Java 2020 Nagy HF: JATacka

2020. március 24.

1. Projekt célja

A nagyházi célja a méltán híres zatacka játék egyszerűsített verziójának megvalósítása lesz. A játék lényegében egy túlélő – pontszerző játék, ahol az a játékos nyer, akinek a végén a legtöbb pontja van. Pontot szerezni a 1. táblázat alapján lehet.

2. Játékmenet

A pálya egy négyzetrácsból áll, amin a játékosok körönként 1 mezőt lépnek. A játék kezdetekor a játékosokon kívül nincs semmi a pályán. Minden körben a játékosok lépnek a 4 irány valamelyikébe, az elhagyott mezőn pedig 1 a saját színüknek megfelelő falat hagynak maguk után. A játékosok egyenesen haladnak alapértelmezetten, kivéve, ha bal, vagy jobb kanyar utasítást nem adnak az adott körben. Ha egy játékos falra lépne, kiesik a játékból. A pálya szélein át lehet jutni a pálya túloldalára.
A játékosok lépésük során néha (valamekkora valószínűséggel) nem hagynak falat maguk mögött, ekkor szakadás kerül az útjukba, amin bármely játékos áthaladhat, ezáltal plusz pontokat szerezve. Legyen egy opcionális játékmód, ahol bizonyos időközönként a pályán megjelennek bónuszok. Ezeket vagy rálépéssel, vagy kattintással lehet megszerezni, amelyik játékos először kattint az objektumra, az szerzi meg a pontot.
A játékot egyszerre maximum 6 játékos játszhatja hálózaton keresztül. A játékosok irányítására mindössze 2 gomb szükséges (jobra és balra kanyarodás 90 fokkal). Ha egy mezőre egyszerre lép két játékos, akkor mindkét játékos meghal.

A játékot vagy egy gépen több játékos játszhatja (ekkor nincsenek kattintással kezelendő objektumok), vagy több gépről hálózaton keresztül.

Esemény	Egy játékos kiesik	Egy játékos útvonalában levő szakadáson áthaladás	Kattintható bónuszok
Pontszám	Minden játékban levő játékosnak 5	1	1-5

1. táblázat. Pontszerzési lehetőségek.

3. User Interface

Kétféle User Interface-t szükséges elkészíteni. Egyet konzolról való játékhoz, egyet pedig grafikus felülettel.

Mindkét esetben a felhasználó először dönt, hogy hálózaton, vagy lokálisan szeretne játszani. Lokális játéknál megmondja, hány játékossal szeretne játszani, majd elindul a játék.

Hálózaton keresztül történő játéknál a felhasználó megad egy felhasználónevet és egy IP-t, amire csatlakozik. A szerver fogadja a kapcsolatokat és visszajelez a felhasználónak. Ha sikeres a kapcsolódás, akkor a felhasználó bekerül egy szobába, ahol várja a többi játékos kapcsolódását, megtudja, hogy hanyas játékos ő, illetve jelezheti, hogy készen áll (gombbal vagy szöveges üzenettel). Ha van legalább 2 játékos, és minden játékos készen áll, a szerver elindítja a játékot. A játék indulása után nem lehet csatlakozni a szerverre a játék végéig, ha valaki lekapcsolódik a szerverről, akkor az a játékos kiesik, mintha falnak ment volna. A játék végén minden játékosnak (aki még csatlakozva van a szerverre) a szerver elküldi az eredményeket, amit a kliensek valamilyen formában megjelenítenek a felhasználónak.

A UI nem tudja, hogy lokális vagy hálózati játék történik ezért minden potenciális eseményt továbbítani fog egy `MessageHandler`-nek.

3.1. Text UI

Szöveges UI esetén minden parancs/lépés a játékostól a bemenetre beírt szöveg lesz (karakterek stb). Mivel a végső verzióban a játék időközönként frissíteni fogja az állapotot (és nem nagyon lehet programból törölnia konzolt IDEA-ban), elég káoszos lesz a dolog. A pálya mellett mindig jelenjen meg a játékosok pontszáma is. Mivel konzolon nincs kattintás, így konzolos játékosok csak rálépéssel tudnak extra objektumokat megszerezni.

3.2. GUI

A GUI-s felhasználói felület esetén is látszódjon egymás mellett a pálya és a játékosok pontszáma.

4. Követelmények

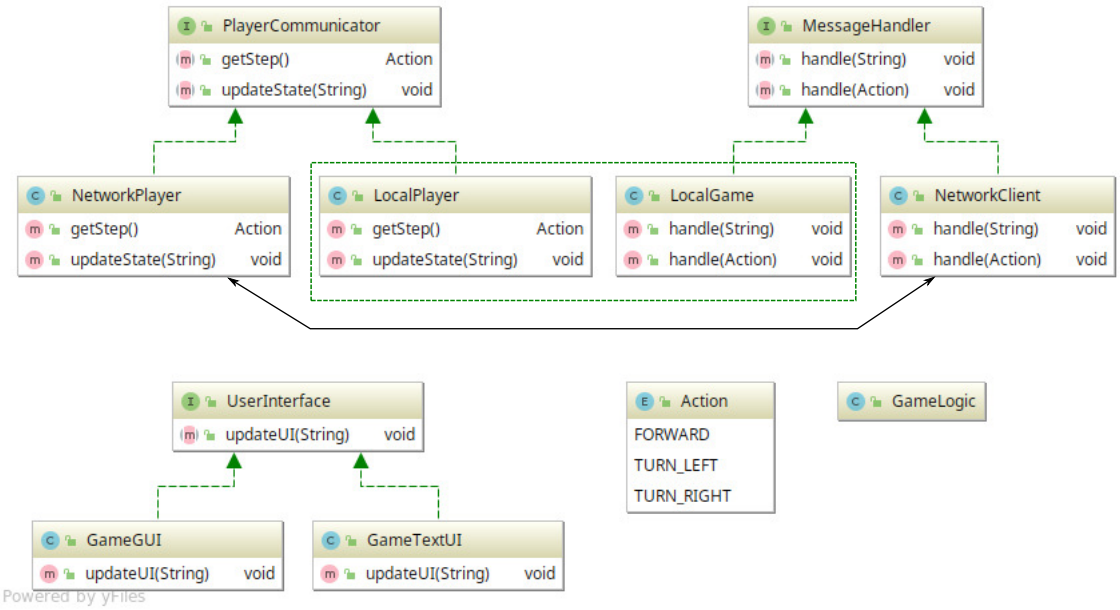
4.1. Projekt felépítése

Az ebben a fejezetben leírt követelmények azért vannak, hogy a különböző komponensek (játéklogika, megjelenítés és a kettő közötti "kommunikációs réteg") jól elkülöníthetőek legyenek. Adott esetben akár a kiadott interface-ek mentén cserélhetőek legyenek a komponensek.

Szükséges egy játéklogikával foglalkozó osztály. Ez hálózat esetén a szerveren értékeli ki a játékosok lépését és kezeli az esetleges halálokat. A lépések kezelésére szüksége van az egyes játékosok (adott körben megtett) lépésére. Ehhez egy `PlayerCommunicator` interface-t megvalósító osztály-t használ, ugyanezen az interface-n keresztül küldi el a játékosoknak a frissített állapotot körönként (illetve extra eseményenként). Emögé az interface mögé kell elrejteni a hálózati kommunikációval, vagy lokális játék esetén a user input feldolgozásával foglalkozó komponenseket.

A `UserInterface` játék közben lokális és hálózati játék esetén is ugyanúgy működik, az eseményeket egy `MassageHandler` objektumnak továbbítja, ami eldönti, hogy azt hogyan kell feldolgoznia és továbbítania a logika felé.

A "középen" levő kommunikációs réteg felelős, azért hogy a felhasználótól érkező üzeneteket továbbítsa a megfelelő `PlayerCommunicator`ok felé. Ezzel párhuzamosan a játéklogikától érkező



1. ábra. JATacka sematikus felépítés. Természetesen nem minden tagfüggvény látható az ábrán. A zöld szaggatott téglalap által bekeretezett két osztály összevonható kisebb ügyeskedéssel, de nem szükséges. A fekete nyíl jelzi a hálózato kommunikáció helyét. Az osztályok package-ekbe szervezése sem látszik az ábrán. Az `Action` enum és az interface-k megtalálhatóak dokumentálva a projekthez tartozó zip-ben.

frissítéseket továbbítja a UI-nak. Mivel itt sem lényeges melyik UI-t használjuk, ezért ez a réteg `UserInterface` interface statikus típussal vár egy UI példányt, akinek továbbítja az üzeneteket.

Az elképzelt felépítésről egy vázlatos UML-t láthattok az 1 ábrán.

4.2. GUI

Figyelj, hogy a JAT szálon ne legyen hálózatkezelés és a GUI-t csak a JAT szálról módosítsd! A játék vége egyértelműen látszódjon a játékosoknál.

Törekedj a szálbiztosságra!

4.3. Szerver-kliens kapcsolat

A játékosok és a szerver külön Java alkalmazások legyenek. A játékosok a szervernek küldik el a lépéseiket (irányváltoztatás, kattintások), a szerver ellenőrzi a lépéseket, végrehajtja azokat, és minden játékosnak küld üzenetet a történtekről. A kapott infók alapján a kliensek frissítik a GUI-t.

Törekedj a szálbiztosságra! A játék vége a hálózati kapcsolat tekintetében is értelmesen legyen kezelve. A játékosok időközbeni kilépése azok automatikus vesztésével jár, ez is legyen kezelve.

4.4. Egyéb követelmények

A kód legyen dokumentálva! Legyen könnyen érthető hogyan működik, hogyan lehet tesztelni. A dokumentáció és intuitív működés hiánya akár 0 pontot is vonhat maga után.

A program során törekedj a kivételek értelmes kezelésére! A kivételes helyzeteket valamilyen értelmes viselkedéssel próbáld helyreállítani, vagy kezelni.

5. CheckPoint határidő: 2020.04.23.

Javasoljuk, hogy a megtanultak felhasználásával egy olyan játékvariánst adj be, ahol a lokális játék működik. Fontos a kód újrafelhasználása miatt, hogy a játéklogikát és a megjelenítést jól külön lehessen választani (hálózatkezeléssel két külön Java program fog futni, amik üzenetekkel kommunikálnak). Ezért a játéklogika (ami állhat több osztálból is akár) a kommunikációs rétegből tényleg csak a PlayerCommunicator interace-t lássa és a Játékmenet UI-hoz kapcsolódó osztályokat egyáltalán ne.

Leadható csak konzolos verzió is, ami csak akkor lép, amikor a játékosok beírták, hogy mit szeretnének lépni. Ehhez szálkezelés se szükséges. De akár leadható olyan verzió is ahol a lokális játék működik és lehet választani konzol vagy GUI között (a két verzió futhat akár külön mainből, nem kell GUI és konzol között váltani tudni a program futása alatt).