Tartalomjegyzék

Bevezetés

Az alkalmazás célja, hogy a testek lejtőn való mozgását a szimuláció segítségével könnyebben, egyszerűbben és jobban megértesse azokkal, akik ezzel még nem ismerkedtek meg.

A program általános és középiskolások részére készül, de bármilyen korú érdeklődőnek hasznos lehet. Az ötletet az adta, hogy a tanári magyarázat sokszor kevés ennek a problémának a megértéséhez. A szemléltetés pedig kimerül táblai rajzokban és esetleg a fizikakönyvbeli ábrákban. Ezek a módszerek kevésbé látványosak, ami lekötné az erről tanulók figyelmét, és nehezebben lehet megérteni velük a problémát. Ezeknek a nehézségeknek a kiküszöbölésére készül ez a szimulációs program.

A szimulációnak különböző adatokkal, beállításokkal kell futtathatónak lennie. Fontos az egyszerű kezelhetőség, hogy a fiatal gyerekeknek ne annak megtanulásával kelljen foglalkozniuk.

A program különösen nagy segítséget nyújt majd a megértésben úgy, hogy a diákok által tanult grafikonokat is folyamatosan a futás közben kirajzolja. Ezzel nem egy kész diagramot kell megérteniük, amit a tanár szóbeli magyarázat mellett a táblára felrajzol, hanem folyamatosan látják a grafikon időbeli alakulását. Az általam fontosnak tartott diagramokat rajzoltatom ki. Ez a három az út – idő, sebesség – idő és a gyorsulás – idő diagramm. Ezek nélkülözhetetlenek a lejtőn mozgó test megértéséhez, és ezeket tanultuk mi is a középiskolában.

Ezzel a programmal majd játékosan tanulhatják meg a testek lejtőn való mozgását a diákok. Érdekesebbé téve a fizikaórát azok számára is, akiket eddig nem nagyon foglalkoztatott ez a tantárgy. Fontosnak tartom az oktatás érdekesebbé tételét, mert a tanároknak egyre inkább a médiával kell megküzdeniük a diákok figyelméért. Ez a program egy lépés e felé.

1. Követelményanalízis

1.1. Áttekintés

Ebben a fejezetben határozom meg, hogy a készülő programnak milyen követelményeknek kell megfelelnie. Az UML[1] (Unified Modelling Language, azaz egységesített modellező nyelv) segítségével terveztem meg a projektet. Ez a vizuális ábrázolási forma lehetővé teszi a könnyebb átláthatóságot mások és saját magam számára is. Ezért több helyen fogom a szakdolgozat során használni.

A program megvalósításához a JAVA nyelvet fogom használni (JAVA™ 2 SOFTWARE DEVELOPMENT KIT (J2SDK), STANDARD EDITION, VERSION 1.4.2_X). Grafikusan jelenítem meg, szimulálom a lejtőn való mozgást a felhasználótól bekért adatokkal. Ennek segítségével játékosan megtanulhatja, könnyen megértheti a program használója ezt a fizikai jelenséget. Főként tanulók, diákok részére készítem, de bárki másnak is hasznára válhat. Ezért törekszem az egyszerű, és egyértelmű kezelés megvalósítására.

1.2. Használati eset diagram (use case)

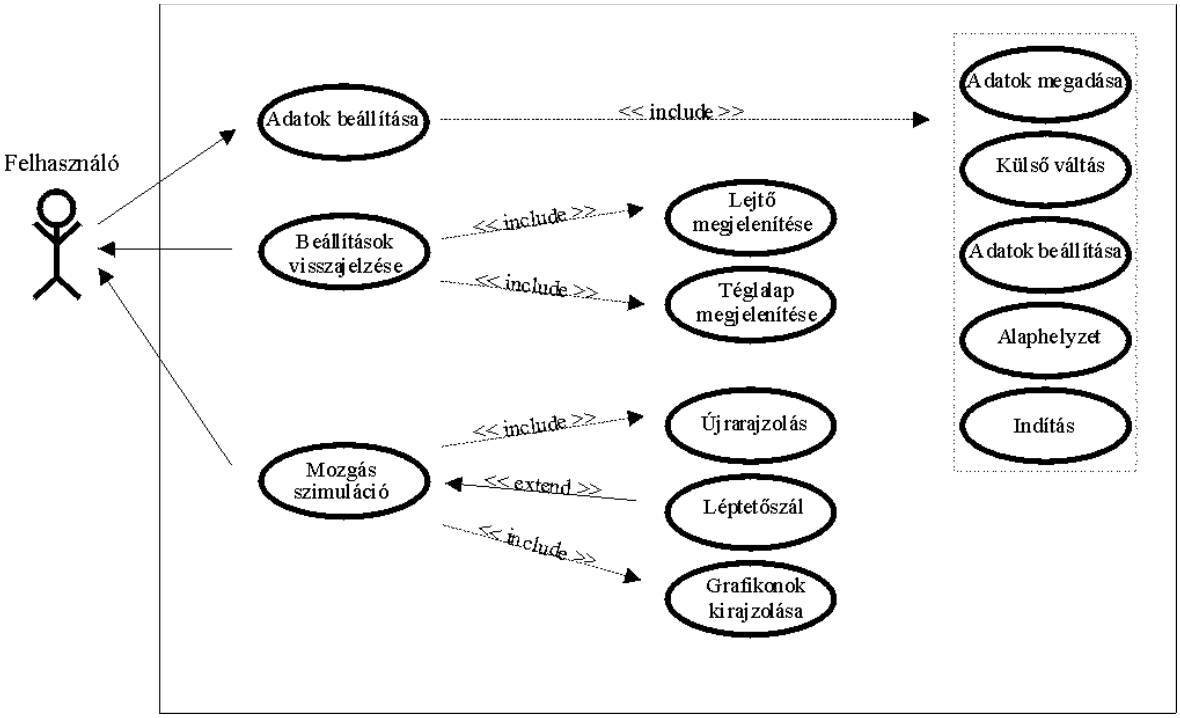
A használati eset diagram[2] a rendszer viselkedését írja le, ahogy az egy külső szemlélő szemszögéből látszik. Ezt használom a rendszerkövetelmények összegyűjtésére, és a kész rendszer tesztelésére.

Részei:

- Használati eset:
  - Jelölése: egy ellipszis névvel ellátva
  - Tevékenységek sorozatát mutatja
- Szereplő (actor):
  - Jelölése: egy embert ábrázoló figura névvel ellátva
  - Személy, felhasználó, aki kapcsolatba lép a rendszerrel
- Rendszerhatár:
  - Jelölése: egy téglalap
  - A rendszer és a külső közötti választó határt jelöli

[1] Tilly Károly: Az UML nyelv alapjai, 3.5 fejezet
[2] Majzik István: UML alapú rendszermodellezés, 12-23. o.

- Kapcsolatok:
    - Jelölése: egy vonal a végén egy nyíllal, ami a kapcsolat irányát jelöli
    - A használati esetek között lépnek fel
    - << include >>: a rész, a tartalmazott, ahova a nyíl mutat
    - << extend >>: változat, kiterjesztő



Használati eset diagram

1. ábra

Az *1. ábra* mutatja be az általam elkészítendő rendszer működését. Látható, hogy a képernyő előtt ülő felhasználó szemszögéből készítettem el a diagramot.

Adatok beállítása: ezen a felületen adhatja meg, és állítja be a programot használó személy a szimuláció paramétereit, és ezekkel az adatokkal indíthatja el a magát a szimulációt is. Ezenkívül itt választhatja ki a felhasználó a megszokott, vagy a legjobban tetsző külsőt az ablaknak.

Beállítások visszajelzése: az adatok beállítása esetén még az indítás előtt megjeleníti a testet és a lejtőt. Ezzel a rosszul beállított adatokat rögtön észre lehet venni, és ki lehet javítani.

Mozgás szimuláció: a *Léptetőszál* a fizikai képletek segítségével időegységenként megváltoztatja a test helyét és sebességét a lejtőn. Ezek folyamatos *Újrarajzolás*ával megkezdődik a szimulálás. Ez addig folytatódik, ameddig a test le nem ér a lejtő aljára, vagy el nem telik a maximálisnak beállított idő. Eközben a *Grafikonok kirajzolása* a képernyő jobb oldalán történik meg folyamatosan. Ez azt jelenti, hogy az előre megrajzolt tengelyek közé az idő elteltével arányosan kirajzolódik a függvény képe.

Ezek alapján látható, hogy a program külső szemlélő számára nagyon egyszerűen és könnyen kezelhető. Nem kell foglalkozni a használatának megtanulásával, figyelmét csak a szimulációra fordíthatja. Ez célszerű figyelembe venni egy program tervezésénél, mert a felhasználónak nincs ideje, vagy türelme bonyolult dolgokkal foglalkozni.

1.3. Alkalmazási példa

Adatok megadása, beállítása, megtekintése, szimuláció indítása:

- A felhasználó megadja a futtatás paramétereit.
    - Beírja a megfelelő szövegmezőbe a test kezdeti sebességét, tömegét, magasságát és hosszát, a lejtő dőlésszögét és súrlódását, majd kiválasztja a hozzá tartozó mértékegységet.
    - Ezen kívül meg kell még adni, hogy a szimulációt maximálisan mekkora ideig akarja nézni. Erre azért van szükség, mert speciális paraméterek megadása esetén hosszú ideig is eltarthat.
- *Beállítás* gomb lenyomásával a megadott adatoknak megfelelően kirajzolódik a szimuláció kezdeti állapota.
    - Megrajzolódik a lejtő és a tetejére a test a megfelelő méretekkel és formában.
- *Alapállapot* gombbal visszaállíthatók a paraméterek az alapértelmezettekre.
- *Kinézete* is beállítható itt az ablaknak.
- *Indítás* gomb a szimulációt elindítja.

Szimuláció és grafikonok megtekintése:

- A csúszás szimuláció, az út-idő, a sebesség-idő és a gyorsulás-idő diagrammok megjelenítését látja a felhasználó.
- A megtekintés után, vagy közben az előző, az adatok beállítása ablakra lehet váltani a *Vissza* gomb segítségével. Visszakerülve az előző ablakra a szimulációt új adatokkal futtathatja, próbálgatva ezt a fizikai jelenséget.

A használati eset diagram és az alkalmazási példa alapján láthatóak a program iránt nyújtott fontosabb igények megvalósulása.

Például:

- A felhasználó könnyen az általa igényelt formában futtathatja le a szimulációt.
- Megjelenítésre kerülne az iskolai oktatást, tanulást segítő grafikonok.
- Hibás adatokkal történő futtatás esetén könnyen vissza lehet lépni, és vissza lehet állítani a helyes paraméterekre a programot.
- Különböző mértékegységeket lehet használni az egyes paraméterek megadásánál.

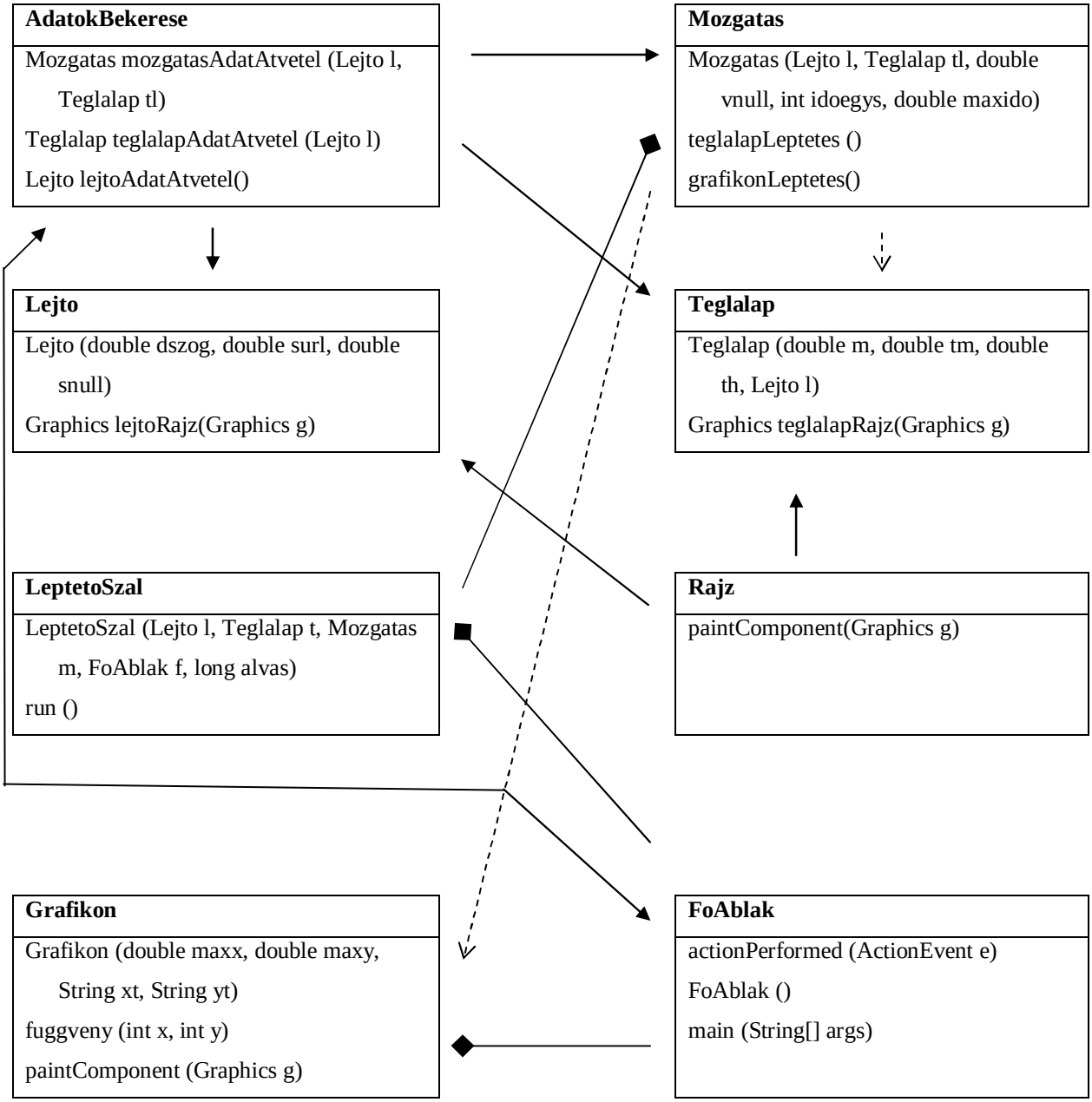
2. Modellezés

Az UML nyelv segítségével az elkészítendő alkalmazás modellezése egyszerű és egyértelmű. Az objektumorientált programozás tervezését nagymértékben elősegíti, és áttekinthetővé teszi a szoftver működését mások számára is.

2.1. Osztálydiagram

Az Osztálydiagramok[3] az objektumok és a közöttük lévő kapcsolatok rendszerét mutatja meg. A *2. ábrán* látható az általam készített osztálydiagram. Ez nem teljes, csak a lényegesebb osztályokat emeltem ki itt, és nem tüntettem fel benne az adattagjaikat.

[3] Nyékyné Gaizler Judit: Java 1.1 útikalauz programozóknak, 318-328 o.
Majzik István: UML alapú rendszermodellezés, 26-29. o.



Osztálydiagram

2. ábra

Jelmagyarázat:

⟶ *Asszociációk (társítás):* valamilyen használati kapcsolat a két osztály között, a két osztály egymástól független.

⟶◆ *Kompozíció:* erősebb, mint az asszociáció, tartalmazási viszonyt jelent.

- - - -➔ *Függőség:* a két elem közötti kapcsolat, ahol egyik változása befolyásolja a másikat.

2.2. Osztályok leírása:

A részletes osztályterv[4] elkészítése a korábbi lépések figyelembevételével történik.

- **Foablak** osztály: az alkalmazás főosztálya.
    - **Feladatai:**
        - Létrehozza a megjelenítéshez a felületet, ablakot.
        - Adatok bekéretése és átvétele.
        - A megkapott adatokból létrehozza a grafikonokat.
        - Kezeli a felhasználótól érkező üzeneteket.
    - **Metódusai:**
        - *public static void main(String[] args):* A program főfüggvénye. Meghívja az osztály konstruktorát, és láthatóvá teszi.
        - *FoAblak():* Az osztály konstruktora. Létrehozza a felületet. Létrehozza az AdatokBekerese objektumot. Átveszi tőle a Lejto, a Teglalap és a Mozgatas objektumot azok adataival.
        - *public void actionPerformed (ActionEvent e):* akciófigyelő függvény.

- **AdatokBekerese** osztály: feladata az adatok bekérésére, és azok eljuttatása a megfelelő objektumok részére.
    - **Metódusai:**
        - *public AdatokBekerese(FoAblak fa):* egy FoAblak típusú objektumot kap, és erre létrehozza és ráhelyezi a mezőket és cimkéikét.
        - *public Lejto lejtoAdatAtvetel():* létrehozza a lejto objektumot a bekért adatokkal.
        - *public Teglalap teglalapAdatAtvetel(Lejto l):* létrehozza a teglalap objektumot, és a lejtő tetejére rakja.
        - *public Mozgatas mozgatasAdatAtvetel(Lejto l,Teglalap tl):* helyfoglalás a mozgatas objektumnak a megfelelő adatokkal.
        - *public void actionPerformed(ActionEvent e):* akciófigyelő függvény.

[4] Nyékyné Gaizler Judit: Java 1.1 útikalauz programozóknak, 329-332 o.

- **Grafikon** osztály: a grafikonok kirajzolásáért, és adataik tárolásáért felel.
    - **Metódusai:**
        - *Grafikon(double maxx,double maxy,String xt,String yt):* megkapja a grafikon nevét és maximális értékét.
        - *public void fuggveny(int x,int y):* hozzáadja egy új pont koordinátáit minden meghíváskor egy tömbhöz. Kirajzolásnál, frissítésnél ez is kirajzolódik, és ezzel elkészül a függvény.
        - *public void visszaall():* a pontokat tartalmazó tömb indexét állítja 0-ára, hogy új adatokkal való futtatáskor újrakezdődjön a kirajzolás.
        - *public void paintComponent(Graphics g):* kirajzolja a tengelyeket, azok neveit, mértékegységeit és méreteit. Azután kirajzolja a függvényt az idő elteltével arányosan.
        - *public void meretezo(double x,double y):* méretezi a grafikon tengelyeit, és a kirajzolandó pontokat.

- **Alapallapot** osztály: segítségével visszaállíthatók a benne tárolt adatok.
    - **Metódusai:**
        - *public Alapallapot():* az adatok tárolására szolgál.
        - *public void visszaallit(JTextField v0,JTextField alfa,JTextField surl,JTextField m,JTextField tm,JTextField th,JTextField maxs,JTextField maxt):* beállítja az alapértelmezett paramétereket a megfelelő mértékegységgel.

- **Lejto** osztály: a lejtő objektumot és annak kapcsolatait hozza létre.
    - **Metódusai:**
        - *public Lejto(double dszog,double surl,double snull):* az osztály konstruktora. Feladata a lejtő kezdeti adatainak beállítása.
        - *private void helyzet():* meghatározza a lejtő pontjait.
        - *public Graphics lejtoRajz(Graphics g):* kirajzolja a lejtőt egy Graphics objektumra.
        - *public void meretezo():* képernyőhöz igazítja, méretezi a lejtőt.

- **Teglalap** osztály: a téglalapot megjelelő méretben a helyére teszi. Kiszámítja a mozgás közben az új koordinátákat.
    - **Metódusai:**
        - *public Teglalap(double m,double tm,double th,Lejto l):* ez a metódus az osztály konstruktora. A kezdeti adatok átvétele, és beállítása a feladata.
        - *public int[] getTeglalapX():* szimuláció közben kialakuló új X koordinátákat számítja ki, és állítja be.
        - *public int[] getTeglalapY():* szimuláció közben kialakuló új Y koordinátákat határozza meg, és aktivizálja azokat.
        - *public Graphics teglalapRajz(Graphics g):* Graphics objektumra helyezi az éppen aktuális téglalapot.

- **LeptetoSzal** osztály: időegységenként újraszámolja a szükséges értékeket, ezzel idővel arányos mozgást tesz lehetővé.
    - **Metódusai:**
        - *public LeptetoSzal(FoAblak f, double alvas):* az osztály konstruktora. A megkapott adatokat állítja be.
        - *public void run():* a szál indulásakor indul, és a fizikai szimuláció végéig tart. A téglalapnak és a grafikonoknak új értéket beállító metódusokat hívja meg, majd a képernyőre való újrarajzolásukról gondoskodik.

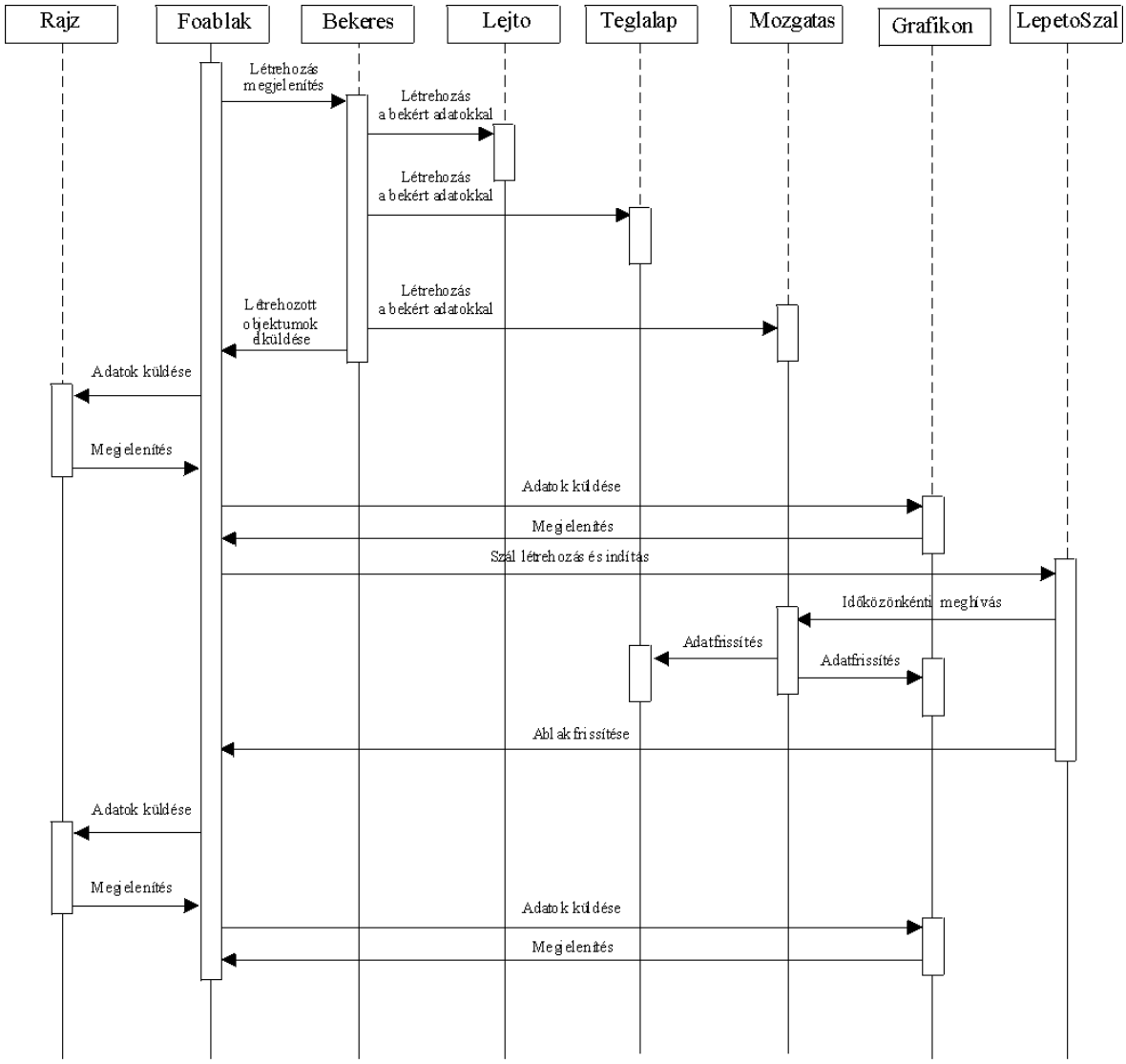
- **Rajz** osztály: feladata a lejtő és a rajta lévő test egy komponensre helyezése.
    - **Metódusa:**
        - *public Rajz(Lejto l,Teglalap tl,Mozgatas mozg):* átveszi a megfelelő objektumokat, és beállítja a szükséges adatokat.
        - *public void paintComponent(Graphics g):* a megfelelő helyre, különböző színben rajzolja ki az objektumokat.

- **Mozgatas** osztály: a megfelelő mozgás, vagy a megállás meghatározása, és az ehhez szükséges függvények meghívása.
    - **Metódusai:**
        - *public Mozgatas(Lejto l,Teglalap tl,double vnull,double idoegys,double maxido):* az osztály konstruktora. Egyrészt beállítja az objektum megkapott adatait, másrészt meghívja a *erokSzamolasa() és maxErtekek()* metódust, ami további kezdeti értékeket számít ki, és állít be.
        - *public void erokSzamolasa():* fizikai képletek és a kapott adatok segítségével az erőket és a gyorsulást számítja ki.
        - *public void utSzamolasa():* a szimuláció kezdetétől eltelt idő állítja be, tartja nyilván. Kiszámítja az aktuális sebességet, és segítségével meghatározza a kezdeti helyzetétől megtett utat.
        - *public void maxErtekek():* a futás közben kialakuló maximális utat, sebességet, időt és gyorsulást számítja ki, és állítja be. Erre a grafikonok tengelyeinek meghatározásához, és kirajzolásához van szükség.
        - *public void teglalapLeptetes():* meghívja az *utSzamolasa()* metódust, és a megtett út segítségével kiszámítja a téglalap új koordinátáit.
        - *public void grafikonLeptetes(Grafikon grafikon1,Grafikon grafikon2,Grafikon grafikon3)*: megkapja a grafikonokat, meghívja a pontok méretezését végző függvényt, és megadja nekik az új kirajzolandó pont koordinátáit.

2.3. Szekvencia diagram

A szekvencia diagram[5] az egyes objektumok közötti kölcsönhatásokat mutatja be az idő függvényében, és az általuk küldött üzenetek sorrendjét ábrázolja. Vízszintesen helyezkednek el egymás mellett azok az objektumok, amelyek részt vesznek a folyamatban. A függőleges tengely az időt reprezentálja. Az objektumok közötti nyilak jelölik az üzeneteket. A nyíl az üzenet küldőjétől indul, és a hegye mutat az üzenet fogadójára.

Szaggatott vonal jelzi, hogy az adott objektum még nem létezik, vagy már megszűnt. A folyamatos vonal azt jelenti, hogy az objektum létezik, de nem aktív. Téglalap mutatja, amikor az objektum aktív, műveletet hajt végre és/vagy másik objektum vezérli.



Szekvencia diagram

3. ábra

[5] Tilly Károly: Az UML nyelv alapjai, 3.5.4. fejezet

3. Tesztelés

A tesztelés[6] magába foglalja a rendszer próbafuttatását, a valós működés szimulálását, annak ellenőrzése, hogy jól értettük-e a követelményeket, a követelmények mindegyikének van-e megfelelője a modellekben, és a dokumentáció / kód átolvasását.

Egyre szükségesebbé válik a programok tesztelése:

- A tapasztalatok azt mutatják, hogy a szoftverben hibák vannak, és ezek közül minél többet szeretnénk megtalálni, még mielőtt a felhasználó találná meg őket, és mielőtt bajt okoznának.
- Egyre nagyobb és összetettebb szoftverek készülnek. A szoftverfejlesztés korszerűsítésének köszönhetően ezek a programok is átláthatók, de ezek sem nyújtanak tökéletes biztonságot. Ezért nehéz jó minőségű, tökéletesen működő alkalmazást létrehozni.

Általában a terméket, különböző kritériumok alapján teszteljük:

- funkcionalitás
- megbízhatóság
- teljesítmény
- egyéb jellemzők

A tesztelés információt ad a program minőségéről, a programban maradt hibákról.

3.1. Funkcionális tesztelés[7]

Ennek a tesztelési formának az a célja, hogy a program működését ellenőrizze rendeltetésszerű használat közben, és felderítse az esetleg felmerülő hibákat.

[6] http://rs1.szif.hu/~heckenas/okt/sga1.pdf
[7] www.itware.hu/szakteruleteink.php?pgid=szoftteszt

Installációs teszt:

Itt az alkalmazás telepítését ellenőrizzük különböző hardver-szoftver konfigurációk esetén. A JAVA nyelv lehetőséget nyújt arra, hogy különböző operációs rendszereken is fusson a program. Ehhez fel kell telepíteni a futtatókörnyezetet, ami a felhasználó által használt operációs rendszer (linux, MS Windows, stb.) alá terveztek.

Általános funkcionális teszt:

Feladata a rendszer megfigyelése normál működés esetén.

Ebben az esetben, különböző – a felhasználó által megadott – adatokkal futtattam a programot. Hibás működést nem tapasztaltam.

- A bevitt adatoknak megfelelően, egymáshoz arányosan rajzolódnak ki a szimulációhoz szükséges egységek.
- A fizikai képletekkel utána számoltam, és megegyezett a program által adott végeredményekkel.

Szélsőérték funkcionális teszt:

A rendszer működését vizsgálja meg ez a teszt abban az esetben, ha a bemenetére szélsőséges adatokat kap. Itt vizsgáljuk meg azt, hogy a bekért adatok megfelelnek e a követelményeknek. Például csak számot adott e meg a felhasználó, esetleg nem túl nagyok, vagy nem túl kicsik az általa beírt számok.

Ide vettem még azt is, amikor a megengedett határértékeket adta be a felhasználó.

Például:

- A lejtő dőlésszögét 90°-nak állította be a felhasználó, akkor eltűnik a lejtő, és a szabadesést szimulálja a program.
- Amikor olyan esetet ad meg a felhasználó, hogy a lejtőn lévő test felborulna, az ablakban egy üzenet figyelmezteti erre.

Konfigurációs teszt:

Ez a rendszer különböző hardver és szoftver környezetben történő használatát ellenőrzi. Elméletileg a futtatókörnyezettől függ a program működése. Ahol a futtatókörnyezet megfelelően tud működni, ott ennek az alkalmazásnak is futnia kell. Ennek gyakorlati tesztelését úgy végeztem el, hogy több számítógépen is futtattam a programot. Problémát nem tapasztaltam telepítés és használat közben sem.

3.2. Teljesítménytesztelés[8]

Itt vizsgáljuk meg, hogy a program mennyire használja, foglalja le a rendszer erőforrásait.

Az alkalmazás tervezésénél és elkészítésénél figyelembe vettem, hogy kisebb és nagyobb teljesítményű gépeket fogják futtatni. Ezért beépítettem az adatok bekéréséhez egy olyan mezőt, ahol a felhasználó beállíthatja, hogy a program milyen gyakorisággal frissítse az ablakot. Ezt ezredmásodpercben kell megadni. Alapértelmezetten ez 25 ezredmásodperc, ami 40 Herznek felel meg. Ebben az esetben szemmel nem igazán vezető észre vibrálás, és viszonylag kisebb teljesítményű gépeken is tökéletesen fut.

3.3. Tesztelés középiskolában

A programot a Kandó Kálmán Szakközépiskola és Szakiskola 11 / A osztályának 11 fős csoportjának diákjaival teszteltettem, majd a véleményüket kértem.

A diákok szerint a program használata egyértelmű, és segítette a tananyag megértését.

Többen hiányolták belőle, hogy rosszul megadott adat esetén a program automatikusan visszaállította az alapértéket, de nem írta ki milyen tartományban lehet változtatni az értékeket. Ezt a problémát ki is javítottam, mert én is úgy látom, hogy a minimum és a maximum határok nem minden esetben egyértelműek.

Néhányan szerették volna, ha többféle extrém adattal is futtatható lett volna a program, de ezt fölöslegesnek, és problémásnak érzem. A beépítését nem tartanám jónak.

4. Üzembe helyezési útmutató

3.2. Teljesítmény tesztelés pont alatt említettem, hogy a felhasználó állíthatja be a saját számítógépének teljesítményéhez képest a hardver igényeket.

Fel kell telepíteni a JAVA virtuális gépet, ami lehetővé teszi azt, hogy a különböző operációs rendszeren is ugyanúgy tudjon futni a program.

Az alkalmazás használata ezek után „gyerekjáték”, bárki számára érthető és egyértelmű. A *fizika.bat* állományt kell elindítani, majd a megjelenő adatbekérő mezőkbe kell beírni a kezdeti, induló adatokat. Az *Indítás* gombbal indulhat is a szimuláció.

[8] www.itware.hu/szakteruleteink.php?pgid=szoftteszt

5. Program részletek

Ebben a fejezetben kiemelek néhány általam alkalmazott, a szoftver szempontjából fontos részletet. Ezzel próbálom egyszerűen végigvezetni, hogy hogyan működik a program. Az UML nyelv segítségével a tervezést és a működést leírtam már. Itt a megvalósítás utáni kódot fogom elemezni, persze csak főbb pontjaiban.

5.1. Méretezés

A program induláskor lekérdezi a képernyő felbontását, és ennek segítségével az ablakot és a lejtőt méretezi.

A *meretezo()* függvény kiválasztja a lejtő hosszabbik oldalát, és elosztja vele az ablakban neki szánt méretet. Ennek az eredményét helyezi a *szorzo* nevű változóba, ami segítségével minden arányosan rajzolódik ki a képernyőre.

```
public void meretezo(){
  if(-felsoPont.y>alsoPont.x) nagyobb=-felsoPont.y;
  else                        nagyobb=  alsoPont.x;
  szorzo=meret/nagyobb;
}
```

5.2. A csúszó test pontjai

A téglalap bal alsó pontja kezdeti állapotban a lejtő csúcspontjával egyezik meg. Ezért az origót áthelyezem ebbe a pontba az egyszerűség kedvéért. A mozgás elkezdése után ettől a ponttól lévő távolság lesz a test új koordinátája, majd kiszámítja a téglalap x és y koordinátáit a képernyőre rajzoláshoz.

```
public int[] getTeglalapX(){
  int[] x=new int[4];
  x[0]=getBalAlsoPont().x;
  x[1]=x[0]+xfuggoleges;
  x[2]=x[1]+xvizszintes;
  x[3]=x[2]-xfuggoleges;
  return x;
}
```

```
public int[] getTeglalapY(){
  int[] y=new int[4];
  y[0]=-(getBalAlsoPont().y);
  y[1]=y[0]-yfuggoleges;
  y[2]=y[1]+yvizszintes;
  y[3]=y[2]+yfuggoleges;
  return y;
}
```

5.3. Borulás, vagy csúszás

A test és a lejtő formájából, és a súrlódási együttható értékétől függ, hogy a test csúszik, vagy felborul. Ennek figyelésére ez az egyszerű függvény szolgál. Ha nem csúszik a test, akkor egy üzenetet kap a felhasználó ezzel kapcsolatban. Ezt azért tartottam fontosnak megemlíteni, mert a benne lévő rész kiszámítása komoly feladatot jelentett. Nem található meg így egy fizika könyvben sem, ezért ki kellett számítani, hogy mi is kerüljön az if() függvény belsejébe.

```
public boolean getCsuszik_e(){
  if(hossz/magassag >= SINALFA/COSALFA || hossz/magassag >= surlodas ) return true;
  return false;
}
```

5.4. Az erők és az út kiszámítása

Fizikai képletek segítségével meghatároztam az erőket. Az erők segítségével a test gyorsulását, ami fontos szerepet játszik a megtett út kiszámításában.

Az előző futtatás óta eltelt idővel (idoegyseg) megnövelem a *t* változó értékét, és megkapom, hogy az indítástól mennyi idő telt el. Ennek segítségével az aktuális sebességet, az aktuális sebességgel pedig a megtett utat tudom beállítani. A megtett út ismeretében határozom meg a téglalap új koordinátáit a kirajzoláshoz. Ezenkívül az út, a sebesség, a gyorsulás és az idő pillanatnyi értéke szerint rajzolódik ki a grafikonokon az újabb pont, ami segítségével kialakul a függvény gráfja.

```
public void utSzamolasa(){
  t+=idoegyseg;
  if(v>0 || v>=0 && a>=0)v=a*t/1000+v0;
  else v=0;
  s+=v*idoegyseg/1000;
}
```

Összefoglalás

Az eddig leírtak alapján látszik, hogy a program a kitűzött célokat teljesíti. Megfelel azoknak az elvárásoknak, amiket a bevezető részben leszögeztünk. Könnyen kezelhető, a mozgást tökéletesen szimuláló, és az eredményeket, a folyamatot grafikonokkal ábrázoló szoftver született.

A JAVA nyelv jelenleg az egyik leggyorsabban fejlődő programozási nyelv. Ez a tisztán objektumorientált nyelv lehetőséget biztosít a későbbi igények és problémák egyszerű és gyors megoldására.

Az alkalmazás kinézetének javítását, praktikusabbá tételét, és bővítését javaslom. Próbáltam a bővíthetőséget is szem előtt tartani, hogy a későbbiekben ebből egy szimulációkat összefoglaló nagyobb méretű alkalmazás lehessen. Például bővíteni lehetne a rezgések, a felhajtóerő, az ütközések és sok más probléma, érdekesség bemutatására is. Ezáltal egybegyűjtene minden ilyen típusú feladatot, és az azok közötti összefüggések is sokkal jobban láthatóak lennének.

Irodalomjegyzék

Tilly Károly: Az UML nyelv alapjai.
BME Méréstechnikai és Információs Rendszerek tanszék, Budapest (2004)
http://www.inf.mit.bme.hu/~varro/uml/slides/uml1.pdf

Majzik Istvan: UML alapú rendszermodellezés.
BME Méréstechnikai és Információs Rendszerek Tanszék, Budapest (2004)
http://www.inf.mit.bme.hu/~varro/uml/slides/UML.ppt

Nyékyné Gaizler Judit (szerk.): Java 1.1 útikalauz programozóknak
ELTE TTK Hallgatói Alapítvány, Budapest (1998)

Dirk Luis, Peter Müller: Java – Bevezetés az internet programozás világába
Panem Kiadó, Budapest (2002)

Sike Sándor, Varga László: Szoftvertechnológia és UML
ELTE Eötvös Kiadó, Budapest (2003)

Daniel J. Berg, J. Steven Fritzinger: Java felsőfokon
Kiskapu Kft., Budapest (1999)

http://rs1.szif.hu/~heckenas/okt/sga1.pdf

http://www.itware.hu/szakteruleteink.php?pgid=szoftteszt

Mellékletek jegyzéke

1. számú melléklet

1 CD rajta az általam használt JDK telepítője, az alkalmazással és a forráskóddal

2. számú melléklet

A szakközépiskolában végzett felmérés megoldott feladatai (11 db)