Kígyófarm

9 – **D**emonic **D**evelopment **T**eam

Konzulens:
Kovacsics Péter

Csapattagok

Csuzdi Csaba Dávid	NMXKRD	csuzdy@freestart.hu
Fehér Zsolt	S4I88L	kondvezer@gmail.com
Györgyey Tamás	LT96FV	patate@sch.bme.hu
Major Péter	SDI6MA	majorpetya@gmail.com

2008. május 14.

Tartalomjegyzék

1. fejezet

Követelmény, projekt, funkcionalitás

1.1. Követelmény definíció

1.1.1. A program célja, feladata

Ez a program a népszerű kígyó játéknak egy változata, melyben a cél, hogy az idő lejártakor a kígyó hosszabb legyen mint az ellenfélé. A kígyókat irányítva különböző bogyókat kell megenni. Részletesebb leírás megtalálható az 1.3. részben.

1.1.2. A projekt célja

A projekt célja, hogy a csapat készítsen egy játékprogramot, amely hiba nélkül fut minden olyan gépen, amelyen legalább Java 1.4.2 futtatókörnyezet található. Fontos szempont továbbá, hogy a játék minél élvezhetőbb legyen. Fejlesztés során törekedünk az UML 2.0 tökéletes használatára.

1.1.3. A fejlesztő környezet

A fejlesztéshez a NetBeans 6.0.1 szoftvert használjuk, mely támogatja a CVS-t és a forward engineering-et, azaz képes az osztálydiagramból kódot generálni. A cél, hogy a játékprogramot lehessen fordítani és futtatni a HSZK számítógépein, ezért a Java SDK 1.4.2 kompatibilitást állítjuk be a NetBeans-ben. A dokumentumokat szintén a NetBeans-ben írjuk, hogy a csapattagok bármikor elérhessék a teljes projektet. A dokumentumok végleges verzióinak szerkesztését LaTeX-hel valósítjuk meg, mely képes pdf-be konvertálni.

1.1.4. A játék rendszerkövetelménye

A játék futtatható lesz minden olyan rendszeren, amely képes futtatni a Java Runtime Environment-et. Sun ajánlása PC esetén: Minimum Pentium 166 Mhz 32 MiB RAM-mal. A játék használatához szükség van billentyűzetre, egérre és természetesen grafikus képernyőre.

1.1.5. A felhasználói felület

A játékprogram végleges verziója grafikus felhasználói felülettel fog rendelkezni. A felhasználók a játékot egér segítségével indíthatják és a billentyűzettel vezérlik.

1.1.6. A minőségi követelmények

Teljesítmény: A cél, hogy a játék tökéletesen fusson a rendszerkövetelmények résznél meghatározott számítógépen, a grafikus felület animációi és a játékmenet folyamatos legyen.

Újrafelhasználhatóság: Mivel a grafikus felület teljesen külön lesz a program többi részétől, ezért egy esetleges grafikus felület változtatás egyszerűen és gyorsan végrehajtható.

Rugalmasság: A játékprogram futtatható lesz minden olyan környezetben, melyben megtalálható a megfelelő Java futtatókörnyezet.

Felhasználhatóság: A játék egyszerűsége révén nem szükséges semmilyen alapismeret vagy számítástechnikai tudás, bárki könnyedén használhatja, akár a felhasználói kézikönyv elolvasása nélkül is.

1.1.7. A szoftver minősítése

A végső játékprogram akkor megfelelő, ha minél pontosabban teljesíti a fentebb leírtakat. Ez ellenőrizhető a játék futtatásával és kipróbálásával, valamint a kód és a modell összevetésével.

1.1.8. A kibocsátás

A játékprogram kibocsátása a forráskóddal és dokumentációval együtt a konzulens előtt fog megtörténni, majd az ellenőrzés és értékelés után elérhető lesz az interneten is.

1.2. Projekt terv

1.2.1. Bevezetés

A feladat egy a fent említett követelményeknek megfelelő program, és az ahhoz tartozó tervek, valamint dokumentációk elkészítése. A projekt megvalósításának alapvetően három fő fázisa van:

A *szkeleton* elkészítése és működőképessége bizonyítja, hogy az előtte készített objektum és dinamikus modellek működőképesek, és valóban alkalmasak a specifikációban leírt feladat végrehajtására. Ennek ellenőrzése érdekében a programnak futása során előre meghatározott pontokon üzeneteket kell kiírnia, ezzel megkönnyítve a tesztelést.

A következő fázis a *prototípus* elkészítése. Ekkor illesztjük be a különböző algoritmusokat. A program már tulajdonképpen kész, csak grafikus megjelenítés nincs. A jó tesztelhetőség érdekében a működést alfanumerikus képernyőn kell megjeleníteni, valamint a be– és kimenetnek fájlok használatával is vezérelhetőnek, illetve megfigyelhetőnek kell lennie.

Az utolsó fázis a *grafikus* változat elkészítése. Ekkor kapja meg a végleges felhasználói felületet a program. A játékélmény szempontjából fontos a jó áttekinthetőség, egyszerű kezelhetőség, valamit a vizuálisan is hibamentes működés. Éppen ezért ebben a fázisban is fontos szerepet kap a tesztelés.

1.2.2. Fejlesztési ütemterv

Az 1.2.1. részben említettek alapján a fejlesztés három lépcsős:

Szkeleton A modellezési fázis. Törekedni kell a minél általánosabb modellre, főleg annak ismeretében, hogy a követelmény változni fog a projekt folyamán. A legtöbb intuíciót és kreativitást igénylő fázis. A tesztelése során felfedhetjük a modell esetleges megbúvó elvi hibáit, hiányosságait.

Prototípus A modell alapján elkészített, részben generált alfanumerikus felületet használó program. A tesztelés során fényt deríthetünk az implementáció hibáira.

Grafikus változat A prototípus alfanumerikus felületét felváltja a grafikus felhasználói interfész. A program elnyeri végső külalakját.

1.2.3. Kockázatok kezelése

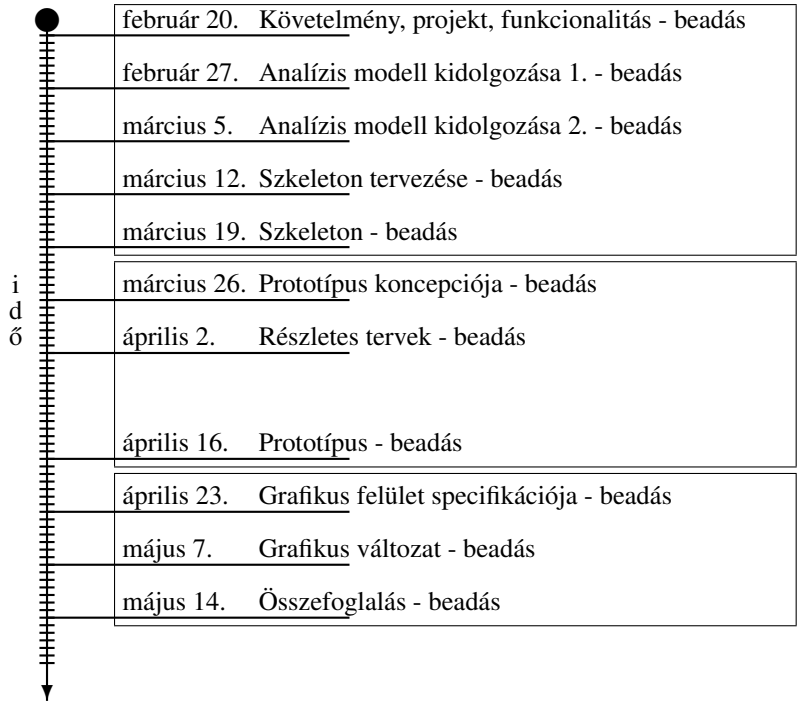
	Kockázat	Valószínűség	Hatás
1	Egy csapattag távozik	alacsony	kezelhető
2	Több csapattag távozik	nagyon alacsony	katasztrofális
3	Tönkremegy a CVS szerver	alacsony	kezelhető
4	Követelmények változása	nagyon magas	kezelhető
5	Megszűnik az internetkapcsolat a VPK-ban	alacsony	kezelhető

1.1. táblázat. Kockázat analízis

Az 1.1. táblázatban a projektet befolyásoló eseményeket foglaltuk össze. Ezek kezelésére, és elkerülésére a következő stratégiákat dolgoztuk ki:

1. Egy csapattag kilépése esetén a csapat a projekt definíció szerint 3 fővel még működhet tovább. Ez esetben a kilépő tag feladatait szét kell osztani. Új ember bevétele nem lehetséges. (*Brooks tv.*)
2. Több csapattag távozása esetén a projekt sikeres befejezése kétségessé válik, azonnal a konzulenshez kell fordulni.
3. A csapat saját CVS szervert használ, ezért a meghibásodások okozta adatvesztés elkerülése érdekében rendszeresen biztonsági másolatot készítünk a repository-ról.
4. A projekt elkészítése során várható a követelmények változása, ekkor az új követelményt be kell illeszteni a modellbe, elemezni kell a változás projektre gyakorolt hatását.
5. Amennyiben megszűnik az internetkapcsolat, akkor a CVS szerver elérhetetlenné válik hálózaton kívülről, azonban belső hálózaton még elérhető, tehát az ilyen hiba csak hétvégén kritikus.

1.2.4. Fontos időpontok



1.1. ábra. Időbeosztás

1.2.5. Csapat működése

A csapat négy emberből áll. Tagjai egy kollégiumi szobában laknak, ami jelentősen leegyszerűsíti a kapcsolattartást. Az aktuális feladatok kiosztása mindig egy megbeszélésen történik, ekkor törekszünk arra, hogy mindenki azonos nehézségű feladatot kapjon, és olyat, amiben kedvét leli. A megbeszélés befejezése után a naplóba minden esetben rögzítjük a meghozott döntéseket, hogy ki, milyen feladatot kapott, és milyen határidővel.

Munka közben, amennyiben dokumentációról van szó, mindenki saját fájlba dolgozik a CVS szerveren. Ez a fájl a *doc/unstable* katalógusban található. A naplóját szintén mindenki külön vezeti egy nevével ellátott *log* kiterjesztésű fájlban. A megbeszéléseket külön naplófájlban, a *discuss.log*-ban jegyzőkönyvezzük. A naplófájlok formátuma kötött az egyszerű feldolgozhatóság végett. Beadás előtt lefuttatjuk a *loggen* nevű szkriptet, ami egy közös fájlba egyesíti a naplókat, időrendbe rendezi a kezdés ideje szerint, és átkonvertálja LaTeX-es táblázattá. Ezután az egyénileg készített anyagokat formázzuk, majd elkészítjük a beadandó dokumentumot. Ennek elkészültével a csapat megbeszélésen dönt a dokumentum véglegesítéséről, és ha mindenki egyetért, akkor az átkerül a *doc/stable* katalógusba, kinyomtatjuk és beadjuk.

1.3. A játékprogram leírása

A játékban kígyók küzdenek egymás ellen a győzelemért. A játéktéren két különböző színű kígyó mozog, amelyet egy számítógépen különböző nyomógombokkal (amelyet majd akár át is lehet állítani) vezérlünk. A kígyónak a fejét tudjuk vezérelni, hogy melyik irányba menjen, a teste pedig folyamatosan követi ugyanazokat az irányokat, tehát amerre megy a kígyó feje, arra fog menni a kígyó teste is. Egy kígyót összesen négy gombbal lehet irányítani, ahol az egyes gomb azt mondja meg hogy melyik irányba menjen a kígyó, de nem lehet egy vonalban megfordulni. A kígyók addig élnek, amíg falnak, másik kígyónak vagy saját maguknak (ún. becsavarodás) nem ütköznek. A játékot lehet szüneteltetni: ez az 'Esc' gomb lenyomásával történik, majd újbóli megnyomásával lehet folytatni a játékot.

A játéktéren különböző fajtájú bogyók találhatók, amelyek különböző hatásokkal lehetnek a kígyókra. Ha egy mezei bogyót eszünk meg, akkor a kígyó hossza egy egységgel hosszabb lesz. Ha fűrészbogyót eszünk, a kígyónk harapása egy rövid ideig olyan erős lesz, hogy amikor a másik kígyónak (vagy akár saját magának) nekimegy, áthalad rajta, levágva az ellenfél (vagy a saját) farkát. Ha viszont egy kőbogyót fogyasztunk el, akkor az folyamatosan megy hátra a kígyó testében egészen a farokig, ahol majd felhalmozódik. A kőbogyó az érintett testszakasz fenti módon leírt megharapását akadályozza meg, ezzel a köves részre harapó kígyót elpusztítva.

A játéktéren állandóan két mező- valamint egy-egy fűrész -és kőbogyó van. A játék elindításakor a kígyók a pályától függően a megadott helyekről megadott irányban indulnak el. A játék kezdésekor a kígyók hossza egyenként 3 egység. A játék időre megy, ami alapesetben 2 perc, de ezt akár a felhasználó is beállíthatja. Be lehet majd továbbá azt is állítani, hogy milyen sebességgel haladjanak a pályán a kígyók. Az a játékos nyer, akinek az utolsó pillanatban hosszabb a kígyója, vagy egységes kígyóhossz esetén az számít, hogy kinek a testében van kevesebb kőbogyó. Ha a kőbogyók száma is egyenlő, akkor az eredmény döntetlen. Ha a játék folyamán az egyik kígyó meghal, akkor a pálya egy másik részén új életet kap és folytathatja a játékot, de a hossza újra 3 egység lesz. Ha valamelyik játékos két lépés között több irányt is megad, akkor az utoljára megadott lesz érvényes.

Speciális esetek:

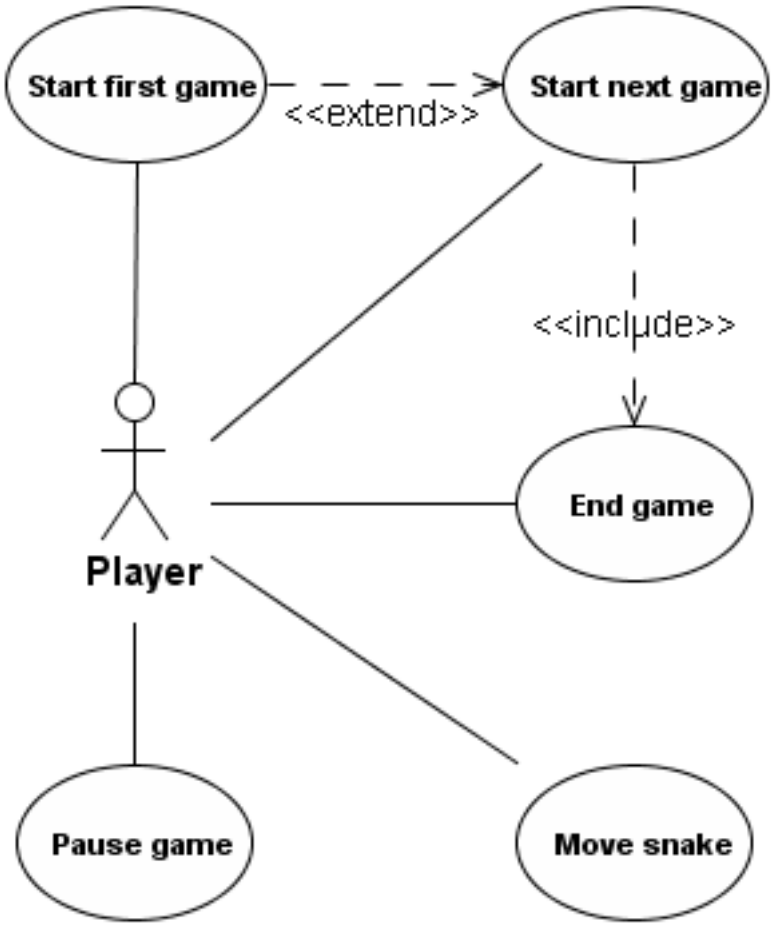
- Ha egy fűrészbogyó hatása alatt álló kígyó fejjel nekimegy egy másik (fűrészbogyót nem fogyasztott) kígyó fejének, akkor az a kígyó hal meg, amelyik nem evett fűrészbogyót. Minden egyéb esetben a fej-fej ütközés mind a két kígyót megöli.
- Ha egy kígyó olyan sok kőbogyót megevett már, hogy a feje után már csak köves részek vannak, akkor ha egy új kőbogyót akar felvenni, meghal a kígyó.
- Ha egy fűrészbogyós kígyó úgy harapja át a másikat, hogy az rövidebb lenne a harapás után 3 egységnél, akkor a megharapott kígyó meghal.
- Ha két kígyó egyszerre lép rá egy olyan mezőre a pályán, ahol valamilyen bogyó található, akkor a fej-fej ütközés szabálya lép életbe, és ha van túlélő kígyó, az eszi meg a bogyót.

1.4. Szótár

átharap	Egy kígyó úgy ütközik egy másik kígyóval, hogy azzal a másikat megrövidíti, miközben ő maga életben marad.
becsavarodás	Amikor az egyik kígyó saját magának megy neki.
bogyó	Amit, ha egy kígyó megeszik különböző hatással lesz a kígyóra, függően a bogyó fajtájától.
fal	Egy olyan rész a játéktéren, aminek, ha egy kígyó nekimegy, meghal.
fűrészbogyó	A játéktéren található. Ha ezt megeszi egy kígyó, akkor egy rövid ideig képes lesz átharapni egy másik kígyót.
játéktér	Ezen a területen mozoghatnak a kígyók. Falak határolják körbe.
kezdeti hossz	A játék elején, vagy amikor a kígyó új életet kap, ilyen hosszúsággal jelenik meg a pályán.
kígyó	Ezt irányítja a játékos a játék során.
kígyó farka	Alapértelmezésben a kígyó feje utáni rész, de ha egy fűrészbogyós kígyó átharapja a kígyó testét, akkor a farok az átharapott testrész utáni területet jelenti.
kígyóhossz	Egy szám, ami azt mutatja meg, hogy a kígyó (fejjel együtt) hány egység hosszú.
kígyó megeszik egy bogyót	A kígyó fejével egy olyan pontra kerülünk, amin eredetileg valamilyen bogyó volt. A bogyó eltűnik a pályáról, majd a fajtájának megfelelően hatással lesz az őt megevő kígyóra.
kőbogyó	A játéktéren helyezkedik el. Amennyiben egy kígyó megeszi ezt, védettséget nyújt számára a test azon részén (ahol éppen van) egy fűrészbogyós kígyó harapásától.
kőbogyók száma	Azt mutatja meg, hogy a kígyó testében hány darab kőbogyó található.
meghal a kígyó	A kígyó ebben az esetben eltűnik a pályáról, majd egy másik helyen új életet kap.
mezei bogyó	A játéktéren található, amelyet ha egy kígyó felvesz, akkor megnő a hossza egy egységgel.
pálya	A játéktér egyik formája.
új élet	A kígyó egy bizonyos helyen újra megjelenik a megadott kezdeti hosszal.
ütközés	Az amikor az egyik kígyó egy másik kígyónak, falnak, vagy épp saját magának megy neki.

1.5. Essential Use-Case-ek

1.5.1. Use-Case diagram



1.2. ábra. Essential Use-Case diagram

1.5.2. Use-Case leírások

Use-Case:	Start first game
Actor:	Player
Feltétel:	Nincs folyamatban lévő, futó játék.
Leírás:	A felhasználó új játékot indít.

Use-Case:	Start next game
Actor:	Player
Feltétel:	Folyamatban van egy játék.
Leírás:	A felhasználó új játékot indít.

Use-Case:	End game
Actor:	Player
Feltétel:	Folyamatban van egy játék.
Leírás:	A pillanatnyilag futó játék befejezése.

Use-Case:	Move snake
Actor:	Player
Feltétel:	Folyamatban van egy játék.
Leírás:	Valamelyik játékos megváltoztatja a kígyója haladási irányát.

Use-Case:	Pause game
Actor:	Player
Feltétel:	Folyamatban van egy játék.
Leírás:	A felhasználó szünetelteti a játékot.

2. fejezet

Analízis modell kidolgozása

2.1. Osztályok leírása

2.1.1. CONTROL package

Control: Ez az osztály felel a program irányításáért, a felhasználó utasításai alapján vezérli a modellt és a megjelenítést.

2.1.2. MODEL package

Berry: A különböző típusú bogyók absztrakt ősosztálya, mely a bogyók viselkedésének közös részét valósítja meg (ütközések kiértékelése, új pozíció felvétele). A *GameObject* osztályból származik.

FieldBerry: Mezei bogyót reprezentáló osztály, a *Berry* osztályból származik.

FieldElement: A pálya egy darabja, erre lehet rálépni. Van egy négy elemű tömbje, ami a szomszédos *FieldElement*-ek referenciáját tartalmazza. Tudja, hogy a pályadarabon éppen milyen *GameObject*(ek) van(nak).

GameField: Maga a pálya. A pálya létrehozásnál van fontos szerepe, a *FieldElement*-eket és összeköttetéseiket hozza létre, a *GameObject*-ek rajta keresztül adhatók hozzá a pályához. Játék közben ő ad üres *FieldElement*-eket az új pozícióra létrejövő objektumoknak.

GameObject: A pályán előforduló elemek közös absztrakt ősosztálya, tartalmazza, hogy melyik *FieldElement*-en tartózkodik, valamint a leszármazott osztályok által felüldefiniálandó metódusokat (ütközések, meghalás kezelése).

Model: A játékmenetet kezelő osztály. Felelős a pálya és a *GameObject*-ek, valamint a kígyók létrehozásáért, mozgatásáért, a játék időzítéséért.

SawBerry: Fűrészbogyót reprezentáló osztály, a *Berry* osztályból származik.

Snake: Egy kígyót reprezentáló osztály. Felelős a kígyó létrehozásáért, mozgatásáért, valamint lekérdezhető tőle a kígyó állapota (hossza, tartalmazott kőbogyók száma, hátralevő fűrészbogyó-idő). Referenciája van a kígyó fejére, azon keresztül irányítja a kígyót.

SnakeBody: A kígyó törzsének egy szegmensét reprezentáló osztály, a *SnakeElement* osztályból származik. Tartalmaz egy referenciát a következő törzsszegmensre.

SnakeElement: A kígyó testének egy szegmensét reprezentáló absztrakt ősosztály. Tartalmazza az adott testrész állapotát (halott-e, tartalmaz-e követ), valamint a kígyó működéséhez és állapotának lekéréséhez szükséges függvényeket. A *GameObject* osztályból származik.

SnakeHead: A kígyó fejét reprezentáló osztály, a *SnakeElement* osztályból származik. Tartalmazza, hogy fűrészbogyó állapotból még mennyi van hátra, valamint a kígyó mozgásának irányát. Felelős a kígyó mozgatásáért.

StoneBerry: Kőbogyót reprezentáló osztály, a *Berry* osztályból származik.

Wall: Egy falszakaszt reprezentáló osztály, a *GameObject* osztályból származik.

2.2. Objektum katalógus

Név	*Berry*
Ősosztály	`GameObject`
Komponensei	–
Relációk	`GameObject-ből származik`
Példányok száma	`nem példányosítható`

A különböző bogyók közös absztrakt ősosztálya.

Változók:

Típus	Név	Leírás
GameField	***refGameField***	Referencia a *GameField* objektumra, hogy új bogyó készítésekor tudjunk kérni egy üres helyet.

Metódusok:

Típus	Név	Leírás
boolean	***collide()***	Absztrakt függvény, mely megmondja, hogy az adott objektum túlélte-e az ütközést.
void	***die()***	Eltünteti a pályáról a bogyót, majd egy újat generál.
void	***setrefGameField (GameField)***	Beállítja a *GameField*-re mutató referenciát.

Név	Control
Ősosztály	`Object`
Komponensei	`Model, View`
Relációk	`Model-lel, View-val asszociál`
Példányok száma	`1`

Ez az objektum irányítja a játék működését, tartja a kapcsolatot a modell, a megjelenítés és a felhasználó között.

Változók: –

Metódusok:

Típus	Név	Leírás
static void	***main(String[])***	A program belépőpontja.

Név	FieldBerry
Ősosztály	`Berry`
Komponensei	–
Relációk	`Berry-ből származik`
Példányok száma	2

Egy mezei bogyót reprezentáló objektum.

Változók: –

Metódusok: –

Név	*FieldElement*
Ősosztály	`Object`
Komponensei	–
Relációk	`GameObject-tel asszociál, GameField erősen aggregálja`
Példányok száma	`n`

A pályát alkotó mező, az egyes objektumok ezeken helyezkednek el.

Változók:

Típus	Név	Leírás
FieldElement	***neighbours[4]***	Itt tároljuk el a mezők szomszédjait.

Metódusok:

Típus	Név	Leírás
void	***add(GameObject)***	Egy mezőhöz hozzáadja a megadott objektumot.
int	***getGameObjectCount()***	Visszaadja, hogy egy mezőn hány objektum van.
GameObject[]	***getGameObjects()***	Visszaadja az adott mezőn levő objektumokat.
void	***remove(GameObject)***	Egy mezőről eltávolítja a megadott objektumot.
void	***setGameObjects (GameObject[])***	Itt állíthatjuk be, hogy egy mezőn milyen objektumok helyezkednek el.
FieldElement[4]	***getneighbours()***	Visszaadja egy adott mező szomszédos elemeinek referenciáit.
void	***setneighbours (FieldElement[])***	Beállítja egy megadott mező szomszédokra mutató referenciáit.

Név	GameField
Ősosztály	`Object`
Komponensei	`FieldElement`
Relációk	`Erősen aggregálja a FieldElement-eket, Model erősen aggregálja`
Példányok száma	`1`

A játékteret reprezentáló osztály.

Változók:

Típus	Név	Leírás
Int2	***size***	Ebben tároljuk el az pálya méretét.
Int2	***startPositions[2]***	Itt tároljuk el a kígyók kiindulási koordinátáit.

Metódusok:

Típus	Név	Leírás
int	***add(GameObject, FieldElement)***	A megadott objektumot elhelyezi a specifikált mezőn.
FieldElement	***getFreePosition()***	Visszaad egy olyan mezőt, amely szabad, így tehetünk rá egy új objektumot.
FieldElement	***getFieldAt(Int2)***	A megadott számpárral azonosított mezőt adja vissza.
FieldElement[]	***getFieldElements()***	Visszatér az összes *FieldElement* referenciáját tartalmazó tömbbel.
Int2	***getsize()***	Visszaadja a pálya méretét egy struktúrában.
FieldElement	***getSnakeStartPosition (int)***	Visszaadja, hogy a megadott kígyó melyik mezőről indul.
Int2[2]	***getstartPositions()***	Visszaadja a kígyók kiindulási helyeit.
void	***setsize(Int2)***	Beállítja a pálya méretét.
void	***setstartPositions (Int2[2])***	Beállítja a kígyók kiindulási helyeit.

Név	*GameObject*
Ősosztály	`Object`
Komponensei	–
Relációk	`asszociál a FieldElement-tel`
Példányok száma	`nem példányosítható`

A pályán előforduló elemek közös absztrakt ősosztálya.

Változók: –

Metódusok:

Típus	Név	Leírás
boolean	***collide()***	Absztrakt függvény, mely megmondja, hogy az adott objektum túlélte-e az ütközést.
void	***die()***	Megöli az objektumot.
FieldElement	***getground()***	Visszatér annak a mezőnek a referenciájával, amelyen az objektum elhelyezkedik.
void	***setground (FieldElement)***	Beállítja egy adott objektumnak, hogy épp melyik mezőn van.

Név	Model
Ősosztály	`Object`
Komponensei	`GameField, Snake`
Relációk	`erősen aggregálja a GameField-et és a Snake-et`
Példányok száma	`1`

Ez az objektum reprezentálja magát a játékot.

Változók: –

Metódusok:

Típus	Név	Leírás
int	***getRemainingGameTime()***	Visszaadja a játékból hátralévő időt.
Snake[*]	***getSnakes()***	Visszaadja az egyes kígyók referenciáit, hogy azok állapotait le tudjuk kérdezni.
Object[*]	***getVisibleObjects()***	Visszaadja a pályán található összes felrajzolandó objektumot.
int	***loadMap(URL)***	Ez a függvény tölti be a pályát.
void	***reset()***	Ez a függvény felelős az időzítő visszaállításáért.
void	***setNextMove(int playerID, Direction newDirection)***	Ezzel a függvénnyel tudjuk beállítani a soron következő irányt.
void	***start()***	Az időzítőt újra elindítja a játék szüneteltetése után.
void	***stop()***	Leállítja az időzítőt.

Név	SawBerry
Ősosztály	`Berry`
Komponensei	`-`
Relációk	`Berry-ből származik`
Példányok száma	`1`

Egy fűrészbogyót reprezentáló objektum.

Változók: –

Metódusok: –

Név	Snake
Ősosztály	`Object`
Komponensei	`SnakeHead`
Relációk	`erősen aggregálja a SnakeHead-et, aggregálja a Model`
Példányok száma	`2`

A kígyót reprezentáló objektum.

Változók:

Típus	Név	Leírás
int	***playerID***	Segítségével könnyen tudjuk azonosítani az egyes kígyókat.

Metódusok:

Típus	Név	Leírás
void	***detectCollision()***	Az ütközések kezelése a feladata.
int	***getLength()***	Visszaadja a kígyó hosszát.
int	***getplayerID()***	Visszaadja a kígyó azonosító számát.
int	***getRemainingSawTime()***	Visszaadja azt az időmennyiséget (másodpercben), amely megmutatja, hogy mennyi ideig érvényes még a fűrészbogyó hatása.
int	***getStoneCount()***	Visszaadja a kígyóban levő kőbogyók számát.
void	***move()***	A kígyót lépteti egy egységgel.
void	***setNextMove (Direction)***	Itt tudjuk beállítani, hogy a kígyó következő mozgásakor melyik irányba haladjon.

Név	SnakeBody
Ősosztály	`SnakeElement`
Komponensei	–
Relációk	`SnakeElement-ből származik`
Példányok száma	`n`

A kígyó testének egy szegmensét tartalmazó kígyókomponens.

Változók:

Típus	Név	Leírás
boolean	***isTail***	Két állapotú változó, mely megmondja, hogy az adott kígyóelem a kígyó farka-e.
SnakeBody	***next***	Referencia a soron következő kígyóelemre.

Metódusok:

Típus	Név	Leírás
boolean	***getisTail()***	Visszaadja, hogy az adott kígyóelem a kígyó farka-e.
void	***setisTail(boolean)***	Beállítja, hogy az adott kígyóelem a kígyó farka-e vagy sem.

Név	*SnakeElement*
Ősosztály	`GameObject`
Komponensei	`-`
Relációk	`öröklődik a GameObject-ből`
Példányok száma	`nem példányosítható`

A kígyó testelemek absztrakt ősosztálya.

Változók:

Típus	Név	Leírás
boolean	***hasStone***	Megmondja, hogy az adott kígyóelem tartalmaz-e kőbogyót.
boolean	***isDead***	Megmondja, hogy az adott kígyóelem halott-e.
int	***playerID***	A kígyó egyéni azonosítója.

Metódusok:

Típus	Név	Leírás
boolean	***collide ()***	Felüldefiniált függvény, amely megmondja, hogy az adott objektum túlélte-e az ütközést, illetve kezeli a bogyókkal való ütközést.
void	***detectCollision()***	Feladata az ütközések kezelése.
void	***die()***	Felüldefiniált függvény, amely meghívásakor eltüntetjük az adott elemet a pályáról.
int	***digestStone()***	Ez felelős az elfogyasztott kőbogyók fokozatos hátrajutásáért egészen a farokig.
boolean	***gethasStone()***	Megadja, hogy az adott kígyóelem tartalmaz-e kőbogyót.
boolean	***getisDead()***	Megmondja, hogy az adott kígyóelem halott-e.
int	***getLength()***	Megadja rekurzív módon, hogy milyen hosszú a kígyó.
int	***getplayerID()***	Visszaadja a kígyó azonosító számát.
int	***getStoneCount()***	Megadja, hogy mennyi kőbogyót tartalmaz a kígyó.
void	***grow()***	Kígyó hosszát növeli meg egy egységgel.
void	***move(FieldElement)***	Rekurzív mozgatófüggvény.
void	***sethasStone()***	Beállítja, hogy az adott kígyóelem tartalmaz-e kőbogyót.
void	***setisDead(boolean)***	Beállítja egy adott kígyóelemre, hogy halott-e.
void	***setplayerID(int)***	Beállítja a kígyó azonosító számát az adott kígyóelemre.

Név	SnakeHead
Ősosztály	`SnakeElement`
Komponensei	–
Relációk	`SnakeElement-ből származik, aggregálja a Snake`
Példányok száma	2

A kígyó fejét reprezentáló objektum.

Változók:

Típus	Név	Leírás
SnakeBody	***body***	Referencia a kígyó testére.
Direction	***newDirection***	Ebben a változóban tároljuk a soron következő mozgás irányát.
GameField	***refGameField***	referencia a *GameField* objektumra, hogy le tudja kérni a starthelyet.
int	***remainingSawTime***	Megmutatja, hogy mennyi ideig érvényes még a fűrészbogyó hatása.

Metódusok:

Típus	Név	Leírás
boolean	***collide (FieldElement)***	Felüldefiniált függvény, amely megmondja, hogy az adott objektum túlélte-e az ütközést.
int	***digestStone()***	Felüldefiniált függvény, ez felelős az elfogyasztott kőbogyók fokozatos hátrajutásáért egészen a farokig. Amennyiben ez nem sikerül a kígyó meghal.
int	***getremainingSawTime()***	Visszaadja, hogy a fűrészbogyó hatásából hány másodperc van még hátra.
void	***move()***	Ezt a függvényt meghívva fog a kígyó egy egységgel előrébb mozogni a *newDirection*-ben meghatározott irányban.
void	***setnewDirection(int)***	Beállítja a soron következő mozgás irányát.
void	***setrefGameField (GameField)***	Beállítja a *GameField*-re mutató referenciát.
void	***setremainingSawTime(int)***	Beállítja, hogy a fűrészbogyó hatásából mennyi másodperc van még hátra.

Név	StoneBerry
Ősosztály	`Berry`
Komponensei	–
Relációk	`Berry-ből származik`
Példányok száma	1

Egy kőbogyót reprezentáló objektum.

Változók: –

Metódusok: –

Név	Wall
Ősosztály	`FieldElement`
Komponensei	–
Relációk	`FieldElement-ből származik`
Példányok száma	n

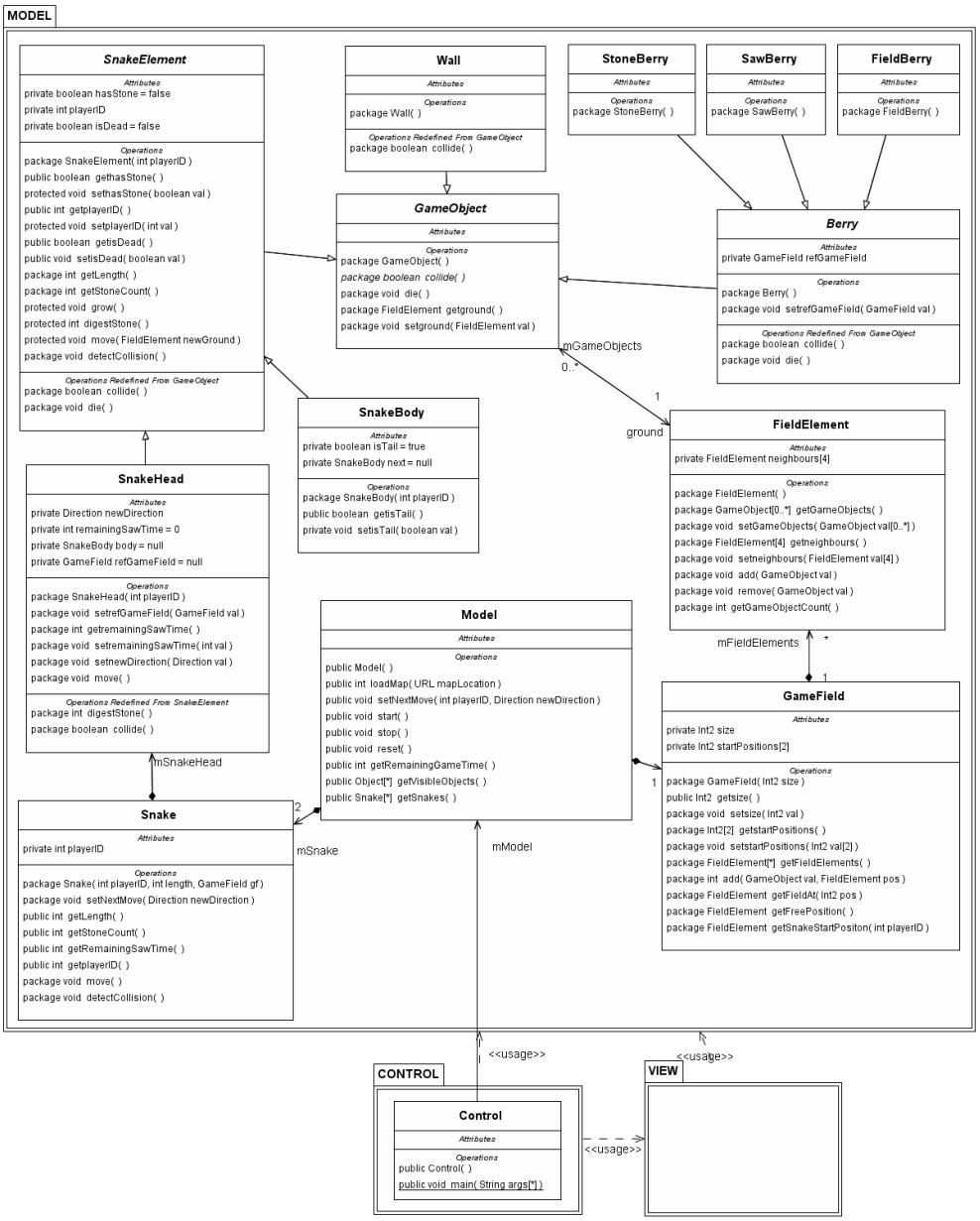
Egy falat reprezentáló objektum.

Változók: –

Metódusok: –

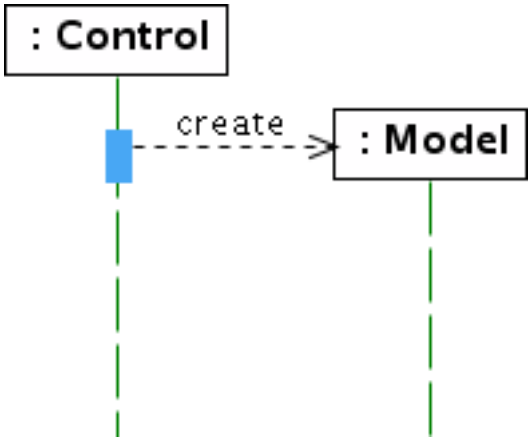
Típus	Név	Leírás
boolean	***collide (FieldElement)***	Ez a függvény megmondja, hogy az adott objektum túlélte-e az ütközést.

2.3. Statikus struktúra diagram

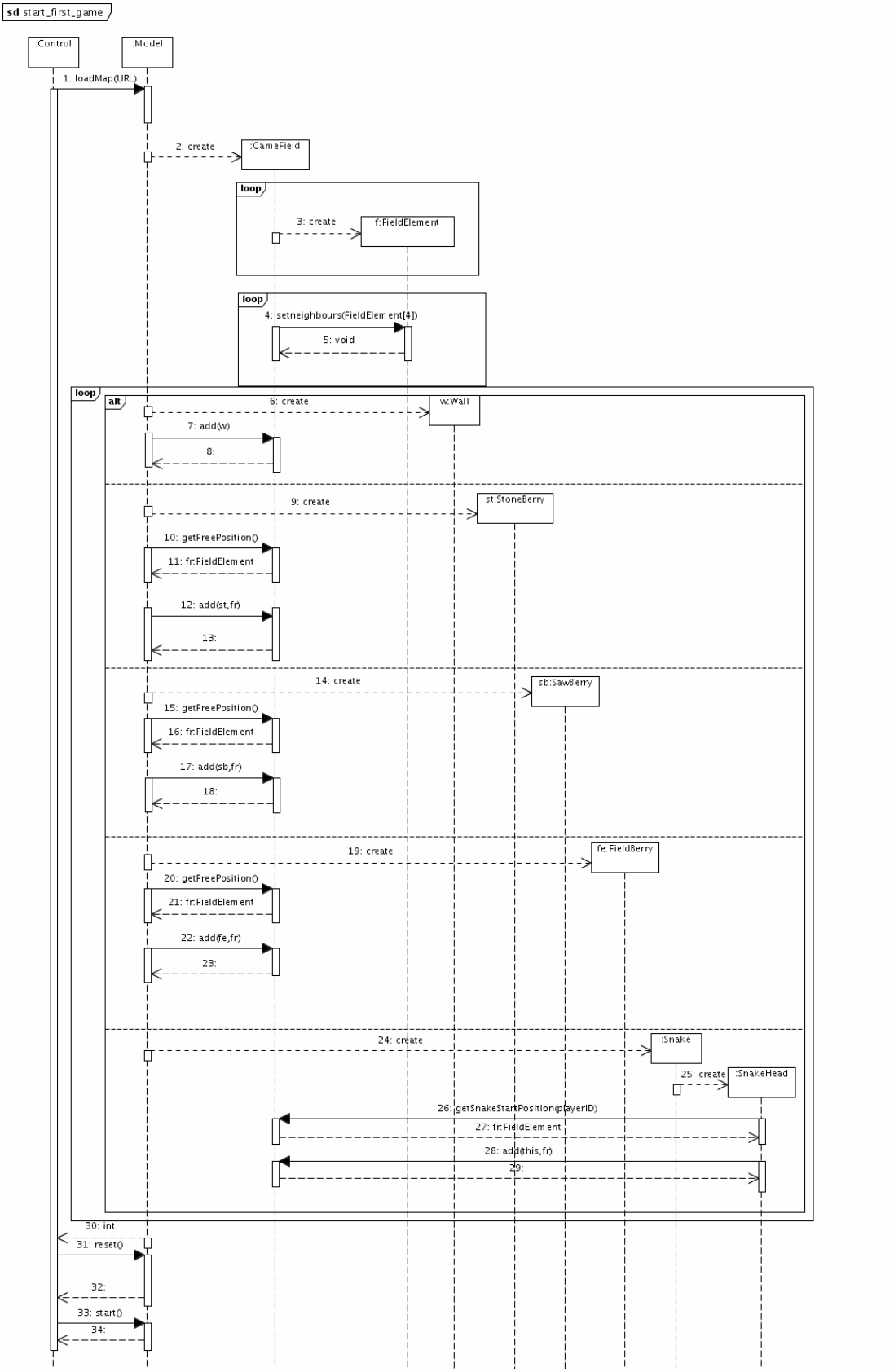


2.1. ábra. Osztály diagram

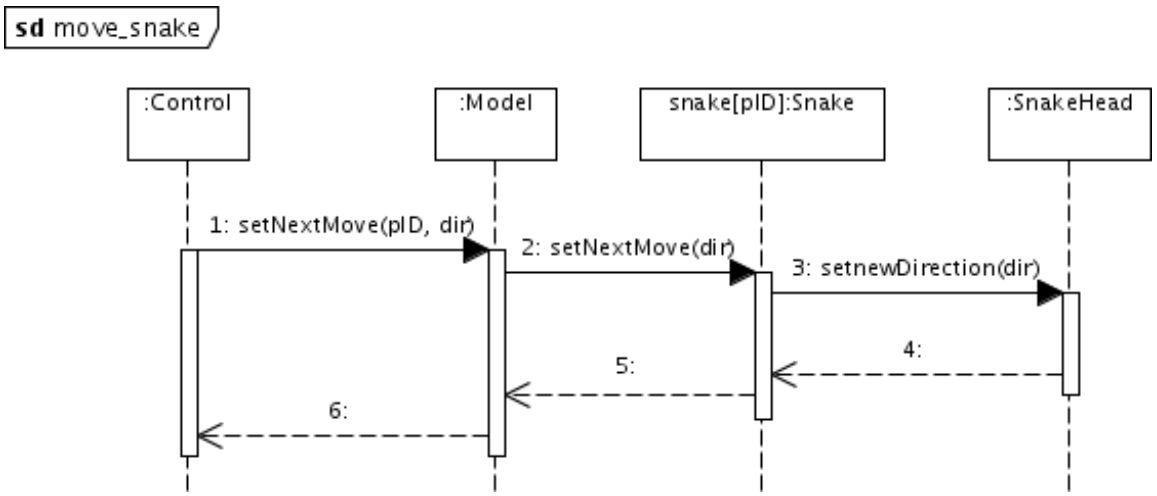
2.4. Szekvencia diagramok



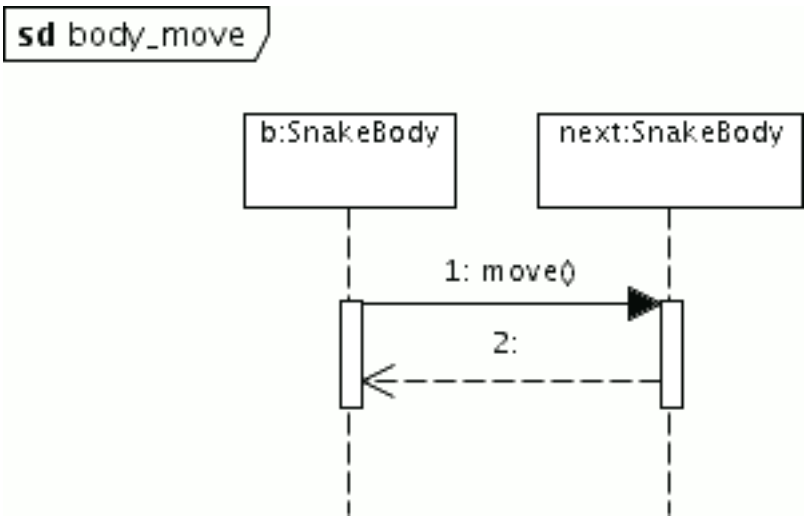
2.2. ábra. Program indítása



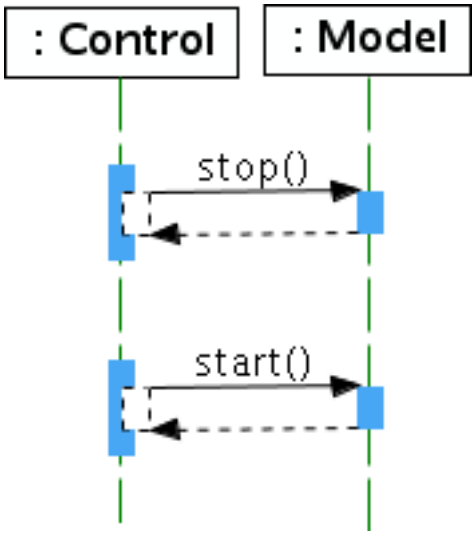
2.3. ábra. Játék indítása



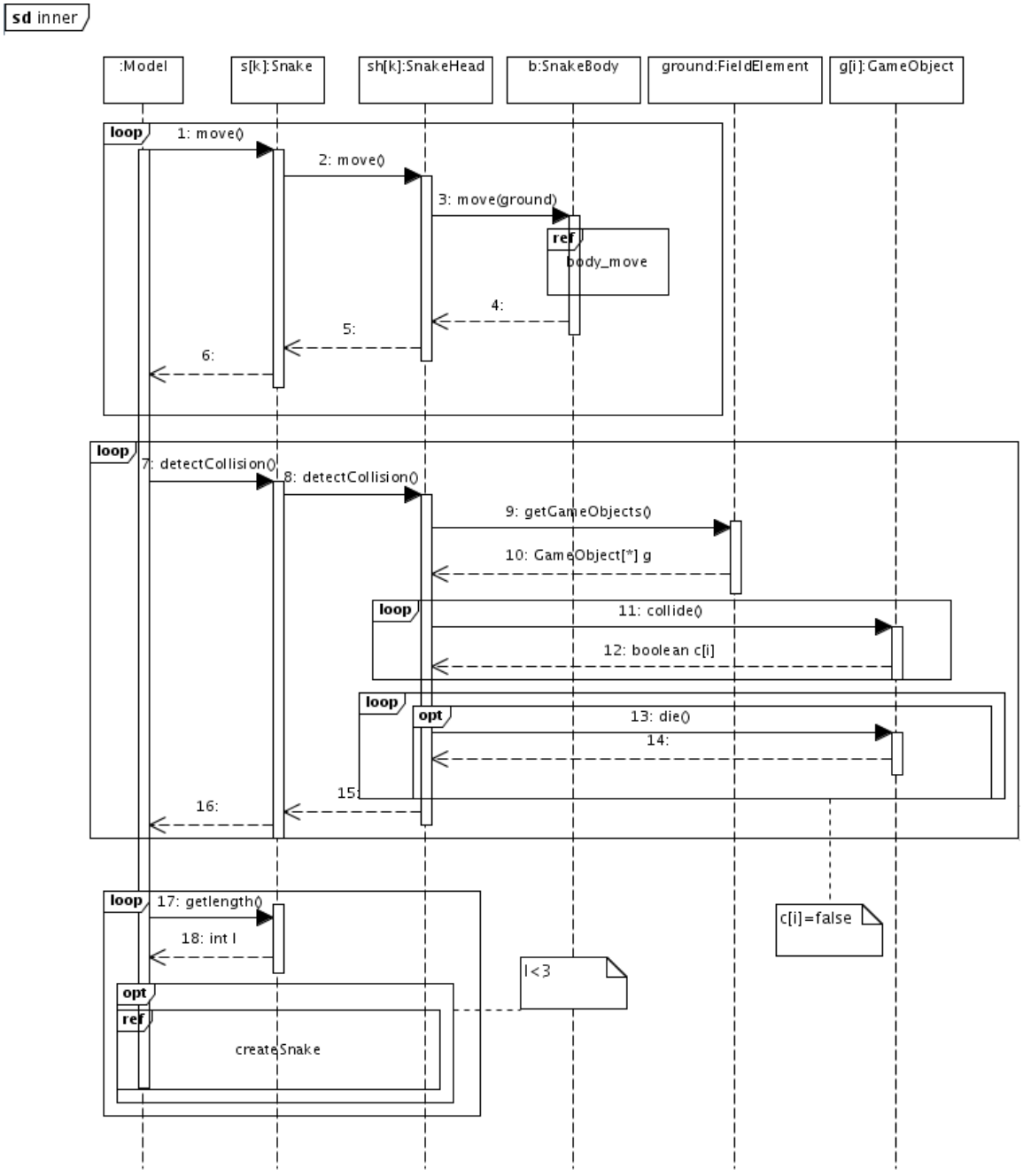
2.4. ábra. Kígyó mozgatása



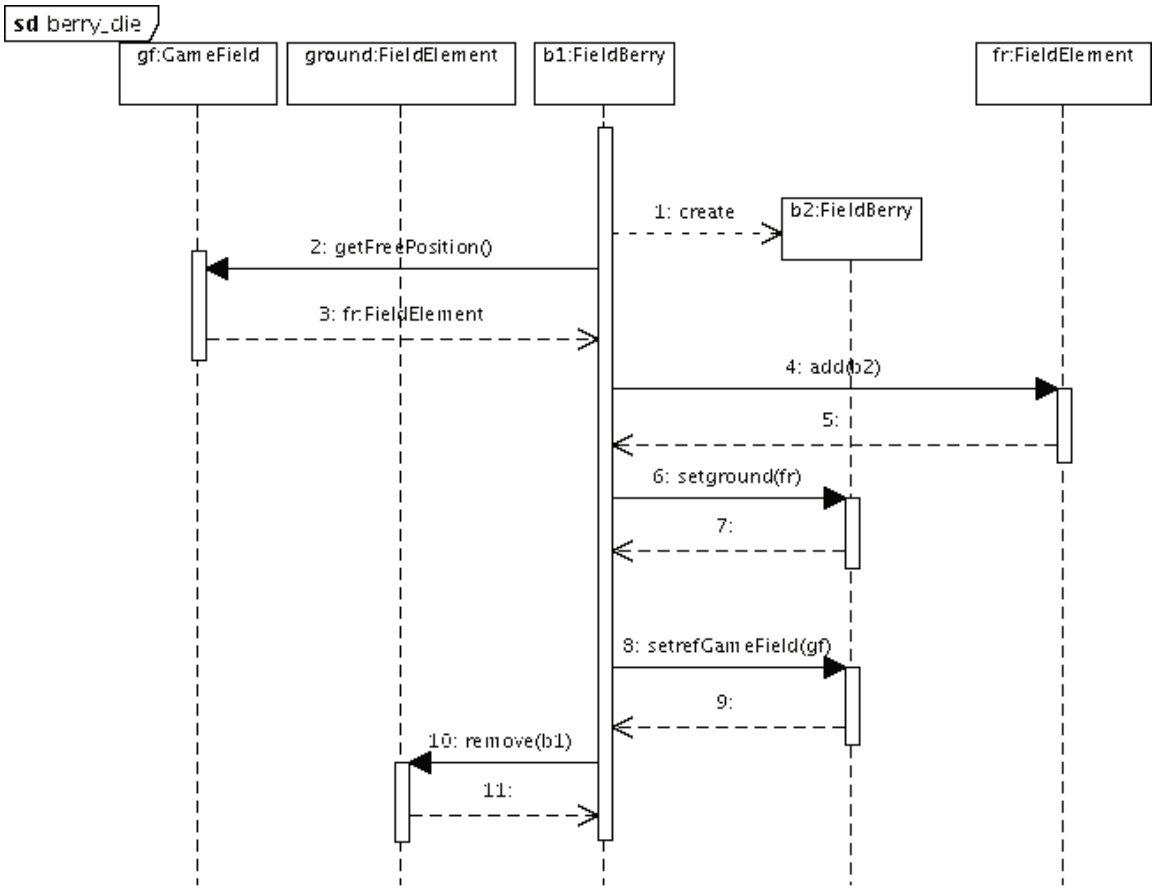
2.5. ábra. Kígyó törzsének mozgása



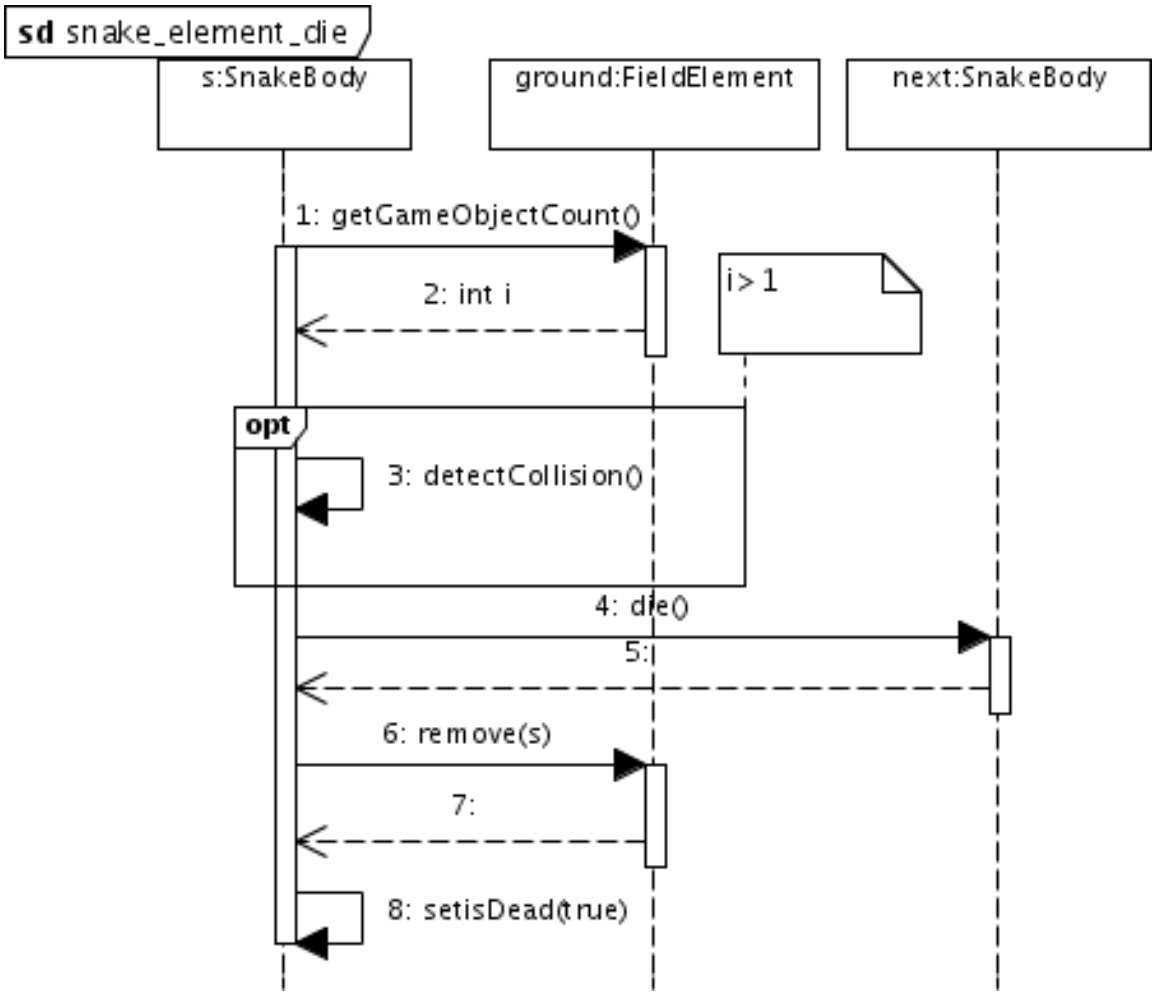
2.6. ábra. Játék szüneteltetése



2.7. ábra. Kígyók léptetése, belső működés

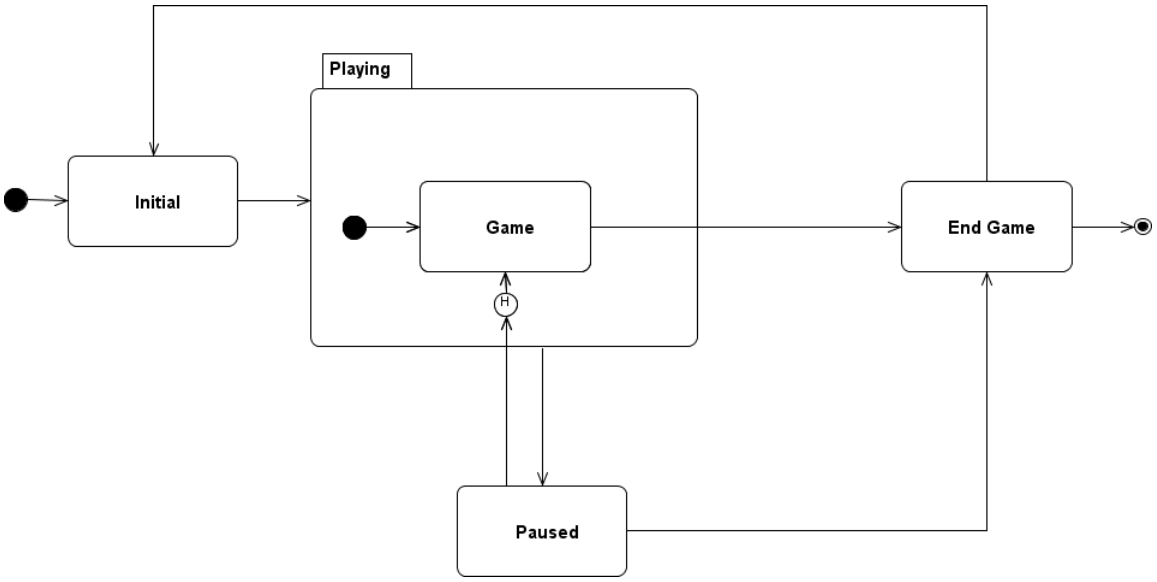


2.8. ábra. Egy bogyó halála

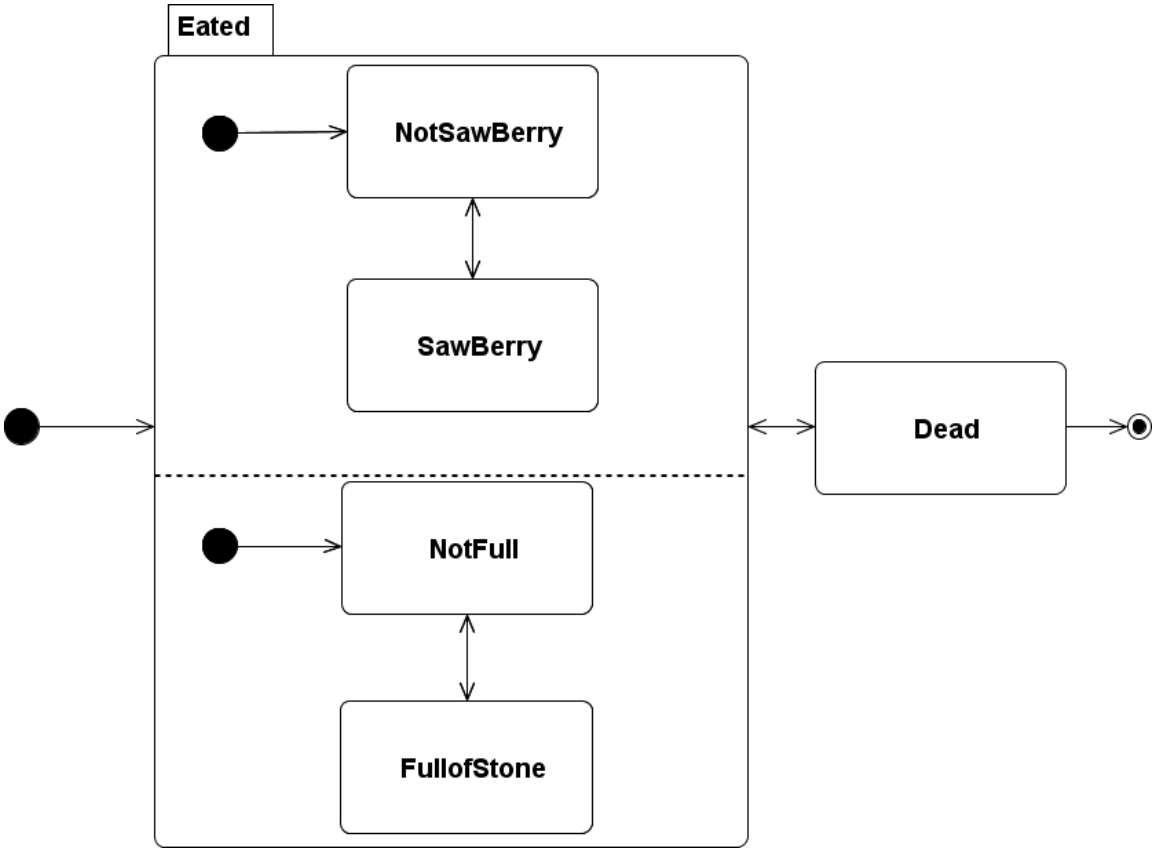


2.9. ábra. Kígyó testelem halála

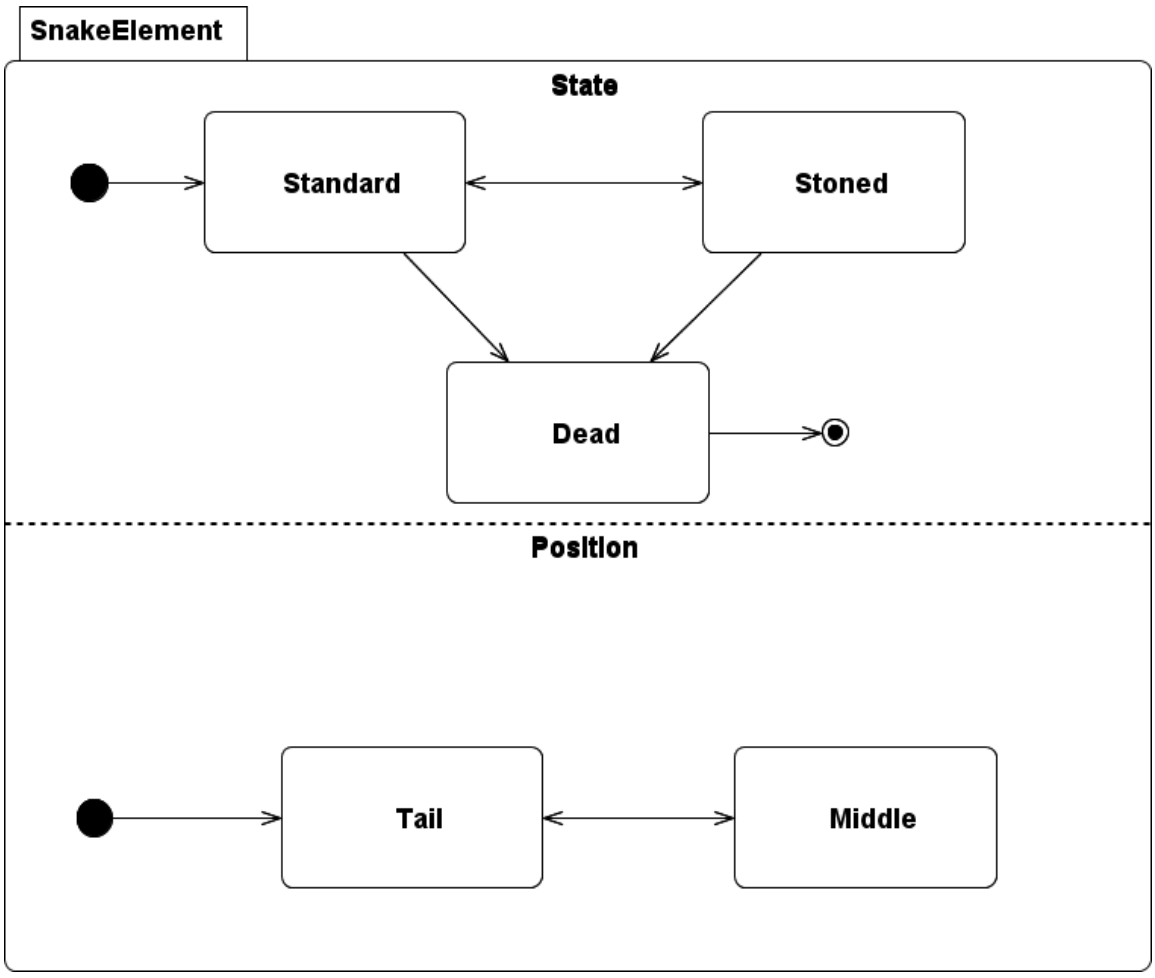
2.5. Állapot diagramok



2.10. ábra. A játék állapot diagramja



2.11. ábra. A kígyó állapotai



2.12. ábra. A kígyó testrészeinek állapotai

3. fejezet

Szkeleton tervezése

3.1. A szkeleton modell valóságos use-case-ei

3.1.1. Use-Case diagram

Lásd a 3.1. ábrát.

3.1.2. A valóságos use-case-ek leírásai

Use-Case:	Initialization
Actor:	Player
Leírás:	Program inicializálása.

Use-Case:	Start Game
Actor:	Player
Leírás:	Felhasználó új játékot indít.

Use-Case:	End Game
Actor:	Player
Leírás:	A pillanatnyilag futó játék befejezése.

Use-Case:	Pause Game
Actor:	Player
Leírás:	A felhasználó szünetelteti a játékot.

Use-Case:	Move Snake
Actor:	Player
Leírás:	Valamelyik játékos megváltoztatja a kígyója haladási irányát.

Use-Case:	Eat FieldBerry
Actor:	Player
Leírás:	A kígyó megeszik egy mezei bogyót.

Use-Case:	Eat SawBerry
Actor:	Player
Leírás:	A kígyó megeszik egy fűrészbogyót.

Use-Case:	Eat StoneBerry
Actor:	Player
Leírás:	A kígyó megeszik egy kőbogyót.

Use-Case:	FatEat StoneBerry
Actor:	Player
Leírás:	Kővel teli kígyó megeszik egy kőbogyót.

Use-Case:	Eat Snake
Actor:	Player
Leírás:	A fűrészbogyós kígyó beleharap egy sima kígyódarabba.

Use-Case:	Eat SnakeHead
Actor:	Player
Leírás:	A fűrészbogyós kígyó beleharap egy kígyó fejébe vagy nyakába.

Use-Case:	Eat SawSnake
Actor:	Player
Leírás:	A fűrészbogyós kígyó beleharap egy fűrészbogyós kígyófejbe.

Use-Case:	Wall Hit
Actor:	Player
Leírás:	A kígyó nekiütközik egy falnak.

Use-Case:	Snake Hit
Actor:	Player
Leírás:	A kígyó nekiütközik egy kígyónak.

Use-Case:	StonedSnake Hit
Actor:	Player
Leírás:	A fűrészbogyós kígyó nekiütközik egy olyan kígyódarabnak, amiben kő van.

3.1. ábra. A szkeleton valóságos use-case-ei

3.2. Architektúra

Ahhoz, hogy a játékban minden megfelelően működjön, a szkeletonban több olyan pályát valósítunk meg, amelyekkel a játékban előforduló eseteket vizsgálni tudjuk.

3.2.1. Első eset

Létrehozunk egy üres pályát. Ezzel azt vizsgáljuk meg, hogy létrejönnek-e az üres mezők. Itt teszteljük a Initialization, Start Game, End Game és Pause Game use-case-eket.

3.2.2. Második eset

Létrehozunk egy üres pályát és elhelyezünk rajta egy kígyót. A kígyó fejéhez egy kígyótestrész és egy farok csatlakozik. Ezen meg tudjuk vizsgálni a kígyó mozgását. Itt teszteljük a Move Snake use-case-t. Először megnézzük, hogy a kígyó egyenesen halad, majd az irányváltoztatásokat is ellenőrizzük.

3.2.3. Harmadik eset

Létrehozunk egy üres pályát, majd elhelyezünk rajta két kígyót, a három fajta bogyót és egy falat. Így meg tudjuk vizsgálni az alapvető ütközéseket. Itt teszteljük a Eat FieldBerry, Eat SawBerry, Eat StoneBerry, FatEat StoneBerry, Wall Hit és Snake Hit use-case-eket. Megnézzük, ahogy a kígyó rálép és felveszi a különböző fajtájú bogyókat. Megnézzük ahogy a kígyó falnak, másik kígyónak ütközik. Végül egy kővel teli kígyóval felveszünk egy kőbogyót.

3.2.4. Negyedik eset

Egy üres pályán elhelyezünk két olyan kígyót, amik fűrészbogyós állapotban vannak, és a farkukban kő van. Ezzel vizsgálni tudjuk a speciális ütközéseket. Itt tudjuk tesztelni az Eat Snake, Eat SnakeHead, Eat SawSnake és StonedSnake Hit use-case-eket. Megnézzük, ahogy az egyik kígyó rálép a másik kígyó egy darabjára, és átharapja azt. Ezután a fejébe majd nyakába harap, így láthatjuk, ahogy a kérdéses kígyó a rövidsége miatt hal meg. Aztán azt nézzük meg, ahogy a két fűrészfogas kígyó feje egyszerre lép egy mezőbe, majd mindkettő elpusztul. Végül azt vizsgáljuk meg, ahogy a fűrészfogas kígyó egy olyan mezőbe lép, ahol egy olyan kígyódarab van amiben kő található.

3.3. A szkeleton kezelői felületének terve, dialógusok

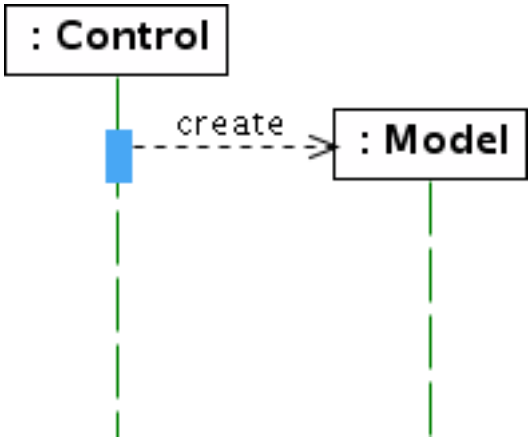
A szkeleton feladata, hogy a szekvencia diagramokban vázolt működés ellenőrizhető legyen. Ennek megfelelően a szkeleton programnak olyan speciális módon kell futnia, hogy minden belső működésre vonatkozó információt a felhasználótól kell bekérnie, ilyen eset lehet például, ha egy belső változótól függ, hogy milyen irányban fut tovább a program. Ilyenkor a CLI (Command Line Interface) segítségével beolvasott adat alapján folytatja futását a program.

A jól követhető futás érdekében minden függvényhívást – amennyiben szükséges, a függvény paraméterét is – ki kell írnia a CLI-nek, továbbá a létrejövő objektumok esetén a keletkezés tényét, valamint a keletkezett objektum nevét. Az egyes objektumoknak nem szükséges külön nevet adnunk, mivel programunk nem használ szálakat, működése nagy mértékben soros. Amennyiben egy meghívott függvény újabb függvényt hív, azt nagyobb behúzással jelöljük. A függvényen belüli közléseket ugyanakkora behúzással írjuk. A kérdés feltétele után meg kell adjuk a válaszlehetőségeket. Amennyiben a felhasználó hibás választ adott, a kérdést újra feltesszük.

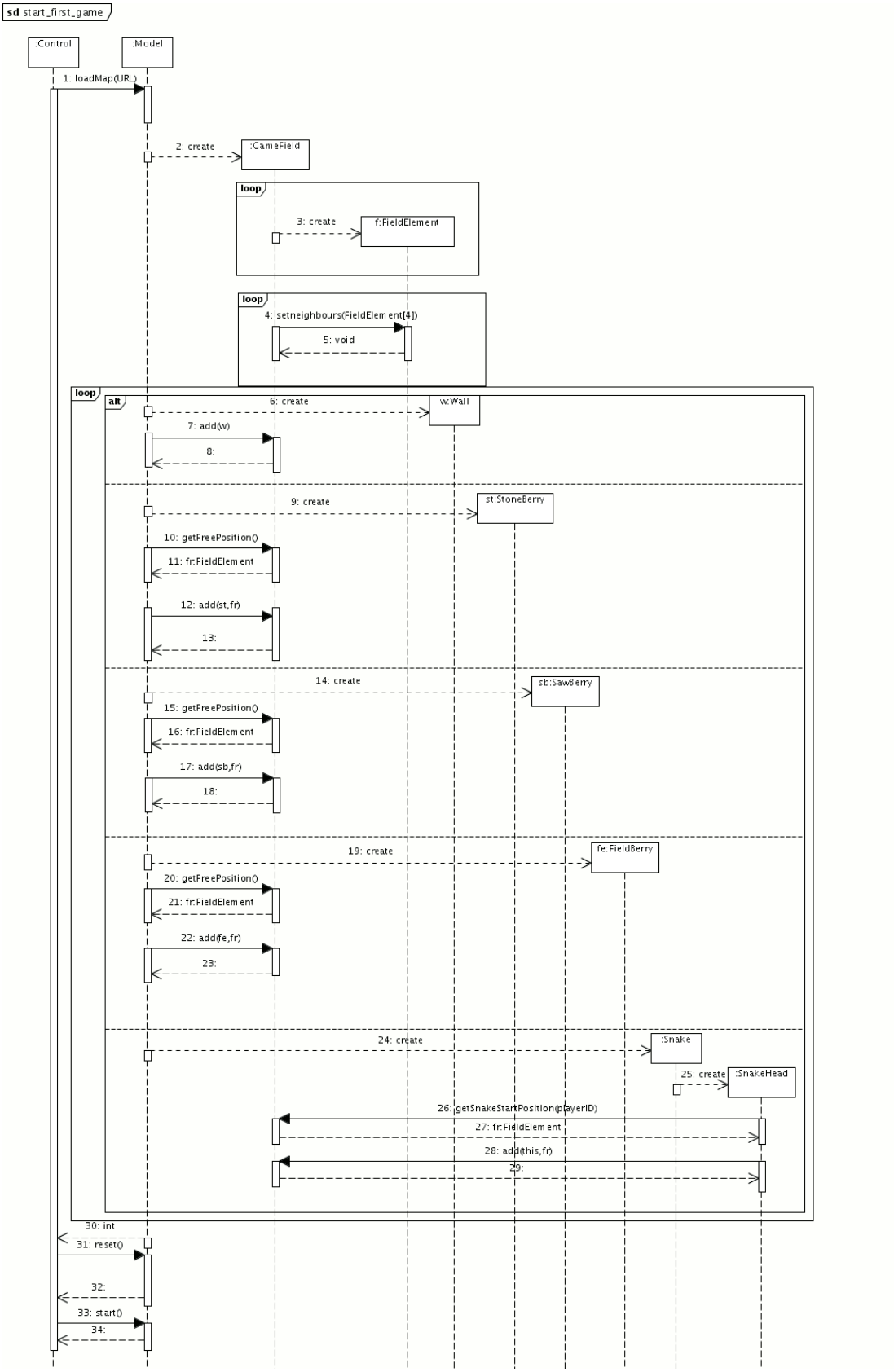
Pl.:

```
->Snake.move()
  ->SnakeHead.move()
    ->SnakeBody.move(ground)
      | body_move
    <-void
  <-void
<-void
->Snake.detectCollision()
  ->SnakeHead.detectCollision()
    ->FieldElement.getGameObjects()
    <-GameObject[] g
    ->GameObject.collide()
      | Do I survive? [true/false] false
    <-boolean[] c
    ->GameObject.die()
    <-void
  <-void
<-void
```

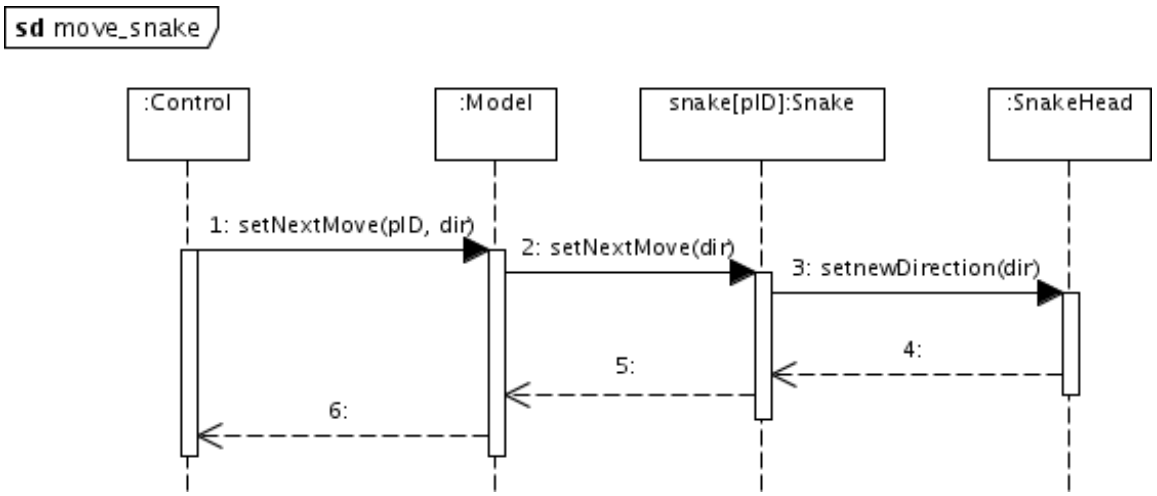
3.4. Szekvencia diagramok



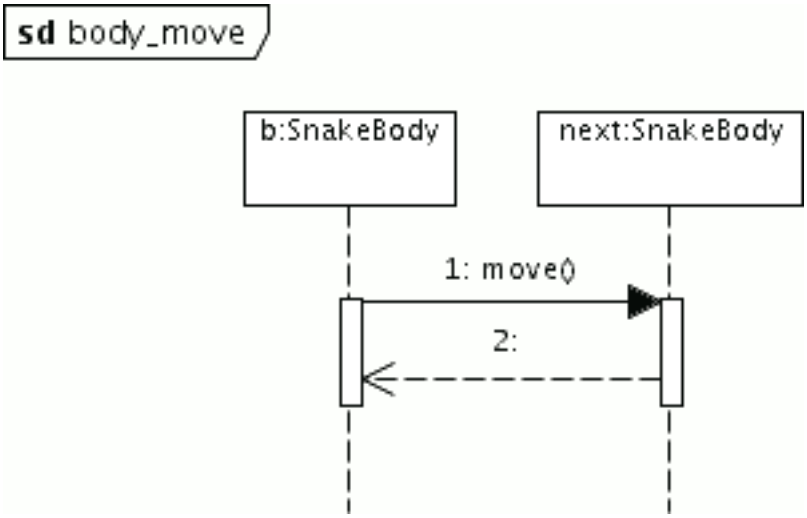
3.2. ábra. Program indítása



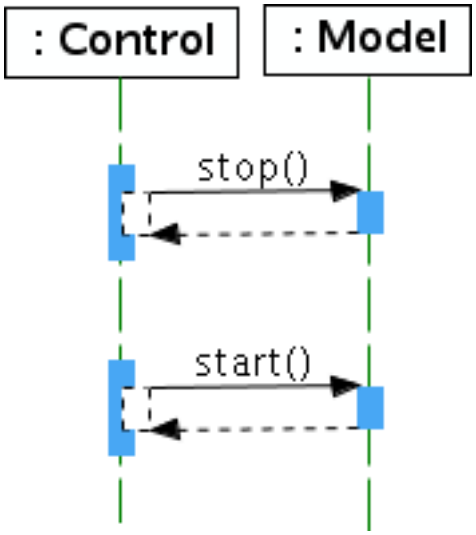
3.3. ábra. Játék indítása



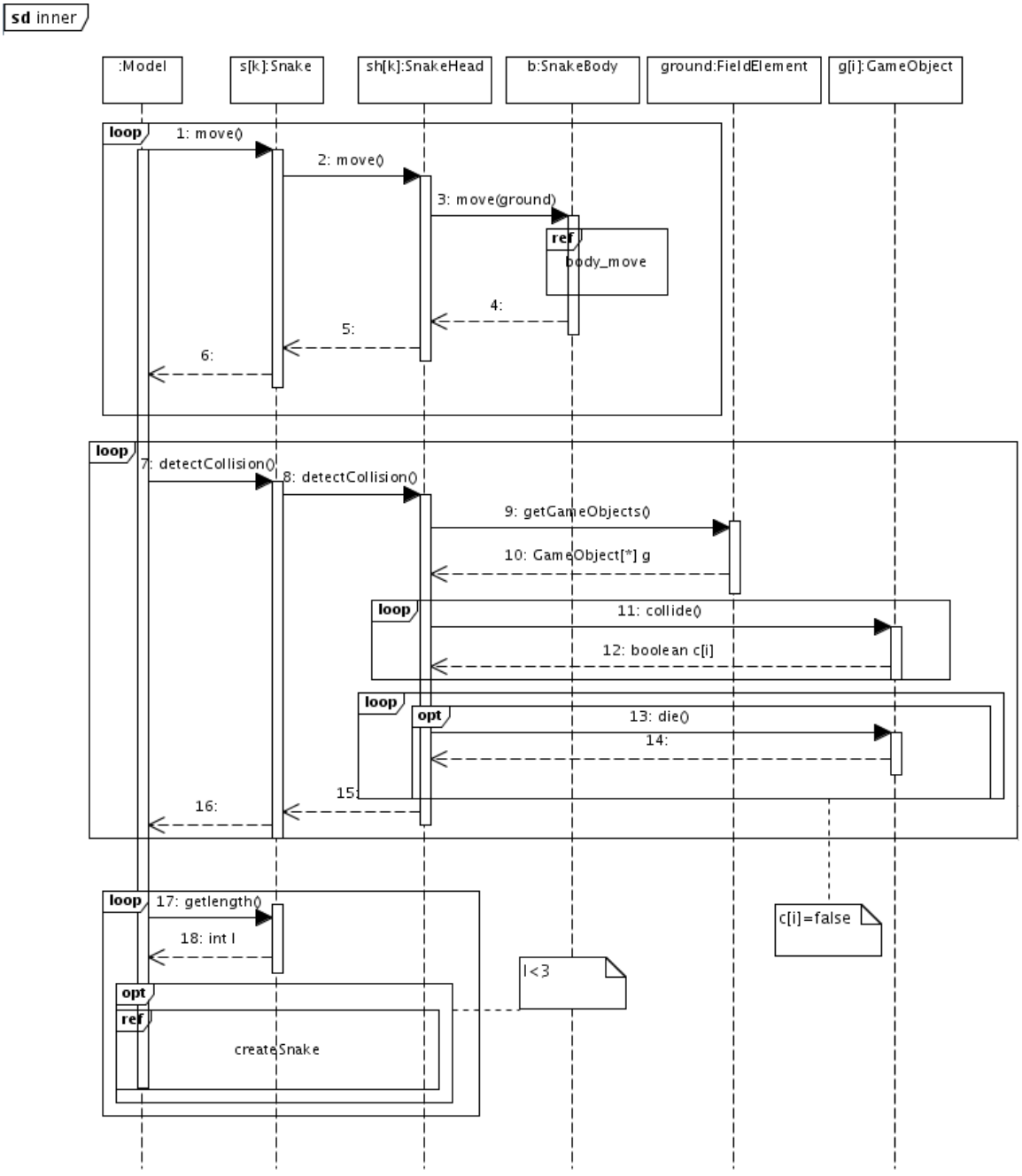
3.4. ábra. Kígyó mozgatása



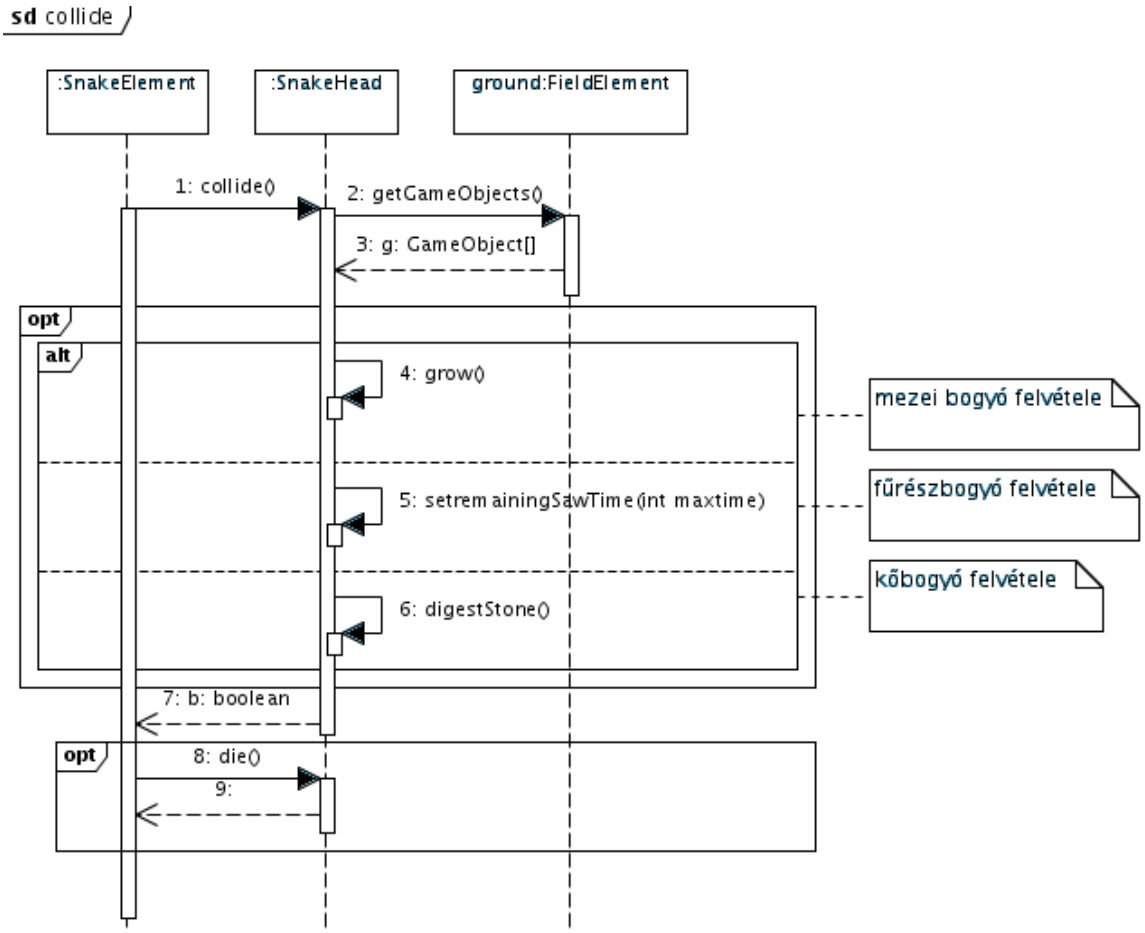
3.5. ábra. Kígyó törzsének mozgása



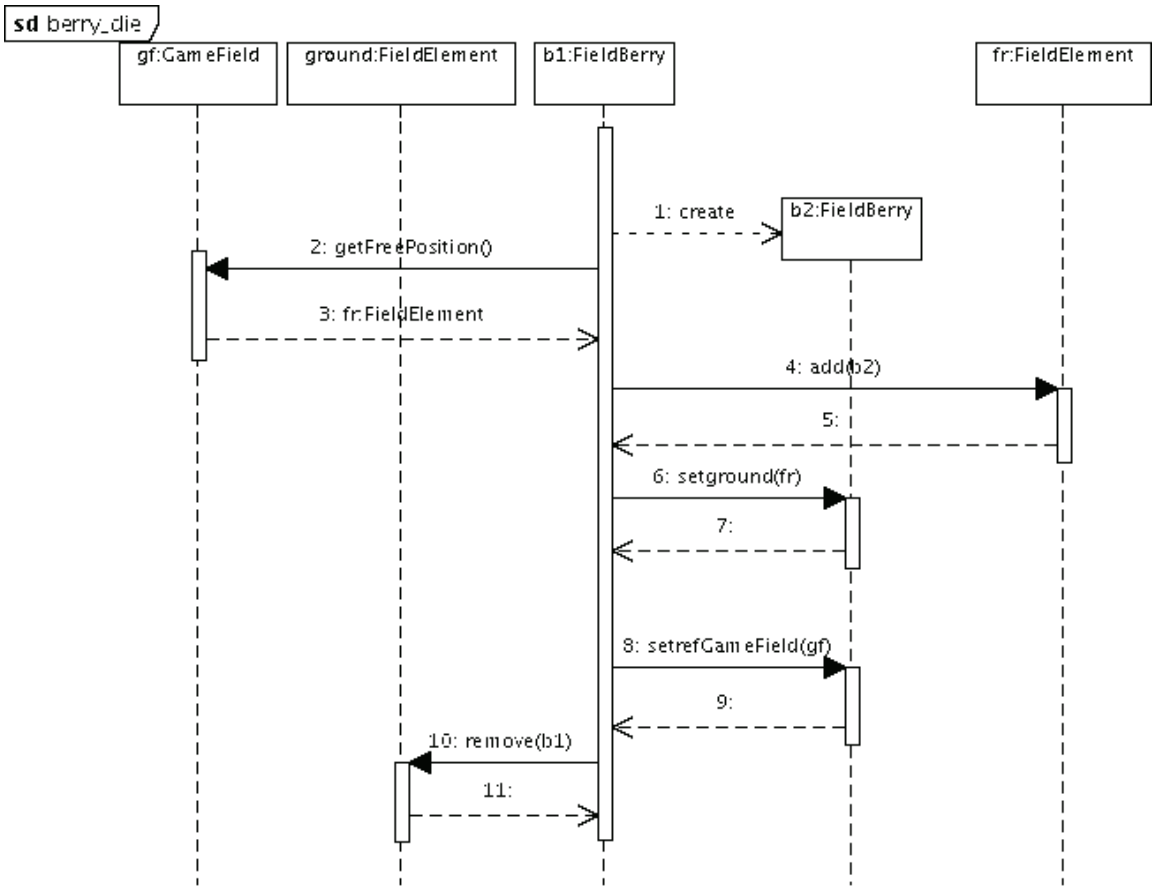
3.6. ábra. Játék szüneteltetése



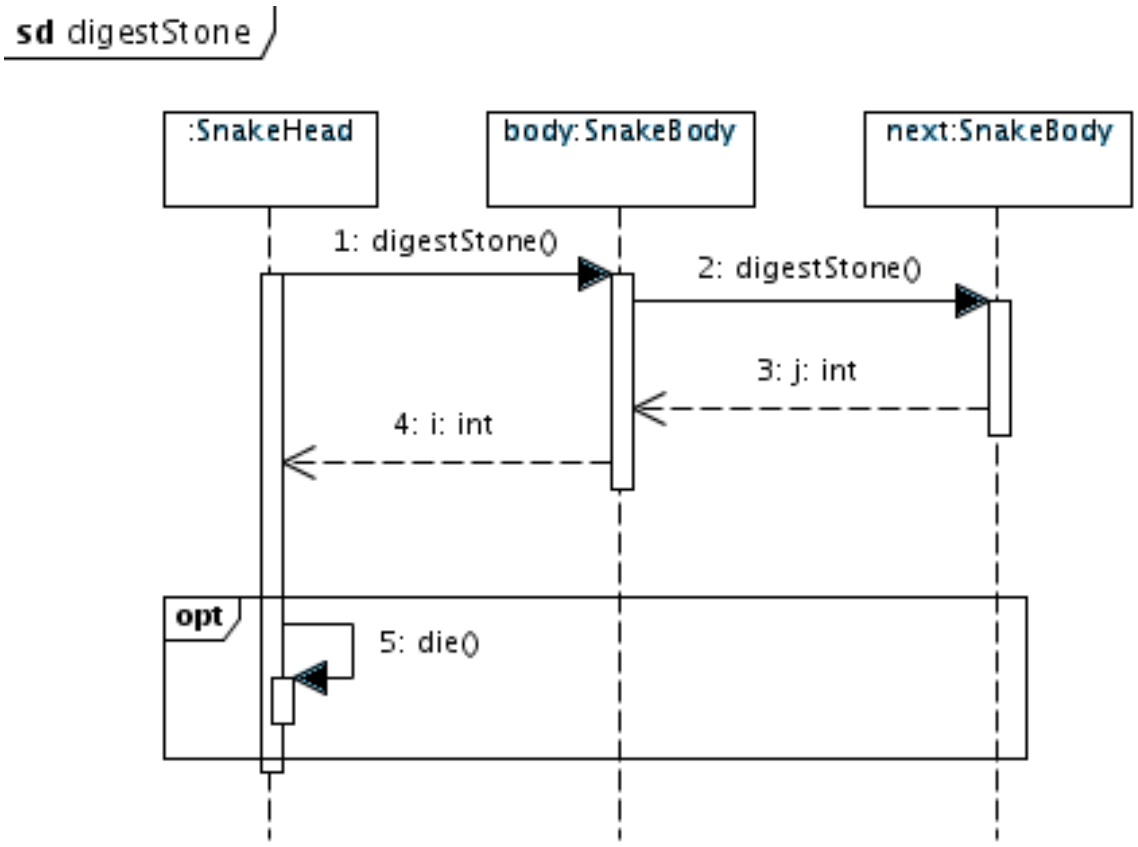
3.7. ábra. Kígyók léptetése, belső működés



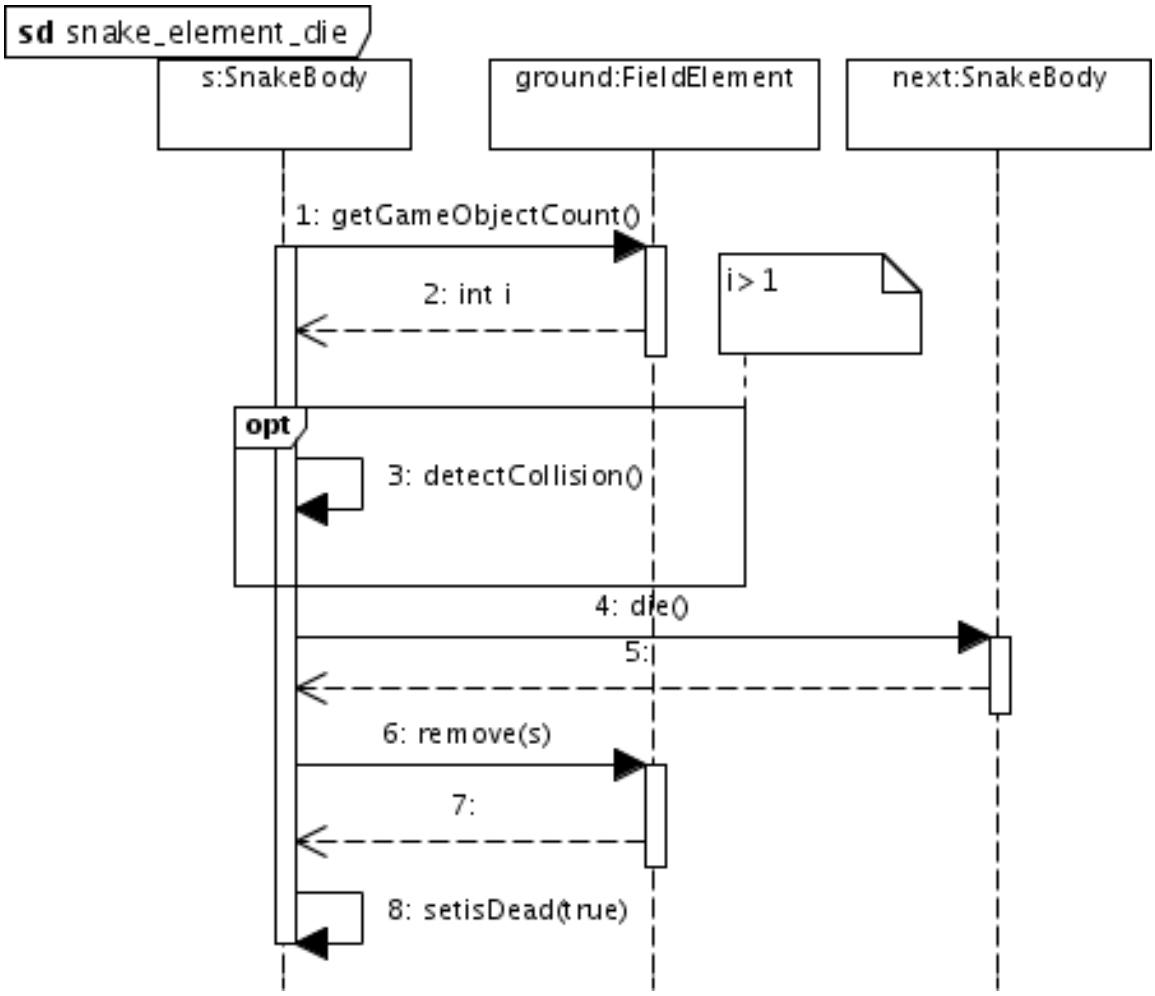
3.8. ábra. Kígyófej ütközése



3.9. ábra. Egy bogyó halála



3.10. ábra. Egy kőbogyó megemésztése



3.11. ábra. Kígyó testelem halála

4. fejezet

Szkeleton beadása

4.1. Szkeleton fordítása és futtatása

Először is a forrásfájlok eléréséhez a fájlokat ki kell tömöríteni a *ddt_szkeleton.zip*-ből, majd az operációs rendszertől függően fordíthatjuk a programot:

4.1.1. DOS parancssorból

Futtassuk a *compile.bat* fájlt, ennek hatására elkészülnek a forrásokhoz a *.class* fájlok is, így futtathatóvá válik a program.

Ezután indítsuk el a *run.bat* fájlt, ennek hatására megjelenik konzolos felületen a program, amelynek használatáról később lesz szó.

Ha a JAVA dokumentációt szeretnénk legeneráltatni akkor futtassuk a *javadocgen.bat* fájlt.

4.1.2. UNIX shell-ből

Amennyiben UNIX alapú operációs rendszerünk van, a kicsomagolási mappában gépeljük be az alábbi parancsot a kód lefordításához:

```
javac -encoding UTF8 MODEL\*.java CONTROL\*.java
      VIEW\*.java
```

A program elindításához a következő sorra lesz szükség:

```
java CONTROL.Control
```

A JavaDoc dokumentációt pedig a következő paranccsal generálhatjuk le:

```
javadoc -private -verbose -encoding UTF8 -charset UTF8 -d
        javadoc -header "<h1>DDT - Demonic Development
        Team </h1>" -windowtitle "DDT - Demonic
        DevelopmentTeam" -docencoding UTF8 CONTROL\*.java
        MODEL\*.java VIEW\*.java
```

4.2. A pálya fájl formátuma

A program az inicializálás során a pályát egy szabványos ASCII szövegfájlból olvassa be, a fájl kiterjesztése „*map*”. Ebben a fájlban minden karakter egy játékobjektumot reprezentál a következők szerint:

Karakter	ASCII kód	Játékobjektum
#	(35)	fal
X	(79)	üres terület
0-9	(48-57)	adott számú kígyó kezdőhelye

A fájlban minden sornak ugyanolyan hosszúnak kell lennie.

4.3. A szkeleton használata

A szkeleton program elindítása után egy menü fogad minket, amiben kiválaszthatjuk, hogy melyik tesztesetet (use-case) akarjuk lefuttatni. A teszteset futtatásához a neve mellett feltüntetett számot írjuk be, vagy kilépés esetén az *Exit* parancsot adjuk ki. A megfelelő teszteset kiválasztása után a program elkezdi kilistázni a futás során történt eseményeket. Ilyen esemény lehet egy függvény meghívása, melyet a

`->«`*`Objektumnév`*`».«`*`függvénynév`*`»(«`*`argumentumok`*`»)`

formátumban ír ki a program. A visszatérés formátuma:

`<-«`*`Típus`*`» «`*`név`*`».`

A függvény meghívása és visszatérése közti műveletekről, vagy egy új objektum létrejöttéről adhat tájékoztatást a *megjegyzés*, melynek formátuma:

`| «`*`Megjegyzés szövege`*`».`

Futás közben a szkeleton program gyakran szorul felhasználói beavatkozásra, mivel a függvényekben majdan implementálásra kerülő algoritmusok még nincsenek megírva. Ekkor egy kérdés jelenik meg a kimeneten, melynek formátuma hasonló a *megjegyzés*éhez, de a kérdés után szögletes zárójelen belül megjelennek a lehetséges válaszok is. A válasz megadása után a program az új információ ismeretében dönt arról, hogy melyik ágon folytatódjon a végrehajtás.

A kimenet vizsgálata során zavaró lehet, hogy egy objektum létrejöttekor először az ősosztály konstruktora hívódik meg. Ekkor úgy tűnhet, hogy több objektum keletkezett egyszerre, de valójában csak egy objektum jött létre. Például, ha egy SnakeHead objektumot hozunk létre, ezt a kimenetet kapjuk:

```
| A GameObject objektum létrejött
| A SnakeElement objektum létrejött
| A SnakeHead objektum létrejött
```

A felhasználói felület: a menü és a CLI függvényeit nem írattuk ki a kimenetre, mert ezek a működés szempontjából nem lényegesek.

4.4. Értékelés

Az eddigi feladatok során a csapat minden tagja egyformán jól teljesített. Még ha az egyes beadásoknál nem is volt teljesen egyenlő a munkamegosztás, összességében arra törekedtünk, hogy mindenki ugyanannyit dolgozzon, és ez sikerült is. A tagok egyhangúan amellett döntöttek, hogy mindenki egyformán részesüljön a pontokból.

A projekt során a tagok közötti kommunikációval nem volt gond, így a gyakori megbeszéléseknek hála, gyorsabban voltunk képesek teljesíteni a kitűzött feladatokat.

A csapat összeszokásával nem volt probléma, mert már korábban is dolgoztunk egy közös projekten. A feladatok kiosztása mindig gyorsan és hatékonyan ment végbe. Az információáramlással sem volt egyszer sem probléma, mindig tudta mindenki, hogy mikorra mit kell elvégeznie, illetve tisztában volt a többiek által végzett feladatokkal is, olyan mélységig, mintha ő csinálta volna. Ha néha akadt is nézeteltérés, kompromisszumokkal hamar megoldottuk a problémát.

Név	**Elvégzett munkahányad**
Csuzdi	25%
Fehér	25%
Györgyey^{CSK}	25%
Major	25%

4.5. Melléklet

4.5.1. A fájlok listája

A *ddt_szkeleton.zip* tartalma:

Utolsó módosítás	Idő	Fájlméret (byte)	Fájlnév	Leírás
2008-03-18	23:45:13	2	1.map	Pálya
2008-03-19	02:49:12	71	2.map	Pálya
2008-03-19	02:49:12	71	3.map	Pálya
2008-03-19	02:49:12	71	4.map	Pálya
2008-03-19	05:41:42	1851	Berry.java	Berry osztályt tartalmazza.
2008-03-17	18:31:21	4621	CLI.java	CLI osztályt tartalmazza.
2008-03-19	06:25:40	62	compile.bat	Fordító szkript.
2008-03-19	05:41:58	7635	Control.java	Control osztályt tartalmazza.
2008-03-19	00:54:18	455	Direction.java	Direction osztályt tartalmazza.
2008-03-18	21:28:03	323	FieldBerry.java	FieldBerry osztályt tartalmazza.
2008-03-19	05:41:42	4115	FieldElement.java	FieldElement osztályt tartalmazza.
2008-03-19	05:41:42	5158	GameField.java	GameField osztályt tartalmazza.
2008-03-19	00:54:18	1516	GameObject.java	GameObject osztályt tartalmazza.
2008-03-19	00:54:18	508	Int2.java	Int2 osztályt tartalmazza.
2008-03-19	06:28:28	222	javadocgen.bat	Dokumentációt generáló szkript.
2008-03-19	05:44:57	10489	Model.java	Model osztályt tartalmazza.
2008-03-19	06:26:50	22	run.bat	Futtató szkript.
2008-03-19	02:34:16	307	SawBerry.java	SawBerry osztályt tartalmazza.
2008-03-19	05:41:43	1107	SnakeBody.java	SnakeBody osztályt tartalmazza.
2008-03-19	05:44:57	6847	SnakeElement.java	SnakeElement osztályt tartalmazza.
2008-03-19	05:41:43	4042	SnakeHead.java	SnakeHead osztályt tartalmazza.
2008-03-19	04:11:34	3342	Snake.java	Snake osztályt tartalmazza.
2008-03-18	21:28:03	332	StoneBerry.java	StoneBerry osztályt tartalmazza.
2008-03-19	02:34:16	678	Wall.java	Wall osztályt tartalmazza.

5. fejezet

Prototípus koncepciója

5.1. Prototípus interfész definíciója

Ebben a részben definiáljuk a program bemeneti, illetve a kimeneti formátumát, vagyis azt a felületet, amivel ellenőrizni tudjuk, hogy a program a specifikációnak megfelelően működik.

5.1.1. A pálya fájl formátuma

A program az inicializálás során a pályát egy szabványos ASCII szövegfájlból olvassa be, a fájl kiterjesztése „*map*”. Ebben a fájlban minden karakter egy játékobjektumot reprezentál a következők szerint:

Karakter	ASCII kód	Játékobjektum
#	(35)	fal
X	(79)	üres terület
0-9	(48-57)	adott számú kígyó kezdőhelye
k	(107)	kőbogyó helye
m	(109)	mezei bogyó helye
f	(102)	fűrészbogyó helye

A fájlban minden sornak ugyanolyan hosszúnak kell lennie.

5.1.2. A bemeneti interfész

A program futtatásakor argumentumokat adhatunk meg. Így a bemenet két helyről is érkezhet:

konzolról Amennyiben nem adunk meg futtatási argumentumot, akkor a program konzolról várja a parancsokat.

fájlból Ez esetben az argumentum azt mondja meg, hogy melyik fájlt olvassuk be, majd végrehajtja az abban megadott utasításokat.

Mind a kettőnél azonos leírásmódot használunk, melyet részletesebben az 5.4. fejezetben tárgyalunk.

5.1.3. A kimeneti interfész

Az előzőek alapján a kimenet is kétféleképpen alakulhat:

konzol módban A konzolon csak az aktuális pálya ASCII karakteres megjelenítését látjuk, ami a lépések hatására frissül folyamatosan. A főbb eseményekről szóló tájékoztatásokat pedig egy fájlba írjuk ki. Ebben az esetben a teszt akkor lesz sikeres, ha az ellenőrző személy nem észlelt hibát a megjelenítés során.

fájl módban Ebben a módban a kimenet két fájlban jelenik meg. Az egyik fájl a pályák grafikus tartalmait, a másik pedig a főbb részfolyamatait tartalmazza. A teszt során két referenciafájlt hasonlítunk össze a teszt kimenetével, amennyiben nincsen különbség a fájlok között, sikeres volt a teszt.

A kimeneti fájlok a bemeneti fájl nevével azonos *.out*, illetve *.gout* kiterjesztésű elnevezést kapnak majd.

5.1.4. A beállításokat tartalmazó fájl

Ebben a fájlban tároljuk el a fontosabb beállításokat, úgymint például a kígyó sebessége, vagy a játék hossza. Az egyes beállításokhoz tartozó sorok a következő formátumúak lesznek:

`Név␣=␣Érték`

Ahol a `Név` a paraméter szóközök nélküli neve, az `Érték` pedig a hozzá tartozó mennyiség. Minden paramétert külön sorba írunk. Ezen kívül lehetőség van komment hozzáfűzésére is, amelyet a sor elejére tett '#' karakterrel tudunk jelezni a program számára.

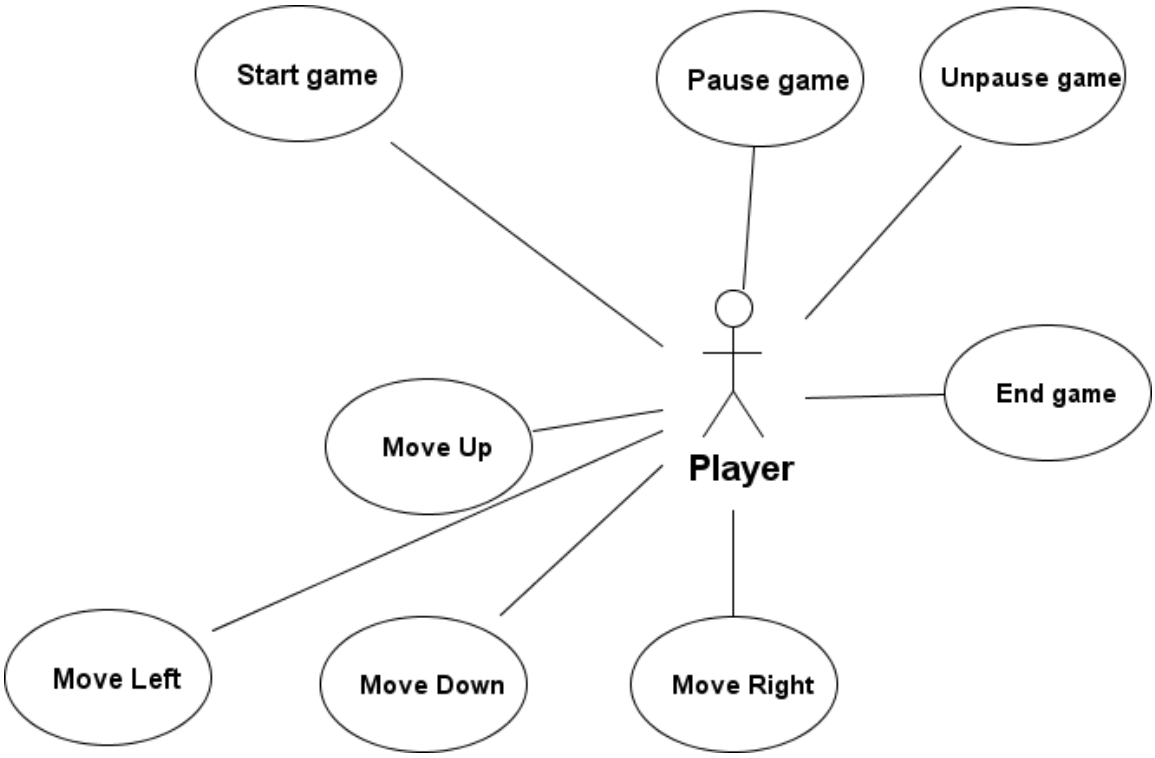
5.1.5. A kimenet specifikálása

A program kimenete a parancsok tekintetében az 5.1 táblázat alapján alakul majd.

Parancs	Kimenet
uj_jatek «argumentum»	Palya sikeresen betoltve.
leptet	Kigyok leptetese.
szunet	Jatek szunetel..
folytatas	Jatek folytatodik.
«irány» «kígyó azonosító»	«Nincs kimenet»
«mezei bogyóra lépés»	«kígyó azonosító»-s kigyo mezei bogyot vett fel.
«fűrészbogyóra lépés»	«kígyó azonosító»-s kigyo fureszbogyot vett fel.
«fűrészbogyó hatása elmúlik»	«kígyó azonosító»-s kigyo fureszfoga megszunt.
«kőbogyóra lépés»	«kígyó azonosító»-s kigyo kobogyot vett fel.
«kígyó meghalt»	«kígyó azonosító»-s kigyo meghalt.
	Uj «kígyó azonosító»-s kigyo jott letre.
«átharapás»	Uj zombi kigyo jott letre.
vege	Jatek vege.

5.1. táblázat. Az egyes parancsokhoz tartozó kimenetek

5.2. Részletes use-case-ek



5.1. ábra. Use-case diagram

Use-Case:	Start game
Actor:	Player
Leírás:	A felhasználó új játékot indít. Létrejön a pálya, elhelyezkednek rajta a kígyók.

Use-Case:	Pause game
Actor:	Player
Leírás:	A felhasználó szünetelteti a játékot.

Use-Case:	Unpause game
Actor:	Player
Leírás:	A felhasználó folytatja a szüneteltetett játékot.

Use-Case:	End game
Actor:	Player
Leírás:	A pillanatnyilag futó játék befejezése.

Use-Case:	Move Left
Actor:	Player
Leírás:	A játékos megváltoztatja a kígyója haladási irányát balra, ha a kígyó épp nem jobbra halad.

Use-Case:	Move Up
Actor:	Player
Leírás:	A játékos megváltoztatja a kígyója haladási irányát felfelé, ha a kígyó épp nem lefele halad.

Use-Case:	Move Right
Actor:	Player
Leírás:	A játékos megváltoztatja a kígyója haladási irányát jobbra, ha a kígyó épp nem balra halad.

Use-Case:	Move Down
Actor:	Player
Leírás:	A játékos megváltoztatja a kígyója haladási irányát lefelé, ha a kígyó épp nem felfele halad.

5.3. Tesztelési terv

5.3.1. A tesztelés menete

A teszteléshez előre létrehozunk tesztelési forgatókönyveket, melyek magukban foglalják a játék során előforduló összes esetet. A tesztelés során ezeket a forgatókönyveket futtatjuk le, és megvizsgáljuk a bemenet alapján, hogy a kapott eredmény megfelel-e az elvárásainknak. Megbízható eredmény érdekében a teszteket többször is le fogjuk futtatni.

5.3.2. A tesztforgatókönyvek

Inicializálás

- Játék indítása.
- Pálya létrehozása, bogyók és kígyók elhelyezése után a játék szüneteltetése.
- Játék folytatása.
- Játék befejezése.

Mozgás Mindkét kígyóra:

- Irány beállítása balra és két léptetés.
- Irány beállítása lefele és két léptetés.
- Irány beállítása jobbra és két léptetés.
- Irány beállítása felfele és két léptetés.

Bogyók felvétele

- A kígyó felvesz egy mezei bogyót.
- A kígyó felvesz egy fűrészbogyót.
- A kígyó felvesz egymás után (nem közvetlenül) négy kőbogyót.

Ütközések

- A kígyó egy kígyódarabnak ütközik.
- A kígyó falnak ütközik.

Fűrészbogyós esetek

- Kígyó felvesz egy fűrészbogyót.
- Fűrészbogyós kígyófej sima kígyódarabra fut és új kígyó jön létre a levágott részből.
- Fűrészbogyós kígyófej kígyónyakra fut.
- Másik kígyó felvesz egy fűrészbogyót.
- Fűrészbogyós kígyófej fűrészbogyós kígyófejre fut.
- A kígyó felvesz egy fűrészbogyót, a másik kígyó pedig egy kőbogyót.
- Fűrészbogyós kígyófej olyan kígyódarabnak ütközik amiben kő van.
- Fűrészbogyós állapot letelik.

5.4. Tesztelő nyelv

A tesztelő nyelv a tesztelés során a programnak adható parancsokat, és azok argumentumait határozza meg. A nyelv megalkotása során oda kell figyelnünk arra, hogy minden, a tesztesetekben meghatározott funkciót meg lehessen hívni parancsokkal. Így tehát a parancsok a felhasználó billentyűleütését, valamint a majdani menüből kiválasztható funkciók aktiválását emulálják. A kígyók irányának beállítása után a `leptet` paranccsal léptethetjük a kígyókat.

Az átláthatóbb irányítás érdekében minden parancshoz tartozhat egy szám – amit közvetlenül a parancs neve elé írunk, és egybeírjuk a parancs nevével –, ami megadja, hogy hányszor ismételjük az adott parancsot (ugyanazokkal az argumentum értékekkel). Pl.: „`5jobbra 1`” jelentése: Az 1-es számú játékos leüti a jobbra billentyűt 5-ször. Az argumentumokkal rendelkező parancsok argumentumait mindig kötelező megadni, s hasonlóan, ha nincs argumentuma egy parancsnak, akkor nem szabad megadni semmit, különben a feldolgozó logika hibaüzenettel kilép.

A tesztelő nyelv parancsait „*.test*” kiterjesztésű fájl(ok)ban tároljuk. A fájlon belül a #(kettőskereszt)-tel kezdődő sorok kommentnek számítanak, nem kerülnek feldolgozásra. Egy sorban több parancsot is megadhatunk egyszerre, a parancsokat szóközzel elválasztva. A parancsokat és argumentumaikat az 5.2. táblázatban foglaltuk össze.

Név	Argumentumok	Leírás
[n]jobbra	«playerID»	Az adott játékos irányának beállítása.
[n]balra	«playerID»	Az adott játékos irányának beállítása.
[n]fel	«playerID»	Az adott játékos irányának beállítása.
[n]le	«playerID»	Az adott játékos irányának beállítása.
[n]leptet	–	Kígyók léptetése.
[n]uj_jatek	«filename»	Új játék kezdése a filename nevű pályán.
[n]szunet	–	Játék szüneteltetése.
[n]folytatas	–	Szüneteltetett játék folytatása.
[n]vege	–	Játék vége.

5.2. táblázat. Parancsszavak és argumentumaik

Példa egy parancsfájlra:

```
#új játék, palya1.map térkép fájl használatával
uj_jatek palya1.map
#mindkét kígyónak megadunk haladási irányokat
jobbra 0 jobbra 1 balra 0 balra 1 le 0 le 1
#léptetjük a kígyókat az utolsó megadott irány alapján
leptet
#játék szüneteltetése
szunet
#pizza elfogyasztása
folytatas
leptet
#kilépünk
vege
```

5.5. Tesztelést támogató segédprogramok leírása

5.5.1. Tesztelő szkript

A tesztelést első körben egy szkript végzi, amely meghívja a programot a megfelelő parancsfájlokat adva argumentumként, majd a program által generált kimenetet összehasonlítja az elvárt kimenettel. Ha nem talált különbséget, akkor sikerült a teszt. Ugyanezt végigjátssza az összes tesztesetre.

5.5.2. Mydiff

A fentiek alapján szükségünk van egy két fájlt összehasonlító programra (nevezzük `mydiff`-nek), melynek visszatérési értéke alapján eldönthetjük, hogy egyezett-e a két fájl, ha különbség volt, megmondja a program, hogy melyik sorban, hányadik karakternél. A programnak két argumentuma van: a két fájl neve.

5.5.3. Snapshotviewer

A képernyőképek könnyebb követhetősége érdekében szükségünk van még egy, a karakteres „képeket” kérésre egymás után megjelenítő programra is. Ez egy egyszerű grafikus felületű program, amely az argumentumban megadott szövegfájlt feldolgozva a képet reprezentáló karaktertömböt betölti egy szövegdobozba, és azt a léptetőgomb minden megnyomásakor a következő „képre” frissíti.

5.6. Módosítások

5.6.1. Hibák javítása

A szkeleton készítése során adódtak kisebb problémák a modellel, ezeket orvosoltuk:

- Az irányváltoztató függvényekben az eddigi szimbolikus *Direction* típust *int*-re cseréltük, és létrehoztunk egy új *Direction* osztályt, mely *int* típusú konstansokat tartalmaz az egyszerűség és átláthatóság kedvéért.
- A *Snake* osztály kapott egy új *die()* függvényt. Erre azért volt szükség, hogy a modell el tudja távolítani a halálosan megrövidült kígyókat és újra tudja teremteni őket.
- A *SnakeHead body* nevű és a *SnakeBody next* nevű, *SnakeBody* típusú referenciái megszűntek, helyettük az ősosztályba, a *SnakeElement*-be került egy *SnakeBody* típusú referencia, hogy implementálhatóak legyenek a *SnakeElement* azon függvényei, melyeket már nem definiálunk felül a leszármazott osztályokban.

5.6.2. A specifikáció változásainak értelmezése

"1. A leharapott kígyófarok nem pusztul el, hanem egy új kígyóvá alakul (amely kezdetben NE a harapással őt létrehozó kígyófej irányába mozogjon!)"

Az ily módon létrejött kígyók "zombi" kígyók, azaz mozgási irányuk az őket létrehozó kígyó mozgási irányának éppen az ellenkezője lesz, és a későbbiekben sem irányíthatóak a játékosok által, tehát ezt a mozgási irányukat megtartják. A modell minden léptetésekor rendes kígyó módjára lépnek, felvehetnek bogyókat, ütközhetnek, haraphatnak, azonban halál esetén nem teremnek úgy újra, mint a játékosok kígyói, hanem végleg elpusztulnak.

"2. A kígyóban a kőbogyók vándorlási sebessége eltérhet (nagyobb vagy kisebb is lehet!) a kígyó mozgási sebességétől."

A kőbogyók kígyóbeli vándorlási sebessége globálisan azonos (tehát egyik kígyó sem emészt gyorsabban mint a másik, és egyik kőbogyó sem halad gyorsabban mint a másik), valamint a játék során sem változik. Értéke az inicializáláskor dől el a konfigurációs fájlból beolvasott érték alapján.

5.6.3. A specifikáció változásának hatásai

A feladat specifikációjának megváltozása miatt szükségessé vált a modell kisebb mértékű módosítása:

- A *SnakeElement* osztály több módosítást is igényelt. Az osztály kapott egy statikus referenciát a *Model* objektumra, hogy az új kígyó létrehozásának szükségességét jelezhesse a modellnek. Az új *becomeNewSnake()* függvény hozza létre átharapás esetén az új kígyófejet a testelem helyére és végzi el a szükséges referencia beállításokat. Ez a függvény a *die()* függvényben hívódik meg, a meghalás alternatívájaként, amennyiben az új *boolean becomingHead* attribútum igaz. Ezt az attribútumot a *collide()* függvényben állítjuk igazra átharapás észlelésekor.

- A *Model* osztályban szükséges egy új függvény, a *createNewSnake(SnakeHead sh)*, amely az átharapás során keletkezett új kígyót felveszi a modellbe.
- A *Snake* osztálynak kell egy új konstruktor, a *Snake(int playerID, SnakeHead sh)*, mely nem hoz létre új testrészeket, pusztán a paraméterként kapott kígyófejet veszi irányítása alá.
- A *Snake* osztályban szükséges egy új, *double* típusú attribútum a vándorlási sebesség kezeléséhez. Ennek az egészrésze fogja meghatározni, hogy az adott lépésben hányszor hívódjon meg a *digestStone()* metódus, azaz hány szegmenssel kerüljenek hátrébb a kőbogyók.

6. fejezet

Részletes tervek

6.1. Objektumok és metódusok tervei

Control

«public» **Control** ()
Konstruktor.

«public static» **void main** (String[] args)
A program belépési pontja. Paraméterként kaphat egy tesztfájl nevet, melyet lefuttat, az eredmény fájlba íródik. Amennyiben nem kapott tesztfájlt, a bemenő parancsokat a konzolról várja és végzi el a *Model* megfelelő függvényeinek meghívásával.

Berry

«package» **Berry** ()
Konstruktor.

«package» **boolean collide** ()
Ütközéskezelő függvény. *True*-val tér vissza, ha a bogyó túlélte az ütközést. A vizsgálathoz lekéri az adott *FieldElement*-re lépett objektumok listáját annak *getGameObjects*() függvényével, és azok (illetve azok tulajdonságai) alapján dönti el az ütközés kimenetelét.

«package» **void die** ()
A bogyó megöléséért, és egy új, ugyanilyen bogyó véletlenszerű új helyre létrehozásáért felelős függvény. Saját halálát a *FieldElement*-jéről való *remove*() hívással éri el, míg az új bogyót egy a *GameField*-től a *getFreePosition*() függvényben megkapott szabad helyre teszi annak *add*() metódusával. Ezek után még átadja a pálya referenciáját az új bogyónak a *setrefGameField*() hívással.

«package» **void setrefGameField** ()
Beállítja a kapott értékre a bogyó pályareferenciáját.

Direction

«public» **Direction** ()
Default konstruktor. A *Direction* egy irány osztály, mely négy darab public *final static int* adattagot tartalmaz az irányok használatának megkönnyítése érdekében. Ezek (és értékeik) az alábbiak: *UP*(0), *RIGHT*(1), *DOWN*(2), *LEFT*(3).

FieldBerry

«package» **FieldBerry** ()
Konstruktor.

FieldElement

«package» **FieldElement** ()
Konstruktor. A *neighbours* és *mGameObjects* tömböket inicializálja.

«package» **void add** (GameObject val)
Az *mGameObjects* tömböt megnöveli eggyel, és felveszi bele a *val* objektumot mint új elemet.

«package» **int getGameObjectCount** ()
Visszaadja a rajta lévő objektumok számát, azaz az *mGameObjects* tömb hosszát.

«package» **GameObject[] getGameObjects** ()
Visszaadja a rajta lévő elemeket egy tömbben.

«package» **FieldElement[] getneighbours** ()
Visszaadja a mező szomszédait egy tömbben. A tömbben az elemek helye a *Direction* konstansai szerint meghatározott.

«package» **void remove** (GameObject val)
Leveszi *val*-t a mezőről, azaz kiveszi az *mGameObjects* tömbből a *val* objektumot, ha az benne volt.

«package» **void setGameObjects** (GameObject[])
Az *mGameObjects* tömbbe berakja a kapott objektumokat, azaz elhelyezi őket a mezőn.

«package» **void setneighbours** (FieldElement[])
Beállítja a mező szomszédreferenciáit a tömbben kapott mezőkre, a *Direction* konstansoknak megfelelő indexeléssel.

GameField

«package» **GameField** (Int2 size)
Konstruktor, mely létrehoz egy *size* méretű pályát. Ehhez létrehozza a szükséges számú *FieldElement*-et , majd ezután beállítja mindegyiknek a megfelelő szomszéd referenciáit.

«package» **int add** (GameObject val, FieldElement pos)
A *pos* mezőn elhelyezi a *val* objektumot a mező *add*() függvényével, ha a mező nem üres. Hiba esetén nem 0-val tér vissza.

«package» **FieldElement getFieldAt** (Int2 pos)
A mező *pos* pozíción lévő mezőjének referenciáját adja vissza.

«package» **FieldElement[] getFieldElements** ()
Visszaadja az összes mező referenciáját egy tömbben.

«package» **FieldElement getFreePosition** ()
Visszaadja a pálya egy szabad mezőjének referenciáját.

«package» **FieldElement getSnakeStartPosition** (int playerID)
Az adott azonosítójú játékos kígyójához tartozó start mező referenciáját adja vissza.

«public» **Int2 getsize** ()
Visszaadja a pálya méretét.

«package» **Int2[] getstartPositions** ()
Visszaadja a starthelyek tömbjét.

«package» **void setsize** (Int2 val)
Beállítja a pálya méretét *val*-ra.

«package» **void setstartPositions** (Int2[])
Beállítja a kígyók starthelyét.

GameObject

«package» **GameObject** ()
Default konstruktor.

«package» **boolean collide** ()
Absztrakt ütközéskezelő függvény. *True*-val tér vissza, ha az objektum túlélte az ütközést.

«package» **void die** ()
Az objektum halálát kezelő függvény. Törli az objektumot a mezőről, amin áll.

«package» **FieldElement getground** ()
Visszaadja azt a mezőt, amin az objektum tartózkodik.

«package» **void setground** (FieldElement val)
Az objektumnak beállítja a mező referenciáját *val* értékére.

Int2

«public» **Int2** (int a, int b)
Konstruktor, mely létrehoz egy új *Int2* objektumot, melynek két tagváltozója *a* és *b* lesz.

Model

«public» **Model** ()
Konstruktor. Beállítja a *SnakeElement* osztály statikus *Model* referenciáját önmagára.

«package» **void createNewSnake** (SnakeHead sh)
Az *mSnake* tömb méretét eggyel megnöveli, és felvesz bele egy új kígyót, melyet a *Snake(int playerID, SnakeHead sh)* konstruktorral hoz létre. Az új kígyó *playerID*-je konstans 9-es lesz, hiszen ez a függvény csak a "zombi" kígyók létrehozására használatos.

«public» **int getRemainingGameTime** ()
Visszaadja a hátralevő játékidőt másodpercben.

«public» **Snake[] getSnakes** ()
Visszaadja a kígyók tömbjét.

«public» **Object[] getVisibleObjects** ()
Visszaadja a pálya látható elemeit egy tömbben. Ezek az objektumok gyakorlatilag a pálya mezői, amiket a *GameField getFieldElements*() függvényével kér le.

«public» **int loadMap** (URL mapLocation)
Activity diagram: 6.5. ábra. Létrehoz egy új játékot az *URL*-ben megadott pályafájl tartalma alapján: létrehoz egy új *GameField*-et a megfelelő mérettel, majd a megadott objektumokat létrehozza, és elhelyezi a pályán annak *add*() metódusának meghívásával. Amennyiben a beolvasás során hiba történik (hibás fájlformátum, nem található a fájl, stb.), nem nulla értékkel tér vissza.

«public» **void reset** ()
Alaphelyzetbe állítja a játékot: kezdeti értékre állítja a játékidő-számlálót, a kígyókat pedig a starthelyükre helyezi.

«public» **void setNextMove** (int playerID, int newDirection)
A *playerID* azonosítójú kígyó irányát megkísérli *newDirection*-re változtatni az adott kígyó *setNextMove*() metódusának meghívásával.

«public» **void start** ()
Szüneteltetett játékot folytatja.

«public» **void step** ()
Activity diagram: 6.6. ábra. A modellt lépteti egyel, azaz: a kígyókat egyesével lépteti a *move*() metódusukkal, így mindegyik arra a helyre kerül, ahova szeretne lépni. Ekkor azonban előfordulhat, hogy ahova lépett, más objektum is tartózkodik, ezért minden kígyóra meghívja annak *detectCollision*() metódusát, mely az ütközéseket kezeli. Miután az ütközések lezajlottak megvizsgálja, hogy van-e olyan kígyó, amelyik túlságosan megrövidült a lépés során. Ha ilyet talál, akkor azt a kígyót megöli, majd - amennyiben nem "zombi" kígyó - újrateremti a megfelelő starthelyen.

«public» **void stop** ()
Futó játékot szünetelteti.

SawBerry

«package» **SawBerry** ()
Konstruktor.

Snake

«package» **Snake** (int playerID, int length, GameField gf)
Konstruktor, mely létrehoz egy *length* kezdeti hosszú kígyót (azaz létrehoz egy új *SnakeHead*-et, ami a kígyó feje lesz, és az első *length*-1 *move*() hívásban az egyszerű léptetésen kívül növeszti is a kígyót).

«package» **Snake** (int playerID, SnakeHead sh)
Konstruktor "zombi" kígyók létrehozására, mely nem hoz létre új fejet, csak a paraméterként kapott *SnakeHead*-re állítja a saját fej-referenciáját, és beállítja annak *playerID*-jét.

«package» **void detectCollision** ()
Továbbadja a hívást a *SnakeHead*-nek annak saját *detectCollision*() metódusának meghívásával.

«package» **void die** ()
Meghívja a hozzá tartozó kígyófej *die*() függvényét, majd *null*-ba állítja a rá mutató referenciáját.

«public» **int getLength** ()
Visszaadja a kígyó hosszát. A visszaadandó értéket a fej *getLength*() függvénye szolgáltatja, amennyiben a fej nem *null*. Ellenkező esetben a hossz 0. Kezdetben (a kígyó létrejötte utáni első két lépésben) a kígyó hosszaként fix 3-at ad vissza, hogy a modell ne törölje az újszülött kígyót rögtön mint halálosan megrövidült kígyót.

«public» **int getRemainingSawTime** ()
Visszaadja a kígyó hátralevő fűrészfogas idejét. Ezt a fej hasonló nevű függvényétől kéri le, ha az nem *null*.

«public» **int getStoneCount** ()
Visszaadja a kígyóban található kőbogyók számát. Ez a fej *getStoneCount*() függvénye által visszaadott érték, ha a fej nem *null*, egyébként nulla.

«public» **int getplayerID** ()
Visszaadja a kígyó azonosító számát.

«package» **void move** ()
A kígyót lépteti a kígyófej *move*() metódusának meghívásával. A konfigurációs fájlban meghatározott sebességnek megfelelő számszor meghívja a fej *digestStone*() metódusát, hogy a kövek vándoroljanak hátrafelé. Csökkenti eggyel a fej hátralevő fűrészbogyós idejét.

«package» **void setNextMove** (int newDirection)
A kígyó haladási irányát megváltoztatandó meghívja a kapott paraméterrel a fej *setnewDirection*() metódusát.

SnakeBody

«package» **SnakeBody** (int playerID)
Konstruktor. Meghívja az ősosztály konstruktorát a kapott paraméterrel.

«package» **boolean getisTail** ()
Visszaadja, hogy az adott testelem farok-e, azaz ő-e az utolsó elem.

«package» **void setisTail** (boolean val)
Beállítja, hogy az adott testelem farok-e, azaz ő-e az utolsó elem.

SnakeElement

«package» **SnakeElement** (int playerID)
Konstruktor.

«package» **void becomeNewSnake** (int newDir)
Átalakul egy új, "zombi" kígyó fejévé, ami kezdetben a *newDir* által megadott irányba halad. Létrehozza az új fejet, beállítja a referenciáit (hozzáadja a mezőhöz amin áll, beköti a következő testelemet), és beállítja az új haladási irányát. Ezek után saját referenciáit nullázza a talajra és a következő testelemre, és átadja az új fejet a modellnek a *Model.createNewSnake*() metódus meghívásával.

«package» **boolean collide** ()
Ütközéskezelő függvény. Lekéri a talajától az ott tartózkodó egyéb objektumokat, és eldönti, hogy túlélte-e az ütközést, *true*-val tér vissza, ha igen. Amennyiben átharapást érzékel, az átharapó kígyótól lekéri annak irányát, és ennek megfelelően (épp ezzel ellentétesre) beállítja a *becomingHead* változó irányát.

«package» **void detectCollision** ()
Activity diagram: 6.4. ábra. Ütközést érzékelő függvény. Lekéri a talajtól a rajta lévő objektumok számát, és ha ez nagyobb mint 1, azaz van más objektum is rajta kívül ugyanazon a talajon, akkor levezényel egy ütközést: lekéri az ott lévő objektum listát a talajtól , és mindegyikre *collide*()-ot hív, az eredményt eltárolva egy tömbben. Akik azt jelezték, hogy meghalnak az ütközéstől, azoknak meghívja a *die*() metódusát.

«package» **void die** ()
A *GameObject*-ben szereplő *die*() függvény egy felüldefiniált változata, amely meghívja a függvény paraméteres változatát *false* paraméterrel.

«package» **void die** (boolean detect)
A *SnakeElement* halálát elrendező metódus. A *detect* paraméter igaz értéke jelöli, ha további *detectCollision* hívásokra lehet szükség. Először megvizsgáljuk, hogy a *becomingHead* változó értéke mennyi. Ha ez egy érvényes irányt jelöl, meghívjuk a *next becomeNewSnake*() metódusát *becomingHead* paraméterrel. Ez azt jelenti, hogy a mögöttünk lévő elem lesz az új fej. Ezek után a *SnakeElement* törli magát a pályáról. Ha nem volt érvényes irány, akkor nem átharapás miatt haltunk meg, hanem egy előző *SnakeElement die*() hívása érkezett tovább hozzánk (pl. ütközés volt fallal, és a kígyó meghalt). Ekkor a *detect* paraméter függvényében további ütközéseket vizsgálunk (nem jött-e bele valaki a meghaló félben lévő testbe?) a *detectCollision*() metódussal, majd ha van még mögöttünk testelem, annak is meghívjuk a *die*() metódusát *true* paraméterrel, végül töröljük magunkat a pályáról.

«package» **boolean digestStone** ()
Kő emésztése. *True*-val tér vissza, ha a folyamat végén a testelem üres, kőmentes, azaz ha nem is volt benne kő, vagy ha volt, de sikerült hátrébb küldenie a testben. Hátrébb akkor tudja küldeni, ha a *next digestStone*()-ja *true*-val tért vissza, azaz volt benne szabad hely a kő fogadására, vagy sikerült felszabadítania. Ekkor a *next* kőtartalmát this kőtartalmára állítjuk, *this* kőtartalmát pedig *false*-ra. Ha nem sikerült továbbküldeni a követ, *false*-szal térünk vissza.

«package» **int getLength** ()
Visszaadja a kígyó testének hosszát ettől a testelemtől számítva a test végéig, azaz a *next.getLength*() által visszaadott érték eggyel növelt értékét, ha *next* nem *null*. Ha *next* *null*, akkor a visszaadott hossz 1.

«package» **int getStoneCount** ()
Visszaadja a kígyó testében lévő kövek számát ettől a testelemtől számítva a test végéig, azaz a *next.getStoneCount*() által visszaadott érték megnövelve a saját kőtartalommal (0 vagy 1), ha *next* nem *null*. Ha *next null*, akkor a visszaadott szám a saját kőtartalom (0 vagy 1).

«public» **boolean gethasStone** ()
Igazat ad vissza, ha a testelem tartalmaz követ.

«public» **boolean getisDead** ()
Igazat ad vissza, ha a testelem halott.

«public» **int getplayerID** ()
Visszaadja a kígyó azonosítóját.

«protected» **void grow** ()
Testhossznövelő metódus. A hívás végigfut a teljes testen egészen a legutolsó elemig, amelyik létrehoz egy új testelemet maga mögött.

«package» **void move** (FieldElement newGround)
Mozgató metódus. Meghívja a következő testelem *move*() metódusát a saját talajával, mint paraméterrel, majd saját talajának beállítja a *newGround*-ot (az előző talajról törli magát *remove*()-val, az újhoz hozzáadja magát *add*()-dal). Ha *next* nem *null*, viszont halott, akkor *next*-et *null*-ra állítja.

«protected» **void sethasStone** (boolean val)
A testelem kőtartalmát *val*-ra állítja.

«protected» **void setisDead** (boolean val)
Beállítja az *isDead* flag-et *val*-ra.

«protected» **void setplayerID** (int val)
A saját kígyóazonosítóját *val*-ra állítja.

SnakeHead

«package» **SnakeHead** (int playerID)
Konstruktor.

«package» **void collide** ()
Activity diagram: 6.7. ábra. Ütközéskezelő függvény. Lekéri a talajától az ott tartózkodó egyéb objektumokat, és eldönti, hogy túlélte-e az ütközést, *true*-val tér vissza, ha igen. Amennyiben átharapást érzékel, az átharapó kígyótól lekéri annak irányát, és ennek megfelelően (épp ezzel ellentétesre) beállítja a *becomingHead* változó irányát. Amennyiben túléli az ütközést, és van felvehető bogyó, akkor azt felveszi, azaz meghívja a megfelelő kezelőfüggvényt (*grow*(), *setremainingSawTime*() vagy *digestStone*()).

«package» **boolean digestStone** ()
A fejnek a bekerült követ mindenképpen továbbítania kell a testnek. Amennyiben *next null*, vagy *next.digestStone*() *false*-szal tér vissza, az azt jelenti, hogy a kőbogyó nem tud hova tovább kerülni a fejből. Ekkor a kígyó meghal, meghívódik a *die*().

«package» **int getremainingSawTime** ()
Visszaadja a hátralévő fűrészes időt.

«package» **void move** ()
Mozgató függvény. A következő testelemet a saját helyére mozgatja annak *move*() metódusával, aminek a saját talaját adja paraméterül, saját maga pedig a pillanatnyi talaja *newDirection* által meghatározott szomszédjára lép, amennyiben az nem *null*, egyébként meghal.

«package» **void setnewDirection** (int val)
A következő lépés irányát *val*-ra állítja. Ez a korábbi értéket felülírja.

«package» **void setrefGameField** (GameField val)
Saját pálya-referenciáját *val*-ra állítja.

«package» **void setremainingSawTime** (int val)
A hátralévő fűrészes időt *val*-ra állítja.

StoneBerry

«package» **StoneBerry** ()
Konstruktor.

Wall

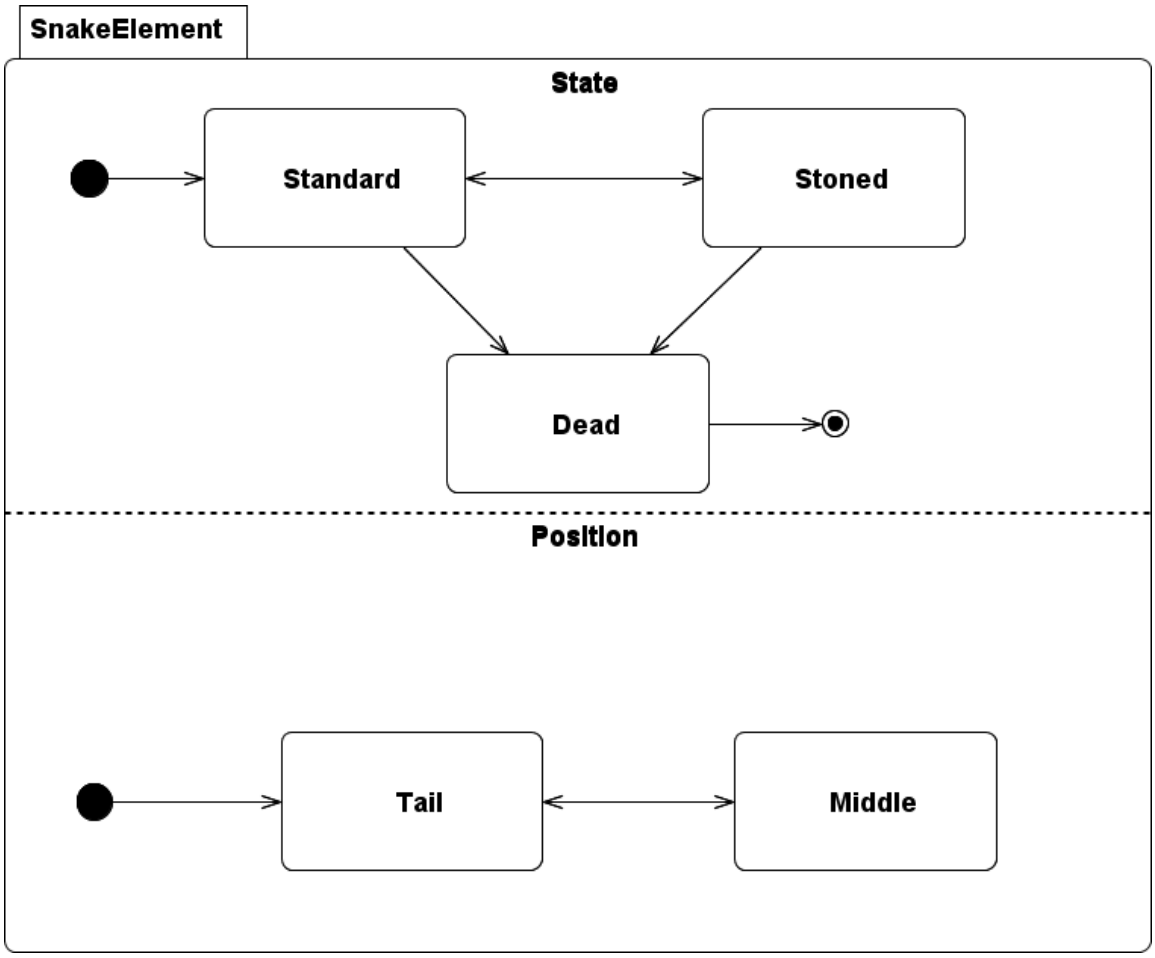
«package» **Wall** ()
Konstruktor.

«package» **void collide** ()
Felüldefiniált ütközéskezelő függvény, mely mindig igazzal tér vissza, hiszen a fal minden ütközést túlél.

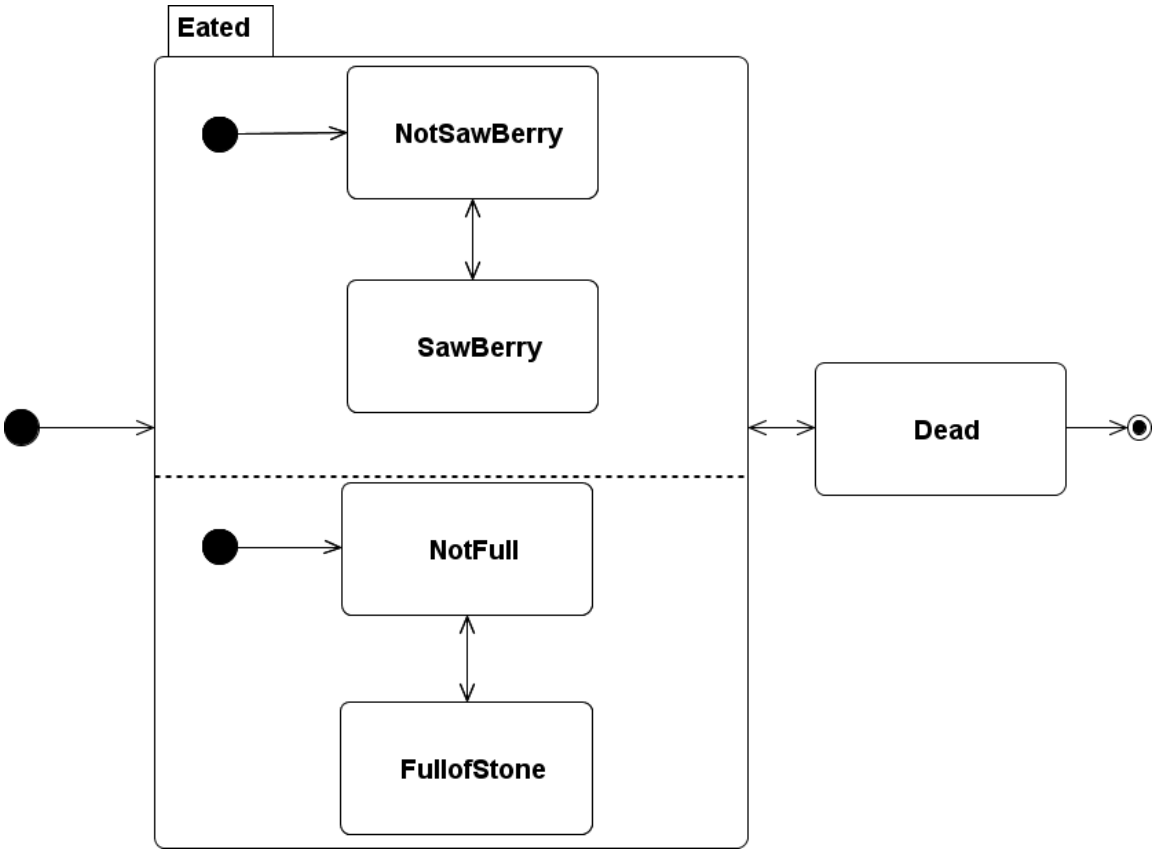
CLI

«public static» **void writeLog** (String message)
A message *String* tartalmát hozzáfűzi a logfájl végéhez. A logfájl egy statikus osztályváltozón keresztül érhető el, melynek az értékének a beállításáról a *Control* gondoskodik.

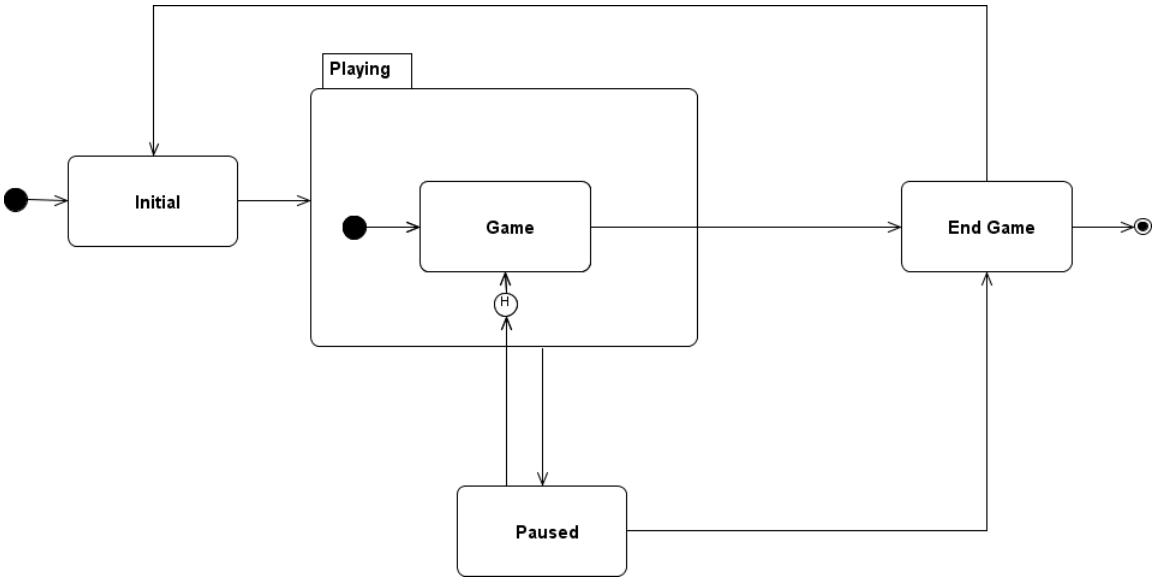
«public static» **void draw** (Writer out)
Kirajzolja a modell pillanatnyi állapotát a megadott kimenetre. Ez lehet fájl, vagy a konzol is. A kirajzoláshoz a *Model*-referenciáján keresztül lekéri a pálya kirajzolandó elemeit a *getVisibleObjects*() függvénnyel.



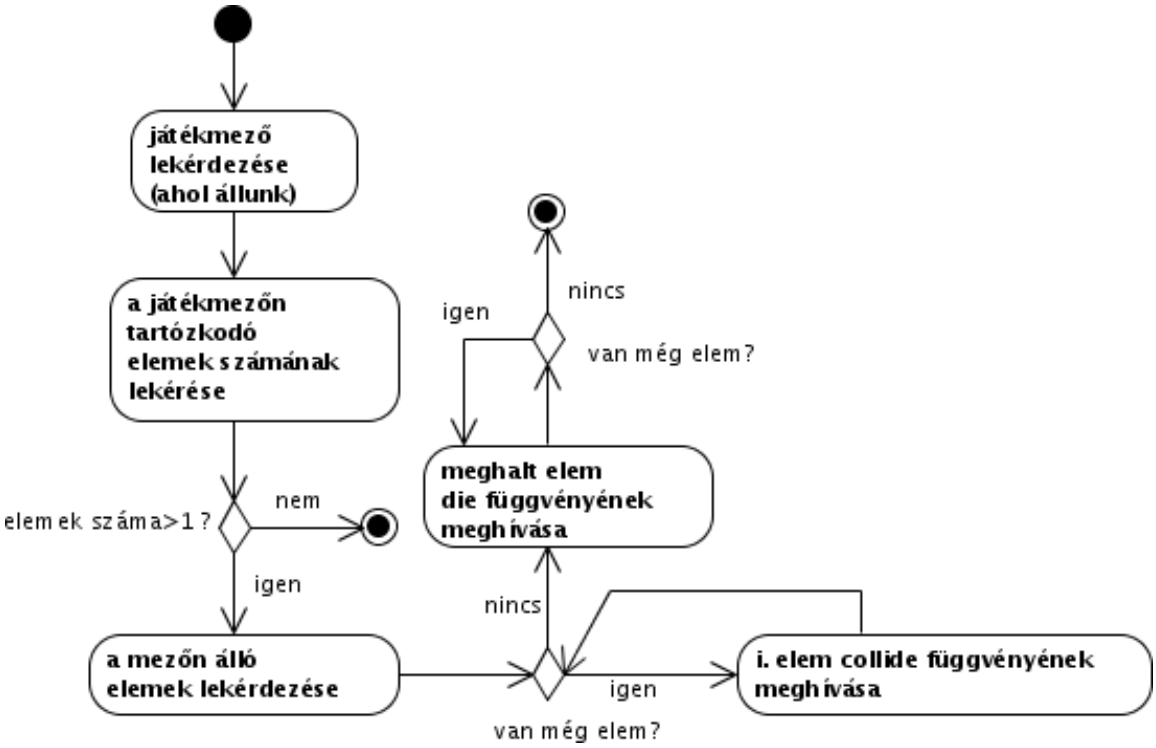
6.1. ábra. A SnakeBody állapotgépe



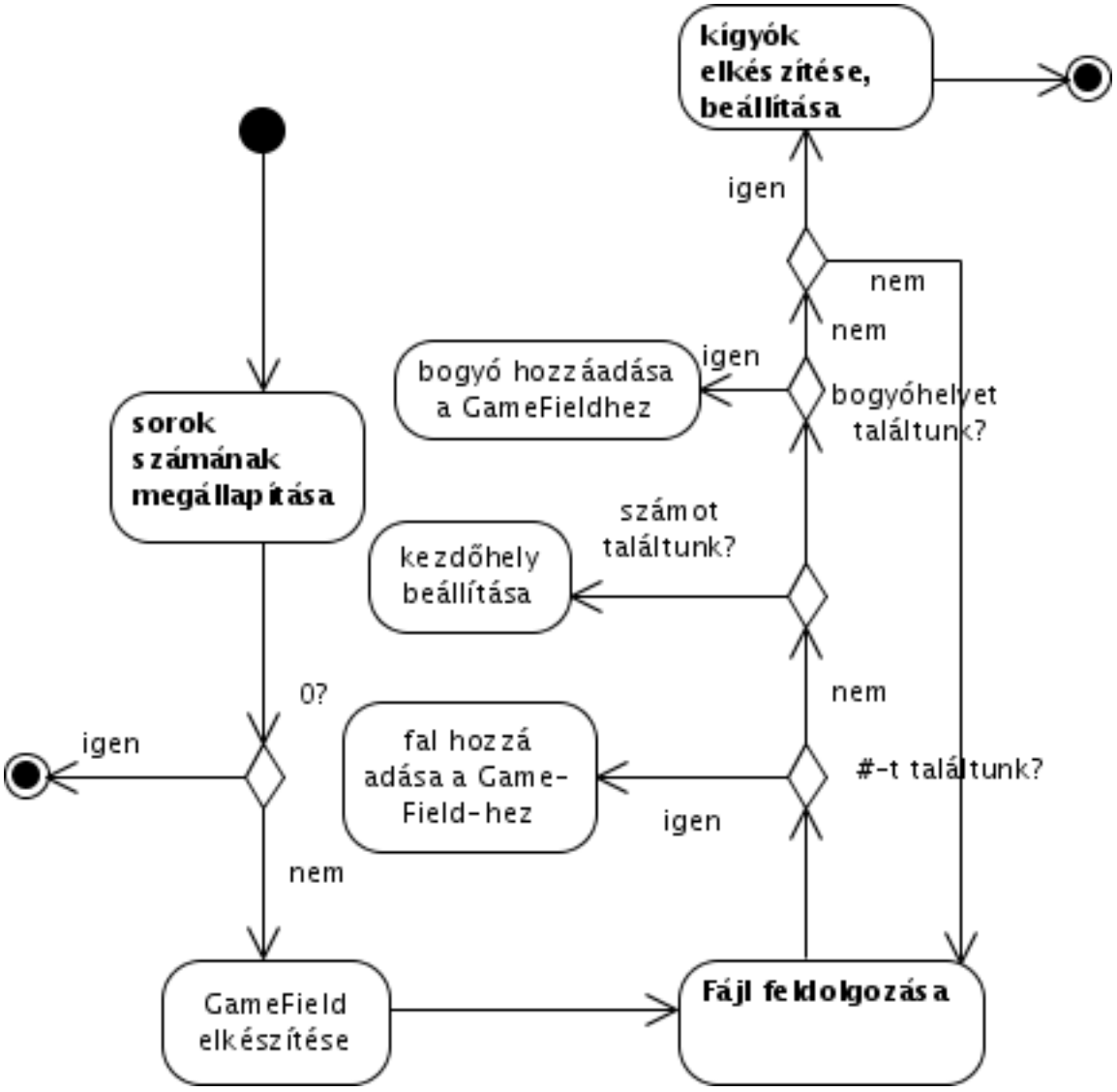
6.2. ábra. A Snake állapotgépe



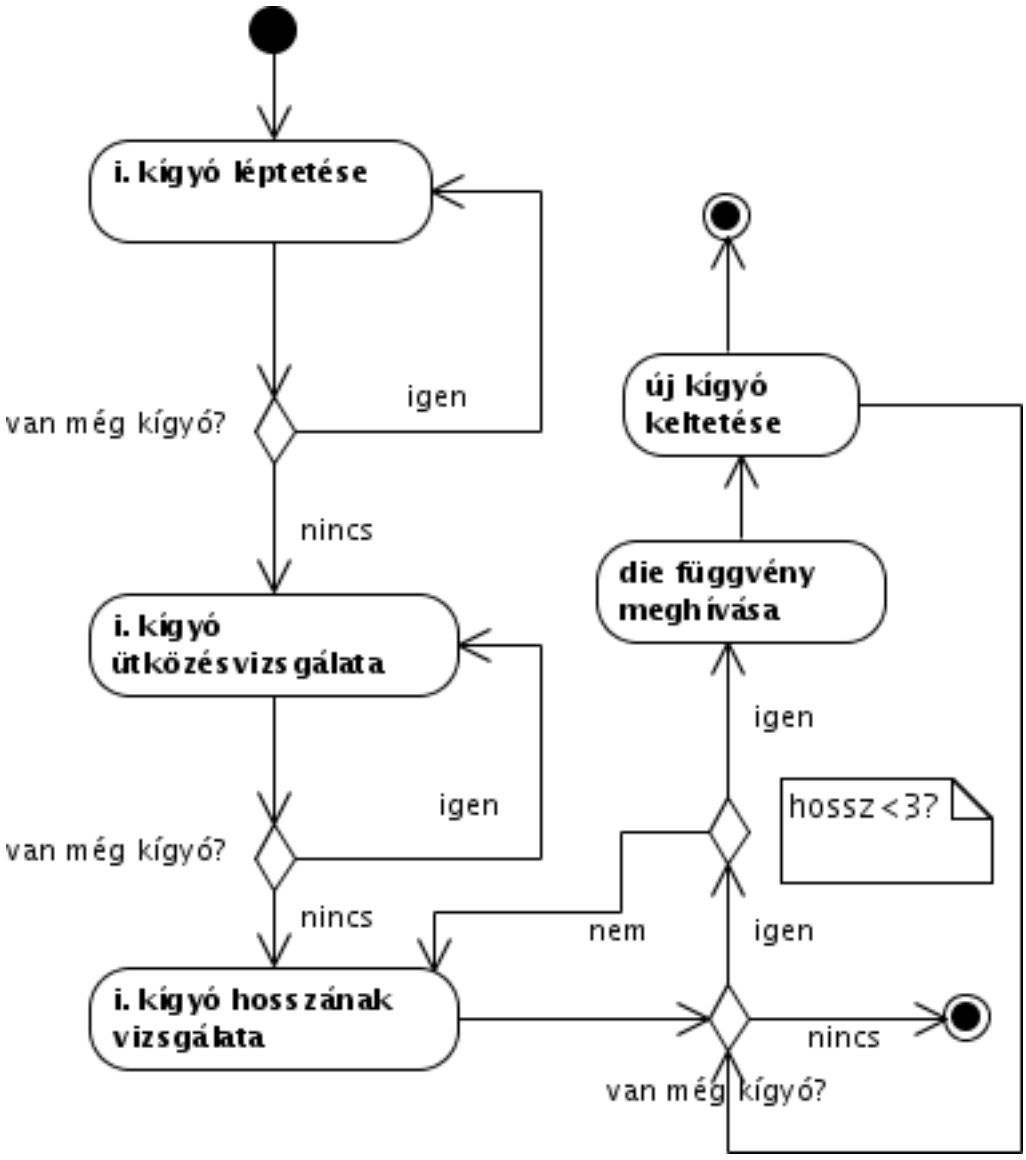
6.3. ábra. A Model állapotgépe



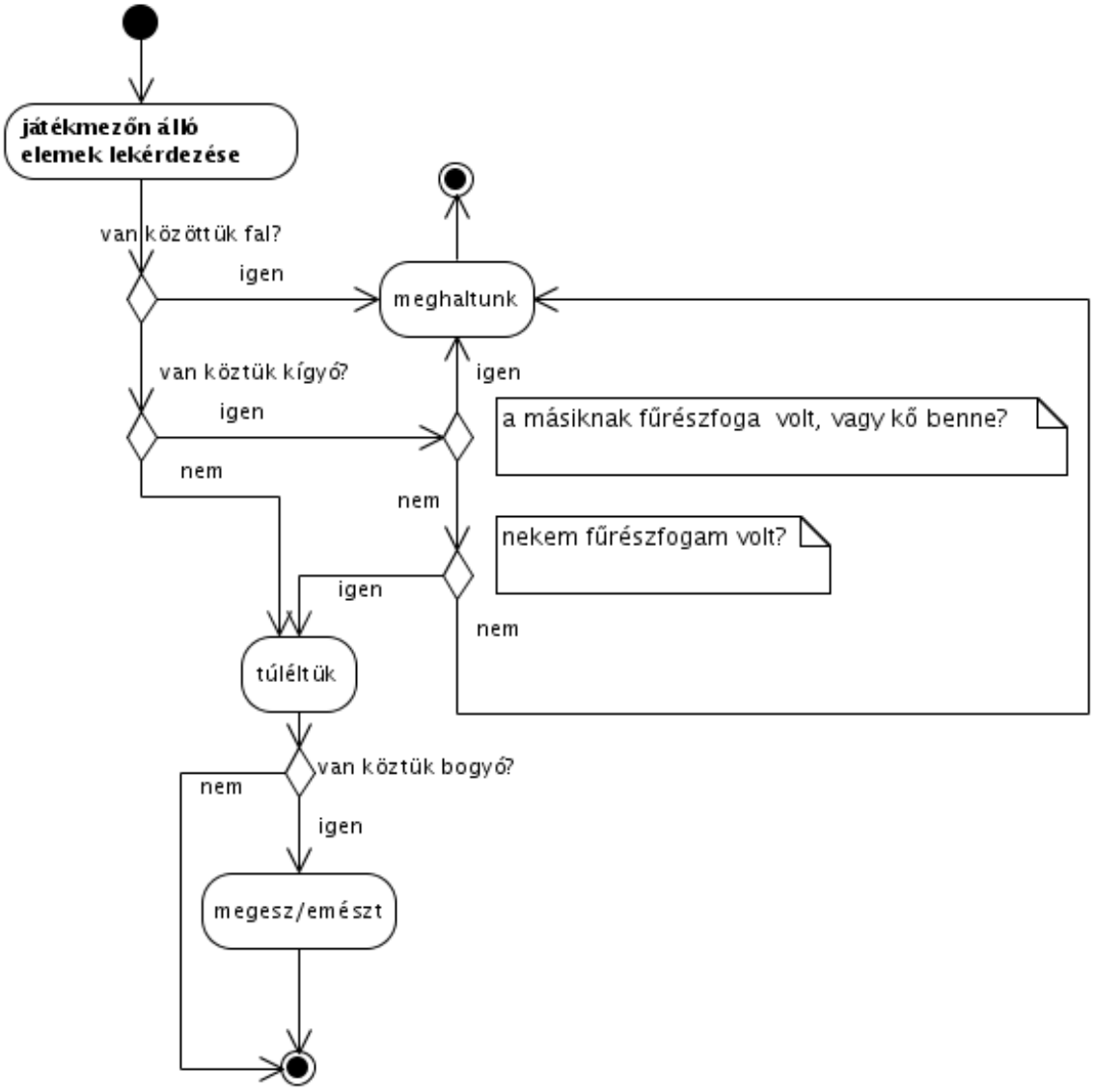
6.4. ábra. detectCollision activity diagramja



6.5. ábra. loadMap activity diagramja



6.6. ábra. step activity diagramja



6.7. ábra. SnakeHead collide activity diagramja

6.2. A tesztek részletes tervei, leírásuk a teszt nyelvén

6.2.1. Inicializálás, Pálya létrehozása

Környezet:
Minden elemből elhelyezünk rajta egyet. Megnézzük helyesen jön-e létre, megfelelően kapcsolódnak-e az elemek.
Bemeneti Adatok:

```
uj_jatek Init.map
```

Várt kimenet:

```
Palya sikeresen betoltve.
```

Lehetséges hibák:
Nem jön létre pálya, nem találja a megadott nevű pályát.
Hiba oka, helye:
A hiba a *Model* osztály *loadMap* metódusában keresendő.

6.2.2. Játék szüneteltetése, folytatása és befejezése

Környezet:
Egy üres pályát létrehozunk, szüneteltetjük a játékot, majd ezután folytatjuk. Végül befejezzük.
Bemeneti Adatok:

```
uj_jatek startstop.map
leptet
szunet
leptet
folytatas
leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Jatek szunetel..
Jatek folytatodik.
Kigyok leptetese.
Jatek vege.
```

Lehetséges hibák:
Játék nem áll meg, nem folytatódik vagy nem fejeződik be.
Hiba oka, helye:
A hibát a *Model start*, *stop* metódusaiban keressük.

6.2.3. Kígyó irányának megadása

Környezet:
Egy kígyónak egy üres pályán több irányt adunk meg mielőtt lépne, megnézzük, hogy a legutolsó adott irányba halad-e.

Bemeneti Adatok:

```
uj_jatek irany.map
balra 0 jobbra 1 le 0 fel 1 jobbra 0 balra 1 fel 0 le 1
leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Jatek vege.
```

Lehetséges hibák:
Kígyó az elsőnek megadott irányba mozog.

Hiba oka, helye:
A hiba a *Snake* ill. *SnakeHead* osztályok *setNextMove* ill. *setnewDirection* metódusaiban keresendő.

6.2.4. Kígyó mozgása

Környezet:
Két kígyó egy üres pályán mind a négy irányba mozog.

Bemeneti Adatok:

```
uj_jatek mozgas.map
balra 0
jobbra 1
2leptet
le 0
fel 1
2leptet
jobbra 0
balra 1
2leptet
fel 0
fel 1
le 1
2leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
```

Lehetséges hibák:

Kígyó meghal, nem változtat irányt, darabjai nem követik.

Hiba oka, helye:

A hiba a *Snake*, *SnakeHead*, *SnakeElement* osztályok metódusaiban keresendő. A hiba lehet a *Model setNextMove*-jában is (ha rossz kigyóhoz küldi az irányt).

6.2.5. Mezei bogyó felvétele

Környezet:

A kígyó rálép egy mezei bogyóra. Megnézzük helyesen veszi-e fel és nő-e tőle. A bogyó újraterem-e üres helyen.

Bemeneti Adatok:

```
uj_jatek mezeibogyo.map
fel 0
6leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
Jatek vege.
```

Lehetséges hibák:

A kígyó meghal, a bogyó ottmarad, nem terem újra. A kígyó nem nő meg, a növés nem áll meg.

Hiba oka, helye:

A *Snake detectCollision* ill. a *SnakeHead collide* és *Berry die*, *collide* metódusaiban lehet hiba. A *SnakeElement grow* metódusa nem megfelelő.

6.2.6. Fűrészbogyó felvétele

Környezet:
A kígyó felvesz egy fűrészbogyót. Megnézzük, hogy helyesen veszi-e fel és fűrészes állapotba kerül-e meghatározott ideig.
Bemeneti Adatok:

```
uj_jatek fureszbogyo.map
fel 0
10leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo fureszbogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo fureszfoga megszunt.
Jatek vege.
```

Lehetséges hibák:
A kígyó meghal, a bogyó ottmarad, nem terem újra. A kígyó nem lesz fűrészes, vagy nem megfelelő ideig.
Hiba oka, helye:
A *Snake detectCollision* ill. a *SnakeHead setremainingSawTime*, *move*, *collide* és *Berry die*, *collide* metódusaiban lehet hiba.

6.2.7. Kőbogyó felvétele

Környezet:
A kígyó felvesz egy kőbogyót. Megnézzük helyesen veszi-e fel és emészti meg.
Bemeneti Adatok:

```
uj_jatek kobogyo.map
fel 0
5leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo kobogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
Jatek vege.
```

Lehetséges hibák:

A kígyó meghal, a bogyó ottmarad, nem terem újra. A kígyóban nem marad kő, a kő nem mozog hátrafelé.

Hiba oka, helye:

A *Snake detectCollision* ill. a *SnakeHead collide* és *Berry die*, *collide* metódusaiban lehet hiba. A *SnakeHead*, *SnakeElement digestStone* metódusa nem megfelelő.

6.2.8. Ütközés saját kígyódarabbal

Környezet:

A kígyó visszakanyarodik és egy saját darabjának ütközik.

Bemeneti Adatok:

```
uj_jatek utkozdarabsajat.map
balra 0
leptet
le 0
leptet
jobbra 0
leptet
fel 0
2leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
0-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
0-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo meghalt.
Uj 0-s kigyo jott letre.
Kigyok leptetese.
Jatek vege.
```

Lehetséges hibák:
A kígyó nem hal meg.
Hiba oka, helye:
A *Snake detectCollision* ill. a *SnakeHead* és *SnakeElement collide* metódusainál lehet hiba, valamint a *SnakeHead die*-nál.

6.2.9. Ütközés kígyódarabbal

Környezet:
Az egyik kígyó nekiütközik a másik kígyó egy darabjának.
Bemeneti Adatok:

```
uj_jatek utkozdarab.map
balra 0
jobbra 1
2leptet
fel 1
2leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo meghalt.
Uj 0-s kigyo jott letre.
Kigyok leptetese.
Jatek vege.
```

Lehetséges hibák:
Egyik kígyó sem hal meg, mindkettő meghal, rossz kígyó hal meg.
Hiba oka, helye:
A *Snake detectCollision* ill. a *SnakeHead* és *SnakeElement collide* metódusainál lehet hiba, valamint a *SnakeHead die*-nál.

6.2.10. Ütközés fejjel

Környezet:
A két kígyó fejjel egymásnak megy.
Bemeneti Adatok:

```
uj_jatek utkozfej.map
balra 0
jobbra 1
5leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo meghalt.
1-s kigyo meghalt.
Uj 0-s kigyo jott letre.
Uj 1-s kigyo jott letre.
Kigyok leptetese.
Kigyok leptetese.
Jatek vege.
```

Lehetséges hibák:

Nem hal meg mindkét kígyó.

Hiba oka, helye:

A *Snake detectCollision* ill. a *SnakeHead* és *SnakeElement collide* metódusainál lehet hiba, valamint a *SnakeHead die*-nál.

6.2.11. Ütközés kőbogyóval

Környezet:

Egy kővel teli kígyó kőbogyót próbál meg felvenni.

Bemeneti Adatok:

```
uj_jatek utkozko.map
6leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo kobogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo kobogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo kobogyot vett fel.
0-s kigyo meghalt.
Uj 0-s kigyo jott letre.
Jatek vege.
```

Lehetséges hibák:

Nem hal meg a kígyó.

Hiba oka, helye:
A *Snake detectCollision* ill. a *SnakeHead* és *SnakeElement collide* metódusainál lehet hiba, valamint a *SnakeHead die*-nál.

6.2.12. Ütközés fallal

Környezet:
Egy kígyó egy faldarabnak megy.
Bemeneti Adatok:

```
uj_jatek utkozfal.map
5leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo meghalt.
Uj 0-s kigyo jott letre.
Kigyok leptetese.
Jatek vege.
```

Lehetséges hibák:
A kígyó nem hal meg.
Hiba oka, helye:
A *Snake detectCollision* ill. a *SnakeHead*, *SnakeElement* és *Wall collide* metódusainál lehet hiba, valamint a *SnakeHead die*-nál.

6.2.13. Fűrészes ütközés

Környezet:
A kígyó felvesz egy fűrészbogyót, és ezután a másik kígyót kettéharapja. Megnézzük új kígyó helyesen jön-e létre.
Bemeneti Adatok:

```
uj_jatek utkozfuresz.map
balra 0
fel 1
3leptet
jobbra 1
3leptet
le 1
7leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
0-s kigyo fureszbogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
Uj zombi kigyo jott letre.
Kigyok leptetese.
Jatek vege.
```

Lehetséges hibák:

Mindkét kígyó meghal, rossz kígyó hal meg, nem jön létre új kígyó, az új kígyó rossz irányba indul el.

Hiba oka, helye:

A *Snake detectCollision* ill. a *SnakeHead die* és *collide* metódusainál lehet hiba, valamint a *SnakeElement becomeNewSnake*-nél és *collide*-nál és *Model createNewSnake*-nél.

6.2.14. Fűrészes nyakharapás

Környezet:

A kígyó felvesz egy fűrészbogyót, és ezután a másik kígyó nyakába harap. Megnézzük, hogy a maradék kígyó a rövidsége miatt meghal-e.

Bemeneti Adatok:

```
uj_palya utkozfuresznyak.map
balra 0
fel 1
3leptet
jobbra 1
3leptet
le 1
5leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
0-s kigyo fureszbogyot vett fel.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
1-s kigyo meghalt.
Uj zombi kigyo jott letre.
Jatek vege.
```

Lehetséges hibák:

Mindkét kígyó meghal, rossz kígyó hal meg, nem jön létre új kígyó, az új kígyó rossz irányba indul el. A rövid kígyó nem pusztul el.

Hiba oka, helye:

A *Snake detectCollision* ill. a *SnakeHead die* és *collide* metódusainál lehet hiba, valamint a *SnakeElement becomeNewSnake*-nél és *collide*-nál. A *Snake getLength*, *Model step* függvénye nem megfelelő.

6.2.15. Fűrészes kőütközés

Környezet:

Egy fűrészfogas kígyó egy olyan kígyódarabnak ütközik, amiben kő van.

Bemeneti Adatok:

```
uj_palya utkozfureszko.map
balra 0
fel 1
3leptet
jobbra 1
3leptet
le 1
5leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
0-s kigyo fureszbogyot vett fel.
1-s kigyo mezei bogyot vett fel.
Kigyok leptetese.
0-s kigyo kobogyot vett fel.
Kigyok leptetese.
0-s kigyo meghal.
Uj 0-s kigyo jon letre.
Jatek vege.
```

Lehetséges hibák:

Mindkét kígyó meghal, rossz kígyó hal meg, nem jön létre új kígyó, az új kígyó rossz irányba indul el. A fűrészfogas kígyó nem hal meg.

Hiba oka, helye:

A *Snake detectCollision* ill. a *SnakeHead die* és *collide* metódusainál lehet hiba, valamint a *SnakeElement becomeNewSnake*-nél és *collide*-nál.

6.2.16. Fűrészes fejharapás

Környezet:

Két fűrészbogyós kígyó egymással szemben ütközik.

Bemeneti Adatok:

```
uj_jatek utkozfureszfuresz.map
balra 0
jobbra 1
4leptet
vege
```

Várt kimenet:

```
Palya sikeresen betoltve.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo fureszbogyot vett fel.
1-s kigyo fureszbogyot vett fel.
Kigyok leptetese.
Kigyok leptetese.
0-s kigyo meghalt.
Uj 0-s kigyo jott letre.
1-s kigyo meghalt.
Uj 1-s kigyo jott letre.
Jatek vege.
```

Lehetséges hibák:

Nem hal meg mindkét kígyó.

Hiba oka, helye:

A *Snake detectCollision* ill. a *SnakeHead* és *SnakeElement collide* metódusainál lehet hiba, valamint a *SnakeHead die*-nál.

6.3. Grafikus felület elvárt kimenetei

```
Init.map:
X#kmf
X1X0X

Init.gout:
X#kmf
X1X0X

startstop.map:
XXXXX
XXXXX
X1X0X
XXXXX

startstop.gout:
XXXXX
XXXXX
X1X0X
XXXXX

XXXXX
X1X0X
X*X*X
XXXXX

XXXXX
X1X0X
X*X*X
XXXXX

X1X0X
X*X*X
X*X*X
XXXXX

irany.map:
XXXXX
X1X0X
XXXXX

irany.gout:
XXXXX
X1X0X
XXXXX
```

```

XXX0X
X*X*X
X1XXX

mozgas.map:
XXXXXXX
XXXXXXX
XX0X1XX
XXXXXXX
XXXXXXX

mozgas.gout:
XXXXXXX
XXXXXXX
XX0X1XX
XXXXXXX
XXXXXXX

XXXXXXX
XXXXXXX
X0*X*1X
XXXXXXX
XXXXXXX

XXXXXXX
XXXXXXX
0**X**1
XXXXXXX
XXXXXXX

XXXXXXX
XXXXXX1
**XXX**
0XXXXXX
XXXXXXX

XXXXXX1
XXXXXX*
*XXXXX*
*XXXXXX

XXXX1**
XXXXXXX
XXXXXXX
XXXXXXX
**0XXXX

XXXX**X
XXXX1XX
XXXXXXX
XX0XXXX
X**XXXX

XXXX*XX
XXXX*XX
XX0X1XX
XX*XXXX
XX*XXXX

mezeibogyo.map:
XXX
XXX
XmX
XXX
XXX
XXX
X0X

mezeibogyo.gout:
XXX
XXX
XmX
XXX
XXX
XXX
X0X
```

```
XXX
XXX
XmX
XXX
XXX
X0X
X*X

XXX
XXX
XmX
XXX
X0X
X*X
X*X

XXX
XXX
XmX
X0X
X*X
X*X
XXX

XXX
XXX
X0X
X*X
X*X
XXX
XXX

XXX
X0X
X*X
X*X
X*X
XXX
XXX
```

fureszbogyo.map:

```
X0X
X*X
X*X
X*X
XXX
XXX
XXX

XXX
XXX
XXX
XXX
XfX
XXX
XXX
XXX
XXX
X0X

fureszbogyo.gout:
XXX
XXX
XXX
XXX
XXX
XXX
XfX
XXX
XXX
XXX
X0X

XXX
XXX
XXX
XXX
XXX
XXX
XfX
XXX
XXX
X0X
X*X
```

```
XXX
XXX
XXX
XXX
XXX
XXX
XfX
XXX
X0X
X*X
X*X
XXX
XXX

XXX
XXX
XXX
XXX
XXX
XXX
XfX
XXX
XXX
X0X
X*X
XXX
XXX

XXX
XXX
XXX
XXX
XXX
X&X
X*X
X*X
XXX
XXX
```

```
XXX
XXX
XXX
XXX
XXX
X&X
X*X
X*X
XXX
XXX
XXX

XXX
XXX
XXX
XXX
X&X
X*X
X*X
XXX
XXX
XXX
XXX

XXX
XXX
XXX
X&X
X*X
X*X
XXX
XXX
XXX
XXX
XXX
```

```
XXX
XXX
X&X
X*X
X*X
XXX
XXX
XXX

XXX
X&X
X*X
X*X
XXX
XXX
XXX
XXX
XXX

X0X
X*X
X*X
XXX
XXX
XXX
XXX
XXX
XXX
XXX

kobogyo.map:
XXX
XXX
XkX
XXX
XXX
X0X
```

```
kobogyo.gout:
XXX
XXX
XkX
XXX
XXX
X0X

XXX
XXX
XkX
XXX
X0X
X*X

XXX
XXX
XkX
X0X
X*X
X*X

XXX
XXX
X0X
X@X
X*X
XXX

XXX
X0X
X*X
X@X
XXX
XXX

X0X
X*X
X@X
XXX
XXX
XXX
```

utkozsajat.map:
```
XXXXX
XXXXX
Xm0XX
XmXXX
XXXXX
```

utkozsajat.gout:
```
XXXXX
XXXXX
Xm0XX
XmXXX
XXXXX

XXXXX
XXXXX
X0*XX
XmXXX
XXXXX

XXXXX
XXXXX
X**XX
X0XXX
XXXXX

XXXXX
XXXXX
X**XX
X*0XX
XXXXX

XXXXX
XXXXX
XX0XX
XXXXX
XXXXX
```

utkozdarab.map:
```
XXXXXXXX
XXXXXXXX
X1XXXX0X
XXXXXXXX
```

utkozdarab.gout:
```
XXXXXXXX
XXXXXXXX
X1XXXX0X
XXXXXXXX

XXXXXXXX
XXXXXXXX
X*1XX0*X
XXXXXXXX

XXXXXXXX
XXXXXXXX
X**10**X
XXXXXXXX

XXXXXXXX
XXX1XXXX
XX**XX0X
XXXXXXXX

XXX1XXXX
XXX*XX0X
XXX*XX*X
XXXXXXXX
```

utkozfej.map:
```
XXXXXXXX
XXXXXXXX
X1XXXX0X
XXXXXXXX
```

utkozfej.gout:
```
XXXXXXXX
XXXXXXXX
X1XXXX0X
XXXXXXXX

XXXXXXXX
XXXXXXXX
X*1XX0*X
XXXXXXXX
```

```
XXXXXXXX
XXXXXXXX
X**10**X
XXXXXXXX

XXXXXXXX
XXXXXXXX
X1XXXX0X
XXXXXXXX

XXXXXXXX
X1XXXX0X
X*XXXX*X
XXXXXXXX

X1XXXX0X
X*XXXX*X
X*XXXX*X
XXXXXXXX
```

utkozko.map:
```
XkX
XXX
XkX
XXX
XkX
XXX
X0X
```

utkozko.gout:
```
XkX
XXX
XkX
XXX
XkX
XXX
X0X

XkX
XXX
XkX
XXX
XkX
X0X
X*X
```

```
XkX
XXX
XkX
XXX
X0X
X@X
X*X

XkX
XXX
XkX
X0X
X*X
X@X
XXX

XkX
XXX
X0X
X@X
X@X
XXX
XXX

XkX
X0X
X@X
X@X
XXX
XXX
XXX

XXX
XXX
XXX
XXX
XXX
XXX
X0X
```

utkozfal.map:

```
X#X
XXX
XXX
XXX
X0X
```

utkozfal.gout:

```
X#X
XXX
XXX
XXX
X0X
X*X

X#X
XXX
X0X
X*X
X*X

X#X
X0X
X*X
X*X
XXX

X#X
XXX
XXX
XXX
X0X
X*X
```

utkozfuresz.map:

```
XmXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
X1XXmXfXXXXXXXXX0X
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XmXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
X1XXmXXXXXXXXXXXXX
X*XXmXfXXXXXXXX0*X
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XmXXmXXXXXXXXXXXXX
X1XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXfXXXXXXX0**X
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X1XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
XXXXmXfXXXXXX0**XX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
```

utkozfuresz.gout:

```
XmXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
X1XXmXfXXXXXXXXX0X
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XmXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
X1XXmXXXXXXXXXXXXX
X*XXmXfXXXXXXXX0*X
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XmXXmXXXXXXXXXXXXX
X1XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXfXXXXXXX0**X
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X1XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
XXXXmXfXXXXXX0**XX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
```

```
X*1XmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
XXXXmXfXXXXX0**XXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X**1mXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfXXXX0**XXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X***1XXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfXXX0**XXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X****XXXXXXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfXX0**XXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X****XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXmXfX0**XXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
```

```
X****XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX1Xf0**XXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X****XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*X&**XXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XX***XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*&**XXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XXX**XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX9XXXXXXXXXXXXX
XXXX&**XXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*9XXXXXXXXXXXX
XXX&**XXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX1XXXXXXXXXXXXX
```

```
utkfuresznyak.map:
XmXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
X1XXmXfXXXXXXXX0XX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

utkfuresznyak.gout:
XmXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
X1XXmXfXXXXXXXX0XX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XmXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
X1XXmXXXXXXXXXXXXX
X*XXmXfXXXXXXX0*XX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XmXXmXXXXXXXXXXXXX
X1XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXfXXXXXX0**XX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X1XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
XXXXmXfXXXXX0**XXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X*1XmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
XXXXmXfXXXX0**XXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
```

```
X**1mXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfXXX0**XXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X***1XXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfXX0**XXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X****XXXXXXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfX0**XXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X****XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXmXf0**XXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X****XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX1X&**XXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X****XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*&**XXXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
```

```
XX***XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*9XXXXXXXXXXXX
X1XX&**XXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

utkozfureszko.map:
XmXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfXXXXXXXX0XX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

utkozfureszko.gout:
XmXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfXXXXXXXX0XX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XmXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
X1XXmXXXXXXXXXXXXX
X*XXmXfXXXXXXX0*XX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XmXXmXXXXXXXXXXXXX
X1XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXfXXXXXX0**XX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X1XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
XXXXmXfXXXXX0**XXX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
```

```
X*1XmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
XXXXmXfXXXX0**XXXX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X**1mXXXXXXXXXXXXX
X*XXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfXXX0**XXXXX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X***1XXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfXXX0**XXXXX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
```

```
X****XXXXXXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXmXXXXXXXXXXXXX
XXXXmXfX0**XXXXXXX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X****XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXmXf0**XXXXXXXX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

X****XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX1X&**XXXXXXXXX
XXXXkXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX
```

```
X****XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX@&**XXXXXXXXX
XXXX1XXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX

XX***XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX*XXXXXXXXXXXXX
XXXX@XXXXXXXXXX0X
XXXX*XXXXXXXXXXXXX
XXXX1XXXXXXXXXXXXX

utkfureszfursz.map:
XXXXXXXXXXX
X1XfXXXfX0X
XXXXXXXXXXX
```

```
utkfureszfursz.gout:
XXXXXXXXXXX
X1XfXXXfX0X
XXXXXXXXXXX

XXXXXXXXXXX
X*1fXXXf0*X
XXXXXXXXXXX

XXXXXXXXXXX
X**&XXX&**X
XXXXXXXXXXX

XXXXXXXXXXX
XX**&X&**XX
XXXXXXXXXXX

XXXXXXXXXXX
X1XXXXXXX0X
XXXXXXXXXXX
```

6.4. A tesztelést támogató programok tervei

6.4.1. Tesztelő szkript

A tesztelő szkript egy konzolban futtatható *.bat* fájl, ami meghívja a programunkat a tesztfájlokat adva neki argumentumként. Ekkor a program felhasználói beavatkozás nélkül végrehajtja a tesztfájlban leírt utasításokat, és létrehozza az eseményeket, valamint a karakteres-grafikus kimenetet tartalmazó fájlokat. (*.out, .gout*). A létrejött fájlokat összehasonlítjuk a referenciafájlokkal. Ezt a *Mydiff* program segítségével tesszük meg, paraméterül adva a két összehasonlítandó fájl nevét. Az esetleges eltéréseket, hibaüzeneteket a *Mydiff* írja ki. Ezek alapján a szkript így fog kinézni: (A szkeleton tesztelésénél tapasztaltakból okulva ezúttal az ékezetektől eltekintünk a kimenetekben.)

```
echo "Teszteles inditasa"
REM Főprogram futtatása
java Control.CONTROL init.test
REM Megtörtént események és az elvárt kimenetek összehasonlítása
java TestProgs.Mydiff init.out init.ref
REM Grafikus kimenet ellenőrzése
java TestProgs.Mydiff init.gout init.gref
REM és így tovább az összes .test fájlra...
echo "Teszteles vege."
```

6.4.2. Mydiff

Ez a program a parancssori argumentumában kapott két fájlt hasonlítja össze soronként. Alapvetően minden karaktert összehasonlít, és ha különböznek, akkor a hibás sor számát és a karakter soron belüli helyét kiírja a kimenetre. Az összehasonlításnál azonban az üres mező és a bogyó között nem teszünk különbséget, mivel a bogyó újrakeletkezési helye véletlenszerűen kiválasztott, így a referenciafájlban ezt nem tudhatjuk előre. Az összes sor összehasonlítása után, ha nem talált különbséget, `OK` kiírása után kilép.

6.4.3. Snapshotviewer

Azért, hogy a képernyőképeket könnyebben ellenőrizhessük, akár kézileg is, szükségünk van egy a képernyőképeket egymás után lejátszó programra, amely nem konzolban fut. Erre szolgál a Snapshotviewer, ami mindössze egy ablakból (Frame-ből) áll, amin megtalálhatunk egy TextArea-t – amiben a pálya képét jelenítjük meg –, valamint két gombot az előre és hátra lapozáshoz. A beolvasandó fájlt a program parancssori argumentumaként adjuk meg.

7. fejezet

Prototípus beadása

7.1. Prototípus fordítása és futtatása

Először is a forrásfájlok eléréséhez a fájlokat ki kell tömöríteni a *ddt_prototipus.zip*-ből, majd az operációs rendszertől függően fordíthatjuk a programot:

7.1.1. DOS parancssorból

Futtassuk a *compile.bat* fájlt, ennek hatására elkészülnek a forrásokhoz a *.class* fájlok is, így futtathatóvá válik a program.

Ezután indítsuk el a *run.bat* fájlt, ennek hatására megjelenik konzolos felületen a program, amelynek használatáról később lesz szó.

Amennyiben az automatizált tesztet szeretnénk futtatni akkor a *test.bat* fájlt kell elindítani. Ekkor a kimeneten láthatjuk az egyes tesztesetek nevét és a tesztelés eredményét, amelyet sikeres lefutás után egy *OK*-val jelzünk a felhasználó számára.

Ha pedig a JAVA dokumentációt szeretnénk legeneráltatni, akkor a *javadocgen.bat* nevű fájlt kell futtatnunk.

7.1.2. UNIX shell-ből

Amennyiben UNIX alapú operációs rendszerünk van, a kicsomagolási mappában gépeljük be az alábbi parancsot a kód lefordításához:

```
javac -encoding UTF8 MODEL/*.java CONTROL/*.java
      VIEW/*.java SNAPSHOTVIEWER/*.java TestProgs/*.java
```

A program elindításához a következő sorra lesz szükség:

```
java CONTROL.Control
```

A képernyőkép-nézegető indításához ezt kell beírnunk, ahol az argumentum egy *.gout* vagy *.gref* kiterjesztésű fájl neve:

```
java SNAPSHOTVIEWER.SnapshotViewer [argumentum]
```

Az egyes tesztesetek helyességének ellenőrzéséhez az alábbi kód szükséges, itt az egyes argumentumok *.out* vagy *.gout* kiterjesztésű fájlok nevei:

```
java TestProgs.Mydiff [argumentum] [argumentum]
```

A JavaDoc dokumentációt pedig a következő paranccsal generálhatjuk le:

```
javadoc -private -verbose -encoding UTF8 -charset UTF8 -d
        javadoc -header "<h1>DDT - Demonic Development
        Team </h1>" -windowtitle "DDT - Demonic
        DevelopmentTeam" -docencoding UTF8 CONTROL/*.java
        MODEL/*.java VIEW/*.java TestProgs/*.java
        SNAPSHOTVIEWER/*.java
```

7.2. Tesztprogramok használata

7.2.1. Tesztesetek futtatása

A tesztesetek automatikus és gyors futtatásának megkönnyítésére egy szkript fájlt használunk, melynek neve *test.bat*. Ez a szkript kiírja a teszteset rövid leírását a képernyőre, majd a megfelelő *.test* kiterjesztésű parancsfájllal meghívja a prototípust megvalósító programot. Ekkor, amennyiben a program nagyobb hibák nélkül lefutott, létrejön egy *.out* illetve egy *.gout* kiterjesztésű fájl a *TEST* könyvtárban. (A fájlok nevei megegyeznek a tesztfájl nevével.) Ezek a fájlok tárolják a program grafikus kimenetét (*.gout*), valamint a program működéséről beszámoló naplóbejegyzéseket (*.out*). Ezek után a szkript meghívja a Mydiff programot, amely összehasonlítja a kimeneteket az elvárt kimenetekkel. Az elvárt kimenetek a megfelelő *.gref* és *.ref* fájlokban találhatók, szintén a *TEST* könyvtárban. Amennyiben a két fájl megegyezik, két *OK* íródik ki a képernyőre, egyéb esetekben hibaüzenettel találkozhatunk.

7.2.2. Mydiff

A *Mydiff* egy fájlok összehasonlítását végző program. Mindig pontosan két parancssori argumentummal kell meghívni, a két összehasonlítandó fájllal. Az összehasonlításkor érzékeli, hogy grafikus kimenetről, vagy naplófájlról van-e szó, és aszerint végzi az összehasonlítást. A grafikus kimenetnél az üres mező, és a bogyók között nem teszünk különbséget, mivel a program ezek helyét véletlenszerűen választja ki. (Nemdeterminisztikus módban.)

7.2.3. Snapshotviewer

Azért, hogy a grafikus kimenetet kényelmesen tudjuk ellenőrizni, készítettünk egy GUI-val rendelkező kis programot, ami a kimentett karakteres képernyőképeket egymás után levetíti, így könnyen követhetőek a képek közti változások, események. Lehetőség van kézi vezérlésre, ekkor az ablak alsó részén lévő nyilakkal válthatunk a képek közt, vagy animálhatjuk a képeket, ekkor azok automatikusan váltják egymást. Ez a funkció az Animate gomb segítségével hívható elő. Indításkor meg kell adnunk parancssori argumentumként a beolvasandó *.gout* vagy *.gref* fájl nevét.

7.2.4. A prototípus használata

A prototípust alapvetően kétféle módban futtathatjuk: fájl módban és a konzol módban. Fájl módban a programnak parancssori bemenetként megadunk egy *.test* kiterjesztésű parancsfájlt. Ezután a program ebből kiolvassa és végrehajtja a parancsokat. A program ekkor a képernyőre csak a hibaüzeneteket írja ki, felhasználói beavatkozás nem lehetséges. A képernyőképeket egy a parancsfájl nevével azonos nevű, de *.gout* kiterjesztésű fájlba menti. A naplóbejegyzések egy hasonló nevű *.out* fájlba kerülnek.

Konzol módban a program nem kap semmilyen parancssori argumentumot, ezért a parancsokat az alapértelmezett bemenetről várja. Először mindenképp be kell töltenünk a pályát az *uj_jatek «filename»* paranccsal, ezután már szabadon adhatjuk ki a többi parancsot, akár egy sorba az összeset (ügyelve az operandusok helyes megadására). Ha megun-

tuk a játszadozást, a *vege* paranccsal befejezhetjük a játékot. A kiadható parancsok listája a 7.1. táblázatban található. A pálya aktuális állása minden léptetés után kirajzolódik az alapértelmezett kimenetre. A naplóbejegyzések a *test.out* naplófájlba kerülnek.

Név	**Argumentumok**	**Leírás**
jobbra	«playerID»	Az adott játékos irányának beállítása.
balra	«playerID»	Az adott játékos irányának beállítása.
fel	«playerID»	Az adott játékos irányának beállítása.
le	«playerID»	Az adott játékos irányának beállítása.
[n]leptet	–	Kígyók léptetése.
uj_jatek	«filename»	Új játék kezdése a filename nevű pályán.
szunet	–	Játék szüneteltetése.
folytatas	–	Szüneteltetett játék folytatása.
vege	–	Játék vége.

7.1. táblázat. Parancsszavak és argumentumaik

Karakter	**ASCII kód**	**Játékobjektum**
#	(35)	fal
X	(79)	üres terület
0-9	(48-57)	adott számú kígyó feje
∗	(42)	kígyó testelem
m	(109)	mezei bogyó
f	(102)	fűrészbogyó
k	(107)	kőbogyó
&	(38)	fűrészfogas fej
@	(64)	köves testelem

7.2. táblázat. Karakteres kimenet jelmagyarázata

7.3. Tesztek jegyzőkönyvei

init.test

Teszt leírása: Minden elemből elhelyezünk a pályán egyet. Megnézzük helyesen jön-e létre, megfelelően kapcsolódnak-e az elemek.

Tapasztalat: A részletes tervekben a *.test* fájl nem tartalmazta a *vege* utasítást, ezért nem fejeződött be a program futása. A pálya betöltésekor a bogyók nem jöttek létre.

Módosítások: Megváltoztattuk a *.test* fájlt, valamint emiatt a *.ref*-et is. Javítottuk a *Model.loadMap()* függvényt.

---

startstop.test

Teszt leírása: Egy üres pályát létrehozunk, melyen léptetjük a kígyókat, közben pedig szüneteltetjük, majd folytatjuk a játékot. Végül befejezzük.

Tapasztalat: A játék nem állt le szüneteltetéskor egy elírás miatt.

Módosítások: A *Model*-ben kijavítottuk a *stop()* függvényt.

---

irany.test

Teszt leírása: Egy kígyónak egy üres pályán több irányt adunk meg mielőtt lépne, megnézzük, hogy az utoljára megadott irányba halad-e.

Tapasztalat: A kígyó nem mozdult el a helyéről. Ennek oka, hogy állandóan magától meghalt a hibás *digestStone()* hívások miatt. Ezt a hibát javítva a kígyók megfelelően mozogtak. Csak nem azok, amelyikeknek kellett volna, mivel a kígyók azonosítása hibás volt.

Módosítások: *SnakeHead* osztályba felvettünk egy *eatStone()* függvényt, ami akkor hívódik meg, ha felvettünk egy kőbogyót, a *digestStone()* pedig csak akkor, ha van is a kígyóban kő. A *Model.loadMap()* függvényben a kígyók starthelyeinek kígyókhoz való hozzárendelését helyrehoztuk, így az irányításuk és számozásuk egyértelművé vált.

---

mozgas.test

Teszt leírása: Két kígyó egy üres pályán mind a négy irányba mozog.

Tapasztalat: A várt kimenetnek megfelelő eredményeket kaptunk.

Módosítások: Nincs.

---

mezeibogyo.test

Teszt leírása: A kígyó rálép egy mezei bogyóra. Megnézzük helyesen veszi-e fel és nő-e tőle. A bogyó újraterem-e üres helyen.

Tapasztalat: A várt kimenetnek megfelelő eredményeket kaptunk.

Módosítások: Nincs

---

fureszbogyo.test

Teszt leírása: A kígyó felvesz egy fűrészbogyót. Megnézzük, hogy helyesen veszi-e fel és fűrészes állapotba kerül-e meghatározott ideig, az állapot elmúlik-e.

Tapasztalat: A várt kimenetnek megfelelő eredményeket kaptunk.

Módosítások: Nincs

---

kobogyo.test

Teszt leírása: A kígyó felvesz egy kőbogyót. Megnézzük helyesen veszi-e fel és emészti-e meg.

Tapasztalat: A várt kimenetnek megfelelő eredményeket kaptunk.

Módosítások: Nincs

---

utkozdarabsajat.test

Teszt leírása: A kígyó visszakanyarodik és egy saját darabjának ütközik.

Tapasztalat: A program *StackOverFlowError*-ral elszállt, mert a *SnakeHead die()* függvénye végtelen ciklusba került (a test mentén a *die()* hívási lánc végén a megharapott faroknál egy újabb ütközéskezelő függvény ismét megölte a *SnakeHead*-et).

Módosítások: Biztosítottuk a *SnakeHead.die()* függvényt rekurzív hívások ellen, így ha már benne vagyunk a függvényben, újabb hívásakor azonnal visszatér.

---

utkozdarab.test

Teszt leírása: Az egyik kígyó nekiütközik a másik kígyó egy darabjának.

Tapasztalat: A várt kimenetnek megfelelő eredményeket kaptunk.

Módosítások: Nincs

---

utkozfej.test

Teszt leírása: A két kígyó fejjel egymásnak megy.

Tapasztalat: A várt kimenetnek megfelelő eredményeket kaptunk.

Módosítások: Nincs

---

utkozko.test

Teszt leírása: Egy kővel teli kígyó kőbogyót próbál meg felvenni.

Tapasztalat: A várt kimenetnek megfelelő eredményeket kaptunk.

Módosítások: Nincs

---

utkozfal.test

Teszt leírása: Egy kígyó egy faldarabnak megy.

Tapasztalat: A várt kimenetnek megfelelő eredményeket kaptunk.

Módosítások: Nincs

---

utkozfuresz.test

Teszt leírása: A kígyó felvesz egy fűrészbogyót, és ezután a másik kígyót kettéharapja. Megnézzük az új zombi kígyó helyesen jön-e létre.

Tapasztalat: Először nem jelent meg az új kígyó feje a grafikus kimeneten, de a következő lépésnél már helyes volt a megjelenítés és a továbbiakban a kígyó megfelelően mozgott.

Módosítások: A hiba oka a *SnakeElement.becomeNewSnake()* metódusban volt: egy elmulasztott referencia-beállítás (a mező nem kapta meg a rajta lévő fej referenciáját).

---

utkfuresznyak.test

Teszt leírása: A kígyó felvesz egy fűrészbogyót, és ezután a másik kígyó nyakába harap. Megnézzük, hogy a maradék kígyó a rövidsége miatt meghal-e.

Tapasztalat: A fej felőli kígyószakasz csak egy lépéssel később halt meg, mint kellett volna.

Módosítások: A hiba oka az volt, hogy a kígyó hosszába beleszámoltuk a már halott, de még nem leválasztott farokrészeket is, ezt javítottuk.

---

utkozfureszko.test

Teszt leírása: Egy fűrészfogas kígyó egy olyan kígyódarabnak ütközik, amiben kő van.

Tapasztalat: A várt kimenetnek megfelelő eredményeket kaptunk.

Módosítások: Nincs

---

utkfureszfuresz.test

Teszt leírása: Két fűrészbogyós kígyó egymással szemben ütközik.

Tapasztalat: A várt kimenetnek megfelelő eredményeket kaptunk.

Módosítások: Nincs

---

szabadkézi tesztelés

Teszt leírása: Különböző *conf.ini* paraméterek hatásának vizsgálata. *Conf.ini* fájl tartalmának beolvasása.

Tapasztalat: A kőbogyó-emésztési sebesség módosításakor a működés nem volt megfelelő, ha az emésztési sebesség kisebb volt mint a mozgási sebesség.

Módosítások: Hiba oka: az ezt kezelő állapotváltozó értéke akkor is változott, amikor a kígyóban nem volt még kőbogyó, így az emésztés gyakorlatilag előre megkezdődött. Ezt a hibát is javítottuk.

7.4. Értékelés

A prototípus befejezéséig a csapat minden tagja egyformán jól teljesített. Még ha az egyes beadásoknál nem is volt teljesen egyenlő a munkamegosztás, összességében arra törekedtünk, hogy mindenki ugyanannyit dolgozzon, és lehetőleg olyan feladatokat kapjon, amelyek számára könnyebbek, szimpatikusabbak. A tagok egyhangúan amellett döntöttek, hogy mindenki egyformán részesüljön a pontokból.

A tesztelés során előjött hibák nagyrészt elírásokból, vagy kihagyásokból adódtak. Ezen hibák egy részét a tesztelő programunk segítségével gyorsan és egyszerűen megtaláltuk, más részüket kemény munkával debuggoltuk ki, így fontos tapasztalatokat szereztünk ezen eszköz használatával kapcsolatban. Bebizonyosodott, hogy a specifikációnk és a modellünk megfelelő volt, nem volt szükség sehol sem lényegi változtatásra, csupán implementációs hibák javítására.

Név	**Elvégzett munkahányad**
Csuzdi	25%
Fehér	25%
Györgyey^{CSK}	25%
Major	25%

7.5. Fájllista

A *ddt_prototipus.zip* tartalma:

Utolsó módosítás	Idő	Fájlméret (byte)	Fájlnév	Leírás
2008-04-16	19:54:50	205	conf.ini	
2008-04-16	22:07:48	4520	Conf.java	
2008-04-16	22:09:04	13569	Control.java	
2008-04-14	19:00:42	55	fureszbogyo.map	*Térkép fájlok*
2008-04-14	19:11:34	14	init.map	
2008-04-14	19:00:42	21	irany.map	
2008-04-14	19:00:42	28	kobogyo.map	
2008-04-14	19:00:42	35	mezeibogyo.map	
2008-04-14	19:00:42	45	mozgas.map	
2008-04-14	19:00:42	28	startstop.map	
2008-04-14	19:00:42	39	utkfureszfuresz.map	
2008-04-14	19:00:42	120	utkfuresznyak.map	
2008-04-14	19:00:42	40	utkozdarab.map	
2008-04-14	19:00:42	25	utkozfal.map	
2008-04-14	19:00:42	40	utkozfej.map	
2008-04-14	19:00:42	120	utkozfureszko.map	
2008-04-14	19:00:42	160	utkozfuresz.map	
2008-04-14	19:00:42	35	utkozko.map	
2008-04-14	19:00:42	35	utkozsajat.map	
2008-04-16	05:25:54	1752	Berry.java	
2008-03-19	01:33:40	485	Direction.java	
2008-04-07	21:11:12	266	FieldBerry.java	
2008-04-07	21:06:54	4602	FieldElement.java	
2008-04-16	05:25:54	6356	GameField.java	
2008-04-14	23:45:02	1293	GameObject.java	
2008-04-07	21:11:12	465	Int2.java	
2008-04-16	22:32:52	8935	Model.java	
2008-04-07	21:11:12	255	SawBerry.java	
2008-04-15	19:53:10	877	SnakeBody.java	
2008-04-16	15:36:22	8850	SnakeElement.java	
2008-04-16	15:36:46	6220	SnakeHead.java	
2008-04-16	17:53:44	5257	Snake.java	
2008-04-07	21:11:12	275	StoneBerry.java	
2008-04-07	21:11:12	554	Wall.java	
2008-04-16	21:58:30	9028	SnapshotViewer.java	
2008-04-14	19:14:38	625	fureszbogyo.gref	*Referencia fájlok*
2008-04-16	15:13:08	320	fureszbogyo.ref	

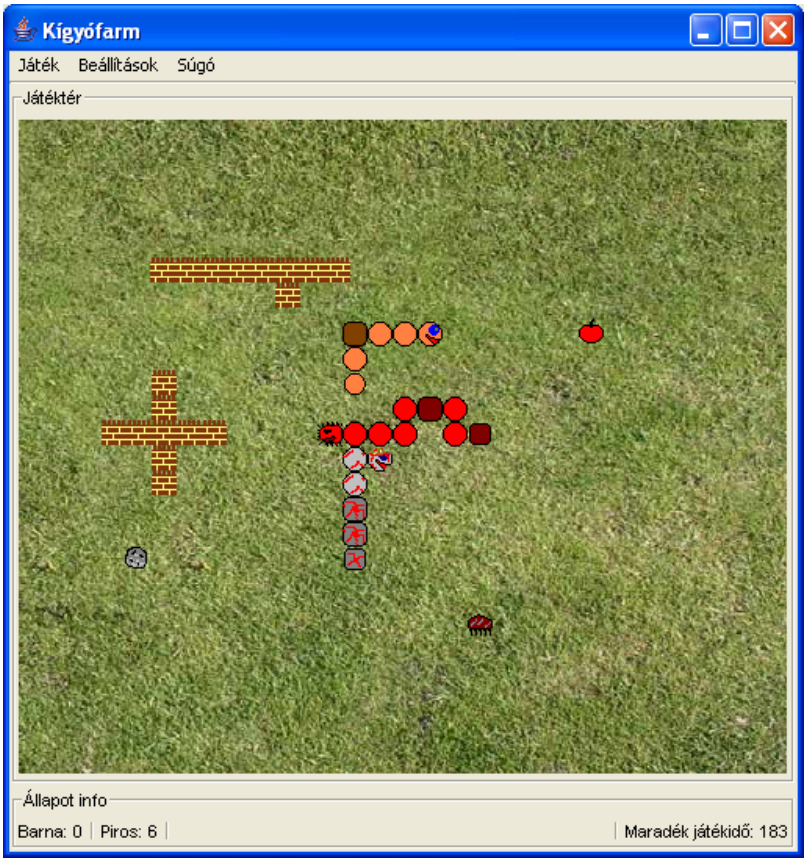
2008-04-14	19:31:14	47	fureszbogyo.test
2008-04-14	19:12:34	14	Init.gref
2008-04-16	15:13:08	90	Init.ref
2008-04-16	15:13:08	23	Init.test
2008-04-14	19:14:38	44	irany.gref
2008-04-16	15:13:08	109	irany.ref
2008-04-14	19:31:14	89	irany.test
2008-04-14	19:14:38	190	kobogyo.gref
2008-04-16	15:13:08	189	kobogyo.ref
2008-04-14	19:31:14	42	kobogyo.test
2008-04-14	19:14:38	257	mezeibogyo.gref
2008-04-16	15:13:08	212	mezeibogyo.ref
2008-04-14	19:31:14	45	mezeibogyo.test
2008-04-14	19:14:38	421	mozgas.gref
2008-04-16	15:13:08	242	mozgas.ref
2008-04-14	19:31:14	132	mozgas.test
2008-04-14	19:14:38	118	startstop.gref
2008-04-16	15:13:08	166	startstop.ref
2008-04-14	19:31:14	71	startstop.test
2008-04-14	19:14:38	203	utkfureszfuresz.gref
2008-04-16	15:13:08	326	utkfureszfuresz.ref
2008-04-16	12:18:58	62	utkfureszfuresz.test
2008-04-16	12:30:02	1584	utkfuresznyak.gref
2008-04-16	15:13:08	596	utkfuresznyak.ref
2008-04-16	13:20:54	91	utkfuresznyak.test
2008-04-14	19:14:38	208	utkozdarab.gref
2008-04-16	15:13:08	212	utkozdarab.ref
2008-04-14	19:31:14	73	utkozdarab.test
2008-04-14	19:14:38	160	utkozfal.gref
2008-04-16	15:13:08	205	utkozfal.ref
2008-04-14	19:31:14	36	utkozfal.test
2008-04-14	19:14:38	250	utkozfej.gref
2008-04-16	15:13:08	277	utkozfej.ref
2008-04-14	19:31:14	55	utkozfej.test
2008-04-16	15:26:10	2266	utkozfuresz.gref
2008-04-16	15:46:20	1462	utkozfureszko.gref
2008-04-16	15:46:20	579	utkozfureszko.ref
2008-04-16	15:13:08	91	utkozfureszko.test
2008-04-16	15:13:08	569	utkozfuresz.ref
2008-04-14	19:31:14	89	utkozfuresz.test
2008-04-14	19:14:38	257	utkozko.gref
2008-04-16	15:13:08	314	utkozko.ref
2008-04-14	19:31:14	35	utkozko.test
2008-04-16	15:24:52	220	utkozsajat.gref
2008-04-16	15:13:08	273	utkozsajat.ref
2008-04-16	13:45:34	94	utkozsajat.test

2008-04-16	22:32:52	6512	Mydiff.java	
2008-04-16	05:25:54	6729	CLI.java	
2008-04-16	16:20:10	103	compile.bat	*Szkriptek*
2008-04-16	16:20:10	211	javadocgen.bat	
2008-04-16	16:20:10	24	run.bat	
2008-04-16	16:42:16	4304	test.bat	
2008-04-16	15:23:38	4509	test.sh	

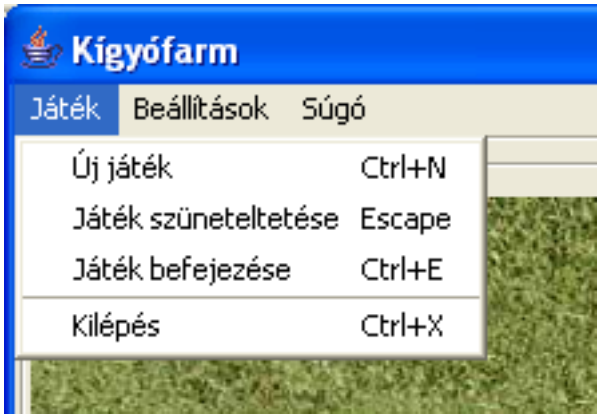
8. fejezet

Grafikus felület specifikálása

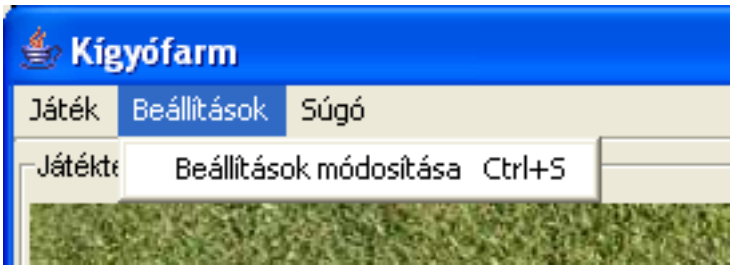
8.1. A kezelői felület grafikus képe



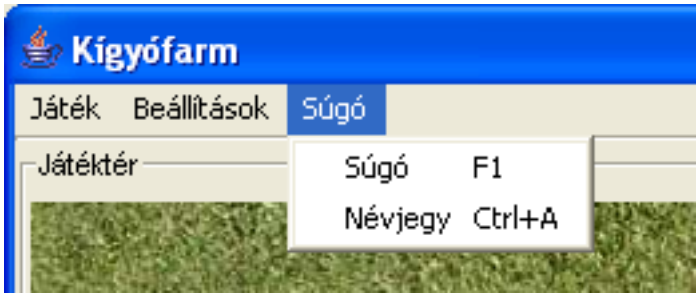
8.1. ábra. Felhasználói felület terve



8.2. ábra. Játék menü



8.3. ábra. Beállítások menü



8.4. ábra. Súgó

8.2. A grafikus felület működési elve

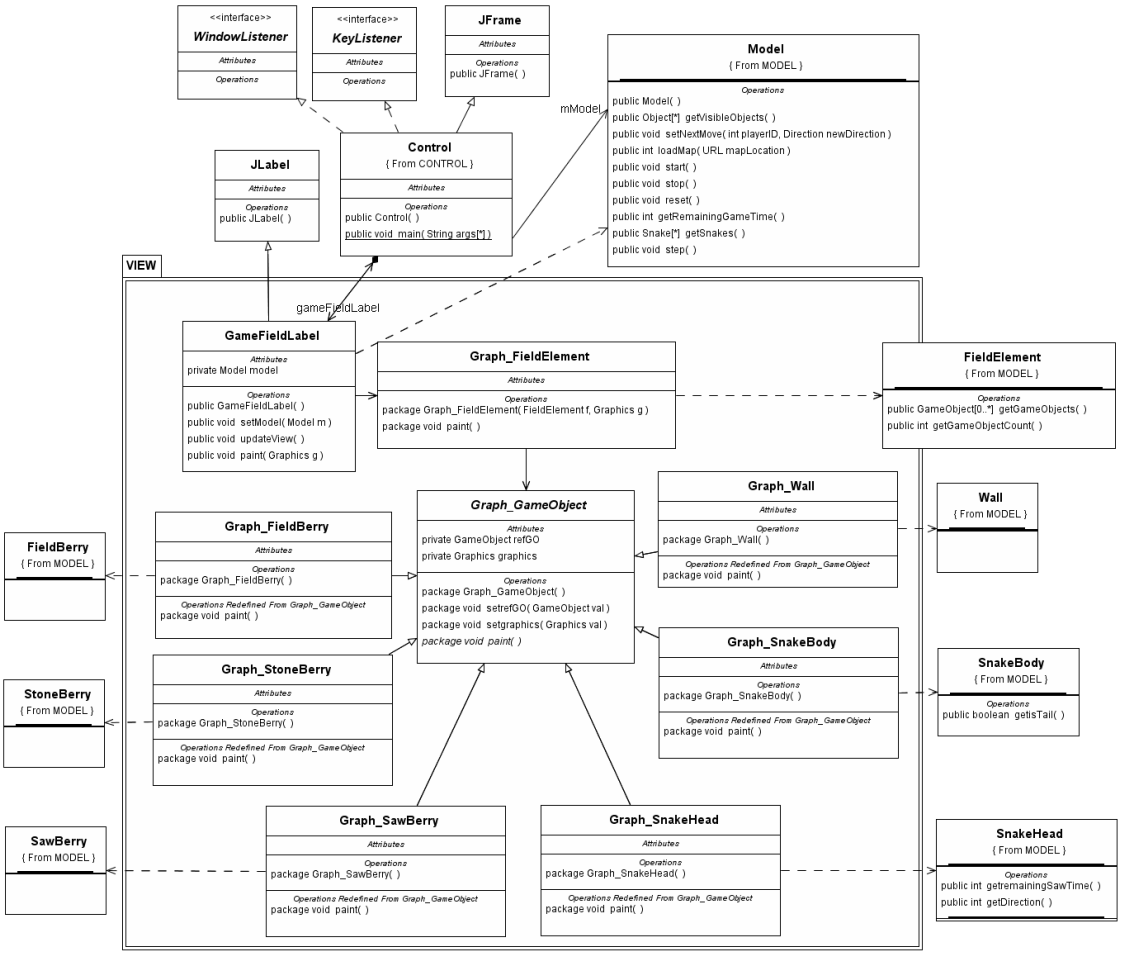
8.2.1. A grafikus felület szerkezete

A felületet két fő részre bonthatjuk, a modellt megjelenítő komponensre és az ezt tartalmazó kezelői felületre. A modell megjelenítéséért egy saját osztály, a *GameFieldLabel* a felelős. Ez referenciával rendelkezik a modellre, így egy lekérdezéssel elérhetjük a játék pillanatnyi állapotát, majd ez alapján ki tudjuk rajzolni azt (bővebben lásd később). A kezelői felület kialakításáért a *Control* osztály a felelős, mely a *javax.swing.JFrame* osztály leszármazottja, tehát ez maga a program főablaka. Az ablak tartalmaz továbbá egy menüsort, egy saját állapotsort és a játék állapotát megjelenítő *GameFieldLabel*-t. A *Control* felel azért, hogy a megjelenítés a megfelelő modell-referenciával rendelkezzen. Továbbá az osztály implementálja a *WindowListener* és *KeyListener* interfészeket, hogy a felhasználói beavatkozásokat le tudja kezelni (A *GameFieldLabel KeyListener*-e maga a *Control* lesz, így a felhasználói bevitel rögtön az ablakhoz kerül). A *Control* felelős a modell bizonyos időközönkénti léptetéséért (ehhez egy *Timer* áll rendelkezésére), majd minden léptetés után a grafikus felület frissítéséért.

8.2.2. A játékállás kirajzolása

A megjelenítésért felelős komponens a rajzterületre kirajzolja egymás után a pálya *Model*-től lekért mezőit. Ehhez minden mezőhöz létrehoz egy *Graph_FieldElement* objektumot, mely a konstruktorában megkapott mezőt a szintén a a konstruktorában megkapott rajzterületre (az egész rajzterület megfelelő részére) rajzolja. Ehhez a *Graph_FieldElement* lekéri a mezőn található *GameObject*-et. Ezeket az objektumokat eltároljuk egy ideiglenes panelen, majd mikor mindegyik elemet lehívtuk és megrajzoltuk az új képet a régi helyére tesszük, ezzel is biztosítva a játék villódzásmentes megjelenítését.

8.3. Statikus struktúra diagram



8.5. ábra. Grafikus osztályok struktúra diagramja

8.4. Grafikus osztályok leírása

Név	GameFieldLabel
Ősosztály	`JLabel`
Komponensei	-
Relációk	`A Control aggregálja, asszociációban van a Graph_FieldElement-tel.`
Példányok száma	1

Ezen a Label-en ábrázoljuk az egyes képernyőképeket.

Változók:

Típus	Név	Leírás
Model	***model***	Referencia a modell objektumra, hogy le tudjuk kérdezni a pályán lévő elemeket.

Metódusok:

Típus	Név	Leírás
void	***setModel(Model)***	Itt állíthatjuk be a Model-re mutató referenciát.
void	***paint(Graphics)***	Ezzel a függvénnyel definiáljuk felül a JLabel eredeti paint() függvényét.
void	***updateView()***	Ez a függvény felelős a pálya újrarajzolásáért, ha változott a modell állapota.

Név	Graph_FieldBerry
Ősosztály	`Graph_GameObject`
Komponensei	-
Relációk	`Graph_GameObject-ből származik, függ a FieldBerry-től`
Példányok száma	1

A mezei bogyók grafikus megjelenítéséért felelős osztály.

Metódusok:

Típus	Név	Leírás
void	***paint()***	Felüldefiniált függvény, mely a mezei bogyók grafikus megjelenítéséért felelős.

Név	Graph_FieldElement
Ősosztály	–
Komponensei	–
Relációk	Függ a FieldElement-től, illetve asszociál a Graph_GameObject-tel.
Példányok száma	1

Egy játékmező kirajzolásáért felelős osztály.

Metódusok:

Típus	Név	Leírás
void	***paint()***	Ezzel rajzoljuk ki a pálya egy játékmezőjét.

Név	Graph_GameObject
Ősosztály	–
Komponensei	–
Relációk	Belőle származnak az egyes pályaelemeket megjelenítő osztályok.
Példányok száma	nem példányosítható

Absztrakt ősosztály. Az egyes pályaelemek egységes kezeléséhez szükséges.

Változók:

Típus	Név	Leírás
GameObject	***refGO***	Referencia arra a GameObject-re, amelynek a kirajzolásáért felelős.
Graphics	***graphics***	A grafikus felület referenciája, amire refGO képe kirajzolandó.

Metódusok:

Típus	Név	Leírás
void	***paint()***	Absztrakt függvény a játékobjektum kirajzolására.
void	***setrefGO(GameObject)***	Itt állítjuk be a GameObject-re mutató referenciát.
void	***setgraphics(Graphics)***	Itt állítjuk be a Graphics-ra mutató referenciát.

Név	Graph_SawBerry
Ősosztály	`Graph_GameObject`
Komponensei	`-`
Relációk	`A Graph_GameObject-ből származik, függ a SawBerry-től.`
Példányok száma	`1`

Ez az osztály felelős a fűrészbogyók megjelenítéséért.

Metódusok:

Típus	Név	Leírás
void	***paint()***	Felüldefiniált függvény, amely a fűrészbogyó grafikus megjelenítéséért felelős.

Név	Graph_SnakeBody
Ősosztály	`Graph_GameObject`
Komponensei	`-`
Relációk	`A Graph_GameObject-ből származik, függ a SnakeBody-tól.`
Példányok száma	`1`

Ez az osztály rajzolja újra a kígyó törzsének elemeit.

Metódusok:

Típus	Név	Leírás
void	***paint()***	Felüldefiniált függvény, amely a kígyótest elemeinek megrajzolásáért felelős.

Név	Graph_SnakeHead
Ősosztály	`Graph_GameObject`
Komponensei	`-`
Relációk	`A Graph_GameObject-ből származik, függ a SnakeHead-től.`
Példányok száma	`1`

Ez az osztály felelős a kígyók fejének megjelenítéséért.

Metódusok:

Típus	Név	Leírás
void	***paint()***	Felüldefiniált függvény, amely a kígyó fejének grafikus megjelenítéséért felelős.

Név	Graph_StoneBerry
Ősosztály	`Graph_GameObject`
Komponensei	`-`
Relációk	`A Graph_GameObject-ből származik, függ a StoneBerry-től.`
Példányok száma	`1`

Ez az osztály felelős a pályán lévő kőbogyók megjelenítéséért.

Metódusok:

Típus	Név	Leírás
void	***paint()***	Felüldefiniált függvény, amely a pályán lévő kőbogyót rajzolja fel a megfelelő területre.

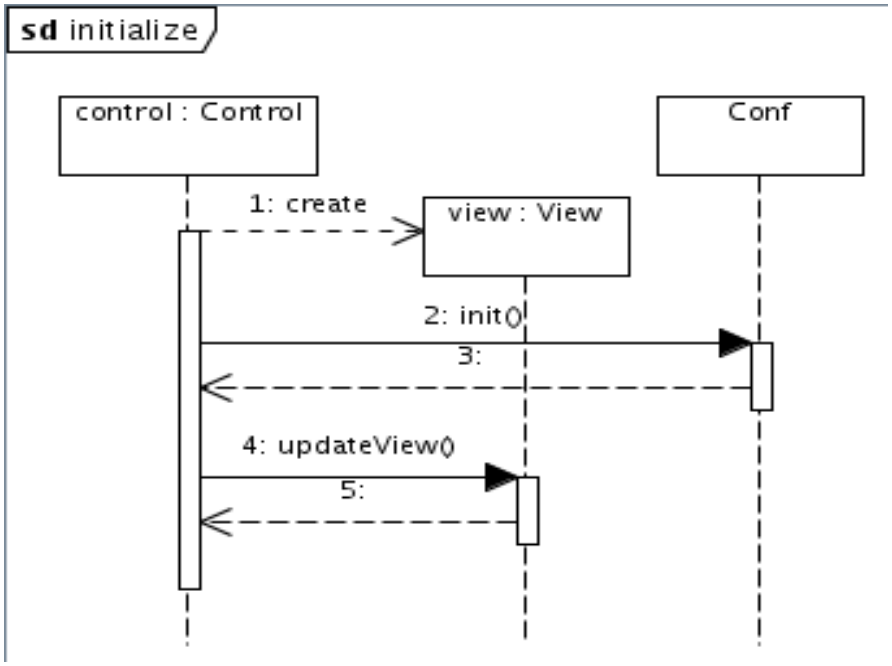
Név	Graph_Wall
Ősosztály	`Graph_GameObject`
Komponensei	–
Relációk	`A Graph_GameObject-ből származik, függ a Wall-tól.`
Példányok száma	1

Ez az osztály felelős a pályán lévő fal objektumok megjelenítéséért.

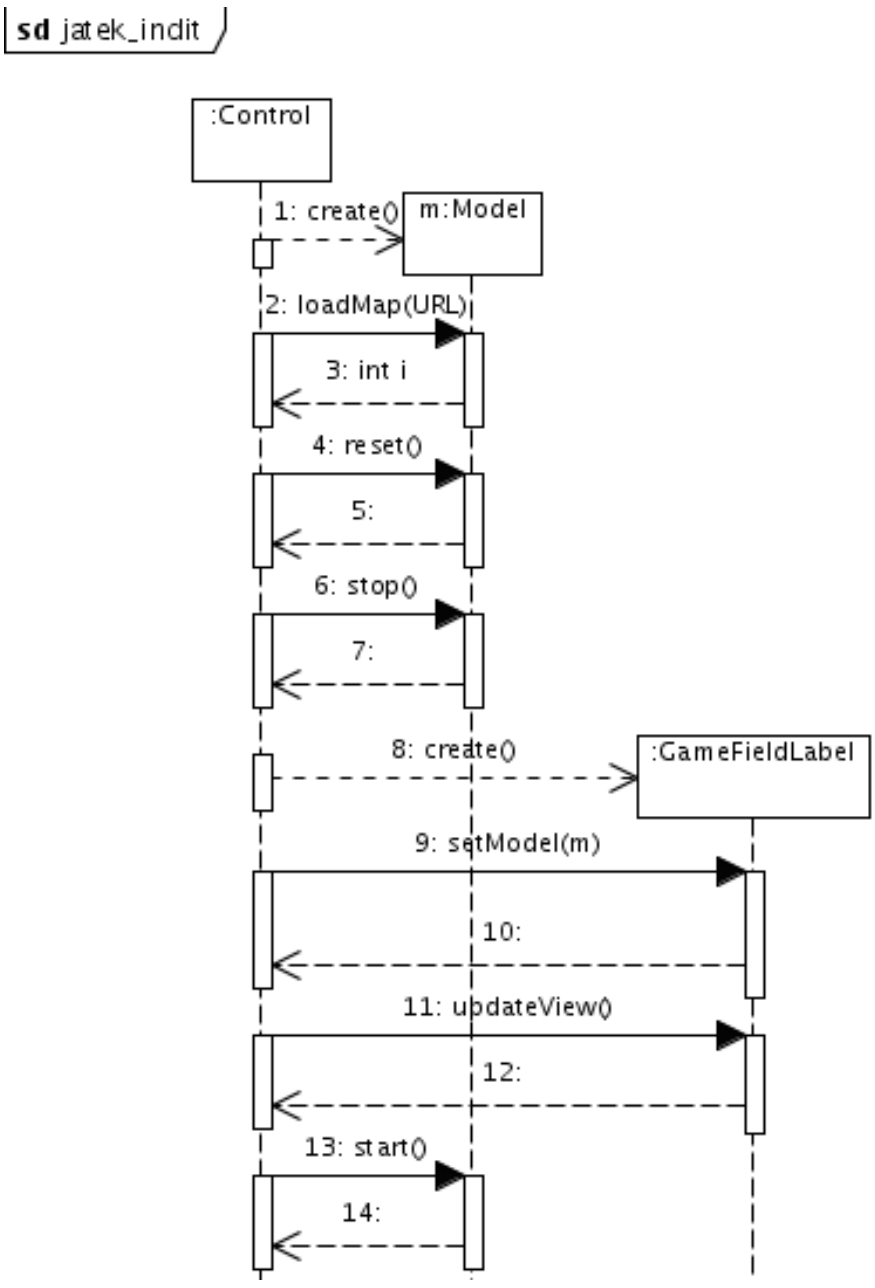
Metódusok:

Típus	Név	Leírás
void	***paint()***	Felüldefiniált függvény, mely a fal grafikus megjelenítéséért felelős.

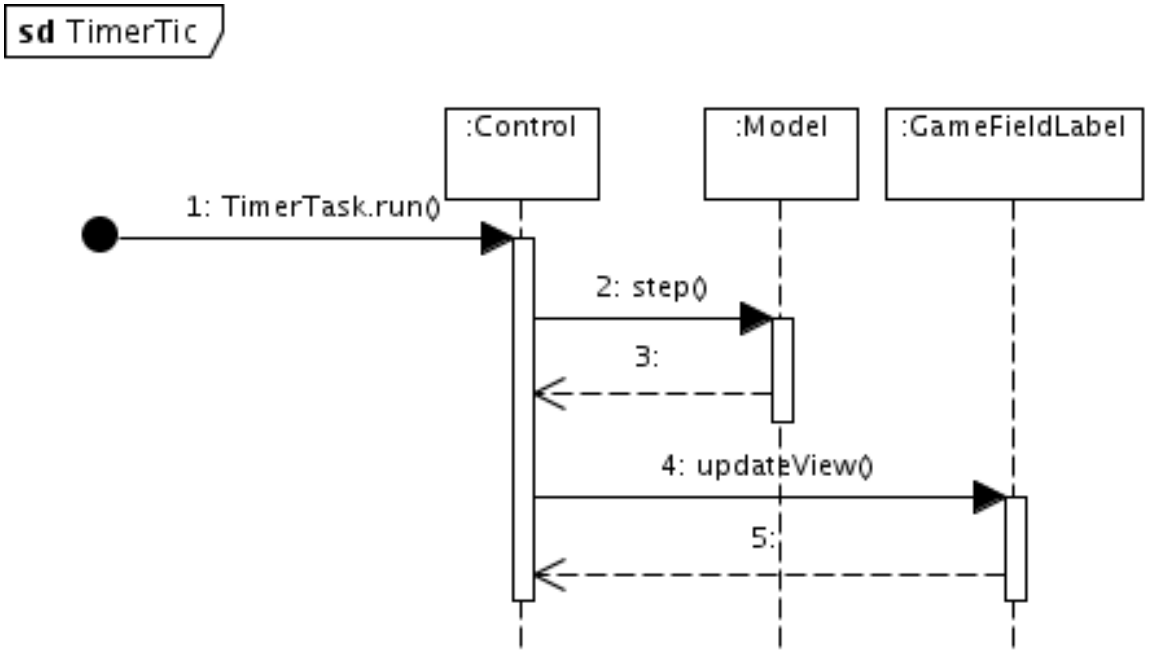
8.5. Szekvencia diagramok



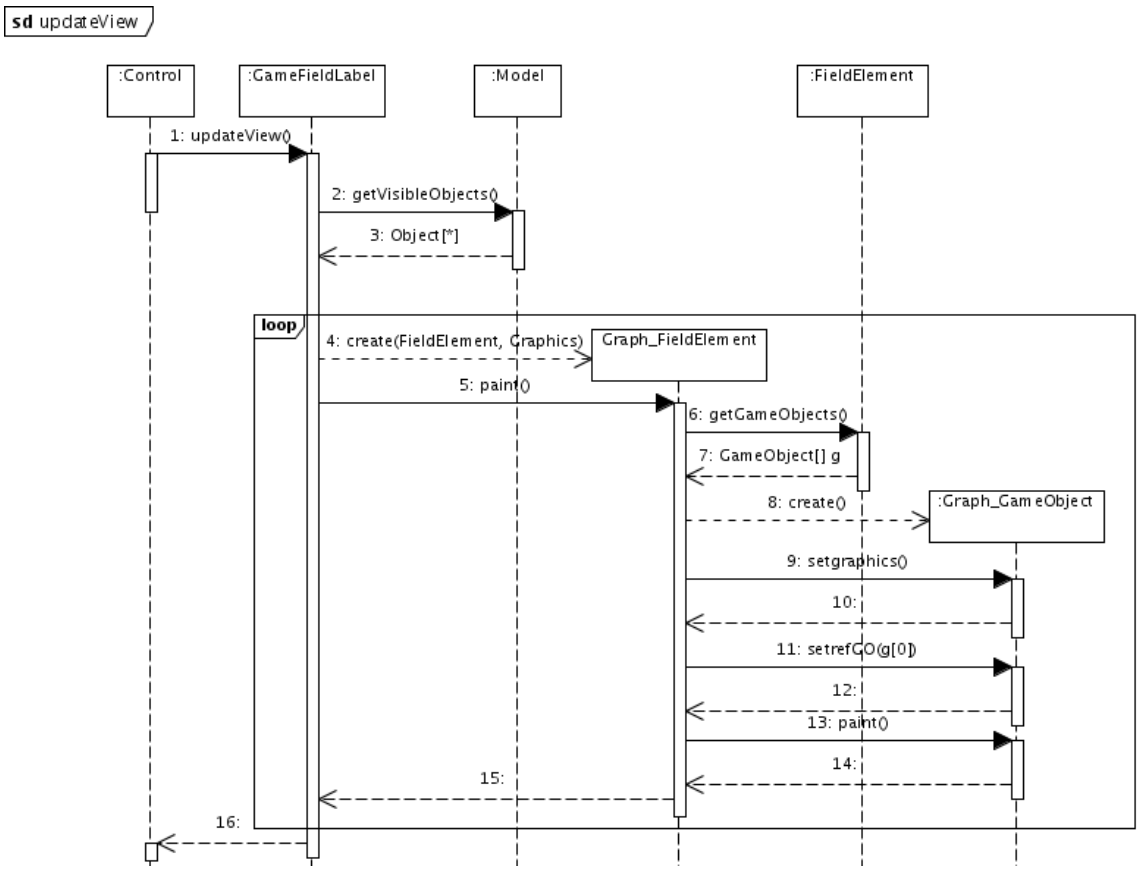
8.6. ábra. Inicializálás



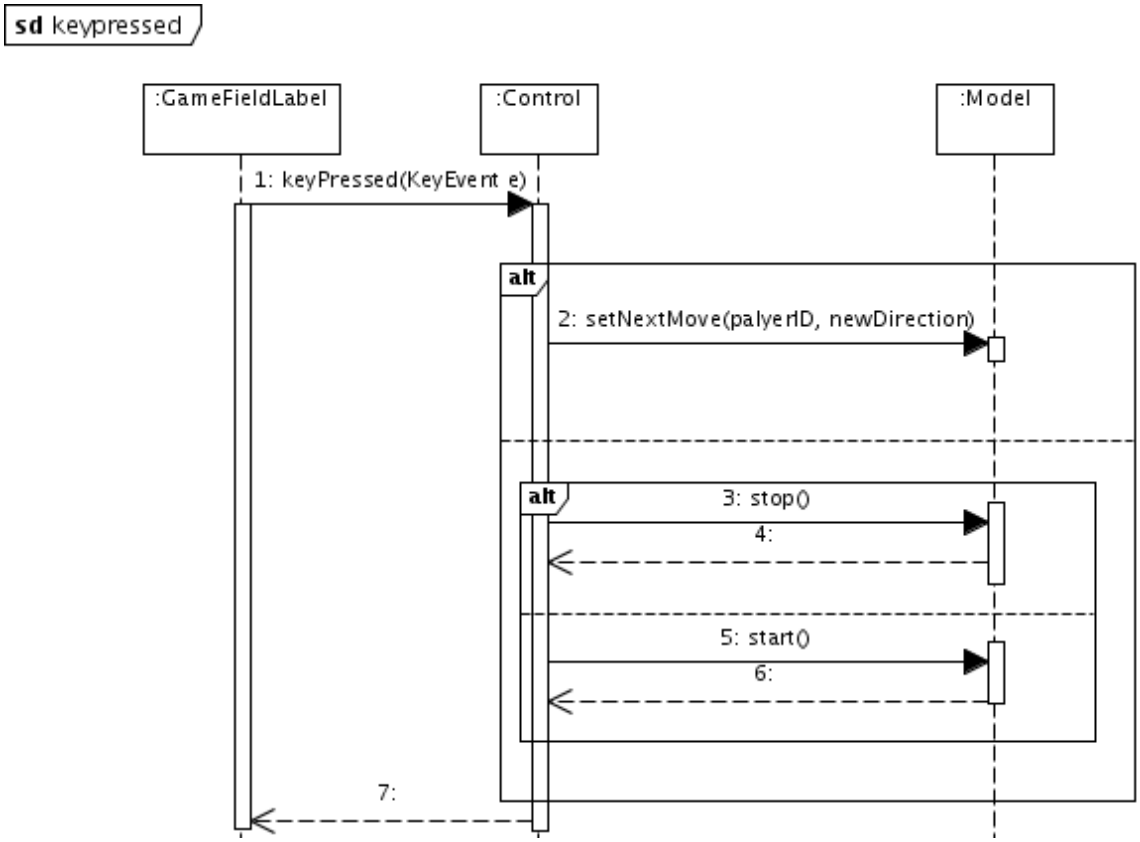
8.7. ábra. Játék indítása



8.8. ábra. Játék léptetése



8.9. ábra. Játéktér frissítése



8.10. ábra. Billentyűleütés feldolgozása

9. fejezet

Grafikus változat beadása

9.1. Grafikus változat fordítása és futtatása

Először is a forrásfájlok eléréséhez a fájlokat ki kell tömöríteni a *ddt_grafikus.zip*-ből, majd az operációs rendszertől függően fordíthatjuk a programot:

9.1.1. DOS parancssorból

Az installálható változat előállításához futtassuk a *compile.bat* fájlt, ezután a *dist* mappa tartalmazza a telepítéshez szükséges fájlokat.
A telepítés menete: A *dist* mappa teljes tartalmát másoljuk át a kívánt telepítési mappába.
Futtatás: A program a telepítési mappában a *run.bat* fájl futtatásával indítható el.
Ha a JAVA dokumentációt szeretnénk legeneráltatni, akkor a *javadocgen.bat* nevű fájlt kell futtatnunk.

9.1.2. UNIX shell-ből

Amennyiben UNIX alapú operációs rendszerünk van, a kód lefordításához az alábbi parancsot kell begépelnünk a kitömörítési mappába:

```
javac -encoding UTF8 MODEL/*.java CONTROL/*.java
      VIEW/*.java
```

A program elindításához a következő sorra lesz szükség:

```
java CONTROL.Control
```

A JavaDoc dokumentációt a következő paranccsal generálhatjuk le:

```
javadoc -private -verbose -encoding UTF8 -charset UTF8 -d
        javadoc -header "<h1>DDT - Demonic Development
        Team </h1>" -windowtitle "DDT - Demonic
        DevelopmentTeam" -docencoding UTF8 CONTROL/*.java
        MODEL/*.java VIEW/*.java
```

9.2. Értékelés

A projekt során arra törekedtünk, hogy mindenki egyenlően vegye ki a részét a munkából. Nagyrészt sikerült is ez alapján szétosztani a feladatokat, ezért egyhangúan úgy döntöttünk, hogy mindenki egyenlően részesüljön a pontokból. Az eltelt idő során többször fordult elő eltérő vélemény a csapattagok között, de mindig hamar megegyeztünk kompromisszumokkal. Egyszer fordult csak elő késés, mert idő hiányában későn álltunk neki a feladatnak, és inkább az igényes munkát választottuk egy esetleges elkapkodott helyett. Végül így egy nagyon jó, használható játékot sikerült készítenünk. Már a prototípusnál bebizonyosodott, hogy a modellünk megfelelő, így a grafikus rész megírásánál csak apróbb, implementációs hibák kerültek elő.

Név	**Elvégzett munkahányad**
Csuzdi	25%
Fehér	25%
Györgyey^{CSK}	25%
Major	25%

9.3. Módosítások jegyzéke

(A modell osztályainak utólagos módosításai)

SnakeBody - bővítés: Az isTail változó kezelése a protóból kimaradt, hiszen jelentősége csak a grafikus programnál van, így e fölött korábban átsiklottunk. A kezeléséhez a *grow()* és *move()* függvényeket kellett módosítani.

SnakeHead - hibajavítás: A fej eddig bevette a mozgási irányával ellentétes új irányt, és önmagába visszafordulva meghalt. A *setnewDirection()* függvény javítva.

SnakeElement - hibajavítás: Zombi kígyó létrejöttekor a playerID-je nem állítódott át rendesen. A megoldáshoz a *setplayerID()* függvény rekurzívvá vált.

Model - hibajavítás és bővítés: A léptetés során a zombi kígyók nem haltak meg, ha túl röviddé váltak, valamint haláluk után nem tűntek el, a modell továbbra is léptette őket. A *step()* függvény javítva. Újításként a zombi kígyók véletlenszerűen mozognak minden lépéskor. (A beállításoknál az "MI mozgékonyság" paramétert 0%-ra állítva a kígyók az eredeti specifikációnak megfelelően csak egyenes vonalban mozognak.)

9.4. Fájllista

A *ddt_grafikus.zip* tartalma:

Utolsó módosítás	Idő	Fájlméret (byte)	Fájlnév	Leírás
2008-05-07	14:32:00	618	compile.bat	
2008-05-07	14:08:00	172	javadocgen.bat	
2008-04-16	16:20:00	24	run.bat	
				MODEL
2008-05-06	23:33:00	1752	Berry.java	
2008-03-19	01:33:00	485	Direction.java	
2008-04-07	21:11:00	266	FieldBerry.java	
2008-04-29	01:32:00	4609	FieldElement.java	
2008-04-16	05:25:00	6356	GameField.java	
2008-04-14	23:45:00	1293	GameObject.java	
2008-04-07	21:11:00	465	Int2.java	
2008-05-07	13:11:00	10388	Model.java	
2008-04-07	21:11:00	255	SawBerry.java	
2008-04-29	17:58:00	1496	SnakeBody.java	
2008-04-29	19:25:00	8926	SnakeElement.java	
2008-05-06	23:36:00	5974	SnakeHead.java	
2008-05-07	14:03:00	5221	Snake.java	
2008-04-07	21:11:00	275	StoneBerry.java	
2008-04-07	21:11:00	554	Wall.java	
				VIEW
2008-05-07	10:03:00	5330	GameFieldLabel.java	
2008-05-07	09:51:00	829	Graph_FieldBerry.java	
2008-05-07	09:52:00	1997	Graph_FieldElement.java	
2008-04-29	01:54:00	1480	Graph_GameObject.java	
2008-05-07	09:52:00	830	Graph_SawBerry.java	
2008-05-07	09:57:00	2062	Graph_SnakeBody.java	
2008-05-07	10:00:00	2357	Graph_SnakeHead.java	
2008-05-07	10:00:00	822	Graph_StoneBerry.java	
2008-05-07	10:01:00	807	Graph_Wall.java	
2008-05-07	13:54:00	39362	SettingsDialog.java	
				CONTROL
2008-05-07	13:54:00	652	conf.ini	
2008-05-07	10:01:00	20107	Conf.java	
2008-05-07	13:52:00	27099	Control.java	
2008-05-06	20:16:00	789	Int3.java	
				MAPS
2008-05-05	01:01:00	1678	l01.map	
2008-05-06	20:18:00	32156	l01.png	
2008-05-05	14:54:00	1678	l02.map	
2008-05-06	20:18:00	32279	l02.png	
2008-05-05	14:54:00	1678	l03.map	

2008-05-06	20:18:00	32244	l03.png
2008-05-05	14:54:00	1678	l04.map
2008-05-06	20:18:00	34523	l04.png
2008-05-05	14:54:00	1678	l05.map
2008-05-06	20:18:00	34516	l05.png
2008-05-05	01:01:00	958	m01.map
2008-05-06	20:18:00	31564	m01.png
2008-05-05	15:57:00	958	m02.map
2008-05-06	20:18:00	31516	m02.png
2008-05-05	18:50:00	958	m03.map
2008-05-06	20:18:00	33649	m03.png
2008-05-05	18:50:00	958	m04.map
2008-05-06	20:18:00	33453	m04.png
2008-05-05	18:50:00	958	m05.map
2008-05-06	20:18:00	34115	m05.png
2008-05-05	01:01:00	438	s01.map
2008-05-06	20:18:00	31973	s01.png
2008-05-05	15:18:00	438	s02.map
2008-05-06	20:18:00	32022	s02.png
2008-05-06	23:17:00	438	s03.map
2008-05-06	20:18:00	32011	s03.png
2008-05-05	15:18:00	438	s04.map
2008-05-06	20:18:00	32892	s04.png
2008-05-05	15:57:00	438	s05.map
2008-05-06	20:18:00	33495	s05.png
2008-05-05	01:01:00	118	vs01.map
2008-05-06	20:18:00	30540	vs01.png
2008-05-07	00:31:00	118	vs02.map
2008-05-07	00:31:00	28924	vs02.png
			pics
2008-05-05	00:31:00	656055	background.PNG
2008-05-05	00:30:00	2919	bogyofuresz.PNG
2008-05-05	00:30:00	2952	bogyoko.PNG
2008-05-05	00:30:00	2898	bogyomezei.PNG
2008-05-05	14:52:00	2815	fal.PNG
2008-05-05	00:30:00	2877	pl0farokkoves.PNG
2008-05-05	00:30:00	2897	pl0faroksima.PNG
2008-05-05	00:30:00	2962	pl0fejbalfuresz.PNG
2008-05-05	00:30:00	2948	pl0fejbal.PNG
2008-05-05	00:30:00	2926	pl0fejfelfuresz.PNG
2008-05-05	00:30:00	2956	pl0fejfel.PNG
2008-05-05	00:30:00	2969	pl0fejjobbfuresz.PNG
2008-05-05	00:30:00	2949	pl0fejjobb.PNG
2008-05-05	00:30:00	2927	pl0fejlefuresz.PNG
2008-05-05	00:30:00	2944	pl0fejle.PNG
2008-05-05	00:30:00	2882	pl0koves.PNG

2008-05-05	00:30:00	2908	pl0sima.PNG
2008-05-05	00:30:00	2870	pl1farokkoves.PNG
2008-05-05	00:30:00	2886	pl1faroksima.PNG
2008-05-05	00:30:00	2943	pl1fejbalfuresz.PNG
2008-05-05	00:30:00	2935	pl1fejbal.PNG
2008-05-05	00:30:00	2921	pl1fejfelfuresz.PNG
2008-05-05	00:30:00	2942	pl1fejfel.PNG
2008-05-05	00:30:00	2943	pl1fejjobbfuresz.PNG
2008-05-05	00:30:00	2936	pl1fejjobb.PNG
2008-05-05	00:30:00	2915	pl1fejlefuresz.PNG
2008-05-05	00:30:00	2941	pl1fejle.PNG
2008-05-05	00:30:00	2871	pl1koves.PNG
2008-05-05	00:30:00	2894	pl1sima.PNG
2008-05-05	00:30:00	2917	plzfarokkoves.PNG
2008-05-05	00:30:00	2935	plzfaroksima.PNG
2008-05-05	00:30:00	2990	plzfejbalfuresz.PNG
2008-05-05	00:30:00	2997	plzfejbal.PNG
2008-05-05	00:30:00	2940	plzfejfelfuresz.PNG
2008-05-05	00:30:00	2996	plzfejfel.PNG
2008-05-05	00:30:00	2994	plzfejjobbfuresz.PNG
2008-05-05	00:30:00	3015	plzfejjobb.PNG
2008-05-05	00:30:00	2993	plzfejlefuresz.PNG
2008-05-05	00:30:00	3012	plzfejle.PNG
2008-05-05	00:30:00	2934	plzkoves.PNG
2008-05-05	00:30:00	2944	plzsima.PNG
2008-05-06	23:17:00	66906	splashscreen.PNG

A. Függelék

Napló

Dátum	Résztvevő(k)	Tárgy
2008.02.15. 18.30 – 2008.02.15. 20.15	Csuzdi, Fehér, Györgyey, Major	*Megbeszélés* munkamegosztásról és a kiírt feladatról. Döntések: – A kiírt feladat specifikációjának pontosítása. (lásd. feladatleírás) – *Csuzdi* feladata a dokumentáció szerkesztése (LaTeX), valamint a projektterv megírása hétfői határidővel. – *Fehér* feladata a követelmény definíció elkészítése hétfői határidővel. – *Györgyey*^{CSK} feladata az essential use-case-ek elkészítése hétfői határidővel. – *Major* feladata a feladatleírás, valamint a szótár elkészítése hétfői határidővel.
2008.02.15. 23.00 – 2008.02.15. 24.00	Csuzdi	A naplókat összefésülő, és LaTeX formátumba átkonvertáló szkript megírásának elkezdése.
2008.02.16. 16.00 – 2008.02.16. 17.00	Csuzdi	A naplókat összefésülő, és LaTeX formátumba átkonvertáló szkript megírásának befejezése.
2008.02.16. 17.00 – 2008.02.16. 19.00	Major	Követelmények leírása.
2008.02.17. 14.00 – 2008.02.17. 15.00	Major	Szótár megírása.
2008.02.17. 15.00 – 2008.02.17. 18.00	Csuzdi	Projekt terv írása.
2008.02.17. 18.00 – 2008.02.17. 20.00	Fehér	Követelmény definíció megírása.

2008.02.17. 20.00 – 2008.02.17. 22.00	Csuzdi	Projekt terv befejezése.
2008.02.17. 23.30 – 2008.02.18. 00.00	Györgyey	Essential Use-Case diagram elkészítése.
2008.02.18. 00.00 – 2008.02.18. 00.30	Györgyey	Essential Use-Case-ek leírásainak elkészítése.
2008.02.19. 23.00 – 2008.02.20. 00.00	Csuzdi, Fehér, Györgyey, Major	*Megbeszélés* a dokumentumok véglegesítéséről. Döntés a beadásra alkalmas végleges változat elfogadásáról.
2008.02.21. 22.00 – 2008.02.22. 00.00	Csuzdi, Fehér, Györgyey, Major	*Megbeszélés*, az analízis modell kidolgozása.
2008.02.22. 15.00 – 2008.02.22. 18.00	Csuzdi, Fehér, Györgyey, Major	*Megbeszélés*, az analízis modell kidolgozása. Döntések: Fehér elkészíti a statechart-okat. Györgyey elkészíti az osztálydiagramot. Major elkészíti az objektum katalógust.
2008.02.23. 19.00 – 2008.02.23. 22.00	Györgyey	Osztálydiagram készítése.
2008.02.23. 23.00 – 2008.02.23. 23.30	Györgyey	Osztálydiagram készítése.
2008.02.24. 00.30 – 2008.02.24. 03.00	Györgyey	Osztálydiagram készítése.
2008.02.24. 22.00 – 2008.02.24. 23.00	Fehér	Statechart diagramok elkezdése.
2008.02.25. 18.00 – 2008.02.25. 19.00	Fehér	Statechart diagramok folytatása.
2008.02.26. 23.30 – 2008.02.26. 24.00	Fehér	Statechart diagramok befejezése.
2008.02.26. 23.45 – 2008.02.27. 01.00	Györgyey	Osztályleírások készítése.
2008.02.27. 01.45 – 2008.02.27. 04.30	Györgyey	Szekvencia diagramok készítése.
2008.02.27. 21.00 – 2008.02.27. 23.00	Csuzdi	Szekvencia diagram készítés (new game).
2008.02.27. 21.00 – 2008.02.28. 00.00	Major	Szekvencia diagramok készítése.
2008.02.27. 23.00 – 2008.02.28. 04.00	Csuzdi	Objektumkatalógus szerkesztése, diagramok szerkesztése.
2008.02.28. 00.00 – 2008.02.28. 02.00	Major	Objektum katalógus megírása.

2008.02.28. 02.00 – 2008.02.28. 04.00	Major	Dokumentumok véglegesítése, áttekintés.
2008.03.03. 15.00 – 2008.03.03. 16.00	Csuzdi, Fehér, Györgyey, Major	*Megbeszélés* a teendőkről és a modellről. Döntés a modell módosításáról.
2008.03.03. 17.00 – 2008.03.03. 19.30	Györgyey	Osztálydiagram módosítása.
2008.03.03. 22.30 – 2008.03.04. 00.30	Györgyey	Osztálydiagram módosítása.
2008.03.04. 20.30 – 2008.03.04. 21.30	Fehér	Osztályleírások átírása, frissítése.
2008.03.04. 22.00 – 2008.03.05. 01.00	Major	Objektumkatalógus módosítása.
2008.03.05. 00.00 – 2008.03.05. 04.30	Csuzdi	Szekvencia diagramok készítése.
2008.03.05. 01.00 – 2008.03.05. 02.00	Major	Dokumentumok szerkesztése.
2008.03.05. 02.00 – 2008.03.05. 03.30	Major	Dokumentumok véglegesítése, áttekintés.
2008.03.09. 22.00 – 2008.03.09. 23.00	Fehér	A szkeleton modell valóságos use-case-einek megírása.
2008.03.10. 15.00 – 2008.03.10. 16.00	Fehér	Az architektúra megírásának elkezdése.
2008.03.11. 16.00 – 2008.03.11. 17.00	Csuzdi, Fehér, Györgyey, Major	*Megbeszélés* munkamegosztásról és a kiírt feladatról. Döntések: – *Csuzdi* feladata a kezelői felület megtervezése. – *Fehér* feladata a valóságos use-case-ek leírása, és az architektúra leírása. – *Györgyey*^{CSK} feladata a szekvencia diagramok pontosítása. – *Major* feladata az elkészült anyagok ellenőrzése, szerkesztése.
2008.03.11. 22.30 – 2008.03.11. 24.00	Fehér	Az architektúra megírásának befejezése.
2008.03.11. 23.00 – 2008.03.12. 01.00	Csuzdi	A szkeleton kezelői felületének tervének elkészítése.
2008.03.12. 00.00 – 2008.03.12. 01.00	Fehér	A use-case diagram befejezése.
2008.03.12. 01.30 – 2008.03.12. 02.30	Györgyey	Szekvencia diagramok pontosítása.

2008.03.17. 20.00 – 2008.03.17. 20.30	Fehér	Értékelés megírása.
2008.03.17. 20.30 – 2008.03.17. 23.30	Fehér	*Wall, StoneBerry, SnakeHead, SnakeElement* osztályok függvényhívásainak és kommentek megírása.
2008.03.18. 18.00 – 2008.03.18. 21.00	Major	Szkeleton forrásfájlok kiegészítése.
2008.03.18. 21.00 – 2008.03.18. 22.00	Csuzdi	A szkeleton használati utasításának megírása, *Model loadMap* függvényének megírása, CLI osztály megírása.
2008.03.18. 21.00 – 2008.03.18. 22.00	Fehér	Dokumentáció írása.
2008.03.18. 21.00 – 2008.03.18. 22.00	Major	Dokumentumok szerkesztése.
2008.03.18. 23.00 – 2008.03.18. 23.30	Fehér	Pályák elkészítése.
2008.03.19. 00.00 – 2008.03.19. 03.00	Györgyey	*Control* osztály függvényeinek megírása.
2008.03.19. 03.00 – 2008.03.19. 04.30	Györgyey	*Model* osztály use-case függvényeinek megírása.
2008.03.24. 23.00 – 2008.03.25. 00.00	Csuzdi, Fehér, Györgyey, Major	*Megbeszélés* munkamegosztásról és a megváltozott specifikáció értelmezéséről. Döntések: – A kiírt feladat specifikációjának pontosítása. – *Csuzdi* feladata a tesztelő nyelv definiálása, a tesztelést támogató segédprogramok specifikálása. – *Fehér* feladata a részletes use-case-ek elkészítése, és a tesztelési tervek kidolgozása. – *Györgyey*^{CSK} feladata a modellben szükséges változtatások kidolgozása, dokumentálása. – *Major* feladata a prototípus interfész definíció kidolgozása.
2008.03.25. 20.00 – 2008.03.25. 20.30	Fehér	Részletes use-case-ek megírása.
2008.03.25. 20.30 – 2008.03.25. 22.00	Fehér	Tesztelési terv megírása.

2008.03.25. 21.30 – 2008.03.25. 23.00	Major	Prototípus interfész definíció elkészítése.
2008.03.25. 22.00 – 2008.03.26. 00.00	Csuzdi	A tesztelési nyelv specifikációjának kidolgozása, dokumentálása, tesztelési segédprogramok leírása.
2008.03.26. 00.00 – 2008.03.26. 01.30	Györgyey	Az analízis modell szükséges módosításainak kidolgozása.
2008.03.31. 21.00 – 2008.03.31. 24.00	Fehér	Tesztelési terv elkezdése.
2008.04.01. 21.00 – 2008.04.01. 24.00	Fehér	Tesztelési terv folytatása.
2008.04.01. 22.00 – 2008.04.01. 00.00	Csuzdi	LaTeX template írás, formázás.
2008.04.02. 00.00 – 2008.04.02. 03.00	Fehér	Tesztelési terv befejezése.
2008.04.02. 01.00 – 2008.04.02. 06.00	Csuzdi	Activity diagramok rajzolása, tesztelést segítő programok terveinek kidolgozása, szerkesztés.
2008.04.02. 01.00 – 2008.04.02. 06.30	Györgyey	Objektumok és metódusaik részletes tervének elkészítése.
2008.04.07. 14.00 – 2008.04.07. 17.00	Major	*Mydiff* program megírása.
2008.04.08. 17.00 – 2008.04.08. 19.00	Csuzdi	*Snapshotviewer* program elkészítése, tesztelése.
2008.04.08. 21.00 – 2008.04.08. 22.00	Fehér	*Berry, FieldBerry, SawBerry, StoneBerry, Wall* osztályok megírása.
2008.04.14. 18.00 – 2008.04.14. 19.00	Csuzdi	CLI osztály elkészítése.
2008.04.14. 18.45 – 2008.04.14. 19.00	Csuzdi, Fehér, Györgyey, Major	*Megbeszélés* a feladatok kiosztásáról. Döntések: – *Csuzdi* feladata a VIEW package osztályainak megírása. – *Fehér* feladata a GAMEDATA és TEST package-ek állományainak elkészítése. – *Györgyey*^{CSK} feladata a MODEL package osztályainak megírása. – *Major* feladata a CONTROL package osztályainak megírása.
2008.04.14. 19.00 – 2008.04.14. 20.00	Fehér	A teszteléshez szükséges fájlok elkészítése (*.map, .test, .ref, .gref*).

2008.04.14. 19.00 – 2008.04.14. 20.00	Györgyey	*GameField* osztály módosítása.
2008.04.14. 22.30 – 2008.04.14. 23.45	Györgyey	*GameField* osztály befejezése.
2008.04.14. 23.45 – 2008.04.15. 00.45	Györgyey	*GameObject* és *Model* osztályok módosítása, befejezése.
2008.04.15. 16.00 – 2008.04.15. 18.30	Major	*Control* osztály megírása.
2008.04.15. 19.00 – 2008.04.15. 20.00	Csuzdi	*SnakeHead* függvényeinek írása.
2008.04.15. 19.00 – 2008.04.15. 20.00	Györgyey	*Snake* osztály módosítása, befejezése.
2008.04.15. 20.00 – 2008.04.15. 20.30	Fehér	Értékelés megírása.
2008.04.15. 20.30 – 2008.04.15. 21.00	Fehér	Tesztek jegyzőkönyvének megírásának elkezdése.
2008.04.15. 22.00 – 2008.04.16. 01.00	Major	A prototípus debuggolása.
2008.04.16. 00.00 – 2008.04.16. 00.30	Györgyey	*SnakeElement* osztály módosítása, befejezése.
2008.04.16. 10.00 – 2008.04.16. 14.00	Györgyey	Tesztelés, debuggolás, javítás.
2008.04.16. 12.00 – 2008.04.16. 15.00	Csuzdi	Debuggolás.
2008.04.16. 14.30 – 2008.04.16. 16.00	Fehér	Debuggolás.
2008.04.16. 15.00 – 2008.04.16. 16.00	Csuzdi	Tesztprogramok használatának leírása.
2008.04.16. 16.00 – 2008.04.16. 18.00	Major	A prototípus tesztelése.
2008.04.16. 19.00 – 2008.04.16. 23.00	Györgyey	Tesztelés, hibajavítás.
2008.04.16. 22.00 – 2008.04.16. 23.00	Csuzdi	Szerkesztés.
2008.04.22. 19.00 – 2008.04.22. 20.00	Györgyey	Grafikus rendszer architektúrájának megtervezése.
2008.04.22. 20.00 – 2008.04.22. 21.00	Csuzdi	Objektumleírás szerkezetének kialakítása, sablon elkészítése.

2008.04.22. 23.00 – 2008.04.23. 00.00	Györgyey	Statikus struktúra diagram készítése.
2008.04.22. 23.00 – 2008.04.23. 01.30	Major	Objektumkatalógus készítése, dokumentumok ellenőrzése.
2008.04.23. 01.00 – 2008.04.23. 03.00	Csuzdi	Szekvencia diagramok készítése.
2008.04.23. 02.30 – 2008.04.23. 03.00	Györgyey	A grafikus felület működési elvének leírása.
2008.04.27. 17.00 – 2008.04.27. 20.00	Györgyey	*Control*, *GameFieldLabel*, *Graph_FieldElement*, *Graph_GameObject* osztályok írása.
2008.04.28. 18.30 – 2008.04.28. 20.00	Györgyey	*GameFieldLabel*, *Conf* osztályok módosítása.
2008.04.29. 01.00 – 2008.04.29. 03.30	Györgyey	*Control*, *Graph_FieldElement* osztályok módosítása, grafikus rajzoló osztályok megírása.
2008.04.29. 16.00 – 2008.04.29. 19.00	Györgyey	*Control*, *Conf* osztály írása, modell javítása.
2008.04.30. 18.00 – 2008.04.30. 21.00	Major	A Settings panel elkészítésének elkezdése.
2008.04.30. 21.00 – 2008.04.30. 23.00	Fehér	A grafikus elemek elkészítése.
2008.05.04. 20.00 – 2008.05.04. 22.00	Fehér	Pályák készítése.
2008.05.05. 15.00 – 2008.05.05. 16.00	Györgyey	Hibajavítás, *Model* módosítása.
2008.05.05. 22.00 – 2008.05.05. 23.00	Fehér	Pályák készítése és tesztelése.
2008.05.06. 20.00 – 2008.05.06. 03.00	Major	A Settings panel elkészítésének befejezése.
2008.05.06. 20.00 – 2008.05.06. 21.00	Fehér	Grafikus elemek javítása, kiegészítése.
2008.05.06. 22.00 – 2008.05.06. 22.30	Fehér	Értékelés megírása.
2008.05.06. 22.00 – 2008.05.07. 02.00	Csuzdi	*Conf* függvényeinek implementálása. Győztes eldöntő logika megírása.
2008.05.06. 23.00 – 2008.05.06. 23.30	Fehér	Súgó megírása.