PROJEKT ASTEROIDY

OBJEKTOVÝ PRÍSTUP K RIEŠENIU PROBLÉMOV

[MATERIÁL PRE ŠTUDENTOV]

TERÉZIA SLIACKA
JÁN KUČERA
MICHAL VARGA
NORBERT ADAMKO

EURÓPSKA ÚNIA
Európsky sociálny fond
Európsky fond regionálneho rozvoja

OPERAČNÝ PROGRAM
ĽUDSKÉ ZDROJE

MINISTERSTVO
ŠKOLSTVA, VEDY,
VÝSKUMU A ŠPORTU
SLOVENSKEJ REPUBLIKY

it akadémia

Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu
a Európskeho fondu regionálneho rozvoja v rámci Operačného programu Ľudské zdroje

www.minedu.sk www.employment.gov.sk/sk/esf/ www.itakademia.sk

Projekt Asteroidy

Objektový prístup k riešeniu problémov

[Materiál pre študentov]

Spracované v rámci národného projektu IT Akadémia – vzdelávanie pre 21. storočie

Žilina, 2021

Projekt Asteroidy - Objektový prístup k riešeniu problémov [Materiál pre študentov]
Spracované s finančnou podporou národného projektu IT Akadémia – vzdelávanie pre 21. storočie

Autori: Terézia Sliacka, Ján Kučera, Michal Varga, Norbert Adamko

Neprešlo jazykovou úpravou.

Vydavateľ: Žilinská univerzita v Žiline
Rok vydania: 2021
Vydanie: 1.

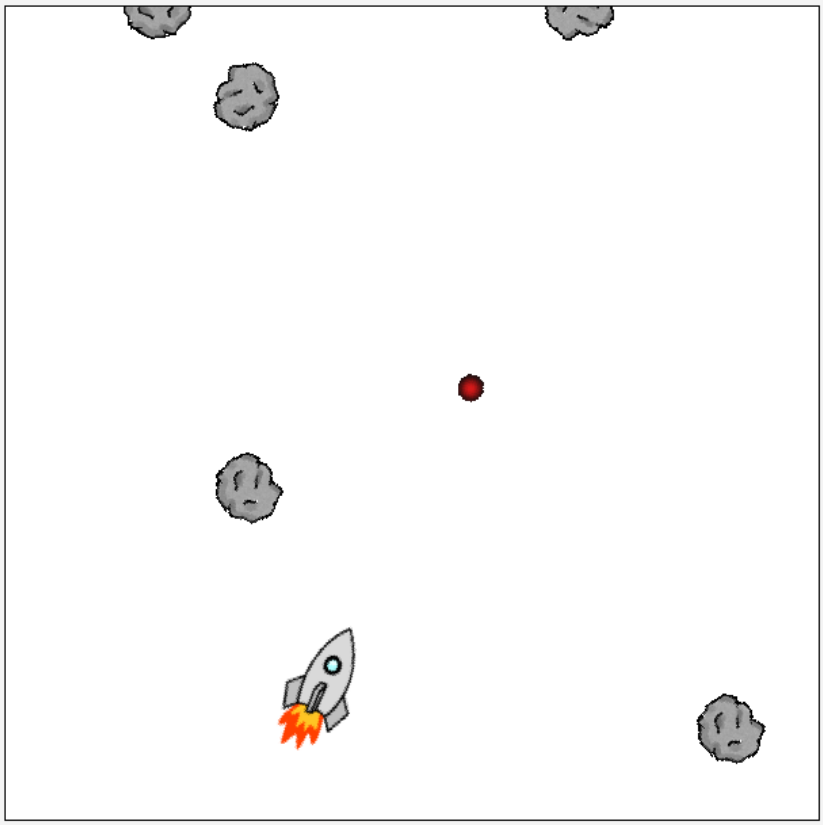
Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu v rámci Operačného programu Ľudské zdroje.

OBSAH

ÚVOD

POPIS A PRAVIDLÁ HRY

Hra *Asteroidy* je určená pre jedného hráča. Hráč ovláda jednu vesmírnu loď okolo ktorej lietajú asteroidy. Cieľom hry je zabrániť nárazu vesmírnej lode s asteroidom a zároveň ničiť asteroidy pomocou striel. Hráč hru vyhrá, ak sa mu podarí zostreliť všetky asteroidy. V prípade, že dôjde ku kolízii vesmírnej lode s niektorým z asteroidov, hra končí neúspechom.



Obrázok 1 - Ukážka hry Asteroidy v prostredí Greenfoot

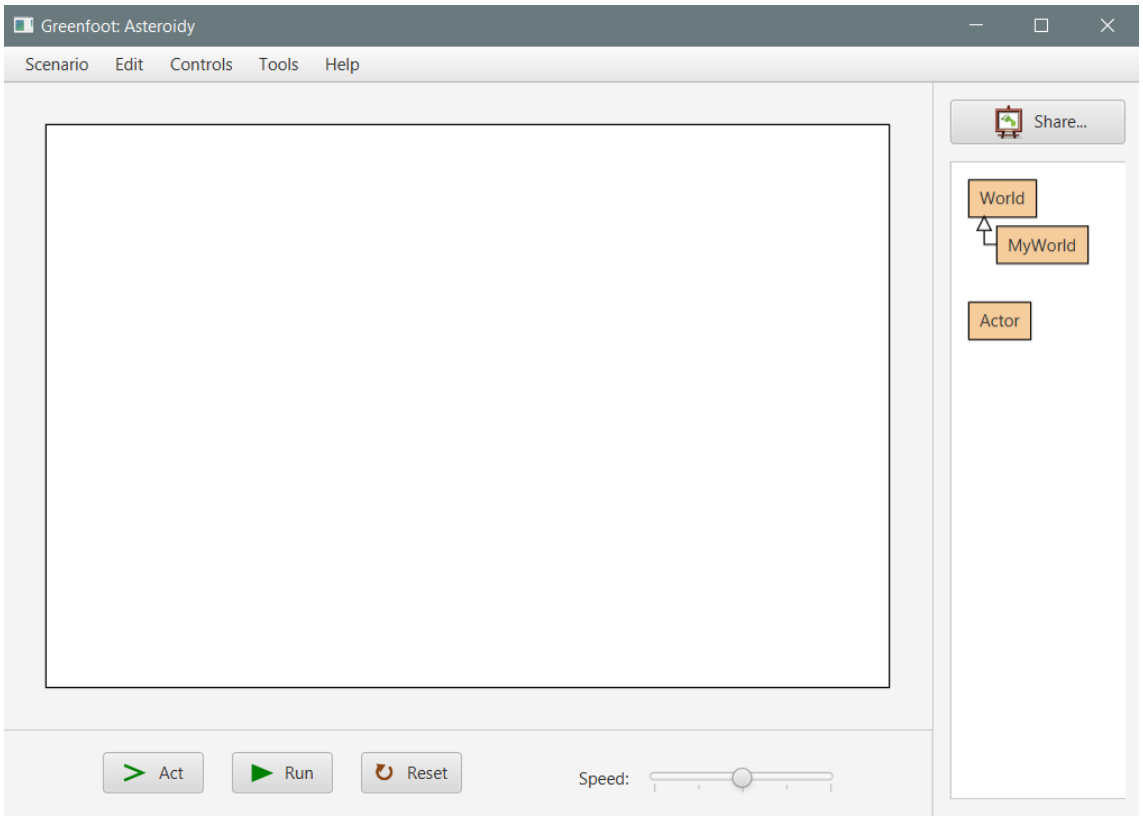
1 VESMÍRNA LOĎ

V prvom kroku vytvoríme základ hry – hracie pole a vesmírnu loď, ktorú sa neskôr naučíme ovládať. Po zvládnutí tejto časti by ste mali v prostredí Greenfoot vidieť vesmírnu hraciu plochu a uprostred nej vesmírnu loď.

ÚLOHA 1.1

Vytvorte nový **Java Scenár**, ktorý bude zastrešovať našu hru. Pomenujte ho *Asteroidy*, aby jeho názov jasne identifikoval hru, ktorú vytvárate. Následne zvoľte vhodnú zložku, do ktorej projekt uložíte. Po vytvorení bude projekt obsahovať triedy **`World`**, **`Actor`** a **`MyWorld`**.

V nástroji Greenfoot vyberte z menu **Scenario** položku **New Java Scenario**.
Name: Asteroidy
Location: *umiestnenie, ktoré uznáte za vhodné.*

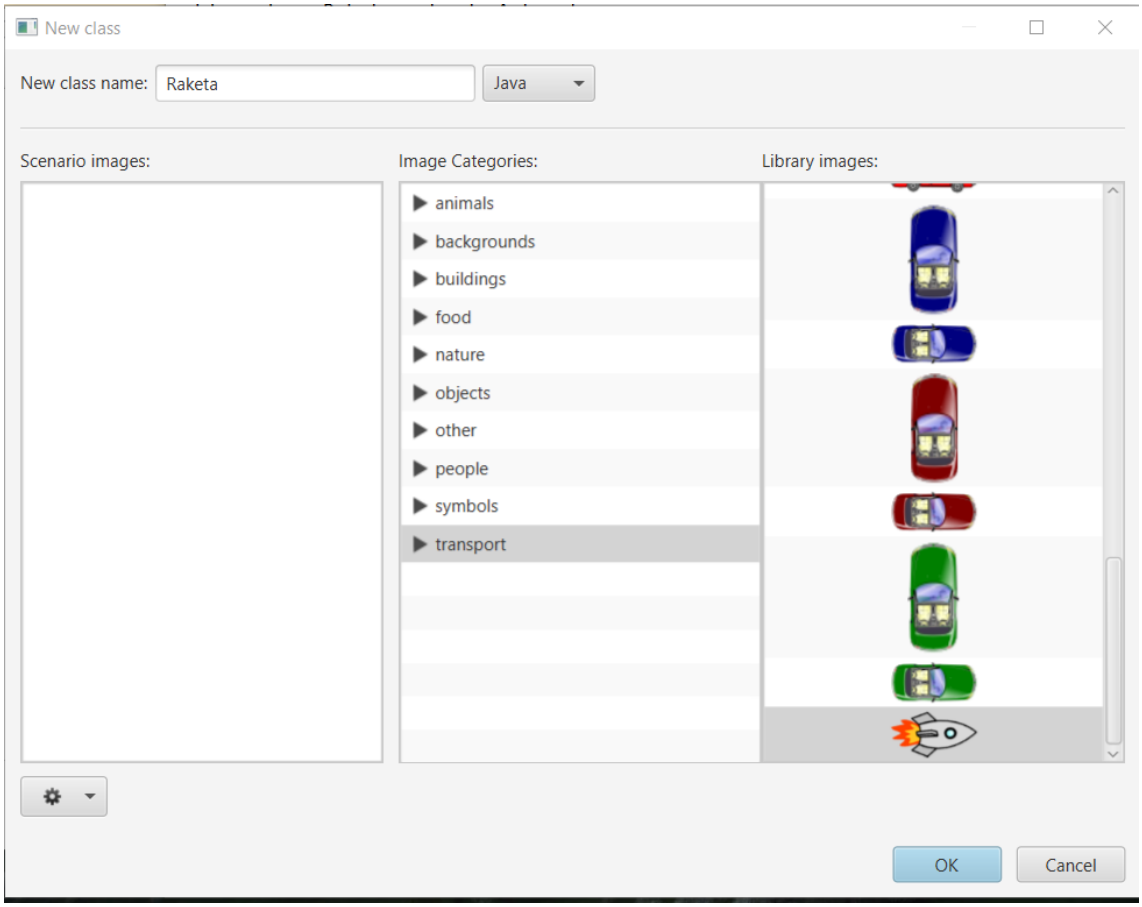


Obrázok 2 - Projekt Asteroidy po vytvorení nového Java Scenaria

ÚLOHA 1.2

Prvým aktorom, ktorý pribudne na vesmírnu scénu je vesmírna loď (Raketa). Vytvorte triedu **`Raketa`** a nastavte jej obrázok zo skupiny **Transport**.

Pravým tlačítkom myši kliknite na triedu `Actor` a zvoľte **New subclass**.
New class name (názov triedy) zvoľte Raketa, pretože pomenovanie Vesmírna loď nie je vhodné použiť kvôli združenému pomenovaniu – obsahuje medzeru a tá sa pri názvosloví v programovaní nepoužíva.
V **Image Categories** kliknite na **transport** – následne v **Library images** nájdite obrázok rakety a stlačte **OK**.



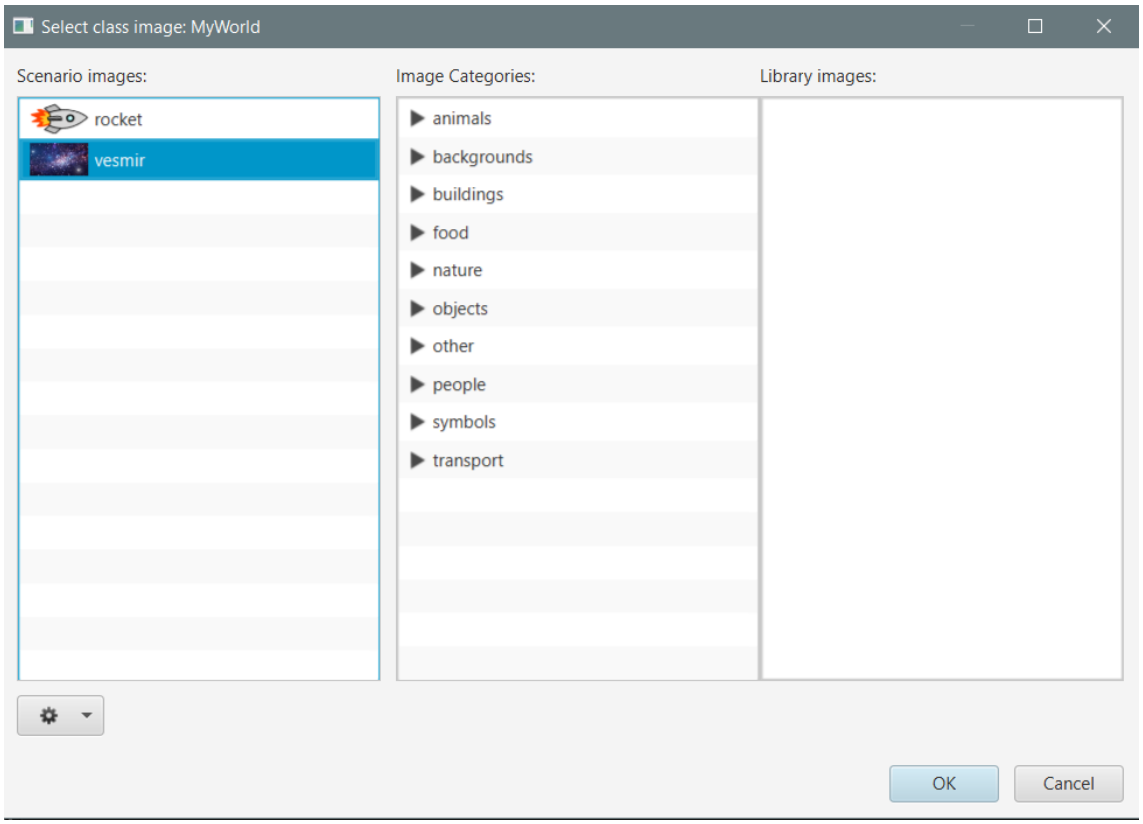
Obrázok 3 - Vytvorenie triedy a nájdenie obrázku Raketa

ÚLOHA 1.3

Trieda `MyWorld` bude reprezentovať hracie pole hry. Nastavte pozadie hracieho poľa na obrázok vesmíru. Obrázok stiahnite z internetu.

Nastavenie pozadia nie je z hľadiska funkcionality hry dôležitým krokom, ide iba o vizuálnu stránku na ktorú však pri tvorbe hier netreba zabúdať. Hra Asteroidy je s vesmírnym pozadím zobrazená len na obrázku 5. Ostatné obrázky sú pre lepšiu prehľadnosť bez vesmírneho pozadia .
Obrázok stiahnite z internetu a uložte ho do priečinka **Asteroidy/images**. Obrázok je potrebné vhodne pomenovať (napríklad *vesmir).*
Následne sa vráťte do prostredia Greenfoot a pravým tlačítkom myši kliknite na triedu `MyWorld`. V menu, ktoré sa vám zobrazí zvoľte **Set Image**. Zobrazí sa vám okno **Select class**

image: **MyWorld** a ak ste obrázok správne uložili, bude sa nachádzať medzi **Scenario images** (Obrázok 4) . Zvoľte obrázok s názvom *vesmír* a stlačte **OK**.



Obrázok 4 - Nastavenie pozadia hracieho poľa

ÚLOHA 1.4

Zmeňte hraciu plochu tak, aby jej veľkosť bola 600x600 buniek, pričom každá bunka bude veľká jeden pixel.

V konštruktore triedy MyWorld upravíme jediný riadok:

```
public MyWorld () {
     super (600, 600, 1);
}
```

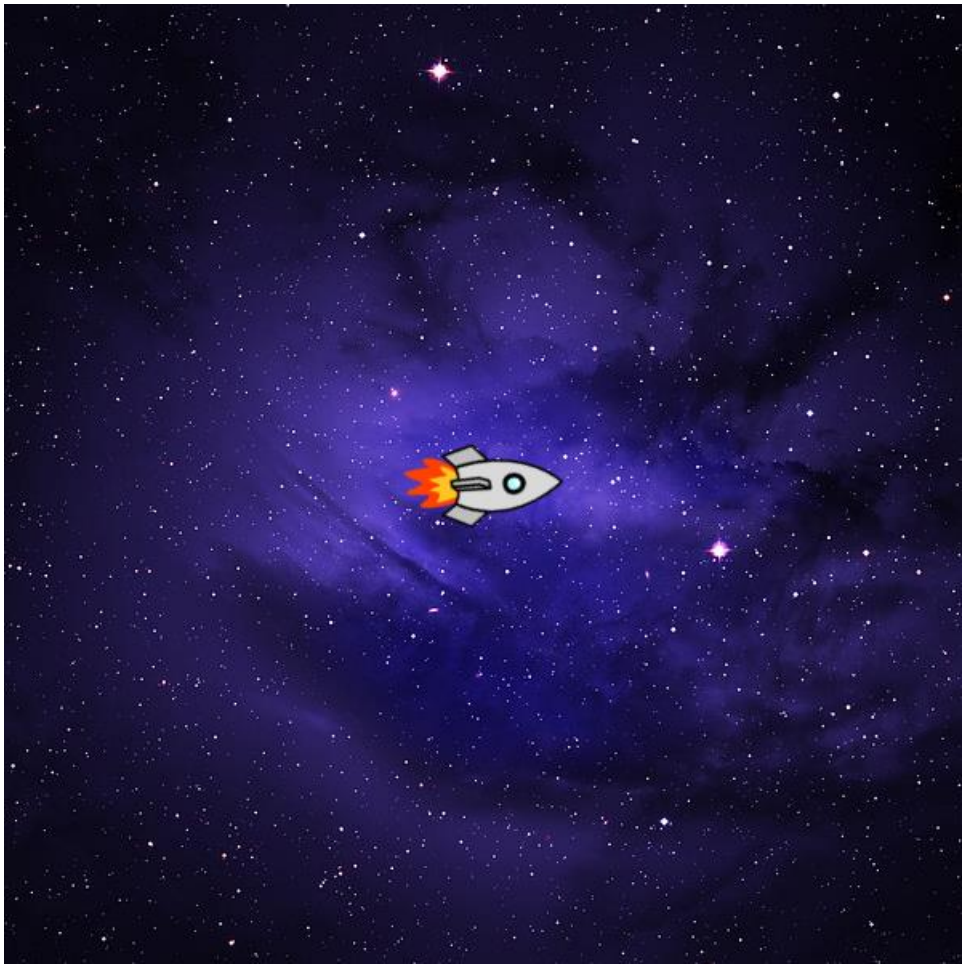
ÚLOHA 1.5

Vytvorte inštanciu vesmírnej lode tak, aby bola zobrazená v hracom poli (vo vesmíre).

Aby sme raketu pridali do scény, najprv ju musíme vytvoriť. Vytvorenú raketu si uložíme do pomocnej premennej a potom ju do sveta vložíme napr. do stredu okna.

```
public MyWorld() {
     super(600, 600, 1);
```

```
        Raketa raketa = new Raketa();
        addObject(raketa, 300, 300);
}
```



Obrázok 5 - Vesmírna loď v hracom poli [1]

2 POHYB VESMÍRNEJ LODE VPRED

V tejto časti začneme vesmírnu loď učiť ako sa má pohybovať vo vesmíre. Nezabudnite, že to, čo sme nenaprogramovali sa nemá ako stať. Zatiaľ sme vesmírnu loď nenaučili na aký povel sa má pohnúť a ani čo pohnutie vlastne znamená – preto sa to v našej hre nemá ako diať.

Základným pohybom, ktorý je potrebné naučiť vesmírnu loď je pohyb vpred. Po zvládnutí tejto kapitoly sa raketa bude posúvať vpred po stlačení medzerníka, avšak len v smere v ktorom je otočená po pridaní do sveta.

Keď chceme vesmírnu loď naučiť pohybu – potrebuje atribút reprezentujúci rýchlosť. Ak je rýchlosť rakety 0, nepohne sa. Čím väčšiu hodnotu rýchlosti priradíme, tým väčšiu vzdialenosť raketa prejde, keď od nej budeme chcieť aby sa pohla.

ÚLOHA 2.1

Pridajte vesmírnej lodi atribút, ktorý bude reprezentovať jej rýchlosť. Rýchlosť vesmírnej lode zadefinujte ako celočíselnú premennú, ktorej hodnotu nastavte v konštruktore na 5.

V triede **`Raketa`** vytvorte nový atribút. Keďže v zadaní je uvedené, že rýchlosť má byť celočíselná premenná – bude typu int**.**

```
private class Raketa extends Actor {
     private int rychlost;

     public Raketa() {
           this.rychlost = 5;
     }
}
```

ÚLOHA 2.2

Vesmírna loď už má určenú svoju rýchlosť, avšak stále dokáže stáť len na jednom mieste, pretože nemá schopnosť *pohnúť sa*. Naučte vesmírnu loď, aby sa pohla o toľko bodov, koľko určuje jej rýchlosť a pohyb vykonala po každom stlačení medzerníka. Keďže chceme, aby sa v hre neustále kontrolovalo, či nedošlo k stlačeniu medzerníka – vytvorte podmienku v metóde **`act()`**, po ktorej splnení vesmírna loď vykoná pohyb vpred.
Na jednoznačné vymedzenie ovládania použite **`Greenfoot.isKeyDown("space")`** – čo znamená, že bola stlačená klávesa space (medzerník). Loď posuniete vpred zavolaním **`this.move(this.rychlost)`**.

V triede **Raketa** by mala metóda **act()** vyzerať nasledovne:

```
public void act() {
     if (Greenfoot.isKeyDown("space")){
           this.move(this.rychlost);
     }
}
```

Ak by sme rýchlosti priradili v konštruktore zápornu hodnotu (napríklad -5), raketa by sa hýbala opačným smerom.

3 OTÁČANIE VESMÍRNEJ LODE ZA MYŠOU

V predošlej kapitole ste naučili vesmírnu loď vykonávať pohyb vpred pri stlačení klávesy medzerníka. Cieľom tejto kapitoly je umožniť vesmírnej lodi pohyb do všetkých strán. Raketa sa teda nebude hýbať len do jedného smeru, ale pomocou kurzora myši bude schopná otočiť sa kdekoľvek a následne sa pomocou stlačenia medzerníka pohnúť daným smerom.

Informácie o myši a kurzore sú uložené v objekte **`MouseInfo`**. Tento objekt obsahuje funkcie na zistenie pozície myši, kód kliknutého tlačidla myši (1 – ľavé tlačidlo, 2 – stredné, 3 – pravé), atď. Tieto informácie nám budú nápomocné pri nasledujúcich úlohách.

Príkazom **`Greenfoot.getMouseInfo()`** získame objekt **`MouseInfo`**. Ten si uložíme do lokálnej premennej, ktorú môžeme nazvať **`mouse`**. O deklarácii (lokálnych) premenných sa dočítate viac aj v knihe v kapitole 4.3 Deklarácia premenných. Pripomeňme si, že na deklarovanie premennej potrebujeme najmä jej typ a názov. Vieme, že názov bude **`mouse`** a typ bude **`MouseInfo`**, pretože ukladáme objekt **`MouseInfo`**:

```
MouseInfo mouse = Greenfoot.getMouseInfo();
```

Názov metódy	Návratový typ	Popis
getActor()	Actor	Získanie aktora, s ktorým kliknutie súvisí
getButton()	Int	Číslo kliknutého tlačidla
getClickCount()	Int	Počet kliknutí (identifikácia dvojkliku)
getX()	Int	Pozícia myši na x-ovej osi.
getY()	Int	Pozícia myši na y-ovej osi.

Teraz môžeme informácie o myši získavať jednoduchým spôsobom. Ku všetkým metódam objektu **`MouseInfo`** sa vieme dostať cez lokálnu premennú s názvom **`mouse`**. Ak chceme zistiť aktuálnu x-pozíciu kurzora myši, stačí použiť príkaz **`mouse.getX()`**. Táto metóda nám vráti celé číslo reprezentujúce aktuálnu x-pozíciu kurzora myši. Keby sme si na začiatku práce s myšou nevytvorili lokálnu premennú, ku všetkým informáciám týkajúcich sa práce s myšou by sme museli pristupovať cez metódu **`getMouseInfo()`**. Na príklade získavania aktuálnej x-pozície by to vyzeralo nasledovne:

```
int xPoziciaKurzora = Greenfoot.getMouseInfo.getX();
```

Teraz nám pre získanie pozície x bude stačiť kratší kód, čo nám zabezpečí lepšiu prehľadnosť kódu:

```
int xPoziciaKurzora = mouse.getX();
```

Upozorňujeme, že príkazom **`getMouseInfo()`** získavame vždy **aktuálne** informácie. Keď by sme si informácie o myši vyžiadali v konštrukture rakety, vedeli by sme získať informáciu o x-pozícii (a ďalšie iné informácie o myši) len v momente, kedy boli tieto informácie vyžiadané, t.j. pri vytváraní rakety.

ÚLOHA 3.1

Pre otočenie rakety za kurzorom potrebujete vedieť jeho pozíciu na hracej ploche. Informácie o myši a kurzore získate príkazom **`Greenfoot.getMouseInfo()`**.
Tieto informácie o myši a kurzore uložte do premennej typu **`MouseInfo`**. Dobre si premyslite, v ktorej časti kódu by sme mali programovať otáčanie vesmírnej lode.

V triede **`Raketa`** v metóde **`act()`** uložíme informácie o myši a kurzore do premennej **`mys`** typu **`MouseInfo`**:

```
MouseInfo mys = Greenfoot.getMouseInfo();
```

ÚLOHA 3.2

V prechádzajúcej úlohe sa nám podarilo získať informácie o kurzore a myši. Teraz ich budeme potrebovať, aby sme vedeli povedať rakete, ktorým smerom sa má otočiť.
Otočte raketu za kurzorom. Pozíciu kurzora myši získate funkciami **`getX()`** a **`getY()`**. Raketu k týmto koordinátom otočíte príkazom **`this.turnTowards(x,y)`**.

Pod riadok, ktorý ste napísali v úlohe 3.1 dopíšte do kódu:

```
this.turnTowards(mys.getX(), mys.getY());
```

ÚLOHA 3.3

Už viete otočiť myš za kurzorom a zároveň s využitím medzerníka a pohybu naprogramovaného v úlohe 2.1 dokážete raketu pohnúť ktorýmkoľvek smerom. Môže sa však stať, že myš sa nenachádza na hracom plátne a preto informácie o nej nie sú dostupné. Ak by sme sa snažili raketu otočiť za myšou, ktorú program nepozná (je mimo hracieho plátna), program by spadol a hra by nemohla ďalej pokračovať. Takýto prípad je potrebné vopred ošetriť.
Vytvorte podmienku, ktorá zabezpečí, že raketa sa bude otáčať za myšou len v prípade, že ju program dokáže rozpoznať. (Môžete využiť porovnanie premennej **`mys`** s hodnotou **`null`**).

Prechádzajúci kód triedy **Raketa** v metóde **act()** doplníme o podmienku a bude vyzerať nasledovne:

```
MouseInfo mys = Greenfoot.getMouseInfo();
if (mys != null) {
     This.turnTowards(mys.getX(), mys.getY());
}
```

4 PRIDANIE ASTEROIDOV DO HRY

Raketa sa už vo vesmíre dokáže pohybovať, ďalším krokom pre nás bude vytvorenie asteroidov, ktorým sa bude musieť uhýbať. Asteroidy vytvoríme rovnakým spôsobom ako raketu, s rozdielom, že asteroidy budeme chcieť vytvárať hromadne, vo väčšom počte.

ÚLOHA 4.1

Vytvorte triedu **`Asteroid`** a nastavte jej obrázok asteroidu, ktorý nájdete v skupine **Nature**.

Asteroid bude vytvorený ako **`Actor`**, rovnako ako **`Raketa`**.

ÚLOHA 4.2

Ak by sme chceli asteroidy do sveta pridávať rovnakým spôsobom, ako sme si ukázali v úlohe 1.5, kód by bol príliš dlhý a v počte asteroidov by sa dalo ľahko stratiť. Preto pomocou cyklu **`for`** vytvorte viacero asteroidov naraz s použitím jednoduchého kódu. Asteroidy vzniknú spolu so svetom, preto dopíšte kód v konštruktore triedy **`MyWorld`**.

V konštruktore triedy **`MyWorld`** pridáme cyklus, v ktorom sa opakovane vykoná v ňom obsiahnutý kód.

```
for (int i = 0; i < 5; i++) {
     Asteroid asteroid = new Asteroid();
     this.addObject(asteroid, 0, 0);
}
```

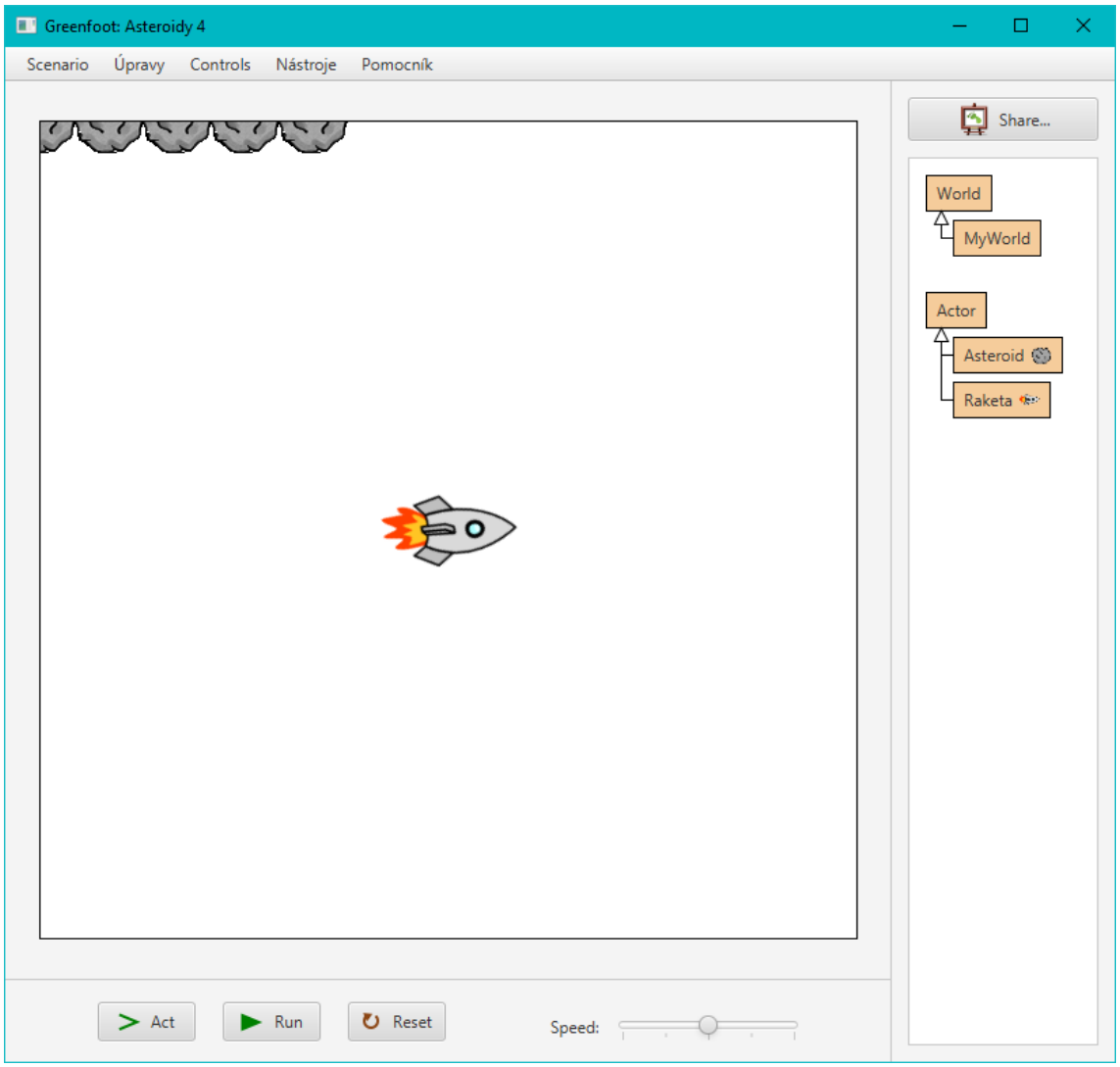
Po spustení hry sa budú všetky asteroidy nachádzať na rovnakej pozícií a preto budú vyzerať ako jeden. V skutočnosti ich je však vytvorených viac, len sa dokonale prekrývajú. Uloženie asteroidov na sebe neovplyvní priebeh hry, pretože asteroidy sa po spustení program začnú hýbať náhodným smerom. To znamená, že asteroidy sa ďalej pravdepodobne prekrývať nebudú.

ÚLOHA 4.3

Použite číslo v premennej **`i`** na vytvorenie asteroidov na rôznych pozíciách, aby ste sa presvedčili, že ich je naozaj viac. Využite fakt, že číslo **`i`** sa zvyšuje pri každom novom asteroide.

Premennú `i` môžeme prenásobiť konštantou, čím získame čísla s väčšími rozostupmi a asteroidy vďaka tomu umiestnime na rôzne pozície.

```
for (int i = 0; i < 5; i++) {
     Asteroid asteroid = new Asteroid();
     addObject(asteroid, i*50, 0);
}
```



Obrázok 6 - Vytvorenie asteroidov pomocou cyklu for

5 POHYB ASTEROIDOV

Asteroidy sa v hre budú pohybovať stále rovnakou rýchlosťou po celú dobu letu. Aby sme ich od seba odlíšili, každý asteroid pri jeho vytvorení otočíme náhodným smerom, ktorým bude následne letieť. Okrem náhodnej rotácie umožníme asteroidom aj let náhodnou rýchlosťou.

Nevyhnutnou súčasťou tejto kapitoly je povolenie aktorom pohyb mimo okna, pretože inak by sme sa pri každej hre dostali do situácie, kedy aktori zostanú zaseknutí na niektorom z okrajov hracieho poľa.

ÚLOHA 5.1

Pridajte asteroidom atribút pre uloženie rýchlosti, rovnako ako pri rakete v úlohe 2.1 a použite túto rýchlosť v metóde **`act()`** na jeho pohyb.

```
public class Asteroid extends Actor {

     private int rychlost;

     public Asteroid() {
           this.rychlost = 5;
     }

     public void act() {
        this.move(this.rychlost);
     }
}
```

ÚLOHA 5.2

Aby sa asteroidy pohybovali rôznymi smermi, musíme ich potočiť. Na to budeme využívať generovanie náhodných čísel, ktoré budú predstavovať uhol, o ktorý sa pri vytvorení asteroid otočí. Asteroid chceme otočiť iba raz, preto ho otočíme rovno pri jeho vytvorení – v konštruktore.

Na získanie náhodného čísla použijeme **`Greenfoot.getRandomNumber(360)`**, pričom parameter 360 je horná hranica generovaného čísla. Asteroidy budú teda otočené o niečo medzi 0 až 359 stupňov. Viac o generovaní čísel si prečítajte v knihe v kapitole 10 Náhodné čísla.

```
public Asteroid() {
     int uhol = Greenfoot.getRandomNumber(360);
     this.turn(uhol);
}
```

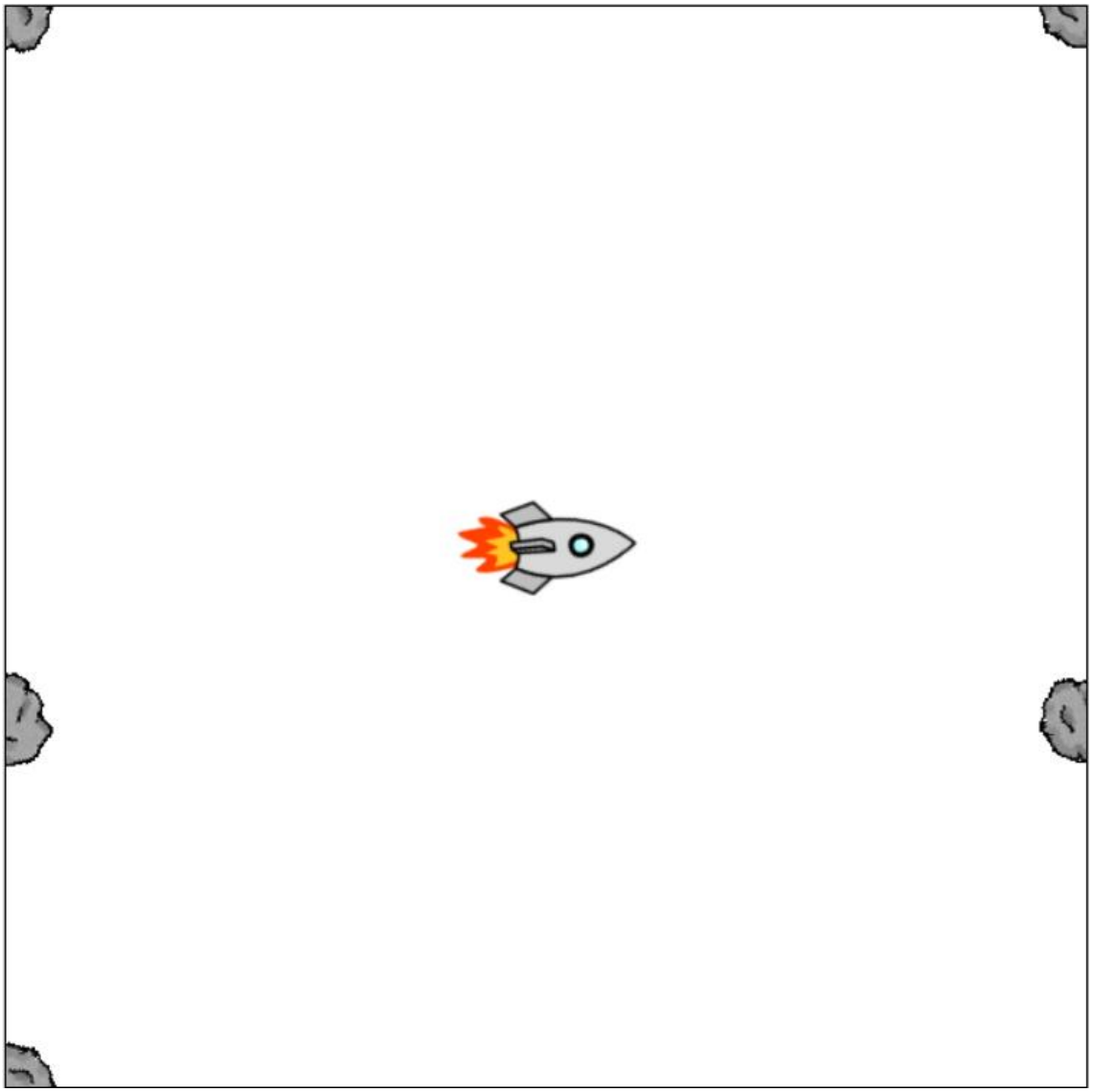
ÚLOHA 5.3

Upravte rýchlosť asteroidov na náhodné číslo, aby sa nepohybovali všetky rovnakou rýchlosťou. Aby sme sa vyhli vygenerovaniu čísla 0, vygenerované číslo môžeme navýšiť o 1.

```
private int rychlost = Greenfoot.getRandomNumber(5) + 1;
```

ÚLOHA 5.4

Nastavte, aby bol možný pohyb aktorov mimo hracej plochy. Ohraničenosť sveta sa určuje štvrtým parametrom v konštruktore triedy **`MyWorld`** po parametroch šírka okna, výška okna a veľkosť buniek. Tento štvrtý parameter je voliteľný a ak ho nepoužijete, bude automaticky nastavený na hodnotu **`true`**, čo znamená, že vychádzanie aktorov mimo okna bude zakázané. To by znamenalo, že svet bude mať vytvorené hranice za ktoré nemožno prejsť tak, ako to bolo doteraz (túto situáciu môžeme vidieť aj na obrázku 7).

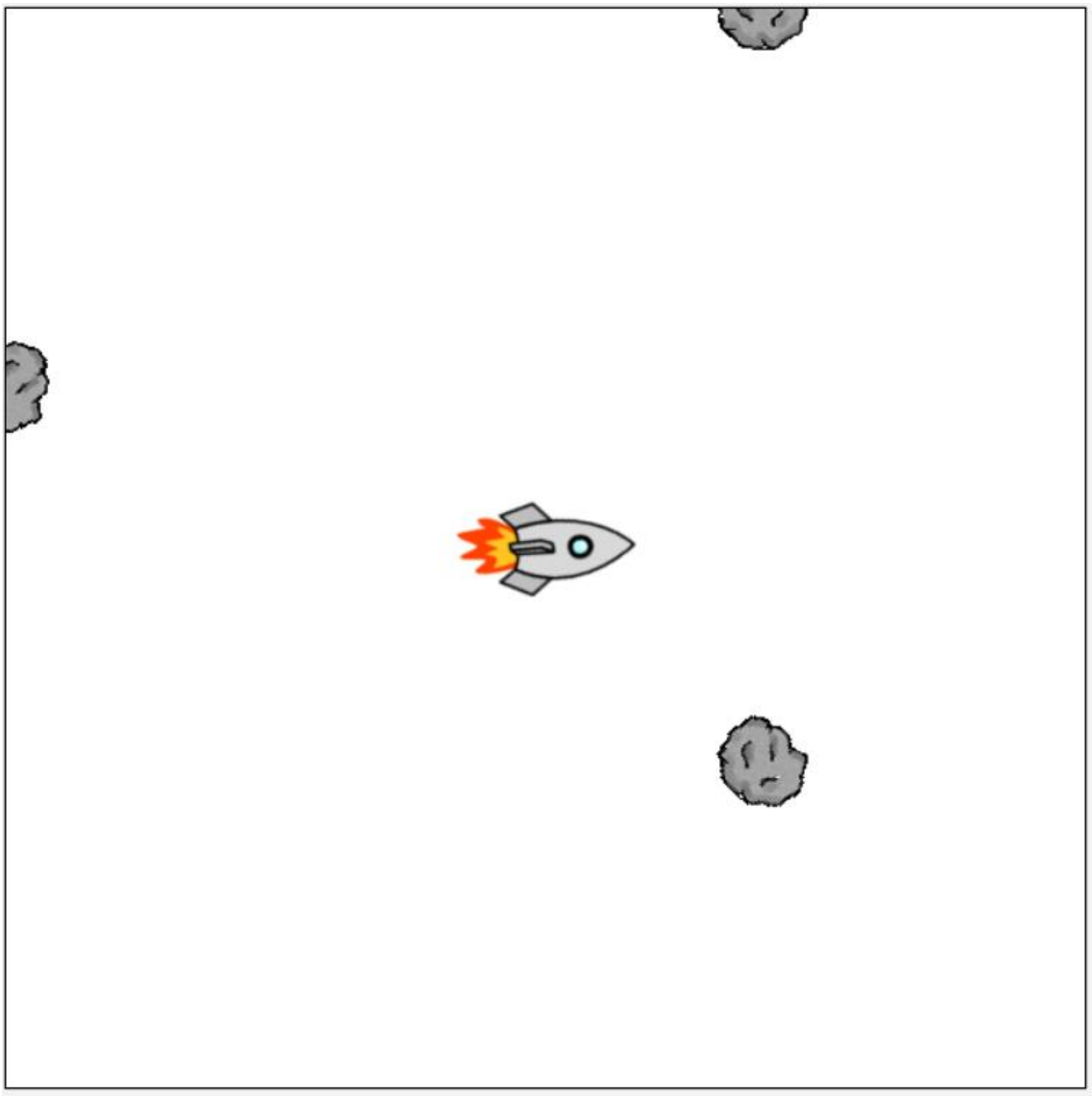


Obrázok 7 - Asteroidy zaseknuté na hraniciach sveta

V konštruktore triedy **`MyWorld`** upravíme jediný riadok:

```
public MyWorld () {
     super (600, 600, 1, false);
}
```

Štvrtý parameter slúži na umožnenie pohybu objektov mimo hracieho poľa – keď ho nastavíme na **`false`**, tento pohyb umožníme. Na obrázku 8 vidieť, že 2 asteroidy vyšli za hranice sveta.

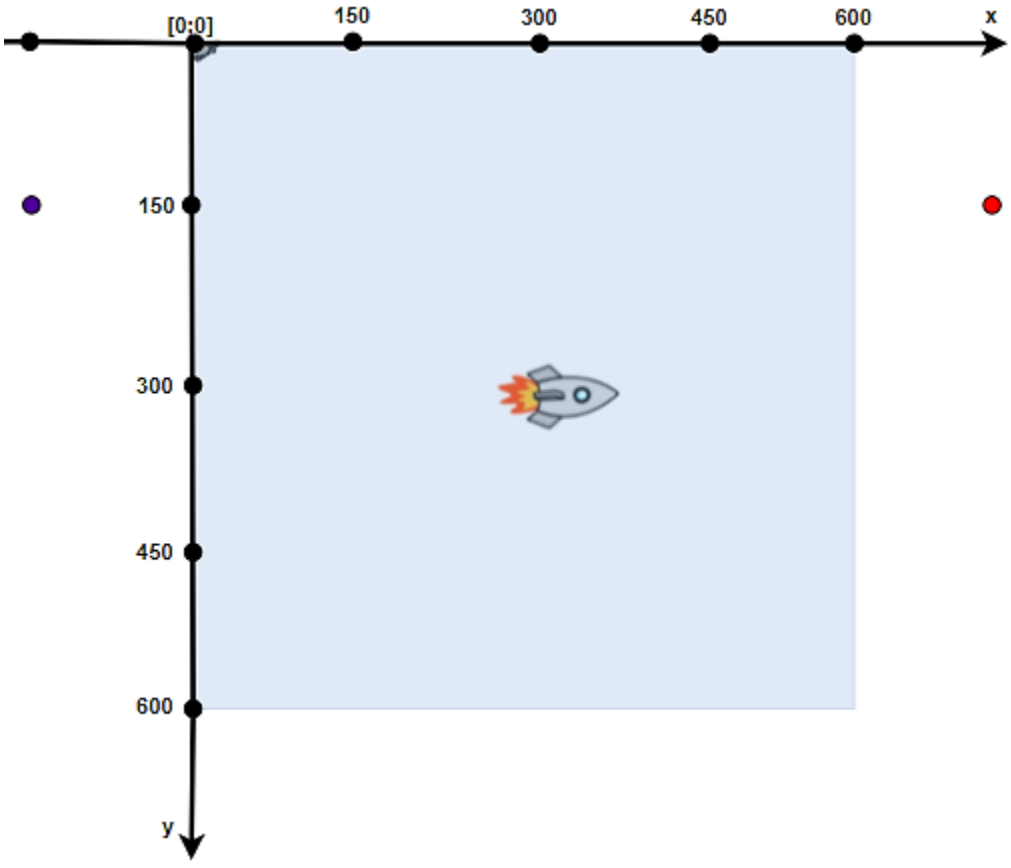


Obrázok 8 - Vychádzanie aktorov za hranice sveta

6 PRESUN ASTEROIDOV, KTORÉ ODÍDU Z OKNA

Pretože sme v úlohe 5.4 povolili aktorom pohyb mimo okna, stáva sa, že asteroidy z neho vyletia preč a nikdy sa nevrátia. Preto upravíme spôsob, ktorým sa budú pohybovať – keď presiahnu hranicu okna, vrátime ich na opačnú stranu. Po implementovaní takého spôsobu premiestňovania, budú asteroidy vyzerať, akoby dokázali prechádzať z jednej strany na druhú. Na to potrebujeme poznať pozíciu ateroidu, rozmery okna a pochopiť funkciu *modulo* (zvyšok po delení). Upravovať budeme metódu `act()` v triede `Asteroid`, pretože asteroid môže okno prekročiť kedykoľvek v priebehu hry.

Operáciu modulo vysvetlíme pomocou nasledujúcich obrázkov. Na obrázku 9 je zobrazené naše hracie pole s rozmermi 600x600 (zvýraznené jemnou modrou farbou). Keby asteroid chcel dosiahnuť pozíciu napríklad [750;150] (červený bod), bol by mimo hracieho poľa a pre hráča neviditeľný. Rovnaký problém by mohol nastať so zápornými číslami, napríklad na pozícii [-150;150] (fialový bod).

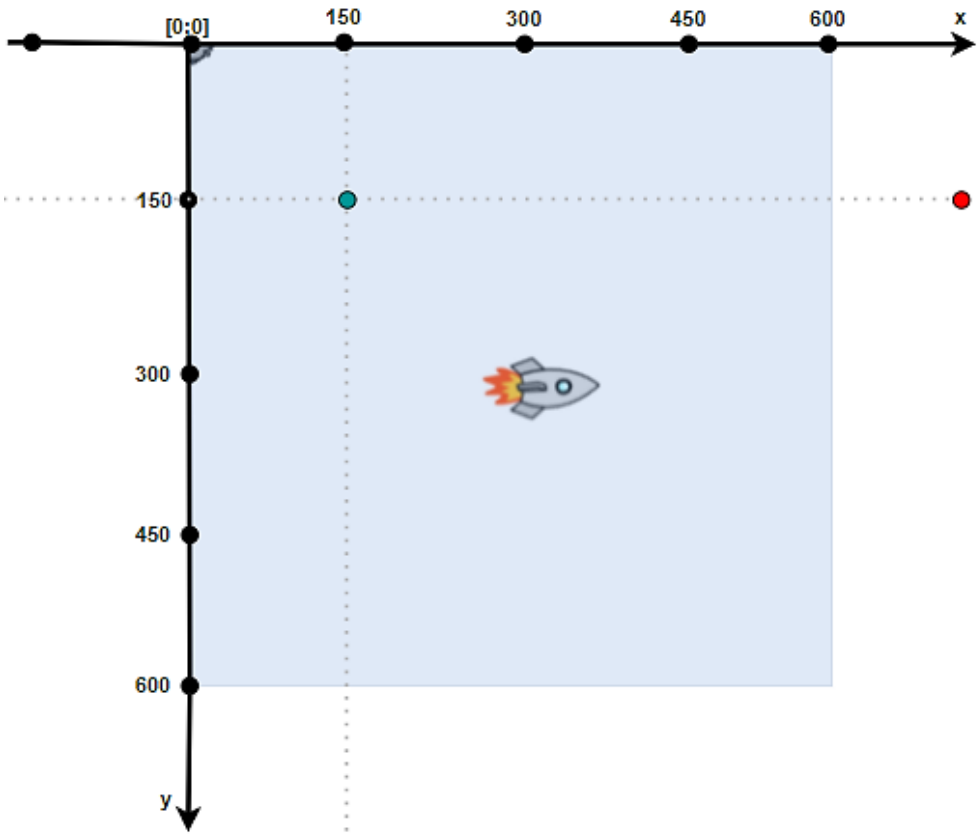


Obrázok 9 – Pozície bodov bez použitia operácie modulo

Keď použijeme operáciu modulo (zvyšok po delení celočíselnom delení, ktoré budeme označovať znakom **//**), situácia z predchádzajúceho obrázku sa zmení. Pri modulovaní používame operátor **%** (ako pri sčítavaní +, odčítavaní -,…). Ak hodnotu x-pozície zmodulujeme x-rozmerom hracieho poľa, z čísla 750 dostaneme 150:

750 % 600 = **150** (700 // 600 = 1, zvyšok **150**).

Rovnaký postup je potrebné použiť pre y koordináty. Na obrázku 10 sú znázornené koordináty [750 % 600 ; 150 % 600]. Z červeného bodu sa stal modrý, viditeľný na hracej ploche.



Obrázok 10 - Pozícia [750;150] bez a s použitím operácie modulo

Ak by asteroid chcel prejsť do záporných koordinátov, samotné modulo nám v jazyku Java nepostačí. Upozorňujeme, že funkciu modulo majú rôzne programovacie jazyky implementované rozličným spôsobom. Pri výpočte modula kladných čísiel by sme mali v každom jazyku dostať rovnaké hodnoty, avšak pri negatívnych hodnotách je potrebné zvýšiť pozornosť. Z matematiky vieme, že záporná hodnota ako zvyšok po delení nedáva zmysel. Ak kladné číslo delíme číslom 10, zvyšok po delení môže byť od 0 po 9.

17 % 10 = **7** (17 // 10 = 1, zvyšok **7**)

Ak je delenec záporný (-17) a deliteľ opäť 10, možnosti zvyšku po delení by mali byť rovnaké (0-9). Teraz však zisťujeme, koľko násobkov čísla 10 je pripočítaných k -17, kým sa dostaneme do rozmedzia hodnôt 0-9. Hodnota na ktorú sa pripočítavaním dostaneme je zároveň aj výsledkom.

-17 % 10 = **3** (-17 + 10 + 10 = **3** -> Hodnotu 10 sme pripočítali 2x a dostali sme sa na číslo **3**)

Problémom je, že v jazyku Java je výsledok modula z negatívneho čísla rovnaký ako z kladného, avšak pred výsledok je dopísané mínus. Z predchádzajúceho príkladu by sme teda v jazyku Java dostali iný výsledok:

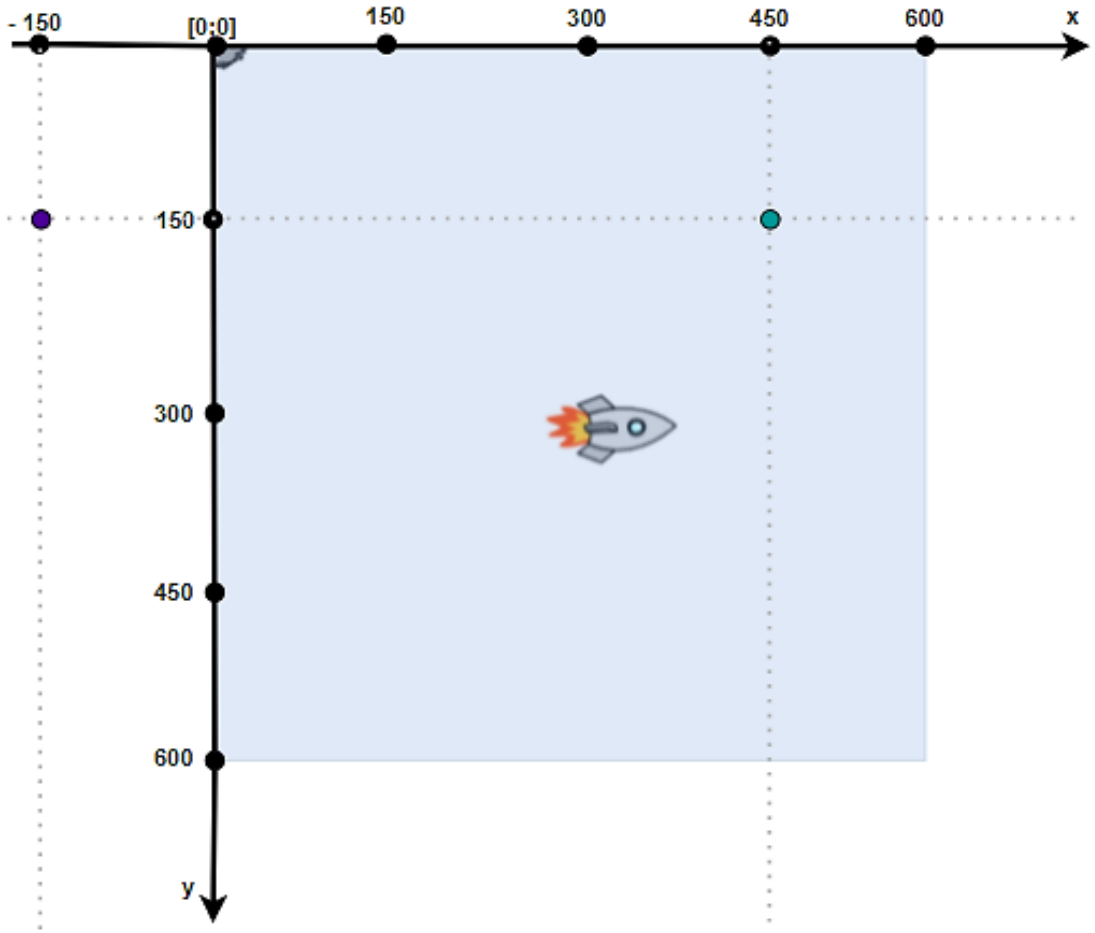
-17 % 10 = **-7** (rovnako ako 17 % 10 = 7, ale z kladnej hodnoty spravíme zápornú)

Teraz si vysvetlíme spôsob akým získame požadovanú hodnotu modula zo záporného čísla aj v jazyku Java. Z negatívnej hodnoty musíme najskôr spraviť pozitívnu a až potom môžeme použiť

modulo. Máme koordináty [-150;150], čo znamená, že fialový bod by sa z ľavej strany mal presunúť na pozíciu x, ktorá bude maximálnaX (600) zmenšená o 150. Dostaneme teda pozíciu [450;150], čo je na obrázku 11 bod znázornený modrou farbou.

Hodnoty maximálnych rozmerov môžeme pripočítavať aj v prípade kladných čísel, pretože výsledok týmto sčítaním ovplyvnený nebude. Na príklade hodnoty 750 si to môžeme potvrdiť nasledujúcim výpočtom:

(750 + 600) % 600 = 1350 % 600 = **150** (1350 // 600 = 2, zvyšok **150**).



Obrázok 11 - Pozícia [-150;150] bez a s použitím operácie modulo

ÚLOHA 6.1

Zadefinujte premenné `x` a `y` do ktorých uložíte aktuálne koordináty (pozíciu) asteroidu v rámci hracieho okna.

```
int x = this.getX();
int y = this.getY();
```

ÚLOHA 6.2

Získajte informácie o veľkosti hracieho okna. Zistite šírku aj výšku hracieho poľa a uložte ich do premenných **`maxX`** a **`maxY`**. Tieto hodnoty budú reprezentovať maximálne pozície v smere x a y, ktoré môže asteroid dosiahnuť.

```
int maxX = this.getWorld().getWidth();
int maxY = this.getWorld().getHeight();
```

ÚLOHA 6.3

Vypočítajte pomocou funkcie modulo (`%`) novú pozíciu pre asteroid. Keď asteroid presiahne hranicu okna, zvyšok po delení jeho koordinátov veľkosťou okna nám určí jeho novú pozíciu na opačnej strane.

```
x = (x + maxX) % maxX;
y = (y + maxY) % maxY;

this.setLocation(x, y);
```

7 STREĽBA Z VESMÍRNEJ LODE

Ďalším ovládacím prvkom rakety je strieľanie z nej. Po kliknutí myšou chceme, aby vystrelila na asteroid pred seba. V časti 3.1 sme pracovali s objektom **`MouseInfo`**, v tejto časti nám pomôže identifikovať kliknutie, teda moment za behu, kedy sa má vytvoriť nová strela.

ÚLOHA 7.1

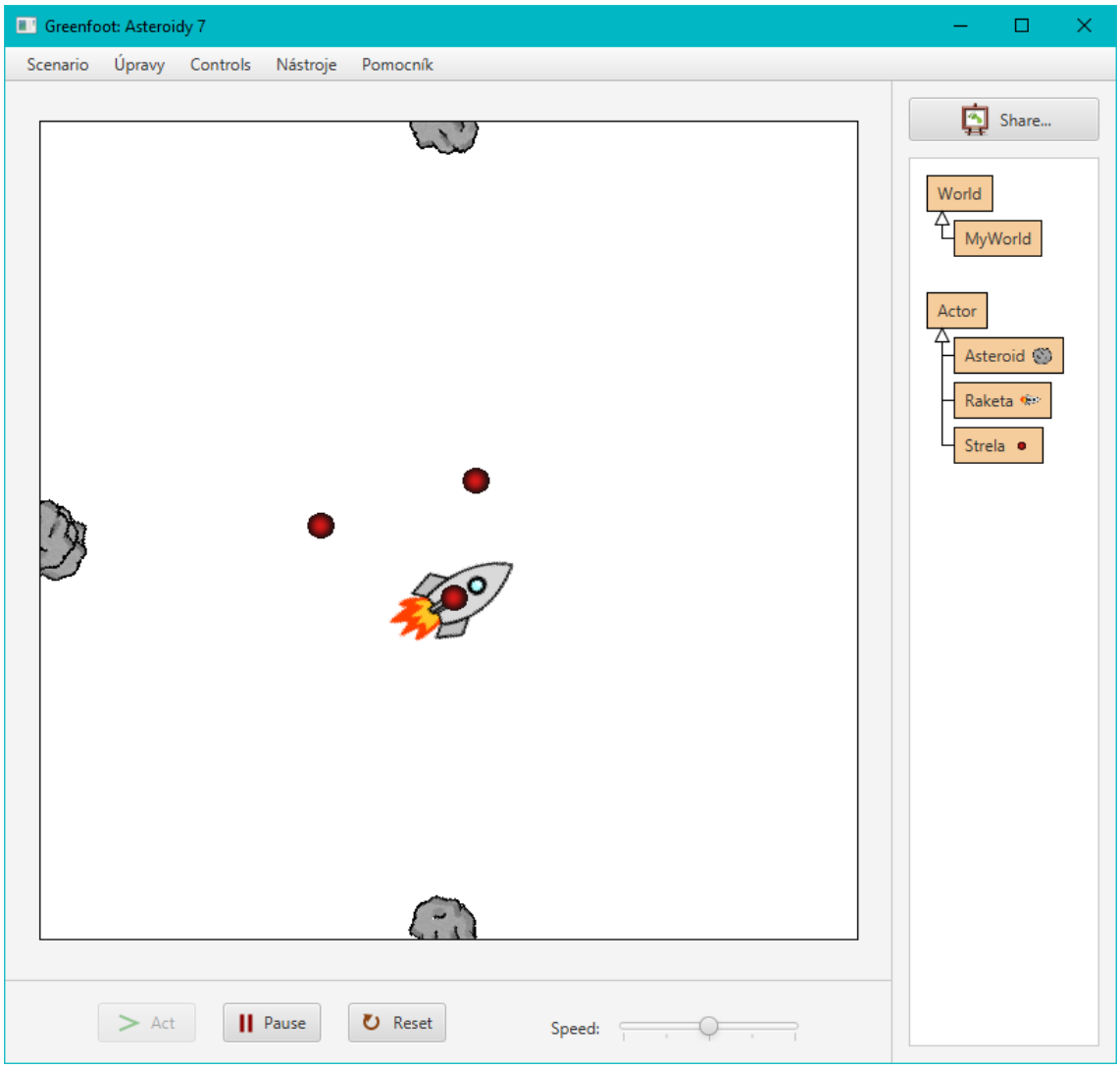
Do projektu pridajte nového aktora **`Strela`** a zvoľte vhodný obrázok. My sme použili obrázok zo sekcie **`Other`**.

ÚLOHA 7.2

V triede **`Raketa`** vytvorte parametrickú metódu **`mozeStrielat(MouseInfo mys)`** v ktorej skontrolujeme, či raketa môže vystreliť. V tejto úlohe opäť využijeme funkcie triedy **`MouseInfo`**, ktoré sme prvýkrát použili v úlohe 3.1 pri zisťovaní aktuálnej polohy kurzora. Podmienky pre vystrelenie rakety sú 2:

1. Bolo stlačené ľavé tlačítko myši **`(mys.getButton() == 1)`**
2. **`Greenfoot.mouseClicked(null) == true`**, aby sa tlačidlo nedalo držať (podmienka slúži na identifikáciu kliknutia). Ak by sme túto podmienku nepoužili, stačilo by držať ľavé tlačidlo myši a raketa by neustále strieľala (ako samopáľ). Týmto spôsobom by však bolo príliš jednoduché hru vyhrať a preto chceme, aby na jeden klik raketa vystrelila len jednu strelu.

Po úspešnej kontrole podmienok metóda **`mozeStrielat()`** vráti hodnotu **`true`**, inak **`false`**, preto jej návratová hodnota musí byť typu **`boolean`**.



Obrázok 12 - Vytvorenie nehybných striel po kliknutí

V triede **Raketa** vytvoríme nasledujúcu metódu:

```
public boolean mozeStrielat(MouseInfo mys) {
  if (mys.getButton() == 1 && Greenfoot.mouseClicked(null) == true) {
     return true;
   } else {
     return false;
   }
}
```

Pretože sa nachádzame v triede `Raketa`, na získanie jej pozície nám stačí podobne ako v úlohe 6.1 iba zavolať funkcie **`getX()`** a **`getY()`**.

ÚLOHA 7.3

V triede **Raketa** vytvorte metódu **strielaj()**, ktorá vytvorí novú inštanciu triedy **Strela** a vloží ju do hry na pozíciu rakety. Rozmyslite si, akú návratovú hodnotu bude mať táto metóda a taktiež či je potrebné, aby mala nejaké parametre.

V triede **Raketa** vytvoríme nasledujúcu metódu **strielaj()**, ktorá má návratovú hodnotu typu void, pretože nepotrebujeme, aby niečo vrátila. Jej jedinou úlohou je vytvoriť strelu a pridať ju do sveta.

```
public void strielaj() {
        Strela strela = new Strela();
        this.getWorld().addObject(strela, this.getX(), this.getY());
}
```

Pretože sa nachádzame v triede **Raketa**, na získanie jej pozície nám stačí podobne ako v úlohe 6.1 iba zavolať funkcie **getX()** a **getY()**.

ÚLOHA 7.4

V metóde **act()** triedy **Raketa** zavolajte metódu **mozeStrielat(MouseInfo mys)** z úlohy 7.2. Ak sú podmienky vystrelenia splnené, zavolajte metódu **strielaj()**, ktorú ste definovali v úlohe 7.3. Keďže pracujeme s myšou, kód súvisiaci so strieľaním nezabudnite písať na miesto, ktoré sme vytvorili v úlohe 3.3 – musí platiť, že premenná **mys** nie je rovná **null**.

Rozšírime kód, ktorý sme písali v úlohe 3.3:

```
public void act() {
    MouseInfo mys = Greenfoot.getMouseInfo();
        if (mys != null) {
            this.turnTowards(mys.getX(), mys.getY());

            if (mozeStrielat(mys) == true) {
                strielaj();
            }
    }
}
```

Vytvorené strely sa ešte nepohybujú. Rovnako ako pri rakete aj asteroidoch, aj strele pridáme atribút **rychlost** a rozhýbeme ju pomocou **this.move(this.rychlost)**. Po zapísaní tohto kódu strely ešte stále nebudú letieť správnym smerom. Smer, ktorým majú letieť je rovnaký ako smer rakety v momente kliknutia.

ÚLOHA 7.5

Rovnako ako v úlohe 5.1 rozhýbte strelu.

```
public class Strela extends Actor {

     private int rychlost;

     public Strela () {
           this.rychlost = 7;
     }

     public void act() {
           this.move(this.rychlost);
     }
}
```

ÚLOHA 7.6

Otočte strelu rovnakým smerom, akým bola otočená raketa v momente kliknutia. Zamyslite sa nad tým, kde je správne miesto na nastavenie rotácie.

Po vytvorení novej inštancie strely ju rovno otočíme smerom, ktorým je ovládaná raketa otočená. Keďže strelu vytvárame v triede **`Raketa`** pomocou metódy strielaj(), strelu môžeme otočiť pridaním zvýrazneného riadku do tejto metódy.

```
public void strielaj() {
     Strela strela = new Strela();
     this.getWorld().addObject(strela, this.getX(), this.getY());
     strela.setRotation(this.getRotation());
}
```

8 ZNIČENIE ZASIAHNUTÉHO ASTEROIDU

Strela počas svojho letu bude kontrolovať, či nie je dostatočne blízko nejakého asteroidu, ktorý by mala zničiť. Pretože asteroidov môže strela prekrývať naraz aj viac, budeme pracovať so zoznamami (listami) asteroidov. V knihe sa na podobné účely používa v úlohe 10.12 funkcia **`getOneIntersectingObject()`**. V našom prípade sa však hra skladá z okrúhlych objektov (asteroid, strela, čiastočne aj loď) a preto použijeme presnejší spôsob zisťovania kolízií ako v uvedenej úlohe. Docielime tým vizuálne lepší výsledok pri hraní hry a zároveň spresníme ovládanie.

ÚLOHA 8.1

Triede, v ktorej chceme zoznam asteroidov použiť, musíme najskôr oznámiť, že zoznam budeme používať. Zoznam (List) je samostatná trieda, ktorá už je naprogramovaná a vy si ju musíte importovať pomocou príkazu **`import java.util.List.`**

V triede **`Strela`** dopíšeme do sekcie import nasledovný riadok:

```
import java.util.List;
```

ÚLOHA 8.2

Získajte zoznam asteroidov, ku ktorým je strela dostatočne blízko na ich zničenie. Pomôže vám veľkosť obrázku strely a funkcia **`this.getObjectsInRange(vzdialenosť, typ)`**. Výsledok uložte do premennej typu **`List<Asteroid>`**, čiže zoznam asteroidov.
Funkcia hľadá objekty v kruhovej oblasti zo stredu strely. Preto hľadáme tie asteroidy, ktoré sa od strely nachádzajú do vzdialenosti rovnej *šírka obrázku strely / 2*.

```
List<Asteroid> trafeneAsteroidy =
     this.getObjectsInRange(this.getImage().getWidth() / 2,
     Asteroid.class);
```

ÚLOHA 8.3

Ak zoznam asteroidov nie je prázdny, znamená to, že strela zasiahla cieľ a má ho zničiť spolu so sebou. Ak teda zoznam asteroidov nejaký obsahuje, zvoľte prvý z nich a vymažte ho zo sveta spolu so strelou.

```
if (trafeneAsteroidy.size() > 0) {
     Asteroid trafenyAsteroid = trafeneAsteroidy.get(0);
     this.getWorld().removeObject(trafenyAsteroid);
     this.getWorld().removeObject(this);
}
```

9 NEÚSPEŠNÝ KONIEC HRY

V predošlej kapitole sme naučili strely ničiť asteroidy. Ďalším krokom pre funkčnosť hry je, aby sme naučili loď zistiť kolíziu s asteroidom – v takom prípade má totiž hra skončiť. Použitím rovnakých príkazov ako pri strele získame zoznam asteroidov, ktoré sú príliš blízko rakety. Pokiaľ nejaké sú, hru ukončíme príkazom **`Greenfoot.stop()`**.

V knihe v kapitole 7.2 Identifikácia zasiahnutých hráčov nájdete iný spôsob riešenia rovnakého problému. V úlohe 7.7 (v knihe) každá bomba pomocou metódy **`act()`** neustále zisťuje, či nezasiahla niektorého z hráčov. Následne môžete v úlohe 7.9 vidieť, že po zásahu hráča bombou je v triede **`Hrac`** vyvolaná metóda **`zasah()`**. V prípade hry Asteroidy by sme mohli použiť rovnaké riešenie a v triede **`Asteroid`** v metóde **`act()`** kontrolovať, či nedošlo ku kolízii s raketou. Zasiahnutú raketu môžeme v triede **`Asteroid`** získať napríklad pomocou metódy **`getOneIntersectingObject()`**. Keďže hráča (raketu) máme len jedného, bolo by zbytočné použiť metódu **`getObjectsInRange()`** pretože tá namiesto jedného objektu získava zoznam objektov. Následne skontrolujeme podmienku, či raketa vôbec bola zasiahnutá (nie je **`null`**) a ak áno, vyvoláme jej metódu **`zásah()`**.

```
Raketa zasiahnutaRaketa =
(Raketa)this.getOneIntersectingObject(Raketa.class);
if(zasiahnutaRaketa!= null) {
    zasiahnutaRaketa.zasah();
}
```

V triede **`Raketa`** by sme museli vytvoriť public metódu **`zasah()`** a v nej zavolať všetko, čo chceme, aby sa udialo po zásahu rakety (napísali by sme rovnaké príkazy ako v úlohe 9.2 alebo 9.3).

V predošlej kapitole sme vysvetľovali, že v našom prípade nie je použitie metódy **`getOneIntersectingObject()`** príliš vhodné, pretože objekty sú ohraničené skôr kružnicou ako obdĺžnikom. Metóda **`getOneIntersectingObject()`** sa pozerá na actorov ako na obdĺžniky a preto zvolíme iný spôsob kontrolovania kolízii medzi asteroidmi a raketou ako sme ukázali v predchádzajúcom texte a tiež aj v knihe.

ÚLOHA 9.1

Získajte zoznam asteroidov, ku ktorým je raketa príliš blízko, podobne ako v úlohe 9.1

```
List<Asteroid> trafeneAsteroidy =
     this.getObjectsInRange(this.getImage().getWidth() / 2,
     Asteroid.class);
```

ÚLOHA 9.2

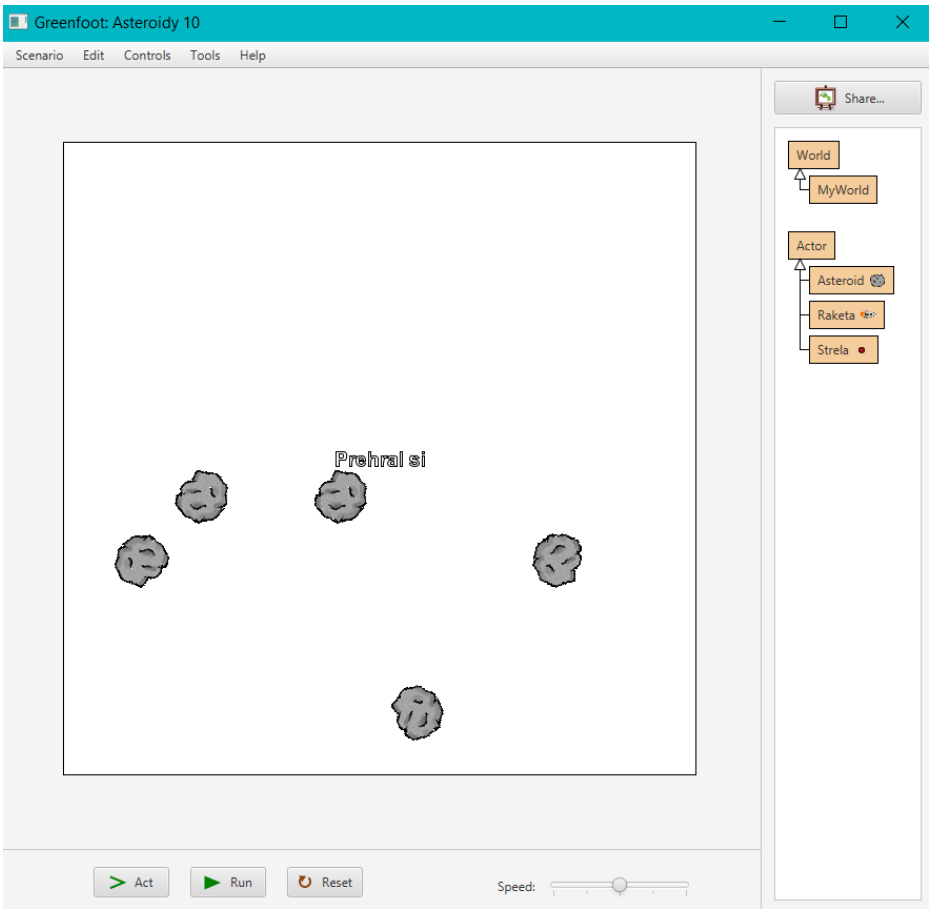
Ak raketa zasiahla nejaký asteroid, ukončite hru.
Pomocou metódy **`this.getWorld().showText()`** vypíšte informáciu o prehre.
Chceme, aby text bol zobrazený uprostred hracieho poľa. Vytvorte si lokálne premenné **`stredX`** a **`stredY`** typu **`int`**, do ktorých si uložíte x a y pozíciu stredu hracej plochy (sveta). Celkovú šírku sveta získate pomocou metódy **`this.getWorld().getWidth()`** a rovnako výšku zavolaním **`this.getWorld().getHeight()`**.
Parametre metódy **`showText()`** sú nasledovné: text, ktorý sa má zobraziť; x – pozícia zobrazeného textu; y – pozícia zobrazeného textu.
Následne odstráňte zničenú raketu z vesmíru zavolaním metódy **`this.getWorld().removeObject(this)`** – tento riadok zabezpečí, že **`MyWorld`** vymaže objekt, ktorý sme určili.
Hru ukončite zavolaním príkazu **`Greenfoot.stop()`**.

```
if (trafeneAsteroidy.size() > 0) {
     int stredX = this.getWorld().getWidth() / 2;
     int stredY = this.getWorld().getHeight() / 2;
     this.getWorld().showText("Prehral si", stredX, stredY);
     this.getWorld().removeObject(this);
     Greenfoot.stop();
```

ÚLOHA 9.3

Do prechádzajúcej úlohy doplňte riadok, ktorý zabezpečí zvukový efekt výbuchu po zasiahnutí vesmírnej lode asteroidom.
Do priečinka **Asteroidy\sounds** uložte zvuk znázorňujúci výbuch. Typ súboru musí byť **wav**.
Na prehratie zvuku zavolajte metódu **`Greenfoot.playSound("nazov.wav").`**

```
if (trafeneAsteroidy.size() > 0) {
     Greenfoot.playSound("vybuch.wav");
     int stredX = this.getWorld().getWidth() / 2;
     int stredY = this.getWorld().getHeight() / 2;
     this.getWorld().showText("Prehral si", stredX, stredY);
     this.getWorld().removeObject(this);
     Greenfoot.stop();
```



Obrázok 13 - Neúspešný koniec hry

10 ÚSPEŠNÝ KONIEC HRY

Výhra nastáva vtedy, keď sa vo svete nebudú nachádzať žiadne asteroidy. Túto kontrolu môžeme vykonávať v triede **`MyWorld`**, do ktorej pridáme metódu **`act()`**. Zoznam všetkých asteroidov vo svete získame príkazom **`this.getObjects(Asteroid.class)`**. Pokiaľ získaný zoznam neobsahuje žiadne asteroidy, opäť hráčovi oznámime výsledok hry ako v predošlej úlohe a ukončíme hru.

ÚLOHA 10.1

Získajte zoznam asteroidov vo svete, ktoré ešte neboli zničené.

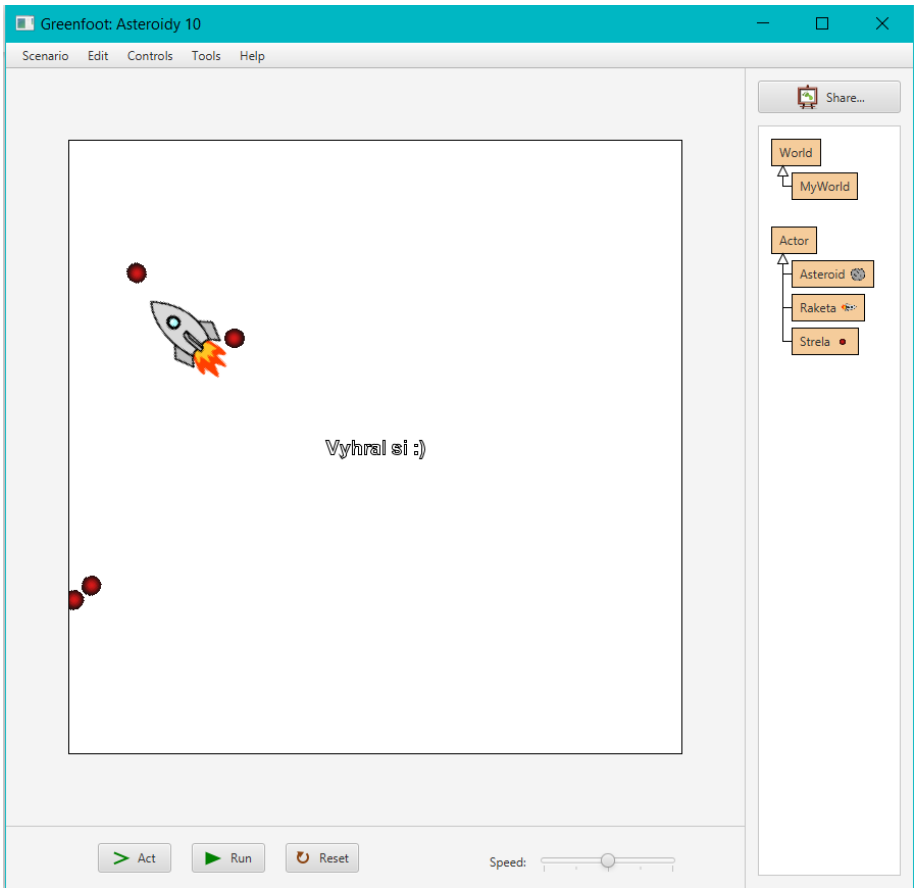
```
List<Asteroid> vsetkyAsteroidy =
     this.getObjects(Asteroid.class);
```

ÚLOHA 10.2

Ak už neexistujú žiadne asteroidy, oznámte hráčovi že vyhral a ukončite hru. Rovnako ako v úlohe 9.2, aj tu vypíšte informáciu o výhre do stredu hracej plochy.

Keďže sa nachádzame v triede **`MyWorld`**, informáciu o šírke a výške sveta dostaneme pomocou **`this.getWidth()`** a **`this.getHeight()`**. V úlohe 9.2. sme tieto informácie chceli získať z editora triedy **`Raketa`** a preto sme k metódam **`getWidth()`** a **`getHeight()`** museli pristupovať cez **`this.getWorld()`**.

```
if (vsetkyAsteroidy.size() == 0) {
     int stredX = this.getWidth() / 2;
     int stredY = this.getHeight() / 2;
     this.showText("Vyhral si :)", stredX, stredY);
     Greenfoot.stop();
}
```



Obrázok 14 - Úspešný koniec hry

INDEX OBRÁZKOV

BIBLIOGRAFIA

[1] Thorslotsdal, „Pixabay,“ 2017. [Online]. Available: https://pixabay.com/photos/space-nasa-purple-2901400/.