Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Výučbové scenáre v rozšírenej realite využívajúce procesné grafy

Systémová príručka

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Meno učiteľa:	Ing. Štefan Korečko, PhD.

Košice 2022 **Bc. Adam Kašela**

Obsah

Zoznam obrázkov

1 Funkcia programu

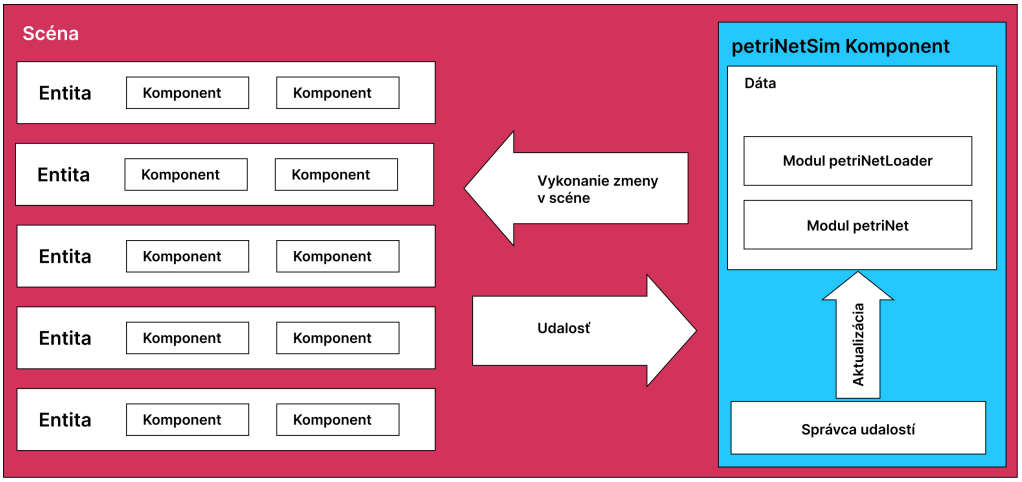
Cieľom tohto projektu bolo vytvorenie konkrétnej výučbovej aplikácie, využívajúcej technológie rozšírenej reality. Aplikácia je navrhnutá ako múzejná expozícia, kde používatelia môžu plniť úlohy a dozvedieť sa zaujímavé informácie o vystavených exponátoch vo virtuálnom prostredí. Riadenie scény je implantované na základe navrhnutej Petriho siete. Riešenie je založené na základe učiva dejepisu pre žiakov 7. ročníka ZŠ alebo 2. ročníka gymnázia s osemročným štúdiom. Ako aplikačná platforma sa využil A-Frame. Táto technológia zabezpečila jednak jednoduchú dostupnosť virtuálnych prostredí priamo vo webovom prehliadači bez nutnosti inštalácie dodatočného softvéru. Program je implementovaný v jazyku JavaScript s využitím ES6 modulov.

2 Popis programu

V tejto časti sa venujeme popisu riešenia. Popisu dôležitých algoritmov, údajových štruktúr a globálnym premenným.

2.1 Popis riešenia

Riešenie je navrhnuté pomocou vývojového rámca A-Frame. A-Frame poskytuje deklaratívnu, zostaviteľnú a opakovane použiteľnú architektúru entita-komponent-systém. Architektúra ECS je bežným a často používaným vzorom pri návrhu 3D aplikácií a vývoji. Naše riešenie je nadstavba na A-Frame rámcom, uskutočnená pomocou pomocou Javascriptu s využitím ES6 modulov. Riadenie scény je uskutočnené pomocou Petriho siete. Preto bolo potrebné na základe A-Frame architektúry navrhnúť naše riešenie. Obrázok 2.1 znázorňuje architektonické riešenie založené na A-Frame ECS architektúre. Architektúra využíva JavaScript a DOM API na komunikáciu medzi A-Frame komponentmi. Hlavný Komponent je **petriNetSim** (modrá farba), ktorý zaobaľuje celú funkcionalitu nad Petriho sieťou. Tento komponent je deklarovaný na entite scény a počúva zverejnené udalosti ostatných komponentov, ktoré súvisa s návrhom scény.



Obr. 2.1: Architektúra riešenia

2.2 Štruktúra projektu

Nasledujúci obr. 2.2 znázorňuje výslednú štruktúru projektu. Riešenie je rozdelené na základe bežnej klientskej webovej aplikácie. Kde HTML dokumenty sú v priečinku **src/views**. Riešenie logiky je rozdelené v priečniku **src/js**, kde sú komponenty a moduly použité vo výslednom riešení. **src/models** je JavaScript objekt, obsahujúci všetky možné udalosti scény.

Obr. 2.2: Štruktúra riešenia klientskej aplikácie.

2.3 Popis algoritmov a údajových štruktúr, globálnych premenných

V tejto kapitole sa venujeme popisu jednotlivých algoritmov, údajových štruktúr použitých v riešení aplikácie, ktorej štruktúra je zobrazená v predchádzajúcej kapitole.

2.3.1 Popis modulov

Pri implementácii sme využili ES6 moduly, ktoré zaobaľujú funkcionalitu na spravou Petriho siete. Riešenie obsahuje 2 moduly, ktoré využívajú objekt Petriho siete:

Údajové štruktúry použité v moduloch

- **Place** - JavaScrip objekt predstavujúci miesto Petriho siete.
  - *Typ:*
    - {id: string, name: string, marking: number}
- **Transition** - JavaScrip objekt predstavujúci prechod Petriho siete.
  - *Typ:*
    - {id: string, name: string}
- **Arc** - JavaScrip objekt predstavujúci hranu Petriho siete.
  - *Typ:*
    - {id: string, markingWeight: number, source: string, target: string}
- **PNet** - JavaScrip objekt predstavujúci Petriho sieť. Objekt obsahuje polia miest, prechodov a hrán.
  - *Typ:* - Objekt
    - places (Array<Place>)
    - transitions (Array<Transition>)
    - arcs (Array<Arc>)

Modul petriNet.mjs

Obsahuje inštanciu triedy **PetriNet**. V triede **constructor()** je miesto, kde je môžete inicializovať vlastnosti inštancie. Trieda obsahuje jednu vlastnosť a to objekt Petriho siete - **netData**, ktorý je typu údajovej štruktúry **PNet** popísanej vyššie. Funkcie, ktoré umožňuje trieda nad objektom **netData** vykonávať sú:

- **fire(transitionName)** - Metóda na odpálenie prechodu. Keď je možné odpáliť prechod, tak aktualizuje značenie Petriho siete.
  - *Parameter:*
    - transitionName (string) - meno prechodu
- **isEnabled(transitionName: string)** - Metóda skontroluje, či je možné odpálenie špecifikovaného prechodu.
  - *Parameter:*

        * transitionName (string) - meno prechodu

- **getMarking(placeName: string)** - Metóda vráti značenie špecifikovaného miesta.
  - *Parameter:*
    * placeName (string) - meno miesta
  - *Vracia:*
    * (number) - značenie miesta
- **isMarked(placeName: string, arcWeight: number)** - Metóda skontroluje, či miesto ma dostatočný počet tokenov, ktorý musí byť rovný alebo väčší hodnote *arcWeight*.
  - *Parameter:*
    * placeName (string) - meno miesta
    * arcWeight: (number) - Váha hrany
- **updateMarking(connectedArcs: Array<Arc>)** - Metóda aktualizuje objekt netData na nasledujúce značenie.
  - *Parameter:*
    * connectedArcs (Array<Arc>) - Pole hrán
- **findPlace(placeName: string): Place** - Metóda vráti špecifikované miesto podla mena.
  - *Parameter:*
    * placeName (string) - Meno miesta
  - *Vracia:*
    * Place - objekt miesta
- **findTransition(transitionName: string): Transition** -Metóda vráti špecifikovaný prechod podla mena.
  - *Parameter:*
    * transitionName (string) - Meno prechodu
  - *Vracia:*
    * Transition - objekt prechodu

- **findAllInputPlaces(transitionName: string): Array<{Place, arcWeight: number}>**
  - *Parameter:*
    - transitionName (string) - Meno prechodu
  - *Vracia:*
    - Array<{Place, arcWeight: number}> - Vráti pole miest, ktoré poskytujú vstup pre prechod
- **findAllConnectedArcs(transitionName: string): Array<Arc>**
  - *Parameter:*
    - transitionName (string) - Meno prechodu
  - *Vracia:*
    - Array<Arc> - Vráti pole hrán, ktoré sú spojené s vybratým prechodom
- **consumeInputTokens(placeIndex: number, markingWeight: number)** - Keď sa prechod odpáli, spotrebuje požadované vstupné tokeny.
  - *Parameter:*
    - placeIndex (number) - Index miesta
    - markingWeight (number) - Počet žetónov, ktoré sa spotrebujú
- **createOutputTokens(placeIndex, markingWeight)** - Keď sa prechod odpáli, vytvorí na svojich miestach požadované výstupné žetóny.
  - *Parameter:*
    - placeIndex (number) - Index miesta
    - markingWeight (number) - Počet žetónov, ktoré sa vytvoria
- **isInputPlace(placeName: string, transitionName: string)** - Skontroluje, či miesto napája špecifický prechod.
  - *Parameter:*
    - placeName (string) - Meno miesta
    - transitionName (string) - Meno prechodu

Modul petriNetLoader.mjs

Obsahuje metódy, ktoré načítavajú Petriho sieť vo formáte PNML. Modul exportuje jedinú metódu **loadXMLDoc**. Metódy sú:

- **loadXMLDoc(url: string)** - Metóda zabezpečujúca načítanie Petriho siete.
  - *Parameter:*
    - url (string) - Špecifická URL adresa, kde je uložená Petriho sieť vo formáte PNML.
  - *Vracia:* - Vráti objekt PNet.
- **xmlToJson(xml: XMLDocument)** - Metóda vracia objekt parsovaný dokument XML.
  - *Parameter:*
    - xml (XMLDocument) - XML dokument, ktorý sa má parsovať. Dedí z generického dokumentu a nepridáva k nemu žiadne špecifické metódy alebo vlastnosti.
  - *Vracia:* - Vráti parsovaný XML dokument na objekt.

2.3.2 Popis komponentov

Navrhnuté komponenty sme požili pri implementácii scény. Komponenty sú vytvorené na základe A-Frame dokumentácie.

- **petriNetSim.component.js** - Komponent sleduje udalosti, ktoré súvisia s odpálením prechodov Petriho siete. Komponent zaobaľuje moduly **petriNet** a **petriNetLoader**.
- **collisionDetectorEventHandler.component.js** - Komponent deteguje kolízie nad entitou, ktorá ho deklaruje. Je možné špecifikovať udalosti, ktoré budú odpálené pri kolízii a ukončení kolízie s entitou.
- **clkSingleEventHandler.component.js** - Komponent sleduje udalosť kliknutia nad entitou deklarujúcou tento komponent. Po kliknutí na entitu, vyšle udalosť komponentu **petriNetSim**.
- **clkMultiEventHandler.component.js** - Komponent sleduje udalosť kliknutia nad entitou deklarujúcou tento komponent. Po kliknutí na entitu, strieda odoslanie udalostí, ktoré sú špecifikované pri použití komponentu.

- **toggleInfo.component.js** - Komponent nastavuje špecifickej entite viditeľnosť v scéne po kliknutí na entitu deklarujúcu tento komponent.

2.4 Popis vstupných a výstupných súborov

Naše riešenie negeneruje ani nevytvára žiadne výstupné súbory. Vstupné súbory sú umiestnené v priečinku **assets**. Tieto súbory sú potrebné pre fungovanie celej scény. Priečinok obsahuje:

- **3dModels** - Priečinok obsahujúci všetky 3D modely.
- **fonts** - Priečinok obsahujúci vlastné použité písma.
- **images** - Priečinok obsahuje obrázky a textúry použité na dotvorenie VR scény.
- **petriNetFile** - Priečinok obsahuje testovacie ale aj finálnu použitú Petriho sieť. Sieť je vo formáte PMNL.
- **sounds** - Zvuky použité v scéne.
- **videos** - Priečinok obsahuje video návod pre používanie VR scény.

3 Použité technológie

Pre vývoj tejto webovej aplikácie sme využili technológie, ktoré nám pomohli pri vývoji. Tieto technológie sú potrebné pre nasledujúci vývoj a testovanie aplikácie.

3.1 Programové prostriedky

V súčasnosti takmer všetky webové stránky obsahujúce JavaScript, bežia na webovom prehliadači návštevníka. Preto je potrebné pre nasledujúci vývoj mať nainštalovaný akýkoľvek dostupný webový prehliadač. Pri našom testovaní sme používali Microsoft Edge. A-Frame podporuje VR pre akýkoľvek prehliadač, ktorý implementuje špecifikáciu WebXR, a ploché 3D pre väčšinu prehliadačov. Veľkí predajcovia prehliadačov pomaly prechádzajú na špecifikáciu WebXR, aj keď nemá veľa zmien na prednej strane vývojárov A-Frame, ktoré zahŕňajú väčšinou premenovanie API. Ďalšie dostupné dostupné a podporované prehliadače sú:

- Supermedium
- Firefox
- Oculus Browser
- Samsung Internet
- Chrome
- Exokit

Pre vývoj a implementáciu sme potrebovali vývojové prostredie. V našom prípade sme využili Visual Studio Code ale je možné použiť akékoľvek vývojové prostredie.

3.2 Využité technológie

Pre vývoj aplikácie boli využité technológie, ktoré sú nevyhnutnou súčasťou implementácie. Všetky hlavné technológie sú uvedené v nasledovnom zozname aj s príslušnou verziou.

- A-Frame (1.0.4) - Webový rámec na vytváranie 3D/AR/VR zážitkov.
- Node.js (8.16.0) - Softvérový systém navrhnutý pre vytváranie vysoko škálovateľných webových aplikácií.
- express (4.17.1) - je minimálny a flexibilný rámec webových aplikácií Node.js, ktorý poskytuje robustnú sadu funkcií na vývoj webových a mobilných aplikácií.
- parse (3.4.1) - Knižnica, ktorá poskytuje prístup k výkonnému backend serveru Parse z klientskej aplikácie.
- bootstrap (5.1.3) - populárny CSS vývojový rámec na vývoj responzívnych a mobilných webových stránok.
- laravel-mix (6.0.41) - Zjednodušená verzia Webpacku.

4 Preklad programu

Pre lokálny vývoj na danom projekte je potrebné nainštalovanie knižníc, ktoré sú špecifikované v **package.json** nachádzajúceho sa v koreňovom priečinku zdrojové kódu. Nasledujúce kroky popisujú postup prekladu riešenia pre lokálny vývoj.

1. Rozbalenie projektu do akéhokoľvek priečinku v počítači.
2. Premiestenie sa do koreňového priečinku **aframe-petri-net-sim**.
3. Inštalácia potrebných závislostí, špecifikovaných v súbore **package.json**, pomocou príkazu *npm install*.
4. Nasledovné spustenie webovej aplikácie na vývojovom serveri pomocou príkazu *npm start*.