Házi feladat

Vizhányó Miklós Ferenc
NVY1AG

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!

Specifikáció

A feladatban leírt sakk modellt szükséges megvalósítani. A leíráshoz képest, az egyeztetés alapján minden figura megvalósításra kerül.

A program egy konzolos applikáció lesz, mely Windows operációs rendszeren lesz futtatható. A játék indítása után rövid parancsok kiadásával lehet majd irányítani a figurákat. Ha helytelen parancsot adott ki a felhasználó azt a program jelezni fogja és a táblán nem történik változás. (pl.: nincs bábu a megadott mezőn) Egy-egy lépés után a program kiírja, az adott játékállást. A játéknak sakk-mattal van vége, vagy ha közben megszakítja a felhasználó a program futását. Ha véget ért a játék, kiírásra kerül a győztes szín.

Ezeken felül még egy tesztprogram is megírásra kerül, mely demonstrálja a fő program működését. Ilyenkor a bábuk kiválasztását is a program végzi, nem szükséges felhasználói bemenet.

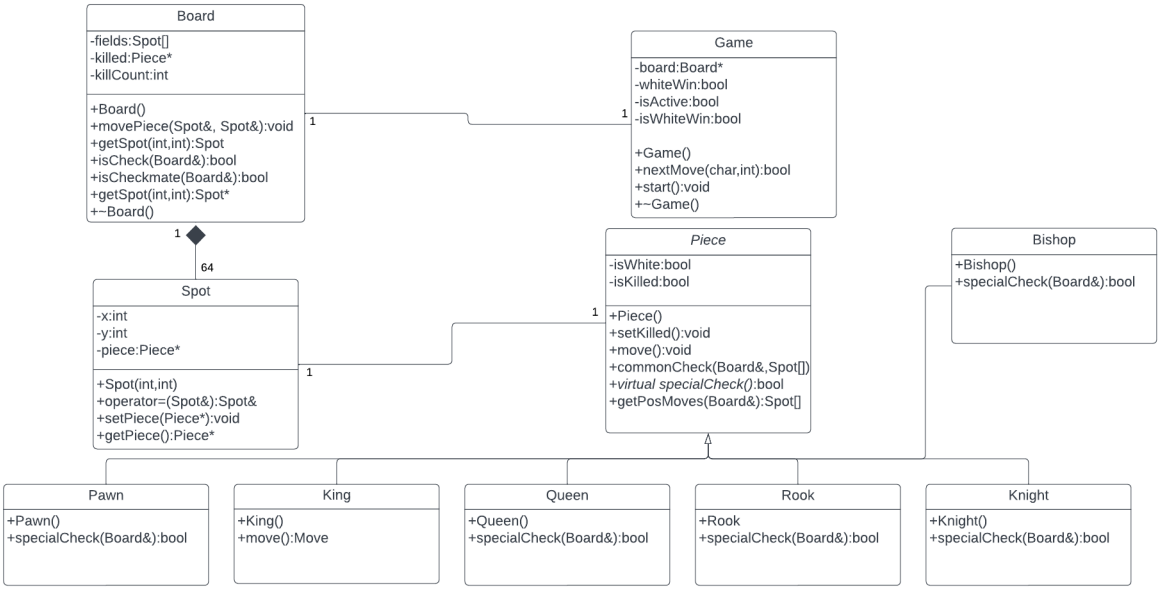
Terv

Az alábbi UML diagrammon látható osztályok szerint lesz felépítve a program. A játék (Game) objektum egy tábla (Board) objektumhoz kapcsolódik, ez tartalmazza a mezőket (Spot). Ezen a táblán mozognak a bábuk (Piece).

A bábuk lépési algoritmusa először megnézi, hogy hova léphet az adott bábu a saját szabályai szerint, utána ellenőrzi, hogy a Board állása alapján melyikeket nem lépheti meg (pl.: azonos színű bábu helyére nem léphet). Végül a maradékból választ egyet, és ezt továbbítja a Board-nak.

A játék minden lépés előtt ellenőrzi, hogy kialakult-e matt, vagy kilép-e a játékos.

A gyalogoknak ismerniük kell az en passant szabályt, és ehhez tudniuk kell az ellenfél gyalog utolsó lépését. A tesztprogram unit teszteket fog tartalmazni az algoritmusok ellenőrzésére.



Dokumentáció

Áttekintés

A program egy Windowsos futtatható konzolos sakkjáték, amelyben a felhasználó kiválaszthat egy bábut a bábu mezőjének meghatározásával. Ezt a mező két koordinátájának egymás utáni megadásával történik (pl.: „a1” vagy „d2”). A játékból kilépni a „q1” kombinációval lehet (vagy EOF karakterrel). A játék addig fut amíg nem lesz valamelyik oldalon matt vagy a felhasználó ki nem lép.

A programban lévő bábuk egy általános Piece osztályból származnak, amely tartalmazza a különböző bábuk közös műveleteit. A bábuk egyedi szabályokkal is rendelkeznek, amelyeket a leszármazott osztályokban lettek implementálva.

A program a Game osztályból áll, amelynek a Board objektuma tartalmazza a Spot objektumokból álló táblát. A Spot objektumok tárolják a bábukat, amelyek az adott helyen vannak.

Főbb függvények

Game osztály

void start()

> Elindítja a játékot, majd addig működteti, amíg nem lesz matt vagy a felhasználó nem lép ki. A játékot a *nextMove* függvény meghívásával működteti.

bool nextMove()

Kommunikál a felhasználóval. Kijelzőre az adott játék állást, bekéri a következő lépésre szánt bábu helyét a *getInput* függvény segítségével. Elkapja a dobott kivételeket és kiírja a hiba üzenetet.

bool getInput(int& row, int& col)

Bekéri a következő lépésre szánt bábu helyét. Ellenőrzi a bemenetet és addig olvas be a paraméterként kapott változókba, amíg nem sikerül beolvasnia egy karaktert és egy számot. Ellenőrzi a kilépési feltételeket.

void validateCoordinates(int x, int y)

Ellenőrzi a paraméterként kapott koordinátákat. Kivételt dob, ha a felhasználó nem létező koordinátákat adott meg, vagy a megadott koordinátán nincs bábu.

Board osztály

Board()

Konstruktor az Board osztályhoz. Létrehozza a 8x8-as fields tömböt és inicializálja az elemeit az *initializePieces* függvénnyel. Létrehozza a leütött tömböket. Minden tömb felszabadítása a destruktor dolga lesz.

~Board()

Felszabadítja a tömböknek foglalt területeket.

void initializePieces()

Inicializálja a táblát a kezdőállapotban. Elhelyezi az összes bábut a megfelelő helyre a fields tömbben.

void display()

Kiírja a kijelzőre a tábla tartalmát. Ha megfelelő környezetben van törli a kijelzőt mielőtt kiír.

void clear()

A királyok kivételével törli a bábukat a tábláról, felszabadítja őket.

void capturePiece(int x, int y)

Ha van bábu a paraméterként kapott mezőn, akkor azt áthelyezi a tábláról a leütött bábuk tömbjébe.

Spot getKingSpot(Color color)*

Visszaadja a paraméterként kapott színű király helyét.

void pawnPromotion(Spot end)*

Ellenőrzi, hogy a gyalog a tábla szélére ért-e és ha igen, akkor lecseréli egy random választott leütött bábura a leütött bábuk tömbjéből (ha van leütött bábu).

void enPassant(Spot start, Spot* end)*

Ellenőrzi, hogy a lépés egy en passant lépés-e és ha igen, akkor kezeli az ellenfél gyalog leütését.

void movePiece(Spot start, Spot* end)*

Átmozgatja a start helyről az end helyre a bábut, ha nem volt üres a mező. Ha volt ott ellenfél bábu, akkor azt áthelyezi a leütöttek tömbjébe. Meghívja az *enPassant* és *pawnPromotion* függvényeket.

bool isKingSafe(Color color)

A megadott színű király megkeresése után (*getKingSpot*) visszaadja, hogy biztonságban van-e a király.

bool isCheckmate(Color color)

Visszaadja, hogy a paraméterként kapott színből bármelyik bábu tud-e még lépni.

Piece osztály

void move(Board& board)

Meghívva a bábu lépni fog, amennyiben tud. Ehhez lekéri a lehetséges lépéseket a *getPossibleMoves* függvénnyel és ha van lehetséges lépés, akkor meglépi a *movePiece* függvény meghívásával.

bool commonValidation(Spot currentSpot, Spot* destinationSpot)*

Ellenőrzi az általános szabályokat a bábu számára. Pl.: nem akar-e kilépni a bábu a tábláról vagy nincs-e ott saját bábu. Ez után ellenőrzi a bábu-specifikus szabályokat a *specificValidation* függvénnyel, illetve az en passant szabályt az *enPassantValidation* függvénnyel. Végül ellenőrzi, hogy van-e az útjában valami az *isPathObstructed* függvénnyel, és ha minden ellenőrzésen átment, akkor igazat ad vissza.

virtual bool specificValidation(Board& board, Spot currentSpot, Spot* destinationSpot) const = 0*

Virtuális függvény, amely ellenőrzi a bábu sajátos szabályait. Implementálva a leszármazott osztályokban van.

*virtual Spot** getPossibleMoves(Board& board)*

Összegyűjti egy tömbbe az összes lehetséges lépést, amiket a bábu megtehet a szabályok alapján úgy, hogy közben nem kerül sakkba a saját királya. A tömb felszabadítása a hívó feladata.

bool isMoveResolvingCheck(Spot currentSpot, Spot* destinationSpot, Board& board)*

Megadja egy lépésről, hogy utána sakkban lesz-e a király vagy nem. Meghívható anélkül is, hogy kialakult volna előtte sakk helyzet.

Fő algoritmusok

Lépések érvényességének ellenőrzése

- Ellenőrzi, hogy a kiinduló mező és a cél mező érvényesek-e a táblán és nem azonosak-e.
- Ellenőrzi, hogy a cél mezőn nincsen-e saját színű bábuja.
- Ellenőrzi a bábu-specifikus szabályokat.
- Ellenőrzi, hogy en passant lépés lehetséges-e.
- Ellenőrzi, hogy nincs-e akadály a bábu útjában (ha a bábu nem huszár).

Lehetséges lépések meghatározása

- Készít egy dinamikusan foglalt Spot** tömböt a lehetséges helyek tárolására.
- Végig nézi a tábla összes mezőjére, hogy lehetséges-e oda lépnie a bábunak. Ha igen, hozzáadja a tömbhöz.
- Az utolsó elem nullptr, ez jelzi a tömb végét.

Sakk feloldása

- Ellenőrzi az összes saját bábura, hogy van-e bármi olyan lehetséges lépés, amivel a király újra biztonságban lenne. Ha nincs, akkor matt helyzet alakult ki, vége a játéknak.

Teszt program

Unit teszteket tartalmaz az algoritmusok ellenőrzésére. Az egyes tesztekhez a programkódban van magyarázat komment formájában. A TESTING makró beállításával futtatható. *gtest_lite* segítségével működik.