SpaceBattle - specifikáció

1 A JÁTÉKRÓL

Ez egy stratégiai játék, ahol a számítógép ellen lehet játszani. A játék célja, hogy elfoglaljuk az összes bolygót a galaxisban.

Eleinte kis kolóniával kezdünk, ahol lassan növekedni kezdenek az egységeink. A zöld bolygó az általunk elfoglalt terület, ahol gyarapodnak egységeink.

Az ellenséges bolygót a piros szín jelöli, ez a számítógép kolóniája, aki szintén folyamatosan fejlődik és próbálja megszerezni a számára kedvező bolygókat.

Végül vannak a semleges bolygók, melyeket még nem ural semmilyen nemzedék. Ezek a területek is tartalmaznak ellenálló seregeket, de azok nem fejlődnek és nem is támadnak senkit, Ők semlegesek.

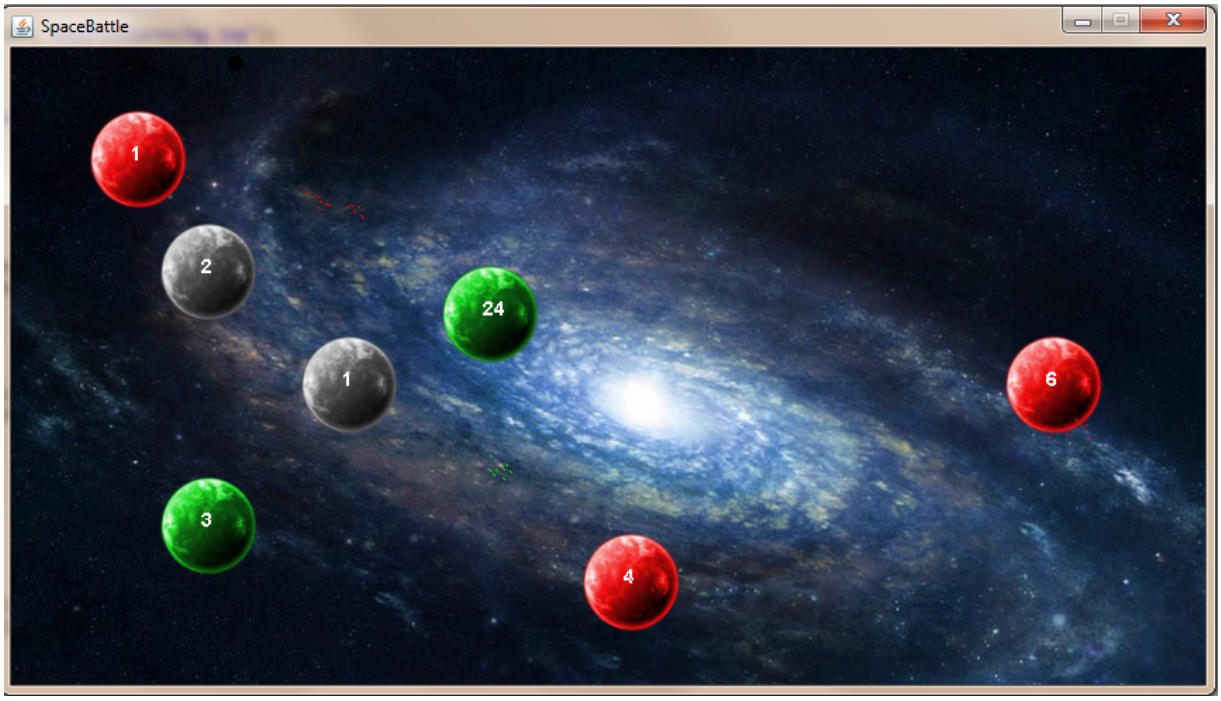
Ha ilyen mezőt foglalunk el, akkor szintén megszerezzük a bolygót és a mi kolóniánk fog ott növekedni, ezzel erősítve az egységünket.

Nagyon fontos a játékban a taktika, hogy mely bolygókat foglaljuk el elsőként és melyeket hagyjuk a végére, illetve milyen összhangban indítjuk a támadásokat az ellenséges bolygókra, ugyanis képesek vagyunk egyszerre több bolygót is kijelölni és az ott tartózkodó egységek egy részével egyszerre támadni.

A támadás úgy néz ki, hogy a meglévő kolóniánk fele hagyja el a bolygót és indul a csatába.

A számítógép is hasonlóan cselekszik, ő is képes egyszerre több bázisából indítani támadó egységeket felénk.

Ha egy bolygót elfoglalunk, akkor ott a mi kolóniánk fog gyarapodni, de ha a csata egál, tehát pont úgy jött ki az egységek száma, hogy egymást kivégzik, akkor a bolygó semlegessé válik, és nem fog egység gyarapodni ott. Ezekre a bolygókra érdemes gyorsan újabb támadást indítani, mert könnyen megszerezhető és ezzel újabb egységeket tudunk építeni.



A képen látható az ellenséges egységek illetve a szövetséges egységek mozgása. Továbbá van még két semleges fehér bolygó, melyek még nem képeznek támadó egységeket.

2 USE-CASE

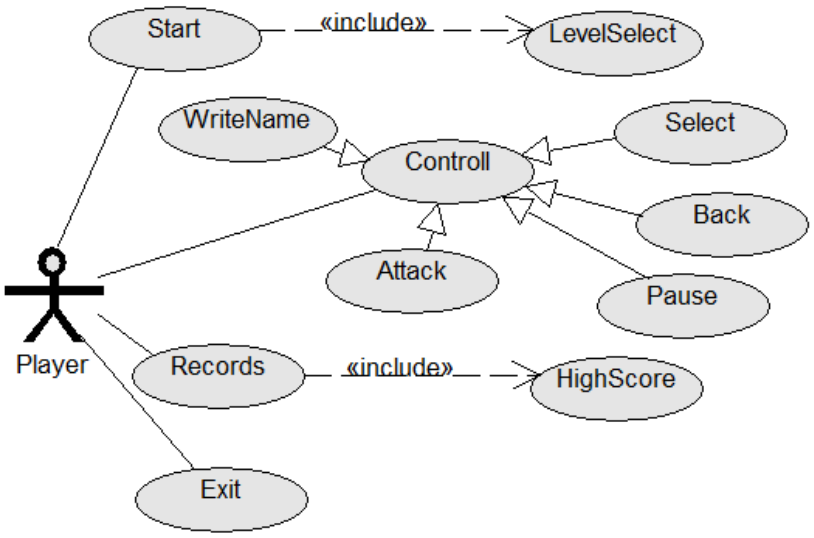
Egér

- Ball gomb - Kijelöli a bolygót ezzel felkészítve a támadásra.
- Jobb gomb – A kolóniákat támadásnak indítja a rákattintott bolygóra.

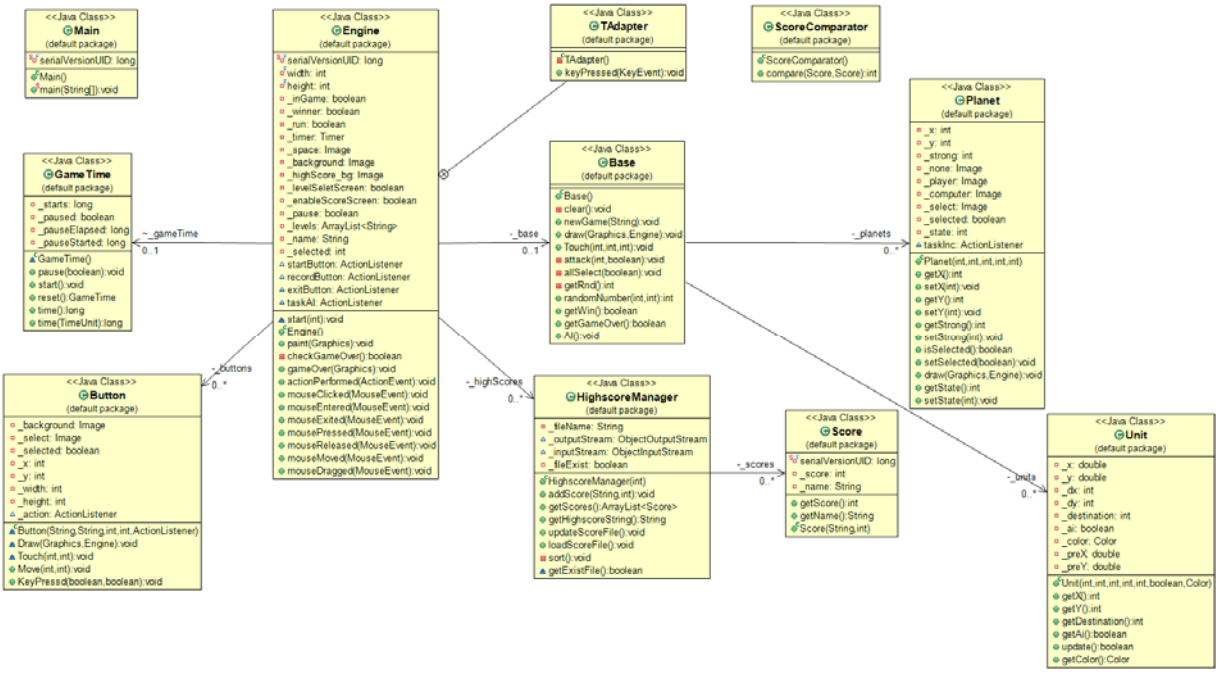
Billentyű

- Menüben/pályaválasztóban/rekord listában a nyilakkal tudunk navigálni
- Enter gomb az aktuális menüpontot kiválasztja
- P gomb lenyomása szünetelteti a játékmenetet

További use-case-ek később fognak belekerülni, mikor elkészül a mobilos változata.

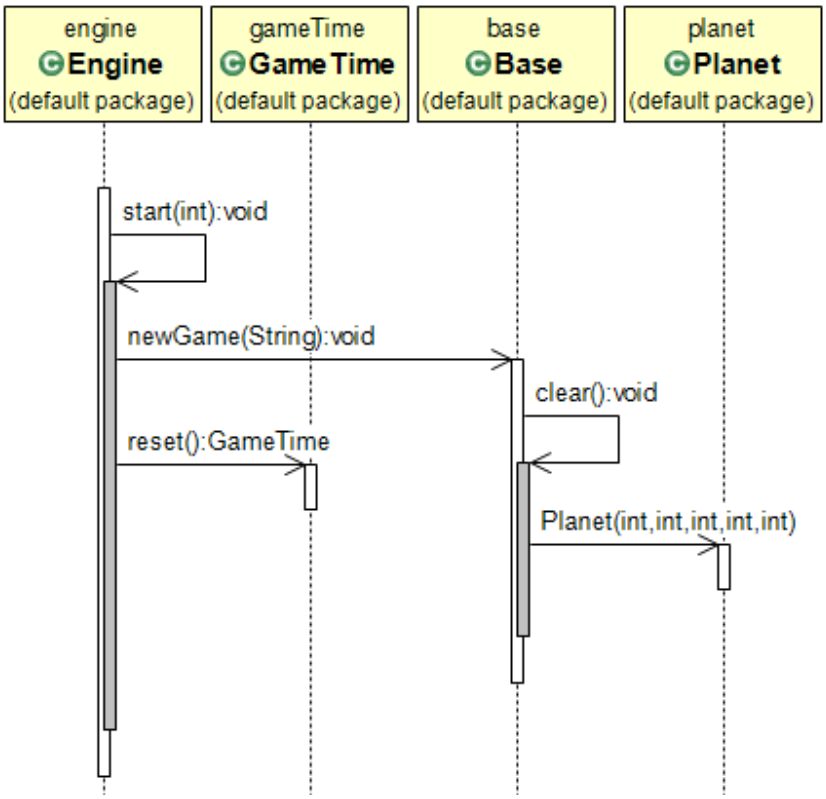


3 UML

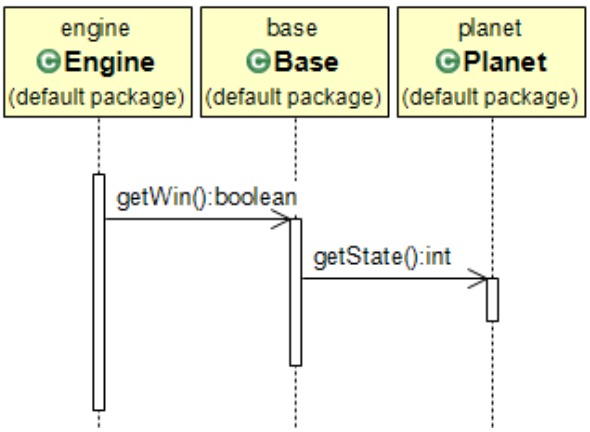

<<Java Class>>
Main
(default package)
serialVersionUID: long
Main()
main(String[]):void
<<Java Class>>
GameTime
(default package)
_starts: long
_paused: boolean
_pauseElapsed: long
_pauseStarted: long
GameTime()
pause(boolean):void
start():void
reset():GameTime
time():long
time(TimeUnit):long
<<Java Class>>
Button
(default package)
_background: Image
_select: Image
_selected: boolean
_x: int
_y: int
_width: int
_height: int
_action: ActionListener
Button(String,String,int,int,ActionListener)
Draw(Graphics,Engine):void
Touch(int,int):void
Move(int,int):void
KeyPressd(boolean,boolean):void
<<Java Class>>
Engine
(default package)
serialVersionUID: long
width: int
height: int
_inGame: boolean
_winner: boolean
_run: boolean
_timer: Timer
_space: Image
_background: Image
_highScore_bg: Image
_levelSeletScreen: boolean
_enableScoreScreen: boolean
_pause: boolean
_levels: ArrayList<String>
_name: String
_selected: int
startButton: ActionListener
recordButton: ActionListener
exitButton: ActionListener
taskAI: ActionListener
start(int):void
Engine()
paint(Graphics):void
checkGameOver():boolean
gameOver(Graphics):void
actionPerformed(ActionEvent):void
mouseClicked(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
mouseMoved(MouseEvent):void
mouseDragged(MouseEvent):void
<<Java Class>>
TAdapter
(default package)
TAdapter()
keyPressed(KeyEvent):void
<<Java Class>>
ScoreComparator
(default package)
ScoreComparator()
compare(Score,Score):int
<<Java Class>>
Base
(default package)
Base()
clear():void
newGame(String):void
draw(Graphics,Engine):void
Touch(int,int,int):void
attack(int,boolean):void
allSelect(boolean):void
getRnd():int
randomNumber(int,int):int
getWin():boolean
getGameOver():boolean
AI():void
<<Java Class>>
Planet
(default package)
_x: int
_y: int
_strong: int
_none: Image
_player: Image
_computer: Image
_select: Image
_selected: boolean
_state: int
taskInc: ActionListener
Planet(int,int,int,int,int)
getX():int
setX(int):void
getY():int
setY(int):void
getStrong():int
setStrong(int):void
isSelected():boolean
setSelected(boolean):void
draw(Graphics,Engine):void
getState():int
setState(int):void
<<Java Class>>
HighscoreManager
(default package)
_fileName: String
_outputStream: ObjectOutputStream
_inputStream: ObjectInputStream
_fileExist: boolean
HighscoreManager(int)
addScore(String,int):void
getScores():ArrayList<Score>
getHighscoreString():String
updateScoreFile():void
loadScoreFile():void
sort():void
getExistFile():boolean
<<Java Class>>
Score
(default package)
serialVersionUID: long
_score: int
_name: String
getScore():int
getName():String
Score(String,int)
<<Java Class>>
Unit
(default package)
_x: double
_y: double
_dx: int
_dy: int
_destination: int
_ai: boolean
_color: Color
_preX: double
_preY: double
Unit(int,int,int,int,int,int,boolean,Color)
getX():int
getY():int
getDestination():int
getAi():boolean
update():boolean
getColor():Color
-_gameTime
0..1
-_base
0..1
-_planets
0..*
-_buttons
0..*
-_highScores
0..*
-_scores
0..*
-_units
0..*

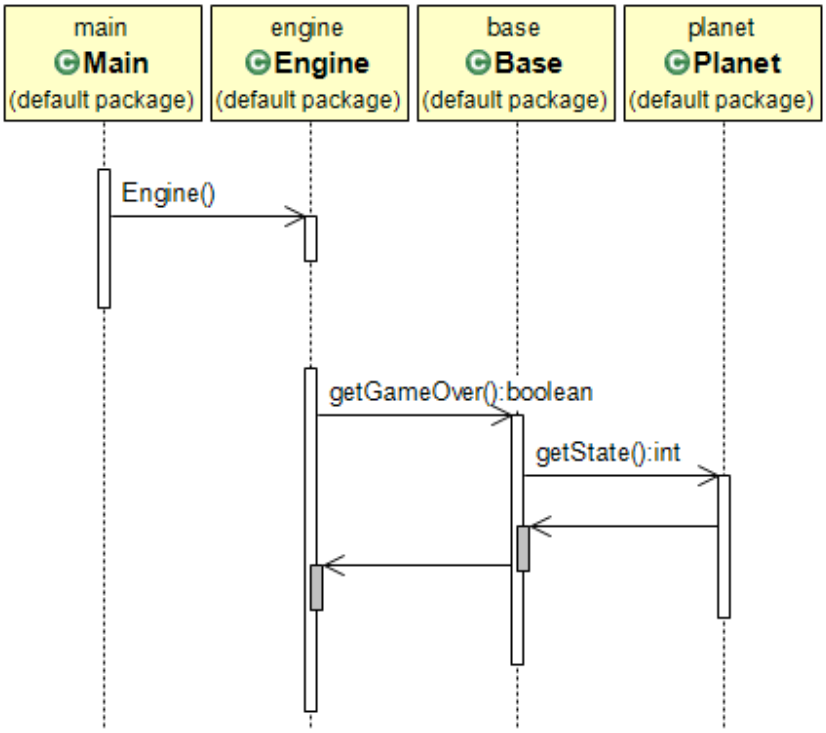
4 SEQUENCE DIAGRAM

Pálya betöltésének menete. A start() függvény átveszi a betöltendő pálya indexét, aztán meghívódik a newGame() függvény, ami betölti a pályát, törli az aktuális jelenetben szereplő tagokat és hozzáadja a bolygókat a jelenethez, majd a reset() függvény fut le, ami az időt indítja el.



Lekérdezzük, hogy nyertünk-e vagy sem. Ennél a getWin() metódust kell meghívni, ami végignézi az összes bolygót, hogy ki birtokolja getState() és ha mindet a játékos birtokolja, akkor nyertünk.





5 MEGVALÓSÍTÁS

```
public class Engine extends JPanel implements ActionListener, MouseListener {
...
}
```

A grafikai elemeket a Graphics osztály segítségével valósítottam meg. Ezzel rajzolom ki a képernyőre a hátteret, bolygókat, támadó egységeket, egységek számát, valamit az értesítő szövegeket. Játék logikájának is itt valósítom meg.

```
    private boolean _inGame = false; // játékban, ez akkor igaz, ha nem a menüben vagyunk
    private boolean _winner = false; // ha nyertünk, akkor az értéke igaz lesz
    private boolean _run    = false; // folyamatban van a játék
    private Base _base;              // ez tartalmazza a bolygókat

    private boolean _levelSeletScreen = false;      // pályaválasztó képernyő aktív
    private boolean _enableScoreScreen = false;     // rekord lista képernyő aktív
    private boolean _pause = false;                 // játék szüneteltetése

    private ArrayList<Button> _buttons;             // gombok a menüben
    private ArrayList<String> _levels;              // pályákat tartalmazó lista
    private String _name;                           // játékos nevét tároló string
    private int _selected = 0;                      // kiválasztott menüpont

    private ArrayList<HighscoreManager> _highScores;       // rekord lista

    GameTime _gameTime = new GameTime();            // játék idejének mérése
```

Játék indítása

```
void start(int index){
      _inGame = true;
      _levelSeletScreen = false;

      _base.newGame("base/configs/level"+index+".map");
      _gameTime.reset();
 }
```

AI cselekvése az időzítő szerint

```
new Timer(5000, taskAI).start();
  5000ms-enként lefuttatja az AI taskot. Ez a gép cselekvése.

ActionListener taskAI = new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
    if(_run)
            _base.AI();
    }
};
```

Kép betöltése

```
ImageIcon imageBg = new ImageIcon("base/textures/bg.jpg");
bg = imageBg.getImage();
```

Kirajzolás metódus

```
public void paint(Graphics g) {
        super.paint(g);

        if (inGame) {

             g.drawImage(bg, 0, 0, this);
             base.draw(g,this);

             checkGameOver();

        } else {

            gameOver(g);

        }
}
```

Az osztály figyeli az egér/billentyű beviteli eszközök feldolgozását.

Beviteli eszközök

```
public void keyPressed(KeyEvent e) {
          int key = e.getKeyCode();

          if ((key == KeyEvent.VK_ENTER) && !inGame ) {
           if(!inGame && index == 4)
                 System.exit(0);

           if(winner)
                 base.newGame("base/configs/level"+ (++index) + ".map");
           else
                 base.newGame("base/configs/level"+ index + ".map");

           inGame = true;
          }
      }
  }
```

```
public class Base {

...

}
```

A Base osztály tartalmazza a bázisokat, más néven bolygókat, melyeket egy txt típusú fájlból olvasok be az alábbi formátumban:

Bolygó tulajdonosa | X | Y | egységek száma

level4.map

Player 50 40 29
Player 100 300 15
Computer 700 200 25
Computer 400 340 20
None 200 200 1
None 100 120 2
None 300 150 1

Beolvasás menete

```
while ((strLine = br.readLine()) != null) {
	String[] words = strLine.split(" ");

	if (words[0].equals("None")) {
		planets.add(new Planet(0,Integer.parseInt(words[1]),
		Integer.parseInt(words[2]), Integer.parseInt(words[3])));
	} else if (words[0].equals("Player")) {
		planets.add(new Planet(1,Integer.parseInt(words[1]),
		Integer.parseInt(words[2]), Integer.parseInt(words[3])));
	} else if (words[0].equals("Computer")) {
		planets.add(new Planet(2,Integer.parseInt(words[1]),
		Integer.parseInt(words[2]), Integer.parseInt(words[3])));
	}
}
```

```
public class Planet {

...

}
```

A bolygókat a Planet osztály valósítja meg, mely tartalmazza a bázis pozícióját és típusát, valamint az ott lévő egységek számát.

Konstruktor

```
public Planet(int state, int x, int y, int strong) {
	this.setState(state);

	this.x = x;
	this.y = y;
	this.strong = strong;

	none = new ImageIcon("base/textures/planetW.png").getImage();
	player = new ImageIcon("base/textures/planetG.png").getImage();
	computer = new ImageIcon("base/textures/planetR.png").getImage();

	ImageIcon selImg = new ImageIcon("base/textures/planetS.png");
	select = selImg.getImage();
}
```

```
public class Unit{

...

}
```

A Unit osztály egy darab egységet valósít meg. Az egység tartalmaz egy x és y koordinátát, ez lesz az ő kezdő helye, ahol megszületik, illetve tartalmaz egy cél állomást is, amit szintén x és y koordinátákban tárolja, hogy hova tart az egység.

```
this.x = x; // source X
this.y = y; // source Y

this.dx = dx; // destination X
this.dy = dy; // destination Y
```

Az egység addig mozog, amíg el nem éri a célállomást, ezt követően becsapódik és elvégzi feladatát.

Egységek frissítése

```
public boolean update() {

	double rtx = 1;
	double rty = 1;

	double distx = dx-x;
	double disty = dy-y;
	double rate = Math.abs(distx/disty);

	if( distx >= -10 && distx <=10 &&
		disty >= -10 && disty <=10){
		return false;
	}

	rtx = Math.sqrt( (8)/(1+rate*rate) );
	rty = rtx;

	if(x < dx){
		x +=rtx*rate;
	} else if(x > dx){
		x -=rtx*rate;
	}

	if(y < dy){
		y +=rty;
	} else if(y > dy){
		y -=rty;
	}

	return true;
```

```
public class GameTime{

...

}
```

Játék idejének mérése, ez elengedhetetlen a highScore lista felállításához.

```
private long _starts;
private boolean _paused;
private long _pauseElapsed;
private long _pauseStarted;
```

idő lekérdezése

Úgy kapjuk meg az időt, hogy az aktuális időből levonjuk a kezdéskor rögzített időt, valamint a szünetben eltöltött időt.

```
public long time() {
      long ends = System.currentTimeMillis();

      return ends - _starts - _pauseElapsed;
    }
```

```
public class HighScoreManager{

...

}
```

Ez az osztály felel a rekordok mentésére/betöltésére. Le tudjuk kérdezni a listában szereplő játékosok pontszámát, és rendezni tudjuk a ranglistát pontok szerint.

Lista rendezése saját komparátorral

```
private void sort() {
            ScoreComparator comparator = new ScoreComparator();
            Collections.sort(_scores, comparator);
}
```

```
public class Score  implements Serializable {
	private int _score;
	private String _name;
}
```

Ez az osztály tárolja a pontszámot és a hozzá tartozó nevet.
A ScoreComparator rendezi a pontszámokat.

```
public class ScoreComparator implements Comparator<Score> {
	public int compare(Score score1, Score score2) {

		int sc1 = score1.getScore();
		int sc2 = score2.getScore();

		if (sc1 > sc2){
			return +1;
		}else if (sc1 < sc2){
			return -1;
		}else{
			return 0;
		}
	}
}
```

```
public class Button {

	private Image _background;
	private Image _select;
	private boolean _selected;

	ActionListener _action;

}
```

Konstruktor

- Átvesszük a gomb pozicióját x,y értékkekkel
- Gomb textúra
- Akció, ami akkor fut le, mikor megnyomjuk a gombot

```
	Button(String base_texture_name, String select_texture_name,
		int x, int y, ActionListener action){

	ImageIcon base_texture = new ImageIcon(base_texture_name);
	 _background = base_texture.getImage();

	 ImageIcon select_texture = new ImageIcon(select_texture_name);
	 _select = select_texture.getImage();

	 _selected = false;

	 _x = x;
	 _y = y;
	 _width = base_texture.getIconWidth();
	 _height = base_texture.getIconHeight();
	 _action = action;
	}
```

Kirajzolás

```
	void Draw(Graphics g, Engine engine){
		if(_selected)
			g.drawImage(_select, _x, _y, engine);
		else
			g.drawImage(_background, _x, _y, engine);
	}
```

Touch esemény

```
	void Touch(int x, int y) {

		if(x>_x && x< _x+_width &&
		   y>_y && y< _y+_height){

			ActionEvent e = new ActionEvent(this, 0, "click");
			_action.actionPerformed(e);
		}
	}
```

6 KÖVETELMÉNYVIZSGÁLAT

6.1.1 Kötelező funkcionalitások

- Swing-alapú GUI
  - vagy JTable, JTree, JComboBox valamelyikének alkalmazásával, menüvel
  - vagy alacsonyszintű grafikai rutinok (Graphics osztály) használatával
- Gyűjtemény keretrendszer alkalmazása
  - ArrayList<Planet> bases; // bolygók tárolása
- Fájlba írás, fájlból olvasás szerializálás segítségével
  - Játék mentése/betöltése
- Tesztelés-támogatás (JUnit)
  - Legalább 2 osztály összesen 5 metódusának tesztelése

Kész

Még hátra van

ÖSSZEGZÉS

Az engine *osztály vezérli a játék logikáját, és hívja meg a kirajzolásért felelős metódusokat, valamint az irányítás feldolgozása is itt történik meg.*

A base *osztály tartalmazza a bolygókat (* Planet *) ,ami három típusú lehet (none,player,computer). Itt történik meg az AI cselekvése, aki random választ egy ellenfelet és megpróbálja elfoglalni a területet.*
A támadó egységeket is a base *osztály tárolja egy listába (* Unit *), amik addig mozognak, míg el nem érik a célpontot.*

Button *osztály valósítja meg a menüben található menüpontokat.*

A GameTime *felel a játék idejének mérésére.*

A HighScoreManager *tölti és menti a rekord listánkat szerializálással. Képes új adatot felvenni a listába és rendezni, valamit le lehet kérni tőle az aktuális pályához tartozó ranglistát pontszám és név szerint.*

7 TOVÁBBFEJLESZTÉS

Az alkalmazás tovább lesz fejlesztve mobil platformokra is!

Elérhető lesz Windows Phone, MeeGo készülékekre, illetve esetleg Android rendszerekre is.

A project oldala: http://irsoftware.darktl.com