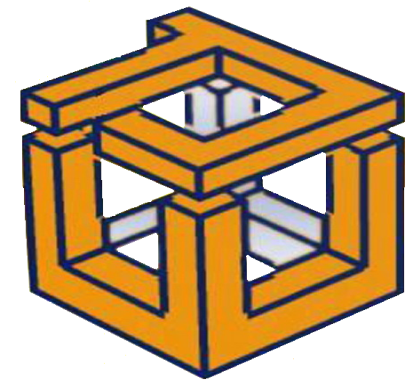


10.06.11



נושא הפרויקט: משחק "אורב-גראב"

שם תלמיד: רוסובסקי

יחידות לימוד: 5

ת.ז.: 311790273

שם המנחה: גנאדי רשאפ

תוכן עניינים

**מבוא........................................................................................................3**

מדריך למשתמש........................................................................................4

מדריך למפתח...........................................................................................7

תאוריה.....................................................................................................9

סיכום אישי...............................................................................................11

**...............................................................................................12** JavaDoc

קודים......................................................................................................48

מקורות....................................................................................................89

מבוא

" **אורב-גראב**" משחק פשוט. ישנם 2 כדורים, אחד מהם מסתובב סביב השני. בלחיצה על כפתור השמאלי של העכבר, הכדורים מחליפים תפקיד והכדור השני מתחיל מסתובב סביב הראשון. המטרה של המשחק היא לפגוע בקוביות. אבל תזהרו! אם תפגעו בקהות המסך תוחזרו אחורה ותאבדו נקודות!

**OrbGrab** is a simple game. You control a group of two balls. One of the balls spins around the other one. When you press the left mouse button, the balls switch roles and the second ball starts spinning around the first. The object of the game is to move the group and collect the cubes. But be careful! If you hit the edges of the screen, you will get bounced back and lose points!

המשחק הוא בשביל אנשים שרוצים להעביר קצת את הזמן. הוא אינו דורש יותר מדי מיומנות או חשיבה.

This game is for everyone that wants to kill a little time. It’s a casual game that doesn’t require too much skill or thinking.

מדריך למשתמש

כדי להריץ את הפרויקט ניתן להריץ את הקובץ

.OrbGrab.jar

הדבר היחיד שנדרש כדי להריץ את הקובץ הוא

Java (http://www.java.com/)

To play the game, simply double-click on the JAR (OrbGrab.jar) file. You must have the Java Runtime Environment installed (http://www.java.com/).

לאחר הרצת התכנית תופיע התפריט המרכזית

After you’ve launched the game, the following screen should pop up

בשביל להתחיל לשחק במשחק יש ללחוץ על אנטר או רווח. במהלך המשחק יש ללחוץ על "י" אם אתם צריכים עזרה.

To start the game, press ENTER or SPACE. If you need help during gameplay, press H.

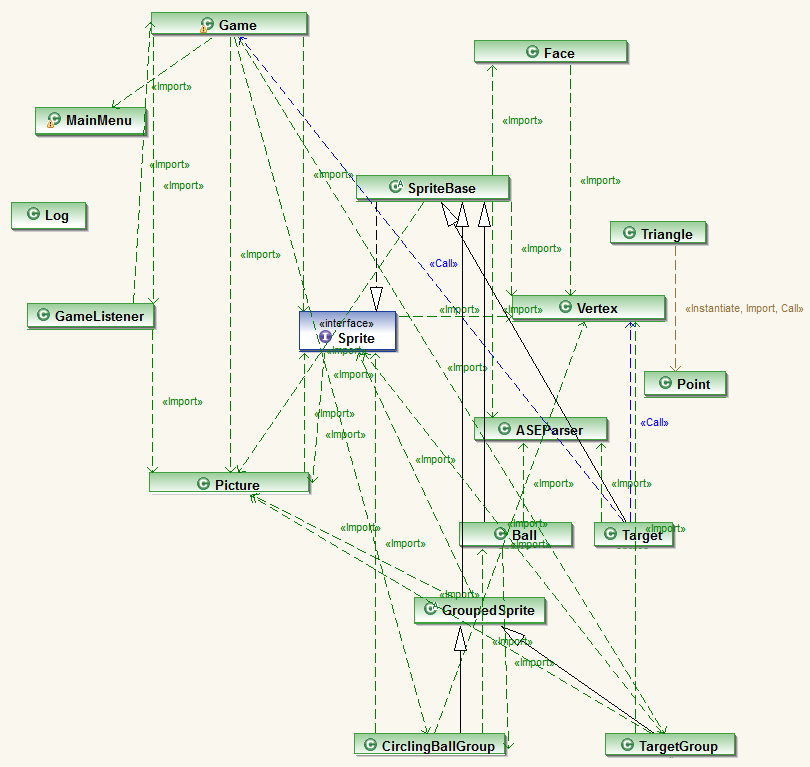


המשחק עצמו נראה כך. כדי לדעת איך לשחק יש להסתכל בעמוד 3.

The game itself looks like this. In order to find out how to play it, please refer to page 3 of this manual.

מדריך למפתח

|  |  |
| --- | --- |
| **שם הקובץ** | **תפקיד הקובץ** |
| Target.java | קלאס המייצג את הקובייה ואת הפעולות הקשורות אליה |
| Face.java | קלאס המייצג את הפאה |
| Game.java | קלאס האחראי על יצירת חלון בו מופיה כל הפאנלים |
| Picture.java | קלאס האחראי על ציור של הכל |
| GameListener.java | קלאס יישום ש כולל מאזינים לעכבר ומקלדת |
| ASEParser.java | קלאס שמנתח מודל |
| Vertex.java | קלאס המייצג את הקודקוד של פאת (נקודה תלת-מימדית) |
| Point.java | קלאס המייצג את הקודקוד של פאת |
| GroupedSprite.java | קלאס אב של קבוצות של ספרייטים |
| Sprite.java | ממשק אב של ספרייטים |
| Log.java | לוגר פשוט בשביל לעשות דיבאגינג |
| MainMenu.java | קלאס המייצג את התפריט הראשי |
| SpriteBase.java | קלאס אב של ספרייטים |
| Triangle.java | קלאס המייצג את הפאה בשני ממדים, משולש |
| Ball.java | קלאס המייצג את הכדור |
| CirclingBallGroup.java | קלאס המייצג את הקבוצה של הכדורים, השחקן |
| TargetGroup.java | קלאס המייצג את הקבוצה של המטרות |

****

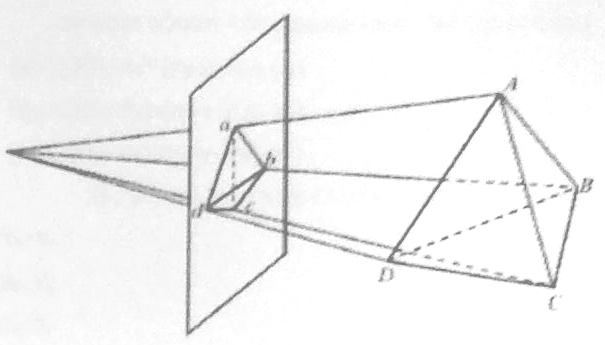
תאוריה

**הטלה פרספקטיבית**

הטלה זו היא טרנספורמציה של קואורדינאטות המאפשרת ייצוג של אובייקטים תלת ממדיים כתמונה דו ממדית.

הטלה פרספקטיבית מבוססת על הרעיון שעצמים רחוקים יותר נראים קטנים יותר. בניגוד להטלה מקבילית בה קווים מקבילים נמתחים עד שהם חותכים את מישור ההטלה, קווי ההטלה הפרספקטיבית אינם מקבילים ומתכנסים לנקודה הנקראת "מרכז ההטלה" שממוקמת על מישור ההטלה. נקודות חיתוך של קווי ההטלה עם מישור ההטלה קובעים את הטלתו של אובייקט על המישור.

התמונה המוטלת תלויה בנקודת מבט. מרכז ההטלה נמצא באינסוף ביחס לצופה.



**הטלה פרספקטיבית**

בדיקת הקוסינוס הזווית שבין וקטור נורמל של מישור הפאה לבין ווקטור העובר בין נקודת מרכז של פאה לנקודת האחרת.

וקטור נורמל של הפאה מחשבים לפי מכפלה קוטורית:

וקטור שבין נקודת הארה ונקודה על המישור:

את הקוסינוס מחשבים לפי מכפלה סקלארית של וקטורים:

לפי סימן של הקוסינוס ניתן לקבוע האם נראית בהתחשב בנקודת הארה.

**הצללה**

אלגוריתמים של הצללה זהים לאלגוריתמים של הסתרת פאות. באלגוריתמים של הסתרת פאות קובעים את הפאות שניתן לראות מנקודת הארה. לכן לקביעת הצבע בהתחשב במקור איר יש להכפיל את כל רכיבי הצבע של הפאה בקוסינוס הזווית שבין וקטור נורמל של מישור הפאה לבין ווקטור העובר בין נקודות מרכז של פאה לנקודת הארה.

סיכום אישי

במהלך עבודתי על הפרויקט קיבלתי ידע וניסיון בדברים כמו בשימוש שיטות שונות של גרפיקה תלת-מימדית ובניית תוכנות מורכבות. למדתי להתמודד עם פרויקטים גדולים ולארגן את עבודתי.

JavaDoc

## org.br.game  Class ASEParser

java.lang.Object

extended by **org.br.game.ASEParser**

public class **ASEParser**

extends java.lang.Object

Reads .ASE (and similar format) files and parses them. Saves the needed information (Vertexes and faces) into arrays.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **ASEParser**(java.lang.String filePath) |  |
| **ASEParser**(java.lang.String filePath, int bufferSize) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| int | **getBufferSize**() |
| java.lang.String | **getFilePath**() |
| int | **getNUMFACES**() |
| int | **getNUMVERTEX**() |
| java.util.ArrayList<java.lang.String> | **read**()            Reads the ASE file and returns an ArrayList which contains all information, also saves number of Faces and vertexes. |
| double[] | **readASEFace**()            Returns a double[] array which contains information for every face (Index of face and numbers of the vertexes in the face). |
| double[] | **readASEVertex**()            Returns a double[] array which contains information for every vertex (x, y, and z). |
| int | **readFACENUM**()            Sets the value for the amount of faces (FACENUM). |
| int | **readVERTEXNUM**()            Returns the value for the amount of vertexes (VERTEXNUM). |

## org.br.game.sprites  Class Ball

java.lang.Object

extended by org.br.game.SpriteBase

extended by **org.br.game.sprites.Ball**

**All Implemented Interfaces:**

Sprite

public class **Ball**

extends SpriteBase

The Ball sprite.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **Ball**(ASEParser filereader, java.awt.Color color, java.lang.String name) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| Ball | **getCenterBall**()            Returns the center (non-spinning) ball. |
| void | **init**()            Checks if ball has a center ball, then checks if it's active (spinning) and calls the circleAround() function, if not, stops the thread, and if there's no center ball, do nothing. |
| void | **move**(double x, double y, double z)            If the ball hits the edge of the screen, bounce it back 200 pixels. |
| void | **setActive**(boolean active) |
| void | **setCenterBall**(Ball centerBall) |
| void | **setGroup**(CirclingBallGroup owner)            Sets the group for each ball so it's aware of the other ball. |
| java.lang.String | **toString**() |

|  |
| --- |
| **Methods inherited from class org.br.game.SpriteBase** |
| getCenter, getColor, getFaces, getFileReader, getName, getPicture, getVertexes, isActive, paint, perspectiveProjection, repaintAll, setColor, setName, setPicture,turnX, turnY, turnZ, zoom |

## org.br.game.sprites  Class CirclingBallGroup

java.lang.Object

extended by org.br.game.SpriteBase

extended by org.br.game.GroupedSprite

extended by **org.br.game.sprites.CirclingBallGroup**

**All Implemented Interfaces:**

Sprite

public class **CirclingBallGroup**

extends GroupedSprite

The concrete implementation of GroupedSprite with the group of 2 moving ball sprites. One of the sprites is still, the second circles around it.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **CirclingBallGroup**(java.util.List<Sprite> sprites) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| Ball | **getCenterBall**()            Returns the stationary ball. |
| Ball | **getCirclingBall**()            Returns the circling ball. |
| boolean | **isActive**() |
| void | **move**(double x, double y, double z)            Calls the move() method for all faces in a sprite. |
| void | **setActive**(boolean flag) |
| void | **switchSpinning**()            Switches between the active sprites. |
| void | **turnX**(double a, Vertex center)            Calls the turnX() method for all faces in a sprite. |
| void | **turnY**(double a, Vertex center)            Calls the turnY() method for all faces in a sprite. |
| void | **turnZ**(double a, Vertex center)            Calls the turnZ() method for all faces in a sprite. |
| void | **zoom**(double a, Vertex center)            Calls the zoom() method for all faces in a sprite. |

## org.br.game  Class Face

java.lang.Object

extended by **org.br.game.Face**

public class **Face**

extends java.lang.Object

The Face (3D triangle) class. Has X, Y, and Z coordinates.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **Face**(Vertex p1, Vertex p2, Vertex p3, java.awt.Color color) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| Vertex | **getCenter**()            Returns the center vertex of the face. |
| java.awt.Color | **getColor**() |
| Vertex | **getP1**() |
| Vertex | **getP2**() |
| Vertex | **getP3**() |
| double | **howSeen**()            Calculates the cosine between the view angle and the normal of the face. |
| void | **move**(double dx, double dy, double dz)            Moves the face by the given coordinates |
| Triangle | **perspectiveProjection**(java.awt.Color color)            Calls perspectiveProjection() for each vertex. |
| void | **scale**(double sx, double sy, double sz)            Scales the face up or down depending on the given values. |
| void | **setColor**(java.awt.Color color) |
| java.awt.Color | **setRGBColor**(double viewAngle)            Changes the face color depending on the view angle given. |
| void | **turnX**(double a)            Turns the face around the X axis |
| void | **turnY**(double a)            Turns the face around the Y axis |
| void | **turnZ**(double a)            Turns the face around the Z axis |
| void | **zoom**(double k, Vertex center)            Moves the face to 0, 0, 0, scales it by the given coordinate, and then moves it back to its previous position. |

## org.br.game  Class Game

java.lang.Object

extended by java.awt.Component

extended by java.awt.Container

extended by java.awt.Window

extended by java.awt.Frame

extended by javax.swing.JFrame

extended by **org.br.game.Game**

**All Implemented Interfaces:**

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

public class **Game**

extends javax.swing.JFrame

The main class of OrgGrab, starts the game itself.

**See Also:**

Serialized Form

|  |
| --- |
| **Nested Class Summary** |

|  |
| --- |
| **Nested classes/interfaces inherited from class javax.swing.JFrame** |
| javax.swing.JFrame.AccessibleJFrame |

|  |
| --- |
| **Nested classes/interfaces inherited from class java.awt.Frame** |
| java.awt.Frame.AccessibleAWTFrame |

|  |
| --- |
| **Nested classes/interfaces inherited from class java.awt.Window** |
| java.awt.Window.AccessibleAWTWindow |

|  |
| --- |
| **Nested classes/interfaces inherited from class java.awt.Container** |
| java.awt.Container.AccessibleAWTContainer |

|  |
| --- |
| **Nested classes/interfaces inherited from class java.awt.Component** |
| java.awt.Component.AccessibleAWTComponent, java.awt.Component.BaselineResizeBehavior, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy |

|  |
| --- |
| **Field Summary** |

|  |
| --- |
| **Fields inherited from class javax.swing.JFrame** |
| accessibleContext, EXIT\_ON\_CLOSE, rootPane, rootPaneCheckingEnabled |

|  |
| --- |
| **Fields inherited from class java.awt.Frame** |
| CROSSHAIR\_CURSOR, DEFAULT\_CURSOR, E\_RESIZE\_CURSOR, HAND\_CURSOR, ICONIFIED, MAXIMIZED\_BOTH, MAXIMIZED\_HORIZ, MAXIMIZED\_VERT, MOVE\_CURSOR, N\_RESIZE\_CURSOR, NE\_RESIZE\_CURSOR, NORMAL, NW\_RESIZE\_CURSOR, S\_RESIZE\_CURSOR, SE\_RESIZE\_CURSOR, SW\_RESIZE\_CURSOR, TEXT\_CURSOR, W\_RESIZE\_CURSOR, WAIT\_CURSOR |

|  |
| --- |
| **Fields inherited from class java.awt.Component** |
| BOTTOM\_ALIGNMENT, CENTER\_ALIGNMENT, LEFT\_ALIGNMENT, RIGHT\_ALIGNMENT, TOP\_ALIGNMENT |

|  |
| --- |
| **Fields inherited from interface javax.swing.WindowConstants** |
| DISPOSE\_ON\_CLOSE, DO\_NOTHING\_ON\_CLOSE, HIDE\_ON\_CLOSE |

|  |
| --- |
| **Fields inherited from interface java.awt.image.ImageObserver** |
| ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **Game**(java.lang.String st) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| java.awt.Color | **getCenterBallColor**() |
| static Game | **getGame**() |
| java.awt.Color | **getMovingBallColor**() |
| CirclingBallGroup | **getPlayer**() |
| int | **getRadius**() |
| int | **getScore**() |
| long | **getSleepTime**() |
| boolean | **getStartGame**() |
| java.awt.Color | **getTargetColor**() |
| TargetGroup | **getTargets**() |
| void | **hit**(Sprite target)            Tells the game what to do when the player hits a target. |
| static void | **main**(java.lang.String[] args) |
| void | **outOfBorder**()            If the player hits the edge of the canvas, subtract 1 from the score. |
| void | **setCenterBallColor**(java.awt.Color centerBallColor) |
| void | **setMovingBallColor**(java.awt.Color movingBallColor) |
| void | **setStartGame**(boolean startGame) |
| void | **setTargetColor**(java.awt.Color targetColor) |
| void | **startGame**()            If not in the main menu, starts the game. |

## org.br.game  Class GameListener

java.lang.Object

extended by **org.br.game.GameListener**

**All Implemented Interfaces:**

java.awt.event.KeyListener, java.awt.event.MouseListener, java.awt.event.MouseMotionListener, java.util.EventListener, javax.swing.event.MouseInputListener

public class **GameListener**

extends java.lang.Object

implements javax.swing.event.MouseInputListener, java.awt.event.KeyListener

Listener for the game class. Contains mouse and key actions for controlling the game.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **GameListener**(Game game, Picture pic) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| void | **keyPressed**(java.awt.event.KeyEvent e) |
| void | **keyReleased**(java.awt.event.KeyEvent e) |
| void | **keyTyped**(java.awt.event.KeyEvent e) |
| void | **mouseClicked**(java.awt.event.MouseEvent e)            On mouse click, calls the switchSpinning() function which switches the active ball. |
| void | **mouseDragged**(java.awt.event.MouseEvent e) |
| void | **mouseEntered**(java.awt.event.MouseEvent e) |
| void | **mouseExited**(java.awt.event.MouseEvent e) |
| void | **mouseMoved**(java.awt.event.MouseEvent e) |
| void | **mousePressed**(java.awt.event.MouseEvent e) |
| void | **mouseReleased**(java.awt.event.MouseEvent e) |

## org.br.game  Class GroupedSprite

java.lang.Object

extended by org.br.game.SpriteBase

extended by **org.br.game.GroupedSprite**

**All Implemented Interfaces:**

Sprite

**Direct Known Subclasses:**

CirclingBallGroup, TargetGroup

public abstract class **GroupedSprite**

extends SpriteBase

Abstract class which is a parent of all grouped sprites.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **GroupedSprite**(java.util.List<? extends Sprite> group) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| java.util.List<Sprite> | **getGroup**() |
| protected  int | **getRadius**() |
| void | **init**() |
| void | **paint**(java.awt.Graphics g) |
| void | **setPicture(Picture picture)** |

## org.br.game  Class Log

java.lang.Object

extended by **org.br.game.Log**

public class **Log**

extends java.lang.Object

Simple logger TODO - use some logging framework !

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **Log**() |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| static void | **error**(java.lang.Class<?> clazz, java.lang.Throwable ex) |
| static void | **info**(java.lang.Class<?> clazz, java.lang.String mess) |
| static void | **warn**(java.lang.Class<?> clazz, java.lang.String mess) |

## org.br.game  Class MainMenu

java.lang.Object

extended by java.awt.Component

extended by java.awt.Container

extended by javax.swing.JComponent

extended by javax.swing.JPanel

extended by **org.br.game.MainMenu**

**All Implemented Interfaces:**

java.awt.event.KeyListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible

public class **MainMenu**

extends javax.swing.JPanel

implements java.awt.event.KeyListener

Paints Main Menu and adds listeners to it.

**See Also:**

Serialized Form

|  |
| --- |
| **Nested Class Summary** |

|  |
| --- |
| **Nested classes/interfaces inherited from class javax.swing.JPanel** |
| javax.swing.JPanel.AccessibleJPanel |

|  |
| --- |
| **Nested classes/interfaces inherited from class javax.swing.JComponent** |
| javax.swing.JComponent.AccessibleJComponent |

|  |
| --- |
| **Nested classes/interfaces inherited from class java.awt.Container** |
| java.awt.Container.AccessibleAWTContainer |

|  |
| --- |
| **Nested classes/interfaces inherited from class java.awt.Component** |
| java.awt.Component.AccessibleAWTComponent, java.awt.Component.BaselineResizeBehavior, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy |

|  |
| --- |
| **Field Summary** |

|  |
| --- |
| **Fields inherited from class javax.swing.JComponent** |
| accessibleContext, listenerList, TOOL\_TIP\_TEXT\_KEY, ui, UNDEFINED\_CONDITION, WHEN\_ANCESTOR\_OF\_FOCUSED\_COMPONENT, WHEN\_FOCUSED, WHEN\_IN\_FOCUSED\_WINDOW |

|  |
| --- |
| **Fields inherited from class java.awt.Component** |
| BOTTOM\_ALIGNMENT, CENTER\_ALIGNMENT, LEFT\_ALIGNMENT, RIGHT\_ALIGNMENT, TOP\_ALIGNMENT |

|  |
| --- |
| **Fields inherited from interface java.awt.image.ImageObserver** |
| ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **MainMenu**() |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| void | **keyPressed**(java.awt.event.KeyEvent arg0) |
| void | **keyReleased**(java.awt.event.KeyEvent arg0) |
| void | **keyTyped**(java.awt.event.KeyEvent arg0) |
| void | **paint**(java.awt.Graphics g) |

## org.br.game  Class Picture

java.lang.Object

extended by java.awt.Component

extended by java.awt.Container

extended by javax.swing.JComponent

extended by javax.swing.JPanel

extended by **org.br.game.Picture**

**All Implemented Interfaces:**

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

public class **Picture**

extends javax.swing.JPanel

This class handles all of the painting.

**See Also:**

Serialized Form

|  |
| --- |
| **Nested Class Summary** |

|  |
| --- |
| **Nested classes/interfaces inherited from class javax.swing.JPanel** |
| javax.swing.JPanel.AccessibleJPanel |

|  |
| --- |
| **Nested classes/interfaces inherited from class javax.swing.JComponent** |
| javax.swing.JComponent.AccessibleJComponent |

|  |
| --- |
| **Nested classes/interfaces inherited from class java.awt.Container** |
| java.awt.Container.AccessibleAWTContainer |

|  |
| --- |
| **Nested classes/interfaces inherited from class java.awt.Component** |
| java.awt.Component.AccessibleAWTComponent, java.awt.Component.BaselineResizeBehavior, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy |

|  |
| --- |
| **Field Summary** |

|  |
| --- |
| **Fields inherited from class javax.swing.JComponent** |
| accessibleContext, listenerList, TOOL\_TIP\_TEXT\_KEY, ui, UNDEFINED\_CONDITION, WHEN\_ANCESTOR\_OF\_FOCUSED\_COMPONENT, WHEN\_FOCUSED, WHEN\_IN\_FOCUSED\_WINDOW |

|  |
| --- |
| **Fields inherited from class java.awt.Component** |
| BOTTOM\_ALIGNMENT, CENTER\_ALIGNMENT, LEFT\_ALIGNMENT, RIGHT\_ALIGNMENT, TOP\_ALIGNMENT |

|  |
| --- |
| **Fields inherited from interface java.awt.image.ImageObserver** |
| ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **Picture**() |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| Sprite | **getCenterBall**() |
| java.util.List<Sprite> | **getListOfSprites**() |
| Sprite | **getSpinningBall**() |
| void | **paint**(java.awt.Graphics g) |
| void | **setCenterBall**(Sprite centerBall) |
| void | **setListOfSprites**(java.util.List<Sprite> listOfSprites) |
| void | **setSpinningBall**(Sprite spinningBall) |

## org.br.game  Class Point

java.lang.Object

extended by **org.br.game.Point**

public class **Point**

extends java.lang.Object

Basic point class. Has x and y coordinates.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **Point**(double x, double y) |  |
| **Point**(Point p) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| double | **getX**() |
| double | **getY**() |
| void | **move**(double dx, double dy)            Moves the point by the given coordinates. |
| void | **scale**(double sx, double sy)            Scales the point up or down depending on the given values. |
| void | **setX**(double x) |
| void | **setY**(double y) |
| void | **turn**(double a)            Turns the point in a circle based on the given angle. |

## org.br.game  Interface Sprite

**All Known Implementing Classes:**

Ball, CirclingBallGroup, GroupedSprite, SpriteBase, Target, TargetGroup

public interface **Sprite**

Sprite interface. Contains all needed methods for sprites.

|  |  |
| --- | --- |
| **Method Summary** | |
| Vertex | **getCenter**() |
| void | **init**() |
| boolean | **isActive**() |
| void | **move**(double x, double y, double z) |
| void | **paint**(java.awt.Graphics g) |
| void | **repaintAll**() |
| void | **setActive**(boolean flag) |
| void | **setPicture**(Picture picture) |
| void | **turnX**(double a, Vertex center) |
| void | **turnY**(double a, Vertex center) |
| void | **turnZ**(double a, Vertex center) |
| void | **zoom**(double a, Vertex center) |

## org.br.game  Class SpriteBase

java.lang.Object

extended by **org.br.game.SpriteBase**

**All Implemented Interfaces:**

Sprite

**Direct Known Subclasses:**

Ball, GroupedSprite, Target

public abstract class **SpriteBase**

extends java.lang.Object

implements Sprite

Base abstract class for all sprites.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **SpriteBase**() |  |
| **SpriteBase**(ASEParser filereader, java.awt.Color color, java.lang.String name) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| Vertex | **getCenter**()            Returns the center vertex of the sprite. |
| java.awt.Color | **getColor**() |
| Face[] | **getFaces**() |
| ASEParser | **getFileReader**() |
| java.lang.String | **getName**() |
| Picture | **getPicture**() |
| Vertex[] | **getVertexes**() |
| void | **init**() |
| boolean | **isActive**() |
| void | **move**(double x, double y, double z)            Calls the move() method for all faces in a sprite. |
| void | **paint**(java.awt.Graphics g) |
| java.util.ArrayList<Triangle> | **perspectiveProjection**()            For each face, checks if it should be seen using the howSeen() method, gives it shading using the setRGBColor() method and turns the faces into triangles using perspectiveProjection(). |
| void | **repaintAll**() |
| void | **setActive**(boolean active) |
| void | **setColor**(java.awt.Color color) |
| void | **setName**(java.lang.String name) |
| void | **setPicture**(Picture picture) |
| void | **turnX**(double a, Vertex center)            Calls the turnX() method for all faces in a sprite. |
| void | **turnY**(double a, Vertex p)            Calls the turnY() method for all faces in a sprite. |
| void | **turnZ**(double a, Vertex p)            Calls the turnZ() method for all faces in a sprite. |
| void | **zoom**(double a, Vertex center)            Calls the zoom() method for all faces in a sprite. |

## org.br.game.sprites  Class Target

java.lang.Object

extended by org.br.game.SpriteBase

extended by **org.br.game.sprites.Target**

**All Implemented Interfaces:**

Sprite

public class **Target**

extends SpriteBase

The target class.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **Target**(ASEParser filereader, java.awt.Color color, java.lang.String name) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| void | **move**(double x, double y, double z)            Checks if the cube went out of bounds of the screen and if it did, move it back. |
| void | **rotate**()            Basic thread to animate rotation of the cube. |

## org.br.game.sprites  Class TargetGroup

java.lang.Object

extended by org.br.game.SpriteBase

extended by org.br.game.GroupedSprite

extended by **org.br.game.sprites.TargetGroup**

**All Implemented Interfaces:**

Sprite

public class **TargetGroup**

extends GroupedSprite

The concrete implementation of GroupedSprite with the group of target sprites.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **TargetGroup**(java.util.List<Sprite> sprites) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| Sprite | **collisionDetected**(Sprite other)            Used for collision detection. |
| Vertex | **getCenter**()            Returns the center vertex of the sprite. |
| boolean | **isActive**() |
| void | **move**(double x, double y, double z)            Calls the move() method for all faces in a sprite. |
| void | **repaintAll**() |
| void | **setActive**(boolean flag) |
| void | **setPicture**(Picture picture) |
| void | **turnX**(double a, Vertex center)            Calls the turnX() method for all faces in a sprite. |
| void | **turnY**(double a, Vertex center)            Calls the turnY() method for all faces in a sprite. |
| void | **turnZ**(double a, Vertex center)            Calls the turnZ() method for all faces in a sprite. |
| void | **zoom**(double a, Vertex center)            Calls the zoom() method for all faces in a sprite. |

## org.br.game  Class Triangle

java.lang.Object

extended by **org.br.game.Triangle**

public class **Triangle**

extends java.lang.Object

Triangle class made up of 3 Points.

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **Triangle**(Point p1, Point p2, Point p3, java.awt.Color color) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| Point | **getCenter**()            Returns the center point of the triangle. |
| void | **move**(double dx, double dy)            For each point in the triangle, calls the move() method and moves all the points. |
| void | **paint**(java.awt.Graphics g) |
| void | **rotate**(double a)            Rotates all the points in the triangle. |
| void | **scale**(double sx, double sy)            Scales each point in the triangle. |
| void | **turn**(double a)            Turns each point in the triangle. |
| void | **turn**(double a, Point center)            Turns each point in the triangle. |
| void | **zoom**(double k)            Zooms each point in the triangle. |

## org.br.game  Class Vertex

java.lang.Object

extended by **org.br.game.Vertex**

public class **Vertex**

extends java.lang.Object

Same as Point class, but has 3 coordinates: X, Y, and Z.

|  |  |
| --- | --- |
| **Field Summary** | |
| static int | **DIST** |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| **Vertex**(double x, double y, double z) |  |
| **Vertex**(Vertex p) |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| boolean | **equals**(java.lang.Object other)            Used for collision detection. |
| double | **getLength**()            Gets the length of the vertex. |
| double | **getX**() |
| double | **getY**() |
| double | **getZ**() |
| void | **move**(double dx, double dy, double dz)            Moves all the coordinates by the given values. |
| Point | **perspectiveProjection**()            Turns a 3D vertex into a 2D point. |
| void | **scale**(double sx, double sy, double sz)            Scales all the coordinates by the given values. |
| void | **setX**(double x) |
| void | **setY**(double y) |
| void | **setZ**(double z) |
| java.lang.String | **toString**() |
| void | **turnX**(double a)            Turns the vertex around the X axis. |
| void | **turnY**(double a)            Turns the vertex around the Y axis. |
| void | **turnZ**(double a)            Turns the vertex around the Z axis. |

Code

**package** org.br.game;

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.util.ArrayList;

**import** java.util.StringTokenizer;

/\*\*

\* Reads .ASE (and similar format) files and parses them. Saves the needed information (Vertexes and faces) into arrays.

\*

\* **@author** Boris

\*/

**public** **class** ASEParser {

**private** String filePath = **null**;

**private** **int** bufferSize = 1000;

**private** **int** NUMVERTEX, NUMFACES;

**public** ASEParser(String filePath) {

setFilePath(filePath);

setBufferSize(1000);

}

**public** ASEParser(String filePath, **int** bufferSize) {

setFilePath(filePath);

setBufferSize(bufferSize);

}

**public** **int** getBufferSize() {

**return** bufferSize;

}

**public** String getFilePath() {

**return** filePath;

}

**public** **int** getNUMFACES() {

**return** NUMFACES;

}

**public** **int** getNUMVERTEX() {

**return** NUMVERTEX;

}

**private** **void** setBufferSize(**int** bufferSize) {

**this**.bufferSize = bufferSize;

}

**private** **void** setFilePath(String filePath) {

**this**.filePath = filePath;

}

/\*\*

\* Reads the ASE file and returns an ArrayList which contains all information, also saves number of Faces and vertexes.

\*/

**public** ArrayList<String> read() **throws** java.io.FileNotFoundException, java.io.IOException {

FileReader fr = **new** FileReader(getFilePath());

BufferedReader br = **new** BufferedReader(fr);

ArrayList<String> aList = **new** ArrayList<String>(getBufferSize());

String line = **null**;

**while** ((line = br.readLine()) != **null**) {

aList.add(line);

}

br.close();

fr.close();

**return** aList;

}

/\*\*

\* Sets the value for the amount of faces (FACENUM).

\*/

**public** **int** readFACENUM() {

**int** numOfFaces = 0;

**try** {

ArrayList<?> arrFile = read();

String line = (String) arrFile.get(1); // Saves first line in String

StringTokenizer token = **new** StringTokenizer(line, " ");

**while** (token.hasMoreTokens()) {

token.nextToken();// Skips \*MESH\_NUMVERTEX

numOfFaces = Integer.*parseInt*(token.nextToken());// Saves the 2nd value in NUMFACES

NUMFACES = numOfFaces;

}

}

**catch** (Exception e) {

System.*out*.println(e.getMessage());

}

**return** numOfFaces;

}

/\*\*

\* Returns the value for the amount of vertexes (VERTEXNUM).

\*/

**public** **int** readVERTEXNUM() {

**int** numOfVertex = 0;

**try** {

ArrayList<?> arrFile = read();

String line = (String) arrFile.get(0); // Saves second line in String

StringTokenizer token = **new** StringTokenizer(line, " ");

**while** (token.hasMoreTokens()) {

token.nextToken();// Skips \*MESH\_NUMFACES

numOfVertex = Integer.*parseInt*(token.nextToken());// Saves the 2nd value in NUMVERTEX

NUMVERTEX = numOfVertex;

}

}

**catch** (Exception e) {

System.*out*.println(e.getMessage());

}

**return** numOfVertex;

}

/\*\*

\* Returns a double[] array which contains information for every vertex (x, y, and z).

\*/

**public** **double**[] readASEVertex() {

String line;

**double**[] arr = **new** **double**[0];// Declared

**try** {

ArrayList<?> arrFile = read();

**this**.readVERTEXNUM();

arr = **new** **double**[NUMVERTEX \* 3];// Makes array which is NUMVERTEX\*3 long, because there are 3 values for every vertex

**if** (arrFile.size() > 4) // Checks if the file has more than 1 vertex

{

**int** index = 0;// index of the array

**for** (**int** i = 3; i < NUMVERTEX + 4; i++) { // starts at line 4, i is for lines, j is for vertex values

line = (String) arrFile.get(i); // next line

parseLine(line, arr, index);

index += 3;

}

}

}

**catch** (Exception e) {

System.*out*.println(e.getMessage());

}

**return** arr;

}

/\*\*

\* Returns a double[] array which contains information for every face (Index of face and numbers of the vertexes in the face).

\*/

**public** **double**[] readASEFace() {

String line;

**double**[] arr = **new** **double**[0];

**try** {

ArrayList<?> arrFile = read();

**this**.readFACENUM();

arr = **new** **double**[NUMFACES \* 3];// Makes array which is NUMFACES\*3 long, because there are 3 values for every faces

**int** linecounter = getNUMVERTEX() + 2;

String templine = (String) arrFile.get(linecounter);

String test = "\*MESH\_FACE";

String test1 = "\*MESH\_FACE\_";

**while** (!templine.contains(test1) && !templine.contains(test)) {// We have to find the line where vertexes end and faces begin

linecounter++;

templine = (String) arrFile.get(linecounter);

}

**if** (templine.contains(test1)) {

linecounter++;

}

**int** index = 0;// index of the array

**for** (**int** i = linecounter; i < NUMFACES + linecounter; i++) {

line = (String) arrFile.get(i); // next line

parseLine(line, arr, index);

index += 3;// parseLine can't change the global index, so we have to change it here

}

}

**catch** (Exception e) {

System.*out*.println(e.getMessage());

}

**return** arr;

}

/\*\*

\* Gets current line, the array to fill, and the current index of the array. Parses the line and fills the values from the current line into the array.

\*

\* **@param** line

\* **@param** arr

\* **@param** index

\*/

**private** **void** parseLine(String line, **double**[] arr, **int** index) {

StringTokenizer st = **new** StringTokenizer(line, ": \t");

**int** counter = 0;

**if** (line.contains(("FACE"))) {// for Faces

**while** (st.hasMoreTokens()) {

**if** (counter == 3 || counter == 5 || counter == 7) {// We need the 4th, 6th, and 8th values, the rest we can discard

arr[index] = Double.*parseDouble*(st.nextToken());

index++;

}

**else** {

st.nextToken();// Skips the token.

}

counter++;

}

}

**else** {

**while** (st.hasMoreTokens()) {// for Vertexes

**if** (counter == 2 || counter == 3 || counter == 4) {// We need the 3rd, 4th, and 5th values, the rest we can discard

arr[index] = Double.*parseDouble*(st.nextToken());

index++;

}

**else** {

st.nextToken();// Skips the token.

}

counter++;

}

}

}

}

**package** org.br.game;

**import** java.awt.\*;

/\*\*

\* The Face (3D triangle) class. Has X, Y, and Z coordinates.

\*

\* **@author** Boris

\*/

**public** **class** Face {

**private** Vertex p1, p2, p3;

**private** Color color;

**public** Face(Vertex p1, Vertex p2, Vertex p3, Color color) {

**this**.p1 = **new** Vertex(p1);

**this**.p2 = **new** Vertex(p2);

**this**.p3 = **new** Vertex(p3);

**this**.color = color;

}

**public** Vertex getP1() {

**return** **new** Vertex(p1);

}

**public** Vertex getP2() {

**return** **new** Vertex(p2);

}

**public** Vertex getP3() {

**return** **new** Vertex(p3);

}

/\*\*

\* Moves the face by the given coordinates

\*

\* **@param** dx

\* **@param** dy

\* **@param** dz

\*/

**public** **void** move(**double** dx, **double** dy, **double** dz) {

p1.move(dx, dy, dz);

p2.move(dx, dy, dz);

p3.move(dx, dy, dz);

}

/\*\*

\* Turns the face around the X axis

\*

\* **@param** a

\*/

**public** **void** turnX(**double** a) {

p1.turnX(a);

p2.turnX(a);

p3.turnX(a);

}

/\*\*

\* Turns the face around the Y axis

\*

\* **@param** a

\*/

**public** **void** turnY(**double** a) {

p1.turnY(a);

p2.turnY(a);

p3.turnY(a);

}

/\*\*

\* Turns the face around the Z axis

\*

\* **@param** a

\*/

**public** **void** turnZ(**double** a) {

p1.turnZ(a);

p2.turnZ(a);

p3.turnZ(a);

}

/\*\*

\* Scales the face up or down depending on the given values.

\*

\* **@param** sx

\* **@param** sy

\* **@param** sz

\*/

**public** **void** scale(**double** sx, **double** sy, **double** sz) {

p1.scale(sx, sy, sz);

p2.scale(sx, sy, sz);

p3.scale(sx, sy, sz);

}

/\*\*

\* Moves the face to 0, 0, 0, scales it by the given coordinate, and then moves it back to its previous position.

\*

\* **@param** k

\* **@param** center

\*/

**public** **void** zoom(**double** k, Vertex center) {

move(-(center.getX()), -(center.getY()), -(center.getZ()));

scale(k, k, k);

move((center.getX()), (center.getY()), (center.getZ()));

}

**public** Color getColor() {

**return** color;

}

**public** **void** setColor(Color color) {

**this**.color = color;

}

/\*\*

\* Changes the face color depending on the view angle given.

\*

\* **@param** viewAngle

\* **@return**

\*/

**public** Color setRGBColor(**double** viewAngle) {

viewAngle = Math.*abs*(viewAngle);

**int** red, green, blue;

red = color.getRed();

green = color.getGreen();

blue = color.getBlue();

**return** **new** Color((**int**) (red \* (0.35 + 0.65 \* viewAngle)), (**int**) (green \* (viewAngle \* 0.35 + 0.65)), (**int**) (blue \* (viewAngle \* 0.65 + 0.35)));

}

/\*\*

\* Returns the center vertex of the face.

\*

\* **@return**

\*/

**public** Vertex getCenter() {

**return** (**new** Vertex((p1.getX() + p2.getX() + p3.getX()) / 3, (p1.getY() + p2.getY() + p3.getY()) / 3, (p1.getZ() + p2.getZ() + p3.getZ()) / 3));

}

/\*\*

\* Calculates the cosine between the view angle and the normal of the face. If the result is positive then we can see it, if negative, then we can't see it and there's no need to draw it.

\*

\* **@return**

\*/

**public** **double** howSeen() {

Vertex normal, middle;

Vertex p1 = getP1();

Vertex p2 = getP2();

Vertex p3 = getP3();

**double** tmp1, tmp2, tmp3;

middle = getCenter();

p1.move(-p2.getX(), -p2.getY(), -p2.getZ());

p3.move(-p2.getX(), -p2.getY(), -p2.getZ());

tmp1 = p1.getY() \* p3.getZ() - p1.getZ() \* p3.getY();

tmp2 = p1.getZ() \* p3.getX() - p1.getX() \* p3.getZ();

tmp3 = p1.getX() \* p3.getY() - p1.getY() \* p3.getX();

normal = **new** Vertex(tmp1, tmp2, tmp3);

middle.move(0, 0, Vertex.*DIST*);

tmp1 = 1 / normal.getLength();

tmp2 = 1 / middle.getLength();

normal.scale(tmp1, tmp1, tmp1);

middle.scale(tmp2, tmp2, tmp2);

tmp3 = normal.getX() \* middle.getX() + normal.getY() \* middle.getY() + normal.getZ() \* middle.getZ();

**return** tmp3;

}

/\*\*

\* Calls perspectiveProjection() for each vertex.

\*

\* **@param** color

\* **@return**

\*/

**public** Triangle perspectiveProjection(Color color) {

**return** **new** Triangle(getP1().perspectiveProjection(), getP2().perspectiveProjection(), getP3().perspectiveProjection(), color);

}}

**package** org.br.game;

**import** java.awt.BorderLayout;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.Graphics;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.io.FileReader;

**import** java.io.Reader;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Properties;

**import** java.util.Random;

**import** javax.swing.JFrame;

**import** javax.swing.JMenu;

**import** javax.swing.JMenuBar;

**import** javax.swing.JMenuItem;

**import** javax.swing.JOptionPane;

**import** org.br.game.sprites.Ball;

**import** org.br.game.sprites.CirclingBallGroup;

**import** org.br.game.sprites.Target;

**import** org.br.game.sprites.TargetGroup;

/\*\*

\* The main class of OrgGrab, starts the game itself.

\*

\* **@author** Boris

\*/

**public** **class** Game **extends** JFrame {

**private** Picture pic;

**static** Game *GAME*;

**private** MainMenu mainMenu = **new** MainMenu();

**private** GameListener gamelistener;

**private** CirclingBallGroup player;

**private** TargetGroup targets;

**private** **boolean** startGame = **false**;

**private** Graphics g;

**private** List<Sprite> sprites = **new** ArrayList<Sprite>();

**private** String ballModel = "models/small\_ball.ase";

**private** String cubeModel = "models/cubik1.ase";

**private** **int** score = 0;

// ///////////////////////////////////// Default properties

**private** **int** targetNumber = 2;

**private** **int** radius = 60;

**private** **long** sleepTime = 100L;

**private** Color movingBallColor = Color.*GREEN*, centerBallColor = Color.*GRAY*, targetColor = Color.*RED*;

**public** Game(String st) {

**super**(st);

setPreferredSize(**new** Dimension(800, 600));

setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);

g = getGraphics();

}

/\*\*

\* Adds GameListener (MouseListener + KeyListener) to pic (JPanel)

\*/

**private** **void** addPicListener() {

gamelistener = **new** GameListener(**this**, pic);

pic.addMouseListener(gamelistener);

pic.addMouseMotionListener(gamelistener);

pic.addKeyListener(gamelistener);

}

/\*\*

\* Creates the Ball sprites, set the starting center ball, creates list of target sprites.

\*

\* **@return** Returns picture with the list of all the sprites for painting.

\*/

**private** Picture getGameContent() {

pic = **new** Picture();

Ball ball1 = **new** Ball(**new** ASEParser(ballModel), getMovingBallColor(), "Ball 1");

Ball ball2 = **new** Ball(**new** ASEParser(ballModel), getCenterBallColor(), "Ball 2");

List<Sprite> targetSprites = **new** ArrayList<Sprite>(5);

**for** (**int** i = 0; i < getTargetNumber(); i++) {

Target target = **new** Target(**new** ASEParser(cubeModel), getTargetColor(), "Target " + i);

targetSprites.add(target);

}

List<Sprite> playerSprites = **new** ArrayList<Sprite>(2);

playerSprites.add(ball1);

playerSprites.add(ball2);

**for** (Sprite sprite : playerSprites) {

sprite.zoom(2.5, sprite.getCenter());

}

**for** (Sprite sprite : targetSprites) {

sprite.zoom(0.2, sprite.getCenter());

}

ball1.setCenterBall(ball2);// Set Center ball as ball2

ball1.setActive(**true**);// Activate ball 1 (make it spin)

setPlayer(**new** CirclingBallGroup(playerSprites));

setTargets(**new** TargetGroup(targetSprites));

getPlayer().setPicture(pic);

getTargets().setPicture(pic);

ball1.setGroup(getPlayer());

ball2.setGroup(getPlayer());

sprites.add(getPlayer()); // adds player (CirclingBallGroup) to sprites (List of Sprites).

sprites.add(getTargets());// Adds target to list of sprites

pic.setListOfSprites(sprites);

addPicListener();

**return** pic;

}

/\*\*

\* If not in the main menu, starts the game. Moves the active ball to the starting orbit location (by radius). Initializes the group and the targets.

\*/

**public** **void** startGame() {

setVisible(**true**);

**if** (getStartGame()) {

getContentPane().remove(mainMenu);

getContentPane().add(getGameContent());

setJMenuBar(buildMenu());

pack();

repaint();

getPlayer().getCenterBall().move(getRadius() \* Math.*cos*(45), getRadius() \* Math.*sin*(45), 0);// Moves the center ball

getPlayer().move(150, 150, 0);// Moves the player group

getPlayer().init();

pic.setCenterBall(getPlayer().getCenterBall());

pic.setSpinningBall(getPlayer().getCirclingBall());

**for** (**int** i = 0; i < getTargets().getGroup().size(); i++) {

Random rand = **new** Random();

Target target = (Target) getTargets().getGroup().get(i);

target.move(rand.nextDouble() \* 200 \* i, rand.nextDouble() \* 300 \* i, 0);

target.rotate();

}

pic.grabFocus();

}

**else** {

setJMenuBar(**null**);

**if** (pic != **null**) {

getContentPane().remove(pic);

}

getContentPane().add(mainMenu, BorderLayout.*CENTER*);

pack();

mainMenu.grabFocus();

}

}

/\*\*

\* Builds the JMenu that's used in the game and returns it

\*

\* **@return**

\*/

**private** JMenuBar buildMenu() {

JMenuBar menubar = **new** JMenuBar();

JMenu filemenu;

JMenuItem exit, help, about;

filemenu = **new** JMenu("File");

menubar.add(filemenu);

help = **new** JMenuItem("Help");

filemenu.add(help);

about = **new** JMenuItem("About");

filemenu.add(about);

help.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

JOptionPane.*showMessageDialog*(pic, "OrbGrab is a simple game. You control a group of two balls. One of the balls spins around the other one. \nWhen you press the left mouse button, the balls switch roles and the second ball starts spinning around the first. \nThe object of the game is to move the group and collect all of the cubes to raise your score. \nBut stay away from the edges of the screen! If you hit the edge you get bounced back and lose points!");

}

});

about.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

JOptionPane.*showMessageDialog*(pic, "Made by Boris Rossovsky in 2011 for Shevah Mofet.\nSpecial thanks to Gennady Rashap and Michael Rossovsky.");

}

});

exit = **new** JMenuItem("Exit");

filemenu.addSeparator();

filemenu.add(exit);

exit.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

System.*exit*(0);

}

});

**return** menubar;

}

**public** CirclingBallGroup getPlayer() {

**return** player;

}

**private** **void** setPlayer(CirclingBallGroup player) {

**this**.player = player;

}

**public** **void** setStartGame(**boolean** startGame) {

**this**.startGame = startGame;

}

**public** **boolean** getStartGame() {

**return** startGame;

}

**public** **static** Game getGame() {

**return** *GAME*;

}

**public** TargetGroup getTargets() {

**return** targets;

}

**private** **void** setTargets(TargetGroup targets) {

**this**.targets = targets;

}

**public** **int** getScore() {

**return** score;

}

/\*\*

\* Tells the game what to do when the player hits a target. Increases the score by 1 and randomly changes the hit target's position.

\*

\* **@param** target

\*/

**public** **void** hit(Sprite target) {

score++;

Random rand = **new** Random();

**double** random1 = rand.nextDouble();

**double** random2 = rand.nextDouble();

**if** ((random1 < 0.5) || (random2 > 0.5)) {

random1 = -1 \* random1;

random2 = -1 \* random2;

}

target.move(random1 \* 100, random2 \* 100, 0);

}

/\*\*

\* If the player hits the edge of the canvas, subtract 1 from the score.

\*/

**public** **void** outOfBorder() {

score--;

}

/\*\*

\* Reads the configuration file "orbgrab.properties" and saves various properties from the file.

\*/

**private** **void** loadConfiguration() {

Properties properties = **new** Properties();

**try** {

Reader propertiesReader = **new** FileReader("orbgrab.properties");

properties.load(propertiesReader);

Log.*info*(getClass(), "Loaded config: " + properties);

setTargetNumber(Integer.*valueOf*(properties.getProperty("target.number")));

setRadius(Integer.*valueOf*(properties.getProperty("radius")));

setSleepTime(Long.*valueOf*(properties.getProperty("sleep.time")));

Color centerBall = Color.*decode*(properties.getProperty("color.center.ball"));

Color movingBall = Color.*decode*(properties.getProperty("color.moving.ball"));

Color target = Color.*decode*(properties.getProperty("color.target"));

setMovingBallColor(movingBall);

setCenterBallColor(centerBall);

setTargetColor(target);

}

**catch** (Throwable ex) {

Log.*error*(getClass(), ex);

}

}

**private** **int** getTargetNumber() {

**return** targetNumber;

}

**private** **void** setTargetNumber(**int** targetNumber) {

**this**.targetNumber = targetNumber;

}

**public** **int** getRadius() {

**return** radius;

}

**private** **void** setRadius(**int** radius) {

**this**.radius = radius;

}

**public** **long** getSleepTime() {

**return** sleepTime;

}

**private** **void** setSleepTime(**long** sleepTime) {

**this**.sleepTime = sleepTime;

}

**public** Color getMovingBallColor() {

**return** movingBallColor;

}

**public** **void** setMovingBallColor(Color movingBallColor) {

**this**.movingBallColor = movingBallColor;

}

**public** Color getCenterBallColor() {

**return** centerBallColor;

}

**public** **void** setCenterBallColor(Color centerBallColor) {

**this**.centerBallColor = centerBallColor;

}

**public** Color getTargetColor() {

**return** targetColor;

}

**public** **void** setTargetColor(Color targetColor) {

**this**.targetColor = targetColor;

}

**public** **static** **void** main(String[] args) {

*GAME* = **new** Game("OrbGrab");

*GAME*.loadConfiguration();

*GAME*.startGame();

}

}

**package** org.br.game;

**import** java.awt.event.InputEvent;

**import** java.awt.event.KeyEvent;

**import** java.awt.event.KeyListener;

**import** javax.swing.JOptionPane;

**import** javax.swing.event.MouseInputListener;

**import** java.awt.event.MouseEvent;

/\*\*

\* Listener for the game class. Contains mouse and key actions for controlling the game.

\*

\* **@author** Boris

\*/

**public** **class** GameListener **implements** MouseInputListener, KeyListener {

**private** Game game;

**private** Picture pic;

**public** GameListener(Game game, Picture pic) {

**this**.game = game;

**this**.pic = pic;

}

/\*\*

\* On mouse click, calls the switchSpinning() function which switches the active ball.

\*/

**public** **void** mouseClicked(MouseEvent e) {

**if** (e.getModifiers() == InputEvent.*BUTTON1\_MASK*) {

game.getPlayer().switchSpinning();

}

}

**public** **void** mouseReleased(MouseEvent e) {

}

**public** **void** mouseEntered(MouseEvent e) {

}

**public** **void** mouseExited(MouseEvent e) {

}

**public** **void** mouseDragged(MouseEvent e) {

}

**public** **void** mouseMoved(MouseEvent e) {

}

**public** **void** mousePressed(MouseEvent e) {

}

**public** **void** keyTyped(KeyEvent e) {

}

**public** **void** keyPressed(KeyEvent e) {

}

**public** **void** keyReleased(KeyEvent e) {

**if** (e.getKeyCode() == 72) { // 72 = H

JOptionPane.*showMessageDialog*(pic, "OrbGrab is a simple game. You control a group of two balls. One of the balls spins around the other one. \nWhen you press the left mouse button, the balls switch roles and the second ball starts spinning around the first. \nThe object of the game is to move the group and collect all of the cubes to raise your score. \nBut stay away from the edges of the screen! If you hit the edge you get bounced back and lose points!");

}

}

}

**package** org.br.game;

**import** java.awt.Graphics;

**import** java.util.ArrayList;

**import** java.util.List;

/\*\*

\* Abstract class which is a parent of all grouped sprites.

\*

\* **@author** Boris

\*/

**public** **abstract** **class** GroupedSprite **extends** SpriteBase {

**private** List<Sprite> group = **new** ArrayList<Sprite>();

**public** GroupedSprite(List<? **extends** Sprite> group) {

**this**.group.addAll(group);

}

**public** List<Sprite> getGroup() {

**return** group;

}

@Override

**public** **void** paint(Graphics g) {

**for** (Sprite groupMember : group) {

groupMember.paint(g);

}

}

**public** **void** init() {

**for** (Sprite groupMember : group) {

groupMember.init();

}

}

**protected** **int** getRadius() {

**return** Game.*getGame*().getRadius();

}

**public** **void** setPicture(Picture picture) {

**for** (Sprite groupMember : group) {

groupMember.setPicture(picture);

}

}

}

**package** org.br.game;

**import** java.util.Calendar;

/\*\*

\* Simple logger **TODO** - use some logging framework !

\*

\* **@author** Boris

\*/

**public** **class** Log {

**private** **static** **final** Calendar *c* = Calendar.*getInstance*();

**public** **static** **void** info(Class<?> clazz, String mess) {

System.*out*.println("INFO:" + *format*(clazz.getName(), mess));

}

**public** **static** **void** warn(Class<?> clazz, String mess) {

System.*out*.println("WARNING:" + *format*(clazz.getName(), mess));

}

**public** **static** **void** error(Class<?> clazz, Throwable ex) {

ex.printStackTrace();

System.*out*.println("ERROR:" + *format*(clazz.getName(), ex.toString()));

}

**private** **static** String format(String className, String mess) {

**return** String.*format*("%1$tm %1$te,%1$tY-[%2$s]: %3$s", *c*.getTime(), className, mess);

}

}

**package** org.br.game;

**import** java.awt.Graphics;

**import** java.awt.Image;

**import** java.awt.event.KeyEvent;

**import** java.awt.event.KeyListener;

**import** javax.swing.ImageIcon;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

/\*\*

\* Paints Main Menu and adds listeners to it.

\*

\* **@author** Boris

\*/

**public** **class** MainMenu **extends** JPanel **implements** KeyListener {

**private** Image bgimg = (**new** ImageIcon("img/title.png").getImage());

**public** MainMenu() {

setFocusable(**true**);

addKeyListener(**this**);

}

**public** **void** paint(Graphics g) {

g.drawImage(bgimg, 0, 0, getWidth(), getHeight(), **this**);

}

@Override

**public** **void** keyReleased(KeyEvent arg0) {

**if** (arg0.getKeyCode() == 10 || arg0.getKeyCode() == 32) { // 10 = Enter, 32 = space bar

Game.*GAME*.setStartGame(**true**);

Game.*GAME*.startGame();

}

**else** **if** (arg0.getKeyCode() == 72) { // 72 = H

JOptionPane.*showMessageDialog*(**this**, "OrbGrab is a simple game. You control a group of two balls. One of the balls spins around the other one. \nWhen you press the left mouse button, the balls switch roles and the second ball starts spinning around the first. \nThe object of the game is to move the group and collect all of the cubes to raise your score. \nBut stay away from the edges of the screen! If you hit the edge you get bounced back and lose points!");

}

}

@Override

**public** **void** keyPressed(KeyEvent arg0) {

}

@Override

**public** **void** keyTyped(KeyEvent arg0) {

}

}

**package** org.br.game;

**import** java.awt.Graphics;

**import** java.awt.Image;

**import** java.util.ArrayList;

**import** java.util.List;

**import** javax.swing.ImageIcon;

**import** javax.swing.JPanel;

/\*\*

\* This class handles all of the painting.

\*

\* **@author** Boris

\*/

**public** **class** Picture **extends** JPanel {

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** List<Sprite> listOfSprites = **new** ArrayList<Sprite>();

**private** Sprite centerBall, spinningBall;

**private** Image bgimg = (**new** ImageIcon("img/bg.png").getImage());

**public** Picture() {

**this**.setSize(800, 600);

**this**.setDoubleBuffered(**true**);

}

**public** List<Sprite> getListOfSprites() {

**return** listOfSprites;

}

**public** **void** setListOfSprites(List<Sprite> listOfSprites) {

**this**.listOfSprites = listOfSprites;

}

**public** **void** paint(Graphics g) {

Vertex one = centerBall.getCenter();

Vertex two = spinningBall.getCenter();

g.clearRect(0, 0, 800, 600);

g.drawImage(bgimg, 0, 0, getWidth(), getHeight(), **this**);

g.drawLine((**int**) one.getX(), (**int**) one.getY(), (**int**) two.getX(), (**int**) two.getY());

g.drawString("Score: " + Game.*getGame*().getScore(), Game.*getGame*().getWidth() - 100, 20);

**for** (Sprite sprite : getListOfSprites()) {

sprite.paint(g);

}

}

**public** Sprite getCenterBall() {

**return** centerBall;

}

**public** **void** setCenterBall(Sprite centerBall) {

**this**.centerBall = centerBall;

}

**public** Sprite getSpinningBall() {

**return** spinningBall;

}

**public** **void** setSpinningBall(Sprite spinningBall) {

**this**.spinningBall = spinningBall;

}

}

**package** org.br.game;

/\*\*

\* Basic point class. Has x and y coordinates.

\*

\* **@author** Boris

\*/

**public** **class** Point {

**private** **double** x, y;

**public** Point(**double** x, **double** y) {

**this**.x = x;

**this**.y = y;

}

**public** Point(Point p) {

**this**.x = p.getX();

**this**.y = p.getY();

}

**public** **double** getX() {

**return** x;

}

**public** **double** getY() {

**return** y;

}

/\*\*

\* Moves the point by the given coordinates.

\*

\* **@param** dx

\* **@param** dy

\*/

**public** **void** move(**double** dx, **double** dy) {

x += dx;

y += dy;

}

/\*\*

\* Scales the point up or down depending on the given values.

\*

\* **@param** sx

\* **@param** sy

\*/

**public** **void** scale(**double** sx, **double** sy) {

x = x \* sx;

y = y \* sy;

}

/\*\*

\* Turns the point in a circle based on the given angle.

\*

\* **@param** a

\*/

**public** **void** turn(**double** a) {

**double** x1 = x, y1 = y;

setX(x1 \* Math.*cos*(a \* Math.*PI* / 180) - y1 \* Math.*sin*(a \* Math.*PI* / 180));

setY(x1 \* Math.*sin*(a \* Math.*PI* / 180) + y1 \* Math.*cos*(a \* Math.*PI* / 180));

}

**public** **void** setX(**double** x) {

**this**.x = x;

}

**public** **void** setY(**double** y) {

**this**.y = y;

}

}

**package** org.br.game;

**import** java.awt.Graphics;

/\*\*

\* Sprite interface. Contains all needed methods for sprites.

\*

\* **@author** Lone Wolf

\*/

**public** **interface** Sprite {

**void** init();

**void** move(**double** x, **double** y, **double** z);

**void** turnX(**double** a, Vertex center);

**void** turnY(**double** a, Vertex center);

**void** turnZ(**double** a, Vertex center);

**void** zoom(**double** a, Vertex center);

Vertex getCenter();

**void** setActive(**boolean** flag);

**boolean** isActive();

**void** paint(Graphics g);

**void** setPicture(Picture picture);

**void** repaintAll();

}

**package** org.br.game;

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** java.util.ArrayList;

/\*\*

\* Base abstract class for all sprites.

\*

\* **@author** Boris

\*/

**public** **abstract** **class** SpriteBase **implements** Sprite {

**private** Picture picture;

**private** Face[] faces;

**private** ASEParser filereader;

**private** Vertex[] vertexes;

**private** Color color;

**private** String name;

**private** **boolean** active = **false**;

**public** SpriteBase() {

}

**public** SpriteBase(ASEParser filereader, Color color, String name) {

**this**.filereader = filereader;

**this**.color = color;

**double**[] vertexarr = filereader.readASEVertex();

**double**[] facearr = filereader.readASEFace();

faces = **new** Face[filereader.readFACENUM()];

vertexes = **new** Vertex[filereader.readVERTEXNUM()];

**for** (**int** i = 0, j = 0; i < vertexes.length; i++, j += 3) {

vertexes[i] = **new** Vertex(vertexarr[j], vertexarr[j + 1], vertexarr[j + 2]); // vertexarr[j] is x, vertexarr[j+1] is y, vertexarr[j+2] is z

}

**for** (**int** i = 0, j = 0; i < faces.length; i++, j += 3)// makes all the triangles with the points that are in the facearr array

{

faces[i] = **new** Face(vertexes[(**int**) facearr[j]], vertexes[(**int**) facearr[j + 1]], vertexes[(**int**) facearr[j + 2]], color); // instructions are in ASEParser

}

**this**.name = name;

}

@Override

**public** **void** init() {

}

/\*\*

\* Calls the move() method for all faces in a sprite.

\*/

**public** **void** move(**double** x, **double** y, **double** z) {

**for** (**int** i = 0; i < faces.length; i++) {

faces[i].move(x, y, z);

repaintAll();

}

}

/\*\*

\* Calls the turnX() method for all faces in a sprite.

\*/

**public** **void** turnX(**double** a, Vertex center) {

**this**.move(-center.getX(), -center.getY(), -center.getZ());

**for** (**int** i = 0; i < faces.length; i++) {

faces[i].turnX(a);

}

**this**.move(center.getX(), center.getY(), center.getZ());

}

/\*\*

\* Calls the turnY() method for all faces in a sprite.

\*/

**public** **void** turnY(**double** a, Vertex p) {

**this**.move(-p.getX(), -p.getY(), -p.getZ());

**for** (**int** i = 0; i < faces.length; i++) {

faces[i].turnY(a);

}

**this**.move(p.getX(), p.getY(), p.getZ());

}

/\*\*

\* Calls the turnZ() method for all faces in a sprite.

\*/

**public** **void** turnZ(**double** a, Vertex p) {

**this**.move(-p.getX(), -p.getY(), -p.getZ());

**for** (**int** i = 0; i < faces.length; i++) {

faces[i].turnZ(a);

}

**this**.move(p.getX(), p.getY(), p.getZ());

}

/\*\*

\* Calls the zoom() method for all faces in a sprite.

\*/

**public** **void** zoom(**double** a, Vertex center) {

**for** (**int** i = 0; i < faces.length; i++) {

faces[i].zoom(a, center);

}

}

/\*\*

\* For each face, checks if it should be seen using the howSeen() method, gives it shading using the setRGBColor() method and turns the faces into triangles using perspectiveProjection().

\*

\* **@return**

\*/

**public** ArrayList<Triangle> perspectiveProjection() {

ArrayList<Triangle> triangles = **new** ArrayList<Triangle>();

**for** (**int** i = 0; i < faces.length; i++) {

**double** vertex = faces[i].howSeen();

**if** (vertex > 0) {

Color color = faces[i].setRGBColor(vertex);

triangles.add(faces[i].perspectiveProjection(color));

}

}

**return** triangles;

}

**public** ASEParser getFileReader() {

**return** filereader;

}

**public** Face[] getFaces() {

**return** faces;

}

**public** Vertex[] getVertexes() {

**return** vertexes;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Color getColor() {

**return** color;

}

**public** **void** setColor(Color color) {

**this**.color = color;

}

**public** Picture getPicture() {

**return** picture;

}

**public** **void** setPicture(Picture picture) {

**this**.picture = picture;

}

**public** **void** paint(Graphics g) {

ArrayList<Triangle> triangles = perspectiveProjection();

**for** (**int** i = 0; i < triangles.size(); i++) {

triangles.get(i).paint(g);

}

}

**public** **void** repaintAll() {

getPicture().repaint();

}

/\*\*

\* Returns the center vertex of the sprite.

\*/

**public** Vertex getCenter() {

Vertex vertex = **new** Vertex(0, 0, 0);

**for** (**int** i = 0; i < getFaces().length; i++) {

vertex.move(getFaces()[i].getCenter().getX() / (getFaces().length), getFaces()[i].getCenter().getY() / (getFaces().length), getFaces()[i].getCenter().getZ() / (getFaces().length));

}

**return** vertex;

}

**public** **boolean** isActive() {

**return** active;

}

**public** **void** setActive(**boolean** active) {

**this**.active = active;

}

}

**package** org.br.game;

**import** java.awt.\*;

/\*\*

\* Triangle class made up of 3 Points.

\*

\* **@author** Boris

\*/

**public** **class** Triangle {

Point p1, p2, p3;

Color color;

**public** Triangle(Point p1, Point p2, Point p3, Color color) {

**this**.p1 = **new** Point(p1);

**this**.p2 = **new** Point(p2);

**this**.p3 = **new** Point(p3);

**this**.color = color;

}

/\*\*

\* For each point in the triangle, calls the move() method and moves all the points.

\*

\* **@param** dx

\* **@param** dy

\*/

**public** **void** move(**double** dx, **double** dy) {

p1.move(dx, dy);

p2.move(dx, dy);

p3.move(dx, dy);

}

/\*\*

\* Rotates all the points in the triangle.

\*

\* **@param** a

\*/

**public** **void** rotate(**double** a) {

p1.turn(a);

p2.turn(a);

p3.turn(a);

}

/\*\*

\* Returns the center point of the triangle.

\*

\* **@return**

\*/

**public** Point getCenter() {

Point p = **new** Point((p1.getX() + p2.getX() + p3.getX()) / 3, (p1.getY() + p2.getY() + p3.getY()));

**return** p;

}

/\*\*

\* Scales each point in the triangle.

\*

\* **@param** sx

\* **@param** sy

\*/

**public** **void** scale(**double** sx, **double** sy) {

p1.scale(sx, sy);

p2.scale(sx, sy);

p3.scale(sx, sy);

}

/\*\*

\* Zooms each point in the triangle.

\*

\* **@param** k

\*/

**public** **void** zoom(**double** k) {

**double** xc, yc;

xc = (p1.getX() + p2.getX() + p3.getX()) / 3;

yc = (p1.getY() + p2.getY() + p3.getY()) / 3;

**this**.move(-xc, -yc);

**this**.scale(k, k);

**this**.move(xc, yc);

}

/\*\*

\* Turns each point in the triangle.

\*

\* **@param** a

\*/

**public** **void** turn(**double** a) {

**double** xc, yc;

xc = (p1.getX() + p2.getX() + p3.getX()) / 3;

yc = (p1.getY() + p2.getY() + p3.getY()) / 3;

**this**.move(-xc, -yc);

**this**.rotate(a);

**this**.move(xc, yc);

}

/\*\*

\* Turns each point in the triangle.

\*

\* **@param** a

\* **@param** center

\*/

**public** **void** turn(**double** a, Point center) {

move(-center.getX(), -center.getY());

rotate(a);

move(center.getX(), center.getY());

}

**public** **void** paint(Graphics g) {

g.setColor(color);

**int**[] x = **new** **int**[3];

**int**[] y = **new** **int**[3];

x[0] = (**int**) p1.getX();

x[1] = (**int**) p2.getX();

x[2] = (**int**) p3.getX();

y[0] = (**int**) p1.getY();

y[1] = (**int**) p2.getY();

y[2] = (**int**) p3.getY();

g.fillPolygon(x, y, 3);

}

}

**package** org.br.game;

/\*\*

\* Same as Point class, but has 3 coordinates: X, Y, and Z.

\*

\* **@author** Boris

\*/

**public** **class** Vertex {

**double** x, y, z;

**public** **final** **static** **int** DIST = 3000; // Used for perspective projection.

**private** **final** **int** dif = 30; // The max difference used in the collision detection.

**public** Vertex(**double** x, **double** y, **double** z) {

**this**.x = x;

**this**.y = y;

**this**.z = z;

}

**public** Vertex(Vertex p) {

**this**.x = p.getX();

**this**.y = p.getY();

**this**.z = p.getZ();

}

/\*\*

\* Moves all the coordinates by the given values.

\*

\* **@param** dx

\* **@param** dy

\* **@param** dz

\*/

**public** **void** move(**double** dx, **double** dy, **double** dz) {

x += dx;

y += dy;

z += dz;

}

/\*\*

\* Scales all the coordinates by the given values.

\*

\* **@param** sx

\* **@param** sy

\* **@param** sz

\*/

**public** **void** scale(**double** sx, **double** sy, **double** sz) {

x \*= sx;

y \*= sy;

z \*= sz;

}

/\*\*

\* Gets the length of the vertex.

\*

\* **@return**

\*/

**public** **double** getLength() {

**return** Math.sqrt((x \* x) + (y \* y) + (z \* z));

}

/\*\*

\* Turns the vertex around the X axis.

\*

\* **@param** a

\*/

**public** **void** turnX(**double** a) {

**double** y1 = y, z1 = z;

y = y1 \* Math.cos(a \* Math.PI / 180) - z1 \* Math.sin(a \* Math.PI / 180);

z = y1 \* Math.sin(a \* Math.PI / 180) + z1 \* Math.cos(a \* Math.PI / 180);

}

/\*\*

\* Turns the vertex around the Y axis.

\*

\* **@param** a

\*/

**public** **void** turnY(**double** a) {

**double** x1 = x, z1 = z;

x = x1 \* Math.cos(a \* Math.PI / 180) - z1 \* Math.sin(a \* Math.PI / 180);

z = x1 \* Math.sin(a \* Math.PI / 180) + z1 \* Math.cos(a \* Math.PI / 180);

}

/\*\*

\* Turns the vertex around the Z axis.

\*

\* **@param** a

\*/

**public** **void** turnZ(**double** a) {

**double** x1 = x, y1 = y;

x = x1 \* Math.cos(a \* Math.PI / 180) - y1 \* Math.sin(a \* Math.PI / 180);

y = x1 \* Math.sin(a \* Math.PI / 180) + y1 \* Math.cos(a \* Math.PI / 180);

}

/\*\*

\* Turns a 3D vertex into a 2D point.

\*

\* **@return**

\*/

**public** Point perspectiveProjection() {

Point p2d = **new** Point((getX() \* DIST) / (getZ() + DIST), (getY() \* DIST) / (getZ() + DIST));

**return** p2d;

}

**public** **void** setX(**double** x) {

**this**.x = x;

}

**public** **void** setZ(**double** z) {

**this**.z = z;

}

**public** **double** getZ() {

**return** z;

}

**public** **void** setY(**double** y) {

**this**.y = y;

}

**public** **double** getX() {

**return** x;

}

**public** **double** getY() {

**return** y;

}

/\*\*

\* Used for collision detection. Checks if the vertex is equal to another vertex (centers of sprites) plus or minus the dif amount.

\*/

@Override

**public** **boolean** equals(Object other) {

**boolean** eq = **false**;

**if** (other **instanceof** Vertex) {

Vertex otherVertex = (Vertex) other;

**if** (Math.abs(otherVertex.getX() - **this**.getX()) <= dif)

**if** (Math.abs(otherVertex.getY() - **this**.getY()) <= dif)

eq = **true**;

}

**return** eq;

}

**public** String toString() {

**return** **new** StringBuilder(getClass().getName()).append(" X:").append(getX()).append(" Y:").append(getY()).append(" Z:").append(getZ()).toString();

}

}

**package** org.br.game.sprites;

**import** java.awt.Color;

**import** org.br.game.ASEParser;

**import** org.br.game.Game;

**import** org.br.game.Sprite;

**import** org.br.game.SpriteBase;

**import** org.br.game.Vertex;

/\*\*

\* The Ball sprite.

\*

\* **@author** Boris

\*/

**public** **class** Ball **extends** SpriteBase {

**private** **long** sleep = Game.*getGame*().getSleepTime();

**private** Ball centerBall;

**private** Thread movingThread;

**private** **boolean** circling = **false**;

**private** CirclingBallGroup owner;

**public** Ball(ASEParser filereader, Color color, String name) {

**super**(filereader, color, name);

}

/\*\*

\* Sets the group for each ball so it's aware of the other ball.

\*

\* **@param** owner

\*/

**public** **void** setGroup(CirclingBallGroup owner) {

**this**.owner = owner;

}

**private** CirclingBallGroup getGroup() {

**return** owner;

}

/\*\*

\* Checks if ball has a center ball, then checks if it's active (spinning) and calls the circleAround() function, if not, stops the thread, and if there's no center ball, do nothing.

\*/

@Override

**public** **void** init() {

**super**.init();

**if** (centerBall != **null**) {

**if** (isActive()) {

centerBall.setActive(**false**);// Deactivates the center ball

circleAround();

}

**else** {

stop();

}

}

**else** {

}

}

/\*\*

\* A thread that's used for the active ball. Inside the loop it keeps calling the spin() method which updates its coordinates.

\*/

**void** circleAround() {

movingThread = **new** Thread() {

**int** inc = 0;

**public** **void** run() {

**while** (circling) {

spin(inc);

Sprite targetDetected = Game.*getGame*().getTargets().collisionDetected(Ball.**this**);

**if** (targetDetected != **null**) {

Game.*getGame*().hit(targetDetected);

}

**try** {

Thread.*sleep*(sleep);// Sleeps for 0.1 seconds

inc += 1;

}

**catch** (InterruptedException e) {

}

}

}

};

movingThread.start();

}

/\*\*

\* Updates the spinning ball's coordinates

\*

\* **@param** inc

\*/

**private** **void** spin(**int** inc) {

**double** newX = getRadius() \* Math.*cos*(Math.*toDegrees*(inc));

**double** newY = getRadius() \* Math.*sin*(Math.*toDegrees*(inc));

move(newX, newY, 0);

}

/\*\*

\* Stop moving

\*/

**void** stop() {

circling = **false**;

setActive(**false**);

**if** (movingThread != **null**) {

movingThread.interrupt();

}

}

/\*\*

\* Returns the center (non-spinning) ball.

\*

\* **@return**

\*/

**public** Ball getCenterBall() {

**return** centerBall;

}

**public** **void** setCenterBall(Ball centerBall) {

**this**.centerBall = centerBall;

}

**public** **void** setActive(**boolean** active) {

**super**.setActive(active);

circling = active;

}

**public** String toString() {

**return** "CirclingBall: " + getName() + " circling:" + isActive();

}

**private** **int** getRadius() {

**return** Game.*getGame*().getRadius();

}

/\*\*

\* If the ball hits the edge of the screen, bounce it back 200 pixels.

\*/

**public** **void** move(**double** x, **double** y, **double** z) {

Vertex center = getCenter();

**if** (center.getX() + x < 0) {

getGroup().move(200, 0, 0);

getGroup().getCirclingBall().move(-getRadius() \* Math.*cos*(45), -getRadius() \* Math.*sin*(45), 0);

Game.*getGame*().outOfBorder();

}

**if** (center.getX() + x > Game.*getGame*().getWidth()) {

getGroup().move(-200, 0, 0);

getGroup().getCirclingBall().move(-getRadius() \* Math.*cos*(45), -getRadius() \* Math.*sin*(45), 0);

Game.*getGame*().outOfBorder();

}

**if** (center.getY() + y < 0) {

getGroup().move(0, 200, 0);

getGroup().getCirclingBall().move(-getRadius() \* Math.*cos*(45), -getRadius() \* Math.*sin*(45), 0);

Game.*getGame*().outOfBorder();

}

**if** (center.getY() + y > Game.*getGame*().getHeight()) {

getGroup().move(0, -200, 0);

getGroup().getCirclingBall().move(-getRadius() \* Math.*cos*(45), -getRadius() \* Math.*sin*(45), 0);

Game.*getGame*().outOfBorder();

}

**super**.move(x, y, z);

}

}

**package** org.br.game.sprites;

**import** java.util.List;

**import** org.br.game.GroupedSprite;

**import** org.br.game.Sprite;

**import** org.br.game.Vertex;

/\*\*

\* The concrete implementation of GroupedSprite with the group of 2 moving ball sprites. One of the sprites is still, the second circles around it.

\*

\* **@author** Boris

\*/

**public** **class** CirclingBallGroup **extends** GroupedSprite {

**public** CirclingBallGroup(List<Sprite> sprites) {

**super**(sprites);

}

@Override

**public** **void** setActive(**boolean** flag) {

// nothing

}

/\*\*

\* Switches between the active sprites.

\*/

**public** **void** switchSpinning() {

Ball centerBall = getCenterBall();

Ball circlingBall = getCirclingBall();

circlingBall.stop();

centerBall.setActive(**true**);

circlingBall.setCenterBall(**null**);

centerBall.setCenterBall(circlingBall);// set Center ball as ball1

**double**[] delta = getDeltaPos(circlingBall, centerBall);

centerBall.move(delta[0], delta[1], delta[2]); // Moves the (previously) center ball on top of the (previously) circling ball.

centerBall.move(-getRadius() \* Math.*cos*(45), -getRadius() \* Math.*sin*(45), 0);// Moves the (previously) center ball to the starting point on the orbit.

init();

}

/\*\*

\* We need the delta position of the sprites in order to move the sprites correctly.

\*

\* **@param** ball1

\* **@param** ball2

\* **@return**

\*/

**private** **double**[] getDeltaPos(Ball ball1, Ball ball2) {

Vertex ball1Pos = ball1.getCenter();

Vertex ball2Pos = ball2.getCenter();

**double** dX = ball1Pos.getX() - ball2Pos.getX();

**double** dY = ball1Pos.getY() - ball2Pos.getY();

**double** dZ = ball1Pos.getZ() - ball2Pos.getZ();

**return** **new** **double**[] { dX, dY, dZ };

}

@Override

**public** **boolean** isActive() {

**return** **false**;

}

/\*\*

\* Returns the stationary ball.

\*

\* **@return**

\*/

**public** Ball getCenterBall() {

Ball center = **null**;

**for** (Sprite sprite : getGroup()) {

**if** (((Ball) sprite).getCenterBall() == **null**) {

center = (Ball) sprite;

**break**;

}

}

**return** center;

}

/\*\*

\* Returns the circling ball.

\*

\* **@return**

\*/

**public** Ball getCirclingBall() {

Ball circling = **null**;

**for** (Sprite sprite : getGroup()) {

**if** (((Ball) sprite).getCenterBall() != **null**) {

circling = (Ball) sprite;

**break**;

}

}

**return** circling;

}

@Override

**public** **void** move(**double** x, **double** y, **double** z) {

**for** (Sprite sprite : getGroup()) {

sprite.move(x, y, z);

}

}

@Override

**public** **void** zoom(**double** a, Vertex center) {

**for** (Sprite sprite : getGroup()) {

sprite.zoom(a, center);

}

}

@Override

**public** **void** turnX(**double** a, Vertex center) {

**for** (Sprite sprite : getGroup()) {

sprite.turnX(a, center);

}

}

@Override

**public** **void** turnY(**double** a, Vertex center) {

**for** (Sprite sprite : getGroup()) {

sprite.turnY(a, center);

}

}

@Override

**public** **void** turnZ(**double** a, Vertex center) {

**for** (Sprite sprite : getGroup()) {

sprite.turnZ(a, center);

}

}

}

**package** org.br.game.sprites;

**import** java.awt.Color;

**import** org.br.game.ASEParser;

**import** org.br.game.Game;

**import** org.br.game.SpriteBase;

**import** org.br.game.Vertex;

/\*\*

\* The target class.

\*

\* **@author** Lone Wolf

\*/

**public** **class** Target **extends** SpriteBase {

**public** Target(ASEParser filereader, Color color, String name) {

**super**(filereader, color, name);

}

/\*\*

\* Basic thread to animate rotation of the cube.

\*/

**public** **void** rotate() {

Thread thread = **new** Thread() {

**public** **void** run() {

**while** (**true**) {

**try** {

Thread.*sleep*(100L);

turnX(5, getCenter());

turnY(5, getCenter());

turnZ(5, getCenter());

}

**catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

};

thread.start();

}

/\*\*

\* Checks if the cube went out of bounds of the screen and if it did, move it back.

\*/

**public** **void** move(**double** x, **double** y, **double** z) {

Vertex center = getCenter();

**if** (center.getX() + x < 0) {

x = -x;

}

**if** (center.getX() + x > Game.*getGame*().getWidth()) {

x = -x;

}

**if** (center.getY() + y < 0) {

y = -y;

}

**if** (center.getY() + y > Game.*getGame*().getHeight()) {

y = -y;

}

**super**.move(x, y, z);}}

**package** org.br.game.sprites;

**import** java.util.List;

**import** org.br.game.GroupedSprite;

**import** org.br.game.Picture;

**import** org.br.game.Sprite;

**import** org.br.game.Vertex;

/\*\*

\* The concrete implementation of GroupedSprite with the group of target sprites.

\*

\* **@author** Boris

\*/

**public** **class** TargetGroup **extends** GroupedSprite {

**public** TargetGroup(List<Sprite> sprites) {

**super**(sprites);

}

@Override

**public** **void** move(**double** x, **double** y, **double** z) {

**for** (Sprite sprite : getGroup()) {

sprite.move(x, y, z);

}

}

@Override

**public** **void** turnX(**double** a, Vertex center) {

**for** (Sprite sprite : getGroup()) {

sprite.turnX(a, center);

}

}

@Override

**public** **void** turnY(**double** a, Vertex center) {

**for** (Sprite sprite : getGroup()) {

sprite.turnY(a, center);

}

}

@Override

**public** **void** turnZ(**double** a, Vertex center) {

**for** (Sprite sprite : getGroup()) {

sprite.turnZ(a, center);

}

}

/\*\*

\* Used for collision detection. Returns the collided-with sprite (target). "Other" is the spinning ball in this case.

\*

\* **@param** other

\* **@return**

\*/

**public** Sprite collisionDetected(Sprite other) {

Sprite detected = **null**;

**for** (Sprite sprite : getGroup()) {

**if** (sprite.getCenter().equals(other.getCenter())) {

detected = sprite;

**break**;

}

}

**return** detected;

}

@Override

**public** **void** zoom(**double** a, Vertex center) {

**for** (Sprite sprite : getGroup()) {

sprite.zoom(a, center);

}

}

@Override

**public** Vertex getCenter() {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**public** **void** setActive(**boolean** flag) {

// **TODO** Auto-generated method stub

}

@Override

**public** **boolean** isActive() {

// **TODO** Auto-generated method stub

**return** **false**;

}

@Override

**public** **void** repaintAll() {

// **TODO** Auto-generated method stub

}

**public** **void** setPicture(Picture picture) {

**for** (Sprite groupMember : getGroup()) {

groupMember.setPicture(picture);

}

}

}

Sources

<http://members.tripod.com/~Paul_Kirby/vector/Vplane.html>

<http://www.codeproject.com/KB/cs/abstractsvsinterfaces.aspx>

<http://www.oracle.com/>