ITI0011-2016:Gomoku

Allikas: Kursused

Tagasi kursuse lehele

Tähtaeg: **20. mai 2016**.

Ülesanne täpsustub. Ülesande püstitus on paigas. Punktide osa jm võib täieneda.

Sisukord

Ülesande lühikirjeldus

Ülesandeks on lisada etteantud Gomoku programmile nö mõtlemiskomponent: jupid programmi, mis leiavad etteantud seisust võimalikult hea käigu. Nende juppide lisamise tulemusena peab sinu programm mängima Gomokut nö "mõistliku headuse" tasemel.

Konkreetselt nõutakse, et sinu programm peab rõhuval enamikul juhtudel võitma õppejõu poolt etteantavat Gomoku võrdlusprogrammi, eeldades, et mõlemal on iga käigu jaoks aega üks sekund. Igal juhul on garanteeritud, et:

- kui sinu programm kasutab korrektselt minimaxi
- oled programmeerinud mõistliku lihtsa seisuhinde (vaatab viieste olemasolu, loeb kokku lahtised neljased, poollahtised neljased, lahtised kolmesed)

siis sinu programm võidab etteantud võrdlusprogrammi, kui mõlemal on ajapiir üks sekund käigu kohta.

NB! Sinu programm peab suutma mõistliku headusega mängida nii 10x10 kui 20x20 laual.

Peate kirjutama `StudentStrategy.getMove()` sisu selliselt, et see mängiks piisavalt hästi. Võite sinna faili teha uusi meetodeid. Väliste librarite ja täiendavate klasside kasutamine ei ole lubatud! Ehk siis kogu koodi peate sellesse faili panema.

Kaitsma tulles peab strateegia töötama viimase versiooniga antud rakendusest. Üldiselt midagi olulist teie jaoks ei muutu. Üks konkreetne näide, mis võib muutuda, on ajapiirang. Hetkel füüsilist ajapiirangut rakenduses pole. Süsteem prindib välja tehtud käigud ja selleks kulunud aja. Kui käik võtab 1.1 sekundit, pole otseselt probleemi. Kui käik võtab 2 sekundit või kauem, tuleb miskit ette võtta.

Loe täpsemalt: #Punktide saamine, #Lähtekoodi seletus

Rakendus

Selleks, et ülesannet realiseerida, on teil ette antud GUI koos vastaste strateegiatega. See peatükk kirjeldab rakendust ja selle käimasaamist.

Viimane versioon 0.999 GUI-st: Meedia:Gomoku-v0.999.zip (seisuga 16. mai 2015)

Uuendused: 0.999:

- 2013. aasta võitja kood parandatud (nüüd on lib kaustas kaks jar-faili - üks fail iga eelmise aasta võitja kohta).

0.99:

- dünaamiline mängija laadimine gomoku.strategies kaustast
- 2014. aasta sügiskursuse turniiri võitja vastane
- mängude logi salvestamine ja laadimine

0.5:

- skoori arvestamine (mängijate võidud-viigid-kaotused loetakse kokku)
- esimese mängija käigu aja arvestamine hakkab sellest hetkest, kui laud jms objektid on loodud (ehk siis mõõdetakse reaalselt strateegia failis getMove() meetodi käivitusaega).

0.4:

- 2013. aasta turniiri võitja fix
- lisatud 20x20 laud
- OpponentStrong ja OpponentWeak tehtud lihtsamaks (teevad kiiremini käigu)
- veel mingeid bugifixe

0.3:

- 2013. aasta turniiri võitja (OpponentWinner)

0.2:

- Lisatud vastased OpponentWeak ja OpponentStrong
- Nimetatud vastased pole veel lõplikud. Küll võib arvestada, et 10p saamiseks peate vähemalt OpponentWeak'ist jagu saama. Võib juhtuda, et OpponentWeak ise muutub natuke lihtsamaks.

Käivitamine IntelliJ's

Paki zip-fail lahti. Tee IntelliJ's uus projekt (HW04 näiteks). Kopeeri zip-failis src sisu uue projekti src kausta alla (seal on package gomoku). Kopeeri zip-failis lib kaust projekti kausta (HW04 alla jääb siis src ja lib).

File -> Project structure -> Modules. Vali HW04 moodul (kui muid mooduleid pole, siis peaks see ainuke olema). Ava Dependencies sakk. Paremal üleval roheline "+". Add jars or directories. Otsi üles HW04/lib (terve kaust) ja pane OK.

Nüüd saad Main klassist programmi käima panna.

Käivitamine Eclipse'is

Soovitatav on vaadata kohe alternatiivset käivitamist.

Zip-fail tuleb lahti pakkida näiteks workspace'i sisse (tekib workspace'i alla gomoku kaust). Nüüd eclipse'is file -> import -> General->Existing projects into Workspace -> nüüd valite kausta kettal ja finish. Peaks tekkima projekt nimega "gomoku".

Alternatiivne käivitamine

Kui eelmine meetod ei tööta, siis tuleb failid käsitsi kopeerida:

- Tõmba alla zip-fail
- tee Eclipse'is uus Java projekt
- kopeeri kõik zip-failis "gomoku/src" kaustas olevad failid loodud projekti "src" kausta (loodud projekti asukoha leiad näiteks projekti peal parem klikk - properties -> Resource, seal on "Location", mis näitab asukohta kettal).
- paki zip-failis olev "lib/strat_impl.jar" fail kuskile kettale lahti. Näiteks projekti alla loo uus kaust "lib" ja kopeeri sinna.
- projekti peal parem klikk - properties -> Java Build Path -> Libraries -> "Add External JARs" nupp (kui panid projekti alla, siis võta "Add JARs...")
- otsi avanevas dialoogis strat_impl.jar fail üles ja "OK" või "Open"

Nüüd peaks Main.java failist projekt käivitatav olema.

Andke teada, kui projekti importimine ei õnnestu.

Esimesed sammud

Paketis gomoku.strategies on klass nimega StudentStrategyExample (ehk siis fail projekti kaustas src/gomoku/strategies/StudentStrategyExample.java). Tehke sellest failist koopia, näiteks AgoStrategy.java (muutke ka klassi nimi vastavalt). Näiteklassi ülekirjutamine ei ole soovituslik (kuigi töötab ja otseselt ühtegi probleemi ei tekita).

Kõik klassid, mis gomoku.strategies kausta all on, peaks olema nähtavad rakenduses (saate neid mängijatena valida ja neid testida).

IntelliJ IDE-l tundub olevat probleem meetodiga

```
initializeGameStatusListener()
```

Sest seal kasutuses lambda (täpne error siis):

```
Lambda expressions are not supported at this language level.
```

Stacki juhendamisel läksin siis

```
File -> Project Structure (Ctrl+Alt+Shift+S)
```

ning "Project" tabi alt sai muuta Language Levelit. IntelliJ-l on see vaikimisi 6 peal kuid lambdade jaoks peab 8-ks muutma nähtavasti.

Kui tekib JavaFX-iga viga, mis ütleb midagi keelatud klasside kohta, võib abi olla sellest: http://stackoverflow.com/questions/9266632/access-restriction-is-not-accessible-due-to-restriction-on-required-library#answer-9586411

Kui sul on tekkinud mõni probleem ja oled selle ära lahendanud, palun saada email: ago.luberg @ ttu.ee . Saan siis selle ka siia lisada.

Lisapunktid

Kes leiab mõne vea ja selle jaoks ka koodiparanduse antud rakenduses, võib teenida lisapunkte. Ka on oodatud kõiksugu ettepanekud parendamise osas (kasutajamugavus, väljanägemine, töökindlus, turvalisus jms).

Punktide saamine

Korrektne minimax koos enam-vähem seisuhindamisega (kolmest ja suuremaid oskab takistada) - 2 punkti.

Nõrgema vastase võitmine ilma minimaxita (või mõne analoogse algoritmita) annab 3 punkti.

Nõrgema vastase (OpponentWeak) vastu üle 50% mängudest võidud (6 mängu 10st) annab 4 punkti.

Tugevama vastase (OpponentStrong) vastu üle 50% punktidest (näiteks 9 viiki ja 1 võit) annab 6p.

Tugevama vastase (OpponentStrong) vastu üle 50% mängudest võidud (6 mängu 10st) annab 8 punkti.

Hinde "5" saamiseks tuleb saada vähemalt 4 punkti.

Võistlus ja lisapunktid

NB! Plagiaadi esitamisel saavad selle kõik autorid ülesande eest 0 punkti

Pärast praktikumitöö esitamise tähtaja möödumist korraldab õppejõud programmide vahel võistluse, ning esimese kahe parema programmi autorid saavad eksami jaoks lisapunkte:

- Esikoha võitnud programmi autorid kumbki 10 punkti.
- Teise koha võitnud programmi autorid kumbki 5 punkti.

Võistlus ei ole kohustuslik. Kuidas sellest osa võtta: **20. maiks** (k.a) tuleb saata viimane versioon StudentStrategy.java failist (mida võib soovi korral veel täiendada) õppejõu emaili peale.

Võistlusel võib osaleda kahekesi.

Õppejõule tuleb saata emailiga järgmine info, kusjuures Subject real (emaili pealkiri) peab olema sõna GOMOKU:

- autorite nimed, matriklinumbrid, emailid
- StudentStrategy.java lähtekood

Sisulised nõuded:

- Teie programmi mõtlemiskomponendid peavad olema üleni teie enda kirjutatud. Artikleid lugeda ja teisi programme uurida tohib, neid otse kopeerida aga mitte.
- Kui õppejõul tekib eelmise punkti osas kahtlus, siis ta uurib autoritelt asja

edasi, ning kahtluse püsimisel diskvalifitseerib kahtlase programmi võistluselt.

Tehnilised nõuded:

- Programm ei tohi AK arvutiklassides ühtegi käiku mõelda kauem, kui ühe sekundi.
- Programm peab suutma mängida nii 10x10 kui 20x20 laual.
- Programm peab suutma käia ka siis, kui etteantud laud on tühi (alustaja must).
- Programm saab ette laua seisu ja käigul oleva mängija (ehk siis selle mängija, kellena teie strateegia mängib). Vt täpsemat juhendit sektsioonist #Lähtekoodi_seletus #Lähtekoodi_seletus
- Programm ei tohi kasutada võrguühendust
- Programm ei tohi kasutada väliseid teeke (või siis kui kopeerite lähtekoodina StudentStrategy.java sisse).
- Programm võib kasutada erinevaid lõimeid (thread), kuigi peab arvestama, et arvuti, milles turniir toimub, ei pruugi olla kasutada rohkem kui ühte. Seega, te ei saa eeldada, et teil on multiprotsessoriga arvuti.
- Et välistada suurte andmebaaside kaasapanemist, peab lähtekoodi faili suurus olema maksimaalselt 1 MB (mis tegelikult on päris suur maht, ehk siis ühtteist te sinna saate soovi korral panna).
- Programm ei tohi tekitada mängimise ajal faile kettale (küll aga võib kasutada mälu "piiramatult", arvestades, et mälu võib olla limiteeritud mõnesaja megabaidiga).
- Võistlus kahe vastase vahel toimub reeglina ühe programmi käivitamise jooksul. See tähendab seda, et kui vastased mängivad 10 mängu, siis kõik need mängud käivitatakse sama sessiooni jooksul. StudentStrategy objekt instantsieeritakse igal mängul. Küll aga saate staatiliste muutujatega salvestada infot läbi erinevate mängude. Seega on teil võimalik näiteks õppida eelmistest mängudest.

Õppejõud korraldab laekunud programmide vahel turniiri, kus edasi pääseb igast matšist parem. Kaks programmi pannakse omavahel mängima, vaheldumisi mustade ja valgetega, kuni üks programm jõuab teisest 2 punkti võrra ette (kaotus ja viik 0 punkti, võit 1 punkt). Mängitakse 10x10 laual. Kui kümne partiiga võitjat ei selgu, mängitakse edasi 20x20 laual. Kui seal ka kümne partiiga võitjat ei selgu, valitakse võitja loosiga.

Õppejõud jätab õiguse teha korralduse poolel muudatusi. Sõltuvalt esitatud tööde arvust, võib korraldada ka alagruppidega turniiri, kus kohe esimese mängu kaotamise korral välja ei kuku. Aga täpsemat infot saab jagada siis, kui on teada, palju osalejaid on.

Kuidas minimaxi ja seisuhinde arvutamist programmeerida

Alusta sellest, et tee valmis lihtne minimax (ilma alpha-betata) ja lihtne seisuhindaja, kus vaadatakse ainult seda, kas mõni osapooltest on võitnud: selleks võid vaadata praktikumides tehtud seisuhindamisi/minimaxi ja .

Debugi see programm ära, mängides talle ise vastu. Tee katseid nii otsinguga sügavuseni kaks, kolm, neli jne, kuni otsiaeg läheb liiga suureks. Uuri lihtsalt, kas programm leiab üles ühekäigulise võidu, kahekäigulise võidu jne, ning kas ta suudab blokeerida sinu ühekäigulise võidu, kahekäigulise võidu jne.

Seejärel asu täiustama seisuhindajat. Peamised ideed:

- Kas on viiest laual (otsene võit)?
- Loe kokku üleni lahtised neljased mõlema jaoks.
- Loe kokku poollahtised neljased mõlema jaoks.
- Loe kokku üleni lahtised kolmesed mõlema jaoks.
- Võibolla ka: Loe kokku poollahtised kolmesed.
- Võibolla ka: Loe kokku lahtised kahesed.

Seisuhinne arvuta kokku nendest parameetritest, pannes võimsamatele ähvardustele kõvema koefitsiendi (üleni lahtine neljane on praktiliselt juba võit, lahtine kolmene aga veel mitte). Mõistlik võib olla viimati käija värvi ja järgmisena käjia värvi ähvardustele erinevate koefitsientide panemine: järgmisena käija ähvardused on hullemad (kui ta juba kaotanud pole)!

Debugi täiustatud seisuhinne ära, mängides jälle ise vastu.

Tee katseid aja-arvestusega ja määra otsingusügavus selline, et AK klassides kunagi ei mõeldaks üle ühe sekundi (NB! 20x20 laual on otsisügavus ilmselt väiksem, kui 10x10 laual).

Nüüd peaks sinu programm olema piisavalt OK, võitmaks võrdlusprogrammi.

Ideid, kuidas programmi veel palju paremaks teha:

- Vii sisse alpha-beta (modifitseeri minimaxi). Programm peaks leidma sama käigu, mis minimaxiga, aga rutem.
- Sundkäikude puhul (neljase sulgemine, võibolla ka lahtise kolmese otsa sulgemine) ära otsi kõiki muid alternatiive läbi, vaid vaata ainult sundkäike.
- Vali võimalikeks käikudeks ainult ruute, mis on mõne täidetud ruudu ligidal.
- Tee seisuhindamist selliselt, et enne lõppsügavusele jõudmist tehakse vahepealse seisu seisuhinne ja antakse see lõppsügavusele ette. Lõppsügavuse seis erineb aga ainult ühe välja poolest, seega saab hinde arvutada nii, et vaatad läbi ainult viimase käigu ümbruskonna ja modifitseerid üks aste kõrgemal saadud seisuhinnet vastavalt.
- Sundkäikude puhul, kus ainult üks valik, analüüsi sügavamale: unikaalne sundkäik ju ei laienda puud! Mh otsi sundkäikude ahelad lõpuni välja.
- Loe läbi lisaviited #Lisaviited

Lähtekoodi seletus

Tuleb implementeerida interface'i:

```
package gomoku;

/**
 * @author Ago
 *
 * Interface for computer strategy.
 */
public interface ComputerStrategy {
        /**
         * Takes the game state and return the best move
         * @param board Board state
         * @param player Player indicator. Which player's
         * strategy it is. Possible values: SimpleBoard.PLAYER_*.
         * @return A location where to make computer's move.
         *
         * @see SimpleBoard
         * @see Location
         */
        public Location getMove(SimpleBoard board, int player);

        /**
         * Name will be shown during the play.
         * This method should be overridden to
         * show student's name.
         * @return Name of the player
         */
        public String getName();
}
```

Objekt, mis antakse "getMove" meetodile kaasa:

```
package gomoku;

/**
 * Simple 2-dimensional presentation
 * of the game board.
 * Every cell in the board is represented
 * by one integer, the Values are either
 * PLAYER_BLACK, PLAYER_WHITE or EMPTY.
 * This object also knows the size of the board
 * and the last move (Location object).
 *
 * @author Ago
 * @see Location
 */
public class SimpleBoard {

        /**
         * Cell value for black player's piece
         */
        public static final int PLAYER_BLACK = 1;

        /**
         * Cell value for white player's piece
         */
        public static final int PLAYER_WHITE = -1;

        /**
         * Empty cell value
         */
        public static final int EMPTY = 0;

        /**
         * The height of the board.
         * Indicates the number of rows.
         */
        private int height = -1;
        /**
         * The width of the board.
         * Indicates the number of columns.
         */
```

```
        private int width = -1;

        private int[][] board;

        /**
         * Returns the height (number of rows)
         * of the board.
         * @return Number of rows
         */
        public int getHeight() {
                return height;
        }

        /**
         * Returns the width (number of columns)
         * of the board.
         * @return Number of columns
         */
        public int getWidth() {
                return width;
        }

        /**
         * Returns 2-dimensional
         * array of integers with values
         * PLAYER_WHITE, PLAYER_BLACK or EMPTY.
         * The values correspond to
         * White player's piece,
         * Black player's piece,
         * or an empty cell accordingly.
         * @return
         */
        public int[][] getBoard() {
                return board;
        }

        /**
         * Constructor to instantiate the board.
         * @param simpleBoard 2-dimensional
         * array for the board.
         */
        public SimpleBoard(int[][] simpleBoard) {
                height = simpleBoard.length;
                if (height > 0) width = simpleBoard[0].length;
                board = simpleBoard;
        }
}
```

Teil tuleb "getMove" meetodis tagastada käigu asukoht Location objektina:

```
package gomoku;

/**
 * @author Ago
 * Location on the board, 0-based.
 */
public class Location {
        /**
         * Index of the row.
         */
        private final int row;
        /**
         * Index of the column.
         */
        private final int column;

        public Location(int row, int column) {
                this.row = row;
                this.column = column;
        }

        /**
         * @return Row index
         */
        public int getRow() {
```

```
                return row;
        }

        /**
         * @return Column index
         */
        public int getColumn() {
                return column;
        }

        @Override
        public String toString() {
                return String.format("(%d, %d)", row, column);
        }
}
```

Näidis random strateegia:

```
/**
 * A random strategy implementation
 * for the gomoku game.
 * @author Ago
 *
 */
public class RandomStrategy implements ComputerStrategy {

        @Override
        public Location getMove(SimpleBoard board, int player) {
                int[][] b = board.getBoard();
                while (true) {
                        // let's loop until we find an empty spot
                        int row = (int)(Math.random() * board.getHeight());
                        int col = (int)(Math.random() * board.getWidth());
                        if (b[row][col] == SimpleBoard.EMPTY) {
                                // if empty, let's return this location
                                return new Location(row, col);
                        }
                }
        }

        @Override
        public String getName() {
                return "Random computer strategy";
        }

}
```

Tudeng peab looma gomoku.strategies paketi alla uue klassi, kasutades näidisena **StudentStrategyExample** klassi. Näiteks: **AgoStrategy.java** (see on täpselt sama kood, mis näiteklassis):

```
package gomoku.strategies;

public class AgoStrategy implements ComputerStrategy {

        @Override
        public Location getMove(SimpleBoard board, int player) {
                // let's operate on 2-d array
                int[][] b = board.getBoard();
                for (int row = b.length - 1; row >= 0; row--) {
                        for (int col = b[0].length - 1; col >= 0; col--) {
                                if (b[row][col] == SimpleBoard.EMPTY) {
                                        // first empty location
                                        return new Location(row, col);
                                }
                        }
                }
                return null;
        }

        @Override
```

```
        public String getName() {
                return "Ago I";
        }

}
```

Praegune tudengi kood hakkab alt paremalt järjest nuppe mööda rida käima (kui on vaba koht).

Lisaviited

Loengu-/praktikumimaterjalid eelmisest aastast: Loeng minimax, alpha-beta, ITI0011:praktikum 13, ITI0011:praktikum 14

Lisaks:

- http://en.wikipedia.org/wiki/Gomoku
- http://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html
- http://web.archive.org/web/20040607185428/www.cs.vu.nl/~victor/thesis.html
- http://www.cs.ualberta.ca/~chinook/

Pärit leheküljelt "https://courses.cs.ttu.ee/w/index.php?title=ITI0011-2016:Gomoku&oldid=4539"

- Viimane muutmine: 16:27, 15. mai 2016
- Seda lehekülge on külastatud 1359 korda.