Szegedi Innovatív Informatika Verseny

2022

SZIIV Program (2022. április 8. – péntek)

Időpont	Műszaki informatika szekció	Informatika szekció
8:30–	**Regisztráció**	
9:00–9:15	**Megnyitó**	
9:20–-11:00	**Előadások I.**	
9:20–9:40	**Kandó Racing Team:** Multikar 2.0	**The Avengers:** Jarvis
9:40–10:00	**OFK:** Okos, fűthető kutyatál	**Kerielit:** Kerikriell
10:00–10:20	**BOT:** IAQ BOX	**csapatnév_végleges(1)másolata:** ElmeSéma
10:20–10:40	**EMRLD:** Xnano és Xos	**Szücs Barna:** A mozgás és a játékélmény összekapcsolása
10:40–11:00	**Pipeline:** Csőtörés-jelző rendszer	**Level 40 wizard:** Vírus szimuláció
11:00–11:20	**Kávészünet**	
11:20–13:00	**Előadások II.**	
11:20–11:40	**ThatNetworkGuy:** wSnake hálózat	**Triangular Frogs:** Laser Cats
11:40–12:00	**αlpha:** Alfa sugárzás kvalitatív és kvantitatív vizsgálata	**Dark Themes Only:** CRVER
12:00–12:20	**Team Elon:** LEGO X-Y plotter – Elon's Plotter	**IceyLeagons:** Icicle keretrendszer
12:20–12:40	**ŰrKukák:** Kockagyűjtő űrkuka	**Bole Team:** Bole
12:40–13:00	**M^5:** masKey	**Peaceful Nature Team**: Peaceful Nature
13:00–14:00	**Ebédszünet**	
13:30–14:00	**Projektbemutató előkészületek**	
14:00–15:00	**Projektbemutatások**	**Projektbemutatások**
16:00–17:00	**Eredményhirdetés**	

Tartalomjegyzék

Előszó

A Szegedi Innovatív Informatika Verseny 2022-ben nyolcadik alkalommal kerül megrendezésre, ahol a középiskolás diákok mutathatják be megvalósított innovatív ötleteiket. A versenyt a Szegedi Tudományegyetem Természettudományi és Informatikai Karán belül az Informatikai Intézet indította útjára.

A verseny keretein belül ezúttal sem kötöttük meg a diákok kezét és kreativitását, bármilyen, őket foglalkoztató informatikai témájú pályamunkával nevezhettek az alábbi két szekció valamelyikére:

A ***Műszaki informatika*** szekcióba olyan pályamunkák beérkezését vártuk, melyeknél csupán egyetlen kikötésünk volt: a feladat ne csak programozási feladatból álljon. A diákoknak meg kellett építeni, vagy már meglévő elemekből össze kellett állítani egy rendszert, amely működtetéséhez szükséges szoftvert is a nekik kellett elkészíteni.

Az ***Informatika*** szekcióba olyan pályamunkákat vártunk, amelyekben a diákok egy elkészített szoftvert mutatnak be. A fejlesztéshez tetszőleges programozási nyelv és programozói függvénykönyvtár használható volt. A szoftver kategóriáját illetően nem tettünk megkötést, lehetett játék, asztali számítógépen futtatható alkalmazás, webes alkalmazás, mobil applikáció, vagy Kinect-es alkalmazás.

Az idei versenyre összesen 30 pályamunka érkezett be. Versenyünk kétfordulós, az első fordulóba beérkezett pályamunkából a zsűri szekciónként a legjobb 10-10 pályamunkát benyújtó csapatoknak ad lehetőséget, hogy a döntőben bemutassák elért eredményeiket. Remélhetőleg idén ismét élőben találkozhatunk a diákokkal és ismerhetjük meg alkotásaikat.

A rendezvényt idén is számos neves informatikai és műszaki cég támogatta, amelyért ezúton is hálásak vagyunk. A zsűri által legjobbnak ítélt pályamunkákat értékes és hasznos díjakkal jutalmazzuk. A verseny szervezői számára kiemelten fontos, hogy ne csupán a csapatok tagjait és felkészítő tanáraikat, de iskoláit is díjazzuk, hogy ily módon támogassuk tehetséggondozó tevékenységüket.

Szeged, 2022. április

Dr. Németh Gábor
a SZIIV verseny szervezője

Programbizottság

Műszaki informatika szekció

A zsűri elnöke:

Dr. Pletl Szilveszter
főiskolai tanár
SZTE Műszaki Informatika Tanszék

A zsűri tagjai:

Dr. Mingesz Róbert
adjunktus
SZTE Műszaki Informatika Tanszék

Dr. Kincses Zoltán
adjunktus
SZTE Műszaki Informatika Tanszék

Antal András
szoftverfejlesztő
BITOTRON Kft.

Trauer János
középiskolai tanár (fizika-informatika szak)
Makói József Attila Gimnázium, Makó

Informatika szekció

A zsűri elnöke:

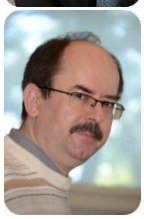
Dr. Nyúl László
intézetvezető, tanszékvezető egyetemi docens
SZTE Képfeldolgozás és Számítógépes Grafika Tanszék

A zsűri tagjai:

Dr. Vinkó Tamás
egyetemi docens
SZTE Számítógépes Optimalizálás Tanszék

Dr. Pflanzner Tamás
adjunktus
SZTE Szoftverfejlesztés Tanszék

Téglásy Nóra
szoftverfejlesztő
ExxonMobil Hungary Kft.



Fodor Zsolt
középiskolai tanár (matematika-fizika-informatika szak)
Baranya Megyei SZC Komlói Technikum, Komló

Szervezők

Dr. Nagy Antal
egyetemi docens, intézetvezető helyettes
SZTE Képfeldolgozás és Számítógépes Grafika Tanszék

Dr. Németh Gábor
adjunktus
SZTE Képfeldolgozás és Számítógépes Grafika Tanszék

Dr. Kincses Zoltán
adjunktus
SZTE Műszaki Informatika Tanszék

Tóth-Ságiné Kun Eszter
ügyvivő szakértő
SZTE Informatikai Intézet

Műszaki informatika szekció

Multikar 2.0

Kandó Racing Team

Giricz Dávid, Csollár Ferenc

Felkészítő tanár: Ladányi Sándor

Kecskeméti Kandó Kálmán Technikum, 6000 Kecskemét, Bethlen krt. 63.

1. Bevezetés

Az iskolánkban elektronikai szakon vagyunk. Ennek ellenére a szoftver fejlesztése is igen nagy hangsúlyt kapott. Úgy gondoljuk, projektünk egy olyan innováció, mely nem csak egy unalmas fizika órát dobna fel, hanem felkeltené olyan emberek érdeklődését is, akik eddig még nem ismerhették meg a robotika ezen fantasztikus világát. A cél elsősorban, hogy hobbielektronikai piacon építőkészletként tudjon helytállni és közben oktatási célokra is felhasználható legyen. Fejlesztéseink közel másfél éve tartanak. Az 1.0-ás verzióval már kipróbáltuk magunkat ezen a versenyen. A tavalyi év tapasztalatai alapján határoztunk úgy, hogy érdemes újra jelentkeznünk és megmutatnunk hová fejlődött projektünk.

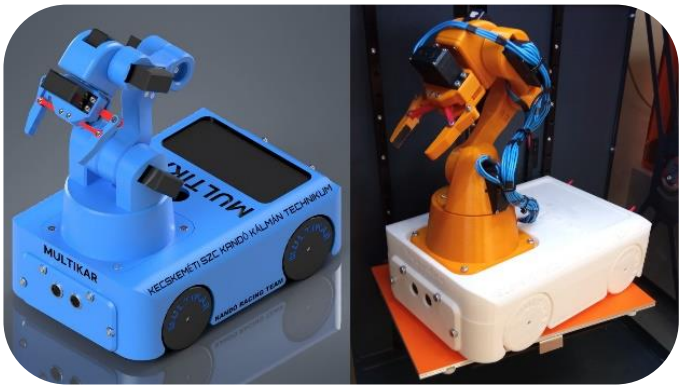
2. Probléma megoldásának menete

A koncepciónk egy olyan robot, amely motorok és szenzorok segítségével képes kisebb félautomata folyamatok végrehajtására. A szerkezetét 3D nyomtatott, saját tervezésű elemekkel kiviteleztük. Ezekbe csatlakoznak a motorok, vezérlések és a szenzorok. Az irányítás WiFi-s kommunikációval lehetséges, melyet a robot egyik mikrovezérlője biztosít. Az irányítás és a megfigyelés, a saját magunk által fejlesztett weboldalakon történik. Ezek az oldalak akkor aktívak, amikor a robot bekapcsolt állapotban van. A kezelő felületen tudjuk irányítani robotkarunkat, illetve annak előre meghatározott mozdulatsorait. Az autó 8 irányba képes helyváltoztatásra, illetve a motorok sebessége is változtatható. A szenzorokat a vonalkövetéshez és az ütközés elkerüléséhez használtuk fel. A folyamatok ellenőrizhetősége érdekében csináltunk naplózást is, ahol láthatjuk a szenzorok jelenlegi értékeit sok más információval együtt. Ezekkel a tulajdonságokkal szeretnénk egy több funkciós robotot biztosítani a megcélzott felhasználók számára. A projektet teljes egészében ketten kivitelezzük, azonban az anyagbeszerzéshez kaptunk segítséget.

2.1. Mechanika

Az alkatrészeket az Autodesk Inventor Professional 2022-es programjával terveztük. Miután elkészültek az elemek, a CraftWare és a Cura programokkal tettük gyárthatóvá. A nyomtatáshoz műanyag fillamentet (PLA) használtunk. A robotban felhasznált motorok belső fém áttétellel rendelkeznek, így nem kell

aggódnunk rövid élettartamuk miatt. Az 1. ábrán a 3D tervező programban készült és mellette a megvalósított modell látható.



1. ábra: Multikar 2.0 robot

2.2. **Elektronika**

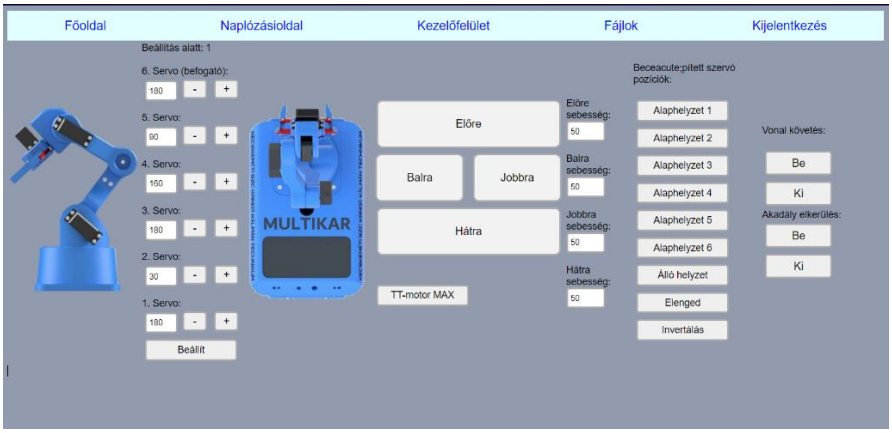
A robot belső elektronikáját négy panel alkotja. Amelyben a főpanel vezérlési és feldolgozói feladatokat végez. Az infravörös fényvisszaverődéses technikát alkalmazó panel a vonal követesért felel. A táp panel a megfelelő védelmi, töltési és áramellátási célokat szolgálja. A mérési panel a segédenergia és a beavatkozó szervek állapotát továbbítja a mikrovezérlőknek. A tápellátásról tíz darab lítium-ion akkumulátor gondoskodik. A 2. ábrán a 2022 februárjában alkalmazott paneljeink láthatóak.

2. ábra: A panelek

2.3. **Szoftver**

Az itt felmerülő feladatokat három mikrovezérlő látja el. Ezek fel programozása az Arduino IDE programjában történtek. A vezérlőegység az előre megadottak közül a legmegfelelőbb WiFi hálózathoz csatlakozik és a beállított porton létrehoz egy webszervert. A weboldalakat alapszintű biztonsági rendszerrel láttuk el. A robot forrás kódja C++ nyelven íródott és

HTML oldalakat is tartalmaz. A weboldalak asztali és mobil eszközökön egyaránt használhatóak. A 3. ábrán a kezelőfelület látható, ahol irányítani lehet a funkciókat. A következő képen a naplózási weboldal látható, ahol a szenzorok jellenlegi állapotát, illetve a kezelő felületen történő változtatásokat figyelhetjük meg.



3. ábra: kezelő felület

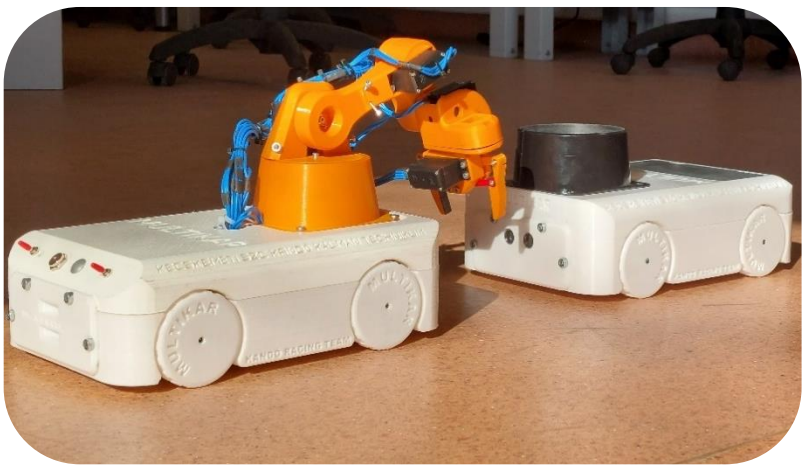
4. ábra: naplózási oldal

3. Elért eredmények

Célunk, hogy ezen területek valamelyikében innovációt és megfelelő konstrukciót mutathassunk be. Nagyban változott eredményeink összehasonlítása az 1. táblázatban látható. Versenyen két modell is előadásunk része lesz, hogy bemutathassuk robotunk gyárthatóságát. Az 5. ábrán 2022 februári helyzete látható ahol a bal oldali működő képes, a jobb oldali pedig gyártás alatt van.

Szempontok	1.0 robot	2.0 robot
Akkumulátorok kapacitása	55,5Wh	92,5Wh
Töltési teljesítmény	2x5W	33,6W
Kar össz tömege	1,2kg	0,74kg
Kar terhelhetősége	~20g	~300g
Kar hossza	412mm	260-310mm
Saját panelek aránya az összeshez viszonyítva	7%	58%
Szenzorok száma	3	12
Előreprogramozott mozdulat sorok	2	9
Akkumulátorok cserélhetősége	Nincs	Van
Hardveres, szoftveres védelem		
Alvó üzemmód		
Visszanézhető tevékenység napló		
Töltöttség visszajelzés		

1. táblázat: összehasonlítás



5. ábra

Okos, fűthető kutyatál

OFK

Nyolczas Áron

Felkészítő tanár: Rózsa Tamás

Verseghy Ferenc Gimnázium, 5000 Szolnok, Tisza park 1.

1. Bevezetés

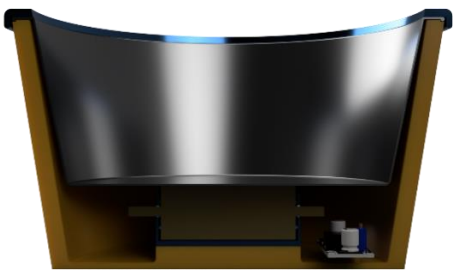
Múlt évben lett egy kutyánk, és télen felmerült az a probléma, hogy amikor odakinn fagy volt, a kutyánk vize befagyott, ezért egy nap többször is ki kellett cserélni a vizét. Ennek a problémának a megoldása érdekében megnéztem, hogy milyen fűtött kutyatálakat lehet kapni, de egyik sem tűnt megfelelőnek: védtelen kábele volt vagy túl kicsi volt és a legtöbb egyenesen a falból jövő hálózati feszültségről üzemelt, amit mondani sem kell, hogy nem biztonságos, mert megrághatja a kutya. Ezért úgy döntöttem, hogy készítek egyet.

2. Probléma megoldásának menete

A célom az volt, hogy egy relatív kis feszültséggel működő – ez esetben 12 Volt – rágástól minél jobban védett, állítható hőmérsékletű és Google Assistant-en keresztül vezérelhető, körülbelül 2 literes okos itatótálat hozzak létre.

2.1. A tál felépítése

A tál két fő részből áll: egy sima 1,8 literes acél tál és egy 3D nyomtatott „vödör”, amiben az elektronika is helyet kapott. Az elkészített munkadarab műanyag részeit magam terveztem és készítettem el 3D-s nyomtatómmal. A 150 wattos IP67-es tápegység a projekt áramellátását biztosítja. A tálat és a tápegységet 1,5mm^2-es vezetékek kötik össze. A vezetékek egy fém gégecsőben vannak elhelyezve, ez rágás elleni védelmet is biztosít és hajlékonyabb, mint a sima műanyag gégecső.



1. ábra: A tál keresztmetszete a CAD programból

2.2. Biztonság és víz elleni védekezés

Az itatótál élőlény részére, szabadtéri használatra készült. Úgy kellett mindent megterveznem, hogy a lehető legbiztonságosabb és a legvízállóbb legyen.

Ennek érdekében egy hőbiztosítékot helyeztem el a tálon, ami 77 C° elérése esetén leolvad, ezáltal az egész tálnak az áramellátását megszakítja. Viszont ez a legvégső védelem abban az esetben, ha a mikrokontroller vagy a hőérzékelők meghibásodnának. Víz ellen többfajta védelmet is használtam: TPU-ból 3D nyomtatott gumigyűrűt helyeztem a fém tál és a műanyag rész közé, amit 15 csavar tart összenyomva, IP67-es tápegység, tömszelence használata és végsősorban a műanyag részek és a NYÁK lapok bevonása akril lakkal.

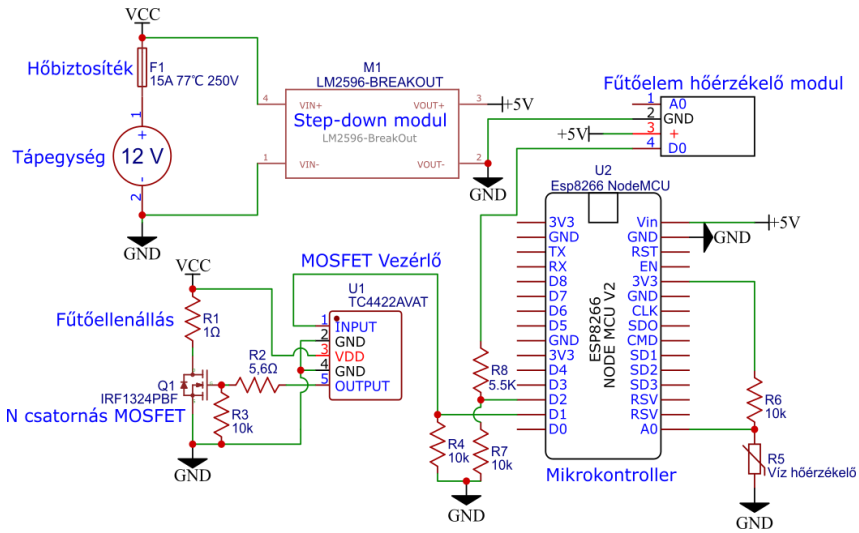
2. ábra: A kész tál tápegységével

2.3. **Az elektronika**

Eredetileg a tál tudott volna fűteni és hűteni egy Peltier-elem segítségével, de ezt tesztelése után nem találtam jó megoldásnak, mivel aktív hűtés kell az elemnek, ha hűteni akarunk vele, ezt pedig nehéz lett volna kivitelezni úgy, hogy a tál vízhatlan maradjon. Ezért egy 1Ω, 100Wattos huzalellenállást használtam fűtésnek.

Ahhoz, hogy teljesen kihasználjam a fűtőelememet, nagyobb áramerősség szükséges 12 Volt esetén, mint 230 Voltnál. Ezért olyan eszközt kellett használni a fűtőelem ki-bekapcsolására, ami ezt a terhelést elbírja. Először egy relében gondolkoztam, de megbízhatóbbnak gondoltam egy tranzisztort. Ezért egy régi tápegységből származó IGBT-vel próbálkoztam, de ez még úgy is túlságosan melegedett, hogy hővezető ragasztóval volt ráragasztva a fém tálra. Azért, mert túl nagy volt bekapcsolt állapotban az ellenállása 10 Amperen majdnem 10 Wattnyi hőt képzett, emiatt lecseréltem egy kis ellenállású MOSFET-re, ami pedig annyira jó választás volt, hogy még hűtőborda sem kellett rá a disszipálódó hő elvezetéséhez. Körülbelül 0,15 Wattnyi energia a hőveszteség, így nem csak hűvösebbé tette, hanem jobb lett a hatásfoka is a rendszernek.

Mikrokontrollernek egy Esp8266-os NodeMCU-t használtam, mivel elérhető és olcsó volt. Egy gond viszont akadt vele: csak egy darab analóg bemenete van, közvetlenül csak egy darab hőérzékelőt lehet hozzácsatolni, ezt a problémát úgy oldottam meg, hogy egy külön modulon lévő hőérzékelőt használtam, aminek a lényege az volt, hogyha a hőmérséklet elér egy potenciométerrel beállítható értéket, akkor a kimenete magas lesz, egyébként pedig alacsony. Ez lehetővé tette, hogy egy digitális bemenettel használjam a hőérzékelőt. Ezzel még egy probléma merült fel. A modul csak 5 Volton működik, miközben az Esp8266 3,3 volton, de ezt két ellenállással meg lehetett oldani (R8, R7). Mivel a mikrokontroller 3,3 Volton dolgozik, ezért nem lett volna képes bekapcsolni a MOSFET-et, ezért egy vezérlőt (U1) használtam, hogy 12 Volton kapcsolhassam a tranzisztorom. Az 5 Voltot egy olyan step-down modul biztosítja, aminek közös negatív pólusa van a bemenetén és kimenetén.



3. ábra: Az áramkör

2.4. A program és a tál működése

Bekapcsoláskor a mikrokontroller felcsatlakozik a kódjában megadott Wifi hálózatra és szinkronizálja magát az AdafruitIO-n található feedek értékeivel. Összesen 4 darab feedből kettő bemenet, itt mi adjuk meg az értékét, kettő kimenet, ami visszajelzést ad az eszközről. A feedeknek az a feladata, hogy a fűtést ki-bekapcsolja, a kívánt hőmérséklet beállítsa (0-25C° tartományban állítható), valamint a jelenlegi hőmérséklet és fűtőtest állapotának jelzése (kap-e áramot vagy nem). A mikrokontroller méri a víz és a fűtőelem hőmérsékletét. A víz mérése esetén megnézi a hőérzékelőnek az ellenállását, majd a Steinhart–Hart (1) egyenlettel kiszámítja a hőmérsékletét.

Ennek lehetnek kilengései, ezért 10 db mérés értékét átlagolja. A fűtőelem hőmérsékletét egy külön áramkör méri, ha onnan magas jelet kap, akkor a fűtőelem átlépte a megengedett hőmérséklethatárt. Majd eldönti, hogy bekapcsolja-e a fűtőtestet 4 db kritérium alapján: ha a víz hőmérséklete kisebb, mint a megadott hőmérséklet és a maximum hőmérséklet (25C°), ha a fűtőelem a megadott maximum hőmérséklet alatt van és ha a fűtés be van kapcsolva, akkor a fűtőelem kap áramot, ha ebből csak egy is hamis, akkor nem kap áramot. A jelenlegi hőmérséklet értékét és a fűtőtest állapotát ezután feltölti a feedekbe. Ez minden 30-adik esetben történik meg, hogy ne érje el a feltöltési korlátot. Ezután a feedek olvasásától a folyamat elölről újra elkezdődik és addig tart, amíg le nem állítjuk. A feedek értékeit pedig nem csak az Adafruit weboldaláról tudjuk frissíteni, hanem IFTTT segítségével a Google Assistant-en keresztül akár a hangunkkal is meg tudjuk őket adni. Sajnos ezt használva mi nem tudunk értékeket lekérdezni az asszisztensen keresztül, de cserében ez a megoldás egész könnyen hordozható, nem kell még egy külön kisszámítógépet (Pl: Raspberry Pi) használni hozzá, hogy kezelje az asszisztenst. Így viszont visszajelzésként csak annyit tudunk kapni, hogy elérte-e a kívánt hőmérsékletet a tál.

$$\frac{1}{T} = A + B * ln(R) + C * ln(R)^3 \qquad (1)$$

3. Elért eredmények

Tavaly az első fagyok beköszöntével merült fel a kutyánk vize hőmérsékletének fagypont felett tartásának igénye. Esztétikus, érintésvédelmileg biztonságos és okos-fűthető kutyatálat sikerült elkészítenem. Az első munkadarab működésének praktikus megvalósítása több hónapig tartott. Széleskörű használata lehetséges, a mintadarab elkészülte után „házi megvalósításban” 1-2 nap alatt elkészíthető.

IAQ BOX

BOT

Nagy Benedek

Felkészítő tanár: Harangozó Zsolt

BMSZC Bláthy Ottó Titusz Informatikai Technikum, 1032 Budapest, Bécsi út 134.

1. Bevezetés

A BMSZC Bláthy Ottó Titusz Informatikai Technikum diákja vagyok, és tanulmányaimhoz kapcsolódóan, 2 éve jelentős érdeklődést ébresztettek bennem a beágyazott rendszerek, azon belül is az alacsony energiafogyasztású IoT rendszerek. Belépő szintű projektjeim után, mint például egy teljesen önellátó kültéri hőmérő, amelynek adatait ezen projektem is használja, illetve egy ultratakarékos talajnedvesség-mérő, úgy döntöttem, hogy tapasztalataimat felhasználva belekezdek eddigi legnagyobb projektembe.

Még ma is, a fogyasztói piacon körbenézve, a legtöbb levegőminőség-mérő eszköz LCD vagy LED kijelzőt használ, mindezt meglehetősen magas áron. A LED kijelzők esetében ez esztétikus megjelenést biztosít, viszont cserébe az eszköz csak interakció hatására, vagy hálózati tápellátással, vagy rövid üzemidővel képes működni. Erre terveztem egy megoldást a projektemmel.

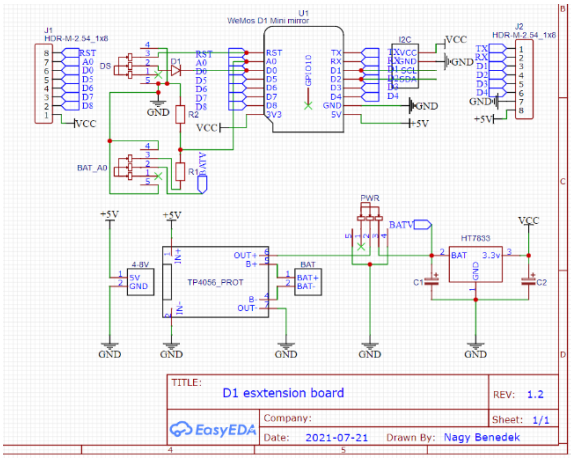
2. Probléma megoldásának menete

A projektem célját még 2021 nyarán találtam ki. Egy, a piacon akkor még nem elérhető, ultratakarékos levegőminőség-mérőt kezdtem el tervezni. Legfőbb elemeként E-Ink technológiával, amely áramellátás nélkül is képes kijelezni a ráírt adatokat, nem igényel háttérvilágítást, és erős környezeti fények esetén is probléma nélkül olvasható. További előnye az én megoldásomnak az ESP-NOW alapú, energiatakarékos felhő szinkronizáció, amely márkafüggetlen integrációt biztosít okosotthon rendszerekbe.

2.1. Tervezés

Az alkatrészek kiválasztása után az egyszerűbb összeszerelés és a megbízhatóbb csatlakozások érdekében megterveztem (1. ábra) és megrendeltem az egyedi PCB-met az alkatrészekkel együtt.

Az alkatrészek megérkezése után megkezdődhetett a projekt házának a 3D modellezése, amelynél egy mágneses rögzítőrendszert terveztem a kijelzőhöz és a napelemhez is, utóbbi esetében a mágnesek biztosítják az elektromos csatlakozást is a könnyű szerelhetőség érdekében. Ezenfelül külön figyelmet fordítottam a projekt esztétikus megjelenésére és az akkupakk izolációjára.



1. ábra: A PCB áramköri rajza

2.2. 3D nyomtatás

Mivel beltéri használatra szánt eszköznek terveztem, lehetőségem nyílt dekor műanyagok használatára, így farészecskéket tartalmazó és fehér PLA filament mellett döntöttem. A fa filamentnek és pár szeletelési beállításnak köszönhetően a doboznak fa illata és érdes textúrája lett.

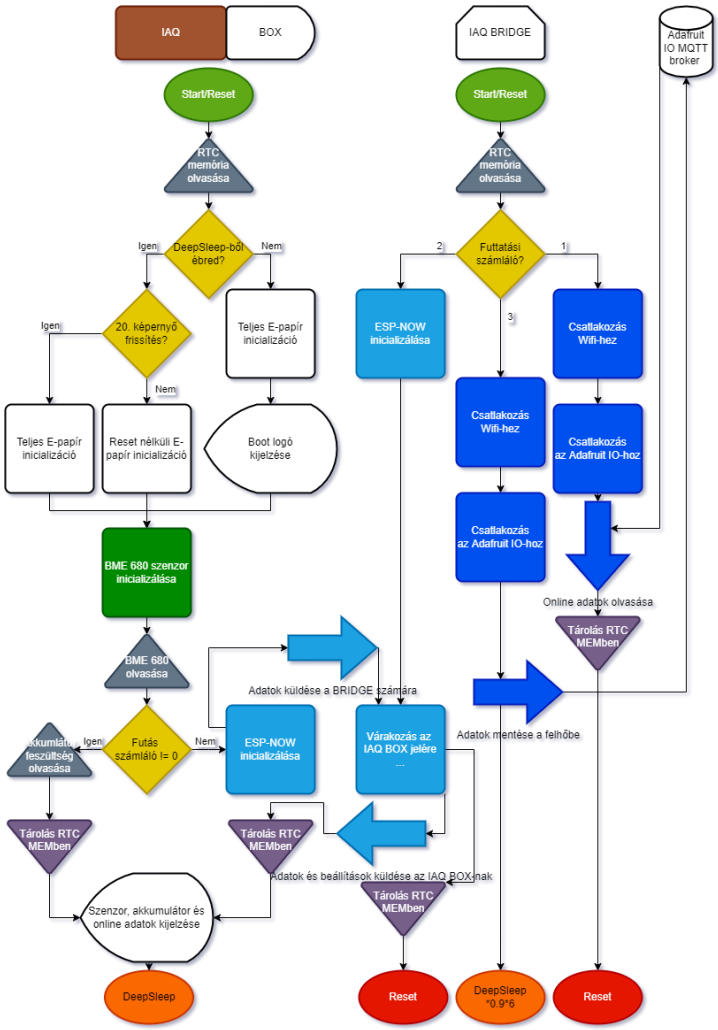
2.3. Összeszerelés

A főbb alkatrészek és indoklás a kiválasztásukra:

- 4x 18650 3400mah li-ion akku „biztonságos túlnyomáskezelés”.
- 5V 250mah napelem „maximum reális méret”.
- PCB „egyszerűbb, biztonságosabb csatlakozások, kompaktság”.
    - D1 mini v4.0 ESP8266 wifi modul és csatlakozó shield „gazdaságos, és megoldja az SMD csatlakozók forrasztását”.
    - HT7833 lineáris feszültségszabályozó „minimális passzív fogyasztás és alacsony drop-out feszültség”.
    - TP4056 lítium töltésvezérlő és DW01A akkuvédő chip, „elengedhetetlen a nagykapacitású akkupakk miatt”.
- BME 680, hőmérséklet-, páratartalom-, légnyomás- és gázszenzor.
- 2.9inch E-Ink kijelző „takarékos, szép, modern”.

A csatlakozó shield 10 tűs SH1.0 csatlakozót használ, amelynek különbözik a kiosztása az E-Papír 8 tűs PH2.0 csatlakozójától, így azt megfelelő krimpelő fogó hiányában kész kábelek keresztbe forrasztásával oldottam meg. A PCB-n szükség volt egy további felhúzó ellenállás bekötésére, amely szerencsére könnyű módosítás volt az extra csatlakozásoknak köszönhetően.

2.4. **Programozás**



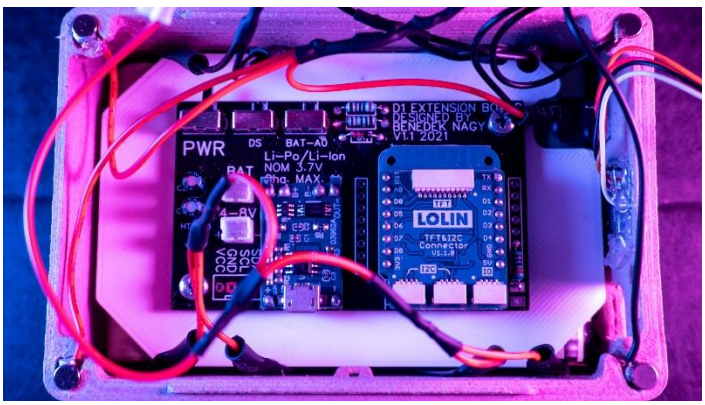
2. ábra: A programkód folyamatábrája

A kód (2. ábra) lényege az energiatakarékosság és az annak megvalósításához szükséges deepsleep. Az IAQ BOX csak minden 6. futás alkalmával szinkronizál az IAQ BRIDGE-el, amely a kapcsolatot biztosítja az internethez, így többek között az ilyenkor letöltött kültéri adatokat is el kell tárolni a processzor RAM-ján kívül, mivel az törlődik minden egyes deepsleep és reset alkalmával. A modulban található RTC modul memóriája nem kapcsol ki, így tökéletes megoldást nyújt a problémára.

3. Elért eredmények

A pályamunka (3. és 4. ábra) elért eredményei:

- egyedileg tervezett PCB és 3D nyomtatott ház,
- napelemes töltés, kellő fény mellett miközben az eszköz alszik,
- energiatakarékos áramellátás a lineáris feszültség szabályzó által,
- takarékos üzemelés deepsleep-el és automatikus ébresztéssel,
- adatok mentése és olvasása RTC memóriából,
- állandó adatmegjelenítés E-Ink kijelzőnek köszönhetően,
- energiatakarékos felhő szinkronizáció és okosotthon-integráció ESP-NOW és egy Gateway-ként funkcionáló másik ESP8266 által,
- könnyű vezérlés és adatvizualizáció Adafruit IO használatával,
- gazdaságosan elkészíthető és márkafüggetlen.



3. ábra: Az elkészült elektronika

4. ábra: Az elkészült pályamunka

Xnano és Xos

EMRLD

Bálint Olivér
Felkészítő tanár: Varga Zsuzsanna
Budapesti Osztrák Iskola, 1126 Budapest, Orbánhegyi út 39-45

1. Bevezetés

2021-ben elkezdtem építeni saját számítógép márkámat, az EMRLD-ot, mivel hiányt látok az olyan cégekben, akik biztonságos Open Source alapú rendszereket és gépeket gyártanának, amik az átlag felhasználóknak és professzionálisnak is egyszerű használatot biztosít. Léteznek már olyan cégek, amik Open Source gépeket és operációs rendszereket gyártanak, de nincs kohézió a hardware és a software dizájn közt, túl fragmentáltak a programok, és csak egy nagyon szűk felhasználói kört céloznak meg. Ezzel szemben az EMRLD nagy hangsúlyt fektet a dizájnba és törekedik arra, hogy az általunk fejlesztett operációs rendszer, az Xos, az átlag felhasználónak is egy használható opció legyen.

2. Problémák és azoknak megoldásainak a menete

A célom az EMRLD-al az, hogy nyitott rendszereket építsek, amelyek egyszerűen használhatóak, esztétikusak és bármely "ökoszisztémába" beillenek. Nagy hangsúlyt fektettek a kompatibilitásra és az újrahasznosításra. Az általam látott problémák a mostani rendszerekben az a bezártság, fejleszthetetlenség, környezetszennyezés és a biztonság hiánya. A rendszerek mint a Windows vagy a Mac csak saját appokat enged meg futtatni, a fejlesztőknek portolniuk kell a programokat, a felhasználónak meg fejtörést okoz, ha egy általa használt applikáció nem érhető el a rendszeren. Az Xos képes Linux, Windows, web és Android appok futtatására, ami hála egy saját fejlesztésű applikációnak egy gombnyomásra letölthető és már használható is. Felhasználók a legnagyobb app-választék közül válogathatnak, fejlesztőknek pedig nem kell még egy platformra fejleszteni, mert kompatibilis a többivel. Modern számítógépek a bezártság mellett nem fejleszthetőek. Nem lehet vásárlás után több memóriát vagy akár háttértárat sem hozzáadni később. Ha az ember 1 év után rájön, hogy kellett volna a nagyobb ram opció, akkor se tud mit tenni, vagy elviseli, hogy annyi van amennyit vett, vagy vehet egy másik gépet. A háttértár pedig már felháborító. Hogy ha valakinek nem elég a 256GB egy idő után, és szeretne többet az hurcolhat magával egy külső meghajtót. Szerintem ez így nincs rendben, éppen ezért az EMRLD gépekben, ahogy az Xnanoban is bővíthető a memória, a háttértár és cserélhető a vezeték nélküli adapter is. Ezzel az élettartamuk az Xnanonak például akár több mint 10 évre is kitolható. És mivel számomra a környezetvédelem és az újrahasznosítás nagyon fontos, ezért szempont volt a tervezésnél ez is. Ha vesszük az átlag

felhasználót, aki 4 évente vesz egy új gépet, akkor 4 évente igazából mondhatni kidob egy működő és használható számítógépet. Tegyük fel, hogy eladja, és valaki még 3 évet kicsikar belőle, de már a bővíthetelenség miatt nem használható, ezért kidobja. Tehát 7 év egy computer élettartama. De mi történik ha mondjuk egy Xnano-t vásárol valaki? Bővíthetősége miatt legalább 6 évig tudja használni első gazdája, és ezek után visszahozhatja az EMRLD-hoz, ha már valóban kell neki egy újabb gép. Itt a moduláris designnak hála újrahasznosítjuk a gép nagy részét, és az alaplapot például egy kioszk-rendszerbe felhasználjuk, ahol feltehetőleg legalább még 5-6 évet lehet használni. Ezzel a módszerrel kevesebb gépet kell gyártani és ritka nyersanyagokat felélni, mivel a gépek akár másfél évtizedig körforgásban maradhatnak. A környezetvédelem és nyitottság mellett egy fontos ismertetőjele az EMRLD által készített programoknak és számítógépeknek az a biztonság. Számomra nagyon fontos, hogy ha írok egy programot, az tartsa tiszteletben a felhasználók magánéletét, és ne lehessen kiskapu támadások esetén. Az Xos Linux alapjainak köszönhetően pontosan ezt garantálja. Vírusoktól mentes biztonságos és szabad platformot, ahol az emberek aggódás nélkül alkothatnak határtalanul. A Windows operációs rendszerek egyszerűen lehetnek vírusosak, és sok fejtörést tud ez okozni. A Mac, amiről az a tévhit terjedt el, hogy nem érheti támadás rácáfol erre, és egyre veszélyesebb platformmá válik. Mivel Európában láthatóan nő az igény a Linux szerű, Open Source operációs rendszerek használatára, ezért nem kérdés, hogy egy, az eddigieknél jobban használható ilyen rendszer, mint az Xos, ami még kompatibilis több rendszerrel is, szélesebb körben felhasználókra fog találni. Amiben nagyban eltér az EMRLD más Open Source-al foglalkozó cégektől az a fragmentáció és az irányelv. Sok nyílt forráskódú programnak több mint száz féle verziója található meg az interneten, mert aki egy új funkciót akart belerakni, ahelyett hogy az eredeti program fejlesztőivel konzultált volna, csak csinált egy külön ágat a programból. Van olyan eset, amikor ez jó is lehet, de álltalában ez átláthatatlanná és nehezen fejleszthetővé teszi a programokat. Az EMRLD ezzel szemben amellett, hogy Open Source programokat ad ki, azt fogja támogatni, hogy egy jól működő verziót tartsanak fönt, és ne 10 elfogadhatót. Ezzel koncentrálni lehet az Open Source fejlesztők erejét, és hatékonyan lehet fejleszteni. Még egy ismertetője a nyílt forrású programoknak az az univerzalitás. Mindent lefedő programokat akarnak fejleszteni, mert mindenkinek más az igénye, és ez azt eredményezi, hogy egy mindenre csak elfogadható megoldást kínálnak. Az én elképzelésem szerint, kell egy erős irányelv az Open Source fejlesztések mögé, amivel irányítani lehet, hogy az adott program abban a dologban, amire használni fogják, kiváló legyen és ne az összesben csupán tűrhető.

3. Elért eredmények

Programozásban már volt korábbi rutinom, amikor nekiálltam Xnano tervezésének, de CAD szoftverekkel még nem dolgoztam, és keveset tudtam a gyártási procedúrákról is. Sok tanulás és álmatlan éjszaka után viszont elkészítettem egy olyan designt, ami egyszerűen és takarékosan gyártható, mindezek mellett esztétikus, és beleillik a legtöbb helyiségbe. A tervezésbe relatív gyorsan belejöttem, de a gyártásokról való tudásom hiánya miatt több nehézség előtt álltam. Első iterációjában még forgácsolással kialakított házat terveztem az Xnanonak, de ez nem valami környezetbarát megoldás, és ehhez a géphez nem is illő. Sok gondolkodás után megálmodtam, hogy hogyan tudom tükrözni a projekt üzenetét a külsőben. Ekkor jött létre az átlátszó akril lapokkal közbezárt külseje az Xnanonak ami az 1.a) ábrán látható.

a) Xnano gépház alaplap nélkül

b) fémöv

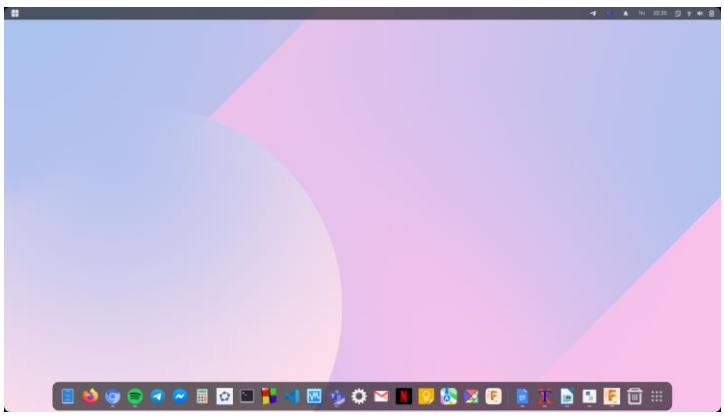
1. ábra: Xnano gépház alaplap nélkül és a legleső fémöv, ami elkészült

A dizájn tükrözi az átláthatóságot és letisztultságot, ezek mellett a gépházat nagyon hatékony és környezetbarát módszerekkel lehet gyártani, és mivel a 3D-fájlok elérhetőek ingyenesen a weboldalamon, ezért bárki akár le tud egy ilyet gyártani. Ha valaki 3D nyomtatókkal szeret dolgozni, az tud akár nyomtatni saját magának egyedi gépházakat. A dizájn 3 fő elemből áll: kettő fedlap ami átlátszó víztiszta akrilból van, és egy sík rozsdamentes acél lapból összehajtott fém középrészből. 8 távtartó és 4 csavar segítségével az alaplap a két akrillap közé fogható, és egy strapabíró, robosztus gépet kapunk. Egy nagyon erős és érzelemmel teli pont a megvalósításban, az az volt, amikor átvettem az elkészült fém középrészeket (lásd a 1.b) ábrán). Ekkor realizáltam, hogy az utóbbi 5 hónap sok munkájának volt értelme. Ekkor éreztem igazán azt, hogy készítettem valamit, én egyedül. Kézben tartani és összerakni az első Xnanot egy hatalmas élmény volt. Folyton az járt a fejemben, hogy mindezt egyes egyedül csináltam, és eljutottam egészen idáig, hogy a kezemben tarthatom.

Mióta a 1.b) ábrán látható fotó készült összeraktam a teljes gépet, és gőzerővel dolgozok az operációs rendszer, az Xos-en. Minden nap az Xos-t

használom, és így aktívan tudom fejleszteni és javítani. Érzem, ha valami nem jó, vagy még törődést igényel. Jelen pillanatban ott tartok, hogy mindössze 5 másodperc alatt boot-ol be, laptopokon, amik Windows 10-el 3,5 óra alatt lemerültek az Xos 6-7 órás üzemidőt biztosít. Web Alkalmazásokat, amit általában böngészőn keresztül érünk el, mint a Gmail vagy a Google Docs natív alkalmazásként tudom használni, és emellett Windows applikációkat is használok nap mint nap, például a Fusion 360-at, amiben tervezem a jövőbeli EMRLD gépeket. Jelen állás szerint az Xos egy 16GB memóriával ellátott Xnano-n 50 program futása, egy Blender renderelés és 400 böngészőablak nyitvatartása közben még képes egy virtuális gépet futtatni bármiféle lassulás nélkül. Ez már az a szint, amit a célközönség, vagyis professzionálisok és poweruserek elvárnak. Összefoglalva eddigi eredményeim: Megtervezett és legyártott, sorozatgyártásra is optimalizált Xnano gépeknek a 15 évre elkészített újrahasznosítási és újrafelhasználási tervezete; Xos operációs rendszer: Kezelői felület kinézete és ergonómiája elkészítése; teljesítmény és hatékonyság optimalizálása; kompatibilitás számos Windows programmal és 850 millió web-alkalmazással; a projekt weboldalának a fejlesztése.

Az Xos fejlesztése során mindenképpen szempont volt, hogy esztétikusan kielégítő legyen, és kezelése egyszerű és intuitív legyen. Sokan azt gondolják, hogy az Xos célközönsége (professzionálisok, prosumerek) nem igénylik az egyszerű felhasználói felületet. Ezzel szemben az ilyen felhasználók sem akarnak bajlódni a mindennapi dolgokkal. Appok letöltése, böngészés és fájlkeresés mind olyan dolog, amiben ezek a felhasználók is preferálják az egyszerű kezelői felületet. A kezelői felület nagy része már hála a Gnome-desktopnak már készen volt, nekem a preferenciák szerint kellett átalakítanom és a fent felsorolt applikációkat/programokat megírnom. A 2. ábrán lehet látni, hogy holt tart esztétikailag az Xos.



2. ábra: Xos Desktop beta

Csőtörés-jelző rendszer

Pipeline

Székely Marcell
Felkészítő tanár: Horváth-Varga Péter
Kempelen Farkas Gimnázium, 1223 Budapest Közgazdász u. 9-11.

1. Bevezetés

Ma minden lakóházban, irodában, gyárban és egyéb intézményben jellemzően van kiépített vízhálózat. Egy váratlan csőtörés esetén akár komoly károk is keletkezhetnek az épületben, az elektromos berendezésekben vagy a berendezési tárgyakban. Rosszabb esetben egy épület hosszabb időre is lakhatatlanná válhat vagy egy irodát csak korlátozottan tudnak használni a helyreállítások ideje alatt. Jól látható tehát, hogy egy csőtörés milyen mértékű károkat okozhat – én ezen károk mérséklésére vagy kiküszöbölésére szeretnék megoldást nyújtani.

A probléma megoldására egy olyan eszközt terveztem, ami időben jelezni tudja a váratlanul megnövekedett vízfogyasztást, így ezzel minimálisra csökkentve egy esetleges csőtörés okozta károkat. Az egyik célnak azt tűztem ki, hogy ezt a rendszert bárki egyszerűen, szakember segítsége nélkül is tudja telepíteni, ehhez ne kelljen megbontania a vízvezetékrendszert. A másik célom pedig az volt, hogy egy csőtörést minden eseteben jelezzen, a téves riasztások minimalizálása mellett. Ezeken túlmenően egy olyan internetes felület elkészítését is terveztem, melyen keresztül a felhasználó könnyen, bárhonnan elérheti az eszközt, hogy távolról is elvégezhessen rajta beállításokat, vagy megtekinthesse az eszköz állapotát.

Ahogy írtam, egy egyszerűen, a vízrendszerre megbontás nélkül beszerelhető eszközt terveztem. Ehhez az kellett, hogy kívülről meg lehessen állapítani, hogy egy csőben folyik-e a víz vagy sem. Erre a célra léteznek ultrahangos áramlásmérők, viszont ezek ára igen magas, emiatt egy másik mérési módszert kellett alkalmaznom. Egy vízvezetékre erősített mikrofonnal a mért frekvenciákból nagy biztonsággal meg lehet határozni, hogy éppen áramlik-e víz, sőt még akár azt is, hogy milyen intenzíven. Azzal a további céllal is készítettem ezt a prototípust, hogy a későbbiekben felhasználható legyen más problémák megoldására is (pl. jellemző hangfrekvenciát kibocsátó eszköz vizsgálatára és hiba-előrejelzésére).

2. Probléma megoldásának menete

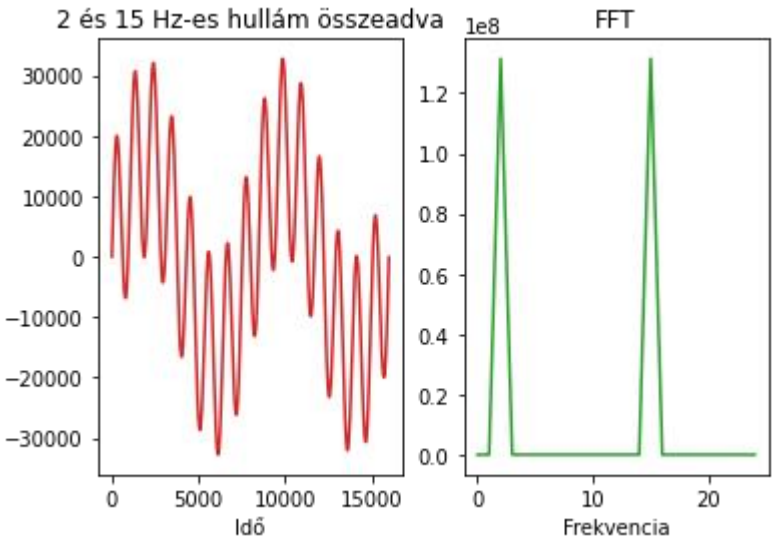
A megoldás első állomása a felhasznált eszközök kiválasztása volt. A szenzor adatainak gyűjtésére és feldolgozására egy STM32 Nucleo-144-es panelt választottam melyen egy 32 bites ARM Cortex M4-es MCU működik, amit C-ben programoztam. A 32 bites mikrokontrollerre azért volt szükségem, mert a

nagy mennyiségű hanganyag feldolgozásához és tárolásához nagy számítási és tárolási kapacitás szükséges. A hang felvételére pedig egy Adafruit digitális mikrofont alkalmaztam. Ezen kívül szükségem volt még egy wifi kiegészítőre, mert így nem kell Ethernet kábelt kivezetékelni az eszközhöz, így még egyszerűbben felszerelhetővé válik. Az eszközök összeszerelése után kezdődhetett el a programozás.

A kihelyezett eszközön felül még fejlesztenem kellett egy olyan szolgáltatást is, melyhez az eszköz képes kapcsolódni és a feldolgozott adatokat felküldeni. Ezen felül ennek a szolgáltatásnak a feladata a beérkező adatok alapján döntést hozni és riasztási eseményeket kiváltani. Szintén ez a komponens biztosítja a webes konfigurációs felületet is. Ennek a szolgáltatásnak a futtatására Amazon felhőszolgáltatást választottam (AWS), mivel ezzel tudtam a leggyorsabban megoldani a feladatot.

2.1. Adatok fogadása, feldolgozása a mikrokontrolleren

Első lépésnek a mikrofonról I2S protokollon keresztül be kellett olvasnom az adatokat. Ezek után következett a hang frekvenciájának kiszámolása, hiszen a frekvenciából tudunk következtetni a folyadék áramlására. A mikrofonból a hang az idő függvényében érkezik, ebből viszont még nem lehet tudni a hang frekvenciáit. Erre kellett alkalmaznom az FFT algoritmust. Ez egy bonyolult matematikai művelet, amivel kiszámolhatjuk a bejövő hang frekvenciáit.



1. ábra: FFT és szinusz hullám

Az 1. ábrát Pythonban példának generáltam 2 Hz-es és 15 Hz-es szinusz hullám összegéből, erre pedig elvégeztem az FFT-t. A jobb oldali

grafikon két kiugrása 2 Hz-nél és 15 Hz-nél van, így mutatja meg a két frekvenciát, amiből az első hullám áll.

Az FFT alkalmazását először nem a valós hanganyagon teszteltem, hanem generáltam szinusz hullámokat, melyekre elvégeztem az FFT-t. Amikor a generált hullámnak megfelelő frekvenciát kaptam eredményül bebizonyosodott az algoritmus helyessége.

2.2. Az adatok kiértékelése és tárolása

A FFT számítás után a frekvencia adatokat ki kell értékelni. Minden vízvezetéknek más hangja van, így először muszáj egy kalibrációt végezni az eszközön. A folyamatot a webes felületen lehet elindítani. A kalibráció után a rendszer megjegyzi, hogy melyik frekvenciatartományban kell figyelni a kiugrást, így mindig tudni fogja, hogy mikor folyik a víz az adott vezetéken. Az idősoros vízáramlási adatokat (hogy mikor folyik vagy nem folyik a víz) Amazon EC2-ben futó InfluxDB adatbázisban tárolom. Az eszköz wifi kapcsolaton keresztül jelentkezik fel a felhőbe és rendszeresen küldi a telemetriai adatokat. Ezeket az adatokat folyamatosan figyelni kell, hogy időben értesíteni tudjuk a felhasználót egy csőtörésről. Erre a célra egy Python-ban írt Lambda függvényt hoztam létre, mely aktiválását egy ütemező váltja ki.

2.3. Riasztási események kezelése

A beérkező adatokból, a beállított paraméterek alapján az alkalmazás algoritmikus megoldással döntést hoz, hogy szükség van-e riasztási esemény kiváltására. Amennyiben a felhasználót értesíteni kell, az alkalmazás ezt emailen és SMS üzeneten keresztül hajtja végre. Email és SMS küldésére külső szolgálatásokat vettem igénybe (AWS SES, Seeme SMS)

2.4. Webes felület fejlesztése

Egy Amazon CloudFront által kiszolgált webes felületet is kellett fejlesztenem, hogy a felhasználók könnyen hozzáférjenek az eszköz beállításaihoz. A felület hátterét egy Pythonban írt Rest API adja, ami szintén Lamdaként fut Amazonon belül. A felhasználó adatait és beállításait Amazon DynamoDB-ben tárolom, ahonnan ezeket a kitelepített eszköz leszinkronizálhatja magának.

A felületről lehet a hang konfigurációt elindítani, beállítani az értesítendő emailt és telefonszámot, ezen kívül a riasztásokat is testre lehet szabni.

3. Elért eredmények

A kitűzött célok szerint sikerült elérnem, hogy az eszköz minden esetben érzékelni tudja, ha egy vezetéken víz folyik keresztül. Az internetes felületen be lehet állítani riasztásokat, például időkorlátot külön napszakra vagy akár konkrét napra is, amikor, ha a beállított paraméternél hosszabb ideig folyik a víz akkor értesíti a rendszer a felhasználót. Emellett az értesítés módját is meg

lehet adni, hogy emailben vagy SMS-ben is jelezzen. A rendszer így nagy megbízhatósággal tudja a felhasználót értesíteni egy adott vízvezeték hibájáról. Ezzel a módszerrel minden nagyobb kárt ki lehet küszöbölni, ahol egy adott vízvezetékben nem folyamatosan áramlik a víz.

Az eszköz arra még nem alkalmas, hogy egy irodaház fő vízvezetékére felszereljük, hiszen nagy valószínűséggel ott mindig folyamatos folyadékáramlás a sok felhasználó miatt, így ott egy algoritmikus rendszer nem tudna jól működni. Időtúllépésnél valószínűleg mindig téves riasztások érkeznének az egyfolytában áramló víz miatt, bár egy éjszakai csőtörés érzékelésére a rendszer itt is alkalmas lenne, mivel akkor már nem tartózkodnak sokan az épületben, így nem lenne állandó vízfogyasztás. Jövőbeli tervem, hogy az algoritmikus módszert gépi tanulásra cseréljem, így az eszköz megtanulná, hogy adott időben egy vezetéken mennyire folyik a víz. Ebből anomáliákat keresve következtetne, hogy a szokásostól eltérő a vízhasználat, ilyen esetben pedig riasztást küldene.

Jelenleg az eszköz fő felhasználási területei a lakóházak vízvezetékrendszere, vagy nagyobb létesítmények különálló kisebb részei, például egy konyha vagy mosdó. Ezeken a területeken a bemutatott rendszer sikeresen alkalmazható a jelentős anyagi károkat okozó lehetséges csőtörések jelzésére.

wSnake hálózat

ThatNetworkGuy

Mikulás Péter Marcell
Felkészítő tanár: Hagymási Gyula
DSZC Mechwart András Gépipari és Informatikai Technikum,
4025 Debrecen, Széchenyi u. 58.

1. Bevezetés

A technológia fejlődésének és az IoT okoseszközök egyre nagyobb népszerűségének köszönhetően elengedhetetlenné válnak ezeket az eszközöket összekapcsoló hálózatok. Ezek az eszközök okosotthonoktól egészen az ipar különböző területein jelen vannak, és gyakran kritikus rendszerek részeit képzik.

A cél egy olyan hálózat és eszközkészlet megvalósítása, amely alkalmas ezen eszközök közötti kommunikáció biztonságos és hatékony megvalósítására, miközben megőrzi az alacsony költségű kiépítést, és skálázhatóságot.

2. Probléma megoldásának menete

2.1. Hálózat elméleti felépítése

A wSnake hálózat alapelemeit képezi egy szerver szolgáltatás, hozzáférési pontok (AP-k) és a végberendezések (Kliensek). Az eszközök közötti kommunikáció saját fejlesztésű protokollal történik, ami az OSI modell 2. és 3. rétegét ötvözi. Az adatfolyamot több, kisebb részre bontja és részletekben továbbítja.

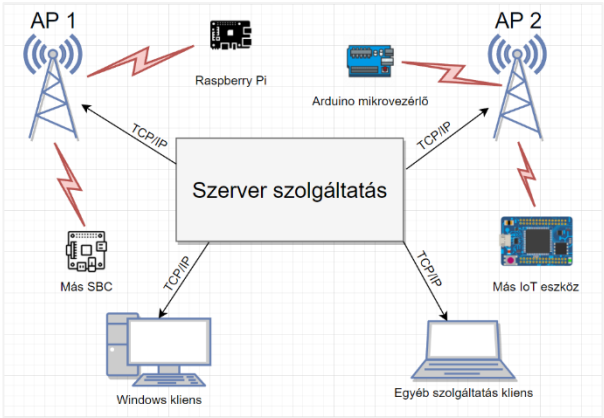
Szerver szolgáltatás

A wSnake elemeit egy Windows vagy Linux rendszeren futó szerver alkalmazás fogja össze. Minden továbbított csomag a szerverre érkezik be, ami továbbítja a megfelelő útvonal felé. Működés közben a szolgáltatás a forgalom alapján folyamatosan tanul, hogy a legjobb és leggyorsabb útvonalon továbbítsa a csomagokat.

A szerver szolgáltatás C# programnyelven készült egy, a 2. ábrán látható konzolos alkalmazás keretein belül.

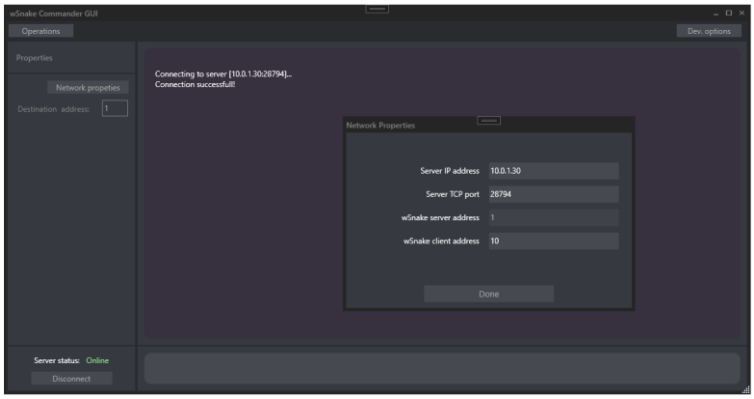
Hozzáférési pontok (AP-k)

A hozzáférési pontok kulcsfontosságú szerepet töltenek be. A vezeték nélküli végberendezések ilyen hozzáférési pontokon keresztül kapcsolódnak a szerverhez és egymáshoz. Ezek a pontok már meglévő Ethernet hálózatokon keresztül kapcsolódnak a szerver szolgáltatáshoz. Egy hálózathoz korlátlan számú hozzáférési pont kapcsolódhat. Az AP-k szoftvere Pythonban készült.



1. ábra: Hálózat felépítése

2. ábra: A szerver szolgáltatás felülete és alapkonfigurálása



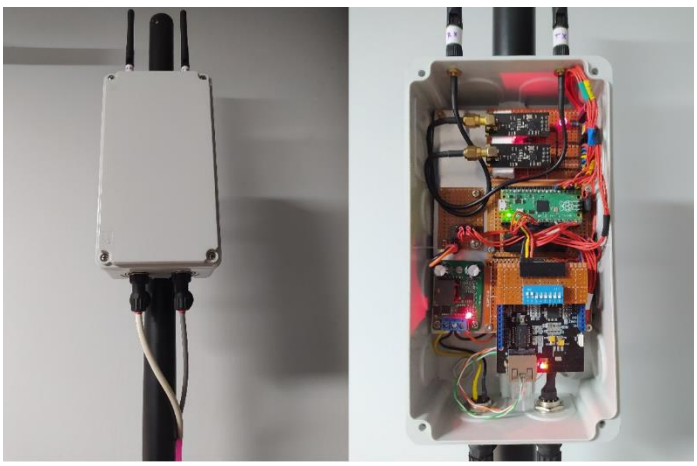
3. ábra: Windows grafikus kliens alkalmazás a hálózat tesztelésére

Végberendezések (Kliensek)

A végberendezések közé tartoznak azon IoT eszközök, vagy programok, amik a hálózatot használják. Kliensek lehetnek vezeték nélküli eszközök (ebben az esetben szükség van egy fentebb említett AP-ra), vagy „vezetékesek", amikor már meglévő Ethernet hálózaton kapcsolódnak közvetlen a szerverhez. A 3. ábrán látható egy Windows wSnake kliens.

2.2. Hálózat hardveres megvalósítása

Hardveres tervezésre és kivitelezésre elsősorban az AP-k felépítéséhez volt szükség. Fontos szempont volt az ellenálló képesség, ezért az AP hardverei egy kültéri villanyszerelési dobozban foglalt helyet (mely a 4. ábrán látható), ezzel IP67 szabványnak megfelelő védelmet biztosítva a komponenseknek.



4. ábra: Hozzáférési pont (1. generáció)

5. ábra: Hozzáférési pont csatlakozói

Az AP-n két GX16 vízálló csatlakozó felület lett elhelyezve, melyek használaton kívüli helyzetekre vízzáró sapkával rendelkeznek. A csatlakozó felület az 5. ábrán látható. Tápfeszültségként 8V és 40V közötti egyenáramra, és minimum 500mA bemeneti áramerősségre van szükség a stabil működéshez. Az adatkapcsolati csatlakozó egyelőre egy 10/100 Ethernet vonalat, és egy 3.3V UART Debug vonalat foglal magában (bővítőmodulokkal az UART vonal leváltható bármilyen kívánt szabványt alkalmazó vonalra).

3. Elért eredmények

A hálózat elemei sikeresen kapcsolódnak, és működik a csomagtovábbítás. A szerver különbséget tesz kliensek és AP-k között. A hálózaton továbbított csomagok forrása alapján tanul a szerver, hogy melyik eszköz melyik úton elérhető, ennek megfelelően a legrövidebb úton küldi felé a csomagokat. Emellett forrás- és célcím alapján szűrők állíthatók be, hogy melyik csomag merre mehet, vagy honnan jöhet.

A szerver szolgáltatás futtatható Windows és Linux operációs rendszereken egyaránt. Futás közben a klienseket több szálon kezeli, ezáltal kihasználva a több magos és több processzoros rendszereket.

A wSnake fejlesztése állandóan folyamatban van, sorra új funkciókkal és elemekkel gazdagodik. Tervek szerint a legközelebbi fejlesztésekként a következő elemek épülnek be:

Végponttól végpontig tartó titkosítás AES szabvánnyal, konfigurálhatóan 128, 192 és 256 bites kulcsméretekkel.

Hardveres gyorsítás beépítése a titkosított adatok visszafejtéséhez a hozzáférési pontokba, és opcionálisan kliens IoT eszközökbe.

Hozzáférési pont saját nyomtatott áramköri lapjának megtervezése, és legyártása alacsonyabb költség és jóval kisebb méret elérése érdekében.

Hozzáférési pontok szoftverének távoli frissítése, a hatékonyabb futás érdekében áttérés C++ programnyelvre.

Webes kliens, mely egy script importálásával lehetővé teszi weboldalak közvetlen csatlakozását a hálózathoz.

Alfa sugárzás kvalitatív és kvantitatív vizsgálata

αlpha
Fábián Gergő, Ignát Péter, Szabó Gergely Imre
Felkészítő tanár: Fegyver Imre
Huszár Gál Gimnázium, 4030 Debrecen, Diószegi út 21.

1. Bevezetés

A radioaktív anyagok külső behatás vagy energiaforrás nélkül energiát hordozó sugárzást bocsátanak ki. A radioaktív sugárzás veszélyes az emberi szervezetre, ugyanakkor sokszor meg kell mérni adott helyeken a sugárzás erősségét. Ezért érdemes a méréseket robot segítségével elvégezni. Mi ezt tettük kicsiben. Az alfa sugárzás kimutatható kamerával, mert az alfa részecskék a kamera receptoraiba csapódva elmozdítják annak töltéseit, ezzel a kijelzőn fehérre színeznek 1-1 pixelt. A sugárzás erőssége egy adott időn belül megjelenő fehér pixelek számából látható.

2. Probléma megoldásának menete

Csapatunk az alfa sugárzás kvalitatív és kvantitatív vizsgálatát tűzte ki célul. Egyrészt „láthatóvá akartuk tenni" a radioaktív sugárzást kamera segítségével, másrészt számítógépes programmal számszerűen ki akartuk mutatni, hogy a sugárzás intenzitása csökken a távolság növekedésével.

2.1. Hardver

A hardver egy dobozból, monitorból, mini számítógépből áll. (2.ábra) A doboz elkészítéséhez elsőnek is szétszereltünk egy régi kis nyomtatót a benne lévő fejmozgató mechanikáért, ehhez rögzítettünk egy Lego ev3 motort, amivel mozgatni lehet a szíjat és ezen keresztül a radioaktív tárgy tartóját. Erre azért volt szükség, hogy a sugárforrást közelíteni, és távolítani lehessen a kamerától.

A kamera és a radioaktív tárgy távolságának méréséhez ultrahang szenzort akartunk használni, de az alumínium doboz miatt változtatnunk kellett a tervünkön. Ugyanis a vas kivételével a fémek jobban verik vissza az ultrahangot, ami megzavarja a mérést. Ezért a motor fokszámolójára bíztuk a távolságmérést. Próbálgatással megnéztük, hány fokot kell forduljon a motor ahhoz, hogy a radioaktív anyag tartója kb. 1 cm-t közeledjen a kamera felé, és az így kapott értéket használtuk a programban a távolság kiszámításához. Fontos megjegyezni, hogy a Lego robot nem alkalmas a pontos mozgatásra, mérésekre; jelen kísérletben néha 1-1 milliméter hibákat ejtett.

A nyomtatófejmozgató mechanikát beleragasztottuk egy szerelődobozba, amit elsötétíthetőre készítettük. Hiszen a mérést csak sötét kameraképpel lehet végezni, hogy alfa-sugárzás okozta fehér felvillanások elkülöníthetők legyenek a többi pixeltől. A radioaktív tárgy betevését úgy oldottuk meg, hogy a doboz oldalát levehetővé tettük.

A szerelődobozba belehelyeztünk egy EV3 lego téglát, amit hozzákötöttünk a dobozon kívüli mini számítógéphez. A tégla működés közben világít, ezért leragasztottuk fekete szigetelő szalaggal. A kamerát egy csipesszel rögzítettük, az összecsíptetett fémes részeket pedig egy papírdarabbal választottuk el egymástól, nehogy rövidzárlatot okozzon. A kész eszköz fényképe az 1. ábrán látható.

2.2. **Szoftver**

A program a mérések elvégzésére egy adott ideig a kamera képéről a fehér pixeleket megszámolja és eltárolja a koordinátáikat. Az eltárolt koordináták segítségével pedig egy képben összesíti az összes felvillanást és különböző színekkel teszi láthatóvá a felvillanások mennyiségét. Ezeket a képeket el is lehet menteni. A mérések elkészítése után a program egy listában mutatja meg az eredményeket (fehér pixelek száma, időtartam, távolság). A programból vezérelhető az ev3 motor (a számítógép és az ev3 között USB kapcsolatra van szükség), amivel a tárgy távolságát lehet állítani és állítható a mérési időtartam, fényérzékenység is. A program képe a 2. ábrán látható.

2.3. **Kísérlet**

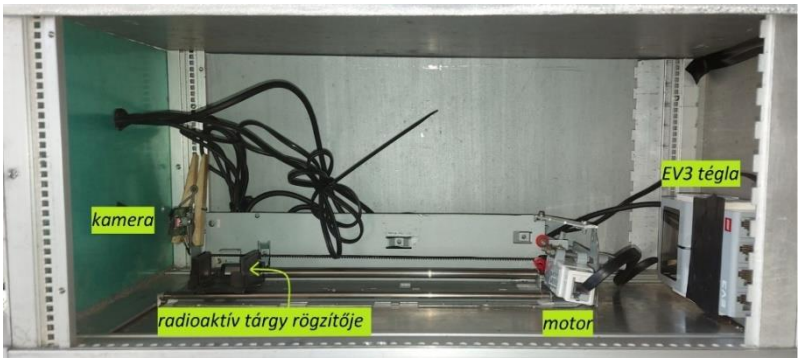
Végig látatlanban dolgoztunk a projekten. A munkánk kipróbálásához el kellett menjünk a Debreceni Egyetem Fizika Karára, ahol fizikusok felügyelete mellett kísérletezhettünk ThB-vel szennyezett tárggyal. A tórium B izotóp béta sugárzást bocsát ki, de leányelemei alfát, és mi ezt észleltük. Ahogy a radioaktív tárgy bekerült a helyére, rögtön látszódtak a felvillanások. A mérések előtt az eszközről készült fénykép a 3. ábrán látható. A radioaktív tárgy a doboz bal oldalán a csipesszel rögzített tárcsán van.

A kamera tesztelése után indulhattak a mérések. Először csak pár másodpercig figyelte a program a becsapódásokat, de úgy irreleváns eredményeket kaptunk. Tehát áttértünk a hosszabb idejű mérésekre. Közben megnéztük a programkészítette összesítő képeket, amik jól sikerültek. A 30 másodperces méréseknél (1. táblázat) megvalósult a célunk. A fél perc alatt történő 131db felvillanást összesítő kép a 4. ábrán látható.

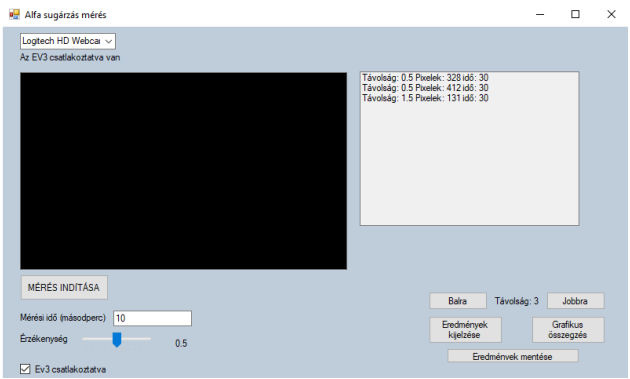
3. **Elért eredmények**

Örültünk, hogy eredményesen zárult a több hónapos munkánk: első próbálkozásra látszódtak a felvillanások a kameraképen és a program is tökéletesen működött. Az eredmények kimutatják, hogy a sugárzás intenzitása csökken a távolság növekedésével. A 30mp-es mérések adatait az 1. táblázat tartalmazza.

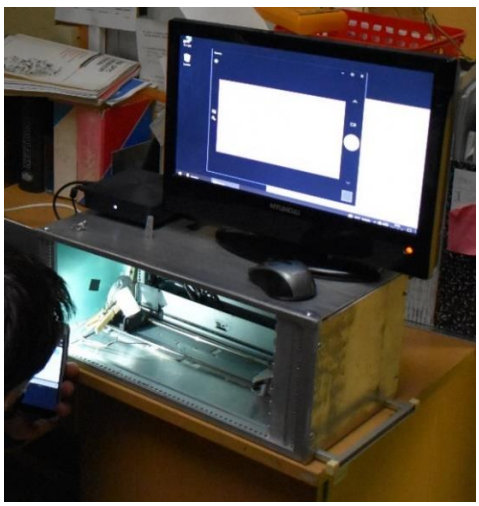
Bebizonyítottuk, hogy egy kamerával ki lehet mutatni az alfa sugárzást, ezt a szemléltetőmódot oktatási célra is fel lehet használni. Emellett, mint ahogy a bevezetésben is említettük fontos lehet a sugárzásmérést robottal elvégezni olyan helyeken, ahol a mérés az emberi egészségre veszélyt jelenthet.



1. ábra: alfa sugárzást vizsgáló doboz belülről



2. ábra: A program működés közben



3. ábra: A mérések előtt az eszközről készített fénykép

4. ábra: Összesítő kép

távolság	**fehérpixelszám**
0,5cm	328db
0,5cm	412db
1,5cm	131db
2,5cm	23db
2,5cm	24db
3,5cm	7db

1. táblázat: mérési adatok

LEGO X-Y plotter – Elon's Plotter

Team Elon

Kiss Lóránt, Szarvas Szilárd, Tulipán Nándor

Felkészítő tanár: Árva Sándor

BSZC Közgazdasági Technikum, 4200 Hajdúszoboszló, Gönczy Pál utca 17.

1. Bevezetés

Az ötletünk abból ered, hogy az általában drága, nehezen megfizethető X-Y plottereket egy szélesebb körben elérhető formában kivitelezzük. Míg számos plotter kit érhető el, ezeket nehezen lehet összeépíteni és beüzemelni. A legnagyobb hibája szerintünk ezeknek az X-Y plottereknek az, hogy ha bármilyen problémája akad a szerkezetnek, akkor az nehezen javítható. Ennek egyik oka, hogy minden ilyen plotter máshogy épül fel mind hardverileg, mind pedig szoftverileg. A problémának az egyetlen megoldása az, ha visszaküldjük a gyártónak, aki remélhetőleg meg tudja javítani, vagy ki tudja cserélni. Ráadásul mindez majdnem annyiba kerül, mint maga a gép.

Minderre a megoldást egy fizikailag és szoftverileg egyszerűbb kialakítású plotterben láttuk. Mivel az egész szerkezet nagyon egyszerű, így viszonylag könnyű átalakítani kisebb vagy éppen nagyobb felületekre is. Ennek oka, hogy semmilyen csúszó vagy görgő csapágyat nem tartalmaz, így szerkezete roppant egyszerű.

2. Probléma megoldásának menete

Kitűzött célunk egy olyan plotter létrehozása volt, melyet egy könnyen megérthető LEGO rendszer segítségével valósíthat meg bárki. A szoftver elkészítéséhez a Python nevű programozási nyelvet használtuk, mivel a Python nyelve az egyetlen nyelv, melyet támogat a LEGO rendszer.

2.1. Hardver

Elsőnek a hardvert raktuk össze. Először is szükségünk volt egy stabil felületre, amin a robot dolgozni fog. Az egésznek a fizikai alapja egy újrahasznosított bútorlap, egy szintén hulladék anyagból épített kihajtható lábbal. Erre lett felcsavarozva a két motor, és az egészet irányító LEGO EV3 okos tégla. A plotter az 1. ábrán látható.

A nagyobb nyomaték és pontosság érdekében 1:5-höz arányú lassító fogaskerekeket szereltünk a motorokra. Ezáltal a nyomaték az ötszörösére növekedett, ahogy a pontosságuk is, azaz ötödére csökkent a minimálisan fordítható szög. Kétféle rajzeszközt állítottunk össze, melyeket egy-egy szervo motor irányít. Komplex mechanikai rendszereken keresztül a motor forgató erejét a megfelelő irányba fordítva létrehoztuk a rajzeszközöket felemelő mechanizmusokat. Ez a 3. ábrán látható.

Mindemellett komoly fejtörést okozott a helyes súlyelosztás kialakítása is. A vezetékek csavarodása és a motor súlya által okozott különféle nemkívánatos erőhatások ellensúlyozása érdekében több ólomdarabot kellett a platformok alá függeszteni. Így ennek köszönhetően a szerszám minden pozícióban függőlegesen áll, és a rajzeszköz mindig megfelelően a papírhoz ér.

2.2. **Szoftver**

A szoftver Pythonban íródott. A projekt elején nem igazán tudtunk ezen a nyelven programozni, mivel főként C#-al foglalkoztunk eddig. Az első problémánk a szoftver megírásakor az volt, hogy meg kellett határoznunk egy G-code-ot, mely az adott kép koordinátáit adja meg. A G-code úgyszintén egy programozási nyelv, amit automatizált szerszámgépek programozására használnak.

Változó	**Leírás**
G	A G parancsok mondják meg a vezérlőnek, hogy milyen mozgást kell végrehajtania. (Pl. egyenes vonalú mozgás, ívelt mozgás balra vagy jobbra, kötetlen vonalú mozgás, stb.)
X	Az X tengely abszolút vagy növekményes helyzete.
Y	Az Y tengely abszolút vagy növekményes pozíciója.
Z	A Z tengely abszolút vagy növekményes pozíciója. (Ebben az esetben csak a toll felemelését, vagy lerakását adja meg.)
I	Meghatározza az ív középpontját az X tengelyen a G02 vagy G03 ívparancsoknál, ez egy relatív koordináta.
J	Meghatározza az ív középpontját az Y tengelyen a G02 vagy G03 ívparancsoknál, ez egy relatív koordináta.
F	Megadja a mozgási sebességet. (A mi robotunk ezt az értéket még nem használja fel.)

1. táblázat: A G-code legfontosabb változói

A G-code-ot egy Inkscape nevű programból vektorkép alapján exportáljuk ki. A robot beolvassa ezt a fájt, majd darabokra szedve külön-külön változókba tárolja le.

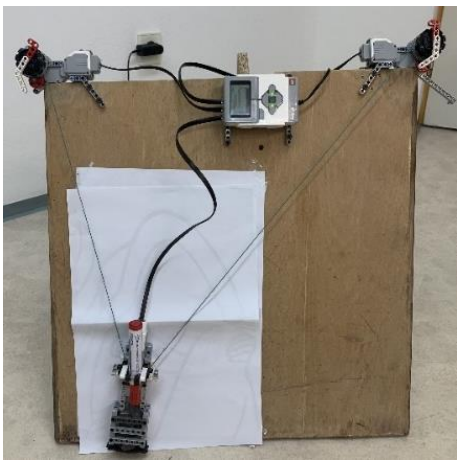
3. **Elért eredmények**

Hosszas próbálkozások sorozata után végül sikerült helyes mozgásra bírnunk a robotot. Nagyjából három A/3-as lap mindkét oldalának betelítése, és több száz programfuttatás után végül létrejöttek az első értelmes rajzok. Például először egy házikóval kezdtünk, amit sokadjára sikerült lerajzolni. Miután a fájlt beolvastuk, a programot az egyenes vonalú mozgással folytattuk. Ahogy az a 2. ábra a) pontjában látható, már ez is komoly problémákat okozott.

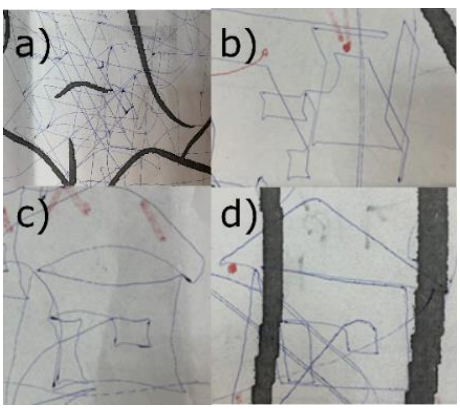
Jó néhány hét fejlesztgetés után lassan sikerült ezt némileg működésre bírni. A 2. ábra b) része már egy házikó kezdetleges képét mutatja. Itt még a pozicionálás sem működött helyesen. Valamilyen okból kifolyólag a házikó ablaka és ajtaja nagyjából megfelelően létrejött, viszont a falak és a tető már

teljesen eltűnt. A 2. ábra c) képén jól látható, hogy mindösszesen néhány hét fejlesztés után a szerkezet már sikeresen eléri a kívánt pontokat, viszont a vonalak még nem egyenesek. Ez a két motor nem megfelelő sebessége miatt történhetett meg. Ráadásul a program azon részében, ahol ez ellen kiszámítjuk a sebességet, pont fordítva adtuk meg nekik a szükséges értékeket, így még rosszabbá téve a problémát. Erre persze nagyjából négy hét kínszenvedés és fejtörés után jöttünk rá. Ezen sok idő munkájának gyümölcse látható a 2. ábra d) részén látható: a házikó már egyenes, megfelelő irányú és méretű vonalakból áll, de itt a toll felemelése még nem működött.

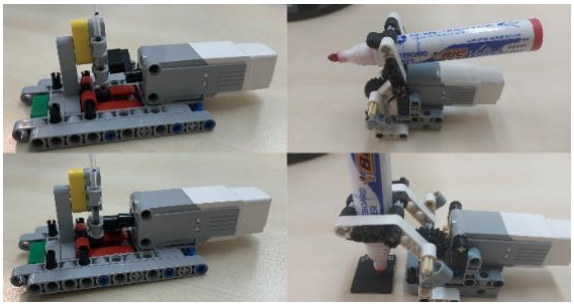
A 3. ábrán látható a két rajzeszköz felemelt és lerakott pozícióban. Ezzel csak akkor foglalkoztunk, amikor már működött az egyenes vonal rajzolása.



1. ábra: A plotter a filctoll szerszámmal.



2. ábra: A ház rajz fejlődése.



3. ábra: A két rajzeszköz felemelve és leengedve.

Az ívelt mozgással folytattuk. Ez még nehezebb volt, mint az egyenes, mivel sokkal több számítást kellett végrehajtanunk. Ez még fejlesztés alatt áll, de már értünk el sikereket. Például feltöltöttünk neki egy kör alakú rajzot, amit *nagyjából* kör alakban rajzolt meg.

4. ábra: Első felismerhető emberi rajz próbálkozásunk.

A 4. ábrán látható, hogy sok próbálkozás után sikerült először az emberi fej kör vonalát megrajzolnia a robotnak. Utána az arc és a kéz megrajzolásával foglalkozott a robot, ami a 4. ábrán látható, hogy nem pontosan sikerült neki. Ennél a rajznál átállítottuk a robotot, hogy az ívelt mozgást is egyenesként hajtsa végre, hogy helyesen működjön.

Jövőbeli fejlesztési tervünk az ívelt mozgás helyes működésre bírása, valamint más rajzeszközök elkészítése is. Emellett még a program esetleg beolvashatná az 1. táblázatban látható F változót a G-code-ból, így az megadná neki a sebességet is. Gondolkodunk még egy feedback rendszer kiépítésén is ami képes visszajelezni a robot rajzoló fej pozícióját), de ez persze még a jövő zenéje. A robot működése a következő videóban látható: Link[1]

[1] https://www.youtube.com/watch?v=wqsAgvHqAJs

Kockagyűjtő űrkuka

ŰrKukák

Joó Balázs, Rózsás Zoltán, Szabó Máté
Felkészítő tanár: Lang Ágota
Berzsenyi Dániel Evangélikus Gimnázium, 9400 Sopron Széchenyi tér 11.

1. Bevezetés: a probléma

1957-ig tiszta volt az ég felettünk. Ma közel 8000 műhold kering a Föld körül, ezeknek mintegy fele nem is üzemel már, ők jelentik az űrszemetet. Tekintve, hogy ezek az űreszközök – körpályájuk sugarától függően – több ezer m/s sebességgel haladnak, ha ütköznek valamivel, akkor a hatalmas mozgási energiájukból kifolyólag kárt tudnak tenni benne, még egy kisebb tömegű darabka is. A technika fejlődésével a műholdak mérete is csökkent. Manapság már nano-,piko- sőt femtokockákról (CubeSat) beszélhetünk, amelyek tömege rendre kevesebb, mint 10 kg, 1 kg illetve 100 g. Ezek történetesen nem járulnak hozzá az űrszeméthez, hiszen alacsony pályán mozogva a légkör közegellenállása folyamatosan fékezi őket, míg végül belezuhannak a sűrűbb légkörbe és elégnek. Csakhogy ezzel elégetünk bennük sok olyan eszközt, amelyet újra lehetne hasznosítani, pl. alapanyagokat kinyerni belőlük. Napjainkban egy aktuális probléma a chiphiány, de ha visszahoznánk ezeket a kockákat a Földre, azzal a környezetvédelmet is támogatnánk.

Ezért próbáltunk megtervezni egy űrkukát, amely összegyűjtené a már nem működő kockákat és visszatérne velük a Földre, majd újból begyűjtő körútra indulna.

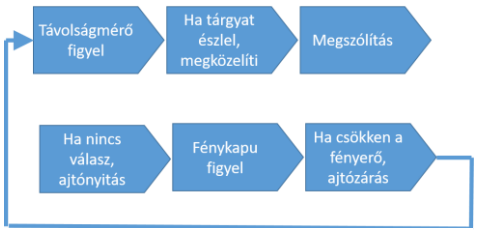
2. Modellünk

A valódi űrkukának a mérete és anyaga olyan kell, hogy legyen, hogy megérje fellőni, azaz minimum 20-30 kockának el kell benne férnie. Ezért mi természetesen az ötletünk modelljét tudjuk csak megvalósítani. Ennek mozgatása egy újabb változat feladata lesz, most a manőverezést nem szemléltetjük.

2.1. A kuka működésének elve

Az űrkuka a fellövés után rááll egy olyan pályára, ahol általában a kockák is keringenek. A rajta lévő – egy vagy több – távolságérzékelő (pl. lézeres szenzor) segítségével figyeli az előtte lévő teret. Ezek forognak is egy kisebb tartományban (a haladási iránytól számítva 90 fokkal jobbra és balra). Ha meghatározott távolságon belül észlel egy tárgyat, akkor megközelíti azt. Egy képelemző szoftver segítségével kideríti, hogy kockával találkozott-e. Ha igen, akkor egy bizonyos távolságból megpróbál rádiókapcsolatba lépni vele. Amennyiben az választ küld, akkor kikerülve azt, továbbrepül. Ha az első üzenetre nem jön válasz, még kétszer próbálkozik. Ha a kocka egyszer sem

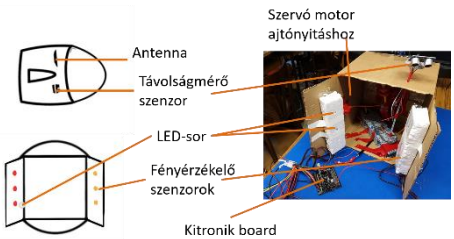
reagál, akkor megkezdi a begyűjtését. Ehhez első lépésként átfordul abba az irányba, amerre a kocka halad. Ezután kitárul a hátulján két ajtó. Az egyiken egy LED-sor van függőlegesen, a másikon – a fényforrásokkal szemben – fényérzékelők. Amennyiben ez a fénykapu a kezdeti értékhez képest kevesebb fényt érzékel (a fényerősség értékeket összeadva), akkor nagy valószínűséggel a kocka a két ajtó között van, így azok lassú becsukódása még beljebb tessékeli a kockát.



1. ábra: A működés lépései

2.2. Modellünk felépítése

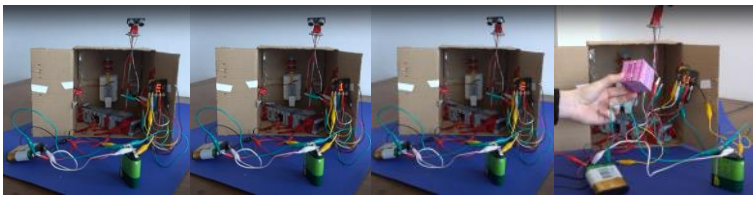
A földi űrkukánk alapja egy papírdoboz, hogy az ajtók nyitását-zárását tudjuk szemléltetni. Ezt a feladatot két szervómotor (nyitás) és egy LEGO-motor (zárás) oldja meg. Ugyancsak egy szervómotor segítségével forgatjuk a távolságérzékelőt, ami modellünkön egy ultrahang-szenzor. Ennek a forgatóegységnek az építésében is segítségünkre voltak a Technics LEGO elemei. 3 darab erős fényű LED-et illetve 3 fotoellenállást hungarocellbe fúrtunk bele, majd a vezetékeknek is csatornát vájtunk. Ezután az „ajtószárnyakra" erősítettük, belülre. A dobozban helyet kapott még egy csiga, amire záráskor feltekeredik a behúzó damil. Nyitáskor pedig a szervók egy gumiszálat húznak meg.



2. ábra: A modell felépítése

2.3. Modellünk működése lépésenként

Mivel szerettük volna a kockákkal való kommunikációt is szemléltetni, ezért a legcélszerűbb vezérlő egységnek a Microbit tűnt, ami rendelkezik rádiós kapcsolattal. Mivel több szenzort és motort kell működtetnie, ezért egy



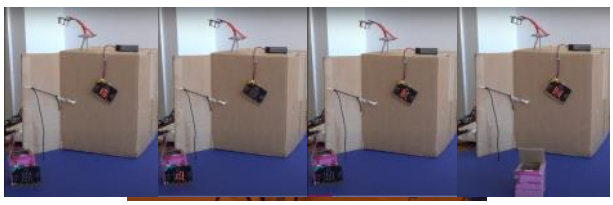
3. ábra: 1. lépés – figyel és meglát

Kitronik board – kiegészítő panelt is beszereztünk mellé. Programozásra a Microbit grafikus programnyelvét használtuk, beemelve az egyes kibővítéseket (pl. a panel, az ultrahangszenzor vagy a szervók kezelése).

Mivel egy viszonylag komplex rendszert próbáltunk összerakni, ezért a működés lépéseinek megfelelően egy-egy részfeladatot oldottunk meg egyszerre. Ezek többségéhez még nem alkalmaztuk a panelt, mert volt annyi szabad pin az alap Microbiten is. Ezek először a LED-ek és fényérzékelők együttes működésénél fogytak el, ekkor vetettük be először a panelt. A tesztelés során az érzékelők által mért adatokat kiírattuk, ha ezek alapján valami akció volt (pl. meg kellett hívnia a következő lépéshez tartozó eljárást), akkor annak a nevét írta ki (pl. nyitás). A meghívott eljárások pedig a Microbiten található gomb megnyomására indultak. Mivel a LED-panelen a Microbit egyszerre csak egy karaktert jelenít meg, ezért a képeken sajnos nem látszik, hogy pontosan mit írt ki.

A kuka tetején elhelyezkedő UH-szenzor 10 fokonként elfordul a szervómotor segítségével és minden helyzetben kijelzi számunkra a távolságot. Ha elé helyezünk egy kockát a megadott távolságon belül, akkor kiírja, hogy „Cube”.

Két űrkocka-modellünk közül az egyikben elhelyeztünk egy Microbitet, rajta egy rádióüzenetre váró programmal, míg a másik üres maradt – ő jelképezi a szemetet. A kukánkba egy másik Microbit került. Az ezen futó program először sorsol egy számot, amit ki is jelez. Ezután egy rádióüzenetet küld ki és várja vissza ugyenezt a számot a kockától. Látható, hogy a kocka megkapja ezt a számot illetve az visszakerül a kukában lévő Microbithez, aki ezt egy OK-kal nyugtázza. Az üres kockától azonban nem jön vissza válasz, így megkezdődik a begyűjtése, amire a tesztben a „Nyitás” felirat utal. A program itt fogja meghívni az azonos nevű függvényt, ami 3. lépésben kinyitja az ajtókat.



4. ábra: 2. lépés - kommunikáció

Az ajtók nyitása után – a teszt során egy gombnyomásra – felkapcsolódnak a LED-ek. A fénykapu működését úgy teszteltük, hogy kiírattuk a Microbit LED-paneljére a fotoellenállások értékét és azok összegét, majd betettük közéjük a kockát. Az így lecsökkent fényerő hatására hívja meg a program az ajtók zárását intéző függvényt.

5. lépésben a Microbit egy relé közbeiktatásával indítja el a LEGO-motort, ami egy damil segítségével behúzza az ajtókat.

3. Elért eredmények

Jelenleg ott tartunk, hogy miután mindent betettünk a dobozba, abba már a begyűjtendő kocka nem fér bele. Ennek legfőbb oka, hogy több áramforrást kell használnunk. Így a közeljövőben nekilátunk megépíteni az Űrkuka2.0-t, egy nagyobb méretű dobozban. Átgondoljuk a nyitás-zárás technikáját is, hátha még kitalálunk egy elegánsabb megoldást. A szenzorok működése és a LED-ek felkapcsolása valamint a kommunikáció rendben van.

6. ábra: teledoboz

masKey

***M*[5]**

Horváth Balázs Milán, Márkus Marcell Dávid, Szécsényi Máté
Felkészítő tanár: Mészáros Róbert
Kaposvári SZC Nagyatádi Ady Endre Technikum és Gimnázium
7500 Nagyatád, Dózsa György u. 13.

1. Bevezetés

A koronavírus-járvány megváltoztatta mindennapjainkat. Az oltásnak köszönhetően részben már visszatért életünk a megszokott mederbe, azonban a vírus, és ezáltal a maszkviselés továbbra is velünk maradt. Sokan nem vettél fel a harmadik oltást, vagy egyáltalán nem olttatták be magukat, így Ők továbbra is veszélyben vannak. A maszkviselési szabályok intézményenként eltérőek, azonban ha kötelező is maszkot használni, ennek betartását nehéz ellenőrizni. Ha belépéskor maszkot visel az illető, nem jelenti, hogy néhány perc múlva nem válik meg tőle, így az intézményi tartózkodás során is szükség van az ellenőrzésre. Ez azonban hatalmas emberi erőforrást igényelne, és terhet róna az iskola alkalmazottjaira, akik így nem tudnák egyéb feladataikat ellátni.

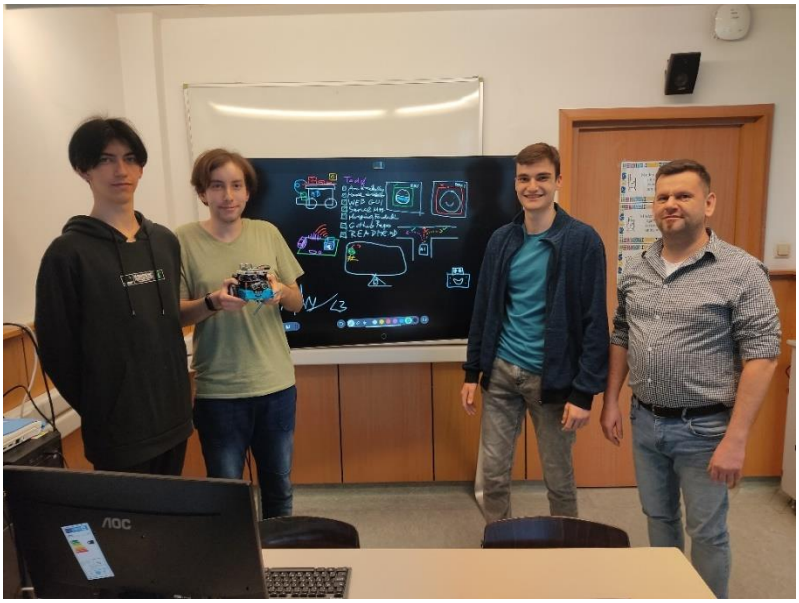
A projekt ezt a problémát hivatott megoldani: egy robotot hoztunk létre, ami önállóan képes közlekedni az iskola folyosóin, ellenőrzi, hogy az illető hord-e maszkot, és ha nem, akkor figyelmezteti a szabályok betartására.

2. Probléma megoldásának menete

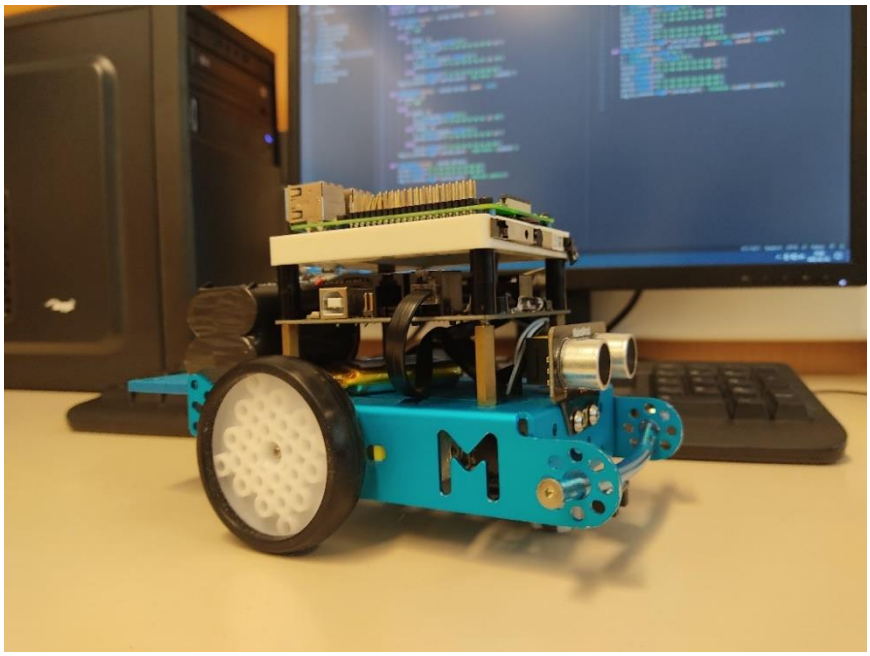
Projektünk célja egy egyszerű, felhasználóbarát, könnyen kezelhető, és költséghatékony eszköz megalkotása volt. Az egyszerűség érdekében felhasználtunk kész eszközöket, de készítettünk saját alkatrészeket is a robothoz.

2.1. Hardveres felépítés

A robot működése két fő vonalon épül fel: az egyik, hogy ellenőrizzük a maszk meglétét, és annak hiánya esetén figyelmeztetjük az érintettet, a másik pedig a robot automatikus közlekedése. A maszkellenőrzés viszonylag könnyen megoldható: szükség van egy kamerára, és egy miniszámítógépre. Kameraként egy már nem használható laptop webkameráját használtuk, USB-n keresztül csatlakoztattuk a miniszámítógéphez. A miniszámítógép pedig egy Raspberry Pi 3-as, Raspbian Lite operációs rendszerrel. A mozgató elektronikát pedig a Makeblock mBot adja, mely az ultrasonic szenzora segítségével tájékozódik. A két építőelem szükség esetén képes soros kapcsolaton keresztül kommunikálni. Az eszköz áramellátásáért egy erre a célra összeállított kétcellás lítium-ion akkupakk felel.



1. ábra: Az eszköz, és a csapat



2. ábra: A robot prototípusa

2.2. Szoftveres felépítés

A robot mozgásáért felelős mBot eszközön egy, a nyílt forráskódú gyári firmware alapján íródott saját firmware fut, mely az Arduino IDE segítségével készült. Ez a firmware olyan módosításokat tartalmaz, mely optimalizálja a működést az ellátott feladathoz. A maszkviselés felismerését és az arra történő reakciót végző Raspberry Pi 3 a Raspbian operációs rendszert futtatja, melyen az általunk írt kód fut. A kód Python nyelvben íródott, de az automatizációhoz bash scripteket is használtunk. A kód megírásakor nyílt forráskódú könyvtárakat is felhasználtunk, ezek pontos listája megtalálható a Github tárolónkon, mely az alábbi linken keresztül érhető el: https://github.com/tcgmilan/mbot-mask-detection

A maszkellenőrzést gépi tanulás, mesterséges intelligencia segítségével oldottuk meg (TensorFlow).

2.3. Működés a gyakorlatban

A robot önállóan közlekedik a folyosón, és pásztázza az ott tartózkodó személyeket. Amint észreveszi, hogy valaki nem hordja a maszkot, hangjelzéssel (text-to-speech megoldás használatával) szóban figyelmezteti az illetőt a szabályok betartására. A hangjelzések egy előre definiált listából választódnak ki, minden figyelmeztetés szövegét véletlenszerűen választja meg a készülék. Az anomáliák elkerülésének érdekében a maszkellenőrzés némi késleltetéssel történik, így elkerülhető, hogy végtelen ciklusba kerüljön, és „beakadjon a robot lemeze". Fejlesztési célunk, hogy valamely módon a maszkot hordókkal is interakcióba lépjünk, és pozitív megerősítést adjunk számukra cselekedetük helyességét díjazva. Ezt például egy egyenként címezhető LED-ekből álló mátrixból kirajzolt szmájlival vinnénk véghez.

2.4. Kihívások

Fontos volt, hogy a projektet úgy valósítsuk meg, hogy az ne ütközzön a GDPR-ral. Mivel kamerás megoldásról beszélünk, ez számos adatvédelmi kérdést felvet, és az eszköz használatát megnehezíti, korlátozza, hogyha adatvédelmi aggályok adódnak vele kapcsolatban. Éppen ezért nem gyűjtünk személyes adatokat, a maszkellenőrzés során kép nem kerül rögzítésre. A robot nem szankcionálja a maszkot nem viselő illetőt, csupán figyelmezteti, ezzel növelve a komfort és biztonságérzetet. Éles használat esetén eleinte minden bizonnyal meglepő, és a gyerekek számára érdekes lesz az eszköz. Ez felveti annak a kockázatát, hogy a „tesztelésük" során megválnak a maszktól, illetve az arcukhoz érnek fertőtlenítés nélkül, azonban a kezdeti izgatottságot követően várhatóan hozzászoknak majd.

3. Elért eredmények

A munka oroszlánrészével elkészültünk: a maszkdetektálás, és az arra adott reakció működik, illetve a robot képes önállóan közlekedni. A tesztek során a maszkviselés érzékelésével nem voltak gondok, ezt azonban a fényviszonyok

és a kamera elhelyezkedése befolyásolhatja, ezzel kapcsolatosan vannak még hátra tesztek. Éles környezetben még nem tudtuk tesztelni, de amint ez megtörténik, az ebből származó tapasztalatok alapján szeretnénk még fejleszteni az eszközt. A pályamű jelenleg nem rendelkezik külső burkolattal, így az informatikát kedvelők számára érdekesebb megjelenést biztosít, a végleges készüléknek azonban szeretnénk egy külső borítást, mely nagyobb biztonságot és barátságosabb megjelenést eredményez

Informatika szekció

Jarvis

The Avengers

Bálint Máté
Felkészítő tanár: Urbánné Jung Andrea
Szegedi Tömörkény István Gimnázium, Művészeti Szakgimnázium és Technikum
6720 Szeged, Tömörkény utca 1.

1. Bevezetés

Jarvis egy Javában írt virtuális asszisztens. Egy virtuális asszisztenssel szemben alapvető követelmény, hogy élő szóban adhassunk neki utasítást, amit a program megért, és reagálni is tud rá. Általában olyan feladatokra használják, amelyeket az emberek gyakran akarnak ismételni, de rendszeres információszerzésre is alkalmas. Az én virtuális asszisztensem az alábbi feladatkörökre képes: idő- és dátuminformációk megszerzése, emlékeztetés, zenelejátszás, időjárás-jelentés illetve filmajánlás.

2. Probléma megoldásának menete

2.1. Input, output

Az első és legfontosabb dolog az volt, hogy a mikrofon inputot Stringgé konvertáljam. Ehhez a sphinx4 könyvtárat használtam. A main metódusban egy végtelen loopot indítok, hogy a hangot rögzíthessem, majd ezt a voiceCommand Stringben tárolom. Innentől kezdve rengetek if, else szerkezettel döntöm el, hogy mit is csináljon a program.

A második legfontosabb egy virtuális asszisztens esetén, hogy válaszolni tudjon. Mivel a program eredetileg privát felhasználásra készült így nem TTS API-t használok, hanem csak simán audiófájlokat. A hangfájlok az eredi IOS mobilapplikációból vannak, de némelyik fájl nevét megváltoztattam, az egyszerűbb használat érdekében (pl. amikor azt kérdezem, hogy milyen nap van ma). A változatosság kedvéért Jarvis nem mindig ugyan úgy válaszol, véletlenszerűen dől el a megerősítés.

2.2. Idő

A program parancsai közül ötnek köze van az időhöz. Három parancs a jelennel kapcsolatos, míg a másik kettő egyfajta emlékeztetőként szolgál. Az első három megoldásához Calendar osztályt használva szerzem meg a parancsnak megfelelő adatokat, majd lejátszom az adatoknak megfelelő hangfájlokat. Mivel a „számos” hangfájlok nullától kilencig úgy kezdődnek, hogy nulla plusz a szám, de a Calendar osztály úgy adja vissza, hogy csak szám, így ellenőriznem kell a Calendar-os String hosszát, hogy a jó audiót játszhassam le.

A triggered metódus a reminder parancs szíve, a felhasználó inputját felosztja a kettőspontoknál, majd kiszámolja, hogy az idő amit beírtunk

összesen hány másodperc. Ezután a Timer és a TimerTask osztály segítségével a kiszámolt idő ezerszeresével indítok egy időzítőt. (Ugyanis a schedule metódus ezredmásodpercet vár paraméterként.) Amikor az időzítő lejár a ReminderClass run metódusa kerül végrehajtásra. Az alarm parancs nagyon hasonló a reminder-höz. A megfelelő másodpercet úgy kapom meg, hogy a várni kívánt időpont másodpercre konvertált értékéből kivonom a jelenlegi időpont másodpercre konvertált értékét.

2.3. Internet

A WebReader osztályt arra használom, hogy oldalak forráskódját olvassam be, illetve hogy ellenőrizzem azt, hogy van-e internetje a felhasználónak. Amennyiben nincs, Jarvis szólni fog, majd kilép. Itt található még a giveTextBetween metódus is, amely az 1. ábrán látható (plusz kétszer felülírva). Az alapkoncepció az, hogy a source Stringben megkeressük a before String első előfordulását, majd ehhez hozzáadjuk a before String hosszát. Ettől az integer értéktől kezdjük el keresni az aftert a source-ban. Ez az end integer. Ezután visszaküldjük a start és az end közötti szöveget, úgy, hogy a végén és elején lévő felesleges space karaktereket is levágjuk a trim metódus segítségével.

```
public static String giveTextBetween(String source, String before, String after) {
    int start = source.indexOf(before);
    start += before.length();
    int end = source.indexOf(after, start);
    return source.substring(start, end).trim();
}
```

1. ábra: A giveTextBetween metódus

2.4. Zenelejátszás

A play vagy a music parancsal zenét hallgathatunk. Mivel a music paramétert elég, hogy tartalmazza (a jarvissal együtt persze) így akár azt is mondhatom, hogy „jarvis, i want to listen to music” akkor is aktiválódni fog. Valójában a parancs nem ez lesz, mivel a beállításoknál megadott szótár fájl nem tartalmazza csak a „jarvis” és a „music” szót a mondatból, de azt a kettőt fel fogja ismerni, tehát aktiválódni fog. A PlayGUI osztályt az AWT Frame osztályból származtatom és kiegészítem az ActionListener és a KeyListener osztállyal, hogy a kattintást és az entert is támogassam. Osztályváltozóként létrehozom a GUI-hoz szükséges objektumokat, változókat, majd a konstruktorban be is állítom őket. A playSong metódus felel a link megszerzéséért és megnyitásáért, ezt a gomb kattintásával, vagy az enter lenyomásával érhetjük el. A textfieldben lévő Stringben kicserélem a szóközt egy plusz jelre, mivel a YouTube azt használja szóközként. Ezután a source Stringben tárolom az oldal forráskódját, ebben megkeresem a "/watch?v=" és az első idézőjel közti Stringet. Ez a videó azonosítója, miután ez megvan, megnyitom a linket a böngészővel.

2.5. Időjárás

Jarvissal lekérhetjük az aznapi Szegedi időjárás jelentést is. Az adatokat az időkép honlapjáról szerzem meg a WebReader osztály segítségével. Először is el kell döntenünk, hogy mínusz fok van-e vagy sem, ezt úgy tehetjük meg egyszerűen, hogy megnézzük, hogy a hőmérsékletet tartalmazó String tartalmazza-e a „-" jelet. Ezután abban az esetben, ha szeles az idő, azt is elmondja Jarvis, majd végül a jelenlegi időjárásra kerül a sor. Erre azért van szükség külön lépésben, mert a napos idő is lehet szeles, ugyanúgy mint az esős, ezért már az oldal is külön vette ezt a két dolgot.

2.6. Filmajánló

A movies paranccsal megtekinthetjük a legújabb és legnézettebb filmek listáját. Az időjáráshoz hasonlóan itt is a WebReadert használom. Az adatokat egy kétdimenziós String tömbben tárolom, amelynek az első dimenziójában annyi elem lehet, amennyi film van éppen az ajánlottak között. Ezt a számot Regex-el nyerem ki, és ez lesz az a szám ami hússzal megszorozva a Frame dinamikus magasságát adja. A második dimenzióba a három kerül, mivel egy filmhez három információt akarok tenni. A while loopban megkezdődik az adatkinyerés és feldolgozás folyamata. Ha az indexek kisebbek nullánál, az azt jelenti, hogy elfogytak az Stringek a sourceban. A filmcím és a nyelvhez a második ábrán látható giveTextBetween-t használom, a minőséghez pedig a harmadik ábrán láthatót. Ebben a metódusban azért használok +2-t mert a nyelv adat vége egy záró kacsacsőrrel és egy idézőjellel végződik, ami kettő karakter. Ezután a két karakter után már jön is a minőség. A loop végén újítom az index változókat úgy, hogy az előző index + 1-től újra elkezdem keresni a keresendő Stringeket. A +1-gyel előzöm meg azt, hogy önmagát érzékelje újra. Miután a loop feldolgozta az adatokat és belepakolta a tömbbe, a JTable osztály segítségével megjelenítem azt a felhasználónak.

```
public static String giveTextBetween(String source, String before, String after, int index) {
    int start = index;
    start += before.length();
    int end = source.indexOf(after, start);
    return source.substring(start, end).trim();
}
```

2. ábra: A felülírt giveTextBetween metódus

```
public static String giveTextBetween(String source, int start, String after) {
    start += 2;
    int end = source.indexOf(after, start);
    return source.substring(start, end).trim();
}
```

3. ábra: A másodjára felülírt giveTextBetween metódus

3. Elért eredmények

Jarvis volt az első projektem ezáltal rengeteget fejlődtem az írása közben. Első sorban a Stringekkel dolgoztam, így volt időm elsajátítani eme karaktertömböknek a metódusgyűjteményét és használatát. Ezzel párhuzamosan elsajátítottam az objektumorientált programozás alapjait. A következő egyik leghasznosabb tudás amit szereztem nem más mint az internetprogramozás. A legelső verziókban a JSoup HTML parsert használtam, majd szépen lassan áttértem a JDK-ban található osztályokra és a manuális adatszűrésre. Pár olyan tudást is elsajátítottam, amely nem látszik a programban. Ilyen például a többszálú programozás. Az emlékeztetőkhöz eleinte a Runnable interfészt használtam, de végül találtam egy jobb megoldást a TimerTask osztály segítségével, így téve Jarvis stabilabbá. Talán a legfontosabb ismeret amit szereztem az nem más mint a git. Mivel a projekt kezdett egyre nagyobb lenni, és szerettem volna új dolgokkal kísérletezni, így elkerülhetetlen volt egy verziókezelő megismerése és elsajátítása. Jarvissal rész vettem még a 17. Neumann, Nemzetközi Tehetségkutató Programtermék Versenyen. Azóta több hibát is kijavítottam, több ellenőrzés van és a hőmérséklet bemondásán kívül a teljes időjárás jelentés is támogatott. Jarvis és a forráskódja az alábbi linken érhető el[1].

.

[1] https://drive.google.com/file/d/1dpzPqq6RAFgSpsPCS3QMKmf4XVgvPXO/view?usp=sharing

Kerikriell

Kerielit

Erdélián Zoltán, Kerekes Márk
Felkészítő tanár: Molnár Zsolt
HSZC Návay Lajos Technikum és Kollégium, 6900 Makó, Posta utca 4-6.

1. Bevezetés

A matematika az egyik olyan tantárgy, amit úgy lehet a legkönnyebben megtanulni, ha gyakoroljuk a feladatokat, feladattípusokat. A tankönyvek egyes feladattípusokból elég hamar kifogynak, mivel ha sok feladat lenne benne, elég vastag lenne a könyv. Az is előfordulhat, hogy a tanár ad fel egy feladatot, ahol az egyes adatok lehet tévesek és így az egész számítás eredménye hibás lehet. Mi erre a problémára próbáltunk egy megoldást alkotni, a Kerikriell-t. A név az iskolánk becenevéből, a Keriből, és egy másik, matematikával foglalkozó weboldal, az Akriell nevéből állítottuk össze.

2. A probléma megoldásának menete

2.1. Programozási nyelvek választásának indoklása

A programhoz HTML, CSS, illetve JavaScript nyelveket használtunk, mivel egy webalkalmazás könnyebben lesz elérhető, mint egy asztali alkalmazás és akár telefonon is lehet használni, mivel a Bootstrap segítségével mindenféle felbontásban megfelelően tud megjelenni az oldal, így nem kell össze-vissza görgetni, meg nagyítani, vagy kicsinyíteni az oldalt. A JavaScript segítségével az oldal lényegét, a matematikai feladatok megvalósítását tudtuk megcsinálni.

A webalkalmazás hatékony elkészítéséhez a programot modulárisan, objektum orientált elvek alapján készítettük el, ezért is választottuk a JavaScript-et. Ennek köszönhetően programot 3 fő részre tudtuk bontani.

2.2. A program főbb részei

Az MData-ban találhatóak a feladattípusok témakörökre lebontva. Egy objektum egy témakört foglal be és ezen belül különböző nehézségi szintek vannak. Nehézségi szinteken belül vannak eltárolva a feladat modulok, amik tartalmazzák a megjelenítendő címet és feladatleírást, illetve az eredményt kiszámoló függvényeket.

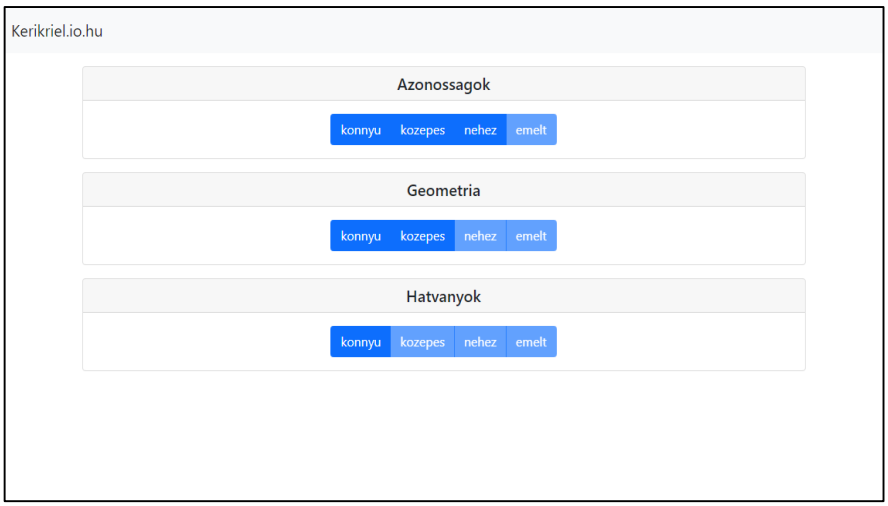
Az MUtils-ban találhatóak olyan kisebb dolgok, mint random egész, vagy tört szám generálása, illetve random karakter generálás. Ugyanitt található a program MDebug része is, ami arra szolgál, hogy a tesztelést egyszerűbbé tegye, és itt lehet például új feladatokat generálni a megadott téma és nehézség alapján, illetve a feladatok megoldását is le lehet kérni.

Az MManage segítségével tudjuk kezelni az előbb említett modulokat. A program itt végzi el a feladatok generálását, értékelését és egyéb, a

programhoz szükséges dolgokat. Például itt kerül ellenőrzésre a felhasználó által beírt eredmény, amit összehasonlít a feladat eredményével. A feladatokat az MManage modul mellett levő MTasks modulban kezeljük, ahol a feladatokat egy tömbben tároljuk el, így több feladatot le lehet generálni. A képen (1. ábra) látható rész felelős a kezdőlapért, amin a témakört ki lehet választani.

```
printTaskSelect() {
    this.show('#taskSelect');
    this.hide('#taskSolve');
    this.hide('#resultsContainer');
    let container = document.getElementById('taskSelect');
    container.innerText = '';
    let btn = document.querySelector('#startBtnTemplate > .btn');
    let rowTemplate = document.querySelector('#startTemplate > .row');
    for(elem in MData) {
        let newRow = rowTemplate.cloneNode(true);
        newRow.querySelector('#rowHeader').innerText = elem;
        for(difficulty in MData[elem]) {
            let newBtn = btn.cloneNode(true);
            newBtn.innerText = difficulty;
            console.log(difficulty+'');
            newBtn.setAttribute(`onclick`,`MManage.selectTask('${elem}','${difficulty}')`);
            if(MData[elem][difficulty].length==0) newBtn.classList.add('disabled');
            newRow.querySelector('#rowBtnGroup').appendChild(newBtn);
        }
        container.appendChild(newRow);
    }
},
```

1. ábra: A kód, ami a kezdőoldalért felelős

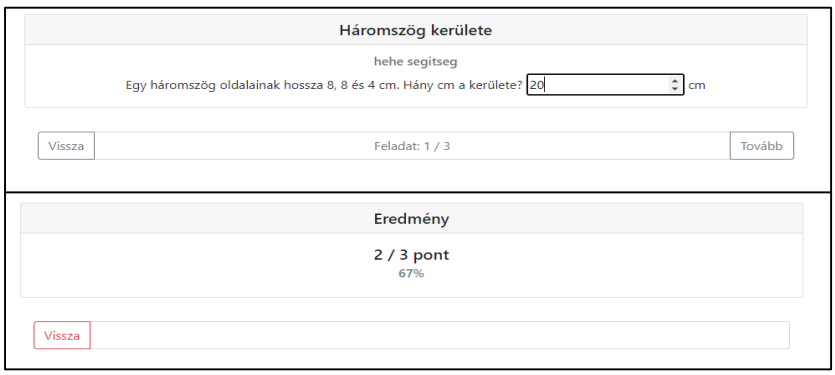


2. ábra: Kezdőoldal

2.3. **A program felépítése**

Az oldal felépítése nincs túlbonyolítva. Az egészhez csak 1 HTML fájl kellett, amihez a tartalmat a JavaScript-ben generálunk le. A HTML csak a vázat biztosítja. Először kiírja, hogy milyen témák vannak, és onnan lehet maximum 4 nehézségi szint közül választani a téma címe alatt levő gombokkal. Ahol nincs egy adott nehézség, akkor az ahhoz tartozó gombot nem lehet megnyomni. Ez a következőképpen néz ki: (2. ábra)

A nehézség kiválasztása után a program alapértelmezetten legenerál 3 feladatot az adott nehézségből. A felhasználó ilyenkor be tudja írni a megoldást a szám, vagy (egyenletek esetén) szöveges mezőbe, a „Következő” gombbal továbblépni a következő feladatra és a „Vissza” gombbal visszamenni a témaválasztó felületre. A feladatok elvégzése után a program kiírja, hogy a feladatokból mennyi lett helyesen megoldva, illetve kiírja százalékos formában az értékelést is (3. ábra).



3. ábra: (felső rész) Egy feladat (alsó rész) A feladatok kiértékelése

ElmeSéma

csapatnév_végleges(1)másolata

Horváth Gergely Zsolt
Felkészítő tanár: Répásné Babucs Hajnalka
BMSZC Neumann János Informatikai Technikum, 1144 Budapest, Kerepesi út 124.

1. Bevezetés

Biztos mindenki átélte már azt az élményt, amikor eszébe jutott egy világmegváltó ötlet. Egy ötlet, ami csak neki és csak akkor juthatott eszébe. Ezt abban a pillanatban tökéletesen szavakba is tudta volna önteni. Csakhogy mire eljutott odáig, hogy egy noteszhoz, illetve íróeszközhöz jusson és papírra tudja vetni, addigra a „heuréka" pillanatnak vége lett, az ötlet pedig elszállt.

Erre a problémára kínál megoldást az ElmeSéma nevezetű applikáció. Segítségével nem szükséges többé hosszasan jegyzettömböt keresni vagy a megfelelő alkalmazást kutatni a telefonon, ha éppen megszállja az embert az ihlet. Helyette gondolatainkat azonnal szavakba önthetjük. Ezzel rengeteg ötlet menthető meg az enyészettől és számos gondolati folyamat leegyszerűsíthető, fejleszthető.

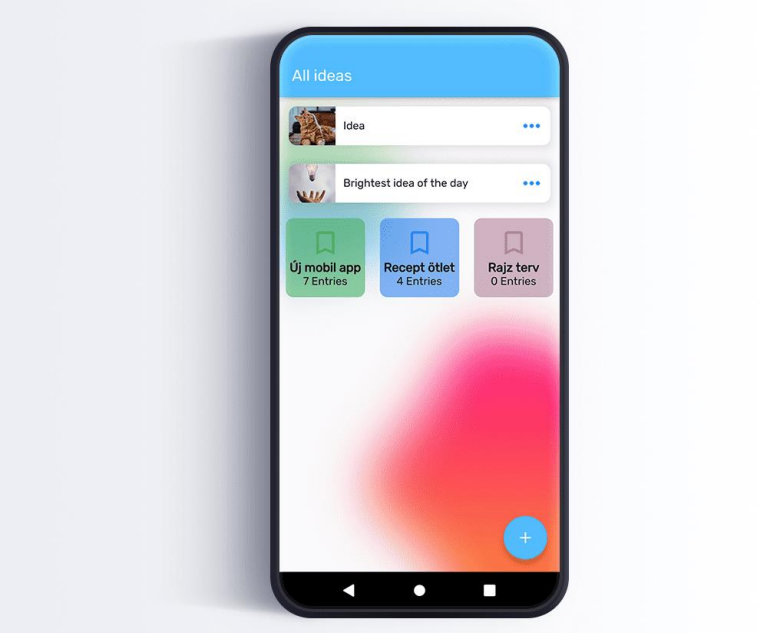
2. Probléma megoldásának menete

E problémakör megoldására a legideálisabb formátumnak a mobilalkalmazás tűnt, hiszen az okostelefon szinte minden ember zsebében ott lapul. Ez egyrészről előnyt jelent, hiszen így nem szükséges más médiumok után kutatni egy ötlet feljegyzéséhez. Másrészről a mobiltelefon az egyénhez viszonyított általános közelsége miatt lehetővé teszi, hogy a többi alternatívánál gyorsabban lehessen elkezdeni az ötlet rögzítését, ráadásul rugalmasabb és változatosabb módokat is kínál erre.

2.1. Az ElmeSéma alkalmazás

Az applikáció középpontjában az ötletek állnak. Éppen emiatt – ahogy az 1. ábra is mutatja – a központi képernyőn egyből ezek láthatóak. A könnyebb megkülönböztethetőség érdekében nem csak elnevezni lehet őket, de azok számára, amelyek képet tartalmazó bejegyzéssel is rendelkeznek, az alkalmazás automatikusan beállít egy előnézeti képet. Itt nem csak láthatóak a már létező ötletek, hanem ezen a képernyőn van lehetőség létrehozni újabbat is. Ezt többféle módon tehetjük meg: manuálisan, illetve a beépített érintésmentes funkciót használva.

Ha létrehoztunk egy ötletet, lehetőségünk van bővíteni azt sokféle elemmel. Ilyen többek között a szöveges bejegyzés, amivel írott módon önthetjük szavakba gondolatainkat.



1. ábra: Az ElmeSéma kezdőoldala

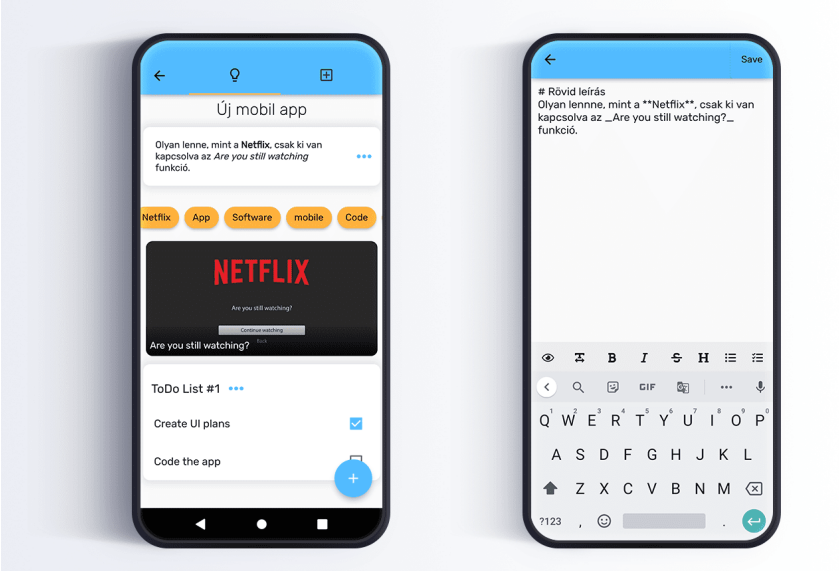
Ahogy a 2. ábrán is jól látható, a Markdown formátum segítségével felhasználó egy olyan robusztus – de ezzel egy időben intuitív és könnyen kezelhető – felületet kap, amellyel játszi egyszerűséggel jegyezheti fel és formázhatja meg az elképzeléseit. A sokrétű személyre szabhatóság biztosítja, hogy ami a gondolat szintjén összeállt, az a képernyőn is hűen tükröződjön.

Ezen jegyzetek mellett akár képekkel is színesíthetjük ötletünk leírását. Például, ha meglátunk valami igazán inspirálót a környezetünkben, abban az esetben elég egy fotót készíteni róla, majd az eszközről ezt néhány gombnyomással betölteni, és máris az elképzelés többi eleme között láthatjuk viszont.

Abban az esetben, ha konkrét teendőket is rendelnénk az ötlethez, az ElmeSéma ebben is a segítségünkre van. Pár érintéssel könnyedén létrehozható egy teendőlista, amelyen jelölni lehet az elvégzendő, illetve a már elkészült teendőket. Természetesen az alkalmazás kardinális pontja, hogy a kreativitásnak a lehető legnagyobb mértékben teret engedjen, éppen ezért a felsoroltak mellett még több bejegyzés típust tartalmaz.

2.2. Érintésmentes használat

Sokak meglátása szerint, illetve az anekdotákat kutató tanulmányok alapján (például Linda A. Ovington et al., (2018) Do people really have insights in the shower?

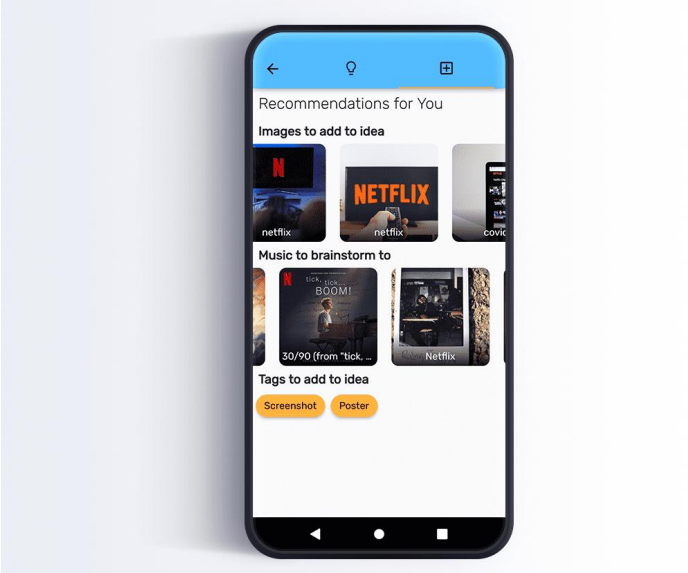


2. ábra: Egy szöveges bejegyzést tartalmazó ötlet és annak létrehozása

The Journal of Creative Behavior, 52. évf. 21-34.) az ötletek, (illetve azok részletei és a hozzájuk tartozó problémák megoldásának) jelentős része a zuhany alatt jut az emberek eszébe. Ilyen és ehhez hasonló esetekben meglehetősen hasznosnak bizonyulhat az alkalmazás érintésmentes használatát segítő eszköz. Az ElmeSéma erőteljes gépi tanulással támogatott megoldásának használatával a felhasználó hangvezérléssel hozhat létre ötleteket és azokhoz bejegyzéseket. Ily módon az érintésszám minimálisra csökkenthető, nagyobb komfortot nyújtva. Nem szükséges tehát egy zuhany alatti „heuréka pillanatban" azonnal papírért és a törölközőért nyúlni, azon aggódva, hogy az ember fejéből bármikor kimehet a gondolat.

2.3. Az ajánlások és ötletelés fül

Néha, amikor az ember sikeresen feljegyezte az ötletét, előfordulhat, hogy hirtelenjében elapad az inspiráció, az ötletelés folyamata pedig lelassul vagy teljesen leáll. Ha ihletre lenne szükség, az ElmeSéma több módon is tud segíteni. A gondolkodási folyamatra serkentő hatással lehet a zene. Éppen ezért a felhasználó által megadott adatokat figyelembe véve az applikáció, a Spotify rendszerét szorosan integrálva, különböző zeneszámokat ajánl. Az alkalmazás arra az esetre is kínál megoldást, amikor ennél valami kézzelfoghatóbb dologra lenne szükség. Segítségével az eddigi bejegyzések alapján releváns képeket és rövid kulcsszavakat tartalmazó címkéket adhatunk hozzá az ötletünkhöz. Mindezt a 3. ábra ábrázolja.



3. ábra: Kép, zene és címke ajánlások

2.4. Felhasznált technológiák

Az ElmeSéma megvalósítása és a lehető legkézenfekvőbbé tétele igazi technikai kihívást nyújtott. A Dart és a Flutter framework használata nagyban megkönnyítette a fejlesztési folyamatot. Az olyan külső szolgáltatások, mint az Unsplash API és a Spotify API zökkenőmentes integrációjában is nagy szerepet játszott. A felhasználói élmény tökéletesítéséhez szükséges volt különböző AI megoldások implementációjára: ilyen többek között a Google ML Kit Vision nevű képfeldolgozási rendszere, illetve a Dialogflow, amely a nyelvfeldolgozásért, ezen belül a felhasználó parancsainak értelmezéséért felelős.

3. Elért eredmények

Az elkészült alkalmazás felhasználási skálája meglehetősen széles spektrumon mozog. A gyors ötlet feljegyzéstől kezdve a brainstorming támogatásáig szinte minden használati esetet lefed. Az ElmeSéma a felhasználó elméjének tartalmát, gondolatait mind konvencionális eszközök, mind gépi tanulás segítségével könnyen érthető, átlátható és visszakereshető sémákba rendezi.

A mozgás és a játékélmény összekapcsolása

Szücs Barna

Szücs Barna

Felkészítő tanár: Vereb Márta

BGSZC Mechatronikai Technikum, 1118 Budapest, Rétköz utca 39.

1. Bevezetés

A mai társadalomban gyakori a játékfüggőség és a részben ehhez köthető elhízás. Az elhízásra egyszerű megoldás lenne a testmozgás, de sok ember nem hajlandó ennek eleget tenni részben a játékfüggőség következményeképpen. A célom ebben a projektben egy egyszerű Python program elkészítése, ami egy élvezhető játékban testesül meg és mozgásra készteti az embereket.

2. Probléma megoldásának menete

Játékként a Flappie bird-öt választottam, ami egy régi és egyszerűen elkészíthető játék. Egy madárról szól, akinek repülés közben csöveket kell kikerülnie. Ezek a csövek 2-es párokban egymás fölött helyezkednek el, egy kis (megadott szélességű) rést hagyva maguk között a madárnak. A játék egy jelzéssel irányítható, ami az ugrásra készteti a madarat. Amennyiben a madár földet ér, a játék véget ér. Ha a madár csőbe ütközik, ugrásképtelenné válik, lezuhan és íly módon ér véget a játék. A madár nem képes a képernyőből kimenni (a plafonról visszapattan).

Ami mozgásra készteti a játékost, az a játék irányítása **-->** az ugrás jelzése a végtagok/test mozgatásával lehetséges. Ehhez a számítógép kameráját használom.

2.1. A programkód

A programkód két kisebb program egybekötésével készült.

2.2. A "Flappie bird"-nevű játék általam programozott (szerény grafikájú) verziója

A megjelenítéshez turtle-t használok, a madár x-koordinátája rögzítve van a képernyő közepéhez és csak az y-tengely mentén mozoghat (zuhanás/ugrás). A csövek közötti rés y-tengely menti elhelyezése randomizálva van, de a szélessége megadott hosszúságú. A framek egy while True-ciklussal vannak megoldva, ahol elsőként a program frissíti a képernyőképet, a madár y-tengely menti sebességét csökkenti (gravitációs effekt), azután pedig az új sebességeknek megfelelően mozgatja a madarat és a csöveket (a csövek x-tengely menti sebessége adott). Miután a képernyőn lévő "szereplőket" elmozgatta, a program ellenőrzi a tárgyak új pozícióit:

- ha a madár belemegy a plafonba akkor az y-tengely menti sebességét lenullázza

- ha madár földet ér akkor a játéknak vége és a program újra indítja önmagát, hogy új menetet kezdhessen
- ha a madár áthaladt egy résen, akkor a pontszámláló pontszámához hozzáad 1-et. Ez az eset úgy van megállapítva, hogyha az (1)-es és (2)-es egyenlet egyaránt teljesül.

$$MadárXKoord - CsőXKoord > \frac{(MadárSzélesség + CsőSzélesség)}{2} \quad (1)$$

$$MadárXKoord + \frac{MadárSzél - CsőSzél}{2} - CsőXKoord \geq CsőXSebesség \quad (2)$$

Ahol

- MadárXKoord – A madár X koordinátája
- CsőXKoord – A cső X koordinátája
- MadárSzél (vagy MadárSzélesség) – a madár X tengely menti kiterjedése pixelekben kifejezve
- CsőSzél (vagy CsőSzélesség) – a cső X tengely menti kiterjedése pixelekben kifejezve
- CsőXSebesség – A cső x tengely menti sebessége

- ha a madár csőbe ütközik akkor nem engedi, hogy ugorjon és a játék vége előtt egy zuhanás jelenetet hoz létre. Ez az eset akkor áll fenn, ha a (3)-as és (4)-es egyenlet egyaránt teljesül.

$$|MadárXKoord - CsőXKoord| < \frac{(MadárSzélesség + CsőSzélesség)}{2} \quad (3)$$

$$|MadárYKoord - CsőYKoord| < \frac{(MadárSzél + CsőHossz)}{2} \quad (4)$$

Ahol CsőHossz – a cső Y tengely menti kiterjedése pixelekben kifejezve.

- ha az egyik csőpár egy cső fentről a másik lentről jön) kimegy a képernyőből akkor egy másik csőpár mögé helyezi új y-tengely menti magassággal (rés magasságát változtatja)

A frame futtatása közben a program méri az időt is és a frame végén annyit vár amennyi még hiányzik az előre megadott frame-időhöz.

2.3. Egy program, ami a kamera aktuális képéből megállapítja néhány a testen elhelyezkedő pontnak a koordinátáit

A pontok megtalálásához a Google Mediapipe programcsomagját használom. Az ugrás-jel küldéséhez sok feltételt megadhatunk (pl. ha a könyök csúcsa magasabban van, mint a váll, akkor ugrás-jelet küld). Ebben a megoldásban a jel küldésének feltétele az, hogy a könyök sebessége nagyobb legyen egy

megadott értéknél. A megoldásban ezt a feltételt használtam, de kiemelném, hogy egyéb pontokat és azok kapcsolatát is ugyanilyen módon lehet elemezni és egy intenzívebb mozgást megadni feltétel-ként. Számomra a legérdekesebb a sebesség kiszámítása volt. A frame-enkénti sebesség kiszámolásához két frame-ben meghatározott koordinátákra van szükség és a frame-ek futási idejére. A számítási képlet a következő:

$$\frac{\sqrt{((Frame1Y-Frame2Y)^2+(Frame1X-Frame2X)^2)}}{Frame1Idő} \quad (5)$$

Ahol

- Frame1Y – az adott pont első frame-beli Y koordinátája
- Frame2Y – az adott pont második frame-beli Y koordinátája
- Frame1X – az adott pont első frame-beli X koordinátája
- Frame2X – az adott pont második frame-beli X koordinátája
- Frame1Idő – az első frame lefutásának ideje

Mivel a framek ideje meg van szabva ezért csakis akkor lehet eltérés, ha egy frame a megadott határértéknél lassabban fut le, de ezt a program figyelembe veszi. Természetesen a programban nem csak az utolsó 2 frame-ben eltelt változásokat figyeljük, ugyanis ily módon pontatlanságok fordulhatnak elő. Ezen pontatlanságok elkerülése érdekében az utolsó 5 frame-ben megállapított sebességet raktározzuk el és azoknak az átlagát vesszük figyelembe a feltételnél.

1. ábra: A Mediapipe-pal azonosítható pontok elhelyezkedése a testen és kamerakép, rajta a megállapított pontokkal

3. Elért eredmények

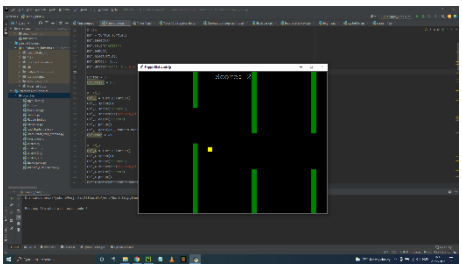
A flappie bird program megvalósítása sikeres volt, bár a madár csak egy sárga kockaként van feltüntetve, de ez nem probléma, hiszen ez a projekt csak egy példa a testmozgással való játék irányításra.

A képelemző program is sikeres lett, pár tized másodpercnyi késéssel megállapítja a sebességet.

A kettő összekapcsolása is teljesült, és bár kicsit lassú a program, de ez nem okoz akadályt a játékban és még mindig élvezhető. A flappie bird játék

újraindítás funkiója nem lett kifejlesztve a projekt során, ezért minden menet után újra kell indítani a programot manuálisan a következő menethez.

Összességében elértem az eredeti célomat: már pár percnyi játék során is komoly fizikai mozgást igényel a játék irányítása. Továbbfejlesztési lehetőségnek az egyéb mozgásformákkal való irányítási módok kidolgozását látom. Ezáltal egy a teljes testet megmozgató, de élvezetes játékot fogunk létrehozni



2. ábra

Vírus szimuláció

Level 40 wizard

Dér Zsombor Zsolt
Felkészítő tanár: Hagymási Gyula
DSZC Mechwart András Gépipari és Informatikai Technikum, 4025 Debrecen, Széchenyi u. 58.

1. Bevezetés

Manapság mindenhol már a koronavírusról hallunk, hogy újabb szigorítások lesznek, vagy közeleg az új hullám. Emiatt fontosnak tartottam, hogy készítsek egy szimulációs programot azzal a céllal, hogy a vírus terjedését látványosan megjelenítsem és a napi adatot naplózzam.

Fő célom az volt, hogy az emberek ne random mozogjanak, hanem legyenek feladataik pl. munkába járás vagy időnként el kelljen menni a boltba, mert elfogyott az élelem otthon.

2. Probléma megoldásának menete

2.1. Pálya generálás

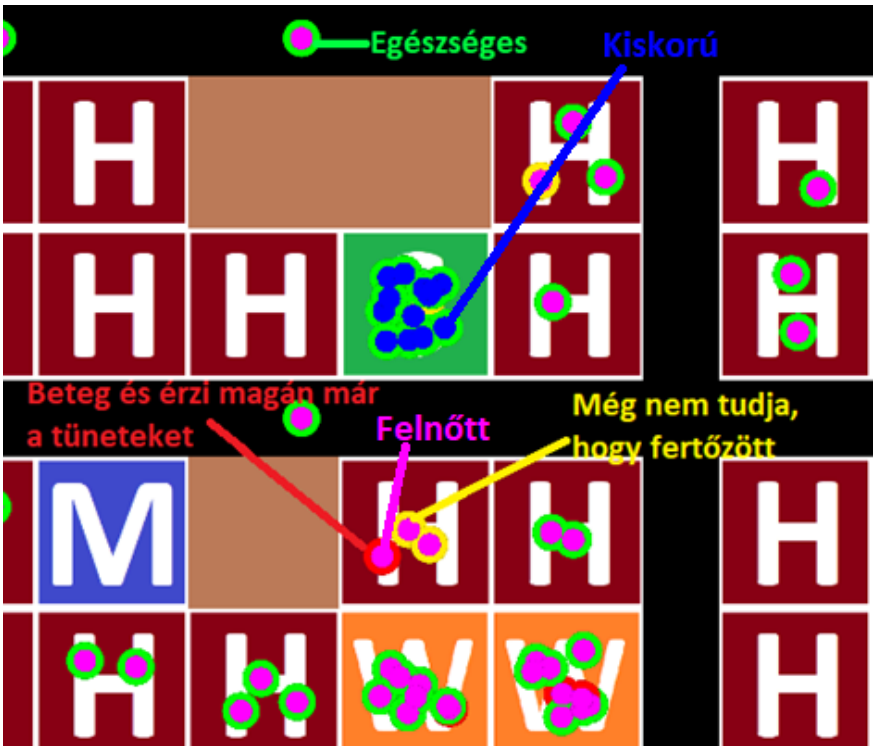
Az első nagy feladat a pálya legenerálása volt. Több épület típuson is gondolkoztam, de végül 4 épület típus mellett maradtam: a ház, ahol laknak majd a családok, bolt ahol vásárolnak, munkahely ahol dolgoznak a felnőttek és az iskola ahova a 18 éven aluliak fognak menni tanulni. (1. ábra)

1. ábra: A lehetséges épületek

2.2. Emberek kezelése

Az emberek megjelenítését úgy oldottam meg, hogy mindenki egy körrel van jelölve. A külső körív színe a jelenlegi fertőzöttséget mutatja (ha zöld akkor egészséges, ha sárga akkor fertőzött, de nem tudja magáról, ha piros akkor pedig fertőzött és tünetes), míg a belső kör azt mutatja, hogy a személy melyik korosztályban van (a kiskorú kékkel van jelölve, míg a felnőtt magentával). (2. ábra) Ezután az emberek mozgását kellett megvalósítani. Ennek a kivitelezéséhez a „greedy” útvonalkereső algoritmus mellett döntöttem, mivel

az sokkal kevesebb erőforrást igényel, mint egy A* algoritmus. A megalkotott útvonalat pedig letároljuk, hogy csak egyszer kelljen kiszámolni és ezzel is CPU időt spóroljunk.



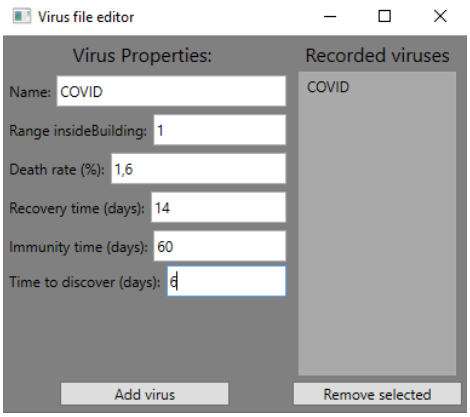
2. ábra az emberek megjelenítése

2.3. **Vírus terjedése**

Számomra a legnagyobb feladat az volt, hogy kitaláljam, hogyan terjedjen a vírus. Végül annál a megoldásnál maradtam, hogy mindig nőni fog egy változó, ha egy fertőzött közelében vannak. A növekedés mértéke függ a helytől és a paraméterektől. Ha a változó elér egy kritikus értéket, akkor az ember elkapja a vírust és ő is hordozó lesz, de még nem tudja magáról, hogy megbetegedett. Majd az idő teltével rájön, hogy ő is elkapta a vírust. Ilyenkor a legtöbb ember otthon marad tudatosan, de vannak kivételek. Ha meggyógyult egy adott ideig immunis lesz a vírusra. A vírus paramétereire készítettem egy beállító programot, amivel újabb vírust tudunk hozzáadni, vagy már meglévőt kitörölni (3. ábra). A program több vírust is tud párhuzamosan kezelni viszont a megjelenítésük, még mindegyiknek ugyanaz.

2.4. Adatok gyűjtése

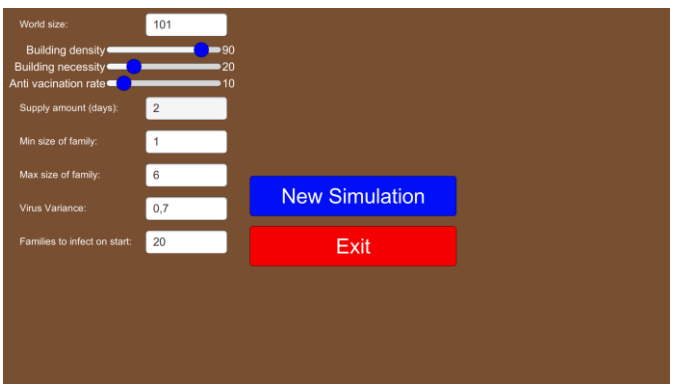
Az adatokat meg tudja nézni a felhasználó futás közben vagy a program naplózza azt napi szinten egy .csv állományba, amit később könnyedén grafikonon ábrázolni is lehet (5.ábra).



3. ábra: Új vírus hozzáadása

2.5. Beállítható paraméterek

A paraméterek bevitelére készítettem egy főmenüt (4. ábra). Itt meg lehet adni a pálya méretét, épületek sűrűségét, a családok létszámát és azt, hogy a szimuláció kezdetén hány család legyen fertőzött és a vírus mennyire térhet el a mediántól. Emellett vannak paraméterek, amiket a program futása közben is lehet állítani, ilyen például a szimuláció sebessége, vagy karantén elrendelése (a település teljes lezárása esetén csak a legfontosabb munkahelyekre mehetnek el az emberek és élelmiszerért a boltba), vagy beállíthatóak a vakcina paraméterei.



4. ábra: A program főmenüje

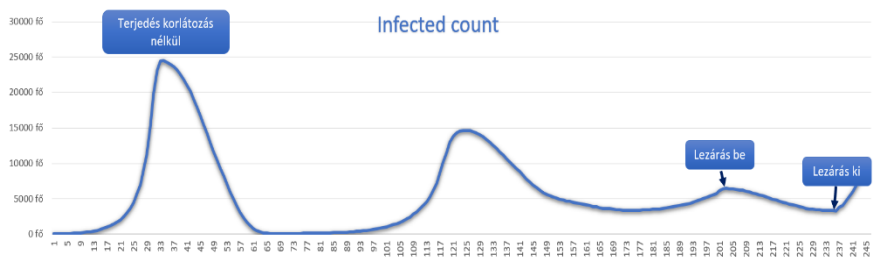
3. Elért eredmények

A program a jelenlegi állapotában stabilan képes szimulálni egy kisebb települést (40-50 ezer fő). Számos paraméterrel lehet testre szabni a szimulációt, ilyen például, hogy az épületek sűrűségének csökkentésével egy ritkábban lakott települést is le tud szimulálni. Az adatokat korosztályokra szétszedi (kiskorú és felnőtt), majd rögzíti annak érdekében, hogy statisztikát lehessen vonni belőle.

4. Továbbfejlesztési ötletek

A program fejlesztése közben sok előre eltervezett plusz funkciót ki kellett hagyni amiatt, mert nem volt időm kivitelezni. Ezeket a verseny után hozzá szeretném adni a programhoz.

- 5 épület típus a kórház. Mivel, ha megbetegednek az emberek nem mindig elég csak otthon maradniuk karanténban sokszor annyira súlyos lesz az állapotuk, hogy ellátásra szorulnak.
- Egy 3. korosztály a nyugdíjasok, akiknél a vírus súlyosabb tüneteket okoz nagyobb százalékban.
- A vírusnak szeretném a megadható paramétereit növelni, például a tünetek, vagy korosztályokra külön paraméter megadása, vagy mutálódásának lehetőségét és annak esélyét.
- A vakcinának meg lehessen adni, hogy mennyi idő alatt alakuljon ki a védettség, mert a jelenlegi kódban egyből kiépül egy adott időre miután megkapták az oltást.
- Az embereknél az is látszódjon, hogy ki immunis jelenleg ne csak az, hogy ki egészséges és ki fertőzött.



5.ábra: Szimuláció eredményének szemléltetése diagramon

Laser Cats

Triangular Frogs

Benei László Barnabás, Lechky Károly Kevin, Pósán Patrik
Felkészítő tanár: Dr. Csapó Gábor
DSZC Brassai Sámuel Műszaki Technikum, 4029 Debrecen, Víztorony u. 3.

1. Bevezetés

Iskolai projektként készült a játék első verziója (1. ábra), amit idővel továbbfejlesztettünk. A feladat az volt, hogy egy olyan játékot kellet csinálni, ami összefüggésben van a lézerekkel és macskákkal. Sokat gondolkoztunk mivel a kettő között nincs sok összefüggés, de végül lett pár ötletünk. A játék Unity-ben készült, viszont a képi elemek megalkotására a Pixel Studio-t használtuk. A zenét pedig FL Studio-ban készítettük el.

2. A játék menete:

Két játékosnak együtt kell működniük és ki kell jutniuk egy „gyárból". A játékosok egy közöttük összekötött lézer segítségével szenzorokat tudnak aktiválni, amik különböző ajtókat nyitnak. Fontos, hogy a két játékos közötti lézer aktív legyen, amikor érintkezik a szenzorral, ugyanis, ha valami blokkolja az útját megszakad. Egyszerű az irányítás billentyűzettel:

Játékos 1: W, A, S, D
Játékos 2: ↑, ↓, →, ← (*Nyilak*)

3. Probléma megoldásának menete

Próbáltuk minél jobban kihasználni a Unity által adott lehetőségeket, ezért a karakterek és a lézer a layer-t nézik, hogy tudják, hogy mivel ütközött és arra hogyan hasson. A pályák könnyű létrehozásához és rajzolásához kihasználtuk a map palette funkciót.

Az első verzióban (1. ábra) rögtön találtunk hibákat, amit megpróbáltunk minél hamarabb kijavítani:

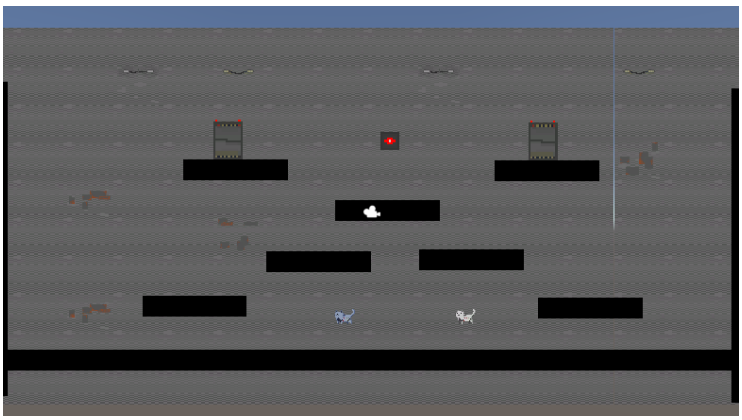
(1) Több zene indul el egyszerre,
(2) Animációs hibák,
(3) Menü beállítások,
(4) Pálya teljesítése utáni gomb nem működik,
(5) Háttér beállítások

(1) A zenéken különböző beállításokat kellett végezni, hogy ne egyszerre induljanak el, illetve megfelelő sorrendben.
(2) Játékos 1 séta animációja nem játszódott le.
(2) Játékos 2 ugrás animációja nem játszódott le.
(3) A főmenü hangbeállításai gomb helyett „csuszkával" állíthatóak.

(4) Miután a játékosok teljesítettek egy pályát nem tudtak a „Next” gombra kattintani, nem volt megfelelően elhelyezve az érzékelője.
(5) Az alap hátterek el voltak csúszva, valamint több helyen nem csatlakoztak egymáshoz.

A jelenlegi játékban már ezek a hibák nem találhatóak meg (2. ábra), minden problémára találtunk valamiféle megoldást.

3.1. Ábrák



1. ábra: A játék legelső játszható verziója.

2. ábra: A jelenlegi játék egyik pályája.

4. Elért eredmények

A játék végleges formájának elérése előtt több szempontból teszteltük, illetve megkértük osztálytársainkat, hogy próbálják ki a játékunkat. A visszajelzéseket figyelembe véve kisebb hibákat kellet javítani (a hibákat

fentebb részletezzük). Hárman készítettük, Benei László Barnabásnak és Pósán Patriknak a programozás az erőssége, Lechky Károly Kevin a részletesen kidolgozott elemeket kedveli a játékokban. A feladat felosztása ennek függvényében történt.

Patrik a játékosok mozgását és animálását csinálta, László a lézerek működését, fő menüvel és a hangokkal foglalkozott, míg Kevin a dizájnt és a pályákat tervezte meg.

A játék korábbi verziójával részt vettünk az iskolánk versenyén, melyen első helyezést értünk el. Ez örömmel töltött el bennünket és motivált minket arra, hogy a játékunkat javítva beküldjük erre a versenyre is.

A pályamunka összeállítása során sok tapasztalatot nyertünk, valamint megismerkedtünk az informatikához kapcsolódó számos eszköz használatával. Ilyenek például:

- Unity mint munkakörnyezet
- A Unity objektum-orientált programozási nyelve (általánosságban a Unity API)
- Pixelart mint használt grafikai stílus
- Unity animátora
- FL Studio alapjai
- Mindhármunkat érdekelnek a számítógépes játékok, azok felépítése, működése, így a játékot továbbfejlesztve a közösség számára elérhetővé is szeretnénk tenni a későbbiekben.
- A pályamunka letölthető az alábbi linkek bármelyikéről:
- Google Drive[1]
- pCloud[2]

[1] https://drive.google.com/file/d/1bw7cPSKpXvHhbU9vw3FsjWrOTFlYjHBb/view?usp=sharing

[2] https://e1.pcloud.link/publink/show?code=XZo5KpZsXolrYrcLfjqEVxbauC794pcEk7X

CRVER

Dark Themes Only

Béri Márk, Krajczár Dávid, Verno Massimo
Felkészítő tanár: Dlusztus Péter
Pécsi Janus Pannonius Gimnázium, 7621 Pécs, Mária utca 2-4.

1. Bevezetés

Csapatunk tagjai az általános iskolában ismerkedtek meg különböző programozási nyelvekkel (C#, Unity, Python). A gimnázium 3D és Innováció szakkörében a 3D tervezés és nyomtatás technológiája, az oktatásfejlesztés a VR-ban s újdonságként a kriptográfia került figyelmünk középpontjába. Ennek tanulmányozása során született meg az ötlet, hogy egy komplex kriptográfiai programot hozzunk létre.

A **CRVER** nevű programunkkal a csapat elsődleges célja egy mindenki által hozzáférhető, nyílt forráskódú és hibamentesen működő kriptográfiai program létrehozása volt.

A kriptográfia egy informatikai tudományág, amely szövegek vagy adatok titkosításával és visszafejtésével foglalkozik. A kriptográfiában nagy szerepe van a matematikának (Pl.: számelmélet, algebra), hiszen a legtöbb titkosítási eljárás matematikai számítások útján hajtható végre. A kriptográfiának óriási a jelentősége olyan szervezetek (pl.: katonai, állami, diplomáciai szervezetek, bankok) működésében, amelyek jelentős és „érzékeny" információkat kezelnek, melyek kiszivárgása nagy kárt okozhat a szervezeteknek, illetve ügyfeleiknek.

2. Probléma megoldásának menete

2.1. Eszközök

A programunkkal öt gyakran használt titkosítási módszert (Caesar, ROT13, Vigenére, Enigma, RSA) kívántunk elérhetővé és könnyebben felhasználhatóvá tenni. Ehhez a Python programozási nyelvet és a Tkinter könyvtárat használtuk fel. A program elkészítésének megkönnyítésére a Visual Studio Code és a PyCharm Community kódszerkesztő programokat használtuk.

2.2. Program leírása

A különböző kódolások programrészeinek elkészítését a csapat tagjai felosztották egymás között. Ezután készítettünk egy felhasználói felületet, amelyből az összes részprogram könnyen elindítható. A programok képesek üzenetek kódolására és dekódolására. (Feltörésre nincs lehetőség.) A szoftver az angol ABC-t használja, a speciális karaktereket (PL.: ~, !, &, #, @) nem ismeri. A felhasználói felület egyszerű, könnyen lehet navigálni a menüpontok között. A program asztali operációs rendszereken futtatható.

2.3. Kriptográfiai módszerek

Caesar

A legegyszerűbb titkosírási módszer. A ROT13 és a Vigenére titkosítás alapja. Minden egyes betűt egy tőle adott távolságra (*n*) lévő betűvel kell helyettesíteni.

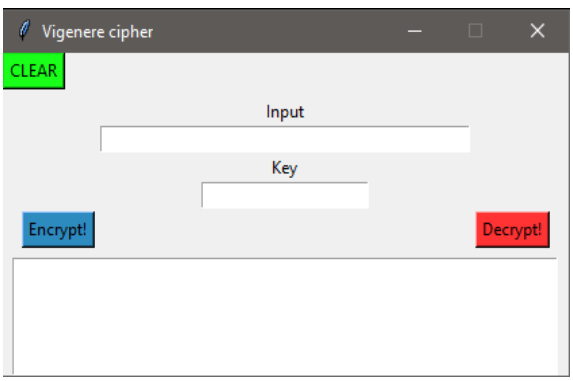
ROT13

A Caesar kódolás egyik változata. A betűket a 13 hellyel arrébb található betűkkel helyettesítjük.

Vigenére

Ehhez a titkosírási módszerhez a felhasználónak szüksége van a Vigenére táblázatra, amely Caesar kódok sorozatából áll. A táblázat minden egyes sora az angol ABC, de a karakterek a sor száma szerint el vannak tolva. A kódolás és a dekódolás folyamatához szükség van egy tetszőlegesen választott „kulcsra”. A kulcs egy tetszőleges hosszúságú szöveg, amit az üzenet közvetítője határoz meg, viszont a vevőnek is ismernie kell.

A kódolás első lépéseként a kulcsot kell „meghosszabbítani”. Ez azt jelenti, hogy a kulcsot addig kell ismételni (egymás után leírni), amíg annak hosszúsága nem egyenlő az üzenet hosszával (mivel szóköz, illetve speciális karakterek nem szerepelnek a táblázatban, ezért ezeket figyelmen kívül hagyhatjuk). A titkosított üzenet karaktereit az üzenet adott karakterének az oszlopa és a kulcs betűjének a sorában találjuk. A dekódoláshoz szintén szükség van a kulcsra. Ilyenkor a titkosítási folyamatot visszafelé kell megismételni.



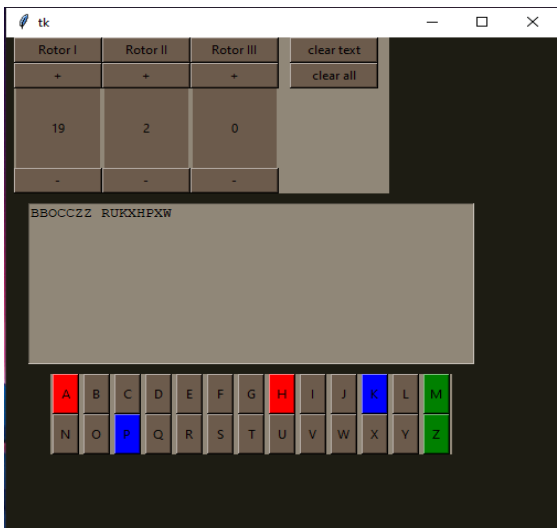
1. ábra: A Vigenére részprogram

Enigma

Az Enigma egy forgótárcsás, elektromechanikus berendezés, mely az 1920-as években eredetileg kereskedelmi célokra készült. A németek saját fejlesztésű Wehrmacht-Enigma katonai rejtjelező modelljüket használták a

második világháború idején, melynek kódját tudósok együttműködése révén Alan Turing brit matematikus törte fel.

Az Enigma korai változata 3 forgatható és egy fordító tárcsából áll, ezeken kívül még egy kapocstáblával is rendelkezik. A kódolást ez egyedi tárcsák forgása hozza létre. A titkosításnak még egy szintjét a kapocstáblán összepárosítható betűk biztosítják. Minden gombnyomás után az első tárcsa fordul egyet, majd 26 fordulás után a második tárcsa is fordul. Az Enigma program elkészítése során a kapocstábla képernyőn való hibamentes megjelenítése problémákat okozott, de nagyobb szintű kódolási technikával megoldható volt.



2. ábra: Az Enigma részprogram

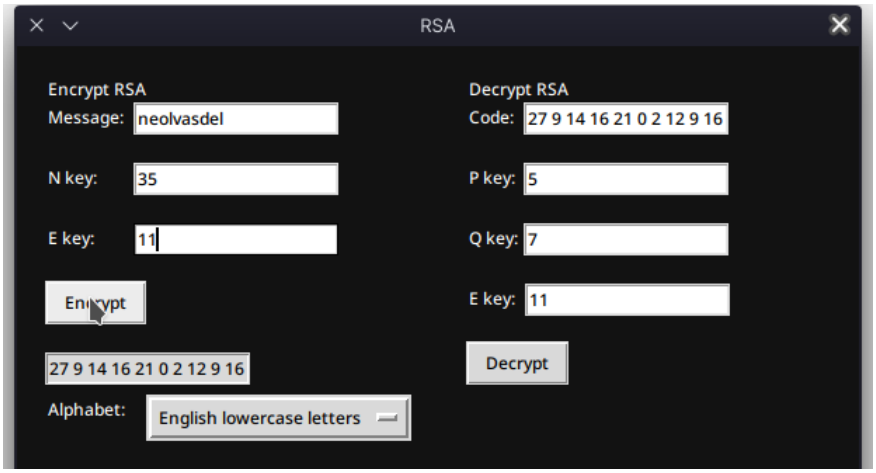
RSA

Az RSA egyirányú üzenetküldésre használható, nyílt kulcsú titkosítás. Az üzenetet fogadó fél rendelkezik titkos és nyilvános kulcsokkal. Ebből a küldő fél csak a nyilvános kulcsot ismeri, amivel az üzenetet küldés előtt titkosíthatja. Ezt a titkosítást innentől csak a címzett tudja visszafejteni a titkos kulcsával.

A fogadó félnek először ki kell választania két prímszámot (p és q), amelyeket összeszoroz (a szorzat n), majd a program kiszámolja a $(p - 1)(q - 1)$ szorzatot (ez $\varphi(n)$). Ezután a fogadó fél kiválaszt egy e számot, ami $\varphi(n)$ relatív prímje. A program ezekből kiszámol egy d számot, ami a legkisebb pozitív egész, aminél d és e szorzatának $\varphi(n)$-nel való osztási maradéka 1. A fogadó a p, q, d számokat titokban tartja, az e, n számokat nyilvánosságra hozza.

Üzenet titkosításához az üzenetet számokká kell alakítani és azokat titkosítani. A kódolt szám az eredeti betű számértékének e-edik hatványának n-el való osztási maradéka.

A fogadott üzenet visszafejtéséhez ismerni kell a küldő fél *e* és *n* számát. Minden kódolt szám eredeti betűjének számértéke a kódolt szám *d*-edik hatványának *n*-el való osztási maradéka. Innen a betűk visszafejthetők. Az RSA titkosítást megvalósító részprogram a 3. ábrán látható.



3. ábra: Az RSA részprogram

3. Elért eredmények

A program elkészítése bővítette a csapat tudását a kriptográfia, a Python programozási nyelv és a felhasználói felületek programozásának terén.

A projekttel egy működőképes kriptográfiai programot hoztunk létre, amely többféle titkosítási módszert felhasználva képes üzeneteket titkosítani és visszafejteni.

A programunkkal nem mellékesen az is célunk volt, hogy rávilágítsunk a tantárgyköziség fontosságára az oktatásban. A szoftver alkalmas szemléltetésre, problémafelvetésre és -megoldásra történelem-, informatika-, matematika- és nyelvtanórákon.

Mivel programunk nyílt forráskódú, így saját igényre szabva szabadon továbbfejleszthető már ismert, illetve új fejlesztésű kódolási technikákkal, s így például saját biztonsági rendszerek is létrehozhatók. Ezáltal a kriptográfia hétköznapi felhasználása is lehetővé válik.

Icicle keretrendszer

IceyLeagons

Tóth Tamás, Kissik Márton

Felkészítő tanárok: Rozgonyi-Borus Ferenc, Motesiczki Ottó

SZTE Vántus István Gyakorló Zeneművészeti Szakgimnázium,

6722 Szeged, Tisza Lajos krt. 79-81

Esztergomi Dobó Katalin Gimnázium, 2500 Esztergom, Bánomi út 9.

1. Mi az az Icicle?

Az Icicle, egy innovatív, Minecraft szerver-oldali plugin készítésére szolgáló, a Spring által ihletett Java/Kotlin keretrendszer. A fejlesztéséhez a szikrát az adta, hogy a Minecraft fejlesztés támogatására nem igazán található alkalmas keretrendszer, és ha vannak is újrafelhasználható eljárások, azok nem egy központi tárházban találhatók, hanem egy-egy specifikus könyvtárban, több különálló projektben érhetők el. A keretrendszer elkészítése egy hullámvasút volt, eredetileg csak néhány hasznos funkciót tartalmazó eljárásgyűjteményből fejlődött ki. A keretrendszert háromszor írtuk újra, a műszakilag megfelelő és legelegánsabb változatot keresve. Keretrendszerünk modulokba van rendezve, amelyek – helytakarékosság céljából – csakis akkor kerülnek letöltésre, betöltésre, valamint frissítésre, ha tényleg szükség van rájuk. Az Icicle-nek a szíve az ún. „***core***" modul, mely tartalmazza a saját „***bean***" és „***dependency injection***" megoldásainkat, a különböző annotáció kezelőket és a keretrendszerhez nélkülözhetetlen alapfunkciókat. A projekt felépítése alapján a „***core***"-t egy szabadon választott környezetbe kell implementálni az interfészek segítségével, így utat nyitva a szerteágazó felhasználásra. Igyekeztünk minden ponton testreszabhatóvá és könnyen használhatóvá tenni szoftverünket, ezáltal biztosítva annak jövőbeli felhasználási lehetőségét is.

2. Probléma megoldásának menete

Kitűzött célunk az egyszerűség, testreszabhatóság, és a kis méret volt. Bármennyire is gondoltuk a feladatot egyszerűnek, nem bizonyult annak – ezt az újraírások száma is jól mutatja. Az első nagy feladat a „***bean***"-ek és a „***dependency injection***" megvalósítása volt. Mivel nem akartunk sok külső szoftvert felhasználni (azok nagy mérete, és lassúsága miatt), így megírtuk a saját megoldásainkat.

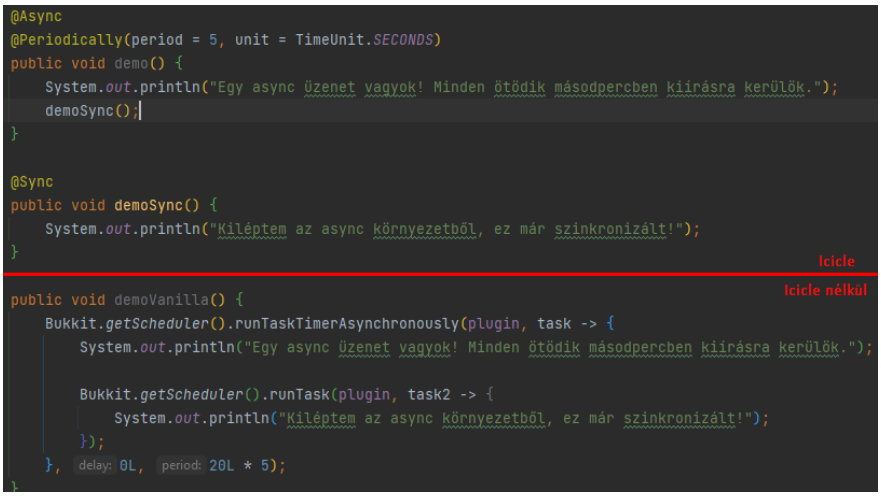
2.1. Babok és függőségek (dependency injection)

Rendszerünkben minden osztály, amelynek az annotációs fája – ehhez is saját megoldást kellett keresnünk, mivel a Java alapból nem kezeli az „*annotációk annotációit*" – tartalmazza az ún. @AutoCreate annotációt, automatikusan létre lesz hozva, és az osztály függőségei is injektálásra kerülnek. Ehhez a

rendszerhez került hozzá a körkörös függőség tesztelése, az osztályok proxizása, és az osztályok futás közbeni létrehozása (az utóbbi kettőhöz egy ***ByteBuddy*** nevű könyvtárat használtunk).

2.2. Osztályok proxyzása, az AOP jelenléte

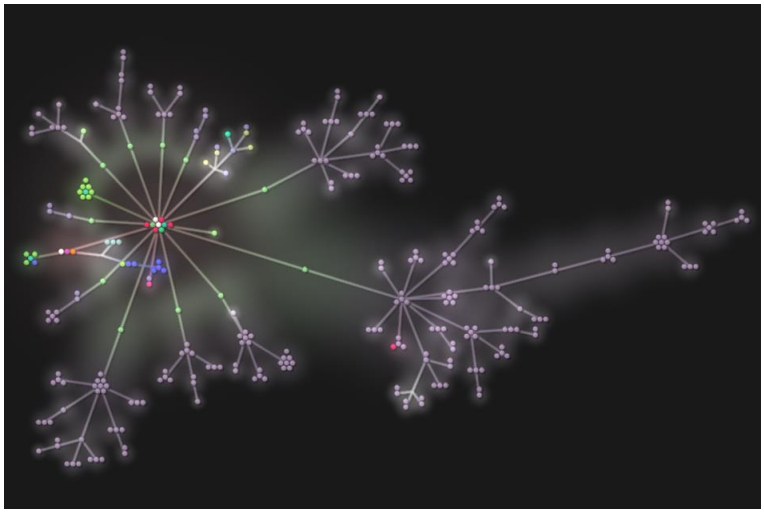
A proxizás az *aspektus-orientált* programozás egyik fő elve. Segítségével képesek vagyunk egy metódus lefutását vagy egy osztály viselkedését alterálni úgy, hogy az eredeti kód elé, közé, illetőleg utána saját logikát injektálunk, ami akár még az eredeti kód lefutását is megakadályozhatja, vagy módosíthatja annak paramétereit, illetőleg a környezetét. Ez a megoldás lehetőséget ad letisztultabb, látszólag „natív" kódok írására, amelyre példát az 1. ábra mutat: a program egy üzenetet ír ki minden 5. másodpercben, egyet szinkronizálva, egyet pedig aszinkron módon. (Míg az Icicle alapút nem, addig a szokványos módszert betöltéskor egyszer manuális meg kell hívni).



1. ábra: Proxy bemutatása

2.3. Felkészülés a jövőre

A jövőben, a saját és mások munkájának megkönnyítése érdekében, a saját annotációk (ebbe beletartozik az @AutoCreate, valamint a saját dependency injection/autowiring annotációk) kezelésére egyszerű módot kínálunk: pár sorban hozzá lehet adni a keretrendszerhez új funkciókat, az annotáció-kezelők segítségével. Továbbá, mivel a Kotlin programnyelv manapság egyre népszerűbb, annak támogatása is bekerült a projektbe. Annak ellenére, hogy a Kotlin és a Java képes integrálódni egymással, a „***bean***"-ek létrehozása során mégis számos plusz funkciót kellett megvalósítanunk, hogy megfelelően kezeljük ezt a feltörekvő nyelvet is.



2. ábra: A projekt kiterjedése 2021. december 31-én (Gource-ábra)

2.4. Alapvető funkciók és kiegészítő modulok

A „***core***” tartalmaz egy konfigurációkezelőt, és egy lokalizációs rendszert is a többnyelvű projektek támogatására. Utóbbi egy kisméretű, egyszerűen bővíthető „script nyelvet” - hasonló az Excelben található függvények formátumához - is használ. Példa erre, egy névelő kereső, ami a ***targy*** paraméter elé a megfelelő névelőt írja ki:
{IF(SW(targy, ‘a','á','o','ó','u','ú','e','é','i','í','ö','ő','ü','ű'), ‘Az’, ‘A’)} {targy} az enyém.

Ezen a ponton kezdtünk el több különböző feladaton dolgozni párhuzamosan. Így készült el, a Minecraft-tól még távolálló Gradle bővítményünk, saját szerializáció könyvtárunk, modul betöltőnk. Majd követték őket a Minecraft modulok, vagyis a parancsrendszer, az NMS (**N**et.**M**inecraft.**S**erver) könyvtár, valamint a protokoll könyvtár. Készül továbbá egy GUI és adatbázis kezelő alrendszer is. Természetesen ezeknek az eszközöknek a támogatásában nem mi vagyunk az elsők, viszont mi egy rendkívül egyszerű formára törekedtünk, minél kevesebb kód használatával, ami a szoftver méretét csökkenti és a teljesítményét jelentősen növeli.

A modulok közül sok időt fektettünk a parancsrendszerre, hiszen a legtöbb esetben, ezen keresztül használunk egy-egy Minecraft plugint. Ez a rendszer követi az eddigi mintákat, vagyis az egyszerű és gyors használatot annotációkkal, illetőleg az erőteljes testreszabhatóság lehetőségével. Jelen esetben az ún. middleware-ekkel a parancs beírása és a végrehajtása között lehet beavatkozni az eseményekbe, a paraméter feldolgozókkal pedig a parancs, illetőleg metódus paramétereit konvertáljuk a megfelelő objektumra egy szöveges bemenetet használva.

```java
@CommandManager("test")
public class TestCommand {

    @PlayerOnly
    @Command(value = "four", returnsTranslationKey = false)
    public String asd2(@CommandSender Player player, String arg, @Optional Player argOpt) {
        return "Executed2 sender: " + player.getName() + " | arg: " + arg + " opt: " + argOpt;
    }
}
```

3. ábra: Példa egy parancs készítésére (9 sor, a szokásos 50+ helyett)

3. Elért eredmények

Keretrendszerünk készítésére és a precizitásra rengeteg időt fordítottunk, amit a jövőbeli projektjeink fejlesztése során kapunk majd vissza, hiszen a hosszú hagyományos kód helyett csak pár sort kell majd írjunk. Ha további funkcióra lesz szükség, akkor csak írunk egy kezelőt a saját annotációkra alapozva. Bármi ami a keretrendszerben rendelkezésre áll, a gyors és hatékony programozást támogatja, ami a Gradle és a jövőben az IntelliJ IDEA bővítményünk segítségével még produktívvá tehető.

Projektünk nyílt forráskódú, jelenleg is fejlesztés alatt áll (még béta). Honlapja megtalálható a https://icicle.iceyleagons.net/ címen. Utoljára egy kis érdekesség: projektünk 2021. december 31-én 18 792 sorból áll.

Bole

Bole Team

Bokor Apor, Juhász Bence

Felkészítő tanár: Galántha Endre

Lánczos Kornél Gimnázium, 8000 Székesfehérvár, Budai út 43

1. Bevezetés

Egy csoport számára gyakori probléma a közös döntéshozatal. Példának okáért a hétvégi mozizásnál a film kiválasztása. Ezen problémákat általában szavazással szokták megoldani. A csoportok nagy része ilyenkor a legegyszerűbb, leggyakoribb módon szavaz: mindenki szavaz egy jelöltre (jelen esetben egy filmre) és a legtöbb szavazatot kapott jelölt nyer. Ezt nevezik plurality vote-nak. A szavazás végén általánosságban elmondható, hogy a csoport nagy része nem elégedett.

Nem közismert, hogy ez az egyik legrosszabb szavazási rendszer. Kettőnél több jelölt esetén előfordulhat, hogy nem a csoport egésze által legjobban kedvelt győzedelmeskedik.

Azonban nem kell elkeseredni, mivel rengeteg alternatív szavazórendszer létezik a választások lefolytatására. Léteznek különböző szolgáltatások is, amik sokféle szavazórendszeren keresztül képesek szavazásokat gyorsan és hatékonyan lebonyolítani.

Felmerül a kérdés: Hogyan válasszuk ki a számunkra legmegfelelőbbet?

Természetesen kellő utánajárással mindegyikről találhatunk leírásokat, amelyek alapján megismerhetjük erősségeiket és gyengeségeiket, de ez nem biztos, hogy elég a döntéshozatalhoz. Ráadásul legtöbbször komplikáltan vannak megfogalmazva, tehát megértésük sok időt vesz igénybe.

Ezért határoztuk el, hogy csinálunk egy weboldalt, amiben ezeket a szavazórendszereket a felhasználó interaktív módon megismerheti.

2. Probléma megoldásának menete

A felmerült problémák a következők voltak:

- A különböző szavazórendszerek implementálása
- A weblap reszponzívvá tétele
- A weblap dizájnja
- A szavazórendszerek vizualizációja
- Taktikus szavazás implementálása és vizualizációja

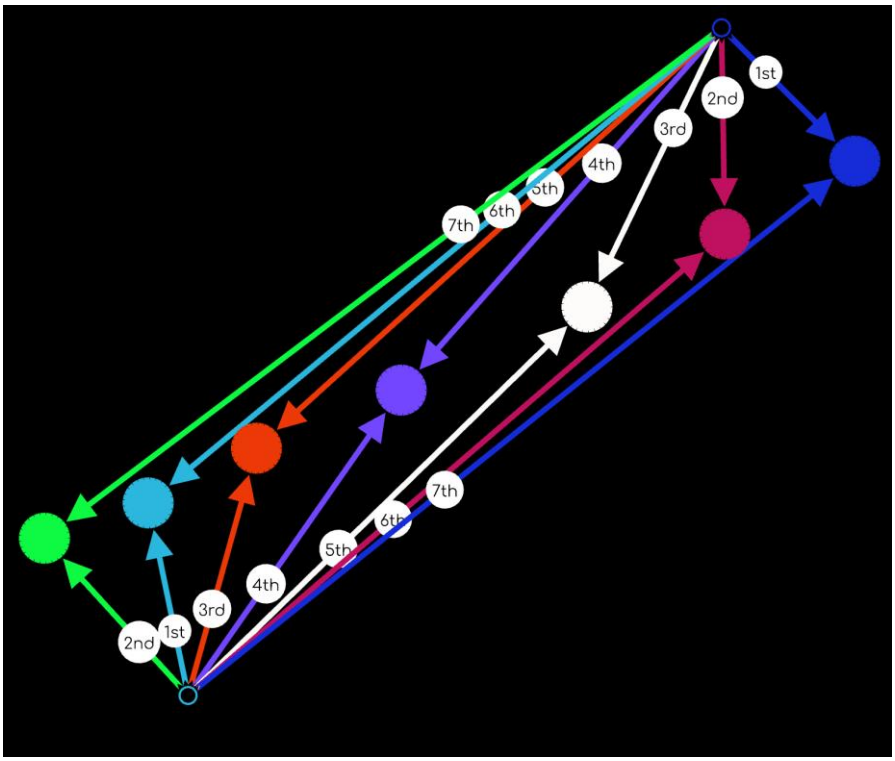
2.1. A vizualizáció alapelve

Ha vizualizálni akarunk különböző szavazórendszereket, először találnunk kell egy módot a szavazók és a jelöltek véleményének megjelenítésére. Ezt az

úgynevezett kétdimenziós véleménysíkkal oldottuk meg. A szimulációnkban a szavazó pozíciója az x- és y- tengelyeken jelképezi a véleményét kettő, egymástól teljesen független témával kapcsolatban. Ez a jelöltekre is igaz, tehát minél közelebb van a szavazó a jelölthöz, annál hasonlóbban gondolkoznak.

Minden szavazónak érdekében áll, hogy az a jelölt kerüljön megválasztásra, akinek a véleménye a leghasonlóbb az övéhez. Vagyis az a jelölt, aki a legközelebb van hozzá a véleménysíkon (1. ábra).

A vizualizációnkban a jelölteket nagy kitöltött körökkel, a szavazókat kis körökkel ábrázoltuk.

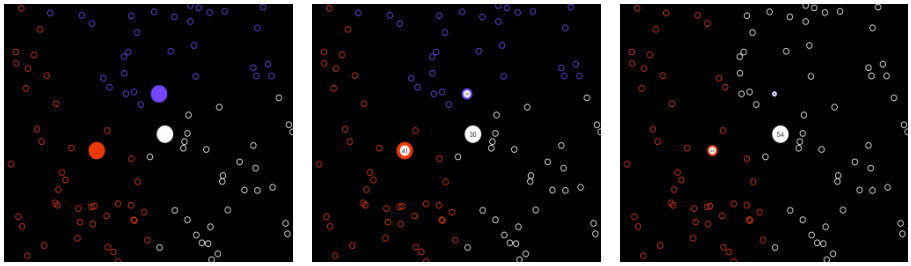


1. ábra: kettő szavazó véleménye 7 jelölttel kapcsolatban

2.2. Lépésről lépésre

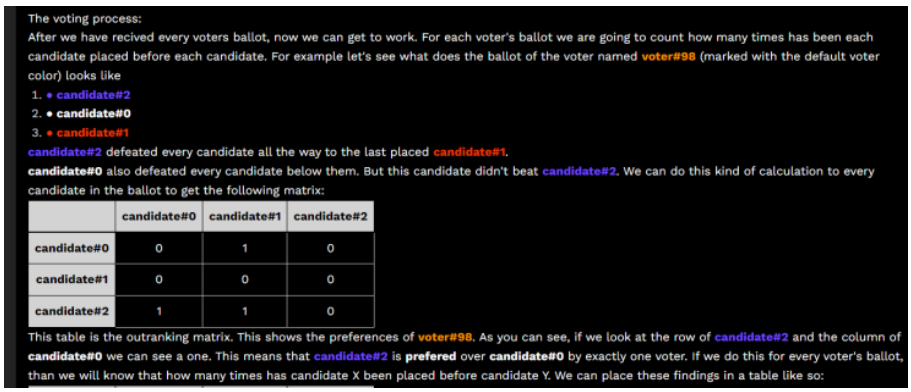
Nem minden szavazórendszer olyan egyszerű, hogy csak összeszámoljuk a szavazatokat, majd egy egyszerű feltétel alapján kiválasztjuk a győztest. Van ahol ciklusosan lépésről lépésre zárunk ki jelölteket, majd újraszámoljuk a szavazatokat (Instant Runoff, Coombs, Contingent vote). Ezért csináltunk egy külön kis részt, ahol akár lépésről lépésre is vizualizálhatjuk ezeket a rendszereket. Minden lépéssel nem csak a magyarázó szöveg változik, hanem

maga a vizualizációban is változnak a jelöltek méretei, újraszínezzük a szavazókat, van, amikor nyilakat rajzolunk stb.



2. ábra: Az Instant Runoff rendszer vizualizálása 3 lépésben

A magyarázó szövegben előszeretettel használtunk színesen kiemelt szövegrészeket az egyszerűbb megértés érdekében (3. ábra).



The voting process:

After we have recived every voters ballot, now we can get to work. For each voter's ballot we are going to count how many times has been each candidate placed before each candidate. For example let's see what does the ballot of the voter named **voter#98** (marked with the default voter color) looks like

1. • **candidate#2**
2. • **candidate#0**
3. • **candidate#1**

candidate#2 defeated every candidate all the way to the last placed **candidate#1**.

candidate#0 also defeated every candidate below them. But this candidate didn't beat **candidate#2**. We can do this kind of calculation to every candidate in the ballot to get the following matrix:

	candidate#0	candidate#1	candidate#2
candidate#0	0	1	0
candidate#1	0	0	0
candidate#2	1	1	0

This table is the outranking matrix. This shows the preferences of **voter#98**. As you can see, if we look at the row of **candidate#2** and the column of **candidate#0** we can see a one. This means that **candidate#2** is **prefered** over **candidate#0** by exactly one voter. If we do this for every voter's ballot, than we will know that how many times has candidate X been placed before candidate Y. We can place these findings in a table like so:

3. ábra: A Copeland rendszer első lépésének szöveges magyarázata részlet

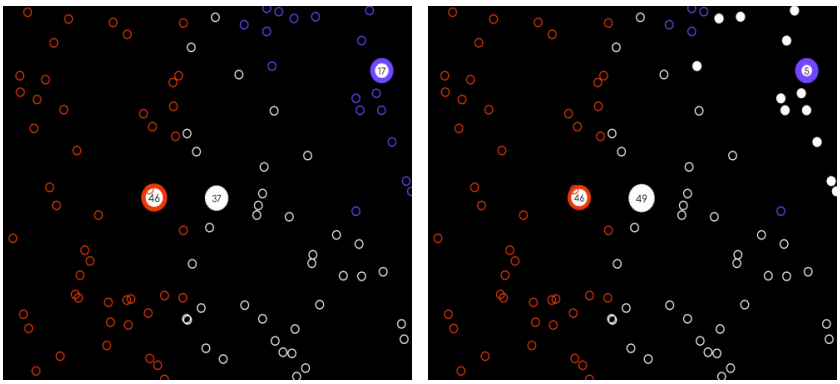
2.3. Taktikus szavazás

Persze a szavazóknak érdekében áll, hogy az általuk legjobban kedvelt jelölt nyerjen. Ezért néhányan, ha tehetik, okosan szavaznak, és nem feltétlen mondják meg őszinte véleményüket, például plularity szavazás közben (4. ábra).

Viszont több esetben ahhoz, hogy okosan tudjanak szavazni, meg kell állapítaniuk, hogy melyek azok a jelöltek, akik közül a nyertes kikerül, és hogy melyek azok a jelöltek, akik biztosan nem nyerhetnek. Máshogy mondva előre meg kell becsülni a választások kimenetelét. Egy jelenlegi becslőalgoritmus a következő kritériumoknak felel meg:

- Könnyű megérteni.
- Nem túlságosan pontos: Mi értelme van választást tartani, ha már előtte is meg tudjuk mondani, hogy ki nyer?

- Valóságban is elképzelhető, hogy a szavazók ilyen vagy hasonló stratégiát használnak a becslésre.



4. ábra: A taktikus szavazók elintézik, hogy plurális szavazásban ne piros nyerjen

3. Elért eredmények

Megalkottuk a weboldalt: getbole.com, mellyel megpróbáljuk egyszerűvé tenni az emberek számára, hogy megértsék a különböző szavazó rendszerek előnyeit és hátrányait, hogy majd ki tudják választani, hogy az ő esetükben melyik a leghasznosabb. Ezek közül a legfontosabb és legjelentősebb az egyedileg megoldott több mint 10 szavazórendszer interaktív vizualizációja. Emellett (ahol van) a rendszerhez tartozó stratégikus szavazást is szimuláljuk és vizualizáljuk.

Ezzel konkrétan egy játszóteret biztosítunk a felhasználónak, aki így szabadon játszadozhat a választás változóival. Ily módon nem csak olvas róluk, hanem fel is fedezi őket és saját maga képes így szinte megtapasztalni az erősségeiket és gyengeségeiket. Ráadásul mindezt egy kidolgozott, reszponzív környezetben tehetik meg.

Peaceful Nature

Peaceful Nature Team

Szabó Levente

Felkészítő tanár: Tari Balázs

Nagyboldogasszony Római Katolikus Gimnázium, 7400 Kaposvár, Zárda utca 2.

1. Bevezetés

Sokat szoktam játszani különböző túlélős játékokkal, de mindegyiknek vannak hiányosságai, illetve olyan részei, ami nekem nem tetszik, vagy amit én máshogy csináltam volna.

Az egyik legnagyobb hiányosság szinte az összes játékban az az NPC-k (Nem játékos által irányított karakterek) programja. A legtöbb játékban alig fordítanak rájuk figyelmet, pedig rendkívül fel lehetne dobni velük a játékmenetet. Belegondoltam, hogy mennyivel élvezetesebb lenne úgy játszani, ha közben az NPC-k is élik a saját életüket és képesek kapcsolatba lépni a játékossal, képesek kommunikálni vele, illetve egymással és a játékos cselekedetei befolyásolják az NPC-k viselkedését.

További hiányosság az NPC-k működésével kapcsolatban, hogy ők általában sokkal gyengébbek, mint a játékos, ezáltal alá is vannak rendelve neki. Éppen ezért szerintem úgy lenne izgalmas a játék, ha az NPC-k mindenre képesek lennének amire a játékos, és az egyetlen különbség köztük az lenne, hogy a játékost ember irányítja.

Így tehát eldöntöttem, hogy megvalósítom ezt a játékot, amiben az NPC-ké a főszerep.

2. Probléma megoldásának menete

A céllom az volt, hogy létre hozzak egy élvezhető, egyszerű túlélős játékot, amiben az NPC-k kifejezetten fejlettek és képesek fejlődni, illetve szaporodni.

A végeredményt elérni nehezebb volt, mint vártam, mivel amellett, hogy egy teljes játékot létre kellett hoznom, amiben szabadon lehet fejlődni; létre kellett hoznom egy működő, egyszerűbb „mesterséges intelligenciát”.

Továbbá az is nehézséget jelentett, hogy az előbbieket össze kellett vonni, hogy a „mesterséges intelligencia” megfelelően tudjon érvényesülni az általam létrehozott virtuális térben.

2.1. A játék

Magának a játéknak (NPC-k nélkül) az elkészítése is több időbe került, mint azt vártam. Mivel egy külső nézetes játékot terveztem, ezért a kamera mozgásra külön oda kellet figyelnem. Végül úgy oldottam meg, hogy az egérrel szabadon tudjuk mozgatni a kamerát, ezáltal minden szögből lehet élvezni a játékot.

Ennek a kameramozgásnak köszönhetően az irányítást egy kicsit komplexebb volt megcsinálni, mint hittem, de végül könnyedén meg tudtam oldani.

A következő nagy probléma a barkácsolással (vagyis a nyersanyagok összerakásával új eszköz, épületi elem stb. szerzése) volt. Mivel ezt a problémát magamtól nem tudtam megoldani, ezért YouTube-on utánanéztem, és így már sikerült leküzdeni ezt a nehézséget.

További nehézséget jelentett az építkezés megoldása, mivel bezavart a kameramozgás, illetve a domborzat is. A program elkészítéséhez Unity-t használtam, És így tudtam használni a beépített funkciókat, és egy kis keresgetés után megtaláltam a megoldást erre a problémára is.

2.2. Az NPC-k

A Legnagyobb bonyodalmat az NPC-k elkészítése jelentette. Mivel eddig egyetlen mesterséges intelligenciát sem csináltam, ezért ez számomra rendkívül újszerű és nehéz része volt a feladatnak. Sok helyen utánanéztem, illetve több embertől is kértem segítséget, és miután összegyűjtöttem a kellő tudást, elkezdtem megírni magát az intelligenciát.

Szerencsére szinte mindent meg tudtam oldani if-ekkel és különböző ciklusokkal. A gondot az jelentette, hogy ezeknek az NPC-knek sok mindennel kell kölcsönhatásba lépniük, sok mindenre kell reagálniuk, ezért a programjuk meglehetősen hosszú és bonyolult lett.

Problémát jelentett még az is, hogy az NPC-knek össze kell tudniuk működni. Nem azt akartam, hogy egymástól teljesen külön éljenek, hanem inkább, hogy együttműködjenek és segítsék egymást. Az NPC-k a játék kezdetén kis (3-4 fős) csoportokban születnek, és az volt a cél, hogy ezek a csoportok együtt maradjanak. Ennek érdekében az ilyen csoportokban nem csak az egyének gondolkoznak, hanem maga a csoport is rendelkezik egy közös intelligenciával, ami segíti, hogy a csoporton belüli egyedek összedolgozzanak.

Az utolsó megoldásra szoruló probléma a játékos, illetve az NPC-k kapcsolata volt. Úgy szerettem volna ezt megoldani, hogy például, ha a játékos bántja az NPC-ket, akkor ők egy idő után elzavarják, ha pedig segíti őket, akkor ők is viszont segítenek neki és befogadják maguk közé.

További funkciója az NPC-knek, hogy képesek szaporodni. Szerencsére ezzel nem volt semmi nagyobb gond.

3. Elért eredmények

Mivel ez elött nem csináltam sok játékot, ezért ez a projekt kihívás volt számomra.

A játék készítése során sikerült létrehoznom egy olyan karaktert, ami teljesen szabadon irányítható, és a speciális kameramozgásnak köszönhetően különböző szemszögekből és távolságokból is rá lehet látni az emberre, ezáltal ki lehet választani a leg előnyösebb nézőpontot.

A Játék készítése során minden 3D-s modellt én szerkesztettem úgynevezett „Low poly” grafikával (pl.: fákat, bokrokat, tábortüzet, szerszámokat stb.). Azért választottam ezt a stílust, mert pont annyira részletes, hogy gyorsan és egyszerűen lehet ilyen objektumokat megszerkeszteni, viszont az egyszerűsége nem megy a kinézet rovására. Külön karaktert ad a modelleknek.

Annak érdekében, hogy a játék élvezhetőbb és változékonyabb legyen, készítettem többféle fát, sziklát, érceket, szerszámokat és ültethető növényeket. A nyersanyagokat úgy csináltam, hogy ha a játékos (vagy az NPC) nem figyel rá, akkor el fognak fogyni. Ez ellen lehet terméseket gyűjteni és új fákat ültetni.

Mivel túl békés volt a játék, ezért adtam hozzá szörnyeket, amik éjjel jönnek, ezáltal arra késztetve a felhasználót, hogy folyamatosan készenlétben álljon. Erre azért volt szükség, hogy egy kicsit felpörgesse a játék menetét.

És végül létre hoztam egy egyszerűbb „mesterséges Intelligenciát”, ami képes szabadon mozogni a 3D-s térben, kikerüli az akadályokat és ugyan úgy él a játékban, mint a játékos. Ugyan úgy gyűjt ételt, nyersanyagokat, építkezik, termeszt élelmiszert és harcol, mint a játékos.

Együttműködő partnereink

Együttműködő partnereink

ExxonMobil Hungary Kft.
www.exxonmobil.hu

SZTE Természettudományi és Informatikai Kar
www.ttik.hu

Knorr-Bremse Vasúti Jármű Rendszerek Budapest
rail.knorr-bremse.com/en/hu/

Deutsche Telekom IT Solutions
www.deutschetelekomitsolutions.hu

Creativ_IT
creativit.hu

Bitotron Kft.
bitotron.com

EPAM Systems Kft.
www.epam.com

3i Fejlesztő és Szolgáltató Kft.
3i.hu

Morgan Stanley Magyarország Elemző Kft.
www.morganstanley.com

Antavo Informatikai Kft.
antavo.com

TEConcept Hungary Kft.
www.teconcept.hu

Optin Kft.
optin.hu

Digital Kft.
www.digital.co.hu

Lufthansa Systems Hungária Kft.
www.lhsystems.com

CAS Software Kft.
cas-software.hu

SZTE TTIK Informatikai Intézet
www.inf.u-szeged.hu

Kiadta: SZTE TTIK Informatikai Intézet

Készítette: Dr. Németh Gábor

Design: Dr. Németh Gábor

Nyomda: Innovariant Nyomdaipari Kft.

Készült: 100 példányban

Együttműködő partnereink:

ExxonMobil

SZTE TTIK 100

KNORR-BREMSE

SZTE TTIK
INFORMATIKAI INTÉZET

T

epam

3i

DEUTSCHE TELEKOM IT SOLUTIONS

Morgan Stanley

creativ_it

TEConcept
Hungary Kft.

CAS SOFTWARE

bitotron

OPTIN Digital

antavo

Lufthansa Systems