ZÁRÓDOLGOZAT

ANDAHÁZY KOLOS

2016

Space Impact

Java programnyelv

Andaházy Kolos

SZÁMALK-Szalézi Szakközépiskola

Szoftverfejlesztő képzés

Nyilatkozat

a záródolgozat eredetiségéről

Alulírott ..... (név) / .....
(anyja neve, szem.ig. szám) büntetőjogi és fegyelmi felelősségem tudatában kijelentem és aláírásommal igazolom, hogy a záródolgozat saját munkám eredménye. A felhasznált irodalmi és egyéb információs forrásokat az előírásoknak megfelelően kezeltem, a záródolgozat készítésre vonatkozó szabályokat betartottam.

Kijelentem, hogy ahol mások eredményeit, szavait vagy gondolatait idéztem, azt a záródolgozatomban minden esetben, beazonosítható módon feltüntettem, a dolgozatban található fotók és ábrák közlésével pedig mások szerzői jogait nem sértem.

Kijelentem, hogy a záródolgozatom CD/DVD adathordozón található elektronikus változata teljes egészében megegyezik a nyomtatott formával.
Hozzájárulok ahhoz, hogy az érvényben lévő jogszabályok és a Számalk-Szalézi Szakközépiskola belső szabályzata alapján az iskola saját könyvtárában megtekinthető (olvasható) legyen a záródolgozatom.

Budapest, ...év .....hónap.....nap

Tanuló aláírása: .....

Tartalomjegyzék

Ábrajegyzék

1 Bevezetés

A szakdolgozat témájául egy kisebb játékszoftver lefejlesztését választottam, mert úgy gondoltam, hogy egy ilyen összetettebb feladat során rengeteg, máshol is előforduló probléma merülhet fel, ezért az elkészítés során szerzett tapasztalatokat talán később is hasznosíthatom. Mivel tanulmányaim során a Java nyelv állt hozzám a legközelebb, ezért ebben akartam dolgozni.

A Java napjaink egyik legnépszerűbb programozási nyelve, azonban a játékfejlesztés kapcsán csak ritkán hallani róla. Az elmúlt két év során sokat használtam a Java-t, de még bőven van mit tanulnom róla ahhoz, hogy később egy jó Java programozó lehessek. A játék megalkotása során lehetőségem van a nyelv új arcát is megismerni, és ezzel más oldalról is fejleszteni a tudásomat.

Ebben a játékban egy vadászgépet irányítva kell minél több ellenséges gépet megsemmisíteni.

2000-ben megjelent a Nokia 3310, és vele együtt jó pár játék, köztük a Space Impact is.

Ezt a játékot még gyerekkoromból ismerem, és nagy rajongója lettem, mivel komplexebb volt a többi sablonos és egyszerű játéknál. Ennek az emléknek a hatására választottam ezt. Érdekelt az, hogy én magam is elkezdjem lefejleszteni PC-s változatban.

Az eredeti játék lényege, hogy ellenséges objektumok és élőlények legyőzésével teljesítsük a pályákat. Az általam készített változatban a vadászgépet irányító játékosnak egy pályán manőverezve kell minél több ellenséges gépet megsemmisítenie. Azt tervezem, hogy egyszer a közeljövőben még tovább tökéletesítem.

A téma és a nyelv eldöntése után még ki kellett választanom egy fejlesztői környezetet. Tanulmányaim alatt a Netbeans-et használtam, elég jól kiigazodom benne, ezért ezt megfelelőnek tartottam. Ebben a fejlesztői környezetben lehetőségem van arra, hogy ezt a játékot az eredetinek megfelelően 2D-s verzióban alkossam meg.

Szakdolgozatom célja tehát a Space Impact játék egy saját PC-s átiratának megvalósítása, melynek elkészítése során bővíthetem ismereteimet a Java nyelvről.

2 Feladatspecifikáció

A program elindítása után rögtön a játék főmenüje jelenik meg az alábbi menüpontokkal:

- Játék indítása
- Toplista
- Kilépés

A Játék indítása gombbal lehet elindítani a játékot, de mielőtt ez megtörténne egy tájékoztató ablak ugrik fel, ahol az irányításról olvashatunk. Az ok gombra kattintva zárhatjuk be.
A Toplistára kattintva megnyithatjuk a játék során elért 10 legjobb eredményt és a hozzájuk tartozó személyek nevét.
A Kilépés gomb megnyomása után bezáródik a program.
A játék elindulásakor létrejön a játéktér, a játékos hajója, és elkezdenek megjelenni az ellenségek is. Célunk ezeknek a hajóknak a megsemmisítése ezzel megvédeni a játékos mögött található frontvonalat illetve elkerülni az ütközést. Minél több ellenséget lövünk ki annál nagyobb esélyünk lesz felkerülni az toplistára. A játék folyamán a képernyő bal felső sarkában található számláló fogja mutatni a jelenlegi eredményünk.
Tíz darab hajó megsemmisítése után az ellenségek felgyorsulnak.
A hajónk irányításához ezt a 3 billentyűt használjuk:

- W – felfelé haladás
- S – lefelé haladás
- Szóköz vagy Space – lövés

A játék akkor ér véget,k amikor a játékos összeütközik az egyik ellenféllel, vagy ha az ellenség átjut rajtunk és eléri a bal oldali falat.

3 Tervezés

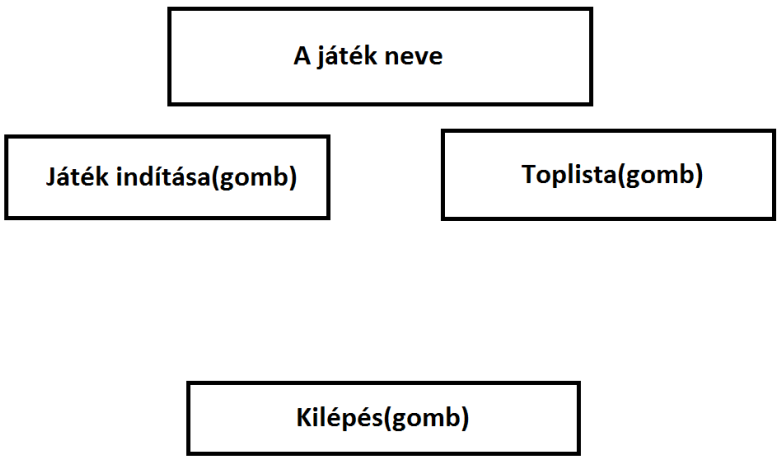
Egy olyan játékprogram elkészítése a cél, ami valamelyest követi a Space Impact játék működési elvét.
A szoftverfejlesztés első lépése a tervezés. Elsősorban meg kell fogalmaznunk a követelményeket, hogy mit várunk el a programtól. Ez általában egy szöveges dokumentum. A követelmények felállítása után meg kell vizsgálni az általunk választott platform lehetőségeit. Java nyelven ezen alkalmazások elkészítése nem olyan egyszerű és a lehetőségek is szűkösebbek, mint például C#-ban, Jelen esetünkben azonban a java nyelv is elegendő, sőt még egy kicsit több kihívást is jelenthet.

A megvalósíthatóság mellett arra is oda kell figyelni, hogy a fejlesztőkörnyezet adottságai pozitív hatással vannak-e a programunkra vagy túl sok a negatívum.
Többször is belefuthatunk tervezési hibába, ami csak a fejlesztés közben derül ki, ilyenkor jön az újratervezés vagy az eltávolítás.
Ezután következik az architektúra megtervezése, amely meghatározza a programunk osztályszerkezetét.

A tervezést követően következik az implementálás, aminek a lényege egy prototípus elkészítése a megtervezett funkciókkal, majd ennek a kibővítése, amíg eljutunk a kész programhoz. Érdemes időnként egy-egy futtatható verziót elmenteni és csak utána folytatni a munkát mivel előfordulhat, hogy valamit elrontunk, és ilyenkor sokat számít, hogy meddig is kell visszamenni, vagyis rengeteg időt és energiát tudunk megspórolni.

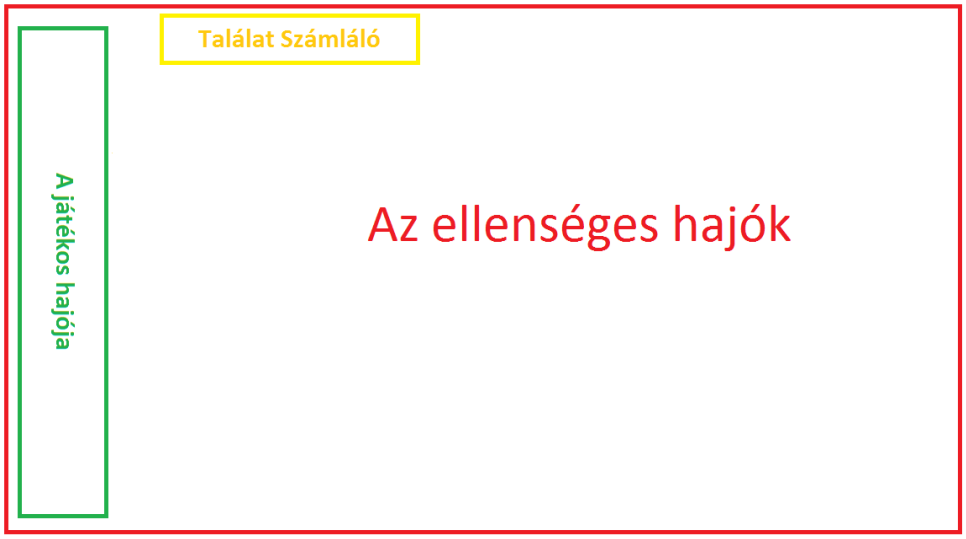
3.1 Menü és játéktér

A program elindítása után egy menü fog minket várni, amelyen az alábbi menüpontok fognak szerepelni:



1. Menü kinézet (Forrás: Saját készítés)

Ezeket egy jFrame ablakon helyezem el. A *játék indítása* gombra való kattintás után egy felugró ablak tájékoztatja a felhasználót, hogy melyik gombokkal tudja vezényelni a játékot. Az ablak bezárása után elindul a játék, ami egy másik felületen játszódik. Ekkor 3 féle különböző elemet láthatunk a pályán. A Játékos hajóját, amellyel a játéktér bal oldalán fel le mozogva kell a megfelelő magasságból elindítani a lövedéket, hogy az összeütközzön és megsemmisítse az ellenséges hajókat. A következő tehát az ellenséges hajók, amelyek a játéktér jobb oldalától a játékos felé mozognak egyenletes sebességgel. A találat számláló azt mutatja meg nekünk, hogy eddig hány ellenséges hajót lőttünk ki. Ezt a számot azért is érdemes figyelni, mert egy bizonyos szám után az ellenséges hajók sebessége megváltozik.



2. Játéktér (Forrás: Saját készítés)

A *toplista* gomb megnyomása esetén a játékok alatt elért 10 legjobb eredményünket fogja kirajzolni nekünk egy külön ablakba.

A *kilépés* gombunk szerepe az ablakok bezárása és program futásának leállítása.

3.2 A Játéktér

Ez az osztály fogja a játékterepet megalkotni. Minden objektumtípus kirajzolására képesnek kell lennie. Egy pontszámlálót is elhelyezünk a képernyő felső részére, amivel az aktuális eredményünket láthatjuk. Ez egy folyamatosan frissülő számot fog mutatni és a lövedék és az ellenség ütközésénél fog eggyel növekedni. Az objektumoknak saját színt kell beállítani a dizájn és a könnyen megkülönböztethetőség végett. A hajók kirajzolása és megfelelőre színezése a paintObjects() metódusban lesz megvalósítva. Minden elemet hozzá kell majd adnunk a játékterephez méghozzá a megfelelő sorrendben. Ügyelnünk kell, hogy az objektumok ne takarják egymást.

3.3 Az Űrobjektumok

Egy absztrakt osztály létrehozása a cél melynek a metódusait ezen osztályokban használunk majd. A játékos, az ellenség és a lövedék. Mindegyiknek a példányosításához minimum 4 alapértéket kell megadnunk.

Az x és y változó fog megfelelni az objektumok megfelelő elhelyezkedéséért, a model adja meg a hajók formáját, és a cellSize adja meg a modellek méretét.

Egy mozgást irányító függvény létrehozása szükséges mivel mind a három objektumtípus valamilyen mozgást fog végezni. A játékos esetében ez egy kicsit másképpen lesz megoldva, mert a másik kettőhöz képest ez felfelé és lefelé fog mozogni és nem vízszintesen.

3.4 A Játékos

A játékos által irányított vadászgép játssza a főszerepet, mivel rajta áll a frontvonal megvédése az ellenséggel szemben. Ha ő elbukik, akkor vége a játéknak. A játékos két irányba fel és le fog tudni mozogni, illetve lövedékeket kibocsájtani az ellenfelekre. A hajónk mozgási sebessége a feladat teljesíthetősége végett lesz megfelelően megadva.
Gépünk rendelkezik majd egy megszabott kinézetettel és színnel. És az osztály az űrobjektumokból fog öröklődni.

3.5 Az Ellenség

Az ellenségek generálása folyamatos lesz a játékmenet során. A pálya jobb oldalától fognak indulni de random pozícióból és random mennyiségben. Ezeknek az objektumoknak egyedi kinézetük és egyedi színük van. Mozgásuk jobbról balra történik és sebességük egyenletes. A játékmenet nehezítéseképpen a sebességüket kicsit változtatni tervezem és a monotonitást is szeretném elkerülni. Ez az osztály szintén az űrobjektumokból fog öröklődni.

3.6 Az ellenség kezelése

Ebben az osztályban lesz az ellenséggel történő folyamatok kezelésének megvalósítása. A hajók generálása lesz az egyik főszerepe. Ennek a metódusnak állandóan futnia kell majd a játék végéig. Ahhoz, hogy időben leállítsuk, a folyamatot ellenőriznünk kell a játékossal illetve a játékos mögött található fallal való ütközést. Mind a két eset vizsgálata nagyon fontos. A hajók sebességének változtatásának kezelése szintén itt lesz megvalósítva. Ekkor gyorsabb egyedeket kell majd létrehoznunk, vagyis ez a folyamat mindenképpen kapcsolatban lesz majd a generáló függvénnyel.

3.7 A lövedék

Létrehozását a játékos által lenyomott billentyű fogja elvégezni. Ennek az objektumnak az ütközése fogja megsemmisíteni az ellenséges hajókat és növelni az eredményünk. Ez is rendelkezik saját színnel és kinézettel.
Ez az osztály is az űrobjektumokból fog öröklődni.

3.8 A lövedék kezelése

A lövedékek létrehozását a játékos fogja szabályozni. Amilyen sokszor nyomjuk meg azt a billentyűt, amelyhez a lövedék generálását szabályozó metódus van kapcsolva olyan sűrűn fognak megjelenni. Természetesen ezt is szeretném szabályozni, mert ha túl sok lövedéket

tudunk kilőni, akkor túl egyszerű lenne a feladatunk. Létrehozásuk tehát elég egyszerű elven alapul. Meg kell határozni a játékos hajójának pozícióját majd annak az orrából fog kiindulni a lövedék ezzel a koordinátával. A továbbiakban lövedékünk egyenletes sebességgel halad az ellenség felé. Ha ütközik az ellenséggel, akkor törlődnek mindketten, de ha nem akkor a pálya szélén eltűnik.

3.9 Hajómodellek

Három kinézetet kell ebben megvalósítani. Egyet a játékosnak, egyet az ellenségnek és egyet a lövedéknek. Ezek a kinézetek fognak a játékmenet során megfelelő pozícióban, színben és méretben megjelenni.

3.10 A toplista

A menüben található gombra kattintva lesz megnyitható ez az ablak.
Minden játék után elmentjük az elért eredményt és bekérünk egy nevet és ezeket kiíratjuk egy fájlba. Toplista ablakunk feladata lesz ennek a fájlnak a beolvasása és tartalmának kiíratása sorrendben. A toplista.txt fájlnak készítése legegyszerűbben úgy oldható meg, ha létrehozunk egy külön osztályt erre a célra. Egy adatként kell kezelnünk az elért eredményt és a bekért személy nevét ahhoz, hogy később a sorba rendezés ne okozzon problémát.

3.11 Főosztály

Terveim szerint ennek az osztálynak az lesz a szerepe, hogy a többi osztály függvényeit meghívja és a futásnak megfelelő sorrendben használja fel.

4 Fejlesztői dokumentáció

4.1 Menü és játéktér(MenuWindow.java & Cancas.java)

A játék elindításakor a főmenü ablak jelenik meg, amin 3 gomb található.

Igyekeztem egy játék nevéhez hű háttérképet beállítani, amivel kicsit feljebb dobhatom a kinézetet. Az ablak alapból úgy van beállítva, hogy a képernyő bal felső sarkában jelenik meg illetve átméretezhető. Ebből kifolyólag írtam egy olyan függvényt, amely a képernyő méretét lekérdezi majd ennek a számnak a felét véve meghatározza, a középpontot majd ennek megfelelően beállítja az ablakunk helyzetét. Az átméretezhetőségen mindenképpen kellett változtatni, mert a fix méretű háttérképünk méretét meghaladva a háttér többi része fehér lesz, és a gombok elhelyezkedése sem középen lesz. Igyekeztem tehát egy olyan fix méretet megadni, amit teljesen le tud fedni a képünk, és a gombok is elférnek rajta.

```
public MenuWindow() {
    this.setResizable(false);
    this.setVisible(true);
    initComponents();
    int xsize = 780;
    int ysize = 500;
    this.setSize(xsize, ysize);
    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();//középre jelenik meg
    int jfrrameWidth = this.getSize().width;
    int jfrrameHeight = this.getSize().height;
    int locationX = (dim.width-jfrrameWidth)/2;
    int locationY = (dim.height-jfrrameHeight)/2;
    this.setLocation(locationX,locationY);
}
```

Játék indítása:

Mielőtt a játék elindulna, egy információs ablak jelenik meg, hogy útbaigazítsa a játékost az irányításról. Ennek az ablaknak bezárása után meghívódik a főosztályunk. Ebben található a saját és a többi osztály metódusai olyan formában, hogy a programunk elindításához ezt az osztályt kell meghívnunk, mivel az összes többinek a meghívása bele van építve. Mindezek után megjelenik a játékterünk és kezdetét veszi a harc.

```
private void jatekinditasaActionPerformed (java.awt.event.ActionEvent evt) {

    String information = "Játék információ!"+"\n"+
                         "Irányítás: "+"\n"+
                         "Felfele haladás - W"+"\n"+
                         "Lefele haladás - S"+"\n"+
                         "Lövés - Space";
   JOptionPane.showMessageDialog (null, information);
   SpaceImpact spi= new SpaceImpact ();
   spi.setResizable (false);

   spi.setDefaultCloseOperation (spi.DISPOSE_ON_CLOSE);
   spi.addWindowListener (new java.awt.event.WindowAdapter () {
    @Override
public void windowClosing (java.awt.event.WindowEvent windowEvent) {
    spi.setKill (true);
   spi.dispose ();
}
});
}
```

A játék elindítása után a játék felülete tárul elénk.

A játékmenet elindulása után a kijelzőn 4 különböző elemet különböztethetünk meg. A játékos által irányított vadászgépet a kijelző bal oldali függőleges skáláján mozogva, a játékos által kibocsájtott lövedékek mozgását a játéktér bal oldalától a jobb oldaláig, az ellenséges gépeket a felület jobb oldalától a bal oldala felé mozogva, és végül a játékos által sikeresen eltalált és kilőtt ellenséges gépek számlálójának mutatóját.

A játék felületének megalkotója a *Canvas* osztály. Ez az úgynevezett vászon, amelyen a játékmenet zajlik. Itt található az objektumokat kifestő,és találatszámlálót felfestő metódusok.

A kinézet és az űrhatás kedvében járva egy csillagos háttérkép lett beállítva.

```
Image img = Toolkit.getDefaultToolkit ().createImage ("test2.jpg");
public void paintBackGround (Graphics g){
g.drawImage (img, 0, 0, null); }
```

Toplista: Megjelenik egy külön JFrame ablak, amely tartalmaz egy JListet, ahol az eddig elért rekordok tömbjének tartalmából készített *HighScore.txt* file adatait olvassa be és jeleníti meg.

```
private void toplistaActionPerformed(java.awt.event.ActionEvent evt) {
    HighScoreWindow hsw = new HighScoreWindow();
    hsw.setVisible(true);
    hsw.beolvas();
}
```

A toplista adatai 2 részből állnak, amelyeket külön elég nehéz lenne rendezgetni, ezért létrehoztam ennek egy külön osztályt *HighScorePlayer* néven. A *HighScorePlayer* típus egy nevet(String) és egy pontszámot(Integer) tartalmaz. Kezdetben nincsenek eredmények azért létre kell hozni egy HighScore.txt fájl és feltölteni 10 darab teszteredménnyel, amelyekhez viszonyítva a később elért eredményekkel könnyen meghaladva feljuthatunk az eredményjelzőre. A fájl létrehozása után következik a fájl visszaolvasása és megvizsgálása egy tömbben, hogy az éppen elért eredményünk vajon változtat e majd a listán. Ha igen akkor a legalacsonyabb elemmel kicserélve majd a tömb sorba rendezése után felülírjuk a fájlunkat, ha pedig nem sikerül jobb eredményt produkálni, akkor az eredeti sorrendet megtartva írjuk felül a fájlt.

Kilépés: bezárja az alkalmazást.

```
private void kilepesActionPerformed (java.awt.event.ActionEvent evt) {
    this.setVisible (false);
    System.exit (0);
}
```

4.2 A játékos (Player.java)

A megvalósításához egy külön osztályt használok *Player* névvel, amelyet a *SpaceObject* absztrakt osztályból származtatok. Ebben az osztályban a főosztály Mozgató metódusait kellett felülírnom. A *moveUp()* és a *moveDown()* függvényeken belül tudom beállítani, hogy a hajóval lehessen mozogni illetve, hogy ne menjen ki a pályáról, ezért a hajónk maximum elmozdulását a pálya széléhez igazítottam.

A *SpaceObject* osztályban létrehoztam a metódusokat és változókat, amelyeket több osztályban is alkalmaznunk kell. A játékos létrehozásánál 4 értéket kell megadnunk:

1. x tengelyen való kezdeti elhelyezkedés: ennek értéke jelen esetben 0 mivel a pálya bal oldalán kell, hogy elhelyezkedjen. Más számot itt nem ajánlott megadunk, mivel a hajónk vízszintesen nem tud mozogni, ezért a nagyobb érték esetén sokkal nehezebb dolgunk lehet.
2. y tengelyen való kezdeti elhelyezkedésének értékét, ez jelen esetünkben 160, ami a pálya tetejétől számítva számoldandó. Mivel hajónkkal függőlegesen tudunk mozogni, ezért a megadott kezdeti elhelyezkedés nem számít.
3. A különböző hajóknak különböző kinézetét egy külön *ShipModels* osztályban hoztam létre ahol a játékos hajója a második elem.
4. Ezzel a számmal állíthatjuk be a hajómodellek képernyőn látható méretét.

```
public Player (int x, int y, int model, int cellSize) {
    super (x, y, model, cellSize);
}
```

A hajó irányítása:

Két irányban mozoghatunk felfele és lefele a „w" illetve az „s" billentyű lenyomásával, lőni pedig az „Szóköz (Space)" gombbal tudunk. A játékos mozgásának sebessége itt állítható a *moveUp()* és a *moveDown()* által tartalmazott szám átírásával. Nem érdemes túl nagy számot megadnunk, mert akkor a hajónk túl nagyokat fog ugrálni és ez a játék élvezetének rovására mehet.

```
public void keyTyped (KeyEvent e) {
                switch (e.getKeyChar ()){
            case 'w':
                player.moveUp (15);
                break;
            case 's':
                player.moveDown (15);
                break;
            case ' ':
                if  (System.currentTimeMillis () - timeLastShot >= delay) {
                shellControl.addShell ();
                timeLastShot = System.currentTimeMillis ();
                }
                break;
    }
}
```

A felfele és lefele mozgás metódusát módosítani kellett a játékosnak megfelelően. A képernyőről kimenést meg kellett akadályozni.

```
public void moveUp (int shift){
    if  (y>10) {
       y -= shift;
    }
}
public void moveDown (int shift){
    if  (y<525) {
       y += shift;
    }
}
```

4.3 Az ellenség (Creep.java)

Az ellenséges hajók szintén egy külön osztályt igényelnek ezt *Creep* néven alkottam meg, amely a játékoséhoz hasonlóan a *SpaceObject*-ből öröklődik.

Ezeknek a hajóknak a létrehozása egy külön osztályt igényelt amit *CreepControll* néven hoztam létre. Ez az osztály generálja le nekünk az ellenségeket random kiindulási koordinátával.

Az ellenség létrehozásánál 5 értéket kell megadnunk.:

1. X tengelyi elhelyezkedés: a program automatikusan generálja le nekünk ezt az értéket.
2. Y tengelyi elhelyezkedés: a program automatikusan generálja le nekünk ezt az értéket.
3. A „ShipModels” osztályban az ellenfél hajóinak a megfelelő sorszáma az egyes.
4. A hajó baloldal irányú mozgási sebességét határozza meg.
5. Ezzel a számmal állíthatjuk be a hajómodellek képernyőn látható méretét.

```
public Creep (int x, int y, int model, int speed, int cellSize) {
    super (x, y, model, cellSize);
    this.speed = speed;
}
```

Az osztály végén egy haladást kezelő funkciót találunk. Itt a SpaceObject move metódusát írjuk felül. x -= speed mivel a negatív irány felé haladunk az x tengelyen.

4.4 Az ellenség kezelése (CreepControl.java)

AZ updateCreeps() egy összetettebb metódus. Itt állítjuk be a hajók felgyorsulását tíz darab ellenség kilövése után, ellenséges hajókat generál majd vizsgálja, az ellenség ütközik-e a játékossal, és ha igen akkor a folyamat lezárul. A timeToSpawn változót is itt tudjuk megváltoztatni, ami a generált ellenségek számát szabja meg.

Az ellenséges hajók állandó generálása történik ebben az osztályban. A hajót példányosítjuk majd random megjelenítjük a játéktéren egyszerre többször is.

```
    public void updateCreeps (){
        if  (timeToSpawn <= 0) {
            Creep creep = new Creep ( SpaceImpact.WIDTH, 0, 0, 2, 3);
            if  (creepList.isEmpty ()) {
               creep.setY (random.nextInt (SpaceImpact.HEIGHT - creep.getShipHeight ()));
            }else{
                boolean collids;
                do{
                    collids = false;
                    creep.setY (random.nextInt  (SpaceImpact.HEIGHT  -  creep.getShipHeight
()));
                    for  (SpaceObject sp : creepList){
                        if  (creep.isCollision (sp)) {
                             collids = true;
                             break;
                        }
                    }
                } while (collids);
            }
            timeToSpawn = random.nextInt (150) + 50;
            creepList.add (creep);
        }
        timeToSpawn--;
    }
```

Amíg a hajó, amíg nem éri el a pálya bal oldalát addig mozog.

```
public boolean update(){
    for (int i = 0; i < creepList.size(); i++) {

        if (creepList.get(i).getX() <= 0) {
            return false;
        }else{
            creepList.get(i).move();
        }
    }
    return true;
}
```

4.5 Hajómodellek (ShipModels.java)

Három különböző modell byte képei találhatóak ebben az osztályban.
Első az ellenséges hajók modellje. A játék folyamán ezek piros színben vannak jelen egyszerre többen is.
Második a játékos hajójának modellje, ezzel kék színben találkozhatunk.
Harmadik pedig a játékos által kibocsájtott lövedék modellje, ami sárga színű.
Ezeket egy byte típusú három dimenziós tömbbe tároljuk.
A hajómodellek színeit a *Canvas* osztályon belül állítom be. Ezeket később hozzá kell adni a kirajzoló függvényhez, hogy ez valóban teljesüljön. Igyekeztem jól látható színeket használni, hogy kellően eltérjenek a sötét háttértől. A színen kívül a hajómodellek mérete is állítható a *Player*, a *Shell* illetve a *Creep* osztály cellSize változójának átírásával.

```
   public static final byte[][][] shipCreep = {{
//ellenség
   {0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0},
   {0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0},
   {0 , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0},
   {0 , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0},
   {1 , 1 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1},
   {1 , 1 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1},
   {1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1},
   {1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1},
   {1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1},
   {0 , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0},
   {0 , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0},
   {1 , 1 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1},
   {1 , 1 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1},
   {1 , 1 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1}, },
   {
//player
   {1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0, 1, 1},
   {1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0, 1, 1},
   {1 , 1 , 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1, 1, 1},
   {1 , 1 , 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1, 1, 1},
   {0 , 0 , 1 , 1 , 1 , 1 , 0 , 0 , 1 , 1 , 1 , 1, 0, 0},
   {0 , 0 , 1 , 1 , 1 , 1 , 0 , 0 , 1 , 1 , 1 , 1, 0, 0},
   {0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0, 0, 0},
   {0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0, 0, 0},
   {0 , 0 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1, 0, 0},
   {0 , 0 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1, 0, 0},
   {0 , 0 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1, 0, 0},
   {0 , 0 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1, 0, 0},
   {0 , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1, 0, 0},
   {0 , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1, 0, 0},
   {0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0, 0, 0},
   {0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0, 0, 0},
   {0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0, 0, 0},
   {0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0, 0, 0},
   {0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0, 0, 0},
   {0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 0 , 0 , 0 , 0, 0, 0},
   },
      {///lövedék
         {1}, {1}, {1}
      }
   };
}
```

Mindegyik modellnek megvan a maga színe, amelyeket az egyesek helyén bájtonként átszínezünk, a nullások helyén meg a háttérszín jelenik meg.

```
public void paintObjects ( List<SpaceObject> list, Color color, Graphics g ){
    for (SpaceObject sp : list){
        byte[][] ship = ShipModels.shipCreep[sp.getModel ()];
        for  (int i = 0; i < ship.length; i++) {
            for  (int j = 0; j < ship[0].length; j++) {
                if  (ship[i][j] == 1) {
                    g.setColor (color);//felveszi a háttér színt
          g.fillRect (i * sp.getCellSize () + sp.getX (), j * sp.getCellSize () + sp.getY
          (), sp.getCellSize (), sp.getCellSize ());
                }
            }
        }
    }
}
```

4.6 A lövedék (Shell.java)

Ez az objektum egy külön hajónak felel meg, amely saját kinézettel és színnel is rendelkezik akár csak a játékos vagy az ellenfél hajója. Ezeket a lövedékeket a játékos bocsájtja ki az ellenfél hajóival szemben, amelyek ha összeütköznek, akkor mindketten megsemmisülnek. A szóköz lenyomásának következményeképpen hozható létre.

Az lövedék létrehozásánál 5 értéket kell megadnunk:

1. A hajó jobb oldal irányú mozgási sebességét határozza meg.
2. X tengelyi elhelyezkedés: A program automatikusan generálja le nekünk ezt az értéket.
3. Y tengelyi elhelyezkedés: A program automatikusan generálja le nekünk ezt az értéket.
4. A *ShipModels* osztályban az ellenfél hajóinak a megfelelő sorszáma a kettes.
5. Ezzel a számmal állíthatjuk be a hajómodellek képernyőn látható méretét.

```
public Shell(int speed, int x, int y, int model, int cellSize) {
    super(x, y, model, cellSize);
    this.speed = speed;
}
```

Az osztály végén egy haladást kezelő funkciót találunk. Itt a *SpaceObject move()* metódusát írjuk felül. x += speed mivel a pozitív irány felé haladunk az x tengelyen.

```
@Override
public void move() {
    x += speed;
}
```

4.7 Lövedékkezelő (ShellControl.java)

Ahhoz, hogy létre tudjuk hozni a lövedéket, mindenek előtt ki kell számítanunk a játékos hajóorrának pozícióját. A space billentyű lenyomásával ebből a pozícióból fog a lövedék elindulni az ellenség felé.

```
public void addShell (){
    Shell shell = new Shell (3, player.getX () + player.getShipWidth (), 0, 2, 3);
    shell.setY ( (player.getY () + player.getShipHeight () / 2) - shell.getShipHeight
() / 2);
    shellList.add (shell);
}
```

A lövedék kitörlődik, ha eléri a pálya szélét, de amíg ez nem történik meg halad tovább. A főosztályban azt is vizsgáljuk, hogy ha a lövedékünk útközben összeütközik egy ellenféllel, akkor mind a két objektumot megsemmisítjük, mielőtt bármelyik is elérné a pálya szélét.

```
public void update (){
    for  (int i = 0; i < shellList.size (); i++) {
        if  (shellList.get (i).getX () > SpaceImpact.WIDTH) {
            shellList.remove (i);
        }else{
            shellList.get (i).move ();
        }
    }
}
```

4.8 Az űrobjektum (SpaceObject.java)

Ez egy absztrakt osztály, amely meghatározza a többi objektumhoz szükséges változókat és metódusokat. Itt hozom létre az x és az y változót, ami később a felületen való elhelyezkedésért felel majd vízszintes illetve függőleges irányban, cellSize ez felel majd a hajónak a méretéért, a model száma adja meg a 3 modell közül, hogy éppen melyiket használjuk.

A mozgásért felelős metódusok vázát is itt adjuk, meg amelyeket később a többi osztályban megfelelően módosítunk. Itt hozzuk létre az *isCollision()* függvényt is, amelyet a *CreepControll* osztályban használtunk arra, hogy vizsgáljuk az ellenség és a játékos ütközését.

```
public boolean isCollision(SpaceObject sp){
    return x  < sp.getX() + sp.getShipWidth() &&
           x + shipWidth > sp.getX() &&
           y < sp.getY() + sp.getShipHeight() &&
           y + shipHeight > sp.getY(); }
```

4.9 Toplistajátékos(HighScorePlayer.java)

Ha még nem létezik a Highscore.txt akkor létrehozzuk tíz darab tömbben tárolt kezdeti eredménnyel. Később ezeket az eredményeket fogjuk felülírni.

```
    public HighScorePlayer[] filemaker(){
        HighScorePlayer[] highscore = new HighScorePlayer[10];
        File f = new File("Highscore.txt");
        String fileoutput = "";
        if (!f.exists()) {
            try {
                FileWriter fw = new FileWriter("Highscore.txt");
                highscore[0] = new HighScorePlayer("Béla", 10);
                highscore[1] = new HighScorePlayer("Réka", 9);
                highscore[2] = new HighScorePlayer("Kati", 8);
                highscore[3] = new HighScorePlayer("aaaa", 7);
                highscore[4] = new HighScorePlayer("bbbb", 6);
                highscore[5] = new HighScorePlayer("asd", 5);
                highscore[6] = new HighScorePlayer("Elemér", 4);
                highscore[7] = new HighScorePlayer("Józsi", 3);
                highscore[8] = new HighScorePlayer("Nóri", 2);
                highscore[9] = new HighScorePlayer("Géza", 1);
                for (int i = 0; i < highscore.length; i++) {
                    fileoutput += highscore[i].playert+"&"+highscore[i].scoret+"₩n";
                }
                fw.write(fileoutput);
                fw.close();
            } catch (IOException ex) {
                Logger.getLogger(HighScorePlayer.class.getName()).log(Level.SEVERE,  null,
ex);
            }
        }
        return highscore;
    }
```

Beolvassuk a Highscore.txt adatait és eltároljuk őket egy HighScorePlayer típusú tömbben. Ha a játékmenet végén sikerül egy toplistára felkerülő eredményt elérni, akkor ezt az eredményt a listánk megfelelő helyére beillesztve felülírjuk a fájlt.

```
        public HighScorePlayer[] Open(){
        String fileoutput2 = "";
            String input;
            try {
                Scanner sc = new Scanner(new File("HighScore.txt"));
                int i = 0;
                while(sc.hasNextLine()){
                    input = sc.nextLine();
                    String [] parts = input.split("&");
                    HighScorePlayer uj1 = new HighScorePlayer(playert, scoret);
                    uj1.playert = parts[0];
                    uj1.scoret = Integer.parseInt(parts[1]);
                    hgsc[i] = uj1;
                    i++;
                }
                change();
                sort(hgsc);//sorbarendezés

            } catch (FileNotFoundException ex) {
                Logger.getLogger(HighScorePlayer.class.getName()).log(Level.SEVERE,   null,
ex);
            }
                FileWriter fw2;
                try {
                    fw2 = new FileWriter("HighScore.txt");
                        for (int j = 0; j < hgsc.length; j++) {
                            fileoutput2 += hgsc[j].playert+"&"+hgsc[j].scoret+"Wn";
                        }
                    fw2.write(fileoutput2);
                    fw2.close();
                } catch (IOException ex) {
                    Logger.getLogger(HighScorePlayer.class.getName()).log(Level.SEVERE,
null, ex);
                }
          return hgsc;}
```

Az Open() metódusban kerül meghívásra a change() és a sort() függvény. A change() függvény vizsgálja meg, hogy a játék után elért eredményünk nagyobb-e a toplista legkissebb eredményénél. Ha igen akkor ezt kicseréljük az új eredményre és névre.

```
public void change(){
        if(hgsc[9].scoret < SpaceImpact.score){
          HighScorePlayer uj = new HighScorePlayer(playert, scoret);
            uj.playert = SpaceImpact.name;
            uj.scoret = SpaceImpact.score;
            hgsc[9] = uj;
        }
}
```

A sort() függvény oldja meg a tömbünk sorba rendezését, így a legutolsó helyre felvitt új eredmény a toplista megfelelő helyére kerül majd.

```
public void sort(HighScorePlayer[] highscore)
 {
     int i = 8;
         while (highscore[i + 1].scoret > highscore[i].scoret && i != 0)//9 > 8
         {
             HighScorePlayer seged = highscore[i + 1];//seged = 9
             highscore[i + 1] = highscore[i];//9 = 8
             highscore[i] = seged;// 8 = 9
             i--;
         }
         if(highscore[1].scoret > highscore[0].scoret){
             HighScorePlayer seged = highscore[1];//seged = 9
             highscore[1] = highscore[0];//9 = 8
             highscore[0] = seged;
         }
 }
```

4.10 Főosztály (SpaceImpact.java)

Ebben az osztályban 5 metódus található:

Az első és egyben legfontosabb az időzítő indító függvény. Amikor elindítjuk, a játékot akkor ez a függvény elindítja az időzítőt és ennek megfelelően a meghívott metódusok életbe lépnek. Létrejönnek az objektumok, elindul a pontszámszámláló és persze fut közben az ellenőrző

függvény is, ami azt vizsgálja, hogy mikor történik ütközés. Ha a lövedék és az ellenfél hajói ütköznek, akkor azok kitörlődnek, de ha a játékos ütközik az ellenféllel vagy az ellenfél ütközik a játékos mögötti falnak, akkor az időzítő leáll. Leállás után bekérjük a játékos nevét majd az elért pontszáma alapján módosítjuk toplistát.

```
public void startTimer(){
    ActionListener al = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
        hg.filemaker();
        if (kill) {
        timer.stop();
        return;
        }
            canvas.setSzoveg(score);
            SpaceImpact.this.repaint();
            removeObjectsWithCollide();
            if(!creepControl.update() || isCollideWithPlayer() ) {
                timer.stop();
                nevbeker();
            }
            creepControl.updateCreeps();
            shellControl.update();
        }
    };
    timer = new Timer(10,al);
    timer.start();
    this.repaint(); }
```

A második a névbekérő metódus. Külön osztályt nem csináltam ennek az ablaknak, mert ebben a függvényben kódból létrehozom. Ez az egyszerű ablak egy textfieldet és egy gombot tartalmaz, ami a játék utáni név bekéréséért felel.

```
public void nevbeker(){
 String string;
JFrame frame = new JFrame("textfield");
JPanel jp = new JPanel();
frame.setSize(200,100);
```

```
JTextField textfield = new JTextField(9);
JLabel jl = new JLabel("Kérem adja meg a nevét!");
JButton jb = new JButton("Ment");
jb.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            name = textfield.getText().toString();
            removeDiacritics(name);
            frame.setVisible(false);
            setVisible(false);
            hg.Open();
        }
});
jp.add(jl);
jp.add(textfield);
jp.add(jb);
frame.add(jp);
frame.setVisible(true);
Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
int jfrrameWidth = frame.getSize().width;
int jfrrameHeight = frame.getSize().height;
int locationX = (dim.width-jfrrameWidth)/2;
int locationY = (dim.height-jfrrameHeight)/2;
frame.setLocation(locationX,locationY);
}//nevbeker()
```

A harmadik az játékossal ütközés(isCollideWithPlayer). A metódus neve egyben meg is határozza, hogy azt vizsgálja, mikor ütközik a játékos az ellenfél hajóival.

```
public boolean isCollideWithPlayer(){//collide ütközik
    for (int i = 0; i < enemyList.size(); i++) {
            if (enemyList.get(i).isCollision(player))  {
                return true;
            }
    }
    return false;
}
```

A negyedik az objektumok törlése ütközés esetében(removeObjectsWithCollide). Itt azt vizsgáljuk, hogy a lövedék és az ellenfél mikor ütközik. amikor ez bekövetkezik, akkor a függvény mindkét objektumot törli.

```
public void removeObjectsWithCollide(){//ütközésnél törlünk
    for (int i = 0; i < enemyList.size(); i++) {
        for (int j = 0; j < shellList.size(); j++) {
            if (enemyList.get(i).isCollision(shellList.get(j)) ) { /////????;
                enemyList.remove(i);//ellenfél eltünik
                shellList.remove(j);//lövedék eltünik
                score++;
                break;
            }
        }
    }
}
```

Az ötödik a billentyűkezelő(keyTyped). Itt állítjuk be az irányításhoz használt billentyűket. A lövedékeket kezelő billentyűzetre itt lehet időzítőt rakni, hogyha azt szeretnénk, ha csak ritkábban tudnánk lőni ezzel nehezítve a játékot.

Ennek a futtatásához a menüben található Játék indítása gombra kattintva lehet meghívni és elindítani.

```
public void keyTyped(KeyEvent e) {
                switch(e.getKeyChar()){
            case 'w':
                player.moveUp(15);
                break;
            case 's':
                player.moveDown(15);
                break;
            case ' ':
                if (System.currentTimeMillis() - timeLastShot >= delay) {
                shellControl.addShell();
                timeLastShot = System.currentTimeMillis();
                }
                break;}}
```

5 Tesztelés

A program tesztelése közben több problémával is találkoztam. A programozó sokat keresgél, míg megtalálja a számára éppen megfelelő kódot. Sajnos ezeknek a kódoknak a beillesztése és a fejlesztői programmal való elfogadtatás nem elegendő. Többször is elvétettem azt a problémát, hogy az ablak háttérképe nem jelent meg valamilyen okból. Arra számítottam, hogy ez egy sokkal egyszerűbb művelet, de tévedtem. A talált kép mérete nem mindig felel meg az elképzeltnek. Van, hogy túl sötét ezért nem lehet megfelelően látni a többi elemet. Ha nem teljes méretű ablakot csinálunk és ezt nem egy teljes méretű képpel fedjük le, akkor az ablak méretezhetőségét ajánlott lefixálni, mert különben az ablak túl nagyra nyitása esetében a kép méretét meghaladó rész fehér lesz és igénytelen. Ügyelni kellett a kép megfelelő elhelyezkedésére a rétegek között vagy különben kitakarhatja a többi elemet. Találtam olyan kódot ahol a kép elérési útját a saját gépeden található mappákon keresztül érjük el, de ezzel sajnos az a probléma, hogy másik gépen ez a funkció nem fog működni, csak ha ezt külön beállítjuk. Korábbi tapasztalataimból kiindulva számítottam erre. Arra jutottam tehát, hogy másik kódot használok, amit elég egyszer egy gépen megírni és az onnantól már működni is fog. A menü hátterét ezzel az univerzális kóddal sikerült is megoldanom.

Az ablakok alapértelmezetten sajnos nem középen, hanem a képernyő bal felső sarkában jelennek meg. Ezeknek a középre igazítása nem okozott különösebb problémát, pár kódsorból megoldottam.

Sokat kísérleteztem a hajó irányításának tesztelésével és a következőkre jutottam. Nem szabad a játékos hajójának sebességét túl gyorsra állítani, mert akadozott mozgást eredményez. Próbálgatások közepette, amikor túl magasra illetve túl alacsonyra mentünk hajónk kiment a pályáról. Erre nem volt megoldás az ablak méretezhetőségének lefixálása. Ennek a megoldása a *Player* osztályban található moveUp() és moveDown() függvényben található, ahol a játékos hajójának az y tengelyen való elmozdulását a pálya szélességéhez igazítottam.

Többen is jelezték felém, hogy nem tudnak lőni mozgás közben, de ez direkt így van, mert emlékeim szerint a mobilon játszott verzió irányítása is így volt megoldva.

Az ellenségek gyorsaságának és számának korlátozását szintén tesztelgetésekkel lehetett a megfelelőre beállítani, mert nem gondoltam, hogy a sebesség megváltoztatása kettőről háromra annyira felgyorsul, hogy rövid időn belül vége is lesz a játéknak. Ebből az okból csak egyes és kettes fokozatot használok.

A toplistának megoldása tartott a legtovább. Minden menet végén függetlenül az elért eredménytől bekérjük a játékos nevét, de az ékezetes és egyéb furcsa neveknél a toString()-es

megoldás nem volt elég. Ezeknek a neveknek a helyes értelmezéséhez és felhasználásához egy külön metódust kellett készíteni. Sokáig az első eredmény felkerülésénél a bekért név helyett egy null értéket kapott, majd a következő felvitelnél az elsőnek bekért nevet használta fel. Ennek okára nem sikerült teljesen rájönnöm, de a sorrendek és pár funkció meghívásának helyét megváltoztatva megoldódott.

6 Továbbfejlesztési Lehetőségek

Az eredeti játék megvalósítása vagy átdolgozása több időt igényel, ebből kifolyólag számos továbbfejlesztési lehetőség adódik:

- Ellenséges hajók váltakozása különböző méretben és kinézetben, amelyek hullámokban érkeznek.
- Főellenséges hajók a pályák végén. Ezen hajók szívósabbak és tudnak lőni is.
- Több pálya más kinézettel és hajókkal.
- Játékfelület nehezítése akadályokkal.
- Speciális lövedékek és élet bevezetése.
- Szüneteltetési lehetőség.

7 Felhasználói dokumentáció

A játék elindítása után a főmenü fog megjelenni, amin 3 gomb található.

- játék indítása
- toplista
- kilépés

Az játék indító gombra kattintva felugrik egy ablak, ami megmutatja nekünk, hogy melyik billentyűt mire tudjuk majd használni. Az ok gombra kattintva elindul a játék.
A játék vége után felugrik egy kis ablak, ahova be kell írnunk a nevünket majd a ment gombra kattintani. Ekkor visszatérünk a fűmenübe ahol a toplista gombra kattintva megnézhetjük az eddig elért legjobb 10 eredményt, vagy új játékot kezdhetünk.
Miután meguntuk a játékot a kilépés gombbal bezárhatjuk a programot.

7.1 Rendszerkövetelmények

Minimum hardver konfiguráció.

Az IDE minimum képernyő felbontása 1024x768 pixel.

- **Microsoft Windows XP Professional SP3/Vista SP1/Windows 7 Professional:**
  - **Processor:** 800MHz Intel Pentium III or equivalent
  - **Memory:** 512 MB
  - **Disk space:** 750 MB of free disk space
- **Ubuntu 9.10:**
  - **Processor:** 800MHz Intel Pentium III or equivalent
  - **Memory:** 512 MB
  - **Disk space:** 650 MB of free disk space
- **Macintosh OS X 10.7 Intel:**
  - **Processor:** Dual-Core Intel
  - **Memory:** 2 GB
  - **Disk space:** 650 MB of free disk space

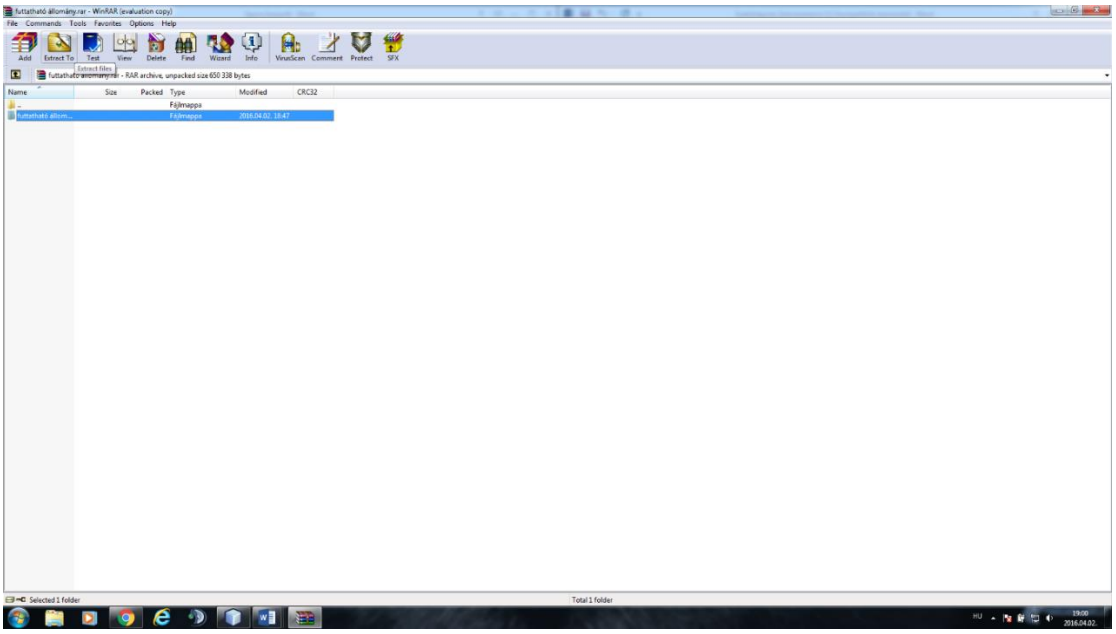
Ajánlott hardver konfiguráció.

- **Microsoft Windows XP Professional SP3/Vista SP1/Windows 7 Professional:**
  - **Processor:** Intel Core i5 or equivalent
  - **Memory:** 2 GB (32-bit), 4 GB (64-bit)
  - **Disk space:** 1.5 GB of free disk space
- **Ubuntu 12.04:**
  - **Processor:** Intel Core i5 or equivalent
  - **Memory:** 2 GB (32-bit), 4 GB (64-bit)
  - **Disk space:** 1.5 GB of free disk space
- **OS X 10.8 Intel:**
  - **Processor:** Dual-Core Intel
  - **Memory:** 4 GB
  - **Disk space:** 1.5 GB of free disk space

8 Telepítés és futtatás

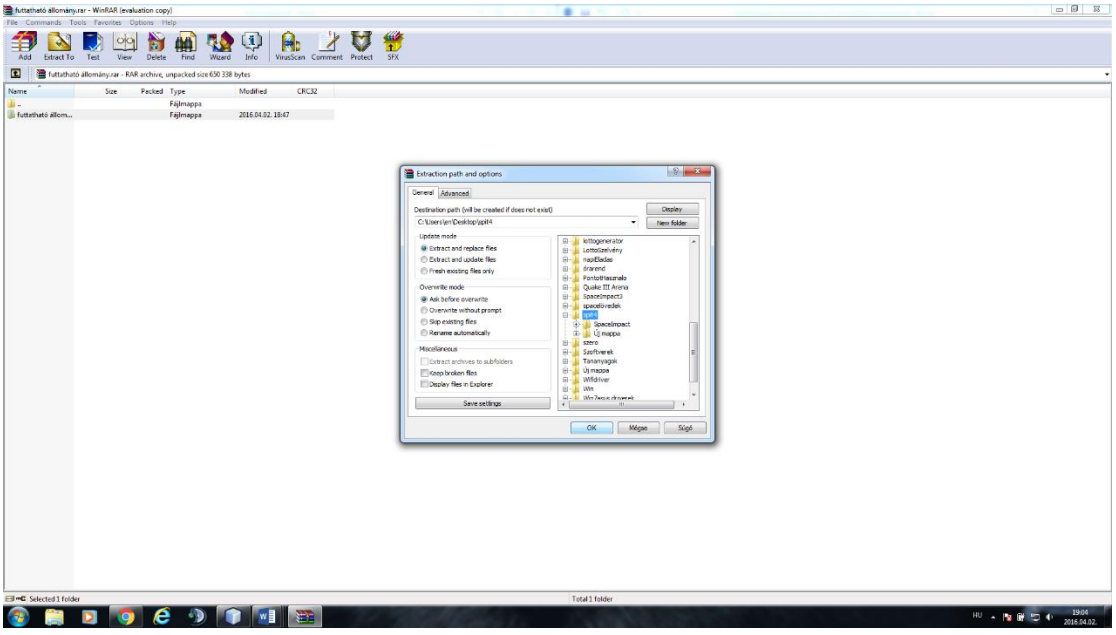
A program használatához szükségünk lesz a tömörített fájlra, ami tartalmazza programot. Az alkalmazás használatához mindenekelőtt ki kell csomagolnunk a tömörített fájlt. Ennek a műveletnek az elvégzéséhez a WinRAR programot használom.

Bal gombbal rákattintva kijelölöm a jelen esetünkben futtatható állomány mappáját, majd a felette található „Extract To” ikonra kattintunk.



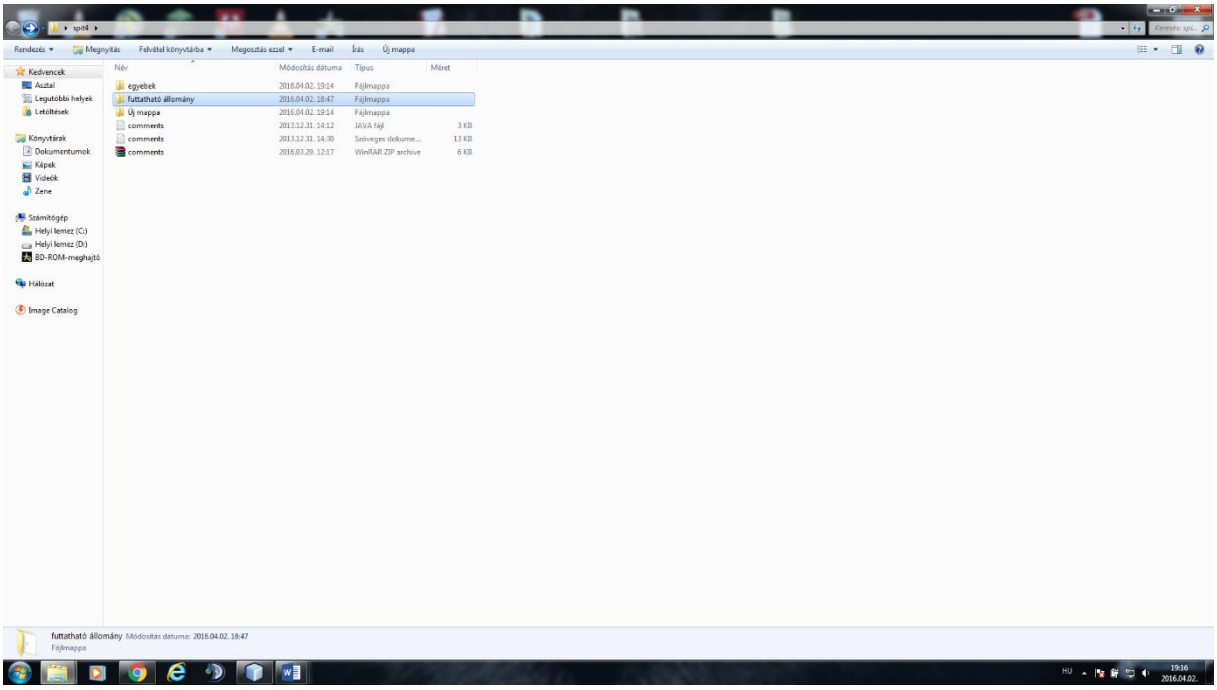
3. Kitömörítés (Forrás: Saját készítés)

Következő lépésként válasszuk ki a kitömörítés célmappáját, majd kattintsunk az ok gombra.



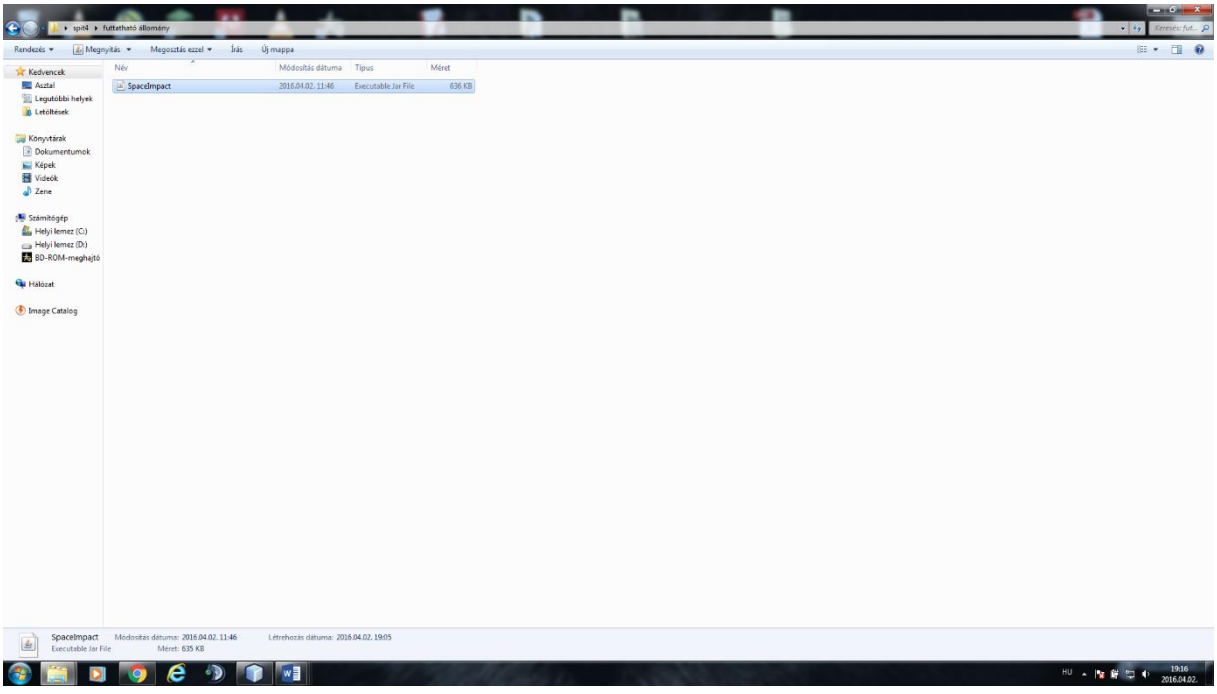
4. Kitömörítés helyének kiválasztása (Forrás: Saját készítés)

A sikeres kitömörítés után menjen a célkönyvtárba és keresse meg a kitömörített mappánkat.



5. Kitömörített mappa helye(Forrás: Saját készítés)

A mappán belül kattintson a SpaceImpact nevezetű jar fájlra.



6. Futtatható fájl (Forrás: Saját készítés)

9 Összefoglalás

Láthatjuk, hogy a Java SE platform mennyi leküzdendő akadályt állít a játékfejlesztő elé, bár ugyanakkor sok helyen segíti is őt. A most megszerzett tudást később akár bonyolultabb játék elkészítését is lehetővé teszi, a platform azonban még további számos lehetőséget tartogat számunkra, emellett folyamatosan fejlődik.

Az elkészült játék fejlesztése során lehetőségem volt jobban megismerni a platformot és megtapasztalni annak erősségeit és hiányosságait. Fejlesztés közben több ötlet is felmerült, amelyek nem kerültek bele a programba, azonban megvalósításuk további érdekes kihívásokat tartogat. Ezek lehetnek például a különböző formájú, méretű és tulajdonságú ellenséges űrhajók hozzáadása a játékhoz.

10 Irodalomjegyzék

[1] Kódok , http://stackoverflow.com/

[2] Ablak középreigazítása , https://www.youtube.com/watch?v=KHlEbYRSz8Q

[3] Ablakméretezés , https://www.youtube.com/watch?v=KDJar8mml0o

[4] Hatterkep , https://www.youtube.com/watch?v=r5S8TI5W2Q8

[5] Szerkezet , https://www.youtube.com/watch?v=UUUXa4so0uI

[6] Névbekérés , https://www.youtube.com/watch?v=KHlEbYRSz8Q

[7] Online Java Könyv , http://nagygusztav.hu/sites/default/files/csatol/java_programozas_1.3.pdf

[8] Netbeans 8.0.2 hardverigény , https://netbeans.org/community/releases/80/relnotes.html

[9] Háttérkép beállítás , http://www.java2s.com/Tutorial/Java/0240__Swing/CreatingFrameswithabackgroundimage.htm