ASSIGNMENT 2

COMP-202B, Winter 2008, All Sections

Due: Wednesday, February 6th, 2008 (23:55)

You **MUST** do this assignment individually and, unless otherwise specified, you **MUST** follow all the general instructions and regulations for assignments.

Part 1:	0 points
Part 2, Question 1:	20 points
Part 2, Question 2:	20 points
Part 2, Question 3:	50 points
Part 2, Question 4:	10 points
	100 points total

Preparation

In assignment 2 you will write Java code that will guide a spaceship through an asteroid field. In order to complete the assignment, you have to first download the `Spaceracer` project from the course homepage. We packaged the project in one big `zip` file. Before you can use the project, you must unzip it. On the Macintosh, Windows XP and Windows Vista, unzipping is as simple as double-clicking the `.zip` file. On Unix/Linux machines, the command `unzip Spaceracer.zip` should do the trick. If you are using an older version of Windows, you might want to try downloading 7-Zip (http://www.7-zip.org/).

Compiling / Running Spaceracer Using Dr.Java

To run Spaceracer in Dr.Java, simply launch Dr.Java[1], then choose "Open Project" in the *Project* Menu (*not* the File Menu), and open the file `SpaceRacer.pjt`. You can then compile the project by clicking the *Compile Project* button, and run Spaceracer by clicking *Run Project*.

Compiling / Running Spaceracer from the Command Line

If you don't want to use Dr.Java, you can compile *Spaceracer* from the command line by typing `javac -cp spaceracer.jar ai/StudentAI.java`. To run Spaceracer, type `java -jar spaceracer.jar`.

Playing the Game

The first screen of the game presents you with a menu on which you can choose the game screen size, and if you want to play in full-screen mode or not. We suggest you choose a resolution of 1024 * 768, and the windowed mode (if you use the fullscreen mode you will not be able to see any output you print using `System.out.println(..)`[2]. On the second screen, you can select the race you want to play, and if you want to control the spaceship with the keyboard or with your programmed auto-pilot. For now, choose keyboard control and click *Play*.

[1] In case you are having problems compiling *Spaceracer* on Unix systems, you might want to type `export LD_LIBRARY_PATH=.` on the command line, and then start Dr.Java by typing `drjava &`.

[2] It is possible that the fullscreen mode does not work on Unix machines on which you do not have the privilege to switch screen resolution

You should then see a grey spaceship waiting at a Formula 1-like start line. After a countdown from 4 to 1 the race begins. You can control your spaceship using the arrow keys to accelerate, decelerate (it is impossible to fly backwards), fly up or fly down. The spacebar activates the shields, which make your spaceship invincible for 5 seconds. Once used, the shield takes 30 seconds to recharge. You can hit the *Escape* key any time to quite the game.

Asteroids are the main obstacles on the race track. They do not move. Your ship should not get too close to the center of an asteroid, otherwise it will explode. But there are not only bad things to be found on the track. *Power-ups* are energy sources that, upon touch, will recharge your shields instantaneously.

What You Need To Know To Start Programming

Spaceracer Environment

Although the visual representation of *Spaceracer* is done using 3D graphics, the race takes place in a plane, i.e. navigation takes place in two dimensions only. The spaceship starts off at the start line at position x = 0.0, y = 0.0. The length of the races vary, and hence the finish line can be anywhere between x = 100.0 and x = 2000.0. The height of the race track varies as well, but the center of the track is always at y = 0.0.

The Auto-Pilot

You are to implement an auto-pilot for your spaceship that will steer the ship safely through the race. The *Spaceracer* game is implemented just as many other simulations using the following algorithm:

1. Set up initial game state (i.e. the ships, the asteroids, etc...).
2. Initialize time t to 0.
3. Repeat until the race is over
   (a) Display the current game state at time t on the screen.
   (b) Get commands from the keyboard or from the auto-pilots of the spaceships.
   (c) Calculate the new game state at time $t + \Delta t$ using the current state and the commands.
   (d) Advance time ($t = t + \Delta t$).

The game has already been implemented for you. The only thing that is missing is the code for the auto-pilot. As you can see from the algorithm above, the auto-pilot is asked to provide guidance for the spaceship *over and over again*. After the current game state is displayed, the auto-pilot is asked to give new commands to the spaceship before the time is advanced and the position of the spaceship updated depending on it's speed and heading.

The auto-pilot class is called `StudentAI`. We've already set up the class for you in the folder `ai` under the name `StudentAI.java`. The code of the class `StudentAI` so far looks like this:

```
package ai;
import mvc.Control;
import spaceracer.Asteroid;
import spaceracer.Comet;
import spaceracer.Constants;
import spaceracer.PowerUp;
import spaceracer.Range;
import spaceracer.Spaceship;
import spaceracer.SpaceshipRadar;
import start.Startup;

public class StudentAI implements Control {
```

```
  private Spaceship myShip;
  private SpaceshipRadar myRadar;

  public StudentAI(Spaceship ship) {
     myShip = ship;
     myRadar = myShip.getRadar();
  }

  public static void main(String[] args) {
     Startup.main(args);
  }

  // add variables to remember previous decisions here, if needed

  public void moveShip() {
     // add local variables here

     // add your code here
  }
}
```

Do not worry about the `package` statement, the many `import` statements, the privat `myShip` and `myRadar` fields, the `StudentAI` method and the `main` method. The only place you have to focus on is the `moveShip` method. This is where you have to add the code for your auto-pilot. Do not modify anything else.

Controlling your Spaceship

Your auto-pilot program has to tell your spaceship what to do. This can be done by calling methods. The following methods of the `Spaceship` class are useful for controlling the spaceship:

- `accelerate()`: This method increments the horizontal speed of the spaceship. Once a spaceship has a non-zero speed, it advances even if no other control methods are called. Calling `accelerate` repetitively will speed up the spaceship more and more until it reaches the maximum speed.
- `decelerate()`: This method decrements the horizontal speed of the spaceship. Calling `decelerate` repetitively will slow down the spaceship more and more until it stands still, i.e. its horizontal speed reaches 0.0.
- `moveUp()`: This method instructs the spaceship to move up, i.e. to increment its y position. In contrast to `accelerate`, `moveUp` has to be called repetitively to keep moving upwards. If the auto-pilot stops calling `moveUp`, the spaceship's vertical movement will slow down, and finally the spaceship will remain at its current y position. Since it is impossible to move the spaceship out of the race tracks, calling `moveUp` when the spaceship is already at its top-most y position has no effect.
- `moveUp()`: This method instructs the spaceship to move down, i.e. to decrement its y position. `moveDown` also has to be called repetitively to keep moving downwards. If the auto-pilot stops calling `moveDown`, the spaceship's vertical movement will slow down, and finally the spaceship will remain at its current Y position. Since it is impossible to move the spaceship out of the race tracks, calling `moveDown` when the spaceship is already at its bottom-most y position has no effect.
- `yStop()`: This method instructs the spaceship to stop moving up or down immediately.
- `activateShield()`: This method turns on the shield of the spaceship if the shield generators are fully charged. The spaceship is then invincible. Once the shield is activated is it impossible to turn off. The shield generators can supply power to the shield for approximately 2 seconds, after which the shield turns off automatically, and the shield generator starts recharging. It takes around 10 seconds for the shield generator to recharge fully.

Your auto-pilot has a reference to the spaceship object in the variable `myShip`. Therefore, in order to send commands to the ship, just invoke the corresponding method on the `myShip` object. For instance, to activate the shield of the spaceship, use the statement `myShip.activateShield()`.

Obtaining Information from the Environment

In order to make decisions on how to command your ship, your auto-pilot needs information about the environment. You can get information from the ship itself (such as position, shield level, status, etc...) and from your radar (such as approaching asteroids and power-ups).

The `Spaceship` class provides the following methods that allow you to obtain information about the state of your ship:

- `float getX()`: This method returns a `float` that represents the current x position of your spaceship. Don't forget that the race starts at position x = 0.0.
- `float getY()`: This method returns a `float` that represents the current y position of your spaceship. Don't forget that the race track is centered around y = 0.0.
- `float getSpeedX()`: This method returns a `float` that represents the current x speed of your spaceship.
- `boolean isShieldOn()`: This method returns a `boolean` that tells you if the shield of your spaceship is currently on.
- `boolean isShieldFullyCharged()`: This method returns a `boolean` that tells you if the shield generator of your spaceship is currently fully charged, i.e. if the shield is ready to be used.
- `float shieldCharge()`: This method returns a `float` that represents the time that the shield generator can still supply power to the shield.

Your auto-pilot can call these methods using the `myShip` object. For instance, the statement `if (myShip.isShieldFullyCharged())` allows the auto-pilot to test if the shield is currently ready to be used.

Your ship is equipped with a *radar* that scans the space in front of the ship up to a distance of 20.0 to detect approaching obstacles. The `Radar` class provides several methods that the auto-pilot can use in order to decide how to command the ship:

- `boolean isAsteroidAhead()`: this method returns *true* if there is an asteroid in front of the spaceship. In such a case, if the spaceship continues to advance on the same y coordinate, the ship will ram the asteroid. In order to avoid the asteroid, the spaceship has to either move up or move down.
- `boolean clearSpotsExist()`: this method determines if there are *clear spots* in the asteroid field in front of the ship. A clear spot is a "horizontal line" in which the spaceship can pass without hitting an asteroid.
- `float getClosestClearSpot()`: this method returns the y coordinate of the clear spot that lies closest to the current y position of the ship.
- `float getDistanceToAsteroidAhead()`: this method returns the distance between the spaceship and the asteroid that is in front of the spaceship.
- `boolean CollisionImminent()`: this method returns *true* if a collision is unavoidable.
- `boolean powerUpDetected()`: this method returns *true* if there is a power up within range of the radar.
- `float getClosestPowerUp()`: this method returns the y position of the closest power up within radar range.

To call a method of your radar, use the `myRadar` object. For instance, the call `if (myRadar.isAsteroidAhead())` allows your auto-pilot to test if the ship is currently heading towards an asteroid.

Part 1 (0 points)

Do **NOT** *hand in this part, as it will not be graded. However, doing this exercise might help you to do the second part of the assignment (that will be graded). If you have difficulties with the questions of Part 1, then we suggest you go to one of the office hours; the TA can help you and work with you through the warm-up questions.*

Warm-up Question 1 (0 points)

Run the game, choose the race *Warmup1*, select *Your AI*, and click *Play*. You'll see a countdown and the race begins. Since the code for your auto-pilot's `moveShip()` method is currently empty, your spaceship does not move. After 10 seconds, your ship will explode. This is normal behavior – the race rules dictate that a ship that does not move for more than 10 seconds is eliminated.

Add code to the `moveShip()` method that tells your ship to accelerate. Compile, and run the race again. You should now win the race!

Warm-up Question 2 (0 points)

Try running your current auto-pilot on the *Warmup2* race. Unfortunately this race contains an asteroid blocking the middle of the track, and your current auto-pilot does not know how to avoid it: your spaceship will ram the asteroid.

Modify the auto-pilot to use the radar to detect the asteroid and avoid it.

Part 2 (20 + 20 + 50 + 10 = 100 points)

The questions in this part of the assignment will be graded.

Question 1 (20 points)

Explore the race *Question1* by using the keyboard-controlled spaceship. Figure out a way to complete this race. Now program your auto-pilot accordingly.

Once you are happy with your auto-pilot, *copy* the file `StudentAI.java` and rename the copy `StudentAIQuestion1.java`. Submit this file when you submit your assignment on WebCT as an answer for question 1. Do not use the "Save as" feature of Dr.Java (or any other editor you are using). Copy the file externally, because you are going to continue modifying this file for question 2, and you don't want to loose your solution for question 1!

Question 2 (20 points)

Explore the race *Question2* using the keyboard-controlled spaceship. Figure out a way to complete this race. Now program your auto-pilot accordingly.

Note that you are not allowed to just use the algorithm of the auto-pilot of question 3 to win this race. You have to write your own auto-pilot in order to get the points for this question. You can solve this question by only using the methods `isAsteroidAhead`, `moveUp`, `moveDown`, `yStop` and `accelerate`, and by defining "variables" that you can use to remember the *previous* decision of the auto-pilot. These variables have to be defined *above* the `moveShip` method, below the line marked with the comment "`// add variables to remember previous decisions here, if needed`". Variables that are placed within the `moveShip` method will not keep their values from one execution of the `moveShip` method to the next one.

Once you are happy with your auto-pilot, copy the file `StudentAI.java` and rename the copy `StudentAIQuestion2.java`. Submit this file when you submit your assignment on WebCT as an answer for question 2.

Question 3 (50 points)

In this question you are to implement a more elaborate auto-pilot. The algorithm you are supposed to implement is depicted in the flow diagram shown in Fig. 1.

We have defined some constants for you: `Constants.PLAYER_MAX_SPEED_X` is the fastest speed the spaceship can travel at. `Constants.PLAYER_MIN_SPEED_X` is the slowest speed the spaceship can travel at.

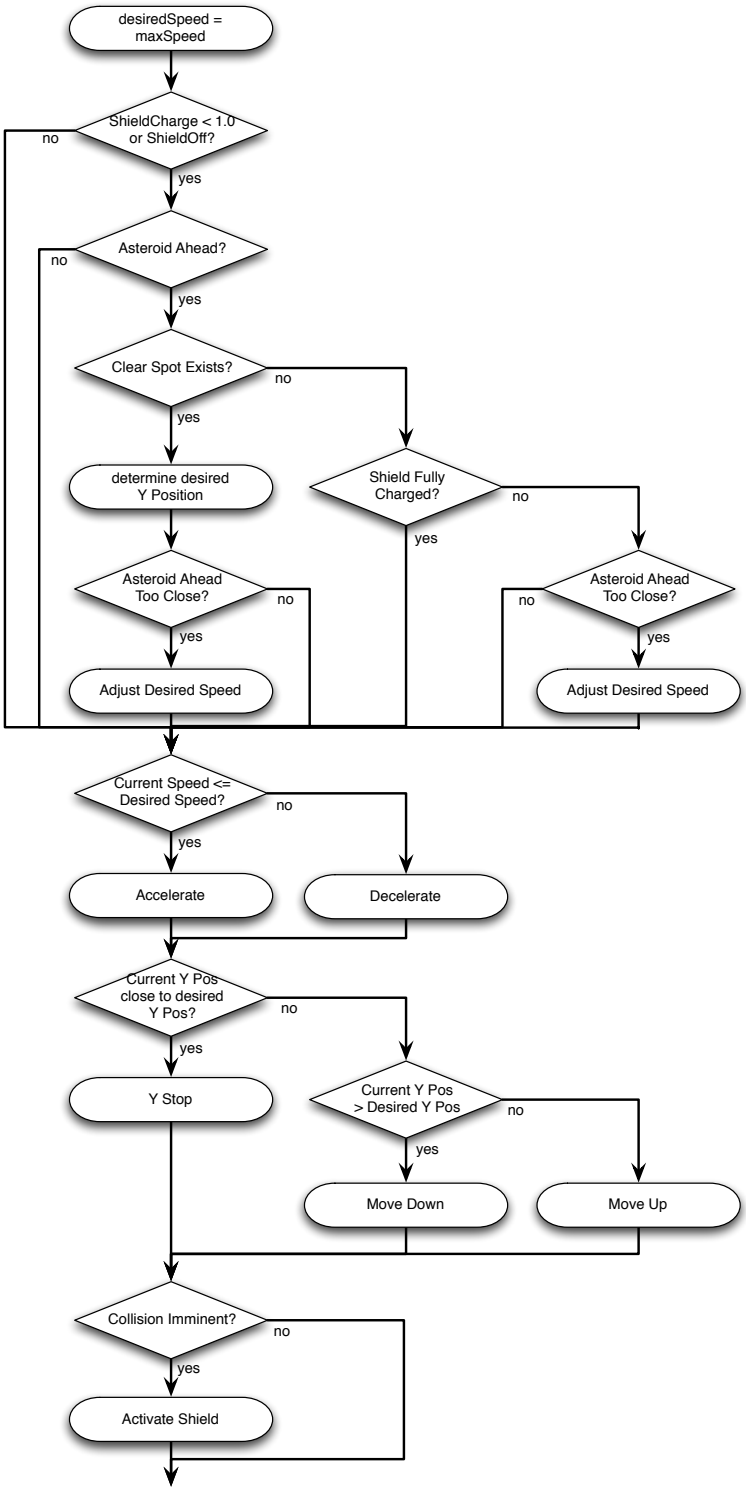

Figure 1: Flowchart of Auto-Pilot Algorithm

`Constants.SLOW_DOWN_DISTANCE` determines at what distance the ship should start slowing down when heading towards an asteroid. `Constants.RADAR_RANGE` determines the maximum range of the radar.

In case of an asteroid close ahead, adjust the desired speed of the spaceship using the following equation:

```
Desired_Speed = Distance_To_Asteroid / Constants.RADAR_RANGE *
                Constants.PLAYER_MAX_SPEED_X
```

Try your auto-pilot on the races *Question3_1*, *Question3_2* and *Question3_3*. Your auto-pilot should be able to safely steer through all the races.

Once you are happy with your auto-pilot, copy the file `StudentAI.java` and rename the copy `StudentAIQuestion3.java`. Submit this file when you submit your assignment on WebCT as an answer for question 3.

Question 4 (10 points)

Try your auto-pilot on the race *Question4*. Is it possible to modify the algorithm of the auto-pilot of question 3 in such a way that the spaceship can finish the race? You don't have to write a program to answer this question. Just simply describe how to you would modify the flow diagram of the previous auto-pilot in a paragraph[1].

[1] Of course you can try to modify the auto-pilot accordingly to see if your solution would works, but you don't have to submit your code.