Tóth Zalán

(CZ72YM)

Feladat leírás:

**Kígyó (Snake)**

Kezdetben egy 2 egység (fej és csörgő) hosszú csörgőkígyóval kell felszednünk a sivatagos játéktéren megjelenő élelmet. A játéktéren egyszerre 1 elemózsia lehet véletlenszerűen elhelyezve olyan mezőn, melyen nem a kígyó található. A kígyó a játéktér közepéről egy véletlenszerűen választott irányba indul. A továbbiakban a felhasználó a billentyűzet segítségével válthat majd irányt. Élelemhez érve, a kígyó mérete egy egységgel nő.

A játékot nehezítse, hogy a sivatagban kövek is találhatók melyeknek, ha nekimegy a kígyó, akkor véget ér a játék. Abban az esetben is elveszítjük a játékot, ha a kígyó saját magának megy neki, vagy a pálya szélének.

Ezekben az esetekben jelenjen meg egy felugró ablak, melyben a játékos a nevét megadva el tudja menteni az adatbázisba az eredményét, mely a játék során a kígyó által elfogyasztott élelem összege. Egy menüpontban legyen lehetőségünk a 10 legjobb eredménnyel rendelkező játékost megtekinteni, az elért pontszámukkal, továbbá lehessen bármikor új játékot indítani egy másik menüből.

# Fejlesztői eszközök:

* IntelliJ 2020.3
* Java 15
* Gradle 6.7
* H2
* Lombok 1.18
* Junit 5.6

Fordítás: *gradle build*   
Futtatás: *java -jar snake-1.0.0.jar*

# Feladat elemzése:

A feladat megoldásához három képernyő szükséges:

* Menü
* Legjobb eredmények
* Játéktér

A menüben két gomb található. Az első a Legjobb eredmények képernyőre navigál, a másikkal a játék indítható.

A Legjobb eredmények megjelenítéséhez szükséges az elmentett adatok betöltése adatbázisból, ezek kiírása a képernyőre, valamint egy gomb, amellyel visszatérhetünk a menübe.   
  
Használhatnánk külső adatbázist, de a felhasználónak kényelmetlenné tenné a szoftver beállítások kezelését, így a játék indulásakor egy H2 adatbázis szervert indít a program a 9092-es porton. Az adatbázist a felhasználó könyvtárában hozza létre. Az alkalmazás leállításakor az adatbázis szerver is leállításra kerül, mely a JFrame-hez csatolt WindowLister-en keresztül kerül implementálásra.  
  
A játéktér felépítése és működtetése több részből áll.  
Először a kígyót helyezzük a kezdőhelyre. Itt kialakítunk egy védett területet, melyen belülre nem kerül akadály, elkerülve, hogy kezdésnél azonnal elpusztuljon. Ezután véletlenszerűen felhelyezzük az akadályokat jelentő sziklákat, ügyelve, hogy egy pozícióra ne kerüljön több. A sziklák száma és mérete bizonyos korlátok között randomizált. Végül az előző szempontokat figyelembe véve felkerül az eleséget szimbolizáló alma egy véletlen helyre a pályán belül.  
  
Az inicializációs rész után elindul a játék órája és a vezérlést átadjuk a játékosnak. A kígyó adott irányba mozog, míg a felhasználó a megfelelő billentyű lenyomásával meg nem változtatja ezt. A kígyó ’előre’ haladása automatikus, bizonyos időközönként egyet lép előre. Ez az idő függ a kígyó hosszától, minél több almát evett, annál gyorsabban mozog (kisebb időközönként hívja az eseményt az időzítő).  
Minden lépés után a program megvizsgálja, sikerült-e almát felvenni (amiért egy pontot jóváírunk a játékos számára), történt-e ütközés sziklával, önmagával, vagy a pálya szélével. Amennyiben nem, a játék folytatódik, ellenkező esetben a játék óráját megállítjuk és bekérjük a játékos nevét. Ha megadja, az eredményét az adott névvel elmentjük az adatbázisba.

A játék irányítása a kígyó mozgásirányának módosításából áll, ez megtehető a WASD billentyűkkel, vagy az irány nyilakkal. A forráskódban bekapcsolható a vizuális debug mód (gizmok), mely megmutatja a sprite-ok körüli ütközési tereket, a kezdőhely körüli védett területet, valamint a C billentyű segítségével tetszőleges méretűvé növelhető a kígyó hossza

# Eseménykezelők:

**Menü:**

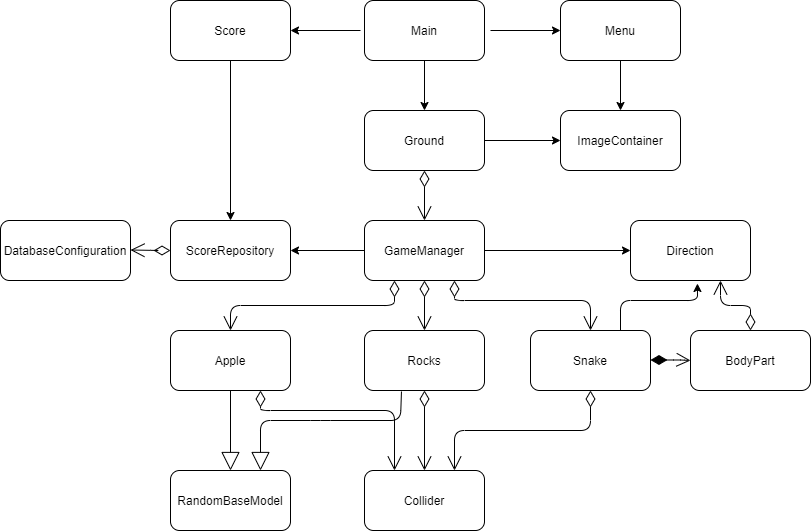
A menüben két gomb lenyomására van lehetőség. A gombok 1-1 MouseListener példánnyal rendelkeznek, melyek a játék-, illetve az eredmények képernyő elindításáért felelős metódust hívják meg.  
  
**Eredmények**:  
  
Az eredmények képernyőn egyetlen gomb található, melyre szintén MouseListener példány van felcsatolva. A kattintás esemény a főmenübe viszi vissza a felhasználót.

**Játék:**

A játék két eseménykezelőt tartalmaz. Az első egy KeyListener, mely a billentyűzeten történő gombok lenyomására figyel. Amennyiben a meghatározott (WASD, Iránynyilak, C) lenyomása történik, az eseménykezelő meghívja az esemény kezelésének megfelelő metódust (Irány változtatás, Csalás).  
  
A másik eseménykezelő a szintén a játéktérre kötött időzített ActionListener, melyet az időzítő bizonyos időközönként meghív. Itt történik meg a játék 1-1 körének kezelése, mint például a kígyó léptetése, az ütközésvizsgálat, a kígyó növelése…

# Diagramok:

Az átláthatóság érdekében a diagramok két részre bontva kerülnek bemutatásra.   
  
Az első diagramon az osztályok kapcsolatai láthatók:



Míg a második az egyes osztályok mezőinek és metódusainak leírása:

# 

# Érdekes algoritmusok:

**Kígyó mozgása:**  
  
A kígyó mozgásának problémája az irányváltoztatásoknál látható. A kígyó egy része más irányba mozog, mint a többi része. Egy hosszabb kígyónál akár több rész is mozoghat másképp.  
Erre megoldásként minden testrésznél nyilvántartjuk a mozgás irányát. Amikor a játékos megváltoztatja a kígyó mozgását, csak a fej irányát változtatjuk meg. Minden léptetésnél a fej a saját irányába mozog egyet előre, a többi testrész pedig megkapja az előtte lévő testrész mozgásának irányát és e szerint kerül mozgatásra. A lánc végén lévő irány pedig kiesik.

public void calculateMoves*() {* Direction previous = getHead*()*.getDirection*()*;  
 for *(*Bodypart bp : snakeBody*) {* computeDirection*(*bp*)*;  
 Direction tmp = bp.getDirection*()*;  
 bp.setDirection*(*previous*)*;  
 previous = tmp;  
 *}  
}*

private void computeDirection*(*Bodypart bp*) {* var rectangle = bp.getRectangle*()*;  
 Direction direction = bp.getDirection*()*;  
 switch *(*direction*) {* case *UP* -> rectangle.y -= Const.*SNAKE\_BODY\_DIMENSION*;  
 case *DOWN* -> rectangle.y += Const.*SNAKE\_BODY\_DIMENSION*;  
 case *LEFT* -> rectangle.x -= Const.*SNAKE\_BODY\_DIMENSION*;  
 case *RIGHT* -> rectangle.x += Const.*SNAKE\_BODY\_DIMENSION*;  
 *}  
}*

**Ütközés vizsgálat:**

A kígyó ütközés vizsgálatánál elég csak a kígyó fejét felhasználni, hisz a test mindig későbbi időpontban ér a fej helyére. Az egyszerűség kedvéért a pályán lévő elemeket listákban tároljuk. Minden pályaelem körül egy bounding box található. Vizsgálatnál a kígyó feje és ezen boxok közötti átfedést vizsgáljuk. Amennyiben van átfedés, a kígyó ütközött az adott pályaelemmel (kő, alma vagy önmaga).  
Figyelni kell, hogy a kígyó fejét ne vizsgáljuk meg önmagával való ütközésre.  
Ezenkívül szükséges a pálya széleivel történő interakciót is számításba venni.

public boolean testSnakeCollision*() {* var headRect = snake.getHead*()*.getRectangle*()*;  
 if *(*testCollidedMapX*(*headRect*)* || testCollidedMapY*(*headRect*)) {* return true;  
 *}* return collideWithElements*(*headRect*)*;  
*}*

private boolean testCollidedMapX*(*Rectangle head*) {* return head.x < 0 || head.x > Const.*WIDTH* - Const.*SNAKE\_BODY\_DIMENSION*;  
*}*

public boolean collideWithElements*(*Rectangle rectangle*) {* var result = testSnakeCollision*(*rectangle*)*;  
 result |= testRocksCollision*(*rectangle*)*;  
 return result;  
*}*

private boolean testRocksCollision*(*Rectangle rectangle*) {* if *(*rocks != null*) {* return rocks.getRocks*()*.stream*()*.anyMatch*(*rectangle::intersects*)*;  
 *}* return false;  
*}*private boolean testSnakeCollision*(*Rectangle rectangle*) {* return snake.getSnakeBody*()*.stream*()* .map*(*Bodypart::getRectangle*)* .filter*(*b -> b != rectangle*)* .anyMatch*(*rectangle::intersects*)*;  
*}*