Pikachu’s Revenge

<https://github.com/richardfoltin/PikachusRevenge>  
ELTE IK - Programozási technológia 1.  
3.feladatsor/egyéni feladat, 1. feladat alapján

*Foltin Csaba Richárd – I37M02*

# Feladatleírás

## Közös követelmények

A megvalósításnak, felhasználóbarátnak és könnyen kezelhetőnek kell lennie. Törekedni kell az objektumorientált megoldásra, de nem kötelező a többrétegű architektúra alkalmazása.

A megjelenítéshez első sorban elemi grafikát kell használni. Az így kirajzolt ’sprite’-ok közül a játékoshoz tartozót billentyűzet segítségével lehessen mozgatni a jelenleg is standard (WASD) billentyűzet kiosztásnak megfelelően. Egyéb funkciókhoz egérhez kapcsolódó esemény vezérlőket is implementálhattok.

Amennyiben nem algoritmussal generáltatod a játékteret, úgy legalább 10 előre definiált játékteret készíts különböző fájlokban eltárolva. Ügyelj arra, hogy mind az algoritmussal generált játékok esetén, illetve az előre definiált esetekben is végig játszható legyen az adott terület.

Minden feladathoz tartozik egy időzítő, mely a játék kezdetétől eltelt időt mutatja.

A dokumentációnak tartalmaznia kell a választott feladat leírását, elemzését, a program szerkezetének leírását (UML osztálydiagrammal), egy implementációs fejezetet a kiválasztott játék szempontjából és/vagy az általad érdekesebbnek gondolt algoritmusok leírásával. (Például: pálya generáláshoz implementált algoritmus.), valamint az eseményeseménykezelő párosításokat és a tevékenység rövid leírását.

A feladatleírás a minimális követelményeket tartalmazza. A játékok tetszőlegesen bővíthetők.

## Maci Laci feladat alapján

A meséből jól ismert Maci Laci bőrébe bújva a Yellowstone Nemzeti Park megmászhatatlan hegyei és fái között szeretnénk begyűjteni az összes rendelkezésre álló piknik kosarat. Az átjárhatatlan akadályok mellett Yogi élelem szerzését vadőrök nehezítik, akik vízszintesen vagy függőlegesen járőröznek a parkban. Amennyiben Yogi egy egység távolságon belül a vadőr látószögébe kerül, úgy elveszít egy élet pontot. (Az egység meghatározása rád van bízva, de legalább a Yogi sprite-od szélessége legyen.) Ha a 3 élet pontja még nem fogyott el, úgy a park bejáratához kerül, ahonnan indult.

A kalandozás során, számon tartjuk, hogy hány piknik kosarat sikerült összegyűjtenie Lacinak. Amennyiben egy pályán sikerül összegyűjteni az összes kosarat, úgy töltsünk be, vagy generáljunk egy új játékteret. Abban az esetben, ha elveszítjük a 3 élet pontunkat, úgy jelenjen meg egy felugró ablak, melyben a nevüket megadva el tudják menteni az aktuális eredményüket az adatbázisba. Legyen egy menüpont, ahol a 10 legjobb eredménnyel rendelkező játékost lehet megtekinteni, az elért pontszámukkal, továbbá lehessen bármikor új játékot indítani egy másik menüből.

## További egyéni célok

* A karakterek képe tényleg mozgás hatását keltse, ne csak egy statikus kép legyen
* A pályák egy spritesheetből legyenek felépítve
* A karakterek ne csak függőlegesen és vízszintesen mozogjanak, hanem koordinátarendszer alapján

# Feladat Elemzése

Első lépésként a játék témájának meghatározása volt a fő cél. Mivel az Interneten Pokémon témában rengetegsok spritesheetet és karaktersprite-okat lehet találni ezért adta magát a téma. A játékban Pikachut lehet majd irányítani, hogy a pályákon elszórt pokélabdákból kiszabadítsa az elkapott pokémonokat. Eközben a pályákon mászkáló pokémontrainerek őt próbálják elkapni, amint a látószögükbe kerül.

A pályákat egy külső spritesheet editor programmal szerkesszük. A program generál egy file-t amely a spritesheeten található 16x16-os sprite-okhoz azonosítót rendel, és még esetleg egyéb paramétereket is megadhatunk nekik. Továbbá minden pályához tartozik egy-egy file ami leírja layer-enként hogy melyik azonosítójú sprite hova kerül. A pályát leíró file tartalmazza továbbá az NPC-k haladási útvonalát és paramétereit.

A játék JFrame ablakokat fog használni a program megjelenítésére. Az ablakon található egyéb elemek (Főmenü, pályák, információs panelek) cserélődni fognak.

A játék egy JScrollFrame-ben zajlik ahová betöltjük a pályát. A játék logikáját egy külön Model objektumban kezeljük. A program egy végtelenített ciklusban 40ms-onként újraszámolja a pozíciókat és újrarajzolja az ablakot. A játékot az irány billentyűkkel lehet irányítani ezért figyeljük a lenyomott billentyűket és ezekből 8 irány (4 alap + 2 egyszerre nyomott billentyű) alapján mozgatja Pikachut. Amennyiben egy a pályán található pokélabdára lép egy játékos, onnan kikel egy random pokémon. Az ellenfelek a pályán járőröznek a megadott útvonal szerint. Amennyiben a játékos a ellenfelek előtti 135°-ban tartózkodik a megadott távolságon belül akkor először megállnak – hagynak egy kis időt a játékosnak elmenekülni – de utána elkapják és a játékos visszakerül a kezdőpozícióba. A kiszabadított pokémonok megmaradnak, de egy élte elveszik. A pályát a minden pálya végén található tábla mellett SPACE-t nyomva lehet tovább vinni, amennyiben elegendő pokélabdát talált meg a játékos.

Azt hogy a játékos ne tudjon áthaladni a tereptárgyakon (házak, fák, víz, hegyek, …) úgy oldjuk meg, hogy a spritesheethez tartozó információs file-ban az collisiont okozó sprite-oknak paramétert adunk. A játékos mozgatásával egy hozzá tartozó láthatatlan négyzetet is mozgatunk, és ha ez egy tereptárgyon akarna áthaladni, akkor nem mozgatjuk oda a játékost. Mivel a pályák több layerből is állhatnak ezért megoldható hogy egy másik layeren a tereptárgyon áthaladható dolog legyen (híd, lépcső, …). A layereken készítünk mindig egy layert, amit majd mindig a játékosok felett rajzolunk ki.

A játéktér felett egy címkén megjelenítjük a játékos életerejét, az pályán található kiszabadított és ki nem szabadított pokémonokat és egy időt, mely a pálya kezdetétől eltelt időt méri.

A játékosok segítségére egy alsó információs címkét tervezünk, mely segítő és információs üzenteket küld a játékos számára.

A játékhoz tartozik egy menüsor is mely a szokásos mentés, betöltés, újraindításon kívül tartalmaz még egy a játék óráját szüneteltető és újraindító menüt. Továbbá egy dicsőséglistát.

# Használt Eszközök

* Tiled (<https://www.mapeditor.org/>) – Könnyen kezelhető közösségi fejlesztésű spritesheet editor, mely tartalmaz a generált file-ok beolvasásához használható java library-t is.
* Pokémon Insurgence (<https://p-insurgence.com/>) – Közösségi fejlesztésű pokémon játék, ahonnan a játékban használt képek és spriteshet származik
* Maze Creator (<http://www.mazegenerator.net/>) – A játék 9. pályáján található labirintus legeneráláshoz
* GitHub (<https://github.com/richardfoltin/PikachusRevenge>) – A játék fejlesztést folyamatosan publikáltam a GitHubra
* GenMyModel (<https://app.genmymodel.com/>) – UML creator GitHub alapján

# Felhasználói Esetek

A felhasználók a következőket tudják tenni a játékelindítása után:

1. Főmenüben
   1. Új játék indítása -> utána kiválasztása a nehézségi foknak
   2. Játék betöltése file-ból
   3. Játék betöltése adatbázisból
   4. Dicsőségtábla megjelenítése
   5. Az ablak bezárása
2. A játékban (ugyanazt mint a főmenüben kiegészítve a következőkkel)
   1. Pikachu irányítása WASD vagy a nyíl billentyűkkel
   2. SPACE lenyomásával interakcióba lépés Pikachu közelében lévő tereptárgyakkal (ha lehetséges)
   3. A játék elmentése file-ba
   4. A játék elmentése adatbázisba
   5. Visszalépés a főmenübe
   6. A játék lepause-olása és újra aktiválása
   7. Az adott pálya újrakezdése
   8. Egy már teljesített pályára visszalépés
   9. Az elkezdett pályák statisztikáinak megjelenítése
   10. A segítség ablak előhívása
   11. A megtalált pokémonokhoz kapcsolódó internetes leírás előhívása

# Program Szerkezete

A programban az összes használt osztály és a hozzájuk tartozó bonyolultabb metódusok ki vannak egészítve a működésüket leíró Javadoc-kal. Itt az érdekesebb, kihívásokat jelentő eseteket mutatom csak be

## A pálya felépítése

Az összes pálya egy sprite-okat tartalmazó képfile elemeiből van felépítve, és a Tiled editor programmal készítettük. A Tiled programnak két fő része van; az egyiken a spritesheethez tartozó tulajdonságokat lehet beállítani, a másikban magát a pályát lehet rétegenként felépíteni a sprite-okból.

### Spritesheet editor

A spritesheet fel van osztva 16x16-os négyzetekre (csempékre) és ezekhez a program egy-egy azonosítót rendel. A csempékhez különböző adattagokat is lehet rendelni. Az általam használt adattagok:

* Sign (boolean) – mellette állva lehet majd SPACE-t nyomva a következő pályára haladni
* Collision (boolean) – ezen a csempén nem lehet áthaladni
* Stairs (boolean) – felülírja a Collisiont és ezen mégiscsak át lehet haladni
* Water (boolean) – ezen a csempén szintén nem lehet áthaladni mert víz
* Bridge (boolean) – felülírja a Water-t és ezen mégiscsak át lehet haladni
* Ball (boolean) – a pokémonokat reprezentáló pokélabda, amire rálépésekor a programban egy esemény történik majd

Mivel a pályák több rétegből állnak, ezért a keresett pozícióban bármelyik rétegen is talál a program valamilyen tulajdonságot, azt az egész pozícióra érvényesnek tartja.

A spritesheeten továbbá be lehet állítani Terrain (terep)-eket is. Ez csak a pálya elkészítésekor van segítségünkre, hogy a Tiled program a megfelelő csempét (oldalak, sarkok) rakja a terepek határára.

### Map editor

A pályák több rétegből állnak. Ezek közül két rétegnek van speciális funkciója:

A szisztematikusan „Above” -nak nevezett réteg (és az esetleg felette lévő rétegeket) a program mindig az egységek felett fogja megrajzolni. A többit az egységek alatt. Ezzel elérhető az a hatás mintha be lehetne menni egy-egy tereptárgy mögé. Például egy 2x2 csempe nagyságú fa esetén az alsó két csempe Collision-ra van állítva (hogy a törzsén ne lehessen átmenni) a felső két csempe pedig az „Above” rétegre van rakva, hogy be lehessen menni a fa mögé.

Minden pályához tartozik egy object réteg, ahol a kézzel rajzolt objektumok vannak. Ebből három félét különböztetek meg:

* „Enter” pont – a pálya kezdetekor Pikachu kezdőpontja
* „Back” pont – a pályára visszalépéskor Pikachu kezdőpontja (általában a „Sign” közelében)
* „NPC” polygon – az NPC-k útvonala. Lehet nyitott vagy bezárt. Általában több tulajdonságot tartalmaz ami leírja az útvonal tulajdonságait

### NPC-k tulajdonságai

Az NPC-ket reprezentáló polygonoknak a következő tulajdonságai lehetségesek (a polygonok pontjainak számozása 0-tól indul):

* Level (int) – a játékban az NPC azonosítója. Ez alapján fogja beállítani a játék az NPC képét, nevét, mozgási sebességét, dobási távolságát, és felkészülési idejét.
* Start (int) – a pálya betöltésekor a polygon hanyadik pontján kezdje az NPC az útját Ha nincs megadva akkor az elején. -1 megadása esetén egy véletlenszerű ponton.
* Reverse (int) – a polygon hányadik pontján forduljon vissza az NPC. Általában a polygon utolsó pontját érdemes megadni nyitott polygon esetén
* Teleport (int) – a polygon hányadik pontján teleportáljon vissza az NPC az elejére. Általában a polygon utolsó pontját érdemes megadni nyitott polygon esetén
* Carry (boolean) – ez egy olyan NPC amire fel lehet szállni (csak a 8. pályán van ilyen)
* Wait%d (int) – a tulajdonság nevében van egy szám, ami azt jelzi, hogy polygon hanyadik pontjáról van szó. A tulajdonság értéke azt jelenti hogy az adott ponton hát játékciklust (40ms) várjon az NPC

## Két lenyomott iránygomb egyszerre érzékelése

A billentyűk lenyomását a MainWindow-hoz adott KeyAdapter érzékeli amit elküld feldolgozásra a KeyPressHandler statikus metódusokat tartalmazó osztálynak, ami tartalmaz egy listát az aktív irányokról. Amikor lenyomunk egy billentyűt, a directionFromKeyCode metódus kiszámolja, hogy ez a játékban melyik iránynak felel meg, és a newDirectionWith metódus megpróbálja hozzáadni ezt a listához a következők szerint:

* ha már benne van a listában, akkor nem törtink semmi
* ha a listában csak egy irány van akkor hozzáadja
* ha a listában két irány van, akkor a régebbit kiveszi onnan, és az újat hozzáadja

Ezután a listából kiszámolt irányt (pl LEFT és UP esetén UPLEFT) elküldi a Player objektumnak hogy próbáljon meg a karakterrel abba az irányba mozogni.

Ha felengedünk egy billentyűt, akkor azt eltávolítja a listából ha benne van, és szintén elküldi a Player objektumnak a maradékból kiszámolt irányt (nincs benne semmi, akkor STOP).

## Pokemonok megpróbálják kikerülni az akadályokat

Minden Pokemon objektum a játék főciklusában kiszámolja a distance adattagját aszerint, hogy az adott pályán ő hányadik megtalált Pokemon a pokémonokat tartalmazó listában. Ezután amikor a Pokemon a következő pozícióját akarja betölteni a következőképpen csinálja:

* ha a MAX\_DISTANCE (330)-nél távolabb van a játékostól, akkor simán elindul a játékos irányába, és nem törődik a terepakadályokkal, mivel ekkor nem látszódik a képernyőn
* ha közelebb van de a distance adattagjánál távolabb akkor első körben megpróbál a játékos irányába mozogni a Direction.getDirection metódus által visszaadott értékkel
* ha nem megy, akkor megpróbálja a Direction.getSecondDirection-t
* ha ez sem megy, vagy a distance-nél közelebb van, akkor megáll

### Direction.getDirection (Position from, Position to)

Két térképen megadott koordinátából próbálja kiszámolni az irányt.

Először kiszámoljuk a két koordináta különbségét (=a hozzá tartozó derékszögű háromszög befogóit előjelesen) Ezáltal kapunk egy az origóból kiinduló irányvektort.

A 8 féle Direction 8 db 45°-os síknyolcadra osztja a teret, viszont elcsúsztatva (pl RIGHT: -22,5°-tól +22,5°-ig) Tehát inkább 16 síktizenhatoddal érdemes számolnunk. (16.,1.: RIGHT; 2.,3.:UPRIGHT; 4.5.:UP…)

Vesszük a két befogó hányadosának arkusztangesét, ami visszaadna egy 1. vagy 4. síknegyedbe lévő szöget. Ennek véve az abszolút értékét akkor csak az 1. síknegyedre vetítünk. Ezt osztva PI/8-cal megkapjuk hogy az első negyedben hányadik tizenhatodba mutat a szög. Ezután a következőképp számolunk:

* 0. (0-22,5°között) – Ha a dx negatív volt akkor LEFT, ha pozitív akkor RIGHT
* 1. vagy 2. (22,5°-67,5°között) – A dx és dy előjelei alapján valamelyik átlós irány
* 3. (67,5°-90°között) – dy előjele alapján UP vagy DOWN
* 4. (90°) – abban a speciális esetben ha pont egy tengelyen van a kettő. Ugyanaz mint 3.

### Direction.getSecondDirection (Position from, Position to)

Ez a metódus arra szolgál, hogy ha játékos irányába nem tud haladni a pokémon, mert ott valamilyen akadály van, akkor azt esetleg ki tudja kerülni. Mivel az akadályok oldala a koordinátarendszer tengelyeivel párhuzamos egyenes, ezért a két pozíció közötti vektor kisebb befogója által mutatott irányt érdemes venni. Ezt hasonlóképpen állapítjuk meg amint a getDirection metódusnál, csak itt elég 8 részre osztani a síkot.

Még egy olyan kiegészítés van, hogy ha ez a befotó kisebb lenne mint amennyi távolságot a játékos (és egyben a pokémon is) egy játékciklus alatt mozog (3) akkor inkább ne mozogjon, hogy elkerüljük azt, hogy pl dx=-1,5 és dx=1,5 között jobbra balra ugrál a pokémon.

## NPC-k különböző állapotai

Azért hogy NPC-k különböző állapotait megkülönböztessük minden NPC-nek van egy NPC\_STATE-eket és NpcState-eket tartalmazó Map-je. Az NpcState egy segítő osztály arra hogy a különböző állapotokról tudjuk hogy éppen aktívak-e és számolja hogy mikor (hány játékcikuls múlva) jár le. Az NPC\_STATE enum a következők lehetnek:

* STOP\_LOOKOUT – az npc megállt mert az útvonalán itt egy várakozási pont van
* STOP\_EXCLAMATION – az npc megállt, mert a látóterébe került a játkos és felkészül a dobásra
* WALKING\_CAUTIOUS – az npc miután felkészült a dobásra ilyen felkészült állapotba kerül, ahol egyből dob, ha játékos a látóterébe kerül
* STOP\_THROW – az npc a dobáshoz is megáll egy rövid időre

A játék fő ciklusában folyamatosan ellenőrizzük, hogy valamilyen állapotba került-e (vagy kikerült-e onnan) az npc, és ez alapján mozgatjuk vagy dobunk vele.

* ha az útvonalán a következő olyan pozícióra fog lépni ahol várakozás van, akkor elindítjuk a STOP\_LOOKOUT-ot
* ha egyik STOP\_.. sem aktív akkor mozgatjuk az npc-t
* ha épp lejárt a STOP\_EXCLAMATION akkor beindítjuk a WALKING\_CAUTIOUS-t

Az npc akkor dob ha:

* nem dob már éppen
* a player már nincs elkapva
* a player dobási távolságon belül van
* a player a látószögében van (vagy a 9. pályán nem takar be fal sem)
* a player nem épp egy szállítóeszközön van

## Elrejtett adatbázis id kinyerése a betöltési táblából

Ahhoz hogy a betöltési dialógusban megjelenő táblázatból jól be tudjuk azonosítani a betöltendő sort a sorhoz tartozó id-t használjuk. Viszont nem szeretnénk hogy ez az id magában a táblázatban is látszódjon.

Ehhez azt kellett tenni, hogy maga a táblázat modelje (LoadTableModel) tartalmazza az id-t (az első oszlopban, viszont a táblázat megjelenítésekor a table .getColumnModel().removeColumn(…) metódussal eltávolítjuk. Ilyenkor csak a megjelenített táblázatból tűnik el az oszlop, magában a modelben még megmarad.

Viszont ilyenkor a tábla getValueAt() metódusára nem tudunk hagyatkozni az érték eléréséhez hanem csak a model getValueAt() metódusára. De mivel a megjelenített tábla sorai nem feltétlen ugyanabban a sorrendben vannak mint a model sorai ezért át kell számolnunk a tábla kiválasztott sorának indexét a model sorának indexére a convertRowIndexToModel() metódussal.

# Osztálydiagram

Mellékelt <uml.png>

# Tesztelési terv

A játék tesztelését két féleképpen végeztem; white box (JUnit tesztekkel) a játék irányításáért, pozíciókért és koordinátákért felelős metódusokat, és black box (a játék használatával) a teljes játék működését teszteltem.

## Black box:

1. Főmenü
   1. játék indítása – SIKERES
   2. játék betöltése file-ból - SIKERES
   3. hibaüzenet hibás file esetén - SIKERES
   4. játék betöltése adatbázisból - SIKERES
   5. hibaüzenet hibás adatbáziskapcsolat esetén betöltéskor – SIKERES
   6. dicsőségtábla betöltése – SIKERES
   7. hibaüzenet hibás adatbáziskapcsolat esetén a dicsőségtáblakor - SIKERES
2. Játék
   1. Játékos
      1. Pikachu mind a 8 irányba megy WASD és nyilak használatával – SIKERES
      2. Pikachu nem át az akadályokon (fák, víz, hegyek, házak, kövek) – SIKERES
      3. Pikachu átmegy a vizen ha van rajta híd – SIKERES
      4. Pikachu átmegy a hegyen ha van rajta lépcső – SIKERES
      5. Pikachu eltűnik a magas tárgyak mögött – SIKERES
      6. a megfelelő animáció jelenik meg az adott mozgás esetén - SIKERES
      7. amennyiben Pikachut elkapják, visszakerül a kezdő pozícióba és egy életet veszít – SIKERES
      8. a pálya végén található táblával a következő pályára lehet továbbmenni - SIKERES
   2. NPC
      1. a pálya file-jában megadott NPC töltődik be a megadott paraméterekkel - SIKERES
      2. az NPC-k a megadott útvonalon mozognak és megfelelő pontokon várakoznak - SIKERES
      3. az NPC-k felett felkiáltójel jelenik meg ha meglátják Pikachut – SIKERES
      4. az NPC-k megállnak ha meglátják Pikachut – SIKERES
      5. az NPC-k eldobják a labdájukat ha már felkészültek, vagy ha Pikachu túl közel került – SIKERES
   3. Pokémonok
      1. amennyiben Pikachu rámegy egy pokélabdára kiszabadul a pokémon és megjelenik a felső címkén – SIKERES
      2. a kiszabadított pokémonok libasorban követik Pikachut – SIKERES
      3. a kiszabadított pokémonok ugyanúgy nem mennek át tereptárgyakon, csak ha Pikachu túl messze van – SIKERES
3. Menüsor
   1. File
      1. új játék indítása - SIKERES
      2. játék betöltése file-ból – SIKERES
      3. játék mentése file-ba – SIKERES
      4. játék mentése a betöltött file felülírásával – SIKERES
      5. játék betöltése adatbázisból – SIKERES
      6. játék mentése adatbázisba – SIKERES
      7. játék mentése adatbázisba felülírással – SIKERES
      8. visszalépés a főmenübe megerősítőablakkal – SIKERES
      9. kilépés megerősítőablakkal – SIKERES
   2. Game
      1. a pálya újrakezdése – SIKERES
      2. pálya újrakezdésekor feltöltődik az élet a pálya betöltésekori értékre – SIKERES
      3. pálya újrakezdésekor Pikachu visszakerül a kiindulási pontra – SIKERES
      4. pálya újrakezdésekor a kiszabadított pokémonok újra labdába kerülnek – SIKERES
      5. pálya újrakezdésekor az NPC a kiindulási helyükre kerülnek – SIKERES
      6. pálya újrakezdésekor az óra lenullázódik – SIKERES
      7. szintlépés másik pályára ha lehetséges (1-10) – SIKERES
      8. szintlépés csak már teljesített pályára lehet – SIKERES
      9. pause/resume – SIKERES
      10. dicsőségtábal megjelenítése – SIKERES
      11. az adott játék összesítő statisztikájának megjelenítése – SIKERES
      12. a segítség megjelenítése – SIKERES
      13. felugró ablakok esetén a játék szünetel és bezáráskor visszaáll – SIKERES
   3. Pokédex
      1. Csak a kiszabadított pokémonok láthatóak benne – SIKERES
      2. menüpontra kattintáskor megjelenik a pokémon html oldala – SIKERES

## White box

1. Position
   1. distanceFrom(Position d)
   2. movePosition(Direction d, double speed)
2. TilePosition
   1. tileCenter(TilePosition tpos)
   2. tileCoordFromMapCoord(double coord)
   3. fromMapPosition(Position p, int level)
3. KeyPressHandler
   1. lenyomunk egy gombot megfelelő-e az irány?
   2. lenyomunk egy másik gombot megfelelő-e a komponált irány?
   3. felengedünk egy gombot, akkor újra megfelelő-e az irány?
4. Direction
   1. getDirection(double x, double y)
   2. getDirection(Position from, Position to)
   3. getSecondDirection(Position from, Position to)
   4. isInDirectionOfSight(Direction from, Direction to)
   5. directionAngleStart(Direction d)