SZAKDOLGOZAT

Révész László

Debrecen

2009

Debreceni Egyetem
Informatikai Kar

MOBILTELEFONOS JÁTÉKFEJLESZTÉS

Témavezető:
Bátfai Norbert
Egyetemi tanársegéd

Készítette:
Révész László
Programtervező informatikus (BSc.)

Debrecen
2009

Tartalomjegyzék

Bevezetés

A szakdolgozatom címe: Mobiltelefonos Játékfejlesztés. Már első hallásra is nagyon érdekesnek ígérkezik a téma.

Manapság, amikor különféle kimutatások szerint hazánkban több a mobiltelefon-előfizetések száma, mint a lakosság lélekszáma, óriási érdeklődésre tart számot ez a technológia. A fejlődésnek pedig még koránt sincs vége!

Jó néhány éve rendelkezem én is előfizetéssel, az évek során a készülékek azonban cserélődtek a kezemben, így saját magam is tapasztaltam azt a robbanásszerű fejlődést, ami ebben a szegmensben bekövetkezett és tart napjainkban is.

Míg évekkel ezelőtt a telefonokat szinte kizárólag az alapfunkciók ellátására használták tulajdonosaik (beszélgetés, szöveges üzenetek), addig az újabbnál újabb készülékek megjelenése egyre inkább lehetővé teszi, hogy szórakoztatási feladatokat is el tudjanak látni a telefonok. Az egyre nagyobb méretű és felbontású kijelzők, az egyre gyorsabb processzorok, az egyre nagyobb kapacitású memóriák, háttértárak mobiltelefonokba történő beépítésével egyre inkább fejlődnek a multimédiás képességei a készülékeknek, és mind nagyobb lehetőség nyílik a játékfejlesztők számára is, hogy egyre jobb és szórakoztatóbb programokat készíthessenek.

Én magam is szívesen játszom a telefonomon egy jó játékkal szabadidőmben vagy, ha éppen várakoznom kell valahol. Mindig is foglalkoztatott a játékok világa, szerettem volna belekóstolni a játékfejlesztés szakmai részébe. Erre most érkezett el az idő, a tanulmányaim során a mobilprogramozás megismerésével.

Dolgozatom célja megmutatni a mobiltelefonokon futtatható játékok programozásának alapjait. Bemutatni, hogy melyek azok az alapvető eszközök, lehetőségek, megoldások, amelyek rendelkezésünkre állnak, és hogyan lehet elindulni azon az érdekes úton, amelyen a mobiltelefonos játékfejlesztés világába érhetünk.

Ennek megfelelően lépésről lépésre építem fel a dolgozatomat. Terveim szerint először áttekintem a mobiltechnológia történetét (persze csak kivonatosan), aztán megnézem, hogy milyen típusú, illetve milyen operációs rendszerrel ellátott készülékeket kínálnak a gyártók. Ezután kiválasztom a játékprogramozáshoz legjobban alkalmazható technológiát, és annak a

legfontosabb, a munkám során használt jellemzőit ismertetem.

Ezt követi majd a fejlesztőkörnyezet kiválasztása és bemutatása, aztán a konkrét mobilplatform és készülék kiválasztása, illetve a jellemzőinek ismertetése fog következni.

Mindezek után szeretnék rátérni egy konkrét játékprogram részletes bemutatására, elmondom, hogy mik azok az eszközök, amelyeket felhasználtam, milyen megoldásokat alkalmaztam a fejlesztés során. A leírást az általam alkalmazott forráskódrészekkel kívánom érthetőbbé tenni.

A végén, a függelékben helyet fog kapni az általam használt fontosabb rövidítések magyarázata is.

A dolgozatban az illusztráció kedvéért ábrákat is el szeretnék helyezni. Ezek a forráskódrészletekkel együtt a megértést segítik.

Ennyit a tervekről – vágjunk bele!

1. A mobiltelefónia rövid történeti áttekintése

A mobiltelefónia az utóbbi évtizedekben óriási fejlődésen ment keresztül, mely napjainkban is tart. Ennek köszönhetően a mobilkészülékek mind több és több funkciót képesek ellátni. Olyanokat is, amelyek megvalósítása a technológia széleskörű elterjedésének kezdetén még óriási kihívásnak vagy egyenesen lehetetlennek tűnt.

Az első, polgári személyek számára is igénybe vehető mobiltelefonos rendszer már 1946-ban üzemelt. Az ún. 0. generációs rendszerek (0G) jellemzője, hogy a lefedett terület bázisállomásokon alapuló cellákra volt osztva, azonban közöttük manuálisan kellett váltani. Így, ha valaki egy másik cellába lépett át a készülékkel, miközben beszélgetést folytatott, a kapcsolat megszakadt.

A következő lépcsőfok a fejlődésben az 1. generációs hálózat (1G) megjelenése volt, melyet a '80-as évektől kezdtek üzemeltetni. Ez a hálózat jóval több csatorna használata mellett már támogatta a cellák közötti automatikus váltást, illetve tervezési szempont volt a kialakításkor, hogy kapcsolhatók legyenek a vezetékes telefonhálózathoz.

Az eddigi rendszerek mind analóg rendszerek voltak, és legfőképpen beszédátvitelre voltak alkalmasak. Az áttörést a 2. generációs hálózatok (2G) megjelenése okozta. Ekkor veszették be a digitális technológiát a jelátvitelre. Mivel a digitálisan kódolt jel jól tömöríthető, jóval több csatornát tudtak alkalmazni, mint az ugyanakkora sávszélességet használó analóg rendszerekben. A digitális készülékek kisebb teljesítményű rádiót használtak, ami kisebb energiafogyasztást, ezáltal jóval kisebb méretet tett lehetővé. Ekkor lettek jelentősen olcsóbbak és hatékonyabbak a mobilkészülékek. Ez volt az egyik ok, amely a mobiltechnológia robbanásszerű elterjedéséhez vezetett. A másik pedig a szolgáltatások bővítése volt. Így az SMS rendkívüli népszerűségre tett szert az egymás közötti kommunikációban, majd az MMS szolgáltatás elindítása következett, később pedig a csomagkapcsolt adatátvitel bevezetése lendített nagyot a mobiltechnológia népszerűségén.

A 3. generáció (3G) olyan új szolgáltatások mobiltelefonon keresztül történő használatát tette lehetővé mint az internetelérés, e-mailfiók elérése, videotelefonálás, mobiltelevíziózás. Mindezt a nagysebességű adatátvitel továbbfejlesztésével, illetve a megnövekedett kapacitással érték el (több felhasználó kiszolgálása a cellákon belül).

A jövő fejlesztési tervében (4G) szerepel a teljes egészében csomagkapcsolt hálózat kialakítása IP alapokon IPv6-ra történő átállással, ami lehetővé tenné, hogy minden egyes készüléknek egyedi IP-címe legyen. Amennyiben azt a – szintén csak tervekben szereplő – feltételt is sikerül teljesíteni, hogy a világ minden pontján legalább 100 Mbit/s adatátviteli sebesség elérhető legyen, soha nem látott mobilitást lehet majd biztosítani a felhasználóknak.

A mobiltelefonos technológia fejlődésével egyidőben a készülékeken használható alkalmazások iránt is egyre nagyobb igény mutatkozik. Ezen alkalmazások egyik fontos szegmense a mobiltelefonos játékoké.

2. Mobilkészülékek és -szoftverplatformok ismertetése

Ma már rengeteg készülék kapható a kereskedelmi forgalomban különböző gyártók kínálatában, melyek között érdemes egy kis csoportosítást végezni.

2.1. Mobilkészülékek csoportosítása

A ma használatos mobilkészülékek több csoportba sorolhatók. Bár a határok egyre inkább elmosódnak közöttük, a főbb csoportoknak vannak olyan jellegzetességei melyek alapján különbséget lehet tenni közöttük.

2.1.1. Általános mobiltelefon

A mindennapi életvitelhez kialakított, viszonylag kevés funkcióval ellátott készülékek tartoznak ide. A legtöbb készülék ebbe a csoportba tartozik. Ebben a szegmensben is igyekeznek egyre sokoldalúbb eszközöket készíteni, ugyan széleskörű üzleti funkciókat ritkán találunk bennük, viszont sok egyéb alkalmazás fejleszthető rá, különösen a játékok népszerűek ebben a kategóriában.

2.1.2. Okostelefon

Az általános célú mobiltelefonok funkcióin kívül bőséges támogatást nyújtanak üzleti felhasználók számára. Általában többféle kommunikációs protokollt támogatnak. Az e-

mailezés és webböngészés az alapfunkciók közé tartozik. Ezen szegmens készülékeire általában akár többféle nyelven is fejleszthetünk különféle alkalmazásokat.

2.1.3. PDA (palmtop)

Ezen készülékeket általában az üzleti felhasználáshoz alakították ki. A hangsúly mindenképpen a szervezőfunkció felé tolódik, amit a nagyméretű érintőképernyő is hangsúlyoz. Noha általában alkalmasak a hagyományos feladatokra is, általában nem a telefonálás az elsődleges cél. Természetesen ezen készülékre is fejleszthetők különféle alkalmazások.

2.2. Szoftverplatformok

Ahhoz, hogy egy készülékre alkalmazást lehessen készíteni, ismerni kell annak szoftverplatformját, ami meghatározza a használható programnyelvet, a fejlesztőeszközt, a rendelkezésre álló lehetőségeket, támogatott eszközöket, illetve a szükséges fejlesztési időt is. A mai készülékek különféle platformot képviselhetnek, melyek közül a legelterjedtebbeket ez a fejezet tartalmazza.

2.2.1. Symbian OS

A '80-as évek végén kezdték fejleszteni, mára az egyik legelterjedtebb mobil-operációs rendszer. Magas rendelkezésreállású, biztonságos, kis erőforrásigényű rendszernek tervezték. Maga a szoftver C++ nyelven íródott, így ezen a nyelven natívan lehet hozzáférni a készülék erőforrásaihoz. A Symbian OS tulajdonképpen a mai változatára épül egy UI réteg, amely készüléktípusonként eltérő lehet. Ez a réteg biztosítja a beviteli eszközök, kijelzők és az operációs rendszer közötti kapcsolatot. Az eltérő kialakítású eszközök hatására különböző keretrendszereket fejlesztettek a Symbian köré. A ma használatos két legelterjedtebb:

- **S60:** eredetileg a Nokia alakította ki kis kijelzős gombbal vezérelt telefonokhoz, mára tetszőleges méretű kijelzőt támogat, sőt az érintőképernyő is felkerült a támogatott eszközök listájába. Jelenleg ez a legelterjedtebb Symbian alapú rendszer.

– **UIQ:** eredetileg érintőképernyős PDA készülékekhez, mára megjelentek már érintőképernyő nélküli UIQ alapú készülékek is.

A két rendszer ugyan nagyon hasonlít egymásra, mégsem kompatibilisek, mert külön osztálykönyvtárak és SDK tartoznak hozzájuk. Symbian OS-re lehet fejleszteni Java ME-ben, Pythonban is, azonban legnagyobb szabadságot a C++ biztosítja a natív támogatás miatt.

2.2.2. Windows Mobile

A Windows Mobile a CE operációs rendszeren alapul. A PC-s változattól teljesen eltérő kernelre épül, kihasználva a mobil készülékek sajátosságait és képességeit. Jelenleg a 6.1-es, ill. 6.5-ös a legfrissebb változata.

Programozása több nyelven is lehetséges, így alkalmazható a Java ME, Python technológia, készíthetők rá natív C/C++ programok is az Embedded Visual C++ környezetben. Legjelentősebb viszont a .NET Compact Framework (.NET CF) technológia, amely a Microsoft .NET keretrendszerének mobil eszközökre fejlesztette változata.

A .NET több nyelvet is támogat, legjelentősebb a C#. Valamennyi nyelven írt programkód ugyanarra az IL nyelvre fordítódik le, amiből közvetlenül az első futáskor készül el a bináris alkalmazás. Ez a JIT-technológia.

Ez a technológia a PDA-k között piacvezető jelenleg, ugyanakkor kevés mobiltelefon használja a Windows Mobile platformot egyelőre.

2.2.3. iPhone OS

Az iPhone OS az Apple mobileszközökre fejlesztett operációs rendszere, amely UNIX alapokra épül, de nem kompatibilis az asztali gépekre szánt termékével. Jelenleg az iPhone, mely mobiltelefonként is funkcionál, és a telefonként nem működő multimédiás eszköz, az iPod épül rá.

A platform programozási nyelve az Object-C, amely a C nyelven alapszik, objektumorientált, de nem kompatibilis a C++-szal. A lefordított kód az eszközökön natívan fut. A készülék nem támogat más technológiát.

2.2.4. Palm OS

A Palm OS a PDA-kra készült mobil operációs rendszer. Ezen platform készülékei fejlett grafikus szolgáltatásokat biztosítanak (pl.: kézírás felismerése), támogatják az TCP/IP protokollt a kommunikációban.

A fő programozási nyelve a Java, illetve natívan képes C/C++ kódot is futtatni. Újabban a nyílt forrású alapokra helyezett változatának (Linux) megjelenését is tervezik.

2.2.5. Android

Az Android egy Linuxra épülő mobilplatform, amelyet az OHA fejleszt. Ennek a konzorciumnak az egyik tagja a Google is. Természetesen nagyfokú nyíltság jellemzi, nagy része nyílt forrású, fontos építőköve a webes technológia, főként a Google termékei, pl.: Gmail, Google Maps. A készülékek teljes értékű billentyűzetet és nagyméretű kijelzőt kaptak.

A platform fő programozási nyelve a Java, azonban nem a Sun, hanem egy saját fejlesztésű virtuális gép futtatja a bájtkódot. Az SDK 1.6 változata lehetővé teszi a natív C/C++ programok írását is Android telefonra.

2.2.6. Maemo

A Maemo egy Linux operációs rendszeren alapuló mobilplatform, melyet elsősorban ún. internettáblák, illetve PDA-k számára kezdett el fejleszteni a Nokia. Felkészítették a platformot a Bluetooth, WLAN, GPS funkciók ellátására, illetve a nagyméretű, magas felbontású érintőképernyők kezelésére is.

A rendszer a Debian Linux-disztribúción alapszik, GNOME felhasználói felülettel. Erre a platformra Python, Java, illetve natívan futó C/C++ programnyelven lehet alkalmazásokat fejleszteni a Scretchbox nevű szoftvercsomag segítségével, ami többféle Linux-alapú platformot is támogat.

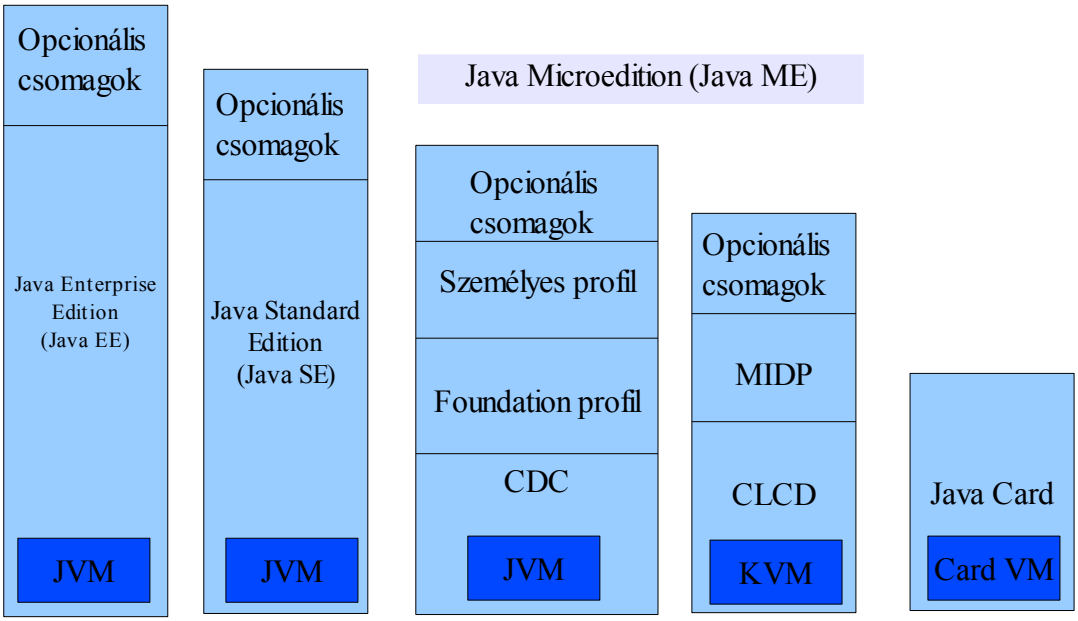
A beágyazott rendszerek és a PC-k, amin a programok készülnek, egyaránt Linux alapúak, így a használt keretrendszerek jórészt azonosak, így pontos emulálást tesz lehetővé a fejlesztés során. Mindezek miatt ígéretes platform válhat a Maemoból a jövőben annak ellenére, hogy ma még kevés eszköz készül ezen platform támogatásával.

3. A Java technológia bemutatása

Amint az előző fejezetből kiderült, a legtöbb készülék támogatja a Java technológiát, így a programfejlesztés ezen a platformon fog zajlani. Ebben a fejezetben bemutatom, hogy mik a fontosabb jellemzői a Java-nak.

A Java-t a Sun Microsystems nevű cég fejleszti a '90-es évek elejétől kezdve. Maga a Java egy teljes rendszert jelent, ami magába foglal egy programozási nyelvet és egy köré épülő környezetet, ami tartalmazza a fejlesztői környezetet, fordítót és a futtatókörnyezetet is, a Java Virtuális Gépet is (JVM). Nagy előnye, hogy platformfüggetlenségre törekszik, azaz a fontos szempont a fejlesztésekor a hordozhatóság. 2006-ban vált nyílt forrásúvá. Ez a döntés is hozzájárult ahhoz, hogy mára az egyik legelterjedtebb technológiává vált, így nagyon fontos szerepet tölt be úgy a multimédiás mint a szerveroldali alkalmazások fejlesztése során.

Annak megfelelően, hogy milyen típusú készülékre íródik egy alkalmazás, különböző nyelvi eszközökből válogathat a programozó. Hiszen különböző funkciókat lát el egy PC-re írt alkalmazás, egy mobil eszközön futó program vagy egy szerverre készült skálázható alkalmazás. A Java technológia mindehhez kínál egységes környezetet (1. ábra). Alapvetően háromféle irányban fejlődött a technológia figyelembe véve a hardvereszközök lehetőségeit.



1. ábra: A Java-platformok

3.1. Java SE

A Java Standard Edition (Java SE) a munkaállomásokra, az asztali számítógépekre készült alkalmazásokhoz kiadott változat. Ez a Java alap kiadása, mely lehetővé teszi a programozó számára többek között a fájlkezelés, a hálózat, a grafikus felület használatát annak érdekében, hogy minél jobb, funkciógazdagabb, felhasználóbarátabb programok készülhessenek a munkaállomásokra. Jelenleg a Java SE legfrissebb stabil változata az 1.6-os, az 1.7-es még fejlesztés alatt van.

3.2. Java EE

A Java Enterprise Edition (Java EE) a szerveralkalmazásokhoz szánt változat. A Java EE jóval gazdagabb, mint az SE, hiszen tartalmazza a teljes SE programkönyvtár-készletét, és olyan bővítéseket is, amelyek az alkalmazásszerverek számára készült moduláris szoftverkomponensek segítségével skálázható, hibatűrő, többrétegű, elosztott rendszerek létrehozását teszik lehetővé. Tartalmazza többek között az adatbázis-kezeléshez, a webszolgáltatásokhoz, az üzenetszolgáltatásokhoz, konkurenciakezeléshez szükséges elemeket. Nagyon fontos szerepet tölt be a Java EE a webalkalmazások, illetve nagyvállalati erőforrásigényes elosztott rendszerek területén. Jelenleg a legfrissebb változata a Java EE 5-ös, a 6-os fejlesztés alatt van.

3.3. Java ME

A Java Micro Edition (Java ME) a kifejezetten mobil eszközök számára kifejlesztett változat. Ezzel a változattal a Java technológia elérhetővé válik mobil eszközökre is, amelyek jelentősen korlátozott hardveres lehetőséggel rendelkeznek akár csak egy alacsonyabb kategóriás PC-hez viszonyítva is. Többek között lassabb processzor, kevesebb memória, kisebb méretű és felbontású képernyő (kijelző) jellemzi ezeket a készülékeket. Ezért csökkentették az API által biztosított funkciókat. Annak érdekében, hogy az eszközök minél szélesebb körében használható legyen a technológia, a ME-architektúrában különféle konfigurációkat, profilokat és opcionális csomagokat (JSR) definiáltak, melyekkel az adott

készülékek lehetőségei jobban kihasználhatók.

A Java ME-konfiguráció a virtuális gép és az alkalmazható könyvtárak összességét jelenti. A konfiguráció összeállításakor a készülékek processzor-, memóriakapacitására, illetve az I/O műveletek kezelésére optimalizálnak. Azonban a készülékek kisebb-nagyobb mértékben eltérnek egymástól, így nem biztosított a teljes mértékű kompatibilitás. A platform virtuális gépe (KVM) eltérő az egyéb platformokon használttól, alkalmazkodva a mobileszközök speciális tulajdonságaihoz. Jelenleg két fő konfiguráció terjedt el:

- **CDC:** összeállításakor a fő szempont a minél nagyobb kompatibilitás volt a Java SE változattal. Természetesen a hardver adottságait figyelembe kellett venni, így a CDC a nagyobb kapacitású mobileszközökön használható konfiguráció. Támogatja a teljes JVM-et, beleértve a lebegőpontost számítást, a beépített könyvtárakat, a szálkezelést és biztonsági szolgáltatásokat.
- **CLDC:** a mobilkészülékek szigorú korlátaihoz igazították. Mind a virtuális gép, mind pedig az osztálykönyvtárak terén megszorításokat kellett életbe léptetni. A CLDC 1.0-ás változatában nem támogatott a lebegőpontos számítás, korlátozott a szálkezelés, a Java SE osztálykönyvtáraiból is sokmindent kihagytak a fejlesztők, így ez a konfiguráció jelentősen különbözik az SE változattól. A CLDC 1.1 változata már több lehetőséget biztosít (pl.: lebegőpontos számítás).

A CLDC az alap objektumokat tartalmazza, erre épül a MIDP, mely definiálja a speciálisabb objektumokat. A CLDC és a MIDP együtt alkotja a Java mobileszközökre készített futtatókörnyezetét. Ez a platform már alkalmas magas szintű grafikával rendelkező, akár hálózati funkciókat használó alkalmazások futtatására is mobilkörnyezetben.

Minderre épül a JSR, mely a készülékekben található olyan egyéb funkciók elérését teszi lehetővé, mint pl. a Bluetooth, fájlkezelés, webszolgáltatások elérése, multimédia-támogatás. Ezen opcionális csomagok segítségével még jobban ki lehet használni a mobileszközök képességeit.

4. A Java programnyelv ismertetése

A Java programnyelvet a '90-es évek elején kezdték el fejleszteni a Sun Microsystems berkein belül, és 1995-ben adták ki az első változatát. Ma az 1.7 változatánál jár a fejlesztés, melyet hamarosan kiadnak. A nyelvet a C++ programnyelv utódjának szánták, annak továbbfejlesztéseként képzelték el. Ennek megfelelően objektumorientált nyelv, bár tartalmaz eljárásorientált elemeket is, és szintaxisa nagyban hasonlít elődjéhez. Az elosztott rendszerek programnyelvének szánták, és a platformfüggetlenséget is fontos megvalósítandó célul tűzték ki a fejlesztői csakúgy, mint a hálózati támogatást és a biztonságot.

4.1. A nyelv fontosabb tulajdonságai

A nyelv legfontosabb jellemzője, hogy az OO-paradigma mentén alakult ki, illetve szem előtt tartották a fejlesztői a forráskód hordozhatóságát (platformfüggetlenség).

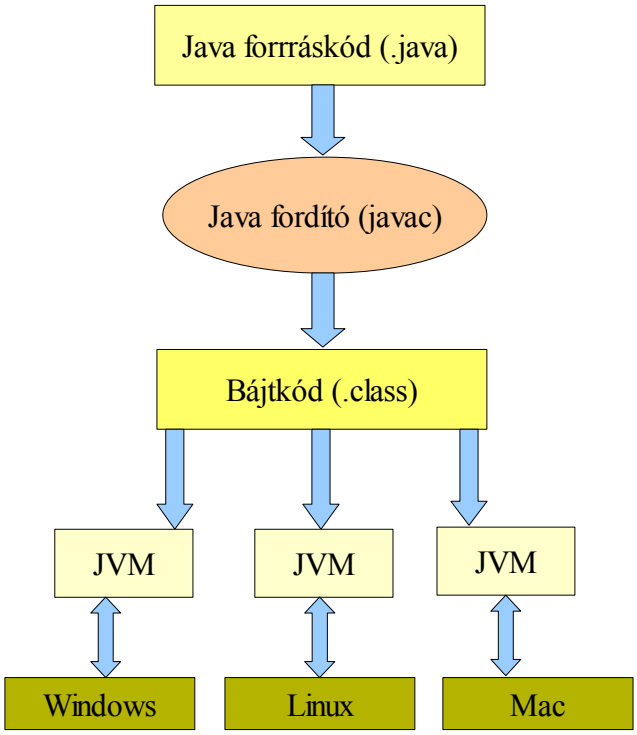
4.1.1. Objektumorientáltság

Ezen tulajdonság a programozási struktúrára és a nyelv szerkezetére is utal. A OO megközelítés azt jelenti, hogy a programot objektumok (adatokat tartalmazó entitások) alapján csoportosítja, nem pedig az elvégzett feladatokra helyezi a hangsúlyt. Ez a megközelítés jobban elősegíti a komplex szoftverrendszerek összehangolt fejlesztését.

A programozás során a programozó osztályokat épít az azonos tulajdonságokkal rendelkező objektumokból arra törekedve, hogy minél jobban modellezze a megvalósítandó feladat világát. Maga az objektum – mely az objektumorientált világ alapeleme – tehát egy adott osztály eleme, példánya. Az osztályok létrehozásakor felhasználhatóak a már meglévő osztályok is, vagyis az osztályok újrafelhasználhatók. Ezekből egy hierarchiát létrehozva megkapjuk az API-t, mely a Java esetén egy fa szerkezetet alkot, melynek a gyökerében a Java áll.

4.1.2. Hordozhatóság

A hordozhatóság azt jelenti, hogy az egyszer megírt kódot más platformon is futtatni lehet. Ezt a Java nyelv többé-kevésbé teljesíteni tudja egy teljesen újszerű megközelítéssel, amely eltérővé teszi a korábbi programnyelvekkel összehasonlítva. A forráskódot nem gépi nyelvre fordítja a javac (Java fordító), hanem ún. bájtkódra, amit a JVM futtat. A forráskódot egyszer kell fordítani, az értelmezés pedig minden futtatáskor megtörténik. A 2. ábrán látható a folyamat, ahogyan a forrásszöveg futtatható kóddá válik. A platformfüggetlenség jegyében alkotott JVM biztosítja a Java programok hordozhatóságát, illetve magasabb absztrakciós szintet valósít meg, mint a korábbi nyelvek.



2. ábra: A forráskódtól a futtatásig

4.2. A nyelv szerkezete

A Java programnyelv sajátosságait érdemes áttekinteni az objektumorientált szemlélet részletes bemutatása érdekében. A Java is tartalmaz nem az OO-paradigma mentén

megalkotott elemeket. Ilyen fogalmak többek között: adattípus, vezérlési szerkezetek, utasítás, blokk, azonosító, tömb, deklaráció, megjegyzés stb.

Azonban a Java nyelv alapvetően objektumorientált nyelv. Így eszközkészlete bővelkedik az ezen szemléletből fakadóan kialakult elemekben. Ebben a fejezetben az eszközkészlet általam használt elemeit mutatom be.

4.2.1. Objektum

Az objektum az OO-paradigma alapja, az eljárásorientált változó fogalmának kiterjesztése, általánosítása. Segítségével a valódi objektumok modellezhetők. Olyan konkrét programozási eszköz, amelynek van állapota és viselkedése. Az állapotát a változói (példányváltozók), míg a viselkedését a metódusai (példánymetódusok) írják le. Egy objektum állapotát magába az objektumba rejti a programozó, az csak metódusok segítségével változtatható. Ez az egységbezárás.

A Java-ban a metódusok is lehetnek kívülről nem láthatók, illetve a változók is lehetnek kívülről láthatók. A hozzáférési szintek segítségével tudja az objektum szabályozni a kívülről láthatóságot, azaz, hogy egy másik objektum hozzá tud-e férni a változóihoz. A Java programban az objektumok hatnak egymásra, aminek következtében megváltozhat azok állapota. Egy program nem más, mint egymással kölcsönhatásban lévő objektumok összessége.

Az egységbezárásnak két nagy előnye van:

- **modularitás:** az objektumok forráskódja teljesen el tud különülni egymástól;
- **információrejtés:** az objektumok csak a publikus csatornákon keresztül tudnak kommunikálni egymással (üzenetek segítségével).

4.2.2. Osztály

Az osztály azonos tulajdonságokkal és a metódusokkal rendelkező objektumok modellje, leírja az adott osztályba tartozó objektumok közös változóit és metódusait. Az osztályhoz köthetők az objektumok, mert az objektumok az osztályokból származnak, azaz minden objektum tartozik valamilyen osztályba. Az osztálynak lehetnek osztályváltozói, melyek valamennyi objektum számára azonosak. Ha ezeket egy objektum megváltoztatja,

valamennyi objektum számára megváltozik. Az osztálynak lehetnek osztálymetódusai is, melyek nem az objektumok viselkedését határozzák meg, hanem magához az osztályhoz tartoznak. (A tulajdonságokat adattagoknak, a metódusokat tagfüggvényeknek is nevezhetjük.)

Az osztály a következőképpen hozható létre:

```
[módosító] class Ujosztalyneve (extends szuperosztály_név)
           {
                Tulajdonsag1;
                Tulajdonsag2;
                … stb.
                Viselkedes1(){
                …
                }
                Viselkedes2(){
                …
                }
                … stb.
           }
```

A fej részben a `class` kulcsszó után le kell írni a nevét, majd a törzsben megadni a tagdefiníciókat. Először az adattagokat, majd a tagfüggvényeket.

```
public class HajosVaszon extends GameCanvas implements Runnable{
  a HajosVaszon osztály törzse
  (adattagok, metódusok)
}
```

Az osztály minden adattagjának külön lehet szabályozni a hozzáférhetőségét. Négy hozzáférési kategória van:

- a jelöletlen tagokra csak az azonos csomagban definiált osztály hivatkozhat (félnyilvános tagok);
- a `public` módosítóval ellátott tagokra minden osztály hivatkozhat (nyilvános tagok);

– a `private` módosítóval definiált tagok csak az adott osztályon belülről érhetők el;
– a `protected` kategória a félnyilvános kategória kiterjesztése.

Az osztály alkalmazásának az újrafelhasználhatóság a nagy előnye. Az objektumok az osztályból példányosítással állíthatók elő, melyet a következőképpen lehet megtenni:

```
HajosVaszon grafikusKepernyo = new HajosVaszon();
```

A példányosítás után már az objektumról mint az adott osztály példányáról beszélünk. Maga a példányosítás tetszőleges alkalommal elvégezhető, és minden művelet eredménye egy új objektum.

4.2.3. Metódus

Amint az már az eddigiekből kiderült, a metódusok az eljárásorientált nyelvekben szereplő eljárásoknak, ill. függvényeknek felelnek meg, azok kiterjesztése az OO-világban. Egy osztály létrehozásakor definiálhatunk példány-, illetve osztálymetódusokat. A Java nyelvben léteznek bizonyos szempontból kiemelt metódusok vagy metódus-szerű programkódok:

Főprogram

Minden Java program tartalmaz egy ún. főprogramot, a `main`-t, melynek alakja a következő:

```
public static void main(String[] args){
        törzs
}
```

A JVM az inicializálások után a főprogramnak adja át a vezérlést, ami végrehajtja az utasításait, átadja a vezérlést a megfelelő programrészeknek. A program akkor ér véget, ha a `main()` befejeződik. (Más módon is kérhetjük a program befejezését, pl.: a `System` osztály `exit` metódusával.)

Konstruktor

Egy objektum létrehozásakor (példányosítás) be kell állítani annak kezdőértékeit, azaz használatra alkalmas állapotba kell hozni. A kezdeti beállítások elvégzésére valók a konstruktorok, melyek nagyon hasonlítanak a metódusokra, azonban mégsem tekintjük őket tagoknak, mert nem öröklődnek. Kizárólag a `new` operátorral hívható meg, a láthatóságának szabályozására van lehetőség.

```
public class HajosVaszon extends GameCanvas implements Runnable {
        ...
        public HajosVaszon() {
              ...
              palya = new int [][]{
                    ...
              }
              ...
        }
        ...
}
```

A konstruktor nevének meg kell egyeznie az osztálya nevével, visszatérési értéke nincs. Egy osztálynak mindig van legalább egy konstruktora (több is lehet, akár ugyanazzal a névvel is, ha más-más paraméterlistával definiáljuk őket). Ha a programozó nem ír konstruktort, akkor a rendszer készít egyet, melynek a törzse üres.

Destruktor jellegű metódus

A Java nyelvben nem kell foglalkoznia a programozónak az objektum megszüntetésével, mert a beépített szemétgyűjtögető (Garbage collector) megteszi ezt. A `finalize()` metódus segítségével értesülhetünk egy objektum megszűnéséről, mely bizonyos esetekben fontos lehet a programozó számára. Ezt a metódust minden osztály örökölheti és átdefiniálhatja. Közvetlenül az adott objektum tényleges felszabadítása előtt automatikusan fut le. (Az objektum „utolsó kívánsága”.)

4.2.4. Öröklődés

Az öröklődés az egyik legfontosabb tulajdonsága az OO-paradigmának. Osztályokhoz kötődő fogalom, legalább két osztály között fennálló aszimmetrikus kapcsolatot fejez ki. A szülőosztályhoz kapcsolódóan lehet létrehozni származtatott osztályokat, melynek során a leszármazott osztályok számára elérhetővé válnak a szülőosztály attribútumai és metódusai, öröklik azokat. A láthatóságot itt is lehet szabályozni a már megismert módon.

```
class Alosztaly extends SzuloOsztaly{
     törzs
}
```

Az öröklés azonnal elérhetővé teszi az `Alosztaly` számára a `SzuloOsztaly` valamennyi tulajdonságát, metódusát. Azonban a származtatott osztálynak mindezeken felül lehetősége van: új attribútumok, illetve metódusok bevezetésére, illetve törlésére, metódusok újraimplementálására, duplikálására, a láthatósági szabályok újraértelmezésére, attribútumok átnevezésére, duplikálására.

Egy szülőosztályból tetszőleges számú alosztály származtatható, viszont minden származtatott osztálynak csak egy szülőosztálya lehet (egyszeres öröklődés). Az öröklődés egy fa szerkezetű osztályhierarchiához vezet a Java nyelvben, aminek a gyökerében az `Object` osztály áll, ez minden osztály szülője.

```
public class Hajozas extends MIDlet implements
      CommandListener{
      ...
}
```

Az öröklődéshez kapcsolódó fogalom a polimorfizmus. Mivel az örökölt metódusokat a leszármazott osztályok újraimplementálhatják, azok polimorfak (többalakúak) lehetnek. Az újraimplementált metódusoknak a szülőosztály metódusához kompatibilisnek kell lenniük, aminek érdekében néhány megkötésnek eleget kell tenni: specifikációjuknak meg kell egyezniük, ugyanazokat a kivételeket kell kiváltaniuk, a láthatóságot csak enyhíteni lehet.

4.2.5. Interfész

Az interfésszel egy új absztrakciós szint bevezetése válik lehetővé. Az interfészek nem részei az osztályoknak, új referenciatípusként jelennek meg a Java nyelvben. Absztrakt metódusok deklarációjának és konstans értékeinek együttese. Ez azt jelenti, hogy a program bizonyos részein nem kell megadni az implementációt, elég csak a specifikációval foglalkozni.

```
public class HajosVaszon extends GameCanvas implements
      Runnable{
      ...}
```

Az interfész használata akkor lehetséges, ha azt egy osztály teljes mértékben implementálja, azaz minden egyes, az adott interfészben specifikált metódust implementál.

Az interfészek között is értelmezhető az öröklődés, azonban itt többszörös öröklődés is lehetséges, tehát egy interfésznek lehet több szülő interfésze, viszont nem létezik közös ősinterfész. Ezzel a módszerrel a Java nyelvben is megvalósul a többszörös öröklődés lehetősége, a specifikáció többszörösen örökölhető, azonban maga a kód csak egyszer öröklődik a szülőosztályon keresztül.

4.2.6. Csomag

A csomag egy programegység, hatásköri egység, melynek segítségével a Java-program forráskódja tagolhatóvá válik. A csomag szorosan összetartozó részeket tartalmaz, használatával a kód áttekinthető és könnyebben módosítható lesz. Csomagok tartalmazzák a Java fejlesztői környezetét és magát a program forráskódját is.

A csomagok egy hierarchikus szerkezetet alkothatnak, egy csomagnak tetszőleges számú alcsomagja lehet. A Java nyelvben fordítási egység az osztály, az interfész, tehát a típusok deklarációi. A fordítási egység mögötti kódnak kötelezően csomagban kell megjelennie. Minden fordítási egység elején lehet deklarálni a `package` kulcsszó segítségével, hogy az melyik csomaghoz tartozik:

```
package hajos_jatek;
```

A csomag hozzáférési szempontból is tagolhatja a kódot, a félnyilvános (módosítóval el nem látott – friendly) tagok csak az adott csomagból látszanak. Amennyiben másik csomagban lévő osztályok használatára van szükség, importálni kell az azokat tartalmazó csomagot, melyet a következőképpen lehet megtenni:

```
import javax.microedition.midlet.*;
```

Ezzel az utasítással az egész csomagot importáltuk, osztályokat is lehet külön, ekkor nincs szükség a '*' karakterre, hanem teljesen ki kell írni az osztályhivatkozást.

4.2.7. Kivételkezelés

Egy program futása során felléphetnek olyan esetek, amelyek hibás működést eredményeznek, ezért a hibákat valahogyan kezelni kell. Egy hiba bekövetkezése esetén olyan állapotba kell hozni a programot, hogy az folytatni tudja működését, vagy annak szabályos befejezését kell megvalósítani. Ez elég nehéz feladat, hiszen minden esetet figyelembe kell venni, ha megbízható kód írása a cél.

A Java a hibák kezeléséhez, mely a kivételkezelésen alapszik, sok segítséget ad a programozónak. A kivételkezelés az osztályokra épül. Amikor valamilyen hiba lép fel a programfutás során, létrejön egy kivételobjektum, mely leírja a kivétel fajtáját, illetve az aktuális állapotot. Ezután a JVM felügyelete alá kerül ezen objektum, tehát az a metódus, amelyikben bekövetkezik a hiba, kiváltja a kivételt.

A kivételek lehetnek nem ellenőrzöttek, amelyek bárhol bekövetkezhetnek. Ezeket nagyon nehézkes és hosszadalmas lenne kezelni, ezért a programozónak nem kell vele foglalkoznia. A másik kivételcsoportba az ellenőrzött kivételek tartoznak, melyeket kezelni vagy specifikálni kell. Minden egyes ellenőrzött kivételt fel kell sorolni, amely kiváltódhat az adott metódus láthatósági körében. A kivételkezelés megvalósítására a `try-catch-throw` utasítások használhatók.

```
try {
     mihajonk = createImage("/mienk.png");
     ellenhajo = createImage("/ellenseg.png");
```

```
            csempek = createImage("/hatter.png");}
        catch(java.io.IOException e) {
            System.out.print("Nem talalom a kepet.");
        }
```

Látható a példában, hogy most elkapjuk a kivételt, ami akkor következik be, ha nem sikerül betöltenie a megfelelő képeket a programnak (ez látható a `try` blokkban). Ekkor az éppen aktuális metódus átadja a vezérlést a futtatórendszernek, amely megnézi, hogy hol kezeli a program a kivételt.

A `catch` blokkban látható a paraméterben lévő típusú kivétel kezelése (most kiírjuk az üzenetet). Amennyiben nem tudtuk vagy nem akartuk volna kezelni a kivételt, akkor továbbadhattuk volna a `throw` utasítás segítségével a specifikációjának megadása után.

4.2.8. Szálak

A Java a szálakat (Thread) biztosítja a programozó számára a párhuzamos programozás támogatására. A szálak mint objektumok a `Thread` osztályból származnak, és a `run()` metódussal lehet őket futtatni. A `Runnable` interfészt implementáló osztály segítségével is megadhatunk szálakat, ekkor azonban az osztálynak kell a `run()` metódust implementálnia.

```
public class HajosVaszon extends GameCanvas implements
      Runnable {
                ...run()
 }
```

A szálak állapota, mely a program futása során változik, a következő lehet:

– kész: futtatható, éppen áll;

– futó: a CPU éppen ezt a szálat futtatja;

– blokkolt: várakozik pl. egy külső esemény bekövetkezésére;

– megszakított: megállított folyamat;

– halott: befejeződött folyamat.

Új szál létrehozása a `new` operátor segítségével történik. Ekkor még csak él, nem fut. A `start()` metódus meghívásával futtatható, míg a `stop()` leállítja a szálat. Egy szál halott lesz leállítás után, illetve, ha a `run()` metódusának a kódja elfogy. Halott szálat nem lehet újraindítani.

A várakozás a szinkronizáció eszköze. Várakozás a `sleep(int n)` és a `wait()` metódusokkal érhető el, a szálak kész állapotba az `n` idő letelte után, illetve a `notify()` metódussal kerülnek. A `suspend()` megszakított állapotba helyezi a szálat, a `resume()` metódus pedig kész állapotba helyezi.

5. fejlesztőkörnyezet

Egy programfejlesztési munka során a megvalósítandó feladatok szerteágazók lehetnek. A kódolás hatékonyságának növelése érdekében célszerű egy komplett fejlesztőrendszert, fejlesztői környezetet használni. Ilyen célt szolgáló eszközökből természetesen sok létezik, a programozó ki tudja választani a számára legmegfelelőbbet, ami legjobban tudja segíteni a munkát. Ebben a fejezetben az általam a játékprogram elkészítéséhez használt eszközrendszert fogom bemutatni.

Egy mobiltelefonos játék elkészítése során több feladatot is meg kell oldani:

- **kódtervezés**, melynek során a program elve tisztázódik, majd az rendszerszerűen ábrázolható, illetve a dokumentáció elkészítésében is segítségünkre lehet;
- **kódolás**, ami maga a program írása. Általában ez a rész jelenti a munka legnagyobb részét, bár egy előzetesen jól megtervezett program gyorsan kódolható;
- **képszerkesztés**, ami egy játékprogram készítésekor elengedhetetlen, hiszen gondoljunk csak a figurák és a háttér megrajzolására.

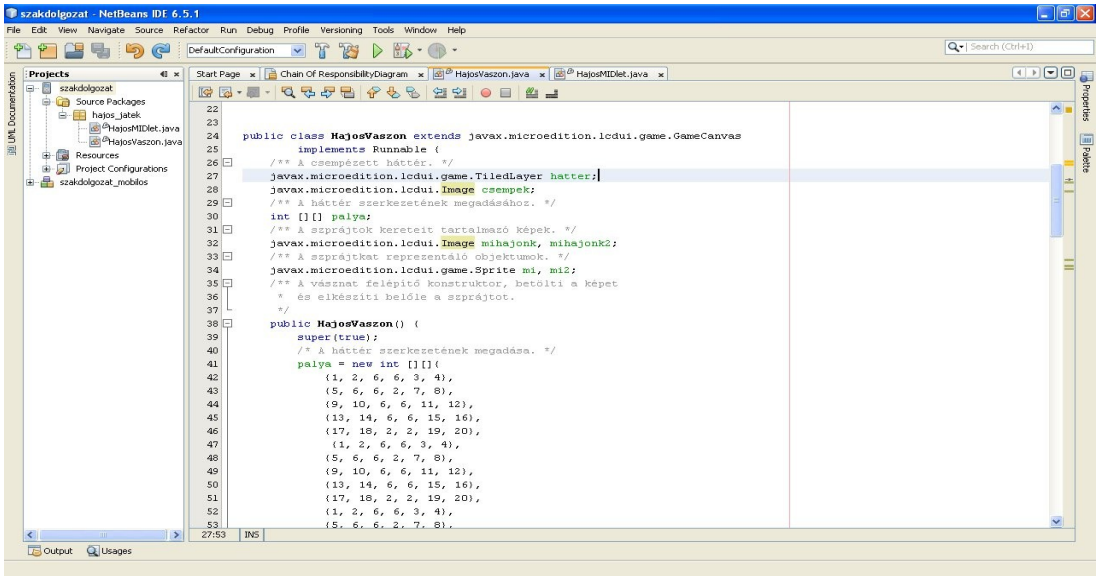
Kódtervezésre és kódírásra a NetBeans, míg rajzolásra, képszerkesztésre a GIMP nevű eszközt használtam. Közös jellemzőjük, hogy ingyenesen használhatók, és elérhetők több platformra is, így Windows és Linux operációs rendszeren is használhatók.

5.1. NetBeans

A NetBeans egy integrált fejlesztői környezet. Nagyon fontos tulajdonsága, hogy szabad, nyílt forráskódú szoftver. Nagyon sok programnyelven történő fejlesztést támogat. Többek között C/C++, PHP, Ruby, Perl, Python nyelven, még az egyik legújabb platformra, az Androidra és természetesen a Java platformjaira is készíthetünk a NetBeans segítségével alkalmazásokat. Ez utóbbi nyelven mobilalkalmazás fejlesztésére is van lehetőség.

A NetBeans mint IDE, természetesen sok segítséget nyújt a kódíráshoz. Íme néhány, melyet én fontosnak tartok, és amik megkönnyítették a munkámat:

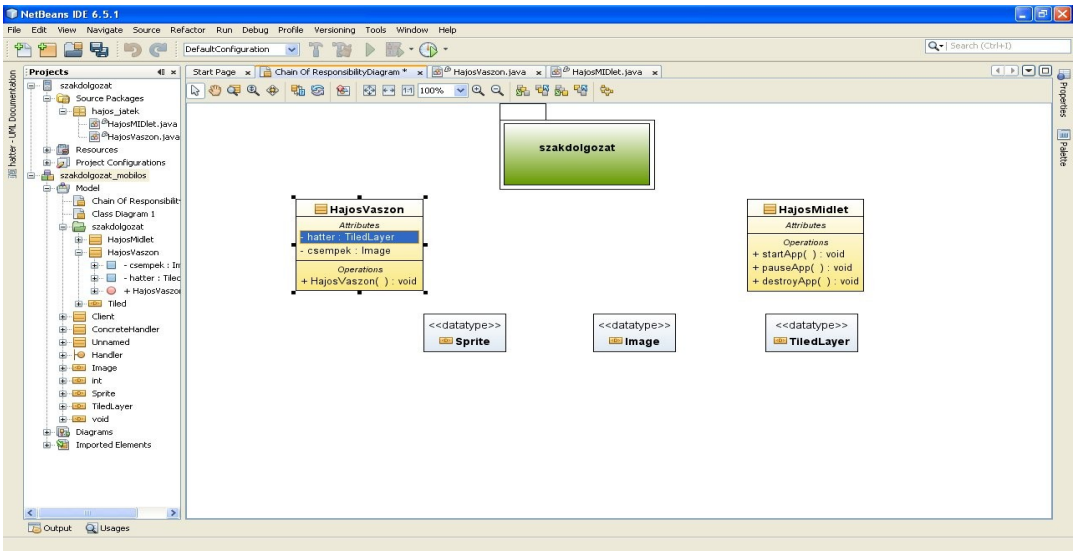
- szintaxiskiemelés (színekkel elkülöníti a rendszer a különböző programelemeket);
- kódkiegészítés (megfelelő osztályelemek, metódusok, változók felajánlása);
- hibafelismerés (szintaktikai hibák jelzése már gépelés közben);
- automatikus kódformázás (az áttekinthető forrásértelmezéshez);
- projektkezelés (egyszerűen áttekinthető, módosítható);
- programtesztelés (emulátor segítségével tesztelhető a program, így nem kell mindig telefonra tölteni a programot a fejlesztés különböző fázisaiban);
- dokumentációkészítés (egyszerű dokumentációgenerálás az elkészült programhoz);
- stb.



3. ábra: NetBeans IDE 6.5

A NetBeans IDE használatához le kell tölteni a honlapjáról, majd telepítés után egyszerűen használatba vehető az eszköz. Ahhoz, hogy a Java ME platformra is tudjunk alkalmazásokat készíteni, szükségünk van még a NetBeans mobiltelefonos kiegészítésére (Mobility Pack), illetve a JDK-ra is. Utóbbi a Sun Microsystems honlapján található meg. (Egyszerűbb megoldásként a NetBeans honlapjáról a komplett csomag egyben is beszerezhető.) Az elkészült program futtatásához a JRE szükséges (a JDK tartalmazza).

A telepített alkalmazáshoz számos egyéb eszközt, kiegészítést tölthetünk le, a program beépített eszközrendszerével. Ilyen fontos modul pl. az UML kiegészítés, mely segítségével a megszokott környezetben használhatjuk az UML nyelvet a program megtervezéséhez, a problémakörök kialakításához, részletezéséhez és megoldásához. A könnyebb áttekinthetőség kedvéért diagramszerkesztőt is használhatunk a munkához.



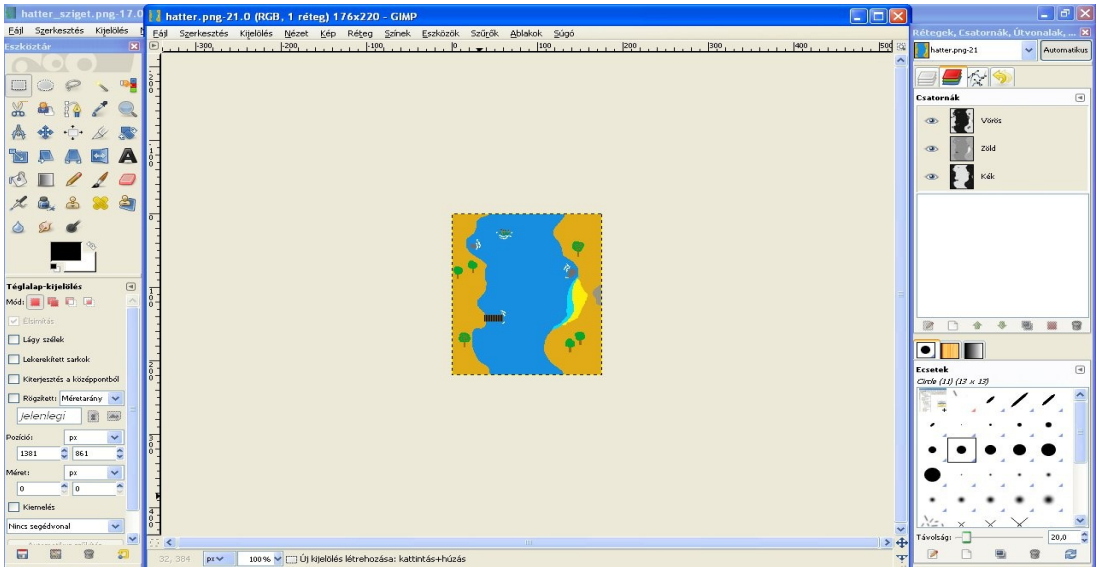
4. ábra: Diagramtervezés a NetBeans segítségével

Láthatjuk, hogy a NetBeans valóban igen sokoldalú eszköz, alkalmas akár nagyobb projektek kivitelezésre is.

5.2. GIMP

Játékprogramok fejlesztése során általában nagy hangsúly kell fektetnie a programozónak az alkalmazás grafikai megjelenésére, mert a játékok figurákból, hátterekből,

tereptárgyakból stb. állnak, amelyek szorosan együttműködnek az egyéb elemekkel a játékélmény fokozása érdekében. A grafikai összetevők megszerkesztésére szintén sok programot találunk. Én a GIMP nevű képszerkesztőt választottam annak sokszínűsége, és szemlélete miatt. A GIMP ugyanis a NetBeanshez hasonlóan nyílt forrású és ingyenesen használható.



5. ábra: A GIMP 2.6.7

A program alkalmas háttérképek, képernyőképek, fényképek szerkesztésére, összetett grafikai műveleteket is végezhetünk vele. Továbbá teljesen új képeket is szerkeszthetünk a segítségével, ahogyan az a játékprogram grafikai elemei esetén látható.

6. A Java ME platform szoftvertervezési megfontolásai

Amint azt már tudjuk, a Java ME a mobileszközök platformja. Sajnos a mobilprogramozó hamar megtapasztalja, hogy igen komoly korlátokba ütközik a készülékek hardverkapacitását illetően. Ezek a megszorítások főként a processzor, a használható memória, a háttértár, illetve a kijelző szerényebb tulajdonságaiban nyilvánul meg. Mindezek miatt, mielőtt belevágunk a fejlesztésbe, érdemes előre kiválasztani egy konkrét készüléket, majd tájékozódnunk a lehetőségeinkről, információt szerezni tulajdonságairól.

A játékprogramot a saját mobiltelefonomra kezdtem el fejleszteni, mely egy Nokia 6300 típusú általános célú készülék. A cég honlapján könnyen tudunk információt szerezni erről a típusról, és egy kis kutatás után kiderül, hogy a sok-sok mobiltelefon típus jellemzői között sok hasonlóságot fedezhetünk fel, legalábbis, ami a készülékekre történő Java alapú fejlesztés szempontjából meghatározó. A tulajdonságokat érdemes két fő csoportra bontani. Az egyik csoportba a készülék általános jellemzői tartoznak, míg a másikba konkrétan a kijelző tulajdonságai.

6.1. A konkrét készülék általános jellemzői

Nézzük a Nokia 6300 típus adatait közül azokat, amelyek fontosak lehetnek az alkalmazásfejlesztés során:

- operációs rendszer: Nokia OS;
- fejlesztői platform: Series 40 3rd Edition, Feature Pack 2;
- kijelző felbontása, színmélysége: 240*320 pixel, 24 bit;
- memória:
  - felhasználható maximális méret: 9 MB;
  - heap maximális mérete: 2 MB;
  - háttértár bővíthetősége: MicroSD, max. 2 GB;
  - JAR fájl maximális mérete: 1 MB;
- támogatott Java technológia:
  - JSR 139 Connected, Limited Device Configuration (CLDC) 1.1;
  - JSR 118 MIDP 2.0;
  - JSR 185 Java™ Technology for Wireless Industry;
  - JSR 75 FileConnection and PIM API;
  - JSR 82 Bluetooth API;
  - JSR 135 Mobile Media API;
  - JSR 172 J2ME™ Web Services Specification;
  - JSR 177 Security and Trust Services API for J2ME™;
  - JSR 184 Mobile 3D Graphics API for J2ME™;

  - JSR 205 Wireless Messaging API 2.0;
  - JSR 226 Scalable 2D Vector Graphics API;
  - Nokia UI API;
- támogatott képformátumok: BMP, GIF87a, GIF89a, JPEG, PNG, WBMP;
- támogatott audioformátumok: AAC, AAC+, AMR-NB, eAAC+, MIDI Tones (poly 64), Mobile XMF, MP3, MP4, SP-MIDI, True tones, WMA;
- támogatott videoformátumok: 3GPP formats (H.263), H.264/AVC, MPEG-4;
- adatátvitel: CSD, EGPRS, HSCSD;
- egyéb: HTML over TCP/IP, WAP 2.0, XHTML over TCP/IP, Flash Lite 2.0.

A lista természetesen nem teljes, a fejlesztési szempontból legfontosabbakat tartalmazza.

6.2. A kijelző

A telefon kijelzője az egyik legfontosabb kommunikációs felülete a készüléknek, ezért érdemes részletesebben is megvizsgálni. Gyártótól és készüléktípustól függően többféle méretű és színmélységű kijelzővel találkozhatunk. A konkrét példánál maradva a 240*320 pixel felbontású, 24 bit színmélységű (16 millió árnyalat) kijelzőt vettem alapul a játékfejlesztés közben.

Mindegyik kijelzőre igaz azonban az, hogy alapvetően három fő részre oszthatók, amit a 6. ábra jól mutat:

- **Status zone**, ahol a telefon működésével kapcsolatos információkat láthatjuk (akkumulátor töltöttsége, hálózati jelerősség), valamint egyéb szolgáltatások be-, illetve kikapcsolt állapotát jelző piktogramok jelennek meg (ébresztés, óra, bluetooth stb.);
- **Main zone**, a főképernyő, ahol telefon éppen működő funkciója jelenik meg, így az alkalmazásunk is itt látható;
- **Softkey zone**, ahol a gyorsbillentyűkhöz rendelt funkciók láthatók.

6. ábra: A kijelző felosztása

Bár a programunk a képernyő középső, legnagyobb részén jelenik meg, mégsem kell a pontos méreteit ismerni, ha teljes képernyősként készítjük el az alkalmazásunkat. Érdemes így szerkesztenünk a játékokat azért is, mert egyrészt jobban látható maga az alkalmazás, másrészt nem zavarják meg a játékélményt a telefon által megjelenített információk.

7. Egy konkrét Java ME alapú mobiltelefonos játék bemutatása

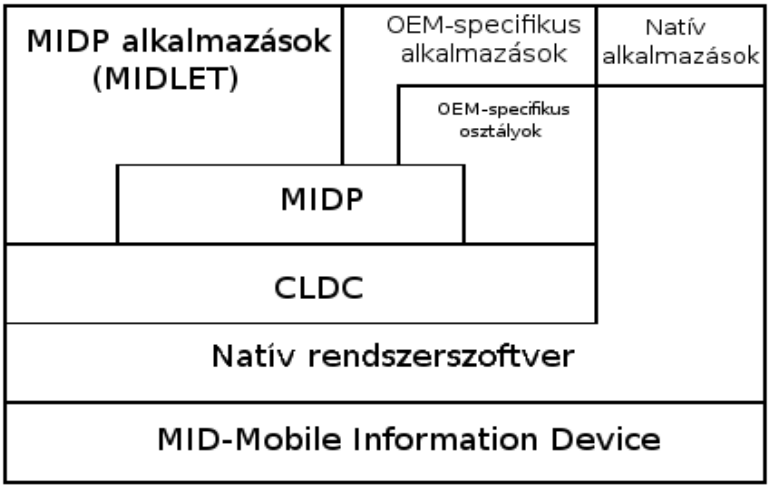
Az előző fejezetekben lépésről lépésre megismertük a Java alapú fejlesztés feltételeit, jellemzőit, a nyelv legfontosabb elemeit, melyeket a konkrét játékprogramhoz fel lehet használni. Kiválasztottam a platformot, egy konkrét mobiltelefon-típust, amelyre a fejlesztést el lehet készíteni. Eldöntöttem, hogy milyen fejlesztőeszköz-rendszert fogok használni a munka során. Ebben a fejezetben a konkrét játékprogram részletes bemutatása következik.

A játékban egy hajót vezetünk egy folyón haladva. Ellenséges hajók jönnek szemben velünk. A cél az, hogy minél tovább tudjunk manőverezni ütközés nélkül. Amennyiben mégsem sikerül elkerülni a „találkozást”, a hajónk felrobban, és a játék véget ér. A játék során számon tartjuk, hogy mennyi ellenséges hajót tudtunk kicselezni, amit folyamatosa meg is jelenítünk.

A játék a kiválasztott telefon CLDC 1.1/MIDP 2.0 támogatásán alapul.

7.1. A MIDP 2.0 jellemzői

Az általam választott telefon egyik, a Java-fejlesztés szempontjából legfontosabb tulajdonsága, hogy milyen konfigurációt támogat. A Nokia 6300 esetén a CLDC 1.1-es és a MIDP 2.0-ás változata használható fel.



7. ábra: A MIDP profil

Ahogy azt a korábbi fejezetekben láthattuk, a CLDC-ben csak alapvető szolgáltatások használatára van lehetőség. Erre épül a MIDP profil a bővített szolgáltatásaival. A profil felépítése a 7. ábrán látható. A kiválasztott telefon által támogatott MIDP 2.0 főbb tulajdonságai a következők:

- multimédiatámogatás, különös tekintettel a hanghatásokra;
- grafikai támogatás, amellyel könnyen megvalósítható a háttérképek, karakterek (Sprite) kezelése;
- adattárolás lehetősége a készüléken;
- fejlett internetelérés támogatása;
- az IO csomag bővülésével Socket, illetve Stream alapú kapcsolatok létrehozása;
- biztonságos kapcsolatok támogatása.

7.2. MIDlet

A korábbi fejezetekben láthattuk, hogy különböző Java-változatok léteznek, melyek különböző környezetben használhatók, és a fejlesztés során különböző lehetőségek állnak rendelkezésre a programozó számára. Ennek megfelelően a platformok számára készült programok elnevezése is igazodik ehhez a változatossághoz.

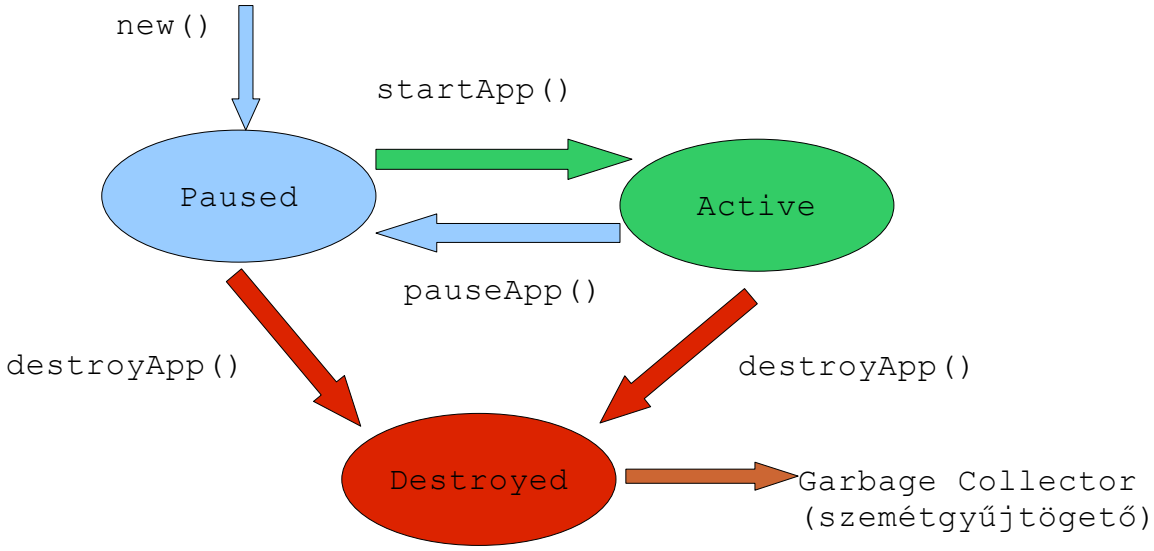
Az asztali környezetben futtatható kisalkalmazás az **Applet**. Ezeknek a programoknak egy része Java-t támogató böngészőkben is képes működni – bizonyos megszorításokkal. A szerveroldalon a futtatókörnyezet részeként futó alkalmazások neve a **Szervlet**. A MIDP környezetben működő program a **MIDlet**. Ez utóbbit nézzük meg részletesebben, mert a mobilalkalmazások tulajdonképpen a MIDletek.

Ha egy alkalmazást töltünk a telefonra, akkor az tulajdonképpen egy MIDletsuit-ból áll, ami egy vagy több MIDletet tartalmaz összecsomagolva. Azaz a MIDletsuit azonos funkciókat ellátó, illetve egymásra hatással lévő MIDletek összessége. Az azonos suit-ban lévő MIDletek osztozhatnak a közös erőforrásokon (adatok, grafikai elemek), és hozzáférhetnek egymás információihoz, míg mások nem. A `MIDlet` osztályt a `java.microedition.midlet` csomag tartalmazza.

A MIDletek saját életciklussal rendelkeznek (8. ábra), melyet egy alkalmazásmenedzser kezel (AMS). Alapvetően három állapotban lehet egy MIDlet: `Active` (aktív), `Paused` (felfüggesztett) és `Destroyed` (törölt). Az állapotváltoztatás a következő metódusok segítségével történhet: `startApp()`, `pauseApp()`, `destroyApp()`.

Alapesetben, amikor még nem töltődött be a JVM-be, `Destroyed` állapotban van a MIDlet. Példányosítás során kerül `Paused`be, amikor betöltődnek a hivatkozott osztályok, az erőforrások azonban még nincsenek lefoglalva. Az alkalmazás indításakor a `startApp()` metódus meghívásával `Active` állapotba kerül a MIDlet. Ekkor lefoglalódnak az erőforrások, felépül a grafikus felület, a kommunikációs kapcsolat, illetve az alkalmazás műkö-dik.

A program futása során szükség van a MIDlet állapotának megváltoztatására pl. ha hívás vagy üzenet érkezik a telefonra, a programot szüneteltetni kell. Ekkor meghívódik a `pauseApp()` metódus, és a program `Paused` állapotba kerül. Az újraindításhoz ismét a `startApp()` metódusra van szükség.



8. ábra: Egy MIDlet életciklusmodellje

Az alkalmazás bezárásához meg kell hívni a `destroyApp()` metódust, ezzel a MIDlet `Destroyed` állapotba kerül, a programfutás befejeződik. Ezt meg lehet tenni mind `Active`, mind pedig `Paused` állapotban. Láthatjuk, hogy a MIDlet tényleges eltüntetésével nem kell foglalkoznunk, azt a Garbage Collector automatikusan elvégzi.

7.3. A játékprogram felépítése

A fejlesztői keretrendszer szimulátor segítségével biztosítja, hogy a mindenkori készültségi állapotban lévő programunkat ne kelljen minden egyes tesztelési fázisban újra és újra a telefonra tölteni. Elegendő ebben a szimulátorban kipróbálni a változtatásokat. A tapasztalatom azt mutatta, hogy a szimulátor és a telefon viselkedése a program futtatása szempontjából megegyezett.

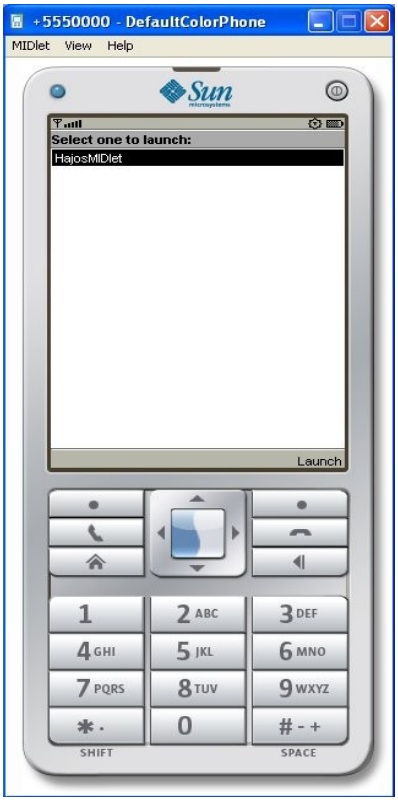
A program egy MIDlet-ből (`Hajozas`) és egy osztályból (`HajosVaszon`) áll. A MIDlet-et úgy is tekinthetjük mint a Java ME program `main()` osztályát, hiszen minden alkalmazásnak kötelező része, és az életciklusa határozza meg az alkalmazás állapotát.

A forrást célszerű belerakni egy általunk készített csomagba, ugyanis a rendszer mindenképpen készít egyet (`default`), és a későbbiek folyamán a program bővítése esetén átláthatatlanná válhat a kód. Ezért az elemeket a `hajos_jatek` nevű csomagban hoztam

létre, ez jelzi az alábbi sor:

```
package hajos_jatek;
```

A szükséges képek pedig a `kepek` csomagban találhatók. Ebből a két csomagból áll a forrás.



9. ábra: Beépített telefonszimulátor

7.3.1. A Hajozas MIDlet

A `Hajozas` MIDlet a `javax.microedition.midlet` kiterjesztéseként jött létre, az eleje tartalmazza a szükséges importállományokat:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Hajozas extends MIDlet implements
 CommandListener { ... }
```

Maga a MIDlet deklarációja elárulja, hogy melyik osztály leszármazottja, illetve azt is, hogy egy interfészt implementál. Ez az interfész (`CommandListener`) az események kezelését látja el. Ezután az adattagok és a metódusok létrehozása következik. Három adattagja van a `Hajozas`-nak:

```
Display kijelzo = Display.getDisplay(this);
Command kilepesGomb = new Command("Kilépés",
                                    Command.EXIT,  20);
HajosVaszon grafikusKepernyo = new HajosVaszon();
```

A `kijelzo` a mobiltelefon kijelzőjét, a `kilepesGomb` a programból való kilépést lehetővé tevő gombját (parancsát), míg a `grafikusKepernyo` a grafikus képernyőt reprezentáló objektum. Ez utóbbi a játék megvalósításáért felelős osztály (`HajosVaszon`) konstruktorának segítségével áll elő.

A `Display` osztály `getDisplay()` metódusa számára azt kell megadni, hogy melyik MIDlethez rendeljük az objektumot. A `Command` osztály objektumának példányosítása során meg kell adni, hogy mi legyen a felirata a parancsnak, hogy mi legyen a parancs hatása, és egy prioritást. Ez a két osztály a `javax.microedition.lcdui` csomagban található.

A három, már korábban ismertetett metódust is itt, a MIDletben kell implementálni:

```
public void startApp() {
        kijelzo.setCurrent(grafikusKepernyo);
        grafikusKepernyo.addCommand(kilepesGomb);
        grafikusKepernyo.setCommandListener(this);
}
```

Az első lépésben telefon a kijelzőjéhez rendelem a grafikusképernyőt, majd kilépésgombot adom hozzá a képernyőhöz, végül a parancskezelőt az aktuális MIDlet

figyelésére állítom. Ehhez szorosan kapcsolódik egy másik metódus, a commandAction(), amelyben azt írom le, hogy mi történjen, egy parancs hatására:

```
public void commandAction(Command command, Displayable
 displayable) {
        if (command == kilepesGomb)
      notifyDestroyed();
}
```

Jelen esetben a kilépés gomb megnyomására egy jelzés érkezik, hogy elpusztul a MIDlet.

Található még két, kötelezően implementálandó metódus, amelyek a programom jelen állapotában üresen maradtak.

```
public void pauseApp() {
...
}

public void destroyApp(boolean unconditional) {
...
}
```

A `Hajozas` MIDlet ismertetése után térjünk rá a `HajosVaszon` osztály elemzésére.

7.3.2. A HajosVaszon osztály

Ez az osztály valósítja meg a grafikai megjelenítést, illetve a karakterek mozgatását, az események kezelését. Mint már említettem, ez az osztály is a `hajos_jatek` csomagban található.

```
package hajos_jatek;
import javax.microedition.lcdui.game.*;
import javax.microedition.lcdui.*;
```

```
import java.util.Random;
public class HajosVaszon extends GameCanvas implements
 Runnable {
...}
```

A `HajosVaszon` osztály is csomagok importálásával kezdődik, mert sok mindent használtam fel azokból. Ezután az osztály deklarációja következik, melyből látható, hogy a `GameCanvas` osztály származtatásából jött létre, illetve azt, hogy a `Runnable` interfészt implementálja. Ez utóbbi segítségével a szálkezelést fogom megvalósítani.

A `HajosVaszon` implementálása során használt osztályok a következők:

- **`GameCanvas`:** egy absztrakt osztály, mely a `Canvas` leszármazottja. A játék grafikus felületének alapját szolgáltatja, illetve az alacsonyabb szintű események kezelését látja el. Olyan, játékok fejlesztése során hasznos lehetőségeket biztosít, mint a gombok állapotának lekérdezése vagy az ún. dupla-puffer technika. Ez utóbbi segítségével nem a kirajzolt képen kell a módosításokat elvégezni, hanem a háttérben történnek a műveletek a kijelzőpufferben, és a kijelzett képpel időről időre megcseréli. A technika folyamatos, villogásmentes képet biztosít és alacsony memóriahasználatot (heap) igényel.
- **`Image`:** grafikus adatok tárolására használható osztály, mely segítségével képeket használhatunk a programokban. Objektumai a kijelzőeszköztől függetlenül csak a kijelzőpufferben léteznek, amíg ki nem íratjuk az eszközre. Érdemes 'PNG' formátumú képeket használni azok kis helyfoglalása és a szabadon történő felhasználásukat lehetővé tevő licenc miatt. A játékban a háttérhez és a karakterekhez használtam az osztályt.

  Először deklarálom a szükséges objektumokat, majd a HajosVaszon konstruktora létrehozza a megfelelő fájlokból a képeket (a biztonság kedvéért kivételkezelést is alkalmaztam):

```
Image csempek;
Image mihajonk, ellenhajo;
```

```
try {
mihajonk = Image.createImage("/kepek/mi.png");
ellenhajo = Image.createImage("/kepek/ellenseg.png");
csempek = Image.createImage("/kepek/hatter.png");
}
catch(java.io.IOException e) {
System.out.print("Nem talalom a kepet");
}
```

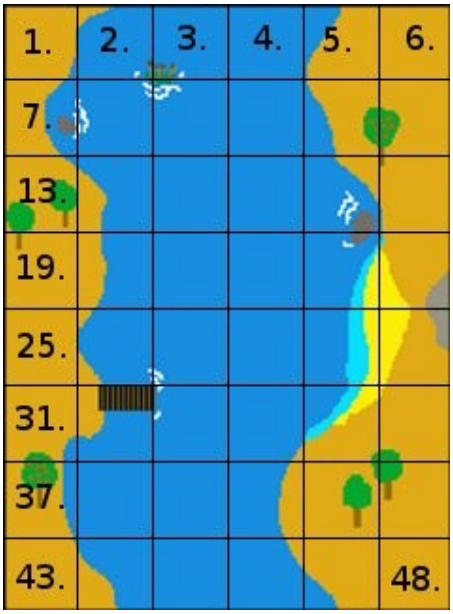
- **`TiledLayer`:** a `Layer` osztály leszármazottja. Vizuális elem, mely olyan rácsozott cellákból áll, amiket külön-külön, cellánként ki lehet tölteni képekkel. Használatával nagy felületek kitöltéséhez sincs szükség nagy képekre. A programomban a háttér kirajzolására használtam.

  Az objektum deklarálása után a konstruktorban a háttér szerkezetét adom meg. A szerkezet meghatározása a háttér képe alapján történik (10. ábra):

```
TiledLayer hatter;
int [][] palya;

palya = new int [][]{
          {1,  2,  3,  4,  4,  5},  {7,  8,  9,  4,  4,  11},
          {13, 14, 4, 4, 4, 17}, {19, 20, 4, 4, 4, 23},
          {25, 26, 27, 4, 4, 29}, {31, 32, 33, 4, 34, 35},
          {37, 4, 4, 4, 40, 41}, {43, 44, 4, 4, 46, 47},

             ...ez többször ismételhető...
};
```



10. ábra: A háttér csempéi

Szintén a konstruktor hozza létre magát a háttérobjektumot, méghozzá s `TiledLayer` osztály konstruktorának meghívásával, melynek meg kell adni a szélességet, a magasságot, a háttér keretét tartalmazó képet, valamint egy keret szélességét és magasságát. Következő lépésként a hátteret feltöltjük egy kettős ciklusban a pálya előzőleg megadott szerkezetének megfelelően.

```
hatter = new TiledLayer(palya[0].length, palya.length,
          csempek, 40, 40);

for(int i=0; i<palya.length; ++i)
    for(int j=0; j<palya[0].length; ++j)
        hatter.setCell(j, i, palya[i][j]);
```

– **`Sprite`:** szintén a `Layer` osztály származtatott osztálya. Egy olyan vizuális elem, amely segítségével animációt lehet előállítani. Ezt úgy tehetjük meg, hogy egy `Image`-ben több részre osztott képet tárolunk, melyek a mozgás különböző fázisait mutatják, és felváltva jelenítjük meg őket. Az azonos méretű képrészek azonos

méretűek, és a hozzájuk rendelt számokra hivatkozva tudjuk elérni őket (a háttér csempéihez hasonlóan). A játékprogramban a hajók mozgását készítettem el a `Sprite` osztály segítségével.

A programban az objektumok deklarációja után a példányosítás következik. Kétféle hajót használtam. A konstruktornak át kell adni az `Image` objektumot, amiben a kép található, illetve a pillanatkép szélességét és magasságát.

```
Sprite mi, ellenseg;
mi = new Sprite(mihajonk, 32, 32);
ellenseg = new Sprite(ellenhajo, 32, 32);
```

Ezután a mozgás animációját állítjuk be az animáció fázisait sorrendben megadva. A két hajó mozgása nem azonos fázisban történik. A három fázisból álló képek a 11. ábrán láthatók.

```
mi.setFrameSequence(new int[]{0, 1, 2, 1, 0});
ellenseg.setFrameSequence(new int[]{0, 1, 2, 1, 0});
```

Fontos még megemlíteni az ún. referenciapixelt, ami a `Sprite` egy kitüntetett pontja. Ezen keresztül (ezt tekintve alappontnak) tudjuk az objektumunkat elhelyezni, transzformálni stb. Az objektumok referenciapixelét a képernyő közepére legfelülre, illetve legalulra helyeztem:

```
mi.defineReferencePixel(0, 0);
ellenseg.defineReferencePixel(0, 0);
ellenseg.setRefPixelPosition(getWidth()/2, 0);
mi.setRefPixelPosition(getWidth()/2, getHeight());
```

12. ábra: Az ellenség, illetve a mi hajónk animációi

Eddig a játék világát építettük fel, még semmi sem rajzolódott a kijelzőre. Ahhoz, hogy életre keljen ez a világ, további osztályok és metódusok szükségesek:

- **`LayerManager`:** kezeli, összefogja a `Layereket`, vagyis a `TiledLayer`t és a `Sprite`-okat. A `paint()` metódus segítségével rajzolhatók ki az elemek. Az elemek egy sorba kerülnek, ami sorrendben hajtódik végre. A sor változtatására is van lehetőség, tehát adhatunk hozzá, törölhetünk belőle, illetve a be is szúrhatunk elemeket.
- **`Graphics`:** két dimenziós grafikai elemeket (pl. vonal, téglalap, szöveg, kép, ív) rajzolhatunk vele a kijelzőre, illetve színezésre és a szövegstílus beállítására is van lehetőség a használatával.

Az játék mozgatása, a kijelzőn bekövetkező események vezérlése a `HajosVaszon` osztály `run()` metódusában történik, melyet egy külön szál segítségével meghívok meg:

```
new Thread(this).start();
```

A metódus első utasításával a grafikus objektumot kell elérnem, hiszen már rajzolni is akarok a vászonra:

```
Graphics g = getGraphics();
```

A következő részben meghatározzuk, hogy hová rajzoljuk ki a hajónkat, illetve a hátteret. A hajót a képernyő közepére terveztem először megjelentetni, a háttér pedig a kijelző alsó szélétől kezdi majd betölteni a játékteret:

```
int miOszlop = getWidth()/2;
int hatterIndex = - (palya.length * 40 - getHeight());
```

Ezután elérkeztünk a játék felélesztéséhez, az animáció elindításához. Maga az animáció egy ciklusban fut. Első futás előtt beállítjuk a kezdőértékeket, majd minden egyes állapot kirajzolása után elvégezzük a szükséges módosításokat, változóbeállításokat, a kijelzőn megjelenő változásokat, aztán a képet újra és újra kirajzoljuk.

Amint azt korábban említettem, a GameCanvas osztály biztosítja a billentyűállapotok lekérdezésének lehetőségét. A programban összesen négy gomb használható a játék irányítására. A ciklus azzal indul, hogy megnézzük, hogy nyomott-e gombot a játékos. Ha igen, akkor változások következnek be.

```
while(z == true){
    int lenyomottBillentyu = getKeyStates();
    if((lenyomottBillentyu&GameCanvas.RIGHT_PRESSED) != 0){
          if(miOszlop+3 < getWidth() + mi.getWidth()-64)
               miOszlop += 3;}
    else if((lenyomottBillentyu&GameCanvas.LEFT_PRESSED)!= 0){
          if(miOszlop-3 > - mi.getWidth()+32)
               miOszlop -= 3;}
    if ((lenyomottBillentyu&GameCanvas.UP_PRESSED)!=0){
          if (gyorsito <= 5)
               gyorsito += 1;}
    else if ((lenyomottBillentyu&GameCanvas.DOWN_PRESSED)!=0){
          if (gyorsito >= 2)
               gyorsito += -1;} ... }
```

Ha a játékos a bal vagy a jobb gombot nyomja meg, akkor a hajó balra vagy jobbra manőverezik a folyón az ellenséges hajókat kikerülve. Ügyeltem arra, hogy a hajó végig a képernyőn maradjon. A fel vagy le gomb megnyomása a hajó sebességét változtatja, gyorsítja vagy lassítja, ezzel szintén a manőverezést teszi könnyebbé. Gyorsítani csak egy bizonyos határig lehet a hajót, hiszen a végsebessége korlátozott, míg lassítani a kezdeti sebesség alá nem tudunk.

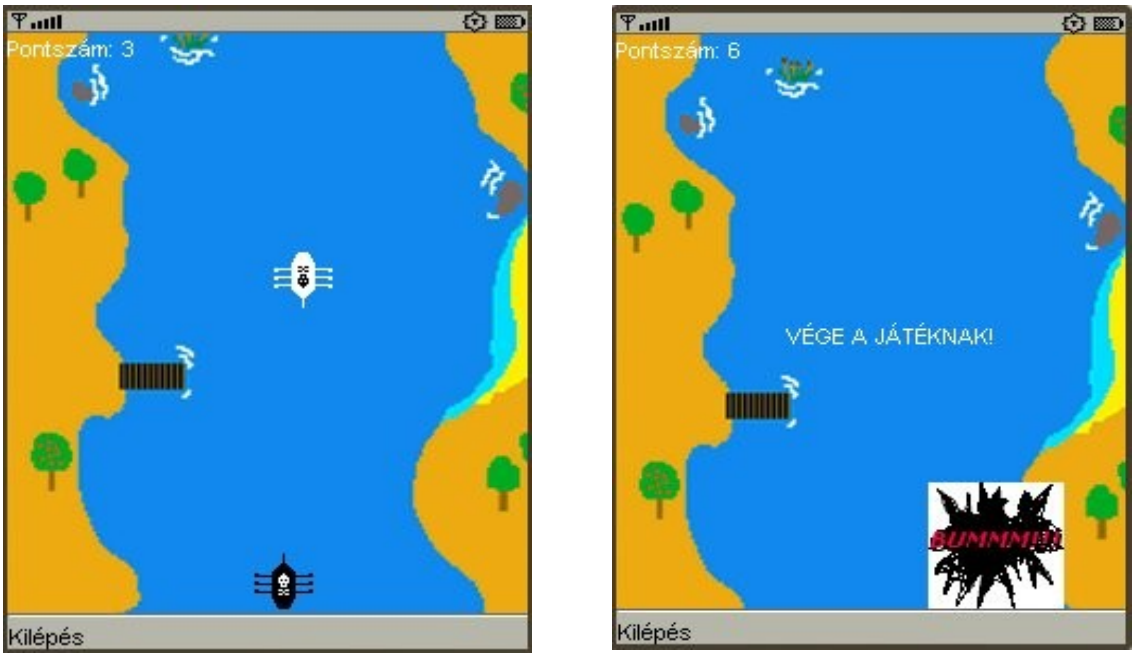
A képfrissítés alakalmával le kell törölnünk a vásznat, hogy az következő „képkockát” ki tudjuk rajzolni az új állapotnak megfelelően elmozdított háttérrel és az új állapotba helyezett hajókkal:

```
    g.setColor(0xffffff);
    g.fillRect(0, 0, getWidth(), getHeight());
```

```
if(hatterIndex < 0)
      hatterIndex = hatterIndex + gyorsito;
else
      hatterIndex = - (palya.length * 40 - getHeight());
hatter.setPosition(0, hatterIndex);
mi.setRefPixelPosition(miOszlop, getHeight() - 32);
ellenseg.setRefPixelPosition(belep, sorlepteto + gyorsito);
```

Ezután kirajzoljuk először a hátteret, majd a Sprite-okat a következő fázisukban végül a képet ténylegesen is megjelenítjük a kijelzőn:

```
hatter.paint(g);
mi.paint(g);
ellenseg.paint(g);
mi.nextFrame();
ellenseg.nextFrame();
flushGraphics();
```



12. ábra: A játék működés közben, és a befejező kép (a szimulátorban)

A játék folyamatosságát biztosítja, hogy a kijelző alsó szélén kiúszó hajó felül ismét megjelenik újabb kikerülendő akadályként. Ezen kívül növeli a játékélményt, ha az új ellenséges hajó a folyó véletlenszerű oszlopában jelenik meg, ezt a `java.util.Random` osztály felhasználásával valósítottam meg. Továbbá az ellenség kicselezéséért pontokat kapunk, az aktuális pontszám folyamatosan kiíródik a kijelző bal felső sarkába (12. ábra).

A játék végéhez érünk, ha beleütközünk egy ellenséges hajóba. Ekkor felrobban a hajónk, betöltődik egy robbanást jelző kép a helyére, és a játék végére figyelmeztető üzenet is megjelenik végül a fő ciklusunkat is leállítjuk (12. ábra).

```
if (mi.collidesWith(ellenseg, true)){
     System.out.println("BUMMM!!!");
     g.drawImage(utkozes,mi.getRefPixelX(),
                                  mi.getRefPixelY()-32, 0);
     g.drawString("VÉGE A JÁTÉKNAK!", 80, getHeight()/2, 20);
     flushGraphics();
     z = false;}
```

Összefoglalás

A dolgozatom célja az volt, hogy betekintést nyerjek a mobiltelefonos játékok fejlesztésének érdekes, kihívásokkal teli világába.

A munka során megtapasztalhattam az első lépéseket, amelyek kijelölik az utat a játékprogramozás felé. Megismerkedtem az alapvető lehetőségekkel, amelyek segítik a tapasztalt programozót is egy játék felépítésében és életre keltésében.

Dolgozatomban a mobiltechnológia történetén keresztül a kezdetektől a jelenbe érkeztem, és a közeljövőbe is próbáltam bepillantást nyerni. Megnéztem, hogy melyek azok a mobilkészülék-típusok, amelyeket a gyártók, és a szolgáltatók jelenleg kínálnak, melyek a népszerű mobilplatformok, illetve az ígéretes újdonságok.

A következő lépés volt az eddigi információk alapján a fejlesztés platformjának kiválasztása volt. Végül a Java technológiára esett a választásom, azon belül is természetesen a ME változatra, amit a mobilkészülékekhez alakítottak ki, hiszen a ma kínált készülékek döntő többsége támogatja, és vélhetően a közeljövő sem fog e tekintetben jelentősen változni. A Java technológiának számos előnye van, így ezeknek és külön a nyelv általam használt eleminek bemutatása sem maradhatott ki a sorból.

Ezután kiválasztottam a fejlesztői eszközrendszert. A Netbeans és GIMP kombinációját választottam a fejlesztéshez. Nagyon hasznos eszközöket kapunk velük a kezünkbe a programozáshoz, rengeteg segítséggel, sok hasznos funkcióval.

A fejlesztés sikerének legbiztosabb módja, főleg a játékfejlesztés kezdeti szakaszában, ha kiválasztunk egy konkrét telefontípust, és arra készítjük el a programunkat. Ezért én is megtettem ezt a célszerű lépést, egy Nokia 6300 típusú készüléket választotva. A telefon Java támogatással kapcsolatos tulajdonságait figyelembe véve – elsősorban a CLDC 1.1-re és a MIDP 2.0-ra gondolok – ezután hozzáláttam a konkrét fejlesztéshez. Munkám végén el is készült egy kisebb játékprogram.

Mindezek alapján úgy gondolom, hogy a célkitűzésemnek eleget tettem, hiszen a munka során önállóan készítettem el a programot. A programban felhasználtam a Java ME platform alapvető elemeit, amelyek nagyban megkönnyítik a fejlesztési munkát. Ezek közül kiemelem a figurák Sprite-okkal, a háttér TiledLayer-rel történő kezelését, az animációk létrehozását,

illetve az ütközések egyszerű figyelését, melynek során az elemeket egyben lehet kezelni, nem kell pontról pontra összehasonlítani a helyzeteiket.

Ez a munka kijelölt számomra egy utat, mert már vannak további ötleteim a játék továbbfejlesztésére, melyek megvalósításával egyre inkább elmélyülhetek a mobilprogramozás ismereteiben. Lehetne kezdőanimációt, menüt tervezni a játéknak, melyben akár nehézségi fokozatokat is lehet választani. Lehetne egyszerre több ellenség a képen, melyeket távolsági fegyverekkel is el lehet pusztítani, a pályán bónuszpontokat összeszedni stb. Más irányú ismeretek megszerzésével a játékhoz akár multimédiás fejlesztések is kapcsolhatók, pl.: zene, hangok.

Új kihívás lehet más mobilplatformokra is átültetni a programot. Elsősorban az újabbak megcélzása lehet ígéretes mint az Andriod, illetve a Maemo.

Irodalomjegyzék

Könyvek, jegyzetek (nyomtatott):

1. Bártfai Norbert: Nehogy Már a Mobilod Nyomkodjon Téged (2007)
2. Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a Mobilprogramozásba (2008)
3. Juhász István: Programozás 2., Egyetemi Jegyzet (2004)
4. Bártfai Norbert: Programozó Páternoszter (2006)
5. Nyékyné G. Judit (szerk.): Java 2 Útikalauz Programozóknak (2008)
6. Nagy Gusztáv: Java Programozás (2007)
7. Carol Hamer: Creating Mobile Games (2007)
8. Ray Rischpater: Beginning Java ME Platform (2008)

Webcímek, leírások, jegyzetek (elektronikus):

9. http://www.javaforum.hu
10. http://java.sun.com/
11. http://hu.wikipedia.org/wiki/Java_%28programoz%C3%A1si_nyelv%29
12. http://www.forum.nokia.com/
13. http://www.tankonyvtar.hu/informatika/javat-tanitok-javat-080904
14. http://netbeans.org/
15. http://www.gimp.org/

1. Függelék

A Java ME forráskódjának lefordítása után elkészül egy `.jar` és egy `.jad` fájl. Az előbbi fájl tartalmazza a lefordított `.class` bájtkódokat, illetve a program által használt egyéb fájlokat (pl.: képeket). Ezt kell futtatni, így a telefonra elég ezt a fájlt letölteni. A .jad fájl pedig erről a futtatható fájlról tartalmaz információkat, amiből kiderül a készülék számára, hogy megfelel-e a szoftver által támasztott követelményeknek.

Az elkészített programom `.jad` fájljának tartalma:

MIDlet-1: Hajos,,hajos_jatek.Hajozas
MIDlet-Jar-Size: 30037
MIDlet-Jar-URL: hajos.jar
MIDlet-Name: hajos
MIDlet-Vendor: Revesz
MIDlet-Version: 2.0
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.0

2. Függelék

A dolgozatban használt rövidítések magyarázata

- **AMS:** Application Management Software – Alkalmazáskezelő Szoftver
- **API:** Application Programming Interface – Alkalmazásprogramozási felület
- **CDC:** Connected Device Configuration – mobilkészülék-konfiguráció
- **CLDC:** Connected Limited Device Configuration – mobilkészülék-konfiguráció
- **GIMP:** GNU Image Manipulation Program – képszerkesztő program neve
- **IDE:** Integrated Development Environment – Integrált Fejlesztői Környezet
- **JAD:** Java Application Descriptor – Java alkalmazásleíró fájl
- **JAR:** Java Archive Resource – Tömörített Java fájl, futtatható program
- **JDK:** Java Development Kit – Java fejlesztőkörnyezet
- **JRE:** Java Runtime Environment – Java futtatókörnyezet
- **JSR:** Java Specification Request – Java opcionális csomag
- **JVM:** Java Virtual Machine – Java Virtuális Gép
- **MIDlet:** MIDP-t használó alkalmazás
- **MIDP:** Mobile Information Device Profile – kibővített eszközkészlet a CLDC-hez
- **OHA:** Open Handset Alliance – egy egyesülés, melynek a Google is tagja
- **SDK:** Software Development Kit – programfejlesztői készlet
- **UI:** User Interface – Felhasználói felület
- **UML:** Unified Modeling Language – szoftvertervezéshez használt modellező nyelv