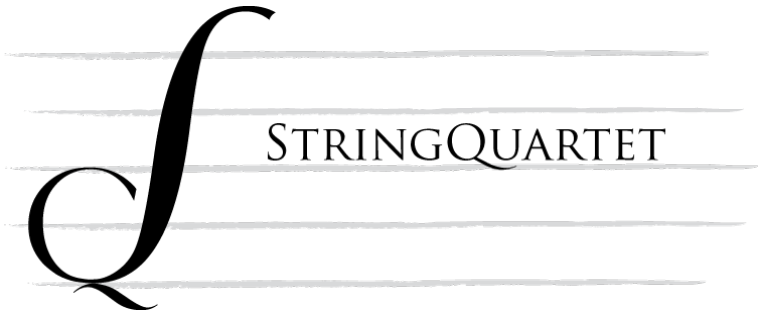
Boulder Dash

4 – String Quartet

Konzulens:
László Zoltán



Csapattagok

Farkas Ádám Attila	GR1M48	wolfee.dm@gmail.com
Siklósi Zsolt	Y42BZ3	zsolt.siklosi@gmail.com
Tóth-Máté Ákos	K4VE4A	akos@tothmate.com
Zsolnai Károly	F29RZ0	keeroy@gmail.com

2009. május 14.

Tartalomjegyzék

1. Követelmény, projekt, funkcionalitás

1.1. Követelmény definíció

1.1.1. A program célja, alapvető feladatai

Az elkészítendő program egy játék, amelyben egy kincskereső indiánt irányítva, előre megtervezett pályák kijáratait kell megtalálni a pálya területén való mozgással, ásással, és az esetlegesen rátámadó ellenfelek kikerülésével vagy megsemmisítésével.

A fejlesztőcsapat célja egy olyan kész program előállítása, mely teljes mértékben kielégíti a specifikációban megkövetelteket, és ami minden olyan gépen lefordítható, futtatható, mely megfelel a későbbiekben megfogalmazott követelményeknek.

1.1.2. A felhasználói felület

A kész program végső változata billentyűzet és egér felhasználásával lesz irányítható, grafikus felhasználói felülettel fog rendelkezni.

1.1.3. A program futtatásához szükséges követelmények

A futáshoz szükséges, hogy a felhasználó számítógépére telepítve legyen a Java Runtime Environment (JRE), a program hardverigénye megegyezik a Sun által meghatározott minimum-konfigurációval: Pentium 166 MHz vagy annál gyorsabb processzor, minimum 32 MB RAM és 125 MB hely a háttértárolón

1.1.4. A szoftver fejlesztésével kapcsolatos alapkövetelmények, elvek, célok

Modellhűség:

Az elkészült szoftver minőségének egyik igen lényeges fokmérője az, hogy mennyire szigorúan elégíti ki a specifikációban meghatározott követelményeket. A fejlesztőcsapat fontosnak tartja ennek szem előtt tartását, már a tervezés kezdetétől fogva.

Továbbfejleszthetőség:
: A kész termék működése, felépítése teljes egészében és részleteibe menően megérhető a hozzáadott dokumentáció felhasználásával. A fejlesztők a tervezés folyamán figyelembe veszik azt a tényezőt, hogy a modell szerkezete lehetővé tegye, hogy az esetleges későbbi továbbfejlesztés akadálymentesen és gördülékenyen megtörténhessen.

Modularitás:
: A fejlesztőcsoport hatékony tervezési minták felhasználásával lehetővé teszi, hogy a program egyes részei jól elhatárolhatóak legyenek egymástól, annak részmoduljai a lehető legkevésbé legyenek összecsatolva, és külön-külön felhasználhatóak, egyenként tesztelhetők legyenek.

Teljesítmény, optimalizálhatóság:
: A teljes tervezési és fejlesztési folyamaton túlmenően hangsúlyos a felhasznált algoritmusok optimalizálásának megszervezése. A csapattagok az optimalizálási elemzések során, ahol lehetséges, matematikai ismereteikre támaszkodva finomítják, hatékonyabbá teszik a felhasznált algoritmusokat, eljárásokat. A program ezáltal kész lesz arra, hogy a specifikációban szereplő számítógép-konfiguráción élvezhetően, maradéktalanul fusson.

Felhasználhatóság:
: A készítők célja, hogy egy olyan program szülessen, mely könnyen kezelhető, melynek használata azonnal elsajátítható, akár a felhasználói leírás elolvasása nélkül is.

Perzisztencia:
: Egy igen fontos irányelv, miszerint a program futása során tapasztalt legfontosabb információkat fájlokba kell írni, így a tesztelés során ellenőrizhető, hogy mikor, milyen körülmények között merül fel a megfogalmazott probléma.

Modern technológiák használata:
: Az UML és a RUP rendszerbe foglalja a rendszertervező és fejlesztő

mérnökök által levezényelt projektjeiből levont következtetéseket, minimálisra csökkenti a tervezés és az implementáció közötti szakadékot, pontos képet ad a fejlesztés, a munkafolyamatok miként jéről és menetéről. A csapat mindezen értékes tudást hasznosítja a feladat megoldása folyamán.

1.2. Projekt terv

1.2.1. A felhasznált fejlesztőeszközök

A csapat választása az Eclipse[1] integrált fejlesztői rendszer használatára esett, elsősorban annak testreszabhatósága, univerzalitása miatt. Felhasználásra került továbbá a Visual Paradigm[2] Community Edition kiadása, mely képes UML-diagramokból Java-kódot készíteni, így hatékonyan támogatja a fejlesztőcsapat munkáját, és illeszkedik annak filozófiájához, miszerint elsősorban a kiadott feladat részletes analízisén és tervezésen van a hangsúly, a kész programkód pedig ezzel kell, hogy szoros kapcsolatban legyen.

A dokumentáció LaTeX leírónyelv használatával nyerte el végleges formáját. A projekt-menedzsmentet és a verziókezelést egy Trac[3] névre hallgató felületen keresztül oldjuk meg Subversionnel[4].

1.2.2. A fejlesztőcsapat tagjai, azok feladatkörei

Név	Feladatok
Zsolnai Károly	csapatvezetés, dokumentáció, diagramok készítése, helyenként kódírás
Farkas Ádám Attila	kódírás, tesztelés
Siklósi Zsolt	dokumentáció- és diagramszerkesztés, kódírás
Tóth-Máté Ákos	dokumentáció, tesztelés, kódírás

[1] http://eclipse.org
[2] http://www.visual-paradigm.com
[3] http://trac.edgewall.org
[4] http://subversion.tigris.org

A csapattagok százalékos eloszlásban is megadták az elvállalt munkák, teendők típusait, minden tag érthetően jelezte, hogy a projektre fordított munkaidejében melyik feladatkörére mennyi időt szán. A felosztás idővel finomodhat, ugyanis az adott feladatkörök különböző mennyiségű munkaórát kívánnak meg, így az első néhány hét folyamán további megfontolások szükségesek.

1.2.3. Kommunikációs modell

A projektben dolgozó csapattagok kiválasztása gondos válogatási eljárás eredménye. Olyan emberek kerültek összeválogatásra, akik jól ismerik egymás szokásait, képesek együtt hatékony munkavégzésre. Fontos kritérium volt, hogy minden feladatkör megfelelően lefedésre kerüljön.

A fejlesztési folyamat során a kommunikáció fontossága alapvető, így az első teendők egyike volt annak részletes átgondolása, megszervezése.

A kapcsolatteremtési formák két nagy osztálya az aszinkron és a szinkron kommunikáció. Mindkettőnek megvannak a maga előnyei és hátrányai. Szinkron kommunikáció esetén a résztvevő felek hatékonyan tudnak információt cserélni, egymás hozzászólásaira reagálni, tehát ideális a közös munka végzésére, azonban nyilvánvaló, hogy nem lehetséges, hogy minden tag mindig elérhető legyen. Az aszinkron kommunikáció elsősorban a tervezésben, kódkészítésben, dokumentációban találkozók megszervezésében nyújt segítséget, azonban az azonnali üzenetváltás hiánya miatt elmarad a szinkron kommunikációs formáktól.

A csapat a kommunikációjának és közös munkájának támogatása alapvető fontosságú - a leghatékonyabb megoldásnak pedig a Trac rendszer bizonyult, amely a következő szolgáltatásokat foglalja magában:

SVN verziókezelés:

> A konkrét kódkészítés folyamán lehetővé teszi, hogy minden tag a szoftver jelenlegi verzióját birtokolja, hogy egyszerre többen szinkron módon dolgozhassanak egy adott probléma megoldásán, valamint részletes naplót vezet a szoftverben elkövetett módosításokról, változásokról.

Levelezési lista:

Az aszinkron kommunikáció konkrét megvalósítása. Minden fejlesztő levelet kap a teljes rendszer bármely részhalmazának változásáról. A szoftverek fejlesztése során nehezen megvalósítható, de alapvető fontosságú, hogy a résztvevők teljes képet kapjanak a rendszerről. Pontosan ezt valósítja meg a változásokról, módosításokról kiküldött értesítés.

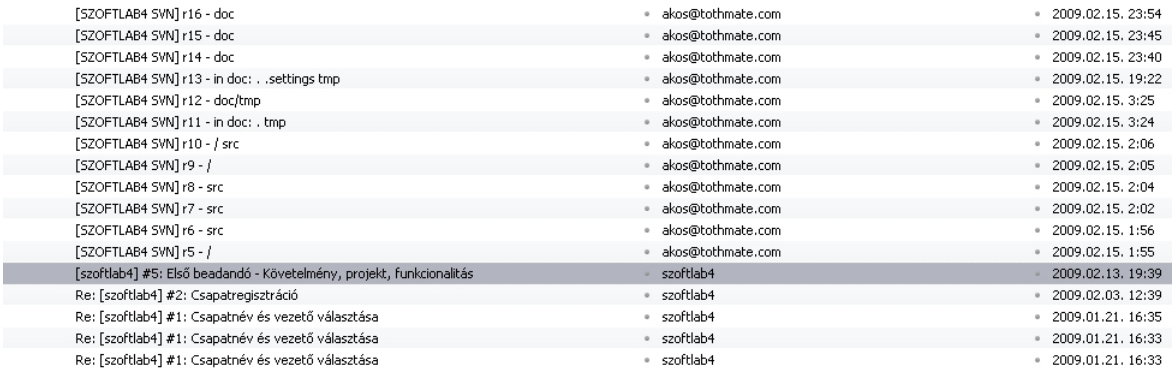
Wiki:

A projekthez tartozó fogalmakat, workflow-kat és tudásanyagot tartalmazó adatbázis. A csapat tagjai több munkakörben is részt fognak venni, így az adott problémán dolgozók által szerzett tapasztalatok megosztása fontos kritérium ahhoz, hogy a mások munkába való bekapcsolódását is lehetővé tegyék.

Bugtracker:

A tesztelések folyamán felderített hibajelenségek specifikálásához és naplózásához szükséges felület, mely segít azok feloldásában, javításában.

Egyéb kommunikációs formák: Microsoft Live Messenger - itt a tagok a szabadidejük jelentős részében elérhetőek -, Skype-konferencia, valamint személyes találkozók és megbeszélések. A feladat bizonyos részei közös jelenlétet igényelnek, ehhez nyújt segítséget az említett két szinkron kommunikációs forma.



[SZOFTLAB4 SVN] r16 - doc	akos@tothmate.com	2009.02.15. 23:54
[SZOFTLAB4 SVN] r15 - doc	akos@tothmate.com	2009.02.15. 23:45
[SZOFTLAB4 SVN] r14 - doc	akos@tothmate.com	2009.02.15. 23:40
[SZOFTLAB4 SVN] r13 - in doc: . .settings tmp	akos@tothmate.com	2009.02.15. 19:22
[SZOFTLAB4 SVN] r12 - doc/tmp	akos@tothmate.com	2009.02.15. 3:25
[SZOFTLAB4 SVN] r11 - in doc: . tmp	akos@tothmate.com	2009.02.15. 3:24
[SZOFTLAB4 SVN] r10 - / src	akos@tothmate.com	2009.02.15. 2:06
[SZOFTLAB4 SVN] r9 - /	akos@tothmate.com	2009.02.15. 2:05
[SZOFTLAB4 SVN] r8 - src	akos@tothmate.com	2009.02.15. 2:04
[SZOFTLAB4 SVN] r7 - src	akos@tothmate.com	2009.02.15. 2:02
[SZOFTLAB4 SVN] r6 - src	akos@tothmate.com	2009.02.15. 1:56
[SZOFTLAB4 SVN] r5 - /	akos@tothmate.com	2009.02.15. 1:55
[szoftlab4] #5: Első beadandó - Követelmény, projekt, funkcionalitás	szoftlab4	2009.02.13. 19:39
Re: [szoftlab4] #2: Csapatregisztráció	szoftlab4	2009.02.03. 12:39
Re: [szoftlab4] #1: Csapatnév és vezető választása	szoftlab4	2009.01.21. 16:35
Re: [szoftlab4] #1: Csapatnév és vezető választása	szoftlab4	2009.01.21. 16:33
Re: [szoftlab4] #1: Csapatnév és vezető választása	szoftlab4	2009.01.21. 16:33

Tárgy: [szoftlab4] #5: Első beadandó - Követelmény, projekt, funkcionalitás
Feladó: szoftlab4 <noreply@devolver.hu>
Válaszcím: noreply@devolver.hu
Dátum: 2009.02.13. 19:39
Másolat: akos@tothmate.com

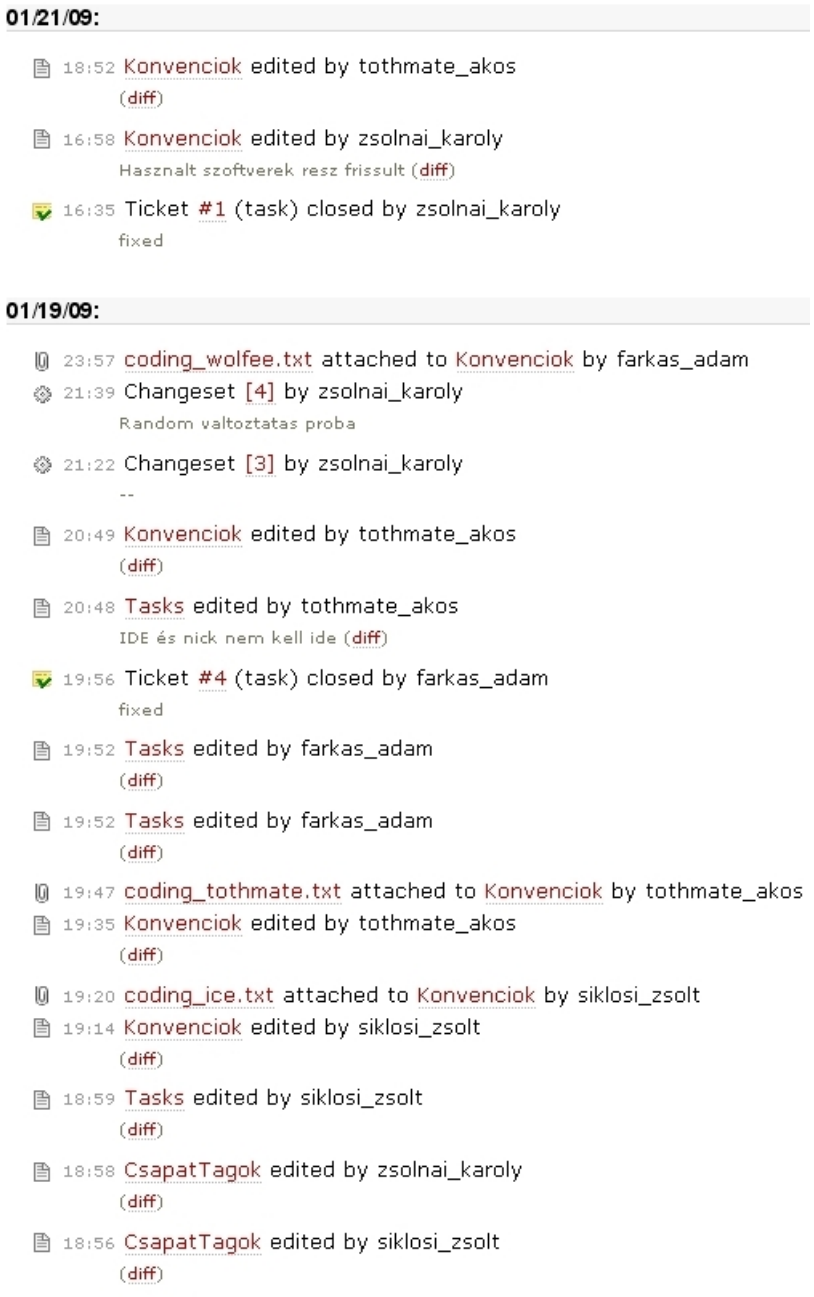
```
#5: Első beadandó - Követelmény, projekt, funkcionalitás
----------------------------+-----------------------------------------------
 Reporter:  zsolnai_karoly  |       Owner:
     Type:  task            |      Status:  new
 Priority:  major           |   Milestone:  Szkeleton
Component:  általános       |    Keywords:
----------------------------+-----------------------------------------------
 El kell kezdeni az első beadandó elkészítését. A konzultáción szerzett
 információk az alábbi wiki-oldalon tekinthetők meg:

 http://trac.devolver.hu/szoftlab4/wiki/Elso_beadando

 Az első beadandó határideje: Február 19.

--
Ticket URL: <http://trac.devolver.hu/szoftlab4/ticket/5>
szoftlab4 <http://devolver.hu>
szoftlab4
```

1. ábra. Levelezési lista



2. ábra. Trac Timeline

1.2.4. Fejlesztési mérföldkövek, ütemterv

Skeleton:

A szó jelentése: váz, vázrendszer. A kész termék minőségét nagymértékben meghatározza a felállított modell részletessége, helyessége, így annak alapos átgondolása, mérnöki módszerekkel való előállítása döntő fontosságú. Ez egy időigényes folyamat, azonban amennyiben sikeresen zárul, a projekt további folyamatai során esetlegesen felmerülő komplikációk száma minimalizálható.

Prototípus (a továbbiakban helyenként röviden proto):

A programtörzs teljes változata, amely a grafikai elemeken kívül már minden alkotóelemet tartalmaz. Ezen mérföldkő elérését követően a program széleskörű tesztelése válik lehetővé.

Grafikus felület (a továbbiakban esetenként GUI[5]):

A szoftver akkor tekinthető teljesnek, mikor a grafikus felhasználói felület is elkészült. A pontos és mérnöki tervezés eredményeképpen a grafikus felület és az algoritmusokat tartalmazó modell egymástól való függetlensége.

[5]Graphical User Interface

1.2.5. Határidők

febr. 13.	14h - A csapatok regisztrációja
febr. 19.	Követelmény, projekt, funkcionalitás - beadás
febr. 26.	Analízis modell kidolgozása 1. - beadás
márc. 5.	Analízis modell kidolgozása 2. - beadás
márc. 12.	Skeleton tervezése - beadás
márc. 19.	Skeleton- beadás
márc. 26.	Prototípus koncepciója - beadás
ápr. 2.	Részletes tervek - beadás
ápr. 9.	
ápr. 16.	Prototípus - beadás
ápr. 23.	Grafikus felület specifikációja - beadás
ápr. 30.	
máj. 7.	Grafikus változat - beadás
máj. 14.	Összefoglalás - beadás

1.2.6. Átadás

A dokumentáció nyomtatott formában hétről-hétre a konzulens irányába kerül továbbításra. A kész szoftver forráskódját a konzulens folyamatosan ellenőrizheti, annak működő verziója pedig időről-időre a HSZK[6] laboratóriumaiban kerül bemutatásra.

1.2.7. Kockázatelemzés

Valószínűségek osztályozása:

Alacsony - Közelítőleg 0-20%-os eséllyel bekövetkező esemény

Közepes - Közelítőleg 20-50%-os eséllyel bekövetkező esemény

Biztos - Közelítőleg 50-100%-os eséllyel bekövetkező esemény

Hatások osztályozása:

Elhanyagolható - Az esemény bekövetkezése nem okoz különösebb problémát, az esetleges javítása nem igényel komoly munkálatokat.

[6]Hallgatói Számítógép Központ - BME

Enyhe - Az esemény bekövetkeztével okozott kár legfeljebb néhány munkaórával javítható, visszaállítható.

Közepes - A csapat több tagjának összehangolt munkáját igénylő probléma, melynek megoldása komolyabb erőfeszítéseket igényel. Ezen változtatások már hatással lehetnek a heti ütemtervre, és a fejlesztőcsapat közös megbeszélését igényelhetik.

Komoly - A projekt sikeres kimenetelét vagy a csapat integritását fenyegető esemény, mely alapvető változtatásokkal jár. Az ilyen jellegű probléma a csapat azonnali tanácskozását igényli.

Eseménytáblázat:

Esemény	Valószínűség	Hatás
Specifikációváltozás	Biztos	Enyhe
Javítandó heti beadandó	Közepes	Enyhe
Sikertelen heti beadandó	Alacsony	Közepes
Csapattag kiválás	Alacsony	Komoly
Csapattag betegsége, távolléte	Alacsony	Közepes
Hardvermeghibásodás	Alacsony	Elhanyagolható
Szoftvermeghibásodás	Közepes	Enyhe
Határidőről lecsúszás	Alacsony	Közepes
Tanácskozásról hiányzás	Közepes	Közepes

1.2.8. Egyéb fontos megjegyzések

A fejlesztés lefordítása és bemutatása a HSZK laborjaiban történik, így a program forráskódjának kompatibilisnek kell lennie a Java Development Kit ennek megfelelő korábbi verziójával. A maximális kompatibilitás elérése érdekében a fejlesztés is pontosan ezeken a verziószámú platformokon történik majd, azaz 1.4.2-es Java SDK és 1.5.0 verziójú JRE kerül felhasználásra.

1.2.9. Szükséges dokumentációk

1. Követelmény, projekt, funkcionalitás
2. Analízis modell kidolgozása 1
3. Analízis modell kidolgozása 2
4. Skeleton tervezése
5. Skeleton
6. Prototípus koncepciója
7. Részletes tervek
8. Prototípus
9. Grafikus felület specifikációja
10. Grafikus változat
11. Összefoglalás

1.3. A feladat részletes leírása

A program egy kincskereső indián, Júz Kéz kétdimenziós világába kalauzolja a felhasználót. A főhős ásójával és tarisznyájával barlangról barlangra jár, feladata pedig a járatokban található gyémántok begyűjtése. Kalandjai során a rá leselkedő veszélyek ügyességi és logikai próbatételt jelentenek a játékos számára.

A bejárandó barlangok főképp laza, kiásható földterületből állnak, ám előfordulnak kemény gránitfalak, amelyeken csak korlátozott mennyiségben rendelkezésre álló robbanószer felhasználásával lehetséges az átjutás, valamint sziklák, melyeket a már kivájt járatokon keresztül lehet mozgatni. Az elhelyezett bombák időzítettek, néhány másodpercen belül robbannak, így a játékosnak van ideje ezalatt biztonságos távolságba mozogni. A játékosra a barlangokban különböző, jellegzetes tulajdonságokkal rendelkező szörnyek is veszélyt jelentenek: léteznek céltalanul bolyongó, de a karaktert követő példányok is, némelyikük halálakor gyémánttá változik. Az egyes pályák

elvégzésének követelménye a benne szereplő szörnyek legalább egy részének eliminálása, ami kétféleképpen lehetséges: robbantással, vagy egy szikla rájuk ejtésével.

A bejárandó világ egy kétdimenziós barlangrendszer, melyet a játékos a billentyűzet segítségével fedezhet fel. Két tetszőleges pont közötti terület szabadon bejárható, amennyiben a játékos nem botlik útközben akadályokba. A játéktér széle áthatolhatatlan mezőkből áll, tehát a pálya elhagyása csak és kizárólag a játék folyamán megnyíló kijáraton keresztül lehetséges. A játékban szereplő elemekre gravitációs erő hat, azaz az alulról megtámasztatlan objektumok óhatatlanul leesnek, az oldaltámasszal nem rendelkező sziklák pedig elgurulnak.

A játék célja az egymás után következő, előre megtervezett pályák megnyerése. Az indián főhős, akit a játékos alakít, igen mohó kincskereső, addig nem hajlandó elhagyni a barlangokat, amíg onnan az összes gyémántot össze nem gyűjtötte. A kijárat csak ezt követően jelenik meg, egy-egy pálya pedig a barlang elhagyásakor tekinthető sikeresen teljesítettnek. Mikor a felhasználó a kijárathoz irányítja karakterét, a program automatikusan betölti a soron következő pályát. A karakter meghal, ha szikla zuhan rá, egy bomba robbanásakor annak hatósugarában tartózkodik, vagy ha a veszélyes szörnyek egyike felfalja – ekkor az aktuális pálya újrakezdhető, annak megnyerésére tetszőleges számú próbálkozás tehető.

Új játékot a főmenüből kezdhetünk, a játék tetszőleges időpontban szüneteltethető, illetve befejezhető. A játéktér egységnyi méretű, négyzet alakú térrészekből épül fel, melyekről egyértelműen megállapítható hogy mit ábrázolnak, így a barlangokban való navigáció sikeressége csakis a játékos ügyességén múlik. A játékmenet teljes mértékben a felhasználó lépéseitől függ: azáltal, hogy mozgatja a karaktert, mozgásba lendíthet kezdetben nyugalmi helyzetben lévő objektumokat. A játékosnak folyamatos figyelemre van szüksége, mert egy-egy lépésével különböző veszélyekbe sodródhat: ráeshet egy szikla, vagy a közelébe kerülhet egy veszélyes lény. Külön ügyelni kell arra, hogy a játék szabályai szerint egy pálya megnyeréséhez a rajta szereplő valamennyi gyémánt megszerzése szükséges. Némely lények elpusztításuk esetén gyémánttá préselődnek, tehát a pályáról való továbblépéshez elengedhetetlen ezeknek a likvidálása. Természetesen e különleges szörnyek

vizuálisan jól megkülönböztethetők lesznek a felhasználó számára.

A program indításakor a felhasználót a főmenü fogadja, ahonnan lehetséges új játék indítása, kilépés, valamint egy megadott pályára való ugrás. A felhasználói felület fő része maga a játéktér, ez teszi ki a megjelenített felület legnagyobb részét. Emellett pedig a felhasználó tájékoztatást kap arról, hogy hány gyémánt szerezhető meg az aktuális pályán, és hogy ezek közül az adott pillanatig mennyit sikerült megszereznie.

A játék valós időben fut, a szörnyek néhány másodpercenként véletlenszerűen, vagy a karakter irányába mozognak, a leeső sziklák elől ki lehet térni, ugyanakkor lehetséges az aláhulló gyémántok levegőben való elkapása. Az egymás után következő pályák egyre komolyabb kihívást jelentenek. A játékból való kilépéskor a program megjegyzi az aktuális pálya sorszámát, így nem szükséges minden játékindításkor az összes, korábban már elvégzett pályát újrajátszani. A már bejárt pályák később újra meglátogathatók a pályaválasztó almenü segítségével.

1.4. Szótár

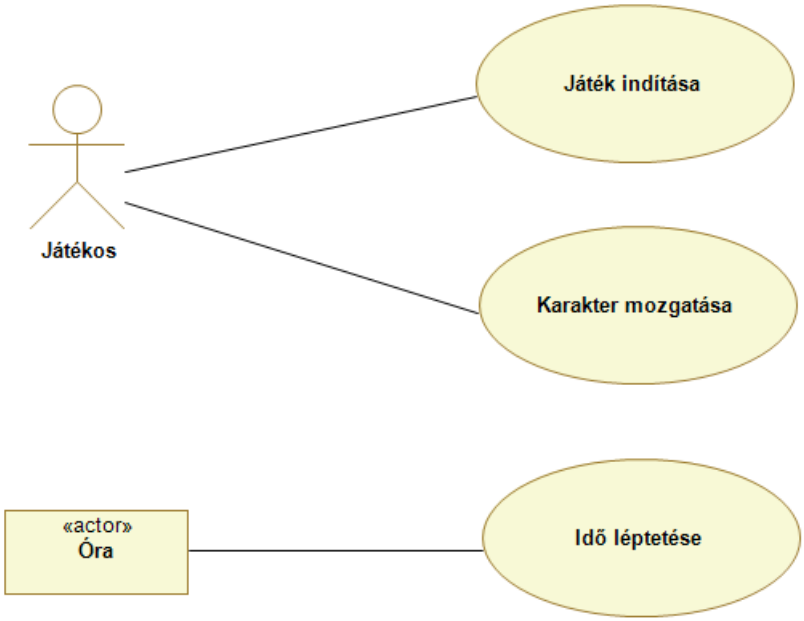
indián (Júz Kéz)	a játékos által irányított kincskereső karakter
felhasználó, játékos	a számítógépet használó személy
karakter	a felhasználó által irányított virtuális személy
barlang	bejárható játékterület
mező	a pálya egységét megtestesítő, négyzet alakú térrész
gyémánt	felszedhető elem, mely a pálya elvégzéséhez szükséges
föld	olyan mezőtípus, melyre rálépve a karakter kiássa azt, így a továbbiakban üres marad
kivái/kiás	amennyiben a játékos föld mezőre lépteti a karaktert, az kiássa, kivájja az adott területet

gránit	nem kiásható mező, csak robbanószerrel lehet eltüntetni
robbanószer	a gránit mezők eltüntetésére és ellenséges lények likvidálására szolgáló eszköz
tarisznya	a robbanóanyag és a gyémántok tárolására szolgáló eszköz
szikla	eltolható, de nem áthatolható mező
lény	a pályákon megjelenő ellenfél, mely veszélyes a karakterre
ráesik	ha egy szikla ráesik a karakterre vagy lényre, az elpusztul
felrobban	ha egy bomba felrobban, a közelében levő karakter vagy lény meghal
szaporodik	a lények egyik speciális fajtája megsokszorozza önmagát
üldöz	a lények egyik speciális fajtája a játékos által irányított karakter felé mozog
gyémánttá alakul	egy speciális fajtájú lényre sziklát ejtve annak elpusztultával egy gyémánt fog hátramaradni
megtámasztott	bizonyos entitások csak akkor maradnak a helyükön, ha alulról egy másik objektum által meg vannak támasztva
leesik	ha megszűnik az alsó támaszték, az objektum a gravitáció hatására zuhanni kezd
legördül	a szabadon, egy oszlopban álló sziklák és gyémántok egymásról legördülve kupacba omlanak le
meghal	a karakter meghal, ha szikla esik rá, a közelében bomba robban, vagy veszélyes lénnyel érintkezik

kijárat	amennyiben a játékos összegyűjti az adott számú gyémántot, megnyílik a kijárat, melyen keresztül továbbjuthat a következő barlangba (pályára)
megnyílik	adott számú gyémánt összegyűjtése esetén megjelenik a pálya kijárata

1.5. Essential use-case-ek

1.5.1. Essential use-case diagram



3. ábra. Essential use-case-ek

1.5.2. Essential use-case-ek leírása

Játék indítása:

A játékos új játékot kezd. A pálya kezdőállapota töltődik be.

Karakter mozgatása:

A játékos vezérli az indián mozgását. Az indián mozgatása a leütött billentyűnek megfelelően fog megtörténni.

Idő léptetése:

Ezt egy belső actor végzi (óra). Periodikus időközönként lépteti az időt, mely az önmaguktól mozgó lények léptetését vonja maga után.

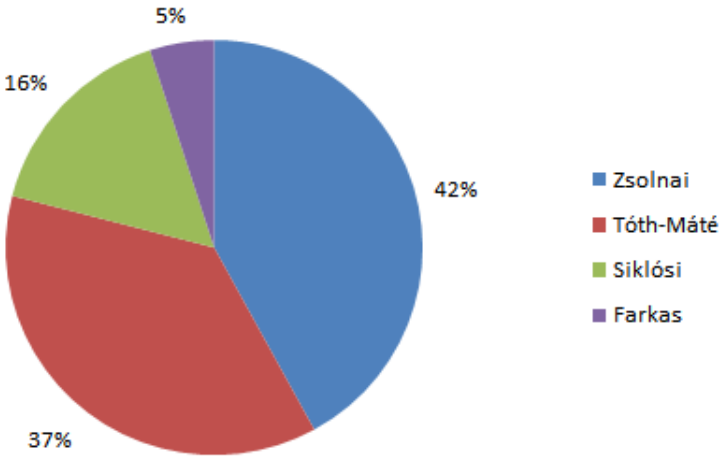
1.6. Napló

2009.01. Csapattagok keresése
2009.02.09. Csapatnév választása: String Quartet
2009.02.10. String Quartet Kapcsolatfelvétel
2009.02.11. String Quartet Feladatkörök százalékos meghatározása (2.2.2)
2009.02.12. Tóth-Máté Trac SVN/CVS rendszer üzembe helyezése (2.2.3)
2009.02.12. Tóth-Máté Határidők és ütemterv felvitele a Trac-be (2.2.5)
2009.02.13. String Quartet Második heti beadandó dokumentumok részeinek kiosztása
2009.02.14. 05:40 Zsolnai, idő: 03:00 Követelmény definíció és Projekt terv első verziójának megírása (2.1, 2.2)
2009.02.14. 18:00 Farkas, idő: 00:30 Specifikációk és egyéb információk beszerzése
2009.02.14. 21:00 Tóth-Máté, idő: 02:30 Dokumentum-formázás, hibajavítás
2009.02.14. 23:00 Zsolnai, idő: 01:00 Csapatszervezés
2009.02.15. 10:30 Siklósi, idő: 00:30 Essential Use-Case diagram megszerkesztése (2.5)
2009.02.15. 11:00 Siklósi, idő: 01:00 Naplófájl készítése MS Excel felhasználásával
2009.02.15. 19:30 Zsolnai, idő: 03:00 Követelmény definíció és Projekt terv további finomítása, a feladat leírása (2.1, 2.2, 2.3)

2009.02.16. 00:00	Tóth-Máté, idő: 02:00
Első beadandó dokumentációhoz szükséges anyagok összeszerkesztése, javítása	
2009.02.16. 10:00	Siklósi, idő: 01:00
Szótár első verziójának megírása (2.4)	
2009.02.16. 19:15	String Quartet, idő: 01:00
Feladat részletes leírásának további finomítása (2.3)	
2009.02.16. 20:00	Zsolnai, idő: 00:30
Szótárban szereplő hibák javítása (2.4)	
2009.02.16. 20:30	Tóth-Máté, idő: 00:30
Dokumentáció hiányzó részeinek feltöltése	
2009.02.16. 21:00	Tóth-Máté, idő: 01:00
Napló áthelyezése Wikibe, majd erre LaTeX konverter írása	
2009.02.16. 22:00	Zsolnai, idő: 01:00
Csapatszervezés	
2009.02.16. 22:00	Farkas, idő: 00:30
Dokumentáció átnézés-szerkesztés	
2009.02.17. 14:00	Tóth-Máté, idő: 02:00
Use-case-ek újrarajzolása és beillesztése a dokumentációba (2.5)	
2009.02.17. 10:00	Siklósi, idő: 01:00
Dokumentáció revíziója, javítások	
2009.02.18. 21:00	Zsolnai, idő: 00:30
Értékelés megírása (2.7), szótár bővítése	

1.7. Értékelés

A csapatban végzett munka százalékos eloszlása:
Zsolnai: 42%
Tóth-Máté: 37%
Siklósi: 16%
Farkas: 5%
Teljes munkaidő: 22,5 óra.



4. ábra. 1. Értékelés

Az első heti dokumentáció szerkesztésének folyamán a munka megoszlása nem volt egyenletes, ennek fő oka, hogy egyes tagok elsődleges feladataként a kódírás és a tesztelés lett megjelölve, így hosszútávon a megoszlás kiegyenlítődése figyelhető majd meg.

2. Analízis modell kidolgozása 1.

2.1. Objektum katalógus

2.1.1. Game

A játékban megjelenő entitások létrehozásáért, tartalmazásáért és megszüntetéséért felelős. Az időzítő jeleit eljuttatja a pályán lévő objektumokhoz.

Alaposztályok: nincs

Példányok száma: 1

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek:

`Cave`	`cave`	a barlang
`Player` [1..*]	`players`	a játékos(oka)t tartalmazó tömb
`Boulder` [0..*]	`boulders`	a sziklákat tartalmazó tömb
`Diamond` [0..*]	`diamonds`	a gyémántokat tartalmazó tömb
`Explosive` [0..*]	`explosives`	a róbbanószereket tartalmazó tömb
`Monster` [0..*]	`monsters`	az ellenséges lényeket tartalmazó tömb
`Element` [0..*]	`staticElements`	a mozdulatlan elemeket tartalmazó tömb
`int`	`playersAlive`	a még életben levő játékosok száma
`int`	`allDiamonds`	a pályán található összes gyémánt száma
`boolean`	`gameFinished`	tárolja, hogy befejeződött-e a játék

Relációk: nincs

Változók: nincs

Szolgáltatások:

`void`	`tick()`	adott sorrendben meghívja az összes objektum `onTick()` függvényét
`void`	`destroyGame()`	kiüríti a barlangot és a játékobjektumokat tároló tömböket.
`void`	`addPlayer(Player p)`	hozzáad egy játékost a `players` tömbhöz
`Cave`	`getCave()`	visszatér a játékhoz tartozó barlang referenciájával
`boolean`	`hasAllDiamonds()`	visszaadja, hogy felvették-e már a karakterek az összes gyémántot
`void`	`addBoulder(Boulder b)`	hozzáad egy sziklát
`void`	`addDiamond(Diamond d)`	hozzáad egy gyémántot
`void`	`addExplosive(Explosive e)`	hozzáad egy robbanószert
`void`	`addMonster(Monster m)`	hozzáad egy szörnyet
`void`	`addStaticElement(Element e)`	hozzáad egy statikus elemet
`void`	`removeElement(Element e)`	eltávolítja az adott elemet
`Player`	`getPlayerById(int id)`	visszaadja a kívánt azonosítójú elemet a karaktereket tartalmazó tömbből
`Explosive`	`getExplosive(int i)`	visszaadja az explosives vektor i-edik elemét
`int`	`getDiamondCount()`	visszaadja a gyémántok számát
`int`	`getCollectedDiamondCount()`	visszaadja a karakterek által idáig felvett gyémántok számát
`Diamond`	`getDiamondOfMonster(Monster m)`	visszatér a paraméterként megadott szörnyhöz tartozó gyémánt referenciájával

Felelősségek: nincs

2.1.2. Cave

A barlangot megvalósító osztály, mely a mezőket megtestesítő `Field` típusú objektumokat tartalmaz és kapcsolatban áll a `Game` osztállyal.

Alaposztályok: nincs

Példányok száma: 1

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek:

`Field [0..*]`	`map`	a mezőket tartalmazó tömb

Relációk: nincs

Változók:

`Game`	`game`	referencia a tartalmazó `Game` objektumra

Szolgáltatások:

`Field`	`createField()`	Létrehoz és hozzáad egy mezőt a `map` tömbhöz
`Field[]`	`getMap()`	a térképet visszaadó függvény
`Game`	`getGame()`	a tartalmazó `Game` referenciájával tér vissza

Felelősségek: nincs

2.1.3. Field

Egy-egy mezőt megvalósító osztály, mely egy `Cave`-ben van és pontosan egy `Element` típusú objektum (indián, lény, szikla, stb.) állhat rajta.

Alaposztályok: nincs

Példányok száma: n

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek:

`Element`	`element`	a mezőn elhelyezkedő objektum

Relációk: nincs

Változók:

`Field[]`	`neighbors`	a mezővel szomszédos mezők referenciái
`Cave`	`cave`	a mezőt tartalmazó barlang referenciája

Szolgáltatások:

`Element`	`getElement()`	a mezőn álló objektumot adja vissza
`void`	`setElement(Element e)`	ráteszi a mezőre a paraméterként kapott objektumot
`void`	`clear()`	leveszi a mezőről a rajta álló objektumot és létrehoz egy üres `Element` a helyére
`Field`	`getNeighbor(int direction)`	visszaadja a mező kért szomszédját

`void`	`setNeighbor(Field f, int direction)`	beállítja `f`-et szomszédos mezőnek
`Cave`	`getCave()`	azzal a barlanggal tér vissza, amelyiken a mező elhelyezkedik

Felelősségek: nincs

2.1.4. Element

A különböző viselkedésű objektumok absztrakt alaposztálya, mely tartalmazza a felüldefiniálandó függvényeket és a közös attribútumaikat.

Alaposztályok: nincs

Példányok száma: 0

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók:

`Field`	`field`	az elemet tartalmazó mező referenciája

Szolgáltatások:

`abstract void onTick()`	ebben a függvényben van definiálva, mit kell tennie az objektumnak, ha órajelet (tick) kap
`abstract void steppedOn(Element e)`	e függvény tartalmazza, mit kell tennie az objektumnak, ha egy `e` objektum rálép

`Field`	`getField()`	az objektumot tartalmazó mezőt adja vissza
`void`	`setField(Field f)`	az objektumot tartalmazó mezőt állítja be
`void`	`move(Field f)`	ezzel jelzi egy objektum a rálépő objektumnak, hogy mozdulhat
`void`	`die()`	az objektum ezzel a függvénnyel megszüntetheti önmagát

Felelősségek: nincs

2.1.5. Player

A játékost megtestesítő osztály. Mozgását a felhasználó irányítja.

Alaposztályok: `Element`

Példányok száma: n (legtöbbször csak 1)

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók:

`int`	`diamondCounter`	a már megszerzett gyémántok száma
`int`	`explosiveCounter`	a tarisznyában lévő robbanószerek száma

Szolgáltatások:

`void`	`incDiamond()`	növeli a gyémántok számát
`void`	`decExplosive()`	csökkenti a robbanószerek számát
`void`	`steppedOn(Boulder b)`	ez hívódik, mikor egy szikla rá akar lépni egy játékost tartalmazó mezőre
`void`	`steppedOn(Diamond d)`	ez hívódik, mikor egy gyémánt rá akar lépni egy játékost tartalmazó mezőre
`void`	`steppedOn(Explosive e)`	ez hívódik, mikor egy robbanószer rá akar lépni egy játékost tartalmazó mezőre
`void`	`steppedOn(Monster m)`	ez hívódik, mikor egy szörny rá akar lépni egy játékost tartalmazó mezőre

Felelősségek: nincs

2.1.6. Boulder

A sziklákat megtestesítő osztály.

Alaposztályok: `Element`

Példányok száma: n

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók:

`boolean`	`falling`	esés közben igaz az értéke

Szolgáltatások:

`boolean`	`isFalling()`	visszaadja, hogy zuhan-e éppen a szikla
`void`	`steppedOn(Player p)`	ez hívódik, mikor egy játékos rá akar lépni a sziklára

Felelősségek: nincs

2.1.7. Diamond

A gyémántok osztálya.

Alaposztályok: `Element`

Példányok száma: n

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók:

`boolean`	`falling`	esés közben igaz az értéke

Szolgáltatások:

`boolean`	`isFalling()`	visszaadja, hogy zuhan-e éppen a gyémánt
`void`	`steppedOn(Player p)`	ez hívódik, mikor egy játékos rá akar lépni a gyémántra

Felelősségek: nincs

2.1.8. Explosive

A robbanószereket megtestesítő osztály.

Alaposztályok: `Element`

Példányok száma: n

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók:

`int`	`counter`	a robbanásig hátralevő időt számolja

Szolgáltatások: nincs

Felelősségek: nincs

2.1.9. Monster

Az ellenséges lényeket megtestesítő osztály. Többféle mutációja létezik eltérő tulajdonságokkal és képes önállóan mozogni.

Alaposztályok: `Element`

Példányok száma: n

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók:

`boolean`	`follower`	igaz, ha a szörny a játékost követi
`boolean`	`selfCloner`	igaz, ha a szörny önmagát meg tudja sokszorozni
`boolean`	`transformer`	igaz, ha a szörny elpusztulásakor gyémánttá alakul át

Szolgáltatások:

`boolean`	`isFollower()`	a szörny követő tulajdonságát adja meg
`boolean`	`isSelfCloner()`	a szörny önsokszorozó tulajdonságát adja meg
`boolean`	`isTransformer()`	a szörny gyémánttá alakuló tulajdonságát adja meg
`void`	`steppedOn(Boulder b)`	ez hívódik, mikor egy szikla rá akar lépni egy szörnyet tartalmazó mezőre

`void`	`steppedOn(Explosive e)`	ez hívódik, mikor egy robbanószer rá akar lépni egy szörnyet tartalmazó mezőre
`void`	`steppedOn(Player p)`	ez hívódik, mikor a játékos rá akar lépni egy szörnyet tartalmazó mezőre

Felelősségek: nincs

2.1.10. Dirt

A kiásható földet megvalósító osztály.

Alaposztályok: `Element`

Példányok száma: n

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók: nincs

Szolgáltatások:

`void`	`steppedOn(Player p)`	ez hívódik, mikor a játékos rálép a földre

Felelősségek: nincs

2.1.11. Exit

A megnyíló kijáratot megvalósító osztály.

Alaposztályok: `Element`

Példányok száma: 1

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók: nincs

Szolgáltatások:

`void`	`steppedOn(Player p)`	ez hívódik, mikor a játékos rálép a kijáratra

Felelősségek: nincs

2.1.12. Granite

Az áthatolhatatlan, de felrobbantható gránitot megvalósító osztály.

Alaposztályok: `Element`

Példányok száma: n

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók: nincs

Szolgáltatások:

`void`	`steppedOn(Explosive e)`	ez hívódik, mikor robbanás történik a gránit közelében

Felelősségek: nincs

2.1.13. Wall

A pályát körülvevő falat megvalósító osztály.

Alaposztályok: `Element`

Példányok száma: n

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók: nincs

Szolgáltatások: nincs

Felelősségek: nincs

2.1.14. Empty

Az üres mezőt megvalósító osztály.

Alaposztályok: `Element`

Példányok száma: n

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók: nincs

Szolgáltatások: nincs

Felelősségek: nincs

2.1.15. Timer

Az időzítésért felelős osztály.

Alaposztályok: nincs

Példányok száma: n

Konkurencia: passzív

Perzisztencia: dinamikus

Komponensek: nincs

Relációk: nincs

Változók: nincs

Szolgáltatások:

`void`	`tick()`	az időzítő jelet kiadó függvény

Felelősségek: nincs

2.2. Osztályok leírása

Game: A játékban szereplő osztályokat tartalmazó objektum, ami összefogja azokat, lehetővé teszi, hogy együttesen is működőképesek legyenek.

Cave: A játékteret reprezentálja, az azt felépítő mezők elhelyezéséért, tárolásáért felelős.

Field: A játéktér egyik alapelemét, a mezőt testesíti meg, melyre térelemeket helyezhetünk, és amely képes számon tartani saját szomszédait.

Element: A különböző térelemeket összegyűjtő absztrakt ősosztály. A belőle származtatott térelemek képesek az időzítővel szinkronban cselekedni, megvizsgálni, hogy mely mezőhöz tartoznak, bizonyos esetekben mozogni, valamint elpusztulásuk pontos körülményeit és menetét is ismerik.

Player: A játékos által irányított karakter. Rendelkezik bizonyos mennyiségű robbanóanyaggal, számon tartja az általa begyűjtött gyémántok számát, valamint pontosan tudja, hogy elpusztul, ha szikla zuhan rá vagy egy szörny megérinti. El tudja helyezni a robbanóanyagot, és gyémánttal való találkozásakor begyűjti azt.

Boulder: A szikla elemtípust reprezentálja, mely képes zuhanni, gurulni, és pontosan képes meghatározni hogy milyen körülmények között mely cselekvést kell végeznie.

Diamond: A mindenfelé elszórt gyémántok működését biztosító osztály, mely képes megvizsgálni, hogy milyen körülmények között kell zuhannia, és hogy felvételét követően el kell tűnnie a játéktérről.

Explosive: A karakter által hordozott időzíthető robbanóanyag képes egyes mezők elpusztítására. Saját beépített számlálója van, amely az elhelyezést követően lép működésbe.

Monster: A szörnyek a játéktéren önállóan mozognak, céljuk a karakter megsemmisítése. Tetszőleges, előre meghatározott tulajdonságok kombinációival rendelkezhetnek, robbanóanyaggal, szikla ráejtésével likvi-

dálhatók. Fel vannak készítve a különböző térelemekkel való találkozásra.

Dirt: A barlangrendszerek jelentős részét kitevő kiásható föld. A kiásást követően üres terület marad utána.

Exit: Az összes gyémánt begyűjtését követően megjelenő kijárat, amin keresztül lehetséges a következő pályára jutás.

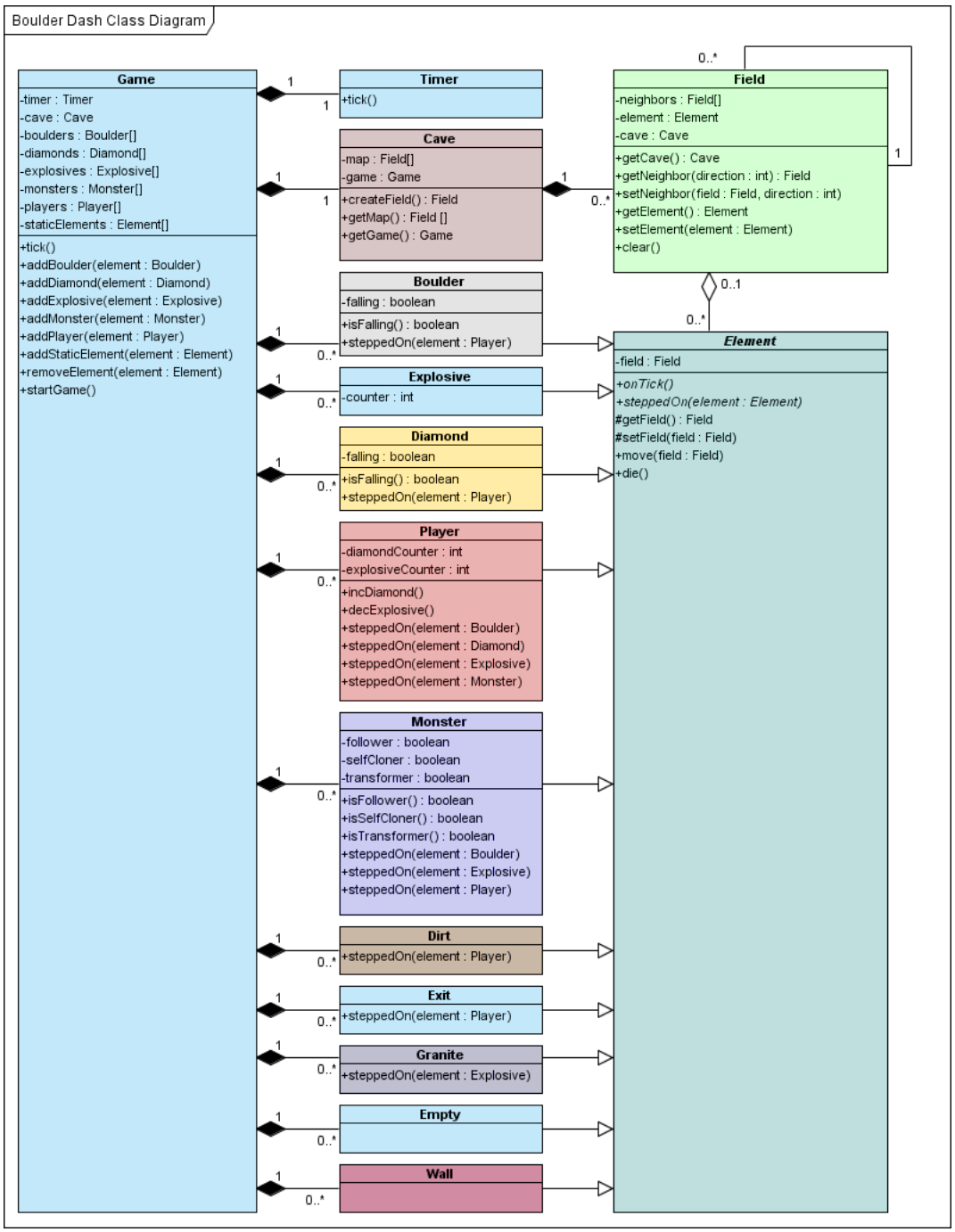
Granite: Stacionárius objektum, melyen az áthaladás csakis felrobbantását követően lehetséges.

Wall: A játéktér széleit határoló, áthatolhatatlan, kirobbanthatatlan fal.

Empty: A kivájt földrészek helyén maradó üres objektum, a szörnyek és a játékos által egyaránt bejárható terület. Amennyiben sziklák és gyémántok alatt üres terület található, azok zuhanni kezdenek.

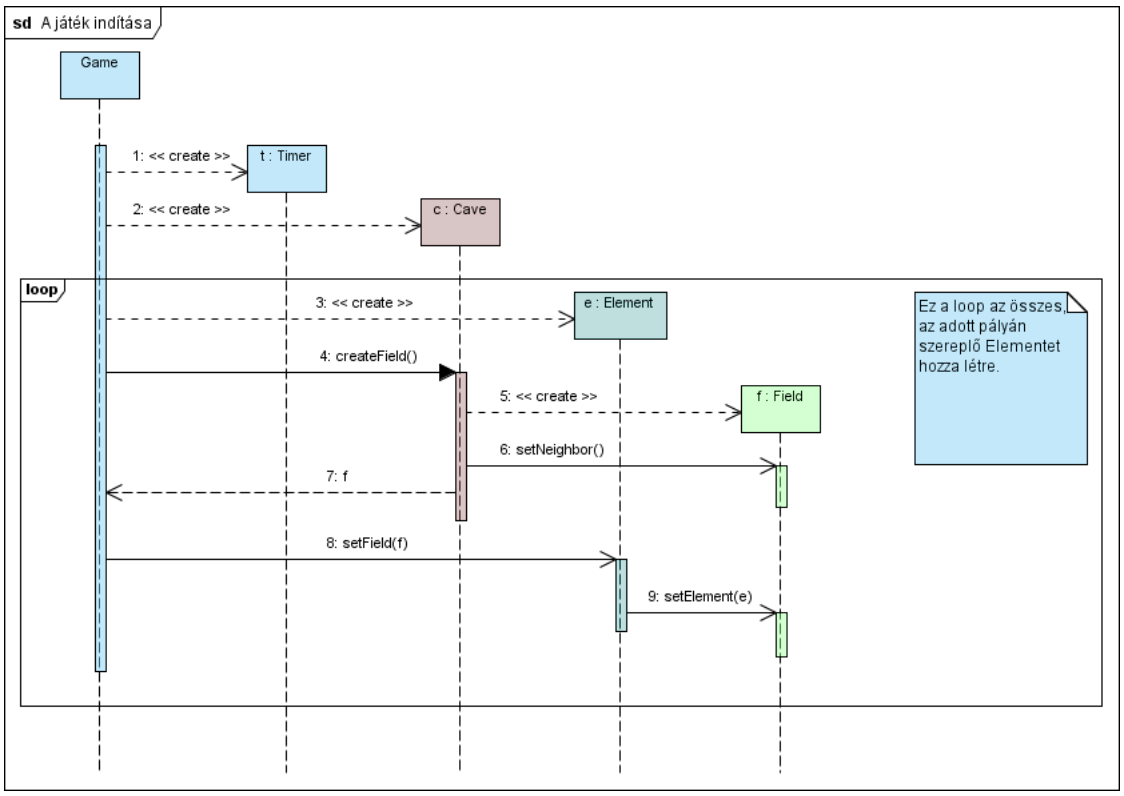
Timer: Az időzítésért felelős osztály, amelynek állítható saját frekvenciája adja meg az órajelet, ami a játék világában lefutó eseményeket időzíti.

2.3. Statikus struktúra diagramok

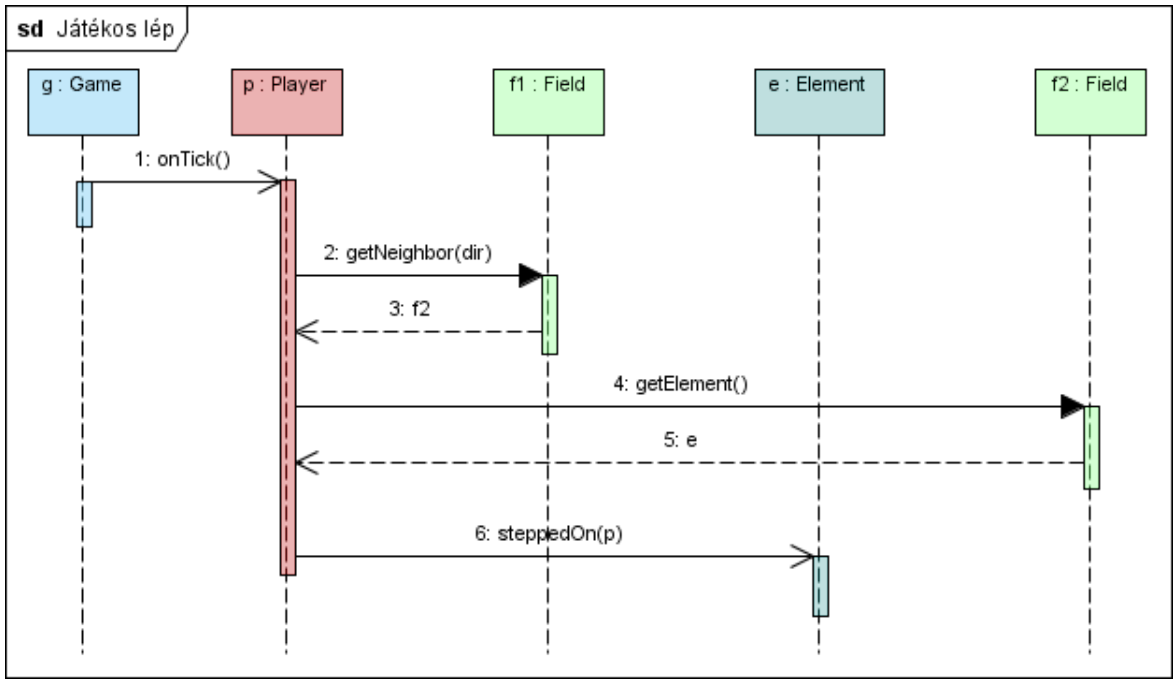


5. ábra. Osztálydiagram

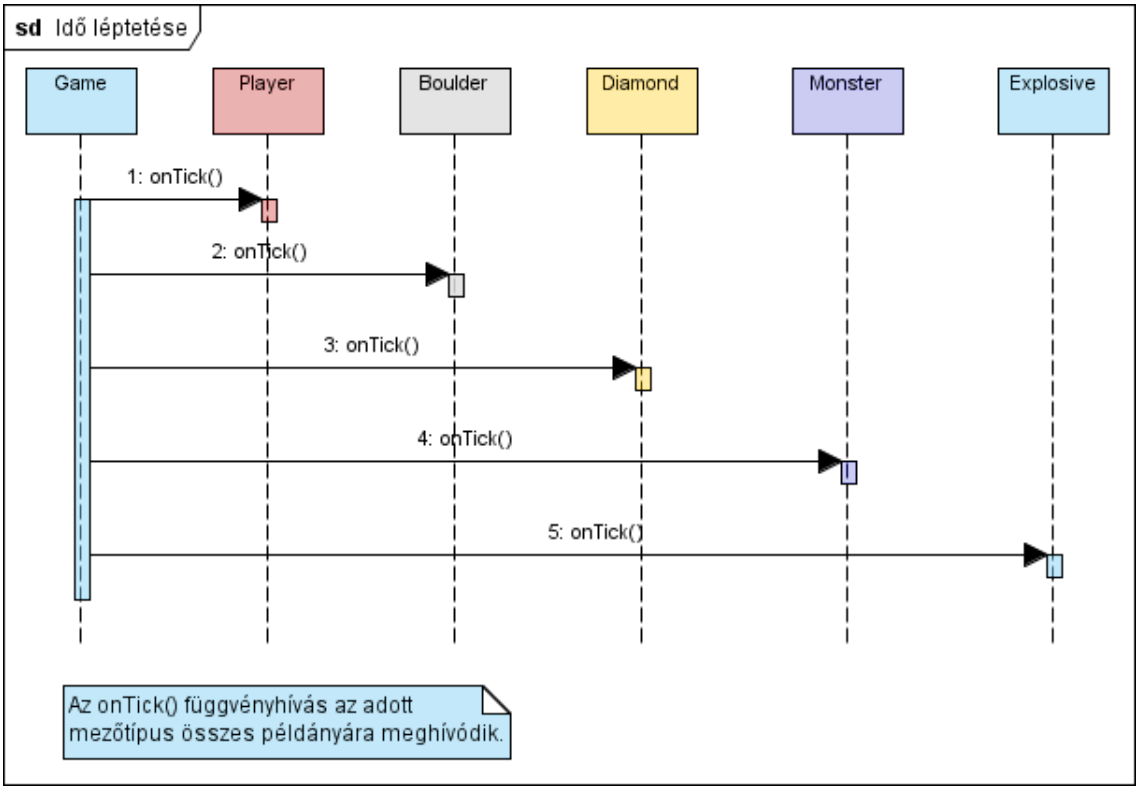
2.4. Szekvencia diagramok



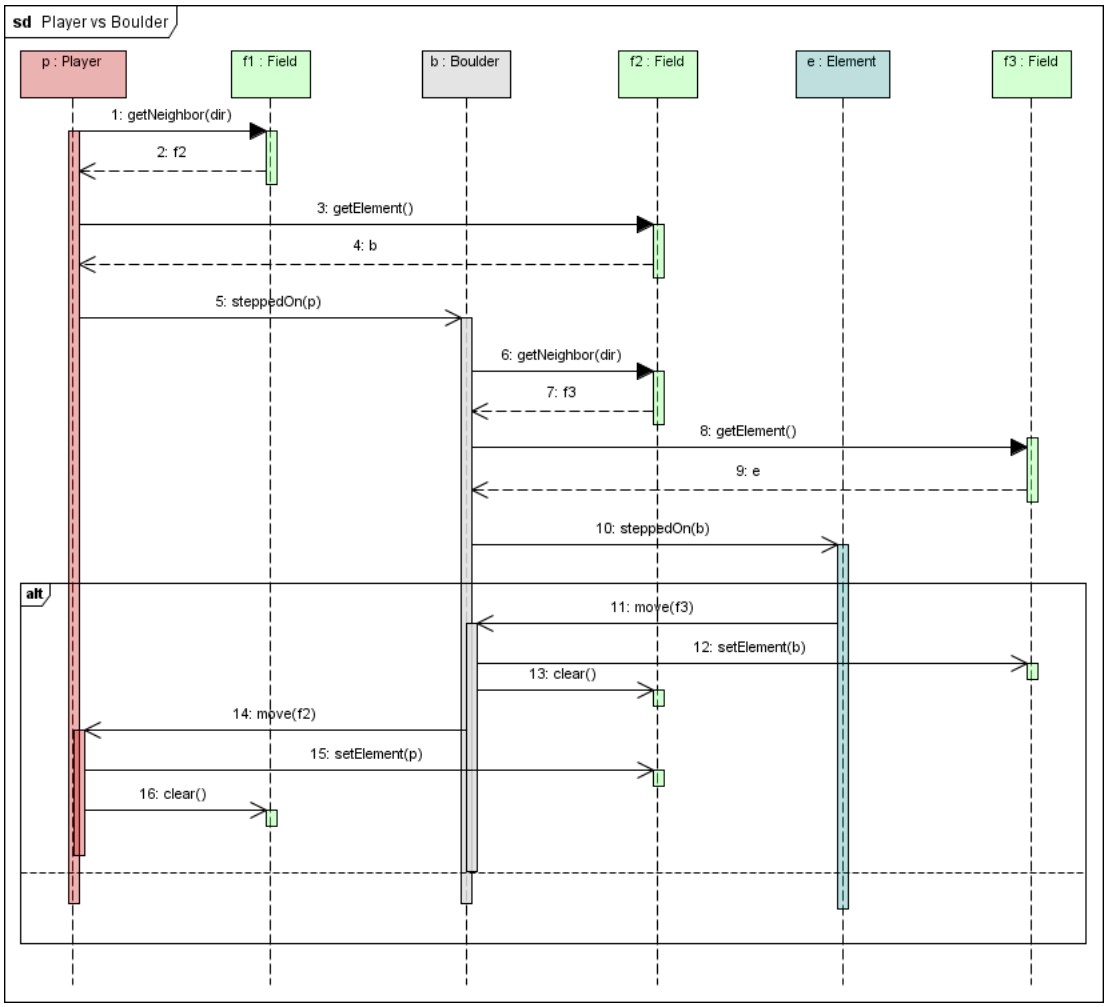
6. ábra. Játék indítása



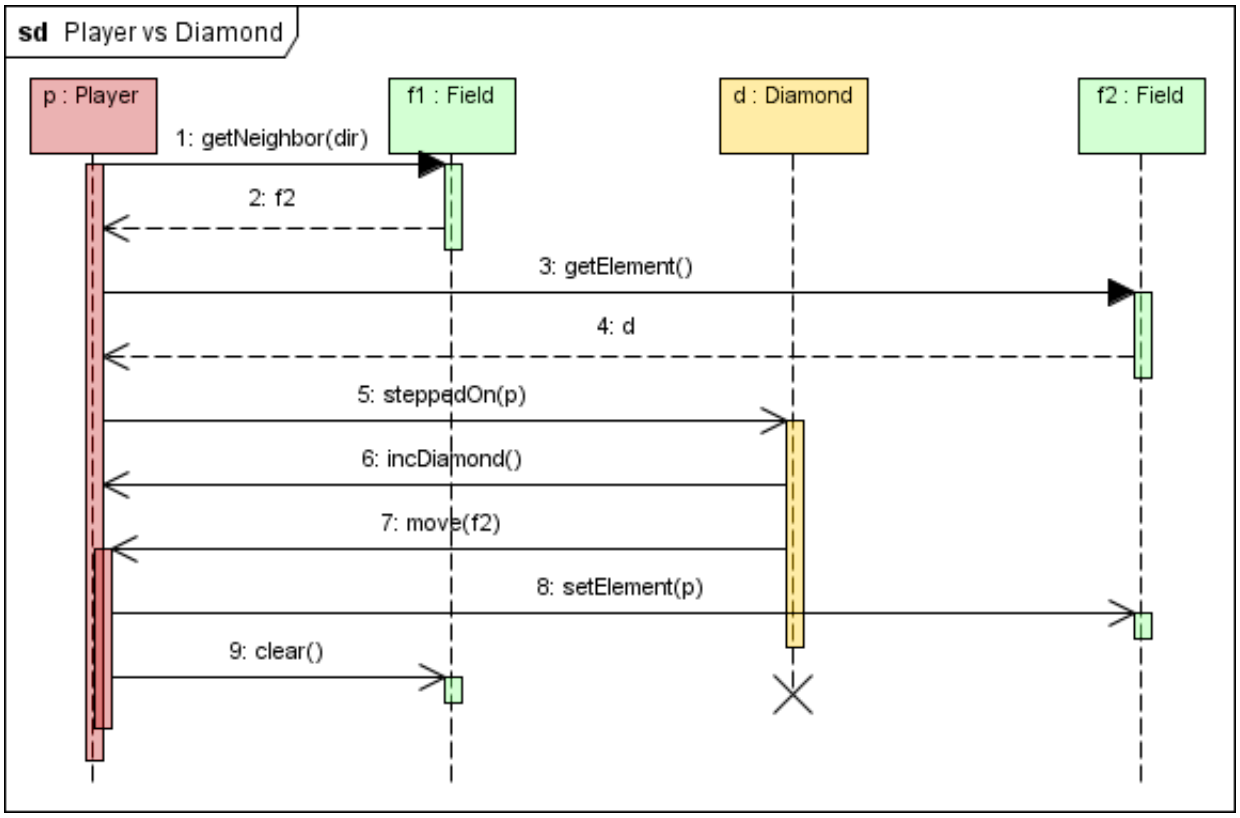
7. ábra. Karakter vezérlése



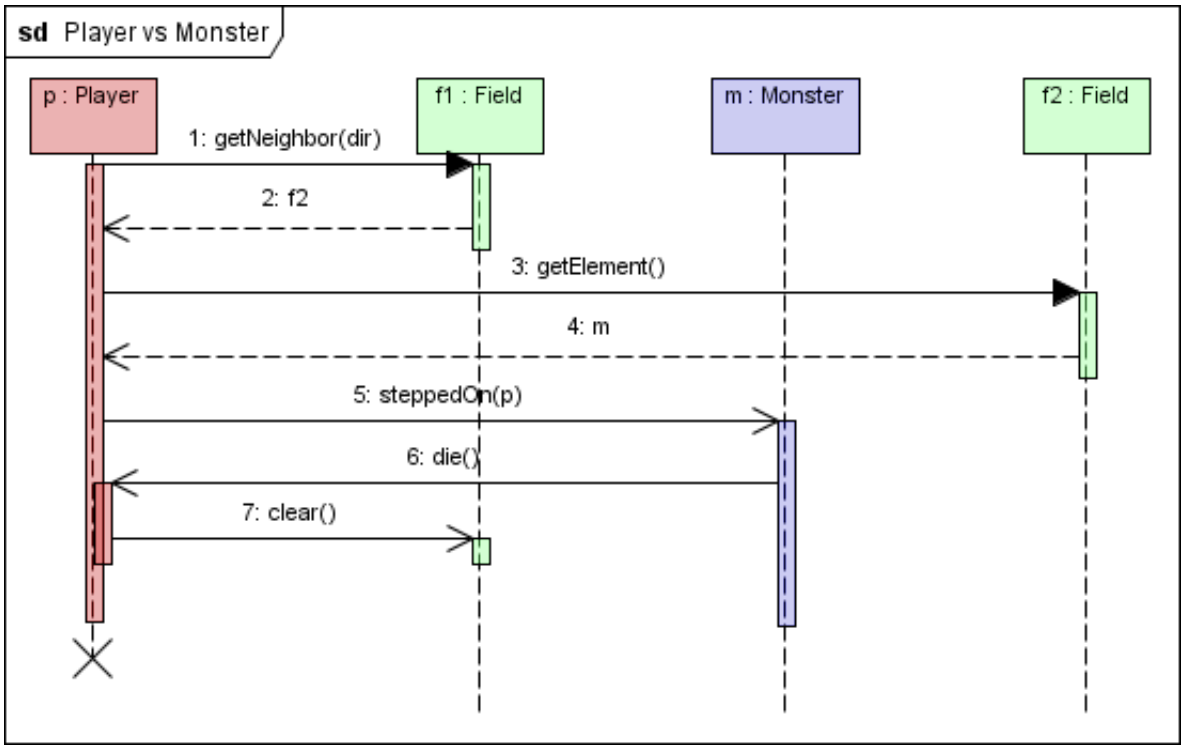
8. ábra. Idő léptetése



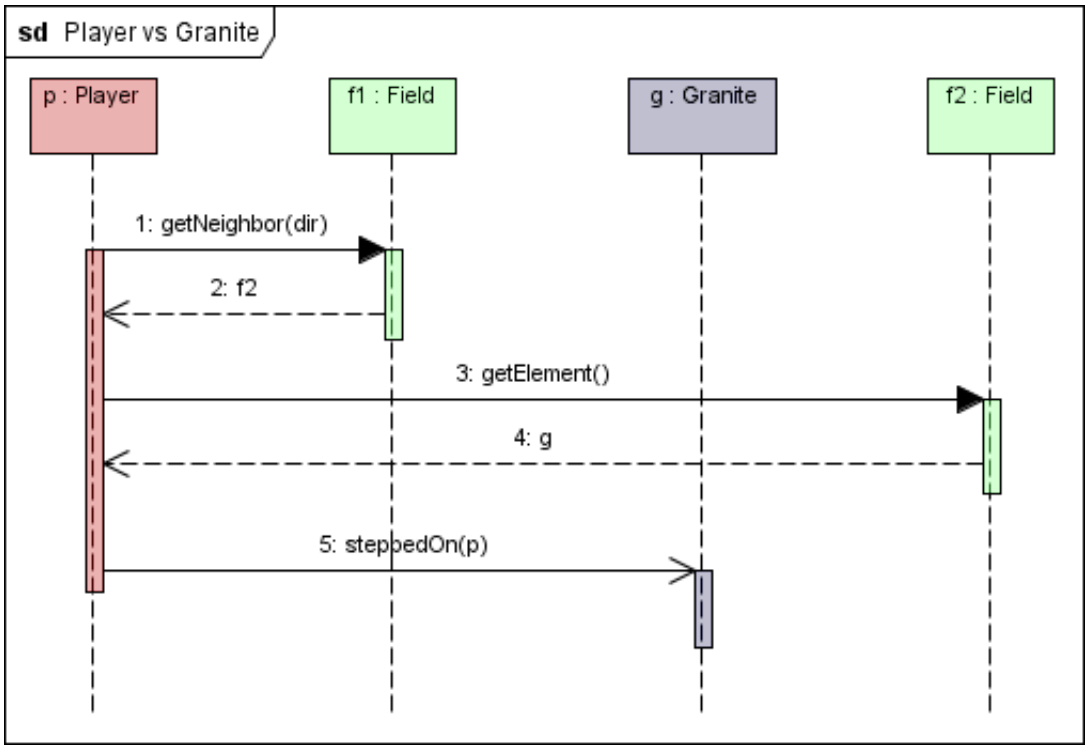
9. ábra. Karakter találkozása sziklával



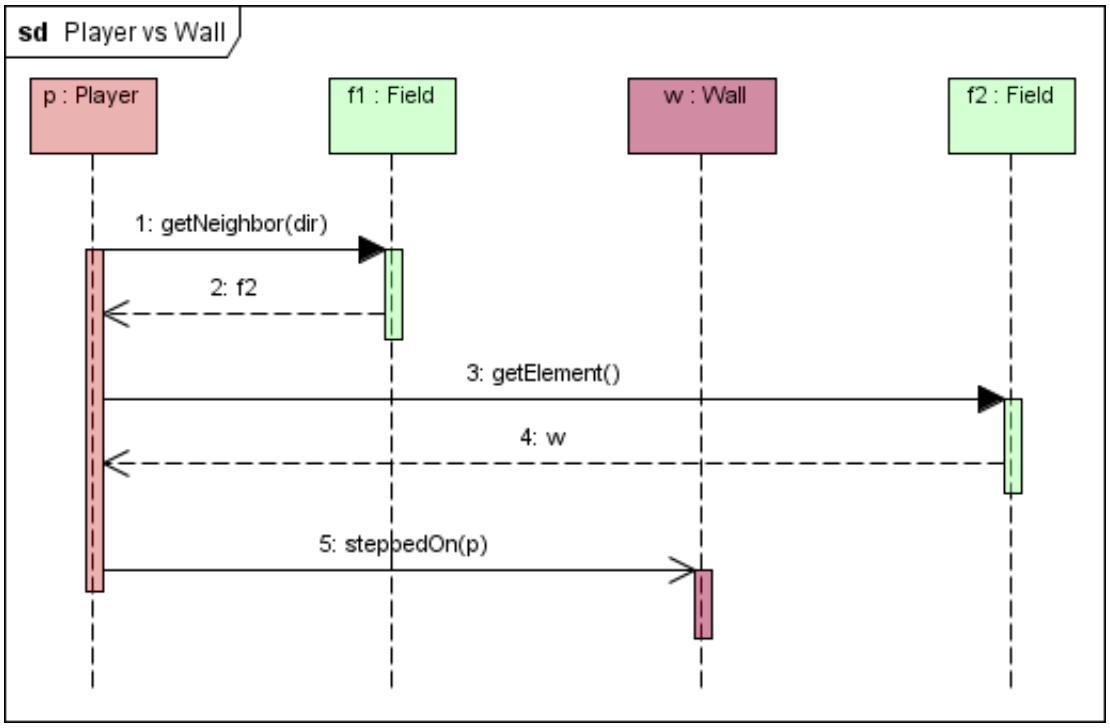
10. ábra. Karakter találkozása gyémánttal



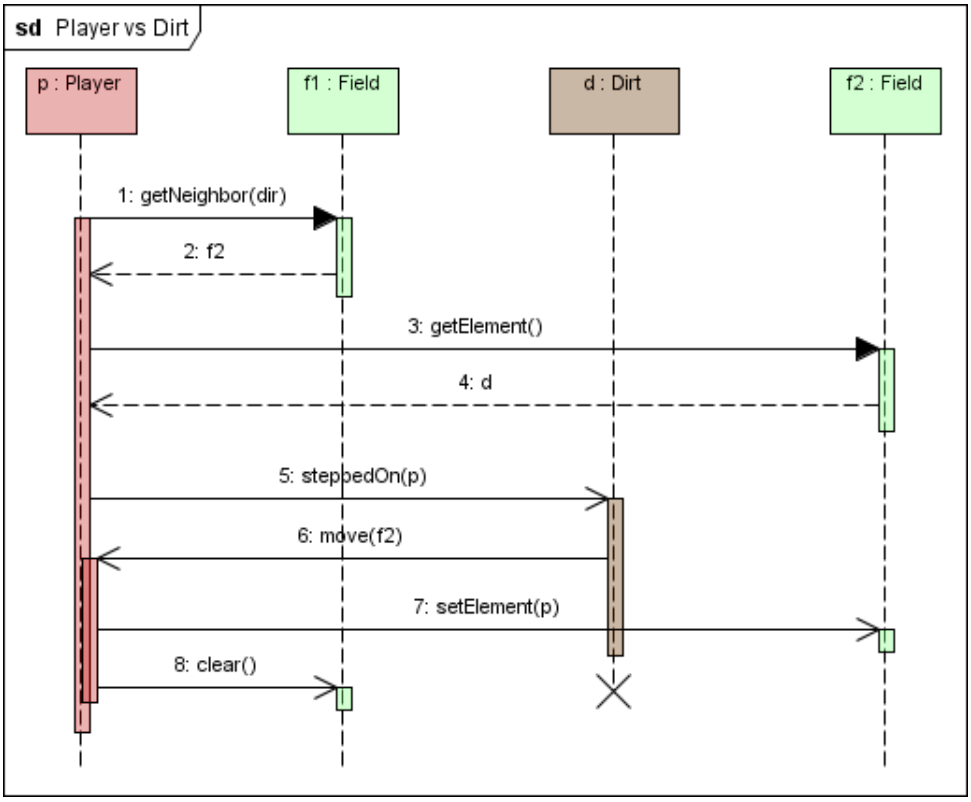
11. ábra. Karakter találkozása szörnnyel



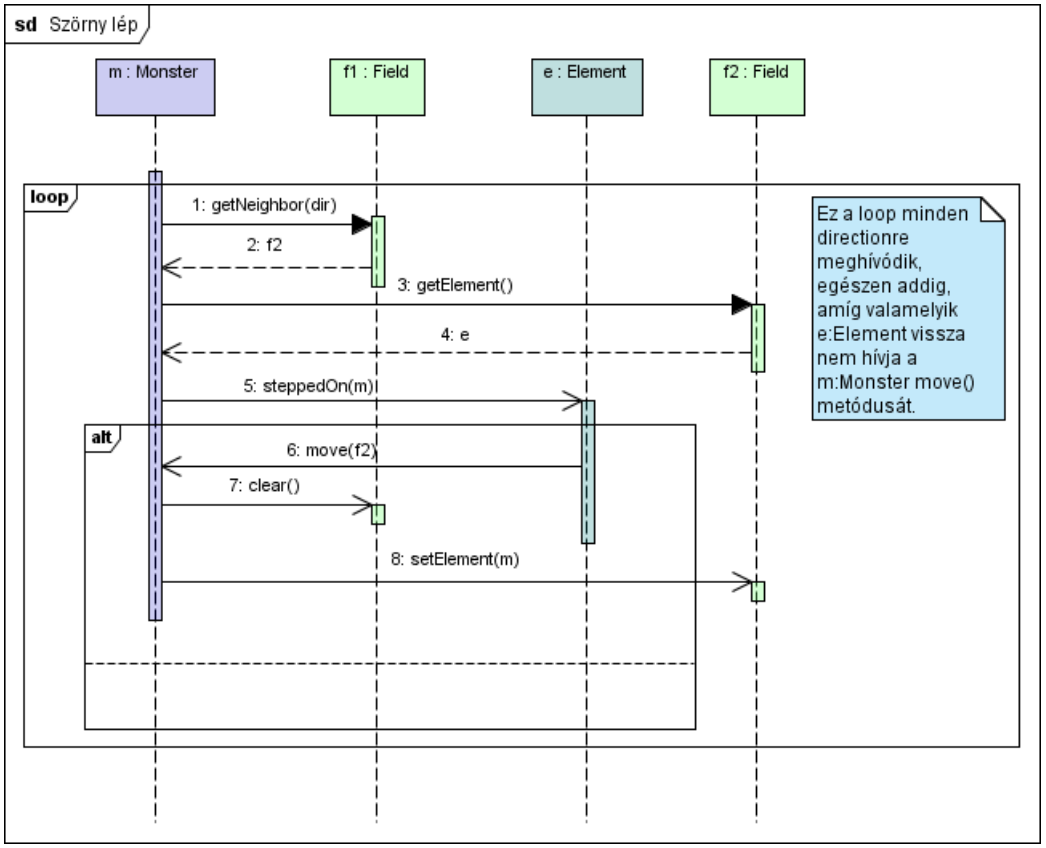
12. ábra. Karakter találkozása gránittal



13. ábra. Karakter találkozása fallal

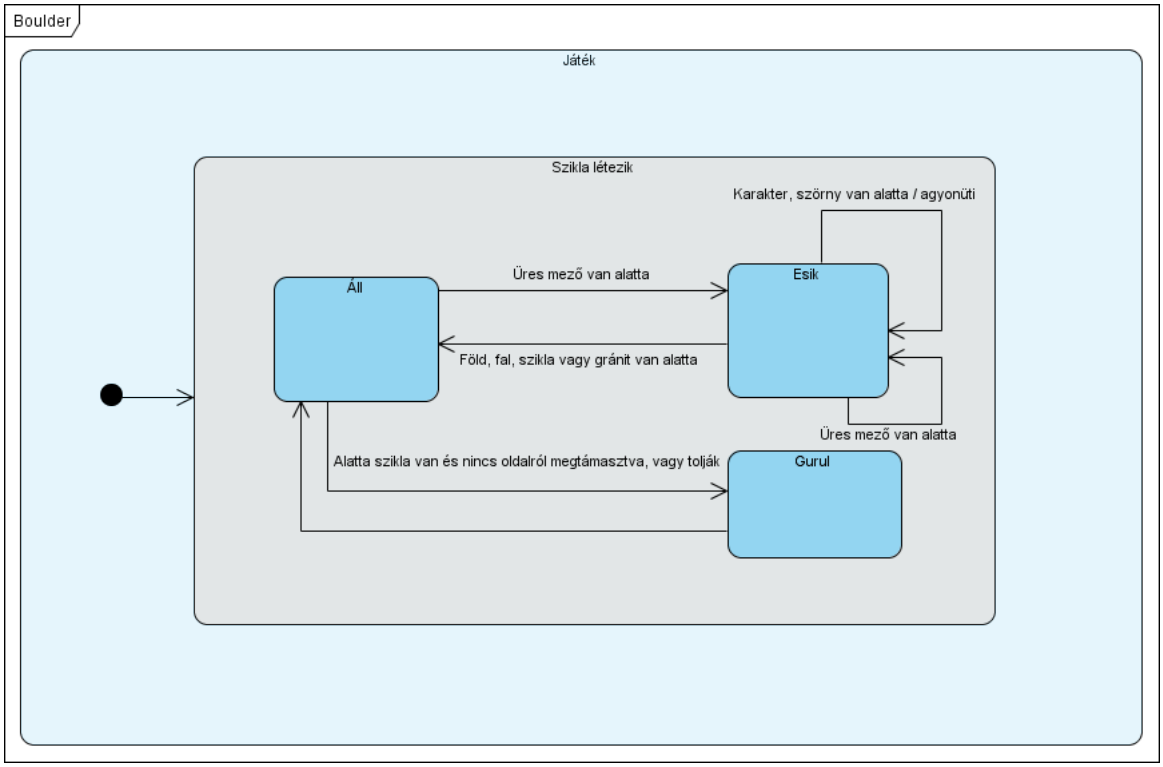


14. ábra. Karakter találkozása földdel

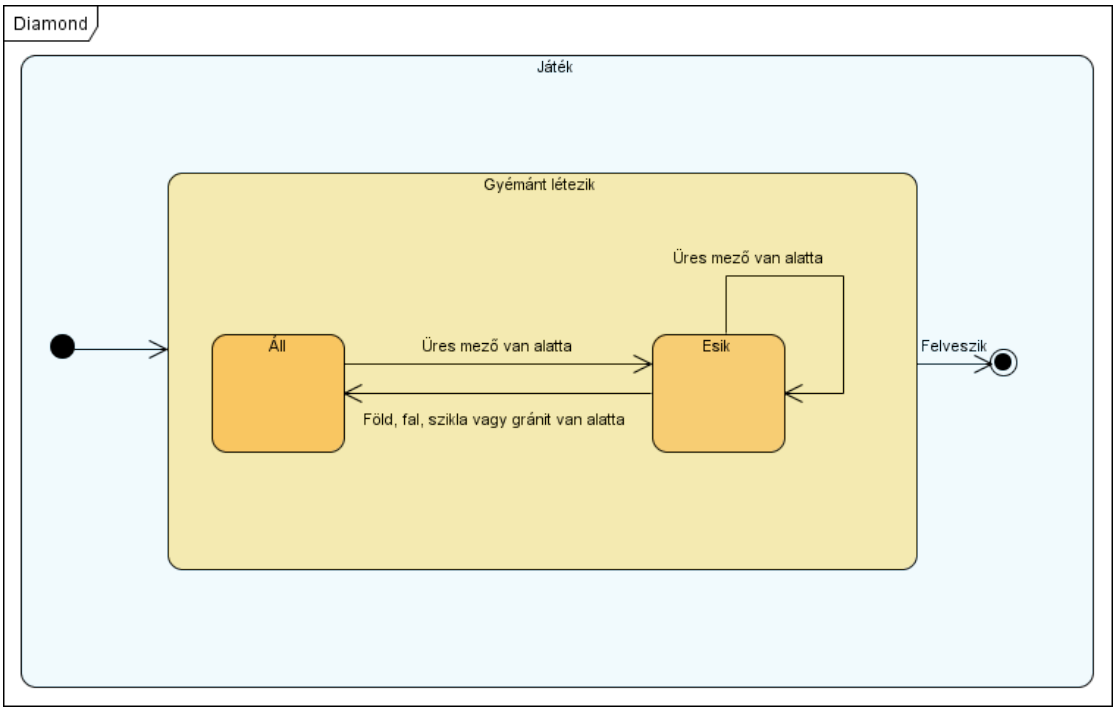


15. ábra. Szörny lépése

2.5. State-chartok



16. ábra. A szikla (Boulder) state-chartja



17. ábra. A gyémánt (Diamond) state-chartja

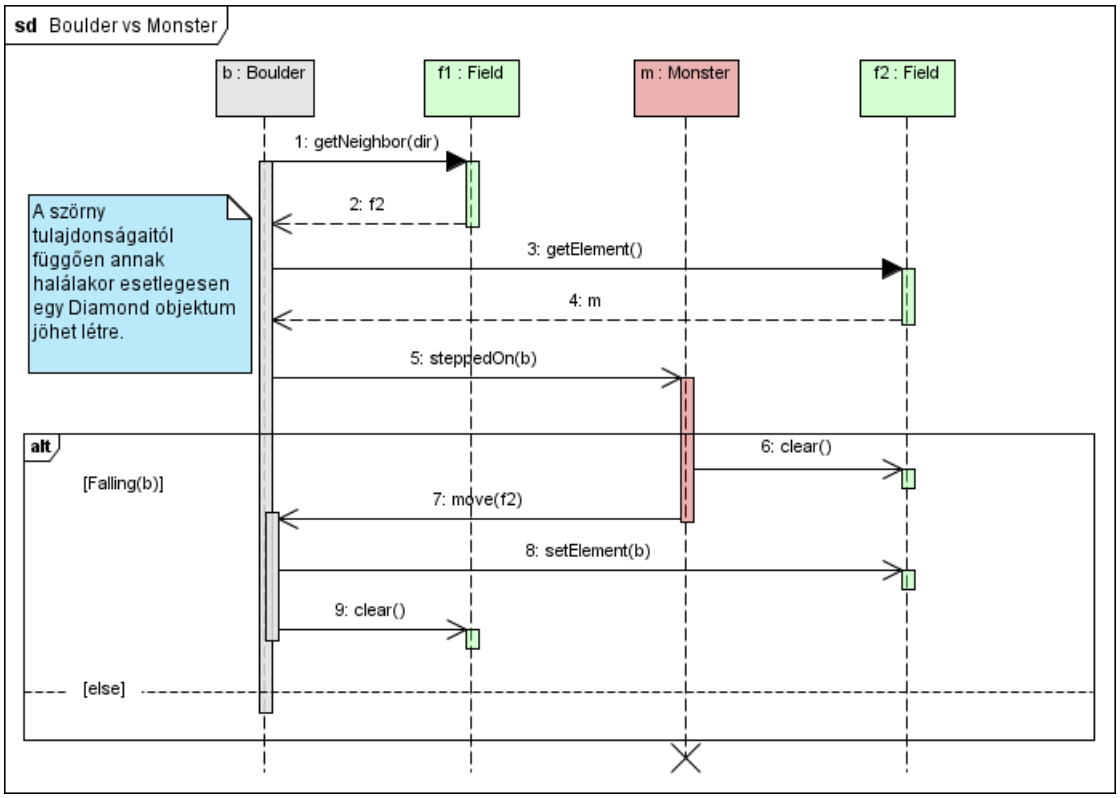
2.6. Napló

2009.02.19. 10:00	Zsolnai, Siklósi, idő: 01:30
Class diagram vázlat rajzolás (3.3)	
2009.02.20. 08:00	String Quartet, idő: 02:00
Class diagram finomítása (3.3)	
2009.02.21. 19:00	Farkas, Siklósi, idő: 01:30
Modell reprezentáció finomítása (3.3)	
2009.02.23. 14:00	String Quartet, idő: 06:00
Objektummodell megalkotása, konzultáció (3.3)	
2009.02.23. 23:00	Zsolnai, idő: 01:00
State chartok létrehozása (3.5)	
2009.02.23. 23:00	Tóth-Máté, idő: 02:00
Class diagram létrehozása (3.3)	
2009.02.24. 00:00	Zsolnai, idő: 00:30
Csapatszervezés	
2009.02.24. 12:00	Farkas, Siklósi, idő: 01:00
Szekvencia diagramok készítése (3.4)	
2009.02.24. 13:00	String Quartet, idő: 03:00
Szekvencia diagramok készítése (3.4)	
2009.02.24. 16:30	Siklósi, idő: 01:30
Szekvencia diagramok egységesítése, javítása (3.4)	
2009.02.24. 18:00	Tóth-Máté, idő: 02:00
Class diagram véglegesítése (3.3)	
2009.02.24. 19:00	Siklósi, idő: 02:00
Objektum katalógus elkészítése (3.1)	
2009.02.24. 20:00	Tóth-Máté, idő: 01:00
Maradék szekvencia diagramok készítése (3.4)	
2009.02.24. 22:00	Zsolnai, idő: 01:00
Szekvencia diagramok készítése (3.4)	
2009.02.24. 23:00	Zsolnai, idő: 00:30
Osztályok leírásának elkészítése (3.2)	

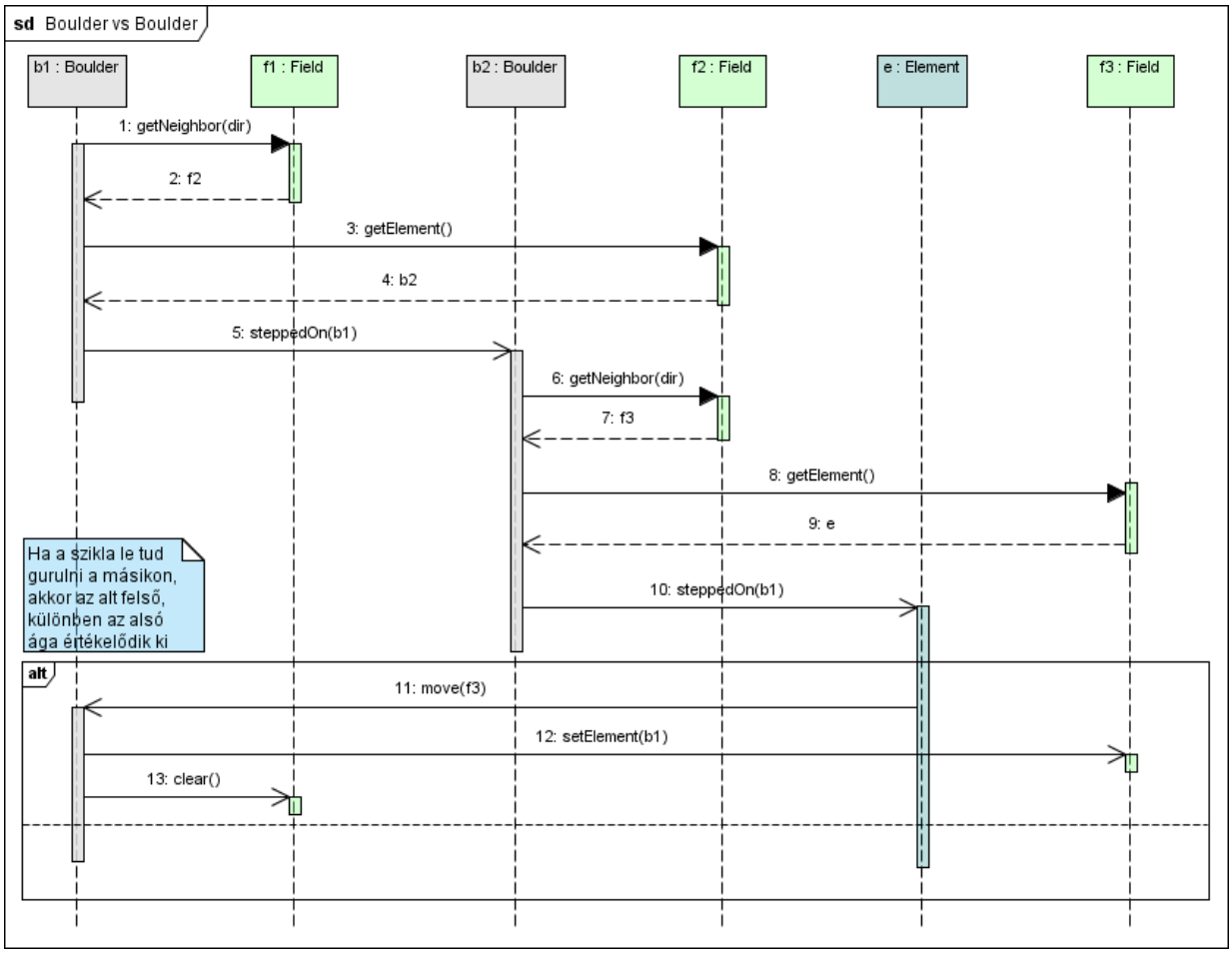
2009.02.25. 15:00	Tóth-Máté, idő: 02:00
Dokumentáció készítése, lektorálás, hibajavítás a megbeszéltek szerint	
2009.02.25. 10:00	Siklósi, idő: 01:00
Objektum katalógus és osztályleírás bővítése, revíziója (3.1)	
2009.02.25. 18:00	Siklósi, idő: 01:30
UML diagramok stílusainak, színvilágának kialakítása	
2009.02.25. 20:30	Farkas, idő: 01:00
Dokumentáció és szekvencia diagramok átnézése, javaslatok	
2009.02.25. 19:00	Tóth-Máté, idő: 01:00
Dokumentáció véglegesítése	
2009.02.25 22:30	Farkas, Zsolnai, idő: 00:30
Lapzárta, javítások	
2009.02.25 22:30	Tóth-Máté, idő: 00:30
Lapzárta, javítások	

3. Analízis modell kidolgozása 2.

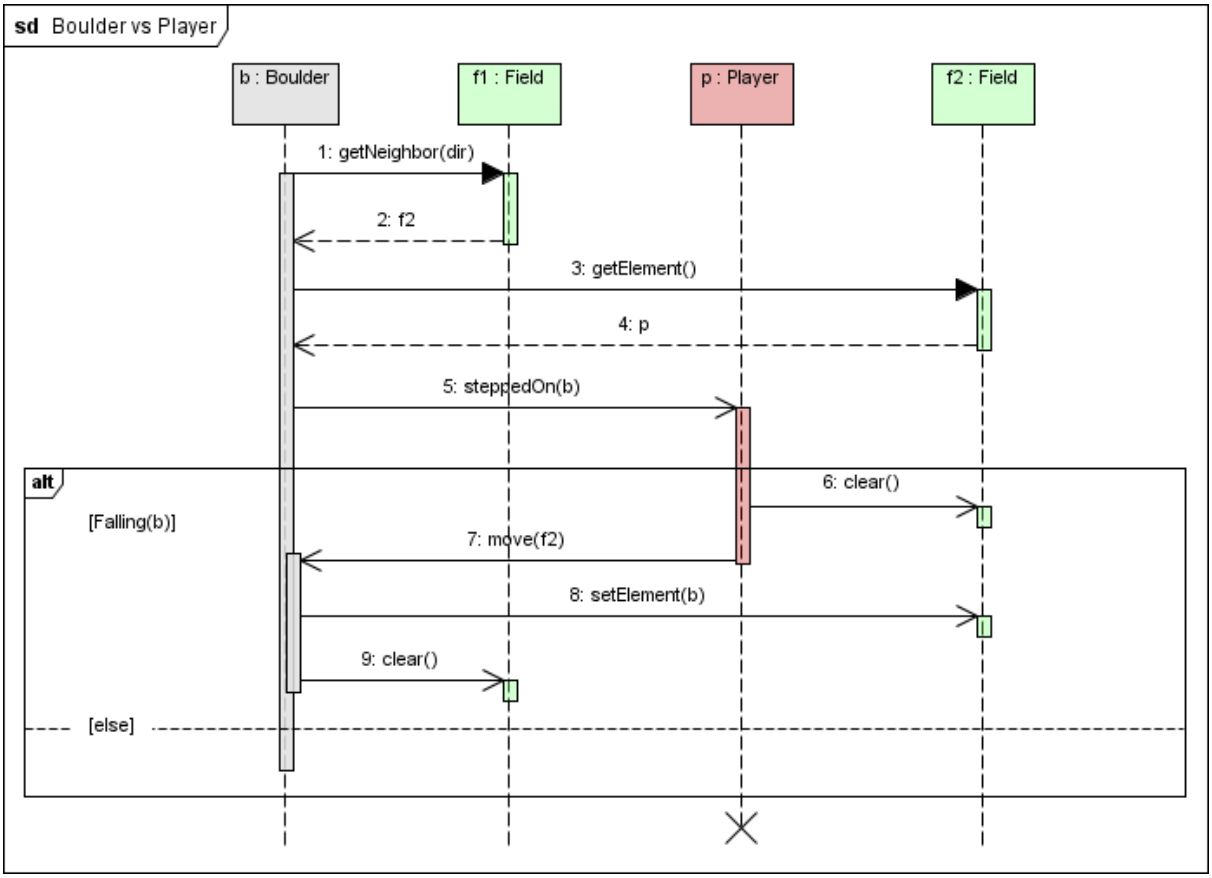
3.1. További szekvencia diagramok



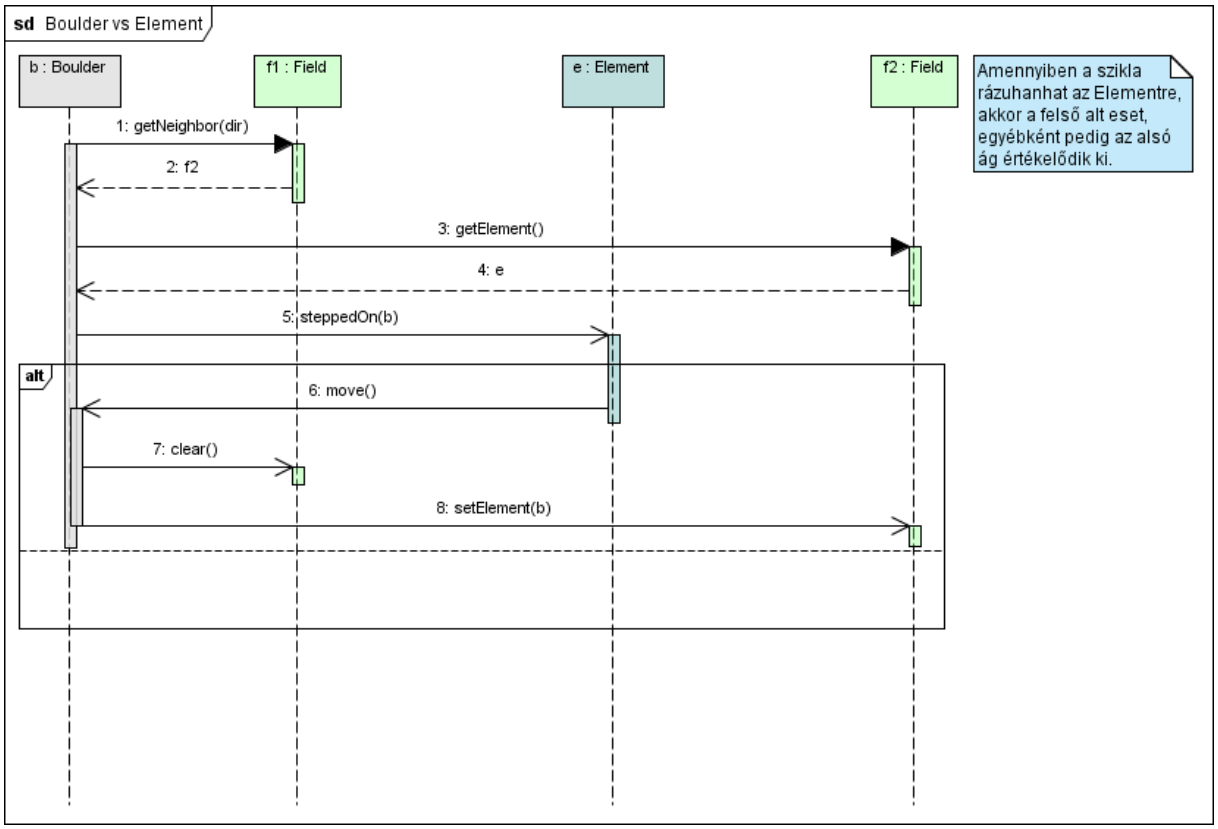
18. ábra. Szikla ráesik egy szörnyre



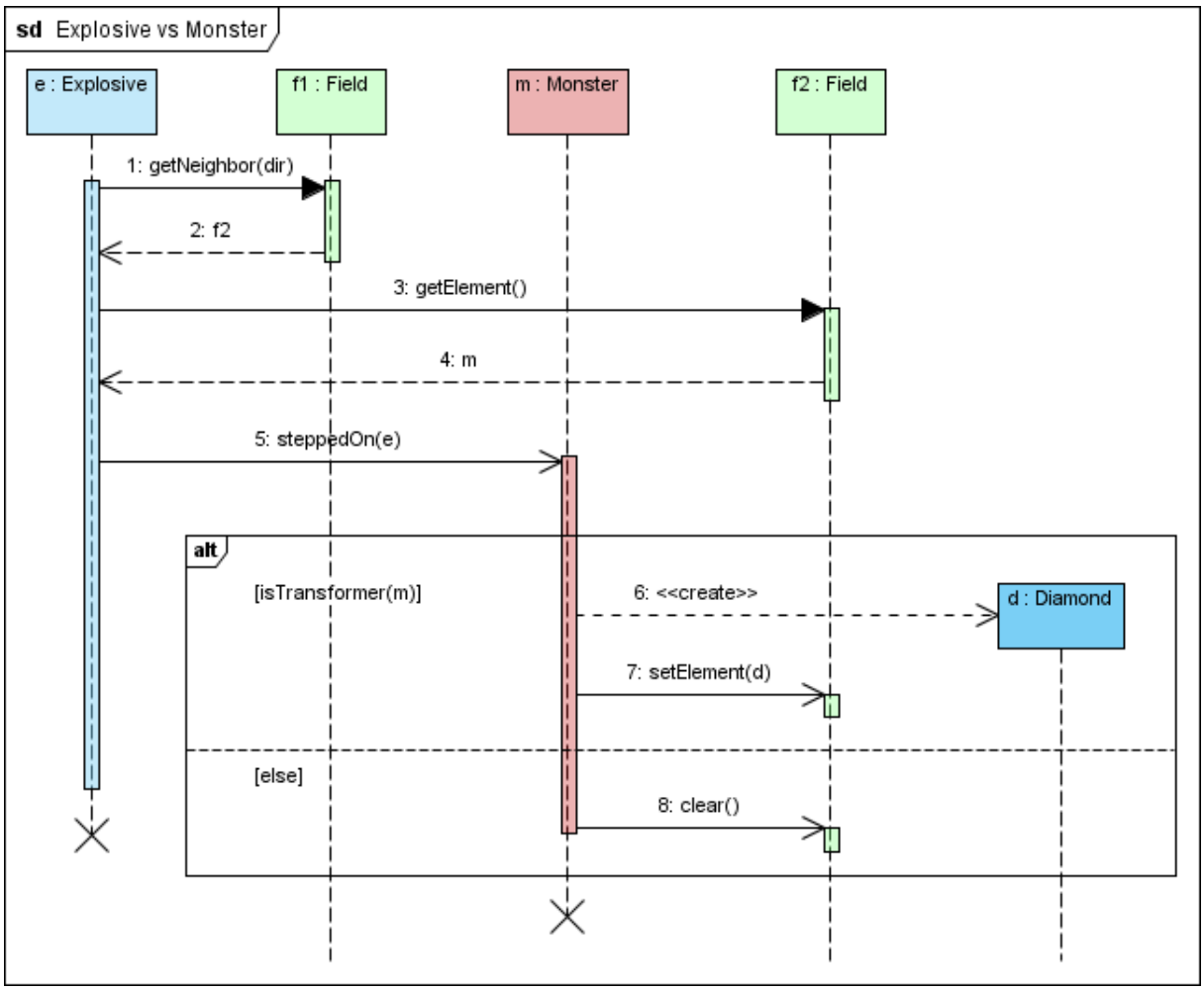
19. ábra. Szikla legurul egy másik szikláról



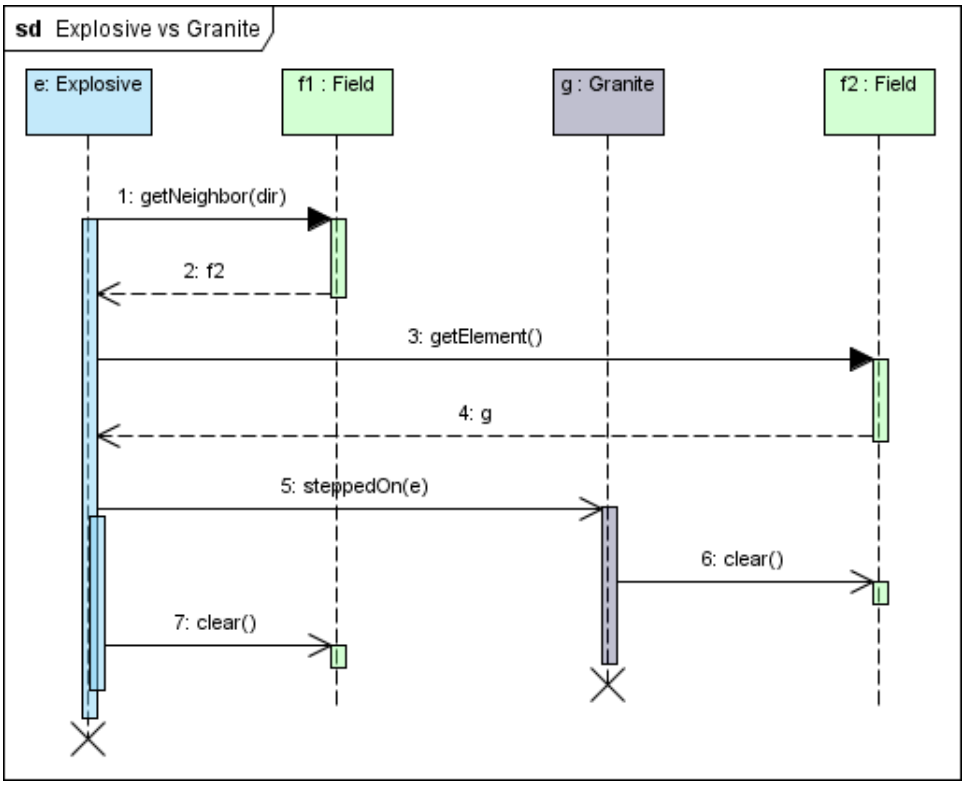
20. ábra. Szikla találkozása karakterrel



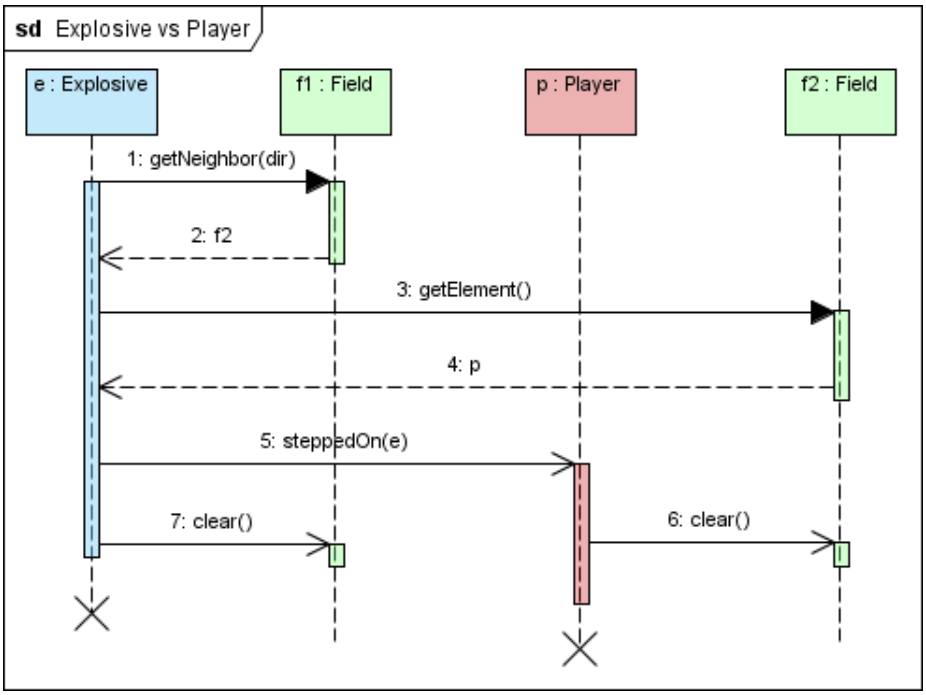
21. ábra. Szikla leesik bármely más elemre



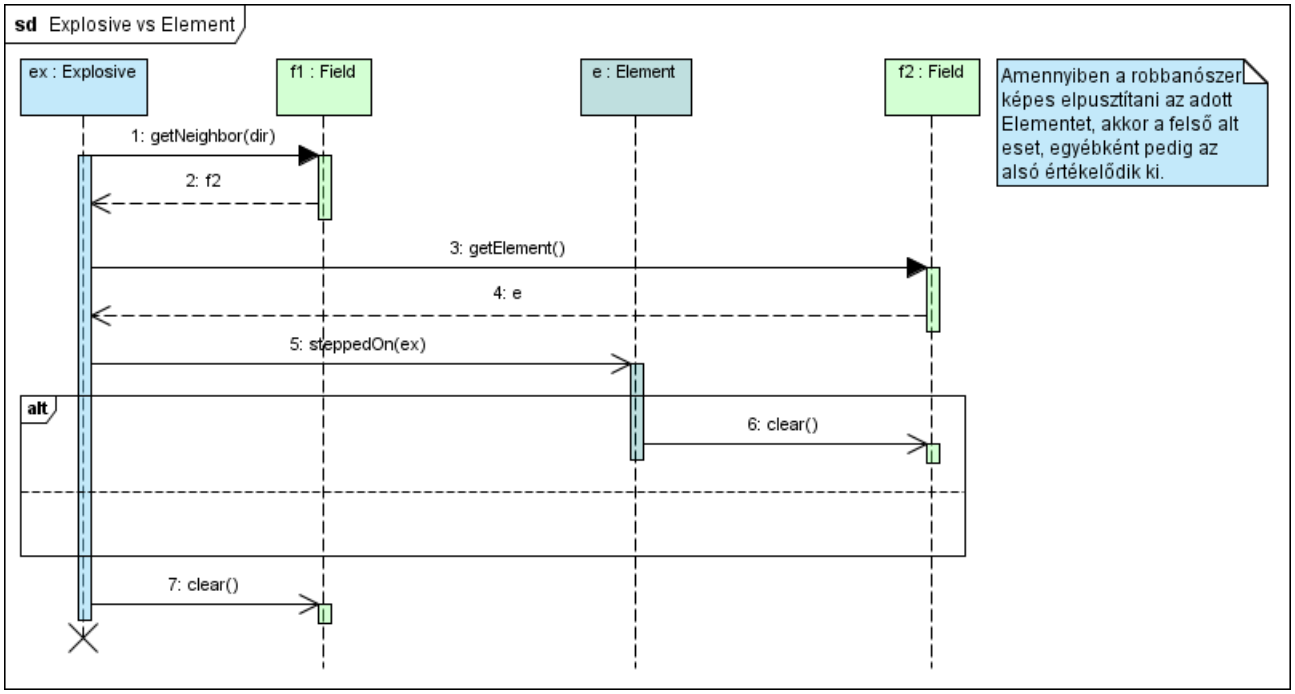
22. ábra. Bomba felrobbantja a szörnyet



23. ábra. Bomba felrobbantja a gránitot



24. ábra. Bomba felrobbantja a karaktert



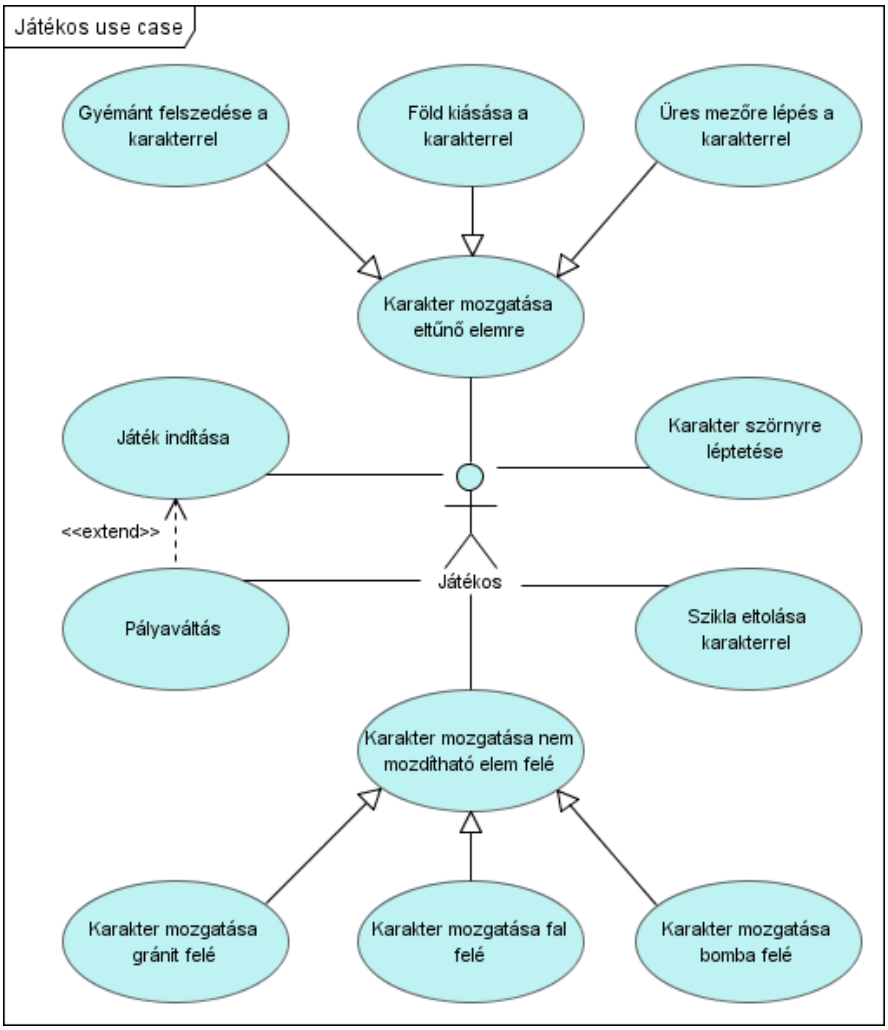
25. ábra. Bomba felrobban bármilyen más elem mellett

3.2. Napló

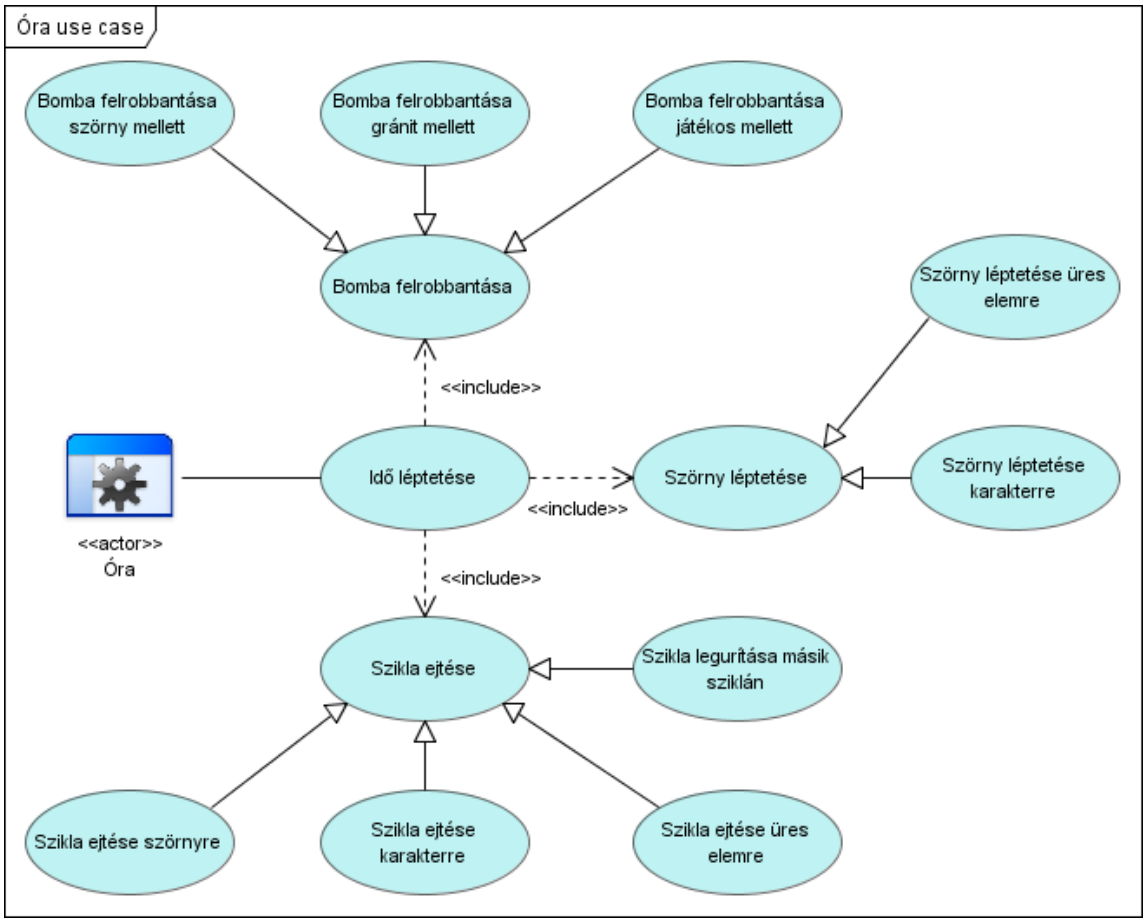
2009.03.03. 12:00	Siklósi, idő: 01:00
Szekvencia diagramok rajzolása (4.1)	
2009.03.03. 14:00	Tóth-Máté, idő: 01:00
Szekvencia diagramok rajzolása (4.1)	
2009.03.03. 17:00	Tóth-Máté, idő: 00:30
Szekvencia diagramok átnézése (4.1)	
2009.03.03. 21:00	Zsolnai, idő: 01:30
Szekvencia diagramok rajzolása, átnézése (4.1)	
2009.03.03. 23:00	Tóth-Máté, idő: 01:00
Szekvencia diagramok lektorálása (4.1)	
2009.03.04. 12:00	Farkas, idő: 01:30
Szekvencia diagramok rajzolása (4.1)	
2009.03.04. 21:00	Zsolnai, idő: 00:30
Szekvencia diagramok lektorálása (4.1)	
2009.03.04. 23:00	Tóth-Máté, idő: 00:30
Leadandó dokumentum apróbb javítások, exportálás	

4. Skeleton tervezése

4.1. A skeleton modell valóságos use-case-ei



26. ábra. Játékos use case-ei



27. ábra. Óra use case-ei

4.2. Architektúra

A skeleton az egyes use case-ek tesztelésére hivatott, melyek lefutása során ellenőrizhető, hogy a valóságban is pontosan a szekvencia diagramokban megállapított módon követik egymást a folyamatok. Ezeket a használati eseteket előre megtervezett és lekódolt pályákon nyílik lehetőség tesztelni, így a skeleton tervezése folyamán perzisztencia használatára nem lesz szükség. A folyamatok egymás után, szekvenciálisan kerülnek meghívásra, így a skeletonban azok konkurens futtatására nem lesz szükség.

Az egyes use case-ek lefutása hasonlóan történik, így az egymással analóg esetek osztályokba sorolhatók, a tesztpályák pedig ezen osztályok tesztelésére szolgálnak. Minden pálya leírásában szerepel, hogy konkrétan mely esetet mutatja be, és hogy ez mely más, azzal analóg típusú use case-eket fed le. Ezek a hasonlóságok a korábban mellékelt szekvencia diagramokról, ellenőrzésképpen leolvashatóak. A csillaggal jelölt analógiák szintén helytállóak, azzal a különbséggel, hogy egy-egy feltétel kiértékelésében különbözhetnek az eredeti példákban szerepeltektől.

A csapat minden eset bemutatására a lehető legminimálisabb pályareprezentációt igyekezett megtervezni, amely képes az objektumok közötti kommunikáció teljes értékű bemutatására.

1. tesztpálya:

Mezők száma: 2
Mezők tartalma: Player, Monster
Analóg esetek: Monster vs Player, Boulder vs Player, Explosive vs Player, Boulder vs Monster*, Explosive vs Monster

Az első tesztpálya azt mutatja be, hogy mi történik, amikor egy szörny mozgáskor arra a mezőre lép, ahol a játékos is tartózkodik.

2. tesztpálya:

Mezők száma: 2
Mezők tartalma: Player, Wall

Analóg esetek: Player vs Granite, Player vs Explosive, Monster vs Boulder, Monster vs Wall, Monster vs Granite, Monster vs Diamond, Monster vs Explosive

A játékos a fal irányába szeretne mozogni, azonban beleütközik. A két elem kommunikációja kerül bemutatásra.

3. tesztpálya:

Mezők száma: 3
Mezők tartalma: Player, Boulder, Empty

A játékos sziklát tartalmazó mezőre mozog, így amennyiben az a másik oldalról nincs megtámasztva (üres elem van rajta), eltolja azt.

4. tesztpálya:

Mezők száma: 4
Mezők tartalma: Boulder, Boulder, Empty, Granite

Az egymás tetején elhelyezkedő sziklák egymásról elgurulnak.

5. tesztpálya:

Mezők száma: 2
Mezők tartalma: Player, Diamond

A játékos gyémántot tartalmazó mezőre lép, és tarisznyájába helyezi azt.

6. tesztpálya:

Mezők száma: 2
Mezők tartalma: Player, Dirt

A játékos földterületre lépve kivájja azt.

4.3. A skeleton kezelői felületének terve, dialógusok

A skeleton, mint program célja annak bizonyítása, hogy a programunk statikus és dinamikus modelljeiről leképzett programváz képes-e az elvárt működést produkálni. A skeletonban minden objektum szerepel, de azoknak csak az interfésze definiált. Minden egyes metódus hívásakor a konzolon (`System.out`) kiírja az őt tartalmazó objektum típusát és azonosítóját, saját nevét, valamint paraméterlistáját. Ezután a metódus meghívja a működéséhez szükséges további metódusokat. Amennyiben döntési helyzet áll elő, a program futása ideiglenesen szünetel, felhasználói interakcióra van szükség. Ilyenkor a felhasználónak feltett eldöntendő kérdésre adott választól függ a folytatás.

A skeletonnak alkalmasnak kell lennie szekvencia diagramok ellenőrzésére. Az egyszerű, karakteres képernyőkezelés biztosítja a rendszer egyszerűségét. Metódushívás esetén a kiírás a következő formában történik (rendre egymás után egy sorban):

- Objektum neve (előfordulhatnak névtelen objektumok, ilyenkor a hashkódja azonosítja az adott entitást)
- Osztály neve
- Objektum hashcode-ja (integer típusú, Java specifikus objektum-azonosító)
- Metódus neve és paraméterlistája (az esetlegesen átadott objektumok típusa, neve, hashcode-ja)
- Hívás esetén `CALL`, visszatérés esetén `RETURN`

Példa:

```
<@Explosive counter reached zero?  (y/n)
@>y
e|Explosive|885| -> f1|Field|1223|getNeighbor(int dir)|CALL
e|Explosive|885| <- f1|Field|1223|getNeighbor(int dir)|RETURN[Field|f2|1235|]
e|Explosive|885| -> f2|Field|1235|getElement()|CALL
e|Explosive|885| <- f2|Field|1235|getElement()|RETURN[Player|p|950|]
e|Explosive|885| -> p|Player|950|steppedOn(Explosive e)|CALL
p|Player|950| -> f2|Field|1235|clear()|CALL
e|Explosive|885| -> f2|Field|1235|clear()|CALL
```

4.4. Szekvencia diagramok a belső működésre

A belső működést leíró use case-ek szekvencia diagramjai elérhetőek a dokumentáció 3.4 és 4.1 fejezeteiben.

4.5. Napló

2009.03.09. 10:00	String Quartet, idő: 02:00
Tennivalók áttekintése, skeleton alapjainak megbeszélése	
2009.03.09. 18:00	Farkas, Zsolnai, Tóth-Máté, idő: 00:30
Use case-ek és belső szekvenciadiagramok összeírása	
2009.03.10. 11:00	Tóth-Máté, idő: 01:00
Use case-ek rajzolása (5.1)	
2009.03.10. 14:00	String Quartet, idő: 01:30
Skeleton megbeszélése	
2009.03.10. 15:30	Farkas, Siklósi, idő: 01:00
A skeleton kezelői felületének terve, dialógusok (5.3)	
2009.03.10. 15:30	Zsolnai, idő: 00:30
Architektúra, szekvencia diagramok a belső működésre (5.2, 5.4)	
2009.03.10. 20:00	Tóth-Máté, idő: 00:30
Leadandó dokumentum szerkesztése	
2009.03.10. 21:00	Zsolnai, idő: 00:30
Leadandó dokumentum lektorálása	

5. Skeleton beadása

5.1. A skeleton

5.1.1. Fordítási és futtatási útmutatás

A fordítás és futtatás egyszerűsítése érdekében az ezeket a feladatokat automatikusan elvégző batch fájlok kerültek elhelyezésre. A könyvtárszerkezet /bin/ mappája alatt található skeleton_compile.bat fájl a gépen található java fordító segítségével a forrásfájlokból állítja elő a használható, kész programkódot, amely ezt követően a skeleton_run.bat állomány futtatásával indítható.

A program a konzolos megjelenítői felület segítségével lehetővé teszi, hogy menüből kiválasszuk a tesztelendő use-caseket, amelyeket a billentyűzet megfelelő gombjainak lenyomásával kérhetünk.

A konkrét tesztesetek futtatásakor a felhasználó kérdéseket kap, melyek legtöbbször eldöntendő típusúak, ezekre a válaszokat pedig a hozzájuk tartozó billentyűk megnyomásával adhat, melyről a felhasználói felület tájékoztatást ad.

Megjegyzés: A szöveg jobb átláthatósága érdekében érdemes lehet a konzol méretének nagyobbra állítása.

Ezenkívül szükséges feltétel, hogy a Java binárisainak elérési útja be legyen állítva a megfelelő környezeti változóban (pl. Microsoft Windows alatt a `set PATH` paranccsal).

5.1.2. A skeleton fájljai

A kész skeleton a következő fájlokat tartalmazza, könyvtáranként való lebontásban: (A kiírt fájlméretek byte-ban értendők)

Az /src/ könyvtár a programot alkotó forráskódokat tartalmazza.

Fájlnév	Méret	Létrehozás	Leírás
Boulder.java	3051	2009.03.11. 16:47	A szikla elemet megvalósító osztály
Cave.java	499	2009.03.11. 16:47	A játékteret megvalósító osztály
Diamond.java	721	2009.03.11. 16:47	A gyémánt elemet megvalósító osztály
Dirt.java	592	2009.03.11. 16:47	A föld elemet megvalósító osztály
Element.java	3755	2009.03.11. 16:47	A térelemek ősosztálya
Empty.java	492	2009.03.11. 16:47	Üres mező elemet megvalósító osztály
Exit.java	433	2009.03.11. 16:47	Kijárat elemet megvalósító osztály
Explosive.java	826	2009.03.11. 16:47	Robbanóanyag elemet megvalósító osztály
Field.java	2267	2009.03.11. 16:47	Mező elemet megvalósító osztály
Game.java	1925	2009.03.11. 16:47	A játék elemeit összefogó osztály
Granite.java	435	2009.03.11. 16:47	Gránit elemet megvalósító osztály
Monster.java	3260	2009.03.11. 16:47	A szörny ellenfeleket megvalósító osztály
Player.java	2789	2009.03.11. 16:47	A játékos karaktert megvalósító osztály
Skeleton.java	5876	2009.03.11. 16:47	A skeleton főosztálya
Timer.java	223	2009.03.11. 16:47	Az időzítésért felelős osztály
Wall.java	96	2009.03.11. 16:47	A fal elemet megvalósító osztály

A /bin/ a futtatható és lefordított állományokat tartalmazó könyvtár:

Fájlnév	Méret	Létrehozás	Leírás
compile.bat	24	2009.03.15. 18:53	A skeleton fordításához szükséges batch fájl
docgen.bat	118	2009.03.15. 18:53	A javadoc dokumentációt elkészítő batch fájl
run.bat	19	2009.03.15. 18:53	A skeletont futtató batch fájl

5.2. Értékelés

A tagok rögtön a közös projekt első hetében szembesültek azzal, hogy az egyéni és a csapatmunka jelentős különbségekkel jár. Hamar kiderült számukra, hogy a kommunikáció és az információcsere, a közös munka hatékony szinkronizációja alapvető fontosságú, melyek megvalósításában a korábban bemutatott wiki és a verziókezelő rendszer volt segítségükre.

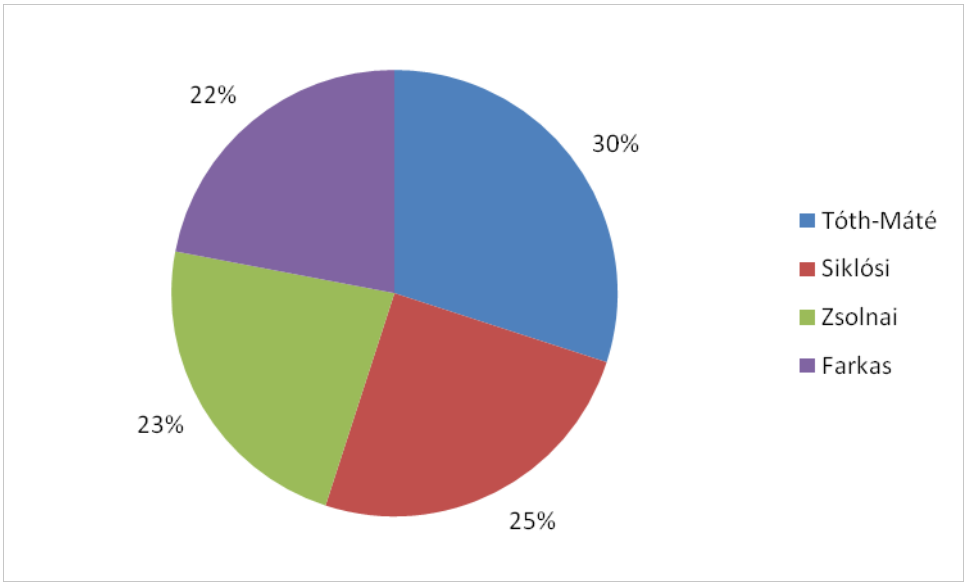
Az első két heti munka elkészítése folyamán a napló és a dokumentáció elkészítése igen sok időt vett igénybe, így a tagok mindezen folyamatok automatizálása mellett döntöttek. Ehhez segítségképpen Tóth-Máté egy PHP-scriptet fejlesztett, amely képes a wikin tárolt naplót automatikusan LaTeX formátummá transzformálni, így az egyetlen kattintással a dokumentációba helyezhető. Ez az időbeli befektetés már többszörösen megtérült a csapat számára.

Szinte minden munkafolyamatban megjelentek apróbb komplikációk, melyek kezelése, és a korábbi dokumentációk ennek megfelelő változtatásai eredményesen zajlottak. A résztvevők aktív kommunikációt folytatnak a többi csapat tagjaival, az ott hallott gyakori kommunikációs és implementációs hibák ismerete segítséget nyújtott azok elkerülésében, megoldásában.

A tagok tudásban megfelelően kiegészítik egymást, így igen fontos tényező a közös munka megfelelő szervezése, ami a vártnál komolyabb erőfeszítéseket igényelt, ám ezzel együtt megtérülő befektetésnek bizonyult, mivel a feladatok megfelelő felosztása és hozzárendelése alapvetően és pozitívan befolyásolta a kész produktum minőségét. Az eddigi feladatmegoldások tapasztalatai alapján a munkafolyamatok lefolyása folyamatos fejlődést mu-

tatott, emellett pedig a társaság különös figyelmet szentelt az első szakasz lehető legtökéletesebb megtervezésének és implementálásának - a további szakaszok erre épülnek rá, így nagyban függenek annak sikerességétől.

A csapatban végzett munkaórák eloszlása:
Tóth-Máté: 45:55h
Siklósi: 38:45h
Zsolnai: 35:45h
Farkas: 35:00h
Teljes munkaidő: 155:25h



28. ábra. Munkaidők megoszlása százalékosan

A fejlesztési és tesztelési folyamatok megkezdésével a munkaórák eloszlását a csapat kiegyenlítettnek, megfelelőnek találta.

5.3. Napló

2009.03.13. 07:00	Tóth-Máté, idő: 02:00
Skeleton vázának írása (6.1)	
2009.03.13. 16:00	Farkas, idő: 01:00
Skeleton implementálása (6.1)	
2009.03.13. 20:00	Farkas, idő: 05:00
Skeleton implementálása (6.1)	
2009.03.14. 18:00	Tóth-Máté, idő: 01:00
Skeleton implementálása (6.1)	
2009.03.14. 16:00	Farkas, idő: 02:30
Skeleton implementálása (6.1)	
2009.03.14. 19:00	Tóth-Máté, idő: 01:00
Skeleton finomhangolása, apró módosítások (6.1)	
2009.03.14. 19:30	Farkas, idő: 02:00
Skeleton implementálása (6.1)	
2009.03.15. 12:00	Siklósi, idő: 01:30
Osztályok és függvények kommentelése Javadoc formátumban	
2009.03.15. 12:00	Tóth-Máté, idő: 01:30
Skeleton egyszerűsítése	
2009.03.15. 16:00	Siklósi, idő: 01:00
Javadoc dokumentáció exportálása, batch-fájlok létrehozása a futtatás megkönnyítésére	
2009.03.15. 18:00	Siklósi, idő: 01:30
Osztályok és függvények kommentelése Javadoc formátumban	
2009.03.16. 19:00	Tóth-Máté, Siklósi, idő: 00:45
Javadoc kommentek javítása, kiegészítése, dokumentáció újraexportálása	
2009.03.16. 19:30	Zsolnai, idő: 00:45
Értékelés, fordítási és használati útmutatók megírása (6.1, 6.2)	
2009.03.17. 15:30	Farkas, Siklósi, idő: 01:00
4-es tesztpálya problémáinak elemzése (6.1)	
2009.03.17. 17:00	Siklósi, idő: 01:00
Skeleton implementálása, 4-es tesztpálya megírása (6.1)	

2009.03.17. 19:00	Siklósi, idő: 01:00
Logger osztály metódusainak átalakítása, egyéb javítások	
2009.03.17. 22:00	Tóth-Máté, idő: 00:30
Skeleton javítása	
2009.03.17. 21:00	Zsolnai, idő: 02:00
Fájllista elkészítése, dokumentáció javítások (6.1)	
2009.03.18. 23:00	Tóth-Máté, idő: 02:00
Skeleton véglegesítése, dokumentáció formázása, exportálás	

6. Prototípus koncepciója

6.1. Prototípus interface definíciója

A program prototípusa egy konzolos megjelenítői felületen keresztül lesz használható, melyben a fejlesztőcsapat által implementált parancsok alkalmazhatók. A fejlesztési folyamat ezen szakasza képet ad a program és az azt leíró rendszer belső működéséről, de a skeletonnal ellentétben már nem az egyes függvényhívások egymás utáni lefutását teszi ellenőrizhetővé, hanem a konkrét működést mutatja be. A prototípus fontos tesztelési célokat lát el, képes a csapat által előre meghatározott tesztesetek bemutatására. Ezek előre felépített pályákon automatizált parancssorozatok lefuttatásával kaphatók meg, és ezen tesztek a korábban deklarált elvárásokat kell, hogy igazolják. Mindezen elvárások a későbbi részletes tervek szekció alatt kerülnek ismertetésre.

A tesztekkel kapcsolatos alapelv, hogy azok átláthatók, megismerhetők és reprodukálhatók legyenek. A csapat ezen elveket szem előtt tartva állítja elő a prototípust, valamint annak tesztfelületét.

6.2. Összes részletes use-case

6.2.1. Játékoshoz kapcsolódó use case-ek

- Játék indítása
- Pályaváltás
- Karakter mozgatása

6.2.2. Órához kapcsolódó use case-ek

- Idő léptetése

6.2.3. Karakterhez kapcsolódó use case-ek

- Karakter találkozásai:
  - Empty: üres mezőre lép

  - Dirt: földre lép, kiássa
  - Diamond: gyémántra lép, felveszi
  - Exit: kijáratra lép
  - Boulder: sziklát tol el
  - Explosive: robbanószer hatósugarába lép
  - Monster: szörnyre lép, meghal
- Karakter robbanószert tesz le
- Karakter meghal

6.2.4. Sziklához kapcsolódó use case-ek

- Szikla találkozásai:
  - Player: karakterre zuhan, megöli azt
  - Empty: üres mezőre kerül
  - Boulder: másik sziklára kerül, azon elgurul
  - Monster: szörnyre zuhan, megöli azt
- Szikla zuhan
- Szikla elgurul

6.2.5. Robbanószerhez kapcsolódó use case-ek

- Robbanószer találkozásai:
  - Granite: gránitra "lépve" felrobbantja azt
  - Player: a karakter mellett robbanva megöli azt
  - Monster: a szörny mellett robbanva megöli azt
- Robbanószer felrobban
- Robbanószer eltűnik (a robbanás után)

6.2.6. Szörnyhöz kapcsolódó use case-ek

- Szörny létrejön (szaporodással)
- Szörny találkozásai:
  - Player: karakterre lép, megöli azt
  - Empty: üres mezőre lép
- Szörny szaporodik
- Szörny gyémánttá alakul
- Szörny meghal

6.2.7. Földhöz kapcsolódó use case-ek

- Föld eltűnik (ha a karakter rálép)

6.2.8. Gyémánthoz kapcsolódó use case-ek

- Gyémánt létrejön (szörny átalakulásából)
- Gyémánt zuhan
- Gyémánt eltűnik

6.2.9. Kijárathoz kapcsolódó use case-ek

- Kijárat megnyílik

Megjegyzés: Azok a találkozásokra vonatkozó use case-ek nem kerültek felsorolásra, amelyek lefutásakor nem történik lényegi interakció a két elem között. Az egyes térelemekhez kapcsolódó léptetési és ütköztetési use case-eket továbbra is a Timer actorhoz kapcsoljuk, azaz az egyes térelemek nem léptek elő külön actorrá, csupán a használati esetek kerültek az alapján rendszerezésre, hogy pontosan mely elemhez tartoznak.

6.3. Tesztelési terv, tesztelő nyelv definiálása

6.3.1. A tesztelés menete

A prototípus fordítása, futtatása a skeletonnal megegyező módon történik. A prototípus képes a parancsokat a standard bemenetről olvasni, illetve a megfelelő parancs hívásával ezt meg tudja tenni fájlból is. Minden újsor karakter (`\n`) az adott parancs végét jelöli. A kimenet alapértelmezetten a standard kimenetre kerül, azonban választható olyan mód is, hogy ez fájlba kerüljön.

6.3.2. A teszteléshez használható parancsok

`loadCommands` *bemeneti_fajl.txt*: A megadott *bemeneti_fajl.txt*-ből beolvassa és végrehajtja a parancsokat.

`setOutput` *mod*: Beállítja a kimenetet (*mod*=0 esetén a képernyőre, *mod*=1 esetén a kimeneti fájlba, *mod*=2 esetén mindkettőre).

`loadMap` *palya_fajl.txt*: Betölti a megadott fájlból a tesztelendő pálya felépítését.

`setPlayerMove` *karakter_szama karakter_iranya*: A *karakter_szama* sorszámú karaktert a következő `tick` híváskor *karakter_iranya* felé próbálja elmozdítani. Ez a pálya felépítéséből adódóan egy 1-6 közötti egész szám.

`allDiamonds`: A pálya teljesítéséhez szükséges számú gyémántot állít be a karaktereknek.

`plantExplosive` *karakter_szama*: A megadott számú karakter a következő lépésekor egy robbanószert helyez el az általa elfoglalt mezőn.

`explode` *robbanoszer_szama*: A következő `tick` hívásakor felrobbantja a *robbanoszer_szama* sorszámú robbanószert.

`tick` *lepesek_szama*: *lepesek_szama* számú mozgatást hajt végre az összes pályán lévő mozgó elemen. Ha nincs megadva paraméter, akkor egyet.

`showMap`: Kiírja a pálya aktuális állapotát (bemenetként is értelmezhető formátumban).

`exit`: Befejezi a tesztelést.

6.3.3. A kimeneten megjelenő hibaüzenetek

`x Error: FileNotFound`: Nem olvasható a megadott bemeneti fájl.

`x Error: WriteError`: Nem írható a megadott kimeneti fájl.

`x Error: ReadError`: Nem sikerült a következő parancs beolvasása.

`x Error: CommandNotFound`: Nem létező parancsot érkezett.

`x Error: ParameterCountMismatch`: Nem megfelelő számú paraméter érkezett.

`x Error: WrongParameter`: Nem megfelelő paraméter érkezett.

`x Error: NoMap`: Nincs betöltve pálya a `loadMap` segítségével.

`x Error: WrongMap`: Nem megfelelően formázott vagy nem létező pályát próbált betölteni.

`x Error: NoPlayer`: Nem létező karakter mozgását próbálta meg irányítani.

`x Error: NoExplosive`: Nem létező robbanószert próbált felrobbantani.

`x Error: OutOfExplosives`: Kifogyott a robbanószer.

`x Error: WrongMove`: Nem létező irányba próbálta meg mozdítani a karaktert.

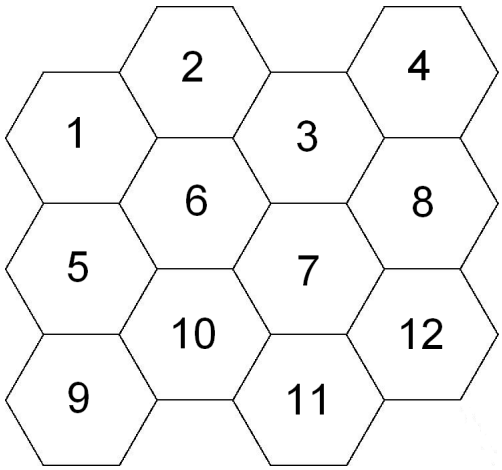
6.3.4. A kimeneten megjelenő egyéb események

A `showMap` parancsra kiíródik a pálya aktuális állapota a kimenetre. Emellett minden pályán található elem kiírja az alábbi formában, ha létrejön, elmozdul illetve eltűnik a pályáról:
o ElemTípusa ElemSorszáma Esemény. (pl. `o Monster 2 has died.`)

Minden pályabetöltés utána képernyőn megjelenik a pálya képe. Emelett a futtatás végén a futtatás ideje is millisec-ben.

6.3.5. A pálya formátuma

A játékteret felépítő elemeket egy szöveges fájlban tároljuk, melyből a kész játék egyes pályái egyértelműen felépíthetők. Minden karakter a játéktér egy mezőjét reprezentálja, annak betűjele pedig meghatározza, hogy ott pontosan melyik térelem található. Az egyes betűjelekhez tartozó térelemek listája ezen dokumentumrész végén, az ábra alatt tekinthető meg. Ez a tárolási módszer lehetővé teszi a hatszögletű pályaelemekkel való munkát.



29. ábra. A pálya felépítése

A térképeket tartalmazó szöveges állományban az ábrán ismertetett szisztéma szerint követik egymást az elemek. A módszer előnye, hogy a fájlban tárolt egyes pályák hasonlítanak a vizuális reprezentációhoz, könnyen olvashatók, megérthetők, további pályák készítéséhez nem szükséges komoly gyakorlat, vagy a program mögött meghúzódó rendszer részletes ismerete.

Jelmagyarázat:
(A könnyebb érthetőség kedvéért a megfelelő karaktereket szögletes zárójellel határoltuk)

[o] - szikla
[D] - gyémánt
[.] - föld
[X] - kijárat
[e] - robbanóanyag
[G] - gránit
[m] - szörny (gyémánttá alakuló: M)
[l] - fal
[P, Q] - játékos karakter
[ ] - üres mező

6.4. Tesztelést támogató segéd- és fordító programok specifikálása

A tesztelés folyamán a tesztpálya aktuális állapota a tesztelőnyelv specifikálása fejezetben ismertetett parancssal tetszőleges időpontban kiíratható, menthető. Mivel a várt kimenet ismert, előre meghatározott formát kell öltsön, a tesztelés további automatizálása érdekében a csapat által használt verziókezelőrendszer egy olyan alrésze kerül alkalmazásra, mely szöveges fájlok tartalmát képes összehasonlítani, azok különbségeiről ad jelzést. A program által generált kimenet feldolgozásával egyértelműen megállapítható, hogy a teszt a várt eredményeket produkálta-e.

6.5. Változtatások a követelmények módosulása miatt

- A játék területe méhkas módjára hatszögekből épül fel. A mozgás során a mozgó elemek egyik hatszögből a másikba lépnek át.
- A játékban két indián, Júz Kéz és testvére, Júz Láb vesz részt. Egymáson nem tudnak átmenni, egymásban nem okoznak kárt. A barlang kijárata akkor nyílik meg, ha az összes felvehető gyémánt fel lett véve.

Az analízis modell gondos megtervezése meghozta gyümölcsét, ugyanis a követelmény változtatásai gond nélkül adaptálhatók a meglévő modellbe. A játékmezők tárolása egyetlen vektorban történik, a mezők a szomszédaikat ismerik. Eddig 4 irányban helyezkedtek el a közvetlen szomszédok, ezen irányok száma a specifikáció változtatásának eredményeképp 6-ra nőtt. Az adatszerkezetek megfelelő megválasztásával lehetővé vált több karakter egyidejű kezelése is. Ezen adatszerkezet bővítésével tetszőleges számú új karakter helyezhető a játéktérre.

6.6. Napló

2009.03.23. 15:00	Tóth-Máté, Zsolnai, Siklósi, idő: 01:30
Prototípus interface-ének megtervezése	
2009.03.24. 09:30	Siklósi, idő: 01:00
Követelmény-változások dokumentálása, összes részletes use-case (7.2)	
2009.03.24. 10:00	Tóth-Máté, idő: 01:00
Prototípusban használható parancsok megírása (7.3)	
2009.03.24. 15:00	String Quartet, idő: 01:30
Prototípus tervének véglegesítése	
2009.03.24. 20:30	Zsolnai, idő: 00:30
Tesztelést támogató segéd- és fordító programok specifikálása (7.4)	
2009.03.24. 21:00	Zsolnai, idő: 00:30
Prototípus interface definiciója, tesztelési terv, tesztelő nyelv definiálása (7.1, 7.3)	
2009.03.24. 22:00	Tóth-Máté, idő: 00:30
Leadandó dokumentum formázása	

7. Részletes tervek

7.1. Objektumok és metódusok tervei

7.1.1. Boulder

`boolean isFalling()`: A metódus megmondja, hogy a szikla zuhan-e éppen. Visszatér a `falling` attribútum értékével.

`boolean isRolling()`: A metódus megmondja, hogy a szikla gurul-e éppen. Visszatér a `rolling` attribútum értékével.

`Field move(Field field)`: Elmozdítja a sziklát a paraméterként kapott mezőre, majd visszatér azzal a mezővel, amelyen korábban állt. Ha a sziklát egy `Player` tolta el, értesíti őt (a `movePlayer`-ben tárolt `Player`-t) arról, hogy el tudott mozdulni, ezáltal a karakter ráléphet a szikla korábbi mezőjére.

`void onTick()`: A szikla megpróbál zuhanni, lekérdezi az alatta levő mezőt és az azon álló `Element`-nek meghívja a `steppedOn` metódusát, paraméterként átadva önmagát. Ezután még beállítja a szikla `falling` tulajdonságát aszerint, hogy sikerült-e lefelé elmozdulni.

`void setRolling()`: Igazra állítja a szikla `rolling` tulajdonságát.

`void steppedOn(Boulder element)`: A sziklára egy másik szikla érkezik. Első lépésként a metódus beállítja a fölülről érkező szikla `rolling` tulajdonságát igazra, majd megpróbálja az érkező sziklát jobbra elmozdítani. Ha ez nem sikerült, megpróbálja balra elmozdítani, különben vége.

`void steppedOn(Player element)`: A metódus először eltárolja a sziklára lépő játékost a `movePlayer` tagváltozóban. Ezután rálépteti a sziklát a játékos érkezésével ellenkező irányban álló mezőre.

7.1.2. Cave

`Field createField()`: A metódus létrehoz egy új mezőt a barlangba és visszatér annak referenciájával.

`Game getGame()`: A metódus visszaadja azt a `Game` objektumot, amelyhez a barlang tartozik.

`ArrayList getMap()`: Visszaadja egy listában a barlanghoz tartozó mezőket.

`void connectNewField(int oldId, int direction)`: Az utoljára létrehozott mezőt összeköti egy létező mezővel a megadott irányban.

`void loadMap(String file)`: A megadott térkép alapján létrehozza és összeköti a mezőket, illetve ráteszi a kért elemet.

`String showMap()`: Visszatér a pályát és a rajta lévő objektumokat reprezentáló Stringgel.

7.1.3. Diamond

`void onTick()`: A gyémánt megpróbál zuhanni, lekérdezi az alatta levő mezőt és az azon álló `Element`-nek meghívja a `steppedOn` metódusát, paraméterként átadva önmagát.

`void steppedOn(Player element)`: A metódus meghívja a gyémánt `die` metódusát, növeli a rálépő `Player` felvett gyémántjainak számát, majd rálépteti a karaktert a mezőre.

7.1.4. Dirt

`void steppedOn(Player element)`: A metódus meghívja a föld objektum `die` metódusát, majd rálépteti a karaktert a mezőre.

7.1.5. Element

`void die()`: A metódus leveszi az objektumot a mezőről a mező `clear` metódusának meghívásával.

`Field move(Field field)`: A metódus ráteszi az objektumot a paraméterként kapott mezőre és visszatér az objektum által addig elfoglalt mezővel.

`Field getField()`: Visszatér az objektum által elfoglalt mező referenciájával.

`void setField(Field field)`: A metódus beállítja az objektum `field` tagváltozóját a paraméterként kapott mezőre és ráteszi önmagát a mező `setElement` metódusának meghívásával.

`char getRepresentation()`: Visszatér az osztályt reprezentáló karakterrel.

`void setRepresentation(char representation)`: Beállítja az osztályt reprezentáló karaktert.

`int getId()`: Visszatér az objektum azonosítójával.

`void setId(int id)`: Beállítja az objektum azonosítóját.

`int getLastDirection()`: Az utolsó elmozdulás irányát adja vissza.

`void setLastDirection(int id)`: Az utolsó elmozdulás irányát állítja be.

`void steppedOn(Element element)`: A metódus ténylegesen nem csinál semmit, az `Element`-ből leszármaztatott osztályokban felül lehet definiálni.

`void steppedOn(Boulder element)`: A metódus meghívja az `Element` paraméterű `steppedOn` metódust.

`void steppedOn(Explosive element)`: A metódus meghívja az `Element` paraméterű `steppedOn` metódust.

`void steppedOn(Monster element)`: A metódus meghívja az `Element` paraméterű `steppedOn` metódust.

`void steppedOn(Player element)`: A metódus meghívja az `Element` paraméterű `steppedOn` metódust.

7.1.6. Empty

`void steppedOn(Element element)`: A metódus bármilyen vizsgálat nélkül rálépteti az üres objektumhoz tartozó mezőre a paraméterben kapott `Element`-et.

7.1.7. Exit

`void steppedOn(Player element)`: A metódus a pályaváltást valósítja meg.

7.1.8. Explosive

`void onTick()`: A metódusban minden híváskor eggyel csökken a számláló összes szomszédos mezőjén álló objektum `steppedOn` metódusát, majd a `die` hívásával megszünteti a robbanószer objektumot.

`void explode()`: Beállítja a számlálót nullára.

7.1.9. Field

`void clear()`: Megszünteti a mezőn álló objektumra való hivatkozást, majd létrehoz a helyére egy üres (`Empty`) objektumot, vagy egy robbanószert (`Explosive`) a `createExplosive` tagváltozó értékétől függően.

`Cave getCave()`: Visszatér annak a barlangnak a referenciájával, amelyhez a mező tartozik.

`void setCave()`: Beállítja, hogy a mező melyik barlanghoz tartozzon.

`Element getElement()`: Visszaadja a mezőn elhelyezkedő objektum referenciáját.

`void setElement(Element element)`: Ráteszi a mezőre a paraméterként kapott objektumot.

`Field getNeighbor(int direction)`: A mező adott irányban elhelyezkedő szomszédját adja vissza.

`void setNeighbor(Field field, int direction)`: A mező adott irányú szomszédjaként beállítja a paraméterként megadott mezőt.

`void plantExplosiveOnClear()`: Igazra állítja a `createExplosive` tagváltozó értékét.

7.1.10. Game

`void startGame()`: A metódus elindítja a játékot, létrehozza a barlangot, benne a `Field`-eket, rajtuk a betöltött pályának megfelelő objektumokat.

`tick`: A játékban szereplő objektumokat típusonként külön-külön vektorban tároljuk. Ez a metódus mindegyik ilyen vektoron végighaladva meghívja az elemeik `onTick` metódusát.

`void addBoulder(Boulder element)`: Hozzáadja a paraméterként kapott sziklát a `boulders` vektorhoz.

`void addDiamond(Diamond element)`: Hozzáadja a paraméterként kapott gyémántot a `diamonds` vektorhoz.

`void addExplosive(Explosive element)`: Hozzáadja a paraméterként kapott robbanószert az `explosives` vektorhoz.

`void addMonster(Monster element)`: Hozzáadja a paraméterként kapott szörnyet a `monsters` vektorhoz.

`void addPlayer(Player element)`: Hozzáadja a paraméterként kapott játékos karaktert a `players` vektorhoz.

`void addStaticElement(Element element)`: Hozzáadja a paraméterként kapott statikus elemet (`Dirt`, `Empty`, `Exit`, `Granite`, `Wall`) a `staticElements` vektorhoz.

`void removeElement(Element element)`: Eltávolítja valamely vektorból a paraméterként kapott objektumot.

`boolean hasAllDiamonds()`: Visszaadja, hogy üres-e a gyémántokat tároló vektor, azaz felvették-e már az összes gyémántot.

`Player getPlayer(int i)`: Visszaadja az i. `Player`-t.

`Explosive getExplosive(int i)`: Visszaadja az i. `Explosive`-ot.

`int getDiamondCount()`: Visszaadja a még fel nem vett gyémántok számát.

7.1.11. Granite

`void steppedOn(Explosive element)`: A metódus meghívja a gránit objektum `die` metódusát.

7.1.12. Monster

`Monster(boolean follower, boolean selfCloner, boolean transformer)`: A Monster osztály konstruktora, mely beállítja az objektum tulajdonságait a megadott paraméterek alapján.

`void onTick()`: A szörny meghívja egy véletlenszerű szomszédja által tartalmazott objektum `steppedOn(Monster element)` metódusát.

`boolean isFollower()`: A metódus visszaadja az objektum `follower` tulajdonságát.

`boolean isSelfCloner()`: A metódus visszaadja az objektum `selfCloner` tulajdonságát.

`boolean isTransformer()`: A metódus visszaadja az objektum `transformer` tulajdonságát.

`void die()`: A metódus meghívja az ősosztály `die()` metódusát, majd megvizsgálja a `transformer` változót, és amennyiben az értéke `true`, egy gyémántot hoz létre a `getField()` visszatérési értékének megfelelő helyen.

`void steppedOn(Boulder element)`: A metódus meghívja a `die()` metódust, majd meghívja az `element move` metódusát a `getField()` visszatérési értékével.

`void steppedOn(Explosive element)`: A metódus meghívja a `die()` metódust.

`void steppedOn(Player element)`: A metódus meghívja az `element` `die()` metódusát.

`void clone()`: A metódus az objektum egy véletlenszerűen kiválasztott szomszédjára próbál létrehozni egy új `Monster` típusú objektumot.

7.1.13. Player

`void onTick()`: A karakter megpróbál elmozdulni a játékos által definiált irányba, és meghívja a megfelelő mező `steppedOn(Player p)` metódusát önmagát átadva paraméterként. Amennyiben nincs definiált irány, nem történik semmi.

`int incDiamond()`: A karakter által birtokolt gyémántok számát növeli eggyel, majd visszaadja a már begyűjtött gyémántok számát (`diamondCounter`).

`int countDiamonds()`: A karakter által már begyűjtött gyémántok számát adja vissza (`diamondCounter`).

`int decExplosive()`: A karakternél lévő robbanószerek számát csökkenti eggyel, majd vissza is adja ezt az értéket (`explosiveCounter`).

`int countExplosives()`: A karakternél lévő robbanószerek számát adja vissza (`explosiveCounter`).

`void steppedOn(Boulder element)`: A metódus megvizsgálja az `element` `isFalling()` függvényének visszatérési értékét. Amennyiben a visszatérési érték `true`, meghívódik a `die()` metódus, majd az `element` `move()` metódusa, paraméterként adva a karakter `getField()` metódusának visszatérési értékét. Amennyiben az `isFalling` értéke `false`, nem történik semmi.

`void steppedOn(Diamond element)`: A metódus meghívja az `incDiamond()` metódust.

`void steppedOn(Explosive element)`: A metódus meghívja a `die()` metódust.

`void steppedOn(Monster element)`: A metódus meghívja a `die()`-t, majd meghívja az `element move` metódusát, paraméterként adva a `getField()` visszatérési értékét.

`void plantExplosive()`: Csökkenti a játékosnál lévő robbanószerek számát, majd megjelöli a mezőt, hogy robbanószert kell letenni, amint a karakter elmozdul.

7.1.14. Timer

`void tick()`: A játékon belüli órajelet szimbolizálja.

7.2. A tesztek részletes tervei, leírásuk a teszt nyelvén

A teszteseteket ellenőrző pályák a következő szempontok szerint kerültek osztályozásra:

Fájlnév - Az adott térképet tartalmazó állomány neve.

Térkép - A megadott jelmagyarázat szerint megadott pálya pontos felépítése, szögletes zárójelekkel határolva a könnyebb átláthatóság érdekében.

Bemenet - Az elvégzendő parancsokat tartalmazó bemeneti fájl tartalma (az állomány neve mindig megegyezik a tesztelt pálya fájlnevével, az „_input” szó hozzáadásában különbözik).

Elvárt kimenet - A teszt eredményeképpen előállított és elvárt eredmények.

7.2.1. 1. tesztpálya

Az első pálya az egyes térelemek példányosításának sikerességét teszteli.
Fájlnév: init.bdm
Térkép:
`[oD.XeMlPQG ]`

Bemenet	Elvárt kimenet

`loadMap init.bdm`
`showMap`
`exit`
	`o Boulder 1 is created.`
	`o Diamond 1 is created.`
	`o Dirt 1 is created.`
	`o Exit 1 is created.`
	`o Explosive 1 is created.`
	`o Monster 1 is created.`
	`o Wall 1 is created.`
	`o Player 1 is created.`
	`o Player 2 is created.`
	`o Granite 1 is created.`
	`[oD.XeMlPQG ]`

7.2.2. 2. tesztpálya

A játékos karakter tevékenységeire vonatkozó esetek tesztelését teszi lehetővé. A karakter a kezdőpontból balra elmozdulva eltolja a sziklát, majd ezt követően néhányszor jobbra haladva kiássa a földterületet, felveszi a gyémántot, majd sikeresen befejezi a pályát.
Fájlnév: player.bdm
Térkép:

```
[ oP.DX]
```

Bemenet	Elvárt kimenet
loadMap player.bdm setPlayerMove 1 4 tick setPlayerMove 1 1 tick setPlayerMove 1 0 tick setPlayerMove 1 1 tick setPlayerMove 1 0 tick showMap exit	o Boulder 1 is created. o Player 1 is created. o Dirt 1 is created. o Diamond 1 is created. o Exit 1 is created « Tick 1 » o Boulder 1 has moved to the [3] direction. o Player 1 has moved to the [4] direction. « Tick 2 » o Player 1 has moved to the [1] direction. « Tick 3 » o Dirt 1 has died. o Player 1 has moved to the [0] direction. « Tick 4 » o Diamond 1 has died. o Player 1 has moved to the [1] direction. « Tick 5 » o Exit 1 permits Player 1 to pass. [PX]

7.2.3. 3. tesztpálya

A sziklákra vonatkozó teszteseteket tartalmazza. A játékos karakter kiássa a fölötte levő, sziklát alátámasztó föld mezőt, majd a szikla útjából elállva, zuhanásnak indítja azt. Ezután a szikla egy másik sziklán is elgurul, végül pedig egy szörnyre érkezik és megöli azt.
Fájlnév: boulder.bdm
Térkép:

```
[o ]
[. ]
[P ]
[o ]
[lm]
[ G]
```

Bemenet	Elvárt kimenet
loadMap boulder.bdm setPlayerMove 1 5 tick setPlayermove 1 0 tick 5 showMap exit	o Boulder 1 is created. o Dirt 1 is created. o Player 1 is created. o Boulder 2 is created. o Wall 1 is created. o Monster 1 is created. o Granite 1 is created. « Tick 1 » o Dirt 1 has died. o Player 1 has moved to the [5] direction. « Tick 2 » o Player 1 has moved to the [0] direction. « Tick 3 » o Boulder 1 has moved to the [2] direction.

	« Tick 4 » o Boulder 1 has moved to the [2] direction. « Tick 5 » o Boulder 1 has moved to the [1] direction. « Tick 6 » o Monster 1 has died. o Boulder 1 has moved to the [2] direction. [ ] [P] [ ] [o] [lo] [G]

7.2.4. 4. tesztpálya

Egy bomba robbanását teszteli, amely elpusztítja a körülötte álló térelemeket.
Fájlnév: explosive.bdm
Térkép:

```
[GMP]
[ e ]
[   ]
```

Bemenet	Elvárt kimenet
loadMap explosive.bdm explode 1 tick 2 showMap exit	o Granite 1 is created. o Monster 1 is created. o Player 1 is created. o Explosive 1 is created. « Tick 1 » o Player 1 has died. o Granite 1 has died. o Monster 1 has died. o Diamond 1 is created. o Explosive 1 has died. « Tick 2 » o Diamond 1 has moved to the [2] direction. [ ] [D] [ ]

7.2.5. 5. tesztpálya

Terhelésteszt. A pálya egy 10x10-es méretű, csak szörnyeket tartalmazó terület, mellyel megvizsgálható, hogy a térelemek egymással való kommunikációja megfelelően gyorsan zajlik-e, így a további fejlesztés menete folyamán minden fordítás után leolvasható az aktuális kód teljesítőképessége.
Fájlnév: stress.bdm
Térkép: Ettől eltekintünk

Bemenet	**Elvárt kimenet**

`setOutput 1`
`loadMap stress.bdm`
`tick 5000`
`exit`
	Az eddigiektől eltérően egy konkrét időeredmény lesz a kimenet, ami alatt a megadott feladat lefutott.

7.3. A tesztelést támogató programok tervei

A tesztelés sikerességének megállapításához illetve az eltérések kimutatásához a DiffUtils[7] csomag `sdiff` és `cmp` binárisait fogjuk használni. A baloldali panelon a valós kimenet lesz látható a jobboldalin pedig az általunk meghatározott. Ha a `cmp` program nem jelez eltérést, akkor a tesztelés sikeresnek bizonyult.

[7] http://gnuwin32.sourceforge.net/packages/diffutils.htm

7.4. Napló

2009.03.30. 18:00	Farkas, Zsolnai, Tóth-Máté, idő: 00:30
Prototípus részletes terveinek megbeszélése	
2009.03.31. 15:00	Siklósi, idő: 02:00
Metódustervek elkészítése (8.1)	
2009.04.01. 12:30	Farkas, idő: 00:30
Metódustervek kiegészítése (8.1)	
2009.04.01. 12:30	Zsolnai, Tóth-Máté, idő: 01:30
A tesztek részletes megtervezése (8.2)	
2009.04.01. 21:00	Zsolnai, idő: 02:00
A tesztek részletes tervei, leírásuk a teszt nyelvén (8.2), kész dokumentáció lektorálása	
2009.04.02. 02:00	Tóth-Máté, idő: 02:30
Részletes tervek véglegesítése	

8. Prototípus beadása

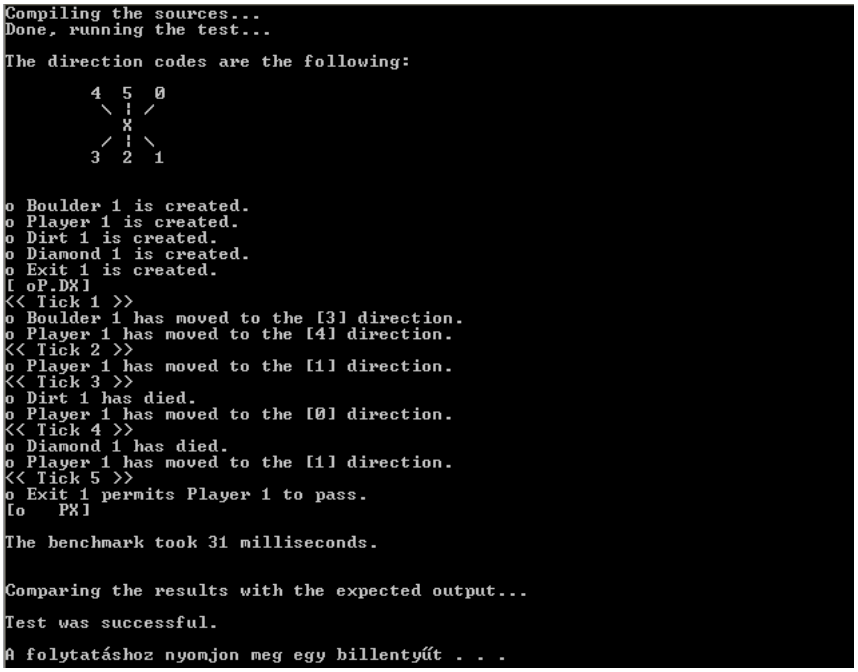
8.1. A prototípus

8.1.1. A prototípus fordítása és futtatása

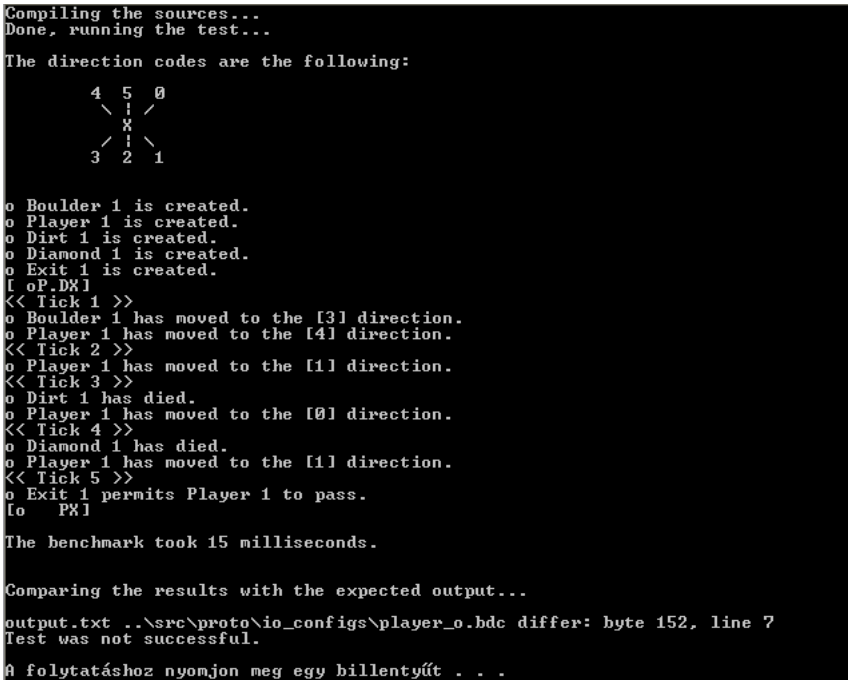
A prototípus lefordítása és futtatása automatizáltan történik, a fejlesztőcsapat által mellékelt indítófájlok segítségével. Ezek az állományok a könyvtárszerkezet /bin/ mappája alatt találhatók, és a célszámítógépen található Java fordító segítségével állítják elő a már futtatható, kész programkódot. A használat további egyszerűsítése érdekében minden tesztpályához külön indítófájl tartozik, melynek nevében szerepel a megadott tesztpálya sorszáma. Például: a proto_test_4.bat a 4. tesztpályát futtatja le.

A fent említett indítófájlok a teszt lefutását követően minden szükséges információt közölnek, ami szükséges lehet az adott teszt sikerességének megállapításához. A megadott tesztpálya konkrét kimenetének kiírásán kívül a program meghív egy külső alkalmazást is, amely összehasonlítja a kimenetet a fejlesztőcsapat által előre meghatározott elvárt kimenetet, és az ezzel való egyezést vagy esetleges különbségeket kiírja.

Megjegyzés: A szöveg jobb átláthatósága érdekében érdemes lehet a konzol méretének nagyobbra állítása. Ezenkívül szükséges feltétel, hogy a Java binárisainak elérési útja be legyen állítva a megfelelő környezeti változóban (pl. Microsoft Windows alatt a set PATH paranccsal).



30. ábra. Egy képernyőkép az egyik teszt sikeres futtatásáról



31. ábra. Egy képernyőkép az egyik teszt sikertelen futtatásáról

8.1.2. A prototípus fájljai

A kész prototípus a következő fájlokat tartalmazza, könyvtáranként való lebontásban: (A kiírt fájlméretek byte-ban értendők)

Az /src/proto könyvtár a programot alkotó forráskódokat és a tesztpályákat tartalmazza:

Fájlnév	**Méret**	**Létrehozás**	**Leírás**
Boulder.java	3037	2009.03.11. 16:47	A szikla elemet megvalósító osztály
Cave.java	6284	2009.03.11. 16:47	A játékteret megvalósító osztály
Diamond.java	684	2009.03.11. 16:47	A gyémánt elemet megvalósító osztály
Dirt.java	495	2009.03.11. 16:47	A föld elemet megvalósító osztály
Element.java	4023	2009.03.11. 16:47	A térelemek ősosztálya
Empty.java	398	2009.03.11. 16:47	Üres mező elemet megvalósító osztály
Exit.java	537	2009.03.11. 16:47	Kijárat elemet megvalósító osztály
Explosive.java	1006	2009.03.11. 16:47	Robbanóanyag elemet megvalósító osztály
Field.java	2426	2009.03.11. 16:47	Mező elemet megvalósító osztály
Game.java	5288	2009.03.11. 16:47	A játék elemeit összefogó osztály
Granite.java	342	2009.03.11. 16:47	Gránit elemet megvalósító osztály
Monster.java	2759	2009.03.11. 16:47	A szörny ellenfeleket megvalósító osztály
Player.java	2457	2009.03.11. 16:47	A játékos karaktert megvalósító osztály

Prototype.java	9072	2009.04.11. 11:49	A prototípus főosztálya
Timer.java	496	2009.03.11. 16:47	Az időzítésért felelős osztály
Wall.java	95	2009.03.11. 16:47	A fal elemet megvalósító osztály
io_configs/ boulder_i.bdc	80	2009.04.11. 16:42	A harmadik tesztpálya bemenete
io_configs/ boulder_o.bdc	572	2009.04.11. 16:42	A harmadik tesztpálya kimenete
io_configs/ explosive_i.bdc	51	2009.04.11. 16:42	A negyedik tesztpálya bemenete
io_configs/ explosive_o.bdc	298	2009.04.11. 16:42	A negyedik tesztpálya kimenete
io_configs/ init_i.bdc	29	2009.04.11. 16:42	Az első tesztpálya bemenete
io_configs/ init_o.bdc	245	2009.04.11. 16:42	Az első tesztpálya kimenete
io_configs/ player_i.bdc	146	2009.04.11. 16:42	A második tesztpálya bemenete
io_configs/ player_o.bdc	479	2009.04.11. 16:42	A második tesztpálya kimenete
io_configs/ stress.bdc	45	2009.04.11. 16:42	Az ötödik tesztpálya kimenete
maps/ boulder.bdm	17	2009.04.11. 12:54	A harmadik tesztpálya
maps/ explosive.bdm	13	2009.04.11. 16:23	A negyedik tesztpálya
maps/ init.bdm	11	2009.04.11. 16:23	Az első tesztpálya
maps/ player.bdm	6	2009.04.11. 16:23	A második tesztpálya

maps/ stress.bdm	118	2009.04.11. 16:23	Az ötödik tesztpálya

A /bin/ könyvtár tartalmazza a futtatható és lefordított bináris állományokat:

Fájlnév	**Méret**	**Létrehozás**	**Leírás**
cmp.exe	57344	2009.04.12. 21:30	A teszteredmények sikerességét ellenőrző alkalmazás
libiconv2.dll	898048	2009.04.12. 21:30	A cmp.exe működéséhez szükséges dll
libintl3.dll	92672	2009.04.12. 21:30	A cmp.exe működéséhez szükséges dll
proto_docgen.bat	130	2009.04.13. 00:27	A javadoc dokumentációt elkészítő batch fájl
proto_run.bat	68	2009.04.12. 14:49	A prototípus kézzel való tesztelését indító batch fájl
proto_test_1.bat	424	2009.04.12. 23:29	Az első tesztpálya indítófájlja
proto_test_2.bat	417	2009.04.12. 23:29	A második tesztpálya indítófájlja
proto_test_3.bat	430	2009.04.12. 23:29	A harmadik tesztpálya indítófájlja
proto_test_4.bat	434	2009.04.12. 23:29	A negyedik tesztpálya indítófájlja
proto_test_5.bat	182	2009.04.12. 23:29	Az ötödik, terheléstesztelő tesztpálya indítófájlja
showdirs.bat	153	2009.04.14. 15:30	A teszteléshez az irányok kódjait bemutató iránytűt kiíró segédprogram

8.1.3. A tesztek jegyzőkönyvei

A prototípus RC verziójának elkészültét követően a 2009.04.13-án 17:00-kor tartott pályatesztek eredményei a következők voltak:

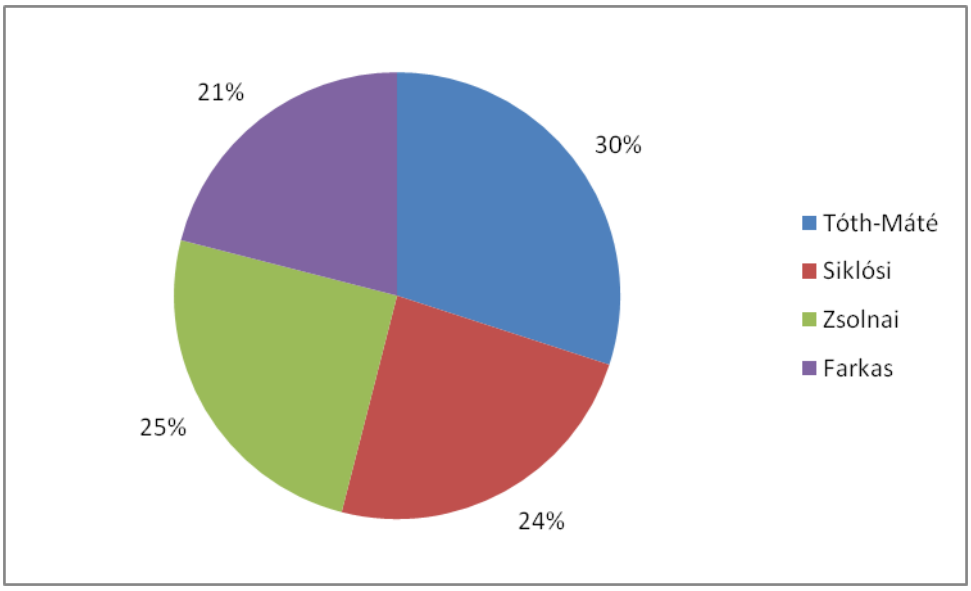
- Az 1. tesztpálya ellenőrzi az inicializálást, és a tesztek az elvárt eredményeknek megfelelően futnak.
- A 2. tesztpálya a játékos karakterre vonatkozó cselekvéseket vizsgálja. Sikeresen lefut.
- A 3. tesztpálya a sziklákkal való műveletek sikerességét ellenőrzi. Hibamentesen fut.
- A 4. tesztpálya a bomba felrobbanását és a körülötte szereplő térelemek elpusztulását vizsgálja. A teszt sikeresen zárult.
- Az 5. tesztpálya ellenőrizhetővé teszi, hogy a prototípus belső működése kielégítő sebességgel történik-e. A tesztelés eredményei alapján a prototípus sebessége kielégíti a specifikációban deklarált kritériumokat.

A lefuttatott tesztek pontosan definiáltak, eredményeik, sikerességük tetszőleges időpontban újra ellenőrizhetők. A csapat a prototípus tesztelését sikeresen zárta, az pontosan az ismertetett elvárásoknak megfelelően működött.

8.2. Értékelés

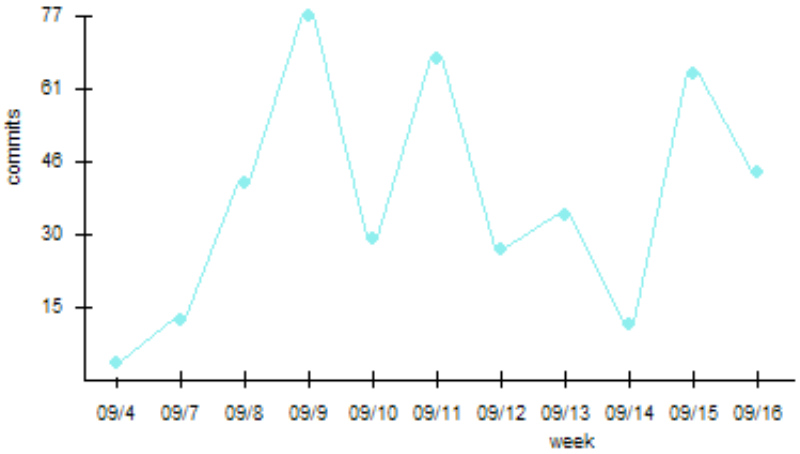
A prototípus implementálásához tartozó munkafolyamatok számos új tapaszalatot tartogattak a csapat számára. A konkrét implementáció egyes alfeladatai jól átgondolt rendszer szerint kerültek szétosztásra a tagok között, így lehetőség nyílt hatékonyan, individuálisan dolgozniuk az egyes feladatok megoldásán. A hangsúlyosabb, komolyabb megfontolásokat igénylő programrészek átbeszélése az interneten keresztül, hangátvitellel történt, azonban a korábban felállított modell és a tennivalókat részletesen meghatározó dokumentáció kapcsán ilyenre csupán két alkalommal volt szükség. A projekt többi részének munkafolyamatai konkurrens módon, párhuzamosan futottak, az egyes modulokon a csapattagok egyszerre, egymástól függetlenül voltak képesek dolgozni. Az egyes objektumok interfészei megfelelő részletességgel kerültek specifikálásra, így lehetőség nyílt a fejlesztés menetének párhuzamosítására, aminek eredményeképpen igen rövid idő alatt, külön kiemelhető probléma nélkül összeállt a kész produktum. A tagok között tapasztalható volt, hogy a hétről-hétre történő összedolgozás során az egyes munkafolyamatok definit formát öltöttek, megfelelő protokollok, menetrendek alakultak ki az egyes feladattípusok megoldására - ez pedig tovább növelte a projektbe fektetett idő és az azzal elvégzett munka arányát.

A csapatban végzett munkaórák eloszlása:
Tóth-Máté: 72:25h
Zsolnai: 60:15h
Siklósi: 59:45h
Farkas: 52:30h
Teljes munkaidő: 244:55h



32. ábra. Munkaidők megoszlása százalékosan

Az alábbi commit statisztikában látszik, hogy a csapat a legkomolyabb munkafolyamatokat együtt, egy időben dolgozva végezte el.



33. ábra. Commit statisztika

8.3. Napló

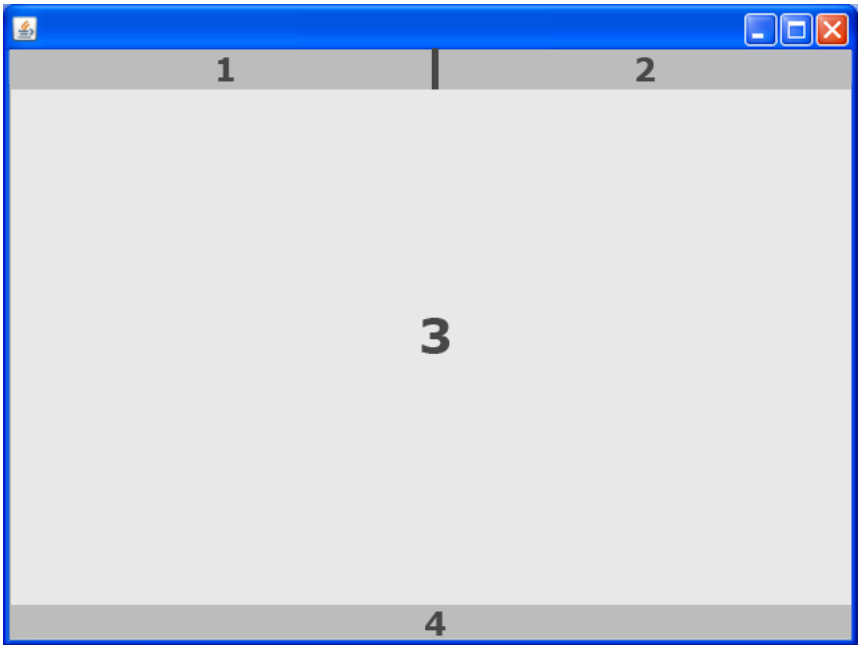
2009.04.11. 10:00	Siklósi, idő: 01:30
Prototípus felhasználói felületének megírása, parancsfeldolgozás	
2009.04.11. 15:00	String Quartet, idő: 01:00
Prototípus implementálási feladatainak megbeszélése	
2009.04.11. 16:15	Farkas, idő: 05:00
Prototípus implementálása	
2009.04.11. 17:00	Zsolnai, idő: 04:30
Prototípus implementálása	
2009.04.11. 16:00	Tóth-Máté, idő: 06:00
Pálya betöltés és kiírás, segítség a többieknek	
2009.04.12. 11:00	Farkas, idő: 01:30
Prototípus implementálása	
2009.04.12. 14:30	String Quartet, idő: 02:30
Prototípus implementálási feladatainak megbeszélése	
2009.04.12. 15:00	Siklósi, idő: 05:00
Prototípus implementálása, hibajavítások, tesztpályák leellenőrzése	
2009.04.12. 16:00	Zsolnai, idő: 02:00
Prototípus implementálása	
2009.04.13. 11:00	Farkas, idő: 02:00
Prototípus implementálása	
2009.04.13. 15:30	Siklósi, idő: 01:00
Hibajavítások, kommentek, egyéb módosítások	
2009.04.13. 16:00	Zsolnai, idő: 02:30
Útmutató, tesztelési jegyzökönyvek, értékelés megszerkesztése (10.1, 10.2, 10.3)	
2009.04.14. 10:00	Tóth-Máté, idő: 02:00
Prototípus kódjának rendbetétele	
2009.04.14. 15:00	String Quartet, idő: 01:30
Prototípus megbeszélése	
2009.04.14. 16:30	Zsolnai, idő: 01:00
Prototípus finomítása	

2009.04.14. 18:30	Tóth-Máté, idő: 02:00
Dokumentáció formázása, változások átvezetése	
2009.04.15. 11:30	Farkas, idő: 01:30
Prototípus implementálása	
2009.04.15. 12:30	Siklósi, idő: 01:00
Prototípus módosítások	
2009.04.15. 15:00	Tóth-Máté, idő: 00:30
Prototípus módosítások javítása	
2009.04.15. 20:00	Tóth-Máté, idő: 00:30
Leadandó dokumentum és feltöltendő csomag készítése	

9. Grafikus felület specifikációja

9.1. A menürendszer, a kezelői felület grafikus képe

A program grafikus verziójának felhasználói felületéről készült látványterv az alábbi képen látható:



34. ábra. A kezelői felület

A kezelői felület megjelenésében az egyszerűségre és a könnyű irányíthatóságra összpontosít. Ahogyan az alapkoncepciót bemutató látványterven is látható, a program főablaka 4 részre oszlik. Az egyes számoknak megfelelő kezelői elemek pedig a következők:

1. „New Game” nyomógomb:
   A játék indításáért felelős.

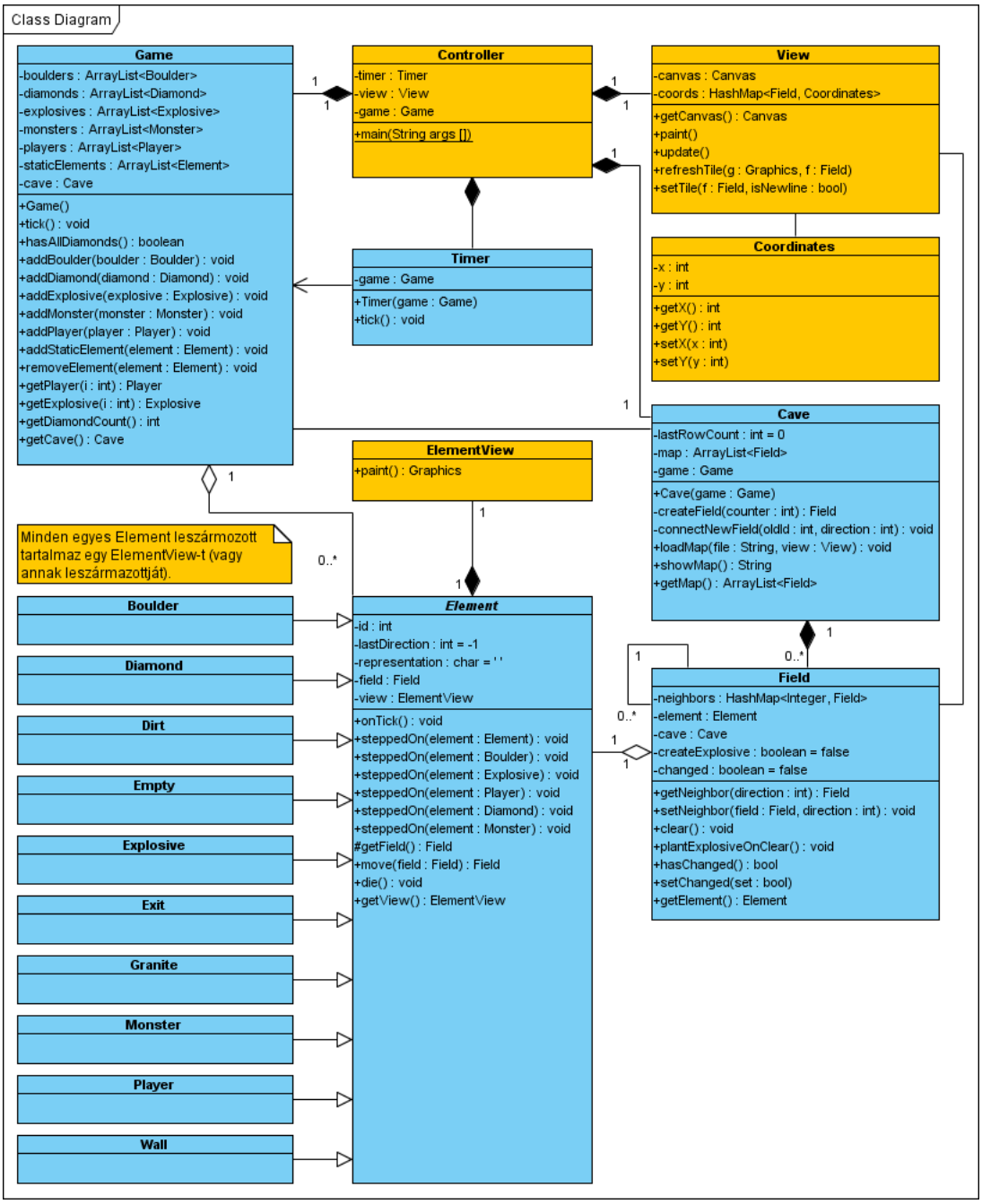
2. „Exit” nyomógomb:
   A játék befejezését illetve a programból való kilépést teszi lehetővé.

3. Játéktér:
   Az ablakterület, ahol maga az érdemi játék folyik. Itt jelennek meg a

hatszög alakú térelemek, és itt követhetők nyomon azok változásai is. Ez a legfontosabb terület.

4. A két játékos karakter, Júz Kéz és Júz Láb adatait tartalmazó állapotsáv:
   Itt követhető nyomon, hogy az egyes játékos karakterek mennyi gyémántot gyűjtöttek be, illetve hogy mennyi robbanóanyaggal rendelkeznek.

9.2. A felület működésének elve, a grafikus rendszer architektúrája



35. ábra. Struktúra diagram

A program kész grafikus verziója a klasszikus Model-View-Controller tervezési minta struktúrájára alapul. Ez a minta alkalmas arra, hogy a program belső adatait elválassza azok konkrét grafikus reprezentációjától, így a kapott program olyan részmodulokból áll, melyek interfészeken keresztül kommunikálnak, és a lehető legkevésbé függenek egymástól - ez az architektúra megfelel a specifikációban korábban deklarált irányelveknek, miszerint a kész programnak modulárisnak, annak részeinek pedig egymástól függetlennek kell lenniük.

Model (modell): Ez a réteg felelős a program belső állapotának, adatainak tárolásáért és kezeléséért. A modell tartalmazza továbbá az alkalmazásban elérhető szolgáltatások implementációját, és a belül megtörtént változásokról folyamatosan értesíti a nézetet.

View (nézet): Magát a modellt jeleníti meg a felhasználói felületen levő elemekkel, melyek lehetőséget adnak a felhasználónak a programmal való interakcióra.

Controller (vezérlő): Az alkalmazás futása során bekövetkező felhasználói műveleteket dolgozza fel, valamint egyszerre kommunikál a modell és nézet modulokkal, változásokra kényszerítve azokat.

9.3. A grafikus objektumok felsorolása, kapcsolatuk az alkalmazói rendszerrel

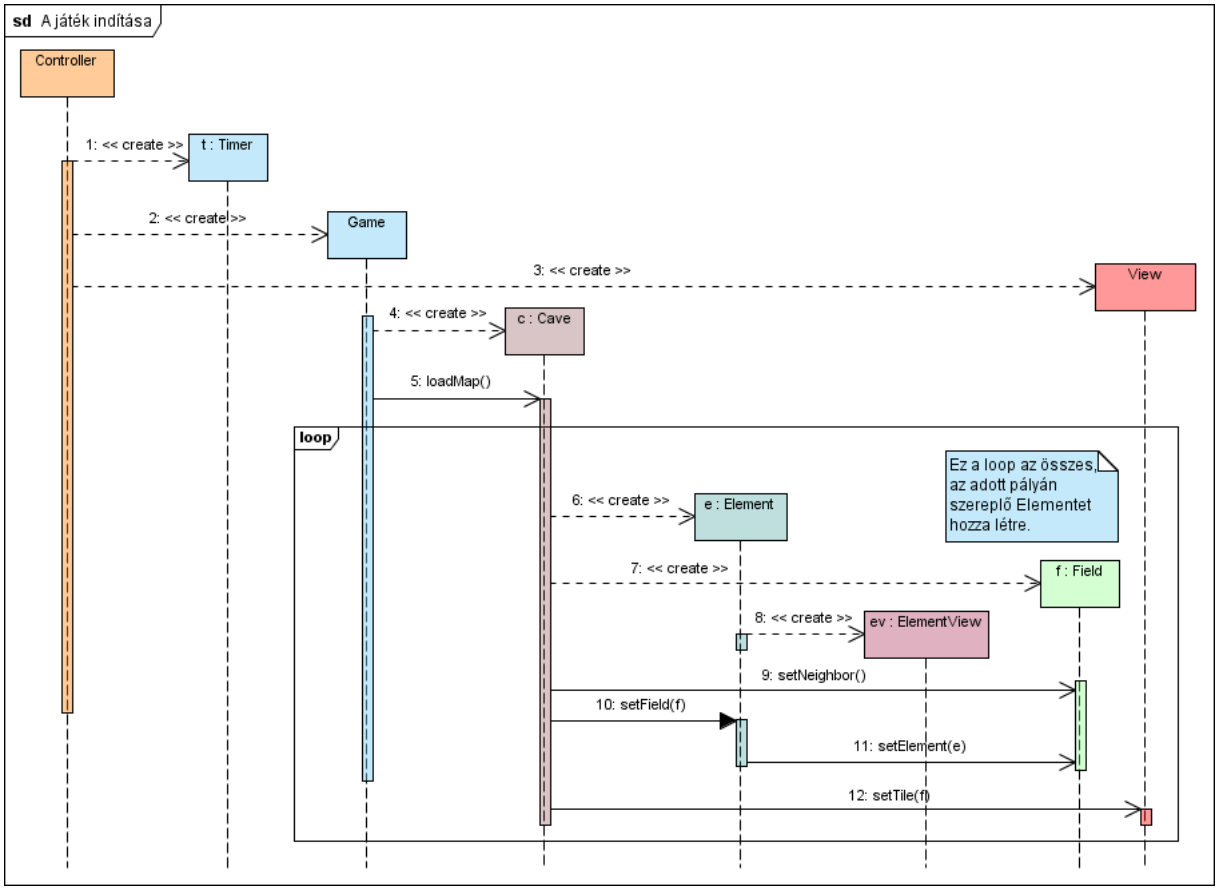
A grafikus verzióhoz, a jelenlegi osztályok mellé, a következők felvétele szükséges:

`Controller`: Feladata a játék indítása, a `Timer`, a `Game` és a `View` objektumok létrehozása, majd a két actor (Játékos és Óra) use case-einek megfelően a Model és a View réteg állapotának változtatása.

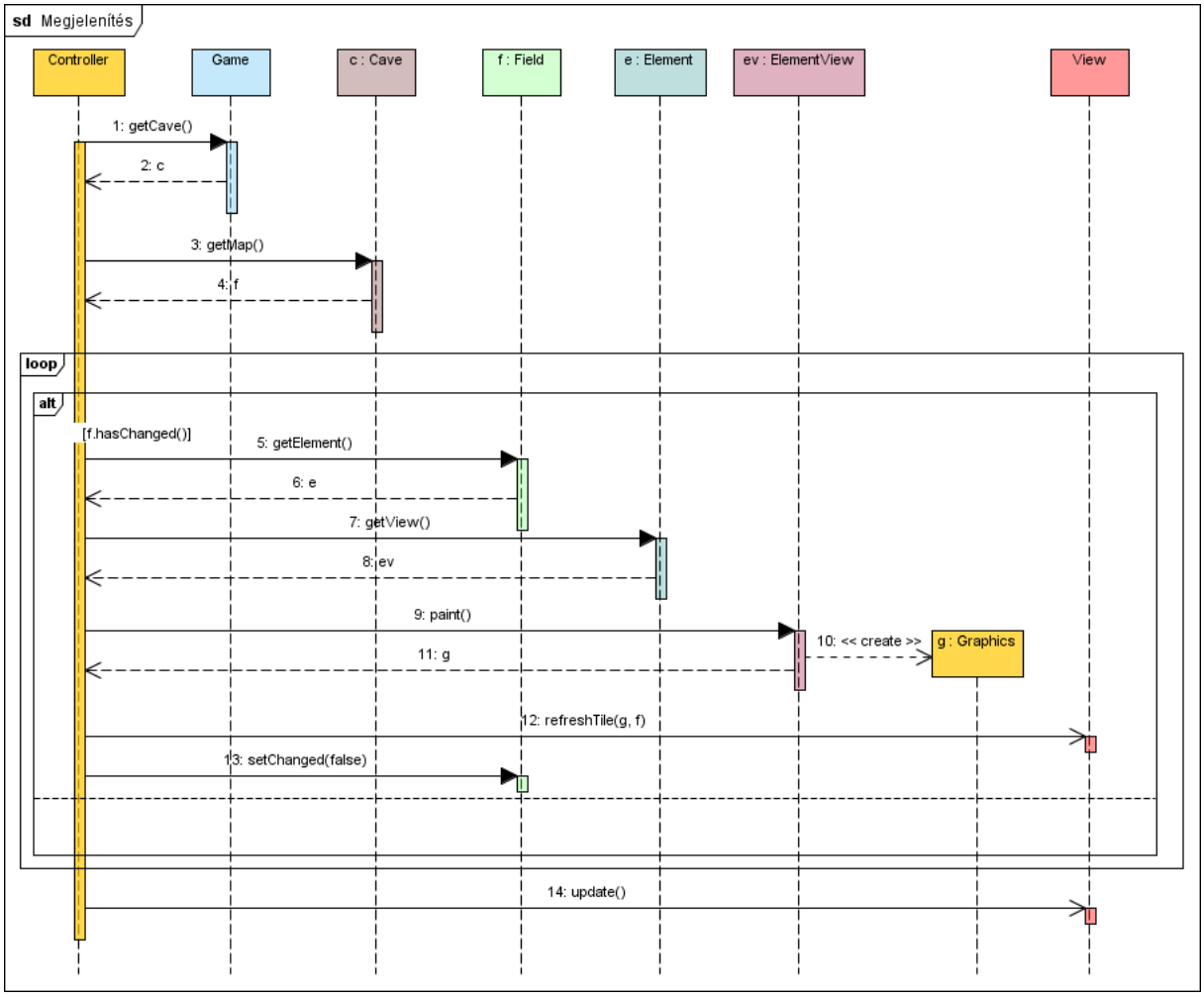
`View`: A megjelenítésért felelős osztály, amely felépíti a kezelőfelületet és a pálya egyes részeinek frissítését lehetővé teszi a `Controller`-nek.

`Coordinates`: Egyszerű segédosztály a `View` számára, ezt rendeli az egyes pályán szereplő elemekhez.

`ElementView` (és leszármozottai): Minden egyes pályán szereplő elem (`Element`) redelkezik az önmaga megjelenítéséért felelős `ElementView` osztállyal.



36. ábra. Grafikus felület indítása



37. ábra. Egyes térrészek frissítése

9.4. Napló

2009.04.21. 10:30	Farkas, Siklósi, Tóth-Máté, idő: 02:30
Grafikus felület specifikációja, tervek	
2009.04.21. 13:00	Tóth-Máté, idő: 01:00
Grafikus felület osztálydiagramjának készítése	
2009.04.21. 23:30	Zsolnai, idő: 01:00
A menürendszer, a kezelői felület grafikus képe, egyéb szövegezések (11.1)	
2009.04.22. 21:30	Siklósi, idő: 01:00
Szekvencia diagramok (inicializáció, megjelenítés)	
2009.04.22. 22:30	Tóth-Máté, idő: 01:00
Véglegesítés, elkészült anyagok rendezése	

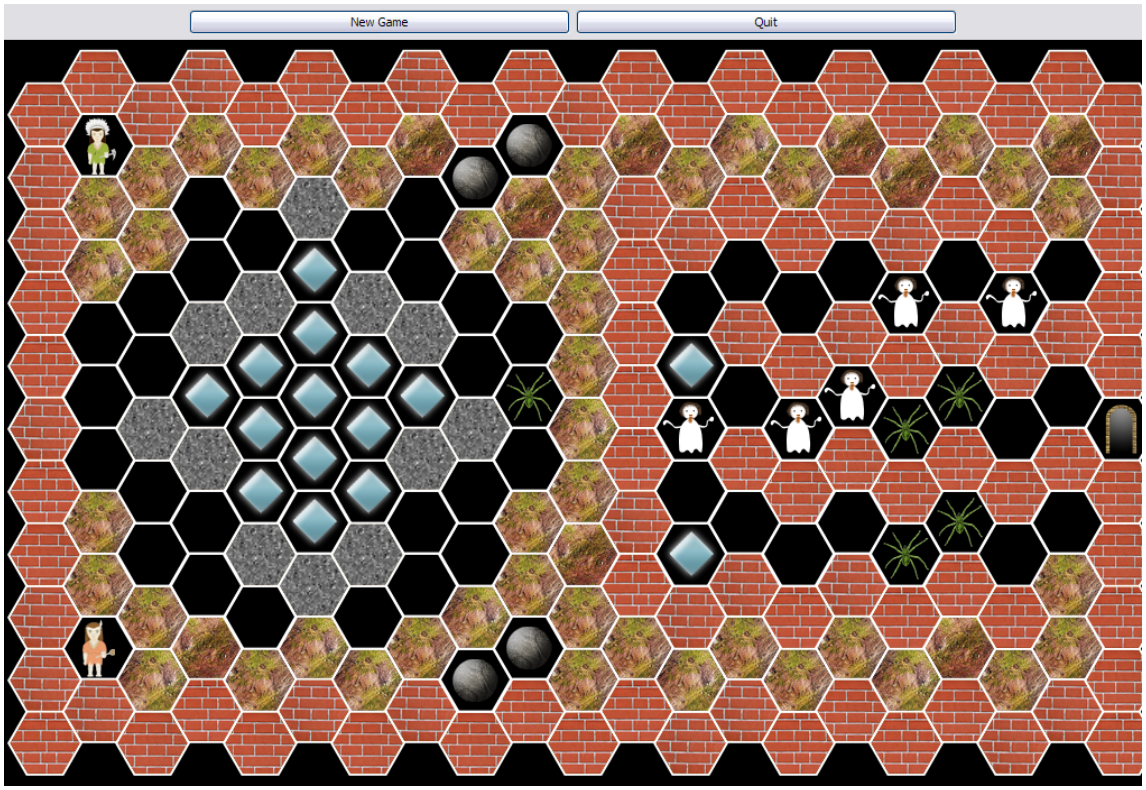
10. Grafikus változat beadása

10.1. A grafikus változat

10.1.1. A grafikus változat fordítása és futtatása

A grafikus lefordítása és futtatása automatizáltan történik, a fejlesztőcsapat által mellékelt indítófájl segítségével. Ez az állomány a könyvtárszerkezet /bin/ mappája alatt található, és a célszámítógépen található Java fordító segítségével állítja elő a már futtatható, kész programkódot. A fordítás a graphic_run.bat indítófájllal történik, ez helyben létrehoz egy futtatható jar fájlt, melyet el is indít (a továbbiak során ez bármikor duplakattintással futtatható, illetve parancssorból a `java -jar BoulderDash.jar` utasítással indítható).

Megjegyzés: Szükséges feltétel, hogy a Java binárisainak elérési útja be legyen állítva a megfelelő környezeti változóban (pl. Microsoft Windows alatt a set PATH paranccsal).



38. ábra. Egy képernyőkép a kész játékból

10.1.2. A grafikus változat fájljai

A kész grafikus változat a következő fájlokat tartalmazza:
(A kiírt fájlméretek byte-ban értendők)
Az /src/BoulderDash könyvtár a programot alkotó forráskódokat, grafikus felület elemeit és a pályákat tartalmazza:

Fájlnév	**Méret**	**Létrehozás**	**Leírás**
Boulder.java	3106	2009.03.11. 16:47	A szikla elemet megvalósító osztály
BoulderDash.java	203	2009.04.30. 20:33	A játék belépési pontját tartalmazó osztály
BoulderView.java	178	2009.05.01. 15:48	A szikla megjelenítéséért felelős osztály
Cave.java	7921	2009.03.11. 16:47	A játékteret megvalósító osztály
Controller.java	6735	2009.04.30. 20:33	A játék vezérléséért felelős osztály
Coordinates.java	399	2009.04.30. 20:33	Koordináták kezelését szolgáló segédosztály
Diamond.java	1043	2009.03.11. 16:47	A gyémánt elemet megvalósító osztály
DiamondView.java	183	2009.05.01. 15:48	A gyémánt megjelenítéséért felelős osztály
Dirt.java	495	2009.03.11. 16:47	A föld elemet megvalósító osztály
DirtView.java	423	2009.05.01. 15:48	A föld megjelenítéséért felelős osztály
Element.java	4318	2009.03.11. 16:47	A térelemek ősosztálya
ElementView.java	322	2009.05.01. 15:48	Az elemek megjelenítéséért felelős osztályok őse
Empty.java	398	2009.03.11. 16:47	Üres mező elemet megvalósító osztály

EmptyView.java	175	2009.05.01. 15:48	Üres mező elem megjelenítéséért felelős osztály
Exit.java	528	2009.03.11. 16:47	Kijárat elemet megvalósító osztály
ExitView.java	174	2009.05.01. 15:48	A gyémánt megjelenítéséért felelős osztály
Explosive.java	979	2009.03.11. 16:47	Robbanóanyag elemet megvalósító osztály
ExplosiveView.java	193	2009.05.01. 15:48	Robbanóanyag megjelenítéséért felelős osztály
Field.java	2661	2009.03.11. 16:47	Mező elemet megvalósító osztály
Game.java	6841	2009.03.11. 16:47	A játék elemeit összefogó osztály
Granite.java	342	2009.03.11. 16:47	Gránit elemet megvalósító osztály
GraniteView.java	182	2009.05.01. 15:48	Gránit megjelenítéséért felelős osztály
ImageCache.java	1524	2009.05.01. 17:41	Képeket előtöltő osztály
Monster.java	5045	2009.03.11. 16:47	A szörny ellenfeleket megvalósító osztály
MonsterView.java	366	2009.05.01. 15:48	A szörny megjelenítéséért felelős osztály
Player.java	3122	2009.03.11. 16:47	A játékos karaktert megvalósító osztály
Player1View.java	179	2009.05.03. 17:15	Az egyik játékos karakter megjelenítéséért felelős osztály
Player2View.java	191	2009.05.03. 17:15	A másik játékos karakter megjelenítéséért felelős osztály
Timer.java	1012	2009.03.11. 16:47	Az időzítésért felelős osztály

View.java	7169	2009.04.30. 20:33	A megjelenítésért felelős osztály
Wall.java	95	2009.03.11. 16:47	A fal elemet megvalósító osztály
WallView.java	421	2009.05.01. 15:48	A fal elem megjelenítéséért felelős osztály
maps/ map1.bdm	128	2009.05.02. 11:41	Az első pálya
maps/ map1.bdm	178	2009.05.05. 22:21	A második pálya
maps/ map3.bdm	218	2009.05.06. 12:14	A harmadikpálya
maps/ map4.bdm	251	2009.05.06. 12:14	A negyedik pálya
res/ boulder.png	7517	2009.05.01. 11:52	Szikla képe
res/ diamond.png	6761	2009.05.01. 11:52	Gyémánt képe
res/ dirt1.png	11546	2009.05.04. 00:19	Föld 1. képe
res/ dirt2.png	11438	2009.05.03. 22:58	Föld 2. képe
res/ dirt3.png	11509	2009.05.03. 22:58	Föld 3. képe
res/ empty.png	3633	2009.05.01. 11:52	Üres elem képe
res/ exit.png	6759	2009.05.01. 11:52	Kijárat képe
res/ explosive.png	5849	2009.05.01. 11:52	Robbanóanyag képe
res/ granite.png	11252	2009.05.01. 11:52	Gránit képe

res/ monster1.png	8762	2009.05.04. 00:19	Szörny 1. képe
res/ monster2.png	6501	2009.05.03. 22:58	Szörny 2. képe
res/ monster3.png	5246	2009.05.03. 22:58	Szörny 3. képe
res/ player1.png	7573	2009.05.03. 22:58	Játékos karakter 1. képe
res/ player2.png	7350	2009.05.03. 22:58	Játékos karakter 2. képe
res/ wall1.png	10964	2009.05.04. 00:19	Fal 1. képe
res/ wall2.png	11003	2009.05.03. 17:02	Fal 2. képe
res/ wall3.png	10967	2009.05.03. 17:02	Fal 3. képe

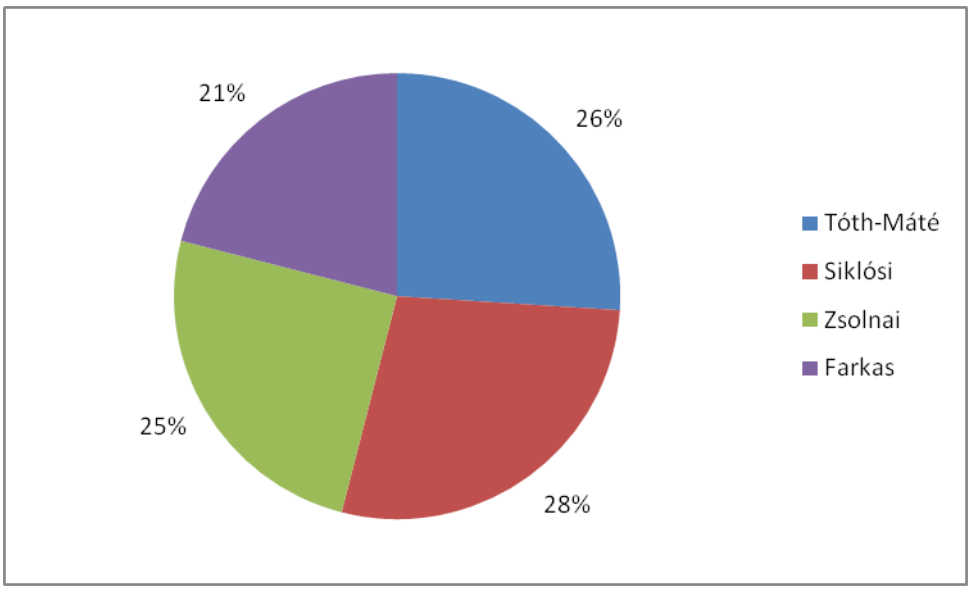
A /bin/ könyvtár tartalmazza a futtatható és lefordított bináris állományokat:

Fájlnév	**Méret**	**Létrehozás**	**Leírás**
graphic_docgen.bat	130	2009.05.06. 01:02	A javadoc dokumentációt elkészítő batch fájl
graphic_run.bat	163	2009.04.30. 20:33	A fordítást végző és a futtatható jar fájlt készítő batch fájl

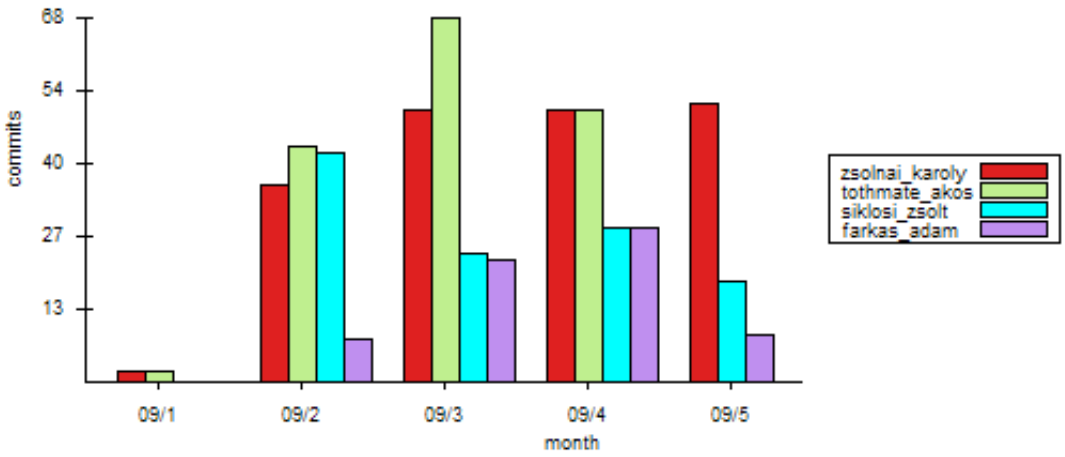
10.2. Értékelés

A grafikus felület létrehozása az eddigi munkamenetekhez képest rendhagyó módon zajlott - a rendszer megtervezése azt igényelte, hogy a csapat tagjai összeüljenek, és együtt vitassák meg a rá vonatkozó részleteket, a konkrét implementáció jelentős része pedig szinte kizárólagosan Siklósi Zsolt keze munkáját dícséri. Az utolsó fejlesztési szakasz folyamán komolyabb problémák nem adódtak, az eddigiekhez képest különbséget jelentett, hogy már a játszhatóságnak és a megjelenésnek is szükséges volt külön figyelmet szentelni, így rengeteg apró megoldatlan feladat várt megoldásra. Fontos volt továbbá az alapvető ergonómiai követelményeket, tényezőket is figyelembe venni, hogy a játékkal való időtöltés során minden magától értetődő legyen, és a lehető legkényelmesebben tudjon lezajlani.

A csapatban végzett munkaórák eloszlása:
Siklósi: 85:15h
Tóth-Máté: 78:30h
Zsolnai: 75:45h
Farkas: 63:00h
Teljes munkaidő: 302:30h



39. ábra. Munkaidők megoszlása százalékosan



40. ábra. Commit statisztika

Number of weeks:				16
Number of authors:				4
Total commits analyzed:				542
Total file changes:				2172
		Average	Min	Max
Commits each week:		33	4	166
Most active author:	zsolnai_karoly	12	2	52
Least active author:	farkas_adam	4	0	29
File changes each week:		135	4	770

41. ábra. További commit statisztikák

10.3. Napló

2009.04.30. 15:00	Siklósi, idő: 02:00
Grafikus tervek	
2009.05.01. 10:00	Siklósi, idő: 06:30
Grafikus felület implementációja, View, ElementView osztályok	
2009.05.01. 10:00	Zsolnai, Farkas, idő: 04:30
Grafikus felület implementációja	
2009.05.02. 10:00	Siklósi, idő: 03:00
Pályabetöltés, megjelenítés implementációja	
2009.05.02. 10:00	Zsolnai, idő: 02:00
Grafikus felület elemeinek szerkesztése, implementációja	
2009.05.03. 15:30	Siklósi, idő: 04:30
Karakterek mozgatása, időzítés, dupla pufferelés, egyéb fejlesztés	
2009.05.03. 11:30	Zsolnai, idő: 03:30
Grafikus felület elemeinek szerkesztése, implementációja	
2009.05.04. 12:00	Siklósi, idő: 00:30
Apróbb fejlesztések, szörny mozgás, bombalerakás, stb.	
2009.05.04. 20:00	Siklósi, idő: 02:00
Új játék kezdése, szörny mozgás optimalizálása, stb.	
2009.05.05. 19:00	Siklósi, idő: 03:30
Pályaváltás, státuszsávon információk megjelenítése, egyéb fejlesztések	
2009.05.06. 00:00	Zsolnai, idő: 02:30
Lépési precedenciák megváltoztatása, tesztelése, egyéb optimalizációk, javadoc újraírása	
2009.05.06. 23:00	Zsolnai, idő: 02:00
Pályaszerkesztés, statisztikák készítése, értékelés (13.2)	
2009.05.06. 23:45	Farkas, idő: 03:30
Szörny tulajdonságainak végleges implementálása	
2009.05.07. 08:00	Tóth-Máté, idő: 01:35
Leadandó dokumentáció szerkesztése	

11. Összefoglalás

11.1. Projekt összegzés

A csapatokra váró feladatot egy furcsa kettősség jellemezte: egyszerre kellett programkód helyett modellben gondolkodni, és meg kellett oldani egy négy ember számára kicsinek mondható feladatot, de úgy, olyan szemléletmódban, ami egy nagy rendszer fejlesztéséhez szükséges. Megoldani a feladatot, megírni a kódot egyszerű - úgy megírni, hogy az megfelelően moduláris, bővíthető, karbantartható legyen, bizony igen komoly feladatnak bizonyult.

Az egyetemi tanulmányokat eddig jellemző önálló munkát a csapatmunka váltotta fel. A közös munkálatok a fejlesztés egy teljesen új aspektusát mutatták be - csapatban már nem elegendő csak elvégezni a munkát, annak folyamatában és annak végeztével gondoskodni kell arról, hogy a csapat többi tagja megfelelő mennyiségű információt kapjon arról, hogy a rendszer egyes részei éppen milyen állapotúak, hogyan, mikor készülnek el. Új dolgokra kell odafigyelni, olyanokra, mint a kódolási konvenciók - az első néhány osztály megalkotását követően ránézésre meg lehetett állapítani, hogy melyik kódrészlet pontosan melyik tag keze munkáját dicséri. A fejlesztési munkafolyamatok tekintélyt parancsoló sebességgel haladhatnak, amennyiben annak részein a tagok individuálisan, párhuzamosan tudnak dolgozni, a feladat tehát adott volt: a teljes rendszer részeire való olymódú dekompozíciója, hogy a lehető legtöbben legyenek képesek rajta egyszerre dolgozni, anélkül, hogy az egyik interfész konkrét implementációja egy másikétól függjön.

Mindezen új aspektusok megmutatták, hogy négy ember közös munkájának szinkronizációja, koordinálása igen komoly kihívást jelent, és cseppet sem meglepő tény az, ha a nagyobb vállalatoknál ezekhez a feladatokhoz külön munkaköröket rendeltek. A csapat tagjai egyetértenek abban, hogy a projekt kivitelezése folyamán számos új tapasztalattal gazdagodtak, és abban, hogy a későbbiekben, ha komolyabb rendszerek tervezésére kerül sor, ott is bőséggel akad majd új ismeret, amit el lehet és el kell sajátítani.