Programozás alapjai 3: Házi feladat specifikáció

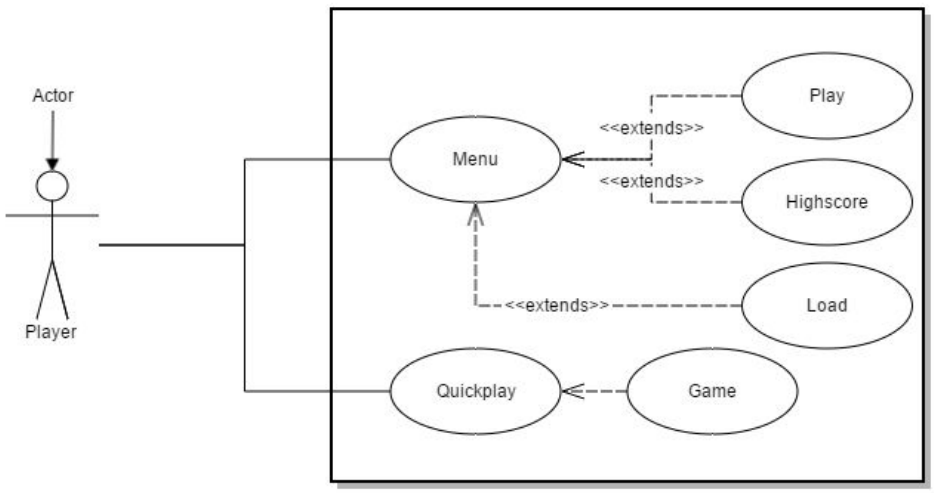
Név: **Hegedüs Dániel**
Neptun: **YBY8BK**

Bubble Game

A házi feladatnak egy egyszerű ügyességi játék megírását választottam, mely a már ismert telefonos játék doodle jump mintájára készül.

A játék lényege minél tovább játékban maradni. Ezt a játékos úgy érheti el, hogy nem hagyja a karakterét lefutni az adott játéktérről. A játékos a játéktéren mozgó platformokat használhatja, hogy minél tovább benn maradjon a játékban, ezen platformok között való "ugrálással". A játék teljesen az ugrálós platformerek zsánerét követi, annyi különbséggel, hogy itt a karakterünk egy kisebb jól megkülönböztetett karika, és a platformok is karikák.

A játék során a buborékok(körök) fentről lefelé esnek lefelé, majd az ablak alján eltűnnek. A játékosnak a karikájával a körök felületén kell "gördülnie" majd egyik körről a másikra ugrálva haladnia felfelé. A kör felületén a mozgásának irányát az egér a képernyőn jobbra illetve balra mozgatásával teheti meg, ugrani kattintással tud.



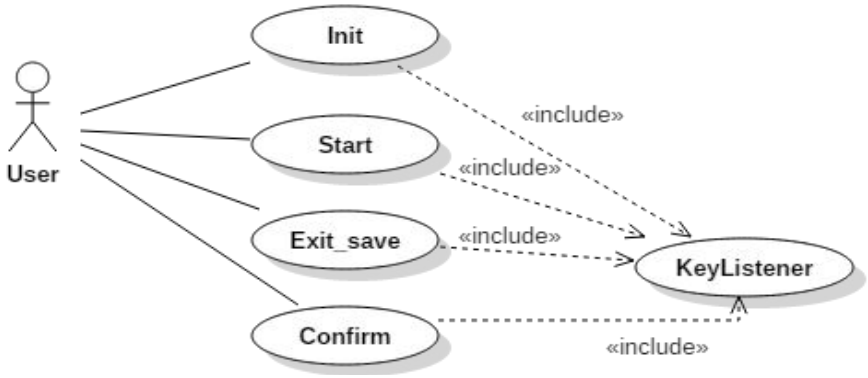
A játékban lesz "high score" lista mely minden játék végén eltárolja a játékos pontszámát. Majd a játék befejeztével menti ezt. A lista előhívható. Ha a játékos félbe kívánja hagyni a játékát, mentheti az adott játékmenetet, majd a következő alkalommal be is töltheti a korábbi mentését.

A játék vezérlése a menu-ből történik. A játékos a játék közben bármikor be tud lépni a menu-be.

Use-Case

A menü felépítése:

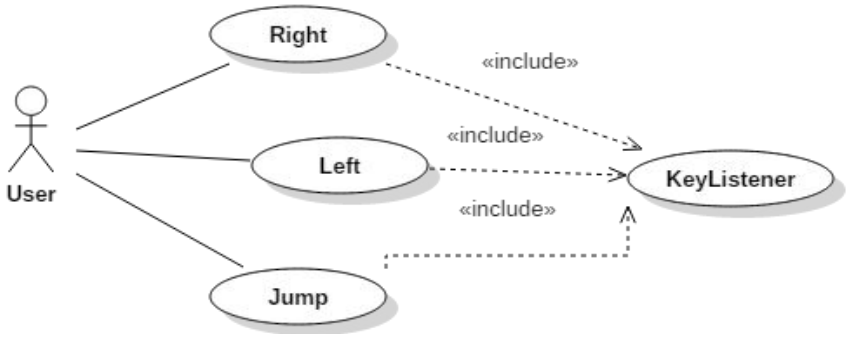
- Start
- Load(nem elérhető)
- Exit/Save
- Confirm



A felhasználó akkor indíthatja el a játékot ha már megadott egy nevet, ezt a nevet a "Confirm" gombbal fogadtathatja el. Confirm után a játékos elindíthatja a játékot.

A játék elindítása után:

- Right
- Left
- Jump



A felhasználó a karikájának a forgási irányát tudja változtatni a billentyűzet nyilainak alkalmazásával. Értelem szerűen csak jobbra és balra. Ezen kívül a játékos elrúgadzkodhat a karikájával az adott platformtól a space billentyű leütésével.

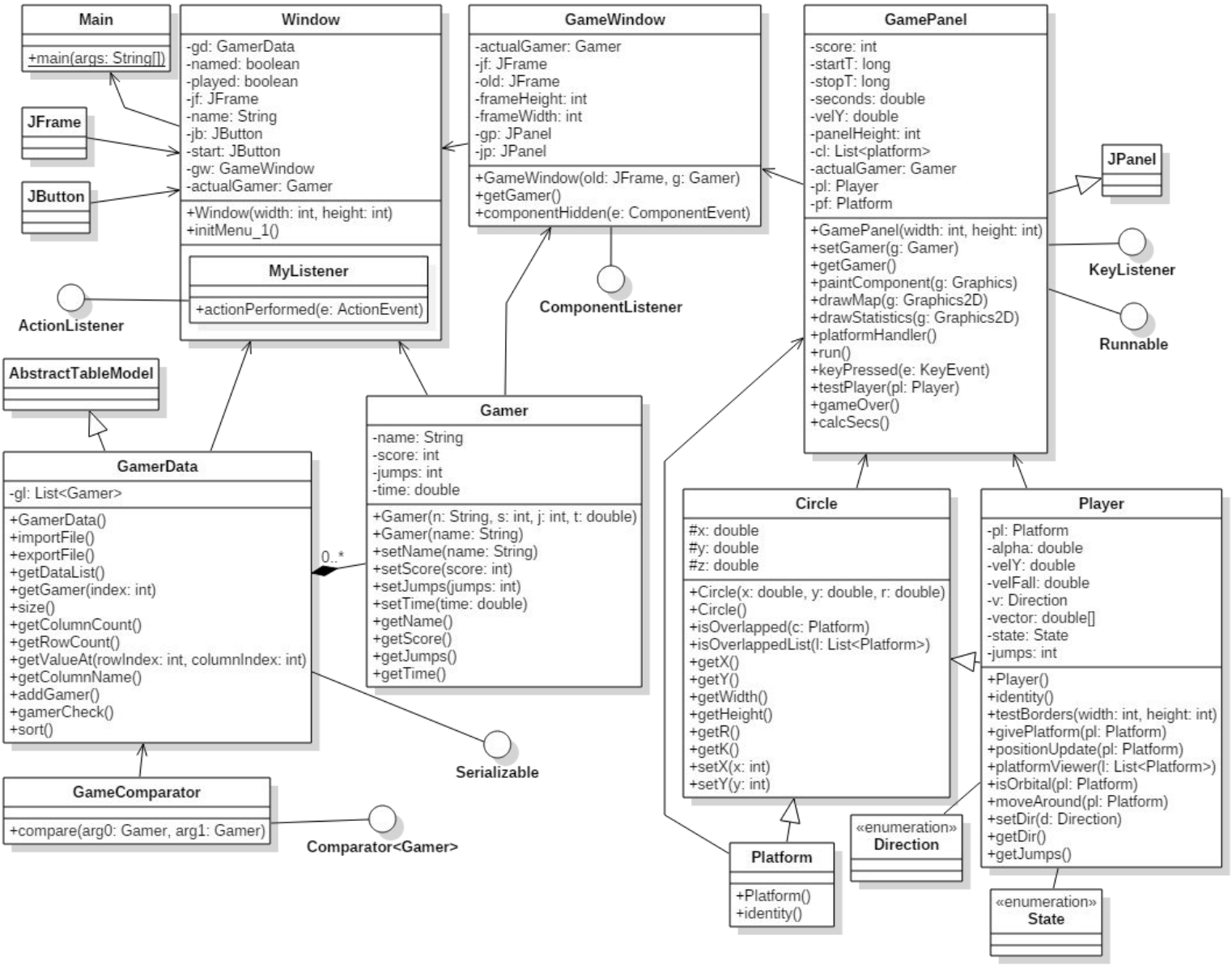
Mindkét esetben a leütött billentyűk monitorozásának feladatát a "java.awt.event.KeyListener" látja el.

A menü ezen kívül megjelenít egy táblázatot. A táblázat egy ranglista első 3 elemét jelenítni meg melyet a korábban már lejátszott játékosok eredményeiből állít össze. Ezt a listát menti a program.

A játék indítása után a játékos csak a játék végeztével tud visszatérni a menübe.

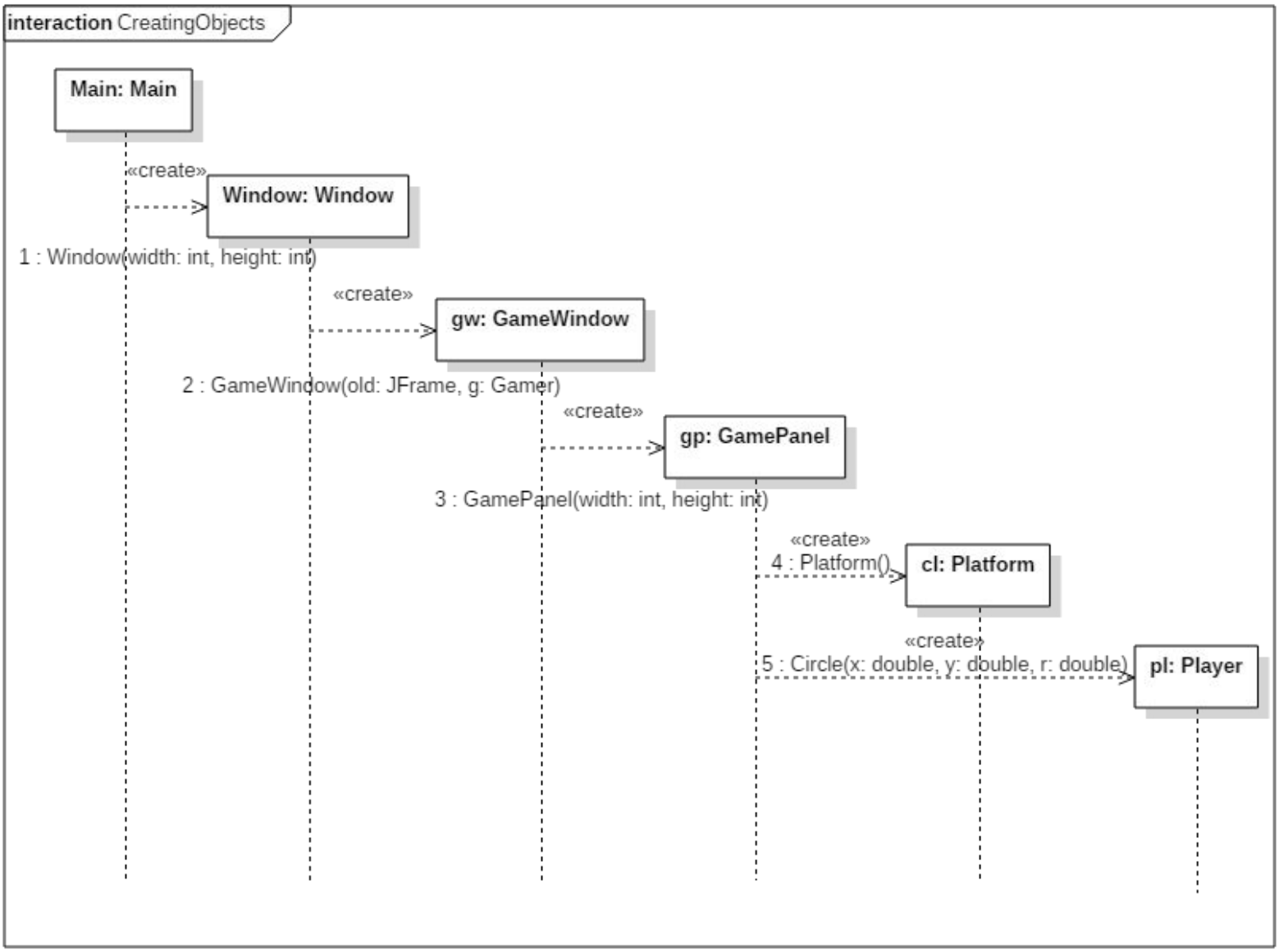
Osztálydiagram

A következő osztálydiagramon próbálom a programom osztályszerkezetét és a köztük lévő asszociációs kapcsolatokat modellezni.



Use-Case szekvenciadiagramok

Construktor hívások sorrendje:



A menu kezelésésének folyamata:

A játék belső mechanikájának folyamata:

Fontosabb osztályok és metódusok bemutatása

Window

A menüt tartalmazó ablak.

public void init_Menu_1()

A menü megjelenítéséhez szükséges osztályok beállítása.

public void actionPerformed(e: ActionEvent)

A billentyűk lenyomásának monitorozásáért felelős osztály metódusa.

GamePanel

A játék terének szolgáló panel. Itt történik a grafikus megjelenítés.

public void paintConponent(g: Graphics)

Az elemek kirajzolásáért felelős függvény. A függvényen belül a kód jobb átláthatósága kedvéért 2 segédfüggvényt használok hozzá.

public void run()

A folyamatos futásáért felelő függvény meg 5 ms-enként meghívódik és meghívja a végén a repaint() függvényt.

public keyPressed(e: KeyEvent)

A gombok lenyomását monitorozó függvény, mely a lenyomott billentyű alapján eldönti a játékos mozgásirányát, amit lehet: RIGHT, LEFT, JUMP.

public testPlayer(pl: Player)

A játékos mozgását figyeli, hogy letér-e a pályáról, ha ez megtörténik meghívja a gameOver() függvényt, ami fokozatosan lebontja az aktualáis ablakot és visszavezet a menübe.

Circle

A platformok és playerek ősosztálya. A kör dimenzióit tárolja és kezeli. Hagyományom matematikai modell szerint tárolja az X, Y koordinátákat de képes azokat visszalakítani a Graphics2D osztály kedve szerint.

public boolean isOverlapped(c: Platform)

Ellenőrzi, hogy két platform ütközik-e egymással. Elsősorban a platformot létrehozásánál használom, hogy a platformok ne legyenek egymásban.

Player

A játékos karikájának az osztálya.

public void testBorders(width: int, height: int)

A játékos helyzetét vizsgálja és ha elhagyja a pályát véget vet a játéknak a State változó beállításával.

public void positionUpdate(pl: Platform)

A játékos pizíciójának frissítése a Platform segítségével ami körül éppen mozog. Ugrás közben nem hívódik meg.

public Platform platformViewer(l: List<Platform>)

Az éppen a játéktéren lévő platformokat figyeli, és ha közel ér egyhez akkor beállítja a saját *pl* változójának értékét az aktuális platformnak megfelelően. Ezen kívül kiszámolja, hogy a platform melyik pontjára érkezett és az alapján álltja be a player *alpha* változójának értékét.

public boolean isOrbital(pl: Platform)

A paraméterként kapott platformtól vett távolságot vizsgálja, ha a játékos és az adott platform sugarának összege egyenlő a kettejük távolságával akkor *true* értékkel tér vissza.

public void moveAround(pl: Platform)

Az aktuális platform körüli mozgást végzi el, illetve a körtől való érintővel merőleges irányú elugrást.

Gamer

A játékosok nevét, pontját, idejét, ugrásainak számát tároló osztály.

GamerData

A ranglistát tároló lista. Egy *Gamer* objektumokat tartalmazó listát tartalmaz.

public void importFile()

A serializált *GamerData* beolvasása a “highscore.dat” fileból.

public void exportFile()

A serializált *GamerData* kiírása a “highscore.dat” fileba.

public void sort()

Egy “Collection.sort()”-t függvényt meghívó feüggvény ami a *GameComparator* osztály segítségével sorba rendezi az adattagokat.