TARTU ÜLIKOOL

MATEMAATIKA-INFORMAATIKATEADUSKOND

Arvutiteaduse instituut

Informaatika eriala

**Sergei Mihhailov**

# *Go* mängu implementeerimine *Android*'i rakendusena

Bakalaureusetöö (6EAP)

Juhendaja: M.Sc V. Leping

TARTU 2014

# *Go* mängu implementeerimine *Android*'i rakendusena

## Lühikokkuvõte:

Antud bakalaureusetöö eesmärgiks oli arendada rakendus Android platvormile, mis kasutaks *Go* mängu tehisintellekti mootorit ning võimaldaks mängijal *Go'*d mängida ning arendusprotsessi analüüsida.

Töö raames analüüsitakse erinevaid *Go* tehisintellekti algoritme, analüüsitakse ning põhjendatakse erinevaid antud projektiga seotud tehnoloogilisi valikuid ning kirjeldatakse rakenduse arendusprotsessi algusest lõppuni.

## Võtmesõnad:

Android, Go, lauamäng, arendusprotsess, tehisintellekt

# Go board game implementation for Android platform

## Abstract

The purpose of this bachelor thesis is to develop an application for GNU *Go* artificial intelligence engine and analyze the development process from project management point of view.

In this thesis author explains all rules of Go board game in details. Author also shows that in order to "teach" computer to play Go, many obstacles should be overcome and explains how those problems are solved today in theory and practice.

This thesis also contains author's clarifications on the subject of development tools choices for Android application development.

## Keywords:

Android, Go, board game, development process, artificial intelligence

# Sisukord

# Sissejuhatus

Eelmise aasta maiks oli müüdud üle ühe miljardi nutitelefoni *Android* operatsioonisüsteemiga.[18] Nutitelefonide müük kasvab nii kiiresti, et nüüd igal päeval aktiveeritakse poolteist miljonit *Android*'i seadet.[19] On tõusnud mitte ainult nutitelefonide arv vaid ka nende arvutusvõimsus. Tegemist ei ole enam lihtsalt seadmetega, mille eesmärgiks on ainult helistada. Tänapäeval tavaline nutitelefon võimaldab lehitseda veebi, teha pilte ning videoid, kuulata muusikat ning mängida mänge.

Autori eesmärgiks on adapteerida *Go* mängu tehisintellekti mootorit mobiilseadmetele, luua loogilise mängu rakendus Android'i platvormile ning analüüsida arenduse protsessi.

## *Miks siht-platvormiks on valitud Android?*

Siht-platvormiks on valitud Android mitmel põhjustel. *Google Android* on kõige populaarsem nutitelefoni operatsioonisüsteem[20] mille arendus toimub *Java* programmeerimiskeelt kasutades. *Android* on avaliku lähtekoodiga operatsioonisüsteem, seega on võimalik luua rakendusi sama koodi-stiiliga nagu *Android* ise on, ning maksimaalselt optimeerida rakenduse tööd.

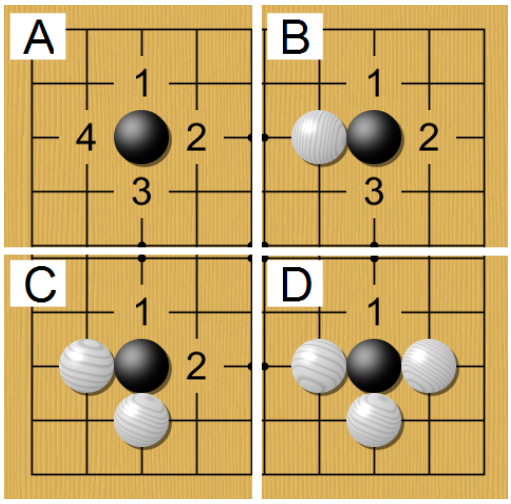
## *Töö struktuurist.*

Töö esimene peatükk annab ülevaate *Go* mängu reeglitest. Teises peatükis vaadeldakse erinevaid *Go* tehisintellekti meetodeid ning räägitakse miks pole lihtne arvutit *Go*'d mängima õpetada. Kolmanda peatüki eesmärgiks on vaadelda kuidas toimub *Android*'i rakenduse arendus ning milliseid tööriistu arenduse protsessis kasutatakse. Töö neljas peatükk annab ülevaate loodud rakenduse arhitektuurist. Viienda peatüki eesmärgiks on analüüsida erinevaid arendusmeetodeid ning kirjeldada kuidas *Go* mängu tehisintellekti rakenduse loomine oli plaanitud ja tehtud.

# 1. *Go* mängu reeglid.

## *1.1 Põhireeglid*

*Go* on Hiinast pärit territoriaalne mäng, mille võitmiseks tuleb oma kividega piirata ja kontrollida suuremat territooriumi mängulauast kui vastane. Mängitakse ruudulisel 19×19 joonega laual. Ühel mängijal on mustad, teisel valged nupud, mida kutsutakse kivideks. Algajad kasutavad ka väiksemaid laudu, näiteks 9×9 ja 13×13. Must alustab mängu. Mängijad asetavad oma kive kordamööda lauale vabadele joonte ristumiskohtadele. Oma käik on lubatud vahele jätta ehk passida. Kord juba lauale asetatud kivi ei ole enam lubatud liigutada. Vastase kive võib "vangistada", piirates need oma kividega sisse sellisel moel, et vastase kividest ei ole ühtki horisontaalset või vertikaalset ühendust ühegi vaba joonte ristumiskohaga.



*Pilt 1: Go mängu triviaalsed positsioonid.*

A näitab ühe musta kivi neli vabadust (vabaduse punkti), ja valge vähendab selle kivi vabadusi (B, C ja D). Kui mustal kivi on ainult üks vabadus (D), siis must kivi on *atari*[1] seisus. Valge võib musta kivi vangistada ehk mängulauast eemaldada (käik 1) [vt. Pilt 1: Go mängu triviaalsed positsioonid. ]

Selleks, et olla “elus”, grupis peab olema vähemalt kaks “silma”. Silm on tühi koht, mis on ümbritsetud sõbralike kividega, kus vastane ei saa kunagi mängida.

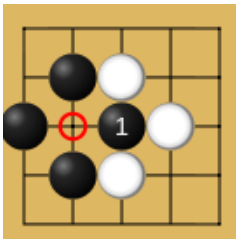
1 Atari – kivi või kivide grupi seis *Go’s*, kui kivil või kivide grupil on ainult üks vabadus.

Mäng lõpeb, kui mõlemad mängijad on järjest käigu vahele jätnud. Pärast mängu lõppu toimub skoorimine, mille käigus otsustatakse võitja. Selleks arvutatakse eraldi musta ja valge mängija skoorid. Mängija skooriks on (jaapani reeglite järgi) tema kivide poolt piiratud vabade ruutude arv, millele on lisatud tema poolt ära võetud vastase kivide arv. Seega pole hea kaotada liiga kergekäeliselt oma kive. Juhul, kui mängijad ei suuda kokku leppida, kellele mis ala kuulub, siis nad peavad edasi mängima. Mängu tulemuseks on mängijate skooride vahe. Sealjuures lisatakse valge skoorile nn "komi" 6.5 punkti. Komi on vajalik, kuna mustal on alustajana väike edumaa, ning komi 6.5 hoiab mängu balansis. (0.5 on lisatud selleks, et ei tekiks viike.) Nii näiteks kui valge on piiranud 40 punkti ja võtnud ära 8 musta kivi, ning must on piiranud 48 punkti ja võtnud ära 4 valge kivi, siis on valge skoor 40+8+6.5=54.5 ning musta skoor 48+4=52, seega valge on võitnud 2.5 punktiga.

## *1.2 Erandid.*

“Enesetapud” on keelatud. Mingisse positsiooni ei tohi käia siis kui selle tagajärjel a) ükski vastase rühm ei sure, ning samal ajal b) sul endal moodustub rühm, millel pole ühtegi vabadust.

Selleks, et vältida lõpmatu pikkusega mänge, on sisse toodu järgnev ko reegel : kui vastane on võtnud ära täpselt ühe kivi ja mänguseis on sarnane mänguseisuga pildis [vt. Pilt 2: Ko-reegel.]. Siis ei saa kohe tema kivi (kivi nr.1) ära võtta, sest see viib selleni, et vastane võib jälle kivi “vangistada” ja mäng kestab lõpmatult.



*Pilt 2: Ko-reegel.*

Seki on selline mängu olukord, kus kivide grupid ei saa teine teist vangistada. See on seotud sellega, et grupp, mis alustab rünnakut vähendab mitte ainult vastase grupi vabadusi, kuid ka enda grupi vabadusi.

# 2. Probleemi püstitus

## *2.1 Tehisintellekt Go mängus.*

Pikka aega arvati, et tehisintellekt *Go* mängus ja males erineb. Ja see erinevus on seotud sellega, et maalemängu kiirotsingu meetodid võrreldes inimese kogemusega pole *Go* mängus efektiivsed. Seega suur osa tööd *Go* tehisintellekti arendusest on kulutatud sellele, et liita eksperdi teadmised ja lokaalne otsing, selleks, et mängu taktikast aru saada. Tulemuseks on arvutiprogrammid, mis on võimelised leidma häid lahendusi lokaalsetes olukordades, aga ei saa analüüsida tervet mängu. Selliste klassikaliste programmide mängu võimsus kasvas aeglaselt. Seega selle valdkonna areng oli aeglane ja arvati,et programm, mis oskab hästi *Go*'d mängida luuakse alles kauges tulevikus. Isegi programmi, kirjutamine mis suudab kindlaks teha, kes võitis lõppenud mängus, ei olnud triviaalne ülesanne.

### 2.1.1 Liiga suur mängu piirkond.

Peamine põhjus, miks arvuti ei oska *Go*'d mängida nii hästi, kui inimene on selle mängu suur laud (19x19, 361 ristuvaid jooni). Probleem on selles, et suures lauas ei saa teha *Alpha-Beta* [17] sügavuti otsingut. Võrreldes malega, tuleb märkida, et käigud *Go*'s praktiliselt pole piiratud mängu reeglitega. Kui males on avakäigu sooritamiseks 20 võimalust, siis *Go*'s esimese käigu mängimiseks on 361 võimalust. Ja mängu ajal tekkivad momendid, et üht ja sama käiku mängitakse mitu korda.

### 2.1.2 “Täpse” avangu teooria puudumine.

Mängu algfaasis - *fuseki* – on olemas määratud printsiibid, kuidas kivide gruppe arendada, aga neid üldisi võimalusi on palju rohkem, kui nt. males. Uued arendamise variandid võivad ilmuda juba kolmanda või neljanda käiguga, ja pädev mängu avang on võimatu, mõistmata strateegilisi arenguperspektiive, mis tekivad laual. *Joseki’*d (skeemid, kuidas mängida piiratud positsioone, eriti nurkades) tehisintellekt kasutada ei saa, see ei anna garanteeritud tulemust, sest nende kasutamise edu sõltub olukorrast kogu mängulaual, isegi valida sobivat *joseki’d* antud seisule konkreetses nurgas, on keeruline intellektuaalne ülesanne.

### 2.1.3 *Ko*-võitlemine.

*Ko* reegel sageli viib selleni, et mängu tempo muutub drastiliselt. Isegi kogenud mängijatele on raske hinnata milline on mäng. Tavaliselt *Ko*-võitlemises[2] inimene kasutab oma kogemusi ja intuitsiooni, kuid aga arvutile on raske formaliseerida neid mõisteid.

### 2.1.4 Hindamise funktsioon.

Veel üks probleem seisneb selles, et on vaja luua hea hindamise funktsioon *Go*’ks. Igal mängu sammul võib esineda mitu head käiku ja selleks, et valida nendest parim, peab arvuti hindama erinevaid võimalikke tulemusi. See ülesanne *Go*’s on kõige raskem. Näiteks võib tekkida selline olukord, kus vastase kivide gruppi vangistamiseks on vaja tugevdada tema gruppi teises kohas. Otsus selle kohta, kas selline vahetus on kasulik või mitte on väga keeruline isegi inimesele.

2 *Ko*-võitlemine - *Go* mängufaas mille jooksul mängijad mängivad *Ko*-reegli järgi

### 2.1.5 Mängu lõppfaas.

Mängu lõppfaasis ehk *ese*'s on vähem võimalikke käike kui alguses või keskel, võiks siis järeldada, et arvutil lõppfaasis on palju lihtsam mängida. Aga selline järeldus ei kehti järgmiste probleemide tõttu :

- Mängu lõppfaas on kõige "matemaatilisem" mängu osa, sest peaaegu igale käigule on võimalik anda hinnangut, ehk mitu punkti see käik annab. Ja isegi see etapp on arvutile liiga raske – sest ta ei oska ko-võitlemise tulemust õigesti hinnata.
- Lõppfaas võib viia selleni, et on vaja hinnata kas mingi kivide grupp on "elus" või "surnud" ja tehisintellektil on seda väga keeruline teha.
- *Ese*'s sageli tekkivad kolmekordsed või neljakordsed Ko-võitlemised, mis viitavad sellele, et üht ja sama mängu on võimalik mängida lõpmatult.

Seega on väga raske programmeerida head algoritmi lõppfaasis mängimiseks, rääkimata tervest mängust.

## *2.2 Go-tehisintellekti põhi-algoritmid.*

### 2.2.1 Minimax otsing mängupuus.

*Minimax*'i [16] otsingu puude kasutamine on traditsiooniline tehnika mängu tehisintellekti loomiseks. *Minimax*'i otsingu algoritm on selline : vaadeldakse kõiki teoreetiliselt võimalikke käike määratud sügavuseni, ja kasutatakse hinnangfunktsiooni selleks, et hinnata iga käigu väärtust. Otsitakse efektiivsem käik ja arvestades sellega, et see käik on juba tehtud, vaadeldakse kõiki ülejäänud käike. Algoritmi tööd saab esitada puuna. Kuigi tehnikad, mis kasutavad otsingu puud, andsid häid tulemusi males, on vähem edukad *Go*'mängus.

See on seotud sellega, et on keeruline luua efektiivset hinnangu funktsiooni, sest teoreetiliste käikude hulk on liiga mahukas. Seega otsingu puu tehnika vajab palju riistvaralist ressurssi.

On olemas tehnikaid, mis saavad parandada otsingu puu kiirust ning mälu kasutust: nt. *Alpha-Beta* meetod võib vähendada hargnevuse tegurit praktiliselt kaotamata mängu käikude kvaliteedis. Kõige võimekam programm, mis kasutab otsingut mängupuus on *GNU Go* [3].

### *2.2.2 Ekspertsüsteemid.*

Algajad sageli õppivad vanade meistrite mänge vaadates. On hüpotees, et teadmiste koondamine – on tugeva tehisintellekti loomise võti. Selles seisneb ekspertsüsteemi tehnika : sõnastada tugeva mängu printsiipe ja reegleid nii palju kui võimalik , et need hiljem masinkoodina formaliseerida ja otsustada kus kasutada.

See meetod, kuni viimase ajani, oli kõige edukaim. Sellist meetodid kasutavad : *Handtalk* (hiljem *Goemate*) [7], *The Many Faces of Go* [8], *Go Intellect* [9] ja *Go* + + [10], igaüks neist oli mingil hetkel nomineeritud parimaks programmiks *Go* maailmas.

### 2.2.3 Monte-Carlo meetod.

*Monte-Carlo* meetodi algoritm on järgmine: esialgselt leitakse käike milliseid on võimalik teha ja pärast mängitakse iga leitud käiguga palju suvalisi mänge. Käik, mis annab kõige rohkem võite, valitakse järgmiseks käiguks. Võrreldes mängupuu otsinguga ja ekspertsüsteemiga ei vaja see meetod palju teadmisi *Go*'st ning kasutab vähe mälu ja protsessori ressurssi. Sellist meetodit kasutavad : Zen [11], The Many Faces of *Go*, Leela [12], MoGo [13], Crazy Stone [14] ja Gobble [16].

# 3. Androidi rakenduse loomine

## 3.1 Sissejuhatus

Esimene *Android'*i seade (*HTC Dream*) oli müüdud aastal 2008. Sinna oli paigaldatud *Android* operatsioonisüsteem versiooniga 1.0. Veebruaris 2013 aastal tehti *Android 4.2.2*. Seal on üle 1000 uue *API*[3]'i, mis annavad võimalusi arendajatele suurepäraseid rakendusi luua.

## 3.2 Programmeerimiskeskkond

Androidi rakenduste loomiseks kasutatakse *Android Software Development Kit (SDK)*, *Android Native Development Kit (NDK)* ning *Android Development Tools (ADT) plugin*'it koos *Eclipse integrated development environment'*ga *(IDE)*. *ADT* laiendab *Eclipse'*i funktsionaalsust ning annab võimalusi *Android* projekti arendamiseks, kasutajaliidese komponentide loomiseks, rakenduse silumiseks ja allkirjastatud (või allkirjastamata) *.apk* faile eksportimiseks.

## 3.3 Tehnoloogia valik

### 3.3.1 Java Native Interface

JNI ehk *Java Native Interface* on raamistik mis võimaldab virtuaalmasinas jooksvas koodis välja kutsuda C/C++ ning *assembler*'i koodi ja vastupidi.

JNI võimaldab kirjutada puhtaid meetodeid selliste olukordade käsitlemiseks, mille puhul rakendust ei saa kirjutada täielikult Java programmeerimiskeeles, näiteks kui standard *Java* klassi teek ei toeta platvormi spetsiifilisi funktsioone või teeke. Paljud standard teegi klasse

3 Programmiliides (inglise keeles Application Programming Interface) on reeglistik olemasoleva valmisprogrammiga suhtlemiseks. Võimaldab programmeerijatel kirjutada lisa- ja abiprogramme, mis täiendavad või laiendavad programmi(de) funktsionaalsust. [1]

kasutavad JNI’d, selleks et anda arendajatele ja kasutajatele teatavat funktsionaalsust, näiteks failide I/O ja heli.

JNI raamistik võimaldab *C* keele meetodites kasutada *Java* objekte samamoodi, nagu *Java’*s neid objekte kasutatakse. *C* keele meetodid saavad luua *Java* objekte ja kasutada neid objekte , et oma ülesandeid täita. *C* keele meetodid saavad ka kasutada objekte, mis on loodud *Java* koodist.

JNI on "pääsetee" *Java* arendajatele, sest see võimaldab neil lisada funktsionaalsust oma Java rakendustele, mida standard Java API ei võimalda. Seda kasutatakse ka ajakriitiliste ülesannete täitmiseks nagu keerulise matemaatilise võrrandi lahendamine, sest puhas *C* kood võib olla kiirem kui JVM kood.

### 3.3.2 Software development kit

SDK [4] sisaldab vajalikke vahendeid Android’i rakenduste loomiseks :

- Dokumentatsioon
- Lähtekood
- SDK tööristad. Sisaldab tööriistu silumiseks ja testimiseks, lisaks muud abivahendid, mis on vajalikud, et rakendust arendada.
- *Android*’i seadme emulaator.
- Valitud *Android*’i versiooni süsteemi kujutis. Ilma selleta ei saa *Android*’i emulaatorit käima panna.
- Näited kuidas SDK tööriste kasutada.
- Lisa teegid. Nt. Litsentside haldamise teek, *Google*’i Arveldus teek.
- *Android Debug Bridge (ADB)*. See vahend võimaldab ühendada ning hallata virtuaalseid või reaalseid *Android*’i seadeid.

Ametlikult toetatavad arenduskeskkonnad, on *Eclipse*, *IntelliJ Idea* ning *NetBeans*. Need

arenduskeskkonnad annavad võimaluse kasutada ADT vidinaid ja luua SDK *Android*'i rakendusi. Lisaks arendajad võivad kasutada suvalisi tekstiredaktoreid Java ning XML-failide redigeerimiseks. Sellisel juhul on vaja ka kasutada käsurea tööriistu (nt. Java Development Kit, Apache Ant), et luua, ehitada ja siluda Android rakendusi. Tuleb öelda ka seda, et SDK toetab ainult programmeerimiskeelt Java.

### 3.3.3 Native development kit

Android NDK [5] on tööriistakomplekt, mis võimaldab sisseehitada komponente, mis kasutavad puhast *C* koodi oma *Android*'i rakendusse. Põhierinevus SDK'ga on see, et NDK pakub tööriistade komplekti rakenduse arendamiseks kasutades *C++* ning C keeli ja NDK's pole *Android*'i seadme emulaatorit.

NDK'd on võimalik kasutada kahel viisil:

- Kirjutada puhta *C* koodis ainult funktsioonide teek ning kasutada seda teeki oma rakendusest JNI kaudu. See lähenemine annab võimaluse kasutada kõiki mugavusi, mis pakub SDK ja samal ajal kasutada puhast *C* koodi, kui seda väga vaja on.
- Kirjutada rakenduse põhiosa puhtas *C* koodis, asendades need funktsioonid, mis pole kättesaadavad *C* koodi abil. See lähenemine muudab rakenduse arendusprotsessi keerulisemaks, sest mõningaid vajalikke funktsioone on võimalik kasutada ainult SDK kaudu. Valides selle tee peab aga loobuma paljudest mugavustest – staatiline süntaksianalüüs on palju parem, kõik kolmanda osapoole teegid peab enamasti ise kompileerima, debugger'i käivitamine ja arusaadava vigu rapordi kättesaamine nõuavad palju lisavaeva ning mäluprofileerijate kasutamisest võib praegu ainult unistada.

### 3.3.4 Kokkuvõtte

Kuna Google ei soovita arendada rakendust, mis sunnib kasutajat vastust oodata kauem kui 200 millisekundit, siis rakenduse reageerimisaeg  on väga kriitiline. Seega on mõistlik kasutada NDK. Teisest küljest, puhta NDK kasutamine muudab rakenduse arendamist palju pikemaks. Just sellepärast otsustati kasutada NDK’d ainult tehisintellekti funktsiooni kasutamiseks ja SDK’d kasutajaliidese arendamiseks.

# 4. Rakenduse arhitektuur

## 4.1 Kasutajaliidese ja tehisintellekti mootori omavaheline suhtlus

Rakenduse kasutajaliides suhtleb tehisintellekti mootoriga kasutades GTP4 protokolli. Aktiivse mängu jooksul iga kasutaja tehtud käik transformeeritakse tekstiks, kujul “black_A15”, kus “A” ja “15” on kivi koordinaadid ja “black” on kivi värv. Vastuseks saadakse tekstirida, mis võib koosneda:

- Tehisintellektiga tehtud käigu koordinaatidest
- Käsust “pass”, juhul kui arvuti otsustanud alustada mängu lõppfaasi
- Käsust “resign”, juhul kui arvuti arvab, et kasutaja võitis

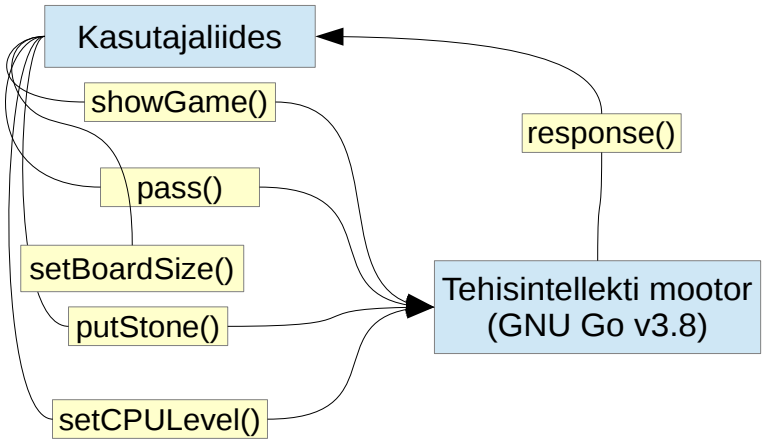
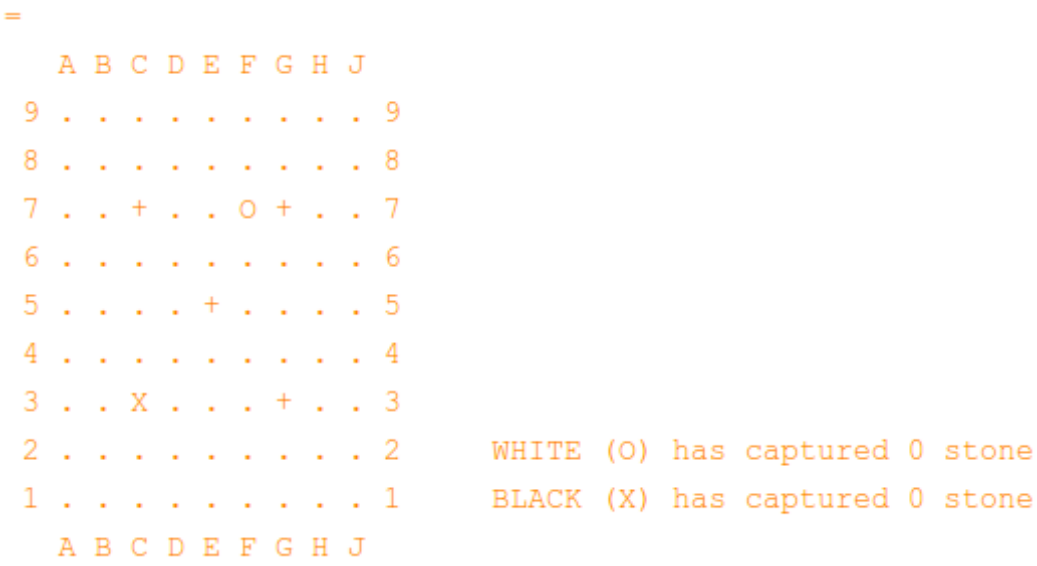



*Diagramm 1: Kasutajaliidese ja tehisintellekti mootori omavaheline suhtlus*




*Pilt 3: Mänguseis GNU Go's*

Kuna GNU *Go* hoiab mälus jookva mängu loogilise pildi, on võimalik seda pilti temalt küsida ja vastuseks antakse Pilt 3: Mänguseis GNU Go's.

## 4.2 Rakenduse vaated

Diagramm 2: Rakenduse vaadete hierarhia näitab kuidas rakenduse kasutajaliides tehtud on. "Main view" näidatakse esimesena. Sellest vaatest on võimalik avada "Settings view" või "New Game view". "Settings view" on tehtud selleks, et tehisintellekti võimsust, vihjeid ning helisid konfigureerida. "New Game view" abil kasutaja saab uue mängu parameetreid määrata: mängulaua suurust, komi, punkti arvutamise süsteemi ning edumaad. Kui kõik mängu algparameetrid on määratud, avatakse rakenduse peavaade, ehk "Game view". Seal toimub interaktiivne *Go* mängu mängimine ning on võimalik ühe sammu tagasi võtta ja alustada mängu lõpp-faasi. Mängu lõppfaasis toimub punktide arvutamine ning tulemuseks näidatakse mängu lõpp-vaade, ehk "Finish Game view" kus mängijate punktid on näidatud ning on võimalik uut mängu alustada.

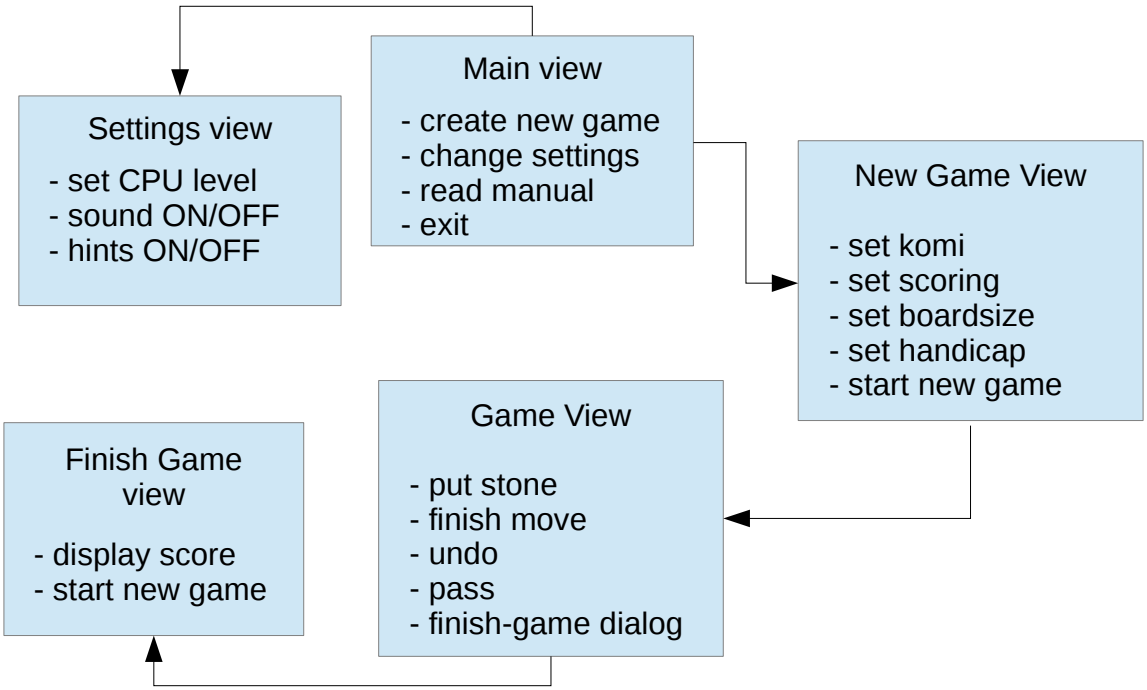



*Diagramm 2: Rakenduse vaadete hierarhia*

## 4.3 Disaini mustrid

Rakenduse objektide struktuur baseerub *MVC*[4] printsiibil.

### 4.3.1 Mudel (Model)

Mudelit kasutatakse informatsiooni hoidmiseks rakenduses. Ta ei sõltu kontrolleritest ja vaadetest. Selles rakenduses mudeli rollis on klassid GoBoard ning GoStone. Nende abil kõik rakendusele vajalik informatsioon on salvestatud.

4 *Model–view–controller (MVC)* on tarkvara arhitektuurne muster, mille põhimõtteks on see, et informatsiooni kuvamine on eraldatud kasutaja tegevustest selle informatsiooniga.

### 4.3.2 Vaade (View)

Vaated koosnevad kasutajaliidese elementidest: nuppudest, piltidest, animatsioonidest. Seda kasutatakse selleks, et kasutajale infot mudelitest näidata. Selle rakenduse kontekstis vaadete rollis on MainView, SettingsView, GoBoardView, GoStoneView, DialogView jms.

### 4.3.3 Kontroller (Controller)

Kontroller on mõeldud selleks, et anda korrektset informatsiooni vaadetele ning uuendada mudeleid uue informatsiooniga. Selles rakenduses kontrolleri rollis on GoBoardController, mille kaudu *Go* mängimine toimub.

# 5. Tarkvara arendusprotsess

## 5.1 Tarkvara arenduse mudelid

On olemas mitmeid tarkvara arenduse mudeleid. Igaühel nendest on oma vaade kuidas tarkvara on vaja arendada nii, et see oleks kiire, kvaliteetne ning loodud õigeks ajaks. Mõned nendest on väga spetsiifilised nt. TDD[5] ja mõned on väga üldised. Lõputöös analüüsitakse kahte väga erinevat mudelit: traditsiooniline ja mitte.

## 5.2 Traditsiooniline tarkvara arendusprotsess

Eestis kasutatakse traditsioonilise tarkvara arendusprotsessina rahvusvahelist standardit ISO 12207[6]. "Standard ei kirjuta ette konkreetset elutsükli mudelit ega tarkvara arendusmeetodit. Standardit järgivate huvipoolte vastutusele jääb elutsükli mudeli valimine tarkvaraprojekti tarbeks ning standardi protsesside, tegevuste ja tööde kajastamine selles mudelis. Huvipoolte vastutusele jääb ka tarkvara arendusmeetodite valimine ja rakendamine ning konkreetse tarkvaraprojekti jaoks kohaste tegevuste ja tööde sooritamine." [ISO 12207] [6]

Aga see standard määrab järgmiseid reegleid:

- Tellija koostab koos analüütikuga detailse ülesandepüstituse, mis on tarkvara arenduslepingu oluline osa.
- Ülesandepüstituse järgi alustab tööd lahenduse arhitekt, kes loob tarkvaralahenduse arhitektuuri. Suure tõenäosusega erineb arhitektuur mingis osas ülesandepüstituses kirja pandust – st tekib viga.
- Pärast arhitekti asub tööle disainer, kes joonistab erinevad ekraanipildid. Siin võib

5 *Test-driven development* - tarkvara arenduse metodoloogia, mis nõuab arendajalt esialgselt testi kirjutamist ning pärast selle testi läbiva funktsiooni kirjutamist.

6 ISO 12207 koostati ISO/IEC2 tehnilise ühendkomitee poolt ning avaldati 1995. aastal. Eesti standardina kinnitas Standardiamet selle 1998. aastal.

samuti tekkida erinevus algsest ülesandepüstitusest.

- Järgneb tarkvara arendusprotsess, kus realiseeritakse lahendus vastavalt loodud arhitektuurile ja disainile. Ka siin võib ilmneda vigu.
- Lõpuks testitakse loodu vastavust ülesandepüstitusele.

## 5.3 Agiilne tarkvara arendus

Agiilne tarkvara arendus on suhtluskeskne ning kasutaja vajaduste rahuldamisele orienteeritud arendusprotsess. Projekti algstaadiumist alates suheldakse tihedalt tulevaste kasutajatega ning muudetakse ja täiustatakse tarkvara vastavalt nende soovidele. Kuna kliendil enamasti puudub väga konkreetne visioon uue tarkvara funktsionaalsuste ja väljanägemise osas, siis on see heaks võimaluseks järk-järgult välja kujundada kasutajale meelepärane tarkvara. See arenduse meetod on kasutusel peamiselt väikestes arendusmeeskondades, kuna see nõuab dünaamilisust, kiiret ümberpaigutumist jne.

Paindliku tarkvaraarenduse eelised:

- Otsuseid saab teha kiirelt kogu arendusprotsessi jooksul.
- Tellijal on selgem arusaam loodava lahenduse tegelikust vastavusest soovitavale.
- Protsessi käigus tekib just selline lõppprodukt, mis on tellijale vajalik.
- Kui selgub, et projektiga pole mõtet edasi minna, saab selle lihtsalt lõpetada, ja kaotus pole nii suur kui tavaprotsessi korral.

## 5.4 Kokkuvõtte

Selleks, et tarkvara arenduse protsess oleks edukas ning tema tulemuseks oleks kvaliteetne toode on vaja sellist protsessi targalt juhtida. Kuna protsessis osaleb ainult üks arendaja ja tarkvara arendatakse mitte müümiseks, siis projekti juhtida on väga raske, sest ainsaks motivatsiooniks projekti arendamiseks on puhas huvi. Ennem koodi kirjutamist sai analüüsitud tarkvara arenduse metoodikad, aga ei leidnud ühtegi sobivamat. Seega autor otsustas luua uue metoodika. Baasiks oli agiilne tarkvara arendus, milles testija, tellija, analüütiku ning arendaja rollis oli üks inimene lisaks arenduse graafikaga tegeles disainer.

## 5.5 Kuidas arendusprotsess pidi olema teoorias

Enne arendust oli koostatud projekti plaan, mis koosnes rangelt püstitatud tähtaegadest ja tehnilistest nõuetest. Lisaks sellele olid riskid hinnatud ja arvesse võetud. Kogu arendusplaan oli jagatud kaheksaks osaks. Iga osa eesmärgiks oli kas kasutajaliidese parandamine või funktsionaalsuse lisamine. Iga osa kestuseks oli täpselt üks nädal(7 päeva). Projekti tööpäev oli mitte 24-tunnine vaid ainult 3-tunnine, sest arendusprotsess oli paralleelne õppetööga.

Kuna tegemist oli uute tehnoloogiate õppimisega paralleelselt õppetööga, siis ilmnesid järgmised riskid:

- Ülesande lahendamine vajab palju aega uute materjalide õppimist ning võtab rohkem aega, kui plaanitud oli.
- Koormus ülikoolis suureneb ning aega projektis tööd tegemiseks ei ole.
- Inimese faktoriga seotud riskid: väsimus, meeleolu või huvi puuduvad jne.

- Selleks et neid riske likvideerida igale osale oli lisatud vähemalt ühe-päevane buffer

ning iga osa koosnes ainult ühest suurest ülesandest vt [Lisa 1].

## 5.6 Kuidas arendusprotsess toimus praktikas

Esimene nädal oli väga produktiivne : kahe päeva jooksul kõik plaanitud tegevused olid ära tehtud. Buffer-ajal alustati kolmanda vt [Lisa 1] nädala esimene ülesande lahendamist, mille kuulutati kogu buffer-aeg. Esimese nädala edukus oli seotud sellega, et Internetis on väga palju õppematerjale kuidas *Android*'ile esimest rakendust luua.

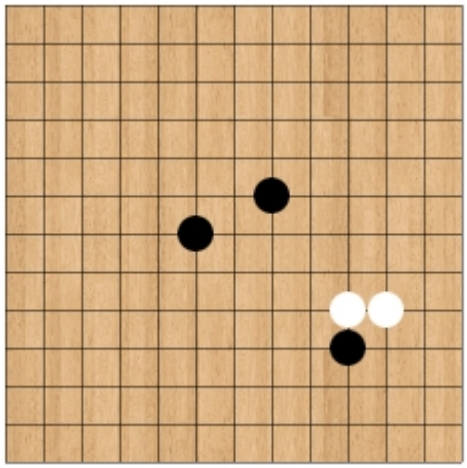
Teise nädala esimene ülesande, ehk *GNU Go* tehisintellekti mootori lähtekoodi uuring võttis väga palju aega. Dokumentatsiooni täieliku puudumise tõttu kestis see terve nädal ja isegi kolmanda nädala alguseks jäi tegemata. Otsustati alustada kolmanda nädala ülesannetega ja neljanda nädala jooksul valmis teha ka tegemata jäänud tööd.

Kolmas nädal nagu esimenegi oli päris kerge. Kõik tööd said tehtud plaani järgi ning kaks buffri päeva oli kasutatud selleks, et lisada *Go* mängimislaua suuruse muutmise võimaluse .

Neljanda nädala alguses projekti seis selline: kasutajaliides oli valmis, aga mingit tehisintellekti rakenduses ei olnud. Neljanda nädala keskel *GNU Go* lähtekood oli aru saadud ning hakati kirjutama JNI kihti tehisintellekti mootorile, et anda võimalust rakenduses tehisintellekti kasutada. See ülesanne võttis neljanda nädala kõik ülejäänud päevad. Neljanda nädala tulemuseks oli rakendus, mille kasutajaliides ja mootor ei olnud veel kuidagi seotud.

Kuna suurem osa kasutajaliidesest oli juba tehtud, otsustati pühendada terve viies nädal rakenduse tehisintellekti mootori ja liidese ühendamisele. Selle protsessi jooksul tuli välja, et on vaja veel kaks funktsiooni JNI kihile lisaks kirjutada: *setRules* ja *initGTP*. Võrreldes peafunktsiooniga – *playGTP* olid need kaks triviaalsed, seega nende kirjutamine võttis ainult

ühe päeva. Ülejaanud päevad olid kulutatud selleks, et rakenduses teha võimalikuks tehisintellekti mootoriga *Go'*d reeglite järgi mängida.



*Pilt 4: Rakenduse kasutajaliides. Esimene versioon.*

Kuuenda nädala ülesanded, mis tegelikult olid viiendast nädalast ülekantud ülesanded, olid edukalt tehtud poole nädalaga. Kuuenda nädala alguseks oli võimalik GTP kaudu tehisintellekti mootoriga suhelda. Selle nädala produktiivse töö tulemuseks oli juba mängimiseks valmis rakendus.

Kuna projekti alguses palju aega oli kulutatud õppimiseks, siis seitsmenda ja kaheksanda nädala ülesanded olid tehtud plaani järgi ja projekt oli edukalt lõpetatud.

## 5.7 Kokkuvõtte

8-nädalalane arendusprojekt oli küll lõppetatud, kuid kahjuks selle tulemuseks ei olnud veel rakendus, mida saaks kasutusele anda ning *Google Play*[7]'i üles panna. See on seotud sellega, et projekti plaan oli päris ebamäärane. Arendusprotsessi jooksul suur osa projekti plaanist oli muudetud ja nende muudatuste tegemine ja plaani korrigeerimine mõjutas lõpp-toote kvaliteeti.

Üks aspekt, mis oli väga tähtis, aga täiesti unustatud on testimine. Alates teisest nädalast hakkasid rakenduses ilmuma defektid : kriitilised neist said kohe lahendatud, aga ülejaanud

7 Google Play – rakenduste pood *Android* platvormile.

jäid avastamata ning hakkasid nii arendust, kui ka rakenduse kasutamist, segama. Seda olukorda oleks võimalik likvideerida testidega, aga ei olnud mõtet neid projekti keskel sisse tuua.

Projekt oli plaanitud valeks ajaks: on ilmne, et töötades täiskohaga ning täiskoormusega õppides on väga raske kõrval-projekti arendada. Suvevaheaeg sobiks selleks paremini.

Kokku oli kirjutatud 2636 *Java* koodi rida, 348 XML-i koodi rida ning 200 C-keele koodi rida. Selle koodi kirjutamine võttis umbes 168 tundi aega.

# Go board game implementation for Android platform

### Bachelor's thesis ( 6 ECTS)


### Sergei Mihhailov


## Summary


The purpose of this bachelor thesis has been fulfilled as the application is ready and provides the user interface for playing Go board game with AI.

Whole development process was carefully managed and analyzed : popular artificial intelligence techniques used in *Go* were taken into account; development tools choice was explained; project plan with most possible risks was developed before the actual development started and whole development process was documented.

The application is developed for Android platform. Eclipse integrated development environment, Android SDK & NDK frameworks were used during development. All tasks planned for this project were successfully implemented, but application requires additional development to be done in order to be published in *Google Play Market*.

## Kasutatud kirjandus


1. Application programming interface [http://en.wikipedia.org/wiki/Application_programming_interface] (29.04.2014).

2. *Go* game. [http://en.wikipedia.org/wiki/*Go*_(game)] (05.05.2014). Pilt 1: Go mängu triviaalsed positsioonid. Pilt 2: Ko-reegel.

3. GNU *Go* [http://www.gnu.org/software/gnugo] (06.05.2014)

4. Android SDK [http://developer.android.com/sdk/index.html] (06.05.2014)

5. Android NDK [http://developer.android.com/tools/sdk/ndk/index.html] (06.05.2014)

6. ISO/IEC 12207:2008 [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43447] (05.05.2014)

7. Goemate [http://senseis.xmp.net/?Goemate] (08.05.2014)

8. The Many Faces Of *Go* [http://www.smart-games.com/manyfaces.html] (08.05.2014)

9. *Go* Intellect [http://www.yutopian.com/go/soft/EAC19.html] (08.05.2014)

10. *Go*++ [http://www.goplusplus.com/] (08.05.2014)

11. Zen [http://senseis.xmp.net/?ZenGoProgram] (08.05.2014)

12. *Leela* [http://senseis.xmp.net/?Leela] (08.05.2014)

13. *MoGo* [http://senseis.xmp.net/?MoGo] (08.05.2014)

14. *Crazy Stone* [http://www.unbalance.co.jp/igo/eng/] (08.05.2014)

15. *Gobble* [http://www.cgl.ucsf.edu/go/Programs/Gobble.html] (08.05.2014)

16. Minimax [http://en.wikipedia.org/wiki/Minimax] (09.05.2014)

17. Alpha-beta [http://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning] (09.05.2014)

18. Android passes one billion activations in 2013 [http://www.engadget.com/2013/09/03/android-kit-kat/ ] (11.05.2014)

19. How smartphones are on the verge of taking over the world [http://www.nydailynews.com/life-style/smartphones-world-article-1.1295927] (11.05.2014)

20. Android, the world's most popular mobile platform [http://developer.android.com/about/index.html] (11.05.2014)

# Lisad

## *Lisa 1. Projektiplaan*

| **Milestone 1. User Interface.Usability** |
|---|
| **1. Create initial application with single Activity class in it.** |
| **2. Implement fixed size *Go* board drawing using Canvas.** |
| **3. Implement fixed color stone placing via touchscreen.** |
| **4. - 7. Buffer.** |
| **Milestone 2. Functionality** |
| **1. - 3. Research GNU *Go* sources.** |
| **4. - 6. Implement ndk-friendly layer on GNU *Go*. (placing stones via GTP.)** |
| **7. Buffer.** |
| **Milestone 3. User Interface. Usability** |
| **1. Allow to place stones only on intersections of the grid on a *Go* board.** |
| **2. Implement dynamic size *Go* board drawing for devices with different screen density.** |
| **3. Implement dynamic color stone placing via touchscreen.** |
| **4. Implement support lines drawing.** |
| **5. - 7. Buffer.** |
| **Milestone 4. Functionality** |
| **1. Implement structure for holding all information about board: size, handicap, stones placed, current move number, information about players, and so on.** |
| **2. - 4. Basic rule implementation: liberty rule, ko rule, seki, organizing stones into groups, “two eyes” rule, stone capturing.** |
| **3. - 6. Implement Japanese scoring system.** |
| **7. Buffer.** |
| **Milestone 5. User Interface. Usability** |
| **1. Implement working main menu with buttons “Start Game”, “Settings”, “Exit”.** |
| **2. Implement “Settings” menu, where komi, scoring system,** |

| |
|---|
| **board size and handicap can be configured.** |
| **3. - 4. Implement board size configuration and drawing + wire this to the “Settings” menu.** |
| **5. Enable “back”button functionality to navigate between application's screens.** |
| **6. Add “Finish Move” button to the main screen of the application.** |
| **7. Buffer.** |
| **Milestone 6. Functionality** |
| **1. Implement komi, scoring system and handicap configuration before actual game start.** |
| **2. - 3. Implement undo functionality during the game** |
| **4. - 6. Wire GNU *Go* to the actual application and make playing with CPU via GTP layer possible.** |
| **7. Buffer.** |
| **Milestone 7. User Interface. Usability** |
| **1. - 2. Improve user interface: add images to button's states (PASSIVE, PRESSED, FOCUSED), add background images to every view in the application.** |
| **3. Add options menu with two buttons: Undo and Pass** |
| **4. - 5. Implement finished-game-dialog which is showed if game has finished. Displays final score, has two buttons Start a New Game and Review finished game.** |
| **6. - 7. Buffer.** |
| **Milestone 8. Functionality** |
| **1. - 2. Make application persist it's state if application is not active** |
| **3. - 4. Implement final score determination & provide all buttons in finished-game-dialog with proper functions.** |
| **5. Application should restore game state after it's been inactive.** |
| **6. - 7. Buffer.** |

**Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks**

Mina Sergei Mihhailov
(*autori nimi*)
(sünnikuupäev: 04.09.1991)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
„*Go* mängu implementeerimine *Android*'i rakendusena“,
(*lõputöö pealkiri*)

mille juhendaja on Vambola Leping,
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **12.05.2014**