TYPESCRIPT

Segédlet a Kliens oldali technológiák c. tárgyhoz

Szabó Gábor

2018

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

1117 Budapest, Magyar tudósok körútja 2.
Q. ép. B. szárny 207. • www.aut.bme.hu
Telefon: 463-2870 • Fax: 463-2871

Bevezetés

Célkitűzés

A labor során megismerkedünk a TypeScript nyelv használatával, a kapcsolódó típusrendszerrel, típusinformációkkal, a TypeScript-ben történő fejlesztés alapvető metodikájával.

Előfeltételek

A labor elvégzéséhez szükséges eszközök:

- **Visual Studio Code** (tesztelve a 1.10.2 verzióval)
  - A kellően vakmerőek próbálkozhatnak Visual Studio IDE-vel is, ebben az esetben a VS2017-es verzió használata javasolt az NPM integráció miatt.
  - A VS Code ZIP verziója a hivalatos oldalról is letölthető, ekkor nem szükséges adminisztrátori jog a telepítéshez (http://code.visualstudio.com/docs/?dv=winzip)
- NodeJS Package Manager (**NPM**)

Amit érdemes átnézned

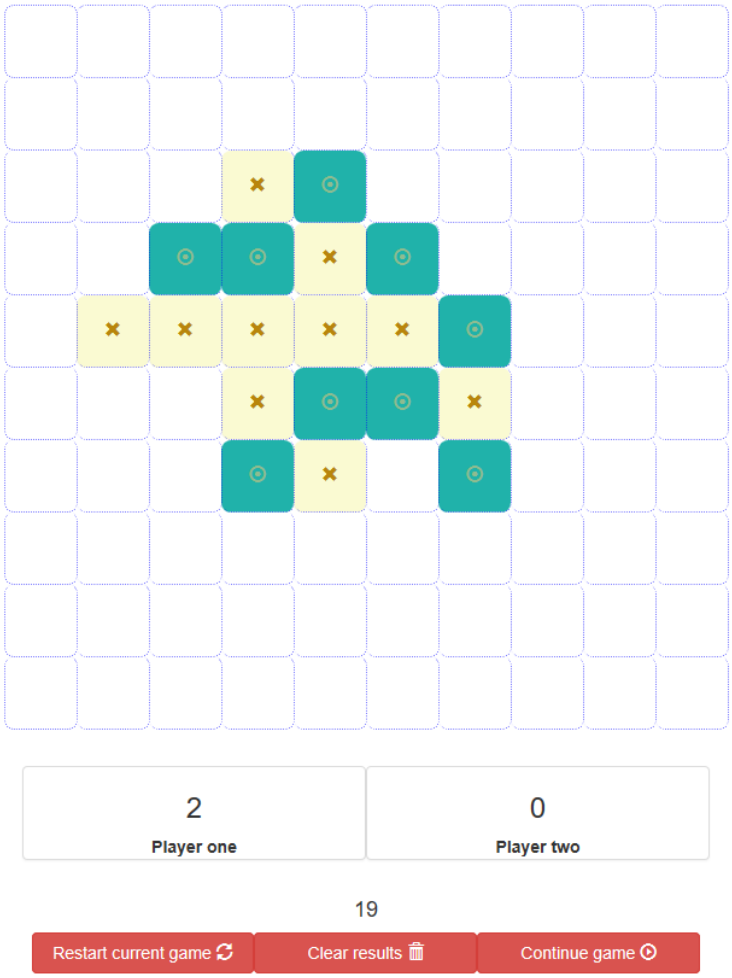
- HTML, JavaScript alapok
- TypeScript típusrendszer, statikus típusosság

Összefoglalás

Az előadásokat követően gyakorlatban is megismerkedünk a TypeScriptben történő fejlesztéssel, az ezt támogató típusrendszer használatával. A laboron egy amőba játékot készítünk.

Jegyzőkönyv

A feladatok megoldását jegyzőkönyvvel kell alátámasztani. A jegyzőkönyv sablont a tárgy honlapjáról lehet letölteni. Az önálló feladatok megoldásához szükséges az első 3 feladat teljesítése, ezért a labor során végig érdemes figyelemmel követni a laborvezető útmutatását.



Feladatok

0. feladat – áttekintés

Első lépésként tekintsük át a kiinduló projektet, végezzük el a szükséges műveleteket a függőségek betöltéséhez, és indítsuk el az alkalmazást!

1. Nyissuk meg a Visual Studio Code szerkesztőt!
   - A szerkesztőben az **F1** megnyomására egy beépített, mindenre alkalmas keresőt és feladat-futtatót kapunk. Az itt megjelenő „>” jelet törölve és a „?” karaktert beírva gyors segítséget kapunk az egyes funkciók elvégzésére. A „**>**” után parancsokat gépelhetünk. A színséma átállításához begépelhetjük a „**>**” után a „**theme**” kulcsszót, majd a **Preferences: Color Theme**-et választva válthatunk a beépített színsémák között.
2. Telepítsük az **IIS Express** bővítményt a Visual Studio Code szerkesztőbe!
   - A baloldali menüsávon válasszuk ki az Extensions fület!
   - A megjelenő keresőbe gépeljük az „IIS Express” kifejezést, telepítsük!
   - Indítsuk újra a Visual Studio Code alkalmazást!
3. A File → Open Folder… paranccsal válasszuk a kiinduló projekt mappáját! Az alábbiakat érdemes áttekintenünk:

- A .vscode mappában különböző konfigurációs fájlok vannak a szerkesztő testreszabásához.
- A styles mappában egy kiinduló amoeba.css fájl írja le az alkalmazás stílusát.
- A main.ts fájlban fogunk dolgozni, ez egyelőre üres.
- A package.json fájlban az alkalmazásunk metaadatai és függőségei találhatók. A függőségeket telepítenünk kell, ehhez indítsuk el az előre definiált npm restore parancsot (F1 → Tasks: Run Task → npm restore)! Ez telepíti a definiált függőségeket, úgy, mint a TypeScript fordítót. A függőségek a node_modules mappába kerülnek, ha nem jelenik meg és meg szeretnénk nézni, a projekt neve mellett megjelenő Refresh gombbal megtehetjük.
- A tsconfig.json fájlban a TypeScript fordító konfigurációja található. Ezen NE módosítsunk a labor során, kivéve, ha az kifejezetten jelezve van a segédletben!
- Az index.html fájl az alkalmazásunk vázát írja le. A fejlécben a szükséges függőségeket (JQuery, Bootstrap) hivatkozzuk be, ezután a fájl törzsében egy táblázat (a játékmező) található, alatta egy eszköztár, ahová gombokat fogunk elhelyezni a későbbiekben, és a két játékos egy-egy kártyája a nevükkel és az általuk megnyert játékok számával.

4. Miután letöltöttük a függőségeket, az **F1 → Tasks: Run Task → tsc watch** lehetőséget választva a TypeScript fordító elindul és a háttérben fut, és figyeli a fájlokban bekövetkező változásokat, amikor pedig újra fordítja azokat.
5. A **Ctrl+F5** megnyomására a **IIS Express**-ben elindul az alkalmazásunk. F5 megnyomására ezután a Chrome debuggerrel felcsatlakozik, így van lehetőség VS Code-ból a hibakeresésre (jelenleg ez valószínűleg nem fog működni a modulbetöltés miatt).

1. FELADAT – A JÁTÉK KERETALKALMAZÁSA

A keretalkalmazásban ügyeljünk, hogy ne függvény-orientáltan, hanem objektum-orientáltan gondolkodjunk! Hozzuk létre az ehhez szükséges kiinduló osztályokat! Fájlt, mappát létrehozni a mappa neve melletti ikonokkal tudunk.

Tipp: kódot formázni a Shift+Alt+F billentyűkombinációval tudunk. További billentyűkombinációkért az F1 → Preferences: Open Keyboard Shortcuts lehetőséget keressük fel!

A játékos, a játéktábla és egy négyzet (csempe) egyértelmű entitások, hozzuk hát ezeket létre!

A játékos egy egyszerű entitás, a nevét tároljuk el, illetve azt, hogy hányszor nyerte meg a játékot eddig.

player.ts:

```
export class Player {
    constructor(public name: string, public id: number, public gamesWon = 0) { }
}
```

A csempe szintén egyszerű, van egy állapota (üres, X vagy O van rajta), illetve az állapot változásakor frissíti a DOM-ot és megjelenít egy X-et vagy egy O-t a hozzá tartozó négyzetben.

tile.ts:

```
export enum TileState { Empty = 0, X = 1, O = 2 }

export class Tile {
    state: TileState = TileState.Empty;
    constructor(public element: JQuery) { }

    setState(state: TileState) {
        if (this.state === TileState.Empty && state !== TileState.Empty) {
            this.state = state;
            if (state === TileState.X) {
                this.element.addClass("mark mark-x");
                this.element.append(
                    $("<i class='glyphicon glyphicon-record'></i>"));
            } else if (state === TileState.O) {
                this.element.addClass("mark mark-o");
                this.element.append(
                    $("<i class='glyphicon glyphicon-remove'></i>"));
            }
        }
    }
}
```

A játékot a játék táblán keresztül kezeljük. A táblának van egy szélessége és egy magassága, ismeri a rajta található csempéket és játékosokat, illetve ezek közül a nyertest. Fontos továbbá, hogy ismeri magát a táblát a DOM-ban, így ezt egy JQuery objektumban adjuk neki át.

game-board.ts:

```
import * as $ from "jquery";
import { Tile, TileState } from "./tile";
import { Player } from './player';

export class GameBoard {
    readonly x = 10;
    readonly y = 10;

    board: Tile[][];
    playerOne = new Player('Player one' , 1);
    playerTwo = new Player('Player two' , 2);

    currentPlayer: Player;
    winner: Player;

    constructor(public tableElement: JQuery) {
    }
}
```

Felépítettük a kezdeti adatmodellünket, most készítsük el magát a logikát, ami összeállítja a táblát!

A tábla felépítését kezdjük a konstruktorban, és bontsuk le a különböző lépésekre! Az első lépés, hogy inicializálnunk kell a táblázatot, és minden cellához hozzárendelni a mögöttes modellt (Tile).

```
export class GameBoard {
    //...
    initializeBoard(tableElement: JQuery, board: Tile[][]) {
        tableElement.children().remove();
        let tBody = $("<tbody></tbody>");
        tableElement.append(tBody);
        console.log(tableElement.children());
        for (let i = 0; i < this.y; i++) {
            var rowTiles: Tile[] = [];
            let row = $("<tr></tr>");
            tBody.append(row);
            for (let j = 0; j < this.x; j++) {
                let column = $("<td></td>");
                row.append(column);
                rowTiles.push(new Tile(column));
            }
            this.board.push(rowTiles);
        }
    }
}
```

Előbb-utóbb szeretnénk elmenteni a játékállapotot. Ez legyen egy loadState() és saveState() páros, a loadState igazzal térjen vissza, ha sikeresen visszaállította az állapotot. Egyelőre csak mock implementációnk legyen:

```
loadState() {
    return false;
}
saveState() { }
```

Most kössük össze az egészet:

```
constructor(public tableElement: JQuery) {
    this.startGame();
}

startGame() {
    if (!this.loadState()) {
        this.initializeBoard(this.tableElement, this.board = []);
        this.currentPlayer =
             this.winner === this.playerOne ? this.playerTwo : this.playerOne;
    }
}
```

Végül pedig indítsuk el a játékot:

main.ts:

```
import { GameBoard } from './game-board';
import * as $ from 'jquery';

var board: GameBoard;
$(() => {
    board = new GameBoard($(".game-board-table"));
});
```

Ha ügyesek voltunk, akkor az alábbit értük el: az alkalmazás indulásakor az X-ekkel kitöltött táblázat helyett egy üres táblázat jelenik meg, emögött viszont már van egy konkrét adatmodell, amit kódból tudunk manipulálni.

Ha a böngésző nem frissül, akkor lehetséges, hogy a TypeScript compiler leállt, ekkor az F1 → Tasks: Run Task → tsc watch újbóli indításával lehet újraindítani.
Lehetséges az is, hogy a böngésző cache-elte a JavaScript fájlokat, ilyenkor az F12-t megnyomva, a böngésző Toolbar Network fülén ürítsük a cache-t és töltsük újra az oldalt.

---

2. FELADAT – BEVITEL- ÉS ÁLLAPOTKEZELÉS

Ahhoz, hogy tudjuk kezelni az állapotot, kommunikálnunk kell a felülettel. Ezt eseménykezelők beregisztrálásával tesszük meg.

A táblán kell tudnunk kezelni az egyes cellákra történt kattintást…

```
onTileClicked(tile: Tile) {
    if (tile.state === TileState.Empty && this.winner === undefined) {
        if (this.currentPlayer === this.playerOne) {
            tile.setState(TileState.X);
            this.currentPlayer = this.playerTwo;
        } else if (this.currentPlayer === this.playerTwo) {
            tile.setState(TileState.O);
            this.currentPlayer = this.playerOne;
        }
        this.checkWinner();
    }
}
checkWinner() {

}
```

… és fel kell tudnunk regisztrálni a klikk eseménykezelőket a tábla cellájára.

```
registerHandlers(board: Tile[][]) {
    for (let i = 0; i < board.length; i++) {
        for (let j = 0; j < board[i].length; j++) {
            let tile = board[i][j];
            tile.element.click(() => this.onTileClicked(tile));
        }
    }
}
```

A registerHandlers() függvényt meg kell hívnunk a startGame() végén:

```
startGame() {
    if (!this.loadState()) { /* ... */ }
    this.registerHandlers(this.board);
}
```

Kezeljük megfelelően a játék végét, amikor 5 elem egy sorban vagy egy oszlopban egymás után ugyanolyan.

```
checkWinner() {
    var points = 0;
    for (let fun of [
        (i: number, j: number) => this.board[i][j],
        (i: number, j: number) => this.board[j][i]
        ]) {
        for (let i = 0; i < this.x; i++) {
            let state = TileState.Empty;
            points = 1;
            for (let j = 0; j < this.y; j++) {
                let tile = fun(i, j);
                console.log(`${i}, ${j}, ${tile.state}, ${state}, ${points}`);
                if (tile.state !== TileState.Empty && tile.state == state) {
                    if (++points >= 5) {
                        this.won(tile.state === TileState.X ?
                                this.playerOne : this.playerTwo);
                    }
                } else {
                    points = 1;
                }
                state = tile.state;
            }
        }
    }
}

won(player: Player) {
    alert("Player " + player.id + " won! Congrats, " + player.name + "!");
    player.gamesWon++;
    var continueButton = $(".continue-game");
    continueButton.removeAttr("disabled").click(() => {
        continueButton.attr("disabled", "disabled");
        this.winner = undefined;
        this.startGame();
    });
    this.winner = player;
}
```

3. FELADAT – ADATOK PERZISZTÁLÁSA LOKÁLIS TÁROLÓBA

Adatok perzisztens tárolására használhatjuk a böngésző localStorage objektumát, ami egy egyszerű, sting-string kulcs-érték tároló. Minden lépés után mentsük el az aktuális állapotot, amit az oldalra történő visszanavigációkor kiolvasunk!

```
onTileClicked(tile: Tile) {
    if (tile.state === TileState.Empty && this.winner === undefined) {
        /* ... */
        this.saveState();
    }
}
```

Fontos, hogy mentéskor stringeket mentünk, így az objektumok valójában egyszerű anonim objektumok lesznek, ha a JSON API-t használva oda-vissza sorosítjuk őket. A SaveData asszertálásakor importáljuk az aktuális kontextusba a Ctrl+. használatával!

save-data.ts:

```
import { TileState } from "./tile";

export interface SaveData {
    playerOne: PlayerInfo;
    playerTwo: PlayerInfo;
    current: 'player-one' | 'player-two';
    x: number;
    y: number;
    tileStates: TileState[][];
}

export interface PlayerInfo {
    name: string;
    id: number;
    gamesWon: number;
}
```

game-board.ts:

```
saveState() {
    localStorage.setItem("amoeba-table", JSON.stringify(<SaveData>{
        playerOne: this.playerOne,
        playerTwo: this.playerTwo,
        x: this.x,
        y: this.y,
        tileStates: this.board.map(row => row.map(tile => tile.state))
    }));
}
```

A sorosításnál ügyeljünk arra, hogy milyen objektumokat használunk, valójában nem tudtuk volna magukat a Tile-okat eltárolni, ugyanis azokat visszasorosítva csak adatobjektumunk lesz, a hozzá tartozó függvények nem lesznek sorosítva!

```
import { SaveData } from "./save-data";

loadState() {
    let data = <SaveData>JSON.parse(localStorage.getItem("amoeba-table"));
    if (!data)
        return false;
    if (this.x !== data.x || this.y !== data.y) {
        localStorage.removeItem("amoeba-table");
        return false;
    }

    this.initializeBoard(this.tableElement, this.board = []);
    for (let i = 0; i < data.x; i++) {
        for (let j = 0; j < data.y; j++) {
            this.board[i][j].setState(data.tileStates[i][j]);
        }
    }
    this.playerOne = data.playerOne;
    this.playerTwo = data.playerTwo;
    this.currentPlayer =
        (data.current === 'player-one') ? this.playerOne : this.playerTwo;
    return true;
}
```

Próbáljuk ki az alkalmazást!

Láthatjuk, hogy amikor újratöltjük az oldalt, az alkalmazás állapota visszatöltődik.

4. FELADAT – ÖNÁLLÓ FELADAT

A tanult megközelítések használatával valósítsd meg az alábbiakat:

1. A játék újraindítását és eddigi eredmények törlését jelző gombok kattintás eseménykezelőjét kezeld megfelelően! Újraindításra a jelenlegi játéktér ürítődik, az eredmények törlésével ezen felül a játékosok pontszáma is nullázódik.
2. Kezeld a duplakattintás eseményt a játékos nevére kattintva: a játékos nevének helyén ilyenkor egy input jelenjen meg, amiben a játékos a nevét tudja szerkeszteni, valamint egy OK gomb, amire kattintva az eredeti állapotba áll vissza a felület, elmentve a játékos nevét!
3. Vezesd ki a felületre, és tartsd karban a játékosok nevét, és a játékosok által elért pontszámokat az erre fenntartott elemekben, illetve az aktuális játék során megtett lépések számát! Ezeket az újratöltés során is vedd figyelembe!