Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Kovács-Bodó Csenge

2024

Szegedi Tudományegyetem

Informatikai Intézet

Beszédfelismerő és képernyő felolvasó játék fejlesztése vakok és látássérültek számára

Szakdolgozat

Készítette:
Kovács-Bodó Csenge
Gazdaságinformatikus
BSc hallgató

Témavezető:
Dr. Jánki Zoltán Richárd
Egyetemi adjunktus

Szeged
2024

Feladatkiírás

Az esélyegyenlőség jegyében számos program érhető el az interneten, amelyeket kifejezetten vakok és látássérültek számára fejlesztettek. Emellett nagy népszerűségnek örvend az általános felhasználók körében is, ha egy program felhasználói felülete minél könnyebben használható és felhasználóbarát. Szakdolgozatom célja olyan frontend oldali technológiák, módszerek tárgyalása, melyekkel akadálymentesíthetünk egy programot látásukban korlátozott felhasználók számára. Példaprogramom egy egyszerű memóriajátékot mutat be, melyet hangvezérléssel irányíthatunk, és a program folyamatos szövegfelolvasással visszajelzést küld, és segít a tájékozódásban.

Tartalmi összefoglaló

A téma megnevezése:
Beszédfelismerő és képernyő felolvasó játék fejlesztése vakok és látássérültek számára

A megadott feladat megfogalmazása:
Szakdolgozatom célja olyan módszerek és technológiák bemutatása, amelyekkel vakok és látássérültek számára kényelmesen használható, kiváló felhasználói élményt nyújtó játékot készíthetek.

A megoldási mód:
Az alkalmazás célja, hogy a felhasználók billentyűzet és egér nélkül, akusztikus módszerekkel irányíthassák a játékot, mivel nem támaszkodhatnak a látásukra. Ehhez a Web Speech API beszédfelismerő és beszédszintetizáló funkcióit használtam.

Alkalmazott eszközök, módszerek:
A demonstrációs projekt a legújabb Angular keretrendszerrel készült. Létrehoztam egy Firebase projektet, amelyben Firestore-t használtam a játék eredményeinek, Storage-t pedig a kártyák képeinek tárolásához, és Hostingot a program kitelepítéséhez. A beszédfelismerést és szintetizációt a Web Speech API két részAPI-jával valósítottam meg.

Elért eredmények:
Létrehoztam egy webes memóriajátékot, amelyhez nem szükséges billentyűzet vagy egér, mert hangparancsokkal irányítható, és beszédszintetizációval ad visszajelzést. Az egér mozgatásával az alkalmazás felolvassa a szövegeket és az elemek nevét. A játék magyar vagy angol nyelven is játszható, testreszabható beállításokkal. Van egyszemélyes és kétszemélyes mód, az egyéni eredmények pedig ranglistára küldhetők.

Kulcsszavak:
beszédfelismerés, beszédszintetizáció, szövegfelolvasás, Web Speech API

Tartalomjegyzék

Bevezetés

Vakok és látássérült emberek számára elérhető eszközök, játékok, könyvek vagy alkalmazások tervezése és készítése napjaink legnagyobb kihívása, viszont az egyik legszebb feladata. Fontos missziója a társadalmunknak ezen emberek integrálása, befogadása közösségünkbe. A testileg vagy szellemileg sérült emberek gyakran szembesülhetnek azzal, hogy a piacon található alkalmazások nem megfelelőek számukra, emiatt kirekesztettnek érezhetik magukat, önbecsülésük csökken, szociális készségeik visszamaradottá válnak. Középiskolában volt egy osztálytársam, aki egy ritka szembetegséggel született, ennek eredményéül maximum tíz százalékot látott a külvilágból. Nehezen nyitott mások felé, nehezen tudott tanulni, lassan írt kézzel, gépelni egyáltalán nem tudott. A technológia rohamszerű fejlődésének köszönhetően már számos olyan programkönyvtár elérhető, amit a legtöbb magasszintű programozási nyelv és keretrendszer támogat. Az elmúlt évtizedben nagy hangsúlyt fektettek a fejlesztő cégek olyan szoftverek értékesítésébe, melyekben különféle akadálymentesítést szolgáló funkciók elérhetőek, ezzel a felhasználói élményt maximalizálták.

Szakdolgozatom célja azon technológiák, frontend fejlesztési alapelvek, módszerek ismertetése, összegyűjtése, melyekkel vakoknak, látássérülteknek és színtévesztőknek is egyaránt tudunk szoftvereket tervezni és fejleszteni. Példaprogramom egy olyan memóriajátékot mutat be, melyben nem szükséges a látásunkra hagyatkozni, illetve billentyűzetre és egérre sem lesz feltétlen szükség. Beszéddel tudjuk kezelni az alkalmazást, mégpedig úgy, hogy az a felhasználó szóban kiadott parancsokkal tudja irányítani az egész programot. Így a játék beszédfelismerésre képes, emellett szövegfelolvasással és beszédszintetizációval segít az oldalon való könnyebb tájékozódásban.

1. fejezet

Terület áttekintése, jelenleg ismert technológiák

A piacon egyre több új szoftver és alkalmazás jelenik meg, amelyek kifejezetten vakok, látássérültek vagy színtévesztők számára kínál új funkciókat. Az alábbiakban bemutatom a legismertebbeket közülük.

1.1. Apple VoiceOver

A VoiceOver [1] egy fejlett képernyő felolvasó funkció Mac Os X operációs rendszerekben, mely lehetővé teszi látássérült felhasználók számára, hogy könnyedén vezérelhessék telefonjukat, táblagépüket vagy számítógépüket billentyűzetparancsokkal [3] és gesztusokkal. [2] A képernyő tartalmában szakaszosan tud lépegetni a felhasználó, miközben a készülék automatikusan felolvassa annak tartalmát. Mac gépeken a Command+F5 billentyűkombinációval kapcsolható be, Iphone-okon és Ipad-eken a beállításokon belül a kisegítő lehetőségek menüpontban található, vagy akár Siri-nek kiadott utasítással is bekapcsolható. A Braille kijelzők [4] és a Multi-Touch trackpad [5] megjelenése óta már egyszerű kézi gesztusokkal is vezérelhető a felolvasás. [2]

1.2. Intelligens személyi asszisztens

A VoiceOver-nél említett Siri [6], Samsung eszközökön a Bixby [7], Amazon Alexa és Alice mind-mind olyan intelligens személyi asszisztens program, mely nagy mértékben egyszerűsíti és gyorsítja eszközeink használatát. Elég egyetlen, szóban kiadott utasítás és a program a felhasználó helyett elvégzi a feladatot, például hívást indít, rákeres bármire a böngészőben vagy megnyitja a letöltött applikációkat. Ez a funkció rendkívül hasznos

látássérülteknek, mivel egyáltalán nem kell a kezüket használni okostelefonjuk kezeléséhez.

1.3. SciFY

A SciFY (Science For You) [8] egy görög nemzetközi szervezet, mely ingyenes és nyílt forráskódú LEAP (Listen - LEArn - Play) játékokat fejleszt kifejezetten vakok és gyengénlátó gyermekek számára. A játékok célja, hogy segítsenek a gyerekeknek új készségek elsajátításában, miközben szórakoztatják őket. Hivatalos weboldalukon, a `https://gamesfortheblind.org` oldalon különböző játékok találhatóak, mint például a Tic Tac Toe, Tennis és Curve, amelyek különböző nehézségi szinteken játszhatók. Ezek a játékok háromdimenziós binaurális hangot [10] használnak, amelyet a játékosok fejhallgatóval hallhatnak, így képesek pontosan érzékelni a hangok helyét a térben. Ennek segítségével pontosabban észlelhetik a hangforrások irányát és távolságát. A weboldalon kívül a Memor-i Online platformon bármelyik felhasználó könnyedén létrehozhat saját, inkluzív memóriajátékot, amelyet mind vak, mind látó játékosok is játszhatnak. A platform célja, hogy lehetővé tegye a vak és látó gyermekeknek a közös játékot, és segítse a gyerekek fejlődését

1.4. Chrome-ban támogatott bővítmények és csomagok

Fellelhető egy kiváló leírás a Google Codelabs jóvoltából a `https://codelabs.developers.google.com/angular-a11y#0` oldalon, ahol pontokba szedve adnak tippeket, hogyan lehet egy Angular-ban írt felhasználói felület minél jobban akadálymentesített, mire figyeljen egy fejlesztő, mikor színtévesztőknek tervez alkalmazást. Egy egyszerű képernyőolvasó módszert is bemutat, melynek lényege, hogy a HTML elemeinket felcímkézzünk ARIA label-ökkel, ahol megadjuk, hogy a program mit olvasson fel, majd a Chrome Web Store-ból letöltjük a megfelelő bővítményt [12], amely felolvassa a label-ök tartalmát. Az 1.1-es kód egy példa arra, hogyan lehet egy olyan Angular Material[11] csúszka elemet létrehozni, amelyet felcímkéztek egy *aria-label*-lel, így egy képernyőfelolvasó bővítmény képes felolvasni az elem értékét.

```
<mat-slider
  aria-label="Dumpling order quantity slider"
  id="quantity" name="quantity"
  color="primary" class="quantity-slider"
  [max]="13" [min]="1" [step]="1"
  [tickInterval]="1" thumbLabel
  [(ngModel)]="quantity">
</mat-slider>
```

1.1. kód. ARIA label-lel felcímkézett Angular Material slider.

1.5. Referencia alkalmazások

Példaprogramom fejlesztése előtt áttekintettem valamennyi nyílt forráskódú programot, publikus GitHub repository-t [13][14][15], melyek beszédfelismeréssel és beszédszintetizációval működnek. A kiválasztott technológia, amivel elkészítettem a játékot, a Web Speech API, mely a következő fejezetben kerül bemutatásra.

2. fejezet

A Web Speech API

2.1. A gépi beszédfelismerés háttere

A számítógépes beszédfelismerés [17] egy összetett folyamat, amely az emberi beszéd digitális feldolgozásán alapul azzal a céllal, hogy azt a számítógép írott formába alakítsa. Az alábbiakban a legfontosabb elemeket és alapelveket foglalom össze.

2.1.1. Alapfogalmak: Az osztályozási feladat

A beszédfelismerés olyan osztályozási probléma, ahol a cél az, hogy adott hangokról eldöntsük, melyik fonémához (beszédhanghoz) tartoznak. Ehhez a hangokat a számítógépes feldolgozás során, jellemzőkkel (adatokkal) írjuk le. Az osztályozás Bayes-döntéselmélet alapján történik, amely azt mondja ki, hogy az a döntés optimális, amelyik a legvalószínűbb kimenetet választja az adatok alapján.

2.1.2. A statisztikai módszerek kihívásai

A statisztikai alapú megközelítés során gyakran két fő probléma merül fel:

- **Adatkorlátok**: Kevés példa esetén a statisztikák pontatlanok lehetnek.
- **Dimenziós problémák**: Sok jellemző és lehetséges kombináció mellett az adatok feldolgozása exponenciálisan nehezebbé válik.

Lehetséges megoldások:

- Több adat gyűjtése.
- A jellemzők számának csökkentése, azaz kizárólag a releváns adatok használata.
- A valószínűségek egyszerűsítése (például az egyes jellemzők függetlenségét feltételezve).

2.1.3. A beszédfelismerő rendszer felépítése

A tipikus beszédfelismerő rendszer több modulból áll:

- **Előfeldolgozás**: A beszédet digitális jellé alakítják, amelyet spektrogram formájában elemeznek. Ezt követően csökkentik az adatmennyiséget, például a kevésbé fontos részek eltávolításával.
- **Akusztikai-fonetikai modell**: Ez a rész a hang és a beszédhangok (fonémák) közötti kapcsolatot térképezi fel. Ehhez gyakran rejtett Markov modelleket (HMM-eket) használnak, amelyek a beszédet kisebb egységekre bontva értékelik.
- **Nyelvi modell**: Ez a modul a lehetséges beszédkombinációk valószínűségét határozza meg. Például az angol nyelvben a szavak előfordulási valószínűségét, míg magyarban a ragozási szabályokat figyelembe véve működik.
- **Dekóder**: Ez az egység összevonja az előző modellek eredményeit, hogy az egész beszédet szöveggé alakítsa. A folyamat során számos hipotézist vizsgál meg, és a legvalószínűbbet választja ki.

Tehát megállapítható, hogy a számítógépes beszédfelismerés lényege, hogy az emberi beszédet digitális jelekké alakítja, és statisztikai, valamint gépi tanulási módszerek segítségével fonémákra, majd írott szövegre fordítja. A módszerek fejlesztésének kulcsa az adatok pontos feldolgozása és a modellek folyamatos finomítása.

2.2. A Web Speech API

A Web Speech API [18] egy böngésző alapú API, amely lehetővé teszi a beszéd alapú alkalmazások létrehozását webfejlesztők számára. Ez az API különösen hasznos a látássérült felhasználók számára, vagy azoknak az alkalmazásoknak, amelyek interaktívabb élményt szeretnének nyújtani. Két fő eleme:

– **SpeechRecognition**: lehetővé teszi a weboldalak számára, hogy felismerjék és feldolgozzák a felhasználó beszédét. Ez hasznos lehet például hangalapú keresésekhez vagy hangvezérelt feladatokhoz.

– **SpeechSynthesis**: lehetővé teszi, hogy a program beszéljen egy adott nyelven. Például, ha egy weboldal szeretne szöveget felolvasni, a SpeechSynthesis segítségével ezt megteheti.

Használata kizárólag JavaScript-ben biztosított, ezen két funkció a *SpeechSynthesisUtterance* és a *SpeechRecognition* objektumok használatával érhető el.[19]

2.2.1. SpeechRecognition

A SpeechRecognition API[20] a Web Speech API része, amely segítségével webalkalmazások képesek felismerni és feldolgozni a felhasználók beszédét. Az API lehetővé teszi a beszédfelismerési folyamat elindítását, amely során a mikrofonon keresztül érkező hangokat elemzi és szöveggé alakítja. Számos eseményt kínál, mint például *onresult*, *onspeechstart*, *onspeechend*, amelyekkel a fejlesztők különböző eseményeket kezelhetnek a felismerési folyamat során. Támogatja a folyamatos felismerést is, ahol a beszédfelismerés nem áll le az első mondat után, hanem folytatódik, amíg a felhasználó beszél. Támogatottsága böngészőnként eltérő lehet, jelenleg Safari-ban még nem támogatott. Emellett biztonsági kockázatokkal is járhat a mikrofon engedélyezése. Bármely program inicializálásakor a felhasználó dolga a mikrofonjának engedélyezése a böngészőnek, ennek hiányában nem fog működni a beszédfelismerés.
Az 2.1-es kódban inicializálom a beszédfelismerést. Létrehozok egy új *recognition* változót, beállítom a nyelvét, valamint hogy folytonos legyen és csak a végső eredményekkel térjen vissza. Az *event.result* tömb tartalmazza majd a felismert szöveget. Végül elindítom a beszédfelismerést a *recognition.start()* metódussal.

```
var recognition = new (window.SpeechRecognition ||
window.webkitSpeechRecognition)();
recognition.lang = 'hu-HU'; recognition.continuous = true;
recognition.interimResults = false;
recognition.onresult = function(event) {
    var transcript = event.results[0][0].transcript;
    console.log(transcript);
};
recognition.start();
```

2.1. kód. Beszédfelismerés inicializálása.

2.2.2. SpeechSynthesis

A SpeechSynthesis [21] API a Web Speech API része, amely lehetővé teszi, hogy egy weboldal beszédszintetizációt valósítson meg, ezzel akár saját képernyőolvasó funkciót is implementálhatunk. Az API szöveget alakít hanggá, és lejátssza a felhasználóknak. Személyre szabhatjuk a szintetizált beszéd olyan tulajdonságait, mint a nyelvét, a beszéd sebességét, annak hangmagasságát és hangerejét. A *SpeechSynthesisUtterance* objektum tartalmazza a szintetizált beszéd szövegét és beállításait, és a *SpeechSynthesis* objektum vezérli a beszédszintetizálási szolgáltatást, például elindítja vagy leállítja a beszédet. Legfontosabb függvénye a *speak()*, mely két paramétert vár: a kimondandó szöveg és a nyelv. A 2.2-es kód bemutatja, hogyan történik a beszédszintetizáló inicializálása. Létrehoztam egy *SpeechSynthesisUtterance* típusú változót, majd beállítottam az attribútumait.

```
var msg = new SpeechSynthesisUtterance();
msg.text = "Heló Világ!";
//msg.lang = "hu-HU" // Nyelv
msg.volume = 1; // Hangerő
msg.rate = 1; // Lejátszási sebesség
msg.pitch = 1; // Hangmagasság
window.speechSynthesis.speak(msg, "hu-HU");
```

2.2. kód. Beszédszintetizáció inicializálása.

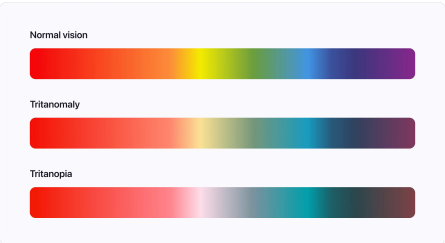
3. fejezet

A felhasználói felület optimalizálása színtévesztők és színvakok számára

3.1. Színtévesztések fajtái

A színtévesztés és színvakság [22] öröklődő látászavarok, amelyekben a retina színérzékelő csapjai gyengén működnek, hiányoznak, vagy egyes színek érzékelése csökken. Ha teljesen kiesik egy szín érzékelése, színvakságról beszélünk. A retina csapjai a színeket (vörös, zöld, kék) érzékelik, míg a pálcikák a szürkületi látásért felelősek. A tökéletes színlátás a csapok megfelelő működésétől és arányától függ. Színtévesztés, illetve színvakság esetén az egyén nehezen, vagy egyáltalán nem képes színek között különbséget tenni. Színtévesztés esetén az egyén nem látja a színeket azonos módon, mint a színlátók, míg színvakság esetén a színérzékelés teljesen kiesik egy adott színre vagy színcsoportra. Három féle színtévesztés ismert:

– **kék-sárga színtévesztés**: az alany összekeveri a kék és sárga árnyalatokat, legsúlyosabb esetben kizárólag sárga és kék árnyalatokat lát.A 3.1-es ábra gradiensábrázolásokkal érzékelteti a kék-sárga színtévesztés különböző fokozatait[24].

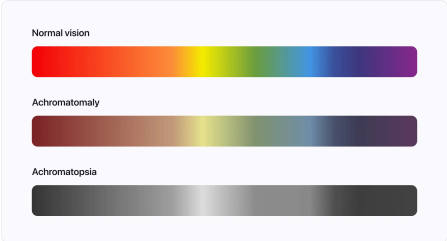


3.1. ábra. Kék-sárga színtévesztés fokozatai.

– **piros-zöld színtévesztés**: az alany összekeveri a piros és zöld árnyalatokat, legsúlyosabb esetben kizárólag sárga és kék árnyalatokat lát. A 3.2-es ábra szemléletesen mutatja be a piros-zöld színtévesztés fokozatait különböző gradiensábrázolások segítségével[23].

3.2. ábra. Piros-zöld színtévesztés fokozatai.

– **teljes színvakság**: az alany a színeket halovány árnyalatában, vagy kizárólag fekete, fehér és szürke árnyalatokat lát. A 3.3-as ábra gradiensek segítségével ábrázolja a teljes színvakság különböző szintjeit[25].



3.3. ábra. Teljes színvakság.

3.2. A felhasználói felület optimalizálási lehetőségei

Számottevő lehetőség áll a rendelkezésünkre, hogy egy olyan felhasználói felületet tudjunk készíteni, melyet színtévesztők és színvakok kényelmesen, akadálymentesen tudnak használni. Ezek közül összegyűjtöttem a legfontosabb szempontokat.

3.2.1. Egyes színkombinációk kerülése

Mivel a színtévesztők hajlamosak összekeverni egyes színeket, ezért érdemes néhány színkombinációt[26] elkerülni az oldalon, melyek megtéveszthetik a felhasználót. A 3.4-es ábra bemutatja azokat a színkombinációkat, melyeket tanácsos elkerülni:

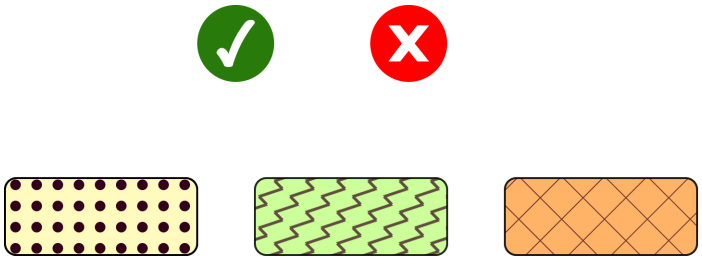
3.4. ábra. Kerülendő színpárok.

3.2.2. Kontrasztos színsémák

Magas kontrasztú színkombinációk[26] választása segíti a színvaksággal élőket az elemek megkülönböztetésében, és biztosítja, hogy minden információ jól látható és érthető legyen. A legjobb kombináció a fekete-fehér párosítás, de a lényeg, hogy sötét színhez a megfelelő párosítás egy minél világosabb szín legyen. Az AudioEye oldalán (`https://www.audioeye.com/color-contrast-checker`) különböző színkombinációkat tudunk tesztelni, hogy elég kontrasztosak-e.

3.2.3. Ikonok, minták használata

Az ikonok színei önmagukban nem feltétlen informatívak, különösen színtévesztők esetében. Például a Piros-zöld színtévesztők könnyen összekeverhetik a piros és a zöld gombokat. A könnyebb átláthatóság érdekében érdemes a színezés mellett az elemeket mintázni [26][27], szöveggel ellátni. A 3.5-ös ábrán vázoltam pár mintázott elemet. A zöld és piros gombok a színük mellett pipa és X ikonnal még kifejezőbbek, míg a mintázott gombok segítenek könnyen megkülönböztetni őket a színtévesztők számára.



3.5. ábra. Elemek mintázása.

3.2.4. Letisztult megjelenítés

Törekedjünk az egyszerűségre. Minél letisztultabb a felhasználói felület, annál könnyebb benne tájékozódni, egy minimalista design professzionálisabb és esztétikusabb megjelenést is kölcsönözhet az oldalnak. Az elemek legyenek megfelelő távolságra egymástól, ne csússzanak össze. Ragaszkodjunk egyféle színpalettához, azon belül is a legjobb, ha maximum kettő-három színt használunk fel az oldal színvilágának tervezése során.

4. fejezet

A program megvalósítása

Ebben a fejezetben bemutatom a szakdolgozat példaprogramjának megvalósítását, tervezéstől a kivitelezésig.

4.1. A program architektúrája

Szakdolgozati szoftverként egy Angular CLI alkalmazást készítettem el. A jelenleg legfrissebb, 17.3.0-s verziójú Angular-ral fejlesztettem, 20.12.1-es Node.js-sel és 10.5.0-s verziójú Node Package Manager-rel. A program teljes logikáját a komponensek TypeScript file-jaiban írtam meg, néhány service-szel kiegészítve, nem készítettem hozzá külön backend programot. Az alkalmazás több komponensből épül fel, melyek a következők:

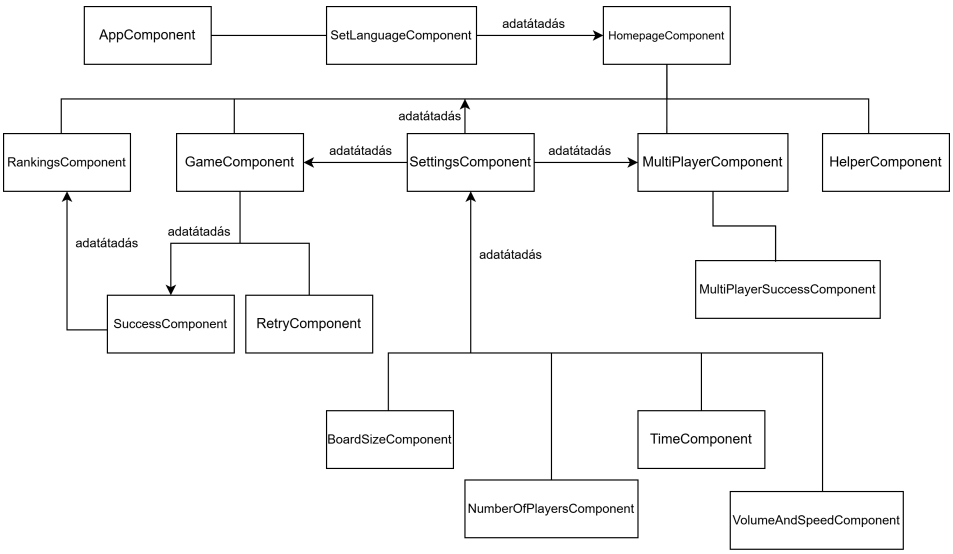
- **AppComponent**: a program magja, ebből indul ki a többi komponens (kivéve a *SetLanguageComponent*) és itt adjuk át a service-eknek az alkalmazás nyelvét.
- **HomepageComponent**: a főoldal komponense.
- **HelperComponent**: egy segítség dialógus, amely felolvassa az összes olyan utasítást, amelyeket a felhasználó ki tud adni, hogy irányíthassa a programot.
- **GameComponent**: az egyszemélyes játék komponense. Egy egyszerű memóriajátékkal tudunk játszani, mely méri az időnket és gyűjti a pontokat.
- **SuccessComponent**: egy dialógus ablak, amely akkor jelenik meg, ha megtaláltuk az idő letelte előtt az összes párt. Itt megadhatjuk a nevet amivel szeretnék szerepelni a ranglistán, és elküldjük az eredményeket a Firestore-ba.
- **RetryComponent**: egy dialógus ablak, amely akkor jelenik meg, ha nem találtuk meg az összes párt. Innen új játékot lehet kezdeni.

- **RankingsComponent**: a korábbi eredményeket listázza ki a Firestore-ból rangsorolva pont és idő alapján.
- **SetLanguageComponent**: a weboldal megnyitásakor mindig ez a komponens jelenik meg. Itt állítjuk be az oldal nyelvét.
- **SettingsComponent**: a beállítások menü komponense, mely az alábbi almenüket tartalmazza:
  - **BoardSizeComponent**: játéktér méretének beállítása. Háromszor négyes, négyszer négyes és négyszer ötös méretek elérhetőek.
  - **NumberOfPlayersComponent**: játékosok számának beállítása. Egyszemélyes és kétszemélyes játék közül lehet választani.
  - **TimeComponent**: játékidő beállítása. 60, 120, 180 vagy 300 másodperc állítható be.
  - **VolumeAndSpeedComponent**: itt a hangerő és a lejátszási sebesség állítható be.
- **MultiPlayerComponent**: kétszemélyes játék komponense. Nem méri az időt és a pontokat, kizárólag a két játákos talált párjainak a számát.
- **MultiPlayerSuccessComponent**: ez a dialógus akkor jelenik meg, ha a kétszemélyes játék során megtalálták az összes párt. Kiírja, illetve felmondja a nyertes nevét, vagy azt, ha döntetlen lett az állás.

A komponensek közötti működésért az alábbi service-ek felelősek:

- **CardService**: kártyák betöltése Storage-ból.
- **GameService**: segít a játék és a beállítás menü között a játék tulajdonságainak kezelésében.
- **LanguageService**: kommunikál a *SetLanguageComponent* és a többi komponens között, beállítja a nyelvet.
- **SpeechRecognizerService**: a beszédfelismerésért felelős service.
- **SpeechSynthesizerService**: a beszédszintézisért felelős service.
- **VoiceoverService**: a képernyőolvasást megvalósító service.

Készítettem egy egyszerűsített diagramot a komponensek közötti kapcsolatokról, hierarchiáról, ez látható a 4.1-es ábrán. Az egész alkalmazás az *AppComponent*-be van ágyazva. Bármely endopint megnyitásakor automatikusan a *SetLanguageComponent* nyílik meg először, mivel kötelesek vagyunk elsősorban a nyelvet kiválasztani. A *LanguageService*-szel átadjuk az alkalmazásnak a nyelvet, ezután megnyílik a *HomepageComponent*, innen navigálhatunk a *HelperComponent*-be, a *SettingsComponent*-be, a *RankingsComponent*-be, a *GameComponent*-be vagy kétszemélyes játék esetében a *MultiPlayerComponent*-be. Sikeres játék esetén megjelenik a *SuccessComponent* dialógusa, majd a játék eredményeit továbbítjuk a *RankingsComponent*-nek. Sikertelenség esetén a *RetryComponent* jelenik meg. A *MultiPlayerComponent*-ben, azaz a kétszemélyes játék végén a *MultiPlayerSuccessComponent* dialógusa jelenik meg. Mindkét féle játékmódban a *CardService*-szel töltjük be a képek elérési útvonalát. A Beállítások menüben, azaz a *SettingsComponent*-ben négy almenü található, melyek komponensei: *BoardSizeComponent, NumberOfPlayersComponent, TimeComponent, VolumeAndSpeedComponent*. A *GameService* segítségével alkalmazza a játék az almenükben beállított opciókat. Az egész weboldalon minden komponensben működik a *SpeechRecognizerService*, a *SpeechSynthesizerService* és a *VoiceoverService*.



4.1. ábra. Komponensek közötti kapcsolat, hierarchia.

Mivel a böngésző az operációs rendszerben beállított nyelvet, illetve letöltött nyelvi csomagjait tudja használni a Web Speech API használatához, így nagyon fontos, hogy a használni kívánt nyelvekhez tartozó csomagokat töltsük le az operációs rendszer beállításaiban. Alapértelmezetten támogatott minden operációs rendszerben az angol nyelv, viszont a magyar nyelv nem, ezért különösen fontos a felhasználóknak erről gondoskodniuk.

4.2. Adatmodellek

Három fő adattal dolgoztam az alkalmazás fejlesztése során. A program elején beállítottam a nyelvet, melyet egy string-ként átadtam a beszédfelismerésért és a beszédszintéziésért felelős service-eknek. Amennyiben magyar nyelvet választ ki a játékos, a "hu-HU", angol nyelv esetén a "en-US" string kerül átadásra. A játékban használt képeket nem a program repository-jában tároltam el, hanem az alkalmazás mögött álló Firebase Storageában. A képeket a Storage-ból az elérési útvonaljukkal és egyedi *Access token*-jük segítségével értem el. A képek betöltéséhez készítettem egy service-t *CardService* néven.
Az alábbi metódus(4.1-es kód) a játék során betölti annak a kártyának a képét, melyet kiválasztott a játékos. Paraméterben megkapja a kép elérési útvonalát (esetünkben a cards/ mappából), a *this.storage.ref(imageUrl)* segítségével pedig létrehoz egy hivatkozást a Firebase Storage-ban található fájlhoz, majd a *getDownloadURL()* metódust hívja meg a hivatkozásra, melynek visszatérési értéke tartalmazza a képeket.

```
getImage(imageUrl: string){
  return this.storage.ref(imageUrl).getDownloadURL()
}
```

4.1. kód. Kártyák képeinek betöltése Strogae-ból.

A kártyák elérési útvonalát egy konstansban tároltam el, mely látható a 4.2-es kódban.

```
export const cardpictures  =[
  "cards/cat.png","cards/ball.png","cards/candy.png",
  "cards/car.png","cards/cloud.png","cards/dog.png",
  "cards/flow.png","cards/rose.png","cards/sun.png",
  "cards/umb.png"
]
```

4.2. kód. Kártyák képeinek elérési útvonala a Storage-ban.

Tehát egy képet az elérési útvonala, valamint az Access token-je azonosít. A kártyák adatmodelljét a 4.1-es táblázat vázolja.

Attribútum	Típus	Leírás
elérési útvonal	string	A kép elérési útvonala a Storage-ból.
Access token	string	A kép egyedi token-je a Storage-ból.

4.1. táblázat. Kártyák attribútumainak táblázata.

Végül a játék során eltároljuk az eredményeket a ranglistába. Az adatokat a Firebase app Firestore-jába mentjük el, a rankings kollekcióba. A rangsori helyezések adatmodelljét a 4.2-es táblázat ismerteti.

Attribútum	Típus	Leírás
documentId	string	Egyedi azonosítója a Firestore dokumentumnak, amely egy rangsori helyezés adatait tárolja.
name	string	Felhasználónév (felhasználói inputról).
points	number	A játék során elért pont.
time	number	A játékban eltöltött másodpercek száma.
date	timestamp	A játék adatainak mentési ideje (mm dd, yy at hh:mm:ss AM/PM timezone formátumban).

4.2. táblázat. Ranglista helyezés attribútumainak táblázata.

A 4.3-as kódban látható az a TypeScript interfész, amelyet az adatok kezeléséhez készítettem.

```
export interface Ranking{
 name:string;
 points: number;
 time: number;
 date: any;
}
```

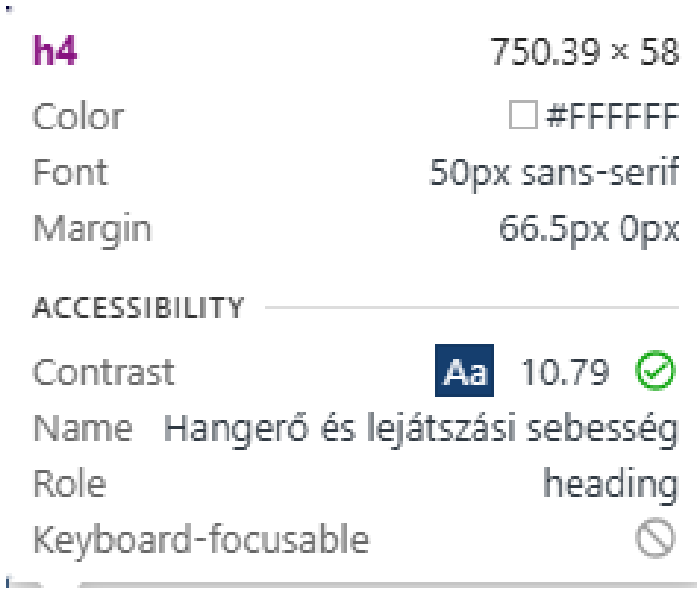
4.3. kód. Ranking interfész.

4.3. Verziókövetés

A fejlesztési folyamat során a programot a kezdetektől a befejezésig verziókövetéssel kezeltem a GitHub platformon, biztosítva a kód folyamatos nyomonkövethetőségét és dokumentáltságát. A végleges alkalmazást a Firebase Hosting szolgáltatás segítségével telepítettem ki, amely stabil és megbízható környezetet biztosított a projekt publikálásához. A `https://blindmemorygame-szakdolgozat.web.app/` címen publikusan elérhető bármely böngészőben az alkalmazás.

4.4. A játék grafikus interfészének elkészítése, jellemzői

A szoftver felhasználói felületének elkészítése során különös figyelmet szenteltem arra, hogy színtévesztők és színvakok számára is megfelelő legyen. Figyelmet fordítottam a színekre, nem használtam olyan kombinációkat, melyeket színtévesztők könnyen összekeverhetnek, mint például a piros és zöld együttese. Egy sötétkék hátteret választottam, és a szövegeket fehérre színeztem, hogy minél kontrasztosabb legyen. Átlagosan 60 pixel nagyságú betűméretet és sans-serif betűtípust adtam meg a css file-okban, amely egy talpatlan betűtípus, ezáltal könnyebben olvasható. Minden gombot, toggle csúszkát, görgetős elemet és ikont a lehető legnagyobb méretűre állítottam. Az elért eredményem pedig, hogy a Developer Tools minden elememre pipát írt a Contrast értékre, amely azt mutatja, hogy könnyen olvasható, akadálymentesített a felület. A 4.2-es ábrán a beállítások menüben teszteltem egy almenü címsorát Developer Tools-ban.



4.2. ábra. Vizsgálat Developer Tools-ban egy tetszőleges elemre.

A játékba beépítettem egy apró funkciót, amely lehetővé teszi, hogy amikor a játékos az egeret egy kártya fölé viszi, annak mérete háromszorosára növekedjen. Ez gyakorlatilag egy nagyító funkcióként működik. Ezen a képernyőfelvételen (4.3-as ábra) látható a példa erre:

4.3. ábra. Kártyák nagyítása hover-nél.

4.5. Angol és magyar nyelvre fordítás implementálása

Annak érdekében, hogy elkerüljem a magyar és angol szövegek közvetlen beleégetését a HTML és TypeScript kódokba, kidolgoztam egy megoldást, amely támogatja a magyar-angol fordítást a programban. Ezáltal a szövegek dinamikusan kezelhetők és könnyen bővíthetők a jövőben. A szoftver elején megnyílik a *SetLanguageComponent*, itt tudjuk beállítani a program nyelvét, itt inicializáljuk a *currentLanguage* értékét. A reguláris kifejezéseknek, a megjelenített- és kimondott szövegeknek külön-külön létrehoztam egy JSON file-t, melyekben a magyar- és angol nyelvhez értékpárokat hoztam létre. Részlet a *text.json* file-ból a 4.4-es kódban:

```
{
  "hu-HU": {
    "notfoundallpairs": "Nem találtad meg az összes párt!",
    }
  "en-US":{
     "notfoundallpairs": "You didn't find all the pairs!",
  }
}
```

4.4. kód. Részlet a text.json file-ból.

Mindkét objektumhoz ugyanolyan nevű értékpárokat rendeltem. Ezeket az file-okat importáltam a komponensekbe, és a *currentLanguage* változó értéke alapján a megfelelő objektumnak az értékeit használtam fel.
A 4.5-ös kód egy részlet a *GameComponent-ből*, ahol a sikeres szöveg tartalmát a *spokenText.json*-ból olvastuk ki, és a *SpeechSynthesis speak()* metódusával mondatjuk ki a programmal.:

```
this.speechSynthesizer.speak(
      this.spokenText.allpairsfound + ". " +
      this.spokenText.gotpoints +
      this.points.toString() + ". " +
      this.spokenText.addaname , this.currentLanguage
    );
```

4.5. kód. Játék eredményeinek felmondása.

4.6. Beszédfelismerés implementálása

A beszédfelismerés megvalósításához alapul vettem egy egyszerű alkalmazást [28], amely a WEB Speech API segítségével több nyelven képes felismerni a felhasználó hangját, majd kiírja a bediktált szöveget a képernyőre, és a nyelv kiválasztásánál fel is mondja a kiválasztott opció nevét. A beszédfelismeréshez elsősorban létrehoztam a *SpeechRecognizerService*-t. Van egy *webkitSpeechRecognition* példánya, amely célja a beszédfelismerés kezelése, a beszéd valós idejű rögzítése és a felismerési események figyelése. Rendelkezik egy *language* változóval, amit a későbbiekben állítunk be hu-HU vagy en-US értékre. Deklaráltam egy *isListening* logikai változót (kezdetben false értékkel), mely jelzi, hogy az alkalmazás hallgat-e, azaz működik-e a beszédfelismerés. A service konstruktorában paraméterként átadtam egy *NgZone* példányt, amely biztosítja, hogy a beszédfelismerési események változásai az Angular-zónán belül kerüljenek kezelésre, és hogy megfelelően frissüljön a felhasználói felület.
Hat függvénye van:

- **initialize()**: ellenőrzi, hogy a *webkitSpeechRecognition* API támogatott-e a böngészőben. Ha támogatott, true értékkel tér vissza, létrehoz egy *webkitSpeechRecognition* példányt, valamint beállítja, hogy a beszédfelismerés folytonos legyen és időközi eredményeket is adjon, illetve a nyelvet is beállítja a saját *setLanguage()* függvényével.

- **setLanguage()**: beállítja a felismerés nyelvét (*recognition.lang*) az *initialize* vagy a futás közbeni változtatások alapján.

- **start()**: az *isListetning* változót true értékre állítja, így elindítja a beszédfelismerési folyamatot.
- **onStart()**: visszaad egy *Observable*-t, amely értesíti a hallgatót, amikor a beszédfelismerés elindul. A *recognition.onstart()* eseményt figyeli, és a *SpeechEvent.Start* eseményt továbbítja.
- **onEnd()**: visszaad egy *Observable*-t, amely értesíti a hallgatót, amikor a beszédfelismerés leáll. A *recognition.onend* eseményt figyeli, és a *SpeechEvent.End* eseményt továbbítja. Valamint itt állítja false-ra az *isListening* állapotot.
- **onResult()**: A recognition.onresult eseményt figyeli, és a tartalmat szöveges formában (transcript) adja vissza, amit az *event.results* tartalmaz. Visszaad egy *Observable*-t, amely a beszédfelismerés eredményeit küldi vissza:
  - **Interim eredmények (időközi tartalom)**: A beszédfelismerés folyamatban van, és még nem biztos az eredményben. Az időközi tartalmat azonnal továbbítja.
  - **Végleges eredmények**: Amikor a felismerés befejeződött egy-egy mondatra, és biztos az eredményben, a végleges tartalmat továbbítja.

Eszerint egy *SpeechEvent*-nek négyféle állapota lehet: *Start, End, FinalContent, InterimContent*. Ezeket egy enumban definiáltam, és a 4.6-os kód szemlélteti.

```
export enum SpeechEvent {
    Start,
    End,
    FinalContent,
    InterimContent
}
```

4.6. kód. SpeechEvent enum.

Ezt a service-t példányosítottam az összes komponensemben, ahol a programot hangalapon kívántam irányítani. Ezalól kivételt képez a *SetLanguageComponent*, mivel itt inicializáltam a nyelv változóját. A komponensekben két függvényt hoztam létre a beszédfelismerés implementálásához:

- **processNotification()**: reguláris kifejezések alapján feldolgozza a felismert szövegeket. A reguláris kifejezéseket a *regex.json*-ból olvastam be. A 4.7-es kód a *GameComponent*-ből azt mutatja meg, hogy hogyan történik a főoldalra navigálás beszédfelismeréssel. Reguláris kifejezést készítünk a *languagePatterns.home* értéké-

nek felhasználásával, ahol a kifejezés a .*főoldal.* mintát tartalmazza, majd megnézzük hogy a felimsert szövegben szerepel-e a "főoldal" szó. Amennyiben igen, visszatérünk a főoldalra.

```
let regexHome = new RegExp(languagePatterns.home);
let testHome = regexHome.test(message);
if (testHome) {
  this.gotohome();
}
```

4.7. kód. Visszatérés a főoldalra beszédfelismeréssel.

– **initRecognition()**: inicializálja a beszédfelismerési folyamatot a komponensben és két adatfolyamot hoz létre:

  • **transcript**: kezeli a beszédfelismerési eredményeket, feldolgozza a szövegeket a *processNotification* segítségével, majd az értesítésekből kinyeri a végleges vagy időközi szöveges tartalmat.

  • **listening**: figyeli, hogy a beszédfelismerés fut-e, ezt az *onStart()* és *onEnd()* eseményekből állapítja meg és true-ra állítja, ha a felismerés aktív (*SpeechEvent.Start*).

4.7. Beszédszintetizáció implementálása

A beszédszintetizáció programozása során szintén abból a programból inspirálódtam, mint a beszédfelismerés fejlesztése során. Ehhez a funkcióhoz létrehoztam a *SpeechSynthesizerService*-t, amely használja a böngésző beépített SpeechSynthesis API-ját. Deklaráltam egy *SpeechSynthesisUtterance* típusú változót, melyet a program később a beszédszintetizálás konfigurálásához és futtatásához használ. A beszédszintézishez az alábbi függvények szükségesek:

– **initSynthesis()**: ez a metódus értékül ad a *speechSynthesizer* változónak egy *SpeechSynthesisUtterance* példányt és beállítja az alapértelmezett hangmagasságot.

– a szintetizált beszéd további tulajdonságának kezeléséhez négy függvényt definiáltam, melyek a későbbiekben a Beállítások menüben szükségesek lesznek:

    - **getVolume()**: hangerő értékének lekérdezése
    - **setVolume()**: hangerő beállítása
    - **getRate()**: lejátszási sebesség értékének lekérdezése
    - **setRate()**: lejátszási sebesség beállítása

  – **speak()**: ez a metódus felelős egy szöveg felolvasásáért. Bemeneti paraméterei a kimondandó szöveg és nyelv, majd a SpeechSynthesis API saját *speak()* függvényét hívja meg.

  – **stop()**: ez a metódus megszakítja az aktuálisan futó beszédszintézist.

A service konstruktorában meghívjuk a *initSynthesis()* függvényt, ezáltal példányosításkor azonnal működni a fog a beszédszintetizáció. A komponensek konstruktorában paraméterként adom át a *SpeechSynthesizerService* egy példányát, és a *speak()* metódusát definiálom felül. A kimondandó szöveget a *spokenText.json*-ból olvastam be a program nyelve szerinti objektumból, a 4.8-as kód példa erre a *HomepageComponent*-ből, ahol kimondatom az alkalmazással az üdvözlő szöveget:

```
this.speechSynthesizer.speak(
  this.spokenText.welcome, this.currentLanguage
);
```

4.8. kód. Üdvözlőszöveg felmondatása a programmal.

4.8. Képernyő felolvasás implementálása

Készítettem egy saját képernyő felolvasó „voiceover” funkciót az alkalmazásnak, melynek lényege, hogy *mouseenter* eseménynél, azaz amikor az egerünkkel belépünk egy HTML elem területére, az oldalon található szövegek tartalmát felolvassa a program. Ehhez elkészítettem a *VoiceoverService-t*, melyben két függvényt definiáltam:

– **voiceover()**: a *SpeechSynthesizerService speak()* metódusával felolvassa a kapott szöveget, melyet a *readingText()* függvény nyer ki a HTML elemből.

– **readingText()**: ez a metódus kezeli (4.9-es kód), hogy adott egéreseményre nyerjük ki az érintett HTML elemből a szöveget, majd az ezzel az értékkel meghívjuk a *voiceover()* függvényt.

```
readingText(event: MouseEvent) {
    const target = event.target as HTMLElement;
    const text = target.innerText;
    this.voiceover(text);
}
```

4.9. kód. A readingText() függvény a VoiceoverService-ből.

Ezután átadtam a komponensek konstruktorában paraméterként a service-t, és a HTML file-ban az összes elemben meghívtam a *readingText()* függvényt a mouseenter eseményre. Ügyeltem arra, hogy ha a felhasználó elhagyja az egérrel az adott elemet, akkor a szoftver szakítsa meg a felolvasást, ezt a *SpeechSynthesizerService stop()* függvényének használatával biztosítottam a *mouseleave* eseményhez kötve. A HTML file-okba a szövegeket dinamikusan töltöttem be a *text.json*-ból, így értem el, hogy magyar vagy angol nyelv kiválasztása után a megfelelő nyelven jelenjen meg az oldal tartalma, és ne legyen beleégetve a forráskódba a szöveg. A 4.10-es kód egy példa a *SettingsComponent-ből*, melynél felolvasható a Beállítások menü címsora:

```
<h2 (mouseenter)="this.voiceoverService.readingText($event)"
(mouseleave)="this.speechSynthesizer.stop()">
{{ loadedText.settings}}</h2>
```

4.10. kód. Beállítások mneű címsorának felolvasása a VoiceoverService-szel.

Néhol a HTML elem nem tartalmaz szöveget, ahogyan az a 4.11-es kódban is látható, viszont fontos információval bír. Erre példa a *HomepageComponent*-ben lévő ikonok. Ezeknél az elemeknél nem használtam a *VoiceoverService* függvényeit, csupán a *SpeechSynthesizerService speak()* metódusával mondattam fel szöveget az ikonok érintésénél.

```
<img style="height: 150px; width: auto;float: right"
src="./assets/settings.jpg" alt="icon"
(mouseleave)="this.speechSynthesizer.stop()"
(mouseenter)="this.speechSynthesizer.speak(
this.spokenText.settings, this.currentLanguage)"
(mouseleave)="this.speechSynthesizer.stop()"
(click)="gotosettings()">
```

4.11. kód. A SpeechSynthesizerService-szel való felolvasás.

4.9. A játék logikájának implementálása

Készítettem egy egyszerű memóriajátékot, melyet a játékos beszéddel is képes irányítani. A játékot egyedül, és párban is lehet használni. Egyszemélyes játékmódban a program visszaszámolja az időt, eközben számolja a játékos pontjait. Sikeres játék esetén megjelenik a *SuccessComponent* dialógusa, bekér egy becenevet és az eredményeket el tudjuk menteni a ranglistára. Ha nem sikerült az összes párt megtalálni, akkor a *RetryComponent* dialógusa ugrik fel, és új játékot kezdhetünk. Kétszemélyes játékmódban kötetlen a játékidő, és pontszámolás sincs, ehelyett a két játékos talált párjainak számát tartja számon a program. A játék végén a *MultiPlayerSuccessComponent* dialógusa kiírja a nyertest vagy a döntetlen eredményt. A játék oldalát a "játék" vagy "game" paranccsal tölthetjük be a főoldalról. A játék és a beállítások közötti kapcsolatot a *GameService* biztosítja, amely felelős a beállítások továbbításáért a *SettingsComponent*-től a *GameComponent* és a *MultiPlayerComponent* felé. Ez a megoldás biztosítja a központi adatkezelést és az összefüggő komponensek közötti hatékony kommunikációt. A két játékmód megfelelő működését az alábbi függvények biztosítják:

- **maxPairs()**(egyszemélyes/kétszemélyes játékban): a *row* és a *col* változók alapján beállítja a játék méretét. Háromszor négyes, négyszer négyes és négyszer ötös játéktáblák érhetők el a Beállítások menüben.

- **shuffle()**(egyszemélyes/kétszemélyes játékban): egy általános véletlenszerűsítő függvény, amely a Fisher-Yates[29] algoritmust használja egy tömb véletlenszerű sorrendbe állításához. Ez a függvény szükséges a kártyák megkeveréséhez a *randomizeCards()* metódusban.

- **randomizeCard()**(egyszemélyes/kétszemélyes játékban): meghívja a *shuffle()* függvényt az *this.cards* tömbre, hogy véletlenszerűsítse a kártyák sorrendjét, majd az első *this.maxpairs* darabszámú elemet kivágja. Az így kapott tömböt megduplázza, hogy minden kártyának legyen egy párja. Az összesített (párosított) kártyákat ismét véletlenszerűsítve összekeveri. Egy üres *this.rows* tömbbe tölti a kártyákat, a *this.row* és *this.col* változók által meghatározott méretű sorok szerint. Végül párhuzamosan létrehozza a *this.flipped* mátrixot is, amely egy-egy kártya fordított állapotát jelzi.

- **setUpCards()**(egyszemélyes/kétszemélyes játékban): betölti a kártyák háttérképét és a játékhoz szükséges kártyaképeket, majd véletlenszerű sorrendbe keveri és elrendezi őket a játéktáblán.

- **selectCard()**(egyszemélyes/kétszemélyes játékban): kezeli a játékos kártyaválasztását. Ha a választás érvényes, felfordítja a kártyát, és ellenőrzi, hogy a két kiválasztott kártya egyezik-e. Ha nem, rövid késleltetés után visszafordítja őket, ha pedig egyeznek, pontokat ad, és ellenőrzi, hogy minden párt megtaláltak-e. A folyamat során visszajelzéseket ad a játékosnak.

- **startTimer()**(egyszemélyes játékban): elindít egy visszaszámlálót, amely másodpercenként csökkenti a hátralévő időt, növeli az eltelt időt, és frissíti ezeket az értékeket a *GameService*-ben. Ha az idő lejár, leállítja az időzítőt, véget vet a játéknak, és megjeleníti a *RetryComponent* dialógusát.

- **stopTimer()**(egyszemélyes játékban): leállítja az időzítőt az aktuális intervalId törlésével, és visszaállítja azt null értékre, biztosítva, hogy az időzítő ne fusson tovább.

- **success()**(egyszemélyes/kétszemélyes játékban): a játék sikeres befejezését kezeli. Leállítja az időzítőt, beállítja a *gameOver* állapotot igazra, és megnyit egy sikerüzenetet tartalmazó párbeszédablakot (*SuccessComponent/MultiPlayerSuccessComponent*).

- **sayCard()**(egyszemélyes/kétszemélyes játékban): a felfordított kártya képének elérési útvonala alapján kimondja a felhasználó felé, hogy mi található a képen.

Természetesen mindkét komponensben megvalósításra került a beszédfelismerés is, így a korábban taglalt *initRecognition()* és *processNotification()* metódusok is szerepet kaptak. A játékot úgy lehet beszéddel irányítani, ha felhasználó kimondja a kiválasztott kártya sorának és oszlopának a számát, például az első sor, első oszlopban lévő kártya felfordításához elég annyit mondani, hogy „első első”, de hosszabb mondatokat is képes feldolgozni az ehhez készített reguláris kifejezés. Erről látható egy részlet a 4.12-es kódban a forráskódból. A *languagePatterns* különböző értéke egy-egy reguláris kifejezése, amelyek a sorok és oszlopok sorszáma szövegesen. Ezeket konvertáljuk át számokká a *this.numbersInWords* objektumban. Ezután a kód teszteli, hogy a megadott sor-oszlop párosban van-e kártya, és felfordítja azt a felhasználónak.

```
this.numbersInWords = {
  [languagePatterns.first]: 1,
  [languagePatterns.second]: 2,
  [languagePatterns.third]: 3,
  [languagePatterns.fourth]: 4,
  [languagePatterns.fifth]: 5
};

let testGame = regexSelectCard.exec(message);

if (testGame) {
  let row = this.numbersInWords[testGame[1]] - 1;
  let col = this.numbersInWords[testGame[2]] - 1;
  if(!isNaN(col) && !isNaN(row)){
    this.selectCard(row, col);
}
```

4.12. kód. Kártyák felfordítása beszédfelismeréssel.

4.10. Egyéb funkciók implementálása

Ebben a fejezetben bemutatom a szoftverem egyéb funkcióit, amelyek a rendszer alapvető működését támogatják.

4.10.1. Segítség ablak

A főoldalról elérjük a *HelperComponent*-et, melynek célja, hogy segítse a felhasználót a program használatában. Egy leírás található az alkalmazásban használt összes parancsról, mellyel irányítani lehet az oldalt. Ez a szöveg felolvasásra is kerül az ablak megnyitásakor. Ez az ablak a „segítség” vagy „help” paranccsal hozható elő.

4.10.2. Ranglista

A *RankingsComponent*-ben a Firestore-ból kilistázzuk az elmentett sikeres eredményeket. Az oldal megjeleníti a helyezésekhez a becenevet, az elért pontot, az időt és a dátumot. Amikor mentésre kerül egy sikeres játék eredménye, az alkalmazás elmondja, hogy hanyadik helyen került a játékos a ranglistára. Az oldal a „ranglista” vagy „rankings” paranccsal hozható elő a főoldalon.

4.10.3. Beállítások menü

A beállítások menü a szoftver egyik kulcsfontosságú része, mivel itt történik a játék különböző elemeinek konfigurálása és az olvasási funkciók egyes tulajdonságainak beállítása. Ezáltal a felhasználók személyre szabhatják a játékélményt és az alkalmazás működését. A „beállítások” vagy „settings” szavakkal hozható elő a *SettingsComponent*, ahol négyféle beállítás áll rendelkezésre:

- **Játéktábla méretének beállítása**: a *BoardSizeComponent* segítségével beállíthatjuk a játéktábla méretét. Háromszor négyes, négyszer négyes és négyszer ötös méret között választhatunk. Alapértelmezetten a háromszor négyes játéktér van beállítva.
- **Játékosok számának beállítása**: a *NumberOfPlayersComponent*-ben kiválaszthatja a játékos, hogy egy- vagy kétszemélyes játékkal szeretne játszani. Alapból az egyszemélyes játék van kiválasztva.
- **Játékidő beállítása**: a *TimeComponent* lehetővé teszi a felhasználónak, hogy beállíthassa az időt az egyszemélyes játékhoz. Választható értékeke: 60, 120, 180 és 300 másodperc. Kezdetben 60 másodperc van kiválasztva.
- **Hangerő és lejátszási sebesség beállítása**: a *VolumeAndSpeedComponent* lehetőséget biztosít a beszédszintetizáció hangerejének és lejátszási sebességének beállítására. A hangerőt 0 és 1 közötti értékekkel szabályozhatjuk, míg a lejátszási sebességet 0.25 és 5 közötti tartományban van lehetőség lassítani vagy gyorsítani. A program megnyitásakor a hangerő és a lejátszási sebesség értéke 1.

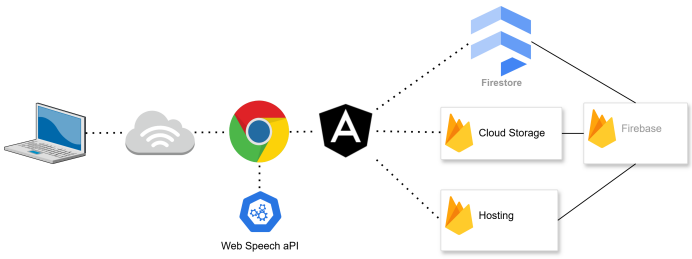
A Beállítások menüben implementálva lett a beszédfelismerés és a beszédszintézis. Az aktuálisan beállított értékeket a program felolvassa, és az opciók kiválasztása hangutasítással is lehetséges.

5. fejezet

Az elkészült alkalmazás ismertetése

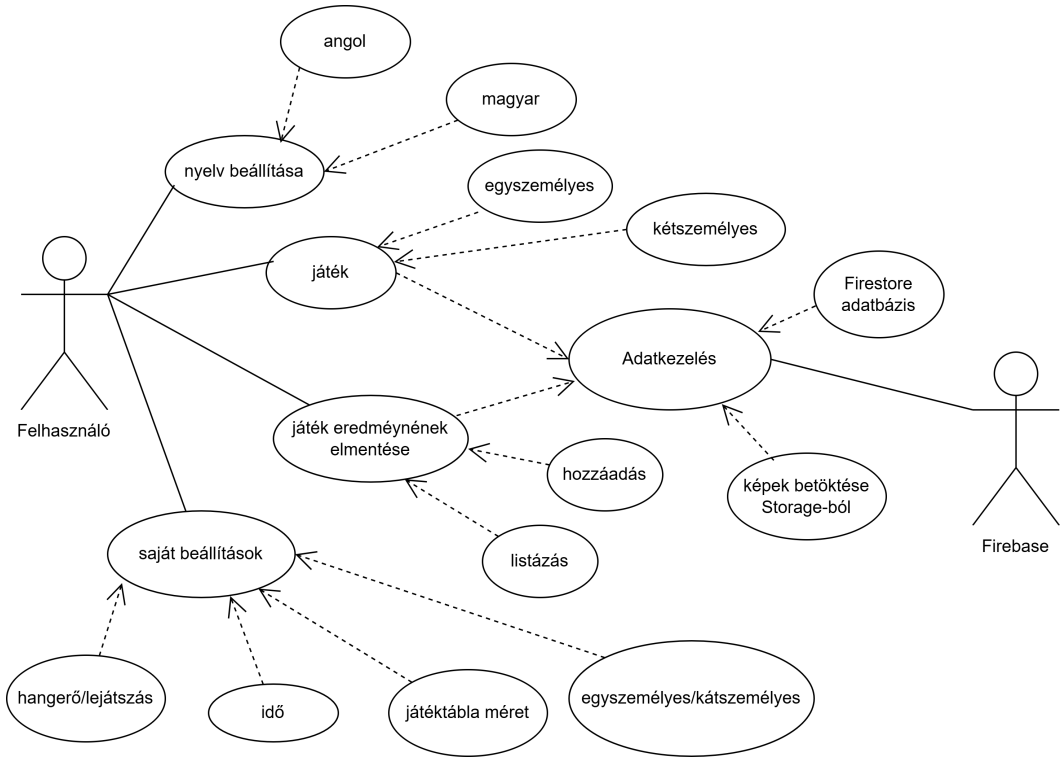
5.1. A szoftver végleges specifikációi

Az elkészült szoftver egy olyan játékot mutat be, amely a Web Speech API SpeechRecognition és SpeechSynthesis részAPI-jait használja, tehát felismeri a felhasználó által adott verbális utasításokat, majd feldolgozza őket a weboldalon, emellett rendszeres visszajelzést ad szóban a felhasználónak beszédszintézissel. A weboldal megnyitásakor a felhasználó kiválaszthatja, hogy magyarul, vagy angol nyelven szeretné használni az alkalmazást, és a program eszerint le lesz fordítva. A program egy egyszerű memóriajátékot tartalmaz, melyet egyedül és párban is lehet játszani. Egyszemélyes játékban pontokat lehet gyűjteni időre. Sikeres játék végén egy becenévvel felkerülhet a játékos egy globális ranglistára. Elérhető egy beállítások menü, ahol a játékos személyre szabhatja a játéktér méretét, időtartamát, beállíthatja, hogy egyszemélyes vagy kétszemélyes játékkal szeretne játszani, illetve a program hangerejét és lejátszási sebességét is megváltoztathatja.
Az alkalmazás egyszerűsített architektúrális nézetét mutatja az 5.1-es ábra. A játék a Firebase szolgáltatásait használja, a Hosting révén az alkalmazás kitelepítésre került a webre, így használatához internet-hozzáférés szükséges. Bármely böngészőben megnyitható, ahol támogatott a Web Speech API. A Firestore szolgáltatással mentésre kerülnek a ranglista adatai, emellett a kártyák képei a Cloud Storage-ban kerültek tárolásra.



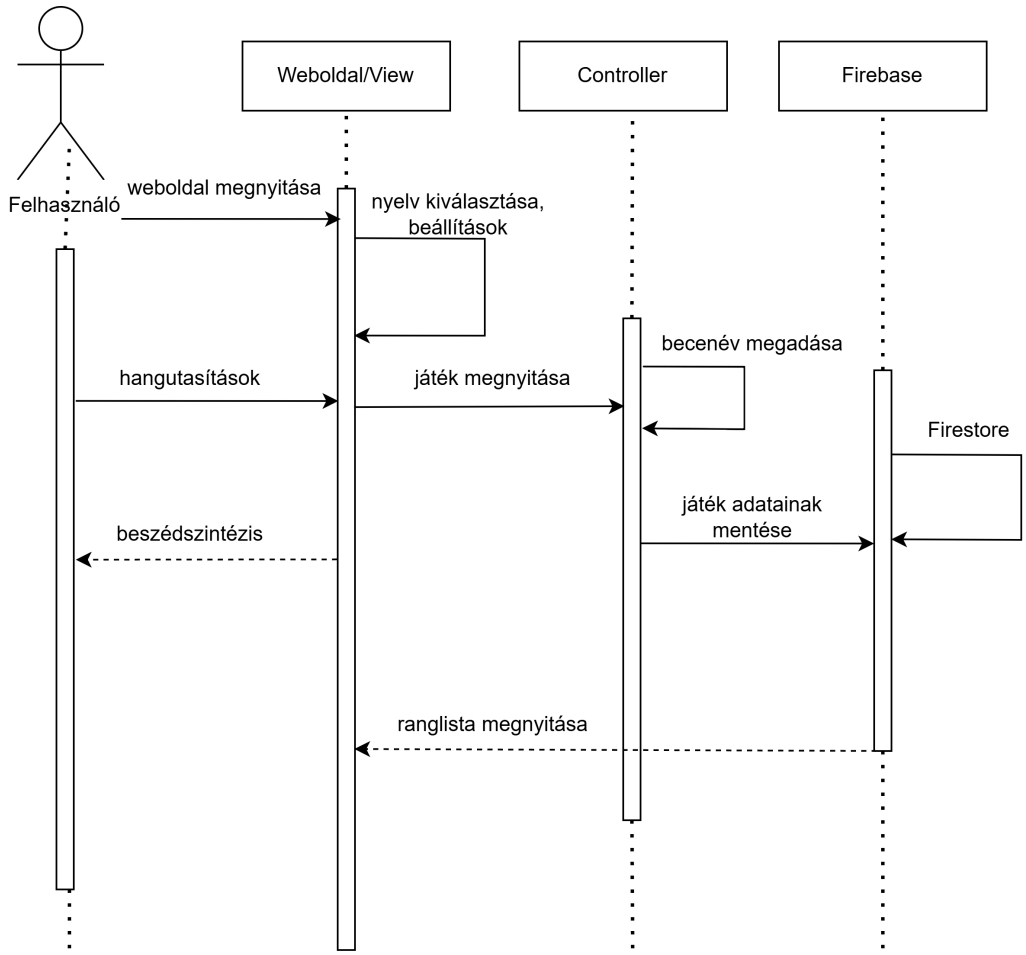
5.1. ábra. A rendszer architektúrája.

A felhasználó képes kétféle játékmódot használni, a program nyelvét beállítani, a beszédszintézis hangerejét és lejátszási sebességét testreszabni, valamint a játéktér méretét, a játék idejét és a játékosok számát is beállítani. Továbbá, a Firestore-ral biztosítva a felhasználó el tudja menteni saját eredményeit a ranglistába a megadott becenévvel együtt, és megtekintheti a használható parancsokat a segítségnyújtó ablakban. Mindezek az esetek a 5.2-es ábrán kerülnek bemutatásra.



5.2. ábra. Use case diagram.

A program szekvenciális diagramja, amelyet a 5.3-as ábrán szemléltettem, bemutatja, hogyan halad a felhasználó a program megnyitásától a játék eredményeinek tárolásáig. A folyamat során a felhasználó képes hangutasításokat kiadni, amelyekre a program beszédszintézis segítségével válaszol. A diagram részletezi, hogyan dolgozza fel a program a felhasználói interakciókat, legyenek azok hangutasítások vagy a grafikus felületen keresztül adott parancsok. A bemenetek feldolgozását követően a program azonnali visszacsatolást küld.



5.3. ábra. Szekvenciális diagram.

A szakdolgaztban szereplő összes ábrát és diagramot a draw.io programmal készítettem el.[16]

5.2. A szoftver hiányosságai, továbbfejlesztési lehetőségek

Igaz a mondás, hogy egy szoftver sosincs kész, mindig tudunk új funkciókat kitalálni a programba, annak működést modernizálni. Ez teljes mértékben helytálló a szakdolgozati programomnál is. Meglehetősen zavaró, hogy a program csak az operációs rendszerben letöltött nyelvi csomagokat tudja használni, mivel, ha a felhasználó nem tölti le a megfelelő csomagokat, akkor a program a beszédet nem ismeri fel és az operációs rendszer alapértelmezett (általában angol) nyelvét használja. Ennek az a hátránya, hogy hiába választottuk ki a nyelvet, a beszédfelismerés és a beszédszintetizáció csak angol nyelven fog működni. A jövőben érdemes lehet olyan megoldások után kutatni, hogy hogyan lehetséges az operációs rendszerben beállított nyelvektől függetlenül implementálni a Web Speech API segítségével a beszédfelismerést és a beszédszintetizációt.
A terveim között szerepelt egy olyan nagyító funkció implementálása, amely képes az egész oldalon minden olyan elemet felnagyítani, amely a nagyító területére esik. Jelenleg JavaScript-ben leginkább képek felnagyítására van lehetőség, de egy általános nagyító eszköz fejlesztése izgalmas és hasznos kihívás lenne.
A weboldal betöltésekor a mikrofon engedélyezése pillanatnyilag manuálisan történik, azaz a felhasználónak kell rákattintani a felugró ablakban az engedélyezés opcióra. Ezt érdemes lehet a jövőben automatizálni, hogy az oldal betöltésekor a program automatikusan engedélyezze a mikrofon használatát, így azt nem kell egy másik, látó embernek beállítani a látássérült felhasználónak.

Összefoglalás

A szakdolgozatom célja egy olyan játék fejlesztése volt, amely vakok, látássérültek és színtévesztők számára készült. A játék hangalapú interakciókat biztosít, így a felhasználók vizuális segítség nélkül is élvezhetik azt. A dolgozatban az akadálymentes alkalmazásfejlesztés elveit is ismertettem, különös figyelmet fordítva a digitális hozzáférhetőség fontosságára, valamint a felhasználói élmény javításának technikai és tervezési aspektusaira. A játék kialakításakor kiemelt figyelmet fordítottam a Web Speech API funkcióira, melyek lehetővé teszik a hangalapú vezérlést és visszajelzéseket, így a felhasználók a szemük használata nélkül navigálhatnak és interakcióba léphetnek az alkalmazással. Megismertem más szolgáltatásokat, technológiákat, melyeket szintén látásukban korlátozott személyek segítésére fejlesztettek, például a képernyőolvasó programokat, a Braille-képernyőket, valamint az audio-alapú navigációs alkalmazásokat, amelyek mind hozzájárulnak a digitális világ akadálymentessé tételéhez és a felhasználók önállóságának növeléséhez. Szakdolgozatom készítése során rendkívül hasznos tudásra tettem szert az akadálymentes alkalmazásfejlesztés terén, különösen a Web Speech API, valamint a felhasználói élmény optimalizálása és a látásukban korlátozott személyek igényeihez igazodó technológiai megoldások alkalmazása terén.

Irodalomjegyzék

[1] Introducing VoiceOver

https://www.apple.com/voiceover/info/guide/$_1121.html$

[2] VoiceOver felhasználói útmutató

https://support.apple.com/hu-hu/guide/iphone/iph3e2e415f/ios

[3] VoiceOver billentyűparancsok

https://support.apple.com/hu-hu/guide/voiceover/cpvokys01/mac

[4] Braille Screen

https://support.apple.com/en-us/101637

[5] Multi-Touch trackpad

https://support.apple.com/hu-hu/102482

[6] Siri

https://www.apple.com/siri/

[7] Bixby

https://www.samsung.com/hu/support/mobile-devices/hogyan-hasznalhatom-a-bixby-alkalmazast/

[8] SciFY

https://scify.org/

[9] gamesfortheblind

https://gamesfortheblind.org

[10] Binaural audio

https://splice.com/blog/what-is-binaural-audio/

[11] Provide control labels with ARIA

https://codelabs.developers.google.com/angular-a11y7

[12] ChromeVox

https://chromewebstore.google.com/search/chromevox?hl=en-USutm$_source = ext_sidebar$

[13] web-speech-angular

https://github.com/luixaviles/web-speech-angular

[14] Pacman

https://github.com/chandradharrao/Voice-Controlled-Games-For-Persons-With-Disabilities

[15] Audio Games For Blind/Low Vision Gamers

https://veroniiiica.com/audio-games-for-blind-low-vision-gamers/

[16] draw.io

https://app.diagrams.net/

[17] A számítógépes beszédfelismerés alapjai

https://www.inf.u-szeged.hu/ tothl/speech/ASRhun.html

[18] Web Speech API

https://developer.mozilla.org/en-US/docs/Web/API/Web$_S peech_A PI$

[19] Using the Web Speech API

https://developer.mozilla.org/en-US/docs/Web/API/Web$_S peech_A PI/Using_t he_W eb_S peech_A PI$

[20] SpeechRecognition

https://developer.mozilla.org/en-US/docs/Web/API/SpeechRecognition

[21] SpeechSynthesis

https://developer.mozilla.org/en-US/docs/Web/API/SpeechSynthesis

[22] Színtévesztés, színvakság

https://bhc.hu/betegsegek/szintevesztes$_s zinvaksag$

[23] Red-Green Color Blindness

https://atmos.style/images/blog/color-blindness-in-ui-design/red-green-color-blindness.png

[24] Blue-Yellow Color Blindness

https://atmos.style/images/blog/color-blindness-in-ui-design/blue-yellow-color-blindness.png

[25] Total Color Blindness

https://atmos.style/images/blog/color-blindness-in-ui-design/full-color-blindness.png

[26] How to Design for Color Blindness

https://www.audioeye.com/post/8-ways-to-design-a-color-blind-friendly-website/

[27] Designing for Color Blindness in UI Design

https://atmos.style/blog/color-blindness-in-ui-design

[28] A Voice-Driven Web Application using Angular and TypeScript

https://github.com/luixaviles/web-speech-angular

[29] Fisher Yates shuffle

https://www.geeksforgeeks.org/shuffle-a-given-array-using-fisher-yates-shuffle-algorithm/

Nyilatkozat

Alulírott Kovács-Bodó Csenge gazdaságinformatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, gazdaságinformatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

Szeged, 2024. december 14. . . . . .

aláíras

Köszönetnyilvánítás

Ezúton szeretném kifejezni mély hálámat Dr. Jánki Zoltán Richárd témavezetőmnek, aki folyamatos támogatásával segítette a dolgozat elkészülését. Különösen hálás vagyok türelméért és segítőkész, támogató hozzáállásáért, amelyek nélkül nem érhettem volna el a kívánt eredményeket.

Hálával tartozom mindazon oktatóknak, tanároknak és szakembereknek, akik egyetemi éveim során nemcsak tudományos ismeretekkel, hanem személyes inspirációval is gazdagítottak. Az általuk átadott tudás és tapasztalat örök értékként kísér majd, és nemcsak a szakdolgozatom elkészítésében, hanem életem minden szakaszában irányt mutat számomra, segítve a folyamatos fejlődésben és a jövőbeni kihívások kezelésében.