Drahos Zsolt – UCZFU3

Programozás Alapjai 3

NHF - Dokumentáció

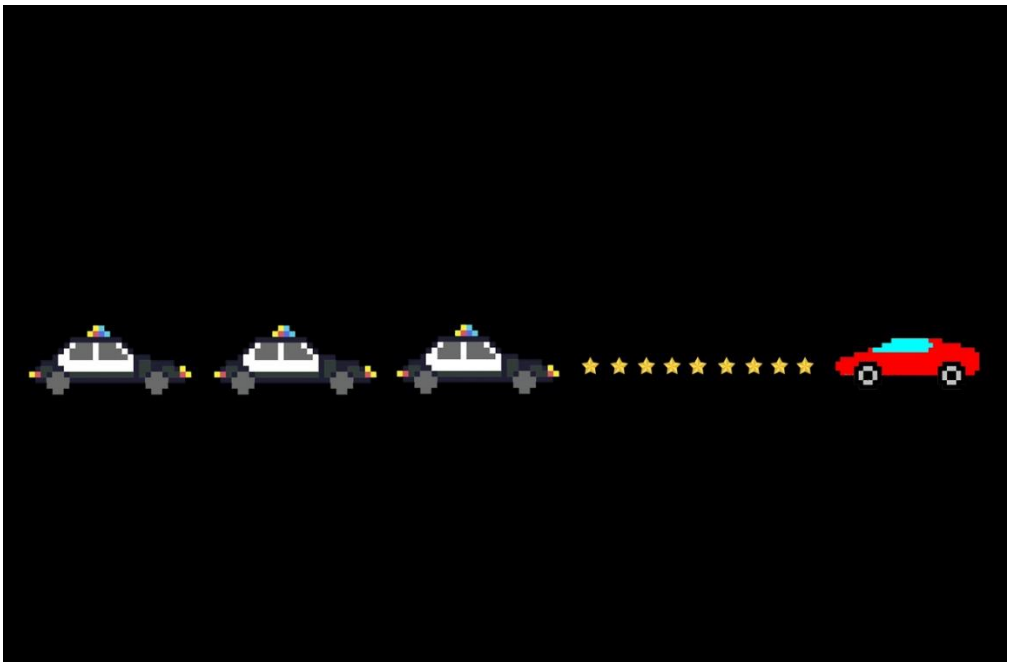
Drahos Zsolt – UCZFU3

HOT PURSUIT

Feladat leírása

A Hot Pursuit nevű játék a PacMan alapjaira épül. A játék egy előre meghatározott pályán játszódik (labirintus), ahol a játékos egy autóval van. A cél, hogy elmenekülj a rendőrök elől anélkül, hogy elkapnának és felszedje az összes csillagot a pályán. A játék úgy kezdődik, hogy egy megadott helyen éled fel a játékos és a rendőrautók pedig elkezdik üldözni egészen addig amig a fel nem szedte az összes csillagot a pályán vagy utol nem érték a rendőrautók. Amennyiben utolérik akkor a három életéből csökken egy és ha ez eléri a nullát akkor a játéknak vége. Minden játék végén kialakul egy pontszám, ami a felszedett csillagok alapján kerül kiszámításra és bekerül az eredménytáblába.

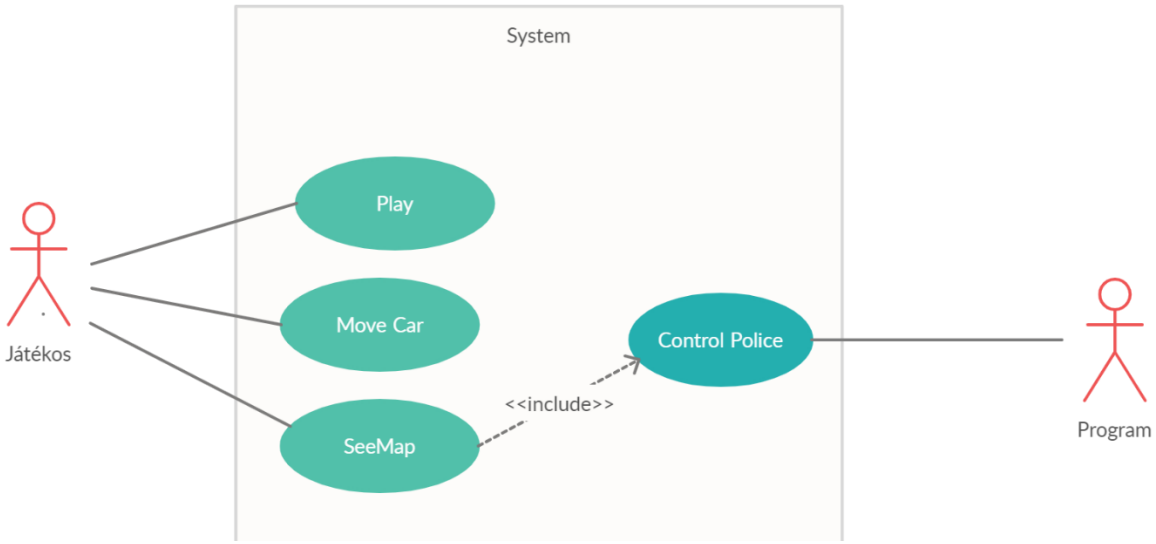
A rendőrök nem tudják felszedni a csillagokat azonban megállás nélkül automatikusan üldözik a menekülő autót



Use-Case

A felhasználó a programban a következőket fogja tudni csinálni:

- Játszani (Play gomb)
- Lekérdezni az eredményeket
- Beállítani a nehézségi szintet.
- Kilépni a programból.



Nev	Play
Aktor	Játékos
Leírás	A játékos elindítja a megfelelő gombbal a játékot a menüben
Forgatókönyv	Rányom a gombra ezáltál megváltozik a scene és megjelenik a pálya és a karakter, amit rögtön el lehet kezdeni irányítani. Vagy be lehet állítani a nehézséget.

Nev	Move Car
Aktor	Játékos
Leírás	A játékos irányítja a karaktert fel, le jobbra és balra
Forgatókönyv	Ezeket majd a nyilakkal tudja működtetni és irányítani amig falba nem ütközik, ahol pedig másik irányt kell választani.

Nev	SeeMap
Aktor	Játékos
Leírás	Játékosnak betölt a pálya és ezáltal megtudja tekinteni, hogy merre szeretne menni
Forgatókönyv	A játékos megtekinti a pálya felépítését

Nev	Control Police
Aktor	Program
Leírás	Rendőrök mozognak a pályán fel le jobbra vagy balra
Forgatókönyv	Játékos meghal, ha rendőrutóval ütközik

Vázlatos ismertetés

A menüben lesz egy play gomb, ahol el lehet kezdeni a játékot tovább be lehet állítani, hogy milyen nehézségi szinten szeretne játszani a játékos. Továbbá lesz egy Scoreboard gomb, ahol pedig a meglehet tekinteni a pontokat, amiket elért eddig.

Az autót a billentyűzeten lévő nyilak segítségével lehet irányítani jobbra, balra fel és le a menüben pedig az egér segítségével lehet elérni dolgokat. A rendőrautókat pedig a program irányítja saját stratégiájuknak megfelelően

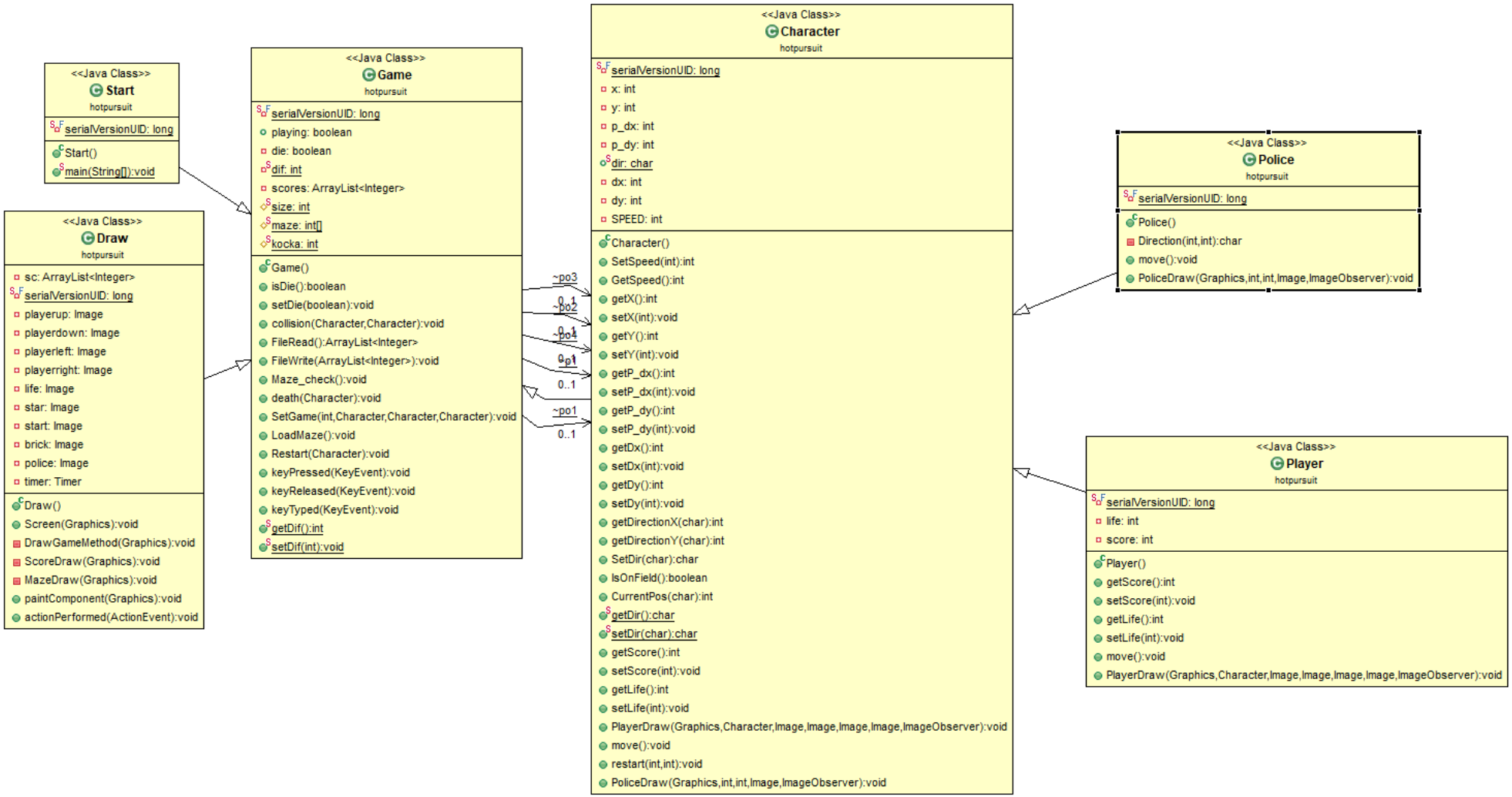
A programban pálya lesz generálva, ami egy n-es tömbben lesz eltárolva és az alapján lesz legenerálva grafikus formában. A pályán lesznek „falak" amiken ha a játékos neki megy akkor nem tud tovább menni, ekkor kénytelen irányt változtatni. A pálya minden [i]. cellájában van egy csillag (ahol nincs fal), ezek a csillagok eltűnnek miután a játékos át ment rajtuk.

A játék során eltárolom az x és az y koordinátát tovább az irányokat h melyik irányba megy az autó. Erre azért van szükség, hogy majd a megfelelő játékos ikon jelenjen meg amikor valamilyen irányba megy.

A scoreboard adatait fájlban lesznek eltárolva, ahonnan mindig beolvassa az előzőleg elért pontokat és meg megjeleníti a játékban. Minden egyes játék után bekerül egy új sor a .txt fájlba azonban nem biztos, hogy a játékban meg fog jelenni mert csak a legjobb 10 pont lesz megjelenítve.

A grafikus megjelenítésért a Swing és a Java AWT fog felelni melyek segítségével fogom létrehozni a pályát a falat meg a karaktereket is.

Drahos Zsolt – UCZFU3



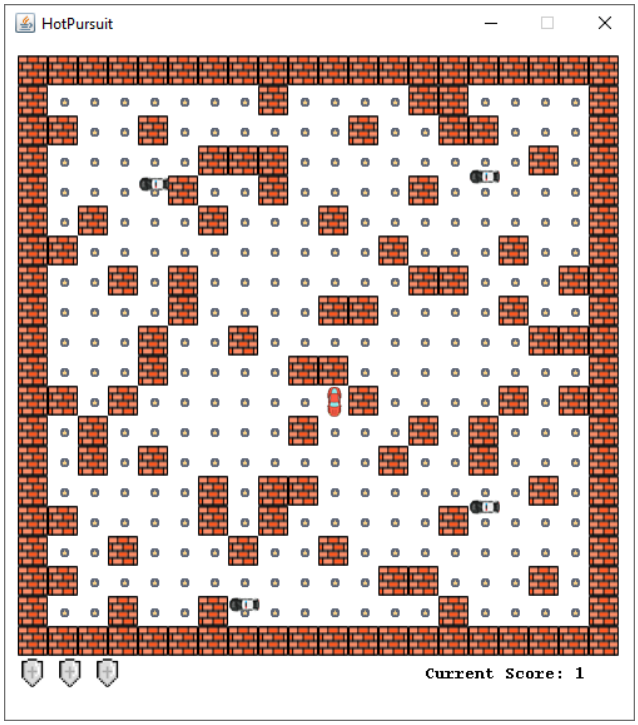
Dokumentáció

A program egy pacmanhez hasonló játékot valósít meg. A különbség, hogy a hagyományos pacmanhez képest nincsnek benne bónusz golyók és a „szellemekből” is csak egy féle van. A program használ alapvető java importokat. A feladatot 3 osztály segítségével valósítottam meg. Az osztályok végleges függvényei az objektum terv fülnél látható. A JUnit teszt pedig a Test1 osztályban jött létre.

A program lényege, hogy a játékos összeszedjen minél több pontot a táblán és ezáltal felkerüljön az eredménytáblára. A játékot a nyilak, space, escape és az 1-3 gombok segítségével lehet vezérelni. A játék a space gomb lenyomásával tud elindulni, azonban még előtte a 1-3 gombok segítségével lehet nehézségi szintet beállítani. Ha ezt nem teszi meg a játékos akkor automatikusan a legelső nehézségi szintet állítja be. Miután elindult a játék, a felhasználó a nyilak segítségével tudja irányítani a karakterét jobbra, balra, fel és le. Továbbá, ha esetleg megunta volna a játékot akkor az ESC billentyűzet segítségével visszatérhet a főmenübe. A játék addig tart amig a játékos el nem veszíti mind a három életét. Ha esetleg összegyűjtötte az össze pontot a pályán akkor a pálya újra kezdődik, de a pontok megmaradnék tovább gyűjtögetheti a pontokat amig meg nem hal. Amint a játéknak vége a pontok kiíródnak az eredménytáblára (csak az első 10) és kezdheti előröl a játékot.

```
21
46
29
31
32
12
11
16
17
97
17
13
16
16
1
19
32
23
51
```

A programban van fájlba írás és olvasás, melyek a highScore.txt fájlt használják. Itt tárolja el a pontokat. A fájlban az adatok sortöréssel vannak elválasztva és csak számok találhatóak meg benne.



A Game osztályban található maga a játék logikája melyben létrejön mapgenerálás és tulajdonképpen a többi osztály meghívása is.

A Draw osztályban történik a kirajzolás és a Move függvények meghívása.

Character osztály felel a megfelelő mozgatható objektumok megjelenítésért (Player/Policeok), egy közös ős a Player és a Polcienak.

A Player osztály a játékos karakteréért felelős. Ez az osztály határozza meg a mozgását, eredeti pozícióját és az ikonjának kirámolását is.

Police osztály pedig a „szellemek” megvalósításáért felelős. Az irányítás és a pozíció meghatározás egyaránt ide tartozik.

Start osztály csak átláthatóság szempontjából került bele, itt található a Main függvény.

A maze generálás 2 szám segítségével jön létre. A 16 szám a csillagot jelenti a maze-en míg a 17-es pedig a falat, amint a player egy olyan mezőre lép ahol van 16os szám (van csillag) akkor azt a mező értékét a tömbben 0-rá változtatva ezáltal tudatva hogy ott felvette a csillagot. Minden egyes újraindításánál a maze alaphelyzetbe áll.

A játékban egy kocka 24*24 pixel, ezt takarja a 24-es szám a program kódban, továbbá a maze egy 20*20-as tömb melyben található csillag és fal is.

Osztályok és azok metódusai részletesen

Game Osztály

Az osztály, ami a játékért logikájáért felelős, itt történik a Character osztály deklarálása

Attributumok:

private boolean	die
private static int	dif
static int	kocka
static int[]	maze
static Character (Player)	p1
public boolean	playing
static Character (Police)	po1, po2, po3, po4,
private java.util.ArrayList<java.lang.Integer>	scores
private static int	size

Konstruktor

Beállítja az alapértelmezett értékeket, inicializálja az attribútumokat.

Tagfüggvények

```
public void collision(Police police1, Player p1)
```

A függvény a Player és a Police ütközését vizsgálja és ha összeütköznek, tehát a két objektum ugyan abban a pozícióban van akkor a „die” attribútum értékét igazra állítja ezáltal tudatva a program többi részével, hogy meghalta a játékos.

Paramétere egy police mert abból több is lehet míg a playerből csak egy van.

```
private  ArrayList<Integer> FileRead()
```

A függvény felel a highScore.txt fájl olvasásáért. Erre azért van szükség, hogy a kezdőképernyőn meg tudjon jelenni a 10 legmagasabb pontszám. Tovább itt történik egy konvertálás melyre azért van szükség mert a Stringként tárolt számokat nem ugyan úgy teszi sorba a sort függvény.

Visszatért az így keletkezett listával.

```
private void FileWrite(ArrayList<Integer> scores2)
```

A függvény lényege, hogy be tudjuk olvasni a highScore.txt fájlt melyben a pontszámok vannak eltárolva. A beolvasáshoz ArrayListet és FileWriter-t használ a függvény és minden írás után tesz egy sortöést.

Paramétere egy ArrayList ami általában a beolvasott lista szokott lenni.

```
private void Maze_check()
```

A függvény feladata, hogy meghatározza, hogy hány darab felszedhető pont van a pályán és meghatározza a maximális pontot aminek megszerzésével újratöltődés a pálya.

```
public void death(Graphics g2, Player p1)
```

A függvény a „halálért” felel. Ha a játékos meghal akkor itt veszít egy élete és vizsgálja, hogy ha az élet egyenlő 0-val akkor vége van a játéknak és elmenti a pontot a fájlba.

Paramétere a Graphics2D osztály mely segítségével tudunk grafikai rutinokat használni. és a játékos osztály (tesztelés miatt kellett).

```
public void SetGame(int difficulties, Player p, Police pol1, Police pol2)
```

A függvény a játék 3 nehézségi szintjét állítja be. Az első szinten (ami az alapértelmezett is) beállítja a 3 életet és megjeleni 4 rendőr. A második szinten már 6 rendőr van és ezek sebessége meg lett növelve. És van a harmadik ahol pedig a játékos sebessége van feljebb véve ezáltal nehezítve a közlekedést.

Paramétere a `difficulties`. és a Player meg a Police értékek (tesztelés miatt)

```
public void LoadMaze()
```

A függvény feladata a labirintus legenerálása, ez jelen esetben egy ciklussal és különböző feltételekkel történik mert a Random map generátor sajnos még nem tökéletes és ott generálódik olyan eset is amit nem lehet kijátszani.

```
public void Restart(Police testmiatt)
```

A függvény a Player és a Police Restart függvényeit hívja meg és állítja a játékot alaphelyzetbe.

```
public void keyPressed(KeyEvent e)
```

A függvény minden egyes gomblenyomásnál meghívódik és megadott gomboknál különböző események történnek. Itt történik az irányítás meghatározása nyilakkal és a 1-3 gombokkal való nehézség beállítása.

```
Továbbá egy két setter és getter (dif, die)
```

Draw Osztály

Az osztály, ami a játékért megjelenítéséért és (logikájáért) felelős.

Attributumok:

private Image	playerup, playerdown, playerleft, playerright, life, star, start, brick, police
private Timer	timer
private java.util.ArrayList<java.lang.Integer>	sc

Konstruktor

Beállítja az alapértelmezett értékeket, inicializálja az attribútumokat.

Tagfüggvények

```
public void Screen(Graphics g2)
```

A függvény segítségével jeleníti meg a start oldalon az adatokat, képeket, scoreboardot, továbbá itt határozza meg a scoreboard 1-3 elemét hogy annak különböző szöveget tudjon adni.

Paramétere a Graphics osztály mely segítségével tud grafikai rutinokat használni.

```
private void DrawGameMethod (Graphics g2)
```

A függvény felel a player és a police osztály kirajzoló függvényének meghívásért. Továbbá itt történik a `collision` függvény meghívás is. Ezen felül itt hívódik meg az labirintus rajzoló függvény is és itt vizsgálódik az is, hogy a die igaza akkor meghívja a death függvényt, ha pedig nem és nem játszik akkor pedig megjelenítőik a kezdőképernyő (Screen).

Paramétere a Graphics2D osztály mely segítségével tudunk grafikai rutinokat használni egy picit bonyolultabban, mint a Graphics-al.

```
private void ScoreDraw(Graphics g2)
```

A függvény feladat az aktuális pont megjelenítése a képernyőn. Tulajdonképpen csak kirajzolja a mindig az aktuális pontot a képernyőre.

Paramétere a Graphics2D osztály mely segítségével tudunk grafikai rutinokat használni.

```
private void MazeDraw(Graphics g2)
```

A függvény a labirintust rajzolja ki a képernyőre. A blokkok egy kép ami 24*24 mértre le vannak korlátozva mert akkora egy blokk. Továbbá a csillagok is egy kép ami pedig a blokk közepén jelenik meg. Ennek megvalósításhoz 2 ciklust használtam amik elmennek a az egész labirintus méretéig és 24 picelenként rajzolnak x és y koordináta alapján.

Paramétere a Graphics2D osztály mely segítségével tudunk grafikai rutinokat használni.

```
public void paintComponent(Graphics g)
```

Egy automatikusan generált függvény melyben hatódik végre a Draw függvény és jelenítődik meg minden a képernyőn.

```
public void actionPerformed(ActionEvent e)
```

A függvény feladata a timer segítségével jelen esetben 35 ms-enként újra rajzolja a komponenseket ezáltal biztosítva a folyamatos mozgást a játékban

Start Osztály

A main függvény található meg benne, nem volt mindenképpen szükség rá de így jobban átláthatóak a dolgok

Tagfüggvények

```
public static void main(String[] args)
```

A main függvény, ahol szimplán meg van hívva a Game osztály és létre van hozva a JFrame ahol meg fog minden jelenni. Továbbá itt van beállítva az ablak mérete és az is, hogy ne lehessen átméretezni.

Character osztály

A Player és a Police osztály őse.

Attributumok:

private static char	dir
private static int	dx, dy
private int	life,
private int	p_dx, p_dy
static int	PACMAN_SPEED
private int	SPEED
private static long	serialVersionUID
private static int	x, y

Konstruktor

```
public Character()
```

Tagfüggvények:

```
private int getDirectionX(char d)
```

A függvény lényege, hogy meghatározza a paraméterként kapott karakterből, hogy milyen koordinátát adjon neki. és visszatér a x koordinátával

Paraméter char d

Return a karakter megfelelő irányának x koordinátájával

```
private int getDirectionY(char d)
```

Ugyan az, mint az előző csak a Y koordinátával tér vissza.

```
public boolean IsOnField()
```

Megvizsgálja, hogy rajta van-e egy kockán, az x koordinátát elosztja 24 pixellel és ha az maradékosan osztva nulla és az y-is akkor rajta van, különben nem.
Return true ha rajta van.

```
public int CurrentPos(char d)
```

Meghatározza a falat attól függően melyik irányba van.

Paraméter d irány

```
public void restart()
```

Lényege, hogy visszaállítja az alapértelmezett értékbe a Charactert.

Getterek és setterek minden egyes privát adattaghoz, hogy azokat elérjük és értéket tudjunk adni nekik máshonnan.

Player osztály a Character osztály leszármazottja

Attributumok:

`private int`	**`life`**
`private int`	**`score`**

Konstruktor

```
public Player()
```

Tagfüggvények:

```
public void move ()
```

A karakter irányításáért felel, ha nincs fal az irányába amerre megy akkor megy tovább, ha pedig falba ütközik akkor megáll. Továbbá beállítja az x és y koordinátát az aktuális irány és a sebesség segítségével.

Override-ra azért van szükség mert a Move függény a Playerben és a Policeban is máshogy történik ezért ilyenkor mindig a megfelelő függvény hívódik meg.

```
public void PlayerDraw(Graphics g2, Player player, Image playerleft, Image playerright, Image playerup, Image playerdown, ImageObserver a)
```

Kirajzolja a Playert az iránynak megfelelően.

Paraméter Graphics2D mely segítségével tudunk grafikai rutinokat használni, a képek, amiket felhasznál, és az ImageObserver.

Override-ra azért van szükség mert a Draw függény a Playerben és a Policeban is máshogy történik ezért ilyenkor mindig a megfelelő függvény hívódik meg.

Továbbá vannak különféle getter-ek meg setterek. (Eclipse által generált függvények).

Police osztály a Character osztály leszármazottja

Tagfüggvények:

```
private char Direction(int x, int y)
```

Meghatározza az irányt és visszatér egy char-al ami segítségével könnyebben lehet majd meghatározni, hogy hol van fal és hol nincs.

```
public void move ()
```

A rendőrök irányításáért felel, ha nincs fal az irányába amerre megy akkor megy tovább, ha pedig falba ütközik akkor megáll. Az irányt egy random generátor segítségével kapja, ezáltal a rendőrök mozgása teljesen random. Továbbá beállítja az x és y koordinátát az aktuális irány és a sebesség segítségével.

Override-ra azért van szükség mert a Move függény a Playerben és a Policeban is máshogy történik ezért ilyenkor mindig a megfelelő függvény hívódik meg.

```
public void PoliceDraw(Graphics g2, int x, int y, Image police, ImageObserver a)
```

Kirajzolja a Police-t a megfelelő koordinátára.

Paraméter Graphics2D mely segítségével tudunk grafikai rutinokat használni, a képek amiket felhasznál, és az ImageObserver.

Fájlok

képek: HotPursuit/src/hotpursuit/images/*.png

java fájlok: HotPursuit/src/hotpursuit/*java

txt fájl: HotPursuit/highScore.txt

Tesztek

A program ellenőrzésére, 12 darab tesztet hoztam létre, melyekből 10 releváns a program működés szempontjából.

```
public void test_playerSetter()
```

A teszt lényeg, hogy megvizsgálja a Character osztály getter és setter metódusait.

```
public void test_player_pos()
```

A teszt lényeg, hogy megvizsgálja, hogy megfélő értéket ad-e vissza különböző irányoknál, és helyen határozza meg a pozíciót.

```
public void test_player_restart()
```

A teszt lényeg, hogy megfelelő helyre kerül-e vissza a karakter a kezdéskor/újraindításkor/halálkor.

```
public void test_police_pos()
```

A teszt lényeg, hogy megvizsgálja, hogy megfélő értéket ad-e vissza különböző irányoknál, és helyen határozza meg a pozíciót.

```
public void test_police_restart()
```

A teszt lényeg, hogy megfelelő helyre kerül-e vissza a rendőr a kezdéskor/újraindításkor/halálkor.

```
public void test_move()
```

A teszt a Character mozgását teszteli, hogy különböző irányokban megfelelő pozíciót vesz e fel.

```
public void test_collision()
```

A teszt a Player és Police összeütközését vizsgálja, ha összeütköznek akkor die igaz lesz.

```
public void test_death()
```

A teszt player halálának megvizsgálásáért felel, ha meghal akkor az élete mindig csökken eggyel.

```
public void test_dif_set()
```

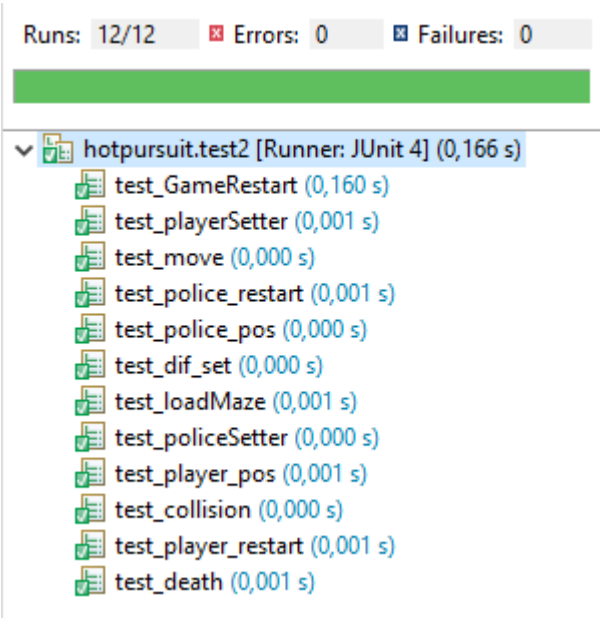
A teszt a nehézségi szintek beállításáért felel.

```
public void test_loadMaze()
```

A teszt a pálya legenerálásáért felel.

```
public void test_GameRestart()
```

A teszt lényeg, hogy visszaálljon minden alapértelmezettbe ha elindul/ vége a játéknak.



Minden teszt sikeresen lefut!