Poglavje 5

Seminarska naloga 1

5.1 Smart Games – IQ Fit

Pri prvi seminarski nalogi se bomo ukvarjali s kombinatoričnim problemom polaganja kosov nepravokotniških oblik v ravnino. Preden natančno definiramo problem, najprej predstavimo motivacijo iz sveta učljivih igrač, na kateri je problem zasnovan. Predstavimo tudi nekatere možne podatkovne strukture, s katerimi lahko problem predstavimo. Ogledamo si tudi nekatere sorodne probleme, ki utegnejo služiti kot pomoč in navdih pri zasnovi rešitve.

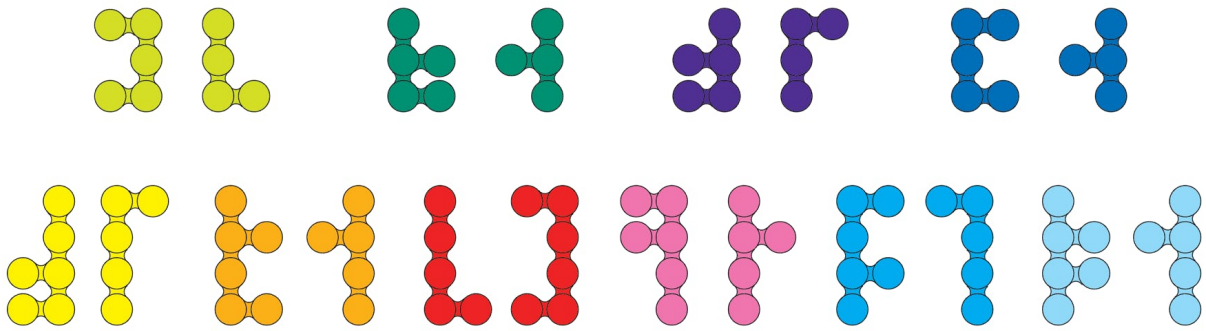
5.1.1 Uvod

Smart je belgijsko podjetje, ki se že več kot 25 let ukvarja z izdelavo učljivih igrač. Njihove glavne blagovne znamke so tri: *SmartGames*, *GeoSmart* in *SmartMax*. *SmartGames* je zbirka več kot 80. večnivojskih logičnih iger, med katerimi je tudi igra *IQ Fit*, ki jo je zasnoval Raf Peeters.

5.1.2 Zasnova igre *IQ Fit*

Igro sestavljata igralna plošča z luknjicami in nabor sestavnih kosov. Cilj je položiti vse sestavne kose na ploščico tako, da so prekrite vse luknjice. Originalna igra vsebuje ploščo velikosti 5 x 10 luknjic, sestavnih kosov pa je 10.

Posebnost igre je, da je igralna plošča dvodimenzionalna, medtem ko so sestavni kosi tridimenzionalni. Sestavni kosi so zgrajeni iz kroglic, povezanih v en kos. Na igralno ploščo jih je vedno potrebno položiti tako, da so prekrite vse luknjice in da položeni kosi nad igralno ploščo tvorijo ravnino, brez štrlečih delov in lukenj. Zaradi tridimenzionalne oblike imajo sestavni kosi šest diskretnih prostostnih stopenj. Z rotacijami je sestavne kose mogoče obrniti tako, da bo v ravnini nad igralno ploščo vidnih 4 ali 5 oz. 5 ali 6 kroglic za posamezni sestavni kos (slika 5.1).



Slika 5.1: Tlorisni pogled na sestavne kose, ki so na sliki obrnjeni tako, da v vidnem delu tvorijo ravnino, na igralno ploščo pa so pritrjeni z eno oz. dvema kroglicama, odvisno od njihove rotacije. Čeprav slika prikazuje 20 kosov, jih je v resnici samo 10 – par sestavnih kosov, ki je obarvan z isto barvo, predstavlja isti sestavni kos v različnih rotacijah. Poleg oblike sestavnega kosa rotacija vpliva tudi na število luknjic, ki jih posamezen kos nad igralno ploščo prekrije. V zgornji vrstici so sestavni kosi, ki prekrijejo 4 oz. 5 lukenj, v spodnji pa jih prekrijejo 5 oz. 6.

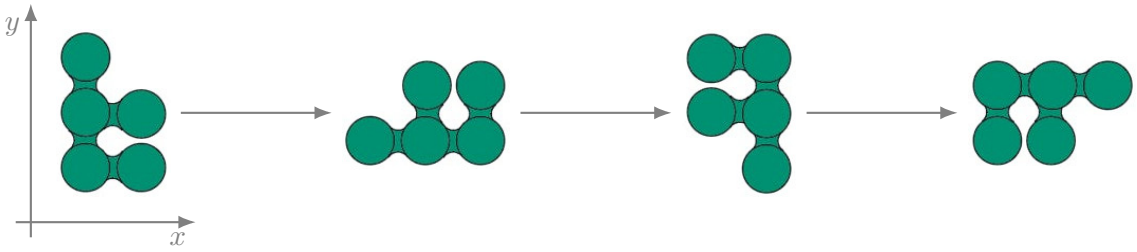
Rotacije in zrcaljenja sestavnih kosov

Sestavne kose je v osnovi mogoče vrteti v 6 prostostnih stopnjah. Rotacije, ki v tlorisnem pogledu sestavnim kosom "spreminjajo obliko", in na sliki 5.1 predstavljajo par sestavnih kosov v isti barvi, se prevedejo na **izbiro ene izmed dveh oblik v paru** (slika 5.2).

Slika 5.2: Izbira sestavnega kosa v paru z isto barvo. Od obeh že podanih oblik, ki so v fizični igri dobljene s 3D rotacijami, izberemo natanko eno, zato se te rotacije prevedejo oz. poenostavijo na izbiro ene izmed obeh možnih oblik.

Od rotacij ostanejo še 2D rotacije okrog osi, ki pravokotno prebada ravnino igralne plošče. Naj bo xy ta ravnina. Za vsak sestavni kos so v ravnini xy možne **štiri orientacije**, katere so

poravnane z osema x in y, med orientacijami pa prehajamo z **rotacijami za 90 stopinj**, kot prikazuje slika 5.3.



Slika 5.3: Možne orientacije sestavnih kosov. Od leve proti desni novo orientacijo dobimo z vrtenjem za 90 stopinj v nasprotni smeri urinega kazalca okrog osi z (ni prikazana), pravokotne na ravnino xy.

Naj opozorimo, da je pri določenih sestavnih kosih, z rotacijami ali brez njih, mogoče dobiti enake sestavne kose. Primer prikazuje slika 5.4.

Slika 5.4: Primera podvojenih (simetričnih) sestavnih kosov, do katerih se pride z rotacijo (desni primer) ali brez nje (levi primer). Ker v obeh primerih sestavna kosa pripadata različnemu paru (sta različno obarvana), obdržimo oba kosa.

Zrcaljenja sestavnih kosov niso možna, saj bi to v ravnini xy povzročilo izbokline. Lahko pa so podani primeri sestavnih kosov, ki so simetrični na zrcaljenje, kot prikazuje primer na sliki 5.5.

Slika 5.5: Primer zrcalno simetričnih sestavnih kosov. Sestavnih kosov ne zrcalimo, vseeno pa je lahko takšen simetričen par že podan. Obdržimo oba kosa.

5.2 Sorodni problemi in strategije reševanja

Polaganje ploščic oz. poliformov raznoraznih oblik je precej dobro preštudiran problem. Standardno ime za poliforme, ki so geometrijski ravninski liki, sestavljeni iz skupka enakih kvadratkov, povezanih v večji lik, je poliomino (ang. *polyomino*). Beseda je skovanka iz dveh besed, *polyform* in *domino*, njen izumitelj pa je matematik Solomon W. Golomb [1]. Glede na število kvadratov, ki tvori poliform, poznamo posebna imena, kot so omino (1 kvadrat), domino (2 kvadrata), tromino (3 kvadrati), tetromino (4 kvadrati), pentomino (5 kvadratov), heksomino (6 kvadratov), itd.

Od vseh naštetih je morda najbolj študiran poliform pentomino, lik, sestavljen iz petih kvadratov. Če teh pet kvadratov povežemo tako, da se med pari povezanih kvadratov robovi stikajo, je

vseh možnih različnih likov, dobljenih na ta način, 12, oz. 18 z vsemi zrcaljenji. Glede na obliko črke, na katero spominjajo, jih je Golomb poimenoval po črkah. Standardni problem je potem polaganje pentominov v ravnino v obliki pravokotnika, sestavljenega iz 60 polj ($60 = 5 \cdot 12$), ali kvadrata iz 64 polj, ki ima na sredini luknjo velikosti 4 polj.

Običajen pristop za reševanje tovrstnih problemov je sestopanje, ki je drug izraz za preiskovanje najprej v globino. Iskanje se prične v korenu, kar predstavlja začetno stanje (prazna plošča), nato pa se glede na možne naslednje poteze v danem trenutku razveji na več možnih podproblemov. Recimo, da izberemo enega izmed 12 pentominov, ki ga najprej v eni izmed 8 možnih orientacij položimo na ploščo. Nato vzamemo drug pentomino, ki ga zopet v eni izmed 8 možnih orientacij položimo v prosta polja na plošči. To ponavljamo, dokler bodisi ne najdemo rešitve bodisi pridemo v stanje, v katerem ni mogoče najti rešitve. V vsakem primeru potem sestopimo korak nazaj in tam poskusimo neko drugo možnost kot prej, tj. drug vrstni red pentominov, drugo orientacijo ali drugo pozicijo na plošči. To tvori t.i. iskalno drevo. Iskanje najprej v globino deluje po tem postopku. Tak pristop je smiseln, ker je problem polaganja likov v pravokotnik NP-poln problem [3].

Dana Scott se je v 60. letih prejšnjega stoletja ukvarjal s problemom polaganja pentominov v kvadrat velikosti 64 polj, za kar je uporabil program s sestopanjem [4]. Njegov program je izpisal vse možne rešitve polaganja ploščic. Da je zmanjšal število vseh možnosti, ki jih program preišče, je pri sestopanju uporabil naslednjo hevristiko. Eden izmed pentominov ima obliko črke X (ali znaka plus). Scott je opazil, da je lik X v vseh možnih orientacijah in zrcaljenjih enak, zato je polaganje ploščic začel s tem likom. Poleg tega je ugotovil, da so za lik X možne samo tri pozicije na plošči, kar opazno zmanjša število vseh možnosti. Podobno je lik v obliki črke P pri enem od dveh možnih zrcaljenjih enak eni izmed rotacij, kar potem 8 možnih orientacij zmanjša na 4.

Golomb in Baumart sta uporabila drugačno hevristiko, in sicer sta na vsakem delu procedure sestopanja iskala najprej tam, kjer je bilo v nekem stanju videti najmanj možnih podproblemov [2]. Podobno hevristiko je uporabil tud Donald E. Knuth, ki pa je bil prvi, ki je problem polaganja likov videl kot poseben primer natančnega prekrivanja [3].

Pri problemu polaganja likov je mogoče izkoriščati tudi simetrije. Če najdemo neko rešitev, lahko ploščo zavrtimo ali zrcalimo, in dobimo še drugo rešitev. Med preiskovanjem različnih možnosti nam to lahko pomaga tudi, če nismo našli rešitve.

Nenazadnje lahko dandanes izkoristimo večprocesorske računalniške arhitekture, tako da različni procesorji sočasno preiskujejo različne možnosti, kar nam lahko pospeši iskanje za nek konstanten faktor, idealno sorazmeren s številom procesorjev.

5.3 Možne podatkovne strukture

V tem poglavju na kratko predstavimo nekatere možne podatkovne strukture za predstavitev problema. Predstaviti je potrebno igralno ploščo, na katero polagamo like, kot tudi same like. Predpostavimo, da je plošča v obliki pravokotnika.

5.3.1 Igralna plošča

2D tabela. Verjetno za mnoge najbolj intuitivna možna predstavitev igralne plošče je 2D tabela, katere indeksi predstavljajo x in y koordinate. Na začetku, ko je plošča prazna, bi imeli v polja zapisane 0. Ko polagamo plošče, v polja zapisujemo neničelna števila, ali pa kar npr. črke likov, na katere njihova oblika spominja. Slednje nam tudi olajša interpretacijo izpisane plošče. Tipično je pri tovrstnih predstavitvah koordinatni sistem po osi y obrnjen, torej y narašča navzdol in pada navzgor. Spodnja koda v Javi prikazuje, kako se lahko sprehodimo po plošči dimenzije $w \times h$, če si elementi sledijo v redu najprej vrstica. Slabost: iskanje prostih mest zahteva prelet tabele.

```
// row-major order
int[][] board = new int[h][w];
for (int y = 0; y < h; y++) {
    for (int x = 0; x < w; x++)
        System.out.print(board[y][x] + " ");
    System.out.println();
}
```

1D tabela. Povsem sorodna podatkovna struktura 2D tabeli je 1D tabela, kjer koordinate preračunavamo v enodimenzionalne indekse. Zapomnimo si širino (w) in višino (h) igralne plošče. Koordinato (x, y) v indeks i preslikamo tako:

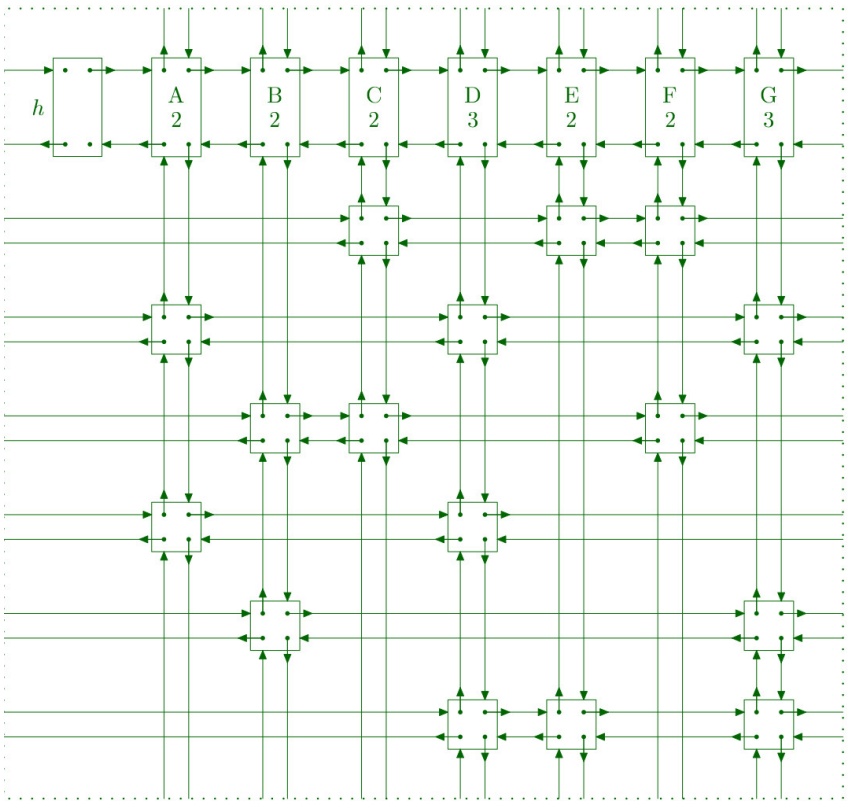
$$i = y \cdot w + x. \tag{5.1}$$

V obratni smeri: $y = \lfloor \frac{i}{w} \rfloor$, $x = i - \lfloor \frac{i}{w} \rfloor \cdot w$. Slabost: iskanje prostih mest zahteva prelet tabele.

Alternativna predstavitev z 1D ali 2D tabelo je, da vloge indeksov in vrednosti zamenjamo. V tem primeru indeks vsake vrstice ustreza enemu izmed likov. Število stolpcev je enako številu kvadratkov, iz katerih je lik sestavljen (5 pri pentominu). Vrednost k v tabeli z indeksom (i, j) pomeni, da j-ti kvadratek i-tega lik je umeščen na koordinato k. Ta vrednost, k, lahko učinkovito zapakira koordinati (x, y). Recimo z bitnimi operacijami `k = x « 16 | (y & ((1 « 17) - 1))` ali pa po (5.1). Prednost: hitro lahko najdemo, kateri lik ni bil uporabljen. Slabost: iskanje prostega mesta na plošči je neučinkovito; število stolpcev določa, koliko kvadratkov sestavlja lik. V našem primeru (pri IQ Fit) to število variira 4–6.

Bitno polje. Polje lahko predstavimo z biti. To je analogno 1D tabeli, le da lahko hranimo samo dve vrednosti, 0 in 1. Ideja je, da se vse operacije, ki jih izvajamo nad poljem, npr. iskanje prostih mest (tj. kateri biti so 0), položitev lika v polje, odstranitev lika iz polja, rotacije, ... izrazi z bitnimi operacijami. Prednost: bitne operacije so običajno zelo hitre, kar lahko pripomore k hitrejšemu izčrpnemu preiskovanju možnosti; majhna poraba pomnilnika; hitro kopiranje stanja. Slabost: običajno težko razumljiva koda, nefleksibilnost pri spremembi dimenzij plošče in/ali likov, hrani lahko samo vrednosti 0 in 1, zato običajno rabimo več bitnih polj.

Četverno povezan krožni seznam. Knuth je v svojem delu [3] uporabil podatkovno strukturo četverno povezan krožni seznam, ki tvori 2D mrežo. Podatkovna struktura vsebuje stolpce, po katerih se lahko preko povezav premikamo gor in dol (krožno). Vezni členi so vozlišča, ki pa so povezani tudi v vrstice, torej ne le gor in dol, pač pa tudi levo in desno. Prednosti: hitro popravljanje strukture – dodajanje in brisanje vozlišč; hitro iskanje nezasedenih mest; naravna uporaba v algoritmih s sestopanjem – zapomniti si moramo le vrednost v vozlišču. Slabost: nekoliko zahtevnejša implementacija.



Slika 5.6: Četverno povezan krožni seznam [3].

5.4 Definicija problema pri seminarju

Pri prvi seminarski nalogi boste izdelali algoritem in podatkovne strukture za igranje igre IQ Fit. Podani bodo liki, ki nastopajo v parih glede na barvo. Izmed vsakega para boste uporabili natanko en lik. Cilj je najti takšen nabor likov, njihovih postavitev in orientacij, da boste (čimbolj) napolnili igralno ploščo. Plošča bo podana v obliki pravokotnika. Operacije nad liki so rotacije okrog z osi, vendar ne zrcaljenja.

Ocenjevalni algoritem. Vsak program bo imel na voljo omejen čas reševanja. Ocenjevalni algoritem bo pripravil vhodne podatke problema, pognal vaš program in mu posredoval vhodne podatke. Po preteku časa bo ocenjevalni algoritem pregledal izhod iz programa in ga ovrednotil. Najboljša možna rešitev je, če se uporabi natanko en lik vsake barve, ki so pravilno položeni na ploščo, tako da ni lukenj in prekrivanja. Takšna rešitev dobi vse možne točke. Vsaka rešitev, ki odstopa od tega, dobi manj točk, in sicer sorazmerno s številom pokritih polj (lukenj) na plošči.

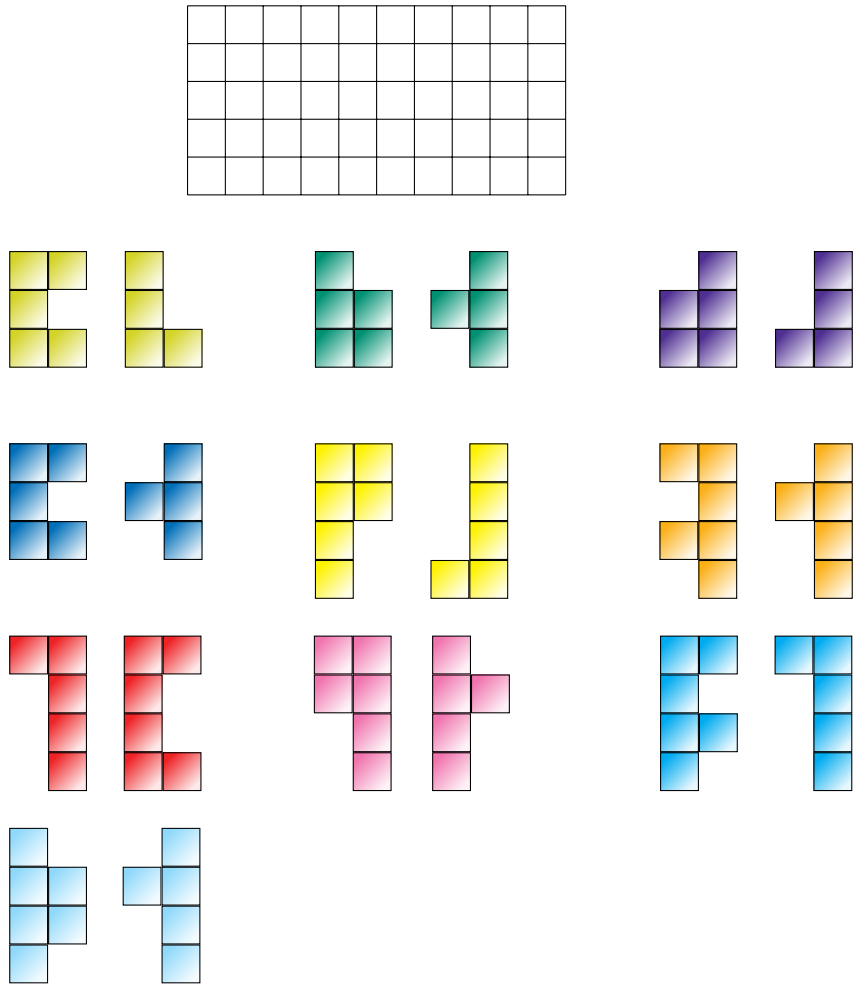
Torej, če že algoritem ne najde natančnega prekrivanja, naj najde takšnega, ki pokrije čimveč polj. V nobenem primeru pa ne smeta biti uporabljena oba lika iz istega para (tj. isto barvo). Takšna rešitev je neveljavna in dobi 0 točk.

Vhod. Podatki bodo podani preko standardnega vhoda (`System.in` v Javi). V prvi vrstici bodo podana tri naravna števila, ločena s po enim presledkom, `w h k`. Parametra w in h označujeta dimenziji igralne plošče, tj. njeno širino in višino. Parameter k označuje število parov podanih likov. Nato sledi $2 \cdot k$ vrstic, po ena za vsak podan lik.

Vsaka vrstica i, ki opisuje lik, je opredeljena s peterko (n, c, w_i, h_i, s_i). Ti parametri so ločeni s po enim presledkom. Parameter n označuje ime lika in je dolžine en znak. Parameter c predstavlja ime barve (ali para) in je prav tako dolžine en znak. Naravno število w_i označuje širino lika, merjeno v številu kvadratkov. Naravno število h_i označuje višino lika, merjeno v številu kvadratkov. Končno, s_i je niz enic in ničel dolžine $w_i \times h_i$, ki opisuje obliko lika, in sicer v mreži dimenzije $w_i \times h_i$ 1 pomeni poln kvadratek, 0 pa prazen (oz. da ga ni).

Primer: IQ Fit, kot predstavljen v poglavju 5.1.2, ima naslednje parametre. Igralna plošča je dimenzije 5×10. Število likov, ki jih polagamo, je 20, oz. 10 parov. V prvi vrstici so zato podana števila 5 10 10. Nato sledi $2 \cdot k = 20$ vrstic, po ena za vsak podan lik.

```
10 5 10
c Y 2 3 111011
l Y 2 3 101011
b G 2 3 101111
t G 2 3 011101
d P 2 3 011111
j P 2 3 010111
c B 2 3 111011
t B 2 3 011101
P y 2 4 11111010
J y 2 4 01010111
7 o 2 4 11011101
1 o 2 4 01110101
L r 2 4 11010101
C r 2 4 11101011
9 p 2 4 11110101
I p 2 4 10111010
F b 2 4 11101110
L b 2 4 11010101
D c 2 4 10111110
1 c 2 4 01110101
```

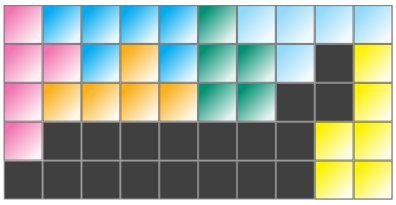


Izhod. Program naj na standardni izhod izpiše stanje igralne plošče. Stanje igralne plošče lahko izpišete večkrat, ocenjevalni algoritem bo prebral vse in upošteval najboljšo rešitev.

Izhod naj bo formatiran na naslednji način. Pri velikosti igralne plošče $w \times h$ naj algoritem izpiše h vrstic. Posamezna izpisana vrstica naj ustreza stanju plošče v isti vrstici. Izpisanih naj bo w stolpcev, ki so ločeni s presledkom. Vsak stolpec naj bo širine 2. Izpisana vrednost v vrstici y in stolpcu x naj bo stanje plošče v koordinati (x, y), in sicer `nc`, kjer je n ime lika, c pa njegova barva. Če v koordinati (x, y) ni lika, naj izpiše dve ničli: `00`. Če program večkrat izpiše stanje, naj se novo stanje začne v novi vrstici pod prejšnjim. Med dvema izpisoma stanj so lahko vmes tudi prazne vrstice. Zadostuje izpis enega stanja, a poskrbite, da bo najboljše.

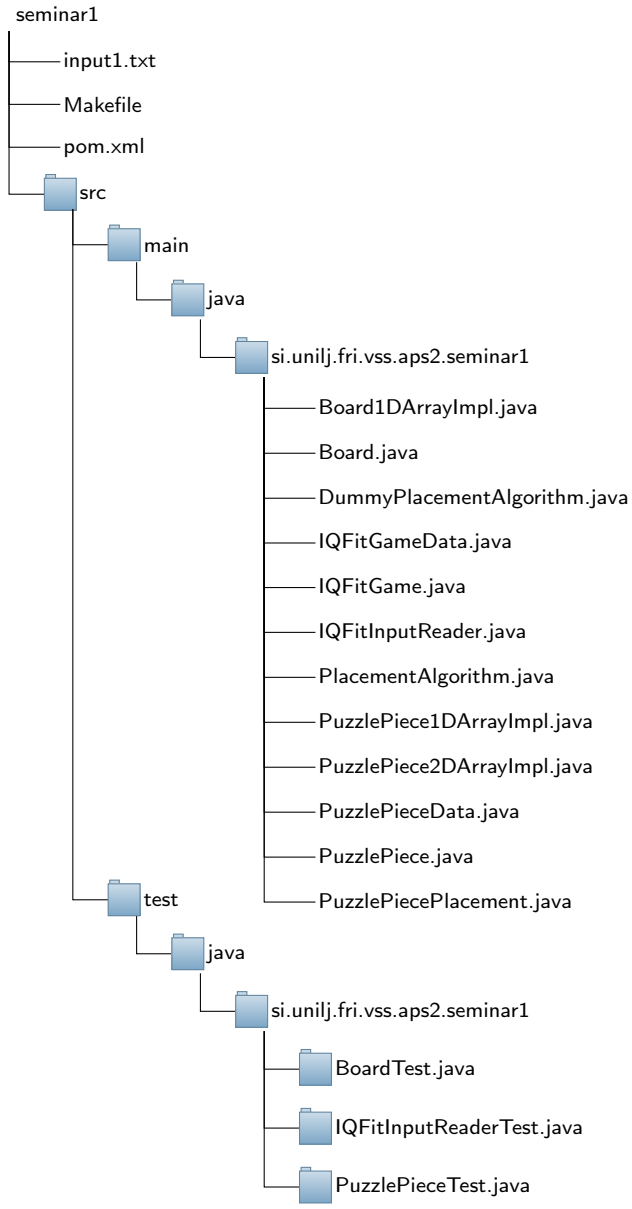
Primer:

```
Ip Fb Fb Fb Fb bG 1c 1c 1c 1c
Ip Ip Fb 1o Fb bG bG 1c 00 Py
Ip 1o 1o 1o 1o bG bG 00 00 Py
Ip 00 00 00 00 00 00 00 Py Py
00 00 00 00 00 00 00 00 Py Py
```



5.5 Struktura predloge

V pomoč pri izdelavi seminarja je pripravljena predloga `seminar1.zip`, ki vsebuje vse potrebne komponente seminarja, napisane v programskem jeziku Java. Datotečna struktura predloge je naslednja:



Vrhnji imenik `seminar1` vsebuje tri datoteke: `pom.xml`, `Makefile` in `input1.txt`. Datoteka `pom.xml` je konfiguracijska datoteka orodja Apache Maven[1]. `Makefile` je konfiguracijska datoteka orodja GNU make[2]. Datoteka `input1.txt` vsebuje primer vhoda v program.

Drevesna struktura sledi priporočilom orodja Apache Maven. Kot je razvidno iz strukture, so vse izvorne datoteke `*.java` v paketu `si.unilj.fri.vss.aps2.seminar1`. Imenik `seminar1/src`

[1] `https://maven.apache.org/`
[2] `https://www.gnu.org/software/make/`

vsebuje dva podimenika: `main` in `test`. V prvem (`main`) se nahajajo vse datoteke programa. V drugem (`test`) se nahajajo enotni testi (ang. *unit tests*), ki preverijo pravilnost nekaterih implementiranih razredov in metod.

Imenik `test` in vso vsebino pod njim se lahko brez škode pobriše. Boljše pa je, da prispevate še svoje teste. Da pa so testi uporabni, datoteka `pom.xml` vsebuje odvisnost na paket `org.junit`, ki jo Apache Maven pridobi in namesti sam.

5.5.1 Vsebina paketa `si.unilj.fri.vss.aps2.seminar1`

Glavni razred predstavlja `si.unilj.fri.vss.aps2.seminar1.IQFitGame`, ki vsebuje metodo `main` in je zato vstopna točka programa. Ta je tudi ustezno nastavljena v konfiguracijski datoteki `pom.xml`.

Razred `IQFitInputReader` prebere podatke s standardnega vhoda. Pri tem uporabi razreda `IQFitGameData` in `PuzzlePieceData` kot podatkovni model. Ta model se uporabi, da se kreira igralno ploščo `Board` in posamezne like `PuzzlePiece`, ki jih želimo položiti na ploščo.

`Board` in `PuzzlePiece` sta abstraktna razreda, ki implementirata samo določene metode. Njune končne implementacije so `Board1DArrayImpl` in `PuzzlePiece1DArrayImpl` oz. `PuzzlePiece2DArrayImpl`, ki dokončno določijo in implementirajo uporabljeno podatkovno strukturo (enodimenzionalno oz. dvodimenzionalno tabelo).

Razred `PuzzlePiecePlacement` vsebuje podatkovni model, ki predstavlja položitev enega lika na igralno ploščo z namenom, da se da stanje igralne plošče povrniti na prejšnje stanje. Abstraktni razred `PlacementAlgorithm` je osnova algoritma za polaganje likov na igralno ploščo, njena konkretna implementacija z zelo preprostim požrešnim algoritmom je `DummyPlacementAlgorithm`. Pričakuje se, da bo vaša implementacija presegla rezultat tega algoritma.

Predlagane datoteke služijo samo kot pomoč in jih ni potrebno uporabiti. **Obvezno** pa uporabite `IQFitGame` kot vstopno točko v program (sicer ustrezno popravite vstopno točno v datoteki `pom.xml`).

5.5.2 Uporaba orodja Apache Maven

Kot omenjena uvodoma, struktura projekta sledi priporočilom orodja Apache Maven. Zelo priporočljivo je, da si to orodje namestite in ga uporabljate, saj bo delo precej lažje. Pri vsakem resnejšem projektu, ki se ga boste lotili, boste prej ali slej morali spoznati tovrstna orodja. Navodila za namestitev lahko najdete na domači strani orodja[3].

Ko je orodje ustrezno nameščeno, ga lahko uporabimo iz ukazne vrstice (v Windowsu `cmd.exe` ali Powershell).

[3] `https://maven.apache.org`

Uporabni cilji orodja Apache Maven

V nadaljevanju si pogledamo osnovne operacije orodja Apache Maven, ki jih lahko uporabimo v sklopu seminarja. Ves čas predpostavljamo, da se nahajamo v imeniku `seminar1` (tj. tam, kjer se nahaja `pom.xml`). Kot je razvidno iz vsebine datoteke `Makefile`, se bo pri prevajanju programa uporabljajo naslednje preproste cilje:

Prevajanje datotek. Izvorne datoteke `*.java` lahko prevedemo z ukazom

```
mvn compile
```

V kolikor še ni, bo Maven v imeniku `seminar1` ustvaril nov imenik, imenovan `target`, ki bo vseboval prevedene razrede imenika `src`. Prevedeni razredi se bodo nahajali v imeniku `target/classes`.

Poganjanje testov. Prevajanje datotek in poganjanje testov lahko dosežemo s ciljem

```
mvn test
```

Priprava paketa. Še bolj kot prevajanje je uporabna priprava paketa, ki izvorne datoteke prevede, požene teste in pripravi paket, tj. `jar`, ki ga lahko potem poženemo.

```
mvn package
```

Rezultat te akcije bodo prevedene `.class` datoteke, odložene v imeniku `target/classes`, izvršljiv paket `seminar1-1.0-SNAPSHOT.jar` v imeniku `target` in pognani testi v `src/test`. Če se kakšen izmed testov ne zaključi uspešno, Maven paketa ne bo pripravil, lahko pa obstaja kakšen izmed prejšnjih izvedb te akcije.

Poganjanje programa. Ko je v `target` pripravljen izvedljiv `.jar` paket, lahko program poženemo tako:

```
java -jar target/seminar1-1.0-SNAPSHOT.jar < input1.txt
```

Pri tem programu na standardni vhod podtaknemo vsebino datoteke `input1.txt`.

Odstranjevanje izvršljivih datotek. Pred oddajo je potrebno počistiti izvršljive `.class` and `.jar` datoteke, kar lahko priročno storimo z ukazom

```
mvn clean
```

Rezultat akcije bo odstranjen imenik `target` in vsa njegova vsebina.

Literatura

[1] S. W. Golomb. *Polyominoes. 1994.* Princeton University Press, Princeton, NJ.

[2] Solomon W. Golomb and Leonard D. Baumert. Backtrack programming. *Journal of the ACM (JACM)*, 12(4):516–524, 1965.

[3] Donald E. Knuth. Dancing links. *arXiv preprint cs/0011047*, 2000.

[4] Dana Scott. *Programming a combinatorial puzzle.* Princeton University Department of Electrical Engineering, 1958.