# Contents

# 1 README

```
roigreenberg


##################
File Description
##################

SpaceShip.java - implement the masterclass of the spaceships
SpaceShipFactory - convert the user input to spaceships
Aggressive - implement the aggressive spaceships (extend spaceships)
Basher - implement the basher spaceships (extend spaceships)
Drunkard - implement the drunkard spaceships (extend spaceships)
Humen - implement the humen spaceships (extend spaceships)
spacial - implement the special spaceships (extend spaceships)
README - this file

########
Design
########

The design I choose to implement is that:
I implement most of the the code for the spaceships in the master-class SpaceShip
which I choose to make an abstract class, so I could make sure every sub-class
of a ship will have to implement the method 'doAction' which is different for
every kind of spaceship but also the masterclass itself implement most off the
require methods that are mostly the same for all or most af the spaceships.
In addition the spaceship class provide the constructor that give the ship
it initial position

Inherete from the SpaceShip class are oll the ships classes.

All of them are overriding doAction and execute there the method they do in the
game:

Humen class - this class inheret 1.teleport, 2.fire and 3.shield and execute
                them in case the condition are good
        Also it's override the methods 1.getMove, since it's behavior is
                 entirly different from ather ships
        and 2.getImage to display the uniqe image of the user ship.

Aggressive class - this class inheret 1.getMove which it execute exactly as it
                    implement in SpaceShip and
                   2.fire which it execute in case the condition are good.

Basher class - this class inheret 1.getMove which it execute exactly as it
                implement in SpaceShip and
        2.shieldOn which it execute in case the condition are good.

Runner class - this class inheret 1.getMove but unlike the previews ship, it work
                on opposite way meaning it run away instead of casing. to do so,
                in it constructor it switch between the right and left value and
                then the getMove method will work as needed without need to
                override it.
        2. teleport  which it execute in case the condition are good.

Drunkard class - this class include lots of variable to determine and save it
                randomly behavior.
        this class inheret 1.teleport, 2.fire and 3.shield and execute
                them in case the condition are good.
```

```
        also it's override the methods getMove, since it's move randomly

Special class - this class behevior is to behave the same as the closest ship.
                there for it inheret all of methods and using
        them according to the closest ship as describe in the other ship's
                class above.
        since some ships are overriding getMove and some not, I decided
                not to override it but implement it(in case of overriding) as
                part of doAction and not as standalone method.


While I start to wrote the code, I had notice that most of the methods are exactly
the same for all the ship and the diffrances are mostly the term that determine
if to activate the mathods or opposite directions(in case of 'getMove' method)

The advantage in such design is that I had very few code repeatition. I almost
didn't had to override a method(s) for a spesific ship and only execute the method
in the term of every ship. almost every ship class has very short code
The disadvantage may be that most of the code is in the class spaceShip whice
made it very long code.


The Drunkard class-

I decided to implement this ship in that way:
1. The ship behavior will be in "session"s.
2. I reset a counter that count the rounds. and draw a number that will determine
    the length(in rounds) of the next "session"
3. Every "session" I draw if the ship will try to teleport, accelerate, active
    shield and also it draw if to turn or not.
4. Every "session", teleporting, shielding and firing(in case of draw as true)
    will try to execute every square of the length of the session
    (for example, if session length is 100 rounds, and teleport draw as true,
    it will try to teleport after 10 rounds, 20 rounds and so.
    if the length is 300, it will try after 17 rounds, 34 and so on)
5. after the session end(after X rounds when X is the length of the session)
    it will draw again all the parameter above and a new session length

*To make sure every session last a while but not to much, I choose to make 2 constant.
 a starting point(which I choose now as 150) and a range(which I choose to be 200).
 that make sure every session will be between 150 to 350 rounds.

The Special class -

I decided that the spacial ship will act in that way:
every rount it will check what is the type of the closest ship, and will act with
that ship behavior (for example, if it close to the basher, it will act like basher)
in case 2 special ship are closest one to each other, they will try to teleport.


######################
Implementation Issues
######################

when I start writing the special class code, I had difficulte how to "know" what
is the closest ship class so I could behave the same.
After lot of thinking I choose to use the 'toString' method which I originaly made
only for the testers and use Switch-Case to exectue
different method for each situation.
Also in case it close to a Drunkard ship, in order to avoid copying the entire
long code I create a varieble of a Drunkard ship and used
it parameters.

Another minor thing, while writing the javadoc, I didn't know how to tag the
"override" when needed, so I decided not to tag it so there want be any error
while creating the javadoc.
```

```
128  The testers -
129  I wrote 2 file of tests. One to the Factory and second for the ships.
130  It under the name of HumenTest but it test thing that are exactky the sa
131  every other ship.
132
133  the 3rd file, Ex3Suite.java is a file that run both tests file.
```

-7/-10 You didn't submit any unit testers. (code='no_junits')

# 2 Aggressive.java

```java
/**
 * This class implement the Aggresive spaceship.
 * This ship available to execute 2 actions
 * The actions are:
 * 1. Move(run to closest ship) 2. fire

 * @author Roi Greenberg
 */
public class Aggressive extends SpaceShip {


    private static final boolean ACCELARATE = true;

    /**
     * Does the actions of this ship for this round.
     * This is called once per round by the SpaceWars game driver.
     * Thde actions that tried to execute for this ship are
     * 1. Move 2. fire
     * Also it regenerate unit of energy at the end of the turn
     *
     * @param game the game object to which this ship belongs.
     *
     * override doAction in class SpaceShip
     */
    @Override
    public void doAction(SpaceWars game) {

        getMove(game, ACCELARATE);

        if (this.getPhysics().distanceFrom(this.closestShip(game))<0.2){
                this.fire(game);
        }
        this.fireCounter++;

        regeneration();
    }

    /**
     * this method is used only testing and for the special ship
     * @return name the type of this ship
     */
    @Override public String toString(){
        String name = "Aggressive";
        return name;
    }
}
```

# 3 Basher.java

```java
/**
 * This class implement the Basher spaceship.
 * This ship available to execute 2 actions
 * The actions are:
 * 1. Move(run to closest ship) 2. activate shield

 * @author Roi Greenberg
 */
public class Basher extends SpaceShip {


        private static final boolean ACCELARATE = true;
    /**
         * Does the actions of this ship for this round.
         * This is called once per round by the SpaceWars game driver.
         * THe actions that tried to execute for this ship are
         * 1. Move 2. activate shield
         * Also it regenerate unit of energy at the end of the turn
         * @param game the game object to which this ship belongs.
         *
         * override doAction in class SpaceShip
         */
        @Override
    public void doAction(SpaceWars game) {
        this.shieldState = false;

        getMove(game, ACCELARATE);

        if (this.getPhysics().distanceFrom(this.closestShip(game))<0.2){
            this.shieldOn();
        }
        regeneration();
    }

        /**
         * this method is used only testing and for the special ship
         * @return name the type of this ship
         */
        @Override public String toString(){
        String name = "Basher";
        return name;
    }
}
```

# 4 Drunkard.java

```java
import java.util.Random;
//import org.omg.PortableInterceptor.SYSTEM_EXCEPTION;

/**
 * This class implement a drunkard spaceship.
 * This ship tring to execute all 4 actions available in a random way
 * The actions are:
 * 1. Teleport 2. Move 3. activate shield 4. fire

 * @author Roi Greenberg
 */
public class Drunkard extends SpaceShip {

    //determine the range of the frequency of the changing
    private static final int CHANGE_RANGE = 200;
    private static final int CHANGE_START = 150;
    //determine the range of the options to turn (-1,0,1)
    private static final int TURN_RANGE = 3;
    private static final int TURN_FIX = 1;
    private final Random randomTurn = new Random();
    private int changeMoveCounter;
    private int change;
    private int turn;
    private boolean acceleration;
    private boolean teleport;
    private boolean shield;
    private boolean fire;

    /**
     * Construct a new drunkard ship.
     * in addition to what it inhirate from the SpaceShip class the ship
     * determine for the first time the action that will happend randomaly
     * by that algorthm:
     *  it draw a random number (named 'change') in the range determind above
     *  also it will reset a counter, and choose where to turn, and if to
     *  accelerate, try to teleport, fire or rise a shield.
     */
    public Drunkard() {
        Random randomTurn = new Random();
        this.changeMoveCounter = 0;
        this.change = this.randomTurn.nextInt(CHANGE_RANGE)+CHANGE_START;
        this.turn = this.randomTurn.nextInt(TURN_RANGE) - TURN_FIX;
        this.acceleration = this.randomTurn.nextBoolean();
        this.teleport = this.randomTurn.nextBoolean();
        this.shield = this.randomTurn.nextBoolean();
        this.fire = this.randomTurn.nextBoolean();

    }

    /**
     * Does the actions of this ship for this round.
     * This is called once per round by the SpaceWars game driver.
     * THe actions that tried to execute for this ship are
     * 1. Teleport 2. Move 3. activate shield 4. fire
     * Every 'change' number of rounds all the parameters are
     *  chanching randomaly.
     * Teleporting, firing and rising a shield are happening every squere
     *  'change' number of rounds iff their state was dtaw 'true'.
     * Also it regenerate unit of energy at the end of the turn  .
```

```
60          *
61          * @param game the game object to which this ship belongs.
62          *
63          * Override doAction in class SpaceShip
64          *
65          */
66         @Override
67         public void doAction(SpaceWars game) {
68
69             this.shieldState=false;
70             if (this.teleport && this.changeMoveCounter%
71                     ((int)Math.floor(Math.sqrt(this.change)))==0) {
72
73                 this.teleport();
74             }
75             getMove(game);
76
77             if (this.shield && this.changeMoveCounter%
78                     ((int)Math.floor(Math.sqrt(this.change)))==0){
79                 this.shieldOn();
80
81             }
82
83             if (this.fire && this.changeMoveCounter%
84                     ((int)Math.floor(Math.sqrt(this.change)))==0){
85                 this.fire(game);
86
87             }
88             this.fireCounter++;
89
90             if (this.changeMoveCounter%this.change==0){
91                 this.change = this.randomTurn.nextInt(CHANGE_RANGE)+CHANGE_START;
92                 this.turn = this.randomTurn.nextInt(TURN_RANGE) - TURN_FIX;
93                 this.acceleration = this.randomTurn.nextBoolean();
94                 this.teleport = this.randomTurn.nextBoolean();
95                 this.shield = this.randomTurn.nextBoolean();
96                 this.fire = this.randomTurn.nextBoolean();
97
98             }
99
100            this.changeMoveCounter++;
101            regeneration();
102
103
104        }
105
106        /**
107         * moving the ship in this round.
108         * This is called once per round by doAction.
109         * the ship will move according to what draw randomaly this time.
110         *
111         * @param game the game object to which this ship belongs.
112         *
113         * override getMove in class SpaceShip
114         */
115        private void getMove(SpaceWars game){
116            this.getPhysics().move(this.acceleration, this.turn);
117        }
118
119
120    //     all the code below is uses for testing and/or for the special ship
121
122
123        /**
124         * determine if the ship is trying to fire
125         * uses for the special ships only
126         * @return if the ship trying to fire
127         */
```

```
128     public boolean toTeleport(){
129         if (this.teleport && this.changeMoveCounter%
130                 ((int)Math.floor(Math.sqrt(this.change)))==0){
131             return true;
132         }
133         return false;
134     }
135     /**
136      * return the current turn draw
137      * uses for the special ships only
138      * @return the current turn draw
139      */
140     public int turn(){
141         return this.turn;
142     }
143     /**
144      * return the current accelerate draw
145      * uses for the special ships only
146      * @return the current accelerate draw
147      */
148     public boolean accelerate(){
149         return this.acceleration;
150     }
151     /**
152      * determine if the ship is trying to fire
153      * uses for the special ships only
154      * @return if the ship trying to fire
155      */
156     public boolean toFire(){
157         if (this.fire && this.changeMoveCounter%
158                 ((int)Math.floor(Math.sqrt(this.change)))==0){
159             return true;
160         }
161         return false;
162     }
163     /**
164      * determine if the ship is trying to rise the shield
165      * uses for the special ships only
166      * @return if the ship trying to rise the shield
167      */
168     public boolean toShield(){
169         if (this.shield && this.changeMoveCounter%
170                 ((int)Math.floor(Math.sqrt(this.change)))==0){
171             return true;
172         }
173         return false;
174     }
175     /**
176      * this method is used only testing and for the special ship
177      * @return name the type of this ship
178      */
179     @Override public String toString(){
180         String name = "Drunkard";
181         return name;
182     }
183 }
```

# 5 Humen.java

```java
import java.awt.Image;


import oop.ex3.*;

/**
 * This class implement the user spaceship.
 * This ship available to execute all 4 actions according to user choose
 * The actions are:
 * 1. Teleport 2. Move 3. activate shield 4. fire

 * @author Roi Greenberg
 */
public class Humen extends SpaceShip {


    /**
     * Does the actions of this ship for this round.
     * This is called once per round by the SpaceWars game driver.
     * THe actions that tried to execute for this ship are
     * 1. Teleport 2. Move 3. activate shield 4. fire
     * Also it regenerate unit of energy at the end of the turn
     *
     * @param game the game object to which this ship belongs.
     *
     * override doAction in class SpaceShip
     */
    @Override
    public void doAction(SpaceWars game) {
        this.shieldState=false;
        if (game.getGUI().isTeleportPressed()==true) {
                this.teleport();
        }
        getMove(game);

        if (game.getGUI().isShieldsPressed()){
                this.shieldOn();
        }

        if (game.getGUI().isShotPressed()){
                this.fire(game);
        }
        this.fireCounter++;

        regeneration();

    }

    /**
     * moving the ship in this round.
     * This is called once per round by doAction.
     * the ship will move according to user choose
     *
     * @param game the game object to which this ship belongs.
     *
     * override getMove in class SpaceShip
     */
    private void getMove(SpaceWars game){
        int turn = STAY_STRIGHT;
```

```
60          if (game.getGUI().isLeftPressed()==true){
61                  turn = this.left;
62          } else if (game.getGUI().isRightPressed()==true){
63                  turn = this.right;
64          }
65          this.getPhysics().move(game.getGUI().isUpPressed(), turn);
66      }
67      /**
68       * Gets the image of this ship. This method should return the image of the
69       * ship with or without the shield. This will be displayed on the GUI at
70       * the end of the round.
71       *
72       * @return the image of this ship.
73       *
74       * override getImage in class SpaceShip
75       */
76      @Override
77      public Image getImage(){
78          if (this.shieldState){
79                  return GameGUI.SPACESHIP_IMAGE_SHIELD;
80          }
81      return GameGUI.SPACESHIP_IMAGE;
82
83
84
85      }
86
87      /**
88       * this method is used only testing and for the special ship
89       * @return name the type of this ship
90       */
91      @Override public String toString(){
92          String name = "humen";
93          return name;
94      }
95
96  }
```

# 6 Runner.java

```java
/**
 * This class implement the Runner spaceship.
 * This ship available to execute 2 actions
 * The actions are:
 * 1. Teleport 2. Move(run away from closest ship)

 * @author Roi Greenberg
 */
public class Runner extends SpaceShip {


    /**
     * Construct a Runner space ship.
     * inhirate from SpaceShip class.
     * also turn the left and right so the ship will run away from closest ship
     */
    public Runner() {
            this.left = -1;
            this.right = 1;
    }

    private static final boolean ACCELARATE = true;

    /**
     * Does the actions of this ship for this round.
     * This is called once per round by the SpaceWars game driver.
     * THe actions that tried to execute for this ship are
     * 1. Teleport 2. Move
     * Also it regenerate unit of energy at the end of the turn
     * @param game the game object to which this ship belongs.
     *
     * override doAction in class SpaceShip
     */
    @Override
    public void doAction(SpaceWars game) {

        if (this.getPhysics().distanceFrom(this.closestShip(game))<0.2 &&
                this.getPhysics().angleTo(this.closestShip(game))<0.2){
            this.teleport();
        }

        getMove(game, ACCELARATE);

        regeneration();


    }

    /**
     * this method is used only testing and for the special ship
     * @return name the type of this ship
     */
    @Override public String toString(){
        String name = "Runner";
        return name;
    }
}
```

# 7 SpaceShip.java

```java
import java.awt.Image;
import oop.ex3.*;

/**
 * The API spaceships need to implement for the SpaceWars game.
 * It is your decision whether SpaceShip.java will be an interface, an abstract class,
 *  a base class for the other spaceships or any other option you will choose.
 *
 * @author oop
 */
public abstract class SpaceShip{

    private static final int START_HEALTH = 20;
    private static final int HEALTH_REDUCE = 1;
    private static final int MAX_ENERGY_LEVEL = 200;
    private static final int ENERGY_SHIELD_REDUCE = 3;
    private static final int ENERGY_FIRE_REDUCE = 20;
    private static final int ENERGY_HIT_REDUCE = 10;
    private static final int ENERGY_TELEPORT_REDUCE = 150;
    private static final int ENERGY_COLISION_RISING = 20;

    private static final int REGENERATE_ENERGY = 1;

    private final int LEFT = 1;
    private final int RIGHT = -1;
    private SpaceShipPhysics position;

    /**
     * all the protected variable as protected because it uses in subclasses
     */

    protected int left = LEFT;
    protected int right = RIGHT;
    protected int maxEnergy=MAX_ENERGY_LEVEL;
    protected int currentEnergy = maxEnergy;
    protected int health = START_HEALTH;
    protected int fireCounter = FIRE_LIMIT;
    protected final int STAY_STRIGHT = 0;
    protected static final int FIRE_LIMIT = 8;
    protected boolean shieldState = false;


    /**
     * Construct a spaceship.
     * create the physics object for the spaceship.
     */
    protected SpaceShip(){
            this.position = new SpaceShipPhysics();
    }

    /**
     * Does the actions of this ship for this round.
     * This is called once per round by the SpaceWars game driver.
     *
     * @param game the game object to which this ship belongs.
     */
    abstract void doAction(SpaceWars game);

    /**
```

```
60          * Gets the physics object of the closest ship
61          * @param game the game object.
62          * @return the physics object of the closest ship
63          */
64         protected SpaceShipPhysics closestShip(SpaceWars game){
65             return game.getClosestShipTo(this).getPhysics();
66         }
67
68         /**
69          * moving the ship in this round.
70          * This is called once per round by doAction.
71          * this used for ships that turn according to closest ship angle
72          * @param game the game object to which this ship belongs.
73          * @param acceleration define if the ship will accelerate
74          */
75         protected void getMove(SpaceWars game, boolean acceleration){
76             int turn = this.right;
77             if (this.getPhysics().angleTo(this.closestShip(game))>0){
78                     turn = this.left;
79             }
80             this.getPhysics().move(acceleration, turn);
81         }
82
83
84         /**
85          * Regenerate the current energy of the ship by REGENERATE_ENERGY
86          * unit(s)
87          */
88         protected void regeneration(){
89             if (currentEnergy < maxEnergy){
90                 currentEnergy+=REGENERATE_ENERGY;
91         }
92         }
93
94         /**
95          * This method is called every time a collision with this ship occurs
96          */
97         public void collidedWithAnotherShip(){
98             if (shieldState == false){
99                 gotHit();
100            } else {
101                maxEnergy+=ENERGY_COLISION_RISING;
102                currentEnergy+=ENERGY_COLISION_RISING;
103            }
104        }
105
106        /**
107         * This method is called whenever a ship has died. It resets the ship's
108         * attributes, and starts it at a new random position.
109         */
110        public void reset(){
111            maxEnergy=MAX_ENERGY_LEVEL;
112            currentEnergy = maxEnergy;
113            health = START_HEALTH;
114            setPhysics(new SpaceShipPhysics());
115        }
116
117        /**
118         * Checks if this ship is dead.
119         *
120         * @return true if the ship is dead. false otherwise.
121         */
122        public boolean isDead() {
123            return health==0;
124        }
125
126        /**
127         * Gets the physics object that controls this ship.
```

```
128         *
129         * @return the physics object that controls the ship.
130         */
131        protected SpaceShipPhysics getPhysics() {
132            return position;
133        }
134        /**
135         * Sets the physics object that controls this ship in case
136         * it need a new position(such as after teleporting or death)
137         *
138         * @param pos new position to set to he ship
139         */
140        protected void setPhysics(SpaceShipPhysics pos) {
141            this.position = pos;
142        }
143
144        /**
145         * This method is called by the SpaceWars game object when ever this ship
146         * gets hit by a shot.
147         */
148        public void gotHit() {
149            if (shieldState == false){
150                this.health-=HEALTH_REDUCE;
151                isDead();
152
153            } else {
154                this.maxEnergy-=ENERGY_HIT_REDUCE;
155                if (this.maxEnergy<0){
156                    this.maxEnergy=0;
157                }
158                if (this.currentEnergy>this.maxEnergy){
159                    this.currentEnergy=this.maxEnergy;
160                }
161
162
163            }
164        }
165
166        /**
167         * Gets the image of this ship. This method should return the image of the
168         * ship with or without the shield. This will be displayed on the GUI at
169         * the end of the round.
170         *
171         * @return the image of this ship.
172         */
173        public Image getImage(){
174            if (this.shieldState){
175                return GameGUI.ENEMY_SPACESHIP_IMAGE_SHIELD;
176            }
177            return GameGUI.ENEMY_SPACESHIP_IMAGE;
178
179        }
180
181        /**
182         * Attempts to fire a shot.
183         *
184         * @param game the game object.
185         */
186        public void fire(SpaceWars game) {
187            if (this.currentEnergy >= ENERGY_FIRE_REDUCE  &&
188                this.fireCounter>=FIRE_LIMIT){
189                this.currentEnergy -= ENERGY_FIRE_REDUCE;
190                game.addShot(this.position);
191                this.fireCounter=0;
192
193            }
194
195
```

```
196         }
197
198         /**
199          * Attempts to turn on the shield.
200          */
201         public void shieldOn() {
202             if (this.currentEnergy > ENERGY_SHIELD_REDUCE){
203                 this.currentEnergy -= ENERGY_SHIELD_REDUCE;
204                 this.shieldState = true;
205             }
206
207
208         }
209
210         /**
211          * Attempts to teleport.
212          */
213         public void teleport() {
214             if (this.currentEnergy>=ENERGY_TELEPORT_REDUCE) {
215                 setPhysics(new SpaceShipPhysics());
216                 this.currentEnergy-=ENERGY_TELEPORT_REDUCE;
217
218             }
219
220         }
221
222          /**
223          * this method is using as getter for the JUNIT tests only
224          * @return the reduce by hit
225          */
226         public static int getHealthReduce(){
227             return HEALTH_REDUCE;
228         }
229         /**
230          * this method is using as getter for the JUNIT tests only
231          * @return the energy reduce by hit
232          */
233         public static int getEnergyReduce(){
234             return ENERGY_HIT_REDUCE;
235         }
236         /**
237          * this method is using as getter for the JUNIT tests only
238          * @return the energy reduce by hit
239          */
240         public static int getEnergyRising(){
241             return ENERGY_COLISION_RISING;
242         }
243         /**
244          * this method is using as getter for the JUNIT tests only
245          * @return the energy regenerate every turn
246          */
247         public static int getRegenerate(){
248             return REGENERATE_ENERGY;
249         }
250         /**
251          * this method is using as getter for the JUNIT tests only
252          * @return the energy reduce while firing
253          */
254         public static int getEnergyFireReduce(){
255             return ENERGY_FIRE_REDUCE;
256         }
257
258
259     }
```

# 8 SpaceShipFactory.java

```java
/**
 * This class is used by the SpaceWar class to create all the sspaceship
 * objects according to the command line arguments.
 * @author Roi Greenberg
 */
public class SpaceShipFactory {

    private static SpaceShip[] spaceShips;
    /**
     * Used by the SpaceWar class to create all the sspaceship
     * objects according to the command line arguments.
     *
     * @param args array of strings supllied by the user.
     *
     * @return spaceShips array os SpaceShip according to args.
     * @author
     */
    public static SpaceShip[] createSpaceShips(String[] args) {
        spaceShips = new SpaceShip[args.length];
        int index = 0;
        for (String ship: args) {
            switch (ship) {
                case "r": spaceShips[index] = new Runner();
                    index++;
                    break;
                case "h": spaceShips[index] = new Humen();
                    index++;
                    break;
                case "b": spaceShips[index] = new Basher();
                    index++;
                    break;
                case "a": spaceShips[index] = new Aggressive();
                    index++;
                    break;
                case "d": spaceShips[index] = new Drunkard();
                    index++;
                    break;
                case "s": spaceShips[index] = new Special();
                    index++;
                    break;
                }

        }
        return spaceShips;
    }

}
```

# 9 Special.java

```java
/**
 * This class implement the Special spaceship.
 * This ship available to execute all 4 actions
 * The ship specialty is that it allwas act as the closest ship
 *
 * @author Roi Greenberg
 */
public class Special extends SpaceShip {


    private static final boolean ACCELARATE = true;
    // helper ship variable. explaind in doAction
    private final Drunkard demoShip = new Drunkard();

    /**
     * Does the actions of this ship for this round.
     * This is called once per round by the SpaceWars game driver.
     * this ship behevior is to always the same as the closest ship
     * (exept when it closest to another special ship then it will try to
     * teleport while accelereting)
     * in case of being closest to Drunkard ship, the randomaly behavior will
     * be calculated in the demoShip, Drunkard class ship that created only for
     * this and then be used by the ship itself.
     *
     * see all other classes for explanation about their doAction
     *
     * @param game the game object to which this ship belongs.
     *
     *
     *
     * override doAction in class SpaceShip
     */
    @Override
    public void doAction(SpaceWars game){

            String ship = game.getClosestShipTo(this).toString();

            switch (ship) {

                case "Runner":
                    this.left = -1;
                    this.right = 1;
                    if (this.getPhysics().distanceFrom(this.closestShip(game))
                            <0.2 &&
                        this.getPhysics().angleTo(this.closestShip(game))<0.2){
                        this.teleport();
                    }

                    getMove(game, ACCELARATE);
                    break;

                case "humen":
                    this.shieldState=false;
                    if (game.getGUI().isTeleportPressed()==true) {
                            this.teleport();
                    }

                    int turn = STAY_STRIGHT;
                    if (game.getGUI().isLeftPressed()==true){
```

```
                turn = this.left;
            } else if (game.getGUI().isRightPressed()==true){
                turn = this.right;
            }
            this.getPhysics().move(game.getGUI().isUpPressed(), turn);

            if (game.getGUI().isShieldsPressed()){
                this.shieldOn();
            }

            if (game.getGUI().isShotPressed()){
                this.fire(game);
            }
            this.fireCounter++;
            break;

        case "Basher":
            this.left = 1;
            this.right = -1;
            this.shieldState = false;

            getMove(game, ACCELARATE);

            if (this.getPhysics().distanceFrom(this.closestShip(game))<0.2){
                this.shieldOn();
            }
            break;

        case "Aggressive":
            this.left = 1;
            this.right = -1;
            getMove(game, ACCELARATE);

            if (this.getPhysics().distanceFrom(this.closestShip(game))<0.2){
                this.fire(game);
            }
            this.fireCounter++;

            break;
        case "Drunkard":


            this.shieldState=false;

            if (demoShip.toTeleport()){
                this.teleport();
            }

            this.getPhysics().move(demoShip.accelerate(),
                    demoShip.turn());

            if (demoShip.toShield()){
                this.shieldOn();
            }

            if (demoShip.toFire()){
                this.fire(game);
            }
            this.fireCounter++;

            break;

        case "Special":

            this.teleport();

            break;

```

```
128                 }
129             regeneration();
130     }
131 
132     /**
133      * this method is used only testing and for the special ship
134      * @return name the type of this ship
135      */
136     @Override public String toString(){
137         String name = "Special";
138         return name;
139     }
140 }
```