Tartalomjegyzék

1. Követelmény feltárás

1.1 Célkitűzés, projektindító dokumentum

A projekt célja egy kétszemélyes Tetris létrehozása, ahol a klasszikus Tetris játékot továbbfejlesztve a játékot egyszerre két felhasználó játszhatja, az eredeti játék játékmenetének megtartásával, plusz funkciók hozzátételével. Ezek a hozzáadott funkciók lehetővé teszik, hogy a két felhasználó egymással versenyezzen játék közben, és lehetőséget adnak a másik felhasználó játékmenetének módosítására, mindkét játékos részéről. A játéknak kezelnie kell ha két játékost ugyanazon számítógépen, valamint ha különböző számítógépeken hálózaton keresztül szeretnének játszani. Amennyiben csak egy játékos van, a másik játékos szerepét lehetőség legyen betölteni AI által. Ezeken felül a változatos felhasználói élmény garantálása érdekében lehetőség legyen különböző karakterek választására, amelyek befolyásolják a játékos milyen módokon tudja módosítani saját vagy ellenfele játékát a játék folyamán.

1.2 Szakterületi fogalomjegyzék

AI – Mesterséges intelligencia, amely egy játékos gondolkodását utánozza, és az alapján tesz lépéseket amelyeket az adott helyzetben optimálisnak ítél
Tetromino – Tetrisben használt alakzatok. 4 egymással összekapcsolt négyzetből álló alakzat. A Tetris-ben ezeket kell lehelyezni a játéktérre.
Alakzat - Tetromino
Játéktér – A hely amin a játék folyik. A lehelyezett tetromino-k egy előre megadott méretű gyűjtőhelye, üresen indul, és folyamatosan bővül a leérkezett alakzatokkal, amíg a megadott magasságig be nem tellik.
Tetromino leérkezése – Amikor egy Tetromino nem tud lejebb mozogni leérkezik a játéktérre, és annak része lesz.
Teli sor – Amennyiben a játéktéren a leérkezett tetromino-k valahol összefüggő sort alkotnak a játéktér egyik oldalától a másikig, a sor teli.
Üres sor – olyan sor amiben nem található egyetlen teli mező sem
Lyukas sor – nem üres és nem teli sor
Leérkezett alakzat – Már nem mozgatható alakzat, amely része a játéktérnek
Aktuális alakzat – Felhasználó által mozgásában és szögében módosítható alakzat
Karakter – Adott játékos által kiválasztott és megszemélyesített, egymástól jól elkülöníthető tulajdonságokkal és képességekkel bíró entitás, ami megadja az adott helyzetben használható speciális képességeket.
Speciális képesség – Adott mennyiségű mana-hoz rendelt előre definiált képességek amik módosítják a saját vagy ellenfél játékát
Avatar – A kiválasztott karakter, játék interfészén megjelenített képe

Mana - Tetromino mezője tartalmazhat véletlenszerűen „Mana” mezőt, ami egy sor eltűntetésénél a játékos tulajdonába kerül, és felhasználhatja speciális képességekre.
Sor eltűntetése – Amennyiben a játéktér teli sort tartalmaz, azt a játék eltűnteti, és minden felette lévő sort egy sorral lejebb mozgat

1.3 Játékszabályok

A szoftvernek követnie kell az 1984-ben kijött klasszikus Tetris játék szabályrendszerét, csak az abban jelenlévő alakzatokat (tetromino-kat) használhatja. A játék csak akkor érhet véget ha egy tetromino nem lehelyezhető úgy, hogy teljes egészében benne legyen a játéktérben. Ezek a szabályok mind egy valódi ember, mind az AI által irányított játékosra érvényesek.

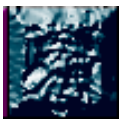
Játék célja

A játék célja az, hogy az ellenfél elveszítse a játékot, vagyis, hogy ne tudja anélkül lerakni az aktuális alakzatát, hogy az kilógjon a játéktérről. Ennek a célnak az eléréséhez több dolog is segítségére van a játékosnak.

Az első segítség ebben, hogy amennyiben egy teli sort sikerült eltűntetnie a játékosnak, az ellenfél játékterén feltűnik egy lyukas sor, a játéktér legalsó sorában.

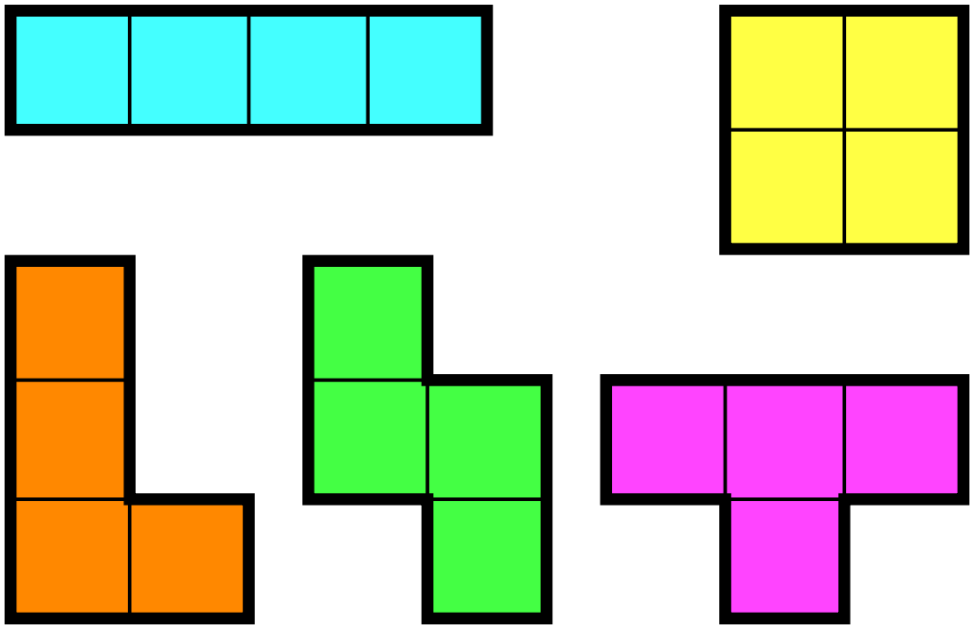
A másik segítség, hogy a játékosok mindegyike kiválaszt egy karaktert a játék elején. Mindegyik karakter különböző speciális képességekkel rendelkezik, szám szerint hárommal. Ezek a képességek abban az esetben használhatók, ha megvan a megfelelő mennyiségű mana a használatukhoz. Minden képességhez van rendelve egy adott manaszám ami a használatához szükséges, és a használt speciális képesség az alapján kerül kiválasztásra, hogy mennyi mana áll rendelkezésre (a legmagasabb költségű képesség ami még belefér a rendelkezésre álló manába).

Mana úgy gyűjthető, ha a játékos egy olyan sort tűntet el, amiben legalább az egyik mező, mana mező. A mana mező a következőképp néz ki:



Alakzatok

A játékban használt alakzatok (tetromino-k) megegyeznek a hivatalos tetris-ben használt alakzatok, amelyek az alábbiak:



Forrás: Wikipedia - Tetromino

Karakterek speciális képességei

Ninja

1. **Wind**
    - A játékos játékterén minden mező jobbra tolódik, ameddig ellenállásba nem ütköznek
    - *Költség – 1 mana*
2. **Shove**
    - A játékos legalsó két sora eltűnik, és az ellenfél játékterénnek alján feltűnik 2 új lyukas sor.
    - *Költség – 2 mana*
3. **Copy**
    - A játékos játéktere az ellenfél játékterének pontos mása lesz
    - *Költség - 3 mana*

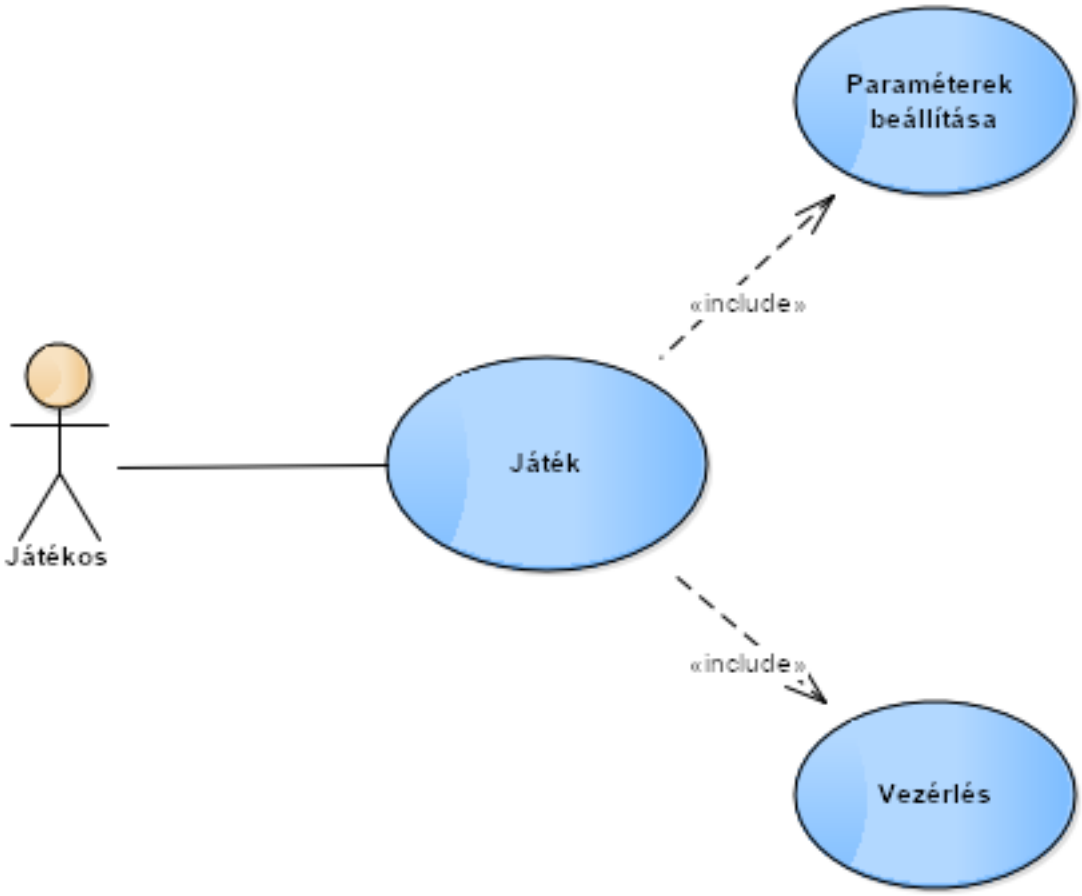
Warrior

1. **Slice**
    - A játékos játékterén a felső 4 sor eltűnik
    - *Költség – 1 mana*
2. **Raise**
    - Az ellenfél játékterének aljához 5 új lyukas sor hozzáadása
    - *Költség – 2 mana*
3. **Swap**
    - A játékos játéktere és az ellenfél játéktere kicserélődik
    - *Költség - 3 mana*

Mage

1. **Erase**
    - A játékos játékterén az alsó 4 sor eltűnik
    - *Költség – 1 mana*
2. **Convert**
    - A játékos aktuális tetromino-ja egy véletlenszerű tetromino-ra változik
    - *Költség – 2 mana*
3. **Mirror**
    - Az ellenfél játékterén minden üres sort telire, és minden teli sort üresre változtat
    - *Költség – 3 mana*

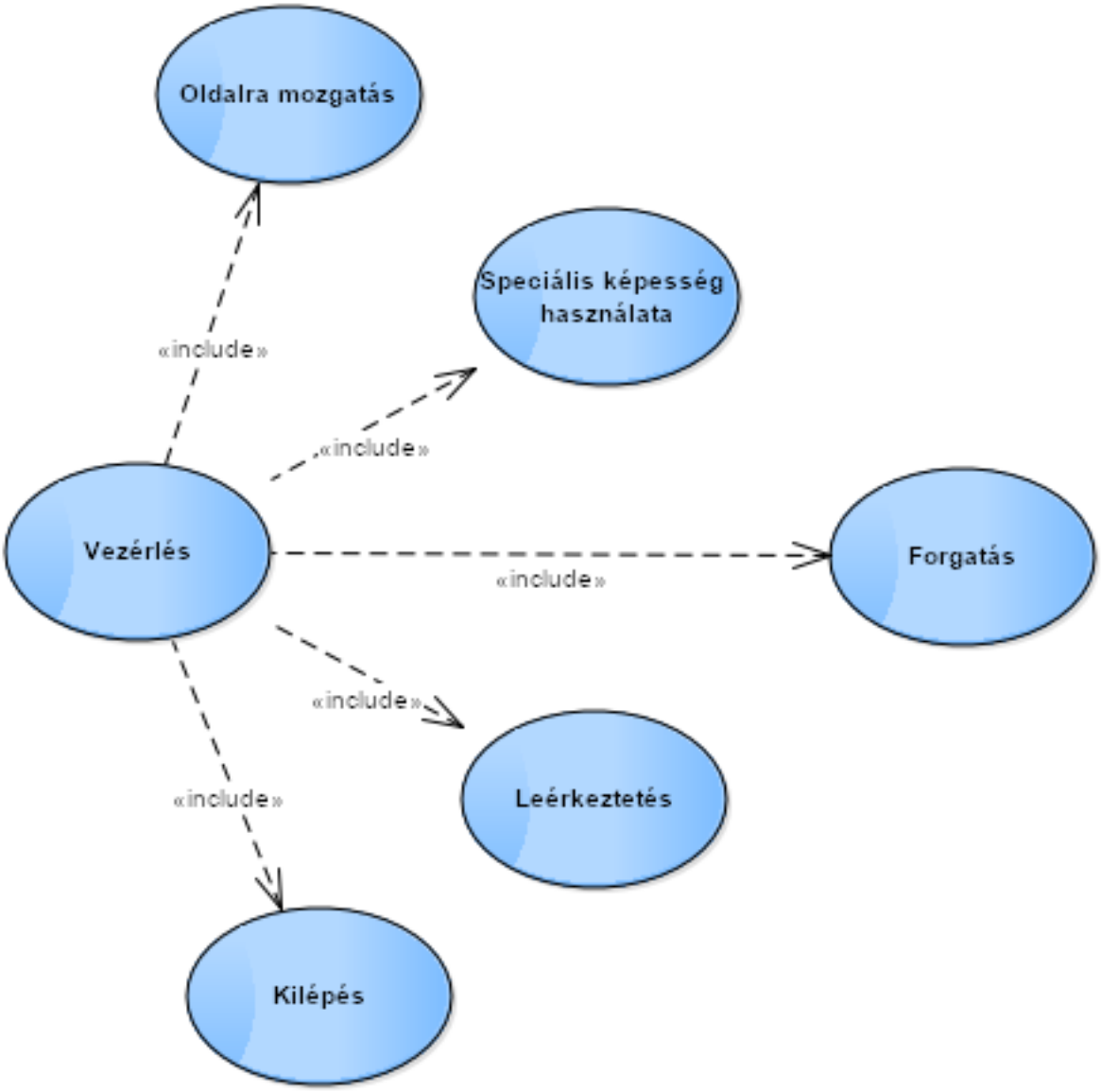
1.4 Használatieset-modell, funkcionális követelmények



A használati eset neve: Játék
Leírás: A játék azonnal elindul a program indítása után, addig tart, míg a játékos ki nem lép. A játékszabályok megtalálhatók az 1.3 fejezetben
Előfeltétel: Paraméterek beállítása
Utófeltétel: Játékos kilép

A használati eset neve: Vezérlés
Leírás: Játékos által játékban használatos valid gomb kerül lenyomásra
Előfeltétel: Folyamatban lévő játék
Utófeltétel: -

A használati eset neve: Paraméterek beállítása
Leírás: A játékos indítás előtt kiválasztja a játékhoz szükséges karaktereket, a játékmódot, valamint hálózati játék esetén beállítja a játékszerver címét.
Előfeltétel: Programindítás
Utófeltétel: -



A használati eset neve: Tetromino mozgatása oldalra
Leírás: Az aktuális tetromino mozgatása egy mezővel a lenyomott irányba
Előfeltétel: A tetromino adott irányba mozgatása nincs akadályozva fal, vagy egyéb alakzat által. Folyamatban lévő játék
Utófeltétel: -

A használati eset neve: Speciális képesség használata
Leírás: Az adott felhasználó karakteréhez, és a meglévő mana mennyiségéhez rendelt speciális képesség végrehajtása
Előfeltétel: Játékosnak legalább 1 manája van. Folyamatban lévő játék

Utófeltétel: -

A használati eset neve: Tetromino forgatása
Leírás: Az aktuális tetromino jobb oldali irányba forgatása 90 fokkal.
Előfeltétel: Folyamatban lévő játék. Az elforgatott tetromino nem ütközik fallal, vagy már leérkezett tetrominoval
Utófeltétel: -

A használati eset neve: Tetromino gyorsított leérkeztetése
Leírás: Tetromino leesésének felgyorsítása az alap esési sebességen felül
Előfeltétel: Folyamatban lévő játék. Aktuális létezik, valamint nincs még leérkezve
Utófeltétel: -

A használati eset neve: Kilépés a játékból
Leírás: Játék bezárása
Előfeltétel: Folyamatban lévő játék
Utófeltétel: A játék nem fut

1.5 Nem funkcionális követelmények

Fejlesztési módszertan:

- Agilis

A fejlesztéshez szükséges hardver:

- CPU: Core i5, RAM: 8GB, videó: 1440x900

A fejlesztéshez használt szoftverek:

- Operációs rendszer: OS X Sierra
- Követelmény elemzés: LibreOffice Writer szövegszerkesztővel
- CASE eszköz: Enterprise Architect 11.0
- Verziókezelő rendszer: Git
- C++ fejlesztőeszköz: Vim
- Python fejlesztőeszköz: Vim

A futtatáshoz szükséges operációs rendszer:

- Tetszőleges operációs rendszer amely támogatja az SDL2 library-t, valamint megtalálható rajta olyan fordítóprogram ami támogatja a C++ C++14-es verzióját, valamint a Python2-őt.

2. Tervezés

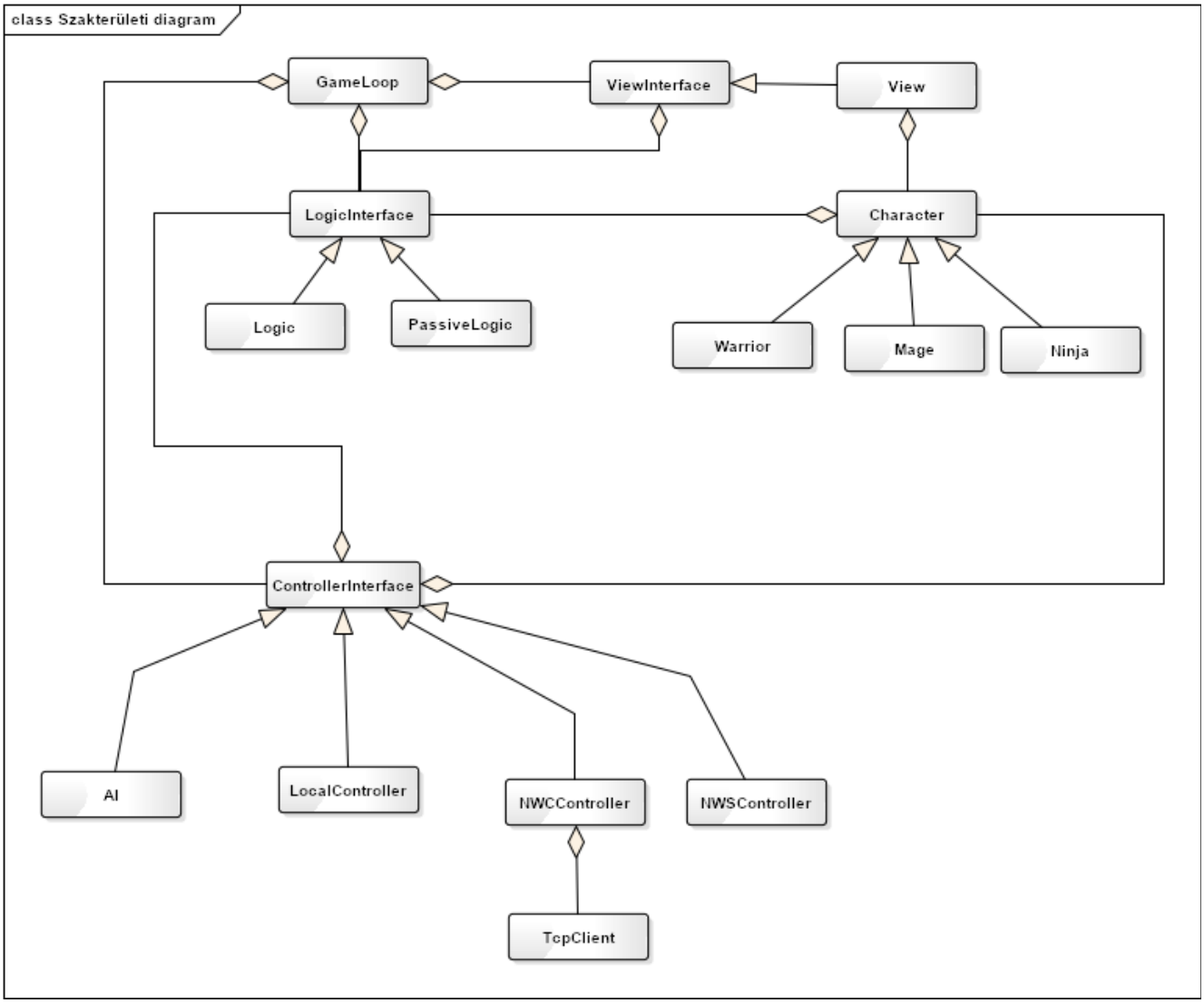
2.1 A program architektúrája

A játék eseményvezérelt architektúrát valósít meg Game Loop programozói minta [1] felhasználásával. Egy központi game loop segítségével figyeli az események bekövetkeztét amelyek lehetnek hálózaton keresztül érkező csomagok, vagy felhasználó által lenyomott gombok. Ezen felül ebben a központi ciklusban értesíti a program megfelelő komponenseit a ciklus újabb iterációjának indulásáról, hogy az ehhez az eseményhez rendelt műveleteket le tudja futtatni az adott komponens.

A hálózati játék kliens szerver architectúrát használ ami abban nyílvánul meg, hogy egy központi szerver alkalmazás fut a háttérben amihez a két kliens csatlakozik, amelyek a felhasználó számítógépén futnak, és rajta keresztül kommunikálnak egymással egy előre definiált üzenetformátummal.

2.2 Osztálymodell

GameLoop



Az osztály ami a program indulásakor jön létre, és tartalmazza a játék főciklusát, amely szabályozza milyen gyorsan hajtódjanak végre a műveletek. Összeköti a különböző komponenseket, meghívja a főbb műveleteiket a megfelelő időben, szabályozza a megjelenítést, valamint poll-ozza a gombnyomásokat, és azokat továbbítja a megfelelő objektumoknak.

ViewInterface

Megjelenítésért felelős interfész. Segítségével lecserélhető a későbbiekben az SDL megjelenítője egy másikra, például konzolos karakteres megjelenítésre, vagy egyszerűen kivehető a megjelenítés, hogy a render ne tegyen semmit

View

Konkrét megjelenítő ami a ViewInterface-ből származik. SDL segítségével képernyőre vetíti a játék állását

LogicInterface

Interface a játék logikájához

Logic

A játék logikája. Tárolja a játékteret, szabályozza hova mozoghat egy alakzat, generál alakzatot, megállapítja ha játék vége van, mozgatja az aktuális alakzatot, valamint eltűnteti a teli sorokat.

PassiveLogic

Passzív logika amit hálózati játékhoz használunk. Működése arra van kihegyezve, hogy a mozgatást, generálást játék vége ellenőrzést nem neki kell csinálni, egyetlen feladata a hálózaton keresztül küldött játéktér és alakzat nyilvántartása, és tárolása

Character

Karakter interface ami tartalmazza a karakterek közös funkcióit, hogy tárolásnál ne kelljen statikusan fix karaktertípust tárolni, hanem választástól függően jöjjön létre a megfelelő.

Warrior

Konkrét karakter, ami megvalósítja a Warrior karakter speciális képességeit

Mage

Konkrét karakter, ami megvalósítja a Mage karakter speciális képességeit

Ninja

Konkrét karakter, ami megvalósítja a Ninja karakter speciális képességeit

ControllerInterface

Iránytásért felelős osztály interface-e

Ai

ControllerInterface-ből származó osztály, ami a mesterséges intelligenciát tartalmazza, és az alapján irányít amit az osztály maga optimálisnak talál

LocalController

ControllerInterface-ből származó osztály ami a lokális játékhoz szükséges logikát tartalmazza. Átvesz egy adott billentyűkiosztást, és ennek segítségével külső input alapján az adott billentyűhöz rendelt irányító akciót hajtja végre.

NWCController

ControllerInterface-ből származó osztály ami a hálózati játékhoz szükséges. Fogadja az üzeneteket a távoli ellenféltől, feldolgozza őket, és azok alapján frissíti az információkat a saját logikájában, hogy az ellenfél játékterét valósághűen tudja megjeleníteni a program.

NWSController

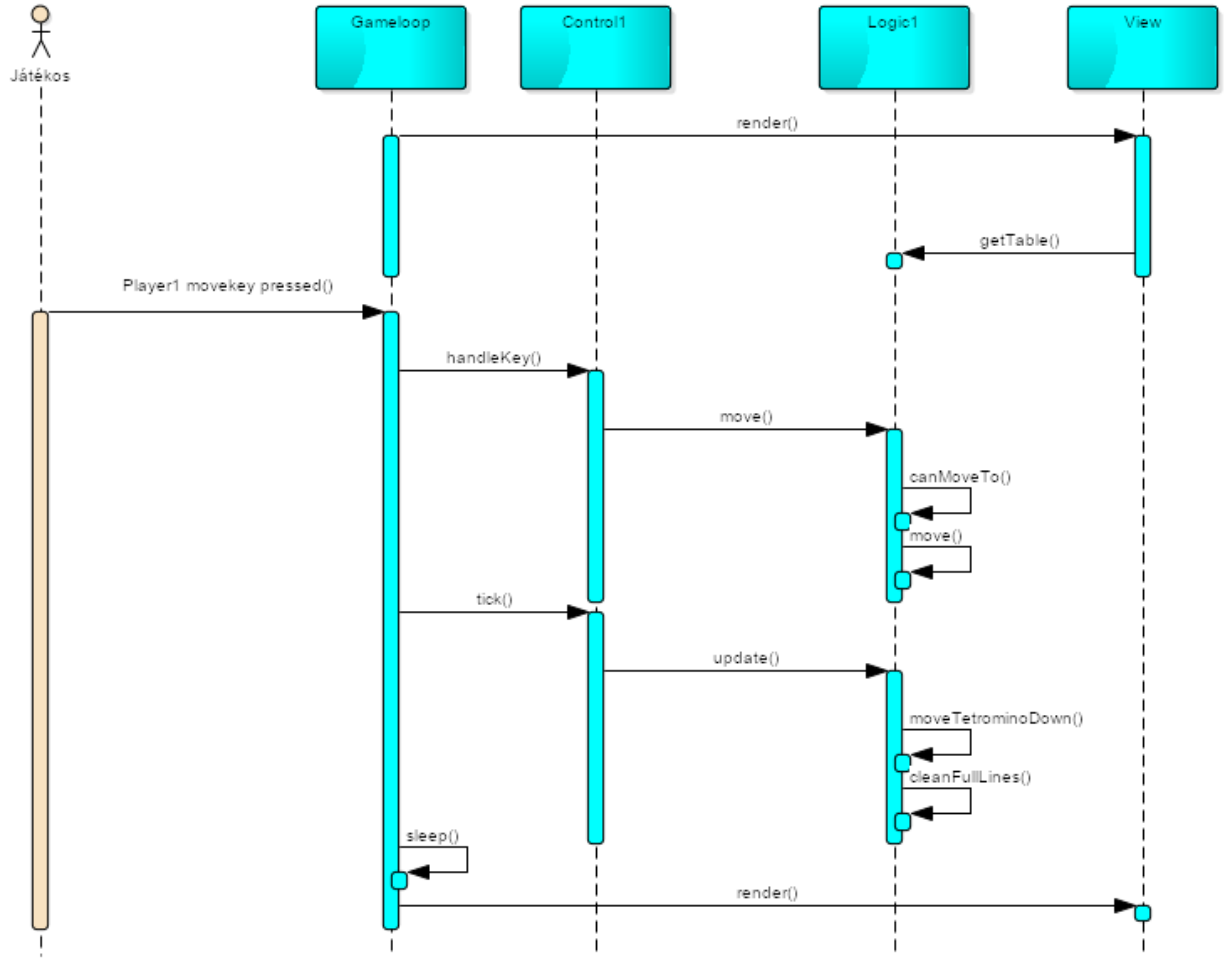
ControllerInterface-ből származó osztály ami a hálózati játékhoz szükséges. Működése a LocalController-hez hasonló, ugyanúgy kap egy adott irányítást és a kapott billentyűinput alapján kitalálja milyen műveletet kell végrehajtani, de ezen felül a feladata, hogy továbbítsa a megfelelő információkat, és a végrehajtott műveleteket a távoli ellenfélnek hálózaton keresztül.

TcpClient

Hálózati csatlakozáshoz használt osztály. Egy adott IP cím, port párosra csatlakozva üzeneteket továbbít és fogad folyamatos TCP kapcsolaton keresztül.

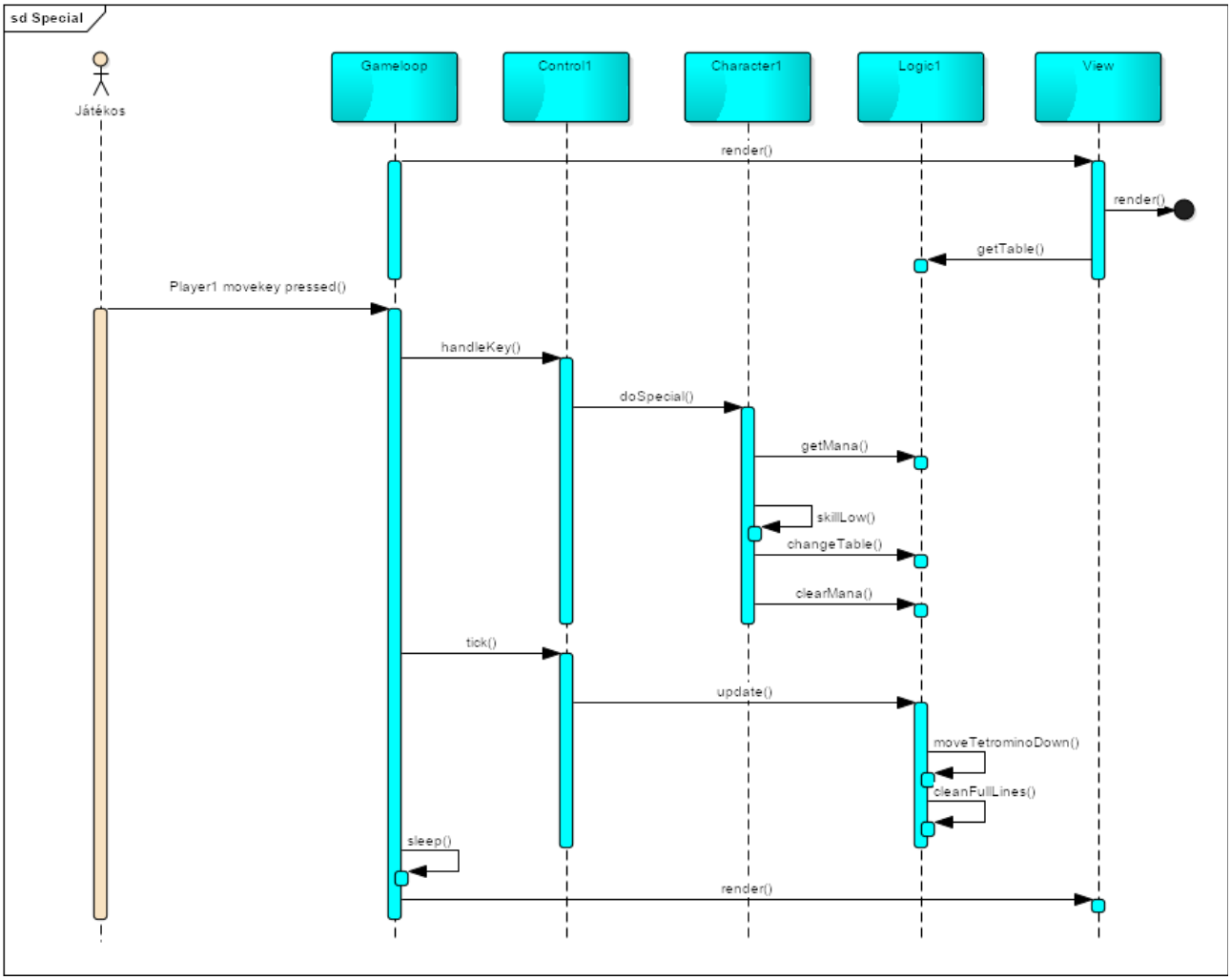
2.3 Dinamikus működés

Mozgatás



A felhasználó által mozgatás gomb lenyomására a program fejében lezajló működés. A felhasználó által lenyomott billentyűt megkapja a GameLoop osztály, azt továbbítja a LocalController-nek, ami lefuttatja a meglelelő mozgatás metódust a Logic osztályon, ami végrehajtja a mozgatást amennyiben az lehetséges. Ezen felül látható az egy gameloop alatt végrehajtott műveletek sorrendje, ami render-el kezdődik, és sleep-el fejeződik be. A forgatás, és ejtés művelet hasonlóan zajlik le, változások csak a logika által végrehajtott metódusban találhatók.

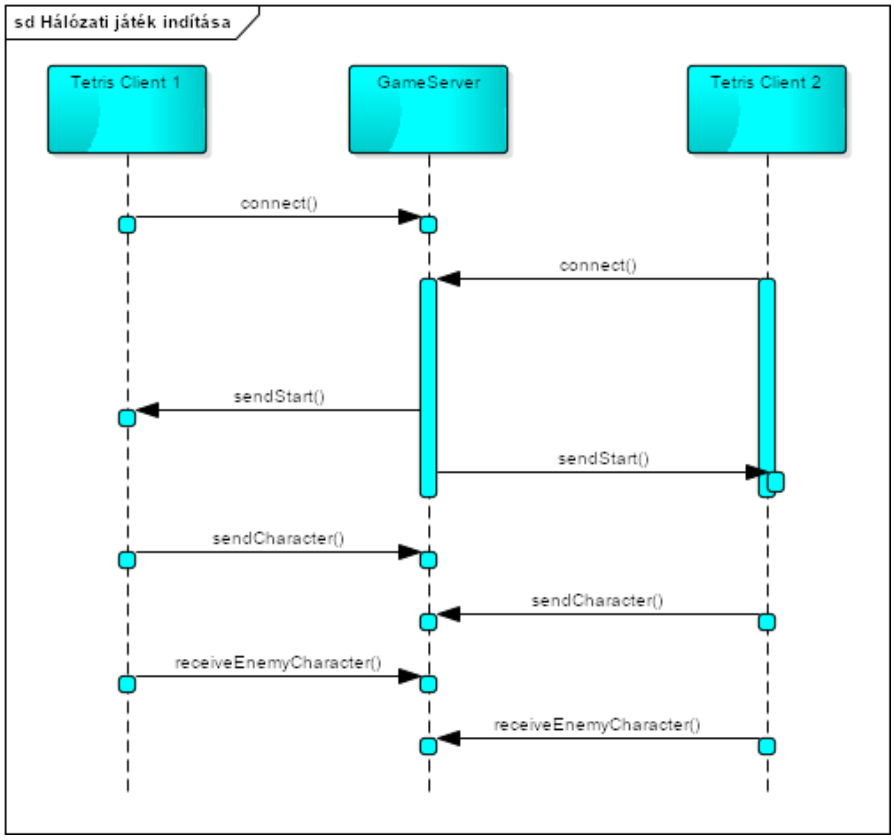
Special



Speciális képesség használatának lezajlása. Hasonló egyéb gombok lenyomásához, például az előzőleg látott mozgatáshoz, de itt a sorrendbe közbeiktatódik a Character osztály is ami ebben az esetben meghívja a Logic osztályt, miután elvégezte a megfelelő ellenőrzésket. A fenti ábrán egy olyan eset található amikor a különleges képesség a saját játékteret módosítja, de lehetséges az ellenfél játékterének módosulása is, ez esetben az ellenfél logikáját hívja, ugyanis a Character osztály ismeri mindkét logikát.

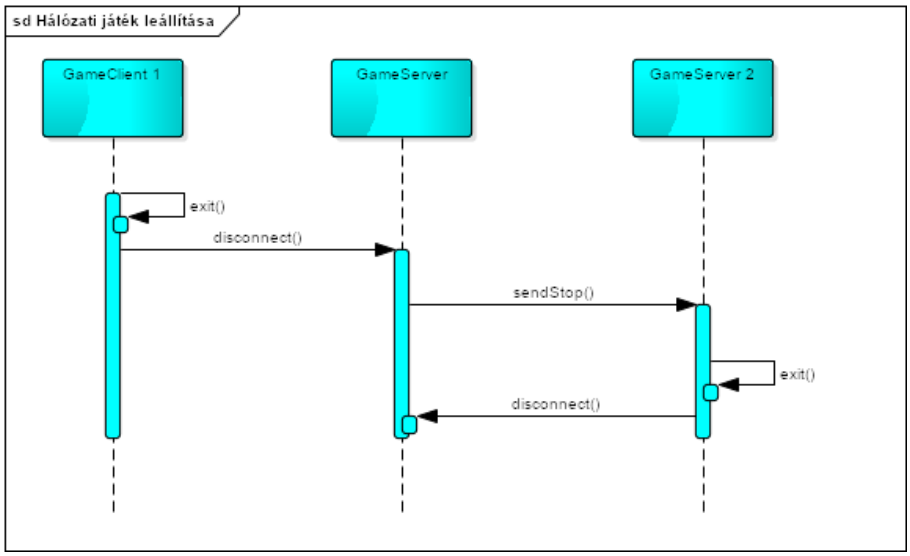
Hálózati játék

Hálózati játék inicializálása



Hálózati játék indítása esetén az első lépés, hogy mindkét kliens csatlakozik a játékszerverhez. Amennyiben a játékszerver azt észlelte, hogy becsatlakozott mindkét játékos, kiküld mindkét kliens-nek egy start üzenetet ami mindkét kliensben azt váltja ki, hogy elkezdik az inicializációt. Az inicializáció első lépése, hogy mindkét kliens elküldi a saját játékosa által kiválasztott karaktert a játékszervernek, ami továbbítja az ellenfélnek. Ezután mindkét kliens fogadja a másik karakterét, és ez alapján létrehozza a megfelelő karaktert az ellenfélnek, és ráteszi az avatar-ját a képernyőre, és a játék elindul.

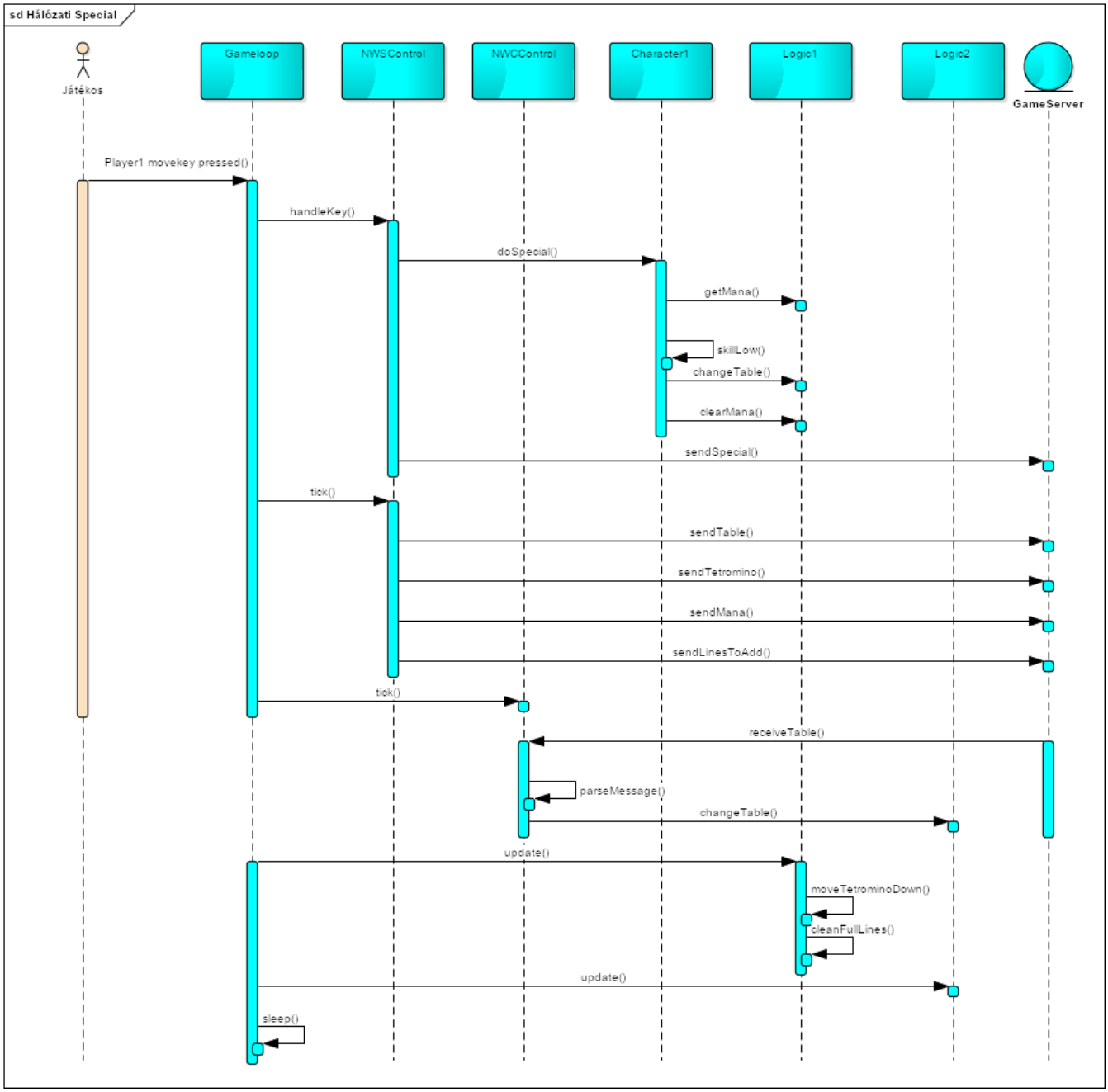
Hálózati játék leállítása



A játék leaállítása a következő módon történik hálózati játék esetén: Az adott kliens-ben a felhasználó kilép a játékból. Ez azt váltja ki, hogy a kliens lecsatlakozik a játékszerverről. A játékszerver ezt észlelve kiküld a megmaradt kliens-nek egy stop üzenetet, aminek a hatására az ellenfélben is meghívódik a kilépés, és értesíti az ellenfelet a program, hogy a kilépés oka a másik fél lecsatlakozása volt.

Hálózati speciális képesség használata

Az alábbi ábra mutatja a játék fejében létrejövő objektumhívásokat hálózati játék alatt, abban az esetben, mikor a felhasználó a speciális képesség használata gombot nyomta le a billentyűzetén, valamint a játékszerver felé kimenő üzeneteket egy GameLoop ciklusiteráció alatt, adott sorrendben.

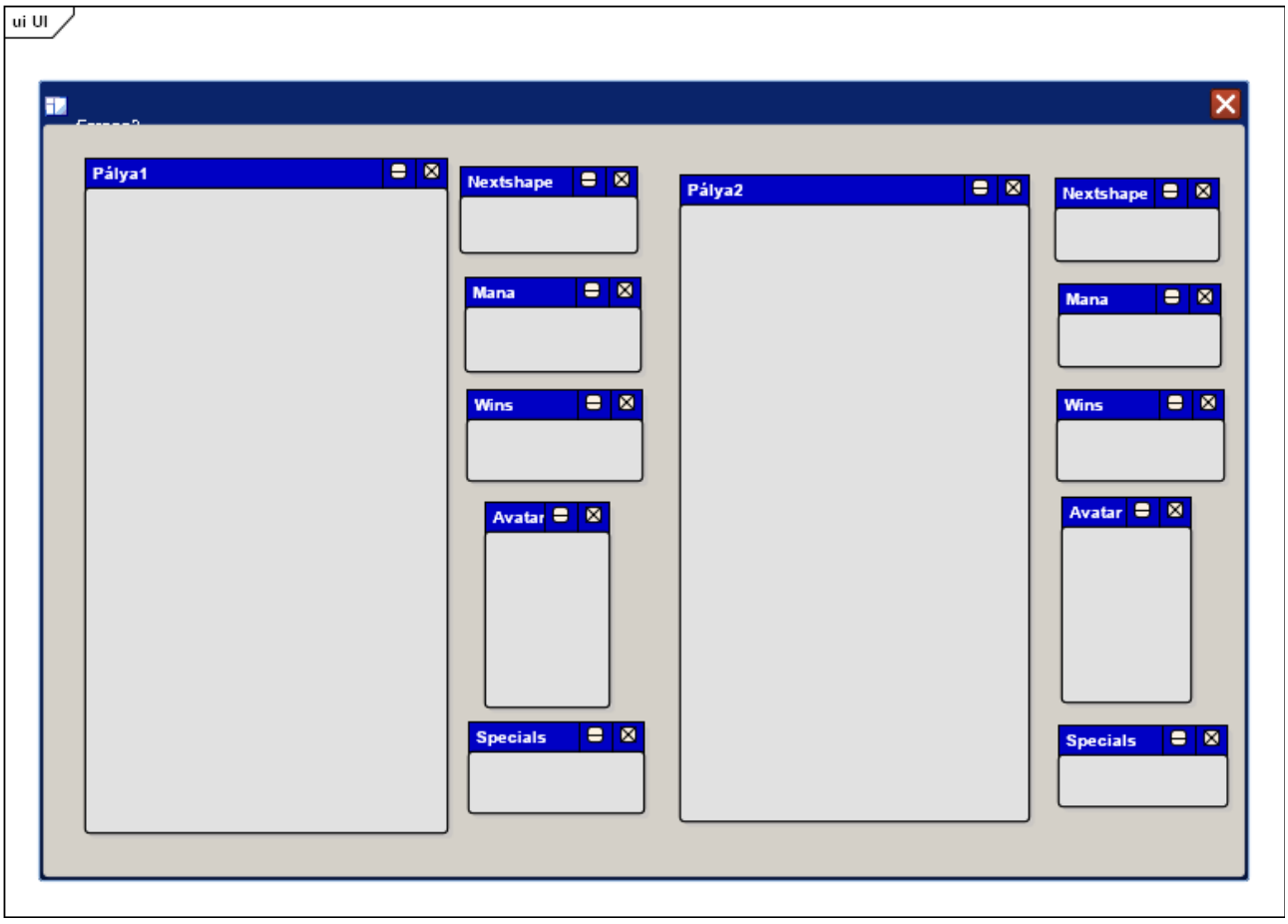


AI működése

Az AI a következő módon működik: A GameLoop minden iterációja kiváltja azt a műveletet ami alapján az AIController kiszámítja melyik lenne az adott helyzetben az optimális lépés. Ezzel a működéssel amennyiben változik a játéktér (például ellenfél speciális képességének használata következményeképp), a változásokat dinamikusan követni tudja, és módosítja az eredeti célt az

aktuális Tetromino lerakásához. Az optimális lépést az alapján állapítja meg, hogy minden lépést pontoz az alapján milyen jó az adott lépés. Ezek közül a pontok közül kiválasztja a legkisebbet minimum kereséssel, és afelé teszi meg a szükséges lépéseket, az alapján forgatja be és mozgatja az alakzatot. Amennyiben a tetromino helyezete és szöge megfelelő, a maradék lépéseiben felgyorsítja az alakzat esését, hogy gyorsabban érkezzen le. Mivel alapesetben ez a gondolkodás túl gyors lenne egy valódi ellenfél ellen, ezért mesterségesen lassítjuk a gondolkodást egy időzítő beiktatásával, amivel szabályozni lehet az AI gyorsaságát. Ezen felül amennyiben az AI veszélyben érzi magát speciális képességet használ. Veszély alatt azt értjük, hogy a pályának egy előre megadott magasságát elérte a játéktér.

2.4 Felhasználói-felület modell



A **Pálya1** és **Pálya2** tartalmazza a két játékos pályáját, amin megtalálható az aktuális irányítható tetromino, valamint a már leérkezett elemek.

A **Nextshape** doboz tartalmazza grafikusan megjelenítve a következő tetromino-t amit a játékos kap miután lehelyezte az aktuálist, ennek a segítségével tud előre tervezni a játékos.

A **Mana** doboz tartalmazza az adott játékos rendelkezésre álló mana-jat, amit az adott játék alatt felhasználhat.

A **Wins** doboz tartalmazza a játékos által nyert körök számát, ami egy minden győzelem után 1-el növekvő számot tartalmaz.

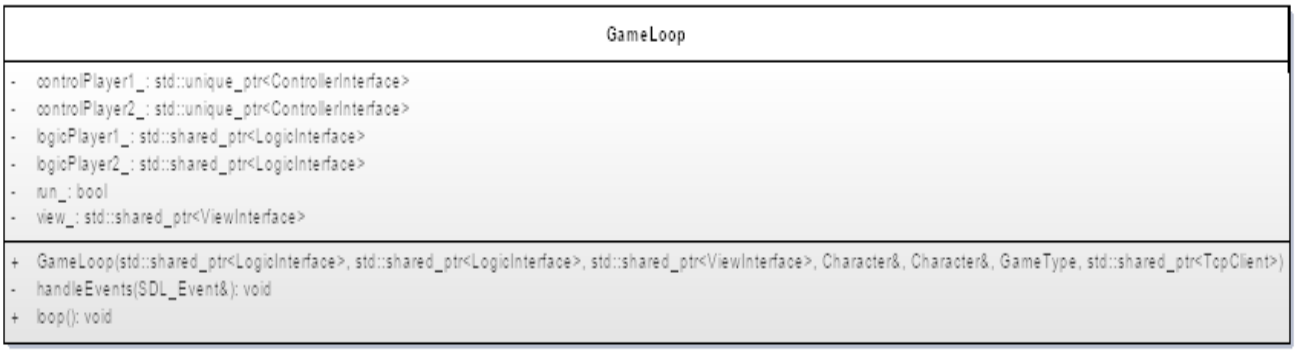
Az **Avatar** doboz az adott játékos által kiválasztott karakter avatar-ját tartalmazza, ami egy kép ami hozzá van rendelve az adott karakterhez.

A **Specials** doboz tartalmazza a játékos által kiválasztott karakter speciális képességeinek nevét a különböző mana költségekhez rendelve szövegesen.

2.5 Részletes programterv

Az itt részletezett osztályok működése és egymással való viszonya korábban definiálásra került, ezért ebben a szekcióban a metódusaik és tagváltozóik kerülnek bővebben kifejtésre

GameLoop



Konstruktor: A konstruktor a kapott paraméterek alapján inicializálja a controller osztályokat, eltárolja a logikákat, összerendeli a logikákat, és a karaktereket az adott controller-ekkel.

handleEvents(): Eseménykezelő függvény, ami továbbítja a lenyomott gombokat az összes kontroller-nek, ezen felül külön kezeli az escape gomb lenyomását, aminek hatására a run_ változó hamissá válik, és a gameLoop leáll. A loop függvényből kerül meghívásra.

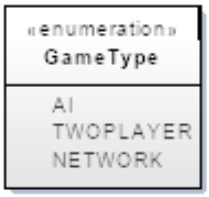
loop() : A program főciklusa. Addig tart a futása amíg a run_ változó értéke igaz, és minden iteráció során végrehatja a működéshez szükséges műveleteket: Események feldolgozása, update hívása, játék végének ellenőrzése, új játék indítása, ciklus késleltetése.

controlPlayer1_, controlPlayer2_ : Az osztály által tárolt control objektumok mindkét játékos számára. Ezeket az osztály a kapott paraméterekből állítja elő

LogicPlayer1_, logicPlayer2_ : Az osztály által tárolt játékosokhoz tartozó logikára való hivatkozás.

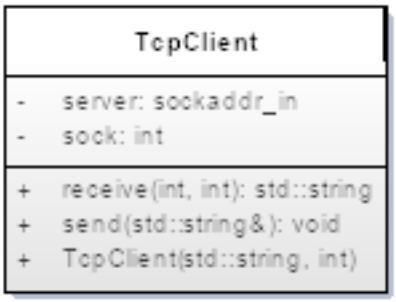
run_ : A futáshoz kapcsolható logikai változó. Igaz értékkel indul, és ha kilépést kezdeményez a felhasználó hamisra áll át.

GameType



GameType segéd enumerátor. Tartalmazza a lehetséges játéktípusokat, amik átadásra kerülnek a GameLoop osztálynak, hogy ennek alapján tudja létrehozni a Controller-eket. AI esetén Ai objektumot gyárt, TWOPLAYER esetén LocalController-t, NETWORK esetén pedig NWCController-t, és NWSController-t.

TcpClient



Konstruktor : a kapott ip-port pároshoz csatlakozik, amennyiben a kapcsolat nem sikerül exception-el kiszáll.

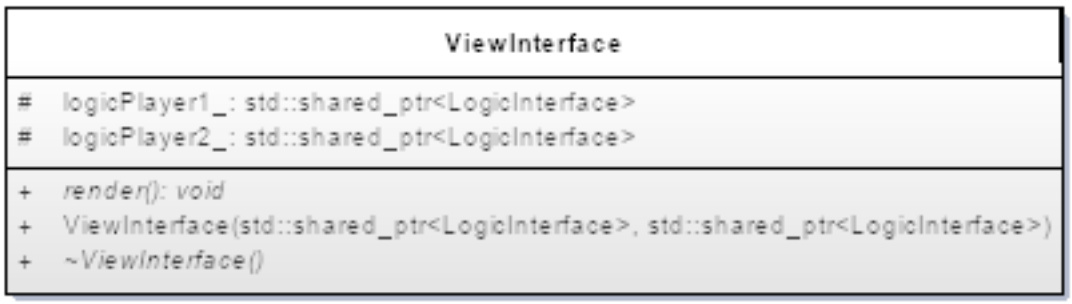
receive() : paraméterben megkapja a maximálisan olvasható adat mennyiségét, valamint a hálózati csomagra várás maximális idejét, és visszaadja string-ként a hálózaton kapott adatot

send() : a paraméterben kapott adatot elküldi a kapcsolaton keresztül

server : a szerver adatait tartalmazó struktúra. (ip, port)

sock : socket fájl deszkriptor

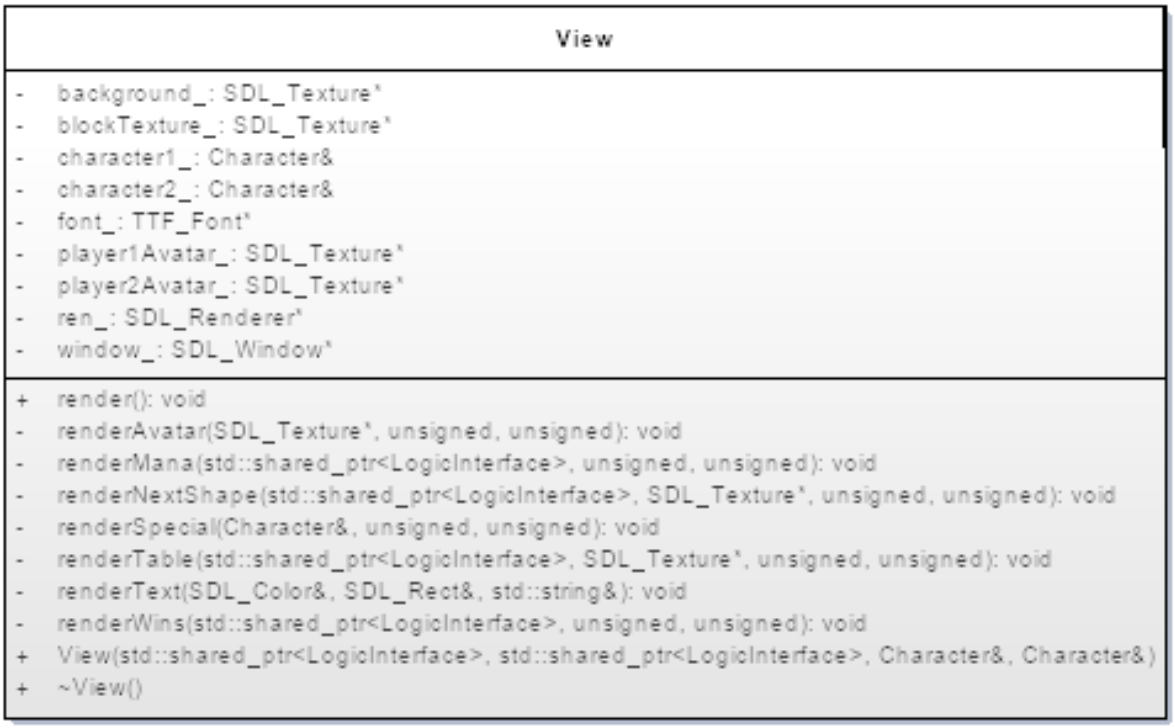
ViewInterface



render() : külső interface a képernyőn való megjelenítés hívására.

logicPlayer1_, logicPlayer2_ : játékoslogika mutató, ami a különböző játékosok logikáját tartalmazza, és lekérhető tőlük a pálya, a mana mennyisége, és egyéb megjelenítéshez szükséges információk.

View



render() : Megjelenítésre szolgáló funkció, megvalósítja az interface-beli render funkciót SDL használatával

renderAvatar() : a render() funkció által használt funkció a játékosok avatar-jának megjelenítésére. Átveszi az avatar-t és a pozíciót ahol az avatárnak el kell helyezkednie az ablakban.

renderMana() : a render() funkció által használt eljárás, ami a kapott LogicInterface-től lekért mana mennyiséget jeleníti meg a játékpályán a megadott pozíción.

renderNextShape() : A játékos következő alakzatának megjelenítésére használt funkció. Paraméternek LogicInterface-t, koordinátát, és egy textúrát kap amiről veheti a különböző színeket, az adott textúra részeinek megjelenítésével.

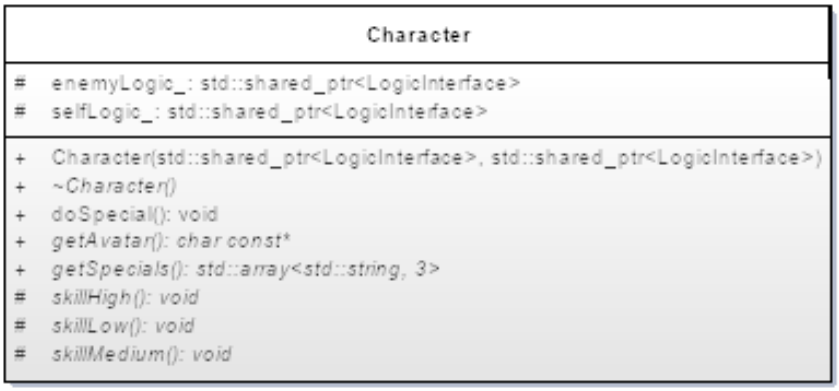
renderSpecial() : Egy adott karakter speciális képességeit jeleníti meg a képernyőn.

renderTable() : Egy adott Logikához tartozó játékteret jelenít meg a képernyőn

renderText() : Szöveg megjelenítésére használt segédfüggvény

r**enderWins()** : Egy adott Logikához tartozó játékos győzelmeinek megjelenítésére használt függvény.

Character



doSpecial() : speciális képesség végrehajtására szolgáló metódus interfész. Adott Controller osztály hívja abban az esetben amennyiben speciális képesség használatára van szükség.

getAvatar() : Az adott karakter avatar-jának fájlnevét lekérdező interfész.

getSpecials() : A speciális képességek nevét visszaadó interfész.

skillHigh() : a doSpecial() függvény által abban az esetben meghívott funkció ha 3 vagy több mana áll rendelkezésre

skillMedium() : a doSpecial() függvény által abban az esetben meghívott funkció ha 2 vagy több mana áll rendelkezésre

skillLow() : a doSpecial() függvény által abban az esetben meghívott funkció ha 1 mana áll rendelkezésre

Ninja

A Ninja karakter specifikus képességeinek és adatainak tárolására, és megvalósítására szolgáló osztály, metódusai megvalósítják a Character osztály metódusait

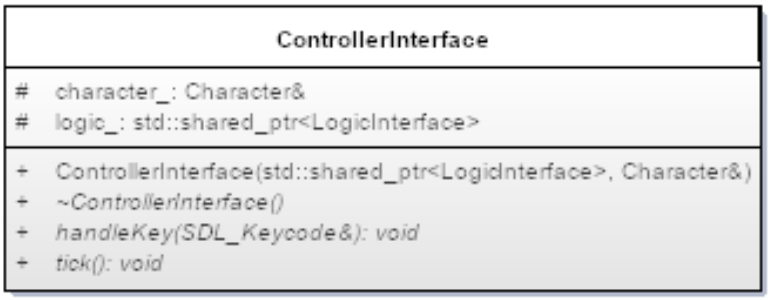
Warrior

A Warrior karakter specifikus képességeinek és adatainak tárolására, és megvalósítására szolgáló osztály, metódusai megvalósítják a Character osztály metódusait.

Mage

A Mage karakter specifikus képességeinek és adatainak tárolására, és megvalósítására szolgáló osztály, metódusai megvalósítják a Character osztály metódusait

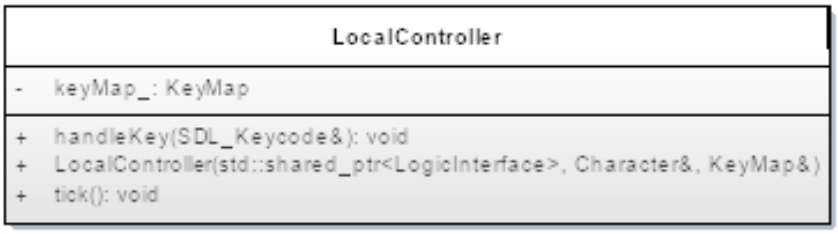
ControllerInterface



tick() : interface ami a megfelelő controller update-je során végrehajtandó feladatokat valósítja meg. A GameLoop loop()-ja hívja minden iteráció során egyszer.

handleKey() : A funkció ami lekezeli a GameLoop által átadott gombokat controller fajtájától függően.

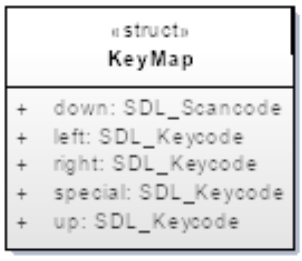
LocalController



tick() : ControllerInterface-ben található tick függvény megvalósítása. Minden iterációban meghívja a logika move() függvényét amivel lefelé mozgatja az aktuális alakzatot amennyiben a lefelé gomb állapota az adott pillanatban lenyomva tartott.

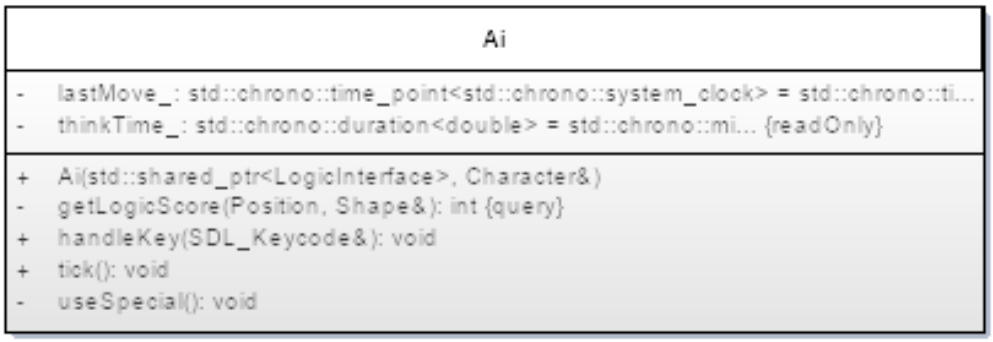
handleKey() : lekezeli a GameLoop által átadott gombnyomásokat, a tárolt KeyMap struktúrában tárolt gombkiosztás alapján

KeyMap



Struktúra ami a különböző akciókhoz rendel gombokat. A Controller ez alapján a struktúra alapján tudja, adott gombra milyen művelet futtatása szükséges.

Ai



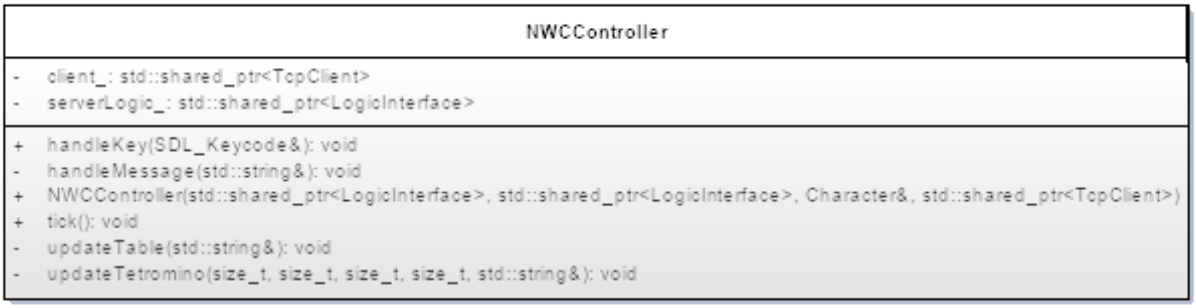
tick() : a ControllerInterface tick() függvényének AI specifikus megvalósítása. Kiszámítja az optimális lépést az adott helyzetben, és az alapján teszi meg a megfelelőt, amennyiben a lastMove_ óta (utolsó Ai által végrehajtott művelet ideje) eltelt a thinkTime_ ami azt tartalmazza két lépés között mennyit gondolkozzon a mesterséges intelligencia.

handleKey() : ebben a megvalósításban ez a függvény nem csinál semmit, csak egy üres hívás

getLogicScore() : A függvény ami megmondja egy adott pozíciót mennyire pontoz a mesterséges intelligencia. Minél kisebb az érték annál jobb a lépés.

useSpecial() : a tick()-en belül kerül meghívásra. Ez a függvény felelős azért, hogy mikor használja fel a különleges képességét a mesterséges intelligencia.

NWCController



handleKey() : ebben a megvalósításban ez a függvény nem csinál semmit, csak egy üres hívás

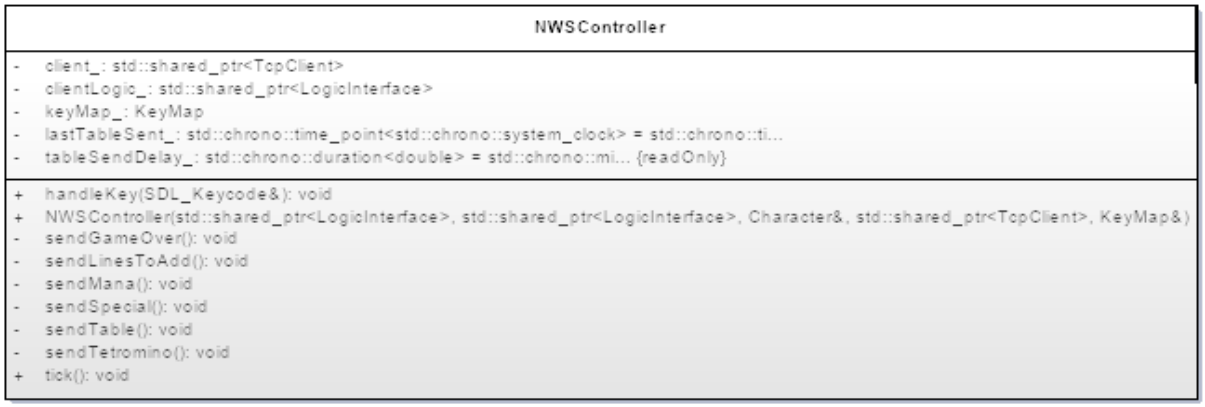
handleMessage() : a hálózaton keresztül kapott üzenetet feldolgozó funkció, ami feldolgozza az üzenetet, és eldönti mit kell vele csinálni.

updateTable() : Amennyiben táblát kapott a kliens frissíti a fejében lévő Logika játékterét

updateTetromino() : Amennyiben tetromino-t kapott a hálózaton keresztül frissíti a Logika aktuális tetromino-ját vele.

tick() : Üzenetet fogad hálózaton keresztül a távoli játékszervertől, darabokra osztja, és feldolgozza a kapott üzentet.

NWSController



tick() : A ControllerInterface tick() függvényének megvalósítása. Minden iteráció során amennyiben letelt a legutolsó küldés óta lastSendDelay_-nyi idő elküldi a játékos tetromino-ját, tábláját, mana-ját, az eltűntetett sorok számát, valamint, hogy vége van-e a játéknak. Ezen felül a LocalController tick() függvényének működését is megvalósítja.

sendTetromino() : Továbbküldi a hálózaton a játékos aktuális tetromino-ját

sendTable() : Továbbküldi a hálózaton a játékos aktuális játékterét az aktuális tetromino nélkül.

sendSpecial() : Amennyiben special képesség került használatra továbbküldi hálózaton keresztül.

sendMana() : A játékos aktuális manájának mennyiségét küldi tovább a hálózaton

sendLinesToAdd() : A játékos által eltűntetett sorok számát küldje át hálózaton keresztül.

sendGameOver() : A játék végéről küld értesítést a másik játékosnak

handleKey() : a GameLoop-tól kapott gombot kezeli le a LocalController-hez hasonlóan azon felül, hogy speciális képesség használatakor hálózaton keresztül továbbküldi ezt az információt az ellenfélhez.

LogicInterface

addPlusLine() : interfész új sor hozzáadásának hívásához

canMoveTo() : Megadja, hogy egy adott alakzat tehető-e a megadott pozícióra, vagyis nem-e ütközik akadályba, vagy kerül átfedésbe egy teli mezővel

changeCurrentShape() : Az aktuális alakzatot megváltoztatja a paraméterben kapottra.

changeTable() : A játékteret megváltoztatja a paraméterben kapottra táblára.

clear() : A tábla minden mezőjét üresre állítja

clearMana() : A mana mennyiségét 0-ra állítja

clearTable(): interfész a tábla ürítés híváshoz

enemyClearedLine() : Sor eltűntetésekor hívott metódus. A linesToAdd_ értékét megnöveli.

finished() : Interface annak lekérdezésére, hogy az adott játék véget ért-e

generateNewCurrentShape(): interface új currentShape_ generálására

linesAdded() : lenullázza a linesToAdd_ értékét

move() : interface alakzat mozgatására

newGame() : új játék kezdéséhez interface

pointIsEmpty() : lekérdezi, hogy egy adott pont a játéktáblán üres-e

removeLine() : interface sor törléséhez

removeTopLines() : interface a paraméterben átadott mennyiségű sor törléséhez a játéktér tetejéről

resetCurrent() : currentShape_ lecserélése nextShape-re interface

rotate() : interfész alakzat forgatásához

setEnemy() : interfész ellenség beállításához

setFinished() : Interfész játék végének beállításához.

setMana() : currentMana értékét állítja

Logic

addPlusLine() : új sor hozzáadása a játéktérhez. Megnöveli 1-el a linesToAdd_ értékét, és az aktuális alakzat lehelyezésekor hozzáadja a sort.

addLine() : sor hozzáadása a játéktérhez

cleanFullLines() : teli sorok törlése

cleanLine() : adott sor törlése

clearStats() : clearMana() hívása

clearTable() : tábla mezőinek üresre állítása

finished() : megnézi a jelenlegi alakzat ütközik-e a játéktérrel, ha igen azt jelenti az új alakzat nem is rakható le, ezért ez jelzi a játék végét.

generateNewCurrentShape() : új currentShape_ generálása véletlenszerűen előállítva

getNextShape() : A következő alakzat lekérése

landCurrent() : resetCurrent(), cleanFullLines() és a hozzáadandó sorokat hozzáadja a játéktér-hez

move() : aktuális alakzat mozgatása a megadott irányba a megadott lépésközzel

newGame() : új játék kezdése, a tábla, és adott játékhoz kötődő értékek nullázása

removeLine() : Egy sor törlése a játéktér aljáról

removeTopLines() : megadott számú sor törlése a játéktér tetejéről

resetCurrent() : nextShape_ kicserélése currentShape_-el, és currentShape_ újragenerálása. A landCurrent() hívja

rotate() : forgatja a jelenlegi alakzatot.

setEnemy() : beállítja az ellenfelet

update() : mozgatja egyel lejebb a jelenlegi alakzatot, ha kell leérkezteti

PassiveLogic

addPlusLine() - üres függvényhívás

clearTable() - üres függvényhívás

finished() - visszaadja a finished_ változó értékét

generateNewCurrentShape() - üres függvényhívás

getNextShape() - nullptr-t visszaadó függvény

move() - üres függvényhívás

newGame() - növeli a gamesWon_ értékét ha nem finished_, és beállítja a finished_ értékét hamisra

removeLine() - üres függvényhívás

removeTopLines() - üres függvényhívás

resetCurrent() - üres függvényhívás

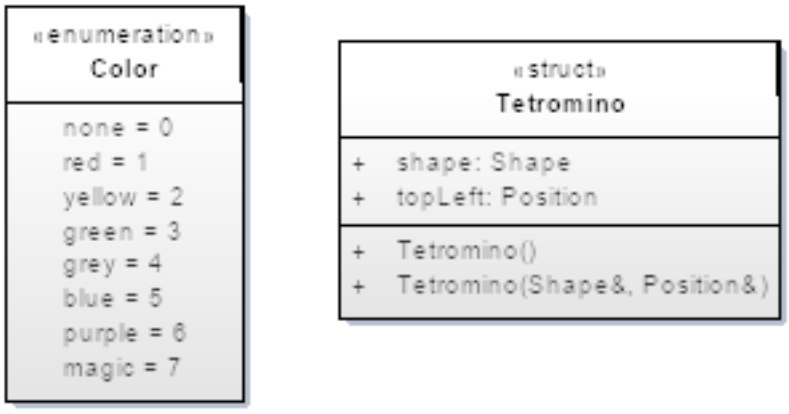
rotate() - üres függvényhívás

setEnemy() - üres függvényhívás

setFinished() - beállítja a finished_ változót

update() - üres függvényhívás

Logic segédosztályok



Segédosztályok a Logika osztályokhoz. A Shape egy vector aminek az elemei Color típusúak. A Color az adott mező színét jelzi. A none azt jelenti, hogy a mező üres, a magic, hogy a mező mana-t tartalmaz, a többi pedig a nevének megfelelő színt jelöli.

A Tetromino egy aott Shape és egy pozíció párosából áll ami azt jelzi, hogy a bal felsó mezője hol helyezkedik el a játéktéren. A Konstruktorban üres paraméterezés esetén véletlenszerűen generál egy alakzatot, vagy a konstruktorban kapott alakzatot, és pozíciót állítja be. Az alakzatok előre definiáltak, azok közül generálja ki amennyiben véletlenszerűen állítja őket elő.

3. Implementáció

3.1 Fejlesztőeszközök

A fejlesztés agilis módszertannal történik, aminek a segítségével több iteráción keresztül a kód folyamatosan közeledik az elvárt működés felé. A hibák és megvalósítandó funckionalitások menedzselése issue-k formájában történik. Ezen issue-k egy-egy olyan problémát tartalmaznak, amik megoldása várhatóan maxálisan 1-2 nap alatt megejthető. Amennyiben egy probléma megoldása hoszabb időbe kerülne, úgy az adott probléma több kisebb részfeladatra kerül osztásra, és azokból keletkeznek az issue-k. A sprint-ek 2 hetes intervallumokban történnek, és minden ilyen iteráció milestone-jába az adott sprint során megoldandó issue-k kerülnek, és minden sprint célja, hogy a 2 hét végére a hozzásorolt issue-k lezárásra kerüljenek.

Az adott sprint issue-inak menedzselésére Kanban tábla kerül alkalmazásra, amelyben megtalálható minden hozzárendelt issue, és követhető rajta a státusza az adott issue-nak, a tábla megfelelő sorába helyezve az issue-t.

A programkód menedzselése a GitHub nevű repository hosting service felhasználásval Git verziókezelőben történik, ami a verziókezelésen kívül segítséget ad az Agilis fejlesztés során használt sprint-ek, release-ek, issue-k menedzselésében, valamint tartalmaz beépített Kanban táblát.

A szoftver fejlesztése során annak érdekében, hogy a programkód több operációs rendszeren is fordítható legyen, a CMake program kerül felhasználásra, ami compiler-függetlenül menedzseli a build process-t, ezzel lehetővé teszi több fordító és több operációs rendszer egyidejű használatát.

Ez a tulajdonság kihasználásra kerül a program automatikus build-je során, amihez Travis CI nevű szervíz kerül felhasználásra, ami lehetővé teszi, hogy összekötve a GitHub verziókezelővel minden commit, push és pull request hatására automatikusan a konfigurációhoz használt yamel fájlban definiált build-ek és tesztek fussanak le, valamint tag hatására a build artifact-ot fel tudja tölteni a GitHub release nyílvántartásába, hogy bármikor letölthető legyen.

A fejlesztés során a C++ nyelv 2014 decemberében megjelent C++14-es verziója kerül felhasználásra, amely a programozási nyelv legfrisebb verziója.

A szoftver által használt egyetlen külső fejlesztői könyvtár a megjelenítéshez használt SDL (Simple DirectMedia Layer) 2-es verziója, amely egy keresztplatformos fejlesztőkönyvtár, ami réteget nyújt a szoftver és a számítógép multimédia hardvereszközei között.

3.2 Alkalmazott kódolási szabványok

A kódoláskor alapvetően a C++ Core Guidelines [2] az irányadó az alábbi pontosításokkal:

Általános szabályok

A forráskódban szereplő sorok hossza legfeljebb 120 karakter lehet. Ha kell, sortörést mindig operátor után iktatunk be.

- Jobban áttekinthető, ha nem kell görgetni.
- Kikényszeríti a túl mély szerkezetek kikerülését. Ha elértünk 3-4 indentálási mélységet, akkor fontoljuk meg egy új függvény vagy metódus bevezetését inkább.

Forráskódban használjunk angol nyelvet.

- Így nemzetközi lesz a kód, ha olyan kell fejlessze, aki nem tud magyarul, az is egyszerűen megteheti.
- Forráskódban kerüljük a nem ASCII karakterek használatát – kommentben is.

Névadás

Angol neveket adjunk.
Csak ASCII karaktereket alkalmazzunk.
Törekedjünk az olvashatóságra.
A nevekben ne aláhúzással, hanem nagybetűkkel válasszuk el a szavakat (kivétel: konstansok).

- Objektum orientált kódoknál ez megszokott.
- Így a nevek rövidebbek lesznek egy kicsit.

Header fájlba csak deklarációk kerülhetnek, kivéve sablonok és rövid inline-ok (getter-ek / setter-ek)
C++ forrásfájlban lehet definíció és deklaráció is – ez utóbbi természetesen csak akkor, ha csak lokálisan használt elemet deklarálunk
Általában is törekedni kell arra, hogy csak abban a legbelső scope-ban deklaráljunk egy elemet (pl. változót), ahol használjuk.

Fájlnevek

A fájl nevét a benne szereplő osztály adja. A fájl neve legyen pontosan ugyanolyan kis- és nagybetűkkel írva, ahogy az osztály neve is.

A fájlok kiterjesztései

A C-ből is használható header fájlok kiterjesztése: .h
Csak C++-ból hívható header fájlok kiterjesztése: .hh
A C++ forrásfájlok kiterjesztése: .cc

Fordítási direktívák

A fordítási direktívák első karaktere (a kettőskereszt) kerüljön a sor első pozíciójába.

Az include direktívák sorrendjét úgy válasszuk meg, hogy a header fájlok között a speciálistól haladjunk az általános felé. Tehát először a saját header fájlokat, majd a programkönyvtárak header-jeit, majd a rendszer header-eket include-oljuk.

- Így kibukik, ha egy header-ben valami olyanra hivatkozunk, ami még nincs deklarálva valahol.

Ha elegendő, akkor előzetes deklarációt használjunk include helyett. Az include-ok spórolásával gyorsabbá tesszük a fordítást.
Macskakörömmel az aktuális (C fájl-lal megegyező) könyvtárban lévő fejléc fájlokat include-oljuk, az -I után megadott könyvtárakból elérhető header-ek esetén pedig relációs jelek közé tesszük a header nevét a beágyazás során.
Törekedjünk arra, hogy csak a minimálisan mindenképp szükséges mennyiségű include könyvtár legyen megadva a fordításnál (kevés -I a CPPFLAGS értékében a Makefile-okban).
Az include könyvtár(ak) legyen(ek) a projekt fa alatt, annak gyökeréhez lehető legközelebb.

Header fájlban ne használjunk using namespace-t. Felesleges és kiszámíthatatlan névtér szennyezést okoz.

Típusok elnevezése

A típusok nevei főnevek.
A típus nevén belüli elkülöníthető szavakat kezdjük nagybetűvel, a nem szókezdő betűk pedig legyenek kisbetűk.
A típusnevek első karaktere kötelezően nagybetű.

Something, TreeNode, PhoneCardEntry

A típusnevekben szereplő rövidítések szónak számítanak, vagyis csak első betűjük nagybetű, a többi kicsi.

- Másként konfliktusokhoz vezetne. A változóneveket úgy kéne deklarálni, hogy pl. hTML, ami nem igazán olvasható, főleg ha a következő szó első nagybetűjét is hozzávesszük. A szavak nem különülnének el úgy, ahogy kellene, és ez nagy mértében rontaná az olvashatóságot.

HtmlTreeNode, MyWysiwygEditor

Ne alkalmazzunk aláhúzás jeleket a szavak elválasztására.

Mutatók és referenciák

Lehetőleg minél kevesebb helyen alkalmazzunk mutatókat.
Ha egy változóra történő hivatkozást kell alkalmazni, akkor csak abban az esetben használjunk mutatót, ha van értelme annak, hogy a hivatkozás NULL legyen. Ha a hivatkozás nem lehet NULL, akkor használjunk referenciát.
A pointer érvénytelenségét a !ptr szerkezettel ellenőrizzük. (Tehát hogy a pointer nulla-e.)

Ne használjunk egymásba ágyazott referencia-mutató szerkezeteket. Ezek olvashatatlanná teszik a kódot:

*char *& something;*

Változók elnevezése

A változók nevei főnevek.
A változó nevén belüli elkülöníthető szavakat kezdjük nagybetűvel, a nem szókezdő betűk pedig legyenek kisbetűk.
A változónevek első karaktere kötelezően kisbetű.
A változónevekben szereplő rövidítések szónak számítanak, vagyis csak első betűjük nagybetű, a többi kicsi.
Ne alkalmazzunk aláhúzás jeleket a szavak elválasztására.

htmlTreeNode, myWysiwygEditor

A globális változókat mindig :: operátorral hivatkozzuk.
A globális változók használatát kerüljük, ajánlott a singleton-ok alkalmazása.
Az általános célú változók neve egyezzen meg a típusával (persze a kis kezdőbetűtől eltekintve).
A nevek hosszúsága legyen arányban a hatókörük nagyságával.

Konstansok elnevezése

C++-ban a define előfordító direktívával létrehozott konstansok használata kerülendő. Használjunk const-ot, vagy enum-ot.
A konstansok és enum értékek elnevezése megfelel a változóknál látottaknak.

const int someNumericLimit = 10;

Tagváltozók

Az osztályok tagváltozóinak nevét aláhúzás jellel zárjuk.

- Így látszani fog pusztán a neve alapján, hogy tagváltozóról van szó.

A tagváltozók deklarációjánál a nagyobb láthatóságú tagváltozókat vegyük előre. Tehát először a sorrend legyen a következő: public, protected, private.
Kerüljük a publikus tagváltozók használatát az adatrejtés elve miatt. Inkább alkalmazzunk publikus getter setter metódusokat.

Függvények elnevezése

A függvények nevei igék.
A függvények nevén belüli elkülöníthető szavakat kezdjük nagybetűvel, a nem szókezdő betűk pedig legyenek kisbetűk.

A függvénynevek első karaktere kötelezően kisbetű.
A függvénynevekben szereplő rövidítések szónak számítanak, vagyis csak első betűjük nagybetű, a többi kicsi.
Ne alkalmazzunk aláhúzás jeleket a szavak elválasztására.

Tagfüggvények

A tagfüggvények nevei ne tartalmazzák az őket tartalmazó osztály nevét, ez csak felesleges redundancia.

```
line.getLength(); // NOT: line.getLineLength();
```

Blokkok

A blokkokban lévő kapcsos zárójeleket a K&R szintaxis szerint helyezzük el. Ez alól kivételt alkotnak a függvények és metódusok definíciói, valamint az osztálydeklarációk, ahol az ANSI-C szintaxist használjuk, vagyis itt a kapcsos zárójelek azonos oszlopba kerülnek, azaz az első oszlopba.

```
void
setLimit(int limit)
{
        if (limit > 0) {
                limit_ = limit;
        }
}
```

Egy sorba csak egy utasítást tegyünk

If

K&R szerint:

```
if (bank.isRobbed()) {
        chooseAnotherBank();
} else {
        bank.visit();
}
```

Goto

Nem használunk goto-t.
C++-ban már a hibakezelésre megfelelő megoldást kínál a try-catch, így a goto feleslegessé vált. Ezen felül elrontja a program strukturáltságát.

Whitespace

Minden blokk növeli a bal oldali behúzást egy egységgel, kivéve a névterek és az osztálydeklarációk blokkjai, illetve az extern C blokkja.
A behúzási egység egy tabulátor karakter.
Behúzáshoz nem használunk szóközt, tabulátort pedig csak behúzáshoz használunk.
A két whitespace karakter szerepe élesen elválik. A tabulátor vezérlőkarakterként pusztán a behúzásért felelős, a szóköz pedig a tartalom szerves része.
A kifejezésekben szereplő operátorok köré egy-egy szóköz kerül.
Zárójel pár belső oldalára nem kerül szóköz, külső oldalára viszont igen, kivéve ha a kettő egybe esik.

```
(a + b * (c + d));
```

Vesszők és pontosvesszők után mindig szóköz áll, kivéve ha sorvégi.

```
for (i = 0; i < 10; ++i) {
}
```

A C++ fenntartott szavait kövesse egy whitespace karakter.
Sorok végén nem lehet whitespace.
Tabulátor után nem állhat szóköz karakter.
Ha egy zárójelben szereplő tartalom olyan hosszú, hogy ezzel sérül az egy sorra vonatkoztatott hossz határ, akkor a zárójel tartalmát külön blokk-ként kell kezelni, hogy az új, mélyebb szintű indentálási egységet alkosson. Vagyis a két zárójelet külön sorba kell elhelyezni közötte a tartalommal.
Egymást nem követhet két üres sor.
Blokkon belül csak az elkülöníthető jelentésű részek közé tegyünk új sort.

Kommentek

A kommentek kerülendők, cél a kód olvashatóvá tétele kommentek használatának szüksége nélkül. Amennyiben valahol komment szükséges, helyette a kód olvashatóságát kell javítani. A változónevek, és függvénynevek utaljanak funkcióikra.

Értékadás

A mágikus számok kerülendők. A 0 és 1 kivételével a számokat konstansok formájában adjuk meg.
A lebegőpontos számokat mindig ponttal adjuk meg. Így kiemeljük a lebegőpontos és az egész aritmetika különböző természetét. Matematikailag a két modell teljesen különböző és nem kompatibilis egymással.

double total = 0.0; // NOT: double total = 0;
double speed = 3.0e8 // NOT: double speed = 3e8;

A lebegőpontos konstansokban a pont előtt mindig álljon számjegy.

double total = 0.5; // NOT: double total = .5;

Használjunk a típusuknak megfelelő nulla értékeket

- egészre 0, karakterre \0, lebegőpontosra 0.0, kivéve NULL.

A NULL használatát inkább kerüljük, mert libc változatonként eltérő módon került definiálásra, és kasztolási hibákat tud eredményezni így. Helyette a C++11-ben bevezetett nullptr legyen használva

4. Tesztelés

4.1 Tetromino mozgatás jobbra

Teszt lépés: Felhasználó megnyomja a jobbra mozgatáshoz kiosztott billentyűt

Elvárt eredmény: a tetromino egy egységnyivel jobbra tolódik, az esés sebessége nem változik

4.2 Tetromino mozgatás balra

Teszt lépés: Felhasználó megnyomja a balra mozgatáshoz kiosztott billentyűt

Elvárt eredmény: a tetromino egy egységnyivel balra tolódik, az esés sebessége nem változik

4.3 Tetromino mozgatás jobbra akadályozás esetén

Teszt lépés: Felhasználó megnyomja a jobbra mozgatáshoz kiosztott billentyűt miközben az alakzat jobb oldalán található közvetlenül a játéktér széle

Elvárt eredmény: a tetromino nem mozdul, az esés sebessége nem változik

4.4 Tetromino mozgatás balra akadályozás esetén

Teszt lépés: Felhasználó megnyomja a balra mozgatáshoz kiosztott billentyűt miközben az alakzat bal oldalán található közvetlenül a játéktér széle

Elvárt eredmény: a tetromino nem mozdul, az esés sebessége nem változik

4.4 A Tetromino leesésének megvárása beavatkozás nélkül

Teszt lépés: A felhasználó megvárja amíg a forma leesik

Elvárt eredmény: A tetromino beilleszkedik a sorba, kitölti a rendelkezésére álló helyet, teli mezőt nem ír felül

4.5 Speciális képesség használata mana nélkül

Teszt lépés: A felhasználó megnyomja a speciális képesség használatához rendelt gombot, miközben az általa birtokolt mana száma 0

Elvárt eredmény: Nem történik semmi változás sem a felhasználó, sem az ellenfel pályarészén, a tetromino tovább esik lefelé változatlan sebességgel.

4.6 Speciális képesség használata 1 mana birtoklásával Ninja karakter esetén

Teszt lépés: A felhasználó megnyomja a speciális képesség használatához rendelt gombot, miközben az általa birtokolt mana száma 1 az általa kiválasztott karakter pedig a Ninja karakter

Elvárt eredmény: A felhasználó játékterén minden alakzat jobb oldalra tolódik kitöltve a lyukakat a játéktéren, miközben minden sorban üres terület csak a sor bal oldalán található egybefüggő, alakzattal nem megszakított sávban. A felhasználható mana mennyisége 0-ra csökken.

4.7 Speciális képesség használata 2 mana birtoklásával Ninja karakter esetén

Teszt lépés: A felhasználó megnyomja a speciális képesség használatához rendelt gombot, miközben az általa birtokolt mana száma 2 az általa kiválasztott karakter pedig a Ninja karakter

Elvárt eredmény: A felhasználó játékterén eltűnik a legalsó két sor, és két új sor tűnik fel az ellenfél térfelén egy-egy véletlenszerűen elhelyezett lyukkal. A felhasználható mana mennyisége 0-ra csökken.

4.8 Speciális képesség használata 3 vagy több mana birtoklásával Ninja karakter esetén

Teszt lépés: A felhasználó megnyomja a speciális képesség használatához rendelt gombot, miközben az általa birtokolt mana száma 3 vagy több az általa kiválasztott karakter pedig a Ninja karakter

Elvárt eredmény: A felhasználó játéktere az ellenfél játékterének a képesség használatakor életben lévő játéktér pontos mása lesz. A felhasználható mana mennyisége 0-ra csökken.

4.9 Speciális képesség használata 1 mana birtoklásával Warrior karakter esetén

Teszt lépés: A felhasználó megnyomja a speciális képesség használatához rendelt gombot, miközben az általa birtokolt mana száma 1 az általa kiválasztott karakter pedig a Warrior karakter

Elvárt eredmény: A felhasználó játékterén eltűnik a felső 4 sor. Amennyiben nincs annyi sor a játéktéren a játéktér üres lesz. A felhasználható mana mennyisége 0-ra csökken.

4.10 Speciális képesség használata 2 mana birtoklásával Warrior karakter esetén

Teszt lépés: A felhasználó megnyomja a speciális képesség használatához rendelt gombot, miközben az általa birtokolt mana száma 2 az általa kiválasztott karakter pedig a Warrior karakter

Elvárt eredmény: A felhasználó játéktere változatlan marad, míg az ellenfél játékterén amint leérkezik az aktuális teromino 5 új sor jelenik meg minden sorban egy véletlenszerűen elhelyezett lyukkal. A felhasználható mana mennyisége 0-ra csökken.

4.11 Speciális képesség használata 3 vagy több mana birtoklásával Warrior karakter esetén

Teszt lépés: A felhasználó megnyomja a speciális képesség használatához rendelt gombot, miközben az általa birtokolt mana száma 3 vagy több az általa kiválasztott karakter pedig a Warrior karakter

Elvárt eredmény: A ellenfél játéktere és a felhasználó játéktere helyet cserélnek. A felhasználható mana mennyisége 0-ra csökken.

4.12 Speciális képesség használata 1 mana birtoklásával Mage karakter esetén

Teszt lépés: A felhasználó megnyomja a speciális képesség használatához rendelt gombot, miközben az általa birtokolt mana száma 1 az általa kiválasztott karakter pedig a Mage karakter

Elvárt eredmény: A felhasználó játékterén a legalsó négy sor eltűnik. Amennyiben nincs annyi sor a játéktéren a játéktér üres lesz. A felhasználható mana mennyisége 0-ra csökken.

4.13 Speciális képesség használata 2 mana birtoklásával Mage karakter esetén

Teszt lépés: A felhasználó megnyomja a speciális képesség használatához rendelt gombot, miközben az általa birtokolt mana száma 2 az általa kiválasztott karakter pedig a Mage karakter

Elvárt eredmény: A felhasználó játékterén eső aktuális tetromino eltűnik, és helyette véletlenszerűen egy új generálódik. A felhasználható mana mennyisége 0-ra csökken.

4.14 Speciális képesség használata 3 vagy több mana birtoklásával Mage karakter esetén

Teszt lépés: A felhasználó megnyomja a speciális képesség használatához rendelt gombot, miközben az általa birtokolt mana száma 3 vagy több az általa kiválasztott karakter pedig a Mage karakter

Elvárt eredmény: A felhasználó játéktere változatlan marad. Az ellenfél játékterén a legfelső olyan sorral bezárólag ahol legalább egy nem üres mező található az összes mező ami üres volt teli lesz, valamint az összes olyan mező ami teli volt üres lesz. A felhasználható mana mennyisége 0-ra csökken.

4.15 Sor hozzáadódása a játéktérhez

Teszt lépés: A felhasználó eltűntetett 1 vagy több sort a játékteréről, vagy olyan speciális képességet használt ami hozzátesz az ellenfél játékteréhez plusz sorokat.

Elvárt eredmény: az ellenfél játékterének alján, amint letette a sor hozzáadása alatt nála lévő tetromino-t, megjelenik az összes addig összegyűlt hozzáadott sor , mindegyik sorban egyetlen véletlneszerűen elhelyezett lyukkal, mielőtt még az ellenfél megkapná az új alakzatot.

4.16 Kilépés a játékból

Teszt lépés: Felhasználó megnyomja az ESC billentyűt, vagy az ablak sarkában található bezárógombot az egérrel.

Elvárt eredmény: A játékablak bezáródik

4.17 MI játék indítása

Teszt lépés: Felhasználó elindítja a játékot MI mód megadásával.

Elvárt eredmény: A játék elindul, az ellenfél aktívan mozgatja és helyezi le a tetromino-kat.

4.18 MI speciális képesség használata

Teszt lépés: Felhasználó MI módban indítva a játékot megvárja amíg az ellenfél játékterén csak a felső 4 sor nem üres

Elvárt eredmény: Amint a felső 4 sor közül nem legalább egy nem üres az ellenfél felhasználja addig meglévő manáit, és végrehajtódik a karakternek és rendelkezésre álló manának megfelelő speciális képesség. Az ellenfél manáinak száma 0 lesz.

4.19 Tetromino forgatása

Teszt lépés: Felhasználó megnyomja a tetromino forgatáshoz kiosztott billentyűt, miközben a forgást nem akadályozza semmi.

Elvárt eredmény: a Tetromino 45 fokkal jobbra fordul

4.20 Tetromino forgatás akadályozás esetén

Teszt lépés: Felhasználó megnyomja a tetromino forgatásához kiosztott billentyűt miközben a forgást fal, vagy más alakzat akadályozza

Elvárt eredmény: a Tetromino nem mozdul, az esés sebessége nem változik

4.21 Lokális játék indítása

Teszt lépés: Felhasználó elindítja a játékot lokális módban

Elvárt eredmény: A játék elindul, a felhasználó mindkét játékteret irányíthatja, egyiket a nyilakkal, másikat a W A S és D billentyűk nyomásával.

4.22 Játék elvesztése

Teszt lépés: Felhasználó addig rakja le a kapott tetromino-kat, amíg már nem tud többet lerakni

Elvárt eredmény: Mindkét játékos pályarésze üres lesz, mindkét játékos mana-ja 0-ra csökken és az ellenfél győzelmeinek száma egyel nő.

4.23 Hálózati játék indítása rossz ip-port párossal

Teszt lépés: Felhasználó a játékot hálózati módban indítja, de rossz portszámot ad meg

Elvárt eredmény: A program hibaüzenettel tér vissza, ami azt mondja, hogy a megadott címhez nem lehet csatlakozni

4.24 Hálózati játék indítása ellenfél nélkül

Teszt lépés: Felhasználó elindítja a játékszervert, valamint a játék egy példányát a megfelelő ip-port páros megadásával

Elvárt eredmény: A játék rövid idő eltelte után visszatér hibaüzenettel, hogy nem sikerült ellenfelet találni.

4.25 Hálózati játék indítása

Teszt lépés: Felhasználó elindítja a játékszervert, valamint a játék két példányát a megfelelő ip-port páros megadásával

Elvárt eredmény: Miután a második példány is elindult a programból, a játék elindul, és a két példányban megkezdődik a játék, ahol a játékosok egymás játékterét megfelelő szinkronban tartva látják hasonlóan a lokális játékhoz.

4.26 Ellenfél kilépése hálózati játék közben

Teszt lépés: Felhasználó elindítja a játékszervert, valamint a játék két példányát a megfelelő ip-port páros megadásával. Eztuán a felhasználó az egyik példányt kilépteti.

Elvárt eredmény: A másik példány is leáll az ellenfél lecsatlakozott hibaüzenettel

5. Felhasználói dokumentáció

5.1 A futtatáshoz ajánlott hardver-, szoftver konfiguráció

A futtatáshoz ajánlott konfiguráció:

Linux

OS: Debian 9

CPU: Intel Core 2 Duo E7300

Memória: 2 GB

Mac

OS: OS X Sierra

CPU: Intel Core i3

Memória: 2 GB

5.2 Telepítés

A következő csomagok megléte szükséges a játék futtatásához, amennyiben ezek nincsenek telepítve, az adott operációs rendszer csomagkezelőjével kerüljenek telepítésre:

- libsdl2
- libsdl2-image
- libsdl2-ttf

A program egy tarball fájlba van csomagolva, ami tartalmazza az összes szükséges fájlt a játék indításához. A telepítéshez szükséges a tarball fájl kicsomagolása egy szabadon választott helyre a használt adattárolón amelyre telepíteni szeretnénk. Ezután a játék indításra készen áll.

A játékszerver is megtalálható a csomagban, a futtatásához szükséges a python 2-es verziójának megléte.

5.3 A program használata

Tetris

A program terminálból indítható a megfelelő paraméterek beállításával. A használható paraméterek a következők:

-h, --help : A játék help-jének kiíratása a képernyőre, ahol a paraméterek magyarázatokkal kiíródnak

---version : A játék verziójának kiíratása

-1 <name>, --character-1 <name> : Szükséges paraméter. Az első játékos karakterének kiválasztása. A **<name>** helyére az alábbi paraméterek kerülhetnek:

- n : Ninja karakter
- w: Warrior karakter
- m: Mage karakter

-2 <name>, --character-2 <name> : Opcionális paraméter, hálózati játékhoz nem szükséges. A második játékos karakterének kiválasztása. A **<name>** helyére az első játékosnál használatos paraméterek közül lehet választani.

Az alábbi paraméterek közül csak egy adható meg egyszerre:

-a, --ai-game : MI elleni játék indítása

-l, --local-game : Lokális két személyes játék indítása

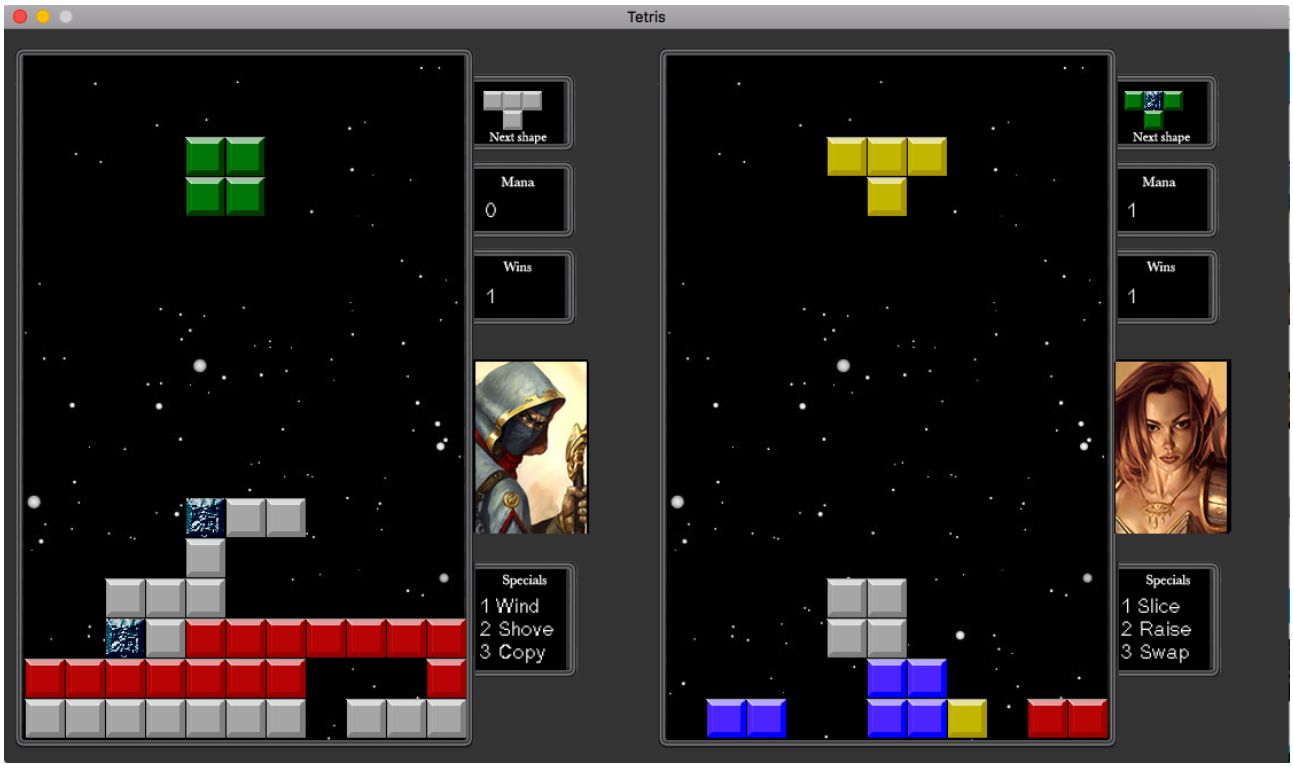
-n <address>, --enemy-addr <address> : Hálózati játék indítása. Az <address> helyére egy ip / port párosnak kell kerülnie ami a játékszerver címét és portját tartalmazza amivel csatlakozik a játék, a következő alakban: **<ip>:<port>**

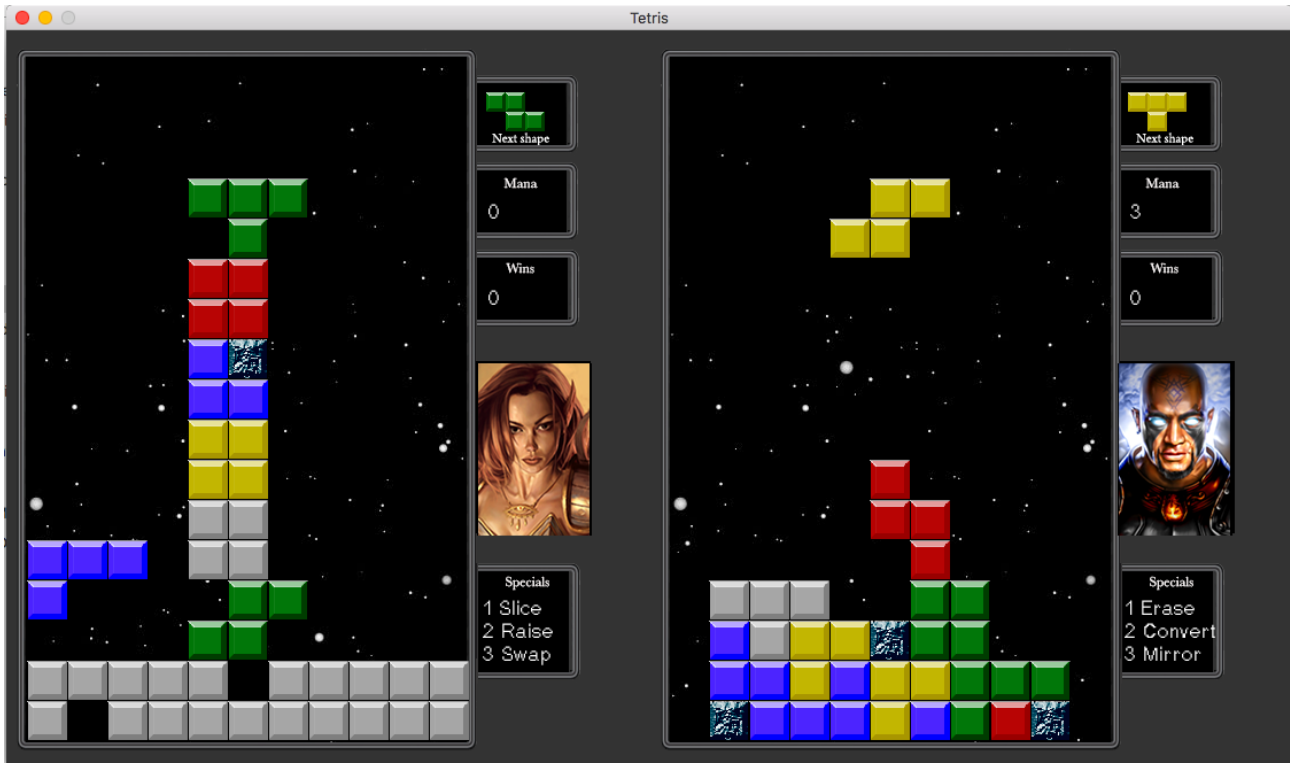
Példák paraméterezésre:

out/Tetris -n localhost:4242 -1 n

out/Tetris -l -1 n -2 w

out/Tetris -n 127.0.0.1:4242 -1 m


Tetris
Next shape
Mana
0
Wins
1
Specials
1 Wind
2 Shove
3 Copy
Next shape
Mana
1
Wins
1
Specials
1 Slice
2 Raise
3 Swap


Tetris
Next shape
Mana
0
Wins
0
Specials
1 Slice
2 Raise
3 Swap
Next shape
Mana
3
Wins
0
Specials
1 Erase
2 Convert
3 Mirror

Gameserver

A játékszerver indításához szükséges a python2 megléte. Egyetlen paramétert vesz át, a portot amin keresztül hallgatózik bejövő kapcsolatok után. Kétféleképp indítható:

python2 src/gameserver.py <port>

valamint

./src/gameserver.py <port>

A játéknak az itt megadott porthoz kell csatlakozni hálózati játék indításakor, valamint a gameservert futtató számítógép ip címéhez.

Felhasznált irodalom jegyzéke

- Robert Nystrom – Game Programming Patterns [1]
- Robert Cecil Martin – Clean Code [2]