**CO452 Programming Concepts**

**PR1 Group Project - Source Code**

Game Title: METAUNIVERSE

Authors**: Name: Marius Boncica Bucks New ID: 22045429**

**Name: Martin Konecky Buck New ID: 22045464**

GitHub Repository: [CW2-CO452/Game1 at main · UnicoAI/CW2-CO452 (github.com)](https://github.com/UnicoAI/CW2-CO452/tree/main/Game1)

GitHub WIKI: [METAUNIVERSE GAME CO452 CW2 · UnicoAI/CW2-CO452 Wiki (github.com)](https://github.com/UnicoAI/CW2-CO452/wiki/METAUNIVERSE-GAME-CO452-CW2)

Video Metauniverse Game Playing: <https://screencast-o-matic.com/watch/c0Vq2kVwKKW>

# WEB PRESENTATION

<https://sway.office.com/GuGFPslhnNyFTSsG?ref=Link&loc=play>

|  |
| --- |
| [A picture containing satellite, night sky  Description automatically generated](https://sway.office.com/GuGFPslhnNyFTSsG?ref=email&loc=play) |
| [CO452 PROGRAMMING CONCEPTS](https://sway.office.com/GuGFPslhnNyFTSsG?ref=email&loc=play) |
| AUTHORS: Name: Marius Boncica Bucks New ID: 22045429 |
| |  | | --- | | [Go to this Sway](https://sway.office.com/GuGFPslhnNyFTSsG?ref=email&loc=play) | |

Graphical user interface, website

Description automatically generated

Table of Contents

[Authors**:** 1](#_Toc125110295)

[Galaxy.class 1](#_Toc125110296)

[Galaxy2ndLevel.class 1](#_Toc125110297)

[Galaxy3rdLevel.class 1](#_Toc125110298)

[Menu.class 1](#_Toc125110299)

[Myworld.class 1](#_Toc125110300)

[Scroll.class 1](#_Toc125110301)

[WorldLevel1.class 1](#_Toc125110302)

[Single.class 1](#_Toc125110303)

[Counter.class 1](#_Toc125110304)

[GalaxySingle.class 1](#_Toc125110305)

[GalaxySingleMenu.class 1](#_Toc125110306)

[Obstacle.Class 1](#_Toc125110307)

[Play.class 1](#_Toc125110308)

[Playmenu.class 1](#_Toc125110309)

[PlaySingle.class 1](#_Toc125110310)

[PlaySingleMenu.Class 1](#_Toc125110311)

[Player1.class 1](#_Toc125110312)

[Player2.class 1](#_Toc125110313)

[PlayerSingle.class 1](#_Toc125110314)

[Projectile.class 1](#_Toc125110315)

[Projectile2.class 1](#_Toc125110316)

[ScrollActor.class 1](#_Toc125110317)

[Earth.class 1](#_Toc125110318)

[Enemy.class 1](#_Toc125110319)

[Mars.class 1](#_Toc125110320)

[Plane.class 1](#_Toc125110321)

[Sun.class 1](#_Toc125110322)

[Step1.class 1](#_Toc125110323)

[Prize.class 1](#_Toc125110324)

[Speedbooster.class 1](#_Toc125110325)

[Gifimage.class 1](#_Toc125110326)

[Simpletimer.class 1](#_Toc125110327)

# Galaxy.class

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Metauniverse Game level 1 description intro

\*

\* @author (Marius Boncica)

\* @version (version2 10/01/2022)

\*/

public class Galaxy extends World

{

/\*\*

\* Constructor for objects of class Quit.

\*

\*/

public Galaxy()

{

// Create a new world with 900x600 cells with a cell size of 1x1 pixels.

super(900,600,1);

//add button to forward on WorlLevel1 when clicked

GalaxySingle button2 = new GalaxySingle();

addObject(button2,470,450); //set position bottom for button

//arrange text display on screen

showText("Single Player - score: " + PlayerSingle.scor, 470, 50);

showText("Drive Space Ship in the Universe To Find Earth like Planet ", 300, 100);

showText("Use Arrows to navigate ", 170, 140);

showText("Mouse click to Shoot Aliens ", 170, 180);

showText("Avoid Touching Sun or Aliens ", 170,220);

showText("Time Limit 1 Minute", 170, 260);

showText("Galaxy View", 470, 480);

}

//set background black and add starts created with creatStars method

public void prepare(){

GreenfootImage bkg = getBackground();

bkg.setColor(Color.BLACK);

bkg.fill();

createStars(300);

}

//create method for stars

private void createStars(int number)

{

GreenfootImage background = getBackground();

for (int i=0; i < number; i++)

{ //declare variables used for X Y position

int x = Greenfoot.getRandomNumber( getWidth() );

int y = Greenfoot.getRandomNumber( getHeight() );

int color = 150 - Greenfoot.getRandomNumber(120);

//set position

background.setColorAt(x, y, new Color(color,color,color));

}

}

}

# Galaxy2ndLevel.class

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Metauniverse Game level 2 description intro

\*

\* @author (Marius Boncica)

\* @version (version2 10/01/2022)

\*/

public class Galaxy2ndLevel extends World

{

/\*\*

\* Constructor for objects of class Quit.

\*

\*/

public Galaxy2ndLevel()

{

// Create a new world with 900x600 cells with a cell size of 1x1 pixels.

super(900,600,1);

//add button to forward on Galaxy2ndLevel when clicked

PlaySingle button = new PlaySingle();

addObject(button,470,450);

//arrange text display on screen

showText("Single Player - score: " + PlayerSingle.scor, 470, 50);

showText("Drive Space Ship Around Earth To Protect Earth ", 300, 100);

showText("Use Arrows to navigate ", 170, 140);

showText("Mouse click to Shoot Aliens ", 170, 180);

showText("Collect Points By Touching Satelites ", 200,220);

showText("Press Space Bar To Remove Steroids--Needed 10 points", 300, 260);

showText("Touch space balls to increase speed", 200, 300);

showText("Time Limit 1 Minute", 170, 340);

showText("Start", 470, 480);

}

//set background black and add starts created with creatStars method

public void prepare(){

GreenfootImage bkg = getBackground();

bkg.setColor(Color.BLACK);

bkg.fill();

createStars(300);

}

//create method for stars

private void createStars(int number)

{

GreenfootImage background = getBackground();

for (int i=0; i < number; i++)

{

int x = Greenfoot.getRandomNumber( getWidth() );

int y = Greenfoot.getRandomNumber( getHeight() );

int color = 150 - Greenfoot.getRandomNumber(120);

background.setColorAt(x, y, new Color(color,color,color));

}

}

}

# Galaxy3rdLevel.class

/\*\*

\* Metauniverse Game level 3 description intro

\*

\* @author (Marius Boncica)

\* @version (version2 10/01/2022)

\*/

public class Galaxy3rdLevel extends World

{

/\*\*

\* Constructor for objects of class Quit.

\*

\*/

public Galaxy3rdLevel()

{

// Create a new world with 900x600 cells with a cell size of 1x1 pixels.

super(900,600,1);

//add button to forward on Galaxy3rdLevel when clicked

Play button = new Play();

addObject(button,470,450);

//add text to screen and define position X AND Y

showText("Player 1 - score: " + Player1.scor, 150, 50);

showText("Player 2 - score: " + Player2.scor, 750, 50);

showText("META UNIVERSE ", 470, 270);

showText("Drive Space Ship Around Earth To Protect Earth ", 300, 100);

showText("Use Arrows and W A S D to navigate ", 190, 140);

showText("Mouse click to Shoot Aliens ", 170, 180);

showText("Collect Points By Touching Satelites ", 200,220);

showText("Press Space Bar To Remove Steroids--Needed 10 points", 300, 260);

showText("Touch space balls to increase speed", 200, 300);

showText("CLICK AND Q TO SHOOT", 170, 340);

showText("Time Limit 1 Minute", 170, 380);

showText("Start", 470, 480);

}

//set background black and add starts created with creatStars method

public void prepare(){

GreenfootImage bkg = getBackground();

bkg.setColor(Color.BLACK);

bkg.fill();

createStars(300);

}

//create method for stars

private void createStars(int number)

{

GreenfootImage background = getBackground();

for (int i=0; i < number; i++)

{

int x = Greenfoot.getRandomNumber( getWidth() );

int y = Greenfoot.getRandomNumber( getHeight() );

int color = 150 - Greenfoot.getRandomNumber(120);

background.setColorAt(x, y, new Color(color,color,color));

}

}

}

# Menu.class

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Menu start class of METAUNIVERSE GAME

\*

\* @author (Marius Boncica)

\* @version (version2 10/01/2022)

\*/

public class Menu extends World

{

/\*\*

\* Constructor for objects of class Quit.

\*

\*/

public Menu()

{

// Create a new world with 900x600 cells with a cell size of 1x1 pixels.

super(900,600,1);

//add buttons to forward on click on the game level selected

PlayMenu button = new PlayMenu();

addObject(button,270,450);

PlaySingleMenu button1 = new PlaySingleMenu();

addObject(button1,670,450);

GalaxySingleMenu button2 = new GalaxySingleMenu();

addObject(button2,470,450);

//show text and positioning on screen

showText("Player 1 - score: " + Player1.scor, 150, 50);

showText("Player 2 - score: " + Player2.scor, 750, 50);

showText("Single Player - score: " + PlayerSingle.scor, 470, 50);

showText("META UNIVERSE ", 470, 270);

showText("Multi Players ", 270, 480);

showText("Single Player", 670, 480);

showText("Galaxy View", 470, 480);

}

//set the background

public void prepare(){

GreenfootImage bkg = getBackground();

bkg.setColor(Color.BLACK);

bkg.fill();

createStars(300);

}

//method to create stars

private void createStars(int number)

{

GreenfootImage background = getBackground();

for (int i=0; i < number; i++)

{

int x = Greenfoot.getRandomNumber( getWidth() );

int y = Greenfoot.getRandomNumber( getHeight() );

int color = 150 - Greenfoot.getRandomNumber(120);

background.setColorAt(x, y, new Color(color,color,color));

}

}

}

# Myworld.class

import java.util.Random;

/\*\*

\* \*Metauniverse Multiplayer level

\* Author Marius Daniel Boncica

\* updated 14/12/2022

\*/

/\*\*

\*

\*

\*/

public class MyWorld extends World

{

//create Counter objects for player1 life and player 2 life counters

public static Counter player1Life = new Counter();

public static Counter player2Life = new Counter();

//declare int timer and fix value for 1 minute

private int timer = 1\*60\*55; //about 1 min at normal speed

//initiat mao and create table to arrange objects od specified positions

int[][] map= {

{1,1,1,3,1,1,6,1,1,1,1,1,1,1,1,2},

{1,1,1,3,6,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,6,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,6,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{5,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

};

//create object random

Random randgen = new Random();

//declare container, timer , Booster

public static int the\_width = 900, the\_height = 600, cell\_size = 1;

int cadran = 1, cadranTimerCounter = 1, cadranTimer = 250, spawnTimerCounterPrize = 0, spawnTimerPrize = 130, spawnTimerCounterBooster = 0, spawnTimerBooster = 260;

//create objects of players

Player1 alfa = new Player1("R.png");

Player2 beta = new Player2("R1.png");

//-----------------CONSTRUCTOR-------------------------

public MyWorld()

{ //define world size

super(900,600,1);

//use if function when creating objects and store objects to array

//array position will be refered in map above

for(int i = 0; i<16;i++)

for (int j =0; j<16; j++)

if (map[i][j]==0)

//create object of step1 and add it to array position 0

addObject(new Step1("sunearth.gif"),j\*50,i\*50);

//create object of Obstacleand add it to array position 2

else if(map[i][j] == 2)

addObject(new Obstacle("blackhole.png"),j\*50,i\*50);

//create object of Obstacle and add it to array position 3

else if (map[i][j] == 3)

addObject(new Obstacle("beeper.png"),j\*50,i\*50);

//create object of step1 and add it to array position 4

else if (map[i][j] == 4)

addObject(new Step1("sunearth.gif"),j\*50,i\*50);

else if (map[i][j] == 5)

addObject(new Asteroid("asteroid.png"),j\*50,i\*50);

else if (map[i][j] == 6){

addObject(new Obstacle("blackhole.png"),j\*50,i\*50);}

else if (map[i][j] == 7)

addObject(new Asteroid("asteroid.png"),j\*50,i\*50);

//call methods for this class object

prepare();

continuing();

addObject(player1Life, 350,20);

addObject(player2Life, 550,20);

player1Life.setValue(5);

player2Life.setValue(5);

Player1 p1 = new Player1("R.png");

addObject(p1, 350,400);

Player2 p2 = new Player2("R1.png");

addObject(p2,650,400);

}

//-------------Set background-------------------

private void prepare(){

GreenfootImage bkg = getBackground();

bkg.setColor(Color.BLACK);

bkg.fill();

createStars(300);

}

//-----------------set music--------------\*/

private void continuing(){

setMusic();

tinScor();

}

//--------------------act method to display timer and score---------------\*/

public void act(){

if( Greenfoot.mouseClicked(null) )

addObject(new prize(), 10 + randgen.nextInt()%(getWidth() - 10 + 1), 10 + randgen.nextInt()%(getHeight() - 10 + 1) );

checkForSpawningPrizes();

checkForSpawningBoosters();

tinScor();

alertAsteroid();

if (--timer == 0)

{

removeObjects(getObjects(null));

if(Player1.scor > Player2.scor){

showText("Player1 WON ||| Score: " + Player1.scor, 350,200);

showText("Player2 Loose ||| Score "+ Player2.scor, 440,300);}

else if(Player1.scor == Player2.scor){

showText("Player1 and Player2 WON ||| Score: " + Player1.scor, 450,300);

}

else if(Player1.scor < Player2.scor){showText("Player2 WON ||| Score " + Player2.scor, 450,200);

showText("Player1 Loose ||| Score "+ Player1.scor, 450,300);

}

//create a delay when time end

Greenfoot.delay(10);

//quit word and enter new world

Greenfoot.setWorld(new Menu());

}

}

//method to animate speed booster object

private void checkForSpawningBoosters(){

spawnTimerCounterBooster = (spawnTimerCounterBooster + 1)%spawnTimerBooster;

if( spawnTimerCounterBooster == 0){

addObject(new speedBooster(), 1 + randgen.nextInt()%900, 1 + randgen.nextInt()%600);

addObject(new speedBooster(), 10 + Greenfoot.getRandomNumber( the\_width - 2\*10 ), 10 + Greenfoot.getRandomNumber( the\_height - 2\*10 ) );

if(spawnTimerBooster == 0) spawnTimerBooster = 260; //initial value

}

}

//method to animate prize object

private void checkForSpawningPrizes(){

spawnTimerCounterPrize = (spawnTimerCounterPrize + 1)%spawnTimerPrize;

if( spawnTimerCounterPrize == 0 ){

addObject(new prize(), 1 + randgen.nextInt()%900, 1 + randgen.nextInt()%600);

addObject(new prize(), 10 + Greenfoot.getRandomNumber( the\_width - 2\*10 ), 10 + Greenfoot.getRandomNumber( the\_height - 2\*10 ) );

spawnTimerPrize -= 1; //speed points selectors

if(spawnTimerPrize == 0) spawnTimerPrize = 130;//initial value

}

}

//set music -create object and declare song, volume and loop

private void setMusic(){

GreenfootSound backgroundMusic = new GreenfootSound("nice\_sp.mp3");

backgroundMusic.setVolume(15);

backgroundMusic.playLoop();

}

//method to create stars

private void createStars(int number)

{

GreenfootImage background = getBackground();

for (int i=0; i < number; i++)

{

int x = Greenfoot.getRandomNumber( getWidth() );

int y = Greenfoot.getRandomNumber( getHeight() );

int color = 150 - Greenfoot.getRandomNumber(120);

background.setColorAt(x, y, new Color(color,color,color));

}

}

//method Prize OBJECT ON CLICK EVENT

private void spawnPrize(){

if( Greenfoot.mouseClicked(null) ){

addObject(new prize(), 10 + randgen.nextInt()%(getWidth() - 10 + 1), 10 + randgen.nextInt()%(getHeight() - 10 + 1) );

}

}

//method SPEEDbOOSTER ONclick event

private void spawnSpeedBooster(){

if( Greenfoot.mouseClicked(null) ){

addObject(new speedBooster(), 30+randgen.nextInt()%(getWidth() - 60 + 1), 30 + randgen.nextInt()%(getHeight() - 60 + 1) );

}

}

//method score to display score for players

private void tinScor(){

// player1:

showText("Player 1- score: " + Player1.scor, 100, 20);

showText("Speed: " + (float)((int)(Player1.speed\*1000) )/1000, 100, 40);

// player2 2:

showText("Player 2- score: " + Player2.scor, 800, 20);

showText("Speed: " + (float)((int)(Player2.speed\*1000) )/1000, 800, 40);

// timer:

showText("Time left: "+ timer/(60\*55) + (timer%(60\*55))/55, 450, 100 );

showText("Position " + cadran, the\_width/2, 10);

}

//create alert when scor>10 alert to remove asteroid

private void alertAsteroid(){

if(Player1.scor>=10 && Player2.scor>=10 ){

showText("GO TO ASTEROID AND PRESS SPACE TO REMOVE IT", 400, 200);

}

}

}

# Scroll.class

/\*\*

\* CLASS: Scroll (subclass of World)<br>

\* AUTHOR: danpost (greenfoot.org username)<br>

\* DATE: February 9, 2015<br>

\* MODIFIED: February 10, 2015 (made grid world compatible)

\* <br>

\* DESCRIPTION: a support superclass for an infinite scrolling world;

\* capable of horizontal, vertical or universal scrolling;

\* scrolls either independent of any actor or by following an actor;

\* scrolls background with or without actor relocations;

\* capable of scrolling actors without scrolling the background image;

\* capable of limited scrolling as access to field values are provided

\* through getter methods;<br>

\* <br>

\* The only limitation, which may be dealt with in a later update, is that

\* the scrolling background image is obtained from the world background

\* set at the time the direction of scrolling is set -- however, this would

\* complicate the code with more fields, checks and controls;

\*/

public class Scroll extends World

{

private GreenfootImage bg; // the scrolling background image

private int scrollDirX, scrollDirY; // the scrolling directions along both axes

private int scrollSpeed; // the scrolling speed

private int scrollX, scrollY; // the current scrolled offset values

/\*\* basic World constructor call \*/

public Scroll(int w, int h, int c) { super(w, h, c); }

/\*\* basic World constructor call (2) \*/

public Scroll(int w, int h, int c, boolean b) { super(w, h, c, b); }

/\*\* sets the direction for future scrolling (left, right, up, or down);

\* the current world background image will be used for scrolling;

\* @param dir the rotational angle for movement divided by 90 \*/

public void setDirection(int dir)

{

// determine scrolling directions along both axes

dir = dir%4;

scrollDirX = (1-dir)\*((dir+1)%2);

scrollDirY = (2-dir)\*(dir%2);

// initialize fields for background image scrolling

bg = new GreenfootImage(getBackground());

scrollX = 0;

scrollY = 0;

}

/\*\*

\* scrolls the background image using current field values

\*/

public void scrollBackground()

{

// set new scroll offsets

scrollX = (scrollX-scrollDirX\*scrollSpeed+getWidth())%getWidth();

scrollY = (scrollY-scrollDirY\*scrollSpeed+getHeight())%getHeight();

// draw scrolling image onto background image

getBackground().drawImage

(

bg,

scrollX\*getCellSize(),

scrollY\*getCellSize()

);

getBackground().drawImage

(

bg,

((scrollX-getWidth())\*scrollDirX\*scrollDirX)\*getCellSize(),

((scrollY-getHeight())\*scrollDirY\*scrollDirY)\*getCellSize()

);

}

/\*\* scrolls all movable actors using current field values;<br><br>

\* NOTE: this method is for use in projects using the QActor class by danpost;

\* if not using that class, remove this method and un-comment the other

\* 'scrollActors' method \*/

public void scrollActors()

{

// scroll actors in world

for (Object obj : getObjects(null))

{

// see if actor is of QActor type

if (obj instanceof ScrollActors)

{

// scroll QActor object

ScrollActors actor = (ScrollActors)obj;

actor.setQLocation

(

actor.getQX()-scrollDirX\*scrollSpeed\*ScrollActors.QVAL,

actor.getQY()-scrollDirY\*scrollSpeed\*ScrollActors.QVAL

);

}

else

{

// scroll non-QActor object

Actor actor = (Actor)obj;

actor.setLocation

(

actor.getX()-scrollDirX\*scrollSpeed,

actor.getY()-scrollDirY\*scrollSpeed

);

}

}

}

/\*\* scrolls all movable actors using current field values;<br><br>

\* NOTE: this method is for use in projects not using the QActor class by danpost;

\* if not using that class, remove the other 'scrollActors' method and un-comment

\* this one \*/

// public void scrollActors()

// {

// // scroll actors in world

// Actor actor = (Actor)obj;

// actor.setLocation

// (

// actor.getX()-scrollDirX\*scrollSpeed,

// actor.getY()-scrollDirY\*scrollSpeed

// );

// }

/\*\* scrolls the background and all movable actors using current field values \*/

public void scrollAll()

{

// scroll background image

scrollBackground();

// scroll actors in world

scrollActors();

}

/\*\* scrolls the background and all movable actors using current field values so that

\* the given actor is scrolled to the location coordinates given;

\* @param actor the actor whose location is to be adjusted

\* @param x the coordinate along the x-axis at which the actor is to be placed

\* @param y the coordinate along the y-axis at which the actor is to be placed \*/

public void setActorAt(Actor actor, int x, int y)

{

// ensure actor is in world

if (actor.getWorld() != this) return;

// scroll all horizontally getting actor to its new x-coordinate location

setScrollSpeed(actor.getX()-x);

setDirection(0);

scrollAll();

// scroll all vertically getting actor to its new y-coodinate location

setScrollSpeed(actor.getY()-y);

setDirection(1);

scrollAll();

}

/\*\* returns the scrolling direction along the x-axis

\* ('1' for right, '0' for none and '-1' for left)

\* @return direction of horizontal scrolling \*/

public int getDirectionX() { return scrollDirX; }

/\*\* returns the scrolling direction along the y-axis

\* ('1' for down, '0' for none and '-1' for up)

\* @return direction of vertical scrolling \*/

public int getDirectionY() { return scrollDirY; }

/\*\* returns the current scrolled amount along the x-axis

\* @return current distance horizontally scrolled \*/

public int getScrollX() { return scrollX; }

/\*\* returns the current scrolled amount along the y-axis

\* @return current distance vertically scrolled \*/

public int getScrollY() { return scrollY; }

/\*\* returns the current value set for the speed of scrolling

\* @return the current scrolling speed \*/

public int getScrollSpeed() { return scrollSpeed; }

/\*\* sets the value for the speed of future scrolling to the given value

\* @param speed the value to set the speed of scrolling to \*/

public void setScrollSpeed(int speed) { scrollSpeed = speed; }

}

# WorldLevel1.class

/\*\*

\* MetaUniverse level Galaxy view single player

\*

\* @author (Marius Boncica & Martin Konecny)

\* @version (version2 10/01/2022)

\*/

public class WorldLevel1 extends Scroll

{//declare variables

private Actor ship; // the main actor

public static Counter planeLife = new Counter(); //counter

private int timer = 1\*60\*55; //time limit to 1 minute

//define constructor

public WorldLevel1()

{

super(800, 600, 1);// the background image

//image scale

GreenfootImage image = new GreenfootImage("galaxy.jpg");

image.scale(getWidth(), getHeight());

setBackground(image);

// set scrolling speed and direction

setDirection(0);

setScrollSpeed(2);

// the ship

addObject(ship = new Plane("spaces11.png"), getWidth()/8, getHeight()/8);

// add enemey object

addObject(Enemy.scoreDisp = new Actor(){}, 80, 20);

Enemy.scoreDisp.setImage(new GreenfootImage(1, 1));

Enemy.created = 0;

// the actor to display the score

addObject(Sun.scoreDisp = new Actor(){}, 40, 40);

Earth.created = 0;

Earth.unhit = 0;

// the actor to display the score

addObject(Earth.scoreDisp = new Actor(){}, 30, 60);

Mars.created = 0;

Mars.unhit = 0;

// the actor to display the score

addObject(Mars.scoreDisp = new Actor(){}, 20, 80);

}

public void act()

{

// keeps ship center screen

setActorAt(ship, getWidth()/2, getHeight()/2);

//add text on screen to customize user experience

showText("WELCOME TO META UNIVERSE ", 470, 20);

showText("Score: " + Plane.score, 100, 40);

// timer:

showText("Time left: "+ (timer%(60\*55))/55, 450, 40 );

// change worlds

if ("enter".equals(Greenfoot.getKey())) {

Greenfoot.setWorld(new Menu());}

// scroll background image

scrollBackground();

// spawn alienships (with increasing frequency)

if (Greenfoot.getRandomNumber(200-Enemy.created/10) == 0)

{

int y = Greenfoot.getRandomNumber(getHeight());

addObject(new Enemy(), getWidth()+40, y);

}

if (Greenfoot.getRandomNumber(200-Sun.created/10) == 0)

{

int y = Greenfoot.getRandomNumber(getHeight());

addObject(new Sun(), getWidth()+40, y);

}

if (Greenfoot.getRandomNumber(200-Earth.created/10) == 0)

{

int y = Greenfoot.getRandomNumber(getHeight());

addObject(new Earth(), getWidth()+40, y);

}

if (Greenfoot.getRandomNumber(250-Mars.created/10) == 0)

{

int y = Greenfoot.getRandomNumber(getHeight());

addObject(new Mars(), getWidth()+40, y);

}

//function when time ends and display text according the number of points

if (--timer == 0)

{

removeObjects(getObjects(null));

Greenfoot.stop();

if(Plane.score >= 0){

showText("YOU WON ||| Score: " + Plane.score, 350,200);

}

else if(Plane.score < 0){

showText("You Loose ||| Score "+ Plane.score, 450,300);

}

//delay response

Greenfoot.delay(5);

//enter new world

Greenfoot.setWorld(new Menu());

}

}

}

# Single.class

import java.util.Random;

/\*\*

\* \*Metauniverse game level 2 single player world

\* Author Marius Daniel Boncica and Martin Konecny

\* updated 14/12/2022

\*/

public class Single extends World

{//create an object on counter class

public static Counter playerSingleLife = new Counter();

//declare int timer and fix value to 1 minute

private int timer = 1\*60\*55; //about 1 min at normal speed

//initiate map table made by array of objects items created

int[][] map= {

{1,1,1,3,1,1,6,1,1,1,1,1,1,1,1,2},

{1,1,1,3,6,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,6,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,6,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{5,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}

};

//declare variables as timer, counter, points, booster, spees

Random randgen = new Random();

public static int the\_width = 900, the\_height = 600, cell\_size = 1;

int spawnTimerCounterPrize = 0, spawnTimerPrize = 130, spawnTimerCounterBooster = 0, spawnTimerBooster = 260;

//create an object of class PlayerSingle

PlayerSingle alfa = new PlayerSingle("R1.png");

//-----------------CONSTRUCTOR-------------------------

public Single()

{

super(900,600,1);

for(int i = 0; i<16;i++)

for (int j =0; j<16; j++)

//create an array of objects and allocate objects to specific location

//use array index to position on map above

if (map[i][j]==0)

addObject(new Step1("sunearth.gif"),j\*50,i\*50);

else if(map[i][j] == 2)

addObject(new Obstacle("blackhole.png"),j\*50,i\*50);

else if (map[i][j] == 3)

addObject(new Obstacle("beeper.png"),j\*50,i\*50);

else if (map[i][j] == 4)

addObject(new Step1("sunearth.gif"),j\*50,i\*50);

else if (map[i][j] == 5)

addObject(new Asteroid("asteroid.png"),j\*50,i\*50);

else if (map[i][j] == 6){

addObject(new Obstacle("blackhole.png"),j\*50,i\*50);}

else if (map[i][j] == 7)

addObject(new Asteroid("asteroid.png"),j\*50,i\*50);

//call methods for this class

prepare();

continuing();

addObject(new PlayerSingle("R1.png"),450,450);

addObject(playerSingleLife, 470,20);

playerSingleLife.setValue(0);

}

//-------------Set Background-------------------

private void prepare(){

GreenfootImage bkg = getBackground();

bkg.setColor(Color.BLACK);

bkg.fill();

createStars(300);

}

//-----------------set music--------------\*/

private void continuing(){

setMusic();

tinScor();

}

//--------------------method act ---------------\*/

public void act(){

if( Greenfoot.mouseClicked(null) )

addObject(new prize(), 10 + randgen.nextInt()%(getWidth() - 10 + 1), 10 + randgen.nextInt()%(getHeight() - 10 + 1) );

//call bolean function

checkForSpawningPrizes();

checkForSpawningBoosters();

//display score

tinScor();

//ALERT REMOVE ASTEROID

alertAsteroid();

//completion text to show on screen related to score value

if (--timer == 0)

{

removeObjects(getObjects(null));

if(Player1.scor > 0 ){

showText("YOU WON ||| Score: " + PlayerSingle.scor, 350,200);

}

else if(Player1.scor == 0){

showText("Try Again " + PlayerSingle.scor, 450,300);

}

Greenfoot.delay(10);

//delay next step

Greenfoot.setWorld(new Menu());

//quit world

}

}

//booster method animation

private void checkForSpawningBoosters(){

spawnTimerCounterBooster = (spawnTimerCounterBooster + 1)%spawnTimerBooster;

if( spawnTimerCounterBooster == 0){

addObject(new speedBooster(), 1 + randgen.nextInt()%900, 1 + randgen.nextInt()%600);

addObject(new speedBooster(), 10 + Greenfoot.getRandomNumber( the\_width - 2\*10 ), 10 + Greenfoot.getRandomNumber( the\_height - 2\*10 ) );

if(spawnTimerBooster == 0) spawnTimerBooster = 260; //initial value

}

}

//prize objects method animation

private void checkForSpawningPrizes(){

spawnTimerCounterPrize = (spawnTimerCounterPrize + 1)%spawnTimerPrize;

if( spawnTimerCounterPrize == 0 ){

addObject(new prize(), 1 + randgen.nextInt()%900, 1 + randgen.nextInt()%600);

addObject(new prize(), 10 + Greenfoot.getRandomNumber( the\_width - 2\*10 ), 10 + Greenfoot.getRandomNumber( the\_height - 2\*10 ) );

spawnTimerPrize -= 1; //speed points selectors

if(spawnTimerPrize == 0) spawnTimerPrize = 130;//initial value

}

}

private void setMusic(){

/\*

\* set music

\* Skrillex - Scary Monsters And Nice Sprites

\*/

GreenfootSound backgroundMusic = new GreenfootSound("nice\_sp.mp3");

backgroundMusic.setVolume(15);

backgroundMusic.playLoop();

}

//method to create stars

private void createStars(int number)

{

GreenfootImage background = getBackground();

for (int i=0; i < number; i++)

{

int x = Greenfoot.getRandomNumber( getWidth() );

int y = Greenfoot.getRandomNumber( getHeight() );

int color = 150 - Greenfoot.getRandomNumber(120);

background.setColorAt(x, y, new Color(color,color,color));

}

}

/\*\* method prize object on click event \*/

private void spawnPrize(){

if( Greenfoot.mouseClicked(null) ){

addObject(new prize(), 10 + randgen.nextInt()%(getWidth() - 10 + 1), 10 + randgen.nextInt()%(getHeight() - 10 + 1) );

}

}

/\*\* method spedBooster object on click event \*/

private void spawnSpeedBooster(){

if( Greenfoot.mouseClicked(null) ){

addObject(new speedBooster(), 30+randgen.nextInt()%(getWidth() - 60 + 1), 30 + randgen.nextInt()%(getHeight() - 60 + 1) );

}

}

//add text with score and speed to be displayed on screen

private void tinScor(){

// player1:

showText("Score: " + PlayerSingle.scor, 100, 20);

showText("Speed: " + (float)((int)(PlayerSingle.speed\*1000) )/1000, 100, 40);

// timer:

showText("Time left: "+ timer/(60\*55) + (timer%(60\*55))/55, 450, 100 );

}

//create alert when scor>10 alert to remove asteroid

private void alertAsteroid(){

if(PlayerSingle.scor>=10 ){

showText("GO TO ASTEROID AND PRESS SPACE TO REMOVE IT", 400, 200);

}

}

}

# Counter.class

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* A Counter class that allows you to display a numerical value on screen.

\*

\* The Counter is an actor, so you will need to create it, and then add it to

\* the world in Greenfoot. If you keep a reference to the Counter then you

\* can adjust its value. Here's an example of a world class that

\* displays a counter with the number of act cycles that have occurred:

\*

\* <pre>

\* class CountingWorld

\* {

\* private Counter actCounter;

\*

\* public CountingWorld()

\* {

\* super(600, 400, 1);

\* actCounter = new Counter("Act Cycles: ");

\* addObject(actCounter, 100, 100);

\* }

\*

\* public void act()

\* {

\* actCounter.setValue(actCounter.getValue() + 1);

\* }

\* }

\* </pre>

\*

\* @updates made by Marius Boncica

\* @version 3.0

\*/

public class Counter extends Actor

{

private static final Color transparent = new Color(0,0,0,0);

private GreenfootImage background;

private int value;

private int target;

private String prefix;

public Counter()

{

this(new String());

}

/\*\*

\* Create a new counter, initialised to 0.

\*/

public Counter(String prefix)

{

background = getImage(); // get image from class

value = 0;

target = 0;

this.prefix = prefix;

updateImage();

}

/\*\*

\* Animate the display to count up (or down) to the current target value.

\*/

public void act()

{

if (value < target) {

value++;

updateImage();

}

else if (value > target) {

value--;

updateImage();

}

}

/\*\*

\* Add a new score to the current counter value. This will animate

\* the counter over consecutive frames until it reaches the new value.

\*/

public void add(int score)

{

target += score;

}

/\*\*

\* Return the current counter value.

\*/

public int getValue()

{

return target;

}

/\*\*

\* Set a new counter value. This will not animate the counter.

\*/

public void setValue(int newValue)

{

target = newValue;

value = newValue;

updateImage();

}

/\*\*

\* Sets a text prefix that should be displayed before

\* the counter value (e.g. "Score: ").

\*/

public void setPrefix(String prefix)

{

this.prefix = prefix;

updateImage();

}

/\*\*

\* Update the image on screen to show the current value.

\*/

private void updateImage()

{

GreenfootImage image = new GreenfootImage(background);

GreenfootImage text = new GreenfootImage(prefix + value, 22, Color.BLACK, transparent);

if (text.getWidth() > image.getWidth() - 20)

{

image.scale(text.getWidth() + 20, image.getHeight());

}

image.drawImage(text, (image.getWidth()-text.getWidth())/2,

(image.getHeight()-text.getHeight())/2);

setImage(image);

}

}

# GalaxySingle.class

/\*\*

\* Game Cw2

\*

\* @Marius Boncica & Martin Konecny

\* @version 12

\*/

public class GalaxySingle extends Actor

{ GreenfootImage myGif = new GreenfootImage("Play1.png");

/\*\*

\* create an object og gif image

\* Constructor class-adjust Height and width

\*/

public GalaxySingle(){

GreenfootImage myGif = getImage();

int h = (int)myGif.getHeight()/1;

int w = (int)myGif.getWidth()/2;

myGif.scale(h,w);

}

// act method to describe animarion behaviour of class when clicked

public void act()

{if(Greenfoot.mousePressed(this))

{getImage().scale((int)Math.round(getImage().getWidth()\*0.9),

(int)Math.round(getImage().getHeight()\*0.9));

}

// Add your action code here.

if(Greenfoot.mouseClicked(this))

{ Greenfoot.delay(5);

Greenfoot.setWorld(new WorldLevel1());

}

}

}

# GalaxySingleMenu.class

/\*\*

\* Game Cw2

\*

\* @Marius Boncica & Martin Konecny

\* @version 12

\*/

public class GalaxySingleMenu extends Actor

//create object of gif image

{ GreenfootImage myGif = new GreenfootImage("Play1.png");

/\*\*

\* Constructor

\* scale image

\*/

public GalaxySingleMenu(){

GreenfootImage myGif = getImage();

int h = (int)myGif.getHeight()/1;

int w = (int)myGif.getWidth()/2;

myGif.scale(h,w);

}

//method to animate behaviour on click event

public void act()

{if(Greenfoot.mousePressed(this))

{getImage().scale((int)Math.round(getImage().getWidth()\*0.9),

(int)Math.round(getImage().getHeight()\*0.9));

}

// Add your action code here.

if(Greenfoot.mouseClicked(this))

{ Greenfoot.delay(5);

//on click event enter word Galaxy

Greenfoot.setWorld(new Galaxy());

}

}

}

# Obstacle.Class

/\*\*

\* update

\*

\* @author Marius Boncica

\* @version (1.01 8/12/2022)

\*/

public class Obstacle extends Actor

{

/\*\*

\* declare variables

\*/

int count;

String obstacleName;

/\*\*

\* constructor

\* scale image

\*/

public Obstacle(String obstacleName)

{

this.obstacleName = obstacleName;

setImage(this.obstacleName);

getImage().scale(50,50); }

//method act for class bahaviour

//set location random

public void act()

{

count++;

if(count % (Greenfoot.getRandomNumber(30) +1) == 0 && this.obstacleName != "news - Copy.gif")

{

setLocation(getX() + 30, getY() );

if(getX() == getWorld().getWidth()-1)

setLocation(1, getY());

}

}

}

# Play.class

/\*\*

\* button redirect to world MyWorld

\*

\* @Marius Boncica

\* @3.19/01/2023

\*/

public class Play extends Actor

//create object of gif

{ GreenfootImage myGif = new GreenfootImage("Play1.png");

/\*\*

\* Constructor

\* scale image

\*/

public Play(){

GreenfootImage myGif = getImage();

int h = (int)myGif.getHeight()/1;

int w = (int)myGif.getWidth()/2;

myGif.scale(h,w);

}

//behaviour of image on click event

public void act()

{if(Greenfoot.mousePressed(this))

{getImage().scale((int)Math.round(getImage().getWidth()\*0.9),

(int)Math.round(getImage().getHeight()\*0.9));

}

// Add your action code here.

if(Greenfoot.mouseClicked(this))

{ Greenfoot.delay(5);

//redirect event on click to world MyWorld

Greenfoot.setWorld(new MyWorld());

}

}

}

# Playmenu.class

/\*\*

\* Button redirect to Galaxy3rdLevel

\*

\* @Marius Boncica

\* @version (3 19/01/2023)

\*/

public class PlayMenu extends Actor

{ GreenfootImage myGif = new GreenfootImage("Play1.png");

/\*\*

\* Constructor--Scale Image

\*/

public PlayMenu(){

GreenfootImage myGif = getImage();

int h = (int)myGif.getHeight()/1;

int w = (int)myGif.getWidth()/2;

myGif.scale(h,w);

}

//METHOD ACT DESCRIBE ANIMATION BEHAVIOUR OF IMAGE ON CLICK EVENT

public void act()

{if(Greenfoot.mousePressed(this))

{getImage().scale((int)Math.round(getImage().getWidth()\*0.9),

(int)Math.round(getImage().getHeight()\*0.9));

}

// Add your action code here.

if(Greenfoot.mouseClicked(this))

{ Greenfoot.delay(5);

//REDIRECT ON CLICK EVENT TO WORLD Galaxy3rdLevel

Greenfoot.setWorld(new Galaxy3rdLevel());

}

}

}

# PlaySingle.class

/\*\*

\* Write a description of class Play here.

\*

\* @author (Marius & Martin)

\* @version (a version number or a date)

\*/

public class PlaySingle extends Actor

{ GreenfootImage myGif = new GreenfootImage("Play1.png");

/\*\*

\* Act - do whatever the Play wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public PlaySingle(){

GreenfootImage myGif = getImage();

int h = (int)myGif.getHeight()/1;

int w = (int)myGif.getWidth()/2;

myGif.scale(h,w);

}

public void act()

{if(Greenfoot.mousePressed(this))

{getImage().scale((int)Math.round(getImage().getWidth()\*0.9),

(int)Math.round(getImage().getHeight()\*0.9));

}

// Add your action code here.

if(Greenfoot.mouseClicked(this))

{ Greenfoot.delay(5);

Greenfoot.setWorld(new Single());

}

}

}

# PlaySingleMenu.Class

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Write a description of class Play here.

\*

\* @author (Marius & Martin)

\* @version (a version number or a date)

\*/

public class PlaySingleMenu extends Actor

{ GreenfootImage myGif = new GreenfootImage("Play1.png");

/\*\*

\* Act - do whatever the Play wants to do. This method is called whenever

\* the 'Act' or 'Run' button gets pressed in the environment.

\*/

public PlaySingleMenu(){

GreenfootImage myGif = getImage();

int h = (int)myGif.getHeight()/1;

int w = (int)myGif.getWidth()/2;

myGif.scale(h,w);

}

public void act()

{if(Greenfoot.mousePressed(this))

{getImage().scale((int)Math.round(getImage().getWidth()\*0.9),

(int)Math.round(getImage().getHeight()\*0.9));

}

// Add your action code here.

if(Greenfoot.mouseClicked(this))

{ Greenfoot.delay(5);

Greenfoot.setWorld(new Galaxy2ndLevel());

}

}

}

# Player1.class

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Author: Marius Daniel Boncica- Group project

\* UPDATE 14/12/2022

\*/

public class Player1 extends Actor

{

//declare variables

public String player1Image;

public static int scor = 0, timerPrizeSound = 0, contorPrizeSound = 10, timerBoosterSound = 0, contorBoosterSound = 10;

static float speed = 5;

static boolean prizeSoundOn = false, boosterSoundOn = false;

static GreenfootSound prizeSound = new GreenfootSound("cherry\_sound(short\_buzzer\_sound).wav"), boosterSound = new GreenfootSound("burger\_sound(sci\_fi\_drill\_alert).wav");

public Player1(String player1Image){

this.player1Image = player1Image;

getImage().scale(80,80); }

/\*\*

\* Method act

\* call methods to control adpoints, speed booster, drop items, shoot projectile

\*/

public void act()

{

control();

addPoints();

speedBooster();

touchAsteroid();

moonLaunch();

fireProjectile();

hitByProjectile2();

}

//method to shoot projectile--add object on click

public void fireProjectile(){

if(Greenfoot.mousePressed(null)){

Projectile projectile = new Projectile();

getWorld().addObject(projectile,getX(),getY());

projectile.turnTowards(Greenfoot.getMouseInfo().getX(),Greenfoot.getMouseInfo().getY());

}

}

//method if touching Projectile2 class- decrease score

public void hitByProjectile2(){

Actor projectile2 = getOneIntersectingObject(Projectile2.class);

if(isTouching(Projectile2.class)){

getWorld().removeObject(projectile2);

MyWorld.player1Life.add(-1);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* movement \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private void control(){

if( Greenfoot.isKeyDown("down") )

setLocation( getX(), getY() + (int)speed );

if( Greenfoot.isKeyDown("up") )

setLocation( getX(), getY() - (int)speed );

if( Greenfoot.isKeyDown("left") )

setLocation( getX() - (int)speed, getY() );

if( Greenfoot.isKeyDown("right") )

setLocation( getX() + (int)speed, getY() );

}

//method to add points if touching prize class and sound control

private void addPoints(){

Actor points = getOneIntersectingObject(prize.class);

if( isTouching(prize.class) ){

scor++;

getWorld().removeObject(points);

MyWorld world = (MyWorld) getWorld();

if( !prizeSoundOn )

{

prizeSoundOn = true;

prizeSound.playLoop();

timerPrizeSound = 1;

}

else

timerPrizeSound = 1;

}

else if( prizeSoundOn )

{

timerPrizeSound = (timerPrizeSound+1) % contorPrizeSound;

if( timerPrizeSound == 0 )

{

prizeSound.stop();

prizeSoundOn = false;

}

}

}

//method to drop points and remove a Asteroid class when intersecting + spacebar

private void moonLaunch(){

Actor moon = getOneIntersectingObject(Asteroid.class);

if ( Greenfoot.isKeyDown("space") && Player1.scor >= 10 && Player1.scor <100){

getWorld().removeObject(moon);

getWorld().showText("ASTEROID DESTROYED!!!YOU SAVED THE WORLD!!!", 400, 400);

}

}

private void touchAsteroid(){

if(isTouching(Asteroid.class)){

setRotation(getRotation()+40);}

if( !prizeSoundOn )

{

prizeSoundOn = true;

prizeSound.playLoop();

timerPrizeSound = 1;

}

else

timerPrizeSound = 1;

}

//method to increase speed if touching speedBooster class+sounds

private void speedBooster(){

Actor booster = getOneIntersectingObject(speedBooster.class);

if( isTouching(speedBooster.class) ){

getWorld().removeObject(booster);

speed = 120 \* speed / 100;

MyWorld.player1Life.add(1);

if( !boosterSoundOn )

{

boosterSoundOn = true;

boosterSound.playLoop();

timerBoosterSound = 1;

}

else

timerBoosterSound = 1;

}

else if( boosterSoundOn )

{

timerBoosterSound = (timerBoosterSound+1) % contorBoosterSound;

if( timerBoosterSound == 0 )

{

boosterSound.stop();

boosterSoundOn = false;

}

}

}

}

# Player2.class

import greenfoot.\*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/\*\*

\* Author Marius Daniel Boncica

\* updated 14/12/2022

\*/

public class Player2 extends Actor

{

public String player2Image;

//declare variables

static int scor = 0, timerPrizeSound = 0, contorPrizeSound = 10, timerBoosterSound = 0, contorBoosterSound = 10;

static float speed = 5;

static boolean prizeSoundOn = false, boosterSoundOn = false;

static GreenfootSound prizeSound = new GreenfootSound("cherry\_sound(short\_buzzer\_sound).wav"), boosterSound = new GreenfootSound("burger\_sound(sci\_fi\_drill\_alert).wav");

/\*\*

\* Constructor

\* scale image

\*/

public Player2(String player2Image){

this.player2Image = player2Image;

setImage(this.player2Image);

getImage().scale(100,100); }

/\*\*

\* Act method- behaviour of class

\* control, earn points, speed, touching asteroid, remove class, shooting

\*/

public void act()

{

control();

earnPoints();

speedBooster();

touchAsteroid();

moonLaunch();

fireProjectile();

hitByProjectile1();

}

//method to shoot-create object of a class

public void fireProjectile(){

if(Greenfoot.isKeyDown("Q")){

Projectile2 projectile = new Projectile2();

getWorld().addObject(projectile,getX(),getY());

projectile.turnTowards(450,0);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* movement\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private void control(){

if( Greenfoot.isKeyDown("w") )

setLocation( getX(), getY() - (int)speed);

if( Greenfoot.isKeyDown("s") )

setLocation( getX(), getY() + (int)speed);

if( Greenfoot.isKeyDown("a") )

setLocation( getX() - (int)speed, getY() );

if( Greenfoot.isKeyDown("d") )

setLocation( getX() + (int)speed, getY() );

}

//method to remove Aasteroid class

private void moonLaunch(){

Actor moon = getOneIntersectingObject(Asteroid.class);

if ( Greenfoot.isKeyDown("space") && Player2.scor >= 10 && Player1.scor <100){

getWorld().removeObject(moon);

getWorld().showText("ASTEROID DESTROYED!!!YOU SAVED THE WORLD!!!", 400, 400);

}

}

//method to change movement if touching Asteroid class

public void touchAsteroid(){

if(isTouching(Asteroid.class)){

setRotation(getRotation()+40);}

if( !prizeSoundOn )

{

prizeSoundOn = true;

prizeSound.playLoop();

timerPrizeSound = 1;

}

else

timerPrizeSound = 1;

}

//method to add point when touching prize class + sound event

private void earnPoints(){

Actor price = getOneIntersectingObject(prize.class);

if( isTouching(prize.class) ){

scor++;

getWorld().removeObject(price);

MyWorld world = (MyWorld) getWorld();

if( !prizeSoundOn )

{

prizeSoundOn = true;

prizeSound.playLoop();

timerPrizeSound = 1;

}

else

timerPrizeSound = 1;

}

else if( prizeSoundOn ){

timerPrizeSound = (timerPrizeSound+1) % contorPrizeSound;

if( timerPrizeSound == 0 ){

prizeSound.stop();

prizeSoundOn = false;

}

}

}

//method to increase speed when touching speedBoostr class+sound

private void speedBooster(){

Actor booster = getOneIntersectingObject(speedBooster.class);

if( isTouching(speedBooster.class) ){

getWorld().removeObject(booster);

speed = 120 \* speed / 100;

MyWorld.player2Life.add(1);

if( !boosterSoundOn )

{

boosterSoundOn = true;

boosterSound.playLoop();

timerBoosterSound = 1;

}

else

timerBoosterSound = 1;

}

else if( boosterSoundOn )

{

timerBoosterSound = (timerBoosterSound+1) % contorBoosterSound;

if( timerBoosterSound == 0 )

{

boosterSound.stop();

boosterSoundOn = false;

}

}

}

//method if touching projectile-remove life

public void hitByProjectile1(){

Actor projectile1 = getOneIntersectingObject(Projectile.class);

if(isTouching(Projectile.class)){

getWorld().removeObject(projectile1);

MyWorld.player2Life.add(-1);

}

}

}

# PlayerSingle.class

/\*\*

\* Author: Marius Daniel Boncica- Group project

\* UPDATE 14/12/2022

\*/

public class PlayerSingle extends Actor

{//declare variables

public String playerSingleImage;

public static int scor = 0, timerPrizeSound = 0, contorPrizeSound = 10, timerBoosterSound = 0, contorBoosterSound = 10;

static float speed = 1;

static boolean prizeSoundOn = false, boosterSoundOn = false;

static GreenfootSound prizeSound = new GreenfootSound("cherry\_sound(short\_buzzer\_sound).wav"), boosterSound = new GreenfootSound("burger\_sound(sci\_fi\_drill\_alert).wav");

/\*\*

\* constructor

\* scale image

\*/

public PlayerSingle(String playerSingleImage){

this.playerSingleImage = playerSingleImage;

getImage().scale(80,80); }

/\*\*

\* act method: bahaviour of the class

\* control, points, speed, shooting, touch class events, release class

\*/

public void act()

{

control();

addPoints();

speedBooster();

moonLaunch();

fireProjectile();

touchAsteroid();

}

//method to shoot-create object on click event

public void fireProjectile(){

if(Greenfoot.mousePressed(null)){

Projectile projectile = new Projectile();

getWorld().addObject(projectile,getX(),getY());

projectile.turnTowards(Greenfoot.getMouseInfo().getX(),Greenfoot.getMouseInfo().getY());

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* movement \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private void control(){

if( Greenfoot.isKeyDown("down") ){

setLocation( getX(), getY() + (int)speed );

getImage().mirrorVertically();}

if( Greenfoot.isKeyDown("up") )

{ setLocation( getX(), getY() - (int)speed );

getImage().mirrorVertically();}

if( Greenfoot.isKeyDown("left") )

{setLocation( getX() - (int)speed, getY() );

getImage().mirrorHorizontally();}

if( Greenfoot.isKeyDown("right") )

{setLocation( getX() + (int)speed, getY() );

getImage().mirrorHorizontally();}

}

//method to add points if touching prize class + sound

private void addPoints(){

Actor points = getOneIntersectingObject(prize.class);

if( isTouching(prize.class)){

scor++;

Single world = (Single) getWorld();

getWorld().removeObject(points);

if( !prizeSoundOn )

{

prizeSoundOn = true;

prizeSound.playLoop();

timerPrizeSound = 1;

}

else

timerPrizeSound = 1;

}

else if( prizeSoundOn )

{

timerPrizeSound = (timerPrizeSound+1) % contorPrizeSound;

if( timerPrizeSound == 0 )

{

prizeSound.stop();

prizeSoundOn = false;

}

}

}

//method to remove Asteroid class + sound

private void moonLaunch(){

Actor moon = getOneIntersectingObject(Asteroid.class);

if ( Greenfoot.isKeyDown("space") && PlayerSingle.scor >= 10 && PlayerSingle.scor <100){

getWorld().removeObject(moon);

getWorld().showText("ASTEROID DESTROYED!!!YOU SAVED THE WORLD!!!", 400, 400);

}if( !prizeSoundOn )

{

prizeSoundOn = true;

prizeSound.playLoop();

timerPrizeSound = 1;

}

else

timerPrizeSound = 1;

}

//method to change movement if touching Asteroid class

public void touchAsteroid(){

if(isTouching(Asteroid.class)){

setRotation(getRotation()+40);}

}

//method to increase speed if touching speedBooster class+ sound

private void speedBooster(){

Actor booster = getOneIntersectingObject(speedBooster.class);

if( isTouching(speedBooster.class) ){

getWorld().removeObject(booster);

speed = 120 \* speed / 100;

Single.playerSingleLife.add(1);

if( !boosterSoundOn )

{

boosterSoundOn = true;

boosterSound.playLoop();

timerBoosterSound = 1;

}

else

timerBoosterSound = 1;

}

else if( boosterSoundOn )

{

timerBoosterSound = (timerBoosterSound+1) % contorBoosterSound;

if( timerBoosterSound == 0 )

{

boosterSound.stop();

boosterSoundOn = false;

}

}

}

}

# Projectile.class

/\*\*

\* Shooting projectile.

\*

\* @Marius Boncica

\* @version (3 19/01/2023)

\*/

public class Projectile extends Actor

{//declare speed value

int speed = 10;

/\*\*

\*constructor

\*\*scale image

\*/

public Projectile(){

getImage().scale(30,30);

}

/\*\*

\* Act method: behavior of class

\* remove if touching Player2 class

\* \*remove is at the edge of world or enemy

\*/

public void act()

{hitEnemy();

move(speed);

hitprize();

if(isTouching(Player2.class))

getWorld().removeObject(this);

else if(isAtEdge())

getWorld().removeObject(this);

// Add your action code here.

}

//remove projectil if touching Enemy class

public void hitEnemy(){

Actor enemy1 = getOneIntersectingObject(Enemy.class);

if( isTouching(Enemy.class))

getWorld().removeObject(enemy1);

}

//remove projectile if touching prize class

public void hitprize(){

Actor projectile = getOneIntersectingObject(prize.class);

if(isTouching(prize.class)){

getWorld().removeObject(projectile);

}

}

}

# Projectile2.class

/\*\*

\* shooting projectile

\*

\* @Marius Boncica

\* @ version 319/01/2023

\*/

public class Projectile2 extends Actor

{//declare variable speed and set value to 10

int speed = 10;

/\*\*

\*constructor

\* scale image

\*/

public Projectile2(){

getImage().scale(30,30);

}

/\*\*

\* Act Method

\* move,remove projectile when touching edge, player1 clas, prize class

\*/

public void act()

{turnProjectile();

move(speed);

hitprize();

if(isAtEdge()) move(0);

if(isTouching(Player1.class))

getWorld().removeObject(this);

}

public void turnProjectile(){

}

//method to remove projectile if touching prize class

public void hitprize(){

Actor projectile2 = getOneIntersectingObject(prize.class);

if(isTouching(prize.class)){

getWorld().removeObject(projectile2);

}

}

}

# ScrollActor.class

/\*\*

\* CLASS: QActor (subclass of Actor)<br>

\* AUTHOR: danpost (greenfoot.org username)<br>

\* CREATION DATE: September 5, 2014<br>

\* MODIFIED: October 6, 2014 - seperated rotation and movement vectors;

\* also, changed from using vector and speed fields to horizontal and vertical speed fields<br>

\* MODIFIED: October 8, 2014 - added two needed 'move' methods and corrected documentation<br>

\* <br>

\* NOTE: not intended for use in gridded worlds (worlds with a cell-size greater than one).<br>

\* <br>

\* DESCRIPTION: an extension of the Actor class that provides smooth movement and turning; also,

\* it provides options for the behavior of the actor, 'boundedAction', when reaching bounded limits

\* (the edges of the world plus or minus an offset value saved in the'boundedRange' field.<br>

\* <br>

\* The class works by dividing each unit (pixel or degree) into 1/100ths of that unit. When adding an

\* actor into the world, setting its location in the world or setting its rotation, normal values are

\* given. When moving or turning (adjusting using the current location or rotation as a reference), one

\* hundred times the actual changes are given. The values given do not have to be even multiples of one

\* hundred, which allows multiple in-between speeds of moving and turning to be available to your actors,

\* including very small rates of speed without the need for delay counters.

\*

\* The class contains horizontal and vertical speed fields, 'vX' and 'vY' that work in the same unitary

\* scale. There are methods to get and set these field values and a method, 'addForce' to adjust these

\* values by way of applying a force which is represented by a length (strength of force) and vector

\* (direction of force). Both parts of the force are given in '100x' for accuracy.

\*/

public abstract class ScrollActors extends Actor

{

protected static final int QVAL = 100; // smoothness value

/\*\* bounds action value -- indicates no movement beyond bounds \*/

protected static final int LIMIT = 0;

/\*\* bounds action value -- indicates actor is removed upon reaching bounds \*/

protected static final int REMOVE = 1;

/\*\* bounds action value -- indicates actor is transported to opposite world edge upon reaching bounds \*/

protected static final int WRAP = 2;

/\*\* bounds action value -- indicates actor turns toward center upon reaching bounds \*/

protected static final int BOUNCE = 3;

private int qX, qY; // fine-tuned location coordinates (100x)

private int vX, vY; // fine-tuned speeds along the horizontal axis and the vertial axis (100x)

private int qR; // fine-tuned rotation (100x)

private int boundedAction, boundedRange; // world edge fields

/\*\*

\* as the main method for movement, uses the current horizontal and vertical speed values to

\* relocate the actor, then checks the bounds of the actor to perform set actions; all 'move'

\* methods call this method and can be called by subclasses for continuous movement along the

\* same direction; the other methods that use it are 'move(int)', which will move the actor in the

\* direction it is facing and 'move(int, int)', which will move the actor in the direction given.

\*/

public void move()

{

qX += vX; // adjust x coordinate value of q-location

qY += vY; // adjust y coordinate value of q-locaation

super.setLocation(qX/QVAL, qY/QVAL); // set current location of actor

boundsAct(); // check for and perform bounded edge action

}

/\*\*

\* overrides the Actor class 'setLocation(int, int)' method to allow the q-location values to be corrected

\*

\* @param x the world x location to place the actor

\* @param y the world y location to place the actor

\*/

public void setLocation(int x, int y)

{

super.setLocation(x, y); // set location of actor

qX = getX()\*QVAL; // set x coordinate value of q-location

qY = getY()\*QVAL; // set y coordinate value of q-location

}

/\*\*

\* overrides the Actor class 'turn(int)' method to allow the q-rotation value to be adjusted

\*

\* @param amount the change in rotation as a fine-tuned value (100x)

\*/

public void turn(int amount)

{

qR = (qR+amount+360\*QVAL)%(360\*QVAL); // adjust q-rotation value

super.setRotation(qR/QVAL); // set current rotation

}

/\*\*

\* overrides the Actor class 'turnTowards(int, int)' method to allow the q-rotation value to be corrected

\*

\* @param x the world x coordinate of the point to face

\* @param y the world y coordinate of the point to face

\*/

public void turnTowards(int x, int y)

{

setQRotation((int)(QVAL\*(Math.atan2(y-getY(), x-getX())\*180/Math.PI)));

}

/\*\*

\* overrides the Actor class 'setRotation(int)' method to allow the q-rotation value to be corrected

\*

\* @param angle the angle in degrees at which the rotation of the actor is to be set

\*/

public void setRotation(int angle)

{

super.setRotation(angle); // set actor rotation

qR = getRotation()\*QVAL; // set q-rotation value

}

/\*\*

\* adds a force to the movement of the actor

\*

\* @param amount the strength of the force as a fine-tuned value (100x)

\* @param direction the direction of the force in fine-tuned degrees (100x)

\*/

public void addForce(int amount, int direction)

{

vX += Math.cos((double)direction\*Math.PI/(180\*QVAL))\*(double)amount; // new horizontal speed

vY += Math.sin((double)direction\*Math.PI/(180\*QVAL))\*(double)amount; // new vertical speed

}

/\*\*

\* applys a force to (moves) the actor

\*

\* @param amount the strength of the force as a fine-tuned value (100x)

\* @param direction the direction of the force in fine-tuned degrees (100x)

\*/

public void move(int amount, int direction)

{

int holdX = vX, holdY = vY; // save the current values

vX = 0; vY = 0; // clear the values

addForce(amount, direction);

move();

vX = holdX; vY = holdY; // restore saved values

}

/\*\*

\* overrides the Actor class 'move(int)' method to use the given fine-tuned distance with the

\* fine-tuned rotation of the actor for the direction

\*

\* @param amount the fine-tuned distanc value to move (100x)

\*/

public void move(int amount)

{

move(amount, qR);

}

/\*\*

\* sets the action and range fields for behavior when bounds are exceeded by actor

\*

\* @param action one of the bounds action values of this class

\* @param range offset from edge the action is to occur

\*/

public void setBoundedAction(int action, int range)

{

boundedAction = action%4; // the given action

boundedRange = range; // the given range offset where positive direction is away from center

}

/\*\*

\* internal method to perform the bounded action that is currently set to the actor

\*/

private void boundsAct()

{

switch(boundedAction)

{

case LIMIT: // no movement beyond bounds

if (qX <= -boundedRange\*QVAL)

{

setQLocation(-boundedRange\*QVAL, qY);

if (vX < 0) vX = 0;

}

if (qY <= -boundedRange\*QVAL)

{

setQLocation(qX, -boundedRange\*QVAL);

if (vY < 0) vY = 0;

}

if (qX >= (getWorld().getWidth()+boundedRange-1)\*QVAL)

{

setQLocation((getWorld().getWidth()+boundedRange-1)\*QVAL, qY);

if (vX > 0) vX = 0;

}

if (qY >= (getWorld().getHeight()+boundedRange-1)\*QVAL)

{

setQLocation(qX, (getWorld().getHeight()+boundedRange-1)\*QVAL);

if (vY > 0) vY = 0;

}

break;

case REMOVE: // actor is removed at bounds

if (getX() <= -boundedRange ||

getX() >= getWorld().getWidth()+boundedRange-1 ||

getY() <= -boundedRange ||

getY() >= getWorld().getHeight()+boundedRange-1)

getWorld().removeObject(this);

break;

case WRAP: // actor is transported to opposite world edge at bounds

if (getX() <= -boundedRange)

setLocation(getX()+getWorld().getWidth()+boundedRange\*2-2, getY());

else if (getY() <= -boundedRange)

setLocation(getX(), getY()+getWorld().getHeight()+boundedRange\*2-2);

else if (getX() >= getWorld().getWidth()+boundedRange-1)

setLocation(1-boundedRange, getY());

else if (getY() >= getWorld().getHeight()+boundedRange-1)

setLocation(getX(), 1-boundedRange);

break;

case BOUNCE: // actor faces toward center at bounds

if (getX() < -boundedRange || getX() >= getWorld().getWidth()+boundedRange) vX = -vX;

if (getY() < -boundedRange || getY() >= getWorld().getHeight()+boundedRange) vY = -vY;

break;

}

}

/\*\*

\* sets the location of the actor to fined-tuned coordinate values

\*

\* @param x the fine-tuned x-coordinate value (100x)

\* @param y the fine-tuned y-coordinate value (100x)

\*/

public void setQLocation(int x, int y)

{

qX = x;

qY = y;

super.setLocation(qX/QVAL, qY/QVAL);

}

/\*\*

\* sets the rotation of the actor to a fined-tuned value

\*

\* @param amount the fine-tuned rotational value for the actor in degrees (100x)

\*/

public void setQRotation(int amount)

{

qR = amount;

super.setRotation(qR/QVAL);

}

/\*\*

\* returns the fine-tuned x-coordinate for the locationof the actor (100x)

\*

\* @return the fine-tuned value of the x-coordinate for the location of the actor

\*/

protected int getQX() { return qX; }

/\*\*

\* returns the fine-tuned y-coordinate for the location of the actor (100x)

\*

\* @return the fine-tuned value of the y-coordinate for the location of the actor

\*/

protected int getQY() { return qY; }

/\*\*

\* returns the fine-tuned rotation of the actor (100x)

\*

\* @return the fine-tuned rotation of the actor

\*/

protected int getQR() { return qR; }

/\*\*

\* returns the fine-tuned speed along the horizontal (100x)

\*

\* @return the fine-tuned value of the speed of the actor along the horizontal

\*/

protected int getVX() { return vX; }

/\*\*

\* returns the fine-tuned speed along the vertical (100x)

\*

\* @return the fine-tuned value of the speed of the actor along the vertical

\*/

protected int getVY() { return vY; }

/\*\*

\* sets the horizontal speed to the given fine-tuned value (100x)

\*

\* @param speed the fine-tuned value the horizontal speed is to be set to

\*/

protected void setVX(int speed) { vX = speed; }

/\*\*

\* sets the vertical speed to the given fine-tuned value (100x)

\*

\* @param speed the fine-tuned value the horizontal speed is to be set to

\*/

protected void setVY(int speed) { vY = speed; }

}

Asteroid.class

/\*\*

\* Author: Marius Daniel Boncica- Group project

\* UPDATE 14/12/2022 Asteroid class to be removed

\*/

public class Asteroid extends ScrollActors

{//declare speed, angle rotation variables

public String NameImage;

private int rSpeed = 2\*QVAL-Greenfoot.getRandomNumber(4\*QVAL+1); // tumble rate

private int angle = 175\*QVAL+Greenfoot.getRandomNumber(10\*QVAL+1); // angular direction

private int speed = 0; // velocity

/\*\*

\* Constructor

\* remove when past 50 edge

\*/

public Asteroid()

{

setBoundedAction(REMOVE, 0); // set bounded action to remove when 50 past edge

}

//scale image

public Asteroid(String NameImage){

this.NameImage = NameImage;;

getImage().scale(400,300);

}

/\*\*

\* method act

\* move and turn

\*/

public void act()

{

// turning

turn(rSpeed);

// moving

move(speed, angle);

// removed

}

}

# Earth.class

/\*\*

\* Author: Marius Daniel Boncica- Group project

\* UPDATE 14/12/2022

\*/

public class Earth extends ScrollActors

{//declare variable for speed, angle rotation, score

public String NameImage;

public static Actor scoreDisp; // actor displaying number of avoided rocks (unhit)

public static int created, unhit; // class counts

private int rSpeed = 2\*QVAL-Greenfoot.getRandomNumber(4\*QVAL+1); // tumble rate

private int angle = 175\*QVAL+Greenfoot.getRandomNumber(10\*QVAL+1); // angular direction

private int speed = QVAL/2+Greenfoot.getRandomNumber(2\*QVAL)+created; // velocity

//constructor

public Earth()

{

created++; // bump counter

setBoundedAction(REMOVE, 0); // set bounded action to remove when 50 past edge

}

//scale image

public Earth(String NameImage){

this.NameImage = NameImage;;

getImage().scale(30,30);

}

//act method of class

public void act()

{

// turning

turn(rSpeed);

// moving

move(speed, angle);

}

}

# Enemy.class

import greenfoot.\*;

//Author : Marius Boncica and Martin Konecny

//Updated jan 2023

public class Enemy extends ScrollActors

{//declare variables for speed, rotation angle, created

public String NameImage;

public static Actor scoreDisp;

public static int created; // class counts

private int rSpeed = 2\*QVAL-Greenfoot.getRandomNumber(4\*QVAL+1); // tumble rate

private int angle = 175\*QVAL+Greenfoot.getRandomNumber(10\*QVAL+1); // angular direction

private int speed = QVAL/2+Greenfoot.getRandomNumber(2\*QVAL)+created; // velocity

//constructor

public Enemy()

{

created++; // bump counter

setBoundedAction(REMOVE, 50); // set bounded action to remove when 50 past edge

}

//scale image

public Enemy(String NameImage){

this.NameImage = NameImage;;

getImage().scale(150,150);

}

/\*\*

\* act method:

\* touch Plane class and turn

\*/

public void act()

{

touchPlane();// turning

turn(rSpeed);

// moving

move(speed, angle);

// removed when touch projectile

touchProjectile();

}

//method to remove this object when touching projectile class

public void touchProjectile(){

Actor bullet = getOneIntersectingObject(Projectile.class);

if( isTouching(Projectile.class ) ){

getWorld().removeObject(bullet);

getWorld().removeObject(this);

}

}

//method to create rotation when touching Plane class

public void touchPlane(){

int dir = 1-2\*((getRotation()+90)/180);

if( isTouching(Plane.class ) )

setRotation(135+90\*(1-dir)+Greenfoot.getRandomNumber(91));

}

}

# Mars.class

/\*\*

\* Author: Marius Daniel Boncica & Martin Konecny

\* UPDATE 14/12/2022

\*/

public class Mars extends ScrollActors

{//declare variable score, speed, rotation angle

public String NameImage;

public static Actor scoreDisp; // actor displaying number of avoided rocks (unhit)

public static int created, unhit; // class counts

private int rSpeed = 2\*QVAL-Greenfoot.getRandomNumber(4\*QVAL+1); // tumble rate

private int angle = 175\*QVAL+Greenfoot.getRandomNumber(10\*QVAL+1); // angular direction

private int speed = QVAL/2+Greenfoot.getRandomNumber(2\*QVAL)+created; // velocity

//constructor

public Mars()

{

created++; // bump counter

setBoundedAction(REMOVE, 0); // set bounded action to remove when 50 past edge

}

//scale image

public Mars(String NameImage){

this.NameImage = NameImage;;

getImage().scale(30,30);

}

//act method tirn rotate and move

public void act()

{

// turning

turn(rSpeed);

// moving

move(speed, angle);

}

}

# Plane.class

/\*\*

\* Author: Marius Daniel Boncica & Martin Konecny

\* UPDATE 19/01/2023

\*/

public class Plane extends ScrollActors

{//declare variables score, prizecount booster speed

//sound event

public String PlaneImage;

public static int score = 0, planeLife = 0, contorPrizeSound = 10, timerBoosterSound = 0, contorBoosterSound = 10;

static int speed = 1;

static boolean prizeSoundOn = false, boosterSoundOn = false;

static GreenfootSound prizeSound = new GreenfootSound("cherry\_sound(short\_buzzer\_sound).wav"), boosterSound = new GreenfootSound("burger\_sound(sci\_fi\_drill\_alert).wav");

// constructor: scale image

public Plane(String PlaneImage){

this.PlaneImage = PlaneImage;

getImage().scale(150,150);}

/\*\*

\* Act Method

\* Control behaviour using arrows

\* add points, shooting

\*/

public void act()

{

// adjusting the rotation

int dr = 0;

if (Greenfoot.isKeyDown("right")) dr++;

if (Greenfoot.isKeyDown("left")) dr--;

turn(dr\*QVAL);

// adjusting the speed

int ds = 0;

if (Greenfoot.isKeyDown("up")) ds++;

if (Greenfoot.isKeyDown("down")) ds--;

speed += ds;

// limiting the speed

if (speed < 0) speed = 0;

if (speed > 5\*QVAL) speed = 5\*QVAL;

// moving

move(speed);

removePoints();

fireProjectile();

addPoints();

removeSun();

}

//method to shoot- create object of projectile class

public void fireProjectile(){

if(Greenfoot.mousePressed(null)){

Projectile projectile = new Projectile();

getWorld().addObject(projectile,getX(),getY());

projectile.turnTowards(Greenfoot.getMouseInfo().getX(),Greenfoot.getMouseInfo().getY());

}

}

//method to add points if touching Earth and Mars class

private void addPoints(){

Actor points = getOneIntersectingObject(Earth.class);

if( isTouching(Earth.class) ){

score++;

}Actor points2 = getOneIntersectingObject(Mars.class);

if( isTouching(Mars.class) ){

getWorld().removeObject(points2);

score++;

}

}

//method to remove points if touching Sun class or Enemy Class

private void removePoints(){

Actor points = getOneIntersectingObject(Sun.class);

if( isTouching(Sun.class ) ){

getWorld().showText("Avoid Sun! You Will Melt!!!Press SPACEBAR",100,400);

score--;

setRotation(getRotation()+40);//set rotation when touching Sun class

//score will decrease as long as plane will rotate near Sun class

}

Actor points2 = getOneIntersectingObject(Enemy.class);

if( isTouching(Enemy.class ) ){

score--;//decrease score

//score will decrease as long as Enemy rotates around Plane

}

}

// remove sun if score > 10

private void removeSun(){

Actor sun1 = getOneIntersectingObject(Sun.class);

if ( Greenfoot.isKeyDown("space") ){

getWorld().removeObject(sun1);

getWorld().showText("Sun Removed!!!", 400, 400);

}

}

}

# Sun.class

/\*\*

\* Author: Marius Daniel Boncica & Martin Konecny Group project

\* UPDATE 14/12/2022

\*/

public class Sun extends ScrollActors

{//declare variables

public String NameImage;

public static Actor scoreDisp; // actor displaying number of avoided rocks (unhit)

public static int created, unhit; // class counts

private int rSpeed = 2\*QVAL-Greenfoot.getRandomNumber(4\*QVAL+1); // tumble rate

private int angle = 175\*QVAL+Greenfoot.getRandomNumber(10\*QVAL+1); // angular direction

private int speed = QVAL/2+Greenfoot.getRandomNumber(2\*QVAL)+created; // velocity

/\*\*

\* CONSTRUCTOR

\*COUNT CREATED AND REMOVE PAST 50 AT THE EDGE OF WORLD

\*/

public Sun()

{

created++; // bump counter

setBoundedAction(REMOVE, 0); // set bounded action to remove when 50 past edge

}

//SCALE IMAGE

public Sun(String NameImage){

this.NameImage = NameImage;;

getImage().scale(30,30);

}

//ACT METHOD-- TURN ROTATE AND SPEED

public void act()

{

// turning

turn(rSpeed);

// moving

move(speed, angle);

}

}

# Step1.class

/\*\*

\*Step1-- center background of the world modified for 3d effect

\*

\* @author Marius Boncica

\* @version (1.01 8/12/2022)

\*/

//constructor gif image

public class Step1 extends Actor

{

private String img;

private GifImage gif;

GifImage myGif = new GifImage("sunearth.gif");

public Step1(String img)

{

this.img = img;

gif = new GifImage( img );

}

//act method: set image current

public void act()

{

setImage( myGif.getCurrentImage() );

// Add sayour action code here.

}

}

# Prize.class

/\*\*

\* Author Marius Daniel Boncica

\* updated 14/12/2022

\*/

public class prize extends Actor

//declare variables: image and speed

{GifImage myGif = new GifImage("satelit.gif");

int speed = 3;

//act method: behaviour: at edge turn, move remove if hit projectile class

public void act()

{

move(speed);

if(isAtEdge()) turn(20);

hitByProjectile1();

hitByProjectile2();

}

public prize(){}

//method to remove object if touching Projectile2 class increase life score

public void hitByProjectile2(){

Actor projectile2 = getOneIntersectingObject(Projectile2.class);

if(isTouching(Projectile2.class)){

getWorld().removeObject(projectile2);

MyWorld.player2Life.add(1);

}

}

//method to remove object if touching Projectile class increase life score

public void hitByProjectile1(){

Actor projectile = getOneIntersectingObject(Projectile.class);

if(isTouching(Projectile.class)){

getWorld().removeObject(projectile);

MyWorld.player1Life.add(1);

Single.playerSingleLife.add(1);

}

}

}

# Speedbooster.class

/\*\*

\* Author Marius Daniel Boncica

\* updated 14/12/2022

\*/

public class speedBooster extends Actor

{

/\*\*

\* behaviour of this class defined in MyWorld \*/

public void act()

{

// Add your action code here.

}

}

# Gifimage.class

import java.io.BufferedInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.net.URL;

import java.util.List;

import java.util.ArrayList;

/\*\*

\* This class can be used to read animated gif image files and extract the individual

\* images of the animation sequence.

\*

\* @author Michael Berry

\* @author Neil Brown

\*

\* Copyright (c) 2011,2013,2014,2018,2021

\*/

public class GifImage

{

/\*\* The images used in the animation. \*/

private GreenfootImage[] images;

/\*\* The delay between each frame. \*/

private int[] delay;

/\*\* The index of the current frame in the GIF file. \*/

private int currentIndex;

/\*\* The time passed since the last frame in ms. \*/

private long time;

/\*\* Whether the animation is paused or not. \*/

private boolean pause;

/\*\*

\* Set the image of the actor. If the image is a normal picture, it will be displayed as normal.

\* If it's an animated GIF file then it will be displayed as an animated actor.

\*/

public GifImage(String file)

{

pause = false;

if(file.toLowerCase().endsWith(".gif")) {

loadImages(file);

}

else {

images = new GreenfootImage[] {new GreenfootImage(file)};

delay = new int[] {1000}; // Doesn't matter, as long as it's not zero

currentIndex = 0;

time = System.currentTimeMillis();

}

}

/\*\*

\* Copy the given GifImage. This is faster, and uses less memory, than loading the same

\* GIF multiple times. The current play state (position in the GIF, paused state) is copied

\* from the given GifImage, but after that they can be independently played/paused.

\*

\* The images making up the GIF are shared between the two images, so any modifications to

\* the images will be shared in both GIFs. You can call this constructor on the same source

\* GIF multiple times.

\* @param copyFrom The GifImage to copy from.

\*/

public GifImage(GifImage copyFrom)

{

pause = copyFrom.pause;

images = copyFrom.images.clone();

delay = copyFrom.delay.clone();

currentIndex = copyFrom.currentIndex;

time = copyFrom.time;

}

/\*\*

\* Get all the images used in the animation

\* @return a list of GreenfootImages, corresponding to each frame.

\*/

public List<GreenfootImage> getImages()

{

ArrayList<GreenfootImage> images = new ArrayList<GreenfootImage>(this.images.length);

for(GreenfootImage image : this.images) {

images.add(image);

}

return images;

}

/\*\*

\* Pause the animation.

\*/

public void pause()

{

pause = true;

}

/\*\*

\* Resume the animation.

\*/

public void resume()

{

pause = false;

time = System.currentTimeMillis();

}

/\*\*

\* Determines whether the animation is running

\* @return true if the animation is running, false otherwise

\*/

public boolean isRunning()

{

return !pause;

}

public GreenfootImage getCurrentImage()

{

long delta = System.currentTimeMillis() - time;

while (delta >= delay[currentIndex] && !pause) {

delta -= delay[currentIndex];

time += delay[currentIndex];

currentIndex = (currentIndex+1) % images.length;

}

return images[currentIndex];

}

/\*\*

\* Load the images

\*/

private void loadImages(String file)

{

GifDecoder decode = new GifDecoder();

decode.read(file);

int numFrames = decode.getFrameCount();

if(numFrames>0) {

images = new GreenfootImage[numFrames];

delay = new int[numFrames];

}

else {

images = new GreenfootImage[1];

images[0] = new GreenfootImage(1, 1);

}

for (int i=0 ; i<numFrames ; i++) {

GreenfootImage image = new GreenfootImage(decode.getFrame(i).getWidth(), decode.getFrame(i).getHeight());

image.drawImage(decode.getFrame(i), 0, 0);

delay[i] = decode.getDelay(i);

images[i] = image;

}

time = System.currentTimeMillis();

}

/\*\*

\* The Rectangle class represents rectangles. This is essentially a re-implementation

\* of the java.awt.Rectangle class, created in order to avoid any dependency on AWT.

\*/

private static class Rectangle

{

public int x;

public int y;

public int width;

public int height;

public Rectangle(int x, int y, int width, int height)

{

this.x = x;

this.y = y;

this.width = width;

this.height = height;

}

}

/\*\*

\* Class GifDecoder - Decodes a GIF file into one or more frames. <br><br>

\*

\* <i>I (Michael) edited this slightly on 10/09/08 to bring class up to date with generics and therefore remove warnings.

\* Also edited so that resources are grabbed from the jar file and not externally, so no security exceptions.</i>

\* <br><br>

\* <pre>

\* Example:

\* GifDecoder d = new GifDecoder();

\* d.read(&quot;sample.gif&quot;);

\* int n = d.getFrameCount();

\* for (int i = 0; i &lt; n; i++) {

\* BufferedImage frame = d.getFrame(i); // frame i

\* int t = d.getDelay(i); // display duration of frame in milliseconds

\* // do something with frame

\* }

\* </pre>

\*

\* No copyright asserted on the source code of this class. May be used for any

\* purpose, however, refer to the Unisys LZW patent for any additional

\* restrictions. Please forward any corrections to kweiner@fmsware.com.

\*

\* @author Kevin Weiner, FM Software; LZW decoder adapted from John Cristy's

\* ImageMagick.

\* @version 1.03 November 2003

\*

\*/

private class GifDecoder

{

/\*\*

\* File read status: No errors.

\*/

public static final int STATUS\_OK = 0;

/\*\*

\* File read status: Error decoding file (may be partially decoded)

\*/

public static final int STATUS\_FORMAT\_ERROR = 1;

/\*\*

\* File read status: Unable to open source.

\*/

public static final int STATUS\_OPEN\_ERROR = 2;

private BufferedInputStream in;

private int status;

private int width; // full image width

private int height; // full image height

private boolean gctFlag; // global color table used

private int gctSize; // size of global color table

private int loopCount = 1; // iterations; 0 = repeat forever

private int[] gct; // global color table

private int[] lct; // local color table

private int[] act; // active color table

private int bgIndex; // background color index

private Color bgColor; // background color

private Color lastBgColor; // previous bg color

private int pixelAspect; // pixel aspect ratio

private boolean lctFlag; // local color table flag

private boolean interlace; // interlace flag

private int lctSize; // local color table size

private int ix, iy, iw, ih; // current image rectangle

private Rectangle lastRect; // last image rect

private GreenfootImage image; // current frame

private GreenfootImage lastImage; // previous frame

private byte[] block = new byte[256]; // current data block

private int blockSize = 0; // block size

// last graphic control extension info

private int dispose = 0;

// 0=no action; 1=leave in place; 2=restore to bg; 3=restore to prev

private int lastDispose = 0;

private boolean transparency = false; // use transparent color

private int delay = 0; // delay in milliseconds

private int transIndex; // transparent color index

private static final int MaxStackSize = 4096;

// max decoder pixel stack size

// LZW decoder working arrays

private short[] prefix;

private byte[] suffix;

private byte[] pixelStack;

private byte[] pixels;

private ArrayList<GifFrame> frames; // frames read from current file

private int frameCount;

/\*\*

\* A single frame

\*/

private class GifFrame {

public GifFrame(GreenfootImage im, int del) {

image = im;

delay = del;

}

private GreenfootImage image;

private int delay;

}

/\*\*

\* Convert an RGB integer value to a Color.

\*/

private Color colorFromInt(int rgb)

{

int r = (rgb & 0xFF0000) >> 16;

int g = (rgb & 0xFF00) >> 8;

int b = (rgb & 0xFF);

return new Color(r,g,b);

}

/\*\*

\* Gets display duration for specified frame.

\*

\* @param n

\* int index of frame

\* @return delay in milliseconds

\*/

public int getDelay(int n) {

//

delay = -1;

if ((n >= 0) && (n < frameCount)) {

delay = (frames.get(n)).delay;

}

return delay;

}

/\*\*

\* Gets the number of frames read from file.

\*

\* @return frame count

\*/

public int getFrameCount() {

return frameCount;

}

/\*\*

\* Gets the first (or only) image read.

\*

\* @return BufferedImage containing first frame, or null if none.

\*/

public GreenfootImage getImage() {

return getFrame(0);

}

/\*\*

\* Gets the "Netscape" iteration count, if any. A count of 0 means repeat

\* indefinitiely.

\*

\* @return iteration count if one was specified, else 1.

\*/

public int getLoopCount() {

return loopCount;

}

/\*\*

\* Creates new frame image from current data (and previous frames as specified

\* by their disposition codes).

\*/

protected void setPixels() {

// fill in starting image contents based on last image's dispose code

if (lastDispose > 0) {

if (lastDispose == 3) {

// use image before last

int n = frameCount - 2;

if (n > 0) {

lastImage = getFrame(n - 1);

} else {

lastImage = null;

}

}

if (lastImage != null) {

image.clear();

image.drawImage(lastImage, 0, 0);

// copy pixels

if (lastDispose == 2) {

// fill last image rect area with background color

Color c = null;

if (transparency) {

c = new Color(0, 0, 0, 0); // assume background is transparent

} else {

c = lastBgColor; // use given background color

}

for (int x = 0; x < lastRect.width; x++)

{

for (int y = 0; y < lastRect.height; y++)

{

image.setColorAt(lastRect.x + x, lastRect.y + y, c);

}

}

}

}

}

// copy each source line to the appropriate place in the destination

int pass = 1;

int inc = 8;

int iline = 0;

for (int i = 0; i < ih; i++) {

int line = i;

if (interlace) {

if (iline >= ih) {

pass++;

switch (pass) {

case 2:

iline = 4;

break;

case 3:

iline = 2;

inc = 4;

break;

case 4:

iline = 1;

inc = 2;

}

}

line = iline;

iline += inc;

}

line += iy;

if (line < height) {

int k = line \* width;

int dlim = Math.min(ix + iw, width);

int sx = i \* iw;

for (int dx = ix; dx < dlim; dx++) {

int index = ((int) pixels[sx++]) & 0xff;

int c = act[index];

if (c != 0) {

image.setColorAt(dx, line, colorFromInt(c));

}

}

}

}

}

/\*\*

\* Gets the image contents of frame n.

\*

\* @return BufferedImage representation of frame, or null if n is invalid.

\*/

public GreenfootImage getFrame(int n) {

GreenfootImage im = null;

if ((n >= 0) && (n < frameCount)) {

im = ((GifFrame) frames.get(n)).image;

}

return im;

}

/\*\*

\* Gets image size.

\*

\* @return GIF image dimensions as an array - [0] = width, [1] = height

\*/

public int[] getFrameSize() {

return new int[]{width, height};

}

/\*\*

\* Reads GIF image from stream

\*

\* @param BufferedInputStream

\* containing GIF file.

\* @return read status code (0 = no errors)

\*/

public int read(BufferedInputStream is) {

init();

if (is != null) {

in = is;

readHeader();

if (!err()) {

readContents();

if (frameCount < 0) {

status = STATUS\_FORMAT\_ERROR;

}

}

} else {

status = STATUS\_OPEN\_ERROR;

}

try {

is.close();

} catch (IOException e) {

}

return status;

}

/\*\*

\* Reads GIF image from stream

\*

\* @param InputStream

\* containing GIF file.

\* @return read status code (0 = no errors)

\*/

public int read(InputStream is) {

init();

if (is != null) {

if (!(is instanceof BufferedInputStream))

is = new BufferedInputStream(is);

in = (BufferedInputStream) is;

readHeader();

if (!err()) {

readContents();

if (frameCount < 0) {

status = STATUS\_FORMAT\_ERROR;

}

}

} else {

status = STATUS\_OPEN\_ERROR;

}

try {

is.close();

} catch (IOException e) {

}

return status;

}

/\*\*

\* Reads GIF file from specified file/URL source (URL assumed if name contains

\* ":/" or "file:")

\*

\* @param name

\* String containing source

\* @return read status code (0 = no errors)

\*/

public int read(String name) {

status = STATUS\_OK;

InputStream resource = this.getClass().getResourceAsStream(name);

if (resource == null) {

name = "images/" + name;

resource = this.getClass().getResourceAsStream(name);

if (resource == null) {

throw new RuntimeException("The gif file \"" + name + "\" doesn't exist.");

}

}

in = new BufferedInputStream(resource);

status = read(in);

return status;

}

/\*\*

\* Decodes LZW image data into pixel array. Adapted from John Cristy's

\* ImageMagick.

\*/

protected void decodeImageData() {

int NullCode = -1;

int npix = iw \* ih;

int available, clear, code\_mask, code\_size, end\_of\_information, in\_code, old\_code, bits, code, count, i, datum, data\_size, first, top, bi, pi;

if ((pixels == null) || (pixels.length < npix)) {

pixels = new byte[npix]; // allocate new pixel array

}

if (prefix == null)

prefix = new short[MaxStackSize];

if (suffix == null)

suffix = new byte[MaxStackSize];

if (pixelStack == null)

pixelStack = new byte[MaxStackSize + 1];

// Initialize GIF data stream decoder.

data\_size = read();

clear = 1 << data\_size;

end\_of\_information = clear + 1;

available = clear + 2;

old\_code = NullCode;

code\_size = data\_size + 1;

code\_mask = (1 << code\_size) - 1;

for (code = 0; code < clear; code++) {

prefix[code] = 0;

suffix[code] = (byte) code;

}

// Decode GIF pixel stream.

datum = bits = count = first = top = pi = bi = 0;

for (i = 0; i < npix;) {

if (top == 0) {

if (bits < code\_size) {

// Load bytes until there are enough bits for a code.

if (count == 0) {

// Read a new data block.

count = readBlock();

if (count <= 0)

break;

bi = 0;

}

datum += (((int) block[bi]) & 0xff) << bits;

bits += 8;

bi++;

count--;

continue;

}

// Get the next code.

code = datum & code\_mask;

datum >>= code\_size;

bits -= code\_size;

// Interpret the code

if ((code > available) || (code == end\_of\_information))

break;

if (code == clear) {

// Reset decoder.

code\_size = data\_size + 1;

code\_mask = (1 << code\_size) - 1;

available = clear + 2;

old\_code = NullCode;

continue;

}

if (old\_code == NullCode) {

pixelStack[top++] = suffix[code];

old\_code = code;

first = code;

continue;

}

in\_code = code;

if (code == available) {

pixelStack[top++] = (byte) first;

code = old\_code;

}

while (code > clear) {

pixelStack[top++] = suffix[code];

code = prefix[code];

}

first = ((int) suffix[code]) & 0xff;

// Add a new string to the string table,

if (available >= MaxStackSize)

break;

pixelStack[top++] = (byte) first;

prefix[available] = (short) old\_code;

suffix[available] = (byte) first;

available++;

if (((available & code\_mask) == 0) && (available < MaxStackSize)) {

code\_size++;

code\_mask += available;

}

old\_code = in\_code;

}

// Pop a pixel off the pixel stack.

top--;

pixels[pi++] = pixelStack[top];

i++;

}

for (i = pi; i < npix; i++) {

pixels[i] = 0; // clear missing pixels

}

}

/\*\*

\* Returns true if an error was encountered during reading/decoding

\*/

protected boolean err() {

return status != STATUS\_OK;

}

/\*\*

\* Initializes or re-initializes reader

\*/

protected void init() {

status = STATUS\_OK;

frameCount = 0;

frames = new ArrayList<GifFrame>();

gct = null;

lct = null;

}

/\*\*

\* Reads a single byte from the input stream.

\*/

protected int read() {

int curByte = 0;

try {

curByte = in.read();

} catch (IOException e) {

status = STATUS\_FORMAT\_ERROR;

}

return curByte;

}

/\*\*

\* Reads next variable length block from input.

\*

\* @return number of bytes stored in "buffer"

\*/

protected int readBlock() {

blockSize = read();

int n = 0;

if (blockSize > 0) {

try {

int count = 0;

while (n < blockSize) {

count = in.read(block, n, blockSize - n);

if (count == -1)

break;

n += count;

}

} catch (IOException e) {

}

if (n < blockSize) {

status = STATUS\_FORMAT\_ERROR;

}

}

return n;

}

/\*\*

\* Reads color table as 256 RGB integer values

\*

\* @param ncolors

\* int number of colors to read

\* @return int array containing 256 colors (packed ARGB with full alpha)

\*/

protected int[] readColorTable(int ncolors) {

int nbytes = 3 \* ncolors;

int[] tab = null;

byte[] c = new byte[nbytes];

int n = 0;

try {

n = in.read(c);

} catch (IOException e) {

}

if (n < nbytes) {

status = STATUS\_FORMAT\_ERROR;

} else {

tab = new int[256]; // max size to avoid bounds checks

int i = 0;

int j = 0;

while (i < ncolors) {

int r = ((int) c[j++]) & 0xff;

int g = ((int) c[j++]) & 0xff;

int b = ((int) c[j++]) & 0xff;

tab[i++] = 0xff000000 | (r << 16) | (g << 8) | b;

}

}

return tab;

}

/\*\*

\* Main file parser. Reads GIF content blocks.

\*/

protected void readContents() {

// read GIF file content blocks

boolean done = false;

while (!(done || err())) {

int code = read();

switch (code) {

case 0x2C: // image separator

readImage();

break;

case 0x21: // extension

code = read();

switch (code) {

case 0xf9: // graphics control extension

readGraphicControlExt();

break;

case 0xff: // application extension

readBlock();

String app = "";

for (int i = 0; i < 11; i++) {

app += (char) block[i];

}

if (app.equals("NETSCAPE2.0")) {

readNetscapeExt();

} else

skip(); // don't care

break;

default: // uninteresting extension

skip();

}

break;

case 0x3b: // terminator

done = true;

break;

case 0x00: // bad byte, but keep going and see what happens

break;

default:

status = STATUS\_FORMAT\_ERROR;

}

}

}

/\*\*

\* Reads Graphics Control Extension values

\*/

protected void readGraphicControlExt() {

read(); // block size

int packed = read(); // packed fields

dispose = (packed & 0x1c) >> 2; // disposal method

if (dispose == 0) {

dispose = 1; // elect to keep old image if discretionary

}

transparency = (packed & 1) != 0;

delay = readShort() \* 10; // delay in milliseconds

transIndex = read(); // transparent color index

read(); // block terminator

}

/\*\*

\* Reads GIF file header information.

\*/

protected void readHeader() {

String id = "";

for (int i = 0; i < 6; i++) {

id += (char) read();

}

if (!id.startsWith("GIF")) {

status = STATUS\_FORMAT\_ERROR;

return;

}

readLSD();

if (gctFlag && !err()) {

gct = readColorTable(gctSize);

bgColor = colorFromInt(gct[bgIndex]);

}

}

/\*\*

\* Reads next frame image

\*/

protected void readImage() {

ix = readShort(); // (sub)image position & size

iy = readShort();

iw = readShort();

ih = readShort();

int packed = read();

lctFlag = (packed & 0x80) != 0; // 1 - local color table flag

interlace = (packed & 0x40) != 0; // 2 - interlace flag

// 3 - sort flag

// 4-5 - reserved

lctSize = 2 << (packed & 7); // 6-8 - local color table size

if (lctFlag) {

lct = readColorTable(lctSize); // read table

act = lct; // make local table active

} else {

act = gct; // make global table active

if (bgIndex == transIndex)

bgColor = colorFromInt(0);

}

int save = 0;

if (transparency) {

save = act[transIndex];

act[transIndex] = 0; // set transparent color if specified

}

if (act == null) {

status = STATUS\_FORMAT\_ERROR; // no color table defined

}

if (err())

return;

decodeImageData(); // decode pixel data

skip();

if (err())

return;

frameCount++;

// create new image to receive frame data

image = new GreenfootImage(width, height);

setPixels(); // transfer pixel data to image

frames.add(new GifFrame(image, delay)); // add image to frame list

if (transparency) {

act[transIndex] = save;

}

resetFrame();

}

/\*\*

\* Reads Logical Screen Descriptor

\*/

protected void readLSD() {

// logical screen size

width = readShort();

height = readShort();

// packed fields

int packed = read();

gctFlag = (packed & 0x80) != 0; // 1 : global color table flag

// 2-4 : color resolution

// 5 : gct sort flag

gctSize = 2 << (packed & 7); // 6-8 : gct size

bgIndex = read(); // background color index

pixelAspect = read(); // pixel aspect ratio

}

/\*\*

\* Reads Netscape extenstion to obtain iteration count

\*/

protected void readNetscapeExt() {

do {

readBlock();

if (block[0] == 1) {

// loop count sub-block

int b1 = ((int) block[1]) & 0xff;

int b2 = ((int) block[2]) & 0xff;

loopCount = (b2 << 8) | b1;

}

} while ((blockSize > 0) && !err());

}

/\*\*

\* Reads next 16-bit value, LSB first

\*/

protected int readShort() {

// read 16-bit value, LSB first

return read() | (read() << 8);

}

/\*\*

\* Resets frame state for reading next image.

\*/

protected void resetFrame() {

lastDispose = dispose;

lastRect = new Rectangle(ix, iy, iw, ih);

lastImage = image;

lastBgColor = bgColor;

int dispose = 0;

boolean transparency = false;

int delay = 0;

lct = null;

}

/\*\*

\* Skips variable length blocks up to and including next zero length block.

\*/

protected void skip() {

do {

readBlock();

} while ((blockSize > 0) && !err());

}

}

}

# Simpletimer.class

\* A simple timer class that allows you to keep track of how much time

\* has passed between events.

\*

\* You use this class by creating a timer as a member field in your actor (or whatever):

\* <pre>

\*

\* private SimpleTimer timer = new SimpleTimer();

\* </pre>

\*

\* Then when you want to start the timer (for example, when a shot is fired), you call the mark() method:

\*

\* <pre>

\*

\* timer.mark();

\* </pre>

\*

\* Thereafter, you can use the millisElapsed() method to find out how long it's been since mark()

\* was called (in milliseconds, i.e. thousandths of a second). So if you want to only allow the player to fire a shot every second, you

\* could write:

\*

\* <pre>

\*

\* if (timer.millisElapsed() > 1000 && Greenfoot.isKeyDown("space"))

\* {

\* // Code here for firing a new shot

\* timer.mark(); // Reset the timer

\* }

\* </pre>

\*

\* @author Neil Brown

\* @version 1.0

\*/

public class SimpleTimer

{

private long lastMark = System.currentTimeMillis();

/\*\*

\* Marks the current time. You can then in future call

\* millisElapsed() to find out the elapsed milliseconds

\* since this mark() call was made.

\*

\* A second mark() call will reset the mark, and millisElapsed()

\* will start increasing from zero again.

\*/

public void mark()

{

lastMark = System.currentTimeMillis();

}

/\*\*

\* Returns the amount of milliseconds that have elapsed since mark()

\* was last called. This timer runs irrespective of Greenfoot's

\* act() cycle, so if you call it many times during the same Greenfoot frame,

\* you may well get different answers.

\*/

public int millisElapsed()

{

return (int) (System.currentTimeMillis() - lastMark);

}

}