



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai Kar

Média- és Oktatásinformatikai Tanszék

Teamfight Tactics in JAVA

Kozsik Tamás
egyetemi docens

Bezzegh Kristóf
programtervező informatikus szak

Budapest, 2021

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

SZAKDOLGOZAT TÉMABEJELENTŐ

Hallgató adatai:

Név: Bezzegh Kristóf

Neptun kód: T4O94U

Képzési adatok:

Szak: programtervező informatikus, alapképzés (BA/BSc)

Tagozat: Nappali

Belső témavezetővel rendelkezem

Témavezető neve: Kozsik Tamás

munkahelyének neve, tanszéke: ELTE IK, Programozási Nyelvek és Fordítóprogramok

munkahelyének címe: 1117 Budapest, Pázmány Péter sétány 1/C

beosztás és iskolai végzettsége: docens, PhD

A szakdolgozat címe: Teamfight Tactics in JAVA

A szakdolgozat témája:

(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását)

Egy JAVA nyelvben megírt egy játékos által irányítható játékot szeretnék elkészíteni, aminek alapja egy TFT-típusú játék (Teamfight Tactics). A játékos kap egy bábut, amit ő irányít, és a hozzá tartozó életeréje a játék történéseitől függően változik. Ezzel a bábuval támadhatjuk meg a számítógép által irányított ellenségeket. A játék menete több körből áll, minden kör végén a játékos kap bizonyos összegű aranyat, amiből vásárolhat vagy újra sorsoltathatja a megkapott kártyákat, illetve fejlesztheti a játékos adott szintjét, és így több karaktert birtokolhat a mezőn, hogy előnyt kovácsoljon a következő körökhöz. A játékos minden kör elején választhat a kisorsolt kártyák közül, ahol a kártyák különböző karaktereket jelölnek, ezeket leteheti és taktikusan elhelyezheti minél jobb pozícióba. Minden karakternek más-más életeréje és támadóereje lesz, valamint a karakterek fejleszthetőek lesznek, hogy erősebbek lehessenek. Az erősítés több egy karaktert jelképező kártyák kombinálásból történik. A karakterek egy meghatározott algoritmus alapján támadják meg az ellenfeleit, figyelembe véve a karakter kasztját, az ellenség elhelyezkedését, és további tulajdonságokat. A játék akkor ér véget, ha a játékos legyőzi az összes ellenfelet, vagy a bábujának elfogy az élete. Egy játszma kap egy adott időintervallumot, amin belül győznie kell valamelyik félnek, ellenkező esetben a karakterek folyamatosan sebződnek, így kikerülve a végtelen játékokat. Egy adatbázisban tervezem tárolni a játszmák kimenetelével kapcsolatos információkat.

Budapest, 2020.11.18.

Tartalom

Bevezetés	3
A motiváció és a cél	3
Játékmenet.....	3
A játék története	3
Felhasznált technológiák.....	4
Felhasználói dokumentáció	5
Célközönség.....	5
Telepítés.....	5
A játék története	7
Játék szabályok	7
User interface	9
Főmenü.....	9
Exit menü	9
Data menü	10
Help menü	10
Options menü	11
Start Game / Loading menü	11
Játék	12
Kártyapanel	13
Megvásárolható karakter kártya.....	14
Kártyák frissítése.....	14
Lakat.....	15
Tapasztalati pont vásárlás	15
Kispad	16
Harcmező	16
Játékos karaktere	17
Harci egységek és tulajdonságai	17
Carrot	18
Red Slime.....	18
Saint	18
Apple Slime.....	19
Spiked Slime	19
Wise	19
Purple Slime.....	20
Karakter kasztokról röviden.....	20
Assassin.....	20
Mage.....	20

Bruiser	20
Healer	20
Summoner	20
Minion	20
Troubleshooting	21
Fejlesztői dokumentáció	22
Specifikáció.....	22
Játékmenet.....	22
Játék szabályok	22
Megoldási terv.....	23
Használati eset.....	25
Multithreading.....	25
Grafikai tervek	26
Adatbázis.....	26
Csomagszerkezet.....	27
Osztályok bemutatása.....	28
Megvalósítás	30
Design decisions	30
Osztályok közti kapcsolatok	31
Osztályok UML diagramjai csomagonként.....	33
Főbb osztályok és metódusai	45
Javításra váró ismert hibák.....	60
Buildelés és futtatás	60
Rögtön indítható.....	60
IntelliJ IDEA	60
Windows 10	61
Linux	61
Tesztek	61
Tesztelés.....	61
Automatizált tesztek.....	61
Manuális tesztek.....	62
Összegzés.....	66
Továbbfejlesztési lehetőségek.....	66
Felhasznált tartalmak	68

Bevezetés

A motiváció és a cél

A főbb motivációm minden egyes dologban, mint az életemben az, hogy minél nehezebb és bonyolultabb feladatokkal foglalkozzak, ami nem hagyja az agyamat kikapcsolni. Ezért is találtam ki, hogy egy ismert játékot példának véve, tökéletes feladat lesz a számomra, amelyen több ember is dolgozik/dolgozott. Természetesen nem lett, és sose lesz olyan 100%-ban, mint az a program, ami az „ötletemet” szülte. Azért is, mert nem szeretnék ugyanolyat alkotni, ami már létezik, valamint nem is tudnám teljesen lemásolni ennyi idő alatt. A főbb célom a motivációm mellett pedig az volt, hogy egy olyan programot hozhassak létre, ami nagyjából tükrözi a programozási készségeimet. Persze ezt nehéz megmutatni egy szűk határidőn belül, mert még bőven tudnám és lehetne tovább fejleszteni a játékot, amelyről szeretném később tájékoztatni az érdeklődő olvasómat.

Játékmenet

A Teamfight Tactics in JAVA című szakdolgozat egy JAVA nyelvben megírt játék, amelynek a célja az, hogy a játékos hadsereget toborozzon, hogy legyőzze az ellenségét. A játékosnak van pénze, és egy irányítható karaktere, amit nevezhetünk, akár tábornoknak. Mint minden tábornok, mi nem veszünk részt a harcban, de nekünk kell kiadni a katonáknak a pozíciót, ezáltal növelve a győzelmi esélyünket. Például logikus, hogy egy távolsági egységet nem az első sorba állítunk be, hiszen ők támogathatják hátulról a közelharcosainkat, akár támadóerővel, akár gyógyítással. Viszont ezek a katonák pénzbe kerülnek, és nincs mély pénztárcánk az elején, bizonyítanunk kell a rátermettségünket, hogy jól helyezzük el a csapatunkat. Minden katona fejleszthető. Természetesen minél nagyobb szintű a harcosunk, annál erősebb lesz a harcmezőn. Minden egységnek más-más támadóereje, életereje, sőt más egyedi képessége van, amit bevethet a harcban az életben maradásért, győzelemért. A játékos minden kör végén kap pénzt, ami a játék kimenetelétől több, vagy kevesebb is lehet.

A játék története

A fiatal John herceg egy kis varázslótanonc, aki megtalálta az élete szerelmét a varázslóiskolában. A szerelme igent mondott neki, ezért megkezdték a házasságuk megtervezését. Azonban a menyegzőn a gonosz sárkány irigyen hallotta, a fiatal

menyecske szépségét, ezért megjelent ott hívatlanul és elrabolta őt. A kis hercegünk elhatározta, hogy megmenti hőn áhított szerelmét és haza viszi. Ezért fogta a kis arany batyuját és elindult egyedül, hogy legyőzze a sárkányt és visszaszerezze a menyasszonyát, hogy boldogan élhessenek tovább, mint a szép mesékben. De a kis herceg tudomására jutott, hogy a gonosz sárkány, hadsereget toborzott, hogy távol tartsa a hősünket. Ezért muszáj lesz neki is szövetségeseket gyűjtenie a célelérésének a reményében, de a szegény herceg nem volt túl gazdag, ez sejthető is volt abból, hogy egyedül indult neki a harcnak, a saját kevéske aranyával. Azonban ebből a kis vagyonból futja neki néhány katonára, zsoldosra, hogy végül megküzdhessen a gonosz sárkány ellen a csapatával e nemes cél érdekében.

Felhasznált technológiák

A programhoz több technológiát használtam fel, Java Swing, MariaDB, JUnit 5 (JUnit Jupiter). A Java Swing maga a grafikai modulhoz tartozik, ez alapján történnek meg a kirajzolások. A MariaDB az adatbázishoz tartozik, amellyel el lettek mentve a játék kimenetelével kapcsolatos információk. Végül pedig a JUnit 5 (JUnit Jupiter) a tesztesetekhez használtam fel, amivel ellenőriztem a programot.

Felhasználói dokumentáció

Célközönség

A program célközönsége minden olyan ember, aki szívesen taktikázna egy gép ellen és játszana egy jót, amikor csak teheti, akár 5-10 perc esetében is. Az ajánlott korosztály körülbelül a 7-99 év közöttiek közé sorolható be.

Telepítés

A játék telepítéséhez szükséges a Java SE JRE, abból is a 14-es verzió, valamint a MariaDB jelenlegi legújabb verziója a 10.5.9-es. A játék futtatásához feltehetőleg szükséges minimum gépigény megfelel a fejlesztői környezet minimumához.

Minimális képernyőfelbontás: 1280x960

Minimális gépigény:

Windows:

- Windows 10 (8u51 és afölött)
- Windows 8.x (Számítógép)
- Windows 7 SP1
- Windows Vista SP2
- Windows Server 2008 R2 SP1 (64-bit)
- Windows Server 2012 and 2012 R2 (64-bit)
- RAM: 128 MB
- Tárhely: 124 MB JRE-hez; 2 MB Java Update-hez + MariaDB és a program mérete
- Processzor: Minimum Pentium 2 266 MHz processzor
- Böngészők: Internet Explorer 9 és afölött, Firefox

Mac OS X:

- Intel-alapú Mac futtatása Mac OS X 10.8.3+, 10.9+
- Adminisztrátor jogosultság a telepítéshez
- 64-bit böngésző
- Egy 64-bit böngésző (Safari, például) elvárt, hogy fusson az Oracle Java Macen.

Linux:

- Oracle Linux 5.5+1
- Oracle Linux 6.x (32-bit), 6.x (64-bit)2

- Oracle Linux 7.x (64-bit)² (8u20 és afölött)
- Red Hat Enterprise Linux 5.5+1 6.x (32-bit), 6.x (64-bit)²
- Red Hat Enterprise Linux 7.x (64-bit)² (8u20 és afölött)
- Suse Linux Enterprise Server 10 SP2+, 11.x
- Suse Linux Enterprise Server 12.x (64-bit)² (8u31 és afölött)
- Ubuntu Linux 12.04 LTS, 13.x
- Ubuntu Linux 14.x (8u25 és afölött)
- Ubuntu Linux 15.04 (8u45 és afölött)
- Ubuntu Linux 15.10 (8u65 és afölött)
- Böngésző: Firefox

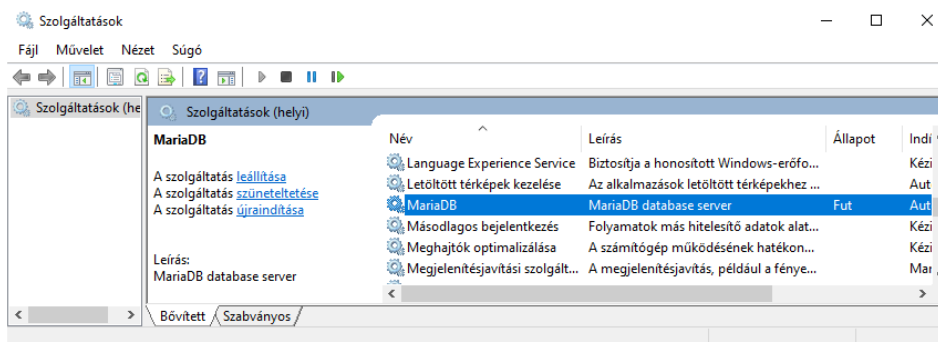
A MariaDB-t a hivatalos oldaláról¹ kell letölteni, a Download menüpont alatt válasszuk ki a legfrissebb verziót, ami jelenleg a MariaDB Server 10.5.9. Majd az operációs rendszert, az architektúrát, illetve a telepítési csomagot is. A telepítés során a Default instance properties 1. ablakában a **jelszó** legyen **bezzegh32**, valamint **legyen bepipálva a(z) UTF-8**, a Default instance properties 2. ablakában pedig az **Install as Service** legyen bepipálva és a **Service Name** legyen **MariaDB**.

1. ábra - MariaDB telepítése 1. ablak

2. ábra - MariaDB telepítése 2. ablak

¹ <https://mariadb.org>

Sikeres telepítés esetén el is indul a szolgáltatás a háttérben. Természetesen, ezt ki- és be is lehet kapcsolni a Szolgáltatásoknál.



3. ábra - MariaDB ki- és bekapcsolása

Ezek után már nyugodtan el lehet indítani a játékot a Teamfight Tactics.jar fájljal, ami a Teamfight Tactics mappán belül található.

A játék története

A fiatal John herceg egy kis varázslótanonc, aki megtalálta az élete szerelmét a varázslóiskolában. A szerelme igent mondott neki, ezért megkezdtek a házasságuk megtervezését. Azonban a menyegzőn a gonosz sárkány irigyen hallotta, a fiatal menyecske szépségét, ezért megjelent ott hívatlanul és elrabolta őt. A kis hercegünk elhatározta, hogy megmenti hőn áhított szerelmét és haza viszi. Ezért fogta a kis arany batyuját és elindult egyedül, hogy legyőzze a sárkányt és visszaszerezze a menyasszonyát, hogy boldogan élhessenek tovább, mint a szép mesékben. De kis herceg tudomására jutott, hogy a gonosz sárkány, hadsereget toborzott, hogy távol tartsa a hősünket. Ezért muszáj lesz neki is szövetségeseket gyűjtenie a célelérésének a reményében, de a szegény herceg nem volt túl gazdag, ez sejthető is volt abból, hogy egyedül indult neki a harcnak, a saját kevéske aranyával, azonban ebből a kis vagyonból futja neki néhány katonára, zsoldosra, hogy végül megküzdhessen a gonosz sárkány ellen a csapatával e nemes cél érdekében.

Játék szabályok

A játékos egy karaktert irányít, aki nem vesz részt a harcban, mint egy jó tábornok, de van életereje, ami a játék kimenetelétől függően változhat, ha az életereje eléri a nullát, akkor elveszti a játékot. A felhasználó karaktere segítségével fel lehet venni az ellenfelektől esett aranyat, aminek a dobási aránya csupán néhány százalék. A játékosnak van egy maximum limitje minden kör elején, hogy hány harcost küldhet harcba a következő körbe. Minden kör után kap +2 tapasztalati pontot, ami a hadsereg

létszámát kitolhatja, ha eléri az adott tapasztalati pont maximumát. A játékos vásárolhat +4 tapasztalati pontot, 4 aranyért cserébe. A megvásárolt egységek nem lesznek egyből a harcmezőn, megvásárlás után, úgymond a kispadra kerülnek. A kispadon lévő egységek nem vesznek részt a harcban, ezeket a felhasználónak kell elhelyezni, viszont, ha ez nem történik meg, de van megvásárolt egység, akkor automatikusan a harcmezőre lesznek rakva, mielőtt elkezdődne a harc, azonban, ha egyetlen egy harcost sem vásárolt, akkor automatikusan kikap az adott körben, ezáltal életerőt veszítve. A katonák fejleszthetők, három egyes szintű, egy kettes szintűvé alakul át és három kettes szintű egy hármas szintűvé, ez mind automatikusan történik meg, tehát nem lehet három azonos egység, azonos szinten, kivéve, ha eléri a maximum szintet, ami a hármas, három hármas szintű már lehet a harcmezőn és a kispadon egyaránt. Ha a játékos kikap, akkor a következő ellenfele ugyanaz lesz, ugyanott fognak elhelyezkedni és ugyanolyan szintűek lesznek. Ha egy egység meghal, újjászületik a következő körre, tehát nem veszíti el a katonákat a felhasználó, viszont ez az ellenfélre is igaz, ha nem győztük le az adott körben, minden egysége feltámad. A játékos minden kör végén kap valamennyi aranyat, ezzel is lehet taktikázni, ha minél többet nyer egymás után, annál többet fog kapni a kör végén, valamint, ha több, mint 10 aranya van, az minden kör végén +1 aranyat ér, még pedig úgy, hogy ha 10-zel osztható a pénzösszege az a maradékkal egyenlően hozzáadja a vagyonunkhoz, de ez maximum 5 arany lehet, tehát 60 arany esetén, 5 aranyat kap ebből. Ha a játékosnak vereség szériája van, a játék megpróbálja visszahozni az adott játékba, ugyanúgy, mint a győzelem esetén, több vereség egymás után több pénzt jelent. Minden új kör elején frissülnek a vásárolható katonák, ezeknek a katonáknak a létszáma 5, viszont frissítheti a játékos is 2 aranyért cserébe, valamint, ha nincs elegendő pénze az adott körben, de megszeretné venni a katonát, és biztosra akar menni, hogy megmaradjon, le lehet lakatolni a kártyákat, de ekkor a felhasználó sem tudja frissíteni az adott paklit, csak ha kikapcsolja ezt a funkciót.

User interface

Főmenü

A játék egy menüvel indul el, a menüben ötféle lehetősége van a felhasználónak.



4. ábra - Főmenü

Exit menü

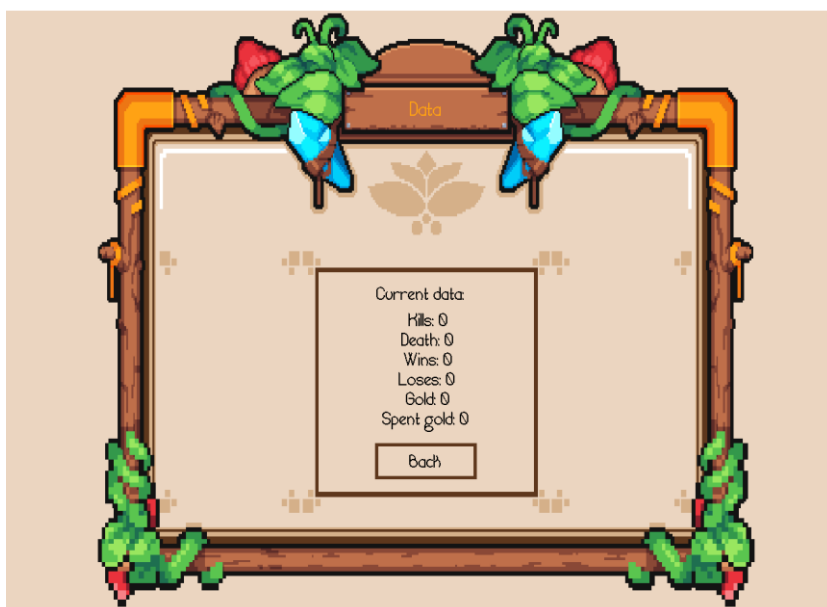
Az **exit** menü segítségével tud kilépni a felhasználó a játékból, ezt megelőzi egy kérdés, hogy biztosan ki akar-e lépni, ha igen, akkor az **igenre** kattintva kiléphet a játékból. Egyébként a **nemmel**, pedig visszatér a főmenübe.



5. ábra - Exit menü

Data menü

A **data menüben** a játékkal kapcsolatos adatok találhatóak meg. Ezek az adatok fentről lefelé a következők: az összes legyőzött ellenfél, az összes elvesztett harci egység, az összes győzelem, az összes vereség, az összes arany, amit szerzett, valamint az összes elköltött arany.



6. ábra - Data menü

Help menü

A **help menüpontban** található némi információ a játékkal kapcsolatban, hogyha valamit nem tudna vagy nem értene még a felhasználó.



7. ábra - Help menü

Options menü

Az **options menüpontban** be lehet állítani, hogy mutassa a játék az fps-t², valamint nullázni az adatbázis adatait.



8. ábra - Options menü

Start Game / Loading menü

A **start gamere** kattintva elindul a játék, amit egy Loading képernyő előz meg, itt várni kell valamennyi időt a számítógép erejétől függően, hogy a játék betöltsön.

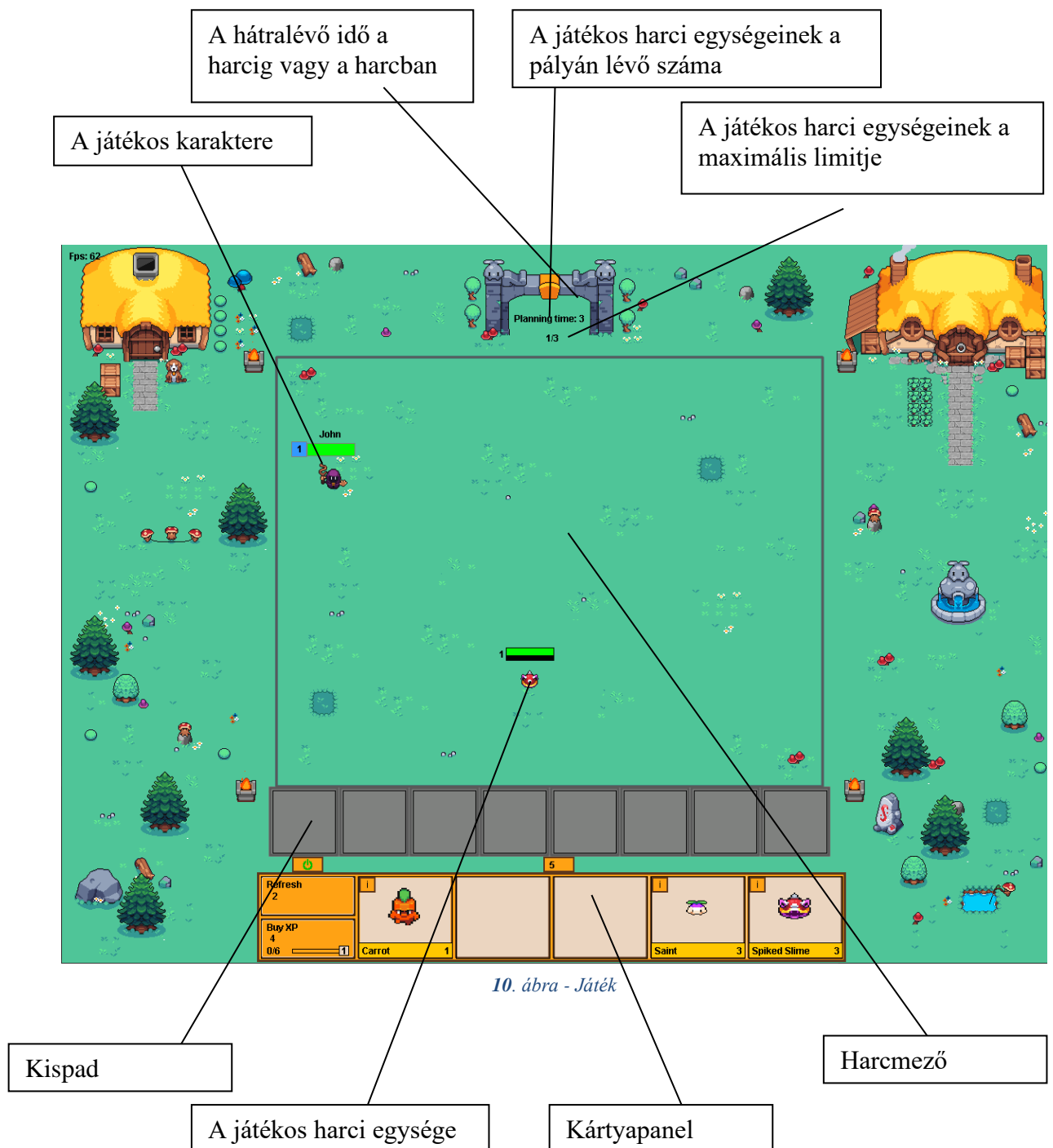


9. ábra - Start Game / Loading Menü

² Frame per second (Képkocka per másodperc)

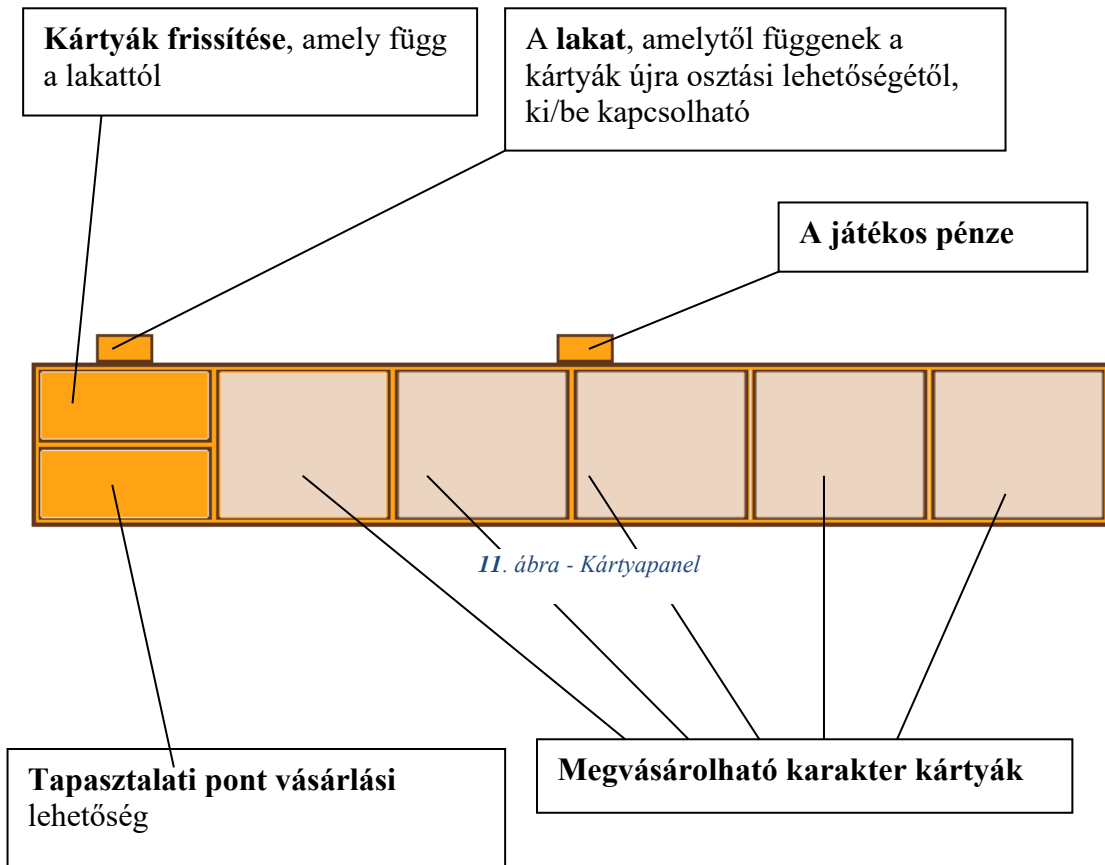
Játék

A(z) user interface több lépésből tevődik össze. Lépésenként a következőképpen néz ki:



Kártyapanel

Ha nem fog meg egy karaktert sem a felhasználó, akkor a következőképpen néz ki a kártyapanel alaphelyzetben:

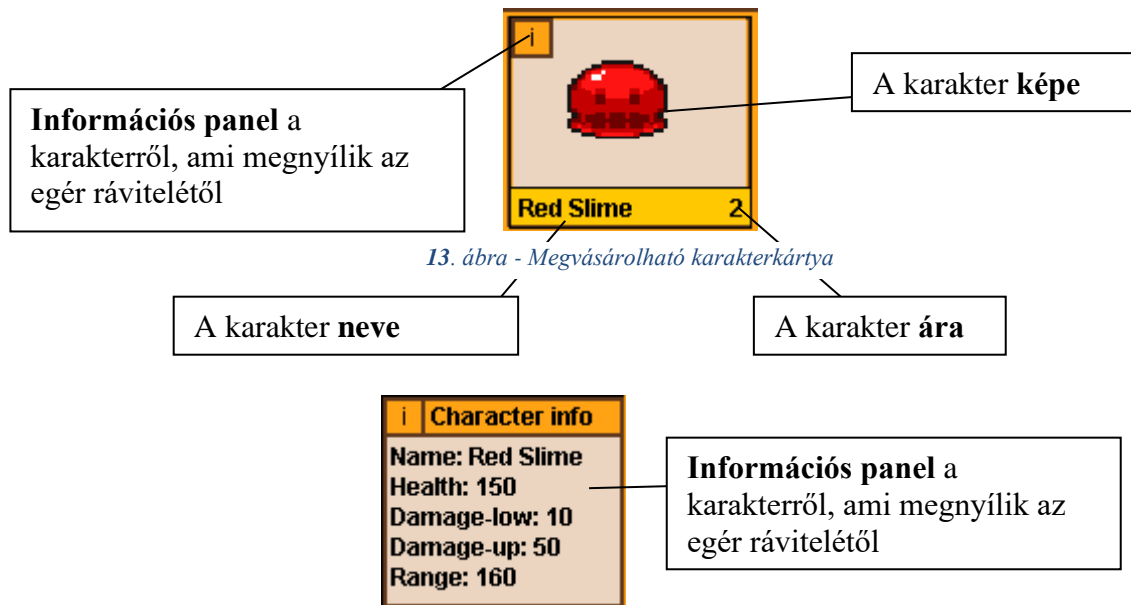


Ha megfog egy karaktert a felhasználó a platformon, akkor a kártyapanel átváltozik a következőre:



Megvásárolható karakter kártya

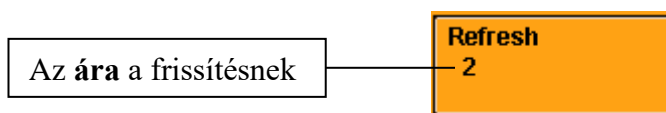
A kártya megvásárolható „X” összegért, ha van rá elegendő pénze a felhasználónak és a kispad sincs teli. Megvétel előtt megnézheti a karakter leírását, ha az „i”-re ráviszi az egeret. Ha megvette a kártyát, akkor a játék automatikusan elhelyezi egy szabad padra, ahonnan a játékos kedve szerint elhelyezheti a csatamezőn, ha van még rá kapacitása (nincs elérve az adott szinten elérhető maximális harci egysége).



Kártyák frissítése

Kártyák frissítése, amely függ a lakat állapotától, ha rákattint a felhasználó erre a területre, akkor frissíti a megvásárolható kártyákat, ami 2 arany árából lehetséges.

Lakat nyitva:



14. ábra - Refresh nyitva

Lakat zárva:




15. ábra - Refresh zárva

Lakat

A lakat által megakadályozhatja a felhasználó azt, hogyha véletlenül rányomna a frissítésre vagy pedig, hogy a kör végén új kártyák legyenek kisorsolva, például, ha 1 arany hiányzik az egység megvásárlásához és feltehetőleg tudja a felhasználó, hogy ezt mindenképpen megszerezheti a következő kör elején, akkor rákattinthat a lakatra, hogy ne legyenek frissítve a karakter kártyák az új kör ellenére.

Lakat kikapcsolt állapotban: 

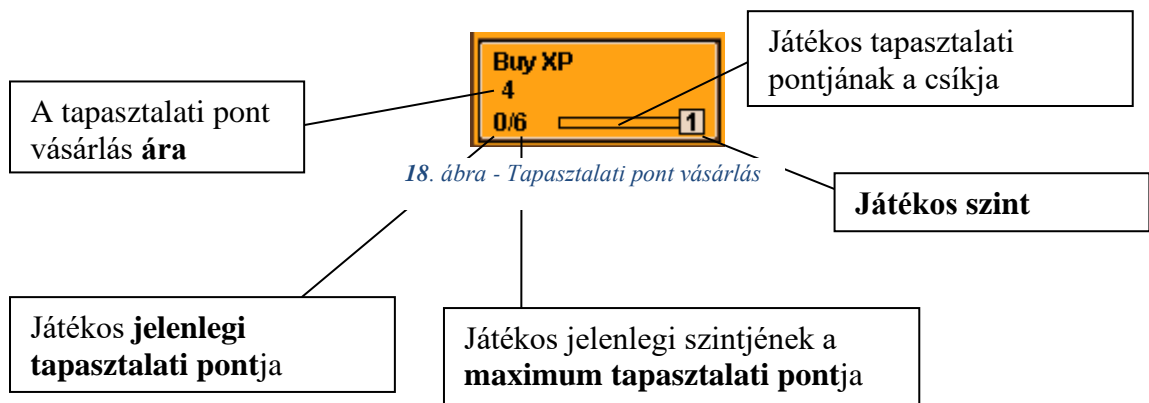
16. ábra - Lakat nyitva

Lakat bekapcsolt állapotban: 

17. ábra - Lakat zárva

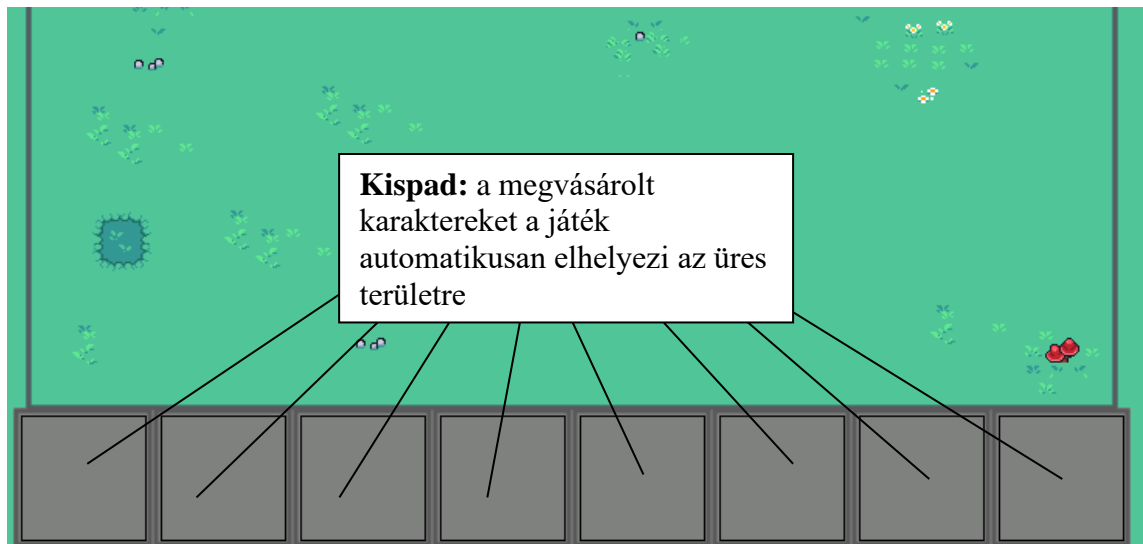
Tapasztalati pont vásárlás

Tapasztalati pontot lehet vásárolni 4 aranyért cserébe. Ha eléri a maximális tapasztalati pontot, akkor fejlődik a karakter, ezáltal több egység tehető le a harcmezőre.



Kispad

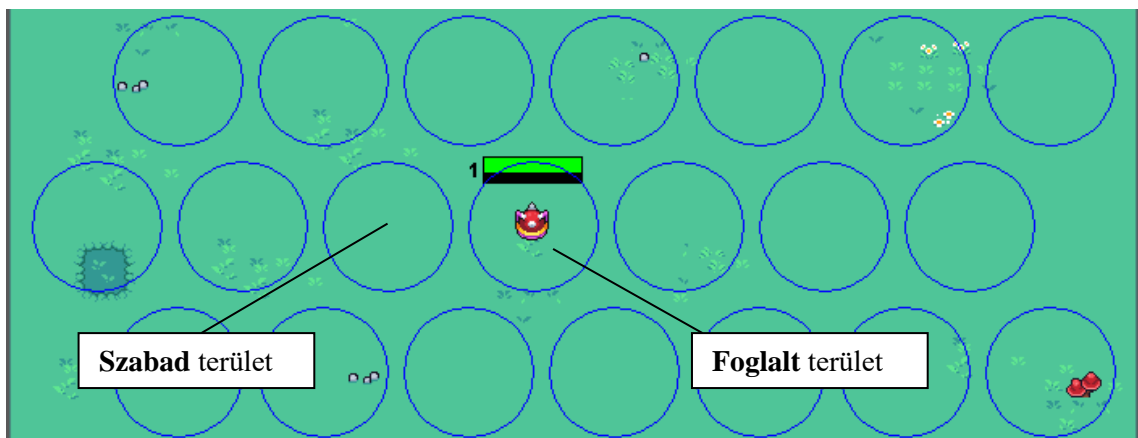
A megvásárolt karakterek úgymond a kispadra kerülnek, ahonnan elhelyezhetők a csatamezőre. Ez a terület egy 1x8-as csík, amelyre összesen 8 karakter helyezhető el. Ezek a karakterek bármikor cserélhetők egymás pozíciójával.



19. ábra - Kispad

Harcmező

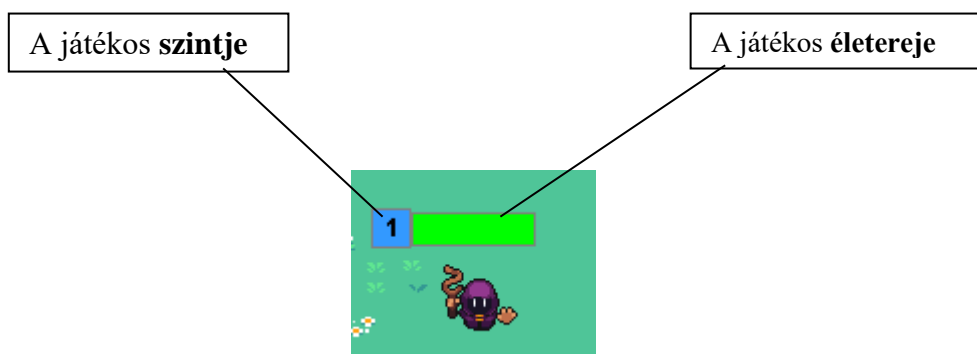
A kispadon lévő karaktereket elhelyezheti a harcmezőn a felhasználó, hogyha még nem érte el a szintjének a maximális harci egységeinek a limitjét. A harcmezőkön lévő egységek fognak megküzdeni minden egyes körben az ellenféllel. A harcmező egy 6x7-es nagyságú terület, ebből a játékosnak van egy 3x7-es, a másik 3x7-es terület az ellenfelé. Minden 1x1-es területre csak 1 karakter rakható le, egyébként cserélni fogja a karaktert, ha olyan helyre szeretné rakni a játékos a kezében lévő, ami már foglalt.



20. ábra - Harcmező

Játékos karaktere

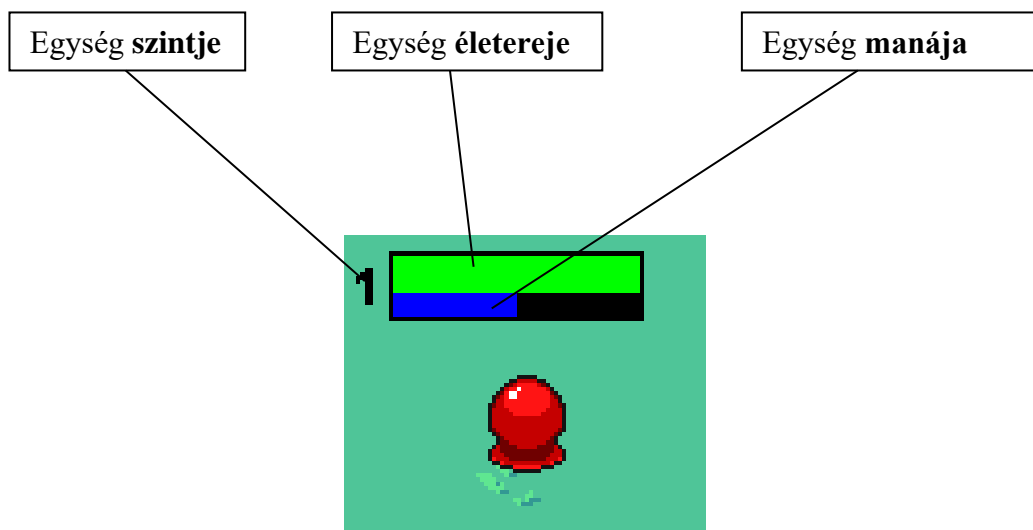
A játékos karakterének néhány feladata van. Az egyik, hogy mutatja a játékos adott életerejének a pontját, a másik, hogy mutatja a játékos szintjét, utoljára pedig a legyőzött karaktereknek van egy kisebb esélye, hogy dobznak +2 aranyat, ami felvehető a karakterrel. Hogyha nem mozgatja a karaktert, akkor el kezd pörögni egy idő után. A játékos maximális életereje 100.



21. ábra - Játékos karaktere

Harci egységek és tulajdonságai

Minden harci egység más-más támadóerővel, életerővel és egyedi képességekkel rendelkezik. Minden egység 1-es szinten kezd.



22. ábra - Harci egység

Carrot

Karakter neve: Carrot

Életereje: 100

Ára: 1 arany

Sebzés: 25-40

Sebzésének távolsága: 160

Kezdő mana: 25

Osztály: Assassin

Egyedi képessége: A minimum és a maximum sebzés közti sebzésének a kétszeresét üti a célpontra



23. ábra - Carrot

Red Slime

Karakter neve: Red Slime

Életereje: 150

Ára: 2 arany

Sebzés: 10-50

Sebzésének távolsága: 160

Kezdő mana: 50

Osztály: Assassin

Egyedi képessége: A minimum és a maximum sebzés közti sebzésének a háromszorosát üti a célpontra



24. ábra - Red Slime

Saint

Karakter neve: Saint

Életereje: 250

Ára: 3 arany

Sebzés: 15-20

Sebzésének távolsága: 160

Kezdő mana: 20

Osztály: Healer

Egyedi képessége: A legkisebb életerővel rendelkező szövetséges gyógyítása, az egység szintjének az ötvenszerese, hatótávolság nincsen az egész pályán érvényes



25. ábra - Saint

Apple Slime

Karakter neve: Apple Slime

Életereje: 150

Ára: 2 arany

Sebzés: 20-30

Sebzésének távolsága: 600

Kezdő mana: 50

Osztály: Mage

Egyedi képessége: Az egység szintjének az ötvenszeresét sebz az adott ellenfélre



26. ábra - Apple Slime

Spiked Slime

Karakter neve: Spiked Slime

Életereje: 200

Ára: 3 arany

Sebzés: 20-30

Sebzésének távolsága: 160

Kezdő mana: 0

Osztály: Bruiser

Egyedi képessége: Egy területi sebzés, ami két körből áll, egy kisebbből és egy nagyobbból. A kisebb körön belül a képesség sebzésének a $(-10 * \text{karakter szint})$ -et sebz, a sebzése a következő szintenként növe: 60, 120, 200



27. ábra - Spiked Slime

Wise

Karakter neve: Wise

Életereje: 125

Ára: 3 arany

Sebzés: 15-35

Sebzésének távolsága: 400

Kezdő mana: 25

Osztály: Summoner

Egyedi képessége: Létrehoz az ő szintjével egyenlő szintű miniont, ami az oldalán harcol tovább



28. ábra - Wise

Purple Slime

Karakter neve: Purple Slime

Életereje: 100

Ára: Nem megvehető karakter

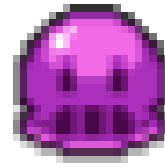
Sebzés: 10-25

Sebzésének távolsága: 160

Kezdő mana: 25

Osztály: Minion

Egyedi képessége: Önmagát gyógyítja a szintjének a harmincötszörösével



29. ábra - Purple Slime

Karakter kasztokról röviden

Assassin

Az assassin osztály kezdésképpen megkeresi a tőle a legtávolabb lévő ellenséges egységet majd célpontjává teszi, ezek után mindig a legközelebbi egységet keresi. Minden sebzését 1300-1500 ms közt teszi meg, valamint minden ütésénél a manája 50-nel nő.

Mage

A mage osztály a legközelebbi ellenséges egységet veszi célpontba. Minden sebzését 1300 - 1500 ms közt teszi meg, valamint minden ütésénél a manája 25-tel nő.

Bruiser

A bruiser osztály a legközelebbi ellenséges egységet veszi célpontba. Minden sebzését 600 - 1000 ms közt teszi meg, valamint minden ütésénél a manája 20-szal nő.

Healer

A healer osztály a legközelebbi ellenséges egységet veszi célpontba. Minden sebzését 1000 - 1500 ms közt teszi meg, valamint minden ütésénél a manája 20-szal nő.

Summoner

A summoner osztály a legközelebbi ellenséges egységet veszi célpontba. Minden sebzését 1300 - 1500 ms közt teszi meg, valamint minden ütésénél a manája 25-tel nő.

Minion

A minion osztály a legközelebbi ellenséges egységet veszi célpontba. Minden sebzését 1000-1500 ms közt teszi meg, valamint minden ütésénél a manája 25-tel nő.

Troubleshooting

Ha nem lett kitörölve fájl a játék mappájából véletlenül (akár a vírusírtó, akár szabadkezüleg), akkor a következő hibák lehetségesek:

- A java verzió nem felel meg, ahogy a kód elvárná, Java Version 14.0-ban készült a program
- Nem fut a háttérben az adatbázis, MariaDB Version 10.5.9
- Ha nem a MariaDB nevet adtuk meg a Servicnek vagy ha nem a jó jelszót, akkor nem fog működni az adatbázis a programhoz, valamint nem fog elindulni, ha ez bekövetkezne a Szolgáltatásoknál állítsuk le a MariaDB Service-t és adatbázis nélkül futtatható a program, esetleg a MariaDB helyes újratelepítése is megoldhatja a problémát
- A processzor nem tud egyszerre annyi szálat kezelni, ahányat megkövetel a program
- Nem bírja el a számítógép a minimális gépigényt

Ha a fenti állapotok közül egyik sem igaz, azaz biztos, hogy minden feltételnek jónak kell lennie, akkor a játék újra telepítése segíthet a probléma megoldásában.

Fejlesztői dokumentáció

Specifikáció

Játékmenet

A Teamfight Tactics in JAVA című szakdolgozat egy JAVA nyelvben megírt játék, amelynek a célja az, hogy a játékos hadsereget toborozzon, hogy legyőzze az ellenségét. A játékosnak van pénze, és egy irányítható karaktere, amit nevezhetünk, akár tábornoknak. Mint minden tábornok, mi nem veszünk részt a harcban, de nekünk kell kiadni a katonáknak a pozíciót, ezáltal növelve a győzelmi esélyünket. Például logikus, hogy egy távolsági egységet nem az első sorba állítunk be, hiszen ők támogatják hátulról a közelharcosainkat, akár támadóerővel, akár gyógyítással. Viszont ezek a katonák pénzbe kerülnek, és nincs mély pénztárcánk az elején, bizonyítanunk kell a rátermettségünket, hogy jól helyezzük el a csapatunkat. Minden katona fejleszthető. Természetesen minél nagyobb szintű a harcosunk, annál erősebb lesz a harcmezőn. Minden egységnek más-más támadóereje, életereje, sőt más egyedi képessége van, amit bevethet a harcban az életben maradásért, győzelemért. A játékos minden kör végén kap pénzt, ami a játék kimenetelétől több, vagy kevesebb is lehet.

Játék szabályok

A játékos egy karaktert irányít, aki nem vesz részt a harcban, mint egy jó tábornok, de van életereje, ami a játék kimenetelétől függően változhat, ha az életereje eléri a 0-t, akkor elveszti a játékot. A játékosnak van egy maximum limitje minden kör elején, hogy hány harcost küldhet harcba a következő körbe. Minden kör után kapunk +2 tapasztalati pontot, ami a hadsereg létszámát kitolhatja, ha eléri az adott tapasztalati pont maximumát. A játékos vásárolhat +4 tapasztalati pontot, 4 aranyért cserébe. A megvásárolt egységek nem lesznek egyből a harcmezőn, megvásárlás után, úgymond a kispadra kerülnek. A kispadon lévő egységek nem vesznek részt a harcban, a játékosnak kell elhelyeznie őket, viszont, ha ez nem történik meg, de van megvásárolt egység, akkor automatikusan a harcmezőre lesznek rakva, mielőtt elkezdődne a harc, azonban, ha egyetlen egy harcost sem vásárolt, akkor automatikusan kikap az adott körben, ezáltal életerőt veszítve. A katonák fejleszthetőek, három egyes szintű, egy kettes szintűvé alakul át, és három kettes szintű, egy hármas szintűvé, ez mind automatikusan történik meg, tehát nem lehet három azonos egység, azonos szinten, kivéve, ha eléri a maximum szintet, ami a hármas, három hármas szintű már lehet a harcmezőn és

kispadon egyaránt. Ha a játékosunk kikap, akkor a következő ellenfele ugyanaz lesz, ugyanott fognak elhelyezkedni és ugyanolyan szintűek lesznek. Ha egy egység meghal, újjászületik a következő körre, tehát nem veszítjük el a katonákat, viszont ez az ellenfélre is igaz, ha nem győztük le az adott körben, minden egysége feltámad. A játékos minden kör végén kap valamennyi aranyat, ezzel is lehet taktikázni, ha minél többet nyerünk egymás után, annál többet fogunk kapni a kör végén, valamint, ha több, mint 10 aranyunk van, az minden kör végén +1 aranyat ér, még pedig úgy, hogy ha 10-zel osztható a pénzeszközünk az a maradékkal egyenlően hozzáadja a vagyonunkhoz, de ez maximum 5 arany lehet, tehát 60 arany esetén, 5 aranyat kapunk ebből. Ha a játékosnak vereség szériája van, a játék próbálja visszahozni az adott játékba, ugyanúgy, mint a győzelem esetén, több vereség egymás után több pénzt jelent. Minden új kör elején frissülnek a vásárolható katonák, ezek a katonák létszáma 5, viszont frissíthetjük mi is 2 aranyért cserébe, valamint, ha nincs elegendő pénzünk az adott körben, de megszeretnénk venni a katonát, és biztosra akarunk menni, hogy megmaradjon, le lehet lakatolni a kártyákat, de ekkor a játékos sem tudja frissíteni az adott paklit, csak ha kikapcsolja ezt a funkciót.

Megoldási terv

A játékosnak van egy irányított karaktere, melynek van életereje, szintje és maximum harci egység limitje. Ez a limit növelhető a játékos szintjével. A karakter a jobb egérgomb lenyomásának a pozíciójába irányítható. A játéknak úgymond három különböző területe van: a kártyapanel, a kispad és a harcmező.

A kártyapanel:

- *Lakat*: a kártyák újra sorsolásának megakadályozása, ha lenyomjuk, a játékos nem tud a Refresh gomb segítségével frissíteni és a játék sem új kör esetén
- A panel mutatja a játékos pénzét

Ha nem fogunk harci egységet:

- 5 harci egység vásárolható meg rajta egyszerre, amely minden új körben megváltozhat
- *Refresh gomb* segítségével újra pörgethetjük az 5 harci egységet, ha nincs lelakatozva
- *Tapasztalati pont vásárlás gomb* esetén vehetünk 4 tapasztalati pontot, hogy fejlődhessen a karakterünk és ezáltal több harci egységet birtokolhassunk a harcmezőn

Ha fogunk harci egységet:

- A fogott karakter eladható, ha lehúzzuk az adott területre

A kispad:

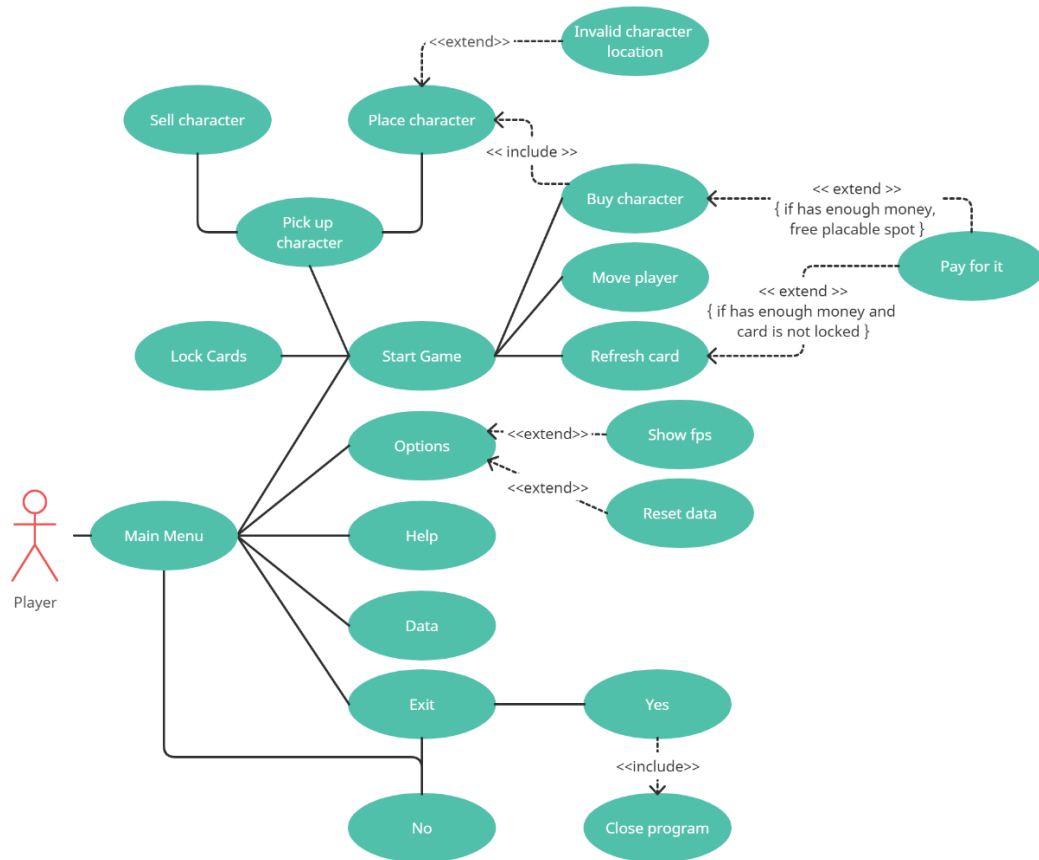
- A kispad egy 1x8-as méretű terület, ahova a játék automatikusan elhelyezi a játékos által vásárolt harci egységeket
- Ha a kispad mind a 8 területe foglalt, nem vásárolható új karakter, míg nincs hely neki

A harcmező:

- A játékosnak és az ellenfelének is van egy 3x7-es harcmezője, ahova szabadon leteheti a harci egységeket
- Annyi harci egységet birtokolhatunk a harcmezőn, ahány a játékos jelenlegi szintje megenged, 1-es szinten 3, 2-es szinten 4, egyesével növekedve szintenként
- A harcmezőn lévő harci egységek harcolnak egymás ellen

A harcmezőn és a kispadon szereplő harci egységek pozícióját kedvünkre váltogathatjuk, például két karakter kicserélhető egymással. Minden kör végén megfelelően osszuk ki a játékosnak a pénzt, amely függ a győzelmi, vagy pedig a vereségi sorozattól, valamint a jelenlegi pénzétől is. Minden jelenlegi 10 arany birtoklása után +1 arany a kör végén, ez maximum 5-ig megy. Vereség esetén a játékos életerejének csökkentése, valamint a vereséget okozó karakterek visszaállítása. Győzelem esetén új ellenfelek létrehozása. Minden kör végén helyre kell állítani a harci egységek életerejét és pozícióját az eredetihez képest, úgymond feltámasztani. Létrehozni kettő Swing Timer-t, ami az egyik a harci egységek elhelyezésére van, a másik pedig a harc döntetlen elkerülésének érdekében. Legyőzött ellenfeleknek beállítani az esélyt arra, hogy pénzt dobjanak. Animációk updatelése, elindítása, megállítása. A karakterek szintjének automatikus fejlesztése, hogyha három azonos szintű van a pályán, kivéve, ha három maximális szintű. A karakterek több szálon való futása. Valamint a játék kimenetelével kapcsolatos adatbázisának létrehozása, amely adatai a menüben lekérdezhető és resetelhető legyen.

Használati eset

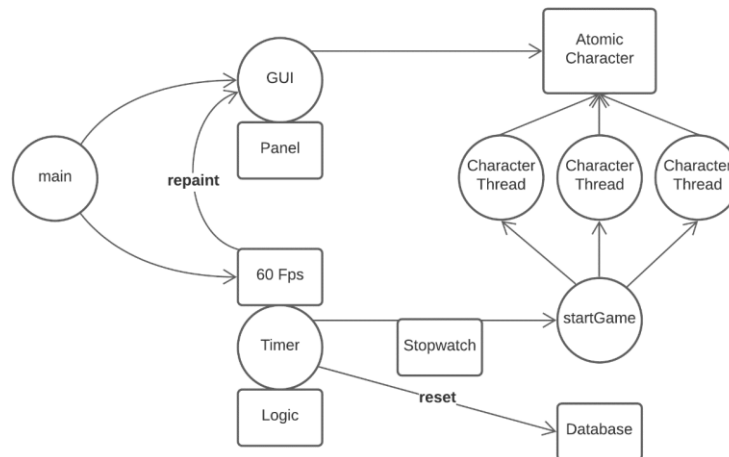


30. ábra - Use case diagram

Multithreading

A program több szálon lett megvalósítva. A játék alapvetően egy szálról kezdődik, ami egyből elindít egy `SwingUtilites.invokeLater()` szálát (GUI szál), valamint egy `Timer` szálát (Logic szál). A GUI szál a grafikai részére lett létrehozva, a rajz animációknak a frissítéséért. A logikai szál, ahogy a nevében is benne van, maga a játék logikájáért létezik. A programban továbbá van két `Stopwatch`, amely segíti a logika menetét, az egyik a `fightClock`, a másik pedig a `planningClock`, ezek egyszerre nem futnak, felváltva cserélgetik egymást. A `Stopwatch` egy adott időponttól folyamatosan lefelé csökken (a `planningClock` 15 másodperces, a `fightClock` 30 másodperces), ha eléri a nullát, akkor váltják egymást. Ha lejár a `planningClock` egy új szálon elindítja a játékot (ha nem indítja el új szálon, akkor a háttérben lévő harc fut, de a grafikai része úgymond megáll, ezért nem lesz látható a játék kimenetele, csak abból tudna következtetni a játékos a kör végén, hogy a karaktere élete változott-e). Amint elindul ez az új szál a karaktereket külön-külön szálon útjára engedi, majd joinoltatja őket a végén. A

karakter szálak atomic változókkal rendelkeznek és kommunikálnak a gui szállal, ezáltal az életerő, mana és hasonlók változása folyamatosan látható lesz a játék közben.



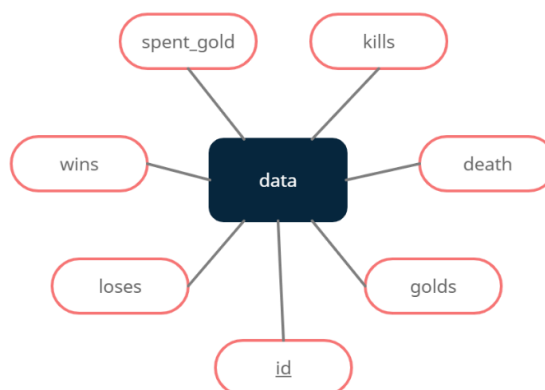
31. ábra - Multithreading

Grafikai tervek

A játék grafikája alapvetően 2D-re lett kitalálva, de természetesen egy 2.5D szebb lenne (felülnézetős 3D-ben látszódó rajzok, de csak 2D-ben van megrajzolva). Az animációk SpriteSheet alapján valósulnak meg, ami azt jelenti, hogy egy adott rajznak az animációi/képkockái egy képen szerepelnek, vagyis több kép egy képen. Például, ha egy karakter balra mozgása négy képkockából jönne létre, akkor ezek egy képen szerepelnének. A játéksíkon lévő karakterek sima Sprite-ok, de megvéve őket már SpriteSheet-ek alkotnák, amit az Animáció váltogathatna.

Adatbázis

A játék adatbázisa nem egy bonyolult rendszert valósít meg. Egyetlen egy táblával dolgozik, melynek több értéke is van, mint például az elköltött arany és a győzelmek száma.



32. ábra - Adatbázis diagram

Csomagszerkezet

A program 5 nagy csomagra van szedve, ez az öt pedig nem más, mint a:

- **gui:** A grafika csomag
- **io:** A beolvasás csomag
- **model:** A logika csomag
- **persistence:** Az adatbázis csomag
- **resources:** A képfájl csomag

A program csomagszerkezete a következőképpen néz ki csomagonként:

- **gui:** *Menu*, *TeamfightTacticsFrame*, *TeamfightTacticsPanel*, *UIConstants*, ezek a fő osztályai a programnak, amelyek a grafikáért szerepelnek
 - **gui.sprite:** *ISprite*, *Sprite*, *SpriteSheet*, a grafikáért szereplő kép beolvasásának és annak a kezelése a szerepük.
 - **gui.sprite.characters:** *AppleSlime*, *Carrot*, *PurpleSlime*, *RedSlime*, *Saint*, *SpikedSlime*, *Wise*, az összes létező karakter, amely lényegében csak grafika egy kicsi logikával.
 - **gui.sprite.other:** *Bullet*, *Gold*, ide tartoznak az olyan Sprite-ok, amelyek még a grafikában szerepelnek.
 - **gui.sprite.player:** *Player*, a *Player* osztály a *Sprite* osztályból leszármazva ide tartozik, amelyben a játékos kirajzolása és a logikája is szerepel egyaránt.
- **io³:** *PropertiesReader*, az adatbázis configját olvassa be.
- **model:** *Character*, *Direction*, *FieldState*, *GameConstants*, *GameState*, *Stopwatch*, *TeamfightTacticsLogic*, ezek az osztályok a program logikája.
 - **model.animation:** *Animation*, *Frame*, az animációért szerepelnek, de a logikához tartoznak, mert számításokat végeznek a képekkel.
 - **model.classes:** *Assassin*, *Bruiser*, *Healer*, *Mage*, *Minion*, *Summoner*, minden olyan osztály, amelyet a karakterek örökölnek és alkalmaznak.
 - **model.map:** *LoadGame*, *Key*, betölti a játék képeit és megcsinálja a pályát, valamint elindítja a játékot a betöltés után.
- **persistence:**
 - **persistence.connection:** *DataSource*, *PropertiesData*, az adatbázishoz való csatlakozásának létrehozása a szerepük.

³ input / output (bemenet, kimenet)

- **persistence.dao:** *DataDao*, *IEntityDao*, az adatbázishoz tartozó lekérdezésekért léteznek.
- **persistence.entity:** *AbstractEntity*, *DataScore*, *Identifiable*, az adatbázis lényege úgymond, innen lehet lekérni az adatokat és frissíteni azokat a dao csomag segítségével.
- **resources:** *ResourceLoader*, a képek beolvasásáért szerepel.
 - **resources.images:** A játék képei itt találhatóak.
 - **resources.images.characters:** A játékban lévő karakterek SpriteSheet-jei ebben a csomagban vannak.
 - **resources.images.textures:** Minden olyan kép, amelyek nem a karakterekért, hanem a játék szépségéért léteznek, például menü animáció és a mozgó kutya.

Osztályok bemutatása

Két főbb modul van, az egyik a logikai, a másik a grafikai.

Logikai modul osztályai és szerepei röviden:

- **Animation:** A karakter animációért felelős osztály, amely *BufferedImage* tömb és *frameDelay* alapján dolgozik
- **Boot:** A játék indításáért felelős, amely létrehozza az ablakot, valamint a logikát
- **ResourceLoader:** A játék összes képének a betöltéséért ő felel
- **Direction:** Egy enum osztály, amely a karakterek mozgásának és animációjának az irányát adja meg
- **FieldState:** Egy enum osztály, amely a harmező és a kispad működéséért vállal szerepet, hogy van-e rajta karakter vagy sem (foglalt vagy üres)
- **Frame:** Az animációban veszi ki a szerepét, hogy éppen melyik az aktuális képe az animációnak
- **GameState:** Egy enum osztály, amely a menüért és a játék működéséért vállal szerepet
- **LoadGame:** A játéknak majdnem az összes képét itt töltjük be a *ResourceLoader* osztály segítségével, valamint létrehozza a kispadot, a karakterpanelt, a harmezőt, a karaktereket és a játékost is
- **Stopwatch:** A játék timere, amelynek az ideje egy megadott időponttól csökken másodpercenként
- **TeamFightTacticsLogic:** A játék fő logika osztálya, szinte ez felel mindenért

- **Key:** Lényegében a Map key-e átalakítva úgy, hogy nem 1 kulcs van 1 értékkel, hanem 2 kulcs, 1 értékkel (x, y koordináta és a FieldState enum)
- **GameConstants:** A játékhoz szereplő konstansok, például frame méret, fps
- **PropertiesReader:** Az adatbázis configjához használt olvasó
- **DataSource:** Az adatforrás, amivel csatlakozni lehet az adatbázishoz
- **PropertiesDataSource:** Az adatforráshoz tartozó tulajdonságok, mint például az url, user, password
- **DataDao:** Az adatbázis lekérdezése/updatesége a szerepe
- **IEntityDao:** Interface osztály a DataDaohoz
- **AbstractEntity:** Abstract osztály, mely tartalmazza az id-t és végrehajtja az Identifiable interfacet
- **DataScore:** A kiépített adatbázisból ez alapján lehet lekérni az adatokat
- **Identifiable:** Interface osztály az id lekérdezéséhez vagy beállításához
- **Assassin:** Egy karakter kaszt
- **Bruiser:** Egy karakter kaszt
- **Healer:** Egy karakter kaszt
- **Mage:** Egy karakter kaszt
- **Minion:** Egy karakter kaszt
- **Summoner:** Egy karakter kaszt

Grafikai modul osztályai és szerepei röviden:

- **ISprite:** A Sprite osztályhoz való interface
- **Menu:** A játék menüjéért vállal szerepet
- **Sprite:** A játékban lévő képekért felel, amely csak egy adott kép
- **SpriteSheet:** Az animációban veszi ki a szerepét, a spritesheet lényegében egy képben lévő több kép, például egy adott karakternek az animációja több képből áll, és nem 5 képként van betöltve, hanem 1 képként
- **TeamFightTacticsFrame:** A játék ablakot hozza létre. a játék nevével
- **Gold:** Az arany kirajzoltatása
- **Carrot:** Egy karakter kirajzoltatása
- **PurpleSlime:** Egy karakter kirajzoltatása
- **RedSlime:** Egy karakter kirajzoltatása
- **Wise:** Egy karakter kirajzoltatása
- **Saint:** Egy karakter kirajzoltatása

- **AppleSlime:** Egy karakter kirajzoltatása
- **SpikedSlime:** Egy karakter kirajzoltatása
- **TeamFightTacticsPanel:** A főbb grafikai osztály, amely minden kirajzolásban szerepet vállal
- **UIConstants:** UI-hoz tartozó konstansok, mint például szín és kép

Vegyes osztályok és szerepei röviden:

- **Bullet:** A karakterek által kilőtt lövedék mozgatása, valamint kirajzoltatása
- **Character:** A karakterek főbb osztálya, amely a karakterek logikájáért, mozgásáért, támadásáért, valamint kirajzoltatásáért is szerepel
- **Player:** A játékos logikájáért, mozgatásáért, valamint kirajzoltatásáért létezik

Megvalósítás

Design decisions

Mint minden program esetében itt is vannak olyan megvalósítások, amelyekre rákényszerült a programozó vagy pedig jobb megoldásnak találta.

Többszál vs. egy szál

Azért is lett megvalósítva a program többszálon, mert egyrészt kihívásnak véltem, úgyhogy elsősorban egy önző döntésként, de mélyebben belegondolva a főindokom a következő volt: hogyha a játék egy szálon futna minden egyes karakterrel és beállítanánk, hogy az Assassin-ok két másodpercenként üssenek, míg a Mage-k másfél másodpercenként, az mind szép és jó, de mi történik, ha mozognak? És mi történik, ha mozgás közben kéne ütniük? Ekkor minden ilyen karakter átugraná ezt az időpontot és nem ütne, majd lehet, hogy várakozik három másodpercet. De több szálon nincs ilyen probléma, mozgás után (vagy előtte, mert így van kivitelezve) megállva és keresve a célpontot, majd bemérve azt, mindig az ütés után várakozna a balanszolás szempontjából a megfelelő időmennyiséget.

TeamfightTacticsPanel

A TeamfightTacticsPanel osztályban szerepel néhány logikai rész is, amit úgy gondoltam, hogy nem kéne, hogy publikus legyen, de mindenképpen kell az adott osztályban szerepelnie, ezek a felemelt karakterek cseréje, mozgatása és ehhez hasonló dolgok.

Adatbázis

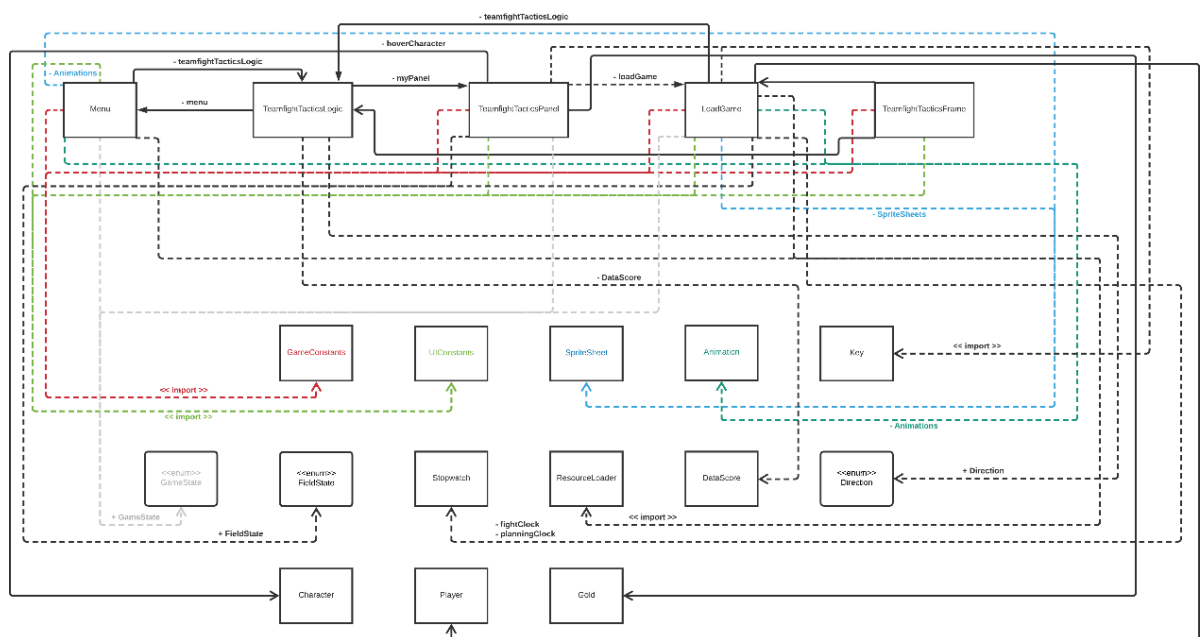
Az adatbázis egy szálon történő megvalósítása (ezt a logikai szál vitelezi ki), hogyha több szálon lenne az adatbázis problémákat tudott volna létrehozni. Például, hogyha a játékos aranyat költ, ami benne van az adatbázisban, és egyből frissítené a program a GUI szálon, valamint a játék körének pont vége lenne és a Logikai szál is meghívna az adatbázist akkor a szálak ütközhetnének egymással.

TeamfightTacticsLogic

A TeamfightTacticsLogic osztályon belül a reset egy lock-kal lett megoldva, de a lock után végig iterál megint az alliances tömbön és ott hívja meg a characterLevelingUp()-ot. Ez mind azért lett így megoldva, mert ha a try – catch – finally-n belül levő alliances ciklusnál alkalmaztam, akkor nem mindig állította vissza a karakterek életerejét és pozícióját helyesen, emiatt a kód szépségének a rovására ment.

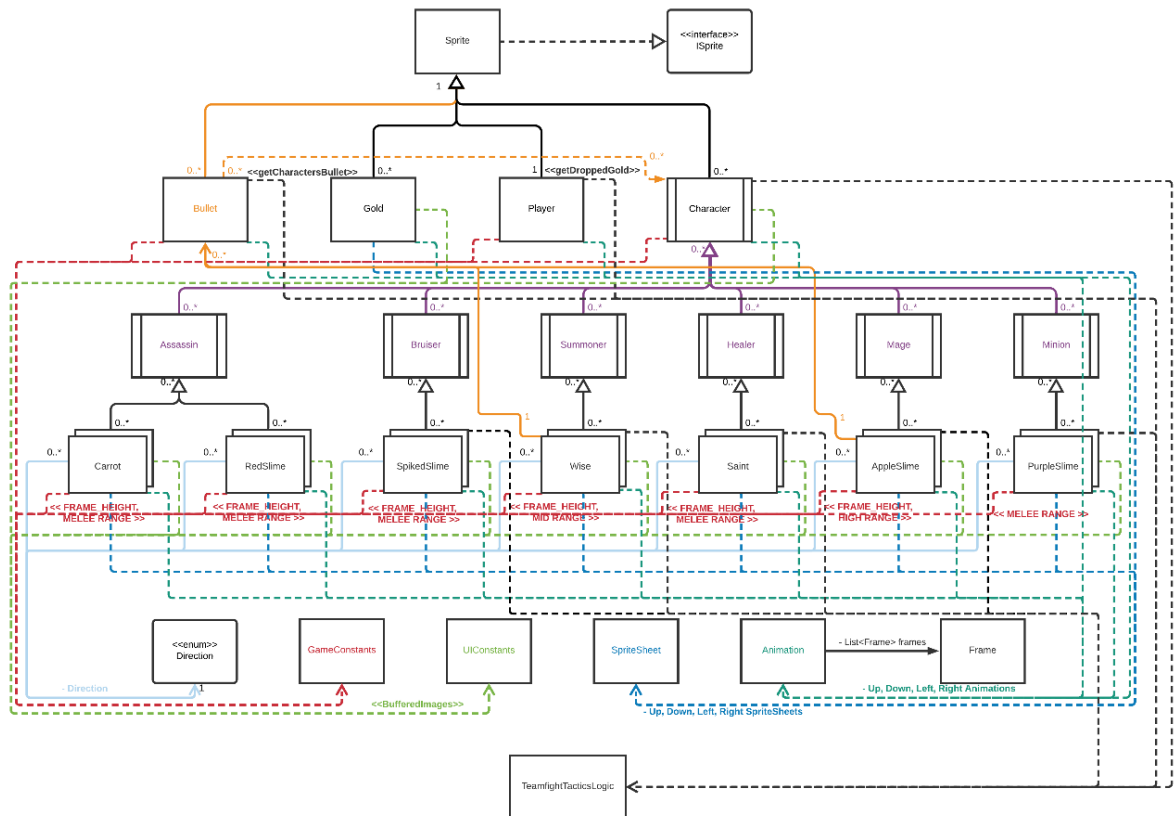
Osztályok közti kapcsolatok

Logikai és GUI kapcsolatok



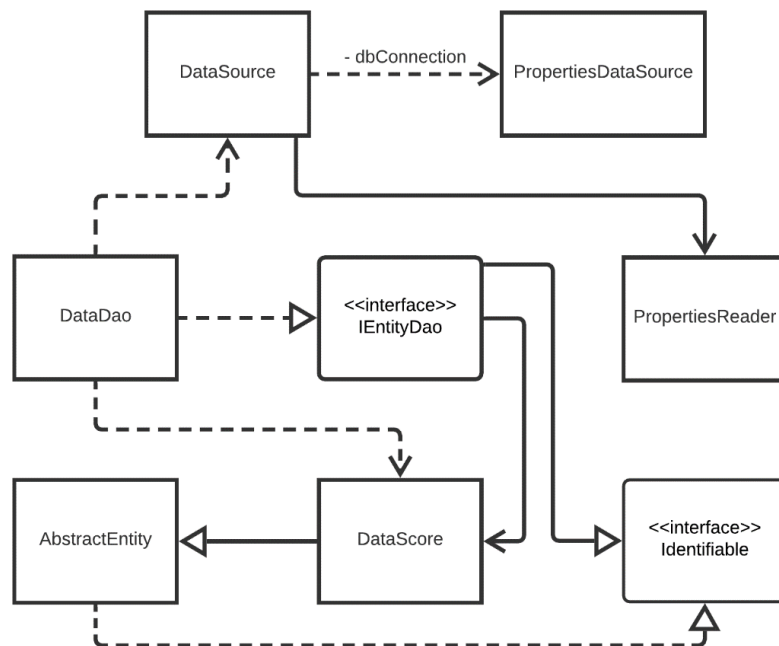
33. ábra - Logikai és GUI kapcsolatok

Karakterek kapcsolatai



34. ábra - Karakterek kapcsolatai

Adatbázis kapcsolatok



35. ábra - Adatbázis kapcsolatok

Osztályok UML diagramjai csomagonként

gui.sprite.characters package:

AppleSlime:

AppleSlime
<ul style="list-style-type: none"> • name: String • health: int • damageThresholdLower: int • damageThresholdUpper: int • range: int • cost: int • mana: int • <u>shootingSlimeMoveUpSpriteSheet: SpriteSheet</u> • shootingSlimeMoveUpAnimationImages: BufferedImage[] • shootingSlimeMoveUpAnimation: Animation • shootingSlimeMoveDownSpriteSheet: SpriteSheet • shootingSlimeMoveDownAnimationImages: BufferedImage[] • shootingSlimeMoveDownAnimation: Animation • shootingSlimeMoveLeftSpriteSheet: SpriteSheet • shootingSlimeMoveLeftAnimationImages: BufferedImage[] • shootingSlimeMoveLeftAnimation: Animation • shootingSlimeMoveRightSpriteSheet: SpriteSheet • shootingSlimeMoveRightAnimationImages: BufferedImage[] • shootingSlimeMoveRightAnimation: Animation • shootingSlimeShootFrontSpriteSheet: SpriteSheet • shootingSlimeShootFrontAnimationImages: BufferedImage[] • shootingSlimeShootFrontAnimation: Animation • shootingSlimeShootBackSpriteSheet: SpriteSheet • shootingSlimeShootBackAnimationImages: BufferedImage[] • shootingSlimeShootBackAnimation: Animation • shootingSlimeShootLeftSpriteSheet: SpriteSheet • shootingSlimeShootLeftAnimationImages: BufferedImage[] • shootingSlimeShootLeftAnimation: Animation • shootingSlimeShootRightSpriteSheet: SpriteSheet • shootingSlimeShootRightAnimationImages: BufferedImage[] • shootingSlimeShootRightAnimation: Animation • shootingSlimeSkillDownSpriteSheet: SpriteSheet • shootingSlimeSkillDownAnimationImages: BufferedImage[] • shootingSlimeSkillDownAnimation: Animation • shootingSlimeSkillUpSpriteSheet: SpriteSheet • shootingSlimeSkillUpAnimationImages: BufferedImage[] • shootingSlimeSkillUpAnimation: Animation • shootingSlimeSkillLeftSpriteSheet: SpriteSheet • shootingSlimeSkillLeftAnimationImages: BufferedImage[] • shootingSlimeSkillLeftAnimation: Animation • shootingSlimeSkillRightSpriteSheet: SpriteSheet • shootingSlimeSkillRightAnimationImages: BufferedImage[] • shootingSlimeSkillRightAnimation: Animation • shootingSlimeDeathSpriteSheet: SpriteSheet • shootingSlimeDeathAnimationImages: BufferedImage[] • shootingSlimeDeathAnimation: Animation • projectileImage: Animation • <u>image: Image</u>
<pre> + AppleSlime(x: int, y: int) # drawCharacter(g: Graphics, dir: Direction):void # shoot(target: Character, value: int):void # drawDeath(g: Graphics):void # drawSkill(g: Graphics):void # drawAttack(g: Graphics):void # mageSkill(target: Character):void </pre>

36. ábra - AppleSlime osztály

Carrot:

Carrot
<ul style="list-style-type: none"> • name: String • health: int • damageThresholdLower: int • damageThresholdUpper: int • range: int • cost: int • mana: int • <u>carrotMoveUpSpriteSheet: SpriteSheet</u> • carrotMoveUpAnimationImages: BufferedImage[] • carrotMoveUpAnimation: Animation • carrotMoveDownSpriteSheet: SpriteSheet • carrotMoveDownAnimationImages: BufferedImage[] • carrotMoveDownAnimation: Animation • carrotMoveLeftSpriteSheet: SpriteSheet • carrotMoveLeftAnimationImages: BufferedImage[] • carrotMoveLeftAnimation: Animation • carrotMoveRightSpriteSheet: SpriteSheet • carrotMoveRightAnimationImages: BufferedImage[] • carrotMoveRightAnimation: Animation • carrotAttackUpSpriteSheet: SpriteSheet • carrotAttackUpAnimationImages: BufferedImage[] • carrotAttackUpAnimation: Animation • carrotAttackDownSpriteSheet: SpriteSheet • carrotAttackDownAnimationImages: BufferedImage[] • carrotAttackDownAnimation: Animation • carrotAttackLeftSpriteSheet: SpriteSheet • carrotAttackLeftAnimationImages: BufferedImage[] • carrotAttackLeftAnimation: Animation • carrotAttackRightSpriteSheet: SpriteSheet • carrotAttackRightAnimationImages: BufferedImage[] • carrotAttackRightAnimation: Animation • carrotDeathSpriteSheet: SpriteSheet • carrotDeathAnimationImages: BufferedImage[] • carrotDeathAnimation: Animation • <u>image: Image</u>
<pre> + Carrot(x: int, y: int) # drawCharacter(g: Graphics, dir: Direction):void # drawDeath(g: Graphics):void # drawSkill(g: Graphics):void # drawAttack(g: Graphics):void # assassinsSkill(target: Character, value: int):void </pre>

37. ábra – Carrot osztály

PurpleSlime:

PurpleSlime
<ul style="list-style-type: none"> • name: String • health: int • damageThresholdLower: int • damageThresholdUpper: int • range: int • cost: int • mana: int • moveUpSpriteSheet: SpriteSheet • moveUpAnimationImages: BufferedImage[] • moveUpAnimation: Animation • <u>moveDownSpriteSheet: SpriteSheet</u> • moveDownAnimationImages: BufferedImage[] • moveDownAnimation: Animation • moveLeftSpriteSheet: SpriteSheet • moveLeftAnimationImages: BufferedImage[] • moveLeftAnimation: Animation • moveRightSpriteSheet: SpriteSheet • moveRightAnimationImages: BufferedImage[] • moveRightAnimation: Animation • attackUpSpriteSheet: SpriteSheet • attackUpAnimationImages: BufferedImage[] • attackUpAnimation: Animation • attackDownSpriteSheet: SpriteSheet • attackDownAnimationImages: BufferedImage[] • attackDownAnimation: Animation • attackLeftSpriteSheet: SpriteSheet • attackLeftAnimationImages: BufferedImage[] • attackLeftAnimation: Animation • attackRightSpriteSheet: SpriteSheet • attackRightAnimationImages: BufferedImage[] • attackRightAnimation: Animation • deathSpriteSheet: SpriteSheet • deathAnimationImages: BufferedImage[] • deathAnimation: AnimationprojectileImage: Animation • <u>image: Image</u>
+ PurpleSlime(x: int, y: int, string: String, level: int) # drawCharacter(g: Graphics, dir: Direction):void # drawDeath(g: Graphics):void # drawSkill(g: Graphics):void # drawAttack(g: Graphics):void # minionSkill():void

38. ábra - PurpleSlime osztály

Red Slime:

RedSlime
<ul style="list-style-type: none"> • name: String • health: int • damageThresholdLower: int • damageThresholdUpper: int • range: int • cost: int • mana: int • moveUpSpriteSheet: SpriteSheet • moveUpAnimationImages: BufferedImage[] • moveUpAnimation: Animation • <u>moveDownSpriteSheet: SpriteSheet</u> • moveDownAnimationImages: BufferedImage[] • moveDownAnimation: Animation • moveLeftSpriteSheet: SpriteSheet • moveLeftAnimationImages: BufferedImage[] • moveLeftAnimation: Animation • moveRightSpriteSheet: SpriteSheet • moveRightAnimationImages: BufferedImage[] • moveRightAnimation: Animation • attackUpSpriteSheet: SpriteSheet • attackUpAnimationImages: BufferedImage[] • attackUpAnimation: Animation • attackDownSpriteSheet: SpriteSheet • attackDownAnimationImages: BufferedImage[] • attackDownAnimation: Animation • attackLeftSpriteSheet: SpriteSheet • attackLeftAnimationImages: BufferedImage[] • attackLeftAnimation: Animation • attackRightSpriteSheet: SpriteSheet • attackRightAnimationImages: BufferedImage[] • attackRightAnimation: Animation • deathSpriteSheet: SpriteSheet • deathAnimationImages: BufferedImage[] • deathAnimation: AnimationprojectileImage: Animation • <u>image: Image</u>
+ RedSlime(x: int, y: int) # drawCharacter(g: Graphics, dir: Direction):void # drawDeath(g: Graphics):void # drawSkill(g: Graphics):void # drawAttack(g: Graphics):void # assassinsSkill(target: Character, value: int):void

39. ábra - RedSlime osztály

SpikedSlime:

SpikedSlime
<ul style="list-style-type: none"> • name: String • health: int • damageThresholdLower: int • damageThresholdUpper: int • range: int • cost: int • mana: int • moveUpSpriteSheet: SpriteSheet • moveUpAnimationImages: BufferedImage[] • moveUpAnimation: Animation • <u>moveDownSpriteSheet: SpriteSheet</u> • moveDownAnimationImages: BufferedImage[] • moveDownAnimation: Animation • moveLeftSpriteSheet: SpriteSheet • moveLeftAnimationImages: BufferedImage[] • moveLeftAnimation: Animation • moveRightSpriteSheet: SpriteSheet • moveRightAnimationImages: BufferedImage[] • moveRightAnimation: Animation • attackSpriteSheet: SpriteSheet • attackAnimationImages: BufferedImage[] • attackAnimation: Animation • spikedDeathSpriteSheet: SpriteSheet • spikedDeathAnimationImages: BufferedImage[] • spikedDeathAnimation: Animation • projectileImage: Image
<pre> + SpikedSlime(x: int, y: int) # drawCharacter(g: Graphics, dir: Direction):void # drawDeath(g: Graphics):void # drawSkill(g: Graphics):void # drawAttack(g: Graphics):void # bruiseSkill(me: Character):void </pre>

40. ábra - SpikedSlime osztály

Wise:

Wise
<ul style="list-style-type: none"> • name: String • health: int • damageThresholdLower: int • damageThresholdUpper: int • range: int • cost: int • mana: int • <u>ss: SpriteSheet</u> • moveDownImages: BufferedImage[] • moveUpImages: BufferedImage[] • moveLeftImages: BufferedImage[] • moveRightImages: BufferedImage[] • moveDownAnimation: Animation • moveUpAnimation: Animation • moveLeftAnimation: Animation • moveRightAnimation: Animation • projectileImage: Animation • <u>image: Image</u>
<pre> + Wise(x: int, y: int) + shoot(target: Character, value: int):void # drawCharacter(g: Graphics, dir: Direction):void # drawDeath(g: Graphics):void # drawSkill(g: Graphics):void # drawAttack(g: Graphics):void # summoningFriend():void </pre>

41. ábra - Wise osztály

Saint:

Saint
<ul style="list-style-type: none"> • name: String • health: int • damageThresholdLower: int • damageThresholdUpper: int • range: int • cost: int • mana: int • saintMoveUpSpriteSheet: SpriteSheet • saintMoveUpAnimationImages: BufferedImage[] • saintMoveUpAnimation: Animation • <u>saintMoveDownSpriteSheet: SpriteSheet</u> • saintMoveDownAnimationImages: BufferedImage[] • saintMoveDownAnimation: Animation • saintMoveLeftSpriteSheet: SpriteSheet • saintMoveLeftAnimationImages: BufferedImage[] • saintMoveLeftAnimation: Animation • saintMoveRightSpriteSheet: SpriteSheet • saintMoveRightAnimationImages: BufferedImage[] • saintMoveRightAnimation: Animation • saintAttackFrontSpriteSheet: SpriteSheet • saintAttackFrontAnimationImages: BufferedImage[] • saintAttackFrontAnimation: Animation • saintAttackBackSpriteSheet: SpriteSheet • saintAttackBackAnimationImages: BufferedImage[] • saintAttackBackAnimation: Animation • saintAttackLeftSpriteSheet: SpriteSheet • saintAttackLeftAnimationImages: BufferedImage[] • saintAttackLeftAnimation: Animation • saintAttackRightSpriteSheet: SpriteSheet • saintAttackRightAnimationImages: BufferedImage[] • saintAttackRightAnimation: Animation • saintSkillDownSpriteSheet: SpriteSheet • saintSkillDownAnimationImages: BufferedImage[] • saintSkillDownAnimation: Animation • saintSkillUpSpriteSheet: SpriteSheet • saintSkillUpAnimationImages: BufferedImage[] • saintSkillUpAnimation: Animation • saintSkillLeftSpriteSheet: SpriteSheet • saintSkillLeftAnimationImages: BufferedImage[] • saintSkillLeftAnimation: Animation • saintSkillRightSpriteSheet: SpriteSheet • saintSkillRightAnimationImages: BufferedImage[] • saintSkillRightAnimation: Animation • saintDeathUpSpriteSheet: SpriteSheet • saintDeathUpAnimationImages: BufferedImage[] • saintDeathUpAnimation: Animation • saintDeathDownSpriteSheet: SpriteSheet • saintDeathDownAnimationImages: BufferedImage[] • saintDeathDownAnimation: Animation • saintDeathLeftSpriteSheet: SpriteSheet • saintDeathLeftAnimationImages: BufferedImage[] • saintDeathLeftAnimation: Animation • saintDeathRightSpriteSheet: SpriteSheet • saintDeathRightAnimationImages: BufferedImage[] • saintDeathRightAnimation: Animation • <u>image: Image</u>
<pre> + Saint(x: int, y: int) # drawCharacter(g: Graphics, dir: Direction):void # drawDeath(g: Graphics):void # drawSkill(g: Graphics):void # drawAttack(g: Graphics):void # healSkill():void </pre>

42. ábra - Saint osztály

gui.sprite.other package

Bullet:

Bullet
<ul style="list-style-type: none">• damage: int• target: Character• animation: Animation
<ul style="list-style-type: none">+ Bullet(x: int, y: int, target: Character, damage: int, animation: Animation)+ update():void+ draw(g: Graphics):void• bulletHit():void

38. ábra - Bullet osztály

Gold:

Gold
<ul style="list-style-type: none">• animationSpriteSheet: SpriteSheet• animationImages: BufferedImage[]• animation: Animation
<ul style="list-style-type: none">+ Gold(x: int, y: int, image: Image)+ draw(g: Graphics):void

39. ábra - Gold osztály

gui.sprite.player package

Player:

Player
<ul style="list-style-type: none">• level: int• levelingLine: int• maxHealth: int• winStreak: int• loseStreak: int• moveLeftAnimation: Animation• moveRightAnimation: Animation• moveDownAnimation: Animation• moveUpAnimation: Animation• boringAnimation: Animation• health: int• gold: int• targetX: int• targetY: int• runTime: double• timer: Timer• startX: double• startY: double• startTime: long• bored: Boolean• start: long• darkBlue: color• levelingLineCap: int• playableCharacterNum: int• animation: Animation• pickUpRange: Ellipse2D.Double• moveStopRange: Ellipse2D.Double
<ul style="list-style-type: none">+ Player(sourceX: int, sourceY: int, image: Image)+ setLevelingLine(levelingLine: int):void+ drawPlayer(g: Graphics):void+ playerMove(targetX: int, targetY: int):void+ calculateMovement(targetX: int, targetY: int):void

40. ábra - Player osztály

gui.sprite package

ISprite

<<interface>> ISprite
+ isVisible():boolean + setVisible(visible: boolean):void + getImage():Image + getWidth():int + getHeight():int

41. ábra - ISprite osztály

Sprite

Sprite
x: int # y: int # image: Image # width: int # height: int • visible: boolean
Sprite(x: int, y: int, image: Image)

42. ábra - Sprite osztály

SpriteSheet

SpriteSheet
• image: BufferedImage
+ SpriteSheet(image: BufferedImage) + grabImage(col: int, row: int, width: int, height: int) : BufferedImage

43. ábra - SpriteSheet osztály

gui package

TeamfightTacticsFrame

TeamfightTacticsFrame
+ TeamfightTacticsFrame(teamfightTacticsLogic: TeamFightTacticsLogic) • setFrameProperties():void

44. ábra - TeamfightTacticsFrame osztály

UIConstants

UIConstants
<ul style="list-style-type: none"> + TITLE + SELL_CARD + REFRESH_CARD + LEVELING_CARD + CHARACTER_CARD + LOCK_CARD + EXIT_QUESTION + HELP_SENTENCES + OPTIONS + HELP + EXIT + DATA + CURRENT_DATA + KILLS + DEATH + WINS + LOSES + GOLDS + SPENT_GOLD + GAME_OVER + wiseMoveAnimation + wiseProjectileAnimation + PLAYER_SPRITESHEET + goldAnimation + shootingSlimeMoveUpAnimation + shootingSlimeMoveDownAnimation + shootingSlimeMoveLeftAnimation + shootingSlimeMoveRightAnimation + shootingSlimeAttackFrontAnimation + shootingSlimeAttackBackAnimation + shootingSlimeAttackLeftAnimation + shootingSlimeAttackRightAnimation + shootingSlimeDeathAnimation + shootingSlimeAttackProjectile + shootingSlimeSkillUpProjectile + shootingSlimeSkillDownProjectile + shootingSlimeSkillLeftProjectile + shootingSlimeSkillRightProjectile + carrotMoveUpAnimation + carrotMoveDownAnimation + carrotMoveLeftAnimation + carrotMoveRightAnimation + carrotAttackUpAnimation + carrotAttackDownAnimation + carrotAttackLeftAnimation + carrotAttackRightAnimation + carrotDeathAnimation + carrotSkillAnimation + spikedSlimeMoveUpAnimation + spikedSlimeMoveDownAnimation + spikedSlimeDeathAnimation + spikedSlimeAttackAnimation + spikedSlimeSkillAnimation + saintDeathUpAnimation + saintDeathDownAnimation + saintDeathLeftAnimation + saintDeathRightAnimation + saintSkillUpAnimation + saintSkillDownAnimation + saintSkillRightAnimation + saintSkillLeftAnimation + saintMoveUpAnimation + saintMoveLeftAnimation + saintMoveRightAnimation + saintMoveDownAnimation + saintAttackUpAnimation + saintAttackDownAnimation + saintAttackLeftAnimation + saintAttackRightAnimation + redMoveUpAnimation + redMoveDownAnimation + redMoveLeftAnimation + redMoveRightAnimation + redAttackUpAnimation + redAttackDownAnimation + redAttackLeftAnimation + redAttackRightAnimation + redDeathAnimation + purpleMoveUpAnimation + purpleMoveDownAnimation + purpleMoveLeftAnimation + purpleMoveRightAnimation + purpleAttackUpAnimation + purpleAttackDownAnimation + purpleAttackLeftAnimation + purpleAttackRightAnimation + purpleDeathAnimation + lockOn + lockOff + infoPanel + hoverAndTitleColor + frameColor + levelColor + standardFont + sellFont

45. ábra - UIConstants osztály

Menu

Menu
<ul style="list-style-type: none"> • halfOfTheFrame: int • startGameBtn: Rectangle • optionsBtn: Rectangle • helpBtn: Rectangle • exitBtn: Rectangle • yesBtn: Rectangle • noBtn: Rectangle • dataBtn: Rectangle • backBtn: Rectangle • onOffBtn: Ellipse2D.Double • resetButton: Ellipse2D.Double • titleFont: Font • fpsOn: boolean • dataReseted: boolean • menuFont: Font • menuAnimation: Animation • dataScore: DataScore • teamfightTacticsLogic: TeamfightTacticsLogic • menuHovered: String
<ul style="list-style-type: none"> + Menu(teamfightTacticsLogic: TeamfightTacticsLogic) + draw(g: Graphics):void + drawString(g: Graphics, text: String, x: int, y: int):void + drawCenteredString(g: Graphics, text: String, rect: Rectangle, font: Font):void

46. ábra - Menu osztály

TeamfightTacticsPanel

TeamfightTacticsPanel
<ul style="list-style-type: none"> • teamfightTacticsLogic: TeamfightTacticsLogic • menu: Menu • dragAndPull: boolean • hoverCharacter: Character • inTheShapes: boolean • wasRectangle: boolean • imagePrevPttfFailedX: int • imagePrevPttfFailedY: int • loadGame: LoadGame
<ul style="list-style-type: none"> + TeamfightTacticsPanel(teamfightTacticsLogic: TeamfightTacticsLogic, loadGame: LoadGame) + paintComponent(g: Graphics):void + loadAndStartGame():void + releasedShapes(s: Shape):void + pressedAlliance(me: MouseEvent, c: Character):void + releasedShapesEllipse2D(s: Shape):void + releasedShapesRectangle2D(s: Shape):void + swapCharacterSpots(s: Shape):void + swapCharacterLists(s: Shape, c: Character):void + drawWaiting(g: Graphics):void + drawFight(g: Graphics):void + drawGameOver(g: Graphics):void + drawInfoPanel(g: Graphics):void + drawCharacterCard(g: Graphics):void + drawRefreshCard(g: Graphics):void + drawLevelingCard(g: Graphics):void + drawTexture(g: Graphics):void

47. ábra - TeamfightTacticsPanel osztály

io package

PropertiesReader

PropertiesReader
<ul style="list-style-type: none">• PropertiesReader()+ <u>readProperties(path: String):Properties</u>

48. ábra - PropertiesReader osztály

model.animation package

Animation

Animation
<ul style="list-style-type: none">• frameCount: int• frameDelay: int• currentFrame: int• animationDirection: int• totalFrames: int• completed: volatile boolean• stopped: boolean• frames: List<Frame>
<ul style="list-style-type: none">+ Animation(frames: BufferedImage[], frameDelay: int)+ start():void+ stop():void+ restart():void+ reset():void+ update():void+ getSprite(): BufferedImage• addFrame(frame: BufferedImage, duration: int):void

49. ábra - Animation osztály

Frame

Frame
<ul style="list-style-type: none">• frame: BufferedImage• duration: int
<ul style="list-style-type: none">+ Frame(frame: BufferedImage, duration: int)

50. ábra - Frame osztály

model.classes package

Assassin

Assassin
<ul style="list-style-type: none">• <code>atEarlyGame</code>: boolean
<pre># Assassin(image: Image, x: int, y: int, health: int, mana: int, damageThresholdLower: int, damageThresholdUpper: int, range: int, cost: int, name: String) # assassinsSkill(target: Character, value: int):void # performAction(target: Character, value: int):void</pre>

56. ábra - Assassin osztály

Bruiser

Bruiser
<pre># Bruiser(image: Image, x: int, y: int, health: int, mana: int, damageThresholdLower: int, damageThresholdUpper: int, range: int, cost: int, name: String) # bruiserSkill(me: Character):void # performAction(target: Character, value: int):void</pre>

57. ábra - Bruiser osztály

Mage

Mage
<pre># Mage(image: Image, x: int, y: int, health: int, mana: int, damageThresholdLower: int, damageThresholdUpper: int, range: int, cost: int, name: String) # mageSkill(target: Character):void # shoot(target: Character, value: int):void # performAction(target: Character, value: int):void</pre>

58. ábra - Mage osztály

Healer

Healer
<pre># Healer(image: Image, x: int, y: int, health: int, mana: int, damageThresholdLower: int, damageThresholdUpper: int, range: int, cost: int, name: String) # healSkill():void # performAction(target: Character, value: int):void</pre>

59. ábra - Healer osztály

Summoner

Summoner
<pre># Summoner(image: Image, x: int, y: int, health: int, mana: int, damageThresholdLower: int, damageThresholdUpper: int, range: int, cost: int, name: String) # summoningFriend():void # shoot(target: Character, value: int):void # performAction(target: Character, value: int):void</pre>

60. ábra - Summoner osztály

Minion

Minion
<pre># Minion(image: Image, x: int, y: int, health: int, mana: int, damageThresholdLower: int, damageThresholdUpper: int, range: int, cost: int, name: String) # minionSkill():void # performAction(target: Character, value: int):void</pre>

61. ábra - Minion osztály

model.map package

LoadGame

LoadGame
<ul style="list-style-type: none"> • progressBar: JProgressBar • teamfightTacticsLogic: TeamfightTacticsLogic • creatingMap: boolean • loadingCharacters: boolean • loadingTextures: boolean • cardPanelShapeList: ArrayList<Shape> • characterWaitingShapeList: ArrayList<Shape> • characterPlayingShapeList: ArrayList<Shape> • enemyPlayingMap: HashMap<Integer, FieldState> • characterWaitingMap: HashMap<Integer, FieldState> • characterPlayingMap: HashMap<Integer, FieldState> • cardPanelMap: HashMap<Integer, String> • cardSellShape: Shape • cardLockShape: Shape • cardPlayerGoldPolygon: Polygon • backgroundImage: Image • cardPanelImage: Image • cardPanelSellImage: Image • playerMoveLeftAnimation: Animation • playerMoveRightAnimation: Animation • playerMoveDownAnimation: Animation • playerMoveUpAnimation: Animation • playerBoringAnimation: Animation • waterFountainAnimation: Animation • treeAnimation: Animation • flameAnimation: Animation • mushroomAnimation: Animation • dogAnimation: Animation • mushroomRopesAnimation: Animation • shrineAnimation: Animation • houseSmokeAnimation: Animation • fishingMushroom: Animation
<ul style="list-style-type: none"> + LoadGame(teamfightTacticsLogic: TeamfightTacticsLogic) + loadGame():void • loadingTextures():void • loadingCharacters():void • loadingSaint():void • loadingRedSlime():void • loadingPurpleSlime():void • loadingWise():void • loadingSpikedSlime():void • loadingCarrot():void • loadingShootingSlime():void • loadingPlayer():void • playerImages(spriteSheet: SpriteSheet): BufferedImage[] • startAnimations():void • createCardPanel():void • createCharacterWaitingPanel():void • createEnemyPlayingPanel():void • createCharacterPlayingPanel():void

62. ábra - LoadGame osztály

Key

Key
<ul style="list-style-type: none"> + key1: K1 + key2: K2
<ul style="list-style-type: none"> + Key(key1: K1, key2: K2) + equals(o: Object):boolean + hashCode():int + toString():String

63. ábra - Key osztály

model package

Character

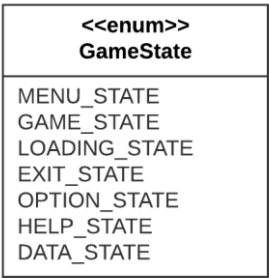
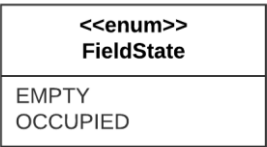
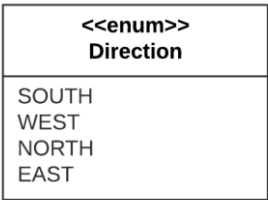


51. ábra - Character osztály

Direction

FieldState

GameState



52. ábra - FieldState enum osztály

65. ábra - Direction enum osztály

67. ábra - GameState enum osztály

GameConstants

GameConstants
<ul style="list-style-type: none"> + <u>FRAME_WIDTH</u> + <u>FRAME_HEIGHT</u> + <u>CARD_SIZE</u> + <u>MELEE_RANGE</u> + <u>MID_RANGE</u> + <u>HIGH_RANGE</u> + <u>CHARACTER_SIZE</u> + <u>FPS</u> + <u>PERIOD</u> + <u>LEVELING_CARD_VALUE</u> + <u>PLANNING_TIME</u> + <u>FIGHT_TIME</u> + <u>LEVELING_LINE_VALUE_PER_ROUND</u> + <u>LEVELING_LINE_VALUE</u> + <u>REFRESH_CARD_VALUE</u> + <u>CHARACTER_CARD_SIZE_W</u> + <u>CHARACTER_CARD_SIZE_H</u> + <u>CHARACTER_CARD_NAME_PLATES_SIZE</u>

68. ábra - GameConstants osztály

Stopwatch

Stopwatch
<ul style="list-style-type: none"> • start: long • maxTime: long
<ul style="list-style-type: none"> + Stopwatch() + elapsedTime():int + stop():void + resetTimer():void

69. ábra - Stopwatch osztály

TeamfightTacticsLogic

TeamfightTacticsLogic
<ul style="list-style-type: none"> + startGame: boolean + isOver : AtomicBoolean + isPlanningTime: AtomicBoolean • gameState: GameState • selectedCharacter: Character • swapSpotCharacter: Character • fightClock: Stopwatch • planningClock: Stopwatch • player: Player • menu: Menu • dataDao: DataDao • logic: Timer • myPanel: TeamfightTacticsPanel • loadGame: LoadGame • dataScore: DataScore • <u>rnd: Random</u> • sellCost: int • <u>alliances: ArrayList<Character></u> • <u>alliancesWaiting: ArrayList<Character></u> • <u>randomCardCharacters: ArrayList<Character></u> • <u>enemies: ArrayList<Character></u> • <u>fightingEnemies: ArrayList<Character></u> • <u>fightingAlliances: ArrayList<Character></u> • <u>droppedGold: ArrayList<Gold></u> • <u>charactersBullet: ArrayList<Bullet></u> • lock: ReentrantLock • <u>deaths: int</u> • spentGold: int • gameLevel: int • <u>kills: int</u> • cardsLock: boolean • fpsTimer: long • frames: int • framePerSecond: int • dbRunning: boolean
<ul style="list-style-type: none"> + TeamfightTacticsLogic() + creatingRandomCharacters(): void + creatingRandomCharacters(): void + characterLevelingUp(character: Character): void + startGame(): void • isProcessRunning(): boolean • creatingRandomNumbers: int • calculateCharactersBehaviour(): void • characterLevelingUpIterators(character: Character, itr2: Iterator<Character>, c: Character): void • reset: void • creatingEnemyByGameLevel(): void • imTooLazyToPlayThisGame(): void

53. ábra - TeamfightTacticsLogic osztály

persistance.connection package

DataSource

DataSource
<ul style="list-style-type: none"> • dbConnection: PropertiesDataSource
<ul style="list-style-type: none"> • DataSource() <ul style="list-style-type: none"> + getConnection(): Connection + getInstance(): DataSource

71. ábra - DataSource osztály

PropertiesDataSource

PropertiesDataSource
<ul style="list-style-type: none"> • url: String • user: String • password: String
<ul style="list-style-type: none"> + PropertiesDataSource(properties Properties) + getConnection(): Connection

72. ábra - PropertiesDataSource osztály

persistance.dao package

DataDao

DataDao
<ul style="list-style-type: none"> • <u>DATA_QUERY</u> • <u>UPDATE_KILLS_QUERY</u> • <u>UPDATE_DEATH_QUERY</u> • <u>UPDATE_WINS_QUERY</u> • <u>UPDATE_LOSES_QUERY</u> • <u>UPDATE_GOLDS_QUERY</u> • <u>UPDATE_SPENT_GOLDS_QUERY</u> • <u>RESET_DATA_QUERY</u> • <u>CREATE_NEW_TABLE_QUERY</u> • <u>ADD_FIRST_DATA_QUERY</u> • <u>ATTR_NAME_ID</u> • <u>ATTR_NAME_KILLS</u> • <u>ATTR_NAME_DEATH</u> • <u>ATTR_NAME_WINS</u> • <u>ATTR_NAME_LOSES</u> • <u>ATTR_NAME_GOLDS</u> • <u>ATTR_NAME_SPENT_GOLDS</u>
<ul style="list-style-type: none"> + getDatas(): DataScore + updateKills(kills: int): void + updateDeath(deaths: int): void + updateWins(): void + updateLoses(): void + updateGolds(gold: int): void + updateSpentGolds(spentGold: int): void + resetData(): void + createDatabase(): void + createTable(): void + addFirstData(): void

73. ábra - DataDao osztály

IEntityDao

<<interface>> IEntityDao
<ul style="list-style-type: none"> getDatas(): DataScore updateKills(kills: int): void updateDeath(deaths: int): void updateWins(): void updateLoses(): void updateGolds(gold: int): void updateSpentGolds(spentGold: int): void resetData(): void createTable(): void createDatabase(): void addFirstData(): void

74. ábra - IEntityDao interface osztály

persistance.entity package

AbstractEntity

AbstractEntity
<ul style="list-style-type: none"> • id: Long
<ul style="list-style-type: none"> + getId(): Long + setId(): Long

54. ábra - Identifiable osztály

Identifiable

<<interface>> Identifiable
<ul style="list-style-type: none"> getId(): T setId(id: T):void

55. ábra - AbstractEntity osztály

DataScore

DataScore
<ul style="list-style-type: none"> • kills: int • death: int • wins: int • loses: int • golds: int • spent_gold: int

56. ábra - DataScore osztály

resources package

ResourceLoader

ResourceLoader
<ul style="list-style-type: none">• ResourceLoader()+ readImage(path: String): BufferedImage

57. ábra - ResourceLoader osztály

Főbb osztályok és metódusai

LoadGame

A LoadGame osztály a menüben lévő Start Game gomb rákattintásával hívható meg, ez az osztály lényegében egy JProgressBar, amely addig tölt, míg az összes képet, feladatot el nem végezte, ha sikerült mindent betöltenie, akkor elindítja a játékot.

Főbb metódusai:

loadingCharacters(): Létrehozza a játékost, beállítja az összes animációját (fel, le, bal, jobb, unatkozás), valamint elindítja vele együtt az összes többi animációt, ami a pályán látható, például a fák mozgását, vagy pedig a kutya csaholását.

createCardPanel(): Létrehozza a kártyapanelt, amelyhez hozzá tartozik a refresh, tapasztalati pont vásárlás és lakat gomb, valamint a karakter eladás is.

createCharacterWaitingPanel(): Létrehozza a kispadot a karaktereknek.

createEnemyPlayingPanel(): Létrehozza az ellenfélnek a harcmezőt

createCharacterPlayingPanel(): Létrehozza a játékosnak a harcmezőt

TeamfightTacticsLogic

A TeamfightTacticsLogic legfőbb feladatai a játék indítása és újratekzdése. Ha lejár a játékosnak a tervezési ideje (eléri a nullát), akkor elindítja a játékot. Valamint, ha már elfogyott az egyik oldalról az összes harci egység, akkor reseteli a karaktereket.

Főbb metódusai:

- **TeamFightTacticsLogic():** A konstruktor, amely létrehozza az adatbázist, valamint felel a timerért. Létrehozz egy timert, mely a timerTaskot hívja meg. A timerTask felülírja a run metódust és elsősorban megnézi, hogy mi a játéknak az állása, ha GameState, akkor megnézi először, hogy a tervezési idő nem null és az eltelt idő egyenlő-e a nullával, ha igen, akkor megállítja a tervezési timert, újraindítja a harci időt és egy új szálon elindítja a startGame()-t. Ha nem teljesül az előző feltétel, akkor megnézi, hogy a tervezési boolean hamis-e és vége van e

a harcnak vagy a harci óra nem nulla és nincs vége a harcnak, valamint a harci óra eltelt ideje egyenlő-e a nullával, ha teljesül ezek közül valamelyik, akkor meghívja a `reset()` metódust, újraindítja a tervezési időt és megállítja a harcnak az idejét. Ezután egy `if`-fel megnézi, hogy a játékos élete, kisebb egyenlő, mint 0, ha igen, akkor a `timert` megállítja, ami futtatja az egész játékot (azaz vége a játéknak). Majd ezt követően megnézi, hogy már nem a tervezési időben van-e, ha nem, akkor végig megy a `fightingAlliance`, `fightingEnemies` és a `charactersBullet` tömbön és updateli az animációjukat. Ezek után végig megy az `alliances`, `alliancesWaiting` tömbön és szintén updateli az animációkat (hogyha nincs harc, akkor is van animáció a kispadon például). Ezek után megnézi, hogy üres-e a `droppedGold` tömb, ha nem, akkor végig megy egy `for` ciklussal a tömbökön és updateli az animációjukat. Ezek után updateli a maradék animációt, mint a háttérben történő dolgok (fák, horgászó gomba, egyéb), valamint a játékost. Egyébként, ha nem `GameState`, akkor lényegében a menü van meghívva és azt updateli, ha a menü nem null. Ezeken kívül pedig az `fps`-t számlálja, majd végül pedig újra rajzoltatja a képernyőt a `gui` szállal.

- **creatingRandomCharacters():** Ez a metódus szerepel a kártyapanel 5 karakteréért, első sorban egy `for` ciklussal kreál 5 véletlenszerű számot 0-tól 5-ig egy tömbbe, majd ezután egy másik `for` ciklussal végig megy a kapott tömbön és létrehozza azt az 5 karaktert.
- **calculateCharactersBehaviour():** Ez a metódus indítja el az új szálakat a karakterek viselkedésével. Hogyha lejár a tervezési idő `clear`eli a `fightingAlliancest` és a `fightingEnemies` tömböt, majd ezután hozzáadja az `alliances` tömb elemét a `fightingAlliances`hoz, valamint az `enemies` tömb összes elemét a `fightingEnemies` tömbhöz. Ezután ellenőrzi, hogy van-e a játékos kezében egy karakter, ha van és ez benne van a `fightingAlliances` tömbben, akkor eldobja a kezéből (hogy kikerülhessük azt a bugot, hogy nem egy szabad/foglalt mezőről indul el a karakter, hanem a felhasználó kezéből). Ezek után végig megy egy `for` ciklussal a `fightingEnemies` és a `fightingAlliances` tömbön külön-külön és hozzáadja a `threads` tömbhöz az új szálakat, miközben meg is hívja a karakterek viselkedését, valamint elindítja az ellenfél animációját is. Majd végül `forEach`-csel elindítja a szálakat és `join`oltatja őket.
- **characterLevelingUp(Character character):** A harci egységek szintjének növeléséért vállal szerepet. Az elején létrehoz két számlálót `count` és

countHigherTwo néven, a count az azonos szintű karaktereket míg a countHigherTwo pedig a kettes szintű azonos karaktereket számolja meg és egy countThree-t, ami a szintfejlesztésnél szükséges. Ezek után, hogyha a character az alliancesWaiting vagy az alliances tömbön belül van, akkor végig megy a tömbökön. Mindegyiken visszafelé megy a tömbből való törlések miatt. Az alliancesWaiting iteráción belül megnézi, hogy azonos-e a neve az adott indexűnek a characterrel, valamint a szint ugyanaz és a szint kisebb, mint három (hármasszintűt nem fejleszt négyesre), ha ez teljesül akkor counthoz hozzáad egyet, egyébként, ha a nevük azonos és az adott indexűnek a szintje kettes és a character szintje kisebb, mint hármasszintű, akkor a countHigherTwo-hoz ad hozzá egyet. Az alliances tömb ciklusa ugyanezzel az elvvel működik, annyi különbséggel, hogy a feltételekben ott van, hogy isPlanningTime, azaz tervezési időben van-e a játék. A két ciklus után egy feltétel következik, mégpedig, hogy a count nagyobb-e, mint 3, ha igen, akkor létrehoz egy wasAlliance boolean-t, ami azért kellett, hogy a végén az alliances, vagy az alliancesWaiting tömbhöz adja hozzá a karaktert fejlesztve. Ezután végig iterál a program az alliances tömbön úgy, hogy közben nézi, hogy a countThree nem egyenlő 0, amíg ez igaz, feltételben megint megnézi, hogy a planningTime igaz-e, azonos szintű a karakter, valamint azonos nevű, ha igen, akkor szabadabbá teszi a characterPlayingMap és characterWaitingMap-ot az adott karakterekkel, valamint beállítja a character eredeti pozícióját az iterált c pozíciójával. Ezek után a wasAlliance-t igazra állítja és a countThree-ből levon egyet. A következő ciklus végig megy az alliancesWaiting tömbön és ugyanezzel az elvvel oldja meg a feladatot, annyi különbséggel, hogy nem nézi, hogy planningTime igaz-e. Az iterációk után foglalttá teszi a character pozícióját a characterPlayingMap-on belül, ha wasAlliance igaz, akkor az alliancesWaiting-ből törölve a character az alliances-hez adja és beállítja az eredeti pozícióját, ha nem igaz, akkor a characterWaitingMap-ot foglalttá teszi és az alliancesWaiting-hez adja a character-t. Az, hogy az if-en kívül a characterPlayingMap-ot már átállította és ha nem igaz a wasAlliance és foglalttá teszi a characterWaitingMap-ot is, nem okoz problémát, mert nem lehetnek azonos kulcsúak, azaz csak az egyik valósul meg. Ezek után a karakter szintjét megnöveli és elindítja az animációját. Végül pedig egy utolsó feltétel, hogy ne legyen teljesen rekurzív a metódus, hogyha a character szintje kisebb, mint 3, és a countHigherTwo nagyobb egyenlő, mint

kettő, akkor létrehoz egy `characterUpToThree`-t és egy `for` ciklussal végig megy az `alliancesWaiting` tömbön a következő feltétellel, hogy ugyanaz a neve és a szintje egyenlő, mint 2, akkor a `characterUpToThree` egyenlő lesz az adott indexű karakterrel, ezután megállítja a ciklust egy `break`kel, és meghívja a `characterLevelingUp`-ot a `characterUpToThree`-vel. A `countHigherTwo` nagyobb egyenlő, mint kettő, azért kettő, mert, hogyha pont a harmadik karaktert fejleszteti kettőre, akkor arról még nem tud a gép, mert az elején számolta ki ennek az értékét.

- **reset():** A `reset` metódust hívja meg, hogyha vége van a harcnak (valamelyik fél győzött, kikapott) vagy pedig lejárt az idő. Ezt a metódust egy `lock`-kal kezdi, hogy mindenképpen végrehajtsódjon és ne szaladjanak el a szálak. Ha ez megtörtént, akkor `clear`eli a `charactersBullet` tömböt, hogy ne legyen a levegőben véletlenül sem lövedék. Majd leellenőrzi, hogy kikapott-e a játékos, vagy pedig lejárt-e az idő és nem nyert a felhasználó, ha ez a feltétel teljesül, akkor a játékos megsebződik az ellenfelének a méretének a kétszerese plusz a játékszintjével. Ezekután `clear`eli a `fightingAlliances` tömböt is, nehogy véletlenül valamilyen bugot kreálva megmaradjanak a pályán a karakterek megduplázódva győzelem esetén. Majd a tervezési időt és a játék végét igazra állítja (itt nem a végleges játék időre gondolunk, hanem a harc végére). Hogyha a játékos életerejé viszont eléri a nullát, akkor a tervezési időt hamisra állítja és majd ezáltal kiíródik a képernyőre a `Game Over` szöveg, tehát veszített a játékos. Azután `try finally`-be rakja a következő feltételeket, hogy a `lock`on belül teljesülhessenek a következő dolgok: az `alliances` tömbön belül végig menve egy `while` ciklussal visszaállítja a karakterek eredeti pozícióját, életerejét, kezdő manáját, `inRange` booleanját hamisra, `target`jét nullra, valamint az animációjukat `restart`olja. A következő `while` ciklussal végig megy az `alliancesWaiting` tömbön és meghívja minden egyes karakterre a `characterLevelingUp(character)` metódust, azért, mert ha a kispadon van például kettő egyes szintű harci egység a harcmezőn pedig egy egyes szintű, akkor fejlődjenek a harc után automatikusan. A harmadik `while` ciklussal pedig végig megy az `enemies` tömbön és ugyanazokat végrehajtja, mint az `alliances` ciklusnál. Ezután megnézi, hogy a `lakat` le van-e nyomva, ha nincsen, akkor meghívja a `creatingRandomCharacters()` metódust és létrehoz további 5 új karaktert a karakterpanelhez. Majd létrehoz egy `playerWin` boolean, ami kezdetben `false`-ra

állít és ellenőrzi, hogy a `fightingEnemies` tömb üres-e, ha igen, akkor a játékos győzött és ha a győzelmi sorozata kisebb, mint öt, akkor hozzáad egyet, cleareli az `enemies` tömböt (újak létrehozásánál nehogy többet kreáljon le a játék, mint kellene) majd a vereségi sorozatot nullára állítja, valamint a játék szintjét megnöveli. Ha kikapott a játékos akkor ugyanúgy megnézi, hogy a vereségi sorozat kisebb-e, mint öt, ha igen, akkor hozzáad egyet és a győzelmi sorozatot nullára állítja. Ezekután a `fightingEnemies` tömböt cleareli, valamint hozzáad 2 tapasztalati pontot a játékoshoz. Majd ellenőrzi a játékos pénzét, amit eloszt 10-zel, ha ez az eredmény nagyobb, mint 5, akkor ötre állítja (ez a maximum 5 arany 50+ arany esetén, ami játékosnál van), valamint megnézi, hogy győzött-e vagy veszített-e a játékos és aszerinti logikával odaadja a pénzt a játékosnak, valamint updateli az adatbázist a győzelem vagy vereség esetén. Ezután szintén updateli az adatbázisunkat, viszont most a legyőzött ellenfeleket, veszített szövetségesek számát és az elköltött pénzt adja meg, ezután a halálok, ölések és az elköltött aranyat nullára állítja, hogy majd, ha elindul a játék ez a számláló megint jól adjon hozzá az adatbázishoz. Ezek után pedig a lock-ot unlockolja és végül végig megy az `alliances` tömbön, és meghívja a `characterLevelingUp`-ot (Design Decision fejezetnél el lett magyarázva, hogy miért).

- **`creatingEnemyByGameLevel()`:** Ez a metódus azért szerepel, hogy minden kör elején létre legyenek hozva az ellenfél egységei. Elsősorban két ellenféllel találkozik a játékos, de ez változik a játék szintjétől (szintek szerint: hármas szinten három, ötös szinten négy, hetes szinten öt, tízes szinten hét ellenfél). Miután végzet az elágazásokkal a játék szintjével kapcsolatban, akkor egy while ciklussal végig megy az ellenfél méretén, míg kisebb a counter és aszerint véletlenül legenerálja az ellenfeleket, mindaddig, amíg nem teljesül a feltétel. Ezután végig megy az `enemies`-en egy for ciklussal és beállítja az ellenfél szintjét a játékszinttől függően tizenkettes szint fölött hármas szintűek lesznek, hatos szint fölött pedig kettes szintűek. Majd létrehoz egy countert, amely a mezőn szereplő ellenfelekért szerepel. Minden mezőt a harcmezőn üresre állít legelőször. Ezután végig megy egy for ciklussal a `map`-en, ha teljesül a cikluson belüli elágazás (a counter kisebb, mint az ellenfél mérete és a mező üres), akkor a mezőt foglalttá teszi és beállítja az ellenfelek kezdeti értékeit.
- **`imTooLazyToPlayThisGame()`:** Ezt az osztályt azért hoztam létre, név szerint is ironikusan jelzi, hogy ha a játékos túl lusta játszani vagy éppenséggel lassú, és

lejárt a tervezési ideje, de viszont nem rakott harci egységet a harcmezőre, akkor ez a metódus automatikusan megteszi helyette. Működése a következőképpen néz ki: lelockolja elsősorban a szálat, hogy mindenképpen végrehajtsanak a dolgok. A try-on belül pedig megnézi, hogy az alliances-nek a mérete kisebb-e, mint a játékos megengedett maximum karaktere (tehát, ha 3 karaktere lehet a játékosnak, de mindegyik a kispadon van, [ha megvette őket] akkor az az alliancesWaiting tömbben vannak, nem az alliancesben, emiatt 0 a mérete), ha ez igaz, akkor megnézi most azt, hogy az alliancesWaiting és az alliances üres-e, ha igen, akkor vége a körnek, ha nem és az alliancesWaiting nem üres (tehát vannak a padon karakterek), akkor létrehoz egy countert az alliances méretével. Ezután végig megy egy for ciklussal az alliancesWaiting tömbön és ha az alliances mérete kisebb, mint a játszható karaktereké, akkor hozzáadja a karaktert az alliances tömbhöz. Ezenfelül pedig ellenőrzi, hogy a játékos kezében van-e az adott karakter, mert ha igen vagy nem, mindkettőhöz másféle kulcs szerepel és ezáltal ahonnan elvette a felhasználó a karaktert a FieldState-jét üresre állítja. Majd, ha vége a for ciklusnak elindít egy következő for ciklust, ami végig megy a getCharacterPlayingMap-en, ebbe a ciklusban van egy feltétel, ami a következő: ha a számláló kisebb, mint az alliances tömb mérete és a kulcs értéke üres, akkor foglalttá teszi, az alliancesWaitingből kiveszi az adott karaktert és beállítja az x és y koordinátáját, valamint az eredeti pozícióját és végül hozzáad egyet a számlálóhoz. Ha ez mind végbe ment, végzet és feloldja a szálat.

- **startGame():** Mint a neve is mondja, ezzel a metódussal indítja el a játékot a program. A tervezési időt false-ra állítja, valamint a mérkőzés végét is, ezután megnézi, hogy az enemies tömb üres-e, ha igen, akkor meghívja a creatingEnemyByGameLevel() metódust, ezután meghív még két metódust, mégpedig az imTooLazyToPlayThisGame()-t és a calculateCharacterBehaviour()-t.

TeamfightTacticsPanel

A TeamfightTacticsPanel osztály szerepel a kirajzoltatásért, ő hívja meg minden további osztály kirajzoltatását is, kisebb logika benne a menü, de természetesen az is a kirajzoltatáshoz tartozik.

Főbb metódusai:

- **TeamfightTacticsPanel(final TeamfightTacticsLogic teamfightTacticsLogic, LoadGame loadGame):** A konstruktor abban vállal szerepet, hogy létrehozza a menüt és az egérre történő Listenereket, a DragListenert és a ClickListenert.
- **loadAndStartGame():** Ha a menüben rákattint a felhasználó a Start Game menüpontra, akkor ez a metódus fog meghívódni, ami betölti a játékhoz tartozó képeket, adatokat, ezután létrehozza a tervezési órát és a harc óráját, beállítja az adatait, valamint a harci órát leállítja kezdésképpen a loadGame() segítségével.
- **releasedShapes(Shape s):** Ez a metódus arra szolgál, hogyha a felhasználó kezében van egy karakter megnézi, hogy éppenséggel körre, vagy pedig négyzetre szeretné elhelyezni az adott karaktert. Ez úgy hajtodik végre, hogy először is megnézi, hogy a dragAndPull igaz-e, azaz folyamatosan tartja a karaktert és hogy a selectedCharacter nem egyenlő null, ha ez teljesül, megnézi, hogy az adott Shape az Ellipse2D vagy Rectangle2D, ha Ellipse2D akkor megnézi még feltételben azt is, hogy tervezési időben van-e, mert játék közben nem kéne a felhasználónak a harcmezőre raknia a karaktert. Ha teljesül valamelyik feltétel, akkor meghívja a hozzátartozó releasedShapes[Ellipse2D/Rectangle2D](s) metódust.
- **pressedAlliance(MouseEvent me, Character c):** Ez a metódus azért szerepel, hogy ellenőrizze, hogy jó helyre kattint-e a felhasználó vagy sem ahhoz, hogy felvehesse a karaktert. Ez úgy hajtodik végbe, hogy megnézi az egér koordinátáit és a harci egység mind a négy sarkát, ha belekattintott a karakter négyzetébe, akkor a selectedCharacter a c lesz és beállítja a hibázott koordinátát, ami azért van, hogyha rossz helyre rakná, akkor az eredeti pozíciójára legyen visszatérve (hibázott koordinátára), valamint elmenti az eredeti pozíciót is.
- **releasedShapesEllipse2D(Shape s):** Megnézi, hogy hol engedi el a selectedCharacter a felhasználó, több lehetőség is van, ha üres mezőre, ha foglalt mezőre, vagy pedig rossz helyre teszi le. Ezt négy elágazással sikerült megoldani. A releasedShapeEllipse2D(Shape s) és a

releasedShapeRectangle2D(Shape s) logikája közt egy különbség van, és az a következő lenne, hogy ha a harcmezőn lévő karakterek száma eléri a játékos limitjét, akkor ne lehessen a pályára húzni újabb karaktert, valamint cserélni a pályán lévő karakter pozícióját, ez az *első elágazás*. Ez úgy valósul meg, hogy először létrehoz két kulcsot, az egyik a Shape koordinátaival a másik pedig az „elhibázott” koordinátaival jön létre. Azután egy feltételben megnézi, hogy a characterPlayingMap FieldState üres-e és az alliances mérete egyenlő a játékos harci egység limitjével. Ezután megnézi, hogy körből vagy pedig négyzetből akarja rakni a felhasználó a karaktert, ha körből, azaz nem négyzetből akkor foglalttá teszi a Shape koordinátaival lévő characterPlayingMap-ot és üressé teszi a „hibázott” koordinátát. Ezután beállítja a selectedCharacter x és y-ját a kör közepéhez és az eredeti pozícióját is. Egyébként, ha nem kör, akkor csak annyi a dolga, hogy visszadobja a kispadra, ahonnét felemelte a felhasználó. A *második elágazás* megnézi, hogy a characterPlayingMap üres-e és a játékos limitje kisebb egyenlő, mint az alliances tömb mérete + 1, ha ez teljesül, akkor megnézi, ismételten, hogy körből vagy négyzetből szeretné-e elengedni a játékos. Ha négyzetből, akkor az alliances-hez hozzáadja a selectedCharacter-t, majd törli a selectedCharacter-t az alliancesWaiting tömbből, ezután a characterWaiting map-et üressé teszi, valamint a characterPlayingMap-et pedig foglalttá, egyébként, ha körből tesszük, akkor csak a characterPlayingMap-et változtatja üressé és foglalttá a két kulccsal, valamint beállítja a selectedCharacter eredeti pozícióját. Ezekután pedig beállítja az x és y koordinátáját a karakternek úgy, hogy középen legyen, valamint az eredeti pozícióját is. A *harmadik elágazás* megnézi, hogy a két kulcs ugyanaz-e, ha nem akkor helyet cserél egymással a selectedCharacter és az éppen kívánt másik karakter. Ez meghívja a swapCharacterSpots(s) metódust, ami külön le lesz írva. Végül pedig a *negyedik elágazás*, hogyha a két kulcs megegyezik, az lényegében az önmagába helyezés, azaz megfogta a karaktert és ugyanoda dobta le a felhasználó, emiatt a selectedCharacter pozícióját beállítja ugyanúgy a kör közepére.

- **swapCharacterSpots(Shape s):** Ez a metódus lényegében annyit csinál, hogy végig megy az alliancesWaiting és az alliances tömbön és meghívja a swapCharacterLists(s, tömb indexű karakter) metódust, ha az alliancesnél vizsgálja, akkor még feltételben ott van, hogy csak tervezési időben lehetséges

cserélni egymással pozíciót, egyébként nem lehet.

- **swapCharacterLists(Shape s, Character c):** A feladata, hogy végrehajtsa a cserét a két karakter közt, ez a következőképpen van megvalósítva: egy feltétellel kezd, ami megnézi, hogy a(z) s (Shape) tartalmazza-e a c (Character) koordinátáit, valamint a két karakter nem ugyanaz, ha ez teljesül, akkor a swapSpotCharacter lesz a c. Ezek után megnézi, hogy tervezési időben van-e a felhasználó, ha igen, akkor jön még egy feltétel, ami nem más, hogy ellenőrzi, hogy a swapSpotCharacter az alliancesWaiting tömbön belül van és a selectedCharacter az alliances tömbben (harcmezőről kispadra való csere). Ha igen, akkor az alliancesWaitingbe beleteszi a selectedCharacter-t és az alliances-be rakja a swapSpotCharacter-t, majd szépen törli mind a kettő karaktert a tömbből, ahol szerepeltek. Ha nem ez a feltétel teljesül, akkor megnézi, hogy a selectedCharacter az alliancesWaitingben van-e és a swapSpotCharacter az alliancesben (kispadról a harcmezőre való csere). Ha teljesül a feltétel ugyanúgy végrehajtja a cserét, mint az előbb (hozzáadás, törlés a logika alapján). A feltételek után beállítja a selectedCharacter és a swapSpotCharacter x és y pozícióját, valamint az eredeti pozíciójukat felülírja. Nincs harmadik lehetősége, azaz, ha harcmezőről harcmezőre cseréli a karaktereket, akkor nem kellene ezek a feltételek és elég csak a koordináták átírása. Egyébként, ha nincs tervezési idő, de cserélni szeretné a karaktereket, akkor csak egy lehetősége van a felhasználónak, kispadról kispadra való csere.
- **paintComponent(Graphics g):** A paintComponent(Graphics g) metódus felel minden kirajzolásért, ami látható a játékban. Három elágazása van, az *első* megnézi, hogy a GameState a Loading_Stateben van-e és az alapján rajzol, alakítja a font méretet. A *második elágazás*, hogyha a Game_State-ben van, azaz elindította a játékot a felhasználó. Ha elindult elsősorban beállítja a háttérképet majd meghívja a drawTexture(g) metódust, ezután egy elágazáshoz jutunk, ha dragAndPull igaz, azaz lenyomva van a bal egérgomb és a selectedCharacter nem null és tervezési időben van a felhasználó, akkor kézzel rajzolja ki a Shapeket, hogy hova teheti le a karaktert a pályán belül. Ezután, ha a selectedCharacter null, akkor annak megfelelően rajzolja ki a cardPanel képét, egy for ciklussal végig megy a cardPanelShapeListen és a cardPanelMap értékein végig menve meghívja a hozzátartozó metódusokat, Leveling_Card -> drawLevelingCard(g, s), Refresh_Card -> drawRefreshCard(g, s), CharacterCard

-> drawCharacterCard(g). Egyébként, hogyha van selectedCharacter, de ez úgy megnézve, hogy a selectedCharacter nincs benne a fightingAlliancesek közt, akkor annak megfelelően rajzolja ki a cardPanelt, ami most az eladó panel. Ezek után lényegében logikusan kirajzolja a többi dolgot is a pályára, a harci egységeket (drawFight(g)), a játékos, a dobott pénzt, a karakter infópanelt (drawInfoPanel(g)), hogyha rávisszük az „i”-re az egeret és végül pedig, hogyha a játékosnak elfogy az életeréje, akkor a drawGameOver(g) metódus kiírja a játékvégét. Egyébként, ha GameState nem a játékban van, akkor a menüt rajzolja ki. A végén pedig megnézi, hogy az fps mutatóját bekapcsolta-e a felhasználó.

Characters

A characters osztály felelős a harci egységek kirajzoltatásáért és logikájáért. Minden karakter egy külön szálon fut és saját Swing Timerrel rendelkezik a mozgáshoz.

Főbb metódusai:

- **behaviour(Character character):** A karakterek viselkedéséért vállal szerepet, amíg nincs vége a játéknak addig úgymond gondolkodik, hogy mi a következő lépése. A következőképpen működik: a legelején először megnézi, hogy milyen karaktereket, hogyan is sleepeltessen, ez részben az *Assassin* osztály miatt kellett létrehozni, mert az *Assassinok* a legelején úgymond beugranak az ellenfél hátvédvonalába. Ha a character *Assassin* akkor sleepel 1000 ms-t majd a karakter gyorsaságát 0.8-ra állítva az ugrás végrehajtásának gyorsasága miatt, ezek után az *Assassin*-nak az earlyGame booleanját true értékre állítja, ez azért kell, hogy az *Assassin* ne 100%-ban az ellenfél fejére ugorjon, hanem elé, ezután a calculateTarget("MAX") metódussal megkeresi a legtávolabbi ellenfelet és beállítja célpontnak, majd megint alszik 1000 ms-t és a speedet 0.1-re visszaállítva, majd pedig az earlyGame-t false-ra állítva kilép az elágazásból. Ha pedig a character *Minion*, akkor csak a calculateTarget("MIN") metódust hívja meg neki, mert ő nem létezik a csata előtt, emiatt nem kell sleepeltetni, ő a csata közben létrejött karakter. Egyébként pedig 2000 ms-t alszanak a karakterek és meghívja a calculateTarget("MIN") metódust. Ezután belép a számításokba, amit egy while ciklus segítségével lett létrehozva, melynek a kikötése, hogy a character ne legyen halott, valamint a csatának ne legyen vége, amíg ezek teljesülnek, addig fut a ciklus. Elsősorban mindig leteszteli, hogy a fightingAlliances vagy a fightingEnemies osztály üres-e, mert ha igen, akkor

megszakítjuk a ciklust. Egyébként pedig meghívja a `calculateTarget("MIN")` metódust, hogy a legközelebbi ellenfelet megtalálja. Ha az ellenfele már halott és van célpontja, akkor az `inRange` boolean hamisra állítja, azaz nincs a hatókörön belül ellenfele. Ha pedig a célpontja nem null és a `range` is igaz, akkor létrehozza a karakterünk sebzését a `min` és a `max` sebzése közt, majd meghívja a `performAction(target, value)` metódust. Egyébként, ha nem teljesül a feltétel egy `busy wait`-tel `sleep`-eltet rajta 250 ms-t (ha nem teszi meg, nem mozognak a karakterek, csak számolnak). Ha meghal a karakter vagy vége a harcnak, akkor kilép a `while` ciklusból és megnézi, hogy a `character` életereje kisebb egyenlő, mint nulla, ha igen, akkor meghívja a `handleHeroDeath(character)` metódust.

- **distance(double x1, double y1, double x2, double y2):** Kiszámolja a két pont közti távolságot.
- **characterMove(int targetX, int targetY):** A karakter mozgásáért felel, hogyha nincs vége a harcnak és nem halott a karakter, akkor megállítja a timert, meghívja a `calculateCharacterMovement(targetX, targetY)` metódust, ami kiszámolja a karakter következő lépését, `startTime`-nak megadja az aktuális rendszer idejét és elindítja a timert.
- **calculateCharacterMovement(int targetX, int targetY):** Kiszámolja a karakter célpontja és a karakter pozíciója alapján a távolságot, majd annak a futási idejét. Természetesen a legelején ellenőrzi, hogy a karakter támadástávolságán belül van-e az ellenfele, mert ha igen, akkor nem számoltatjuk fölősegesen, vagy ha épp meghalt a karakter már.
- **calculateTarget(String distance):** Ez a metódus alapján számolja ki a karakter, hogy hova mozogjon, kit vegyen célba. Elsősorban mindig updateli a karakter támadótávolságát majd meghívja az `updateDirection()` metódust, hogy jó fele nézzen az animációnk. Egy nagyobb elágazásból áll a mozgás, ha a célpont null vagy halott vagy nincs támadástávolságban, akkor egy hosszabb bonyolultabb algoritmust old meg, ha viszont nem igaz, akkor csak mozgatja a karaktert az adott célponthoz. Az algoritmus elsősorban létrehoz két koordinátát, mint a legközelebbi pont (`closestPointX`, `closestPointY`). Ezután ellenőrzi, hogy a karakter a `fightingAlliances` tömbön belül van-e és a `fightingEnemies` tömb nem üres, ha ez teljesül, a `closestPointX` és `closestPointY` értékül kapja a `fightingEnemies` 0. indexű karakter koordinátáit, valamint beállítja

alapcélpontnak is. Ezután kiszámolja a távolságot a két karakter közt (saját és célpontja közti távolságot), ezt `closestDist` néven valósítja meg. Ezekután egy `for` ciklussal végig megy a `fightingEnemies` tömbjén, és létrehozza az adott indexű enemyre a távolságot a karaktertől `dist` néven. Majd minimum vagy maximum keresést hajt végre, hogy megtalálja a célpontját a kért távolsággal (min vagy max). Ha megtalálta a célpontját, megnézi, hogy támadótávolságon belül van-e az adott ellenfél, ha igen akkor az `inRange` boolean `true`-ra állítja majd `return`-özik, egyébként az `inRange` `false`, majd a végén megnézi, hogy az adott karakter `Assassin` osztályú-e és az `earlyGame` igaz-e, ha igen, akkor hozzáadunk a `closestPointY` koordinátához 50-et, hogy az ellenfél szeme elé ugorjon, ne a fejére. Egyébként, ha a karakter `fightingEnemies` tömbben van és a `fightingAlliances` tömb nem üres, akkor ugyanezt a logikát követve kiszámolja az ellenfelének a távolságát. Végül pedig meghívja a `characterMove(closestPointX, closestPointY)` metódust, hogy elinduljon az ellenfele irányába.

- **updateDirection():** Az animációnak az irányát változtatja meg ez a metódus. Ha az animáció nem null és a célpontja nem null a karakternek, akkor elsősorban megállítja az animációt, majd megnézi, hogy melyik irányba kell, hogy mozogjon, ha $a(z) \mid target.x - x \mid$ nagyobb, mint $\mid target.y - y \mid$ (tehát távolabb van vízszintesen a célpont, mint függőlegesen), akkor ezalapján balra vagy jobbra kell mozognia, egyébként felfelé vagy lefelé veszi az irányt, végül pedig elindítja az animációt.
- **handleHeroDeath(Character character):** Ha az adott karakter életeréje kisebb egyenlő, mint nulla, akkor meghívódik ez a metódus és kitörli az adott tömbből a karaktert, valamint lejátssza a karakter halálának animációját, hogyha van. Végül pedig megnézi, hogy üres-e valamelyik harcos oldal, hogy véget vessen-e a játéknak. Ezt a következőképpen hajtja végre, elsősorban megnézi, hogy halott-e a karakter, ha igen, akkor altat 500 ms-t, remélhetőleg ennyi idő alatt a halál animáció kimegy, ezután megnézi, hogy melyik tömb tartalmazza a karaktert, ha a `fightingAlliances`, akkor logikusan belőle törli ki és az adatbázis számításához lévő `deaths` (szövetséges halála) számát megnöveli eggyel. Egyébként a `fightingEnemies` tömbhöz tartozik, létrehozza a `dropChancet` 0 és 100 közt, hogyha ez az érték 85-nél nagyobb egyenlő, valamint a karakter nem `Minion` leszármazott, akkor létrehoz egy aranyat a pályán a meghalt karakter

koordinátaival, végül az adatbázishoz a killshez (ellenfél megölése) hozzáad egyet és kitörli a karaktert a fightingEnemies tömbből. Végül pedig leellenőrzi, hogy a fightingEnemies vagy a fightingAlliances tömb üres-e, ha igen, alszik 1000 ms-t, hogy végrehajtsódjon minden maradék animáció és true-ra állítja az isOver boolean-t, hogy véget vessen a játéknak.

- **draw(Graphics g):** A karaktert rajzolja ki, mégpedig úgy, hogyha tervezési időben van, akkor mutatja minden karakter életerejét, de ha nem, akkor megnézi, hogy él-e az adott karakter, azaz az életereje nagyobb-e, mint 0, ha igen, megnézi, hogy a karakter fightingAlliances vagy fightingEnemies vagy alliances tömbön belül van-e, ha igen, akkor megnézi, hogy a(z) useAttack true, ha igen, akkor a drawAttack(g) módszerrel a támadási animációt rajzolja ki, egyébként kirajzolja a karaktert a drawCharacter(g, dir) segítségével (a támadási animáció és a mozgási animáció a karakter kirajzolása részben, emiatt kell az elágazás). Ezután megnézzük, hogy használ-e képességet, ha igen, kirajzoltatjuk a drawSkill(g) módszerrel. Ha fightingAlliances vagy alliances tartalmazza akkor az életerejük zöld, egyébként az ellenfél életerejének a színe piros. Ezután, ha nem volt igaz ez az elágazásunk, visszamenve, hogy melyik tömb tartalmazza a karaktert, azaz a waitingAlliances tartalmazza (ami a feltételben nem volt), kirajzoltatja a kispadon lévő karaktereket. Végül pedig, ha a karakterünk életereje kisebb, mint 0, akkor, ha az animációja nem ment végbe, kirajzolja a halál animációját a drawDeath(g) módszerrel.

Eseménykezelők

Az eseménykezelők minden egyes kattintásért felelnek, ami a menüben és a játékban egyaránt megtörténhetnek.

Főbb módszerei:

- **AbstractAction:**
 - moving:** Mint a nevében is benne van a mozgásban veszi ki a szerepét, a harci egységek és a játékos is egyaránt egy ilyen Action alapján mozog, viszont eltérnek egy nagyon picit, lényegében a mozgás ugyanaz, csak vannak benne külön feltételek, hogy miszerint álljanak is meg. A mozgás a következőképpen hajtsódik végre: létrehoz egy duration változót a jelenlegi System idejével és ebből kivonja a startTime-ot, kiszámolja a progresszt, mégpedig a duration / runTime alapján, ha ez a progress, nagyobb egyenlő, mint 1.0, akkor megállítja a

timert, egyébként pedig az x és az y koordinátát növeli az adott célponthoz a $kiinduló\ koordináta + (célpont\ koordináta - kiinduló\ koordináta) * progress$ alapján. A különbség a harci egység és a játékos közt, annyi, hogy a harci egységnél megnézzük, hogy vége van-e harcnak vagy rangeben van-e az ellenfél, ha igen, akkor megállítja a timert, valamint megnézi, hogy üres-e valamelyik ellenfél tömbje vagy halott-e a karakter és akkor is megáll. A játékosnál viszont létrehoz egy pickUpRange-t, hogy milyen távolságból tudja felvenni az aranyat automatikusan, végig megy a droppedGold tömbön és megnézi, hogy rangeben van-e, ha igen, akkor törli az adott aranyat és kap +2-t a játékos, majd ezután létrehoz egy moveStopRange-t, ami megállítja a karaktert az adott pontnál, hogyha tartalmazza a kattintott koordinátákat, az animációt leállítja végül pedig kiszámolja, hogy hol áll a karakter és annak megfelelően felfelé vagy lefelé fog nézni az animációja és elindítja az animációt és megállítja a timert.

- **DragListener:**

mouseMoved(MouseEvent me): A mouseMovednek két feladat van, az első az, hogy a menünél az egér pozíciójával az adott menüpontnál ki legyen világítva, maga a látvány és kicsit a segítségért, hogy merre is jár a felhasználó. A másik feladata pedig, hogy a játékban lévő kártyapanelnél a felhasználó megnézhesse a harci egység adatait, mint például az életerejét, támadóerejét és a többi.

mouseDragged(MouseEvent me): A karakter mozgathatásához szükséges metódus, hogyha a Game_Stateben van a játék és nyomva van a bal egérgomb, akkor tudja húzni a felhasználó az adott karaktert, természetesen, ha előtte rákattintott egy karakterre, azaz a selectedCharacter nem null. A végén a repaint függvény meghívásával pedig folyamatosan rajzolja a karaktert az egér körül.

- **ClickListener:**

mousePressed(MouseEvent me): A mousePressed a játék alatt veszi ki a szerepét, azaz, ha Game_State-ben vagyunk. Ha jobb egérgommbal kattint a felhasználó, akkor a játékos karakterét irányíthatja az adott kattintás irányába. Ha balegérgombot nyomja meg, akkor pedig megnézi, hogy Shapebe kattint vagy harci egységre. Ha egy Shapebe kattint, akkor végig megy az adott lehetséges Shapeken (cardPanelShapeList, kártyapanel és a cardLockShape, a lakat). Ha a Character_Card-ra kattint a felhasználó és van ott karakter, akkor megnézi, hogy megtudja-e venni, ha igen, akkor levonja az összeget érte a

program, valamint a kispadra helyezi, ha van üres hely. Ha nincs, akkor nem tudja megvenni a harci egységet. Itt az elköltött aranyat hozzáadja a spentGoldhoz, ami később meghívódva hozzáadódik az adatbázishoz. Leveling_Card és a Refresh_Card ugyanezzel a logikával van. Hogyha van pénze akkor használhatja, valamint a Refresh_Cardnál meg kell nézni, hogy a lakat le van-e nyomva vagy sem. Ezek után, hogyha a tervezési időben van a felhasználó, akkor megengedi, hogy az alliances tömb alapján felvehesse a harci egységet, majd egy for ciklussal megnézi, hogy melyik Shapebe kattintott (hol volt az adott karakter), hogy frissítve legyen a „hibázott” koordináta (imagePrevPtIfFailedX/Y), valamint a wasRectanglet false-ra állítja, mert az alliances tömb karakterei csak a harcmezőn lehetnek, azaz a körben. Ugyanezzel a logikával végig megy a characterWaiting karakterein és elmenti ugyanúgy a „hibázott” koordinátát, valamint a wasRectanglet true-ra állítja.

mouseClicked(MouseEvent me): A menüben van szerepe, meglehetett volna oldani, hogy a mousePressedhez legyen rakva, de jobbnak találtam azt, hogyha a felhasználó véletlenül rosszra kattintott, de letudja reagálni még időben és ha nyomva tartja az egeret majd elviszi a menüponttól a kurzort, akkor nem lesz megnyitva, ez a mousePressednél nem megoldható, kivéve, ha a mouseReleasehez és a mousePresshez írunk egy booleant, de az már a mouseClicked lenne. A szerepe az, hogy ahova kattint a felhasználó arra állítja a játékállását és újra rajzolja a repainttel.

mouseReleased(MouseEvent me): Csak a Game_Stateben hívódik meg és csak a balegérgombra. Ha ezek teljesülnek akkor a dragAndPull és az inTheShapes booleant hamisra állítja, majd végig megy a Shapeken, hogy melyiknél engedte el, ha Shapeben engedte el, akkor az inTheShapes true lesz. Ezután van egy feltételünk, hogyha nincs a Shapeben (not inTheShapes) és a selectedCharacter nem null és a selectedCharacter vagy nem fightingAlliances vagy alliancesWaiting tömbön belül van, akkor visszadobja a „hibázott” koordinátához a karaktert. Ezután megnézi, hogy a karakter a felhasználó kezében van-e és a kurzor a kártyapanel felett van-e (jelen esetben ez az eladó kártyapanel lesz) és az egérgomb elengedésénél így eladja a karaktert és értelemszerűen törli az adott tömbből, valamint üressé teszi a pozícióját. A legvégén nullra állítja a selectedCharacter-t és a swapSpotCharacter-t.

Javításra váró ismert hibák

Időnként néha lehet tapasztalni egy-két hibát a játékban. Sajnos a több szálú programozásban meg van a lehetőség, hogy néha összegabalyodnak a szálak és érdekes hibákat tudnak mutatni. Miután az idő egyre jobban fogyott természetesen vannak ismert hibák, amiket nem tudtam javítani a határidő miatt, de remélhetőleg nem rontják a játék élményt. Ezek a hibák a következők lennének:

- Néha az **animációk nem mennek végig** teljesen, például a karakter halálánál.
- Az **animáció** előfordulhat, hogy a **rossz irányban néz**, egy-két másodperc elteltével változik csak meg.

Több hiba lehetősége is fent állhat, amelyről nem tudhatok.

Buildelés és futtatás

A különböző buildelés és futtatás miatt a config és a ttf fájl kétszer szerepelnek, az IntelliJ IDEA a Teamfight Tactics mappából olvassa be az adatokat, míg a Command Line által a JavaC pedig a Teamfight Tactics/src mappából.

Rögtön indítható

A program rögtön indítható kicsomagolás után a **Teamfight Tactics.jar megnyitása** után. Azonban, ha ez a fájl nem lenne elérhető az IntelliJ IDEA-ban újra létrehozható a következőképpen:

- **Buildelés:** Build -> Build Artifacts -> Teamfight Tactics -> Build
- **Futtatás:** Teamfight Tactics.jar megnyitása

IntelliJ IDEA

A program alapvetően az **IntelliJ IDEA 2019.3.4**-es fejlesztői környezet alapján valósult meg, ezáltal az előbb említett **program által** nyugodtan megnyitható, buildelhető és futtatható is már a program.

- **Buildelés:** Build -> Build Project (Ctrl + F9)
- **Futtatás:** Run -> Run Boot (Shift + F10) [ha nincs ott a Run Boot, akkor a Boot.java osztályt megnyitva a kód 6. sorához odavezérelve az egeret balegérgomb esetén lenyíló menünél a Run 'Boot.main()' (Ctrl + Shift + F10) segítségével létrehozható]

Windows 10

Ha nem állna a rendelkezésünkre az IntelliJ IDEA program, akkor a Teamfight Tactics mappán belül az **src mappában** elindítva a **cmd⁴**-t, az alábbi **parancsokkal** **buildelhető és futtatható:**

- **Buildelés:** javac Boot.java
- **Futtatás:** java -classpath "../libs/mariadb-java-client-2.7.1.jar;" Boot.java

Linux

Linux esetén a Windows 10 cmd parancsok alapján véleményem szerint működnie kell, de ez **nincs tesztelve!**

- **Buildelés:** javac Boot.java
- **Futtatás:** java -classpath "../libs/mariadb-java-client-2.7.1.jar;" Boot.java

Tesztek

A tesztek futtatása az IntelliJ IDEA program által érhető el, mégpedig az osztályra ráírányítva az egerünket majd jobb gomb kattintás esetén a Run '[Class Name]Test' vagy pedig Ctrl + Shift + F10 gyorsgombbal lehetséges.

Tesztelés

Automatizált tesztek

Characters

- **assassinsSkill():** Két Carrot osztály/karakter alapján történő teszt, melynek a képessége a megadott érték kétszerese, az egyik carrot egyszeri megsebzésének történő esete
 - Túléli a sebzést
 - Nem éli túl a sebzést

Classes

- **performAction():** Két Carrot osztály/karakter alapján történő teszt, az egyik carrot egyszeri megsebzésének történő esete.
 - Túléli a sebzést

Character

- **setLevel():** Egy Saint osztály/karakter szintjének megváltoztatása, valamint az adatai ellenőrzése

⁴ Command Prompt

- Kettes szint esetén megfelelő szint, minimális támadóerő, maximális támadóerő, maximum életerő
- Négyes szint esetén hármas szintű-e, valamint az adatai is megfelelnek-e ennek a követelménynek
- **addHealth:** Egy Saint osztály/karakter életerejének hozzáadásának ellenőrzése
 - Nem töltődik túl az életeroje és annak megfelelő az élete a karakternek
 - Túl töltődik a karakter életeroje, mint a maximális életerő, de az élete maximális lesz, nem megy fölé
- **setMaxHealth:** Egy Saint osztály/karakter életerejének a maximumra való visszaállítása
 - Karakter megsebzése után a maximum életerő visszaállítása
- **removeHealth:** Egy Saint osztály/karakter megsebzése
 - Túléli a karakter a sebzés után
 - Nem éli túl a karakter a sebzést és meghal
- **calculateTarget():** Három Saint osztály/karakterrel való tesztelés a célpont helyes kiválasztásáért
 - Két karakter esetén legközelebbi célpont
 - Három karakter esetén legközelebbi célpont
 - Három karakter esetén legtávolabbi célpont

Manuális tesztek

Ide tartozik minden olyan teszt, amit logikailag nem nagyon, de vizuálisan lehet tesztelni, valamint az Automatizált teszteknel kimaradt logikai részek (például a performAction ugyanaz minden Classes-nál) és az összetettebb tesztek, melyeket manuálisan teszteltem.

Eseménykezelők:

- **DragListener**
 - **mouseMoved():** A menünél az egér irányításával a menü gombra ráirányított egér helyén a menü gomb háttere kiszíneződik, valamint a játékban lévő „i” (karakter infópanel) megjelenik az adott karakterre és helyes adatokkal
 - **mouseDragged():** Játék állásban bal egérgomb kattintása és annak nyomva tartása esetén a megfelelő karakter húzása

- **ClickListener:**

- **mousePressed():** A karakter irányítása, valamint a játékpanel gombjai kattintásának tesztelése a következőképpen történt:
 - A játékos jobb egérgomb esetén elindul az adott irányba
 - A megvásárolt karaktereket megfelelően megjeleníti és helyesen vonja le a pénzt
 - A tapasztalati pont vásárlás hozzáadja a vásárolt pontot a játékoshoz és levonja a pénzt
 - A kártyák frissítésének megtörténeése és pénz megfelelő értékének levonása
 - A lakat gomb lenyomása után nem lehet frissíteni a karaktereket, valamint új kör esetén nem frissülnek
 - A megfelelő karakterre kattintva jól működik a dragListenerrel együtt planningTime igaz és hamis állás esetén
- **mouseClicked():** A menüben van szerepe. Kattintás után, azaz bal egérgomb lenyomása után és annak elengedése megfelelően működik a menüben, ha rákattintunk egy menüpontra és elvisszük az egeret és nem a menüpont felett engedjük el, akkor nem nyitódik meg
- **mouseReleased():** Egérgomb elengedése esetén a karaktert elengedi és nem marad a játékos kezében, a megfelelő formát adja tovább ezáltal nem garantál errorrt, ha nem megengedett helyre tesszük a karaktert visszadobja a kezdőpozícióhoz, az eladás gombra ráhelyezve a karaktert majd azt elengedve eladja a karaktert és a megfelelő pénzárték hozzáadódik a játékoshoz

Harc folyamán történő tesztek:

- **Lövedékek:** Megfelelő lövedék rajzolódik ki a karakter esetén, valamint a lövedék becsapódásának, találatának hatására a megfelelő karakter sebződik meg, követi-e a célpontot elmozgás esetén és sebz-e a karaktert, a célpont és a karakter halála esetén a lövedékek törlésének működése
- **Sebzések:** A sebzések megfelelő célpont esetén megfelelően sebződik minden egyes karakter esetén
- **Képességek:** A megfelelő képesség SpriteSheetjével dolgozik-e, jól sebz-e a karaktereket, minden egyes karakterre külön-külön megnézve

- **Minion:** A minion megfelelően hozzáadódik-e a játék menetéhez és beszáll-e a harcba, mint akár célpont

Összetettebb tesztek:

- **reset():** Játékos életerejének változtatása vereség esetén, a karakterek életerejé és pozíciójának visszaállítása
 - Játékos életerejének változtatása
 - Egy karakter megsebzése, melyet túlél, majd életerejének visszaállítása
 - Négy karakter megsebzése, melyet túlél, majd életerejének visszaállítása
 - Két karakter megsebzése, melyet nem élnek túl, majd életerejének visszaállítása, valamint életben vannak-e
 - Egy karakter pozíciójának visszaállítása a kezdetihez
 - Négy karakter pozíciójának visszaállítása a kezdetihez
- **Karakter pozíció változtatás:**
 - A karakter kispadról kispadra helyesen működik-e, szabaddá válik az előző pozíció
 - A karakter harcmezőről harcmezőre helyesen működik-e, szabaddá válik az előző pozíció, valamint hozzáadja-e a harcmezőn lévő kapacitáshoz a karaktert, ha volt kapacitás, ha nem volt természetesen nem engedi a helycserét
 - Két karakter cseréje sikeresen végbe megy-e a harcmezőn
 - Két karakter cseréje sikeresen végbe megy-e a kispadon
 - Két karakter cseréje sikeresen végbe megy-e, ha az egyik a harcmezőn a másik a kispadon helyezkedik
 - Karaktert kézben tartva a kör kezdése előtt, amely automatikusan a pályára helyeződik, mert van rá kapacitása a játékosnak és kiveszi a játékos kezéből
 - Karaktert kézben tartva a kör kezdése előtt, de nem teszi automatikusan a pályára kézben marad-e
- **Adatbázis:**
 - **Adat törlés:** Az Options menüponton belül a Reset Database kör rákattintása esetén megfelelően törölődnek-e az adatok, és 0-ról indítja-e el az adatokat

- **Adat hozzáadása:** A kör vége esetén megfelelően hozzáadódnak-e az adatok az adatbázishoz
- **Stopwatch:** Elindítása és megállítása helyesen működik-e, jól cserélik-e egymást a két megadott Stopwatch
- **LoadGame:** Betölti-e helyesen az összes adatot (képet, betűtípust), létrehozza-e a pályát megfelelően

További tesztek:

- **Megfelelő kép:** Minden karakter esetén végig nézni, hogy a megfelelő SpriteSheetekkel dolgozik, jó a megjelenített Sprite a karakterpanelnél, valamint a SpriteSheetek esetén az animációk jól rajzolódnak ki
- **Swing Timer:** Swing Timer esetén megfelelően megállnak-e a karakterek, ha a célpontot elérik, vagy vége lett a harcnak, a játékos karaktere felveszi-e az aranyat a Timer alapján kirajzolt köre körül
- **Betűtípusok:** Megfelelő betűtípusok jelennek-e meg az adott helyen, menüben, karakterpanel és karakterek esetén

Összegzés

Véleményem szerint sikerült létrehoznom azt a programot, amit elképzeltem és szerettem volna, de természetesen rengeteg továbbfejlesztési potenciál van még benne, valamint javítási, fixálási lehetőség.

Egyetlen hátrányom volt, az pedig a határidő. ha több idő állt volna a rendelkezésemre remélhetőleg még jobb és szebb programot tudtam volna megcsinálni (kódolásiilag, látványilag, játékelményileg).

A játék több szálon futása megvalósult a terveim szerint, valamint jó érzés volt egy ekkora játékot létrehozni, rengeteg mindent tanultam ez idő alatt. Remélem élvezhető kis játékot sikerült alkotnom.

Továbbfejlesztési lehetőségek

A programban még rengeteg potenciál van érzéseim szerint, több ötletem is van, hogy hova lehetne fejleszteni. Az egyik, hogy bevezetném az **Item**-eket, amely erősíthetné az adott karaktert, lenne egy maximális **Inventory**-ja mindegyik egységnek, méghozzá kettő, valamint ezek az Item-ek összeolvadhatnának egy jobb változatért, amely csak egy helyet foglalna, ezáltal úgymond egy karakternek négy Item lenne adható. A másik egyértelműen az **okosabb AI** lenne vagy pedig az **Online** játék megvalósítása. Az okosabb AI esetén létrehoznám neki is a Kispadot, hogy kedvére vásárolgasson és szintezze a karaktereit, ügyeljen a pénzére és taktikázzon, akár beállítva három alap ötletet, ebből egy például, hogy mohón játszana, hogyha gyűjt egy karakterre és látja a megvásárolhatóak közt, mindenképpen megveszi, valamint felállási stratégiával is rendelkezhetne a gép, netán süni alakzat, amely védi a középben lévő távolsági egységet. Az Online működés esetén ugyanúgy lennének AI-ok, de nem az okos kategóriában, hanem előre megírt pozícióban, amelyek természetesen kellenének az Item-ek dobása miatt, körülbelül **8 játékost** tudnék elképzelni, de netán lehetne egy opció egy gyors játékra, amely **4 játékos** lenne. További ötlet még, hogy a **karakterek osztályának fejlesztése**. Például, hogy tűz elem, víz elemű a karakter és ezeknek ugyanúgy szintezve egytől háromig erősítene minden pályán lévő karaktert, például három tűz karakter esetén egyes szintű tűz tulajdonság lenne, amely több támadóerőt adna, a következő szint öt karakter esetén, a végső szint pedig hét karakternél lehetne ezt elérni. Természetesen még ide tartoznak olyan apró ötletek is, hogyha nem Online a **játék**, akkor annak **megállításának lehetősége**, valamint **új játék indításának** való

lehetősége. A **várható arany kiíratása** győzelem és vereség esetén a birtokolt aranyból, amelyet minden kör végén kaphatnánk. **Több karakter bevezetése** is egy fontos fejlesztés lenne. A **harci egységek** a jelenlegi játékban **egymásba tudnak menni**, ennek **javítása** is egy továbbfejlesztési lehetőség (meg lehetne oldani, hogy a pálya 2x3x7-es méretű mezejére csak egyetlen egy karakter léphessen, mint ahogy csak egy karakter helyezhető egy mezőre a kör elején). A **játékos ne mehessen át a háttérben lévő animációkon**, mint például fa, ház és a játékpanel.

Felhasznált tartalmak

A grafikához felhasznált képek szerzője:

- <https://moose-stache.itch.io/rpg-asset-pack>

Eredeti szerző: Moose Stache

A menü betűtípusának szerzője:

- <https://www.dafont.com/humongous-of-eternity-st.font>

Eredeti szerző: Southype

Karakterek mozgásának logikájának a szerzője:

- <https://stackoverflow.com/questions/31041991/moving-a-square-from-a-starting-point-to-the-position-of-a-mouse-click-at-a-fixed-speed>

Eredeti szerző: MadProgrammer

Az animáció logikájának és a Wise karakter képének a szerzője:

- <https://gamedev.stackexchange.com/questions/53705/how-can-i-make-a-sprite-sheet-based-animation-system>

Eredeti szerző: Savlon

Adatbázis felépítése és algoritmusai a Programozási Technológia óráról származik

Eredeti szerző: Nagy Krisztián, Valdar

Adatbázis telepítésének ismertetésének szerzője:

<https://valdar.web.elte.hu/progtech/#!/tools/mariadb>

Eredeti szerző: Nagy Krisztián, Valdar