inda13 - Projekt
Javaga

Gustav Dänsel
Lukas Lundmark

15 maj 2014

1 Programbeskrivning

Vi siktar på att skapa en Galaga-klon i Java. Det ska alltså vara en top-down 2D arkadspel. Vi planerar att använda oss av libGDX-biblioteket.

Spelet fungerar så att man kontrollerar ett rymdskepp som ska skjuta ned utomjordingar som kommer från fönstrets övre kant och försöker röra sig nedåt. Målet är att förstöra dem innan de försvinner ut skärmen. Se http://en.wikipedia.org/wiki/Galaga

2 Användarbeskrivning

Vi tänker oss att personer som är sugna på att spela klassiska retrospel utan att köpa en arkadmaskin kan vara en potentiell målgrupp. Annars ingen, tyvärr.

3 Användarscenarie

3.1 Scenarie 1

Sent på kvällen, dagen innan tentamen för SF1626, pluggångesten liger tungt över Kalle. Han känner att han behöver ta en paus från att inte plugga och göra något annat och får ett tips om den supercoola Galagaklonen Javaga. Han laddar ned spelet från thepiratebay och börjar prokrastinera hårt. Han stry sin rymdhjälte med piltangenterna och skriker avfyra! med mellanslag. Vilken upplevelse det är! Han har aldrig varit med om något liknande! Kalle sitter uppe och spelar hela natten och missar tyvärr tentan, mycket sorgligt.

Based on a true story.

3.2 Scenarie 2

Pappa Per är på väg hem från en lång och slitsam dag på jobbet. På vägen hem ser han reklamen för Game On 2.0 på Tekniska Museet. Han kommer och tänka på alla fantastiska kvällar med sina vänner i arkadhallen när han var liten. Han tänker på Galaga och hur kul det var att spela. När han väl kommer hem så bingar han fram en lista på Galagakloner och den som ligger högst upp och har bäst omdömen är ju givetvis Javaga. Per laddar ned spelet och börjar tidsresan till barndomen. Och vilken resa det är! Sicken resa, Mycket fräck! Han känner sina mossiga gamla färdigheter komma tillbaks och upplever sann eufori. Fantastiskt.

4 Testplan

Vi planerar att muta folk till att spela spelet och ge oss bra feedback. Vi planerar att göra detta vid minst två tillfällen, kanske mer om det finns tid. En viss mängd automatiserade tester kan förekomma.

Testanvändaren förväntas spela spelet tills ögonen blöder och därefter berätta för oss hur fantisktiskt det var.

5 Programdesign

Tanken är att dela upp programmet i flera klasser. Primärt så tänker vi oss att vi har dessa klasser:

- Input
- Render
- Game-Logic
- Units
- File I/O

5.1 Input

Här finns metoder för att läsa indata från kontroller såsom tangentbord och mus, samt skicka vidare dessa till relevanta klasser.

5.2 Render

Här görs det tunga jobbet att rita bilden som ska visas på skärmen.

5.3 Game-Logic

I denna klass sköts all logik - såsom hur enheter ska förflytta sig, om skott träffar eller inte och liknande saker.

5.4 Units

Units innehåller beskrivningar och parametrar för alla olika typer av ssskeppsom finns i spelet.

5.5 File I/O

För att läsa och skriva till hårddisk, för t.ex. inställningsfiler.

6 Tekniska frågor

Den största tekniska frågan ligger i hur vi implementerar biblioteket. I det problemer finns det även fler mindre problem, såsom vilken typ av input vi ska använda o.s.v.

Klassiska problem såsom animation sköter det bibliotek (libGDX) vi använder.

7 arbetsplan

7.1 Tidsplan

- Läsa och sätta sig in i biblioteket och dess dokumentiation - 2/5
- Första fungerande prototyp - 9/5
- Testning under helgen 9/5 till 12/5
- Finslipning mer test till 16/5

Planen är att använda GitHub för att samarbeta på koden. Vi tänker oss att vi delar lika på arbetsuppgifterna och arbetar tillsammans. Då projektet är relativt litet så blir det enklare så då vi båda har koll på all kod och vi slipper läsa ikapp.

8 Programkod

Javaga/core/src/com/me/Javaga/JavagaMain.java

```
1  package com.me.Javaga;
2
3  import com.badlogic.gdx.ApplicationAdapter;
4  import com.badlogic.gdx.Gdx;
5  import com.badlogic.gdx.graphics.GL20;
6  import com.badlogic.gdx.graphics.OrthographicCamera;
7  import com.badlogic.gdx.graphics.Texture;
8  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
9  import com.me.Javaga.managers.GameInputProcessor;
10 import com.me.Javaga.managers.GameKeys;
11 import com.me.Javaga.managers.GameStateManager;
12
13 import java.util.Random;
14
15 public class JavagaMain extends ApplicationAdapter {
16     private SpriteBatch batch;
17     private Texture img;
18     private OrthographicCamera camera;
19     private float WIDTH;
20     private float HEIGHT;
21     private Random rand;
22     private GameStateManager manager;
23
24     @Override
25     public void create() {
26         batch = new SpriteBatch();
27         img = new Texture("badlogic.jpg");
28         rand = new Random();
29
30         //Set Res
31         WIDTH = Gdx.graphics.getWidth();
32         HEIGHT = Gdx.graphics.getHeight();
33
34         //Initiate camera + window.
35         camera = new OrthographicCamera(WIDTH, HEIGHT);
36         camera.translate(WIDTH / 2, HEIGHT / 2);
37         camera.update();
38
39         //Initiate managers.
40         manager = new GameStateManager();
41
42         //Select input processor to our custom one.
43         Gdx.input.setInputProcessor(new GameInputProcessor());
44     }
45
46     @Override
47     public void render() {
48
49         //Draw a black screen.
50         Gdx.gl.glClearColor(0, 0, 0, 1);
```

```
51          Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
52
53          //Update the game state
54          manager.update();
55          GameKeys.update();
56          camera.update();
57
58          //Tell the game manager to initiate drawing of sprites
                and other elements
59          batch.begin();
60          manager.draw(batch);
61          batch.end();
62      }
63  }
```

Javaga/core/src/com/me/Javaga/spaceobject/Boss.java

```
1   package com.me.Javaga.spaceobject;
2
3   import com.me.Javaga.gamestate.levels.EnemyDescription;
4   import com.me.Javaga.managers.GameStateManager;
5
6   import java.util.ArrayList;
7
8   /**
9    * A boss class, a subclass of enemies which generally are
         stronger and
10   * have a little more edge to them
11   * Created by Lukas on 2014-05-13.
12   */
13  public class Boss extends Enemy {
14
15      public Boss(float xPos, float yPos, EnemyDescription
            description,
16                  ArrayList<Bullet> enemyBullets, Player player) {
17          super(xPos, yPos, description, enemyBullets, player);
18      }
19
20      /**
21       * Update the current goals, unlike the enemy,
22       * the boss will start over if it is out of new goals
23       */
24      @Override
25      protected void updateGoal() {
26          if (goalIndex + 1 < goals.size()) {
27              goalIndex++;
28              currentGoal = goals.get(goalIndex);
29          } else {
30              goalIndex = 0;
31              currentGoal = goals.get(goalIndex);
32          }
33      }
34
35      /**
```

```
36      * Fire a bullet att the pllayer, unlike the enemy,
37      * the boss can fire in all directions
38      */
39     @Override
40     public void fire() {
41         if (System.currentTimeMillis() - time < shootLimit ||
               outsideBorder) {
42             return;
43         }
44         float dX = xCenter - player.getX(); // Aim for the
               player
45         float dY = (yCenter - sHeight / 2) - player.getY(); //
               Aim for the player
46         float startDegree = 270;
47         if ((yCenter - sHeight / 2) - player.getY() < 0) { //
               Dont shoot if the player is behind you
48             startDegree = 90;
49         }
50         double radian = Math.atan(dX / dY);
51         float degree = (float) (startDegree - Math.toDegrees(
               radian));
52         float miss = (random.nextBoolean()) ? random.nextFloat()
                * description.getAccuracy()
53                 : random.nextFloat() * -1 * description.
                       getAccuracy(); // Makes their aim awful,
54         //// probably should do it some other this
55         Bullet bullet;
56         if (description.getBulletType().isMotionSeeker()) {
57             bullet = new MotionSeeker(xCenter, yCenter - sHeight
                    / 2, degree + miss, description.getBulletType(),
                    player);
58         } else {
59             bullet = new Bullet(xCenter, yCenter - sHeight / 2,
                   degree + miss, description.getBulletType());
60         }
61
62         enemyBullets.add(bullet);
63         sound.play(GameStateManager.getEffectVolume()); // play
               lazer
64         time = System.currentTimeMillis(); // reset time
65
66     }
67 }
```

Javaga/core/src/com/me/Javaga/spaceobject/Bullet.java

```
1  package com.me.Javaga.spaceobject;
2
3  import com.badlogic.gdx.Gdx;
4  import com.me.Javaga.gamestate.levels.BulletDescription;
5
6  import java.util.ArrayList;
7
8  /**
```

```
 * For projectiles
 * Created by Dansel on 2014-04-30.
 */
public class Bullet extends SpaceObject {


    //private final static float ROTATION = 30;
    protected BulletDescription description;
    protected long startTime;

    /**
     * Create a bullet
     *
     * @param xPos          The start x coordinate
     * @param yPos          The start y coordinate
     * @param degree        The degree which the bullet should be
         fired in, 90 for straigth, 180 to the right, 270 for
         down etc.
     * @param description A BulletDescription object which
         specifies all characteristics of the bullet
     */
    public Bullet(float xPos, float yPos, float degree,
        BulletDescription description) {
        super(xPos, yPos);
        this.description = description;
        dX = (float) Math.cos(Math.toRadians(degree)) *
            description.getSpeed();
        dY = (float) Math.sin(Math.toRadians(degree)) *
            description.getSpeed();
        HEIGHT = Gdx.graphics.getHeight();
        WIDTH = Gdx.graphics.getWidth();
        init();
        sprite.rotate(degree - 90);
        startTime = System.currentTimeMillis();
    }

    /**
     * Initialize all unitilized fields
     */
    @Override
    public void init() {
        setScale(description.getScale());
        spriteSetUp(description.getFilename());
    }

    /**
     * Update the bullet
     */
    @Override
    public void update() {
        if (isHealthy) {
            yPos += dY;
            xPos += dX;
            xCenter = xPos + sprite.getWidth() / 2;
```

```
57              yCenter = yPos + sprite.getHeight() / 2;
58              //sprite.rotate(ROTATION);
59              sprite.setX(xPos);
60              sprite.setY(yPos);
61              hitbox.setCenter(xCenter, yCenter);
62              wrap();
63              if (System.currentTimeMillis() - startTime >
                    description.getLifeTime()) {
64                  isHealthy = false;
65              }
66          } else {
67              hurt();
68          }
69      }
70
71      /**
72       * Start flashing if the bullet is damaged and will soon be
             removed
73       */
74      @Override
75      protected void hurt() {
76          if (disposeIndex > 50) {
77              isDisposable = true;
78          } else {
79              disposeIndex++;
80              if (disposeIndex % 10 == 0) {
81                  draw = (draw) ? false : true;
82              }
83          }
84      }
85
86      /**
87       * Dispose of the bullet if it escapes the games boundaries
88       */
89      @Override
90      public void wrap() {
91          if ((xCenter - sWidth / 2 + 100 < 0) || (xCenter +
                sWidth / 2 - 100 > WIDTH)
92                  || (yCenter - sHeight / 2 + 20 < 0) || (yCenter
                        + sHeight / 2 - 20 > HEIGHT)) {
93              isHealthy = false;
94              isDisposable = true;
95          }
96      }
97
98      /**
99       * Get the damage which the bullet causes
100      *
101      * @return the damage which the bullet deals
102      */
103     public float getDamage() {
104         return description.getDamage();
105     }
106
```

```
107     /**
108      * Check if the bullet is indesctrible and doesn't get
             destroyed after it killed something
109      *
110      * @return True if the bullet doesn't get destroyed on
             impact, false if it does
111      */
112     public boolean isIndestructable() {
113         return description.isIndestructable();
114     }
115
116     /**
117      * Check if the bullet collides with other bullets
118      *
119      * @param bullets An arrayList of bullets
120      * @return true if the colide, false if the don't
121      */
122     @Override
123     public boolean checkForCollision(ArrayList<Bullet> bullets)
             {
124         return false;
125     }
126 }
```

Javaga/core/src/com/me/Javaga/spaceobject/Enemy.java

```
1  package com.me.Javaga.spaceobject;
2
3  import com.badlogic.gdx.Gdx;
4  import com.badlogic.gdx.audio.Sound;
5  import com.badlogic.gdx.math.Rectangle;
6  import com.badlogic.gdx.math.Vector2;
7  import com.me.Javaga.gamestate.levels.BulletDescription;
8  import com.me.Javaga.gamestate.levels.EnemyDescription;
9  import com.me.Javaga.managers.GameStateManager;
10 import com.me.Javaga.managers.InformationDrawer;
11
12 import java.util.ArrayList;
13 import java.util.Iterator;
14 import java.util.Random;
15
16 ;
17
18 /**
19  * Create object of this class to spawn enemies.
20  * Created by Dansel on 2014-05-02.
21  */
22 public class Enemy extends SpaceObject {
23     protected ArrayList<Bullet> enemyBullets;
24     protected ArrayList<Vector2> goals;
25     protected Vector2 direction;
26     protected Vector2 currentGoal;
27     protected Sound sound;
28     protected Player player;
```

```
29      protected long time;
30      protected float shootLimit;
31      protected int goalIndex;
32      protected Random random;
33      protected boolean outsideBorder;
34      protected EnemyDescription description;
35      protected Rectangle directionBox;
36
37      public Enemy(float xPos, float yPos, EnemyDescription
            description,
38                   ArrayList<Bullet> enemyBullets, Player player)
                         {
39          super(xPos, yPos);
40
41          HEIGHT = Gdx.graphics.getHeight();
42          WIDTH = Gdx.graphics.getWidth();
43          this.description = description;
44          health = description.getHealth();
45          this.enemyBullets = enemyBullets;
46          this.player = player;
47          this.time = System.currentTimeMillis();
48          init();
49      }
50
51      @Override
52      public void init() {
53          random = new Random();
54          setScale(description.getScale());
55          spriteSetUp(description.getFilename());
56
57          hitbox.setWidth(sWidth * description.getHitBoxScale()).
58                  setHeight(sHeight * description.getHitBoxScale()
                        );
59          sprite.rotate(180);
60          sound = Gdx.audio.newSound(Gdx.files.internal("lazer.mp3
                "));
61          shootLimit = description.getBulletType().getShootLimit()
                ;
62          goals = new ArrayList<Vector2>();
63          directionBox = new Rectangle();
64          directionBox.setWidth(4).setHeight(4).setCenter(xCenter,
                 yCenter);
65          wrap();
66      }
67
68      @Override
69      public void update() {
70          if (isHealthy) {
71              xPos += direction.x;
72              yPos += direction.y;
73              //System.out.println("y: " + yPos);
74              //System.out.println("x: " + xPos);
75
76              sprite.setX(xPos);
```

```
77              sprite.setY(yPos);
78              xCenter = xPos + sprite.getWidth() / 2;
79              yCenter = yPos + sprite.getHeight() / 2;
80              wrap();
81              hitbox.setCenter(xCenter, yCenter);
82              directionBox.setCenter(xCenter, yCenter);
83              if (currentGoal != null && directionBox.contains(
                    currentGoal.x, currentGoal.y)) {
84                  direction.set(0, 0);
85                  updateGoal();
86              } else {
87                  updateDirection();
88              }
89              if (health <= 0) {
90                  isHealthy = false;
91              }
92              fire();
93          } else {
94              hurt();
95          }
96      }
97
98      @Override
99      protected void hurt() {
100         if (disposeIndex > 50) {
101             isDisposable = true;
102         } else {
103             disposeIndex++;
104             if (disposeIndex % 10 == 0) {
105                 draw = (draw) ? false : true;
106             }
107         }
108     }
109
110     @Override
111
112     /**
113      * Check if the enemy is outside the game or not
114      */
115     public void wrap() {
116         if (xCenter > WIDTH || xCenter < 0 || yCenter > HEIGHT
                || yCenter < 0) {
117             outsideBorder = true;
118         } else {
119             outsideBorder = false;
120         }
121     }
122
123     /**
124      * Check if any bullet hits the enemy and deals the
            appropriate damage
125      *
126      * @param bullets An arraylist of bullets
127      * @return true if there was a collision
```

```
128         */
129         @Override
130         public boolean checkForCollision(ArrayList<Bullet> bullets)
                {
131             Iterator<Bullet> iterator = bullets.iterator();
132             while (iterator.hasNext()) {
133                 Bullet bullet = iterator.next();
134                 if (overlap(bullet)) {
135                     health -= bullet.getDamage();
136                     if (health <= 0) {
137                         InformationDrawer.updatePoints(10);
138                         isHealthy = false;
139                     }
140                     if (!bullet.isIndestructable() || isHealthy) {
141                         bullet.dispose();
142                         iterator.remove();
143                     }
144                     return true;
145                 }
146             }
147             return false;
148         }
149 
150         /**
151          * Fire a shoot
152          */
153         public void fire() {
154             if (System.currentTimeMillis() - time < shootLimit ||
                    outsideBorder) {
155                 return;
156             }
157             float dX = xCenter - player.getX(); // Aim for the
                    player
158             float dY = (yCenter - sWidth / 2) - player.getY(); //
                    Aim for the player
159 
160             if ((yCenter - sWidth / 2) - player.getY() >= 0) { //
                    Dont shoot if the player is behind you
161 
162                 double radian = Math.atan(dX / dY);
163                 float degree = (float) (270 - Math.toDegrees(radian)
                        );
164                 float miss = (random.nextBoolean()) ? random.
                        nextFloat() * description.getAccuracy()
165                         : random.nextFloat() * -1 * description.
                            getAccuracy(); // Makes their aim awful,
166                 //// probably should do it some other this
167 
168                 Bullet bullet = BulletDescription.spawnBullet(
                        xCenter, yCenter - sHeight / 2,
169                         degree + miss, description.getBulletType(),
                            player);
170                 enemyBullets.add(bullet);
171 
```

```
172             sound.play(GameStateManager.getEffectVolume()); // play lazer
173             time = System.currentTimeMillis(); // reset time
174         }
175
176     }
177
178     /**
179      * Push the enemey object in a certain direction
180      *
181      * @param speed  The speed of the direction, or the length of the direction vector
182      * @param degree The degree of the direction
183      */
184     public void setDirection(float speed, float degree) {
185         direction = new Vector2((float) Math.cos(Math.toRadians(degree) * speed),
186                 (float) Math.sin(Math.toRadians(degree)) * speed);
187     }
188
189     /**
190      * Add a new goal to the enemy's list of goals
191      *
192      * @param x x coordinate of the goal
193      * @param y y coordinate of the goal
194      */
195     public void addNewGoal(float x, float y) {
196         goals.add(new Vector2(x, y));
197     }
198
199     /**
200      * Remove the current goal and add new one
201      *
202      * @param x x coordinate of the goal
203      * @param y y coordinate of the goal
204      */
205     public void setCurrentGoal(float x, float y) {
206         goals.remove(goalIndex);
207         Vector2 newGoal = new Vector2(x, y);
208         goals.add(goalIndex, newGoal);
209         currentGoal = newGoal;
210     }
211
212     /**
213      * Update the direction for the enemy object
214      */
215     protected void updateDirection() {
216         if (currentGoal != null) {
217             Vector2 newDirection = new Vector2(currentGoal.x - xCenter, currentGoal.y - yCenter);
218             direction.add(newDirection.nor().scl(description.getSpeed()));
219             //direction.nor().scl(description.getSpeed());
```

```
220             direction.nor().scl(description.getSpeed());
221
222         } else {
223             if (!goals.isEmpty()) {
224                 currentGoal = goals.get(0);
225             }
226         }
227     }
228
229     /**
230      * Update the current goal
231      */
232     protected void updateGoal() {
233         if (goalIndex + 1 < goals.size()) {
234             goalIndex++;
235             currentGoal = goals.get(goalIndex);
236         } else {
237             isHealthy = false;
238         }
239     }
240
241     /**
242      * Dispose all components in the object, no other methods
             can be called after this
243      */
244     @Override
245     public void dispose() {
246         super.dispose();
247         sound.dispose();
248     }
249 }
```

Javaga/core/src/com/me/Javaga/spaceobject/MotionSeeker.java

```
1  package com.me.Javaga.spaceobject;
2
3  import com.badlogic.gdx.math.Vector2;
4  import com.me.Javaga.gamestate.levels.BulletDescription;
5
6  /**
7   * A subclass of bullets which follows a target
8   * Created by Lukas on 2014-05-13.
9   */
10 public class MotionSeeker extends Bullet {
11
12     private SpaceObject target;
13     private Vector2 direction;
14     private float currentDegree;
15
16     public MotionSeeker(float xPos, float yPos, float degree,
17                         BulletDescription description,
                                SpaceObject target) {
18         super(xPos, yPos, degree, description);
19         this.target = target;
```

```
20          direction = new Vector2(dX, dY);
21          currentDegree = degree;
22      }
23
24      /**
25       * Update the bullet
26       */
27      @Override
28      public void update() {
29          xPos += direction.x;
30          yPos += direction.y;
31          //System.out.println("y: " + yPos);
32          //System.out.println("x: " + xPos);
33
34          sprite.setX(xPos);
35          sprite.setY(yPos);
36          xCenter = xPos + sprite.getWidth() / 2;
37          yCenter = yPos + sprite.getHeight() / 2;
38          hitbox.setCenter(xCenter, yCenter);
39
40          if (isHealthy) {
41              updateDirection();
42              sprite.rotate(direction.angle() - currentDegree);
43              currentDegree = direction.angle();
44              if (System.currentTimeMillis() - startTime >
                    description.getLifeTime()) {
45                  isHealthy = false;
46              }
47          } else {
48              hurt();
49          }
50      }
51
52      /**
53       * Update the direction towards the enemy object
54       */
55      private void updateDirection() {
56          Vector2 newDirection = new Vector2(target.getX() -
                xCenter, target.getY() - yCenter);
57          direction.add(newDirection.nor().scl(0.5f));
58          //direction.nor().scl(description.getSpeed());
59          direction.nor().scl(description.getSpeed());
60      }
61  }
```

Javaga/core/src/com/me/Javaga/spaceobject/Player.java

```
1  package com.me.Javaga.spaceobject;
2
3  import com.badlogic.gdx.Gdx;
4  import com.badlogic.gdx.audio.Sound;
5  import com.me.Javaga.gamestate.levels.BulletDescription;
6  import com.me.Javaga.managers.GameKeys;
7  import com.me.Javaga.managers.GameStateManager;
```

```
8  import com.me.Javaga.managers.InformationDrawer;
9
10 import java.util.ArrayList;
11 import java.util.Iterator;
12
13 /**
14  * Class for the Players unit. Contains parameters for position
        as well as the sprite used to draw to canvas.
15  * <p/>
16  * Created by Dansel on 2014-04-30.
17  */
18 public class Player extends SpaceObject {
19
20     private static final String FILENAME = "player3.png";
21     private static long time;
22     //private float rotation;
23     //private float scale;
24     private ArrayList<Bullet> bullets;
25     private Sound sound;
26     private BulletDescription bulletType;
27     private long shootLimit;
28     private static final int MAXHEALTH = 5;
29
30     //Call the super-class's constructor
31     public Player(float xPos, float yPos, ArrayList<Bullet>
           bullets) {
32         super(xPos, yPos);
33         HEIGHT = Gdx.graphics.getHeight();
34         WIDTH = Gdx.graphics.getWidth();
35         this.bullets = bullets;
36         init();
37     }
38
39     @Override
40     public void init() {
41         health = MAXHEALTH;
42         bulletType = BulletDescription.FAST_BULLETS;
43         shootLimit = bulletType.getShootLimit();
44         //Set scalefactor
45         setScale(1f);
46         spriteSetUp(FILENAME);
47         hitbox.setHeight(sHeight * 0.3f).setWidth(sWidth * 0.3f)
               .setCenter(xCenter, yCenter);
48         //Create the sprite with some texture
49         sound = Gdx.audio.newSound(Gdx.files.internal("lazer.mp3
               "));
50         InformationDrawer.setRemainingLife(health - 1);
51     }
52
53     /**
54      * Checks for keypresses and updates the sprite position
55      */
56     @Override
57     public void update() {
```

```
58          if (!isHealthy) {
59              hurt(); // start flashing if the player is hurt
60          }
61
62          if (GameKeys.isDown(GameKeys.UP)) {
63              yPos += 10;
64          }
65          if (GameKeys.isDown(GameKeys.DOWN)) {
66              yPos -= 10;
67          }
68          if (GameKeys.isDown(GameKeys.LEFT)) {
69              xPos -= 10;
70          }
71          if (GameKeys.isDown(GameKeys.RIGHT)) {
72              xPos += 10;
73          }
74
75          if (GameKeys.isDown(GameKeys.SPACE)) {
76              if (System.currentTimeMillis() - time > shootLimit)
                    {
77                  time = System.currentTimeMillis();
78                  fire();
79              }
80          }
81
82          xCenter = xPos + sprite.getWidth() / 2;
83          yCenter = yPos + sprite.getHeight() / 2;
84          //Update position
85          wrap();
86          sprite.setX(xPos);
87          sprite.setY(yPos);
88          //Update hitbox
89          hitbox.setCenter(xCenter, yCenter);
90          InformationDrawer.setRemainingLife(health - 1);
91      }
92
93      /**
94       * Check for collisions
95       *
96       * @param enemyBullets An arraylist of enemybullets
97       * @return true if the player is hit
98       */
99      @Override
100     public boolean checkForCollision(ArrayList<Bullet>
            enemyBullets) {
101         Iterator<Bullet> iterator = enemyBullets.iterator();
102         while (iterator.hasNext()) {
103             Bullet bullet = iterator.next();
104             if (overlap(bullet)) {
105                 health--;
106                 isHealthy = false;
107                 if (!bullet.isIndestructable() || health > 0) {
108                     bullet.dispose();
109                     iterator.remove();
```

```
110                 }
111                 return true;
112             }
113         }
114         return false;
115     }
116 
117     /**
118      * Makes sure the players sprite may not move outside the
             window.
119      */
120     public void wrap() {
121         if (xCenter - sWidth / 2 < 0) {
122             xCenter = 0 + sWidth / 2;
123             xPos = xCenter - sprite.getWidth() / 2;
124         }
125 
126         if (xCenter + sWidth / 2 > WIDTH) {
127             xCenter = WIDTH - sWidth / 2;
128             xPos = xCenter - sprite.getWidth() / 2;
129         }
130 
131         if (yCenter - sHeight / 2 < 0) {
132             yCenter = 0 + sHeight / 2;
133             yPos = yCenter - sprite.getHeight() / 2;
134         }
135 
136         if (yCenter + sHeight / 2 > HEIGHT) {
137             yCenter = HEIGHT - sHeight / 2;
138             yPos = yCenter - sprite.getHeight() / 2;
139         }
140         //Okay, so, since the scaling of the sprite doesn't
                change the boundingbox of it we must
141         //manually find the center of the sprite and from that
                number derive the new edges (the visible edges).
142     }
143 
144     /**
145      * Fire a bullet straight forward
146      */
147     private void fire() {
148         sound.play(GameStateManager.getEffectVolume()); // play
                lazer
149         bullets.add(new Bullet(xCenter, yCenter + sWidth / 2,
                90, bulletType));
150     }
151 
152 
153     /**
154      * Reset the players health
155      */
156     public void resetHealth() {
157         this.health = MAXHEALTH;
158     }
```

```
159
160     @Override
161     public void dispose() {
162         super.dispose();
163         sound.dispose();
164     }
165 }
```

Javaga/core/src/com/me/Javaga/spaceobject/SpaceObject.java

```
1  package com.me.Javaga.spaceobject;
2
3  import com.badlogic.gdx.Gdx;
4  import com.badlogic.gdx.graphics.Texture;
5  import com.badlogic.gdx.graphics.g2d.Sprite;
6  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
7  import com.badlogic.gdx.math.Rectangle;
8
9  import java.util.ArrayList;
10
11 /**
12  * An abstract class describing the essentials of all
13  * space objects in the game
14  * Created by Dansel on 2014-04-30.
15  */
16 public abstract class SpaceObject {
17
18     protected float xPos;
19     protected float yPos;
20     protected Sprite sprite;
21     protected boolean isHealthy;
22     protected boolean isDisposable;
23     protected boolean draw;
24
25     protected float HEIGHT;
26     protected float WIDTH;
27
28     protected float sWidth;
29     protected float sHeight;
30
31     protected float xCenter;
32     protected float yCenter;
33
34     protected float dX;
35     protected float dY;
36     protected float health;
37
38     protected float SCALEFACTOR;
39
40     protected Rectangle hitbox;
41     protected int disposeIndex;
42
43
44     public SpaceObject(float xPos, float yPos) {
```

```
45          this.xCenter = xPos;
46          this.yCenter = yPos;
47          this.isHealthy = true;
48      }
49
50
51      /**
52       * Properly initialize the SpaceObject with sprites and
             logic, should be called
53       * in the subclasses constructor and no other method should
             be called before
54       * this method
55       */
56      public abstract void init();
57
58      /**
59       * Update the logic of the object
60       */
61      public abstract void update();
62
63      /**
64       * Draw the SpaceObject ont the canvas
65       *
66       * @param batch A Sprite batch which draws the sprite onto
67       *              the canvas
68       */
69      public void draw(SpriteBatch batch) {
70          if (isHealthy || draw) {
71              sprite.draw(batch);
72          }
73      }
74
75      public void setScale(float scaleFactor) {
76          SCALEFACTOR = scaleFactor;
77      }
78
79      /**
80       *
81       */
82      public abstract void wrap();
83
84      /**
85       * Check if the object is healthy
86       *
87       * @return True if it is healthy, false if it should be
             discarded
88       */
89      public boolean checkHealthy() {
90          return isHealthy;
91      }
92
93      /**
94       * Check if an object should be discarded and the dispose
             method should be called
```

```
95          *
96          * @return True if the obejct should be disposed as soon as
                possible, otherwise false
97          */
98         public boolean isDisposable() {
99             return isDisposable;
100        }
101
102
103        /**
104         *
105         */
106        public abstract boolean checkForCollision(ArrayList<Bullet>
               bullets);
107
108        /**
109         * See if to spaceobjects overlap each other, which means
                they have colided
110         *
111         * @param obj A spaceObject
112         * @return true if they overlap, false if they don't
113         */
114        public boolean overlap(SpaceObject obj) {
115            return hitbox.overlaps(obj.getHitbox());
116        }
117
118        /**
119         * Return a rectangle stating the area of the sprite
120         *
121         * @return A rectangle
122         */
123        public Rectangle getHitbox() {
124            return hitbox;
125        }
126
127        public void spriteSetUp(String filename) {
128            sprite = new Sprite(new Texture(Gdx.files.internal(
                    filename)));
129
130            xPos = xCenter - sprite.getWidth() / 2;
131            yPos = yCenter - sprite.getHeight() / 2;
132
133            sprite.setX(xPos);
134            sprite.setY(yPos);
135
136            sprite.setScale(SCALEFACTOR);
137            sWidth = sprite.getWidth() * SCALEFACTOR;
138            sHeight = sprite.getHeight() * SCALEFACTOR;
139
140            hitbox = new Rectangle();
141            hitbox.setHeight(sHeight).setWidth(sWidth).setCenter(
                    xCenter, yCenter);
142        }
143
```

```
144     /**
145      * Return the x position of the player
146      *
147      * @return the x position of the player's centrum
148      */
149     public float getX() {
150         return this.xCenter;
151     }
152 
153     /**
154      * Return the y position of the player
155      *
156      * @return the y position of the player's centrum
157      */
158     public float getY() {
159         return this.yCenter;
160     }
161 
162     /**
163      * Creates a flashy effect when the object is damaged
164      */
165     protected void hurt() {
166         if (health <= 0) {
167             isDisposable = true;
168         } else {
169             if (disposeIndex > 100) {
170                 isHealthy = true;
171                 draw = true;
172                 disposeIndex = 0;
173                 return;
174             }
175             disposeIndex++;
176             if (disposeIndex % 10 == 0) {
177                 draw = (draw) ? false : true;
178             }
179         }
180     }
181 
182     public void dispose() {
183         sprite.getTexture().dispose();
184     }
185 }
```

Javaga/core/src/com/me/Javaga/spaceobject/Star.java

```
1 package com.me.Javaga.spaceobject;
2 
3 import com.badlogic.gdx.Gdx;
4 import com.badlogic.gdx.graphics.g2d.SpriteBatch;
5 
6 import java.util.ArrayList;
7 import java.util.Random;
8 
9 /**
```

```
 * A star which flashes in the background
 * Created by Lukas on 2014-05-05.
 */
public class Star extends SpaceObject {

    private final static String FILENAME = "star.png";


    public Star() {
        super(0, 0);
        init();
    }

    @Override
    public void init() {
        Random random = new Random();
        xCenter = random.nextFloat() * Gdx.graphics.getWidth();
        yCenter = Gdx.graphics.getHeight();

        setScale(0.3f);
        spriteSetUp(FILENAME);
        dY = -(random.nextFloat() * 20 + 5);
    }

    @Override
    public void update() {
        yPos += dY;
        yCenter = yPos + sprite.getHeight() / 2;
        sprite.setY(yPos);
        wrap();
    }

    @Override
    public void draw(SpriteBatch batch) {
        sprite.draw(batch);
    }

    @Override
    public void wrap() {
        if ((yCenter - sHeight / 2 < 0)) {
            isHealthy = false;
            isDisposable = true;
        }

    }

    @Override
    public boolean checkForCollision(ArrayList<Bullet> bullets)
        {
        return false;
    }
}
```

Javaga/core/src/com/me/Javaga/managers/BackgroundDrawer.java

```java
package com.me.Javaga.managers;

import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.me.Javaga.spaceobject.Star;

import java.util.ArrayList;
import java.util.Iterator;

/**
 * This class draws the background onto the canvas,
 * it is mostly static to keep the background from changing
 * from gamestate to gamestate
 * Created by Lukas on 2014-05-06.
 */
public class BackgroundDrawer {
    private static ArrayList<Star> stars;
    private static long time; // Keep track of the star
        animation time

    static {
        stars = new ArrayList<Star>();
        time = 0;
    }

    /**
     * Update the background components
     */
    public static void update() {
        if (System.currentTimeMillis() - time > 200) {
            time = System.currentTimeMillis();
            stars.add(new Star());
        }

        Iterator<Star> iterator = stars.iterator();
        while (iterator.hasNext()) {
            Star star = iterator.next();
            if (star.isDisposable()) {
                star.dispose();
                iterator.remove();
            }
        }

        iterator = stars.iterator();
        while (iterator.hasNext()) {
            Star star = iterator.next();
            star.update();
        }
    }

    /**
     * Draw all the background components onto the canvas,
         should be called before all other draw methods
```

```
51          *
52          * @param batch A Sprite bacth
53          */
54         public static void draw(SpriteBatch batch) {
55             //draw stars
56             for (Star star : stars) {
57                 star.draw(batch);
58             }
59         }
60     }
```

Javaga/core/src/com/me/Javaga/managers/Button.java

```
1  package com.me.Javaga.managers;
2
3  import com.badlogic.gdx.Gdx;
4  import com.badlogic.gdx.graphics.Color;
5  import com.badlogic.gdx.graphics.Texture;
6  import com.badlogic.gdx.graphics.g2d.Sprite;
7  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
8  import com.badlogic.gdx.math.Rectangle;
9
10 /**
11  * A button which reacts to user input
12  * Created by Lukas on 2014-05-06.
13  */
14 public class Button {
15
16
17     protected float xPos;
18     protected float yPos;
19     protected Sprite sprite;
20     protected GameStateManager gameStateManager;
21     protected Rectangle rectangle;
22
23     protected float sWidth;
24     protected float sHeight;
25
26     protected float xCenter;
27     protected float yCenter;
28
29     protected float SCALEFACTOR;
30
31     public Button(float xPos, float yPos, GameStateManager
           gameStateManager) {
32         this.xPos = xPos;
33         this.yPos = yPos;
34         this.gameStateManager = gameStateManager;
35     }
36
37     /**
38      * Set the sprite of button, this method should be overriden
              when a button object is created
39      *
```

```
40          * @param filename the name of the sprite
41          */
42         //Should be overriden by all objects
43         public void setSprite(String filename) {
44             sprite = new Sprite(new Texture(Gdx.files.internal(
                   filename)));
45             init();
46         }
47
48         /**
49          * Initialize all fields and components
50          */
51         public void init() {
52             // Does the usuall initializations
53             if (sprite != null) {
54                 setScale(1f); // If we want to scale
55                 sWidth = sprite.getWidth() * SCALEFACTOR;
56                 sHeight = sprite.getHeight() * SCALEFACTOR;
57
58                 //shift position down and to the left so we draw the
                       sprite centered.
59                 xPos -= sprite.getWidth() / 2;
60                 yPos -= sprite.getHeight() / 2;
61
62                 xCenter = xPos + sprite.getWidth() / 2;
63                 yCenter = yPos + sprite.getHeight() / 2;
64
65                 rectangle = new Rectangle();
66                 rectangle.setHeight(sHeight).setWidth(sWidth).
                       setCenter(xCenter, yCenter);
67
68                 sprite.setX(xPos);
69                 sprite.setY(yPos);
70             }
71         }
72
73         /**
74          * Preform an action of some sort when the button is pressed
                ,
75          * this method should be overriden and implemented when a
                button
76          * object is created and added to a buttonContainer
77          */
78         public void preformAction() {
79             // Overide this in all objects
80         }
81
82         /**
83          * Check if the mouse is hovering over the mouse
84          *
85          * @return true if the button is hovering over the button
86          */
87         public boolean isHovering() {
88             return rectangle.contains((float) GameKeys.xMouse(), (
```

```
                float) GameKeys.yMouse());
89      }
90
91      public void setSelected(boolean selected) {
92          if (selected) {
93              sprite.setColor(Color.BLUE);
94          } else {
95              sprite.setColor(Color.WHITE);
96          }
97      }
98
99      /**
100      * Set the scale of the button
101      *
102      * @param scaleFactor a float specifying the scale factor,
103      *                    less than 1 to make it smaller and
                larger than 1 to make it bigger
104      */
105     public void setScale(float scaleFactor) {
106         SCALEFACTOR = scaleFactor;
107         sprite.setScale(SCALEFACTOR);
108     }
109
110     /**
111      * Draw the button onto the canvas
112      *
113      * @param batch A sprite batch
114      */
115     public void draw(SpriteBatch batch) {
116         if (sprite != null) {
117             sprite.draw(batch);
118         }
119     }
120
121     /**
122      * Dispose the button properly when it isn't used anymore
123      */
124     public void dispose() {
125         sprite.getTexture().dispose();
126     }
127
128 }
```

Javaga/core/src/com/me/Javaga/managers/ButtonContainer.java

```
1  package com.me.Javaga.managers;
2
3  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
4
5  import java.util.ArrayList;
6  import java.util.Iterator;
7
8  /**
9   * This class contains the buttons which a menu uses
```

```
10  * Created by Lukas on 2014-05-06.
11  */
12 public class ButtonContainer {
13     private ArrayList<Button> buttons;
14     private Button currentButton;
15     private int currentButtonIndex;
16
17     public ButtonContainer() {
18         buttons = new ArrayList<Button>();
19     }
20
21     /**
22      * Add a button to the container, the order which the
            buttons are added will
23      * affect how the will highligt when you iterate over them
            with the arrow buttons.
24      * The buttons which is highest on the screen should be
            added first etc
25      *
26      * @param button The button which should be added to the
            container
27      */
28     public void addButton(Button button) {
29         buttons.add(button);
30     }
31
32     /**
33      * Handle the user input
34      */
35     public void handleInput() {
36         for (Button button : buttons) {
37             if (button.isHovering()) {
38                 if (currentButton != null) {
39                     currentButton.setSelected(false);
40                 }
41                 currentButton = button;
42                 currentButtonIndex = buttons.indexOf(
                        currentButton);
43                 button.setSelected(true);
44                 if (GameKeys.isMousePressed()) {
45                     button.preformAction();
46                 }
47             }
48         }
49
50         if (GameKeys.isPressed(GameKeys.UP)) {
51             if (currentButton != null) {
52                 if (currentButtonIndex > 0) {
53                     currentButtonIndex--;
54                 }
55                 currentButton.setSelected(false);
56                 currentButton = buttons.get(currentButtonIndex);
57                 currentButton.setSelected(true);
58             } else {
```

```
59                  currentButtonIndex = 0;
60                  currentButton = buttons.get(currentButtonIndex);
61                  currentButton.setSelected(true);
62              }
63          }
64
65          if (GameKeys.isPressed(GameKeys.DOWN)) {
66              if (currentButton != null) {
67                  if (currentButtonIndex != buttons.size() - 1) {
68                      currentButtonIndex++;
69                  }
70                  currentButton.setSelected(false);
71                  currentButton = buttons.get(currentButtonIndex);
72                  currentButton.setSelected(true);
73              } else {
74                  currentButtonIndex = buttons.size() - 1;
75                  currentButton = buttons.get(currentButtonIndex);
76                  currentButton.setSelected(true);
77              }
78          }
79
80          if (GameKeys.isPressed(GameKeys.ENTER)) {
81              if (currentButton != null) {
82                  currentButton.preformAction();
83              }
84          }
85      }
86
87      public void draw(SpriteBatch batch) {
88          for (Button button : buttons) {
89              button.draw(batch);
90          }
91      }
92
93      /**
94       *
95       */
96      public void dispose() {
97          Iterator<Button> buttonIterator = buttons.iterator();
98          while (buttonIterator.hasNext()) {
99              Button button = buttonIterator.next();
100             button.dispose();
101             buttonIterator.remove();
102         }
103
104     }
105
106 }
```

Javaga/core/src/com/me/Javaga/managers/GameInputProcessor.java

```
1 package com.me.Javaga.managers;
2
3 /**
```

```
 * Handle the users input
 * Created by Dansel on 2014-04-30.
 */

import com.badlogic.gdx.Input.Keys;
import com.badlogic.gdx.InputAdapter;

public class GameInputProcessor extends InputAdapter {

    public boolean keyDown(int k) {
        if (k == Keys.UP) {
            GameKeys.setKey(GameKeys.UP, true);
        }
        if (k == Keys.LEFT) {
            GameKeys.setKey(GameKeys.LEFT, true);
        }
        if (k == Keys.DOWN) {
            GameKeys.setKey(GameKeys.DOWN, true);
        }
        if (k == Keys.RIGHT) {
            GameKeys.setKey(GameKeys.RIGHT, true);
        }
        if (k == Keys.ENTER) {
            GameKeys.setKey(GameKeys.ENTER, true);
        }
        if (k == Keys.ESCAPE) {
            GameKeys.setKey(GameKeys.ESCAPE, true);
        }
        if (k == Keys.SPACE) {
            GameKeys.setKey(GameKeys.SPACE, true);
        }
        if (k == Keys.SHIFT_LEFT || k == Keys.SHIFT_RIGHT) {
            GameKeys.setKey(GameKeys.SHIFT, true);
        }
        if (k == Keys.H) {
            GameKeys.setKey(GameKeys.H, true);
        }
        return true;
    }

    public boolean keyUp(int k) {
        if (k == Keys.UP) {
            GameKeys.setKey(GameKeys.UP, false);
        }
        if (k == Keys.LEFT) {
            GameKeys.setKey(GameKeys.LEFT, false);
        }
        if (k == Keys.DOWN) {
            GameKeys.setKey(GameKeys.DOWN, false);
        }
        if (k == Keys.RIGHT) {
            GameKeys.setKey(GameKeys.RIGHT, false);
        }
        if (k == Keys.ENTER) {
```

```
58                GameKeys.setKey(GameKeys.ENTER, false);
59            }
60            if (k == Keys.ESCAPE) {
61                GameKeys.setKey(GameKeys.ESCAPE, false);
62            }
63            if (k == Keys.SPACE) {
64                GameKeys.setKey(GameKeys.SPACE, false);
65            }
66            if (k == Keys.SHIFT_LEFT || k == Keys.SHIFT_RIGHT) {
67                GameKeys.setKey(GameKeys.SHIFT, false);
68            }
69            if (k == Keys.H) {
70                GameKeys.setKey(GameKeys.H, false);
71            }
72            return true;
73        }
74
75
76        @Override
77        public boolean mouseMoved(int screenX, int screenY) {
78            GameKeys.xMouse = screenX;
79            GameKeys.yMouse = screenY;
80            return true;
81        }
82
83        @Override
84        public boolean touchDown(int screenX, int screenY, int
              pointer, int button) {
85            GameKeys.mousePressed = true;
86            //TODO
87            //Probably need to make this less of a "fulhack"
88            return true;
89        }
90
91        @Override
92        public boolean touchUp(int screenX, int screenY, int pointer
              , int button) {
93            GameKeys.mousePressed = false;
94            //TODO
95            //Probably need to make this less of a "fulhack"
96            return true;
97        }
98    }
```

Javaga/core/src/com/me/Javaga/managers/GameKeys.java

```
1   package com.me.Javaga.managers;
2
3   import com.badlogic.gdx.Gdx;
4
5   /**
6    * Contains the state of keys which are pressed (or not pressed)
7    * and the position of the mouse and its state
8    * Created by Dansel on 2014-04-30.
```

```
 */
public class GameKeys {

    public static final int UP = 0;
    public static final int LEFT = 1;
    public static final int DOWN = 2;
    public static final int RIGHT = 3;
    public static final int ENTER = 4;
    public static final int ESCAPE = 5;
    public static final int SPACE = 6;
    public static final int SHIFT = 7;
    public static final int H = 8;
    private static final int NUM_KEYS = 9;
    public static int xMouse;
    public static int yMouse;
    public static boolean mousePressed;
    private static boolean[] keys;
    private static boolean[] pkeys;
    private static boolean prevMousePressed;

    static {
        keys = new boolean[NUM_KEYS];
        pkeys = new boolean[NUM_KEYS];
    }

    /**
     * Update list containing pressed keys.
     */
    public static void update() {
        prevMousePressed = mousePressed;
        for (int i = 0; i < NUM_KEYS; i++) {
            pkeys[i] = keys[i];
        }
    }

    /**
     * Used to create list.
     *
     * @param k int, key ID
     * @param b boolean, pressed or not.
     */
    public static void setKey(int k, boolean b) {
        keys[k] = b;
    }

    /**
     * Checks if a key is "down"
     *
     * @param k int, key ID
     * @return boolean, if key is pressed or not.
     */
    public static boolean isDown(int k) {
        return keys[k];
    }
```

```
63
64      /**
65       * Checks if a key is "down" and previously was "up", aka it
             only returns true on statechange.
66       *
67       * @param k int, key ID
68       * @return boolean, true on statechange.
69       */
70      public static boolean isPressed(int k) {
71          return keys[k] && !pkeys[k];
72      }
73
74      /**
75       * Return the x position of the mouse
76       *
77       * @return the x-value of the mouse
78       */
79      public static int xMouse() {
80          return xMouse;
81      }
82
83      /**
84       * Return the y position of the mouse
85       *
86       * @return the y-value of the mouse
87       */
88      public static int yMouse() {
89          return Gdx.graphics.getHeight() - yMouse; // Sense we
                 have origo in the bottom left corner
90          // we have to convert the y-position like this
91      }
92
93      /**
94       * Return true if the mouse left "button" is held down
95       *
96       * @return true if the mouse is held down
97       */
98      public static boolean isMouseDown() {
99          return mousePressed;
100     }
101
102     /**
103      * Return true if the the mouse left"button" was pressed"
104      *
105      * @return true if the mouse was pressed
106      */
107     public static boolean isMousePressed() {
108         return mousePressed && !prevMousePressed;
109     }
110
111 }
```

Javaga/core/src/com/me/Javaga/managers/GameStateManager.java

```
1  package com.me.Javaga.managers;
2
3  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
4  import com.me.Javaga.gamestate.*;
5
6  /**
7   * Keeps track of the gamestate (play, pause etc) as well as
        updates subclasses.
8   * Created by Dansel on 2014-04-30.
9   */
10 public class GameStateManager {
11     public static final int MENU = 0;
12     public static final int PLAY = 1;
13     public static final int PAUSE = 2;
14     public static final int WELCOME = 3;
15     private static float musicVolume;
16     private static float effectVolume;
17     private GameState currentGameState;
18     private MenuState menu;
19     private PlayState play;
20     private PauseState pause;
21     private WelcomeState welcome;
22
23     public GameStateManager() {
24
25         menu = new MenuState(this);
26         play = new PlayState(this);
27         pause = new PauseState(this);
28         welcome = new WelcomeState(this);
29
30         setMusicVolume(1f);
31         setEffectVolume(0.3f);
32         setState(WELCOME, false);
33         MusicManager.startNewSong(MusicManager.WELCOMESONG);
34     }
35
36     /**
37      * return the volume of the musicplayer
38      *
39      * @return a float between 0-1, 0 is lowest and 1 highest
40      */
41     public static float getMusicVolume() {
42         return musicVolume;
43     }
44
45     /**
46      * Set the music volume of the game
47      *
48      * @param volume a float between 0-1, 0 is lowest and 1
            highest
49      */
50     public static void setMusicVolume(float volume) {
```

```
51          musicVolume = volume;
52      }
53
54      /**
55       * Get the effect volume of the game
56       *
57       * @return a float between 0-1, 0 is lowest and 1 highest
58       */
59      public static float getEffectVolume() {
60          return effectVolume;
61      }
62
63      /**
64       * Get the effect volume of the game
65       *
66       * @param volume a float between 0-1, 0 is lowest and 1
             highest
67       */
68      public static void setEffectVolume(float volume) {
69          effectVolume = volume;
70      }
71
72      /**
73       * Sets the gamestate.
74       *
75       * @param state int number correlating a specific state. 0 =
             menu, 1= play, 2=pause, 3 = welcome screen
76       * @param reset true if the gamestate which you are to set
             as current to should be reset and disposed of before you
             set it
77       */
78      public void setState(int state, boolean reset) {
79          if (state == MENU) {
80              if (reset) {
81                  menu.dispose();
82                  menu = new MenuState(this);
83              }
84              currentGameState = menu;
85          }
86          if (state == PLAY) {
87              if (reset) {
88                  play.dispose();
89                  play = new PlayState(this);
90              }
91              currentGameState = play;
92          }
93          if (state == PAUSE) {
94              if (reset) {
95                  pause.dispose();
96                  pause = new PauseState(this);
97              }
98              currentGameState = pause;
99          }
100         if (state == WELCOME) {
```

```
101                 if (reset) {
102                     welcome.dispose();
103                     welcome = new WelcomeState(this);
104                 }
105                 currentGameState = welcome;
106             }
107 
108         }
109 
110         /**
111          * Updates the game.
112          */
113         public void update() {
114             currentGameState.update();
115         }
116 
117         /**
118          * Draws the entire canvas.
119          *
120          * @param batch SpriteBatch containing a collection of
                 sprites.
121          */
122         public void draw(SpriteBatch batch) {
123             currentGameState.draw(batch);
124         }
125 
126 
127         /**
128          * Dispose the current state and all the things within it
129          *
130          * @param state the state constant
131          */
132         public void dispose(int state) {
133             if (state == MENU) {
134                 menu.dispose();
135                 menu = new MenuState(this);
136             }
137             if (state == PLAY) {
138                 play.dispose();
139                 play = new PlayState(this);
140             }
141             if (state == PAUSE) {
142                 pause.dispose();
143                 pause = new PauseState(this);
144             }
145             if (state == WELCOME) {
146                 welcome.dispose();
147                 welcome = new WelcomeState(this);
148             }
149         }
150 }
```

Javaga/core/src/com/me/Javaga/managers/InformationDrawer.java

```java
package com.me.Javaga.managers;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.Sprite;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;

/**
 * Draws information for the player onto the screen
 * Created by Lukas on 2014-05-14.
 */
public class InformationDrawer {

    private static final String FILENAME = "player3.png";
    private static Sprite sprite;
    private static BitmapFont font;
    private static float spriteWidth;
    private static float remainingLife;
    private static int currentLevel;
    private static long points;
    private static long time;
    private static boolean showInfo;

    static {
        remainingLife = 3;
        currentLevel = 1;
        points = 0;
        sprite = new Sprite(new Texture(Gdx.files.internal(
            FILENAME)));
        sprite.setScale(0.2f);
        spriteWidth = sprite.getWidth() * 0.2f;
        font = new BitmapFont(Gdx.files.internal("white.fnt"),
            Gdx.files.internal("white_0.png"), false);
        font.setScale(0.6f);
    }

    /**
     * Update the information drawer
     */
    public static void update() {
        if (showInfo) {
            if (System.currentTimeMillis() - time > 10000) {
                showInfo = false;
            }
        }
        handleInput();
    }

    /**
     * Draw the information to the canvas, this
     * method should probably be called last of
```

```
51          * all draw method sense this should be in the foreground
52          *
53          * @param batch A SpriteBatch
54          */
55         public static void draw(SpriteBatch batch) {
56             sprite.setY(-sprite.getHeight() / 2 + 20);
57             for (float i = 0, x = -sprite.getWidth() / 2 + 20; i <
                   remainingLife; i++) {
58                 sprite.setX(x);
59                 sprite.draw(batch);
60                 x += spriteWidth;
61             }
62
63             font.draw(batch, "Points: " + Long.toString(points), 0,
                   120);
64             font.draw(batch, "Current Level: " + Integer.toString(
                   currentLevel), 0, 100);
65
66             if (showInfo) {
67                 font.draw(batch, "Use the arrow keys to navigate",
                       Gdx.graphics.getWidth() / 4, Gdx.graphics.
                       getHeight() / 3 + 40);
68                 font.draw(batch, "   Use the space key to fire  ",
                       Gdx.graphics.getWidth() / 4, Gdx.graphics.
                       getHeight() / 3 + 20);
69                 font.draw(batch, "      Try to survive!!!       ",
                       Gdx.graphics.getWidth() / 4, Gdx.graphics.
                       getHeight() / 3);
70
71             }
72         }
73
74         /**
75          * Set the players remaining lifes
76          *
77          * @param life An int stating the amount of lifes the player
                 have left
78          */
79         public static void setRemainingLife(float life) {
80             remainingLife = life;
81         }
82
83         /**
84          * Set the current level of the game
85          *
86          * @param level current level number
87          */
88         public static void setCurretLevel(int level) {
89             currentLevel = level;
90         }
91
92         /**
93          * Add more point to the players score
94          *
```

```
95          * @param point Number of points
96          */
97         public static void updatePoints(int point) {
98             points += point;
99         }
100
101        /**
102         * Reset all the fields
103         */
104        public static void reset() {
105            points = 0;
106            currentLevel = 1;
107            remainingLife = 2;
108        }
109
110        public static void showInfo() {
111            showInfo = true;
112            time = System.currentTimeMillis();
113        }
114
115        public static void handleInput() {
116            if (showInfo) {
117                if (GameKeys.isPressed(GameKeys.ENTER) || GameKeys.
                       isPressed(GameKeys.H)) {
118                    showInfo = false;
119                }
120            } else if (GameKeys.isPressed(GameKeys.H)) {
121                showInfo = true;
122                time = System.currentTimeMillis();
123            }
124        }
125
126
127    }
```

Javaga/core/src/com/me/Javaga/managers/MusicManager.java

```
1   package com.me.Javaga.managers;
2
3   import com.badlogic.gdx.Gdx;
4   import com.badlogic.gdx.audio.Music;
5
6   /**
7    * Keeps track of all the music playing in the game
8    * Created by Lukas on 2014-05-06.
9    */
10  public class MusicManager {
11
12      public static final String PLAYSONG = "Test.mp3";
13      public static final String WELCOMESONG = "Test2.mp3";
14      private static Music musicPlayer;
15
16      /**
17       * Dispose the current song if something was playing and
```

```
         start the new song
18       *
19       * @param filename The filename of the song
20       */
21      public static void startNewSong(String filename) {
22          if (musicPlayer != null) {
23              musicPlayer.dispose();
24          }
25          musicPlayer = Gdx.audio.newMusic(Gdx.files.internal(
                filename));
26          musicPlayer.play();
27          musicPlayer.setVolume(GameStateManager.getMusicVolume())
                ;
28          musicPlayer.setLooping(true);
29      }
30
31      /**
32       * Pause the current song, if nothing is playing, nothing
             will happen
33       */
34      public static void pause() {
35          musicPlayer.pause();
36      }
37
38      /**
39       * Start playing a song, if the song was already playing
40       */
41      public static void play() {
42          musicPlayer.play();
43      }
44
45      /**
46       * Set if the song which is playing should start looping
47       *
48       * @param looping true if the song should start looping,
             false if it should not
49       */
50      public static void setLooping(boolean looping) {
51          musicPlayer.setLooping(looping);
52      }
53
54
55  }
```

Javaga/core/src/com/me/Javaga/gamestate/GameState.java

```
1   package com.me.Javaga.gamestate;
2
3   import com.badlogic.gdx.graphics.g2d.SpriteBatch;
4   import com.me.Javaga.managers.GameStateManager;
5
6   /**
7    * This abstract class is the
8    * Created by Dansel on 2014-04-30.
```

```
 */
public abstract class GameState {

    protected GameStateManager gameStateManager;

    public GameState(GameStateManager gameStateManager) {
        this.gameStateManager = gameStateManager;
    }

    /**
     * Initialize all the components within the state, should be
          called in the constructor of all subclasses
     */
    public abstract void init();

    /**
     * Update the gamestate
     */
    public abstract void update();

    /**
     * Draw something onto the canvas
     *
     * @param batch A sprite batch
     */
    public abstract void draw(SpriteBatch batch);

    /**
     * Do something based on the user input
     */
    public abstract void handleInput();

    /**
     * Disposes all the sprites and sounds within the game state
          to avoid memory leakage
     */
    public abstract void dispose();
}
```

Javaga/core/src/com/me/Javaga/gamestate/MenuState.java

```
package com.me.Javaga.gamestate;

import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.me.Javaga.managers.GameStateManager;

/**
 * Unused class so far
 * Created by Dansel on 2014-04-30.
 */
public class MenuState extends GameState {

    public MenuState(GameStateManager gameStateManager) {
        super(gameStateManager);
```

```
14      }
15
16      @Override
17      public void init() {
18
19      }
20
21      @Override
22      public void update() {
23
24      }
25
26      @Override
27      public void draw(SpriteBatch batch) {
28
29      }
30
31      @Override
32      public void handleInput() {
33
34      }
35
36      @Override
37      public void dispose() {
38
39      }
40  }
```

Javaga/core/src/com/me/Javaga/gamestate/PauseState.java

```
1   package com.me.Javaga.gamestate;
2
3   import com.badlogic.gdx.Gdx;
4   import com.badlogic.gdx.graphics.g2d.SpriteBatch;
5   import com.me.Javaga.managers.*;
6
7   /**
8    * This class shows the pause screen in the game
9    * Created by Dansel on 2014-04-30.
10   */
11  public class PauseState extends GameState {
12
13      private static final String PAUSE = "resume.png";
14      private static final String QUIT = "quit.png";
15      private ButtonContainer currentMenu;
16
17      public PauseState(GameStateManager gameStateManager) {
18          super(gameStateManager);
19          init();
20      }
21
22      @Override
23      public void init() {
24
```

```
25          //Create a new button and override the necisary methods
26          Button pauseButton = new Button(Gdx.graphics.getWidth()
                / 2,
27                  Gdx.graphics.getHeight() / 2, gameStateManager)
                        {
28              @Override
29              public void preformAction() {
30                  gameStateManager.setState(GameStateManager.PLAY,
                            false);
31                  MusicManager.play();
32              }
33          };
34
35          Button quitButton = new Button(Gdx.graphics.getWidth() /
                2,
36                  Gdx.graphics.getHeight() / 2 - 200,
                        gameStateManager) {
37              @Override
38              public void preformAction() {
39                  gameStateManager.dispose(GameStateManager.PLAY);
40                  gameStateManager.setState(GameStateManager.
                        WELCOME, true);
41                  MusicManager.startNewSong(MusicManager.
                        WELCOMESONG);
42              }
43          };
44
45          pauseButton.setSprite(PAUSE);
46          quitButton.setSprite(QUIT);
47          currentMenu = new ButtonContainer();
48          currentMenu.addButton(pauseButton);
49          currentMenu.addButton(quitButton);
50      }
51
52      @Override
53      public void update() {
54          handleInput();
55          BackgroundDrawer.update();
56      }
57
58      @Override
59      public void draw(SpriteBatch batch) {
60          BackgroundDrawer.draw(batch);
61          currentMenu.draw(batch);
62      }
63
64      @Override
65      public void handleInput() {
66          // Lets you exit pause with escape
67          if (GameKeys.isPressed(GameKeys.ESCAPE)) {
68              gameStateManager.setState(GameStateManager.PLAY,
                        false);
69              MusicManager.play();
70
```

```
71          }
72          currentMenu.handleInput();
73      }
74
75      @Override
76      public void dispose() {
77          currentMenu.dispose();
78      }
79  }
```

Javaga/core/src/com/me/Javaga/gamestate/PlayState.java

```
1   package com.me.Javaga.gamestate;
2
3   import com.badlogic.gdx.Gdx;
4   import com.badlogic.gdx.graphics.g2d.SpriteBatch;
5   import com.me.Javaga.gamestate.levels.EnemySpawner;
6   import com.me.Javaga.managers.*;
7   import com.me.Javaga.spaceobject.Bullet;
8   import com.me.Javaga.spaceobject.Enemy;
9   import com.me.Javaga.spaceobject.Player;
10
11  import java.util.ArrayList;
12  import java.util.Iterator;
13
14  /**
15   * This class handles all the game logic and iterates over all
         the objects in the game
16   * Created by Dansel on 2014-04-30.
17   */
18  public class PlayState extends GameState {
19      private Player player;
20      private ArrayList<Bullet> bullets;
21      private ArrayList<Bullet> enemyBullets;
22      private ArrayList<Enemy> enemies;
23      private EnemySpawner spawner;
24
25      public PlayState(GameStateManager gameStateManager) {
26          super(gameStateManager);
27          init();
28      }
29
30      @Override
31      public void init() {
32          bullets = new ArrayList<Bullet>();
33          enemyBullets = new ArrayList<Bullet>();
34          enemies = new ArrayList<Enemy>();
35          player = new Player(Gdx.graphics.getWidth() / 2, 30,
                bullets);
36          spawner = new EnemySpawner(enemyBullets, enemies, player
                , gameStateManager);
37          InformationDrawer.reset();
38          InformationDrawer.showInfo();
39      }
```

```
40
41      @Override
42      public void update() {
43          spawnEnemies();
44          handleInput();
45          checkHealth();
46          player.update();
47          BackgroundDrawer.update();
48
49          for (Enemy enemy : enemies) {
50              enemy.update();
51          }
52
53          for (Bullet bullet : bullets) {
54              bullet.update();
55          }
56
57          for (Bullet bullet : enemyBullets) {
58              bullet.update();
59          }
60      }
61
62      private void checkHealth() {
63          if (player.isDisposable()) {
64              gameStateManager.setState(GameStateManager.WELCOME,
                    true);
65              MusicManager.startNewSong(MusicManager.WELCOMESONG);
66          }
67          Iterator<Bullet> bulletIterator = bullets.iterator();
68          Iterator<Bullet> enemyBulletIterator = enemyBullets.
                iterator();
69          Iterator<Enemy> enemyIterator = enemies.iterator();
70
71          while (bulletIterator.hasNext()) {
72              Bullet bullet = bulletIterator.next();
73              if (bullet.isDisposable()) {
74                  bullet.dispose();
75                  bulletIterator.remove();
76              }
77          }
78
79          while (enemyBulletIterator.hasNext()) {
80              Bullet bullet = enemyBulletIterator.next();
81              if (bullet.isDisposable()) {
82                  bullet.dispose();
83                  enemyBulletIterator.remove();
84              }
85          }
86
87          while (enemyIterator.hasNext()) {
88              Enemy enemy = enemyIterator.next();
89              if (enemy.checkHealthy()) {
90                  enemy.checkForCollision(bullets);
91              }
```

```
92              if (enemy.isDisposable()) {
93                  enemyIterator.remove();
94              }
95          }
96          if (player.checkHealthy()) {
97              player.checkForCollision(enemyBullets);
98          }
99          InformationDrawer.update();
100     }
101
102     @Override
103     public void draw(SpriteBatch batch) {
104         BackgroundDrawer.draw(batch); //draw background
105         player.draw(batch); // draw player
106         // draw player bullets
107         for (Bullet bullet : bullets) {
108             bullet.draw(batch);
109         }
110         //draw enemy bullets
111         for (Bullet bullet : enemyBullets) {
112             bullet.draw(batch);
113         }
114         //draw enemies
115         for (Enemy enemy : enemies) {
116             enemy.draw(batch);
117         }
118
119         InformationDrawer.draw(batch);
120     }
121
122     @Override
123     public void handleInput() {
124         if (GameKeys.isPressed(GameKeys.ESCAPE)) {
125             MusicManager.pause();
126             gameStateManager.setState(GameStateManager.PAUSE,
                    true);
127         }
128     }
129
130     @Override
131     public void dispose() {
132
133         player.dispose();
134         player = null;
135
136         Iterator<Bullet> bulletIterator = bullets.iterator();
137         Iterator<Bullet> enemyBulletIterator = enemyBullets.
                iterator();
138         Iterator<Enemy> enemyIterator = enemies.iterator();
139
140         while (bulletIterator.hasNext()) {
141             Bullet bullet = bulletIterator.next();
142             bullet.dispose();
143             bulletIterator.remove();
```

```
144             }
145
146             while (enemyBulletIterator.hasNext()) {
147                 Bullet bullet = enemyBulletIterator.next();
148                 bullet.dispose();
149                 enemyBulletIterator.remove();
150             }
151
152             while (enemyIterator.hasNext()) {
153                 Enemy enemy = enemyIterator.next();
154                 enemy.dispose();
155                 enemyIterator.remove();
156             }
157         }
158
159         /**
160          * Spawn enemies onto the level
161          */
162         public void spawnEnemies() {
163             spawner.spawnEnemy();
164         }
165     }
```

Javaga/core/src/com/me/Javaga/gamestate/WelcomeState.java

```
1  package com.me.Javaga.gamestate;
2
3  import com.badlogic.gdx.Gdx;
4  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
5  import com.me.Javaga.managers.*;
6
7  /**
8   * This is the class which shows the start screen of the game
9   * Created by Dansel on 2014-04-30.
10  */
11 public class WelcomeState extends GameState {
12
13     private static final String START = "start_game.png";
14     private ButtonContainer menuContainer;
15
16     public WelcomeState(GameStateManager gameStateManager) {
17         super(gameStateManager);
18         init();
19     }
20
21     @Override
22     public void init() {
23         menuContainer = new ButtonContainer();
24
25         Button startGame = new Button(Gdx.graphics.getWidth() /
               2,
26                 Gdx.graphics.getHeight() / 2, gameStateManager)
                       {
27             @Override
```

```
28              public void preformAction() {
29                  gameStateManager.setState(GameStateManager.PLAY,
                        true);
30                  MusicManager.startNewSong(MusicManager.PLAYSONG)
                        ;
31              }
32          };
33          startGame.setSprite(START);
34          menuContainer.addButton(startGame);
35      }
36
37      @Override
38      public void update() {
39          handleInput();
40          BackgroundDrawer.update();
41      }
42
43      @Override
44      public void draw(SpriteBatch batch) {
45          BackgroundDrawer.draw(batch);
46          menuContainer.draw(batch);
47      }
48
49      @Override
50      public void handleInput() {
51          menuContainer.handleInput();
52      }
53
54      @Override
55      public void dispose() {
56          menuContainer.dispose();
57      }
58  }
```

Javaga/core/src/com/me/Javaga/gamestate/levels/Level.java

```
1   package com.me.Javaga.gamestate.levels;
2
3   /**
4    * The enum contains a description of all the levels and all the
         different stages in the game
5    * Created by Dansel on 2014-05-05.
6    */
7   public enum Level {
8
9       LEVEL1(
10              new StageDescription[]{
11                      new StageDescription(true, 10),
12                      new StageDescription(EnemyDescription.
                            HEAVY_ENEMY,
13                              EnemyMovement.MOVEMENT1, 3, -1),
14                      new StageDescription(EnemyDescription.
                            HEAVY_ENEMY,
15                              EnemyMovement.MOVEMENT8, 3, 0),
```

```
16                          new StageDescription(EnemyDescription.
                                HEAVY_ENEMY,
17                                  EnemyMovement.MOVEMENT9, 3, -1),
18                          new StageDescription(EnemyDescription.
                                HEAVY_ENEMY,
19                                  EnemyMovement.MOVEMENT3, 5, 0),
20                          new StageDescription(EnemyDescription.
                                HEAVY_ENEMY,
21                                  EnemyMovement.MOVEMENT4, 5, -1),
22                          new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
23                                  EnemyMovement.MOVEMENT8, 5, 0),
24                          new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
25                                  EnemyMovement.MOVEMENT9, 5, 10),
26                          new StageDescription(EnemyDescription.BOSS1,
27                                  EnemyMovement.MOVEMENT7, 1, -1)
28                  }
29      ),
30      LEVEL2(
31                  new StageDescription[]{new StageDescription(
                        EnemyDescription.FAST_ENEMY,
32                          EnemyMovement.MOVEMENT7, 1, 20),
33                          new StageDescription(EnemyDescription.
                                FAST_ENEMY,
34                                  EnemyMovement.MOVEMENT8, 1, 20),
35                          new StageDescription(EnemyDescription.
                                FAST_ENEMY,
36                                  EnemyMovement.MOVEMENT9, 1, -1),
37                          new StageDescription(EnemyDescription.
                                BOMB_ENEMY,
38                                  EnemyMovement.MOVEMENT1, 3, 0),
39                          new StageDescription(EnemyDescription.
                                BOMB_ENEMY,
40                                  EnemyMovement.MOVEMENT3, 3, 0),
41                          new StageDescription(EnemyDescription.
                                BOMB_ENEMY,
42                                  EnemyMovement.MOVEMENT4, 3, -1),
43                          new StageDescription(EnemyDescription.
                                FAST_ENEMY,
44                                  EnemyMovement.MOVEMENT7, 1, 5),
45                          new StageDescription(EnemyDescription.
                                FAST_ENEMY,
46                                  EnemyMovement.MOVEMENT3, 1, 3),
47                          new StageDescription(EnemyDescription.
                                FAST_ENEMY,
48                                  EnemyMovement.MOVEMENT4, 1, -1),
49                          new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
50                                  EnemyMovement.MOVEMENT1, 3, 0),
51                          new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
52                                  EnemyMovement.MOVEMENT3, 3, 0),
53                          new StageDescription(EnemyDescription.
```

```
                        STANDARD_ENEMY,
54                          EnemyMovement.MOVEMENT4, 3, -1),
55                  new StageDescription(EnemyDescription.
                        FAST_ENEMY,
56                          EnemyMovement.MOVEMENT3, 1, 0),
57                  new StageDescription(EnemyDescription.
                        FAST_ENEMY,
58                          EnemyMovement.MOVEMENT4, 1, -1),
59                  new StageDescription(EnemyDescription.BOSS2,
60                          EnemyMovement.MOVEMENT6, 1, -1)
61          }
62      ),
63      LEVEL3(
64          new StageDescription[]{new StageDescription(
                EnemyDescription.HEAVY_ENEMY,
65                  EnemyMovement.MOVEMENT1, 1, 2),
66                  new StageDescription(EnemyDescription.
                        HEAVY_ENEMY,
67                          EnemyMovement.MOVEMENT1, 3, 2),
68                  new StageDescription(EnemyDescription.
                        HEAVY_ENEMY,
69                          EnemyMovement.MOVEMENT1, 5, 2),
70                  new StageDescription(EnemyDescription.
                        HEAVY_ENEMY,
71                          EnemyMovement.MOVEMENT1, 7, 2),
72                  new StageDescription(EnemyDescription.
                        HEAVY_ENEMY,
73                          EnemyMovement.MOVEMENT1, 9, -1),
74                  new StageDescription(EnemyDescription.
                        BOMB_ENEMY,
75                          EnemyMovement.MOVEMENT1, 1, 2),
76                  new StageDescription(EnemyDescription.
                        BOMB_ENEMY,
77                          EnemyMovement.MOVEMENT1, 3, 2),
78                  new StageDescription(EnemyDescription.
                        BOMB_ENEMY,
79                          EnemyMovement.MOVEMENT1, 5, 2),
80                  new StageDescription(EnemyDescription.
                        BOMB_ENEMY,
81                          EnemyMovement.MOVEMENT1, 7, 2),
82                  new StageDescription(EnemyDescription.
                        BOMB_ENEMY,
83                          EnemyMovement.MOVEMENT1, 9, -1),
84                  new StageDescription(EnemyDescription.
                        FAST_ENEMY,
85                          EnemyMovement.MOVEMENT8, 2, 10),
86                  new StageDescription(EnemyDescription.
                        FAST_ENEMY,
87                          EnemyMovement.MOVEMENT9, 2, 10),
88                  new StageDescription(EnemyDescription.
                        STANDARD_ENEMY,
89                          EnemyMovement.MOVEMENT1, 10, -1),
90                  new StageDescription(EnemyDescription.BOSS3,
91                          EnemyMovement.MOVEMENT6, 1, -1)
```

```
92                  }
93          ),
94          LEVEL4(
95                  new StageDescription[]{
96                          new StageDescription(EnemyDescription.
                                BOMB_SHIELD,
97                                  EnemyMovement.MOVEMENT1, 1, 0),
98                          new StageDescription(EnemyDescription.
                                BOMB_ENEMY,
99                                  EnemyMovement.MOVEMENT1, 1, 3),
100                         new StageDescription(EnemyDescription.
                                BOMB_SHIELD,
101                                 EnemyMovement.MOVEMENT1, 3, 0),
102                         new StageDescription(EnemyDescription.
                                BOMB_ENEMY,
103                                 EnemyMovement.MOVEMENT1, 3, 3),
104                         new StageDescription(EnemyDescription.
                                BOMB_SHIELD,
105                                 EnemyMovement.MOVEMENT1, 5, 0),
106                         new StageDescription(EnemyDescription.
                                BOMB_ENEMY,
107                                 EnemyMovement.MOVEMENT1, 5, 3),
108                         new StageDescription(EnemyDescription.
                                BOMB_SHIELD,
109                                 EnemyMovement.MOVEMENT1, 7, 0),
110                         new StageDescription(EnemyDescription.
                                BOMB_ENEMY,
111                                 EnemyMovement.MOVEMENT1, 7, 30),
112                         new StageDescription(EnemyDescription.
                                FAST_ENEMY,
113                                 EnemyMovement.MOVEMENT10, 1, 10),
114                         new StageDescription(EnemyDescription.
                                FAST_ENEMY,
115                                 EnemyMovement.MOVEMENT10, 2, -1),
116                         new StageDescription(EnemyDescription.
                                HEAVY_ENEMY,
117                                 EnemyMovement.MOVEMENT4, 5, -1),
118                         new StageDescription(EnemyDescription.
                                BOMB_ENEMY,
119                                 EnemyMovement.MOVEMENT1, 3, 0),
120                         new StageDescription(EnemyDescription.
                                BOMB_ENEMY,
121                                 EnemyMovement.MOVEMENT3, 3, 0),
122                         new StageDescription(EnemyDescription.
                                BOMB_ENEMY,
123                                 EnemyMovement.MOVEMENT4, 3, -1),
124                         new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
125                                 EnemyMovement.MOVEMENT1, 3, 0),
126                         new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
127                                 EnemyMovement.MOVEMENT3, 3, 0),
128                         new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
```

```
129                                 EnemyMovement.MOVEMENT4, 3, -1),
130                         new StageDescription(EnemyDescription.BOSS4,
131                                 EnemyMovement.MOVEMENT1, 1, -1)
132                 }
133         ),
134         LEVEL5(
135                 new StageDescription[]{
136                         new StageDescription(EnemyDescription.
                                STANDARD_SHIELD,
137                                 EnemyMovement.MOVEMENT8, 1, 0),
138                         new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
139                                 EnemyMovement.MOVEMENT8, 1, 0),
140                         new StageDescription(EnemyDescription.
                                STANDARD_SHIELD,
141                                 EnemyMovement.MOVEMENT9, 1, 0),
142                         new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
143                                 EnemyMovement.MOVEMENT9, 1, 20),
144                         new StageDescription(EnemyDescription.
                                STANDARD_SHIELD,
145                                 EnemyMovement.MOVEMENT8, 3, 0),
146                         new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
147                                 EnemyMovement.MOVEMENT8, 3, 0),
148                         new StageDescription(EnemyDescription.
                                STANDARD_SHIELD,
149                                 EnemyMovement.MOVEMENT9, 3, 0),
150                         new StageDescription(EnemyDescription.
                                STANDARD_ENEMY,
151                                 EnemyMovement.MOVEMENT9, 3, 20),
152                         new StageDescription(EnemyDescription.
                                HEAVY_SHIELD,
153                                 EnemyMovement.MOVEMENT8, 5, 0),
154                         new StageDescription(EnemyDescription.
                                HEAVY_ENEMY,
155                                 EnemyMovement.MOVEMENT8, 5, 0),
156                         new StageDescription(EnemyDescription.
                                HEAVY_SHIELD,
157                                 EnemyMovement.MOVEMENT9, 5, 0),
158                         new StageDescription(EnemyDescription.
                                HEAVY_ENEMY,
159                                 EnemyMovement.MOVEMENT9, 5, 20),
160                         new StageDescription(EnemyDescription.
                                BOSS_SHIELD,
161                                 EnemyMovement.MOVEMENT6, 1, 0),
162                         new StageDescription(EnemyDescription.
                                BOSS_SHIELD,
163                                 EnemyMovement.MOVEMENT6, 1, 0),
164                         new StageDescription(EnemyDescription.BOSS5,
165                                 EnemyMovement.MOVEMENT6, 1, -1)
166                 }
167         );
168
```

```
169     private StageDescription[] stages;
170     public static final int NUMBER_OF_LEVELS = 5;
171
172     /**
173      * @param stages an array consisting of stageDescriptiont
            which describes the level
174      */
175     private Level(StageDescription[] stages) {
176         this.stages = stages;
177     }
178
179     public static class StageDescription {
180         private EnemyDescription enemyType;
181         private EnemyMovement movementType;
182         private int numberOfEnemies;
183         private int time;
184         private boolean gameOver;
185         private boolean rest;
186
187         /**
188          * @param enemyType        An EnemyDescription object
                which describes the enemy type
189          *                         (if you wish to spawn several
                different enemies, simply create a new stage and set
                 this time to 0)
190          * @param movementType     An EnemyMovement object which
                describes the movement of the enemywave
191          * @param numberOfEnemies the number of enemies which
                should be spawned during this wave
192          * @param time             The time it should take for
                the follow
193          *                         enemy squad to spawn, in
                seconds or -1
194          *                         if all enemies in the current
                squad needs to be defeated beforde
195          *                         the next wave is launched
196          */
197         public StageDescription(EnemyDescription enemyType,
                EnemyMovement movementType,
198                                 int numberOfEnemies, int time) {
199             this.enemyType = enemyType;
200             this.movementType = movementType;
201             this.numberOfEnemies = numberOfEnemies;
202             this.time = time;
203         }
204
205
206         /**
207          * If you want to show that the game is now over, use
                this constructor
208          *
209          * @param gameOver true if the game is over
210          */
211         public StageDescription(boolean gameOver) {
```

```
212             this.gameOver = gameOver;
213         }
214
215         /**
216          * If you want the game to take a pause during a
                specified time without spawning enemies, use this
                constructor
217          *
218          * @param rest true if the game should rest
219          * @param time the amount of the the rest should take,
                in seconds
220          */
221         public StageDescription(boolean rest, int time) {
222             this.rest = rest;
223             this.time = time;
224         }
225
226         /**
227          * Return the enemyDescription
228          *
229          * @return EnemyDescription
230          */
231         public EnemyDescription getEnemyType() {
232             return this.enemyType;
233         }
234
235         /**
236          * An EnenemyMovement object which specifies the
                movement pattern of the enemie
237          *
238          * @return An EnemyMovement Object
239          */
240         public EnemyMovement getMovementType() {
241             return this.movementType;
242         }
243
244         /**
245          * The number of enemies whish should be spawned during
                this stage
246          *
247          * @return number of enemies
248          */
249         public int getNumberOfEnemies() {
250             return this.numberOfEnemies;
251         }
252
253         /**
254          * True if the game is over
255          *
256          * @return true if the game is over, otherwise false
257          */
258         public boolean isGameOver() {
259             return this.gameOver;
260         }
```

```
261
262         /**
263          * True if the game should take a rest without spawning
                any enemies
264          *
265          * @return true if rest, othervise false
266          */
267         public boolean rest() {
268             return this.rest;
269         }
270
271         /**
272          * Retunr the time it should take for the next enemy
                squad to appear, in seconds
273          *
274          * @return and int stating the time for the next
                spawning to occur
275          */
276         public int time() {
277             return this.time;
278         }
279
280     }
281
282     /**
283      * Returns the level based on the input number, 1 returns
            level 1 etc.
284      *
285      * @param level the number of the level
286      * @return The level with that specified number
287      */
288     public static Level getLevel(int level) {
289         if (level == 1) {
290             return LEVEL1;
291         } else if (level == 2) {
292             return LEVEL2;
293         } else if (level == 3) {
294             return LEVEL3;
295         } else if (level == 4) {
296             return LEVEL4;
297         } else if (level == 5) {
298             return LEVEL5;
299         } else {
300             return LEVEL1;
301         }
302     }
303
304     /**
305      * Get a specific stage in the level
306      *
307      * @param index
308      * @return A StageDescription
309      */
310
```

```
311     public StageDescription getStage(int index) {
312         return stages[index];
313     }
314 
315     /**
316      * Returns the amount of stages which the level contains
317      *
318      * @return
319      */
320     public int getLevelLength() {
321         return stages.length;
322     }
323 }
```

Javaga/core/src/com/me/Javaga/gamestate/levels/EnemySpawner.java

```
1  package com.me.Javaga.gamestate.levels;
2 
3  import com.badlogic.gdx.math.Vector2;
4  import com.me.Javaga.managers.GameStateManager;
5  import com.me.Javaga.managers.InformationDrawer;
6  import com.me.Javaga.spaceobject.Boss;
7  import com.me.Javaga.spaceobject.Bullet;
8  import com.me.Javaga.spaceobject.Enemy;
9  import com.me.Javaga.spaceobject.Player;
10 
11 import java.util.ArrayList;
12 
13 /**
14  * The class whish is responisble for spawning new enemies and
        keeping track on the current level
15  * Created by Lukas on 2014-05-12.
16  */
17 public class EnemySpawner {
18     private ArrayList<Bullet> enemyBullets;
19     private ArrayList<Enemy> enemies;
20     private Player player;
21     private GameStateManager gameStateManager;
22     private int time;
23     private boolean rest;
24     private long currentTime;
25     private Level currentLevel;
26     private int stageIndex;
27     private int levelIndex;
28 
29     /**
30      * @param enemyBullets      The arraylist which contains all
            the enemy bullets
31      * @param enemies           The arraylist in which all
            enemies should be spawned
32      * @param player            The player of the playstate class
33      * @param gameStateManager The games gamestate manager
34      */
35     public EnemySpawner(
```

```
36              ArrayList<Bullet> enemyBullets, ArrayList<Enemy>
                    enemies, Player player,
37              GameStateManager gameStateManager) {
38          this.currentLevel = Level.LEVEL1;
39          this.stageIndex = -1;
40          this.levelIndex = 0;
41          this.enemyBullets = enemyBullets;
42          this.enemies = enemies;
43          this.player = player;
44          this.time = 0;
45          this.gameStateManager = gameStateManager;
46          this.currentTime = System.currentTimeMillis();
47      }
48
49      /**
50       * Spawn a new enemy wave
51       *
52       * @param stage The current stage
53       */
54      private void setEnemyWave(Level.StageDescription stage) {
55          EnemyMovement movement = stage.getMovementType();
56
57          Vector2 start = movement.getStartCoordinate();
58          Vector2[] goals = movement.getCoordinates();
59          Vector2 direction = movement.getStartDirection();
60
61          float enemyDifferenceX = movement.getdX() / stage.
                getNumberOfEnemies();
62          float enemyDifferenceY = movement.getdY() / stage.
                getNumberOfEnemies();
63
64          float dX = 0;
65          float dY = 0;
66
67          for (int i = 0, j; i < stage.getNumberOfEnemies(); i++)
                {
68              float degree;
69
70              if (i % 2 != 0) {
71                  j = 1;
72              } else {
73                  j = -1;
74              }
75
76              dX += i * j * enemyDifferenceX;
77              dY += i * j * enemyDifferenceY;
78              degree = (90 / stage.getNumberOfEnemies()) * i * j;
79              direction.rotate(degree);
80
81              Enemy enemy;
82
83              if (stage.getEnemyType().isBoss()) {
84                  enemy = new Boss(start.x + dX,
85                          start.y + dY,
```

```
86                          stage.getEnemyType(), enemyBullets,
                                player);
87              } else {
88                  enemy = new Enemy(start.x + dX,
89                          start.y + dY,
90                          stage.getEnemyType(), enemyBullets,
                                player);
91              }
92
93              enemy.setDirection(direction.x, direction.y);
94
95              for (Vector2 vector : goals) {
96                  enemy.addNewGoal(vector.x + dX,
97                          vector.y + dY);
98              }
99              this.enemies.add(enemy);
100         }
101     }
102
103     /**
104      * If it is allowed to, this method will spawn a new wave of
              enemies onto the screen
105      */
106     public void spawnEnemy() {
107         if (!canSpawn()) {
108             return;
109         }
110         Level.StageDescription stage = getNextStage();
111         if (stage.isGameOver()) {
112             gameStateManager.setState(GameStateManager.WELCOME,
                    true);
113             return;
114         }
115         if (!stage.rest()) {
116             setEnemyWave(stage);
117         } else {
118             rest = true;
119         }
120         time = stage.time() * 1000;
121         currentTime = System.currentTimeMillis();
122     }
123
124     /**
125      * Returns the current stage description for the spawn
              enemies method
126      *
127      * @return The next stageDescritpion or quits the game if
              there are no more levels
128      */
129     private Level.StageDescription getNextStage() {
130         if (stageIndex + 1 >= currentLevel.getLevelLength()) {
131             stageIndex = -1;
132             levelIndex++;
133             InformationDrawer.setCurretLevel(levelIndex + 1);
```

```
134             currentLevel = Level.getLevel(levelIndex + 1);
135             player.resetHealth();
136         }
137         if (levelIndex >= Level.NUMBER_OF_LEVELS) {
138             return new Level.StageDescription(true); // tells
                    the game the level is won
139         }
140         stageIndex++;
141         return currentLevel.getStage(stageIndex);
142     }
143
144     public boolean canSpawn() {
145         if (time == -1000 && !enemies.isEmpty()) {
146             return false;
147         }
148
149         if (rest) {
150             if (System.currentTimeMillis() - currentTime > time)
                        {
151                 rest = false;
152                 return true;
153             }
154             return false;
155         }
156
157         return (System.currentTimeMillis() - currentTime > time)
                    || enemies.isEmpty();
158     }
159 }
```

Javaga/core/src/com/me/Javaga/gamestate/levels/EnemyMovement.java

```
1  package com.me.Javaga.gamestate.levels;
2
3  import com.badlogic.gdx.Gdx;
4  import com.badlogic.gdx.math.Vector2;
5
6  /**
7   * Describes the movement of enemies with predetermined
        coordinates
8   * Created by Lukas on 2014-05-12.
9   */
10 public enum EnemyMovement {
11     MOVEMENT1(
12             new Vector2(Gdx.graphics.getWidth() / 2, Gdx.
                    graphics.getHeight() + 100), // startposition
13
14             new Vector2(0, -1), // start direction
15
16             new Vector2[]{ // Coordinates
17                     new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2),
18                     new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2 - 100),
```

```
19                      new Vector2(Gdx.graphics.getWidth() / 2 +
                            100, Gdx.graphics.getHeight() / 2 - 100),
20                      new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2 - 100),
21                      new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2 + 100),
22                      new Vector2(Gdx.graphics.getWidth() / 2,
                            -100),
23              },
24
25              0, // dY
26              Gdx.graphics.getWidth() //dX
27      ),
28      MOVEMENT2(
29              new Vector2(Gdx.graphics.getWidth() / 2, Gdx.
                    graphics.getHeight() / 2 + 100),
30
31              new Vector2(0, 0),
32
33              new Vector2[]{
34                      new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2),
35                      new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2 - 100),
36                      new Vector2(Gdx.graphics.getWidth() / 2 -
                            100, Gdx.graphics.getHeight() / 2 - 100),
37                      new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2 - 100),
38                      new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2 + 100),
39                      new Vector2(Gdx.graphics.getWidth() / 2,
                            -100),
40              },
41              0,
42              Gdx.graphics.getWidth()
43      ),
44      MOVEMENT3(
45              new Vector2(-20, Gdx.graphics.getHeight() / 2),
46
47              new Vector2(0, 0),
48
49              new Vector2[]{
50                      new Vector2(Gdx.graphics.getWidth() / 2 -
                            50, Gdx.graphics.getHeight() / 2),
51                      new Vector2(Gdx.graphics.getWidth() / 2 -
                            50, Gdx.graphics.getHeight() / 4),
52                      new Vector2(Gdx.graphics.getWidth() + 100,
                            Gdx.graphics.getHeight() / 4)
53              },
54
55              Gdx.graphics.getHeight() / 2,
56              0
57      ),
58
```

```
59      MOVEMENT4(
60              new Vector2(Gdx.graphics.getWidth() + 20, 3 * Gdx.
                    graphics.getHeight() / 4),
61
62              new Vector2(0, 0),
63
64              new Vector2[]{
65                      new Vector2(Gdx.graphics.getWidth() / 2 +
                            50, 3 * Gdx.graphics.getHeight() / 4),
66                      new Vector2(Gdx.graphics.getWidth() / 2 +
                            50, 2 * Gdx.graphics.getHeight() / 4),
67                      new Vector2(-100, 2 * Gdx.graphics.getHeight
                            () / 4)
68              },
69
70              Gdx.graphics.getHeight() / 2,
71              0
72      ),
73      MOVEMENT5(
74              new Vector2(Gdx.graphics.getWidth() / 2, Gdx.
                    graphics.getHeight() + 200),
75
76              new Vector2(0, 0),
77
78              new Vector2[]{
79                      new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() - 20),
80                      new Vector2(Gdx.graphics.getWidth() - 20,
                            Gdx.graphics.getHeight() / 2),
81                      new Vector2(Gdx.graphics.getWidth() / 2, 20)
                            ,
82                      new Vector2(20, Gdx.graphics.getHeight() /
                            2)
83              },
84
85              Gdx.graphics.getHeight() / 2,
86              Gdx.graphics.getWidth() / 2
87      ),
88      MOVEMENT6(
89              new Vector2(Gdx.graphics.getWidth() / 2, Gdx.
                    graphics.getHeight() + 200),
90
91              new Vector2(0, 0),
92
93              new Vector2[]{
94                      new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2 + 100),
95                      new Vector2(Gdx.graphics.getWidth() / 2 +
                            100, Gdx.graphics.getHeight() / 2 + 100),
96                      new Vector2(Gdx.graphics.getWidth() / 2 +
                            100, Gdx.graphics.getHeight() / 2 - 100),
97                      new Vector2(Gdx.graphics.getWidth() / 2 -
                            100, Gdx.graphics.getHeight() / 2 - 100),
98                      new Vector2(Gdx.graphics.getWidth() / 2 -
```

```
                        100, Gdx.graphics.getHeight() / 2 + 100),
99              },
100
101             0,
102             Gdx.graphics.getWidth() / 2
103     ),
104     MOVEMENT7(
105             new Vector2(Gdx.graphics.getWidth() / 2, Gdx.
                    graphics.getHeight() + 200),
106
107             new Vector2(0, 0),
108
109             new Vector2[]{
110                     new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2 + 100),
111                     new Vector2(Gdx.graphics.getWidth() / 2 +
                            100, Gdx.graphics.getHeight() / 2 + 100),
112                     new Vector2(Gdx.graphics.getWidth() / 2 +
                            100, Gdx.graphics.getHeight() / 2 - 100),
113                     new Vector2(Gdx.graphics.getWidth() / 2 -
                            100, Gdx.graphics.getHeight() / 2 - 100),
114                     new Vector2(Gdx.graphics.getWidth() / 2 -
                            100, Gdx.graphics.getHeight() / 2 + 100),
115                     new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() + 200)
116             },
117
118             0,
119             Gdx.graphics.getWidth() / 2
120     ),
121     MOVEMENT8(
122             new Vector2(-100, Gdx.graphics.getHeight() + 100),
123
124             new Vector2(0, 0),
125
126             new Vector2[]{
127                     new Vector2(0, Gdx.graphics.getHeight()),
128                     new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2),
129                     new Vector2(Gdx.graphics.getWidth(), 0),
130                     new Vector2(Gdx.graphics.getWidth() + 100,
                            -100)
131             },
132
133             Gdx.graphics.getHeight() / 2,
134             Gdx.graphics.getWidth() / 2
135     ),
136     MOVEMENT9(
137             new Vector2(Gdx.graphics.getWidth() + 100, Gdx.
                    graphics.getHeight() + 100),
138
139             new Vector2(0, 0),
140
141             new Vector2[]{
```

```
142                     new Vector2(Gdx.graphics.getWidth(), Gdx.
                            graphics.getHeight()),
143                     new Vector2(Gdx.graphics.getWidth() / 2, Gdx
                            .graphics.getHeight() / 2),
144                     new Vector2(0, 0),
145                     new Vector2(-100, -100)
146             },
147
148             Gdx.graphics.getHeight() / 2,
149             -Gdx.graphics.getWidth() / 2
150     ),
151     MOVEMENT10(
152             new Vector2(-100, Gdx.graphics.getHeight() - 10),
153
154             new Vector2(0, 0),
155
156             new Vector2[]{
157                     new Vector2(Gdx.graphics.getWidth() - 10,
                            Gdx.graphics.getHeight() - 10),
158                     new Vector2(10, Gdx.graphics.getHeight() -
                            10),
159                     new Vector2(Gdx.graphics.getWidth() - 10,
                            Gdx.graphics.getHeight() - 10),
160                     new Vector2(10, Gdx.graphics.getHeight() -
                            10),
161                     new Vector2(Gdx.graphics.getWidth() - 10,
                            Gdx.graphics.getHeight() - 10),
162                     new Vector2(10, Gdx.graphics.getHeight() -
                            10),
163                     new Vector2(Gdx.graphics.getWidth() - 10,
                            Gdx.graphics.getHeight() - 10),
164                     new Vector2(10, Gdx.graphics.getHeight() -
                            10),
165                     new Vector2(Gdx.graphics.getWidth() - 10,
                            Gdx.graphics.getHeight() - 10),
166                     new Vector2(10, Gdx.graphics.getHeight() -
                            10),
167                     new Vector2(Gdx.graphics.getWidth() - 10,
                            Gdx.graphics.getHeight() - 10),
168                     new Vector2(-100, Gdx.graphics.getHeight() -
                             10),
169             },
170
171             Gdx.graphics.getHeight() / 2,
172             0
173     );
174
175
176     private Vector2 startCoordinate;
177     private Vector2 startDirection;
178     private Vector2[] coordinates;
179     private float dY;
180     private float dX;
181
```

```
182
183     /**
184      * Create a movement pattern for an enemy
185      *
186      * @param startCoordinates The spawn point for the enemy
187      * @param startDirection   The start direction for the enemy
188      * @param coordinates      An array of coordinates which the
             enemy will visit
189      * @param dY               In the case of several enemies
             using the same movement pattern at the same time
190      *                         this should specify how mush
             space they should share vertically
191      * @param dX               In the case of several enemies
             using the same movement pattern at the same time
192      *                         this should specify how mush
             space they should share horizontally
193      */
194     private EnemyMovement(Vector2 startCoordinates, Vector2
            startDirection,
195                           Vector2[] coordinates, float dY, float
                                dX) {
196         this.startCoordinate = startCoordinates;
197         this.startDirection = startDirection;
198         this.coordinates = coordinates;
199         this.dY = dY;
200         this.dX = dX;
201
202     }
203
204     /**
205      * Get the start coordinates
206      *
207      * @return A Vector2
208      */
209     public Vector2 getStartCoordinate() {
210         return this.startCoordinate;
211     }
212
213     /**
214      * Get the start direction
215      *
216      * @return A Vector2
217      */
218     public Vector2 getStartDirection() {
219         return this.startDirection;
220     }
221
222     /**
223      * The array of coordinates
224      *
225      * @return A Vector2 array
226      */
227     public Vector2[] getCoordinates() {
228         return this.coordinates;
```

```
229     }
230
231     /**
232      * Get the vertically shared space
233      *
234      * @return a float
235      */
236     public float getdY() {
237         return this.dY;
238     }
239
240     /**
241      * Get the horizontally shared spaced
242      *
243      * @return a float
244      */
245     public float getdX() {
246         return this.dX;
247     }
248
249 }
```

Javaga/core/src/com/me/Javaga/gamestate/levels/EnemyDescription.java

```
1  package com.me.Javaga.gamestate.levels;
2
3  /**
4   * This enum contains a description of all enemy classes
5   * Created by Lukas on 2014-05-12.
6   */
7  public enum EnemyDescription {
8      STANDARD_ENEMY(
9              "evil1.png", // filename
10             1f, // scale
11             0.9f, // hitbox scale
12             2f, // speed
13             60, // accyracy
14             BulletDescription.BULLETS, // bullet type
15             1, // health
16             false // is boss
17     ),
18     BOMB_ENEMY(
19             "evil1.png",
20             1f,
21             0.9f,
22             0.5f,
23             0,
24             BulletDescription.BIG_BULLETS,
25             1,
26             false
27     ),
28     HEAVY_ENEMY(
29             "evil6.png",
30             1.2f,
```

```
            0.9f,
            0.5f,
            60,
            BulletDescription.MISSILES,
            1,
            false
    ),
    FAST_ENEMY(
            "evil6.png",
            1f,
            0.9f,
            5f,
            0,
            BulletDescription.FAST_BULLETS,
            1,
            false
    ),
    BOSS1(
            "Boss2.png",
            1f,
            0.9f,
            3f,
            0,
            BulletDescription.FAST_BULLETS,
            20,
            true
    ),
    BOSS2(
            "Boss3.png",
            0.8f,
            0.6f,
            0.5f,
            0,
            BulletDescription.MOTION_MISSILES,
            30,
            true
    ),
    BOSS3(
            "Boss4.png",
            1f,
            1f,
            0.5f,
            0,
            BulletDescription.MOTION_MISSILES,
            40,
            true
    ),
    BOSS4(
            "Boss5.png",
            1.5f,
            0.6f,
            0.5f,
            0,
            BulletDescription.MOTION_MISSILES,
```

```
85              40,
86              true
87      ),
88      BOSS5(
89              "snilsson.png",
90              1.5f,
91              1f,
92              0.5f,
93              0,
94              BulletDescription.MOTION_MISSILES,
95              50,
96              true
97      ),
98      STANDARD_SHIELD(
99              "shield.png",
100             0.8f,
101             0.8f,
102             2f, // speed
103             60, // accyracy
104             BulletDescription.ENERGY_BLAST, // bullet type
105             2, // health
106             false // is boss
107     ),
108     HEAVY_SHIELD(
109             "shield.png",
110             0.8f,
111             0.8f,
112             0.5f,
113             0,
114             BulletDescription.ENERGY_BLAST,
115             5,
116             false
117     ),
118     BOMB_SHIELD(
119             "shield.png",
120             0.8f,
121             0.8f,
122             0.5f,
123             0,
124             BulletDescription.ENERGY_BLAST,
125             5,
126             false
127     ),
128     BOSS_SHIELD(
129             "shield.png",
130             3.5f,
131             0.8f,
132             0.5f,
133             0,
134             BulletDescription.ENERGY_BLAST,
135             80,
136             true
137     );
138
```

```
139     private String filename;
140     private float scale;
141     private float hitBoxScale;
142     private float speed;
143     private float accuracy;
144     private BulletDescription bulletType;
145     private int health;
146     private boolean isBoss;
147 
148     /**
149      * Create a description of an enemy type
150      *
151      * @param filename     The file name of the sprite for the
           enemy
152      * @param scale        A float specifying the scale of the
           sprite
153      * @param hitBoxScale A float specifying how the hitbox
           should be scaled(compared to the scaled sprite
154      * @param speed        The speed if the enemy
155      * @param accuracy     How accurate the enemys aiming should
           be, 0 is perfect and 360 is the worst
156      * @param bulletType  A bullet type
157      * @param health       How much healt the enemy should have
158      * @param isBoss       A boolean stating if the enemy is a
           boss or a normal enemy
159      */
160     private EnemyDescription(String filename, float scale, float
           hitBoxScale,
161                              float speed, float accuracy,
                                     BulletDescription bulletType,
                                     int health, boolean isBoss) {
162         this.filename = filename;
163         this.scale = scale;
164         this.hitBoxScale = hitBoxScale;
165         this.speed = speed;
166         this.accuracy = accuracy;
167         this.bulletType = bulletType;
168         this.health = health;
169         this.isBoss = isBoss;
170     }
171 
172     /**
173      * Get the file name of sprite file
174      *
175      * @return String filename
176      */
177     public String getFilename() {
178         return this.filename;
179     }
180 
181     /**
182      * Get the scale of the enemy
183      *
184      * @return float scale
```

```
185         */
186         public float getScale() {
187             return this.scale;
188         }
189 
190 
191         /**
192          * Get the scale of the hitbox
193          *
194          * @return float hitbox scale
195          */
196         public float getHitBoxScale() {
197             return this.hitBoxScale;
198         }
199 
200         /**
201          * Get the speed
202          *
203          * @return float speed
204          */
205         public float getSpeed() {
206             return this.speed;
207         }
208 
209 
210         /**
211          * Get the enemey accuracy
212          *
213          * @return float accuracy, 0 is perfect and 360 is horrible
214          */
215         public float getAccuracy() {
216             return this.accuracy;
217         }
218 
219         /**
220          * Get the health this enemy type have
221          *
222          * @return int health
223          */
224         public int getHealth() {
225             return this.health;
226         }
227 
228         /**
229          * The type of bullet this enemy type has, is specified in
                 the BulletDescription enum
230          *
231          * @return A BulletDescription
232          */
233         public BulletDescription getBulletType() {
234             return this.bulletType;
235         }
236 
237         /**
```

```
238         * States if the enemy is a boss or not
239         *
240         * @return True if the enemy type is a boss, otherwise false
241         */
242        public boolean isBoss() {
243            return this.isBoss;
244        }
245    }
```

Javaga/core/src/com/me/Javaga/gamestate/levels/BulletDescription.java

```
1   package com.me.Javaga.gamestate.levels;
2
3   import com.me.Javaga.spaceobject.Bullet;
4   import com.me.Javaga.spaceobject.MotionSeeker;
5   import com.me.Javaga.spaceobject.Player;
6
7   /**
8    * This enum class describes all kinds of bullet within the game
9    * Created by Lukas on 2014-05-12.
10   */
11  public enum BulletDescription {
12      BULLETS(
13              "bullet.png", // filename
14              2f,//scale
15              3, // speed
16              2000, // shootlimit in millisecond
17              10000, //life time in millseconds
18              1, // damage
19              false, // indesctructable
20              false // motion seeker
21      ),
22
23      BIG_BULLETS(
24              "bullet.png",
25              4f,
26              1f,
27              3000,
28              7000,
29              1,
30              false,
31              true
32      ),
33      SMALL_BULLETS(
34              "bullet.png",
35              1f,
36              10,
37              200,
38              3000,
39              1,
40              false,
41              false
42      ),
43
```

```
44      MISSILES(
45              "missile.gif",
46              1f,
47              10,
48              1000,
49              10000,
50              10,
51              true,
52              false
53      ),
54
55      HUGE_MISSILES(
56              "missile.gif",
57              1.5f,
58              3,
59              3000,
60              10000,
61              10,
62              true,
63              false
64      ),
65
66      MOTION_MISSILES(
67              "missile.gif",
68              1f,
69              8f,
70              2000,
71              5000,
72              3,
73              true,
74              true
75      ),
76      FAST_BULLETS(
77              "bullet.png",
78              2f,
79              10,
80              20,
81              3000,
82              0.2f,
83              false,
84              false
85      ),
86      ENERGY_BLAST(
87              "energy_blast.png",
88              1f,
89              2f,
90              6000,
91              5000,
92              2f,
93              false,
94              false
95      );
96
97      private String filename;
```

```
98      private float scale;
99      private float speed;
100     private long shootLimit;
101     private long lifeTime;
102     private float damage;
103     private boolean indestructable;
104     private boolean motionSeeker;
105
106     /**
107      * @param filename          the name of the sprite file
108      * @param scale             the scale of the sprite
109      * @param speed             the speed of the bullet
110      * @param shootLimit        the time the gun needs to rest, in
             millseconds
111      * @param lifeTime          the time the bullet will be active
             , in milliseconds
112      * @param damage            the amount of damage the bullet
             deals
113      * @param indesctructable if the bullet should continue to
             exist even after it kills an object, set this to true
114      * @param motionSeeker      true if the bullet should follow
             the player
115      */
116     private BulletDescription(String filename, float scale,
            float speed,
117                               long shootLimit, long lifeTime,
                                      float damage, boolean
                                      indesctructable, boolean
                                      motionSeeker) {
118         this.filename = filename;
119         this.scale = scale;
120         this.speed = speed;
121         this.shootLimit = shootLimit;
122         this.lifeTime = lifeTime;
123         this.damage = damage;
124         this.indestructable = indesctructable;
125         this.motionSeeker = motionSeeker;
126     }
127
128     /**
129      * Returns the filename of the sprite
130      *
131      * @return A string containing the path and name to the
             sprite file
132      */
133     public String getFilename() {
134         return this.filename;
135     }
136
137     /**
138      * Returns the scale factor of the sprite
139      * @return A positive float stating how much the sprite is
             scaled
140      */
```

```
141     public float getScale() {
142         return this.scale;
143     }
144 
145     /**
146      * Returns the speed of the bullets movement(in pixels per
            update)
147      * @return a float stating the speed of the bullet
148      */
149     public float getSpeed() {
150         return this.speed;
151     }
152 
153     /**
154      * See how often the bullet can be fired
155      * @return A long stating the time in millseconds
156      */
157     public long getShootLimit() {
158         return this.shootLimit;
159     }
160 
161     /**
162      * See how long the bullet should live after it fired(in
            milliseconds)
163      * @return A long stating the life time(in milliseconds)
164      */
165     public long getLifeTime() {
166         return this.lifeTime;
167     }
168 
169     /**
170      * Get the damage of the bullet
171      * @return A float specifying the damage
172      */
173     public float getDamage() {
174         return this.damage;
175     }
176 
177     /**
178      * Check if the bullet should be destroyed after impact or
            not
179      * @return True if the bullet should not be disposed after
            it killed an enemy, false otherwise
180      */
181     public boolean isIndestructable() {
182         return this.indestructable;
183     }
184 
185     /**
186      * Check if the bullet is a motion seeker type
187      * @return True if the bullettype is a motionseeker, false
            otherwise
188      */
189     public boolean isMotionSeeker() {
```

```
190
191         return this.motionSeeker;
192     }
193
194     /**
195      * The proper way to spawn bullets, should be used instead
            of the bullet constructor
196      *
197      * @param xPos         The start position of the bullet
198      * @param yPos         The start position of the bullet
199      * @param degree       The degree in which to bullet should
            go
200      * @param description The bulletDescription which specifies
            the bullet attributes
201      * @param player       The current player
202      * @return A bullet
203      */
204     public static Bullet spawnBullet(float xPos, float yPos,
205                                      float degree,
206                                      BulletDescription
                                            description, Player
                                            player) {
207         if (description.isMotionSeeker()) {
208             return new MotionSeeker(xPos, yPos, degree,
                    description, player);
209         } else {
210             return new Bullet(xPos, yPos, degree, description);
211         }
212     }
213
214 }
```