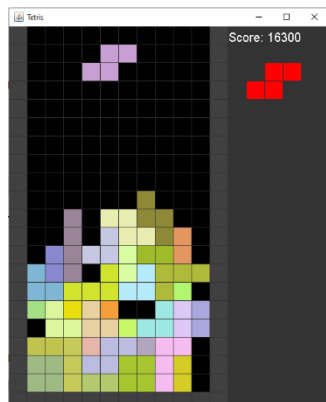


Programozás alapjai 3. – Házi feladat dokumentáció

User Manual

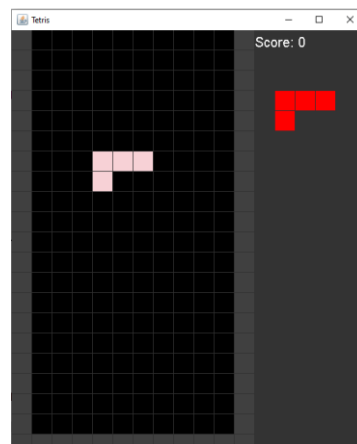
A programom egy Java nyelven írt játékprogram, ami a híres Tetris nevű játék egy általam készített példánya. Ehhez mérten a játék menete ugyanúgy zajlik, mint az említett játékban. A kezdetén egy véletlenszerű alakzat a 7 közül (I, J, L, O, S, T, Z) esik a pálya egy megadott és rögzített pontjából lefelé, a pálya aljára.



Sok idő után a játék egy lehetséges pillanatképe

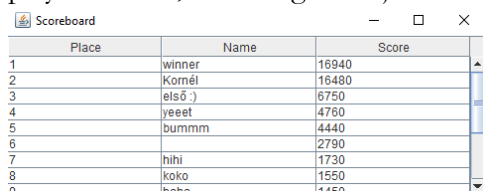
Esés közben a nyilak segítségével tudjuk irányítani az alakzatot, hogy oda essen, ahova kedvező lenne szerintünk. Erre azért van szükség, mivel a játék célja, hogy egy sort telepakoljunk ilyen alakzatokkal, hogy ne legyen rés köztük, mivel ekkor, az adott sor eltűnik.

Ezen kívül még forgatni is tudjuk az alakzatunkat, hogy ezzel is segítsünk a sorok megtöltésén. Sor eltüntetéséért pedig pontot kapunk, arányosan az eltüntetett sorok számával. Végző célunk, hogy a játékban minél több pontot szerezzünk. Ezen kívül ponthoz juthatunk, ha a lefele nyíllal gyorsítjuk a leesését az alakzatnak. A játék nehezítése miatt, az folyamatosan gyorsul.



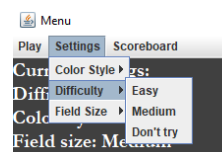
Játék elkezdésekor látható kép

A következőkben az szeretném ismertetni, hogy milyen funkciókat valósítottam meg a játékból. Lehetőség van ugyanis arra, hogy lássuk mi lesz a következő alakzat, látjuk a pontunkat kijelezve, van lehetőségünk beállítani a pálya méretét, nehézségi szintjét valamint azt is, hogy milyen színpompákban szeretnénk látni a Tetrominóinkat.



Place	Name	Score
1	winner	16940
2	Kornél	16480
3	első :)	6750
4	yeet	4760
5	bummm	4440
6		2790
7	hihi	1730
8	koko	1550
9	haha	1450

Dicsőségtábla

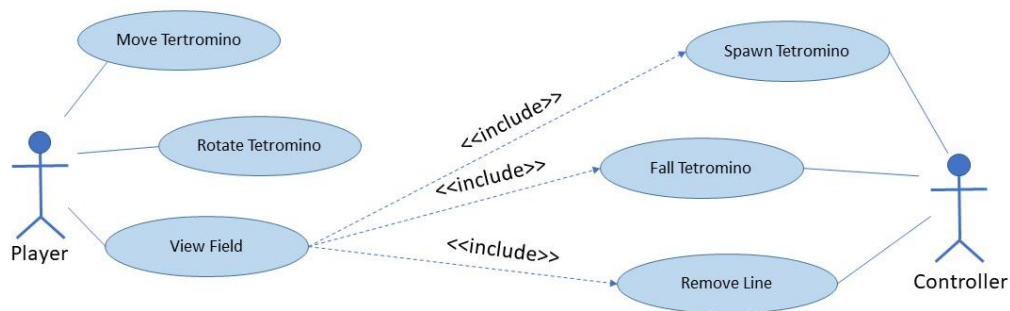


Menü

Lehetőségünk van a játék végeztével,

a nevünket megadva felkerülni a dicsőiséglistára, ahol később látni fogjuk azt. Ha esetleg javítani szeretnénk a pontunkon, arra is van lehetőség, hiszen, ha ugyanazt a nevet adjuk, meg akkor csupán egyszer fog megjelenni nevünk, még hozzá a magasabb ponttal.

Use -case-ek



Cím	Move Tetromino
Leírás	A játékos a tetrominót irányítja a pályán.
Aktorok	Player
Főforgatókönyv	1. A játékos a tetrominót jobbra, balra vagy lefelé mozgatja
Alternatív forgatókönyv	1.A.1. A pálya szélein túl nem lehet mozgatni a tetrominót
Alternatív forgatókönyv	1.B.1. Ha leesett a tetrominó, már nem mozgathatja a játékos azt tovább
Alternatív forgatókönyv	1.C.1. Ha megtelt a pálya a játék véget ér

Cím	Rotate Tetromino
Leírás	A játékos forgatja a tetrominót.
Aktorok	Player
Főforgatókönyv	1. A játékos 90°-al jobbra fordít egyet a tetrominón
Alternatív forgatókönyv	1.A.1. Ha fal mellett van a tetrominó, és belefordulna a falba, akkor nem fordul el
Alternatív forgatókönyv	1.B.1. Ha leesett a tetrominó, azt már nem lehet forgatni
Alternatív forgatókönyv	1.C.1. Egy másik, már a pályán lévő tetrominó is meg tudja akadályozni a forgatást, ha belefordulna

Cím	View Field
Leírás	A játékos megtekintheti a pályát.
Aktorok	Player
Főforgatókönyv	1. A rendszer kirajzolja a pálya aktuális állását 2. A játékos megtekinti a pálya aktuális állását

Cím	Spawn Tetromino
Leírás	Megjelenik egy tetrominó.

Aktorok	Controller
Főforgatókönyv	1. Középen megjelenik egy véletlenszerű tetrominó.
Alternatív forgatókönyv	1.A.1. Ha a pálya tele van, vége a játéknak

Cím	Fall Tetromino
Leírás	A tetrominó esik lefelé
Aktorok	Controller
Főforgatókönyv	1. A tetrominó adott időközönként egy egységnégyzetet esik lefelé
Alternatív forgatókönyv	1.A.1. Ha ráesik egy, már a pályán lévő tetrominóra, akkor rajta marad, és nem esik tovább
Alternatív forgatókönyv	1.B.1. Ha a pálya aljára ér a tetrominó, nem esik tovább
Alternatív forgatókönyv	1.B.1. Ha a pálya tele van, vége a játéknak

Cím	Remove Line
Leírás	Kitörlődik egy sor
Aktorok	Controller
Főforgatókönyv	1. Ha megtelik a tetrominókkal egy sor, az kitörlődik, és a felette lévő sorok egy sorral lejjebb esnek

Be- és kimeneti fájlok

A programhoz van egy scores.dat fájl, amibe történik a kiírás, és amiből történik a beolvasás. Azért használom .dat fájlt, mert ebben szokás a játékhöz (eredménytáblához) szükséges fontos adatokat. Azért szokták ilyen fájlba tárolni, hogy a felhasználó ne férjen könnyen hozzá. Belül sima szöveg van, mint akár egy .txt-ben.

Adatszerkezetek

Az egyik használt adatszerkezetem egy egyszerű kétdimenziós tömb. Ezt használtam a Tetrominók valamint a pálya tárolására. A Tetrominók esetében a tömbben Point-tömöket tárolok, ami egy kézenfekvő osztály erre a célra, hiszen egy pontnak van x és y koordinátája, amivel egyszerűen tudom kezelni az alakzatokat. Azért szükséges ez a forma, hiszen a forgatásra is kell gondolni, és így a koordináta geometria adta lehetőségeket így ki tudom használni. Ezzel szemben a megjelenítéshez a pálya mátrix adott sor és oszlop párosa a helyet megadja, így nem kell nekem ezt külön tárolni, ott elég csupán az adott alakzatra érvényes szint tárolni. Ennek megfelelően a Field egy Colors-okból álló kétdimenziós tömb. Ezekon kívül megjelenik a sima egydimenziós tömb is, mivel egy Tetromino tárolásához elegendő ez, hiszen a Point már magában hordozza a kétféle dimenziót.

A Scoreboardhoz szükséges volt azonban másra is. Ehhez a laboron tanultakat alkalmaztam, és az MVC modellnek megfelelően csináltam egy Scoreboard osztályt, amiben lehet tárolni helyezést, nevet, valamint pontot. Ezt felhasználva hoztam létre a ScoreboardData osztályt, ami

Scoreboardokat tárol egy Listben tárolom, ami egy ArrayList. Erre a könnyű kezelés miatt esett a választásom.

Osztályok leírása

Menu

Ez az osztály azért felel, hogy megjelenítse a menü ablakát, valamint az azon levő gombokat irányítsa. A Menu leszármazik a JFrame osztályból, így benne tudom használni a függvényeit a JFrame-nek, és egyben tudom kezelni az ablakot és a hozzá tartozó dolgokat. A menük megalkotásához a JMenu osztályt használtam. Van egy JMenuBar-om, amire rá tettem 3 főmenüt. A játék indításáért felelős Play-t, a beállításokért felelős Settings-et, és a dicsőséglistánért felelős Scoreboardot. Az almenüket tekintve a Play-nek van egy New Game gombja, amivel elindul a játék. A Settings-nek van egy Color Style, egy Difficulty és egy Field Size almenüje, amivel rendre a szín stílusát (Happy = világosabb színek, Dark = sötétebb színek), a nehézséget (2 esés közben kezdetben eltelt idő: Easy = 1000 ms, Medium = 500 ms, Don't try = 250 ms) és a méretet (Small = 10*15, Medium = 12*21, Large = 16*27) lehet beállítani. Tárolok ezeken kívül Stringeket, amik arra szolgálnak, hogy ki lehessen írni az aktuális menü beállításokat.

A Menu osztály privát attribútumai ezek a menüt irányító dolgok, egy Game típusú változó, ami azért szükséges, hogy a játékot el lehessen indítani, egy Dimension típusú fieldDim, ami a pálya méretért tartalmazza, ami azért kell, mivel a méretet már ismerni kell a Game létrehozásakor, és egy adott mérettel kell létrehozni a Game-et, mivel az hozza létre a Tetrist. Valamint van még kettő int, egy speed, ami a sebességet tárolja, és egy colorStyle ami a színkeveréshez szükséges egész számot tárolja, ezek azért szükségesek, hiszen amíg nincs létrehozva a Game, addig nem lehet változtatni annak attribútumait.

Metódusok

- **initMenu()**: létrehozza a fent leírtak szerint a menü ablakát, létrehozza JMenuBar-t és elhelyez minden menüpontot, beállítja a megadott képet háttérnek és megjeleníti a frame-et.
- **initGame()**: elindítja a játékot, a lambdával megvalósított ActionListener ezt hívja meg
- **main()**: itt van a program belépési pontja
- **scoreboard()**: létrehoz egy ScoreboardFrame-et és láthatóvá teszi
- továbbá setterek, amik színt, nehézséget vagy pályaméretet változtatnak.

Game

A Game osztály azért felel, hogy összekösse a játék logikáját a megjelenítéssel, valamint a játékot olyan módon vezényelje, hogy egy külső GameThread osztályt segítségül hívva elindítsa a játékot, valamint, ha a feltételek úgy diktálják (tele a pálya), akkor véget vet a játéknak. Fő feladata a vezénylés.

Privát attribútumai egy Visual típusú változó, ami a megjelenítésért felelős osztály, egy Tetris típusú változó, ami játék háttér logikájáért felel, egy Keyhandler típusú változó, ami a gombnyomásokért felel, egy int, amiben a gyorsaságot jellemző számot lehet tárolni.

Metódusok

- **Game()**: konstruktor, létrehozza a privát tagváltozókat

- **initGame():** elindítja a játékot, létrehoz egy GameFrame-et, meghívja a megjelenítésért felelős függvényét (startGame()), majd létrehoz egy Thread-et, aminek delegál egy GameThread-et (szálbiztosság miatt ez nagyon fontos), majd elindítja a szálát
- **endGame():** létrehoz egy GameFrame-et, majd meghívja azt a függvényét, ami azért felel, hogy felépítse a játék végén lévő ablakot

GameFrame

Fő feladata, hogy létrehozza a játékhoz szükséges ablakokat, valamint, mivel azt az ablakot is ő kezeli, ami a játék végén bekéri a játékos nevét, ezért azért is felel, hogy a fájlba ki legyen ez a név írva, ne vesszen el. Leszármazik a JFrame-ből.

Ehhez mérten privát változója egy ScoreboardData típusú változó, amiben pontokat kezelem.

Metódusok

- **initData():** beolvassa a fájlból az adatokat a data változóba valamint hozzáad az osztályhoz (ami ugye JFrame-ből származik le) egy WindowListenert, hogy amikor bezárjuk az ablakot, akkor mentse el a beírt nevet ponttal együtt
- **startGame(g: Game):** paraméterül kap egy Game objektumot és ennek az adataival felépít egy ablakot, amiben a játék látszódní fog, és beállítja az ablakhoz a keylistenereket valamint a kirajzoló osztályt is hozzáadja a frame-hez
- **endGame(g: Game):** paraméterül kap egy Game objektumot majd felépít egy ablakot amin van egy JLabel, egy JTextField és egy JButton, amik egy felületet adnak ahhoz, hogy be lehessen írni a játékos nevét, hogy felkerüljön a dicsőséglistára, amit a gombhoz adott ActionListener kezel
- **addName(tf: JTextField, score: int):** meghívja az initData() függvényt, majd a kapott tf és score segítségével a textfieldben lévő névvel és a score ponttal beírja a datába a játékost, valamint megjeleníti a scoreboardot

GameThread

Az osztály leszármazik a Thread osztályból, mivel a feladata annak a szálnak a kezelése, ami azért felel, hogy a játékban adott időközönként történjen egy „lépés” (essen egyet az alakzat).

A privát attribútumai egy Game, amiért felel, int típusú speed, ami a gyorsaságot tárolja és egy volatile boolean típusú stopSignal, amivel a szál leállítását valósítom meg. A volatile azért hasznos, mivel így biztosítva van, hogy egységnyi idő alatt rendelkezésre áll a változó. Ez egy bevett szokás szálkezelésnél különféle anomáliák megelőzése miatt.

Metódusok

- GameThread(game: Game,s: int): konstruktor, beállítja a kapott paraméterek szerint a privát attribútumok értékét
- start(): meghívja a run metódust és a stopSignal false értékre
- run(): egy ciklust indít, amíg nem lesz igaz értéke a stopSignalnak, ez a kapott Game típusú g változóban lévő Tetris típusú változón keresztül jut vissza hozzá, aminek mellékhatása, hogy lép egyet a Tetris, ezután kirajzolja a jelenlegi állást a g-n keresztül elérhető kirajzoló osztállyal, és ez után ha nem lett igaz a stopSignal (ez azért kell, hogy ha vége a játéknak az egyből megtörténjen), akkor várakozik adott ms-ig majd csökkenti egyel speedet, amivel a folytonos gyorsulást valósítottam meg

Keyhandler

Ez az osztály implementálja a `KeyListener` interfészt, mivel a felelőssége a gombnyomások kezelése.

Privát attribútuma egy `Game` típusú `g` változó, ami azért szükséges, hogy a benne lévő `Tetris` típusú változót elérjük, és így a `Tetrominot` léptetni vagy esetleg forгатni tudjuk.

Metódusok

- **keyPressed(e: KeyEvent):** ez az interfész egyik kötelezően definiálandó függvénye, ami a fel, le, jobbra, balra gombokhoz mérten rendre forgat, lefelé-, jobbra- illetve balra mozgat, valamint a lefelé mozgatásnál növeli a pontot
- a másik kettő függvény kötelezően definiálandó, azonban nekem ezekre nem volt szükségem

Scoreboard

Ez az osztály definiálja a sémáját a dicsőségtáblának.

Privát attribútumai egy `int` a helyezéshez, egy `string` a névhez és egy `int` az elért ponthoz.

Metódusok

- getterek és setterek

ScoreboardData

Ez az osztály felel azért, hogy biztosítsa az adatokat a `ScoreboardFrame`-nek. Ezért leszármazik az `AbstractTableModel`-ből. Megjegzés: azért nem a `TableModel` interfészt implementálja, mivel így kevesebb metódust kell felüldefiniálni.

Attribútuma egy `List<Scoreboard>`, ami egy `ArrayList<Scoreboard>`-ként van létrehozva.

Metódusok

- javarészt getterek, azért, hogy a táblázatot fel lehessen majd építeni
- **removeDuplicates():** `List<Scoreboard>` visszatérési értékkel, az osztály attribútumából eltüntet az ismétlődő nevű tagokat, úgy hogy létrehoz egy új listát, majd sorban végighalad az eredeti listán, és ha talál egy olyat, amilyen nevűt még nem tett be az új listába akkor beteszi (megjegyzés: ha az eredeti lista rendezett volt, ez nem rontja el a rendezettséget, és ezenkívül a duplikátumokkal úgy bánt el (rendezettséget feltéve), hogy mindig a legnagyobb pontú az, aki bekerül az új listába – pont, ahogy a program elvárja)

ScoreboardFrame

Ez az osztály azért felel, hogy a meglévő `Scoreboard` és `ScoreboardData` osztályok segítségével megjelenítse a dicsőségtáblát. Ezért `JFrame`-ből származik le.

Privát attribútuma egy `ScoreboardData` típusú adat, amit beletölt a `JTable`-be.

Metódusok

- **initComponents():** beállítja az ablak layoutját valamint létrehozza a `JScrollPane`-t és `JTable`-t benne az adattal, amit megjelenít majd

- **ScoreboardFrame():** konstruktor, induláskor betölti az adatokat majd rendezi és kitörli az esetleges duplikációkat, majd egy hozzáadott WindowListener segítségével eléri, hogy záráskor mentve legyenek az adatok, és végül felépíti az ablakot

Steppable

Interfész, ami a `step()` metódus megvalósítását kéri attól, aki implementálja.

Tetris

Ez az osztály felel a háttérlogikáért, ami a játékot mozgatja.

Privát attribútumai

- **Point[][] Tetrominos** – tárolja a 7 Tetrominót olyan logikával, hogy mindnek legyen egy pontja, ami a (0, 0) pont, ami később referenciapont lehet
- **Color[][] Field** – ez a mezőt írja le
- **int score = 0** – a pontot tárolja 0 kezdőértékkel
- **Point[] currentPlace** – az aktuális alakzat pályán lévő koordinátáinak megfelelő Point tömb
- **int currentPiece** – az aktuális alakzat egy számként, a következő szerint: jelentse ez a szám, aminek intervalluma [0; 6], azt a Tetrominót, amit a Tetrominos számadik helyén találunk
- **int nextPiece** – a következő alakzat, hasonló megkötésekkel, mint az aktuális, ez azért kell, hogy meg lehessen előre jeleníteni a következő alakzatot
- **Color currentColor** – ez az aktuális alakzat színe
- **Random r = new Random()** – egy random szám kreálásában segítő változó
- **int colorStyle = 255** – a színek keveréséhez használt változó (színek keverését később fejtem ki)
- **Dimension fieldDim** – a pálya dimenziója

Metódusok

- **Tetris (d: Dimension):** konstruktor, a kapott paraméternek megfelelően beállítja a dimenzióját a pályának, majd felépíti azt a Field-be (szélei sötét szürkék, közepe fekete négyzetek), létrehoz egy véletlenszerű következő alakzatot, majd kér egy új eső alakzatot
- **initColor():** ez a függvény végzi a színek keverését méghozzá úgy, hogy csinál 3 random számot [0; 255] intervallumon majd ezeket a colorStyle változó irányába eltolja, így azt lehet elérni, a colorStyle variálásával, hogy a generált színek (ugyanis fontos megjegyezni, hogy a játék teljesen véletlenszerű színekkel operál) a világos (minél nagyobb colorStyle érték) vagy a sötét (minél kisebb) felé csúsznak el, így szép pasztell színeket lehet keverni
- **newFallingPiece():** visszatérési értéke az új alakzat kérésének a sikeressége (boolean), ha nincs tele a pálya akkor veszi a következő alakzatot, hogy az legyen az aktuális, csinál egy új következőt, majd az új aktuális alakzatnak megfelelően kiválasztja a Tetrominos tömbből a megfelelő indexen lévő alakzatot, és annak a koordinátáit beállítja a pálya közepének megfelelően (fontos, hogy itt még az alakzat nincs a pályán, így majd ha kirajzolásra kerül a sor, akkor természetesen nem a pályával együtt rajzolódik, hanem külön, de ez nem ennek az osztálynak a felelőssége)
- **isMapFull():** visszatérési értékében (boolean) megmondja, hogy tele van-e a pálya, ami azt jelenti, hogy megnézi, hogy a következő alakzat kezdő pozícióba helyezése lehetséges-e (foglalt-e a hely vagy sem), ezt úgy csinálja, hogy mivel azok az alakzatok jelentenek gondot,

akik már a pályán vannak (ami azt jelenti, hogy a pálya adott koordinátájában lévő szín, nem fekete, hanem az ott lévő alakzatnak a színe), egyszerűen megnézi, hogy az új alakzat által elfoglalni kívánt helyen van-e olyan szín, ami nem fekete

- **collidesAt(here: Point):** visszatérési értékében (boolean) megmondja, hogy az aktuális alakzat a paraméterül kapott here vektorral eltolva okozna-e ütközést (megnézi a színt a helyen, és ha fekete akkor nincs ott semmi), `ArrayIndexOutOfBoundsException`-ot dob ha esetleg túl akarnánk indexelni a `Field`-et
- **rotate():** elforgatja az aktuális alakzatot -90° -al, ha az alakzat nem egy `O`-típusú (ezt nem lehet forgatni, illetve, nem történne semmi ha el lehetne, ezért önkényesen ki is zárva, mint lehetőség), úgy, hogy veszi az aktuális alakzat referenciapontját (ez a 1 indexen levő pont, ami minden alakzat esetében a Tetrominos-ban (0; 0)), majd egy új `rotated Point` tömbbe belerakja az aktuális alakzat pontjait, eltolja negatív irányba a referenciaponttal (ezért fontos ez a pont, hiszen így (0; 0)-ba kerül az alakzat), ott elforgatja (koordináta-geometria origó körüli forgatás alapján) és ez után ellenőrzi, hogy ha visszatolná, az okozna-e ütközést, és ha nem, akkor ehhez képest átalakítja az aktuális alakzat koordinátáit az elforgatottéra, túlindexelés esetére ki van képezve, hiszen az könnyen előfordulhat, azonban ez csak azt jelenti, hogy nem lehet forgatni az alakzatot
- **move(vect: Point):** adott vektorral megpróbálja eltolni az alakzatot, visszatérési értékben (boolean) jelzi, hogy ez sikerült-e vagy sem, esetleges túlindexelés esetére fel van készítve, ha a `collidesAt()` túlindexelést dob
- **clearRow():** ha van üres sor, akkor eltünteti azt, majd lejjebb pakolja az összes felette lévő alakzatot, úgy, hogy bejárja a pályát és lentől megvizsgál minden sort, ha talál egyet, ami tele van (nincs fekete szín a sorban), akkor lejjebb helyezi az összes felette lévő sort, majd megnöveli az `i`-t egyel, hogy újból alulról induljon a vizsgálat, erre azért van szükség, hogy ne legyen egy sor sem kihagyva, továbbá a kitörölt sorokat számolja, és végül négyzetes arányban a törölt sorok számával, növeli a pontot
- **placeToField():** a pályára helyezi az alakzatot, úgy, hogy az alakzat koordinátáinak megfelelő helyre a `Field`-re beírja az alakzat színét
- **step():** mozgat egyet az aktuális alakzaton lefelé, a boolean típusú visszatérési értékben jelzi ennek sikerességét

Visual

Ez az osztály felel a Tetris osztályban lévő kirajzolandó dolgok kirajzolásáért. Leszármazik a `JPanel`-ből, hogy a rajzolás könnyen megvalósítható legyen.

Privát attribútumai egy Tetris típusú változó, az az objektum, amiben megtalálható, az, amit ki kell majd rajzolni, egy `Dimension` típusú változó a pálya méretét tárolva és egy statikus változó a négyzetek méretéhez (igazából annál egyel nagyobb), ez az olvashatóság szempontjából hasznos.

Metódusok

- **Visual(tetris: Tetris):** konstruktor, a kapott attribútumnak megfelelően beállítja a Tetris típusú attribútumát, valamint a dimenziót, amit ugyanúgy a kapott paraméteren keresztül ér el
- **paintShape(Graphics g):** a kapott paraméteren keresztül kirajzolja az aktuális alakzatot megfelelő színre, úgy, hogy 1 pixellel kisebb legyen egy négyzet oldala mint a köztük lévő táv, mivel így könnyen megvalósítható lesz a vonalazása a pályának

- **paintNextShape(Graphics g):** hasonlóan kirajzolja a következő alakzatot, ám ez mindig piros lesz, és fix helyen lesz megjelenítve
- **paintComponent(Graphics g):** egy felüldefiniált függvény, ami a repaint() függvény hívásánál lesz meghívva (JPanel sajátja), ez rajzolja ki a pályát, úgy, hogy a háttér eltérő színű lesz (a vonalak miatt), és a Fieldnek megfelelő színű és index alapján megfelelő elhelyezkedésű négyzeteket rajzol ki shapeSize távolságonként, shapeSize-1 méretben, így marad közöttük egy pixel hely, ami megoldja a vonalazás kérdését, valamint meghívja a paintShape() és a paintNextShape() metódust is, és kirajzolja megfelelő helyre a pontszámot is

Osztálydiagram

