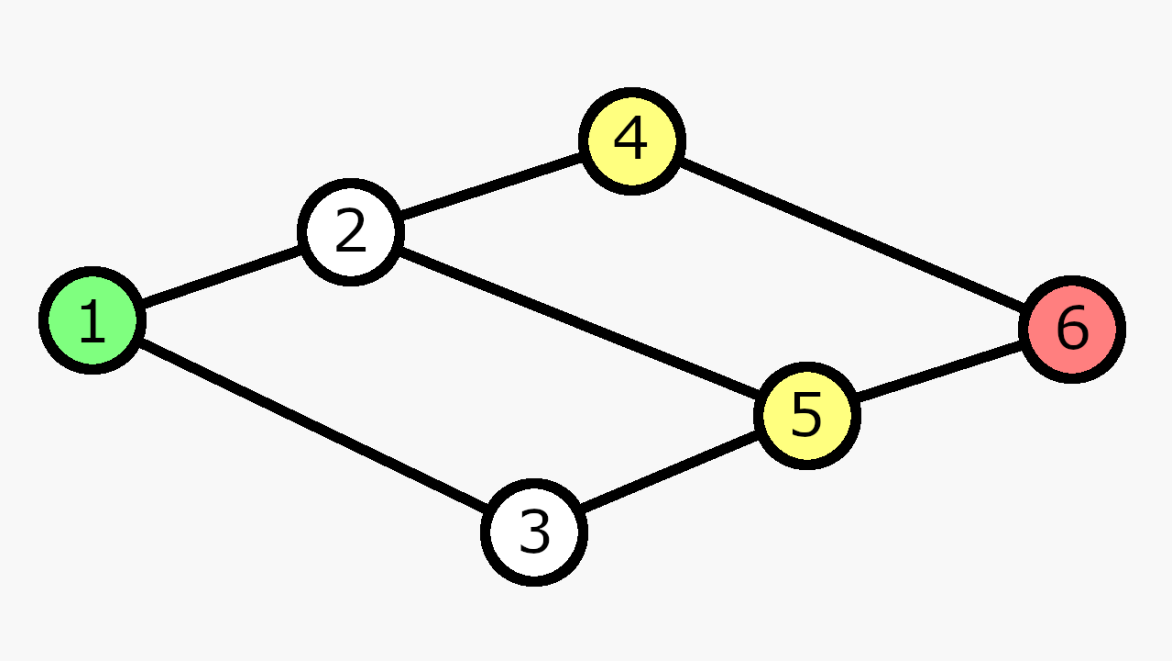
1. Přidáme do grafu vrchol *C*
2. Vrchol *C* připojíme k vrcholu *A* a *B* hranou *h1* a *h2*
3. délku hran *h1* a *h2* nastavíme na 0



Obrázek 12 – Modifikace minulého grafu (Obrázek 11). Přidání nového vrcholu *C* a hran, které spojují vrchol *C* s vrcholem *A* a *B*.

Tato modifikace nám garantuje, že na výsledné Hamiltonové kružnici je vrchol *A* a *B* sousedem vrcholu *C*. Pro získání optimální cesty nám stačí z kružnice odstranit vrchol *C*.

Checkpointy a křižovatky



Obrázek 13 – Graf s mnoha křižovatky, checkpointy a jedním cílem.

Pokud graf obsahuje jeden startovací blok, jeden cílový blok, jedenu nebo více křižovatek a jeden nebo více checkpointů, tak problém spadá do kategorie *STSP*.
Nejdříve graf modifikujeme jako v předchozím problému. Poté pro každý checkpoint, start a cíl definujeme jako povinný vrchol. Následně můžeme postupovat podle algoritmu určený na tuto kategorii.

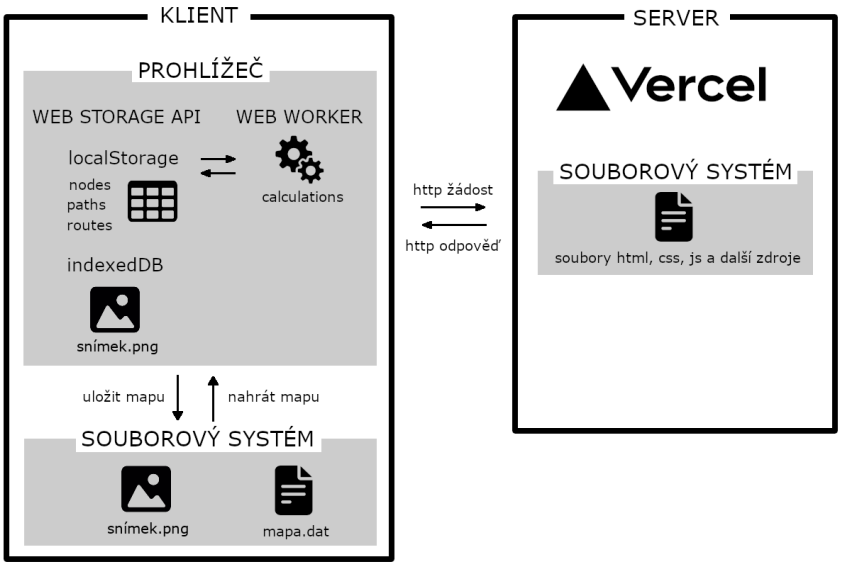
Ostatní případy

Kromě zmíněných podmínek je problém ještě více všeobecný. Např. trať může obsahovat sekci se seskokem, na které je možné jet jen v jednom směru. Ve skutečnosti se nejedná o *prostý graf*, ale *orientovaný multigraf* (mixed multi graph). Zároveň je nutné při výpočtu nejkratší cesty počítat s funkcí „respawn“, která formuli vrátí na poslední checkpoint. Díky této skutečnosti se mi nepodařilo tento konkrétní problém zařadit kategorie již zkoumaných problémů. Jedná se ale o variaci problému TSP, a proto zapadá do složitosti NP-obtížné.
Jelikož tyto případy tvoří většinu, rozhodl jsem se napsat vlastní algoritmus, který najde alespoň aproximaci optimální cesty v rozumné časové době. Zmíněné algoritmy jsou do detailu popsané v praktické části (kapitola 4.4).

4 Praktická část

4.1 Technologie

4.1.1 Architektura



Obrázek 8 - diagram architektury webové aplikace

Klient-Server

Architektura aplikace je poměrně jednoduchá. Jedná se o typ architektury *klient-server*, kde server hostuje statické soubory webových stránek (HTML, CSS, JS) a komunikuje s klientem přes HTTP protokol (požadavek, odpověď). Na serveru je spuštěna instance *SvelteKit* aplikace.

Vercel

Pro backend je využito služby *Vercel*. Vercel je poskytovatel cloudových služeb, jejichž součástí je hostování webových aplikací a správa serveru a infrastruktury[4]. Tento přístup abstrahování správy serveru je znám jako *serverless backend* architektura[5].
Další funkcí, která je využívaná aplikací, je automatické nasazení aplikace do produkce. Vercel zaznamená poslání nejnovější verze aplikace do produkční větve na Git repositáři a automaticky novou verzi nasadí do produkce, pokud při sestavování nenastane chyba[6].
Aplikace je hostovaná na adrese https://tm-cutfinder.vercel.app/

SSG a SPA

SSG (static site generation), česky statické generování stránek, označuje architekturu webové stránky, kde je každá stránka vygenerovaná už před nasazením na server[7]. Její výhodou je rychlejší načtení stránek a lepší hodnocení SEO.

SPA (single page application), česky jednostránková aplikace, označuje architekturu webové stránky, kde všechny požadavky na server vrátí jeden HTML dokument. Všechny následné požadavky změny obsahu se zpracují a generují na straně klienta pomocí *Javascript*. Navigace je řešena na straně klienta v procesu zvaném *client-side routing*, přičemž obsah jednotlivých stránek se aktualizuje a běžné prvky uživatelského rozhraní zůstávají nezměněny[7]. Klient nemusí čekat na odpověď od serveru, a proto je výhodou rychlé vykreslení nových stránek a obsahu zahrnutého v původním HTML dokumentu.

Aplikace využívá hybridní přístup kombinující výhody obou SSG a SPA. Je možné zvolit tento přístup, protože většina výpočtů a ukládání dat probíhá na straně klienta. Nejprve se hlavní stránka vygeneruje při sestavování. Následně při požadavku server vrátí již vykreslenou stránku a veškerá interakce se počítá na straně klienta.

Ukládání dat

Na rozdíl od běžného způsobu nejsou uživatelsky generovaná data ukládaná na straně serveru v databázi, ale na straně klienta. Aplikace používá dva způsoby pro ukládání dat.

První způsob je využití *Web Storage API*[8], [9], přesněji *localStorage* a *indexedDB.* Při každé změně se data automaticky uloží do paměti přiřazené prohlížeči. Při přerušení a navázání nové relace se data automaticky načtou z paměti. Je takto zaručeno, že se data zachovají skrze více relací.

Druhý způsob je uložení dat do souboru. Aplikace nabízí možnost export nebo import dat ve formě binárního souboru (kapitola 4.3.4). Kromě uložení na dobu neurčitou je možné tento způsob využít pro sdílení dat s ostatními uživateli.

Web Worker API

Prohlížeč poskytuje aplikaci jen jediné hlavní vlákno pro většinu operací. Výpočet nejkratší cesty je velice výpočetně náročný, a proto je nutné tyto výpočty provádět mimo hlavní vlákno, jinak by hrozilo jeho zablokování a „zamrznutí“ uživatelského rozhraní. Z těchto důvodů aplikace využívá *Web Worker API*. Interface Web Worker

umožňuje založení objektu *worker*, který spustí skript v novém izolovaném vlákně[10]. *Worker* a hlavní vlákno komunikují mezi sebou pomocí zpráv[11]. Nejdříve se z hlavního vlákna pošle zpráva s informacemi o mapě, *worker* začne s výpočtem a průběžně posílá zpátky zprávy o nalezených cestách a o průběhu výpočtu. Toto API je použito pro výpočet nejkratší cesty (více v 4.4).

4.1.2 Svelte

Pro vývoj interaktivních webových aplikací od základu prohlížeč poskytuje programovací jazyk Javascript (JS) a Document Object Model (DOM) pro prístup k dokumentu a jeho prvkům. Každopádně udržovat stav paměti synchronizovaný s UI je často velmi složitý a nepraktický. Proto byly vyvinuty knihovny a frameworky, které řeší tento problém a nabízí vývojáři mnoho dalších funkcionalit.

Pro vývoj webové aplikace byl zvolen framework *Svelte.js*. Svelte je javascriptový framework vyvinutý Richem Harrisem z The New York Times. Kromě zmíněného synchronizace stavu je klíčovým aspektem rozdělení kódu do menších jednodušších částí, které jsou znovu použitelné. Tyto části jsou známy jako komponenty. Všechny komponenty se nachází v souboru s příponou *.svelte* [12].

```
<script>
  let name = 'world';
</script>

<h1>Hello {name}!</h1>

<style>
  h1 {
    font-style: italic;
  }
</style>
```

Obrázek 9 – ukázka formátu souboru .svelte. Podobá se formátu HTML, kde lze přidat tag *script* a *style*. Rozdíl je v použití složených závorek *{}* pro vložení hodnoty proměnné *name*.

Svelte se drží konvencí existujících webových API a zachovává syntaxe standartního HTML, JS, CSS [13], [14]. Zároveň přináší svou vlastní syntaxi pro rozšiřující funkcionalitu. Zároveň se vyhýbá deklarativní syntaxe jako používá framework *React.js*. Např. React používá metodu *setState()*, kdežto Svelte používá jednoduše přiřazovací operátor = (*let name = 'word', Obrázek9*)[12]. Tento fakt dělá Svelte intuitivní a jednoduchý na naučení.

Známé frameworky jako React nebo Vue používají pro manipulaci dokumentu *Virtual DOM*. Virtual DOM je v podstatě zjednodušená kopie reálného DOM uložená v paměti. Aplikace pak za běhu upravuje tuto kopii (je rychlejší na úpravu), porovná rozdíl mezi skutečném a virtuálním DOM a provede změny na skutečném DOM.[12]
Na rozdíl od tradičních frameworků, Svelte Virutal DOM nepoužívá a namísto přesune co nejvíce práce do kroku při sestavování aplikace. Tento přístup má několik výhod. Zejména menší velikost výsledně sestavené aplikace, rychlejší počáteční načtení a potenciálně rychlejší aktualizace UI. Neobejde se ale i bez nevýhod. Studie od Mattias Levlin[12] ukázala, že při upravování velkého množství prvků najednou je tento přístup výrazně pomalejší oproti Virtual DOM. Naopak úprava jediného prvku je výrazně rychlejší.

4.1.3 SvelteKit

SvelteKit je nadstavba Svelte. Zatím co Svelte pomáhá vývojáři se psaním komponentů a synchronizování stavu proměnných s UI, SvelteKit se soustředí na aplikaci jako celek. Funkce, které framework poskytuje jsou navigace, před načtení stránek před použitím navigace, optimalizace sestavování, podpora off-line režimu, optimalizace obrázků... Dále framework podporuje různé typy architektur, které lze nastavit na každé stránce individuálně: SSR, SSG, SCP, SPA, MPA. Posledně *SvelteKit* používá *adaptér* pro jednoduché nasazení na jakoukoliv platformu. SvelteKit adaptér je rozšíření, které vezme soubory sestavené aplikace jako vstup a vrátí soubory připravené na nasazení pro specifickou platformu.[7]

4.1.4 Typescript

Další technologií, která byla použita při vývoji aplikace je *Typescript* (TS). Typescript je nadmnožina programovacího jazyku Javascript vydaná společností Microsoft v roce 2012. Součástí TS je kompilátor (transpilátor) *tsc* který převádí kód z jazyka Typescript do jazyka Javascript. TS především rozšiřuje Javascript o typový systém. Typový systém umožňuje přiřazení typu proměnným, funkcím a objektům za použití typové notace. Typ určuje, co tyto entity reprezentují, jejich množinu povolených hodnot a jaké operace na nich mohou být provedeny. Zatímco JS kontroluje typy při běhu aplikace, TS zjistí tyto chyby již při psaní kódu [15]. Framework Svelte oficiálně podporuje Typescript.

```
<script>
  let name: string = 'world';
</script>

<h1>Hello {name}!</h1>

<style>
  h1 {
    font-style: italic;
  }
</style>
```

Obrázek 10 - Ukázka použití typové notace Typescript ve Svelte komponentu. Proměnná *name* má přiřazený typ *string*.

Příznivci typového systému tvrdí, že „umožňuje dřívější detekci některých programovacích chyb“[16] a že vede k „lepší dokumentaci ve formě typové anotace“[17]. Zatímco kritici tvrdí, že „bylo snazší dosáhnout správného chování s kratším zdrojovým kódem, kde kód, který nebyl příliš stručný nebo obsáhlý, určoval chování, kde všechny typy mohli být převedeny do stringu pro debuggování“[18], a že „statické typování je příliš přísné a volnost dynamicky typovaných jazyků je ideální pro prototypování systémů měnícími nebo neznámými požadavky“[17].

Cílem studie od J. Bogner a M. Merkel[15] je potvrdit tvrzení, že Typescript aplikace vykazují lepší kvalitu softwaru než Javascript aplikace. Jejich analýza zahrnovala 604 projektů z platformy GitHub, které mají více jak 16 miliónů řádků. Porovnali různé aspekty kvality softwaru liší mezi Typescript a Javascript aplikaci. Výsledky studie potvrdili předpoklady o lepší kvalitě a srozumitelnosti kódu aplikace naprogramované v TS oproti JS.

4.2 Uživatelské rozhraní

4.2.1 Domovská stránka

Obrázek 14 – Domovská stránka.

Domovská stránka je velmi minimalistická. Uprostřed obrazovky se nachází oblast pro nahrání souboru. Pod oblastí je text informující o možnosti přímého vložení snímku obrazovky ze schránky.

Domovská stránka má jen dva účely – založení nové mapy nebo načtení dříve upravované mapy.

Pro otevření existující mapy stačí nahrát dříve vygenerovaný soubor s příponou *.dat*.

Pro založení nové mapy si uživatel může vybrat mezi nahrání souboru typu obrázek přes zmiňovanou oblast nebo přímém vložení snímku obrazovky ze schránky.

4.2.2 Nadřazené okno

Po otevření souboru s daty o mapě se otevře *Nadřazené okno*. Je to prvek uživatelského rozhraní, který v aplikaci obaluje dvě stránky: *Editor* a *Generátor*. Umožňuje plynulý přechod mezi těmito okny. Tento prvek se snaží replikovat klasické softwarové okno.

Nadřazené okno se skládá ze dvou hlavních částí: Titulkový pruh a obsah.

Titulkový pruh

Obrázek 15 – Titulkový pruh.

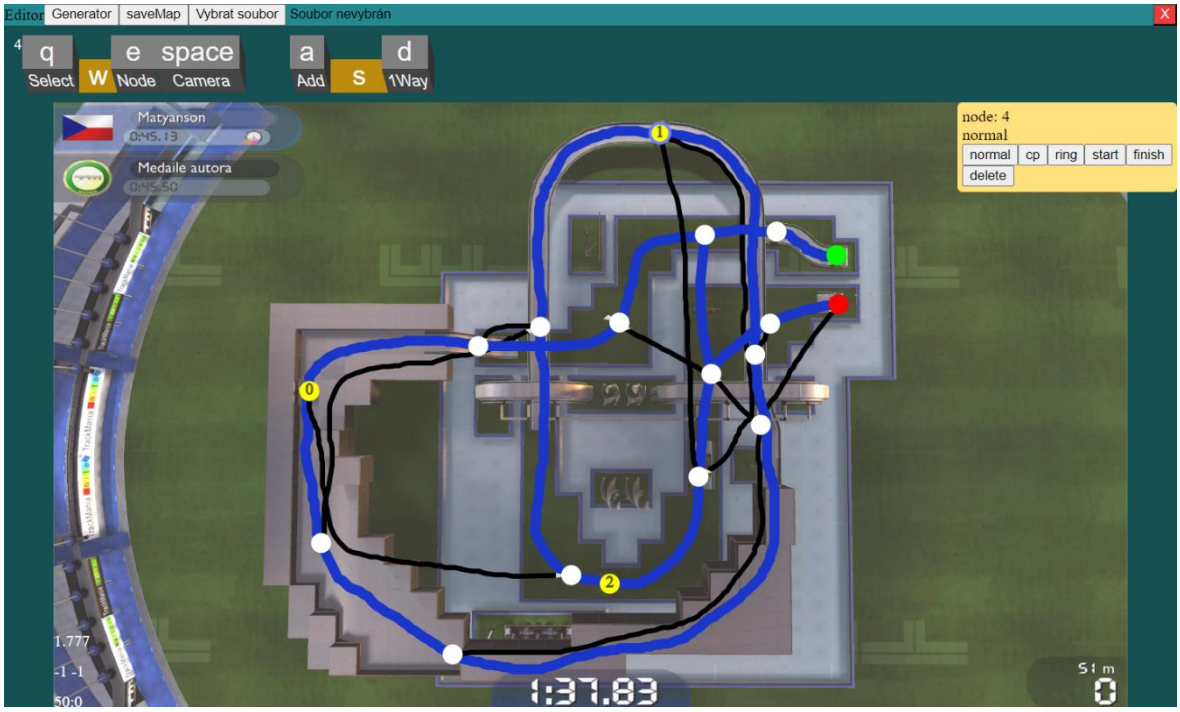
Titulkový pruh obsahuje popořadě tyto elementy:

1. Název okna – v tomto případě *Editor*
2. Tlačítko *Generátor* nebo *Editor* – přepíná obsah okna mezi generátorem a editorem
3. Tlačítko *save map* – převede aktuální mapu do souboru typu .dat a prohlížeč soubor uloží do počítače
4. *Vybrat soubor* – Pole pro načtení jiné mapy
5. Tlačítko *zavřít* – po stisknutí zavře okno a vrátí se na úvodní stránku

Obsah okna

Aktuální obsah okna. Obsah se přepíná mezi stránky Editor a Generátor, které jsou popsány níže.

4.2.3 Editor



Obrázek 16 – Editor, ve kterém je nakreslena trať.

Editor slouží pro obkreslení mapy podle obrázku a vytvoření tak reprezentace ve 2D prostoru. Pro vyhledání nejkratší cesty, aplikace potřebuje graf. Tento náčrt se

využije pro tvorbu grafu. Editor se skládá z několika sekcí, které jsou podrobněji popsány níže.

Toolbar

Obrázek 17 – Toolbar.

Toolbar slouží pro přepínání mezi nástroji a jejími variacemi. Obsahuje tlačítka se vzhledem připomínající fyzické klávesy na klávesnici. Jsou funkčně propojeny se stisknutím klávesy se stejném písmenem jako na softvérovém tlačítku. Jejich netypický vzhled má uživateli napovědět o této funkci, která má za úkol urychlit proces kreslení.

1. *Select* – Přepne na mód výběr prvků na plátně. Veškerá manipulace s prvky je zakázaná
2. *Path* – Přepne na mód kreslení cesty
    a. *Add* – zruší vybrání aktuální cesty a zahájením kreslením se vytvoří nová cesta.
    b. *Normal* – změní typ aktuální cesty na obousměrnou
    c. *1Way* – změní typ aktuální cesty na jednosměrnou
3. *Node* – Přepne na mód kreslení uzlů/vrcholů
    a. *Start* – změní typ aktuálního uzle na start
    b. *CP* – změní typ aktuálního uzle na checkpoint
    c. *RingCP* – změní typ aktuálního uzle na kruhový checkpoint
    d. *Finish* – změní typ aktuálního uzle na cíl
4. *Camera* – Přepne mód na prohlížení, tj. posun a zvětšení plátna

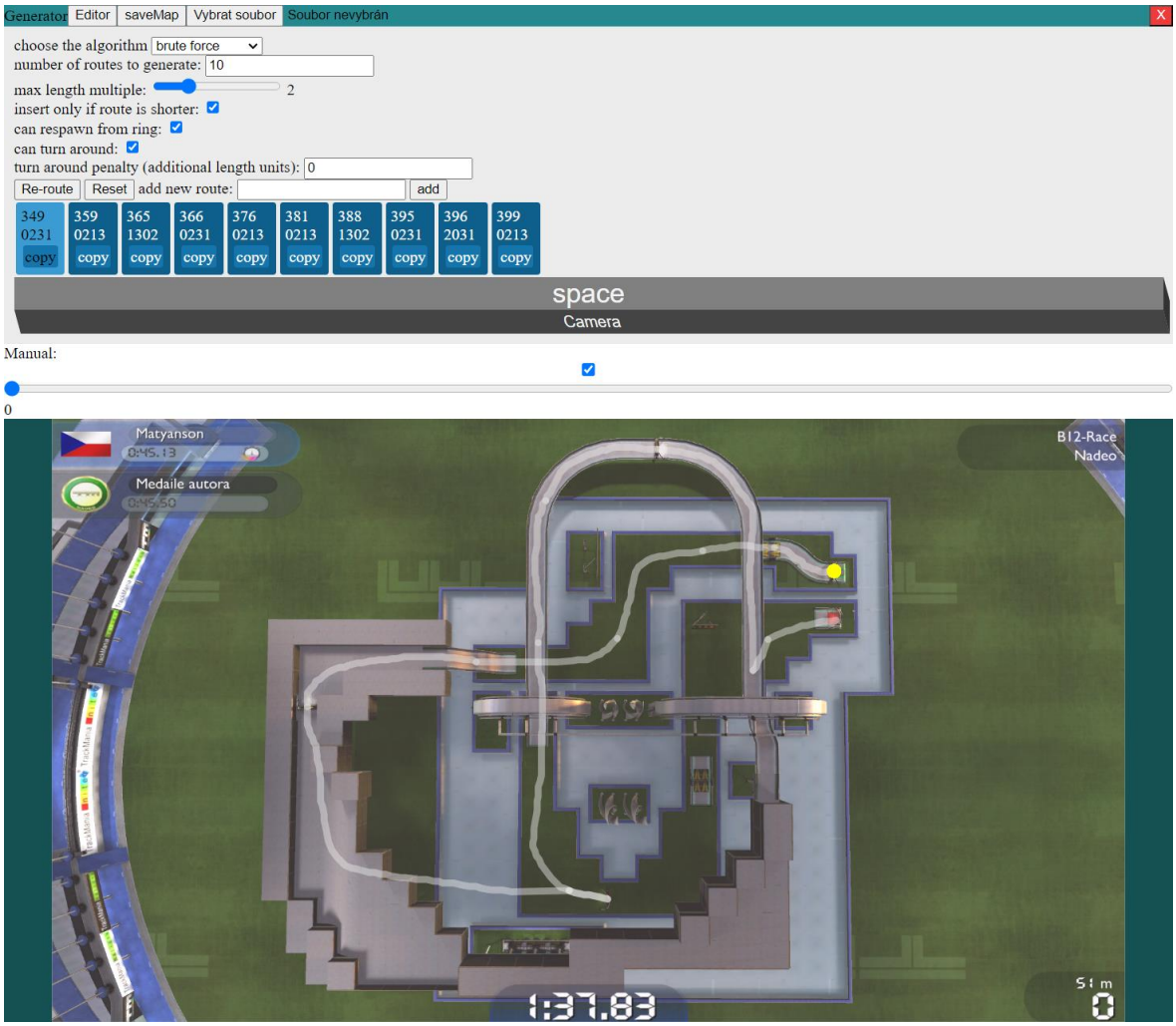
Plátno

Plátno je složeno ze tří vrstev. Základem je obrázek, na kterém je snímek mapy pohledem se shora. Na druhé vrstvě jsou zobrazeny cesty a na třetí vrchní vrstvě jsou zobrazeny uzly. Uživatel může na obrázek volně kreslit cesty a uzle reprezentující mapu ve skutečnosti.

Nevybraný	Vybraný	Typ	Název
		uzel	start
		uzel	checkpoint
		uzel	kruhový checkpoint
		uzel	křižovatka
		cesta	obousměrná cesta
		cesta	jednosměrná cesta

Obrázek 18 – Tabulka zobrazující jednotlivé prvky v editoru.

4.2.4 Generátor



Obrázek 19 – Stránka s generátorem a vizualizací cest.

Funkce této stránky je generování seznamu nejkratších cest na dříve nakresleném náčrtu tratě a zároveň jejich vizualizace.

Nastavení algoritmu

1. *Choose algorithm* – rozbalovací nabídka. Výběr algoritmu, který aplikace použije pro hledání.
2. *Number of routes to generate* – zadání čísla. Určí maximální délku seznamu vygenerovaných cest.
3. *Max length multiple* – zadání čísla. Zadané číslo reprezentuje násobek sumy délky všech cest. Algoritmus ignoruje cesty, které jsou delší než zadaný násobek.
4. *Insert only if route is shorter* – zaškrtávací pole. Pokud zaškrtnuto, algoritmus ignoruje cesty, které jsou delší než doposud nejkratší nalezená cesta.
5. *Can respawn from ring* – zaškrtávací pole. Pokud zaškrtnuto, algoritmus při hledání počítá s cesty, kde hráč využije funkce „respawn“
6. *Can turn around* – zaškrtávací pole. Pokud nezaškrtnuto, algoritmus ignoruje cesty, kde hráč zvolí opačný směr po projetí checkpointem.
7. *Turn around penalty* – zadání čísla. Určuje o kolik jednotek se uměle navýší délka cesty v případě, kdy hráč zvolí opačný směr po projetí checkpointem.

Ovládací prvky

Tato sekce ovládá chod *generátoru*.

1. Tlačítko *Re-route* – spustí generování nejkratších cest
2. Tlačítko *Reset* – zastaví generování
3. Sekce *add new route* – Stisknutím tlačítkem *add* se do seznamu nejkratších cest přidá cesta importovaná z vedlejšího textového pole

Seznam nejkratších cest

Obsahuje seznam prvků, každý reprezentuje jednu trasu. První řádek znázorňuje délku trasy, druhý řádek pořadí, ve kterém trasa vede skrz checkpointy. Na třetím řádku je tlačítko *copy*, které po stisknutí zkopíruje textovou reprezentaci trati.
Po kliknutí na jeden z prvků se v sekci display zobrazí vizualizace trasy.

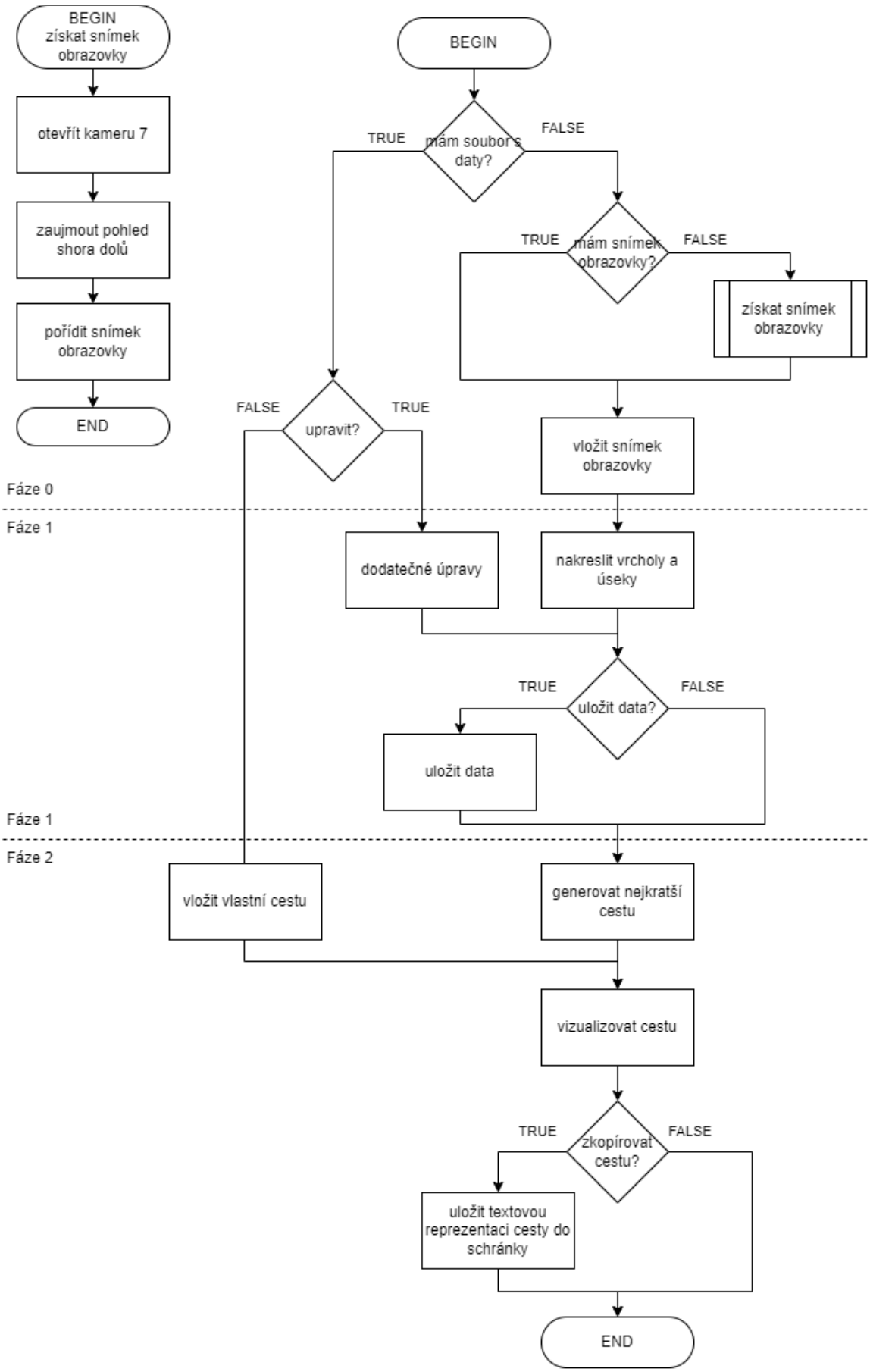
Display

Sekce *Display* slouží pro vizualizaci zvolené trasy na dříve poskytnutém snímku obrazovky. Je rozdělena na dvě části.

V horní části se ovládá animace. Zaškrtávací pole *Manual:* přepíná mezi manuálním a automatickým módem. V manuálním módu hodnota šoupátka (anglicky slider) reprezentuje čas animace. V automatickém módu hodnota reprezentuje rychlost animace.

V dolní části je vizualizace samotná. Trasa je znázorněna bílou průhlednou čarou namalovanou na obrázku a žlutý bod znázorňuje pozici formule.

4.3 Struktura a tok dat



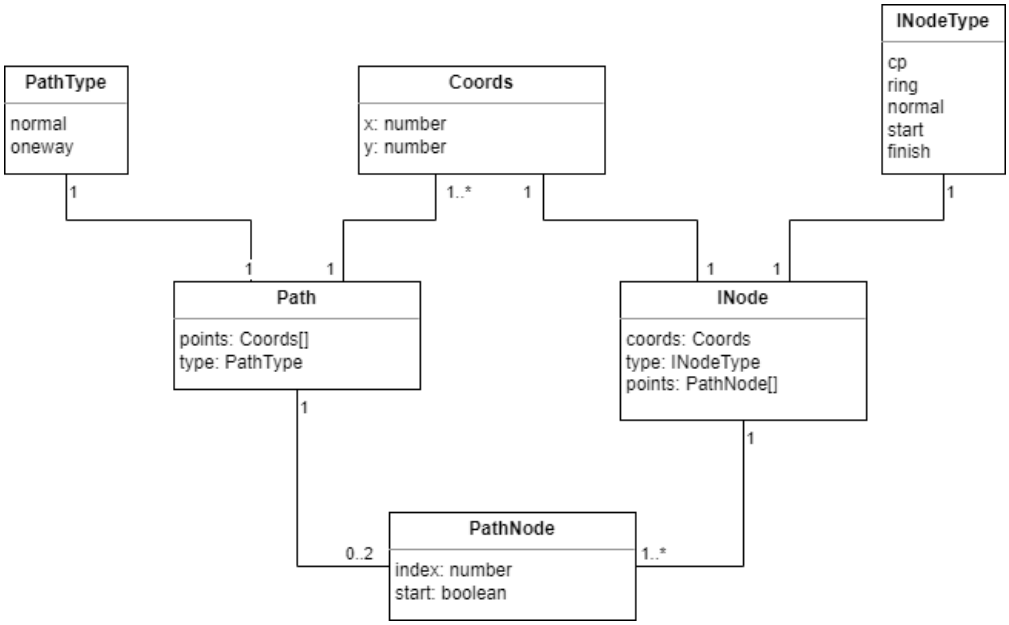
Obrázek 20 – vývojový diagram popisující postup při práci s aplikací.

Tato sekce popisuje strukturu dat a jejich změnu v průběhu použití aplikace. Interakce s aplikací vyobrazená vývojovým diagramem výše jde rozdělit do tří fází.

4.3.1 Fáze 0

Nejprve je nutné sebrat data o mapě a doplnit je do aplikace. Na začátku této fáze jsou všechny data externí v originální podobě uložené ve hře. Data mají podobu 3D prostoru ve hře vykresleného na 2D plochu obrazovky.
Uživatel pořídí snímek obrazovky a nahraje ho do aplikace.

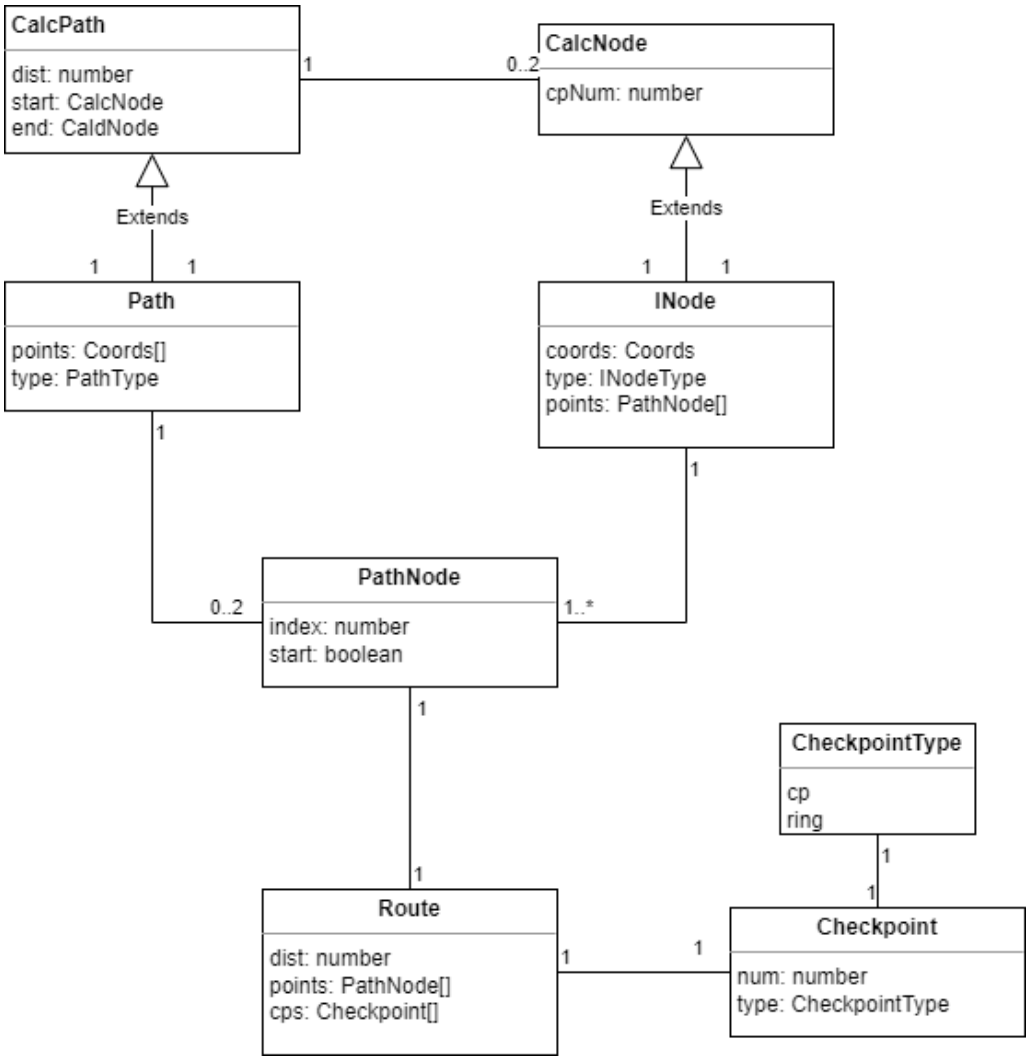
4.3.2 Fáze 1



Obrázek 21 – Entitě Relační diagram datové struktury používané aplikací ve fázi 1.

Po nahrání obrázku se otevře editor. Kromě obrázku se založí dvě pole. První pole s názvem „paths“ ukládá veškeré úseky. Druhé pole s názvem „nodes“ ukládá veškeré vrcholy. Mají typ *Path[]* a *INode[]* v příslušném pořadí. Objekt *INode* má uložen propojení s ostatními úseky ve vlastnosti *points* s typem *PathNode*.
Obrázek je uložen ve formátu *base64* pomocí *IndexedDB API*. Pole jsou uložena do *localStorage*. Hlavní výhodou je, že data přetrvává i při zavření a opětovného otevření prohlížeče. I přesto doporučuji si nakreslený graf uložit do souboru.

4.3.3 Fáze 2



Obrázek 22 - Entitě Relační diagram datové struktury používané aplikací ve fázi 2.

V této chvíli je mapa plně vyobrazena. Graf obsahuje informaci o každém checkpointu, cíli, křižovatce a úseky mezi nimi. Uživatel se přepne do generátoru a začne s generováním.

Před hledáním se pro jednodušší manipulaci s daty pole paths a nodes upraví na typy *CalcPath* a *CalcNode*. Zajistí se tak, že se určité informace nemusí počítat opakovaně. Např. každý úsek má u sebe ještě navíc uloženou jeho délku. Dále se založí nové pole „routes“ s typem *Route[]*, do kterého se budou ukládat vyhledané cesty. Cesta je uložená jako pole bodů (konců úseků). Každý bod určuje, jakým směrem cesta postupuje po úseku. Pokud hodnota *start* nabývá *false*, cesta postupuje ve směru první uložený bod -> druhý uložený bod v úseku.

4.3.4 Soubor s daty o mapě

Obrázek 23 – Struktura souboru s daty o mapě.

Uživatel má kdykoli možnost svojí práci uložit ve formě souboru, data jsou uložená v binární podobě. Soubor je rozdělený na dvě části: metadata a data.

Prvních 32 bitů reprezentuje počet složek. Dalších 32×*n* bitů je rezervováno pro čísla reprezentující velikost těchto složek, kde *n* je počet složek. Jednotlivé složky jsou uloženy v 8bit formátu. Tj. velikost o hodnotě 10 znamená 10×8 bitů.

V tomto případě platí *n=3*. Složky obsahují následující hodnoty:

1. Mime typ obrázku – Binary string
2. Obrázek – *base64* zakódovaný do *Binary string*
3. Ostatní data – *Json* zakódovaný do *Binary string*. Json obsahuje objekt s vlastnosty: `paths: Path[], nodes: INode[]`

4.3.5 Cesta v textové podobě

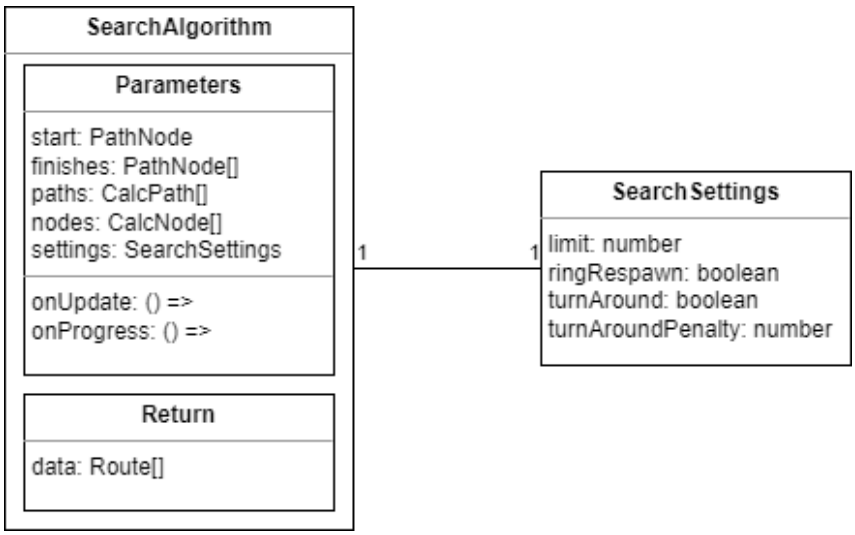
Po vygenerování seznamu cest na trati, lze pro jednoduché sdílení jednotlivé cesty uložit do schránky v textové podobě. Řetězec lze následně vložit do generátoru a cestu si prohlédnout.

Ukázka cesty: `1,E;20,E;16,E;24,E;2,E;11,E;10,E;10,S;9,S;13,S;27,S;3,S;18,E;0,E;`

V řetězci je uložen seznam bodů, ze kterých se skládá cesta. Každý bod je oddělen středníkem. V bodu je uložen index úseku (celé číslo) a označení, zda se jedná o začátek nebo konec úseku (S – začátek, E – konec). Tyto údaje jsou rozděleny čárkou.

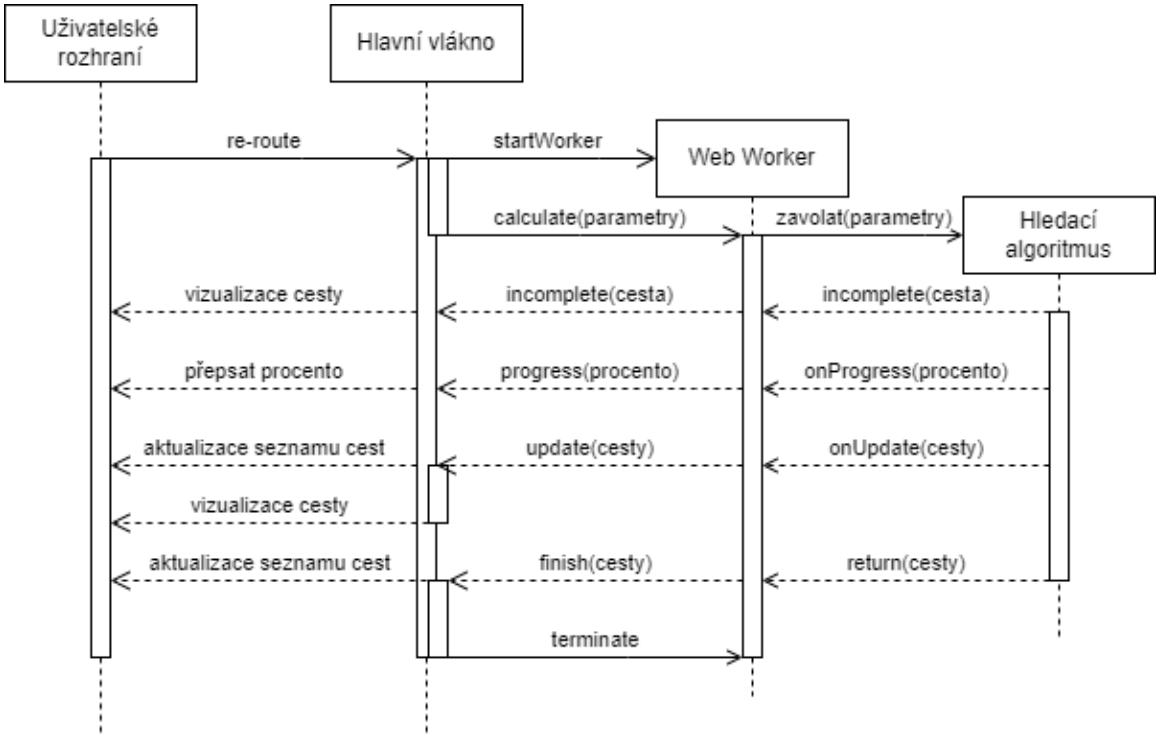
Řetězec následuje tento vzor *(index,end);+*, kde *index* je celé číslo označující index úseku a *end* je jeden ze znaků „S“ a “E“ označující začátek nebo konec úseku, respektive.

4.4 Hledací algoritmy



Obrázek 11 - Datový typ funkce hledacího algoritmu

Ve webové aplikaci jsou všechny hledací algoritmy spuštěny ve *web worker* aby nezatěžovaly uživatelské rozhraní (více kapitola 4.1.1). Web worker je od hlavního vlákna izolovaný a komunikuje s ním pomocí zpráv. Z toho důvodu každý hledací algoritmus musí odpovídat typu *Search Algorithm* a implementovat callback funkce (funkce zpětného volání) *onUpdate* a *onProgress*, které se používají pro komunikaci s web worker.



Obrázek 12 - sekvenční diagram komunikace hlavního vlákna s Web worker

4.4.1 Brute force

Funkce *continueRoute* je hlavní funkce, která je spuštěna při výpočtu algoritmem *brute force*. Jedná se o obdobu rekurzivního algoritmu prohledávání do hloubky. Cílem algoritmu je vrátit seznam nejkratších cest včetně jejich vlastností (pořadí navštívených checkpointů a délka cesty). Funkce má jediný vstup *route*, který obsahuje *points*, *dist* a *cps*. Kde *points* jsou body tvořící cestu, *dist* je délka cesty a *cps* jsou navštívené checkpointy cestou. Při první iteraci *points* nabývá hodnotu jednoho bodu reprezentující startovní blok, *dist* hodnoty 0 a *cps* prázdného pole.

Hlavní myšlenka algoritmu

Hlavní myšlenkou je začít od startovního uzlu a postupně rekurzivně prohledat všechny sousední napojené uzly, které jsou propojeny úsekem ve správném směru. Zároveň do seznamu možných dalších uzlů přidat uzel který odpovídá poslednímu prvku z pole *cps*, jelikož je možné kdykoli při závodu použít funkci *respawn*. Jakmile je při hledání splněna podmínka dokončení závodu, uložit cestu a porovnat její délku s ostatními v seznamu.

```
function continueRoute(route: Route) {
            // rozbalit cestu na body, celkovou délku, checkpointy, poslední
úsek a poslední uzel (uvedeno v tomto pořadí)
            points, dist, cps, lastPath, lastNode
            // přičti délku
            dist += lastPath.dist
            // předčasné ukončení
            IF (dist > distLimit)
                  return
            // přidej dokončenou cestu
            IF (lastNode is 'finish' AND count of cps >= cpCount) {
                  addFinalRoute({ points, dist, cps })
                  return
            }
            // odkaž na druhý konec úseků připojených k aktuálnímu uzlu
            pointsFromHere = getNextpoints(lastNode, points)
            // přidej checkpointy k navazujícím bodům
            pointsFromHere.add(findAlternativePoints(lastNode, points, cps))

            // přidej aktuální uzel ke splněným checkpointům
            IF (lastNode is 'checkpoint') {
                  cps = addCP(lastNode.cpNum)
            }

            // odstraň navazující body které jsou v protisměru
            pointsFromHere = filterNextPoints(pointsFromHere)

            // pokračuj v hledání pro navazující body
            FOR (p of pointsFromHere) {
                  continueRoute(route with p added)
            }
      }
```

4.4.2 FindAlternativePoints

Cílem algoritmu je nalézt alternativní body po kterých by mohla cesta pokračovat. Jedná se o speciální případy, kde hráč zvolí, že se otočí na místě anebo využije funkce *respawn*.

Funkce má tři vstupy: *lastNode*, *routePoints* a *cps*. LastNode je poslední uzel na cestě, routePoints je pole bodů, které tvoří cestu a *cps* je pole splněných checkpointů.

Hlavní myšlenka algoritmu

Algoritmus vytvoří nové prázdné pole, do kterého přidá body pro každý případ zvlášť podle toho, jestli vstup splňuje podmínku. Funkce pole následně vrátí. Pro oba případy platí podmínka, že cesta končí uzlem, který je typem checkpoint, a ještě není splněný. A to z toho důvodu, protože je to jediný způsob, jak může být cesta nejkratší, pokud jsou zahrnuty tyto speciální případy.

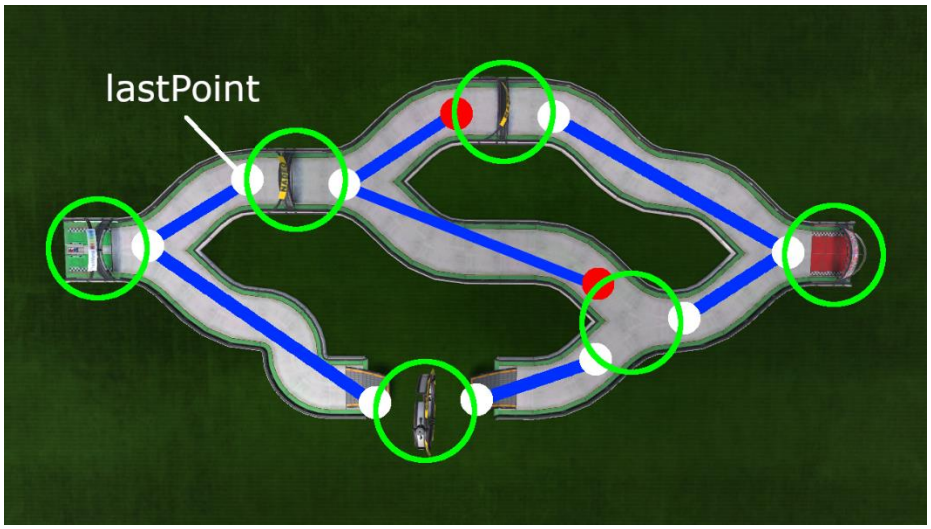
```
findAlternativePoints(lastNode, routePoints, cps) {
	// získat poslední bod ze seznamu
	lastPoint = routePoints[last]
	// definovat pole do kterého bude uložen výsledek
	nextPoints = []
	// pokračovat jen pokud je poslední uzel checkpoint a není v seznamu
	IF (lastNode is 'checkpoint' AND lastNode is not in cps) {
		pathRepetition = getPathRepetition(currentPoints)
		// přidat cestu zpátky (180° otočení)
		IF (settings.turnAround AND pathRepetition < 3) {
			otherEnd = { ...lastPoint, start: !lastPoint.start }
			nextPoints.add(otherEnd)
		}
		// přidat navazující body od kruhového checkpointu
		IF (settings.ringRespawn AND lastNode is 'ring') {
			pointsFromRing = getPointsFromRingRespawn(routePoints)
			nextPoints.add(...pointsFromRing)
		}
	}
	return nextPoints
}
```

4.4.3 getNextPoints

Úkolem tohoto algoritmu je rozhodnout jaké body navazují na vrchol ve směru cesty. Mezi výsledek není započítaný bod, který se vrací po stejném úseku zpátky. Jeho vstupem je *node* a *routePoints*. Node je vrchol, ze kterého jsou hledány navazující body. RoutePoints jsou body ze kterých je stvořena cesta.

Hlavní myšlenka algoritmu

Algoritmus vezme seznam bodů napojených k *node*. Ze seznamu odstraní bod, který je na stejném úseku jako poslední bod cesty. Každý bod ze seznamu nahradí bodem na opačné straně úseku. Vrátí výsledný seznam.



Obrázek 13 - příklad algoritmu getNextPoints. Bod označený „lastPoint“ je poslední bod na cestě. Červeně označené body jsou vrácené body funkcí getNextPoints.

```
function getNextPoints(node, routePoints) {
	// získat poslední bod ze seznamu
	lastPoint = routePoints[last];
	// definovat pole do kterého bude uložen výsledek
	nextPoints = [];
	FOR (p of node.points) {
		// přeskočit cestu zpátky
		IF (p is lastPoint) continue;
		// uložit opačný konec úseku do pole
		oppositeEnd = { ...p, start: !p.start }
		nextPoints.add(oppositeEnd)
	}
	return nextPoints
}
```

4.4.4 AddFinalRoute

Funkce *addFinalRoute* má za úkol přidat cestu do seznamu výsledných nejkratších cest. Tento seznam je uložený v globální proměnné *finalRoutes*. Zároveň je důležité tento seznam seřadit podle délky. Funkce docílí obou za použití třídícího algoritmu

insertion sort (řazení vkládáním). Po dokončení vkládání funkce zavolá *onUpdate*, díky které se pošle zpráva hlavnímu vláknu a seznam se zobrazí na UI.

```
function addFinalRoute(route) {
	// získat délku z route
	{ dist = 0 } = route;
	// najít vhodnou pozici v seznamu
	i = 0
	FOR (i to finalRoutes.length) {
		IF (dist < finalRoutes[i].dist)
			break;
	}
	// vložit do seznamu podle pozice
	insertToIndex(i, route)
	// aktualizovat maximální povolenou délku
	distLimit = finalRoutes[last].dist
	// poslat nový seznam na hlavní vlákno
	onUpdate(finalRoutes)
}
```

4.4.5 Testování algoritmů

Tato sekce se zaměří na testování algoritmů pro hledání nejkratší cesty na mapách hry Trackmania. Cílem je ověřit, zda nalezená cesta je kratší než cesta zamýšlená a zároveň vyhodnotit časovou náročnost výpočtu. Výsledky testování poskytne informace o praktičnosti využití aplikace na reálných případech.
Celkem byly testovány dva algoritmy (*brute force* a *reverse search*) na třech mapách a výkon byl testován na dvou zařízeních. První zařízení je relativně starý notebook a druhé zařízení je výkonnější stolní počítač:

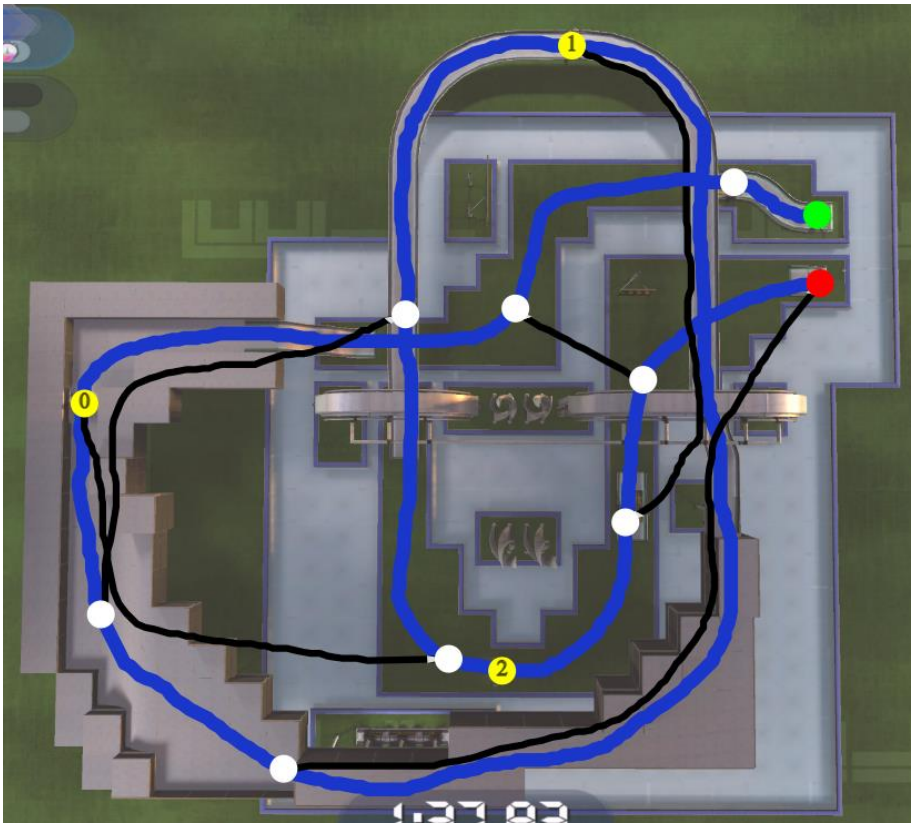
Název	**Informace o procesoru**
zařízení1	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz
zařízení2	Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz 3.60 GHz

Délka cesty není uvedena v konkrétní jednotce. Délka je měřena vůči šířce plátna na kterém byla mapa nakreslena, přičemž jedna jednotka odpovídá 1/100 šířky plátna.

Testování probíhalo následovně:

1. Nakreslit zamýšlenou cestu v editoru
2. Přidat zkratky, které lze na mapě provést. (např. seskok z cesty na níže umístěnou cestu)
3. Spustit vyhledávání nejkratších cest.
4. Zaznamenat si počet nalezených cest kratších, než je cesta zamýšlená a délku nejkratší z nich.
5. Otestovat dobu výpočtu pro oba algoritmy na obou zařízeních.

První testovaná mapa



Obrázek 14 - Ukázka mapy B12 v editoru.

První testovaná mapa má název *B12*. Jedná se o oficiální mapu postavenou vývojáři hry Trackmania.

Výsledky hledání:

Poč. uzlů	Poč. úseků	Délka zamýšlené cesty	Nejkratší nalezená cesta
12	16	478	395

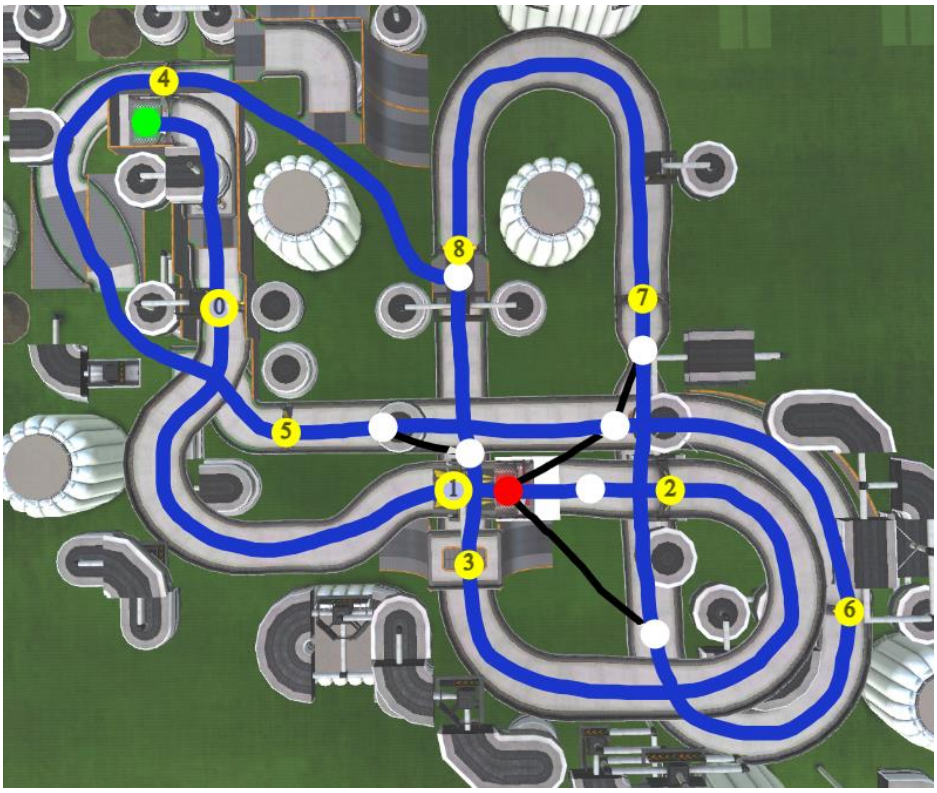
Na mapě B12 byly nalezeny kratší cesty než cesta zamýšlená pomocí obou algoritmů. Oba algoritmy se shodují v nalezení 2 nejkratších cest, kdežto *brute force* nalezlo 6 dalších, které jsou stále kratší než namýšlená cesta. Z těchto výsledku vyplývá zkrácení o 17 %.

Výsledky výkonu:

	Brute force	Reverse search
Zařízení1	136 ms	12 ms
Zařízení2	25 ms	6 ms

Výsledky ukázaly, že oba algoritmy dokončily výpočet v příznivém čase, přičemž algoritmus *reverse search* dosáhl výrazně kratšího času na obou zařízeních. Dále si můžeme všimnout očekávaného výsledku, kde *zařízení2* dokončilo hledání rychleji než *zařízení1*.

Druhá testovaná mapa



Obrázek 15 - Ukázka mapy *Turbulence*

Druhá testovaná mapa má název *Turbulence*. Zamýšlená cesta projíždí některými části mapy více než jednou. Mapa je populární mezi hráči díky velkému počtu nalezených zkratek. Byla vytvořena hráčem *matto* v roce 2009.

Výsledky hledání:

Poč. uzlů	Poč. úseků	Délka zamýšlené cesty	Nejkratší nalezená cesta
17	21	657	497

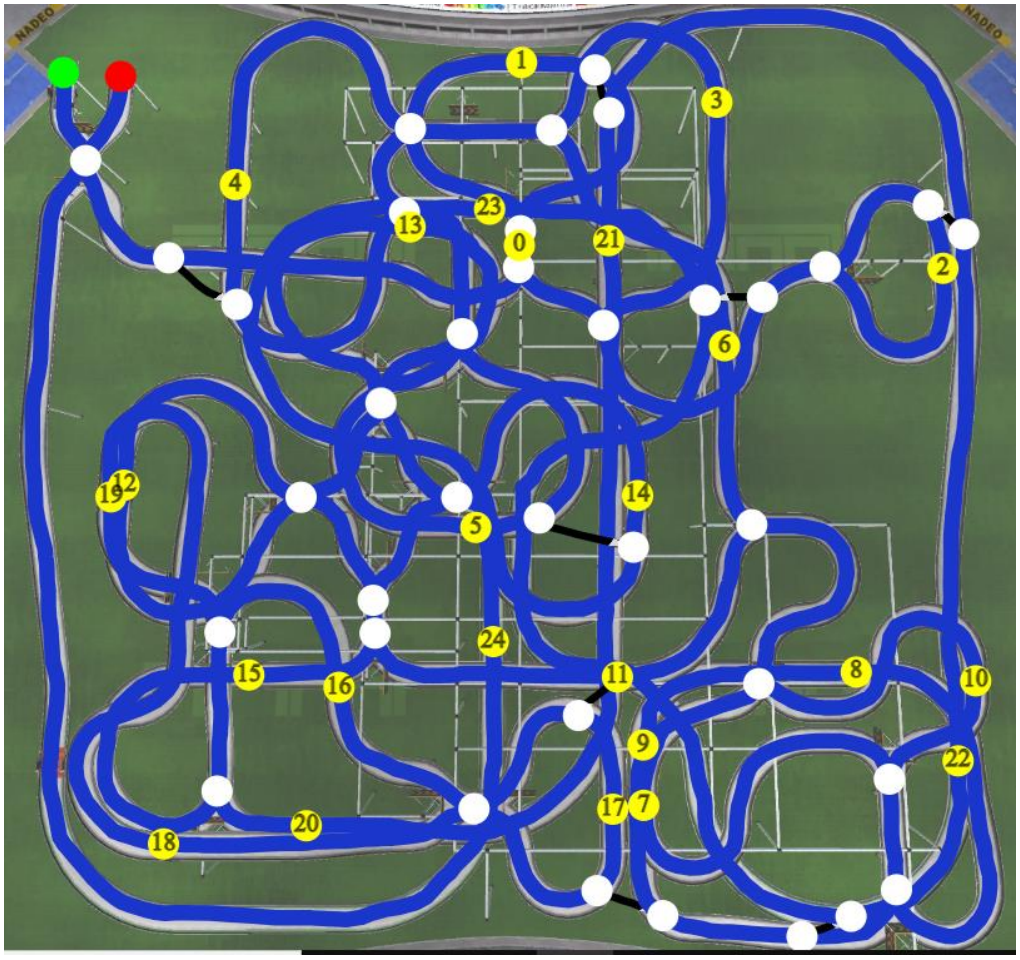
Oba algoritmy se opět shodly na nejkratší nalezené cestě. Tentokrát algoritmus *brute force* našel 73 a *reverse search* 9 cest kratších, než je cesta zamýšlená. Z výsledků lze posoudit 24% zkrácení.

Výsledky výkonu:

	Brute force	Reverse search
Zařízení1	571 ms	4 157 ms
Zařízení2	245 ms	2 418 ms

Výsledky potvrzují trend zhruba poloviční doby výpočtu zařízení2 na rozdíl od zařízení1. Naopak od minulého testu byl algoritmus *reverse search* výrazně pomalejší než *brute force* s průměrnou dobou v měřítku několika sekund.

Třetí testovaná mapa



Poslední testovanou mapou je *E02*. E02 je velmi náročná mapa a jedna z posledních v oficiální kampani. S počtem checkpointů 25 je to jedna z největších a nejsložitějších oficiálních map. Z tohoto hlediska je to ideální kandidát pro účel hledání nejkratší cesty.

Výsledky hledání:

Poč. uzlů	Poč. úseků	Délka zamýšlené cesty	Nejkratší nalezená cesta
62	83	2166	1555

Algoritmus *reverse search* našel více než 100 kratších cest, zatím co *brute force* nebyl schopný vyhledat ani jednu cestu v rozumné časové době. Výsledky tohoto testu vykazují zatím nejvyšší zkrácení délky a to o 28 %.

Výsledky výkonu:

	Brute force	Reverse search
Zařízení1	-	~11 000 ms
Zařízení2	-	~20 000 ms

Při posledním testu byl algoritmus *brute force* na zařízení2 spuštěný déle než čtyři hodiny a přesto nebyl schopný vyhledat jedinou cestu. Algoritmus *reverse search* byl schopný najít kratší cesty v rozmezí desítek vteřin.

5 Shrnutí a diskuse výsledků

Byly testovány tři různé mapy s různými vlastnostmi a složitostí, na kterých byly použity dva algoritmy: *brute force* a *reverse search*.

Z výsledků zle odvodit praktičnost algoritmů v různých situacích. Při malém počtu uzlů a úseků jsou oba algoritmy schopny vyhledat nejkratší cestu. Jelikož algoritmus *brute force* prochází všechny možné cesty, jeho výstup garantuje nalezení nejkratší cesty a zároveň nabízí více možností pro uživatele. Z výsledků lze též vyhodnotit korelace mezi nárůstem počtu uzlů a úseků a zvýšením doby výpočtu. Při dosažení dostatečně velkého počtu se doba výpočtu ale stává nepraktickou. V této situaci má algoritmus *reverse search* navrch.

Výsledky lze považovat za úspěšné, jelikož algoritmy nalezly v průměru cestu, která je o 23% kratší než cesta zamýšlená autorem mapy.

6 Závěry a doporučení

Cílem této práce bylo naprogramovat aplikaci pro hledání nejkratší cesty na jakékoliv trase ve hře Trackmania.

Hlavní překážkou bylo implementovat algoritmus, který funguje pro všechny typy map, dokáže replikovat funkci *respawn* a zároveň je schopný hledat i na složitějších mapách s velkým množstvím křižovatek a checkpointů.

Dalším problémem byl jednoduchý přechod z herní mapy na reprezentaci grafem. Jako řešení jsem zvolil 2D editor, kde se všechny klíčové části obkreslí na obrázku mapy se shora. Tento přístup má ale svoje limitace. Není možné pořídit snímek, když je trasa překrytá střechou, některé části trasy mohou být překryté cestou ve vzduchu, sklopenou cestu nelze rozeznat od vodorovné a pro přesné měření je nutné brát v potaz perspektivní projekce kamery.

Alternativní přistup by mohl být automatické sestavení grafu ve 3D prostoru pomocí dat v souboru, ve kterých jsou mapy uloženy.

Jsem celkově spokojený s funkcemi, které aplikace nabízí. Aplikace je kompatibilní s většinou map hry Trackmania, jak jsme si ověřili při testování, je schopná vyhledat jednu z nejkratších cest a alternativní možnosti, a navíc sdílení vytvořeného obsahu.

7 Seznam použité literatury

[1] K. Mehlhorn a P. Sanders, *Algorithms and data structures : the basic toolbox*. Springer, 2007. Viděno: 1. leden 2024. [Online]. Dostupné z: https://people.mpi-inf.mpg.de/~mehlhorn/ftp/Mehlhorn-Sanders-Toolbox.pdf

[2] David Samuel Johnson a L. A. McGeoch, „The traveling salesman problem":, *Princet. Univ. Press EBooks*, s. 215–310, 2018, doi: 10.2307/j.ctv346t9c.13.

[3] „Computers and Intractability; A Guide to the Theory of NP-Completeness: | Guide books | ACM Digital Library". Viděno: 12. duben 2024. [Online]. Dostupné z: https://dl.acm.org/doi/10.5555/574848

[4] „Get started with Vercel", vercel.com. Viděno: 11. duben 2023. [Online]. Dostupné z: https://vercel.com/docs/getting-started-with-vercel

[5] „What is serverless computing? | Serverless definition". Viděno: 11. duben 2024. [Online]. Dostupné z: https://www.cloudflare.com/learning/serverless/what-is-serverless/

[6] „Deploying Git Repositories with Vercel". Viděno: 11. duben 2024. [Online]. Dostupné z: https://vercel.com/docs/deployments/git

[7] „Glossary • Docs • SvelteKit". Viděno: 11. duben 2024. [Online]. Dostupné z: https://kit.svelte.dev/docs/glossary

[8] „Web Storage API - Web APIs | MDN". Viděno: 11. duben 2024. [Online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

[9] „IndexedDB API - Web APIs | MDN". Viděno: 11. duben 2024. [Online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[10] „An overview of web workers", web.dev. Viděno: 11. duben 2024. [Online]. Dostupné z: https://web.dev/learn/performance/web-worker-overview

[11] „Using Web Workers - Web APIs | MDN". Viděno: 11. duben 2024. [Online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

[12] M. Levlin, A. Soini, a D. Truscan, „DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte", 2020. Viděno: 20. duben 2024. [Online]. Dostupné z: https://www.semanticscholar.org/paper/DOM-benchmark-comparison-of-the-front-end-React%2C-Levlin-Soini/52466a0b58aa2a71857c487842e2cf9b8acf380f

[13] „Docs • Svelte", svelte.dev. Viděno: 1. leden 2023. [Online]. Dostupné z: https://svelte.dev/docs

[14] V. Joshi a B. AshwiniK, „Svelte.js for modern front-end development", *J. Emerg. Technol. Innov. Res.*, čer. 2020, Viděno: 20. duben 2024. [Online]. Dostupné z: https://www.semanticscholar.org/paper/Svelte.js-for-modern-front-end-development-Joshi-AshwiniK./d61cda19397d88026d21e103ec47e5c481e24698

[15] J. Bogner a M. Merkel, „To type or not to type? A systematic comparison of the software quality of JavaScript and TypeScript applications on GitHub", 2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR). [Online]. Dostupné z: https://api.semanticscholar.org/CorpusID:247594541

[16] B. C. Pierce, *Types and Programming Languages*, 1st edition. Cambridge, Mass: MIT Press, 2002.

[17] E. Meijer, „Dynamic typing when needed: The end of the cold war between programming languages", 2004. Viděno: 21. duben 2024. [Online]. Dostupné z: https://www.semanticscholar.org/paper/Dynamic-typing-when-needed%3A-The-end-of-the-cold-war-Meijer/e670c72afa489ffdfab77852a0c9781e082fc266

[18] R. Loui, „In Praise of Scripting: Real Programming Pragmatism", *Computer*, roč. 41, s. 22–26, srp. 2008, doi: 10.1109/MC.2008.228.

8 Přílohy

1. Zdrojový kód projektu: application.zip
   (dostupné také na https://github.com/Matyanson/TMCutFinder2)
2. Složka s testovacími mapami: maps.zip
3. Složka s výsledky testů: tests.zip

Běžící aplikace je dostupná na adrese https://tm-cutfinder.vercel.app/

Příloha č. 1

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Matyáš Peremský**

Studium: I2100261

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Webová aplikace pro hledání cest na herní mapě**

Název bakalářské práce AJ: Web application for finding the shortest route on a game map

Cíl, metody, literatura, předpoklady:

Cílem práce je naprogramovat webovou aplikaci pro hledání nejkratších cest na herní mapě Trackmania Nations Forever. Aplikace bude obsahovat hlavní stránku, editor, vizualizaci cest a vyhledávač cest.

Technologie:
Webová aplikace bude tvořena v jazyce Typescript pomocí frameworku Svelte.js, pro vzhled GUI se použijí vlastní css styly a stránky budou hostované na platformě Vercel. Při tvorbě v editoru se bude mapa průběžně ukládat v paměti prohlížeče pomocí LocalStorage a Indexed Database API. Kód pro výpočet cest bude spouštěn ve Web Workeru, aby nezatěžoval základní funkce stránek.

Funkce:
Uživatel ve hře nastaví kameru na pohled ze shora, tak aby byla v záběru celá mapa a vytvoří snímek obrazovky. Snímek následně vloží do webové aplikace, obkreslí cesty, zvýrazní start, cíl a checkpointy. Dále se uživatel přepne do režimu hledání a může spustit algoritmus hledání nejkratších cest ze startu do cíle. Výsledek pak lze stáhnout pro pozdější úpravy.

-David Samuel Johnson and L. A. McGeoch, "The traveling salesman problem"
-Kan Jun-man and Z. Yi, "Application of an improved ant colony optimization on generalized traveling salesman problem"
-Y. Wang and Z. Han, "Ant colony optimization for traveling salesman problem based on parameters optimization"
-M. Levlin, A. Soini, and Dragos Truscan, "DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte"
-V. Joshi and B AshwiniK, "Svelte.js for modern front-end development"
-J. Bogner and M. Merkel, "To type or not to type? A systematic comparison of the software quality of JavaScript and TypeScript applications on GitHub"

Zadávající pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: Mgr. Vojtěch Vorel, Ph.D.

Oponent: Ing. Karel Malý, Ph.D.

Datum zadání závěrečné práce: 26.1.2021