# Eszköz tervezés

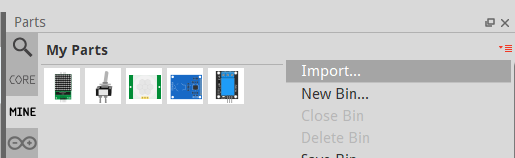
A projekt irányelve, hogy egy bemutató alkalom során az eszközt kipróbáló gyerekeknek megismertessük az Arduino világát. Ehhez olyan kapcsolások kellenek, amik nem túl bonyolultak, viszont látványos eredményt produkálnak egy megfelelően megírt kóddal. A következő oldalakon ismertetni szeretném a különböző részegységek felépítését. Igyekeztem egy olyan sorrendet felállítani, ami a legkevesebb alkatrészt/összeköttetést használttól, ami fizikailag a legegyszerűbben összeállítható, valamint a hozzá tartozó algoritmus minimális programozói ismeretekkel is megérthetőtől halad az egyre összetettebbek felé.

Első sorban ismertetni szeretném a projekt során felhasznált szoftvereket, valamint a projektek közös elemeit. A kapcsolások és az algoritmusok megtervezése is alegységenként lett csoportosítva a dokumentumban.

## Felhasznált szoftverek

### Fritzing

A kapcsolási rajzokat a *Fritzing* nevű, nyílt forráskódú áramkörtervező programmal készítettem, ami bejelentkezés után ingyenesen letölthető és rendkívül egyszerűen használható. Rengeteg Arduino-hoz is használható tervezőprogramot találhatunk az interneten, azonban ami a Fritzinget kiemeli a többi közül, hogy mi magunk is készíthetünk egyedi komponenseket [9], valamint használhatjuk a hatalmas felhasználóibázisnak köszönhetően létrejött rengeteg egyedi alkatrész egyikét. Ilyenkor nincs más dolgunk, mint letölteni a megfelelő *fzpz* kiterjesztésű fájlt és beimportálni azt.



2.1. ábra Saját beimportált alkatrészek, valamint új importálás lehetősége a Fritzing nevű szoftverben.

A szakdolgozat keretein belül nem készítettem saját alkatrészt, viszont felhasználtam több, mások által készített komponenst, melyek a következőek voltak:

* 8x8 LED modul MAX7219-vel [10]
* HC-SR501 PIR modul [11]
* KY-019 5V Relé Modul [12]
* Szolenoid szelep [13]
* Címezhető WS2812 RGB LED szalag [14]

### Tinkercad

A példának hozott kapcsolásokat a Tinkercad online áramtervezőjével valósítottam meg. Azért választottam ezt a példák bemutatására, mivel gyorsabb elérni, mint a Fritzing-et, az egyszerű kapcsolások gyorsabban összerakhatók vele, illetve a leglényegesebb, hogy a szimulációk látványosabbak.

### ClickCharts

Mielőtt nekikezdtem a megvalósításnak, szerettem volna vizuálisan felvázolni, hogy mit is szeretnék a programtól elvárni. Ennek okán minden alegységhez készítettem folyamatábrákat, amik nagyvonalakban felvázolják, hogy a program működése közben az egyes állapotokból mik következnek, illetve, ha több lehetőség van a továbbhaladásra, akkor milyen feltételek mellett fog bekövetkezni a kívánt eredmény.

A flowchartokat a *ClickCharts* [17] nevű szoftverrel terveztem meg, amelynek elérhető egy ingyenes verziója, ami főleg a kényelmi funkciók limitálásában tér el a fizetős verziótól.

### Arduino Software IDE

A forráskód megírásához az Arduino IDE-t használtam, aminek a segítségével kód írható az Arduino kompatibilis alaplapokra. Lefordítja a kódot, szintaktikai ellenőrzést végez, valamint csatlakoztatás után lehetőség van feltölteni azt a kívánt eszközre. Az IDE használatával egyszerű a külső könyvtárak importálása, lehetőség van a soros monitort figyelni, automatikus formázásra ad lehetőséget, illetve még rengeteg kényelmi funkciót biztosít a felhasználónak.

## Közös alkatrészek

A játékdoboz lényege, hogy moduláris legyen, részegységeit külön-külön is használni lehessen, illetve együtt egy egészet alkosson. Így fontos volt, hogy az egységek saját tápellátással rendelkezzenek. Ebből következik, hogy az összes kapcsoláson található egy 9V-os elem, egy tolókapcsolóval, ami rá lett kötve az Arduino megfelelő lábaira, ezzel biztosítva a hordozhatóságot, valamint lehetőség van bármikor ki- és bekapcsolni az eszközöket. Az elem, valamint a kapcsoló mindegyik részegységben megtalálható, ezért ezekre nem térek ki külön.

A tápellátáson kívül közös elem az Arduino Nano, mely az összeszerelés során is a négy egységből háromban megtalálható. Azért válaszottam a Nano méretet, mivel a pin kiosztása elégséges a megvalósításokhoz, méretéből adódóan könnyebben integrálható a projektbe, valamint árát tekintve is jobb választás. A negyedikben egy Arduino Mega lapnak megfelelő *Geekcreit Mega* [15] nevű panel található, ennek oka, hogy a kezdőkészlet [16], amiből az egész projekt elindult, ezt tartalmazta, így logikus volt ezt is felhasználni.

## Modulok megtervezése

## Memória játék

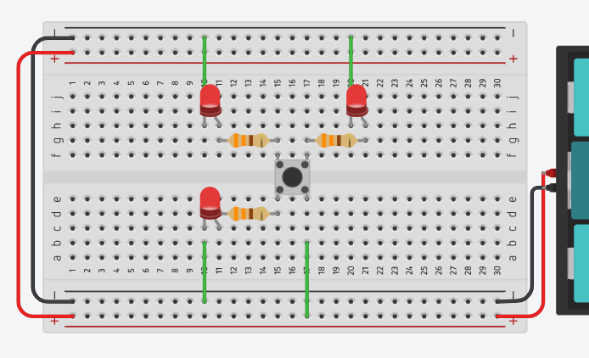
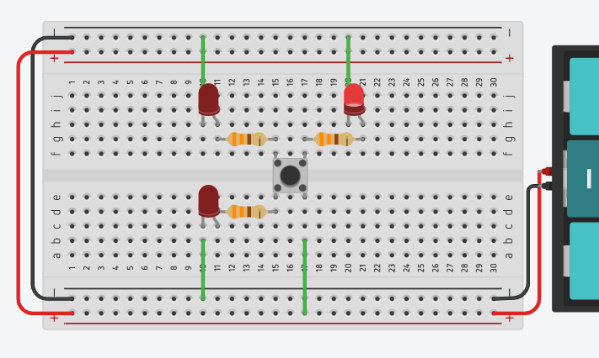
Összeszerelést és algoritmust tekintve is talán kijelenthető, hogy a legegyszerűbben megvalósítható projektről van szó. Kezdők számára tökéletes választás lehet. A memóriajáték lényege, hogy az Arduino mutat egy feladványt, amit a játékosnak el kell tudni ismételni. Amennyiben helyesen ismételt, folytatódik a játék, a feladvány kiegészül még egy taggal. Ez addig ismétlődik, amíg a játékos rosszul nem ismétel. Ilyenkor a játéknak vége és visszaáll az alapállapot addig, amíg új játék nem kezdődik.

### Memória játék hardver tervezése

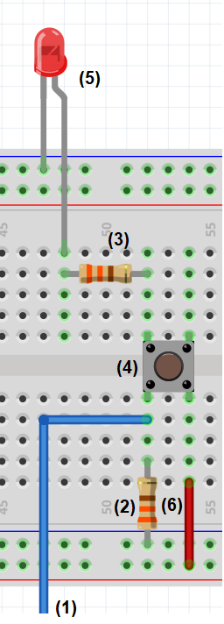
Felhasznált alkatrészek:

* 5 db LED – sárga, piros, zöld, kék, fehér
* 5 db nyomógomb
* 5 db 330Ω ellenállás
* 5 db 10kΩ ellenállás

A kapcsolás előtt célszerű átlátni, hogyan is működik a nyomógomb. A működését egy szemléltető ábra segítségével a legkönnyebb megérteni. Felengedett állapotban a két szemközti láb, míg lenyomott állapotban az összes láb össze van kötve. Erre látható példa az alábbi ábrán.



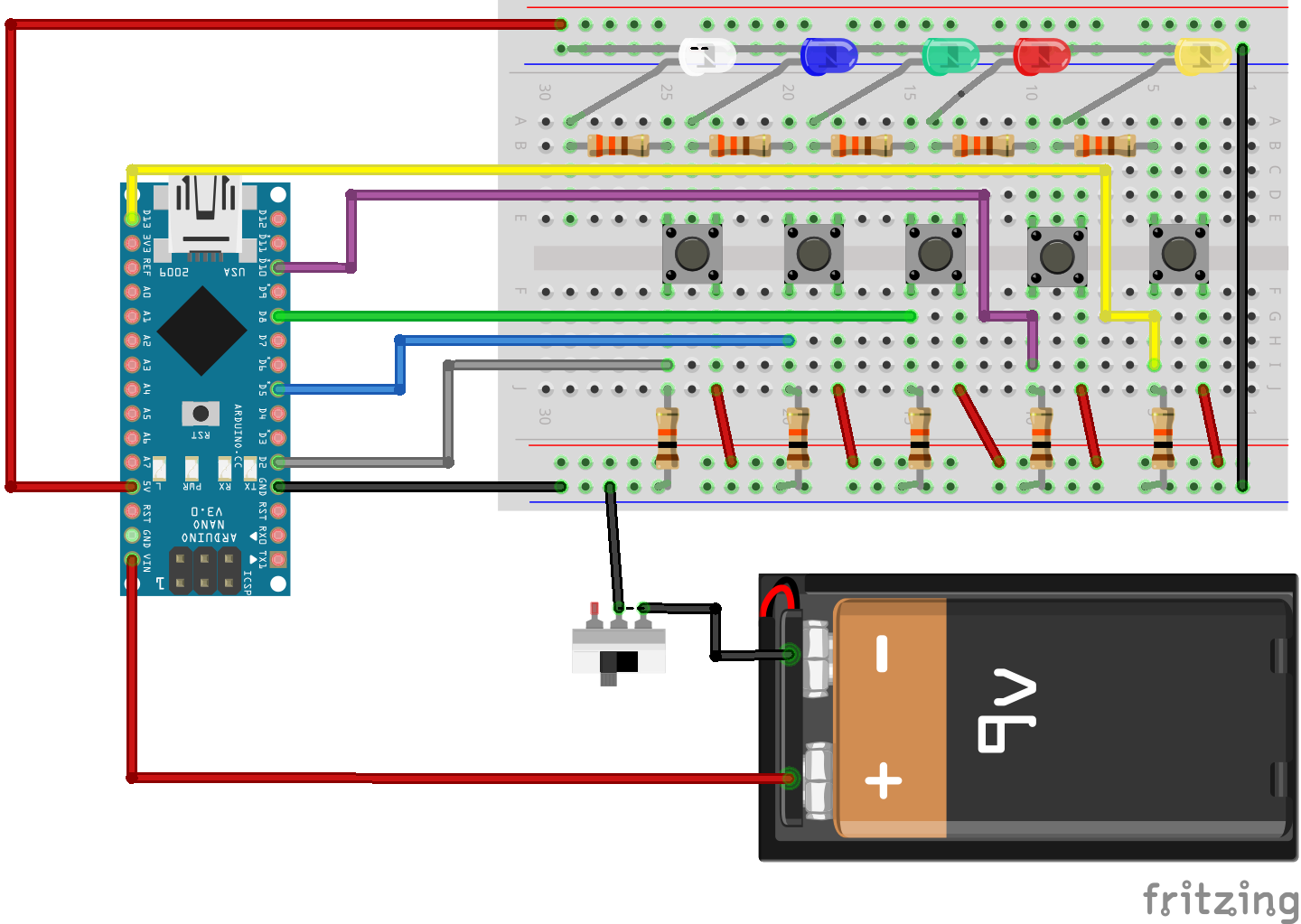
A projekt szabad pinek számáig bővíthető, de célszerű minimum két LED-et alkalmazni. Az összeszerelés egyszerűsége abban rejlik, hogy a következő kapcsolást kell megvalósítani minden LED esetében, amit használni szeretnénk:



A kapcsolás legnyilvánvalóbb része, hogy az ábrán látható (4) -es gombot lenyomva a LED összeköttetésbe kerül az feszültségforrással a (6) -os vezetéken keresztül, ezért az felvillan a gomb lenyomásának idejére. Ennek a funkciója, hogy jelezze a játékosnak, hogy valóban megnyomta a gombot. A LED-ekkel természetesen sorba van kötve egy-egy 330Ω ellenállással. Ez az előtét-ellenállás a LED-en átfolyó áram korlátozására szolgál, megakadályozza a LED-et kiégető túláramot.

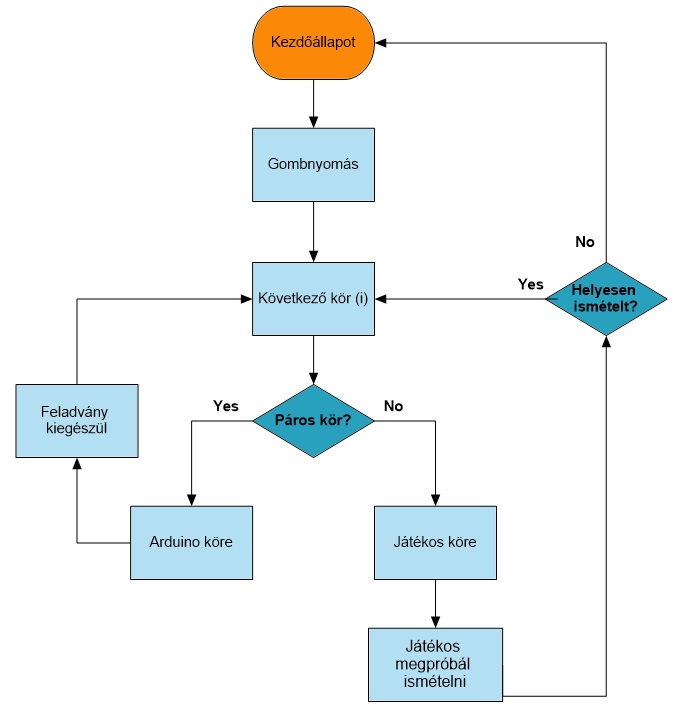
A kapcsolás kevésbé egyértelmű része a gomb bal alsó lábánál látható. Itt az Arduino egyik digitális lába van bekötve (1) a gombra, valamint egy lehúzó ellenállsra (2). Később látható a megtervezett algoritmusban, hogy a megadott pinek egyszerre fognak ki- és bemenetként szolgálni. OUTPUT módban gombnyomáshoz hasonlóan az Arduino logikai magas jelet küld a LED-re, ezért az felvillan. Ami érdekesebb, hogy INPUT módban a pinek. nagy impedanciájú állapotban vannak. A bemeneti pinek rendkívül kis igénybevételt jelentenek az általuk mintavételezett áramkörnek, ami megfelel egy 100MΩ-os soros ellenállásnak a pin előtt. Ez azt jelenti, hogy nagyon kevés áramra van szükség ahhoz, hogy a bemeneti pin egyik állapotból a másikba kerüljön. Ezzel együtt ez azt is jelenti, hogy könnyen zajt vehet fel a környezetből, kéretlen eseteket is gombnyomásnak érzékelhet. Ennek az elkerülésére kell egy 10kΩ-os lehúzó ellenállást (2) építeni az áramkörbe. Ennek segítségével a zavarok elkerülhetőek.

Megvalósításnál, amire érdemes figyelni, az a megfelelő sorrend. Célszerű az Arduino választott lábaival sorban haladni és aszerint megvalósítani a kapcsolást. Ami még problémát okozhat, hogy a LED-ek megfelelő irányba nézzenek, különben azok egészen egyszerűen nem fognak világítani.



2.3. ábra Memória játék kapcsolási rajza

### Memória játék algoritmus tervezése



3.2. ábra Memória játék folyamatábra

Láthatjuk, hogy több állapot van, de ezek igen egyszerűek. A program megvalósításához több feltételt kell kezelnünk és egy kicsit át kell látni a tömböket, illetve azok bejárását. Kezdő projekt lévén for ciklusokból, valamint if - else feltételes elágazásokból fog állni a program nagy része.

A programhoz jónéhány változóra szükség van, ezek a következők:



Mielőtt a loop() függvény tartalmára térnénk, érdemes végig nézni néhány metódust, ami a későbbiekben sokat fog egyszerűsíteni a dolgunkon.



A setPinMode(byte mode) függvénynek a segítségével egyszerre állíthatjuk be, hogy INPUT, vagy OUTPUT módban szeretnénk használni a pin-eket, amik a pins[] tömbben lettek felsorolva .



writeAllPins(byte val) függvénynek köszönhetően az összes LED egyszerre lesz írható, így azok be- vagy kikapcsolhatóak egyszerre.



Az előzőn alapszik a flashLeds(short frequency) függvény, ami arra szolgál, hogy a frequency paraméterben megkapott frekvencián villantsa fel az összes LED-et, ötször egymás után. Ezzel fogjuk jelezni, hogy a játékos hibázott és a játék véget ért.



A playSolution() végigmegy a jelenleg tárolt feladványon, kiírja, valamint felvillantja a megfelelő LED-eket.

Ezeken kívül van még egy ResetGame() függvény, ami a változók alapértékeire való visszaállításra szolgál.

A játék végén az endGame() függvény hívódik meg, ami meghívja a flashLeds(50) függvényt 50-es frekvenciával, valamint a villogás után egy másodperccel meghívja a ResetGame()-et.

A loop() függvényben két fő állapotot kell megkülönböztetni. Az egyik, amikor az Arduino, a másik pedig amikor a játékos következik.

Először vizsgáljuk meg, mi történik, amikor páros kör van, az isPlayersTurn = false, azaz az Arduino van soron.



A setPinMode() segítségével OUTPUT módba állítjuk a pineket, ugyanis ebben az esetben a LED-eket szeretnénk használni. A randomSeed() inicializálja a pszeudo-véletlenszám-generátort, és a véletlen sorozat egy tetszőleges pontjáról indítja. Ez a szekvencia, bár nagyon hosszú és véletlenszerű, mindig ugyanaz. Ahhoz, hogy a random() által generált értékek sorozata későbbi futtatásakor eltérő legyen, akkor a randomSeed() segítségével inicializáljuk a véletlenszám-generátort egy viszonylag véletlenszerű, például esetünkben az analogRead(A0) olvasásával [19]. Így a solution[currentLength] = pins[random(0,numberOfPins)] sor segítségével már egy ténylegesen véletlen értéket adhatunk hozzá a feladványunkat tároló tömb végére. A hozzáadás, illetve feladvány hosszának növelése után a playSolution() segítségével megjelenik a játékos számára a feladvány a LED-eken. Ha ez megtörtént, az isPlayersTurn igazra vált, ezzel jelezve, hogy a játékos következik. Ezt követően az inputTime változóba elmentésre kerül a kör befejezésének ideje a millis() függvény segítségével, ami visszaadja, hogy hány milliszekundum telt el azóta, hogy az Arduino elkezdte futtatni az aktuális programot. [20]

A következő lehetőség, hogy a játékos van soron. Ebben az esetben, mivel a gombok vannak használatban a setPinMode() segítségével INPUT-ra állítjuk a megadott pineket.

A legelső ellenőrzés az lesz, hogy a játékos kifutott-e a gondolkodási időből. 

Amennyiben a játékos több ideig gondolkozott, mint a megengedett, vesztett.

Ha a megadott időn belül döntött, akkor a következő feltétel vizsgálat az lesz, hogy elengedte-e a gombot.



Az elvárt választ a solution[] tömb legvégéről kell kivenni. Ezek után végig kell menni az összes INPUT pinen, és megnézni, hogy van-e olyan aktív pin, ami nem az elvárt megoldás. Ha van, akkor az azt jelenti, hogy a játékos rossz gombot nyomott. Az isGameOver true értéket vesz fel, a játéknak vége.



A következő ellenőrzés megnézi, hogy valóban azt nyomtuk-e meg amit vártunk, valamint ellenőrizzük, hogy nem történik folyamatos gombnyomás. Itt az inputTime felveszi a millis() függvény által felvett értéket, hogy a következő ellenőrzéskor figyelembe tudjuk venni a gombnyomások között eltelt időt. Ebben a kódrészletben növekszik az inputCount, valamint az isButtonPressed igaz értéket vesz fel, hogy biztosan ne ragadjon be a programunk és ne fusson véletlenül se rossz ágra.



A legutolsó rész a gomb felengedése utáni ellenőrzéseket hivatott elvégezni. Abban az esetben, ha erre a részre fut a kód, az isButtonPressed változó hamis értéket vesz fel. Ezután két ellenőrzés fog következni. Amennyiben az isGameOver változó igaz, azaz a játék során rosszul ismételt a játékos az endGame() szcenárió fog lejátszódni. Helyes ismétlés esetén, ha annyi gombnyomás történt, mint amennyi az adott körben elvárt lett volna, akkor a játékos köre befejeződött, újra az Arduino következik.

## Zárszerkezet

A zárszerkezet lényege, hogy bevezetést nyújtson az Arduinok hétköznapi használatába. A modul lényege, hogy egy közönséges fadobozt tesz zárhatóvá. Kinyitásához a megfelelő azonosítóval rendelkező RFID kártyára van szükség. Abban az esetben, ha nincs jogosultságunk, a projektben használt programozható LED szalagon piros színnel jelenik meg animáció, illetve hangjelzés követi. Érvényes beléptetővel zöld, más mintázatú fényjáték jelenik meg a szalagon, ezzel egyidőben a dobozt lezáró szolenoid zár kinyílik, a fedő nyitható megadott ideig, majd visszazár.

### Zárszerkezet hardver tervezése

A zárszerkezet projekt kitűnik a többi közül. A hozzá felhasznált alkatrészek között több olyan is szerepel, ami egy Arduino kezdőkészletben nem található meg, viszont elengedhetetlen az alapműködéshez.

Felhasznált alkatrészek:

* RFID szenzor
* Relé modul
* Szolenoid zár
* 4 x 18650 elemtartó, 18650 akkumulátor cellákkal
* WS2812B címezhető RGBW LED szalag
* Piezo hangszóró

A projekt lelkét az RFID szenzor adja. Ennél kimondottan figyelni kell, hogy nem szabad - sok más alkatrésszel ellentétben - az 5V-os feszültségforrásra csatlakoztatni, mivel a szenzor üzemi feszültsége 3.3V, ennél magasabb az eszköz meghibásodását eredményezi.

A bekötés során az RFID.h könyvtár leírását követtem, ami Arduino Mega típusú alaplapokra is kitért. Ez pontosan megmondja, hogy az SCK, MOSI, MISO melyik digitális lábra legyenek kötve. Az SDA és RST pin lehet általunk definiált, választható helyen.

|  |  |
| --- | --- |
| **MFRC522 RFID szenzor** | |
| **Modul pin** | **Bekötése** |
| *SDA* | Arduino D9-es lábára kötve |
| *SCK* | Arduino D52-es lábára kötve |
| *MOSI* | Arduino D51-es lábára kötve |
| *MISO* | Arduino D50-es lábára kötve |
| *IRQ* | - |
| *GND* | Földre kötve |
| *RST* | Arduino D8-as lábára kötve |
| *3.3V* | 3.3V-ra kötve |

Az SDA pin az SPI-kommunikációban a chip engedélyező pin. Ez akkor kapja a jelet, amikor az Arduinonak SPI-kommunikációt kell végrehajtania. Az SCK az órajel küldésére szolgál. A MOSI és MISO az adatvonalak. Az MOSI a masterből, azaz jelen esetben az Arduinoból, a slave, azaz az RFID olvasó felé, az MISO pedig a slave-ből a master felé továbbítja az adatokat. Ezen az adatvonalon keresztül küldi el az érzékelt kártyák adatait. A készülék alvó üzemmódba kapcsolhat, hogy energiát takarítson meg. Ebben az esetben az IRQ segít felébreszteni. Azonban ezt a projektben nem használtam fel. Az RST a reset pin. Hiba esetén alaphelyzetbe állítja az eszközt, amennyiben az adott ideig nem válaszol.

A szolenoid zár 12V-os feszültséggel tud megfelelően operálni. Szükségünk van egy nagyobb feszültségű külső áramforrásra, ami közvetlen bekötve kisüthetné az Arduinot. A relé az összeköttetés az Arduino és a zár között.

A relék olyan kapcsolók, amelyek elektromechanikusan vagy elektronikusan nyitják és zárják az áramköröket. A relék egy elektromos áramkört egy másik áramkör érintkezőinek kinyitásával és bezárásával vezérelnek. Ha egy relé érintkezője normál esetben nyitott (NO), akkor kontakt van, amikor a relé nincs feszültség alatt. Ha egy reléérintkező normál esetben zárt (NC), akkor egy zárt kontakt van, amikor a relé nincs feszültség alatt. Mindkét esetben az érintkezők állapotát az érintkezőkre adott elektromos áram változtatja meg.

|  |  |
| --- | --- |
| **Relé modul** | |
| **Modul pin** | **Bekötése** |
| *GND* | Földre kötve |
| *VCC* | Tápfeszültségre kötve |
| *IN* | Arduino D3-as lábára kötve |
| *NO* | Elemek pozitív lábára kötve |
| *COM* | Szolenoid zár pozitív lábára kötve |
| *NC* | - |

A relék COM, azaz közös csatlakozója a relé mozgó része. Amikor a relé ki van kapcsolva, a COMMON az NC csatlakozóhoz van csatlakoztatva. A relé NO csatlakozója nincs csatlakoztatva, amíg a relé be nem kapcsol. Amikor a relé bekapcsol, a COMMON az NC-ről NO-ra vált.

A szolenoid zárat nagyon egyszerű bekötni, a pozitív lábán kapja a 12V-os feszültséget, ami hatására a zár nyelve behúzódik. Esetemben a tápellátás a reléből érkezik.

|  |  |
| --- | --- |
| **Szolenoid zár** | |
| **Modul pin** | **Bekötése** |
| *Pozitív* | Relé **COM** csatlakozójára kötve |
| *Negatív* | 12V-os feszültségforrás negatív lábára kötve. |

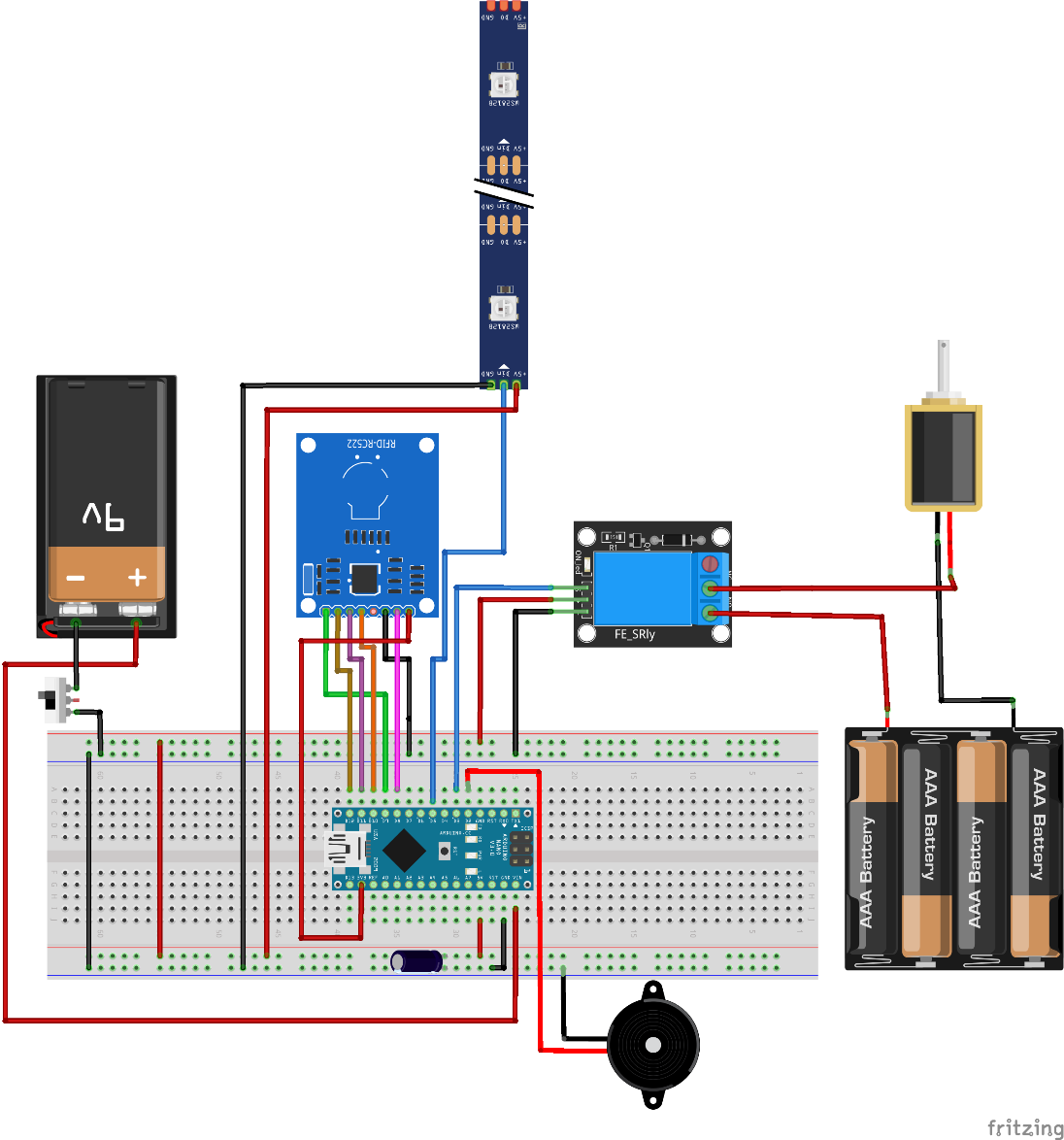
|  |  |
| --- | --- |
| **12V feszültségforrás** | |
| **Modul pin** | **Bekötése** |
| *Pozitív* | Relé **NO** csatlakozójára kötve |
| *Negatív* | Szolenoid zár negatív lábára kötve. |

A hangjelzésként szolgáló piezo hangszóró működéséhez piezokristályt használ, amely feszültség hatására megváltoztatja alakját. Egy membránnak való ütközéssel hang generálódik. A hang frekvenciája a feszültség szabályzásával megoldható, erre beépített függvény is rendelkezésre áll.

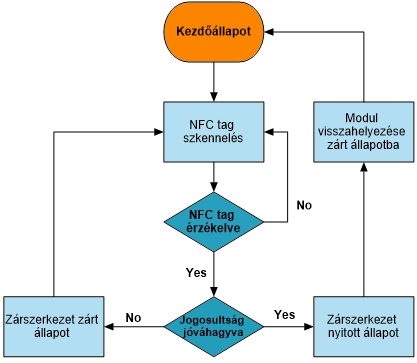
|  |  |
| --- | --- |
| **Piezo hangszóró** | |
| **Modul pin** | **Bekötése** |
| *Negatív láb* | Földre kötve |
| *Pozitív láb* | Arduino D2-es lábára kötve |

A programozható LED szalag nagy előnye, hogy bármelyik LED után elvágható, esetleg újra összeköthető, megtoldható. A használatához 5V-os feszültség alá kell helyezni. A DI pinen keresztül érkezik adat. Ez jöhet az Arduinoból, vagy egy másik LED DO, azaz digitális kimenetéből.

|  |  |
| --- | --- |
| **WS2812B LED szalag** | |
| **Modul pin** | **Bekötése** |
| *VCC* | Tápfeszültségre kötve |
| *DIN* | Arduino **D3**-as lábára kötve. |
| *GND* | Földre kötve. |



### Zárszerkezet algoritmus tervezése



3.1. ábra Zárszerkezet folyamatábra

Már a folyamatábrából is kivehető, hogy az NFC-vel nyíló zárszerkezet programja nem nagy bonyolultságú. Vegyük sorra, minek kell történnie a program futása közben.

Először is, a Serial.begin(9600) segítségével beállítjuk a soros adatátvitel adatátviteli sebességét másodpercenkénti bitben (baud). Ennek segítségével láthatjuk a soros monitoron a kiírt értékeket. Szükségünk van az SPI.h könyvtárra, ami lehetővé teszi az SPI eszközökkel való kommunikációt, ahol az Arduino a master eszköz. A könyvtárból az SPI.begin() az SPI-busz inicializálására szolgál, az SCK, MOSI és SS kimenetekre állításával, az SCK és MOSI alacsonyra, az SS pedig magasra húzásával. Az RFID olvasó használásához elengedhetetlenül szükségünk van az RFID.h könyvtárra, ami MFRC522 és más RFID RC522 alapú modulokhoz is használható. A könyvtár init() függvényével inicializálhatjuk a változók között példányosított RC522 chip-et. Mivel szeretnénk egy relét, valamint egy piezo csengőt használni, ezek pinjeit úgy kell konfiguráljuk, hogy kimenetként működjenek. A buzzernek, ha esetleg lenne generált hangja, a noTone() függvény segítségével leállítjuk a tone() által kiváltott négyszöghullám generálását. Nincs hatása, ha nem generál hangot, viszont a kéretlen zajok elkerülhetőek vele. Fontos, hogy a szolenoid zárat úgy szeretnénk használni, hogy magas jelre nyíljon ki, ezért alapértelmezetten alacsonyra állítjuk a relé digitális jelét.

A következő állapotban az a feladat, hogy keresse, valamint érzékelje a közelben lévő RFID tag-et. Amennyiben nem érzékel neki megfelelő objektumot, folytassa tovább a keresést. Szerencsére az RFID függvénykönyvtárának segítségével igazán könnyen megoldható a folyamatos keresés.



Az isCard() igazat ad vissza, ha egy kártya érzékelhető a közelben. Amennyiben a függvény hamis értékkel tér vissza, az azt jelenti, hogy nem érzékelt kártyát, tehát újra kell kezdeni a keresést. readCardSerial() igazat fog visszaadni abban az esetben, ha beolvasható egy UID az RFID tagből. Amennyiben ez nem lehetséges, újraindul a keresés. Fontos megjegyezni, hogy ezt a függvényt mindenképpen meg kell előzze az előbb említett isCard() metódus.

Amennyiben sikeres a beolvasás, ellenőrizni kell, hogy az adott tagnek mi az azonosítója. Ehhez létrehoztam egy változót, amiben eltárolom az összes karakter. Ahhoz, hogy ez kimenthető legyen, végig kell menni az egész UID-n és kimenteni a karaktereket. Debugolás (?) céljából érdemes kiíratni a soros monitorra az azonosítót.



Szerencsére itt elég egy for ciklus használata, valamint a karakterek összefűzésére szolgáló concat() függvény.

A végső ellenőrzés egy nagyon egyszerű összehasonlítás lesz. Amennyiben a beolvasott UID megegyezik a programnak megadott UID-k valamelyikével, akkor jóváhagyásra kerül a belépés, ellenkező esetben nem.



Jóváhagyott státusz esetén írunk a Serial Monitorra, magas jelet kell küldeni a relére, aminek hatására nyílik a zár, lejátszódik a címezhető LED szalagon az elfogadást jelző animáció, majd előre definiált idő után újra visszaáll a zár a kezdeti csukott pozíciójába.



Amennyiben olyan kártyával próbálkozunk, aminek az azonosítója nincs a jóváhagyottokhoz hozzáadva, akkor egy adott ideig tartó hangjelzés fogja jelezni a meghiúsult kísérletet.



A LED szalag programozásához a <FastLED.h> könyvtár [18] került felhasználásra. Ennek segítségével a WS2812 típusú szalagon az inicalizálás után könnyedén lehet műveleteket végrehajtani. Így a szalag úgy viselkedik, mint egy tömb. Végig lehet az elemeken – amik konkrétan az egyes LED-ek – iterálni (?) és azok színét, fényerősségét külön-külön módosítani.

A playAcceptSequence(), valamint a playDeclineSequence() is csak for ciklusokat használ. Az első balról jobbra felkapcsolja a LED-eket, amik zöld színt vesznek fel. Ehhez elég volt két ciklus egymásba ágyazva. A belső ciklus a sorba következő LED-et kapcsolja, míg a külső reseteli az összeset és újraindítja a folyamatot ötször.



Az elutasítást jelző függvény valamivel bonyolultabb. Mivel itt egyszerre megy középről a két irányba az animáció, így szükség volt egy segédváltozóra, a megvalósítás, pedig kettő-kettő egymásba ágyazott for ciklussal történik. A lényeg nem változik, csak a szín, illetve egyszerre két irányba terjed szét az animáció.

## Snake

A Snake egy mindenki által már jól ismert, klasszikus játék. Lényege, hogy a játékos egy kígyó szerű lényt irányít, amivel minél több, véletlenszerű helyen generált gyümölcsöt kell megenni, amik hatására a kígyó mérete növekszik. Azonban vigyázni kell, hogy milyen útvonalat választ a felhasználó, ha a kígyó önmagával ütközik, elvesztette a játékot. Nyerésre akkor van lehetőség, ha megadott számú gyümölcsöt felszed.

### Snake hardver tervezés

A játék megvalósításához két dologra van szükség mindenképpen. Egy kijelzőre, amin látjuk a játék állapotát, valamint egy vezérlőre, amivel irányítani lehet a játékot. Ezen kívül apró fejlesztés lehet egy tekerő, ami a kígyó sebességét módosítja.

Felhasznált alkatrészek:

* MAX7219 modul 1088AS 8x8 LED mátrix-szal
* Analóg joystick modul
* Tekerhető potenciométer

Kijelzőnek egy 8x8-as LED mátrixot használtam fel. A választás azért esett erre, mivel különböző könyvtárak segítségével egyszerű a használata, letisztult megjelenése megnyerő, valamint az Arduino kezdőkészletek szinte kivétel nélkül tartalmazzák a modult.

|  |  |
| --- | --- |
| **MAX7219 modul 1088AS 8x8 LED mátrix** | |
| **Modul pin** | **Bekötése** |
| *GND* | Földre kötve. |
| *VCC* | Tápfeszültségre kötve. |
| *CS* | Arduino **D11**-es lábára kötve. |
| *CLK* | Arduino **D10**-es lábára kötve. |
| *DIN* | Arduino **D12**-es lábára kötve. |

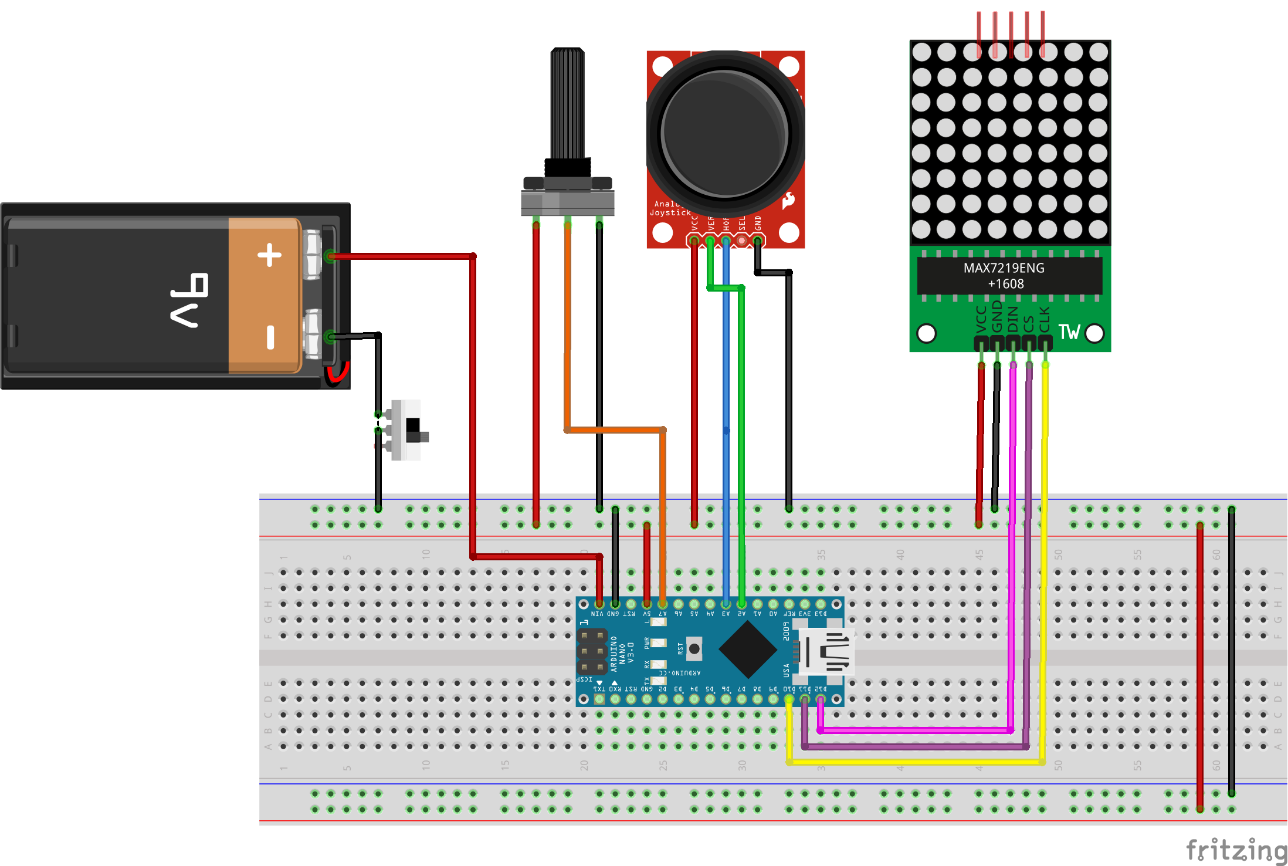
Annak érdekében, hogy eldőljön melyik eszközzel kommunikál az Arduino, szükség van egy chip választó pinre, erre szolgál a modul CS lába. Amikor az Arduino az eszköz CS pinjét logikai alacsonyra állítja, az szolga üzemmódba kerül. A CLK a modulon egy 10MHz maximális sebességű soros óra bemenet. A DIN a digitális jel fogadására szolgál, ezen keresztül érkezik az adat az Arduino felől. Az adatok a CLK emelkedő élére töltődnek be a belső 16 bites shift regiszterbe.

Az irányításra használt joystick felépítése igen egyszerű. Természetesen rendelkezik a tápellátáshoz szükséges pinekkel, emellett pedig egy vízszintes és egy függőleges pozíciót jelző kimenettel. A VRx és VRy valójában egy vízszintes és egy függőleges irányú változtatható ellenállás. Minél jobban elmozdul a joystick valamelyik irányba, annál nagyobb lesz az ellenállás, analóg bemeneten ezt beolvasva az Arduino digitális jelet tud belőle alakítani. Ezeken felül rendelkezik még egy SW pinnel, ami lényegében egy nyomógomb, a joystick lenyomását érzékeli, de ezt a projektben nem használtam.

|  |  |
| --- | --- |
| **Joystick modul** | |
| **Modul pin** | **Bekötése** |
| *GND* | Földre kötve. |
| *VCC* | Tápfeszültségre kötve. |
| *VRx* | Arduino **A2**-es lábára kötve. |
| *VRy* | Arduino **A3**-es lábára kötve. |
| *SW* | - |

A játék sebességének irányítására egy tekerhető potenciométert használtam. Ennek a lényege, hogy tekeréssel változtatható az ellenállás. Ezt az analóg jelet beolvasva az Arduino digitalizálja és használhatja. A VCC és a GND iránya felcserélhető.

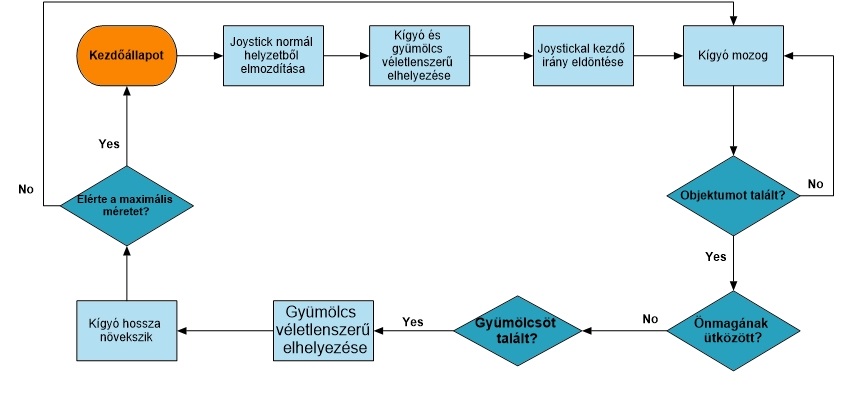
|  |  |
| --- | --- |
| **Potenciométer** | |
| **Modul pin** | **Bekötése** |
| *VCC* | Tápfeszültségre kötve |
| *OUTPUT* | Arduino **A7**-es lábára kötve. |
| *GND* | Földre kötve. |



2.2. ábra Snake játék kapcsolási rajza

### Snake algoritmus tervezés

A Snake játék már a bonyolult kategóriába tartozik. Szükséges a vezérlési szerkezetek mellett még a struktúrák ismerete, valamint a kétdimenziós tömbök megértése is. Ennek átlátásához már valamivel több idő, illetve tapasztalat szükséges, ugyanakkor ez is elsajátítható némi idő ráfordításával.



3.3. ábra Snake játék folyamatábra

A flowcharton is látszik, hogy annak ellenére, hogy nincs sok útvonal, és viszonylag lineáris a folyamat, mégis elég sok állapotból áll a program.

Első sorban a LED mátrix vezérléshez elengedhetetlen a LedControl.h könyvtár használata. Mivel a korábbiakhoz képest több pin lett felhasználva, ezért ebben a projektben a pinek változói egy struktúrában lettek létrehozva, ezzel átláthatóbbá téve a későbbi használathoz.

Sok globális változóra is szükség van, a kígyó kezdeti hosszának megadásához, a pillanatnyi hossz lekéréséhez, aktuális sebesség lekéréséhez, valamint a kígyó menetirányának lekéréséhez, illetve megadásához szükséges konstansok. A játékot szabályzó változók a fényerő szabályzásához szükséges változó, a játékteret tároló kétdimenziós 8x8-as tömb, valamint a játék állapotát tároló boolean változók, amik a nyerést, illetve vesztést hivatottak jelezni. A kígyó, a gyümölcs és a joystick kalibrálásához szükséges adatok tárolására szintén struktúra lett létrehozva, amely a Point és a Coordinate nevet viselik. Mindkettőnek két adattagja van, a Point ami a kígyó és a célpont egy sor és egy oszlop adattaggal, míg a Coordinate egy x és egy y -nal rendelkezik. Default értékük 0,0.



Az Arduino setup() függvényében a soros adatátvitel beállítása történik meg, a joystick kalibrálása a calibrateJoystic() függvény segítségével.

Ezen felül a setup()-on belül megtalálható a játék kezdőállapotának beállítására létrehozott initGame() metódus. Az utóbbi a pinek ki-, valamint bemenetre való állítását, a LED mátrix újraindítását, fényerejének beállítását, valamint a kígyó kezdőpozíciójának beállítását végzi el.

A program futása közben a loop() függvény négy másik metódust fog meghívni folyamatosan egymás után. Ezek sorban a gyümölcs generáló generateFood(), a joystick aktuális állapotát felmérő scanJoystick(), a snake helyzetét kiszámító calculateSnake(), valamint a nyerés illetve vesztés állapotot beállító setState() függvény.

Sorban haladva a generateFood() a gyümölcs véletlenszerű generálását végzi egy nagyon egyszerű ellenőrzéssel.



Abban az esetben, ha a gyümölcs a (-1,-1) -es pozícióban van, az azt jelenti, hogy a kígyó előzőleg „megette”, ezért a pályán kívül jött létre. Ha a kígyó hossza eléri a kívánt hosszúságot, akkor nem kell újra generálni, hiszen nyertünk, a win változó igaz értéket vesz fel. Ha ezt a számot még nem érte el a kígyó hossza, akkor a gyümölcsöt egy új, véletlen helyen kell létrehozni.

A scanJoystick() függvény a kígyó sebességétől mérten olvassa be a joystick pozícióját az analóg inputról. Működése nagyon egyszerű, amennyiben adott irányban átlépi a threshold értéket a joystick pozíciója, a négy irány közül felveszi a megfelelőt.



Egy egyszerű ellenőrzéssel kezelésre kerül, hogy a kígyó ne tudjon 180°-os fordulatot tenni, hiszen az nem lenne a működésnek megfelelő.



A snake útvonalának kiszámításához és megjelenítéséhez szükség van egy fal detektáló függvényre, aminek a lényege, hogy ha falhoz érne a kígyó, akkor a szemközti faltól folytassa az útját.



A kígyó mozgásának érdemi részét a calculateSnake() metódus végzi. Először a snakeDirection változó alapján egy switch-case vezérlési szerkezet megállapítja, hogy melyik irányba kell növelni a kígyó hosszát. Mivel a LED mátrix sorai, illetve oszlopai balról jobbra, valamint fentről lefele növekednek, ha a kígyót feljebb szeretnénk vinni, akkor a sorainak számát csökkentenünk kell. Amennyiben falhoz érne, a wallDetection() átviszi az ellentétes oldalra, a setLed() pedig bekapcsolja az adott LED-et. A program hasonlóképpen működik a négy irányra.



Ezek után az objektumokkal való érintkezés van lekezelve. Amennyiben a kígyó egy olyan szegmensre lépne, ahol a kígyó szerepel, akkor a játék véget ért.

A másik eshetőség, hogy nem saját magával, hanem a célponttal, egy gyümölccsel kerül egy cellába. Ebben az esetben, a gyümölcsöt a pályán kívülre kell elhelyezni, hiszen ez alapján fogja tudni a korábban megírt generateFood(), hogy egy új helyre kell elhelyezni a gyümölcsöt. Ezen kívül a kígyó hosszát tároló snakeLength változó értékét meg kell növelni.



Azoknak a mezőknek az értéke, amik nullánál nagyobbak voltak, azaz volt rajta kígyó, megnövelődnek eggyel. Erre azért lesz szükség, mert az ezt követőben szekcióban, ahhoz, hogy ne egy végtelenbe nyúló vonal legyen, a végéről el kell venni egyet. Ez a következő kódrészletben fog megvalósulni.



A loop() függvény legutoljára a setState() -et hívja, ami nemes egyszerűséggel a győzelmi, valamint a vereség állapotok között vált.



## Dino run

A harmadik játéknak szerettem volna egy olyat megvalósítani, ami széles körben elterjedt és biztosan sokan találkoztak már az ötletadójával. A Dinosaur Game egy beépített böngészős játék a Google Chrome böngészőben. A játékos egy pixelezett Tyrannosaurus rexet vezet egy oldalra gördülő tájon, elkerülve az akadályokat a magasabb pontszám elérése érdekében. [18]. Bonyolultsága abból ered, hogy amellett, hogy szükségünk van több alkatrészre, a kijelző igen sok kapcsolást fog igényelni.

### Dino run hardver tervezés

A megvalósításhoz itt is két dologra lesz szükség a Snake-hez hasonlóan. Kell egy kijelző, valamint egy, a játék irányítására szolgáló alkatrész. Ebbe a játékba szerettem volna belevonni a szenzorok működését, rávilágítva arra, hogy mennyi mindenre lehet az Arduinot használni. Az írányítást egy mozgásérzékelő PIR szenzorral oldottam meg, a képi megjelenést pedig egy LCD modulon valósítottam meg.

Felhasznált alkatrészek:

* Tekerhető potenciométer
* PIR mozgásérzékelő
* LCD1602 LCD modul
* 220Ω ellenállás

A PIR infravörös érzékelő három csatlakozási felülettel rendelkezik. Ezek közül kettő a VCC és GND, a harmadik pedig a digitális jelet továbbító DOUT. A szenzor felhasználás függően szabályozható. Két szabályzóval lehet módosítani a szenzor paramétereit. Az egyik az érzékenységet állítja, néhány tíz centiméterről egészen pár méterig képes érzékelni az eszköz. A másik a késleltetés idejét állítja, ami 1-től 30 másodpercig terjedhet. Fontos megjegyezni, hogy a PIR szenzornak szükséges legalább egy perc bemelegedési idő, addig nem fog megfelelően működni.

|  |  |
| --- | --- |
| **PIR szenzor** | |
| **Modul pin** | **Bekötése** |
| *GND* | Földre kötve |
| *Digital OUT* | Arduino **D2**-es lábára kötve |
| *VCC* | Tápfeszültségre kötve |

A megjelenítésre ebben az esetben egy zöld háttérvilágítású LCD modult választottam.

|  |  |
| --- | --- |
| **PIR szenzor** | |
| **Modul pin** | **Bekötése** |
| *VSS* | Földre kötve. |
| *VDD* | Tápfeszültségre kötve. |
| *V0* | Tekerhető potenciométer **OUT** lábára kötve. |
| *RS* | Arduino **D11** lábára kötve |
| *R/W* | Földre kötve |
| *E* | Arduino **D8** lábára kötve. |
| *D0-D7* | - |
| *D4-D7* | Arduino **D2-D6** lábára kötve. |
| *A* | 220Ω-os ellenállással sorban a tápfeszültségre kötve |
| *K* | Földre kötve |

Az LCD modul VSS pinjén keresztül kapja a negatív, a VDD-n pedig a pozitív tápfeszültséget. A V0-on keresztül szabályozható a kijelző kontrasztja, erre egy potenciométert kötöttem, amivel játék közben is módosítható. Ez kihagyható, ha a kódban van definiált érték a kontrasztnak. Az RS pinen az utasító és az adat regiszter közötti váltás történik meg. Az R/W jelen esetben le van földelve, ez az olvasást/írást választó jel. Az E (Enable) pin a kijelző engedélyezésére szolgál. Ez azt jelenti, hogy amikor ez a pin alacsonyrara van állítva, az LCD nem törődik azzal, hogy mi történik az R/W, RS és az adatbusz vonalakkal. Amikor ez a pin magasra van állítva, az LCD feldolgozza a bejövő adatokat. 4 bites üzemmódban az LCD D4-től D7-ig terjedő adatbuszvonalait kell használni, abban az esetben, ha 8 bites módot kellene, a D0-tól D3-at is csatlakoztatni kellene. Az A és K pin a háttérvilágítás anódja és katódjaként szolgál.

|  |  |
| --- | --- |
| **Potenciométer** | |
| **Modul pin** | **Bekötése** |
| *VCC* | Tápfeszültségre kötve |
| *OUTPUT* | LCD **V0** lábára kötve. |
| *GND* | Földre kötve. |

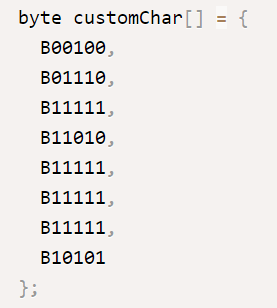
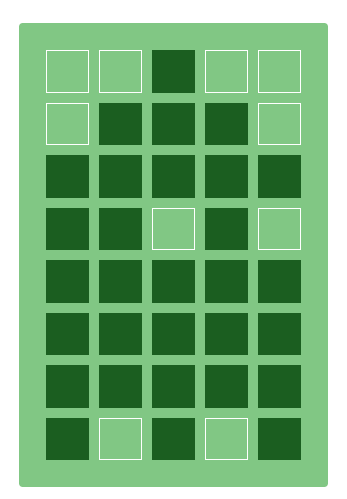
### Dino run algoritmus tervezés

A dolgozat kereteibe nem fért bele, hogy egy teljesen saját játékot írjak a kijelzőre, ezért egy, az Arduino kezdőprojektek között igen népszerű játékot fejlesztettem tovább, ami sok felhasználó által már meg lett valósítva. [19]

A továbbfejlesztés célja az volt, hogy a szenzorok működését is belehessen vonni a bemutató projektbe. Így a sokszor felhasznált gombokkal való irányítás helyett egy PIR szenzort felhasználva hoztam mozgásba a karaktert.

Mivel a PIR szenzor csak négy másodpercenként képes érzékelni, ezért át kellett írni egy kicsit az akadály generálást, hogy nagyobb időközökkel jöjjenek. Ezen felül csak annyi dolog volt, hogy a gomb érzékelésnél az interruptot RISING-ra kellet állítani, így a szenzor érzékelése elején már ugrik a karakter.

Lehetőségünk van egyedi karaktereket létrehozni, ehhez számos online eszköz áll rendelkezésünkre. [21] Én egy szellemet hoztam létre, ennek látható a megvalósítása.



3.4. ábra Egyedi karakter (bal), és a hozzá tartozó kód (jobb)