HÁZI FELADAT

Szoftver laboratórium 2.

Pontosított feladatspecifikáció

Dicse Gábor
E5DBLC

TARTALOM

1. Feladat

Demokratikus sakk

Tervezzen "demokratikus sakk" modellezésére objektummodellt! Ebben a játékban a táblán a figurák önállóan döntenek, hogy hova lépnek. Minden figura tudja a saját szabályait. A megvalósítandó modellben felváltva választunk egy-egy figurát a sötét, ill. a világos mezőkről és megkérjük azokat, hogy lépjenek. Az egyszerűség kedvéért csak gyalogokat modellezzen!
Demonstrálja a működést külön modulként fordított tesztprogrammal! A játék állását nem kell grafikusan megjeleníteni, elegendő csak karakteresen, a legegyszerűbb formában! A megoldáshoz ne használjon STL tárolót!

2. Pontosított feladatspecifikáció

A program egy olyan sakkjátékot modellez, amelyben a bábuk saját maguk döntik el, hogy hova lépnek, tehát a felhasználó csak azt döntheti el, hogy melyik figura lépjen. A játék szabályai, amelyek alapján a lépések történnek, megegyeznek az eredeti sakkjáték szabályaival, annak a kivételével, hogy itt csak gyalogokat modellezünk és nem a király, hanem az összes gyalog leütése, vagy a döntetlen állás (senki sem tud lépni) jelenti a játék végét.

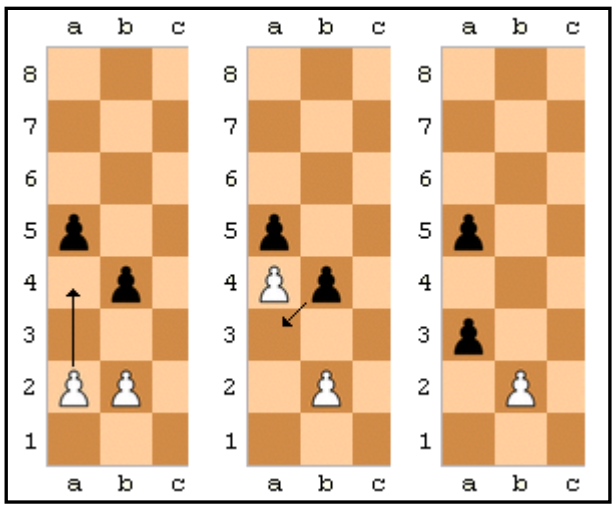
Néhány szabály:

- A tábla 8x8-as méretű.
- A bábuk két csapatra oszlanak: sötét és világos; és ezek felváltva lépnek.
- Az alapfelállás 8-8 gyalog a tábla alsó és felső sorában a két színből.
- A világos csapat kezd.
- Ha az egyik csapatnak elfogynak a bábui, akkor a másik a győztes.
- Ha senki sem tud lépni, olyankor döntetlen az eredmény.

Gyalog lépései:

- Sima: egy mezőt előre.
- Kezdő: első lépésként két mezőt előre.
- En passant: a gyalog leüthet egy ellenséges gyalogot úgy is, ha az egy dupla lépéssel mellé került. Ilyenkor az ellenséges bábu mögé lépve leütheti azt.

En Passant



A játék menete:

1. A felhasználó kiválaszt egy bábut, a sakkban megszokott módon (Pl.: A1, C5, F2…).
2. A kiválasztott bábu lép egyet.

A program ki és bemenetei:

Bemenet: Csak a felhasználótól kap. A kiválasztott bábu koordinátája a fentebb említett formátumban.

Kimenet: A bábu által választott lépés helye, szintén a fentebb említett formátumban, illetve a pillanatnyi játékállás ábrázolása karakteresen.

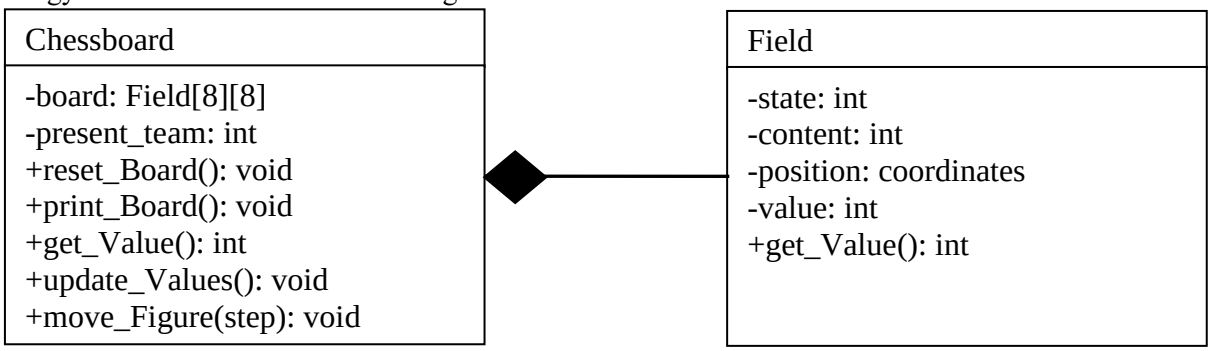
Tesztprogram:

Mivel a lépések helyét előre nem tudjuk meghatározni, ezért a tesztprogram csak a lépések helyességét tudja vizsgálni.

3. Terv

3.1 Objektumterv

Mivel a programban csak gyalogokat modellezek, ezért csak két osztályt használok a játék leírásához, és egy láncolt listát a számítások elvégzéséhez.

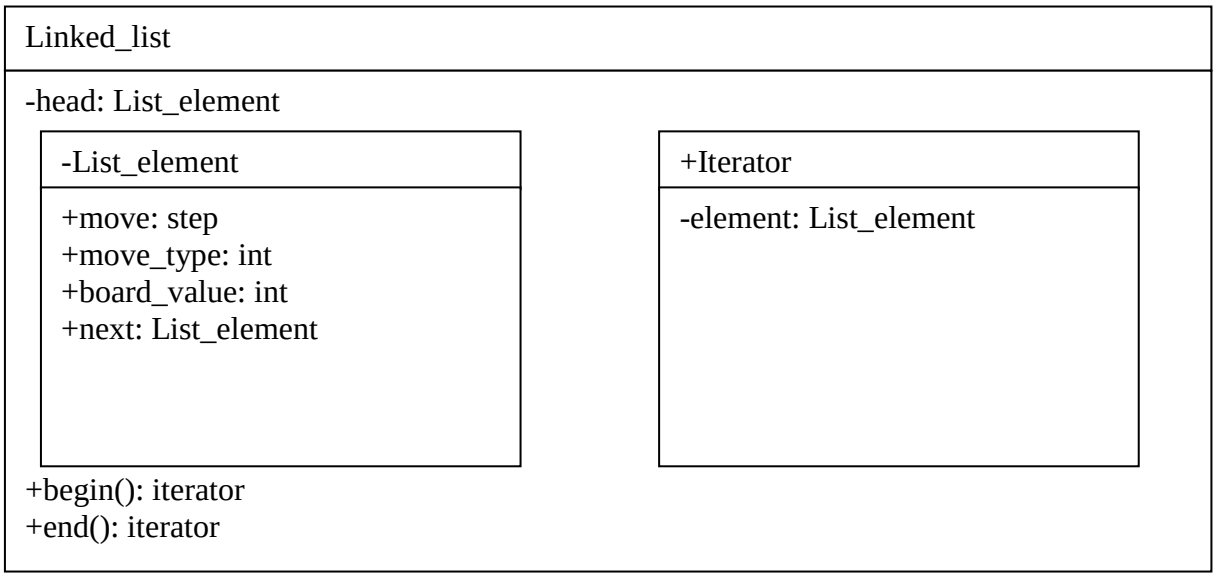


A „**Chessboard**” osztály jelképezi magát a sakktáblát. Az osztály tartalmaz egy 8x8-as 2D-s „**Field**” típusú elemekből álló tömböt, és egy „**present_team**” változót, ami az aktuális csapatot mutatja.

Az osztály főbb függvényei:

- **reset_Board** Visszaállít egy előre megadott állást.
- **print_Board** Kirajzolja a táblát a konzolba.
- **get_Value** Kiszámítja a tábla értékét az állás alapján.
- **update_Values** Frissíti a mezők értékeit a többi mező viszonylatában.
- **move_Figure** A tömbben áthelyez egy „bábut”.

A „**Field**” osztály egy mezőt jelképez, és itt az egyszerűség kedvéért a bábukat is reprezentálja a „**content**” változójával. A **position** változóban a mező a saját „koordinátáit” tárolja, a „**state**” pedig a mező állapotát mutatja (támadott/védett + fokozatok). A „**value**” változó a mező értékét tárolja; inicializáláskor a többi mezőtől függetlenül, azonban ezt frissíteni fogja a mezőt tartalmazó osztály erre dedikált függvénye. Az osztály egyetlen függvénye a „**get_Value**”, ami szimplán visszatér az adattaggal.



A „**Linked_list**” osztály egy sima láncolt lista, amely a „**move**” változójában a lépés kiinduló és végpontját tárolja. A „**move_type**” változóban a lépés fajtáját, és a „**board_value**”-ban pedig a lépéshez tartozó értéket.

3.2 Algoritmusok

A program legfontosabb algoritmusa az amelyik a bábuk „gondolkodását” biztosítja. Erre a problémára, mint a legtöbb táblás játékhoz, a min-max algoritmust fogom használni.

Az algoritmushoz szükség van két szereplőre. Az egyik szereplő a min (itt a fekete), am másik pedig a max (itt a fehér) szerepét tölti be. A lényeg, hogy mindkét szereplő a neve által megjelölt szélsőértékre törekszik a pálya értékét illetően, magyarul a min minél kisebb értéket akar, a max pedig minél nagyobbat.

A pálya értékét jelen esetben a mezők értékének az összege fogja adni, amely a tartalmazott bábutól, illetve attól is függ, hogy védett-e, vagy támadott a mező. Mivel itt csak gyalogokat modellezünk ezért egyszerűen a sötét gyalogok értéke -1, az üres mezőé 0, a világosoké pedig 1 lesz, így, ha az alapállást vesszük a tábla értéke 0-án áll. Minden a gyalogok által védett mező értéke változik, ha arra egy másik gyalog kerül (értelemszerűen a saját színének kedvező irányba), amúgy 0 marad.

Egy másik algoritmus még a mezők értékeinek állítása, amit itt nem részleteznék. (A példában, a védett mezők értékét egyel csökkenti, illetve növeli, színtől függően)

Gyakorlati példa:

Alapfelállás

Tábla értéke: 8 - 8 = 0

1. lépés

Tábla értéke: 8 - 8 = 0

2.lépés

1 sötét gyalog védett mezőre lépett.

Tábla értéke: 7 – 7 = 0

3. lépés

Világos gyalog üt.

Tábla értéke: 8 - 7 = 1

Alternatív 3. lépés

Világos gyalog megvédi az elöl lévőt.

Tábla értéke: 8 - 7 = 1

4. lépés

Sötét támadja a világost védelemből.

Tábla értéke: 7 - 7 = 0

3.3 Tesztprogram

A tesztprogramban azt vizsgálom, hogy a megadott bemenetre, ami egy bábu helyzete, vagy egy lépés helyét kapom vissza, vagy ha nincs ott bábu akkor újra kell próbálkozni. Az első esetben a lépés helyességét lehet vizsgálni, a második eset viszont általában csak a legelején tesztelhető, amikor tudom a bábuk pontos helyzetét, így erre ott lesz több próbálkozás. Tesztelhető lenne még, hogy véget ér-e a játék, de ez sok időt venne igénybe, így csak az első két esetre hegyatkozom.