

Csapat neve: IB028

Konzulens: Dr. Horváth Tamás

Tagok:

Boja Bence (y5yr6u)

Stefán Viktor (xlkvih)

Vass Gergely (a7xiix)

Zajác Balázs (rgx5b3)

*BB132@HSZK.BME.HU*

*SVIKTOR@MAIL.MATAV.HU*

*VGERGO@FRONTFILM.COM*

*ZB120@HSZK.BME.HU*

*2000. május 13.*

***Végleges program programozói  
kézikönyve***

## Tartalom

Tartalom.....	2
Követelmény leírása .....	5
Követelmény definíció.....	5
1 Feladat .....	5
2 Funkciók.....	5
3 Kezelés, funkciók.....	5
4 Futtatási környezet: hardware-software .....	5
5 Felhasználó .....	6
6 Pontossági követelmények .....	6
7 Információ források .....	6
Projekt terv.....	6
1 Életciklus modell .....	6
2 Csapat összetétele .....	7
3 Fejlesztési ütemezés és dokumentumok.....	7
4. Erőforrások kihasználása.....	7
5 Fejlesztési környezet és programnyelv .....	7
6 Bemutató formája .....	7
Essential use-case-ek .....	8
Játék: .....	8
Menü: .....	8
Use case diagram.....	8
Játékkal kapcsolatos objektumok katalógusa.....	8
Fehér golyó:.....	8
Fekete golyó: .....	8
Egyéb golyók:.....	9
Lyuk: .....	9
Asztal: .....	9
Falak:.....	9
Játékos:.....	9
Játék: .....	9
Szabály:.....	9
Dákó:.....	9
Játékkal kapcsolatos osztályok leírása.....	10
Részletes tervek: .....	10
Játék .....	10
Dákó.....	11
Szabály .....	11
Játékos.....	14
Golyó.....	14
Asztal .....	14
Fal .....	14
Lyuk .....	14
Struktúra diagram .....	15
Szekvencia diagramok .....	15

Inicializálás:.....	15
Játék: .....	16
Grafikus megjelenés .....	17
Játék kezelői felület .....	17
Menü megjelenítése .....	17
NEW .....	17
LOAD.....	18
SAVE .....	18
HELP.....	18
QUIT .....	18
A kezelőfelület működésének elve.....	19
Általános megfontolások .....	19
Menürendszer architektúrája.....	19
Menük kiírásával kapcsolatos objektumok .....	19
Menü-objektumok tervei:.....	19
App:.....	19
Be/Ki .....	20
Billentyűzet .....	20
Error .....	20
Dialog:.....	21
Menü szekvencia diagramjai.....	21
Load: .....	21
New:.....	21
Save:.....	21
Játék megjelenítésével kapcsolatos objektumok .....	23
Játék megjelenítésével kapcsolatos objektumok .....	23
Objektumok leírásai.....	23
JatekView: .....	23
GolyoView: .....	23
AsztalView: .....	24
FalView: .....	24
LyukView:.....	24
DakoView: .....	24
Játékkirajzolás szekvencia diagramjai .....	24
Lökés:.....	24
Szimuláció: .....	24
Architektúra, ütemezés .....	25
Architektúra.....	25
Ütemezés .....	25
Tesztelési terv.....	26
Menü tesztelése.....	26
Játék (szabályok) ellenőrzése .....	27
Szimuláció ellenőrzése.....	27
A tesztelést támogató programok tervei .....	28
MELLÉKLET: .....	29
<b>SZABÁLYOK</b> .....	29
MELLÉKLET: .....	31

<b>MEGVALÓSÍTANDÓ RENDSZER FIZIKAI MODELLJE.....</b>	<b>31</b>
1. A golyó-asztal kapcsolat .....	31
2. A golyó-golyó kapcsolata.....	32
3. A golyó-dákó kapcsolat.....	32
A rendszer szimulációja.....	32
Felhasznált irodalom:.....	33
Felhasznált jelölések:.....	34
Értékelés.....	35

## Követelmény leírása

A megvalósítandó feladat egy *biliárdjáték* elkészítése. Azon belül az úgynevezett *pool*, amit a közismert *biliárd asztalon* játszanak. Ez egy posztóval bevont *asztal*, melyen két pont van feltüntetve, a *homlokpont* és a *tőpont*. Ezen kívül 6 *lyuk* helyezkedik el az *asztal* kerete mentén. 4 a sarkokban, 2 a hosszabb falak közepén. Az asztalon kívül *golyók*, valamint egy *dákó* szükséges a játékhoz. A *dákóval* tetszőleges irányban, és (majdnem tetszőleges) erővel lökhetünk. A *golyók* az *asztal* síkjában mozognak, nem ugranak meg és nem csavarodnak. A *szabályok* (a teljesség igénye nélkül) a következők:

Négy fajta *golyót* különböztetünk meg, ezeket meghatározott módon, egy *háromszög* alakú keret segítségével kell a játék kezdetekor felállítani. Az egész játék folyamán csak a fehér *golyót* szabad meglökni a *dákóval*. Két *játékos* van, mindegyikük egészen addig lökhet, amíg vagy ront (nem sikerül "legálisan" egy *golyót* sem "elrakni", ekkor az ellenfél jön), vagy *hibát* követ el (ekkor az ellenfél kétszer lökhet).

A cél: saját *golyóinkat* a *lyukakba* gurítani, majd az utolsó leeső *golyó lyukával* középpontosan szemben levő *lyukba* lökni a fekete *golyót*. Veszít, aki a fekete *golyót* hamarabb, vagy rossz *lyukba* lövi.

A szabályleírás a mellékletben megtalálható.

## Követelmény definíció

### 1 Feladat

Feladatunk egy biliárdprogram létrehozása, mellyel ketten játszhatnak egyszerre. A biliárd játék szabályai a dokumentáció előző részében megtalálható.

### 2 Funkciók

A programban lehetőségünk van a következő, játékkal kapcsolatos funkciók használatára: új játék kezdése, állás betöltése és elmentése, szabályok kiírása és a kilépésre. Ezen szolgáltatások igénybevételén túl természetesen lehetőségünk van a játékkal játszani. Ez a fehér golyó lökését jelenti körről körre. Ehhez a dákóval való lökés paramétereit tudjuk állítani: irány, erősség.

### 3 Kezelés, funkciók

A játék szövegei mind angol nyelvűek lesznek. A program használatához a billentyűzetet használjuk bemenetnek.

### 4 Futtatási környezet: hardware-software

Nem feltételezhető semmilyen speciális software vagy hardware elem a számítógépben. Szabványos VGA felület megléte feltételezhető, 16 színt és 640\*480-as felbontásnál nem

nagyobbat fogunk használni. A programnak egy Pentium processzoros PC-n megfelelő sebességgel kell futnia. A software-t Borland C++ fordítóval fordítjuk feltételezhető, hogy dos/windows környezetben fog futni.

## 5 Felhasználó

A felhasználóról feltételezhető, hogy nem rendelkezik számítástechnikai ismeretekkel, így minden funkciót, szabályt ismertetni kell, illetve megfelelő védelmet kell biztosítani az illegális bemenetek esetére. Azt is feltételezzük, hogy a játékot ketten játsszák.

## 6 Pontossági követelmények

A golyók mozgásának megközelítőleg ki kell elégíteniük a fizika törvényeit. A pattanások irányát a szilárd-test mechanikai törvények alapján számítjuk. A golyók egymással tökéletesen rugalmasan ütköznek, de a posztón való haladás közben lassan lassulnak, így véges időn belül biztosan megállnak. A lökés-erősség maximális értékének beállításánál figyelembe kell venni egy átlagember valódi lökéserejét. Eltekintünk továbbá a nem síkbeli mozgásoktól valamint a golyó forgásából származó erőktől. A mozgások számításánál az elfogadható hibaszázalékot nem határozzuk meg előre, de semmiképpen nem szabad érzékelhetőnek lennie a játékos számára az esetleges számítási pontatlanságoknak.

## 7 Információ források

*Biliárd Játékok Nemzetközi Szabályai, Snooker BT Budapest 1992*

*László, Z. : Programozás technológiája, Műegyetemi Kiadó, Bp., 1994*

*Kondorosi, László, Szirmay-Kalos: Objektum orientált szoftver fejlesztés, ComputerBooks, Bp., 1997.*

Ezekon kívül használhatóak a C++ dokumentációk, a tanulmányaink során összegyűjtött egyéb UML, OMT és C++ jegyzetek, valamint rendelkezésre áll rendszeres konzultációs lehetőség. A projekthez szükséges egyéb információk a <http://eniac.iit.bme.hu/~szglab4/> címen találhatóak meg.

## Projekt terv

### 1 Életciklus modell

A program fejlesztése során az alábbi fő fázisokat különböztetjük meg:

#### *szkeleton*

A program váza, a függvényeknek, eljárásoknak még nincs készen a törzse, de ellenőrizhető a modell működőképessége.

#### *prototípus*

Már működő program, grafikus felület nélkül. Ellenőrizhető a software helyes működése.

### *végleges software*

Végleges verzió, minden funkció és felület tesztelésre kerül.

## 2 Csapat összetétele

A csapatot az alábbi tagok alkotják (Azon munkafázisok kerülnek feltüntetésre, melyekben a tagok aránylag nagyobb részt kívánnak vállalni.)

*Bolya Bence*: Fizikai modellezés és a grafikus megjelenítés megtervezése, kódolás.

*Stefán Viktor*: Kódolás, tesztelés.

*Vass Gergely*: Dokumentációk készítése, tervezés.

*Zajác Balázs*: Kódolás, programtervezés.

A fejlesztés során cél, hogy minden résztvevő ugyan annyi energiát fektessen a projektbe, és elkerülhető legyen az egyenlőtlen díjazás. A feladatok kiosztását hetente közösen végezzük.

## 3 Fejlesztési ütemezés és dokumentumok

A pontos határidők és a dokumentumok listája a <http://eniac.iit.bme.hu/~szglab4/projdef.html> állományban találhatóak meg. Ezen állomány alapján készítendőek a különböző software és dokumentációs termékek.

## 4. Erőforrások kihasználása

A fejlesztés kezdeti szakaszában Vass kolléga végzi el a munka nagyobbik részét, ebben a szakaszban a tervezési dokumentációk elkészítése a fő feladat. A későbbiekben az implementációra kerül a hangsúly, amiben Boja, Zajác és Stefán kollégák vállalják a munka nagy részét. A programozás koordinátora Boja kolléga.

A munkára fordítható energia és idő sajnos limitált, de nem tükrözi a tárgy kredit-pontos súlyozását a többi tantárgyhoz képest (valószínű annál több időt kell a projektbe fektetni).

## 5 Fejlesztési környezet és programnyelv

A programot C++ programnyelvben fejlesztjük, a fordítást a Borland C++ fordítójával fogjuk elvégezni. A fejlesztés az UML módszertan szerint folyik.

## 6 Bemutatás formája

A dokumentációkat az ütemezés szerint folyamatosan prezentáljuk, a kész programot ill. a működő változatokat szintén az elkészítés hetében mutatjuk be. A programot a BME Hallgatói Számítógépközpontjának gépein teszteljük és prezentáljuk.

## Essential use-case-ek

A programmal a felhasználó alapvetően két módon, céllal kommunikál:

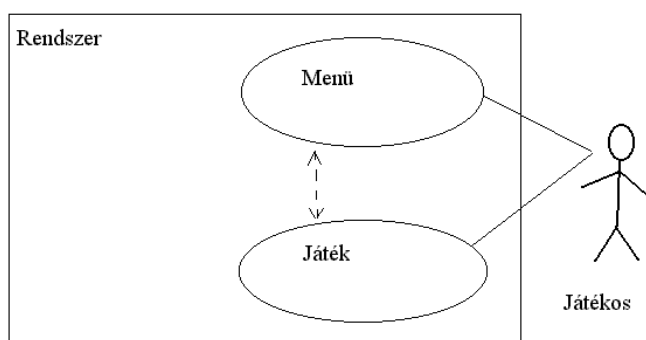
### **Játék:**

A játékos játszik, vagyis a fehér golyót löki a dáköval. Lehetősége van a játékosnak a dákö irányának az állítására valamint a lökés erősségének állítására. A lökés után a golyók a fizikai modell alapján mozognak a képernyőn (ütköznek, lassulnak, lyukba esnek stb.), majd megállnak. Miután megálltak a másik játékos jön illetve speciális esetben ua. a játékos. Természetesen a játék nyeresége/elvesztése esetén nincs újabb lökés.

### **Menü:**

A menü pontjainak használata. Lehetőségünk van új játék kezdésére, mentésre és betöltésre, szabályok lekérdezésére valamint a kilépésre.

## Use case diagram



## Játékkal kapcsolatos objektumok katalógusa

A modellezendő rendszerünkben a játékkal kapcsolatos alapvető objektumok, szereplők - melyek szükségesek a szimulációhoz - az alábbiak.

### **Fehér golyó:**

Egy darab van belőle, a lökéseknél ezt a golyót lökjük. Ütközni képes más golyókkal illetve a fallal, így jellemezhető a sebességével valamint az asztalon lévő pozíciójával. A lyukba is lökhető, ekkor a szabályok szerint kell eljárni.

### **Fekete golyó:**

Egy darab van belőle, az utolsó lökésnél kell lyukba kerülnie. Ellenkező esetben a játéknak vége. Ütközni képes más golyókkal illetve a fallal, így jellemezhető a sebességével valamint az asztalon lévő pozíciójával.



### Egyéb golyók:

14 darab van belőlük: 7 teli és 7 csíkos. Ütközni képesek más golyókkal illetve a fallal, így jellemezhetőek a sebességükkel valamint az asztalon lévő pozíciójukkal. Mindenkinnek a saját színű golyóját kell a tasakba löknie. A színválasztás valamint a lökések szabályai a szabályban megtalálhatóak.

### Lyuk:

8 darab van belőle, a lökéseknél ide lökjük a golyókat. Jellemezhető az asztalon lévő pozíciójával, hiszen nagysága adott lesz. Akármelyik golyó amelyik túl közel kerül a lyukhoz beleesik.

### Asztal:

Egy darab van belőle, ez a játék színtere. Négy fala van, ezek ütközni képesek a golyókkal.

### Falak:

Az asztalnak négy fala van, ezekkel ütköznek a golyók.

### Játékos:

Két darab van belőle, a felhasználókat reprezentálják. A játékosok nevükkel azonosíthatóak. Minden körben más-más játékos van soron, és a jellemző színű golyókat játszhatja meg. A játékot vagy megnyeri vagy elveszti.

### Játék:

A játékvezetővel azonosítható. Felelős a játék vezetéséért.

### Szabály:

Ez egy „elvont” objektum, a szabályokat ismeri pontosan. A *játék*, vagy más objektumok ettől „kérdeszhetik” a játék állását.

### Dákó:

Ezzel löki a játékos a golyót.

## Játékkal kapcsolatos osztályok leírása

Az objektumok közös tulajdonságai alapján jöttek létre az osztályok. Az attribútumok az objektumok által tárolt változók, míg a metódusok azok az akciók, amiket az adott objektum változtatása ill. lekérdezése esetében használunk.

Osztály	Attribútum	Metódus
Golyó: az összes golyó közös osztálya	sebesség, sorszám, pozíció, súrlódási együttható Típus: fehér, fekete, szín	asztalra helyezés, meglökés, egységnyi mozgás, ütközés a többi golyóval, mozgás lekérdezése, átfedés lekérdezése
Lyuk: a lyukak osztálya	középpont, sugár, sorszám	beleesik-e a golyó
Asztal	Asztal mérete, ismeri a falakat	ütközik-e falnak a golyó, legális helyen van-e a golyó
Játék: „játékvezető”	fent lévő golyók, lyukak, ki jön, ismeri: dákó, játékosok, szabály, asztal	játék indítása, tőpontra helyez golyót, megmondja melyik a szemközti lyuk
Fal: a falak osztálya	Fal pozíciója	ütközik-e a golyó
Dákó	Erősség és szög, valamint tudja, hogy melyik a fehér golyó	lök
Játékos: a játékosok osztálya	neve, jöhet-e, mely golyókkal van, hova kell löknie a feketét	névadás/lekérdezés, állapot beállítása, golyócsoport beállítás/lekérdezés, utolsó lyuk beállítás/lekérdezés
Szabály	ki lökött, hányszor, csíkosból/teliből mennyi van fent, lement-e a fehér, játékosok, szabály állapota	kapott esemény feldolgozása

### Részletes tervek:

#### Játék

Az összes többi objektumot (kivéve a *Be/Ki*) ez irányítja. Elérhetővé (címezhetővé) kell tenni számára a többi objektumot.

Metódusai:

*Inicializálás()*: Az előre beállított alapértékek alapján felépíti az objektumokat.

*JátékFuttatás()*: Szimulációs lépések elvégzése, miközben az eseményeket elküldi a szabály objektumnak. Természetesen ott, ahol a szabályokon múlhat a játéklefolyása ott le is kérheti a játékszabály állapotát. A lépések elvégzésének sorrendje az alábbi:

- Lökés elvégzése (megfelelő üzenetek elküldése a megfelelő objektumnak)

- Szimulációs lépés (do\_a\_step) elvégzése. Minden golyóra egyszer le kell futtatni. Ez még nem mozgatja a golyókat, csupán a golyó beállítja maguknak, hogy hová "kellene" gurulniuk.
- Minden golyót meg kell vizsgálni, hogy nem esik-e lyukba.
- Minden golyót meg kell vizsgálni, hogy nem ütközik-e fallal.
- A játékban lévő golyókat meg kell vizsgálni, hogy ütköznek-e egymással. Minden golyópárt meg kell vizsgálni, de csak egyszer! Fizikai modellünkben mindegy a sorrend, ugyanis (jogosan) feltételezhetjük, hogy nem ütközik két golyó egyszerre (ennek 0 a valószínűsége...) így a program által felállított sorrendet nyugodtan tekinthetjük egyfajta ütközési sorrendnek. A golyók ütközésének módusa a golyó objektumban kerül megvalósításra.
- Golyók léptetése az új pozíciójukba. Minden fent lévő golyóra egyszer.

Minden vizsgálat után ha szükség van rá (szabály szerint lényeges esemény történt), a *Szabály* objektumnak egy megfelelő üzenetet kell küldeni.

- A szabályoknak megfelelő változtatásokat el kell végezni.

*MelyikSzemközti(Lyuk):* A lyukkal szemközti lyukat adja eredményül.

*TőpontraHelyez():* A tőpontra helyezi a fehér golyót. Ha a tőpont foglalt a rövidebb fallal párhuzamos egyenes mentén a lehető legközelebb kell helyezni a tőponthoz.

## Dákó

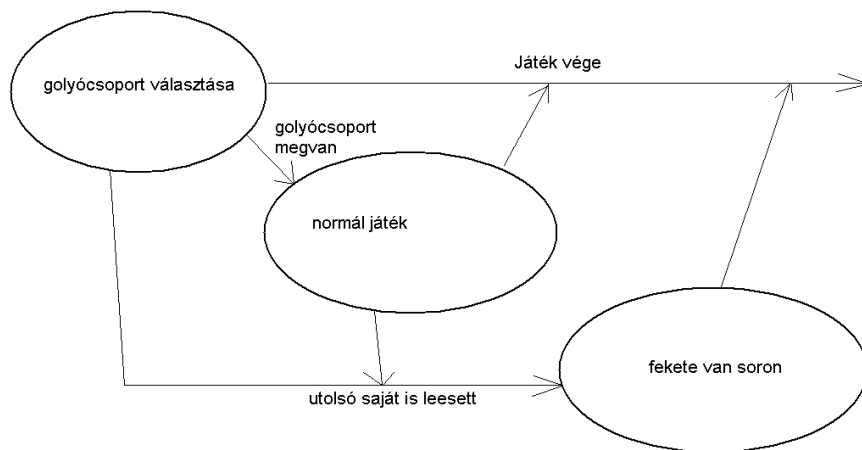
Csupán egy irány attribútummal rendelkezik.

*Lökés():* Meglöki a fehér golyót. Be kell kérni a *Billentyűzetről* a mozgás irányát és a lökés erősségét. Ennek megfelelően kell a fehér golyó paramétereit átállítani. A kirajzolást a paraméterek állítása közben is biztosítani kell majdan.

## Szabály

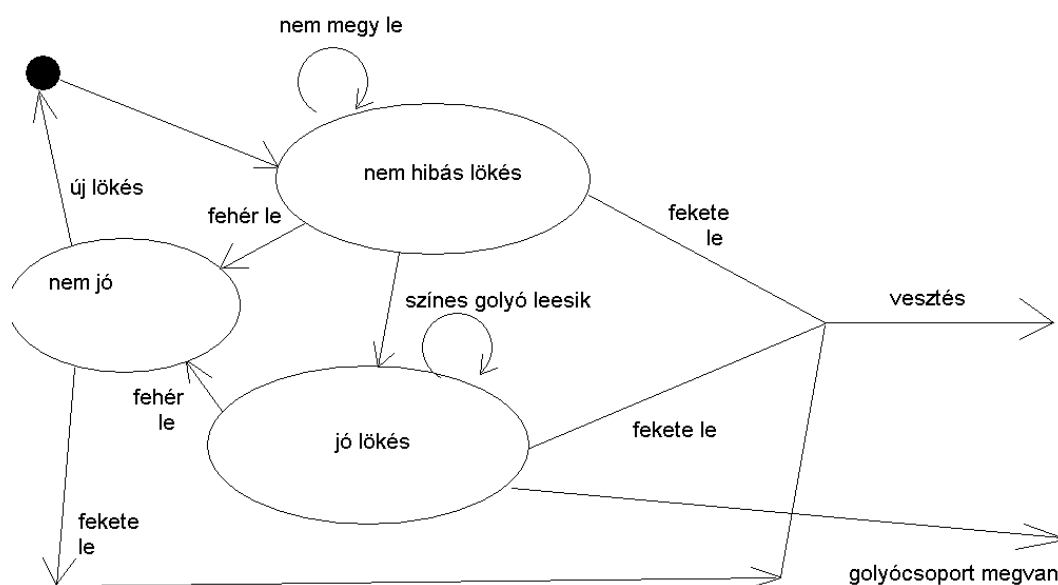
Ez az objektum egy állapotgép szerepét tölti be, így kíséri figyelemmel a játék állását. Az alábbi állapotgépet kell megvalósítani:

*Első szintű felbontás:*

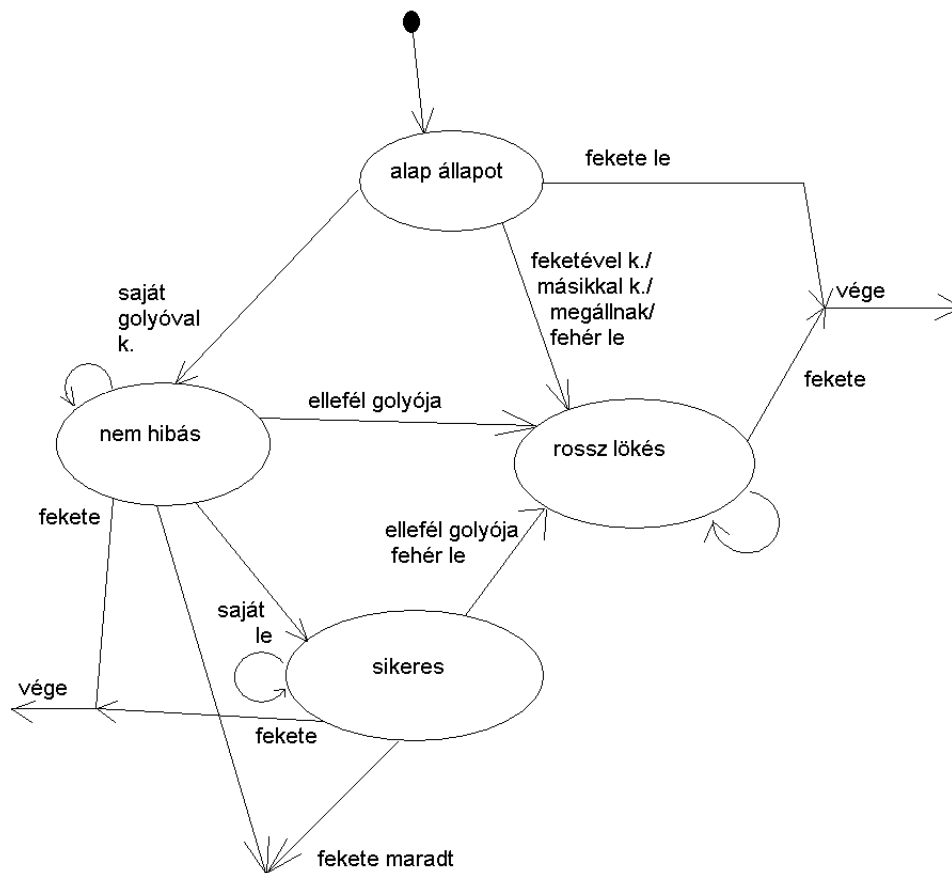


A három fő állapotot a szabályok alapján az alábbi módon valósítjuk meg:

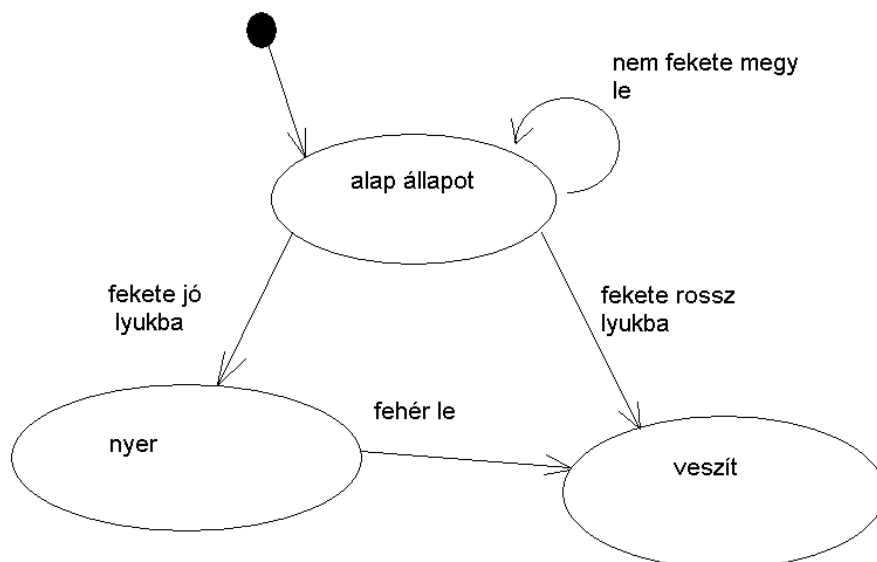
1.



2.



3.



## Játékos

Attribútumai a neve valamint az, hogy melyik golyócsoporthal van (tele/csíkos/még semmi). Azt is eltároljuk, hogy a feketét hova kell lelöknie. Metódusok:

*NévLekérdezés()*, *NévBeállítás()*, *GolyócsoporthLekérdezés()*, *GolyócsoporthBeállítás()*, *UtolsóLyukBeállítás()*, *UtolsóLyukLekérdezés()*

## Golyó

Saját attribútumai a következők: sebesség vektor, hely vektor, új pozíció vektora, típusa (fekete, fehér, tele, csíkos). A megvalósítandó metódusok az alábbiak:

*SetPoz(x,y)*, *GetPoz()*, *SetNewPoz(x,y)*, *GetNewPoz()*, *SetSpeed(x,y)*, *GetSpeed()*

Ezek a függvények nem igényelnek magyarázatot. Az alábbi függvények valamivel bonyolultabbak:

*Do\_a\_step()*: Beállítja az új pozíció paramétert a sebesség vektor és az előre definiált súrlódási együttható figyelembevételével. Lásd melléklet.

*Koccan(golyó)*: Visszatérési értéke igaz vagy hamis. A golyók koccanását az új középpontok alapján számítja ki. Ha koccantak a fizikai modell alapján módosítjuk a sebesség vektort és visszaállítjuk az új pozíciót. Lásd melléklet.

*Mozgat()*: Az új pozíció által meghatározott helyre mozgatja a golyót.

*Érintkezik(golyó)*: A középpontok alapján megmondja, hogy érintkezik-e ("átfedésben" van-e) a golyó az adott másikkal.

## Asztal

A bal alsó ill. a jobb felső sarkok koordinátáit tárolja. Ezeken kívül tároljuk a tőpont és az alappont helyzetét. (Ezek a fehér golyó ill. a kezdőállás felállításához szükségesek).

Metódusok:

*FennLehet(golyó)*: kiszámolja, hogy létező koordinátán van-e a golyó.

*Ütközik(golyó)*: Visszatérési értéke igaz vagy hamis. Az ütközést a golyó középpontja és az egyes falak távolságából kell megállapítani. Ha ütközés esete forog fenn, a golyó sebesség vektorát meg kell változtatni, és meg kell adni, hogy mi lenne így a golyó új helye. Lásd melléklet.

## Fal

Az asztal 4 falát valósítja meg, sarkaival adjuk meg. Metódusa:

*ütközik(golyó)*: egy golyóval való ütköztetést végzi el.

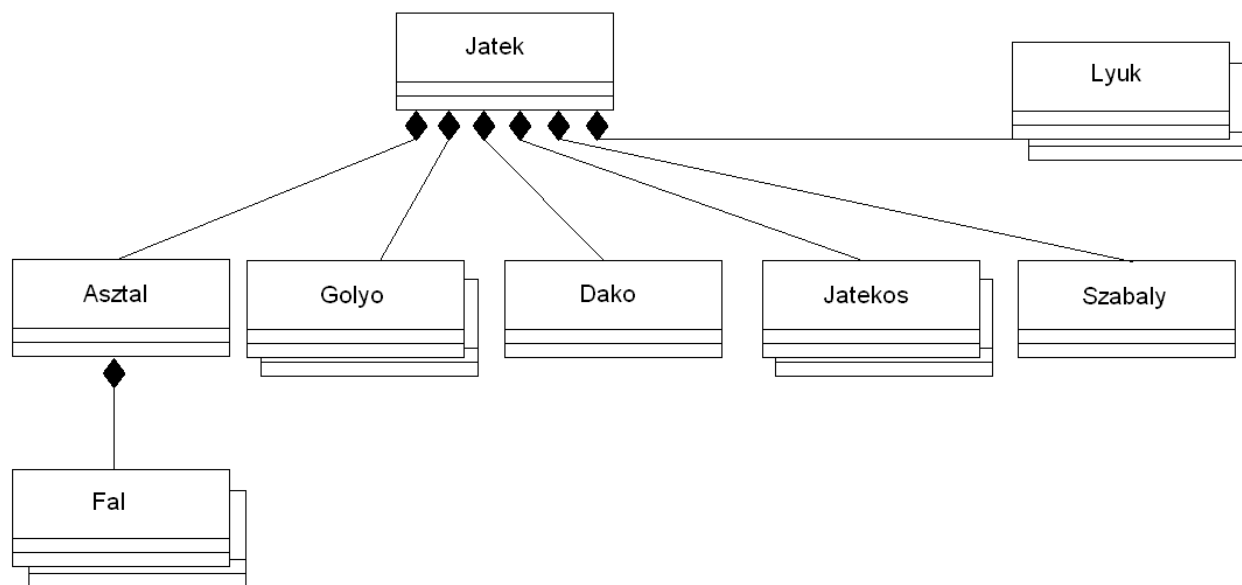
## Lyuk

A lyuk középpontját tárolja, a sugarát előre meghatározzuk. Metódusai:

*SetPoz(x,y), GetPoz()*

*Beleesik(golyó):* Visszatérési értéke igaz, vagy hamis. A beleesés tényét a középpontok távolsága alapján határozza meg.

### Struktúra diagram

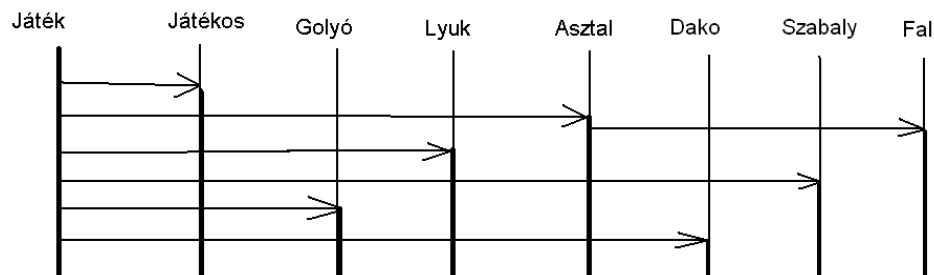


A fenti ábra szemlélteti az objektumok struktúráját. Mint az látható a Játék nevezetű objektum irányítja az egész játékot a szabályok ismeretében.

### Szekvencia diagramok

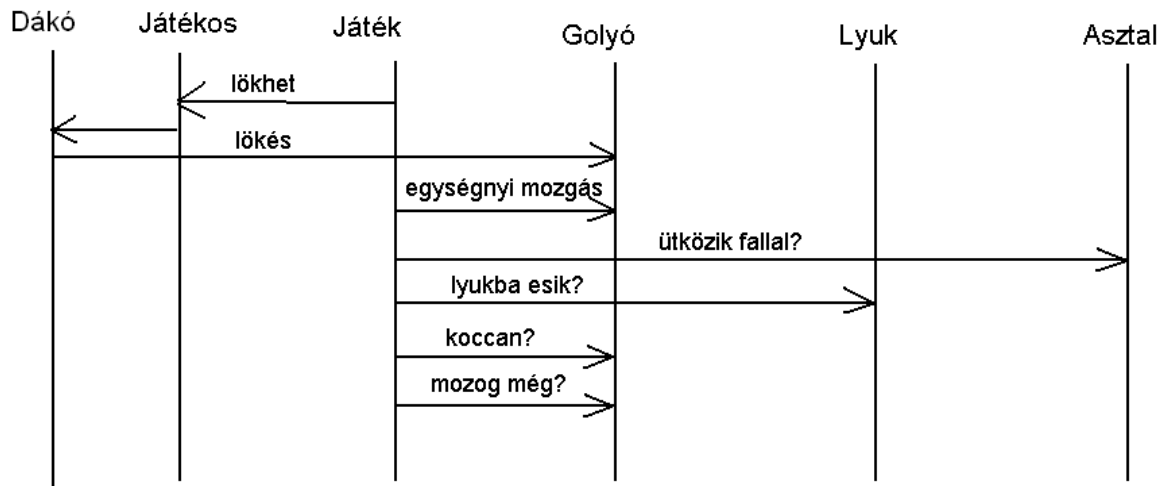
Inicializálás:

Az objektumok létrehozása a játék objektummal kezdődik. Ez az objektum inicializálja a játék résztvevőit:



### Játék:

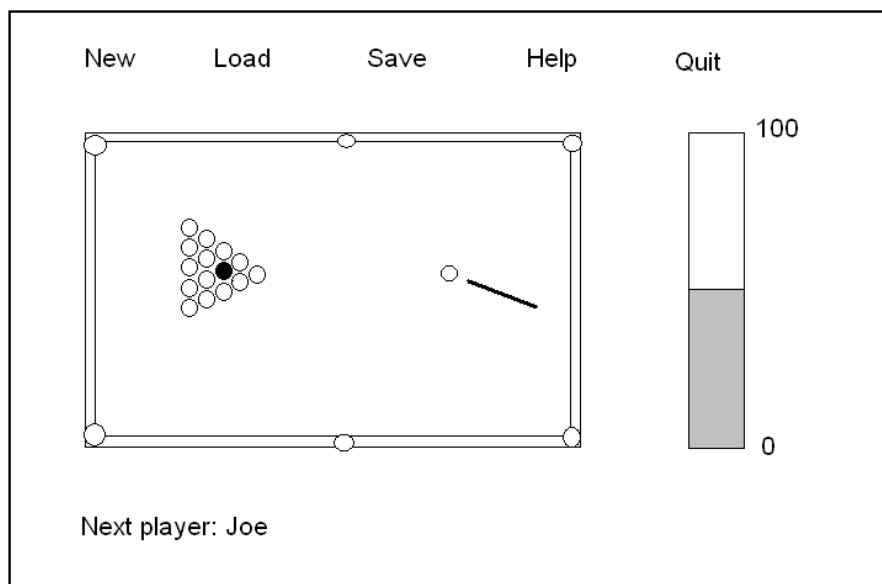
A második szekvencia diagramm egy lökés folyamatát ábrázolja. A játék objektum engedélyezi a lökést, majd a játékos meglöki az egyik golyót. Ezután a játék objektum elvégzett egy szimulációs lépést a golyókkal, majd az egymásrahatásokat lekérdezi és elvégzetteti a szükséges változtatásokat. Ha már nem mozognak a golyók újabb lökést kezdeményezhet, esetleg a játék a végéhez érhet.





## Grafikus megjelenés

### Játék kezelői felület



**Színek:** a játék megjelenése színes lesz, a pontos színértékek a fejlesztés folyamán lesznek meghatározva.

A golyók és lyukak mérete – mivel nem állnak rendelkezésre pontos adatok – szintén a későbbiekben lesznek meghatározva. Nulladik megközelítésben a falak arányai: 35/17. A lyukak mérete megközelítőleg kétszerese a golyók méretének.

Az erősség-indikátor folyamatosan követi a felhasználó gombnyomását.

A menüpontok leírása a későbbiekben következik.

Az asztal alatt helyezkedik el a státusz-sor, amiben a következő játékos neve jelenik meg.

### Menü megjelenítése

Ha lökés helyett egy menüpontot választunk a megfelelő párbeszédablak jelenik meg (kivéve kilépés). Minden ablakból vissza tudunk lépni ESC-lenyomásával, módosítások elvégzése nélkül. Az inputok betűk, számok és szóközök lehetnek, ékezeteket nem fogadunk el.

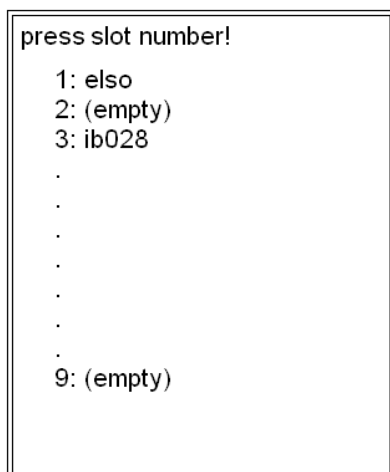
### NEW

Két játékos nevét kell bekérdezni. A párbeszédablak két lépésben kérdezi le a játékosok neveit.

The image shows a simple dialog box with a double-line border. Inside the box, the text 'Player X: .....' is displayed, indicating a prompt for the user to enter a name.

## LOAD

A játékosnak kilenc “slot” közül kell választania egy numerikus billentyű lenyomásával. A slot-ok nevei fel vannak tüntetve, az üres helyek “empty” nevet kapnak.



```
press slot number!
1: elso
2: (empty)
3: ib028
.
.
.
.
.
.
9: (empty)
```

## SAVE

Hasonló a LOAD menüjéhez, kilenc slot közül kell választani egy numerikus billentyű lenyomásával. Miután kiválasztottuk a helyet egy új ablakban kérdezzük le a slot nevét. A párbeszédablakok hasonlóak a fentiekhez.

## HELP

Egy magyarázó szöveget fog kiírni, amely a játék általános leírását tartalmazza. Természetesen angolul.

## QUIT

Nincs ablak, kilép a programból minden mentés nélkül.

## A kezelőfelület működésének elve

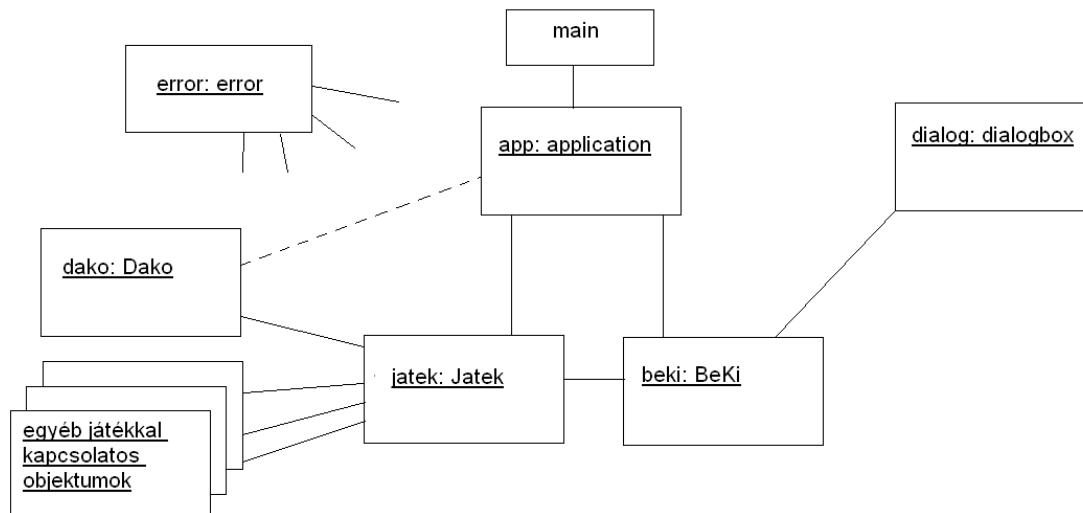
### *Általános megfontolások*

A grafikus felület 640\*350-es felbontásban kerül megvalósításra, a EGAVGA.BGI felhasználásával. A megjelenítéshez 16 színt használunk. Két grafikus lapot frissítünk a villogás elkerülése végett.

## Menürendszer architektúrája

Menük kiírásával kapcsolatos objektumok

Objektum diagram:



Az APP objektum kezeli a játék eseményeit - küldi a megfelelő üzeneteket a dákónak és a be/ki-nek – és tartalmazza a játék és a be/ki objektumokat. A dákó azért van kitüntetett helyzetben (az App-el kommunikál), mert a többi játékkal kapcsolatos objektummal ellentétben a dákót nem a szimuláció alatt, hanem a lökés/menühasználat alatt használjuk. Az error objektumot használja az összes objektum a hibaüzenetek kiírására. A menühívások hatására a be/ki objektum hívja meg a játék és a dialog objektum megfelelő függvényeit. Lásd szekvencia diagram.

### Menü-objektumok tervei:

App:

Ez az objektum kezeli a fellépő eseményeket, vagyis a beérkezett inputoknak megfelelő üzeneteket elküldi azoknak az objektumoknak, amelyek „előfizetnek” erre. Vagyis például egy menühívásnál a Be/Ki objektumnak elküldi a lenyomott billentyűt. Tartalmazza a BeKi, a Bill és a Jatek objektumokat, illetve az „előfizetők” listáját.

*eseményreVár(esemény):* Egy objektum jelzi, hogy egy bizonyos eseményre vár, amit csak neki kell megkapnia.

*előfizet(előfizető):* Előfizetői listába kerül az előfizető, így megkapja majd az eseményeket.

*előfizetMeggzűntet(előfizető):* Előfizetői listából kitörölés.

*futtat():* Elindítja az események kezelését.

## Be/Ki

A beérkezett input alapján lekezeli a New, Load, Save, Help menüpontokat. Kirajzoltatja a dialógus ablakokat, és elvégezteti a műveleteket. Tartalmazza a játékállásokat tároló 10 külső állomány címét.

A külső állomány struktúrájának felépítése:

- fejkomment (“Slot” neve)
- 1. játékos neve
- 2. játékos neve
- ki melyik golyóval van (0=nincs eldöntve, 1 v. 2: ki van a csíkkal)
- ki jön (1 v. 2)
- hányszor jön (1 v. 2)
- 1. játékosnak hova kell a fekete (0 ha nem tudjuk még)
- 2. játékosnak hova kell a fekete (0 ha nem tudjuk még)
- szabály állapota
- golyók típusa és helyzete

*Load()*: Beolvassa a billentyűzetről az inputot és annak megfelelően megpróbálja megnyitni a játékállást. Érvénytelen input vagy file esetén le kell kezelni a hibát (újbóli bekérés vagy visszatérés)

*Save()*: Beolvassa, hogy hova kellene menteni és elmenti a megfelelő paramétereket.

## Billentyűzet

A billentyűzetkezelő függvények megvalósítása:

*LeVanNyomva()*: Nullát ad vissza, ha nincs lenyomva semmi, különben a bill. kódját (ASCII / SCAN).

## Error

Itt tároljuk el a hibaüzeneteket:

1. “Nincs meg a file”
  2. “Hibás file”
  3. “Hibás bemenet”
  4. “File írás hiba”
- stb.

*Kiír(hibakód)*: A megfelelő hibakóddal hivatkozhatunk a hibára, és a függvény megjeleníti azt a képernyőn.

## Dialog:

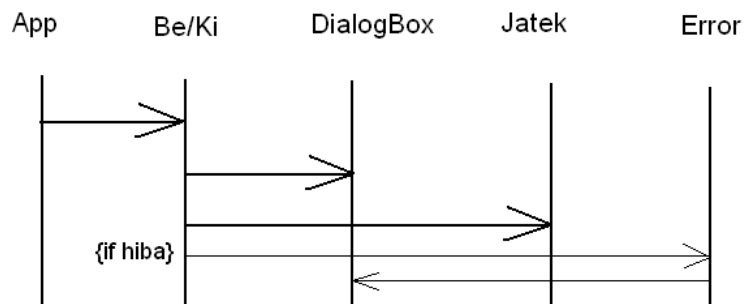
Általános ablakkirajzoló és bekérő objektumok, alapesetben a Be/Ki objektum hozza létre őket. Az App-tól kéri be a beérkező eseményeket.

Metódusa:

*Dialógus*(pozíció, header név, kiírandó szöveg, elfogadó függvény, buffer méret)

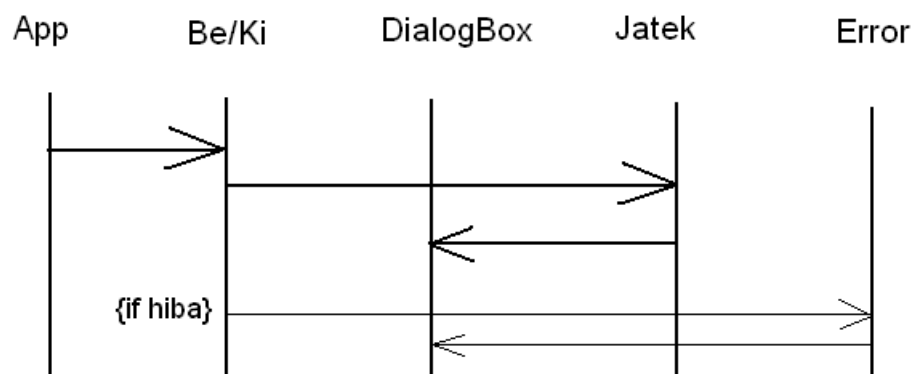
## Menü szekvencia diagramjai

Load:



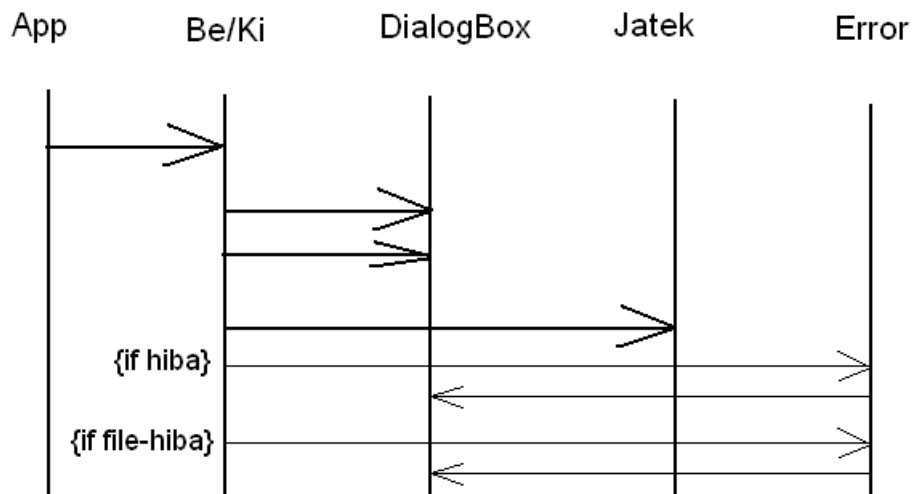
Az application a Be/Ki-vel kezelteti le az L-betű lenyomásával létrejött eseményt. A Be/Ki a DialogBox-szal bekéri a slot-nevet, majd a játék objektum megfelelő függvényével létrehozza a betöltött konfigurációt.

New:



A szekvencia diagram szinte megegyező, de a játék kéri be az új játék paramétereit.

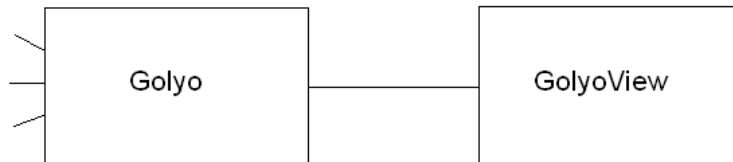
Save:



A dupla lekérdezés a slot sorszám és a slot név miatt kell. A két hibakezelés azért kell, mert egy általános elmentés után nevezzük át az ideiglenes állományt az illető slot-névre.

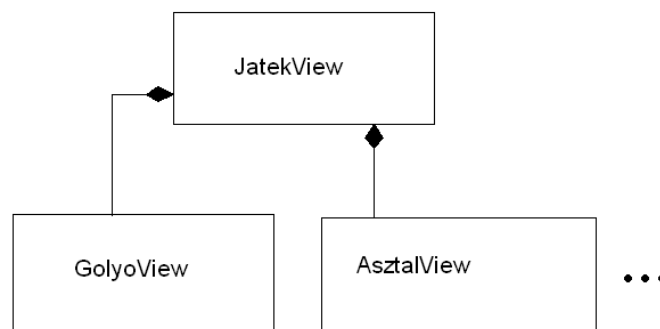
## Játék megjelenítésével kapcsolatos objektumok

Játék megjelenítésével kapcsolatos objektumok



Az összes játékkal kapcsolatos objektumhoz (amik résztvesznek a szimulációban) tartozni fog egy kirajzoló objektum. Ezek végzik a kirajzolást.

Osztályok:



Mint az látható a JatekView osztály tartalmazza az összes kirajzoló osztályt, kivéve a DakoView objektumot. A dákó kezelése némileg különbözik azoktól az elemektől, amik minden szimulációs lépésben kiíródnak. A DakoView meghívásáról maga a dákó objektum gondoskodik

### Objektumok leírásai

JatekView:

Meghívásakor kirajzolja az egész játékeret a különböző \*View objektumok megfelelő függvényeinek meghívásával. A Ez az objektum tudja, hogy miket kell meghívni.

GolyoView:

A kirajzolás meghívásakor kirajzolja a golyót.

### AsztalView:

A kirajzolás meghívásakor kirajzolja az asztalt, a tő- és homlokpontot. Meghívja a fal kirajzolását.

### FalView:

A kirajzolás meghívásakor kirajzolja az illető falat.

### LyukView:

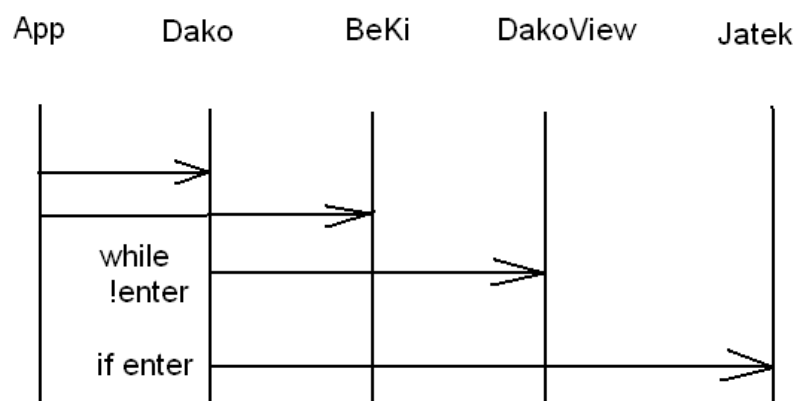
A kirajzolás meghívásakor kirajzolja a lyukat.

### DakoView:

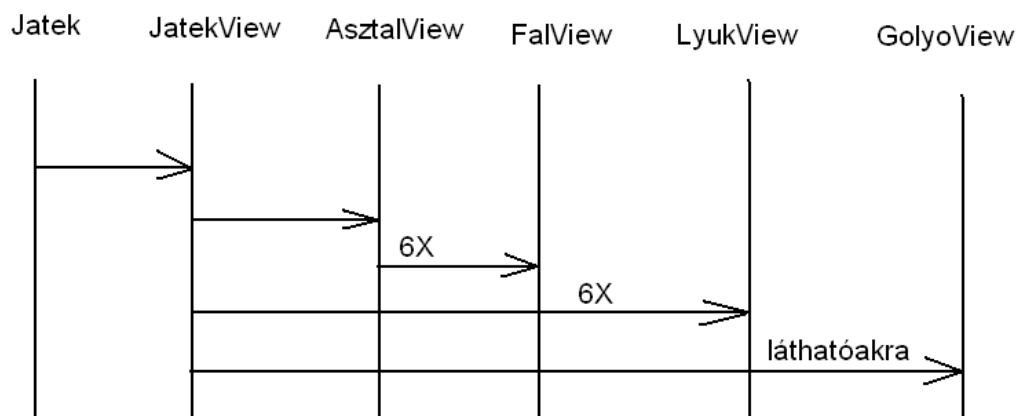
A kirajzolás meghívásakor kirajzolja a dákót az aktuális pozícióra, valamint az erősség indikátort. A kirajzoló metódust maga a Dako objektum hívja meg.

## ***Játékkirajzolás szekvencia diagramjai***

### Lökés:



### Szimuláció:



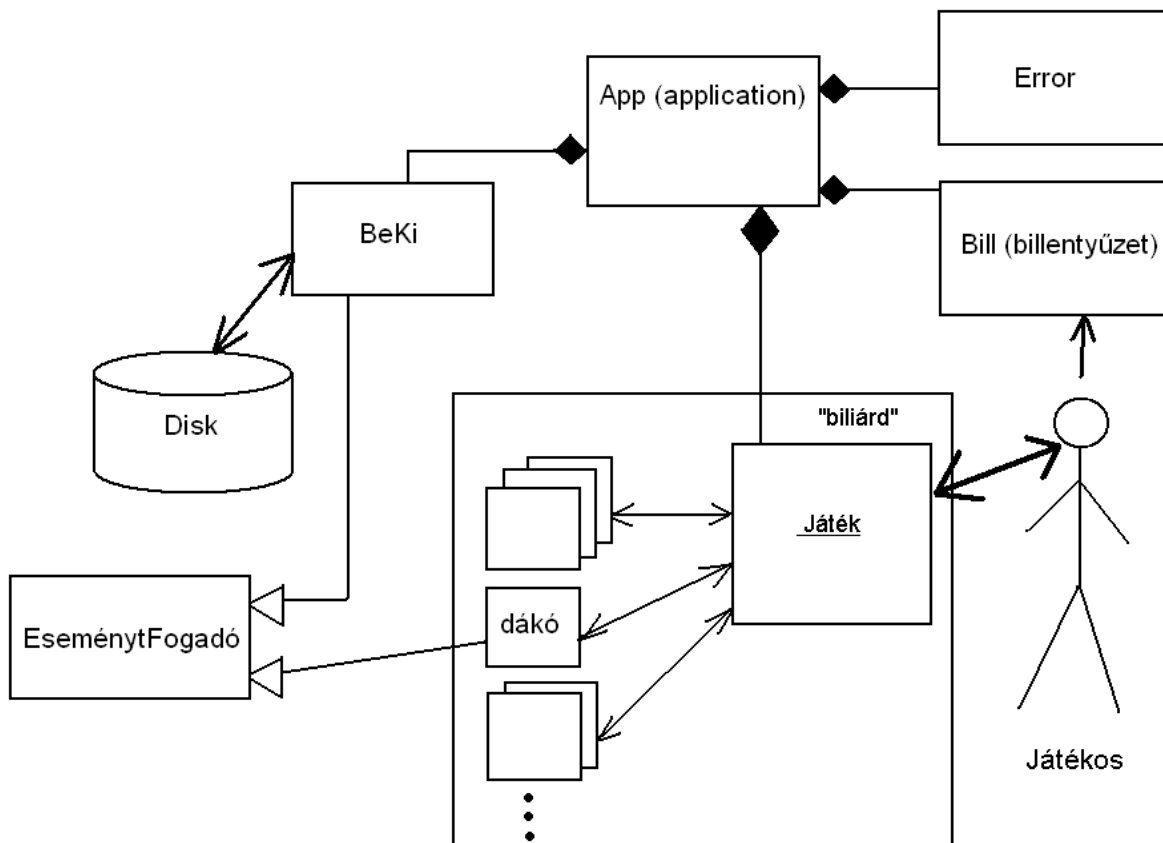
Minden ütemezési lépésben az alábbi sorrendben hívódnak meg a kirajzoló függvények



## Architektúra, ütemezés

### Architektúra

A software felépítése az alábbi ábrán látható:



A „biliárd” nevű doboz a programnak az a része, amivel a dokumentáció első felében foglalkoztunk. Erre épülnek rá az App, Bill, BeKi, és Error objektumok. A kirajzolást végző osztályok nem lettek feltüntetve.

Az „eseményt elfogadó” osztályból örököltettük a dákó és a beki osztályokat, ugyanis ezek az osztályok kaphatnak az app-objektumtól eseményeket.

### Ütemezés

A szimulációhoz időben konstans ütemezést használunk, de ennek pontos formája csak a belövéskor lesz véglegesítve. Az ütemezést úgy kell belőni, hogy a HSZK-ban használatos Pentium, illetve annál gyorsabb gépeken élvezhetően fusson. Ennek megvalósítása az alábbi

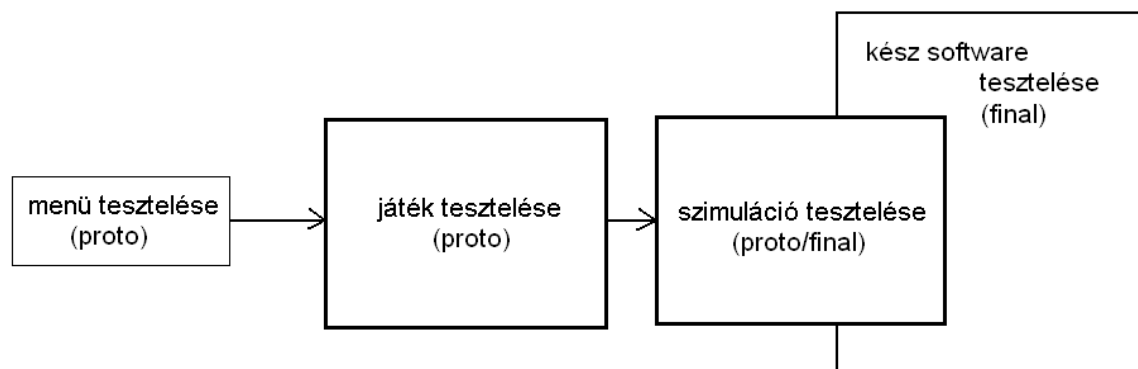
módon történhet: a Jatek objektum figyelemmel kíséri az előző kirajzolás óta eltelt időt, és egy küszöbhatár elérésére esetén rajzolja ki újra az asztal állását.

A menühasználat/lökés fázisban az ütemezés a felhasználói billentyűleütéseken alapszik: egy gomb lenyomása egy eseményt generál, melyet az App objektum küld el az előfizetőknek. A feldolgozás után vagy újabb eseményre vár a program, vagy a szimuláció indul meg.

## Tesztelési terv

A tesztelés három részre tagolódik: a menü tesztelésére, a játék-objektum és az összes játékkal kapcsolatos objektum ellenőrzésére, valamint a szimuláció ellenőrzésére. A menürendszer tesztelése nehézségében és komplexitásában eltörpül az utóbb említett két feladat mellett. Mindegyik teszt-fázis elvégzése után bővíthető a program az újabb implementált részekkel.

A tesztelés időrendi sorrendben:



Az első kettő tesztfázis még nem igényel grafikus felületet. Ezeket a teszteket a prototípussal végezzük el, ahol a képernyőn szöveges formában olvashatjuk a tesztek eredményeit.

Természetesen log-file-t is készítünk, hogy rekonstruálhatók legyenek a teszt-esetek ill. az utólagos analízishez.

A szimuláció tesztelését nem tudjuk (vagy csak nagyon nehezen) elvégezni a grafikus felület hiányában, ezért csak a fatális hibákat próbáljuk detektálni a proto fázisban. A kész software tesztelése a fizikai szimulációt, a grafikus ablakokat, és a korrekt működést teszteli.

## Menü tesztelése

A menü tesztelése igen egyszerű, a prototípus fázisban kezdjük el a teszteket:

A menüt, interfészt az alapján teszteljük, hogy a megfelelő inputokra a megfelelő működés történik ill. a hibás inputok nem okoznak hibát.

Load menü tesztjei:

- Korrektül elmentett állás betöltése
- Hibásan elmentett állás elmentése
- Üres állás elmentése
- Hibás hivatkozás

Save menü tesztjei:

- Korrekt mentés üres helyre
- Korrekt mentés foglalt helyre
- Hibás helyhivatkozás

Help teszt:

- Ki kell íródnia a help szövegnek

Quti teszt:

- No comment

Ha a menü jól működik akkor kezdjük a szabályok tesztelését.

### Játék (szabályok) ellenőrzése

Ebben a fázisban a már teljesen implementált játék és szabály objektumot teszteljük, főleg abból a szempontból, hogy a szabályokat kellőképpen ellenőrzi-e. Hogy ezt elérjük a felhasználó adja meg a játékban fellépő eseményeket. Ebben a fázisban még nem kell a játék-objektumon kívüli objektumok függvényeit kitölteni, hiszen (akár a szkeletonban) a tesztelő adja meg az eseményeket.

Az összes hibát “eljátsszuk”, majd ellenőrizzük a kimenetet. A tesztelésre kerülő események az alábbi négy csoportba oszthatóak:

- 1) Rossz lövés → Az ellenfél jön kétszer.
- 2) “Végzetes lövés” → A játékos elveszti a játékot.
- 3) Sikeres lövés → A játékos újra jöhet.
- 4) Nem hibás lövés → Az ellenfél jön.

Az eseményeket úgy kell megadni a teszt során, hogy a szabály-objektum által megvalósított autómata állapot-diagrammjának összes élén végigmenjünk legalább egyszer.

### Szimuláció ellenőrzése

A szimuláció tesztelésénél a szabályokkal már nem törődünk, csupán a golyók mozgását ellenőrizzük. Természetesen ebben a fázisban csak azt tudjuk ellenőrizni, hogy a megadott fizikai modell alapján mozognak-e a golyók. Az ellenőrzéséhez előre felállított játék-állásokat használunk. *Ezeket egy erre a célra készített (átalakított) programmal (állás-generátor) generáljuk.* Így tetszőleges szimulációs helyzetek generálhatóak. A szükséges tesztek az alábbiak:

- Golyó gurulása ütközés nélkül.
- Golyó pattanása egy falról.
- Golyó pattanása több falat érintve.
- Golyó lyukba esése.
- Két golyó lyukba esése egymás után.
- Két golyó lyukba esése “rex” állásban.
- Mozgó-álló golyó ütközése.
- Mozgó-mozgó golyó ütközése.
- Három golyó “egyidejű” ütközése.

A teszteknek az elvégzéséhez szükséges teszt-állások részletezésétől eltekintünk, mivel jobbára egyértelműek. A szimulációk elvégzése után a log-file-ban ellenőrizhetjük a kapott állás helyességét, hiszen az eseményeket nem tudjuk a képernyőn kellő sebességgel megfigyelni.

### ***A tesztelést támogató programok tervei***

A menü tesztjéhez nincsen szükség külön programra. A szabályok tesztelésekor egy módosított *játék* objektumra lesz szükségünk, amely hasonlít a szkeletonban megvalósítottra. Ennek segítségével a tesztelő adja meg a fellépő eseményeket, és így nyomon követheti a szabály-objektum állapot változásait.

A szimuláció tesztelésénél nélkülözhetetlen egy *állás-generátor* program készítése.

Beállítható paraméterek:

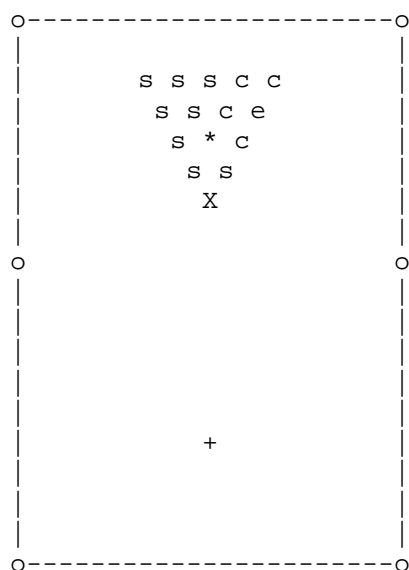
- Asztal mérete
- Falak helyzete és alakja
- Lyukak helyzete, sugara
- Golyók helyzete, sugara
- Golyók sebessége, surlódási együtthatója
- Játék állapota

A program a bemeneteket szöveges képernyőn kéri be, nem szükséges a “hülye”-biztos megvalósítás. Az ezzel a programmal generált pályákat már könnyedén el tudjuk menteni, így a tesztek tetszőlegesen megismételhetők. A tesztek végeredményének megjósolását a fizikai modell alapján részben papíron, részben számítógép segítségével számoljuk ki. A legfontosabb az elvárt események (koccanások, leesések stb.) bekövetkezésének ellenőrzése.

MELLÉKLET:

## SZABÁLYOK

Négy fajta golyót különböztetünk meg: fehér (1 db), fekete (1 db), csíkos (7db), sima (7 db). Ezeket a következő módon kell a játék kezdetekor felállítani:



A fehér golyót a (az asztallapon jelzett) '+' homlokpontra, a fekete golyót a (szintén jelzett) '\*' pontra, a többi golyót pedig a megadott módon a fekete köré kell helyezni (egyenlő oldalú háromszöget alkotva, melynek alapvonala párhuzamos az asztal szélével). X-el jelzett pont a tőpont. Fontos még, hogy a háromszög egyik sarkánál a 'c'-vel jelzett golyók mind egyformák (mind csíkosak vagy mind simák) legyenek, a közrefogott golyó pedig ezekkel ellentétes mintájú.

*Az hivatalos szabály alapján azt is ki kell kötnünk, hogy a háromszög ábra szerinti bal oldalán nem lehet az összes golyó ugyan olyan típusú.*

Csak a fehér golyót szabad meglökní a dákóval. A játékosok felváltva löknek. Aki az első golyót lyukba löki, az ettől kezdve ezekkel a golyókkal van, míg ellenfele a másik fajtaival. Innentől mindenki a fehér golyóval elsőként csak a saját golyóit lökheti meg (ezután persze akármelyik golyó koccanhat).

Ha valaki saját golyót juttat lyukba, akkor még egyszer jöhet.

Ha valaki

- a fekete golyót, vagy ellenfele egy golyóját találja el elsőként, vagy
- ellenfele golyóját is lyukba juttatja, vagy
- a fehér golyót lyukba juttatja, vagy
- nem koccantja a fehér golyót más golyóhoz,

akkor a soron következő játékos jön, és az előbbi egy körből kimarad. Ezek a szabályok nem érvényesek akkor, amikor bárkinek elfogytak a golyói, vagy még valakinek egy golyója sem esett lyukba.

*A hivatalos szabály szerint a hibás ütéssel eltett golyókat vissza kell állítani a tőpontra, ill. ha ez nem tehető meg akkor közvetlen mellé.*

Ha valakinek sikerült az összes golyóját lyukba juttatni, akkor az utolsó leeső golyó lyukával középpontosan szemben levő lyukba kell a fekete golyót juttatnia.

Az nyer, akinek a fenti elsőként sikerül. Veszít, aki a fekete golyót hamarabb, vagy rossz lyukba lövi.

*További kiegészítések: Ha valamelyik játékos lyukba gurítja a fehér golyót akkor azt a homlokpontra kell visszahelyezni. A hivatalos szabályok szerint a fehér golyót az egész homlokmezőben akárhová rakhatjuk kezdőlökés, illetve a golyó (lyukba lövése utáni) visszarakása esetén. (A homlokmező a homlokpont és a fal közötti rész.)*

MELLÉKLET:

## MEGVALÓSÍTANDÓ RENDSZER FIZIKAI MODELLJE

A rendszer leírásához a golyó-asztal, golyó-golyó és a golyó-dákó kölcsönhatást vizsgáljuk. A következő egyszerűsítési feltételekkel élünk:

- A golyók forgásával, gurulásával nem foglalkozunk
- Az ütközések során fellépő energiaveszteséget elhanyagoljuk (teljesen rugalmas ütközéssel számolunk)
- Feltétezzük, hogy a mozgó golyókra azonos nagyságú súrlódási erő hat
- Egyszerre mindig csak két test van kölcsönhatásban

Végeredményben a golyókat pontszerűnek tekintjük, de figyelembe vesszük kiterjedésüket.

Ezek alapján:

### 1. A golyó-asztal kapcsolat

1.1 A golyó a pálya közepén van; mozgásegyenlete:

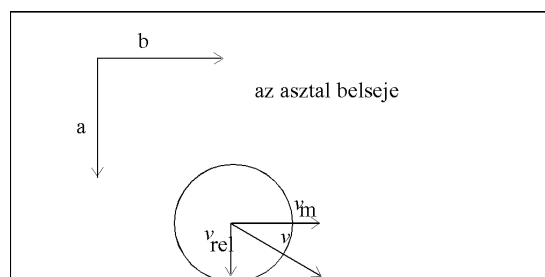
$$m_{\text{golyó}} \cdot (d\mathbf{v}_{\text{golyó}}/dt) = -|\mathbf{F}_{\text{súrlódás}}| \cdot \mathbf{v}_{\text{golyó}}/|\mathbf{v}_{\text{golyó}}|$$

1.2 A golyó középpontja a lyuk felett van

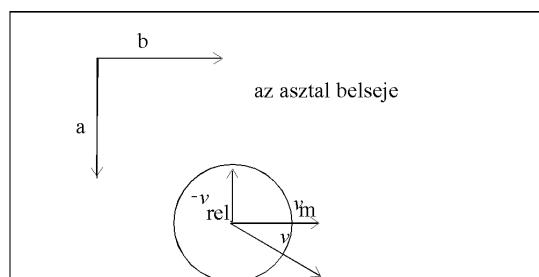
Ekkor az mindentől függetlenül beleesik a lyukba.

Megvitandó pont: Nagy sebesség esetén kipattanhasson-e

1.3 A golyó fallal ütközik



Az ütközés előtt



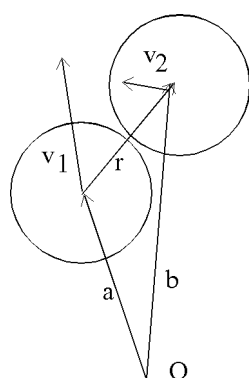
Az ütközés után

Ennek feltétele, hogy a golyó és a fal távolsága legyen kisebb, mint a golyó sugara, és legyen erre a falra merőleges “kifele” mutató sebessége. Legyen  $\mathbf{a}$  az asztal belsejétől ma falra mutató egységvektor,  $\mathbf{b}$  pedig legyen erre merőleges az asztal síkjában. Ekkor az új sebesség:

$$\mathbf{v}_{\text{új}} = -(\mathbf{a} \cdot \mathbf{v}) \cdot \mathbf{a} + (\mathbf{b} \cdot \mathbf{v}) \cdot \mathbf{b} \quad (* \text{ skalárszorozást jelöl})$$

Ahogy az egyszerűsítési feltételeknél kikötöttük a golyó legfeljebb 1 fallal ütközik.

## 2. A golyó-golyó kapcsolata



Két golyó ütközése

Feltéve, hogy az egyik golyót kétszeresére nagyítva a középpontjából a másik golyó középpontja benne van ebben, továbbá  $\underline{r}^* \underline{b} \geq \underline{r}^* \underline{a}$  és  $\underline{r}^* \underline{v}_1 \geq \underline{r}^* \underline{v}_2$  vagy  $\underline{r}^* \underline{b} < \underline{r}^* \underline{a}$  és  $\underline{r}^* \underline{v}_1 < \underline{r}^* \underline{v}_2$  esetben  $\underline{u}_1$  és  $\underline{u}_2$ -val jelölve az új sebességeket

$$\underline{u}_1 = (\underline{r}_m^* \underline{v}_1) * \underline{r}_m + \underline{r}^* \underline{v}_2$$

$$\underline{u}_1 = (\underline{r}_m^* \underline{v}_2) * \underline{r}_m + \underline{r}^* \underline{v}_1$$

összefüggéssel számolhatjuk.

## 3. A golyó-dákó kapcsolat

Csak álló golyók esetében. Ha a dákónak  $\underline{v}$  sebessége van, akkor a labdának is ez lesz a sebessége, feltéve, ha a dákó eltalálja azt.

## A rendszer szimulációja

A feladat számítógépes rendszeren való megvalósításához, egy folytonos idejű rendszert kell szimulálnunk diszkrét idejű rendszeren.

Legyen  $\Delta t$  időnként változtatjuk a rendszer állapotát, azaz ezalatt csak a golyók helyzete változik, és a kölcsönhatásokat nem vesszük figyelembe. Pontosabban legyen a golyó helyvektora a  $t$ . időpillanatban  $\underline{r}(t)$ , sebessége pedig  $\underline{v}(t)$ .

Ekkor:

$$\underline{r}(t+\Delta t) = \underline{r}(t) + \underline{v}(t) * \Delta t$$

$$\underline{v}(t+\Delta t-0) = \underline{v}(t)$$

$$\underline{v}(t+\Delta t) = \underline{v}(t) - |\underline{F}|/m_{\text{golyó}} * \underline{v}(t)/|\underline{v}(t)|$$

Ha  $|\underline{v}(t+\Delta t)| < 0$  akkor a golyó sebességét  $0$ -nak állítjuk be.

A  $t+\Delta t+0$  időpillanatban vesszük számításba a fizikai kölcsönhatásokat. A többi időpillanatban is eszerint a recept szerint járunk el.

Felmerült problémák:

- Milyen egységben mérjük  $\Delta t$
- Mekkora legyen ez a  $\Delta t$  idő, ez nyilván függ a golyók legnagyobb sebességétől (Túl nagy sebességek és  $\Delta t$  esetén a golyók átugorhatják egymást, kis  $\Delta t$  esetén pedig a



számítási pontatlanságok, és lassú szimuláció lesz az eredmény.  $\Delta t > T$  amelzre a gép kiszámítja az összes golyó új állapotát)

- A golyók mozgásának folytonosnak kell látszania, így a megjelenítő adottságait figyelembe kell venni.

Felhasznált irodalom:

*Budó Ágoston: Kísérleti fizika 1*

Felhasznált jelölések:

$\underline{a}$	<i>egy vektort jelent</i>
$ \underline{a} $	<i>a vektor hossza</i>
$\underline{a}*\underline{b}$	<i>vektorok közötti skalárszorítás</i>
$d\underline{a}/dt$	<i>vektor idő szerinti deriváltja</i>

## Értékelés

A projekt elkészítésében mind a négy résztvevő közel ugyan annyit dolgozott, ezért a százalékokat 25%-ra állítottuk be.

Boja Bence	Stefán Viktor	Vass Gergely	Zajác Balázs
25%	25%	25%	25%