TORPEDÓ JÁTÉK Fejlesztői Dokumentáció

Beágyazott Rendszerek Szoftvertechnológiája - 2022 Házi Feladat Fejlesztői Dokumentáció

Simon László

Szalay Szabolcs

Szunyoghy Dóra

# Projekt felépítése

A projektben két jól elkülöníthető modult definiáltunk. Az egyik modul a grafikáért, a játék ablakaiért és a dinamikus elemek megjelenítéséért felelős (View). A másik modul pedig a játék irányításáért felelős. Ez tartalmazza a hálózati kommunikációt, a játékmenetet vezérlő állapotgépet, és az alkalmazás futtatásáért felelős állományokat.

# View

A megjelenítéseket a Java swing és awt könyvtárak segítségével valósítjuk meg. Az alkalmazásban a következő ablakokat definiáltuk:

* WelcomeFrame
* CreateFrame
* JoinFrame
* GameFrame
* EndFrame

A program futása során egy-egy ilyen képernyő van megjelenítve, és ezeken folyik a játék menete.

## WelcomeFrame

Indításkor a WelcomeFrame osztály kerül példányosításra, ami megjeleníti a játék kezdőképernyőjét. Az osztály a következő metódusokat tartalmazza:

* WelcomeFrame(Controller app): konstruktor, ami meghívja az ősosztály (JFrame) konstruktorát, valamint a setupWelcomeFrame() függvényt.
* setupWelcomeFrame(): A függvény ablak paramétereit állítja be, elhelyez a képernyőn egy JLabel üdvözlőszöveget, egy createButton gombot új játék létrehozásához, valamint egy joinButton gombot játékhoz való csatlakozáshoz. Ezek a komponensek a JButton swing osztályból vannak származtatva és MouseInputListener interfészt valósítanak meg. A funkciójuk a kattintás hatására történő megadott függvény hívása, valamint a gomb kinézetének beállítása.
* createGame(): a CreateButton gomb megnyomásakor hívódik meg, átnavigál a CreateFrame képernyőre.
* joinGame(): a JoinButton gomb megnyomásakor hívódik meg, átnavigál a JoinFrame képernyőre.
* hideAllFrames(): elrejti a WelcomeFrame-et és a JoinFrame-et vagy CreateFrame-t, attól függően, hogy melyik volt példányosítva. Ezt a függvényt a Controller hívja a játék indulásakor.

## CreateFrame

Amennyiben új játék létrehozása történik meg, ez a képernyő a következő a kezdőképernyő után. Ennek funkciója az, hogy a képernyőn lévő gomb megnyomásának hatására létrejöjjön egy új Game példány, létrejöjjön egy új Server socket, és a képernyőn megjelenjen a ennek a szervernek az IP-címe, hogy egy kliens csatlakozni tudjon.

Az osztály a következő metódusokat tartalmazza:

* CreateFrame(Controller app): konstruktor, az ősosztály (JFrame) konstruktorát hívja meg, valamint a setupCreateFrame() függvényt.
* setupCreateFrame(): Az ablak megjelenítésének paramétereit beállító függvény. Egy IPSign mezőt hoz létre, ami a szerver socket IP-címének megjelenítésére szolgál, valamint egy CreateGameButton gombot, amit ha megnyom a felhasználó, akkor létrejön az új Game példány és megjelenik a szerver IP-címe az IPSign címkén.
* createGame(): a függvény az ablakban lévő gomb megnyomására hívódik. Hatására a Communication osztály segítségével lekérdezésre kerül a számítógép IP-címe, és ez megjelenik az IPSign mezőben, valamint meghívódik a Controller create() függvénye, ami elindítja az új játék létrehozásának folyamatát.

## JoinFrame

Amennyiben a felhasználó egy létező játékhoz szeretne csatlakozni, ez a képernyő a következő a kezdőképernyő után. Ennek funkciója az, hogy a program a képernyőn lévő bementi mezőben megadott IP-címmel rendelkező szerver socket-re csatlakozzon, mint kliens, valamint, hogy létrehozzon egy lokális Game példányt. Az osztály a következő metódusokat tartalmazza:

* JoinFrame(Controller app): konstruktor, az ősosztály (JFrame) konstruktorát hívja meg, valamint a setupJoinFrame() függvényt.
* setupJoinFrame(): Az ablak megjelenítésének paramétereit beállító függvény. Egy JTextField típusú bemeneti mezőt hoz létre, ami a szerver socket IP-címének bevitelére, valamint egy JoinGameButton gombot, amit ha megnyom a felhasználó, akkor létrejön az új Game példány és az alkalmazás hálózaton keresztül csatlakozik a megadott szerver sockethoz kliensként.
* joinGame(): A JoinGameButton megnyomásának hatására hívódó függvény. Hatására a bemeneti mezőből a program beolvassa a megadott IP-címet, és meghívja a Controller osztály join(ip) függvényét, ami elindítja a játék létrehozását és csatlakozik a szerverhez.

## EndFrame

A játék befejezésekor megjelenő képernyő. A játék végén mindkét játékos számára megjeleníti, hogy mi a játék eredménye. Ez egy statikus képernyő, ami csak egy konstruktorral rendelkezik (JFrame osztályból származik). Példányosításkor megkapja, hogy a játékos, amelyiktől a hívás érkezett nyert-e, vagy sem.

## GameFrame

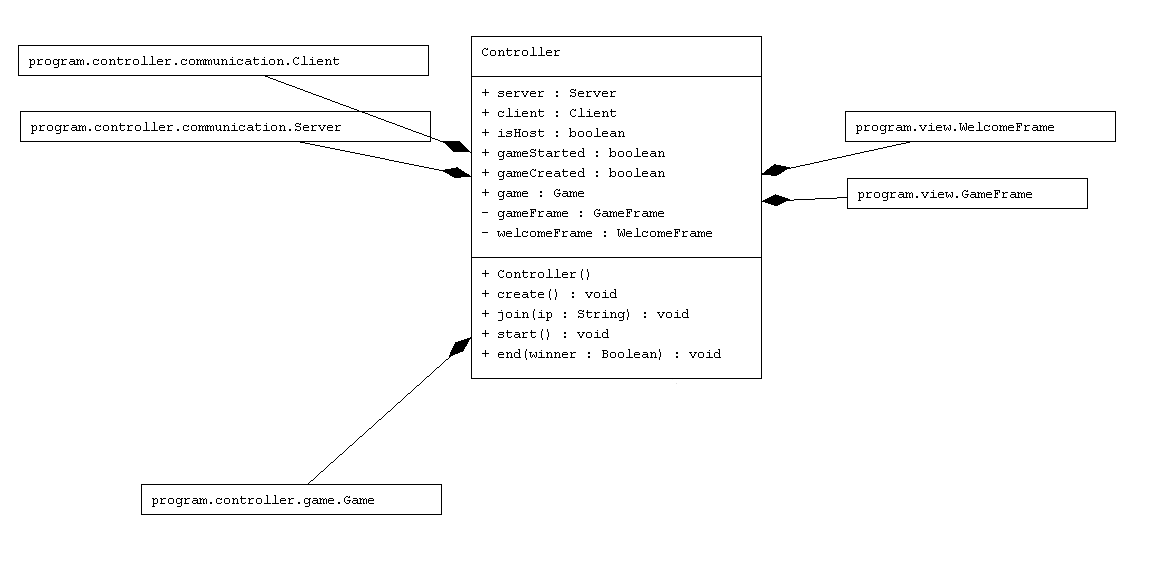
A GameFrame példány a játék főképrenyője. Ezen zajlik a torpedó játék. A képernyő tartalmaz egy GamePanel-t, ahol a játék aktív felülete található, valamint egy InfoPanel-t, ahol a játék során az információk közlése történik. Az osztály a következő metódusokat tartalmazza:

* GameFrame(Controller app, Game myGame): konstruktor, az ősosztály (JFrame) konstruktorát, valamint a setupGameFrame() függvényt hívja meg.
* setupGameFrame(): A képernyő megjelenítéséért felelős függvény. A keret paraméterezéséért felel, valamint létrehozza a képernyőn lévő panelokat, és ezeket a panelokat megtölti tartalommal.
* shootEnemy(int x, int y): Egy lövés hatására meghívja a Game példánynak a shootEnemy() függvényét, ami a lövést kezeli. Visszatérési értékként azt adja meg, hogy a lövés találat volt-e, vagy nem.
* deleteShip(int x, int y): A megkapott koordinátán elhelyezkedő hajóra meghívja a Game deleteShip függvényét, valamint felfedi a hajót és a közvetlen környezetét.
* ready2Play(): Az infopanel gombjainak megjelenítését kezeli az alapján, hogy a játékos ready állapotba kerülhet-e.
* placeShip(int len, int xStart, int yStart, int xEnd, int yEnd): Hajók lerakásakor hívódik a függvény, a Game példány placeShip() függvényét hívja meg. Visszatérési értéke, hogy a hajó elhelyezése sikeres volt-e.
* setHit (int x, int y): Az ellenfél mezőjén beállítja a megkapott mezőt “hit” állapotba.
* gotHit(int x, int y): Az játékos mezőjén beállítja a megkapott mezőt “hit” állapotba.
* isHit(int x, int y): Visszatérési értékként megadja, hogy a paraméterként megkapott mező “hit” állapotban van-e.
* endEnemyShip(int x, int y, int length, Orient or): Az ellenfél mezőjén felfedi az elpusztított hajót és annak környezetét.
* startFight(): Ez a függvény beállítja a játékmezőt kattintható (játszható) állpotba.
* set2setup(): Ez a függvény beállítja a játékmezőt a hajóelhelyezéshez szükséges állapotba.
* set2ready(): Ez a függvény beállítja a játékmezőt alapértelmezett, játék előtti státuszba.
* set2myTurn(): Ez a függvény az InfoPanel kezelését végzi, valamint az ellenfél és a saját mezőt a státuszhoz (myTurn) megfelelő állapotba hozza.
* A képen szöveg látható

  Automatikusan generált leírásset2enemyTurn (): Ez a függvény az InfoPanel kezelését végzi, valamint az ellenfél és a saját mezőt a státuszhoz (enemyTurn) megfelelő állapotba hozza.



# Controller

A Controller osztály felel a program megfelelő működéséért és ez az osztály kezeli az többi modult.

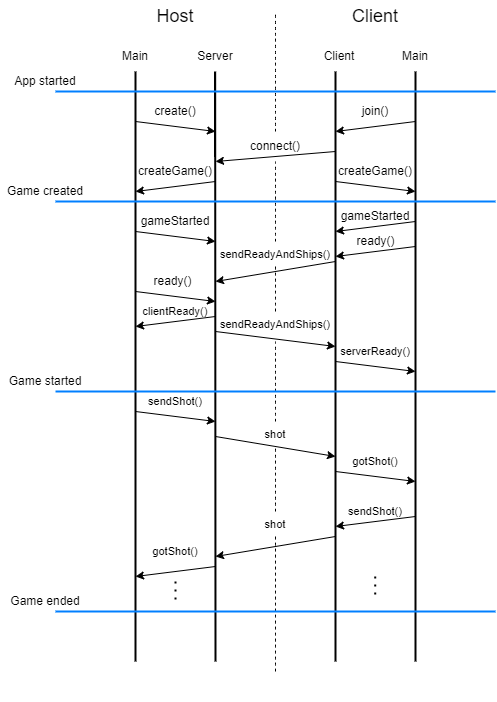
Az osztályban lévő metódusok a következők:

* Controller(): konstruktor, itt történik a fő szál kezelése. Az alkalmazás létrehozza a WelcomeFrame példányt, aminek feladata, hogy a felhasználó hatására elindítsa egy új játék létrehozását. A Game példány saját állapotgéppel rendelkezik, és a Controller osztály ezt az állapotgépet figyeli, és a program különböző státuszai között az állapotgép hatására vált.
* create(): Beállítja az alkalmazást Host-ként, valamint létrehozza a Server példányt, és létrehozza a Game példányt.
* join(): Beállítja az alkalmazást Client-nek, valamint létrehoz egy Client példányt, és létrehozza a Game példányt. A create() és join() függvények közül mindig csak az egyik hívódik meg.
* start(): Elrejti a WelcomeFrame példányt, megjeleníti a GameFrame példányt, és beállítja a gameCreated státuszváltozót true-ba.
* end(): Létrehoz egy EndFrame példányt.

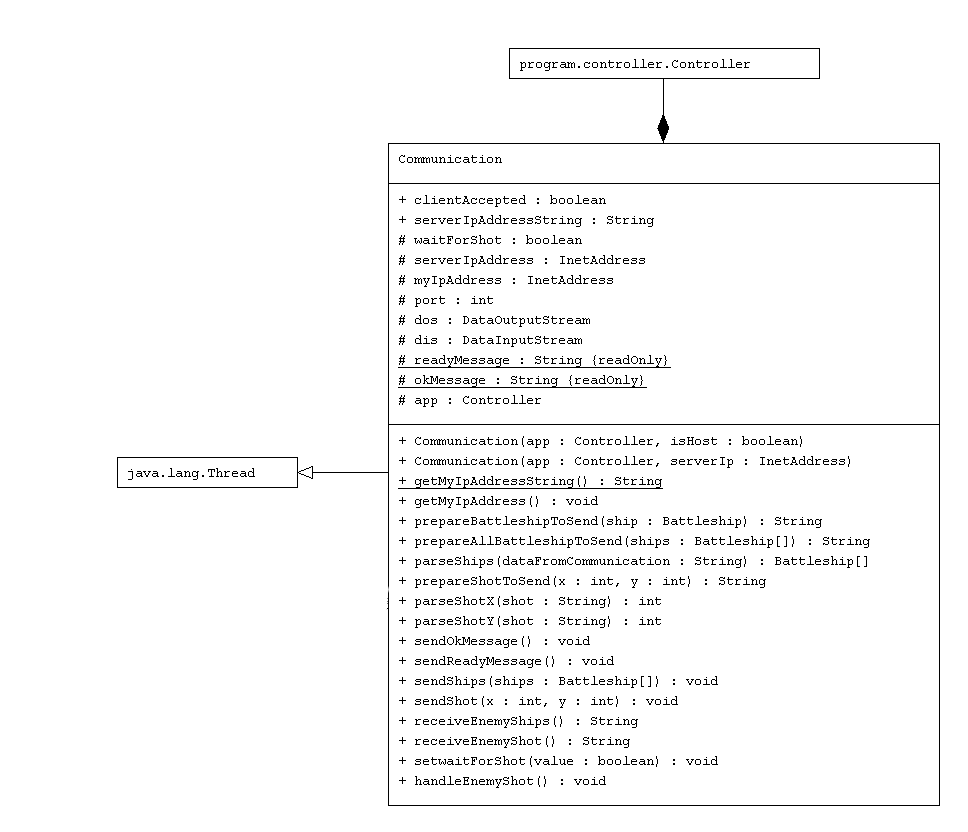
## Communication

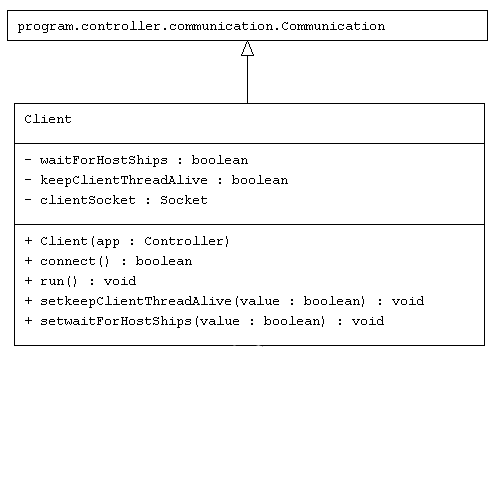
A Communication osztály a Thread osztályból származik, és ebből származnak a Server és Client osztályok, amikből az egyik a megfelelő alkalmazásban példányosításra kerül. A Communication osztályban olyan függvények kerültek implementálásra, amik nem Server vagy Client specifikusak.

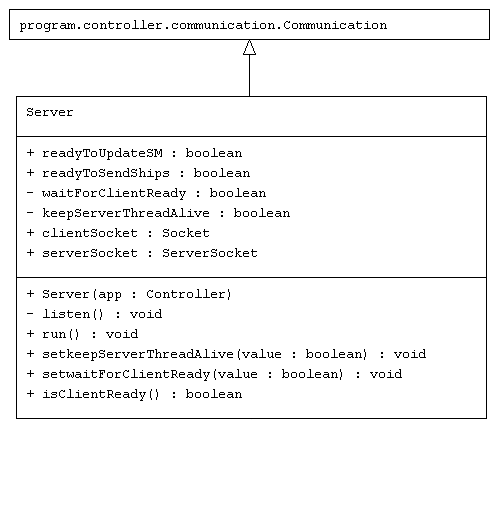
A kommunikáció során mindkét alkalmazás példány csaját Game példánnyal rendelkezik, valamint mindkét alkalmazásban 2 szál fut, egy a főprogram, és egy, ami a kommunikációért felelős. A kommunikáció aszinkronitást hoz a rendszerbe, vagyis a főszál és a kommunikációs szál időnként a másik szálra vár, hogy a futást folytathassa. A folyamatát a következő ábra mutatja be:



Kommunikációs folyamat

A kommunikációval kapcsolatos osztályok osztálydiagrammjai a következők:





Ezekben az osztályokban a kommunikációt kezelő, valamint a kommunikációkat előkészítő függvények vannak implementálva.

A Communication osztályban a következő függvények lettek megvalósítva:

* Communication(app: Controller, isHost: boolean): konstruktor.
* Communication(app: Controller, serverIp): Konstruktor, ami a kliens oldalon hívódik, extra funkció, hogy ez a példányba elmenti a szerver IP-címét.
* getMyIpAddress(): A példány IP-címének beállítására szolgáló függvény.
* getMyIpAddressString(): A példány INetAddress típusú IP címével tér vissza stringké konvertálva.
* prepareBattlesipToSend(ship: Battleship): Egy hajó példányt string formátumba konvertál aminek formája a következő: a hajót 4 adattal azonosítjuk, x,y pozíció, orientáció és hoszz. Ez függvény ezeket az adatokat kinyeri a paraméterként kapott hajó példányból és egy stringbe rakja, ahol vesszővel vannak egymástól elválasztva.
* prepareAllBattleshipToSend(ships: Battleship[]): Ez a függvény megkapja egy játékos hajóit, ezeken végigiterál és a prepareBattlesipToSend() függvény segítségével az összes hajót egy string-be konvertálja. A hajók adatai vesszőkkel, a hajók pontosvesszővel vannak elválasztva.
* parseShips(dataFromCommunication: ships): A függvény a hálózatról kapott hajó stringből csinál egy Battleship[] tömböt, amibe példányosítja a megkapott hajókat.
* parseShotX(shot: String): a hálózatról megkapott lövésből kinyeri a lövés X koordinátáját int-ként
* parseShotY(shot: String): a hálózatról megkapott lövésből kinyeri a lövés Y koordinátáját int-ként
* sendOkMessage(): a hálózaton egy „OK” stringet küld át a felépített socket kapcsolaton keresztül.
* sendReadyMessage(): a hálózaton egy „READY” stringet küld át a felépített socket kapcsolaton keresztül.
* sendShips(ships Battleship[]): Ez a függvény a hajók parse-olását és hálózati küldését végzi el.
* sendShot(x: int, y: int): A függvény segítségével a hálózaton egy X,Y lövés küldését lehet elvégezni.
* recieveEnemyShips(): Ez a függvény az, ami a socket kapcsolaton keresztül várja az átküldött hajókat, és fogadja őket, ha megérkeznek.
* recieveEnemyShot(): Ez a függvény az, ami a socket kapcsolaton keresztül várja az átküldött lövéseket, és kezeli őket, ha megérkeznek.
* setwaitForShot(value: boolean): Ez a függvény a példány waitForShot tulajdonságát állítja be a megkapott igaz/hamis értékre.
* handleEnemyShot(): Ha a szál lövésre vár, ez a függvény az ami ezt kezeli, meghívja a recieveEnemyShot() függvényt, és elküldi a megkapott lövést a Game számára.

A Client osztályban a következő függvények találhatók:

* Client(): konstruktor, ami az ősosztály konstruktorát hívja, valamint létrehozza a kliens socket-et.
* connect(): Hálózaton csatlakozik a szerver socket-re.
* run(): A Thread osztály függvényének túlterhelése. A kliens szál futása ebben a függvényben történik.
* setKeepClientThreadAlive(value: boolean): A példánynak a szál életbentartásáért felelős tulajdonságának írásáért felel.
* setWaitForHostShips(vaule: boolean): ezzel a függvénnyel be lehet állítani azt az állapotát a kliensnek, hogy a szervertől érkező hajók fogadására várjon.

A Server osztályban a következő függvények találhatók:

* Server(): konstruktor, ami az ősosztály konstruktorát hívja, valamint létrehozza a server socket-et.
* listen(): A socket kapcsolat kezdeményezésére szolgáló függvény, hatására a szerver kliens socket csatlakozására vár.
* run(): A Thread osztály függvényének túlterhelése. A szerver szál futása ebben a függvényben történik.
* setKeepServerThreadAlive(value: boolean): A példánynak a szál életbentartásáért felelős tulajdonságának írásáért felel.
* setWaitForClientReady(vaule: boolean): ezzel a függvénnyel be lehet állítani azt az állapotát a szervernek, hogy a klienstől érkező „READY” üzenet fogadására várjon.
* isClientReady(): a kliens „READY” üzenetének fogadásáért felelős, igaz értékkel tér vissza, ha a kliens készen áll a játékra és ez sikeresen át lett küldve a hálózaton.

## Game

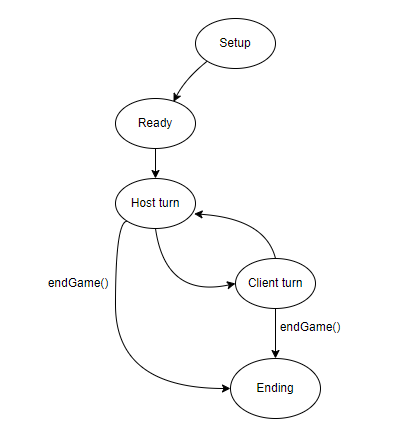
A Game modulhoz tartozik egy Game osztály, egy Battleship osztály, egy GameSM állapotgépet megvalósító enum, és egy Player osztály. A Game osztály feladata minden játékkal kapcsolatos feladatot kezelni. Ide tartoznak a játékosok állapotai, a játékban szereplő csatahajók adatai, valamint a GameFrame kezelése.

### Battleship osztály

Ez az osztály egy csatahajó adatait tartalmazza, hajó elhelyezkedését, paramétereit és státuszait a játék során. Az osztályban ezekhez a tulajdonságokhoz vannak getterek és setterek implementálva.

### GameSM

A játék állapotaiért felelős enum, ami a következő képen látható állapotgépet valósítja meg:

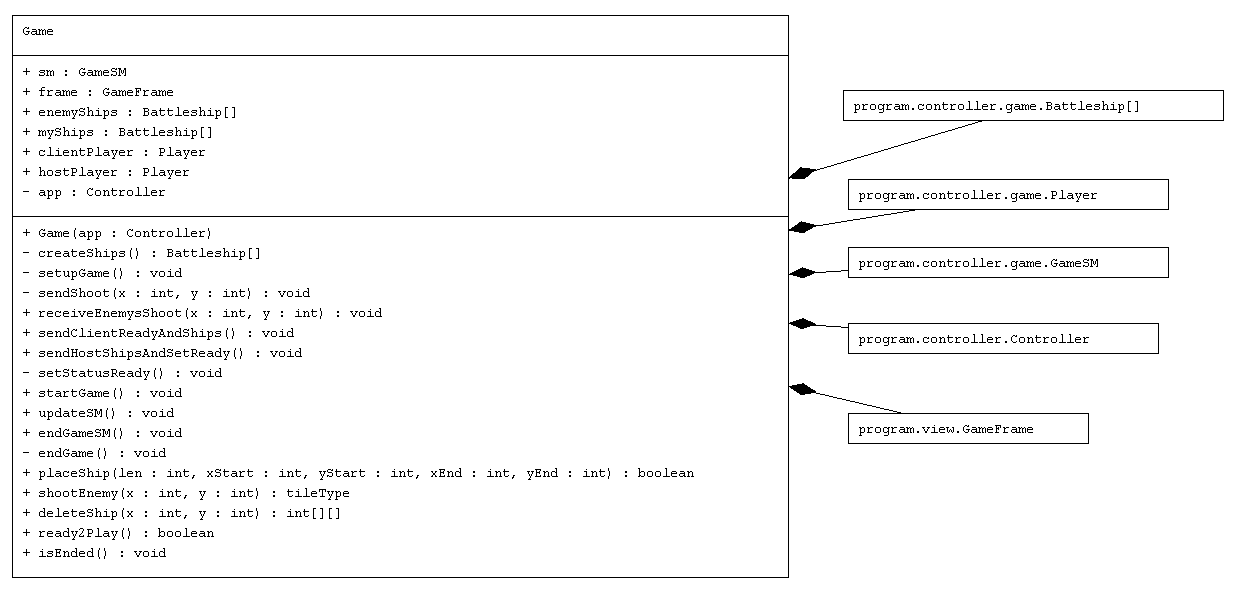


### Player osztály

A játékosok státuszait kezelő osztály, a többi osztály működését segíti. Itt van kezelve, hogy melyik játék az alkalmazásban a játszó játékos és melyik az ellenfél, valamint a játékosok Ready státuszainak kezelésére szolgál az osztály.

### Game osztály



A Game példány az, ami minden játékért felelős tevékenységet kezel, és minden játékkal kapcsolatos szolgáltatással kommunikál.

A megvalósított metódusok a következők:

* Game(app: Controller): konstruktor, a játék élőkészítését végző függvényt hívja meg.
* createShips() létrehoz egy hajók tömböt a megfelelő hajóméretekkel (1,1,2,2,3,4,5) és alapértelmezett adatokkal.
* setupGame(): az új játék inicializálásért felelős függvény. Létrehozza az állapotgépet, létrehozza a két szükséges Player példányt, létrehozza a két Battleship tömböt a játékosok hajóinak tárolásához, és létrehozza a GameFrame példányt, ahol a játék fog folyni.
* sendShoot(x: int, y: int): Ez a függvény meghívja a kommunikációs példány sendShot() függvényét, a játék állapotgépét a következő állapotba küldi, és frissíti a képernyőt.
* revieveEnemyShoot(x: int, y:int): Ha érkezett egy lövés, akkor ez a függvény az, ami kezeli a Game példányban. A frame mezőjének státuszát átállítja, valamint ellenőrzi, hogy történt-e hajónak elpusztítása.
* sendClientReadyAndShips(): Ezt a függvényt a setStatusReady() hívja. Ez a függvény kezeli a kliens oldalon a „READY” üzenet, valamint a hajók hálózati küldésének folyamatát, valamit frissíti a játék állapotgépét.
* sendHostShipsAndSetReady(): Ezt a függvényt a setStatusReady() hívja. Ez a függvény kezeli a szerver oldali „Ready” megnyomás folyamatát, ami összetettebb, és a Communication modulban kerül kezelésre.
* setStatusReady(): A „Ready” gomb megnyomása minden esetben ezt a függvényt hívja meg, és ez a függvény kezeli kommunikáció kezdeményezését és a Player-ek „Ready”-be állítását.
* startGame(): Ha megtörtént mindkét oldal „Ready” állapotban van, akkor a Controller ezzel a függvénnyel elindítja a játék aktív folyamatát, vagyis frissíti az állapotgépet, beállítja a szerver oldalon a frame-et olyan állapotba, hogy a felhasználó kezdeni tudjon, és beállítja a gameStarted flag-et igaz állapotba.
* updateSM(): A függvény az állapotgépet a következő állapotba küldi.
* endGame(): A függvény az állpotgépet „End” állapotba küldi, és ezzel befejezi a játékot.
* placeShip(. . .): Egy hajó lerakását kezelő függvény. Egy hajó lerakása után hívódik a GameFrame által. Ellenőrzések történnek a lerakással kapcsolatban, és ha nincsen semmilyen szabály megszegve, akkor a hajó lerakását és adatainak mentését kezeli.
* shootEnemy(x: int, y: int): Egy lövés kezdeményezésének hatására a Game sendShoot() függvényét hívja meg, valamint ellenőrzi, hogy történt-e találat vagy sem, és ennek függvényében frissíti a meglött mezőt.
* deleteShip(x: int, y: int): Ez a függvény az X,Y kezdőpontokkal rendelkező hajó törlését kezeli (a hajók elhelyezésének a fázisában).
* Ready2play(): Ez a függvény ellenőrzi a játékosok „Ready” feltételét, vagyis azt, hogy minden hajó el lett-e helyezve a pályán.
* isEnded(): Ez a függvény ellenőrzi, hogy egy lövés után a játék véget ért-e.