Ugró Lovag – Dokumentáció

Czumbel Péter – ODZF0M

1. Specifikáció

1.1 A játék leírása

Az *Ugró Lovag* egy egyjátékos, kétdimenziós, ügyességi platformjáték, melyben a játékos feladata az, hogy a platformokon ugrálva feljusson a pálya tetejére a királylányhoz. A cél teljesítéséhez a játékosnak meg kell tanulnia az ugrások technikáját, mivel a platformok elrendezése olyan, hogy egy ugrást elvétve is sokkal lejjebbre eshet vissza a lovag, mint ahonnan indult.

A lovagot a jobbra és balra nyíl, valamint a szóköz billentyűkkel irányíthatjuk. A nyilak segítségével sétálhatunk a lovaggal a nyílnak megfelelő irányba, a szóköz billentyűvel pedig ugorhatunk. Miután lenyomtuk a szóközt, a nyilak lenyomásával adhatjuk meg az ugrás irányát. Az ugrások erejét és szögét az határozza meg, hogy mennyi ideig tartjuk lenyomva a szóközt: minél tovább nyomjuk, annál nagyobb erővel és nagyobb szögben fog ugrani a lovag. Ha elég sokáig tartjuk lenyomba a gombot, az ugrás erőssége eléri a maximumot, és a lovag a gomb felengedése nélkül végrehajtja az ugrást.

Ha a lovag az ugrása közben oldalról vagy alulról egy falnak ütközik, akkor arról teljesen rugalmasan pattan vissza, viszont, ha fentről esik vissza egy platformra, akkor ez teljesen rugalmatlan ütközést eredményez. Ha a lovag egy ugrásával kilépne a képernyőről akkor, ha ez oldalra történik, a képernyő másik oldalán fog visszatérni. Ha a lovag felfele vagy lefele lép ki a képernyőről, akkor a pálya következő – vagy előző – része fog megjelenni. A kamera tehát nem követi a lovagot, nem a platformok mozognak, csak a lovag.

1.2 A program felépítése

A program elindítása után a főmenübe kerülünk. A menü pontjai között a menü gombjainak segítségével válthatunk. Itt lehetőségünk van új játékot kezdeni, betölteni egy korábban elmentett játékállapotot, ki vagy bekapcsolni a platformok körvonalának megjelenítését vagy kilépni a programból.

Miután a főmenüből elindítottuk a játékot, az escape billentyű megnyomásával hozhatunk elő egy másik menüt. Itt a különböző menüpontok kiválasztásával folytathatjuk a játékot, ki vagy bekapcsolhatjuk a platformok körvonalának megjelenítését, elmenthetjük az aktuális játékállapotot, vagy feladhatjuk a játékot, ami kilép a főmenübe és törli a mentésünket.

1.3 Megvalósítás - Terv

A menük swing menükkel lesznek megvalósítva, míg a játék komponensei a graphics osztály segítségével lesznek majd kirajzolva. Mivel a pályának egyszerre mindig csak egy képernyőnyi része fog

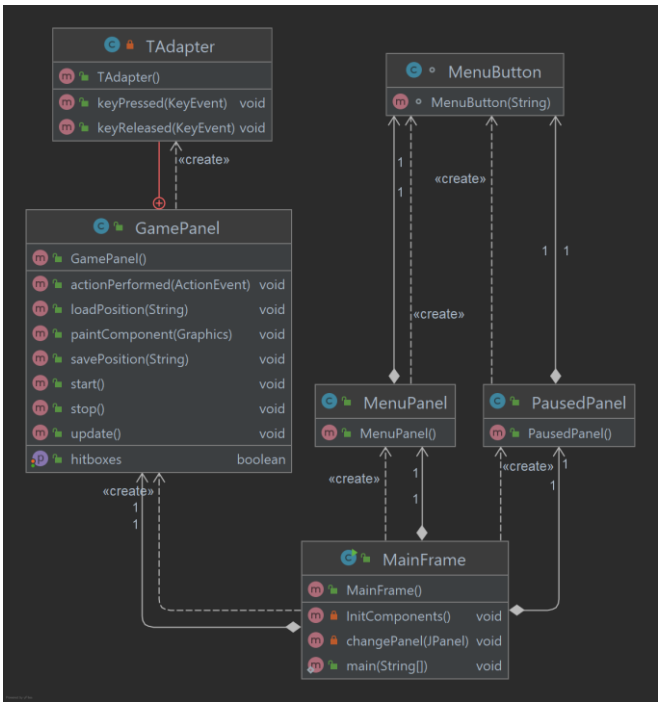
látszani a játékban, ezeket a részeket érdemes egységekként kezelni. Minden ilyen egységhez tartozni fog egy képfájl, ami ábrázolja az adott hátteret és platformokat, valamint egy JSON fájl, ami a platformok koordinátáit tartalmazza. Ezt a JSON fájlt beolvasva a program létre fog hozni minden platformkoordinátához egy objektumot, amikből pedig egy kollekciót készít. Ez a struktúra megkönnyíti majd a lovag ütközéseinek detektálását a platformokkal.

A mentések létrehozásakor elég elmenteni azt, hogy a lovag éppen hányadik egységben volt, a helyzetét a képernyőn és ha éppen egy ugrás közben léptünk ki, a sebességét. Ezek az adatok is egy JSON fájlban kerülnek majd elmentésre.

2. Megvalósítás

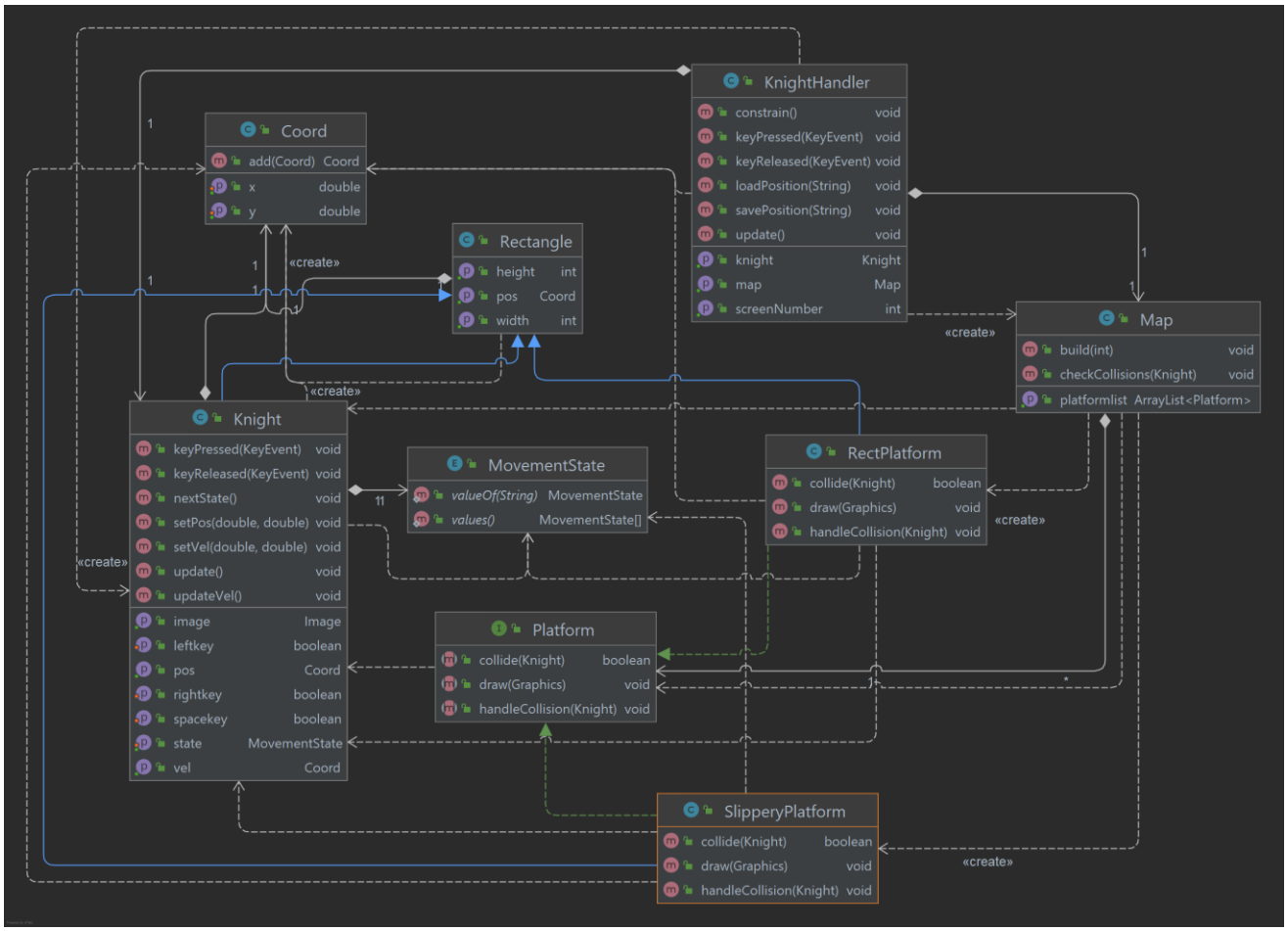
2.1 Osztályok

A program osztályai két csomagban oszlanak szét, a Visual és a Logic package-ekben. A Visual package tartalmazza a menü és a játék megjelenítésével kapcsolatos osztályokat, azaz a UI osztályait. A Logic csomagban a játék működéséért felelős osztályok vannak. Az osztálydiagramokat az IntelliJ generálta.



A balra látható ábrán a Visual package osztálydiagramja látható. A MainFrame osztály a JFrame osztály leszármazottja, ez az osztály felelős az ablak megjelenítéséért. Három JPanel-t tartalmaz, amik közül a program állapotának megfelelően mindig csak egyet jelenít meg egyszerre.
Az első ilyen panel a MenuPanel. A program elindításakor a MainFrame ezt jeleníti meg, ezen a panelen látható egy kép a játék címével, valamint a specifikációban leírt négy gomb.
Ha a játék során megnyomjuk az escape gombot, akkor egy, a főmenühöz kinézetben nagyon hasonló menü fog megjelenni. Az ezt megjelenítő panel a PausedPanel, ami a MenuPanel-től csak a gombjainak funkcióiban tér el, a specifikációban leírtaknak megfelelően. A gomboknak létrehoztam egy saját osztályt, a MenuButton-t, ami a JButton osztályból származik le. Ennek az osztálynak a segítségével minden gombnak egyforma stílust adhattam.

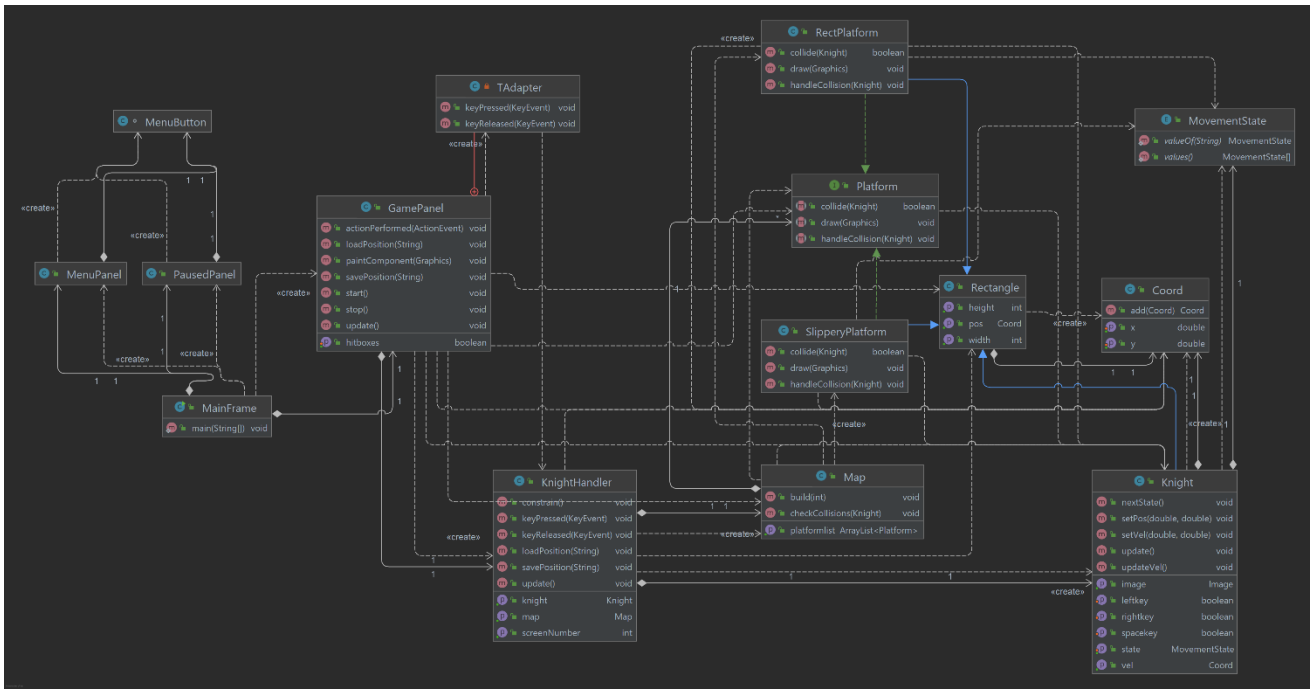
A GamePanel osztály felelős a játék kirajzolásáért és a gombnyomások detektálásáért. Az osztály a JPanel leszármazottja, de implementálja az ActionListener interfészt is. A MainFrame minden swing event után küld egy eventet a GamePanel-nek is, így megoldható a játék folyamatos frissítése. A GamePanel tartalmaz egy timer-t 10 ms késleltetéssel, így a frissítési rátája 100 fps-nek felel meg. A gombnyomások detektálására tartalmaz még egy belső osztályt, ami a KeyAdapter leszármazottja.



A fenti diagram a Logic package osztályai közötti kapcsolatokat ábrázolja. A KnightHandler osztály felelős a lovag és a pálya közti kommunikáció megvalósítására, ezért KnightHandler tartalmaz egy Knight és egy Map tagváltozót. A KnightHandler a Map-en keresztül tudja, hogy a lovag ütközött e platformmal, és annak megfelelően állítja be a lovag új pozícióját és sebességét. Ezen kívül a KnightHandler kezeli a gombnyomások eseményeit is.

A pálya platformjait a Map osztályba lehet betölteni, ami ellenőrizni tudja, hogy a lovag ütközött-e valamelyik platformmal. A Map a platformoknak egy heterogén kollekcióját tárolja, lehetnek benne normál és csúszós platformok is. A Knight osztály felelős a lovag helyzetének, sebességének és állapotának a tárolásáért és frissítéséért. A Coord osztály egy kétdimenziós vektornak(tuple) felel meg, ilyen osztályokban tárolja a program a sebesség és hely vektorokat.

A két package együttes osztálydiagramja a lenti ábrán látható, a sok függőség miatt viszont ez szinte követhetetlen:



2.2 Az osztályok metódusai

2.2.1 A MainFrame metódusai:

private void initComponents():
Inicializálja a frame-et, azaz hozzáadja a szükséges paneleket, és létrehozza a panelek gombjaihoz az ActionListenereket
private void changePanel(JPanel panel):
Kicseréli a jelenleg megjelenített panelt a paraméterként kapott panelre

2.2.2 A MenuPanel, PausedPanel és MenuButton metódusai:

Csak konstruktor.

2.2.3 A GamePanel metódusai:

public void loadPosition(String s) és ***public void savePosition(String s)*:**
Továbbítja a függvényhívást, a KnightHandler fogja betölteni vagy elmenteni a paraméterként kapott nevű fájlba a játék állását.
***public void actionPerformed(ActionEvent e)*:**
A timer minden tickjére meghívódik, továbbadja a hívást az update függvénynek.
***public void update()*:**
Meghívja a KnightHandler update függvényét, ha változott a háttérkép sorszáma, betölti az új képet, majd frissíti a panel tartalmát hogy megjelenjenek a rá rajzolt elemek.
***public void paintComponent(Graphics g)*:**
Kirajzolja a hátteret, a lovagot és a platformokat.

public void stop() és **public *void start()*:**
Elindítja vagy megállítja a GamePanel timer-ét, hogy az ne terhelje feleslegesen a processzort frissítésekkel amikor nem szükséges.

2.2.4 A KnightHandler metódusai:

***public void constrain()*:**
Ha a lovag kilépne a képernyőről, visszahelyezi a megfelelő helyre, és beállítja a megfelelő háttérkép számot.
***public void update()*:**
Meghívja a constraint függvényt, a Knight update függvényét és a Map checkCollisions függvényét.
***public void loadPosition(String s)*:**
Betölti a játékállást a paraméterként kapott Stringnek megfelelő nevű JSON fájlból, majd a fájlból kiolvasott háttérképszámnak megfelelően frissíti a Map-ot.
***public void savePosition(String s)*:**
Kiírja a játékállást a paraméterként kapott Stringnek megfelelő nevű JSON fájlba.
public void keyPressed(KeyEvent e) és ***public void keyReleased(KeyEvent e)*:**
Beállítja a Knight osztály billentyűváltozóit azok állapotának megfelelően.

2.2.5 A Knight metódusai:

A Rectangle osztály leszármazottja.
public void updateVel():
Beállítja a lovag sebességét a lovag állapotának megfelelően.
public void nextState():
Beállítja a lovag következő állapotát a lovag sebessége és aktuális állapota alapján.
public void update():
Meghívja a lovag nextState és updateVel függvényeit, majd hozzáadja a sebességet a pozíciójához, a sebességhez pedig hozzáadja a gravitációt.
public void getImage():
Visszaadja a lovag állapotához tartozó képet, amit a GamePanel fog majd kirajzolni.

2.2.5 A Map metódusai:

public void build(int screenNumber):
Kiolvassa a *platforms.json* fájlból a kapott paraméternek megfelelő objektumot, majd ebből kiolvassa a platformokat és egy láncolt listában tárolja el azokat.
***public void checkCollisions(Knight k)*:**
A láncolt listában tárolt összes platformra leellenőrzi, hogy a lovag ütközik-e vele, és ha igen, meghívja a platform *handleCollision(Knight k)* függvényét.

2.2.6 A Platform interface:

boolean collide(Knight k);
void handleCollision(Knight k);
void draw(Graphics g);

2.2.7 A RectPlatform metódusai

Ez az osztály a Rectangle leszármazottja és megvalósítja a platform interfészt.

boolean collide(Knight k);
Ellenőrzi, hogy a lovag ütközik-e a platformmal.
void handleCollision(Knight k);
Beállítja a lovag sebességét és állapotát az ütközés irányának megfelelően. Ha felülről esik a lovag egy platformra, a sebessége 0 lesz, nem csúszik a platformon.
void draw(Graphics g)
Kirajzolja a platformot, egy piros téglalapként

2.2.8 A SlipperyPlatform metódusai

Ez az osztály is a Rectangle leszármazottja és megvalósítja a platform interfészt.

boolean collide(Knight k);
Ellenőrzi, hogy a lovag ütközik-e a platformmal.
void handleCollision(Knight k);
Beállítja a lovag sebességét és állapotát az ütközés irányának megfelelően. Ha felülről esik a lovag egy platformra, a lovag addig pattog, amíg nem lassul le eléggé a függőleges irányú sebességé, és közben csúszik a platformon.
void draw(Graphics g)
Kirajzolja a platformot, egy kék téglalapként.

2.2.9 A Coord metódusai

public Coord add(Coord c):
Hozzáadja a paraméterként kapott kétdimenziós vektort a tárolt vektorhoz.

2.3 Forrásfájlok és adatszerkezetek

A játék által felhasznált képek és fájlok a *resources* mappában találhatóak. Ezen belül a *screens* mappában van a pálya egységeihez tartozó negyvenhárom háttérkép, valamint a platformokat helyzetét tároló *platforms.json* fájl. Ez a fájl csak az első öt képernyő platformjait tárolja, ugyanis a platformok helyzetének kiszámítása és elmentése kifejezetten időigényes feladat volt.
A *knight* mappában találhatóak a lovag különböző állapotaihoz tartozó képfájlok. Az állapotok közül csak a jobbra és balra sétálásnak, valamint az ugrásra készülésnek van külön képfájl, a többi állapot is a jobbra sétálás képét használja.
A mentések a *saves* mappába kerülnek. Itt a *null.json* fájl tárolja az egy új játékhoz indításához szükséges értékeket. Amikor betöltünk a programból egy játékállapotot, az megkérdezi a nevünket, majd a név + „.json” fájlt fogja betölteni. Hasonlóan mentéskor is így képezi az elmenteni kívánt fájl nevét a program. A mentés fájlok tartalmazzák a lovag helyzetét, sebességét, valamint, hogy éppen melyik képernyőn volt a mentés időpontjában.

3. Tesztelés

A tesztelést a JUnit 4 használatával valósítottam meg. A tesztesetek tesztelik a Coord osztály összeadó függvényét, a MainFrame changePanel függvényét, a KnightHandler constrain függvényét, a Knight nextState, updateVel és update függvényeit, a Map build függvényét, valamint a RectPlatform és a SlipperyPlatform osztályok collide és handleCollision függvényeit.

4. Felhasználói kézikönyv

4.1 A menük

A program elindítása után a játékos a főmenübe kerül, ahol négy gomb van, amit megnyomhat a játékos. Ezek közül a legalsó az *exit* gomb, ennek megnyomására bezáródik az ablak, a program leáll. Az *exit* gomb felett a *hitbox* gomb található. Alap állapotba a platformok kereteinek megjelenítése ki van kapcsolva, de ha erre a gomba kattintunk, ezt bekapcsolhatjuk, újbóli kattintással pedig ismét ki. A főmenü alulról harmadik gombja a *load* button. Ha erre a gomba kattintunk, megjelenik egy dialog box, ahol megadhatjuk a nevünket. Ha már korábban játszottunk a játékkal, és a saját nevünkkel mentettük el az állásunkat, akkor az elmentett játékállás fog betöltődni. Ha még nem, akkor új játékot kezdünk, de a program megjegyzi a nevünk, és mentés esetén erre a névre kerül majd a mentett fájl. A legfelső gomb a *new game* gomb, ezt megnyomva indíthatunk el egy új játékot.

A játékól az escape gomb megnyomásával hozhatjuk elő a játékbeli menüt. Itt és négy gomb van, amikből az alsó kettő működése megegyezik a főmenü alsó két gombjával, azzal a különbséggel, hogy itt a legalsó gomb csak a főmenübe lép vissza.
A legfelső gomb a *resume* gomb, megnyomására visszakerülünk a játékba, és folytathatjuk azt.
Ez alatt található a *save* gomb, ide kattintva, ha a program már ismeri a nevünket (mert betöltéskor megadtuk azt) akkor elmenti a játék állapotát a nevünkre. Ha még nem, akkor megjelenik egy dialog box, ahol ismét megadhatjuk a nevünket amire a játék mentve lesz.

4.2 A játék irányítása

Miután a menüből átkerültünk a játékba, a jobbra és balra nyíllal, valamint a szóköz billentyűvel irányíthatjuk a lovagunkat. A jobb nyíl lenyomására jobbra kezd el sétálni a lovag, a bal nyíl hatására pedig balra. Ha mindkettőt egyszerre nyomjuk, a lovag nem mozdul. Ugrani a szóköz billentyű lenyomásával lehet: ekkor a lovag elkezdi gyűjteni az erejét az ugráshoz. Minél tovább tartjuk nyomva a szóközt, a lovag annál magasabbra fog ugrani. Ha elég sokáig tartjuk nyomva a szóközt, hogy az ugrás elérje a maximális erejét, a lovag magától is el fog ugrani.
Amíg a lovag ugrik, vagyis a levegőben van, nem tudjuk irányítani, csak ha ismét visszaesett egy platformra és megállt. Egy platformnak oldalról vagy alulról nekiugorva a lovag visszapattan, míg, ha felülről ugrik rá, akkor a platformtól függően vagy megáll azon, vagy pedig pattogva csúszni fog amíg meg nem áll.