Négyzetrácsalapú akciójátékok programozása a Scratch-ben

Bernát Péter

bernatp@inf.elte.hu
ELTE IK

Absztrakt. A programozástanítás számos nagy témakörének egyike a játékfejlesztés. A bevezető programozástanítás során a számítógépes játékok sokféle műfaja közül gyakran esik a választás az akciójátékokra. Egy korábbi cikkemben három játéktípuson keresztül ismertettem az akciójátékok legfontosabb kellékeit és megvalósításukat a Scratch-ben. Jelen cikkemben az előzőre építve a négyzetrácsalapú akciójátékok – amelyekben a szereplők egy négyzetrács mezőin mozognak – megvalósításának a sajátosságait járom körül és értékelem továbbra is a Scratch programozási környezetben.

Kulcsszavak: Scratch, programozástanítás, játékfejlesztés, akciójátékok, négyzetrácsalapú játék

1. Bevezetés

Informatikatanárként a bevezető programozástanítás során számos olyan programozási terület közül választhatunk, amelyekkel a programozás alapfogalmai szemléletesen és motiváltan vezethetők be, és amelyeket a kezdőknek szánt programozási környezetek nagymértékben támogatnak [1]. Ilyen terület a teknőcgrafika, a robotika [2], az animációkészítés [3] és a játékfejlesztés is [4].

A játékfejlesztéssel kihasználható a tanulóknak a játékprogramok (működése) iránti érdeklődése, és ez a terület kézenfekvő folytatása az animációkészítésnek: a játékprogram objektumait továbbra is mozgatni és animálni szükséges, ugyanakkor a köztük, illetve a játékos és a játékprogram közti interakciók megvalósításához eseményvezérelt programozásra, az elágazások és a változók gyakori alkalmazására is szükség van [1].

A játékfejlesztésen belül gyakran esik a választás az akciójátékok műfajára, amelyek a kezdőknek szánt, objektumorientált és eseményvezérelt programozási környezetekben kismértékű absztrakcióval, praktikusan elkészíthetők. Ezekben a játékokban a főszereplővel egy leegyszerűsített absztrakt világban kell előre haladni, amelyben jellemzők az áthatolhatatlan falak, az előre haladást nehezítő ellenségek, és az előre jutáshoz szükséges tárgyak, amelyeket fel kell venni és máshol felhasználni. Jellemzően többpályásak, és folyamatosan nyilvántartják a játékos teljesítményét (például a teljesített pályák számát, a pontszámot, a felhasznált időt), és a játékos számára a cél a minél jobb teljesítmény elérése [5].

Korábbi cikkemben [4] a feladattípusorientált programozástanítási módszernek [6] megfelelően három egyre összetettebb feladattípuson – az akadálypályás, a labirintus- és a platformjátékokon – keresztül ismertettem az akciójátékoknak az imént felsorolt jellemzőinek a megvalósítási lehetőségeit a Scratch-ben. A bemutatott példaprogramok és a Scratch-ben alkalmazott megoldási módszerek a sajátjaim voltak.

Jelen cikkemben a négyzetrácsalapú akciójátékok – amelyekben a szereplők egy négyzetrács mezőin mozognak – elkészítésének a sajátosságait járom körül továbbra is a Scratch programozási környezetben, egy mintajáték megvalósításának a bemutatásán keresztül. A bemutatás meghatároz egyúttal egy lehetséges tanítási sorrendet. A megvalósítás ismertetését követően variációs lehetőségeket is megfogalmazok, amelyek további (önállóan megoldandó) feladatokként is felhasználhatók. Végül didaktikai szempontból értékelem a négyzetrácsalapú akciójátékok programozását.

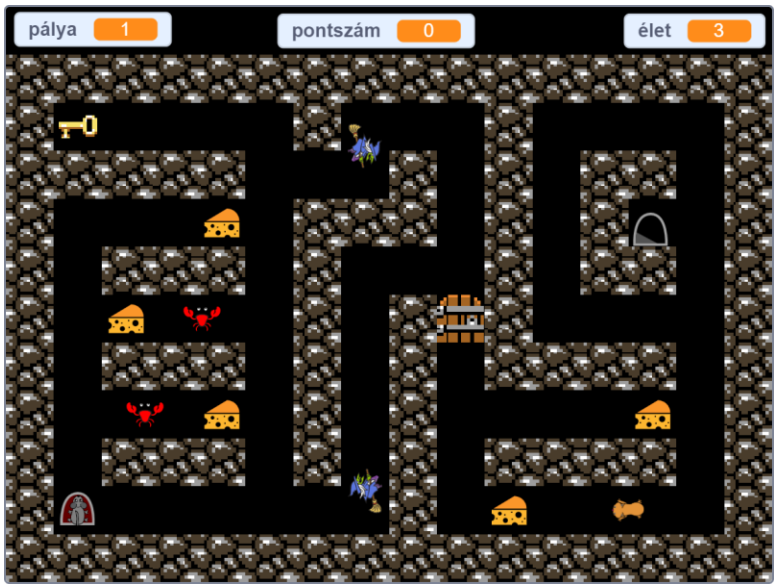
Az ebben a cikkben bemutatott mintaprogram és a Scratch-ben alkalmazott megoldási módszerek is a sajátjaim. Nem ismerek olyan irodalmat, amely a négyzetrácsalapú akciójátékok elkészítésével foglalkozna a Scratch-ben. A Scratch-re építő ismeretterjesztő könyvek egy részét a programozási alapfogalmak tematizálják, és csak egy-két nagyon egyszerű játék elkészítését mutatják be [7]. Pozitív példa Carol Vorderman könyve [8], amely kifejezetten a játékprogramozásról szól a Scratch-ben, de ez sem foglalkozik a négyzetrácsalapú játékok készítésével. Találtam más (alacsonyabb szintű, professzionális) programozási nyelvhez készült, a játékkészítésről szóló és a négyzetrácsalapú játékok készítésével is foglalkozó forrást [9], a benne foglaltakhoz képest azonban a Scratch sajátosságai miatt lényegesen eltérő megoldási módszereket kellett választanom.

2. A mintajáték bemutatása

A kétpályás (de további pályákkal bővíthető) játékban a piros színű *Bejárat* elől (a képernyőképen a bal alsó sarok környékén) induló és a nyílgombokkal irányítható szürke *Egérrel* kell összegyűjteni a pályán található összes *Sajtot,* majd távozni a szürke-fekete *Kijáraton* keresztül (amely a képernyőkép jobb felső sarkától nem messze található). Ennek a véghezvitelét a falakon kívül további akadályok nehezítik: a fából készült (a képernyőkép középső részén látható) *Ajtó* csak a *Kulcs* megszerzését követően nyitható, ha pedig az *Egér* a különböző szabályok szerint mozgó ellenségek valamelyikével ütközik, egy életet veszít és – ha még maradt élete – visszakerül a *Bejárathoz* (a *Bejáratra* az ellenségek nem léphetnek, ott az *Egér* az elindulásig biztonságban van).

Az ellenségek közül a *Rákok* folyamatosan vízszintesen haladnak, és akkor fordulnak vissza, ha falnak ütköznek. A *Boszorkányok* falkövető algoritmus szerint mozognak: mindig az irányukhoz képest balra eső falat követik. Végül a *Macska* (amely a képernyőkép jobb alsó sarkának környékén látható felülnézetben) az elágazásokban azt a (nem a háta mögötti) oldalszomszédos mezőt választja, amelyhez az *Egér* aktuálisan a legközelebb van. A játék a játékos egyes részeredményeit (például a *Sajtok* megszerzését) pontokkal is jutalmazza (1. ábra).

A kész játék és a programkód elérhető a következő címen: *https://scratch.mit.edu/projects/766757769*



1. ábra: A mintajáték első pályája.

3. A mintajáték megvalósítása

A következőkben bemutatom a játékprogram megvalósításának a fontosabb lépéseit az alulról felfelé építkezés elve szerinti sorrendben, amely az elképzelésem szerinti tanítási sorrendnek is megfelel. Az utolsó előtti lépésig a játék egypályás változata készül el, amelyben minden típusú pályaelemre található lesz példa. A több pálya megszervezésével a korábbi cikkemben már foglalkoztam, amelyet a jelenlegi mintajáték megvalósításának az utolsó lépésében használok fel. Az egypályás játék és a programkódja elérhető a következő címen: *https://scratch.mit.edu/projects/778788909*

3.1. Négyzetrácsos pálya rajzolása

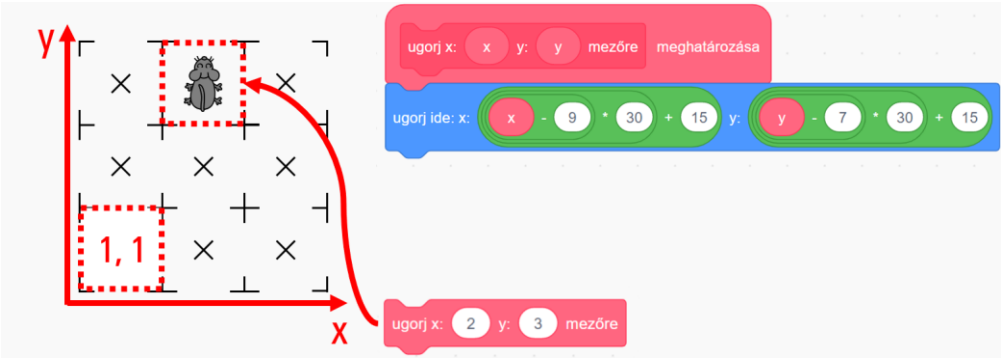
Az *Egér* mozgatásának a megvalósítása során elengedhetetlen lesz, hogy képesek legyünk ellenőrizni a falakkal történő átfedést. A Scratch-ben egy szereplő egy színárnyalat vagy egy másik szereplő érintését tudja érzékelni. A falak színárnyalatai azonban egyes szereplők jelmezeiben is előfordulhatnak, amely szereplőket nem szeretnénk tévesen falként érzékelni. Ezért a falakat egy a *Színpad* méretével megegyező méretű, (például) *Falak* nevű szereplő jelmezére kell rajzolnunk, és az *Egérnek* az ezzel a szereplővel történő átfedését kell majd vizsgálnunk.

A Scratch rajzolóablakában túl körülményes négyzetrácsalapú képet készíteni a négyzetekbe illő textúrákból („csempékből"), de léteznek kifejezetten erre a célra megalkotott könnyen használható és ingyenes rajzolóprogramok. Közülük a mintaprogramban a *Tiled*[1] nevűt használtam, a falak textúrájához pedig az *OpenGameArt*[2] oldaláról jutottam hozzá (szintén ingyenesen). A 480×360 képpontból álló *Színpadot* képzeletben 30×30 képpontos mezőkre osztottam fel, amelynek bizonyos mezőiben elhelyeztem a fal (30×30 képpontos) textúráját.

3.2. Az Egér elhelyezése a négyzetrácson

Az *Egeret* úgy szeretnénk majd mozgatni, hogy csak mezőközéppontról mezőközéppontra haladhasson. Ehhez azonban szükséges, hogy a játék elején már eleve valamely mező középpontjából induljon.

A Scratch beépített *ugorj ide x: y:* nevű parancsa a *Színpad* közepére illesztett koordinátarendszerben értelmezett x és y koordináta szerint pozícionálja a szereplőket. Körülményes lenne a számunkra állandóan kiszámolni egy-egy mező középpontjának a koordinátáit, ezért célszerű létrehoznunk azt az *ugorj x: x y: y mezőre* nevű eljárást, amely a beépített *ugorj ide x: y:* parancs segítségével a paraméterekben megadott *(x, y)* „mezőkoordinátájú" mező közepébe helyezi az eljárást meghívó szereplőt (2. ábra).



2. ábra: Az *Egér* (és a többi szereplő) elhelyezését segítő eljárás meghatározása, és egy példa a használatára.

[1] https://www.mapeditor.org/

[2] https://opengameart.org/

Az eljárás bemutatott megvalósításában az (1, 1) mezőkoordináta-pár a *Színpad* bal alsó sarkában található mezőt határozza meg, és a két mezőkoordináta jobbra, illetve felfelé haladva növekszik.

3.3. Az Egér irányítása a négyzetrácson

Az *Egeret* a játékosnak úgy kell tudnia irányítani, hogy azzal a négy oldalszomszédos mező közül a falakat (és a zárt *Ajtót*) nem tartalmazók középpontjára lehessen átsétálni. Azt szeretnénk, ha nem egyszerűen áthelyeződne az új mezőre, hanem (például) 3 képpontonként haladva jutna el oda úgy, hogy közben a mozgását ne lehessen befolyásolni, tehát például ne lehessen vele visszafordulni vagy kanyarodni.

A korábbi cikkemben láthattuk, hogy a játékos által irányítható szereplő mozgását egy olyan – például *mozdulj el* nevű – eljárással érdemes megvalósítani, amelyet a szereplő egy végtelen ciklusban (a szereplő úgynevezett játékciklusában) folyamatosan meghív.

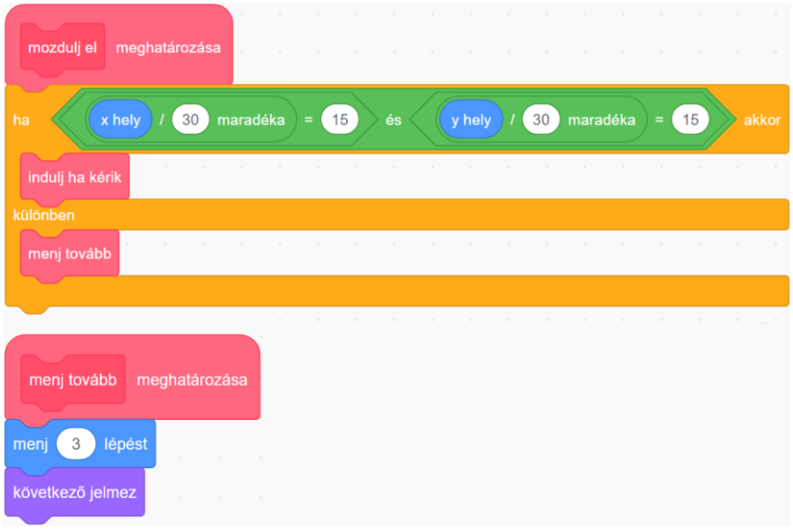
Az *Egér* tervezett működéséből következik, hogy eltérően kell viselkednie

a) egy mező középpontjában, illetve

b) azon kívül,

és hogy az utóbbi esetben a körülményektől függetlenül 3 képponttal kell az aktuális irányába mennie.

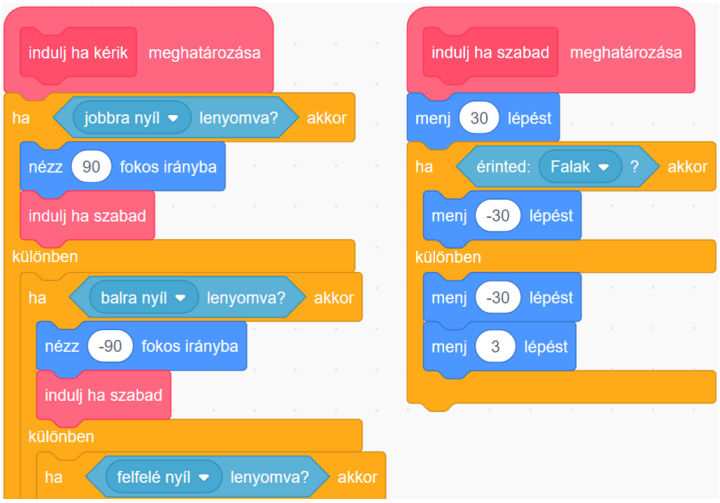
Hogy a két eset közül melyik áll fenn, az a szereplő két (eredeti értelemben vett) koordinátájából kikövetkeztethető: pontosan akkor tartózkodik egy mező közepén, ha mindkét koordinátája 30-cal osztva 15-öt ad maradékul. Ennek megfelelően a *mozdul el* eljárás megvalósítását a 3. ábrán látható módon bonthatjuk fel.



3. ábra: Az *Egér* irányítását megvalósító eljárás (fent),
és a két eset közül a másodikat megvalósító eljárás (lent).

Az *indulj ha kérik* eljárásnak kell gondoskodnia arról, hogy az *Egér* elinduljon valamelyik oldalszomszédos mező irányába 3 képponttal, ha a játékos ezt kezdeményezi, és a szóban forgó mező üres. Ha elindul, akkor a *mozdulj el* eljárás következő meghívásakor már nem mezőközéppontban lesz, ennek megfelelően a *menj tovább* eljárás fogja újabb 3 képponttal előre mozgatni. Ez az utóbbi eljárás pedig addig lesz újra és újra meghívva, ameddig az *Egér* a következő mező középpontjába meg nem érkezik. Onnan a játékos újra elindíthatja őt (ha szeretné) valamelyik megfelelő irányba.

Az *indulj ha kérik* eljárás egy lehetséges megvalósítása a 4. ábrán látható. Az *Egeret* a lenyomott nyílgombnak megfelelő irányba állítjuk (ha le van nyomva valamelyik nyílgomb), majd átadjuk a vezérlést az *indulj ha szabad* eljárásnak. Ebben először átmozgatjuk a szereplőt az oldalszomszédos mezőre *(menj 30 lépést),* hogy azon elvégezhessük a *Falak* szereplővel történő átfedés ellenőrzését. Akár volt átfedés, akár nem, visszahelyezzük a szereplőt az eredetileg elfoglalt mezőre *(menj -30 lépést),* és ha nem történt átfedés, akkor elindítjuk 3 képponttal. Erre az oda-vissza helyezésre – amely olyan gyorsan történik, hogy a Scratch a képernyőn nem jeleníti meg[3] – azért van szükség, mert a szereplők csak az aktuális helyükön képesek érintést ellenőrizni.



4. ábra: Az *Egeret* az *indulj ha kérik* eljárás indítja el a játékos által meghatározott oldalszomszédos mezőre, de csak akkor, ha azon nincs fal[4].

3.4. A Kulcs és az Ajtó elhelyezése és működtetése

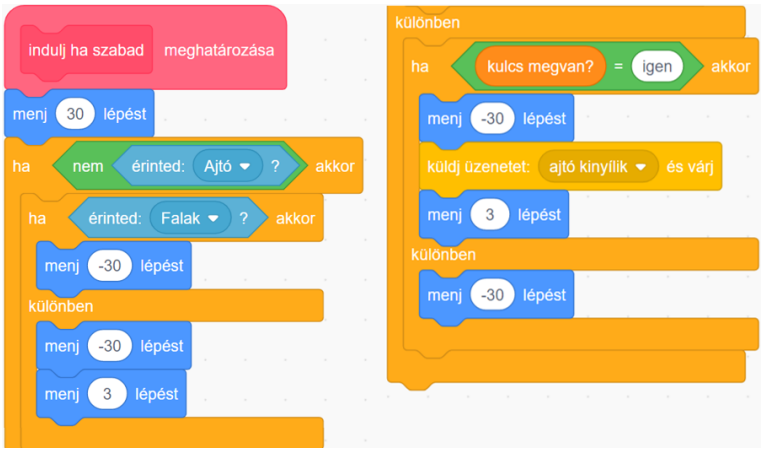
A *Kulcsot* és az *Ajtót* az *Egérhez* hasonlóan az *ugorj x: x y: y mezőre* segédeljárással helyezhetjük el a pályán.

A korábbi cikkemben találkozhattunk a kulccsal nyitható ajtó megvalósításával. A *Kulcs* működtetése a következőképpen történhet. Kezdetben egy (például) *kulcs megvan?* nevű változóban regisztráljuk (például a *nem* változóértékkel), hogy a játékos nem rendelkezik a kulccsal. A játék során pedig a főszereplő – jelenesetben az *Egér* – a saját játékciklusában folyamatosan ellenőrzi egy (például) *érzékeld a kulcsot* nevű eljárásban, hogy átfedésben van-e a *Kulccsal.* Ha igen, akkor üzenetküldéssel felszólítja a *Kulcsot* az eltűnésre, valamint a *kulcs megvan?* változóban annak a tárolására (például az *igen* változóértékkel), hogy a játékos már rendelkezik a kulccsal.

Az *Ajtó* a *Kulcs* megszerzéséig falként kell, hogy funkcionáljon, ezért a fallal való átfedés ellenőrzéséért felelős, korábban ismertetett *indulj ha szabad* eljárást kell kibővítenünk. Természetesen továbbra sem szabad az *Egeret* falat tartalmazó mezőre juttatni. Ha azonban a játékos által választott iránynak megfelelő oldalszomszédos mezőn az *Ajtó* található, akkor pontosan akkor kell azt a mezőt üresnek tekintenünk, ha a játékos már rendelkezik a kulccsal (és ebben az esetben egy üzenetküldéssel fel kell szólítanunk az *Ajtót* az eltűnésre) (5. ábra).

[3] Hasonlóan például ahhoz, ahogyan a teknőcgrafika teknőce „észrevétlenül” rajzol ki egy egyszerűbb alakzatot.

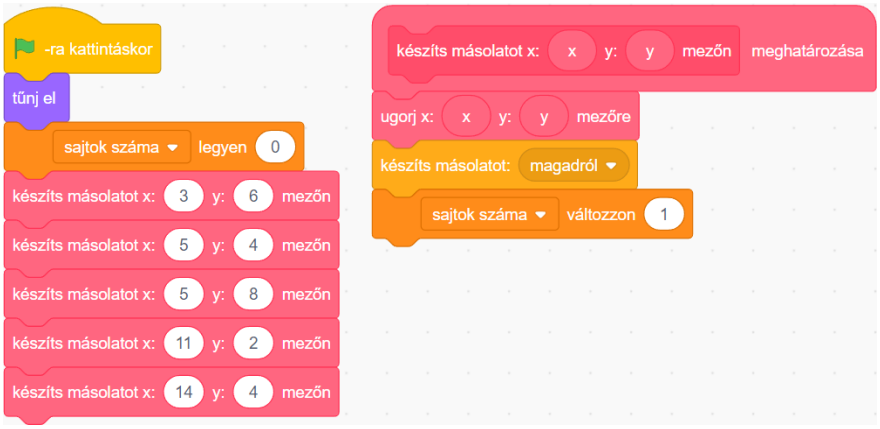
[4] A felfelé és a lefelé nyíl lenyomva tartásának a kezelése hasonlóan történhet.



5. ábra: Az *Egér indulj ha szabad* nevű eljárásának a kiegészítése az *Ajtó* kezelésével.

3.5. A Sajtok elhelyezése és működtetése

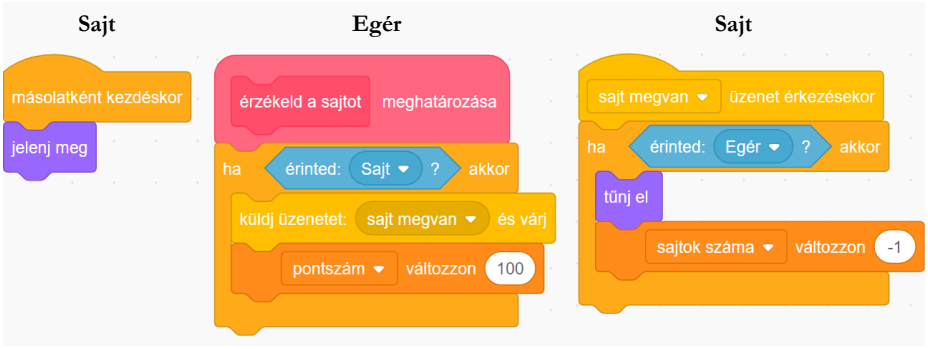
Mivel számos (de egyformán működő) sajtot szeretnénk elhelyezni a pályán, egyetlen *Sajt* szereplőből készíthetünk példányokat a Scratch másolatkészítés (klónozás) lehetőségével. Az eredeti példányát kezdetben láthatatlanná tesszük – ezáltal nem fog részt venni a játékban –, majd „pecsételőként" használjuk: a megfelelő helyeken egy-egy másolatot készítünk belőle, és a *sajtok száma* nevű változóban folyamatosan nyilvántartjuk a másolatok darabszámát (mert az beépített paranccsal nem lekérdezhető) (6. ábra).



6. ábra: A *Sajt* másolatainak az elhelyezése.

Az egyes példányoknak a létrejöttükkor egyetlen dolguk van, láthatóvá válni, hiszen a „pecsételő" minden tulajdonságát, így a pillanatnyi helyzetén túl a láthatatlanságát is megörökölték (7. ábra, balra).

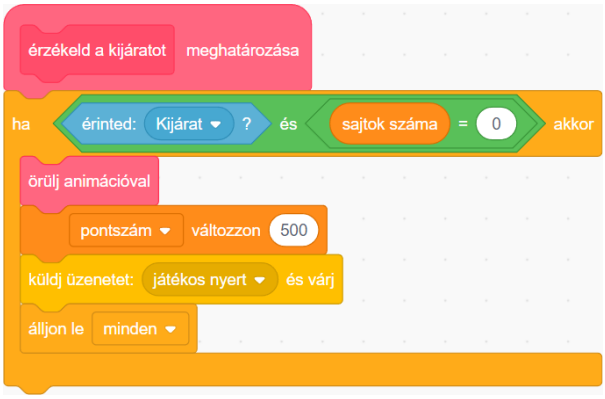
Az *Egér* a *Sajt* valamelyik másolatával történő ütközését is a végtelen ciklusában elhelyezett (például) *érzékeld a sajtot* nevű eljárásában ellenőrzi, amely szerint ha átfedésben van a *Sajt* valamelyik példányával, akkor a *sajt megvan* üzenet kiküldésével felszólítja a megfelelő példányt az eltűnésre (7. ábra, középen). Mivel azonban ezt az üzenetet minden példány megkapja, a példányoknak is ellenőrizniük kell, hogy ők vannak-e átfedésben az *Egérrel*. Ha igen, akkor (csak) a szóban forgó példány eltűnik, aki a *sajtok száma* változóban elkönyveli a találatot (7. ábra, jobbra).



7. ábra: A *Sajtok* működtetése.

3.6. A játék megnyerése

A szokásos módon elhelyezhetjük a *Kijáratot* a pályán, majd foglalkozhatunk avval, hogy ha az *Egér* eljutott a *Kijárathoz* úgy, hogy már minden *Sajtot* begyűjtött, akkor érjen véget a játék (valamilyen győzelmet kihirdető felirat megjelenítését követően). Az *Egér* játékciklusában meghívható a 8. ábrán látható *érzékeld a kijáratot* eljárás (a játékos győzelmét kihirdető felirat a *játékos nyert* üzenet érkezésekor jelenik meg).



8. ábra: A játék megnyerése.

3.7. Az ellenségek elhelyezése és mozgatása

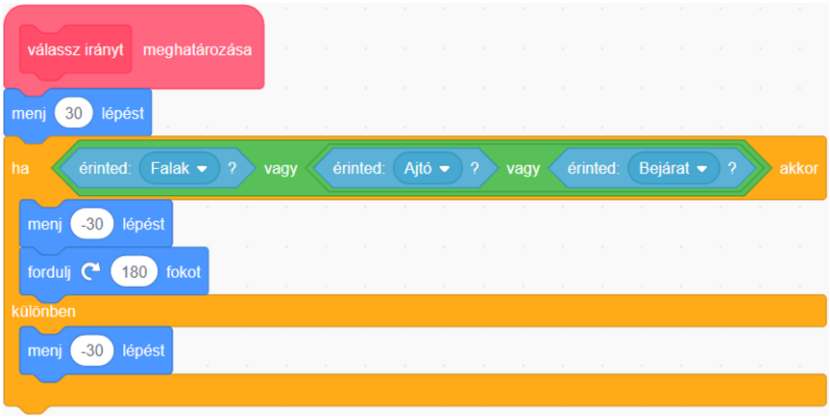
A játékot lényegesen izgalmasabbá tehetjük az ellenségekkel, amelyeket a *Sajtokhoz* hasonlóan egy-egy láthatatlanná tett prototípusuk (a *Rák*, a *Boszorkány* és a *Macska*) másolataiként hozhatjuk létre és helyezhetjük el a játék elején.

A másolatokat a létrejöttükkor meg kell jelenítenünk, majd ezután következhet a mozgatásuk megvalósítása egy végtelen ciklusban a 9. ábrán látható módon. Ha mezőközépponton állnak, akkor valamilyen logika szerint a négy lehetséges irány egyikébe fordulnak (*válassz irányt* eljárás), és mindenképpen (akár mezőközépponton tartózkodnak, akár nem) előre mennek 3 képpontot (*menj tovább* eljárás). A háromféle ellenség – egyre összetettebb – mozgása csak az irányválasztás logikájában fog különbözni.

9. ábra: Az ellenségeket mozgató eljárás.

3.7.1. A Rák mozgatása

A *Ráknak* az irányválasztás során ellenőriznie kell, hogy az iránya szerinti következő mező üres-e, hasonlóan ahhoz, ahogyan az *Egér* ellenőrizte ugyanezt: ideiglenesen rálép a következő mezőre, és ha az foglalt (mert fal van rajta, vagy az *Ajtó*, vagy a játékos számára védelmet biztosító *Bejárat*), akkor az eredeti mezőre visszakerülve ellentétes irányba fordul, ha pedig nem foglalt, akkor az irányának a módosítása nélkül ugrik vissza az eredeti mezőre (ahonnan majd a *menj tovább* eljárás elmozdítja) (10. ábra).



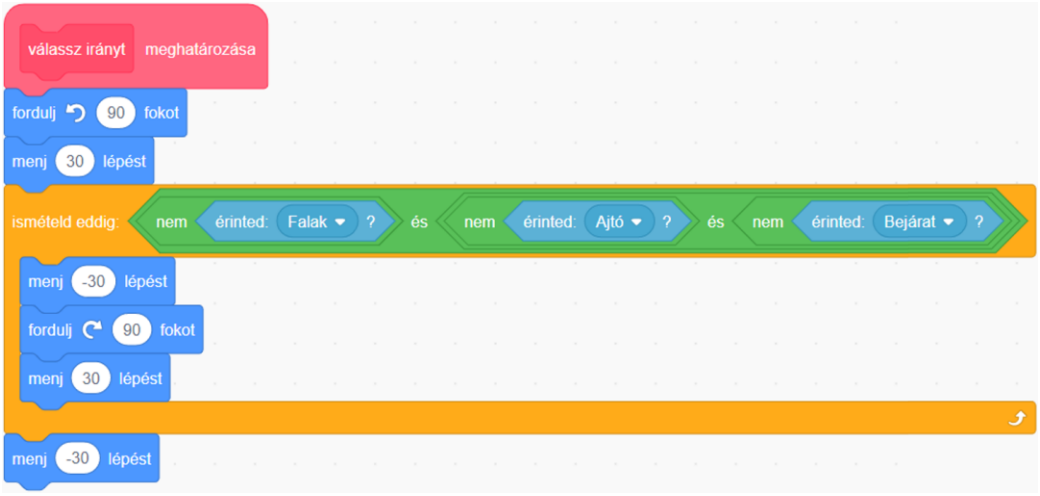
10. ábra: Ha a *Rák* nem léphet az iránya szerinti következő mezőre, visszafordul.

3.7.2. A Boszorkány mozgatása

Mint arról a játék bemutatásakor szó esett, a *Boszorkányokat* falkövető algoritmus szerint szeretnénk mozgatni: mindig az irányukhoz képest balra eső falat kell követniük. A lehetséges eseteket megvizsgálva rájöhetünk, hogy a *válassz irányt* eljárásban az irányukhoz képest balra, előre, jobbra, illetve hátra található mezők közül ebben a sorrendben az első üreset kell kiválasztaniuk.

A *kiválasztás programozási tételt* [10] alkalmazhatjuk: először a balra eső mezőt vizsgáljuk meg, majd amíg foglalt az aktuálisan vizsgált szomszédos mező, addig vizsgáljuk sorban a következőt. Ehhez az *ismételd eddig* feltételes ciklust használhatjuk (amelyben a kilépés feltételét kell megfogalmazni)

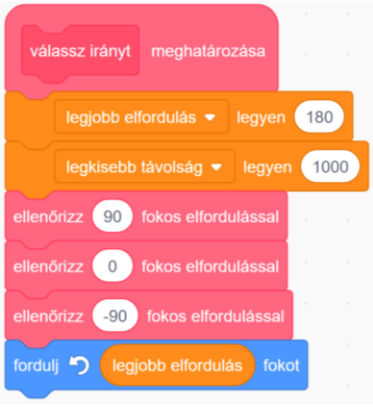
(11. ábra). A *Boszorkányok* háta mögötti mező (ahonnan az aktuális mezőre érkeztek) biztosan szabad, így biztosan találni fognak továbbhaladási irányt.



11. ábra: A *Boszorkány* az iránya szerinti balra, előre, jobbra és hátra található oldalszomszédos mezők közül (ebben a sorrendben) az első szabad mezőt választja.

3.7.3. A Macska mozgatása

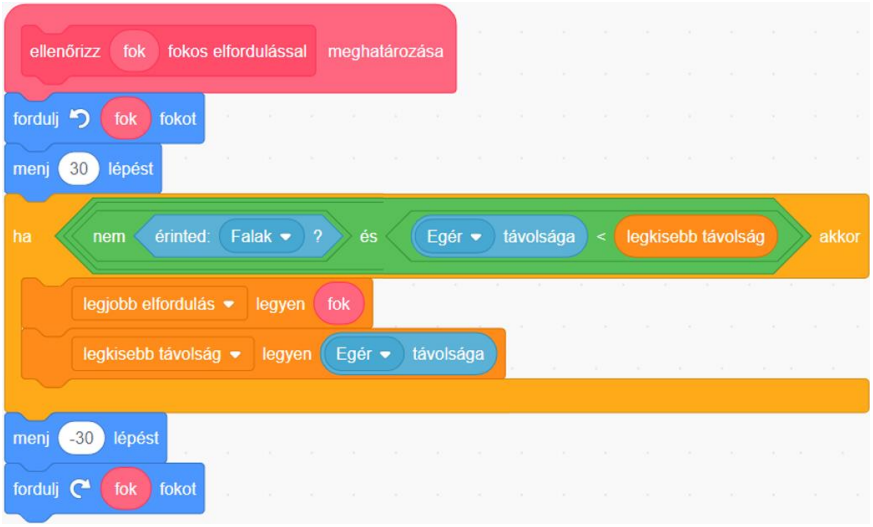
A háromféle ellenség közül a macska lesz a „legintelligensebb" (és a legveszélyesebb), ugyanis figyelembe fogja venni az *Egér* aktuális helyét is: az elágazásokban azt a nem a háta mögötti szabad oldalszomszédos mezőt fogja választani, amelyhez az *Egér* a legközelebb van. A mozgását életszerűbbnek találtam úgy, hogy csak az elágazásokban változtathat az irányán (valamint a zsákutcákban, ahol visszafordul).



12. ábra: A *Macska* az irányához képest balra, előre és jobbra található mezőket ellenőrzi.

A *Boszorkánnyal* ellentétben a *Macskának* az irányához képest balra, előre és jobbra található mezőre is mindenképpen rá kell lépnie (például ebben a sorrendben), mert előfordulhat, hogy egy később ellenőrzött szomszédos üres mező közelebb van az *Egérhez* a már ellenőrzött üres mezőknél.

A feltételes minimumkiválasztás programozási tételt használhatjuk. Kezdetben a 180 fokos elfordulást (mint legrosszabb esetet) tekintjük a legkedvezőbb iránynak (*legjobb elfordulás* változó), és az 1000 képpontot (mint „végtelen” távolságot) az *Egértől* mérhető legkisebb lehetséges távolságnak (*legkisebb távolság* változó). Ha a három ellenőrzésre szánt oldalszomszédos mező valamelyike üres és közelebb van az *Egérhez* az eddigi legkisebb távolságnál, akkor frissítjük a legkedvezőbb irányt és az *Egértől* mérhető legkisebb lehetséges távolságot. Végül a legkedvezőbb irányba fordítjuk a *Macskát* (12. és 13. ábra).

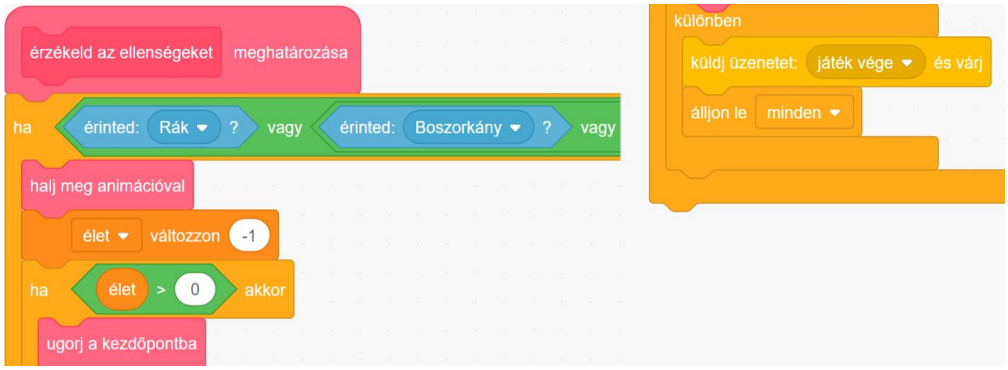


13. ábra: Ha a *Macska* egy adott irányban az *Egérhez* közelebb eső szabad mezőt talál, azt tekinti az addigi legjobb választásnak.[5]

3.8. Az ellenségekkel való ütközés ellenőrzése

Az *Egér* a valamely ellenséggel történő ütközést a végtelen ciklusában meghívott *érzékeld az ellenségeket* eljáráson belül ellenőrizheti, amelynek bekövetkezése esetén az *élet* változó értékét eggyel csökkentenie kell, valamint, ha még maradt a játékosnak élete, akkor vissza kell kerülnie a pálya kezdőpontjába (*ugorj a kezdőpontba*), különben viszont véget ér a játék (14. ábra).

[5] A képről helyhiány miatt az elágazás feltételéből lemaradt az *Ajtóval* és a *Bejárattal* való átfedés ellenőrzése, amelyet a *Falakkal* való átfedés ellenőrzéséhez hasonlóan lehet elvégezni.



14. ábra: Az ellenségekkel való ütközés ellenőrzése[6].

3.9. A többpályás változat létrehozása

A korábbi cikkemben foglalkoztam a több pálya kezelésének módszerével, amelynek a lényegén nem változtat az, hogy a most szóban forgó játék négyzetrácsalapú. A módszer lényege a következő.

Az *Egér* a zöld zászlóra kattintáskor (a játék indításakor) egy-egy üzenet kiküldésével kezdeményezi a játék, majd az első pálya inicializálását (15. ábra, balra).

A *játék előkészítése* üzenet érkezésekor azon változók beállítására kerül sor, amelyeknek csak a játék, és nem minden egyes pálya elején kell kezdőértéket adni: ilyen a *pályák számát* tartalmazó változó, a *pálya* változó (amely az aktuális pálya sorszámát tartalmazza), valamint az aktuális *pontszámot* és *életet* tartalmazó egy-egy változó (15. ábra, középen).

15. ábra: Az *Egér* fő eljárása (balra), valamint tevékenységei a *játék előkészítésekor* (középen) és a *pálya berendezésekor* (jobbra).

A *pálya berendezésekor* üzenetet pedig minden szereplő megkapja, és hatására a szereplők, így például az *Egér*, a pályának megfelelő kezdőpontjukba kerülnek (15. ábra, középen és jobbra), és a *Falak* szereplő is a *pálya* változó értékének megfelelő sorszámú jelmezét állítja be.

Lényeges továbbá, hogy a játékos a pálya megfejtésekor (amikor az *Egér* a *Kijáratnál* van, és már minden *Sajtot* megszerzett), már csak akkor nyeri meg egyúttal a teljes játékot, ha az aktuális *pálya*

[6] A *Macskával* történő átfedés ellenőrzése hasonlóan történhet.

sorszáma megegyezik a *pályák számával*. Különben növelni kell eggyel a *pálya* változó értékét, és ki kell küldeni a *pálya berendezése* üzenetet, amelynek hatására berendezésre kerül a következő pálya, majd folytatódik az *Egér* játékciklusa, aminek köszönhetően a játékos hozzá is láthat a következő pálya megfejtéséhez.

4. Variációs lehetőségek

A bemutatott játéktípus számtalan variációs lehetőséget rejt magában a pálya és a szereplők kinézetének (és így a játék kerettörténetének és hangulatának) a megváltoztatásán túl.

Elkészíthetők például a játéktípusnak olyan variációi, amelyekben a főszereplő „kilép" a négyzetrácsból, és másféle „fizikai törvények" befolyásolják a mozgását, például hat rá a „gravitáció" (ilyen főszereplőirányítást megvalósítottam az előző cikkemben).

Vagy létrehozható egy a *Falak* szereplőhöz hasonló további (például *Akadályok* nevű) szereplő, amelynek a jelmezei az egyes pályák mozdulatlan ellenségeit tartalmazzák, és ennek a szereplőnek az érintése ugyanúgy életvesztéssel jár, mint a találkozás valamely mozgó ellenséggel.

A kulcs mellett vagy helyett szüksége lehet a játékosnak további tárgyak érintésére vagy megszerzésére a pálya bizonyos területeinek az eléréséhez, így például kapcsolók nyithatnak ajtókat.

Meg lehet változtatni a következő pályára lépés feltételét is: például legyen-e kijárat a pályán, vagy össze kelljen-e gyűjteni valamelyik szereplő összes példányát a továbbjutáshoz, vagy csak a pontszámot gyarapítsák?

A mozgó ellenségek mozgási szabályainak is sokféle variációja található ki (vagy leshető el valamilyen létező játékból). Például egy ellenség választhat véletlenszerűen a továbbhaladást biztosító mezők közül. Vagy mozgási útvonalak definiálhatók a pálya megfelelő pontjain elhelyezett, az ellenségeket 90 fokos balra vagy jobbra fordulásra felszólító, a játékos számára nem látható „irányjelző táblákkal" (ezzel a megoldással járőröztek például a *Wolfenstein 3D* című 1995-ös játék ellenséges katonái).

5. Értékelés

Cikkemben – a megelőzőre építve – bemutattam a négyzetrácsalapú akciójátékok legjellemzőbb összetevőinek a megvalósítási lehetőségeit a Scratch programozási környezetben. A „négyzetrácsalapúság" programozási szempontból elsősorban a szereplők elhelyezésének és mozgatásának a módját határozta meg, de ebben a játéktípusban foglalkoztam először a szereplők másolatainak (klónjainak) a kezelésével is, mert a sok mezőből álló pályák nagyon motiválják a sok másolat használatát.

A videójátékok világában az akciójátékok pályái nagyon gyakran négyzetrácsalapúak. Ennek az oka nyilvánvalóan az (amelyet a saját tapasztalataim is alátámasztanak), hogy könnyebb így megtervezni a pályák felépítését, és meghatározni a szereplők (például a főszereplő és az ellenségek) viselkedési szabályait. A szóban forgó játéktípus nagy előnyének tartom, hogy jól megfogalmazható és megvalósítható variációs lehetőségeket rejt magában, ezáltal teret biztosítva a tanulói kreativitásnak. Köszönhetően annak, hogy nagyon sok játék négyzetrácsalapú, számtalan ötlet meríthető (tanárként és diákként is) „valódi" játékok pályáiból és működéséből. Érdemesnek tartom megjegyezni azt is, hogy a szereplők (például az ellenségek) mozgatása kapcsolódhat robotikai típusfeladatokhoz, például az útvonalkövetéshez, a falkövetéshez vagy akár a labirintusok megfejtésének egyéb algoritmusaihoz is.

Gyakran állítják informatikatanárok is néhány tanórányi „Scratch-ezést" követően, hogy „nincs benne több". A korábbi és a jelenlegi cikkemmel célom volt annak a megmutatása is, hogy az egymásra épülő feladattípusok megfelelő összeállításával egy kezdőknek szánt programozási környezetben egy kezdőknek szánt programozási témakörön belül is sokféle programozási fogalom használatát igénylő és igen összetett programok fejlesztéséig lehet eljutni.

Irodalom

1. Bernát, P., Zsakó, L.: *Methods of teaching programming – strategy.* In: New methods and technologies in education and practice: Proceedings of XXX. DIDMATTECH 2017. pp 40-50.
2. Bernát, P.: *Robotika az általános iskolában és a RoboMind programozási környezet* In: Szlávi, Péter; Zsakó, László (szerk.) INFODIDACT 2015, Webdidaktika Alapítvány.
3. Bernát, P.: *Teaching introductory programming by creating animations with Scratch.* In: New Methods and Technologies in Education, Research and Practice 2020. ELTE Informatikai Kar. pp. 30-39.
4. Bernát, P.: *Feladattípusorientált játékfejlesztés a Scratch-ben*. In: INFODIDACT 2017. https://people.inf.elte.hu/szlavi/InfoDidact17/Manuscripts/BP.pdf (utoljára megtekintve: 2022.12.18.).
5. *Action game*. Wikipedia. https://en.wikipedia.org/wiki/Action_game (utoljára megtekintve: 2022.12.18.).
6. Szlávi, P., Zsakó, L.: *Methods of teaching programming.* In: Teaching Mathematics and Computer Science, 1. kötet, pp. 247-257.
7. McManus, S.: *Tanulj meg kódolni 10 lépésben!,* Babilon Kiadó (2017).
8. Vorderman, C.: *Computer Coding Games for Kids,* Dorling Kindersley, London (2015).
9. Egges, A.: *Swift Game Programming for Absolute Beginners,* Apress, New York (2015).
10. Szlávi, P., Zsakó, L.: *Módszeres programozás: Programozási tételek*. NJSZT, Budapest, 1999.