AlgoRythmics Game

Cosma CRISTIAN

Szoftverfejlesztés I év

2021

Tartalom

[Bevezető 3](#_Toc74155979)

[Elméleti megalapozás és bibliográfiai tanulmány 3](#_Toc74155980)

[Célkitűzések 6](#_Toc74155981)

[Elméleti megalapozás 7](#_Toc74155982)

[Gyakorlati megvalósítás 7](#_Toc74155983)

[Projekt menedzsment terv 7](#_Toc74155984)

[Projekt tervezés 7](#_Toc74155985)

[Projekt ütemezési terv 8](#_Toc74155986)

[Rendszerkövetelmények 8](#_Toc74155987)

[Funkcionális követelmények 8](#_Toc74155988)

[Külső interface követelmények 9](#_Toc74155989)

[Nem funkcionális követelmények 9](#_Toc74155990)

[A rendszer részletes leírása 10](#_Toc74155991)

[Szerver Oldal 10](#_Toc74155992)

[A kliensoldal 15](#_Toc74155993)

[A felhasznált technológiák 16](#_Toc74155994)

[Visual Studio 17](#_Toc74155995)

[ASP.NET CORE 2.2 17](#_Toc74155996)

[Entity Framework 17](#_Toc74155997)

[JQuery 17](#_Toc74155998)

[SignalR 17](#_Toc74155999)

[Use casek 17](#_Toc74156000)

[Üzembe helyezés és kísérleti eredmények (szoftverek és hardverek esetében) 18](#_Toc74156001)

[A rendszer felhasználása (szoftverek és hardverek esetében) 18](#_Toc74156002)

[Működési környezet 18](#_Toc74156003)

[Következtetések 18](#_Toc74156004)

[Megvalósítások 19](#_Toc74156005)

[Hasonló rendszerekkel való összehasonlítás 19](#_Toc74156006)

[További fejlesztési irányok 19](#_Toc74156007)

[Irodalomjegyzék 20](#_Toc74156008)

[Függelék (beleértve a forráskódot és dokumentációt tartalmazó adathordozót) 21](#_Toc74156009)

[Project Repository: 21](#_Toc74156010)

[Software Requirements Specification: 21](#_Toc74156011)

[Trello Board join link: 21](#_Toc74156012)

# Bevezető

Az online játékok igen nagy népszerűségnek örvendenek a mindennapokban, a gyakran figyelem lekötő multimédia elemekkel rendelkező játékok órákra le tudja kötni a fiatalok figyelmét, ha ezt a hatást ki lehetne használni komolyabb jellegű oktató célú játékok formájában az javítaná a tudás átadás hatékonyságát. Az ilyen kezdeményezéseket nevezik Game-Based learning-nek (GBL) és kutatások által bizonyítottan hatékony. 1

Az AlgoRythmics kutatócsoport fejlesztett egy újabb és korszerűbb online tanulási és oktatási rendszert, amelyik mint elődje a technológiát és a művészetet ötvöző, interkulturális, számítástechnika oktatási rendszer lett. Továbbá ez az online rendszer külön bejáratú felületet biztosit a tanároknak, kurzusok és azok tanulási lépéseinek a megtervezésére. A tanári felhasználóknak még lehetőségük van kutatási célból eredményeket megtekinteni és diagrammokat készíteni az eredmények alapján.

Ezt az online környezetet később számos kutatásban is felhasználták és további sikereket értek el vele az algoritmus oktatás terén. A rendszert az különbözteti meg a többi átlagos algoritmus oktatási környezetektől, hogy lehetőséget ad többféle tanulási módszerre és azon belül több féle igényre szabhatóságra. Videók és animációk terén többféle interaktivitást biztosit és számos olyan megjelenési beállítást, ami elősegíti a vizualizált algoritmus megértését. Az utóbbi időben a megtekinthető algoritmusok tárháza bővült rekurzív algoritmusok és az N-királynő problémát megoldó backtracking algoritmussal.

A kutatásom során áttekintek néhány cikket a kompetitív tanitó jellegű játékok alkalmazásáról az oktatásban. Megvizsgálom, hogy mások miként és milyen eredményességgel alkalmaztak játékot a tanulási folyamat javításában. A látott ötletek alapján terveztem egy rendezési játékot, melynek az a célja hogy két részvevős rendezési versenyek által a felhasználók nem csak időtöltésként versenyeznek egymás ellen, hanem a játék által képesek elsajátítani a számitástechnikában használt algoritmusokat.

A rendezési játék lényege hogy a felhasználó rendezze a tömböt ez az eredeti rendszerben algoritmus levezénylésnek volt nevezve és teljes mértékű interaktivitást jelent. A magas mértékű interaktivitás sokban javíthatja az algoritmus érthetőségét és a tartós tudás megszerzését.

# Elméleti megalapozás és bibliográfiai tanulmány

Az első áttekintett tanulmányban be vezetik a Game-Based Learning (GBL) koncepcióját, ahol komoly célú játékok formájában javítják a tanulás hatékonyságát. A teljesítmény javulás kutatások által bizonyított, ha olyan módon vannak megtervezve a játékok, hogy egyensúlyban legyenek a szórakoztató és az oktató jellegű részletek, olyan környezetet kell adni a tanulóknak, amiben képesek a megszerzet tudásukat felhasználni. Kimondják, hogy a kinézet és hanghatások ellenére a játékok, ha nem szolgáltatnak megfelelő élményt, akkor hamar felejtésbe süllyednek. Amikor egy ilyen játékot tervezünk fontos, hogy egyensúlyban legyenek az oktató és szórakoztató elemek. A cikkben egy olyan kompetitív játékot és tanulási környezetet terveztek mellyel a fiatal középiskolások könnyeden tudnak játszani. Az oktató anyagért és a vizuális megjelenésért felelős oktatókkal együttműködve dolgoztak ki szcenáriókat és karaktereket. Az általuk megvalósított rendszert ingyenesként közzétették az oktatók közösségének.

Megállapították, hogy a játékosok figyelmét és motivációját az tartja fent, ha a játék képes kihívást jelenteni, kiszámíthatatlan, és versenyhelyzetszerű, mindez jobban hajtja a játékost, hogy győzzön. Beszámoltak még arról, hogy az oktatási aspektus megjelenhet rövid leckék, megbeszélések, videó klipek, interaktív tevékenységek, játék és szerep játék formájában. Nagyfokú interaktivitásra van lehetőség és tanulhatnak a hibáikból. Kimondják, hogy a jó GBL programoknak három komponense kell teljesülön a sikerhez: tanulási tevékenységek és a játék elemek ötvözése, változatos interaktív folyamatok és többféle elérhető tanulási megközelítés.

Az olyan játékok esetén ahol történetek vagy cselekmény narratíva van jelen azok eleve rendelkezhetnek oktatói jelleggel és a játékos tudta nélkül zajlik a tudás megszerzés, a játék előrehaladtával gyarapodik a tapasztalat és a gondolkodás. Kifejtik, hogy gyakorlati tudás mellett még a szociális készség is javítható.

Kimondják, hogy az oktató jellegű komponenseket tartalmazó online játék környezetek, képesek mélyebb és értelmesebb tudással felruházni a tanulókat szemben a hagyományos eszközökkel. Ahhoz, hogy egy GBL program sikeres legyen, szükség van ötletes kialakításra, tiszta és könnyedén értelmezhető felhasználói felületre, és egy hatékony részvételre és cselekvésre ösztönző stratégiára. Ezt követően bemutatták a saját tervezésű online kompetitív játékalapú tanulási rendszerüket.

Első sorban beszámol a két lehetséges problémáról, az egyik a mapping és a másik pedig az internet kapcsolat és a többszálas feldolgozást. Felvázolják, hogy milyen lehetőségek vannak és mit kell figyelembe venni, ha használunk egy játékmotort, akkor az sokban megkönnyíti a fejlesztés folyamatát, legyen szó 3D képekről, ütközés észrevétel vagy a felhasználói felület tervezéséről. Felhívja a figyelmet a játékmotrok hátrányára is, mert ha a cél gépnek nincs meg a szükséges hardware követelménye, akkor a játékok nem működnek folyamatosan és akár hibákhoz is vezethet. Játék tervezéskor a játékosok közötti adatcserét a hálózaton általában két módon végzik: Peer-to-Peer vagy Kliens Szerver. Sok féle megvalósítás létezik, de a lényegük az, hogy a szerver nagy számítási kapacitással rendelkezik a munka nagyrésze történhet a szerveren és a kliensek pedig mentesülnek a számitásigényes folyamatoktól. A cikkben a RakNet hálózati motort használták, mert ezáltal egyszerűen és hatékonyabban tudták kezelni a bit folyamokat, a csomagok és a sávszélességet, továbbá még lehetővé teszi a távoli metódus hívást, az automatikus objektum szinkronizációt. A bemutatott motor még lehetőséget ad titkosításra, erőforrás kezelésre, csomag átküldésre és szál kezelésre, sokban javítva a fejlesztés folyamatát. Az Irrlicht játékmotor által szolgáltatott mapping és környezet felállitással illetve a RakNet hálózati motor funkcióival tervezték meg a cikkben bemutatott játékot.

A cikkben bemutatott rendszer egy online kompetitív többjátékos játék és tanulási rendszer, Miután a játékosok bejelentkeznek, lehetőségük van csatlakozni egy játékteremhez vagy egy új játéktermet létrehozni. Minden játék menetnél beállítható 2-4-ig a játékosok száma, alapvetően nagyon hasonló az elterjedt Monopoly puzzle játékhoz. A résztvevőknek meg kell válaszolniuk egy kérdést minden körben, a játékosoknak érmék és pontok gyűlnek, amik felhasználhatók a játékban. A kapható jutalom az a kérdések és feleletek résznél dől el a teljesítmény szerint. Amikor pedig egy játékos áthalad egy másik játékos területén akkor ismét kérdéseket kell megválaszolnia. A megszerzet pontokat felhasználhatják tippek megszerzésére. A játék paraméterei egy adminisztrátor által adatbázison keresztül állíthatók. A játékot úgy tervezték, hogy a felhasznált tananyagot előre összeállították, és a program fejlesztőkkel segítették a tananyagok megtervezésében. A játék képes magasfokú interaktivitást biztositani, a játékosok felhasználhatják pontjaikat tippekért. Lehetőséget biztosit a személyes fejlődésre azáltal, hogy ha szeretnének fejlődni akkor egy személyes tanuló szobában elérhetnek az oktatók által biztosított tananyagot.

A rendszer felépítését tekintve a kliens oldalon az Irrlicht játékmotor és a RakNet hálózatimotor együtt felel a megfelelő megjelenésért és a szerverrel való kapcsolatért. A szerver oldalon pedig a RakNet hálózatimotor és egy SQL modul felel a kliensekkel való kapcsolatért és az adatok tárolása és olvasásáért.

A lényeges játék mechanizmusok a tanulási pontok gyűjtése, amit felhasználnak a tippek igénylésére, ez fontos az interakció szempontjából. A motiváció és a hatékonyság fokozására egy-egy játékmenetbe 2-4 játékos vehet részt ezzel versengést idézve elő. A gyakorló szoba lehetősége lehetővé teszi a fejlődést, gyakorolhatnak egyjátékos módban. A kérdések és válaszok minden kör végén felmerülnek és gondolkodásra készteti a játékosokat és valós időben kell válaszolniuk. A tippek pedig nem csak oktató jellegűek, de az is a céljuk, hogy a játék élvezhetőbb legyen, kérdésenként 3 tipp volt elérhető és ennek köszönhetően a nehezebb kérdések ez nem rettenték el a játékosokat.

Ezt követően a játék működését részletezték. Az irodalomban beszámolnak egy kísérletről is ahol 35 középiskolás elsőéves diákot mértek fel ebből tizenhét fiú és tizennyolc lány volt. A bemutatott rendszert tíz héten át használták, egy előtesztben és egy utótesztben mérték a megértést és a rendszer hatékonyságát. A felmért tanulók 65.7%-a rendelkezett előzetes tapasztalattal az online játékok területén. Az eredmények kimutatták, hogy a hatékonyságban a nemek között nincs szignifikáns eltérés. Az előzetes tapasztalatok az online játékokkal nem bírt jelentőséggel. További statisztikai tesztek kimutatták, hogy a 10 hét alatt végzet kísérlet végére szignifikánsan jobban teljesítettek a tanulók használva a játékot. A használat és az elégedettség felmérése céljából kérdőívet töltettek ki a tanulókkal. A kérdőívből kiderült, hogy a felhasználói felület sokat számit a játékosok motivációja szempontjából. Alátámasztást nyert, hogy a kompetitív játék élmény növeli a tanulás hatását és az érdeklődést. A tanulás szempontjából az egyjátékos módú tanuló szobák bizonyultak a leghatékonyabbnak.

Az áttekintet irodalomban tehát megterveztek és kiviteleztek egy online többjátékos kompetitív játék és tanulási rendszert, felmérték annak hatékonyságát és követendő támpontokat adtak hasonló rendszerek tervezéséhez.

# Célkitűzések

A fejlesztendő rendszer versenyképes tanulási tevékenységet kínál mindenki számára, aki használja. A felhasználónak lehetősége lesz versenyezni a többi felhasználóval egy rendezési / keresési versenyen. Az a felhasználó nyeri a mérkőzést, aki befejezi a válogatást / keresést a másik előtt. A rendszer az eredeti Algorythmics platform része lesz, és új szempontot nyújt a tanulási folyamat számára, ahol a felhasználók megtanulhatnak vagy gyakorolhatnak, miközben élvezik a verseny légkörét.

A célom tehát egy olyan oktató rendszert tervezni, amelyik az eredeti ötletet tovább gondolva, kompetitív játék mechanikával fejlesztve szórakoztató és egyben oktató jellegű játékot produkáljon.

# Elméleti megalapozás

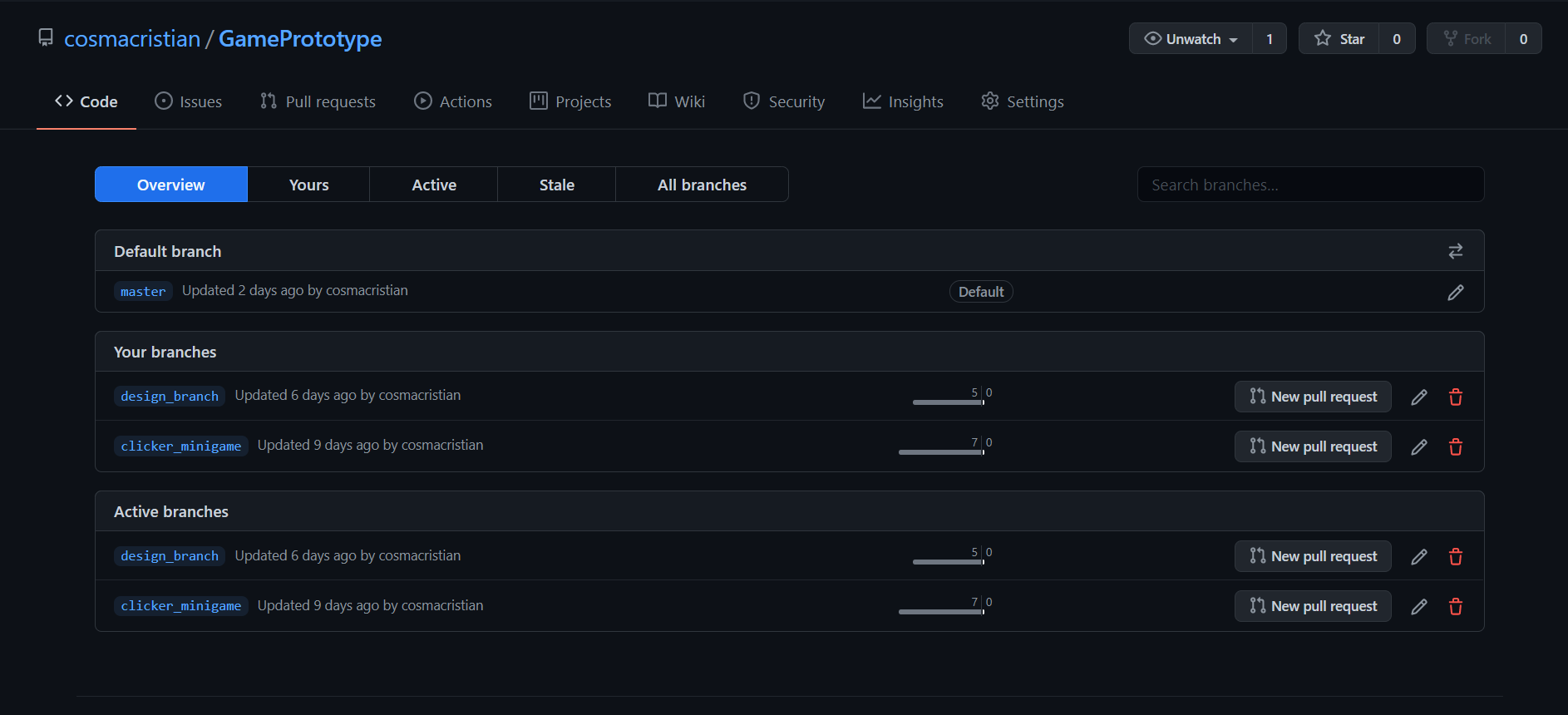
Ahogy az áttekintet irodalomban láttam az oktató jellegű játéknál fontos, hogy milyen mennyiségben van jelen az oktató és a szórakoztató elem. Figyelmet kell fordítani az át adni kívánt tudás fajtájára és hogy miként lehet azt becsempészni a játékélménybe. Láthattam egy példát, hogy a rendszert miként valósították meg és hogy milyen funkcionalitások és interaktivitások lettek integrálva. A látottak alapján terveztem meg a rendszerem és alakítottam ki a játék mechanizmusokat.

# Gyakorlati megvalósítás

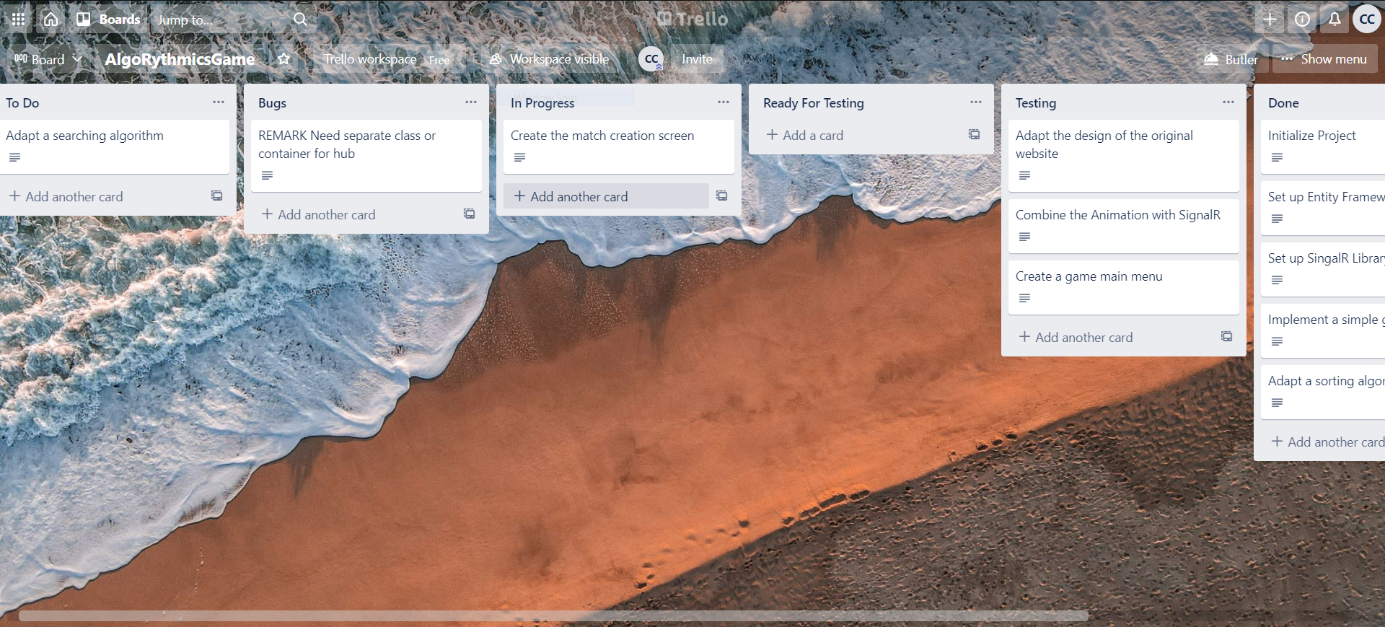
## Projekt menedzsment terv

### Projekt tervezés

A projekt megvalósításában verzió követést használtam GitHub-ban, itt a Visual Stúdio-ba integrált verzió követéssel együtt használtam és szerkesztettem a kódbázist. Itt tárolásra került mind a szerveroldal mind a kliensoldal megvalósítása. Külön figyelmet fordítottam arra hogy az új megvalósításokat minden ticket-nek megfelelően előbb külön branch-en valósítsam meg és ha működik akkor azt később merge-jem a master branch-be.



A feladatokat issue-kra osztottam és lépésenként valósítottam meg a rendszert, az issue-kat Trello projekt management alkalmazásban rendszereztem és tartottam számon. Külön figyelmet fordítottam a feladatok különböző fázisaira ennek megfelelően megállapítottam a következő fázisokat: Todo, Bugs, InProgress, Ready For Testing, Testing, Done.



### Projekt ütemezési terv

|  |  |  |  |
| --- | --- | --- | --- |
| Tervezett dátum | Feladat | Határidő | Számitott Időtartam |
| 2021.04.05 | Bibliográfiai kutatás  Trello beállitás | 2021.04.15 | 5 nap |
| 2021.05.02 | Fejlesztési környezet felállítása  Git verziókövetés | 2021.05.5 | 2 nap |
| 2021.05.06 | Projekt létrehozás  Entity Framework és adatbázis | 2021.05.07 | 1 nap |
| 2021.05.08 | SignalR könyvtár beállítása és kezdetleges játék készítése | 2021.05.09 | 1 nap |
| 2021.05.09 | Egy rendezési algoritmus vizualizáció megvalósítása | 2021.05.11 | 2 nap |
| 2021.05.12 | Kliensoldali design | 2021.05.13 | 1 nap |
| 2021.05.13 | Rendezési játék megvalósítása | 2021.05.19 | 5 nap |
| 2021.05.20 | Játékmenü megalkotása | 2021.05.24 | 3 nap |
| 2021.05.25 | Több algoritmus készítése | Még tart | … |
| 2021.05.26 | Dokumentáció megírása | Még tart | … |
| ??? | Tudományos kísérlet végzés |  |  |

## Rendszerkövetelmények

### Funkcionális követelmények

* Adatkezelés az Entity Framework segítségével: Mivel az eredeti projekt entitás keretrendszerrel jött létre, a projekt az egyszerűbb adatkezeléshez a 2.2.6 verziójú entitás keretrendszert is igényli.
* Keret: Az eredeti projekt .NET Core 2.2 keretrendszerrel készült, a játék prototípusa ugyanazt a keretrendszert fogja használni.
* Programozási nyelv: Mivel .NET Core 2.2-t használunk, az alapértelmezett C # verzió 7.3.
* Kliens oldali könyvtárak: Kliens oldalon a JQuery 3.3.1 és a Bootstrap 3.3.7 programokat használták, és ezeket a könyvtárakat a játék prototípusába is integrálják.
* A rendszer architektúrája kliens-szerver architektúrán fog alapulni. A szerver oldal kezeli a beérkező kéréseket, ez kezeli a játékokat és azok adatait. A kliens oldal megjeleníti a játék felületét és az animációkat a játék során. A két HTTP-kérelem kommunikációként szolgál, a játék előrehaladása és más adatok az AJAX használatával aszinkron HTTP-kérelmekkel kerülnek át a kliensről a szerverre.
* A szerveralkalmazás fejlesztése során az MVC építészeti mintát fogjuk használni a rendszer jobb strukturálásához és áttekintéséhez.

### Külső interface követelmények

* Relációs adatbázis: Az eredeti alkalmazás az SQL Server 2014 Service Pack 1 szervizcsomagot használta, ezért ugyanazt az SQL Server adatbázist fogja használni a fejlesztés során.
* Operációs rendszer: A tárhely-környezetnek a Microsoft .NET Core 2 \* operációs rendszer követelményeinek megfelelő operációs rendszerek egyikének kell lennie.
* Ügyféloldallal: HTML, Javascript támogató böngésző.

### Nem funkcionális követelmények

* Elérhetőség: A játékplatformnak elérhetőnek kell lennie a megadott dátumon és időpontban, mivel a felhasználók idő múlására vagy gyakorlásra használhatják.
* Helyesség: A játéknak egyenlő esélyeket kell kínálnia mindkét játékos számára a győzelem elérésére, így elősegíti a megfelelő algoritmusok használatát a megfelelő bemenethez.
* Karbantarthatóság: A játéknak karbantarthatónak kell lennie, és meg kell felelnie az eredeti rendszerben használt szabványoknak. A játék később beépül a fő rendszerbe.
* Használhatóság: A játéknak maximális számú felhasználót kell kielégítenie.

## A rendszer részletes leírása

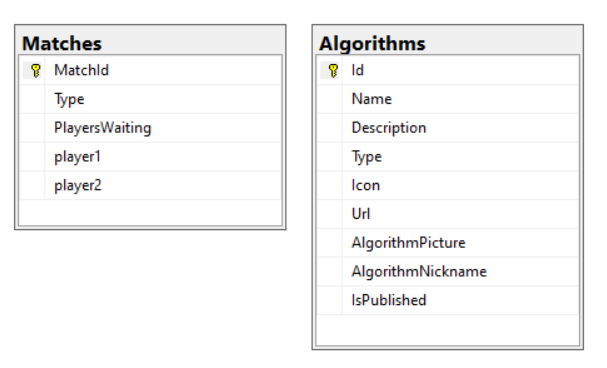
A megvalósított rendszer három részből áll, egy kliens oldali, egy szerver oldali és egy adatbázisból. A szerver oldal C#-ban íródott és APS.NET Core 2.2-es keretrendszer használatával, ezek mellett még Entity Framework 2.2.6-os verziót az adatok kezelésére. A kliens oldal és a serveroldal valós idejű adatcseréjéhez pedig SignalR 1.0-es nyiltforráskódú függvénykönyvtárakat használtam.

A kliens oldal HTML, JavaScript és CSS-re épülő weboldalak összesége. Az egyszerűbb megjelenés tervezéshez Bootstrap 3.3.7-es keretrendszert használtam. Az animációk és egyéb JavaScript feladatok megvalósításában JQuery 3.3.1-es függvény könyvtárat használtam. Itt is a valós idejű adatcseréhez a SingalR kliens oldali JavaScript könyvtárát használtam.

### Szerver Oldal

A rendszer szerver oldala Modell View Controller (MVC) tervezési mintát követ az áttekinthetőbb felépítésért és a könnyebb kezelhetőségért. Ennek a szoftvertervezési mintának az az előnye, hogy az adatok tárolásáért és a különböző felhasznált osztályok a Modell-ben kapnak helyet, ide gyűjtve minden olyan osztályt amit majd a folyamatok működéskor igénybe vesznek. A kliens oldalról érkező HTTP metódus kéréseket a Controller-ek fogadják, a Modell-ek et igénybe véve elvégeznek egy feladatot és válaszolnak a kliens oldalnak egy eredménnyel. A View pedig a megjelenésért felelnek, ezek a forráskódok már a kliens oldalán értelmeződnek, a kliens oldal a web alkalmazások esetén egy böngésző. Amikor egy kliens kérelme kiszolgálásra került akkor a Controller egy View-t térit vissza a kliensnek, amit az majd értelmez és felhasznál.

A szerver oldalon az adatok tartós elmentésére adatbázis vettem igénybe, ez az eredeti rendszerhez hasonlóan egy SQL Server 2014-es adatbázis, a táblák tervezését Code First megoldással végeztem az Entity Framework segítségével. Igy tárolásra került a felhasznált animációk algoritmusainak és a megrendezésre került játék menetek adatai. Ezek azok a modellek, amiket nem csak használtam a működés közben, de el is tároltam hosszabb időre.



Más modellek, amiket még használtam az úgynevezett ViewModellek ezeknek kifejezetten az a céljuk, hogy a View-okal együtt a szükséges külön összeállított adatokkal együtt legyen elküldve a klienshez. Ezek csupán a nézetek külön adat szükségletei miatt lettek létrehozva. Továbbá szót kell ejteni az Enums osztályról, ez olyan előre meghatározott értékeket tartalmaz, amelyek az animációk megjelenését, viselkedését és interaktivitását vezérlik. A Player osztály pedig csupán azért lett létrehozva, hogy egyszerűbben kezelhessem a beérkező felhasználókat és a hozzájuk tartozó WebSocket-eket.

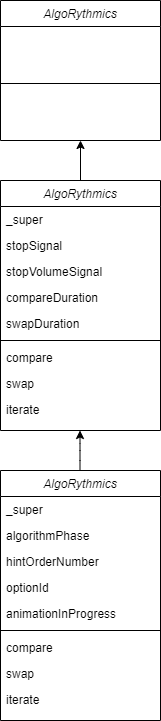
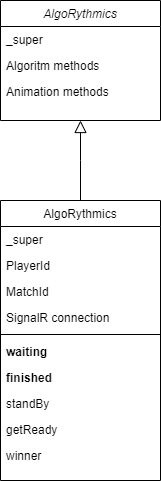
A kontrollerek között két osztály van, az első a HomeController ez fogadja a felhasználókat első érkezéskor és itt megtekinthetik az elérhető játék meneteket, megjelenik a játékmenet azonosító, a belépet játékosok száma és egy játékba belépés gomb. A kontroller ezt a nézetet elkészíti, ki olvassa az adatbázisból az elérhető játék meneteket és együtt visszatéríti azokat a klienshez.

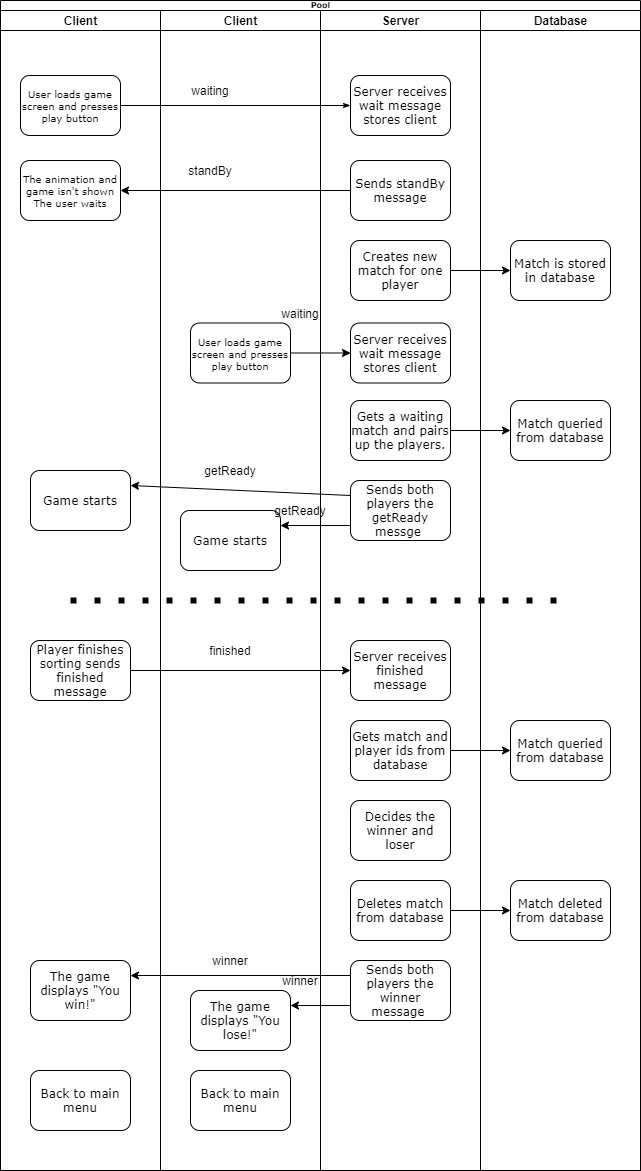
A GameController pedig a játék elemeinek a betöltéséért felel. Ez a vezérlő úgy állítja össze a választ és a szükséges nézetet, hogy a megfelelő algoritmus, a megfelelő megjelenés és a megfelelő interakció szint jelenjen meg a kliensnél. Itt az adatbázisból lekérem az algoritmusom, be állítom a szükséges adatokat az AnimationViewModel és az AlgorithmViewModel-eknek. Itt azért szükséges összeállítani az adatokat, mert ezen a ponton már eldől, hogy a játékmenet milyen algoritmussal fog működni, és hogy milyen beállításokkal fog az animáció rendelkezni. Amikor ezek az adatok megérkeznek a kliens oldalra a nézet már úgy lesz kigenerálva, hogy a megfelelő algoritmus JavaScript kódja és a szükséges beállítások már megjelennek, a JavaScript kódok pedig már úgy futnak, ahogy azt az algoritmus elvárja.

A nézetekből pedig vannak teljes és részleges nézeteim a teljes nézetek önálló HTML dokumentumok, amelyek majd a bővülnek a megfelelő részleges nézetekkel az szerint, hogy mit kell megjeleníteni. Az egyik ilyen teljes nézet az \_Layout.cshtml, ami általános keretet biztosít minden oldal tartalmának, ennek belsejében az @RenderBody() razor kulcsszó hatására fog betölteni egy megjeleníteni kívánt tartalmat. A megjeleníthető tartalmak részleges nézetek formájában vannak jelen például a Game/Index.cshtml, ami önmagában értelmezhetetlen egy böngészőnek, de be illesztve a megfelelő nézetbe megjeleníthető lesz. És ahogy egymásba ágyazhatunk, nézeteket úgy be illeszthetek részlegest a részlegesbe. Maga a játék felülete úgy épül fel a szerveroldalon, hogy a razor kód végrehajtódik, a megfelelő részletek be állitásra kerülnek és a különböző nézetek, mint a \_Layout az Index és a \_CoordinatingPartial a megfelelő sorrendben generálódnak. Végül a megfelelő JavaScript és CSS állományok az összerakott HTML állománnyal a kliens gépén betöltődnek.

Amiről még fontos szót ejteni az a szerver oldalon az aszinkron kommunikációért felelős SignalR Hub. Itt a GameHub osztály kiterjeszti a SignalR.Hub ősosztályt és lehetőség nyílik a megszabni a lehetséges metódus hívásokat. A játékvezérlőként szolgáló GameHub-nak dependency injection-al átadtam egy elérést az adatbázist kezelő Entity Framework kontextushoz. A megrendezett játékokat tárolom az adatbázisban és ameddig használatban vannak, addig ott maradnak és ha már lejártak akkor törlődnek a táblából. A GameHub osztálynak még van egy statikus elérhetőségű kulcs-érték táblázata ahol a fogadott játékosok WebSocket-ét és egyedi azonosítóját tároltam, csupán azért, hogy elkerüljem a regisztráció és a WebSocket állandóságának nehézségeit. Igy a játékhoz bárkinek van hozzáférése és nem igényel időt csatlakozni a játékhoz.

A GameHub osztály két metódust definiál a waiting és a finished üzenetekre. A waiting típusú üzenet a kliens akkor küldi, ha belépet egy játékba és várakozik a kezdésre. Amikor egy játékos várakozik és nincs meg mindkét játékos egy menethez akkor feljegyezi a játékost és „standBy” üzenetet küld vissza. Ha pedig megvan mindkét játékos egy menethez, akkor az OrganizedMatch osztálynak a status adattagját zártnak minősiti és mindkét félnek elküldi a „getReady” üzenetet melynek hatására a kliensoldalon a játék előkészül és indul. A finished tipusú beérkező üzenet hatására a matchId alapján ki olvassa az adatbázisból a játék menetet, az alapján és a kapott userId alapján megkeresi a játékosokat és beállítja a nyertes és a vesztes pozíciókat. A játékmenet törlődik az adatbázisból és a játékosoknak elküld egy „winner” típusú üzenetet, amiben megírja a végeredményt a játékosnak személyre szabottan.



### A kliensoldal

A kliensoldalon a megfelelő animációk dinamikus tartalom betöltéssel állnak össze. Amikor egy felhasználó megpróbál elérni egy algoritmust a játékban akkor még szerveroldalon meghatározódnak a betöltendő JavaScript állományok és különböző részleges nézetek nevei és elérési útvonalai. Az algoritmus vizualizációkhoz szükséges JavaScript állományok: az algoritmus JavaScript implementációja, az összes algoritmus számára elérhető függvények, az éppen használt animáció és interakció JavaScript állománya, a felsorolás típusokat tartalmazó enums.js és a naplózás szempontjából lényeges utilities.js. Mindezek által, áll össze a vizualizációt biztosító rendszer.

Hogy miért beszélek rendszerről és, hogy miként áll ez össze arra a JavaScript Prototipikus Objektum Orientáltsága a válasz. A prototipikus objektum orientált programozás abban tér el a megszokott objektum orientált programozástól, hogy a megszokottnál osztályokat hozunk létre, amiket példányosítva objektumok lesznek és ezek az osztályok ilyenek maradnak, ha megváltoztatjuk az osztályt, akkor egy új fajta objektumot fogunk példányosítani. Ezzel szemben a prototipikus esetében nem írunk osztályokat és nem példányosítjuk azokat, hanem csak létrehozzuk az objektumot és célnak megfelelően előkészítőjük, ha pedig szeretnénk változtatni rajta, akkor nem kell semmit átalakítanunk, hanem klónozzuk a meglévő objektumokat és bővítjük azokat.

A rendszer esetében egy objektumot bővítek mindig a megfelelő adattagokkal és metódusokkal, ezt az objektumot AlgoRythmics-nek neveztem és minden algoritmus vizualizáció esetén betöltődik és elkészül az állandó adattagokkal és függvényekkel, amiket a mutualFunctions.js állományban határoztunk meg. A mutualFunctions.js-ben a központi objektumot mindig kibővítem általános adat tagokkal, amelyeket az animáció és minden algoritmus esetében felhasználunk. Továbbá olyan függvényekkel látom el, amelyek minden algoritmus vizualizációja során előfordulnak, ezek le is lehetnek implementálva vagy előfordulhat, hogy nincsenek le implementálva, hanem ezeket fogják majd később felülírni a frissen betöltött állományok. Az egyik lényeges függvény az az initComponents, ennek az a szerepe, hogy a szerveroldalról érkezett adatokkal inicializálja a szükséges adatmezőket és ezzel előkészítse a megfelelő vizualizációt a beállítások szerint.

A mutualFunctions.js-be kerültek még olyan általános függvények, amelyek az egyszerű algoritmus animációkhoz tartozó vizualizációs elemekért felelnek, illetve a vizualizációt vezérlő felületi elemek kezelő függvényei is és olyan függvények, amelyek nem tartoztak igazán sehova, de fel vannak használva több helyen is.

Az Algorythmics rendszerében az algoritmus vizualizációját az algoritmusok JavaScript kódja irányítja, ezek olyan implementációi az algoritmusoknak melyek tartalmaznak vizualizációs műveleteket. Az algoritmusok JavaScript állományai az algoritmus típusa szerint vannak könyvtárakba osztva, az algoritmus könyvtár alatt található a backtraking, a keresési, és a rendezési algoritmusok könyvtárai tartalmazzák az algoritmusokat és ezek alapján tudjuk dinamikusan betölteni a vizualizációk esetében.

Amiről még fontos szót ejteni az a miniGame.js, ami szintén rá épül az algorythmics rendszerre és felül ír különböző lényeges mozzanatokért felelős függvényeket. Az általános algoritmus vizualizációval szemben itt vezényszóra fog elindulni a játék menet. Amikor be töltődik a játék oldal létre jön egy kapcsolat a szerverrel. Amikor megnyomják a Play gombot akkor elküldődik egy wait üzenet a szerverre és ott attól függően hogy megvan a szükséges játékos szám elindul a játék menet vagy a játékosnak várakoznia kell.

Ha nincs elég játékos, akkor „standBy” üzenet érkezik és nincs megjelenítve a rendezendő tömb. Ha pedig összegyűlt a két játékos akkor „getReady” üzenet érkezik ami hatására az animáció előkészül, megjelenik a tömb és a játék elindult.

Amint az egyik játékos rendezte a tömböt az animáció befejező szakaza beindít egy üzenet küldést ami „finished” üzenetet küld a szerverhez. Az alapján hogy ki küldte eldől a nyertes és a szervertől érkezik egy „winner” üzenet a játékosnak személyre szabva hogy nyert vagy vesztett. Abban az esetben, ha a játékos nem fejezte be a rendezési feladatot, a játék megszakad és mindkét játékos visszatér a fő menübe.

A felhasznált technológiák

A megvalósításhoz használt technológiák:

* Visual Studio 2017
* C# 7.3 ASP.NET CORE 2.2
* Entity Framework 2.2.6
* JQuery 3.3.1
* BootStrap 3.3.7
* SignalR 1.0
* SQL Server 2014 Service Pack 1
* SQL Server Management Studio
* Git

### Visual Studio

A szerver megalkotásához visual studio-t használtam egyrész volt beépített SQL explorer az adatbázis böngészésére, volt beépített verzió követés amit sokat használtam. Könnyedén fel lehetett telepíteni a szükséges technológiákat a projekthez.

SQL Server Management Studio

Az adatok böngészésére és a diagrammok generálására használtam.

### ASP.NET CORE 2.2

A webalkalmazás tervezésére és könnyebb kivitelezésére, platform független web alkalmazás ezáltal bármilyen szerveren deploy-olható.

### Entity Framework

Az adatbázis táblák egyszerűbb karbantartása szempontjából, Code First megoldással és migration-ok segítségével. Az egyszerűbb adat lekérdezésekhez és kezelésekhez.

### JQuery

Az animációk és egyéb kliens oldali műveletek elvégzésére használtam sokkal egyszerűbben lehetett kezelni a felületi elemeket.

### SignalR

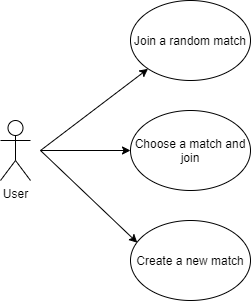
A SignalR nyiltforráskódú függvénykönyvtárak mellyel hatékony kétoldalú aszinkron kommunikáció valósítható meg. A SignalR alapvetően WebSocketre épülő megoldása révén hatékonyabb kommunikációt biztosít a szerver és a kliens között.

## Use casek

A játék nem követeli meg a felhasználóktól a hitelesítést vagy a felhasználói fiók létrehozását. Ez a játék ingyenesen hozzáférhető bárki számára, aki ellátogat a weboldalra. A rendszer felhasználóinak képesnek kell lenniük arra, hogy csatlakozzanak vagy létrehozhassanak egy játékmunkát (1. ábra), ahol beállíthatják a játék különböző paramétereit. A játékfolyamat beállítása után a felhasználók az előcsarnokban várhatnak az ellenfél csatlakozására. Amint egy másik játékos csatlakozik a játékhoz, a mérkőzés a játék felületének bemutatásával kezdődik.

A játék csak maximális játékosszámot fogadhat, ez a szám 100 körül van, ezzel az aktív játékok maximális száma nem haladhatja meg az 50 értékét.

A felhasználók várhatóan napi 8:00-tól 20:00-ig fogják használni a játékot.



# Üzembe helyezés és kísérleti eredmények (szoftverek és hardverek esetében)

# A rendszer felhasználása (szoftverek és hardverek esetében)

## Működési környezet

A játék prototípusának működési környezete:

* MSSQL relációs adatbázis a tárhely eszközön
* IIS bővíthető webszerver az alkalmazás tárolására
* Operációs rendszer Microsoft Windows Server 2016

# Következtetések

Tehát áttekintettem különböző kezdeményezéseket a játékok alkalmazására algoritmus és más témák oktatására, külön megvizsgáltam a kompetitív játékok hatásait az oktatási folyamatra, és megpróbáltam magam is létrehoztam egy hasonló kompetitív játékot. Az AlgoRythmics rendszer bővítése ként elképzelt játék prototípus nem csak szórakoztató időtöltés, hanem játék közben elsajátítható az algoritmusok működése.

## Megvalósítások

## Hasonló rendszerekkel való összehasonlítás

## További fejlesztési irányok

Amint mindenképp szeretnék tovább fejleszteni, az hogy minél több algoritmusra működjön a játék. Egy játékos módot is bevezetni ahol gép vezérelt játékos ellen lehet gyakorolni és tanulni, ahogy a gyakorló szobát megvalósították az áttekintett irodalomban.

Szeretnék kutatásokat végezni ahol diákokon felmérhetném a megvalósított rendszer hatékonyságát.

# Irodalomjegyzék

1. Cheng, Y. M., Kuo, S. H., Lou, S. J., & Shih, R. C. (2012). The Construction of an Online Competitive Game-Based Learning System for Junior High School Students. Turkish Online Journal of Educational Technology-TOJET, 11(2), 214-227.

# Függelék (beleértve a forráskódot és dokumentációt tartalmazó adathordozót)

## Project Repository:

https://github.com/cosmacristian/GamePrototype

Software Requirements Specification: https://docs.google.com/document/d/1pfzMuowi6qIFPPiqnvBiAUle9KgPc51FkotRNgHkt70/edit?usp=sharing

## Trello Board join link:

https://trello.com/invite/b/p5nz7d6p/25f551685fd647d152eab25a994aa264/algorythmicsgame