Welcome to CodeInvaders!

CodeInvaders gives you the opportunity to pit your Java programming skills against other teams in a world of space conquest. Each team will write a MySpaceShip Java class which represents a space ship. Your space ship (class) will be placed in a simulated battle along with space ships from other teams.

CodeInvaders pits space ships against each other in a series of matches. A match consists of up to six space ships competing with each other. Each space ship starts a match at their home planet which is located at a random position in a finite two-dimensional space. Space ships can fire weapons, engage shields to protect themselves, and move by activating their thrusters. Each space ship also has control over several smaller unmanned ships called drones that can be sent out on independent missions under control of their space ship. For the remainder of this guide the term "space ship" will refer to theMySpaceShip class that teams implement, "drone" will refer to unmanned drone ships, and "ship" will refer to either kind of ship.

Entering the UPdateCode Battle

Entering the code battle requires each team to pay a fee of €3. The tournament has a minimum of 3 teams up to a maximum of 16 teams of 3(or less) players each.

The following **rewards** will be presented to the teamsthat end up in 1st, 2nd or 3rd place, respectively:

- €150 FNAC Voucher
- €75 FNAC Voucher
- €30FNAC Voucher

General Information, Caveats, Constraints, and Restrictions

- The Java JRE being use for CodeInvaders is version 1.5.
- Ships may not define constructors.
- Ships may not include static initialization blocks in the class header or call API methods during variable initialization in the class header.
- Ships may not create their own threads, processes, print jobs, files, directly use JNI, AWT or Swing visual components, or other similar system functions.
- Ships may use System.out.println() to display information on the Eclipse console, but the time it takes to do this is charged against the ships move() time limit. (Actions like this are time-consuming.)
- You may run the simulation in Java debug mode. However, understand that when running either the private or public worlds in debug mode, all move and initialization time limits are automatically turned off to allow for breakpoints. If your moves are taking a long time, you should test your Ships in run mode to ensure that it is not going over the time limits.
- You may not talk to anyone other than your own team members once the coding phase starts.
- Teams that use inappropriate names may be disqualified.
- Teams will have access to 2 computers.
- Teams will be allowed to code their ships for 4h30min starting at 1:30pm.

- Any player that submits a ship containing code deemed to be intentionally designed to damage the CodeInvaders environment will be disqualified.
- If anything that which is not portrayed in this documentation, happens during the Code Battle, NuCC supervisors will have the final decision on all matters.

Coding Your SpaceShip – Overview

Before you begin coding, you should try to understand the code structure that your space ship must implement and discuss the strategy your space ship will use during the game. It is permissible for you to openly discuss code structure and strategy during this time, even with other teams.

The coding phase of CodeInvaders starts after the initial presentation. You will use the Eclipse development environment to create and test your space ship. You must use Eclipse to develop your space ship code, since the simulated world runs under Eclipse. During the coding phase you may not talk to other teams.

You can test your space ship during the coding phase in two ways: privately or publicly. For a private test, you run your space ship in a match against a collection of sample space ships with various capabilities that are available on your computer. Only you will be able to see the results of running your space ship in a private test.

For a public test, you must submit your space ship to the public CodeInvaders world. When you submit your space ship to the public world, you will obtain a snapshot on your computer of all other space ships that have been submitted up to that point in time. You will be able to run a public version of the CodeInvaders world to watch the performance of your space ship against space ships that other teams have submitted (and vice versa).

The performance of your space ship in the public world during the coding phase is strictly an opportunity for you to see how your space ship performs against other teams' space ships.

It is not a requirement to submit your space ship to the public world during the coding phase. However, all teams must submit their space ships prior to the end of the coding phase to be included in the final tournament.

SpaceShipTournament

The final versions of all space ships (the last submission you make) will compete in a tournament which will take place during the UPdate Programming Tournament during today's event. Space ships will initially be grouped randomly into matches with up to five other space ships. After each round, space ships will be regrouped according to the points they have accumulated, and a new round will begin.

All space ships will compete in three rounds divided in matches of 5 spaceships.Each space ship will start each round with zero points and random regrouping. The team that earns the most points in the final tournament round will be the winner. In the event of a tie the spaceships with the same number of points will be pitted against each other in another match to break the tie.

The remainder of this guide is intended to help you understand how to design and implement your space ship. JavaDoc files are also available describing the classes and interfaces that relate to coding your space ship.

The Eclipse CodeInvaders Environment

When you start Eclipse you will find a skeleton for the class MySpaceShip in the CodeInvaders project.

This is the class that will contain the code making up your space ship. The MySpaceShip class contains stubs for methods required in space ships. Modifying these methods is the primary manner in which you create the "personality" of your space ship. You may add fields and declare additional methods in MySpaceShip, use inner classes, create other Java classes, or make use of other facets of the Java language to further refine your space ship's characteristics.

When you have a version of MySpaceShip that you want to test in the private world, save your code and then double-click on the games.xml file in the Package Explorer or resource Navigator views. This will open up a game information panel where you will enter the name of your spaceship (name) and the name of your team(organization). These names must be no more than 20 characters. Click the save button or use Ctrl-S to save your name information.

You are now ready to compete in a match. In the Matches section of the game information panel, you can select which space ships you want to compete against. Double-click on teams or use the buttons between the two lists to select which teams will compete in a match. Before you have submitted the first time, only your team and the samples provided will be available to run. When you are ready to view the simulation, click Run. To exit the simulation at any time, click the red X in the top right-hand corner of the simulation or press any key.

To submit your MySpaceShip to the public world, save your code and click the Submit button in the game information panel. When you submit your code, an encrypted copy of all other teams' current submissions will be downloaded to your machine. To compare the performance of your MySpaceShip to other teams' submissions, select teams from the opponents list on the left and add additional teams to the match. If you have not yet successfully submitted code, or if you are the first team to submit code, only the samples will be available to test against. Your last submission will be the one used in the final tournament.

CodeInvaders API

The class SpaceShip is the superclass of MySpaceShip. Do not modify anything in SpaceShip. When you run your space ship, it will actually run with a different version of the SpaceShip class from the one you see in your environment. In particular, theSpaceShip class you will see contains some dummy initialization and return value code that will be replaced when you run in the private or public world.

In addition to the previously defined classes, your environment will contain Java interfaces that define the interfaces presented by various components of the game, plus a helper class:

- **IObject.java**: This is the interface of all objects in the simulated world. Every object implements this interface, which declares methods getX() and getY() that return the location

of the object in the world. All coordinates are non-negative values of type double. **IObject** also contains handy methods to return the distance between any two objects and the heading from one object to another.

- **IShip.java**: This interface extends **IObject** and defines the interface of all ships (space ships and drones) that are currently active in the simulation. Ships can fire their thrusters to move through space or rotate. Ships can also gather energy and attack other ships.
- **ISpaceShip.java**: This interface extends **IShip** and defines the interface of all space ships (not including drones) that are in the match. Every space ship implements this interface. Methods are described in further detail below and in the JavaDocs for the CodeInvaders environment.
- **IControlledShip.java**: This interface extends **IShip** and defines the interface for all ships that you can command. Your space ship and your drones all implement this interface, so you can use it as a common interface for helper methods.
- **World.java**:This helper class provides static methods for the CodeInvaders world. You can find out who the other ships are, as well as information about other objects in space such as bullets and energy sources. These methods are described in further detail below and in the JavaDocs for the CodeInvaders environment.
- **IBullet.java**: This is the interface of all bullets flying through space. It allows you to query the position, direction, and range of bullets to determine if they will hit a ship.
- **IPulse.java**: This is the interface of all pulses expanding in space. It allows you to query the position, range, and radius of pulses to determine if they will hit a ship.

The CodeInvaders Simulation

Display

The screen display for each match shows ships, drones, and other space artifacts against a background of stars and planets. The right hand side of the screen shows information about the space ships competing in the match. For each space ship, the information includes its name, team name, current score (points gained so far during the match), and the total units of energy it currently has. There is a clock at the bottom right that makes one complete revolution for each match. To exit the match at any time, click the red X in the upper right hand corner of the screen.

Initialization

When your space ship is placed into the world, the simulator invokes your space ship's initialize() method. Put any initialization code you want to be executed into this method. You may make use of the entire API in your initialize() method except for beaming energy and firing weapons. Be aware that the simulator will provide only 1 second for your initialization code to execute before it begins the game. If your initialization code fails to complete within one second, your space ship will enter the world in an uninitialized state, with unpredictable results.

Moving Your Objects

After the simulator finishes its calls to each space ships initialize() method, the game proceeds through a series of turns. During each turn, the move() method of each MySpaceShip is called to give the player a chance to choose the next action for its space ship and drones. Methods are available on your ship to query its status, on the World class to both query the status of other ships and find the location of other objects (for example, your opponents' space ships or drones).

Your space ship has methods to move itself and its drones. For instance, "setThrust(MAX_THRUST);" makes your space ship accelerate as quickly as possible. For each turn, you should call these methods on your space ship and drones to let them know what to do. For instance, to tell your first drone to rotate clockwise, you would call: getMyDrones()[0].setRotationThrust(MAX_ROTATION_THRUST).

move() has one parameter, which is the time (in milliseconds) that your move() method used the previous time it was called. Each call to move() has a time limit of 500 milliseconds, so this parameter is useful in determining whether your space ship is in danger of exceeding the maximum amount of time. If a space ship's move() method does not return within 500 milliseconds of the time when it is called, its move() method will not be called again for the rest of the match.

CodeInvader Details

A CodeInvaders world is a two-dimensional world of 1000 units in X by 1000 units in Y, with the origin in the top left corner. There is a wall around the outside edge of the world, and objects cannot go beyond the wall. Objects can move freely within the world, unless they bump into another object or hit the wall. Collisions are inelastic.

The figure below describes the world coordinate system. 0 degrees is "north" and angles increase clockwise:
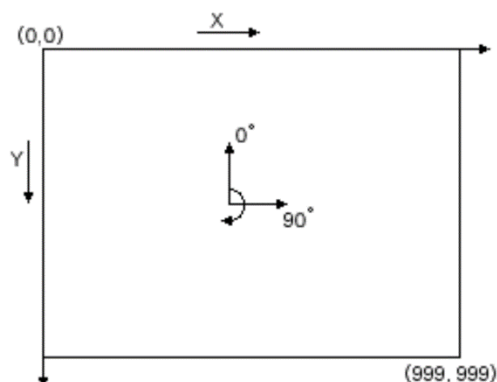
The world has the following characteristics:

- The world is driven by a ticking clock whose value is World.getCurrentTurn(). The total (maximum) number of turns in each match is defined by the World.MAX_TURNS constant.
- Ships can always "see" all other objects in the world. Methods on the World class allow a ship access to information on all objects in the match.
- The stars and planets are background images. Ships do not interact with them except for space ships beaming energy to their home planet.
- Each spaceship starts the match with 500 units of energy and 3 drones, each with 300 units of energy. Space ships can hold a maximum of 1000 units of energy, and drones can hold a maximum of 400 units of energy. If your space ship or any of its drones runs out of energy then it will drift without power (no thrusters, shields, or weapons) unless energy is beamed to it. Your move() method will continue to be called.
- At the start of each match, there are 6 randomly placed energy sources in the world. When a ship has the capacity to hold an additional 200 units of energy and comes within 25 units

- distance of an energy source, it will gain 200 units of energy and the energy source will immediately move to a new random location.
- Ships can raise their shields to protect themselves from opponents' weapons. Once shields are engaged they will stay up for 30 turns. If a request to raise shields is made again during this period, the shields will stay up for 30 turns from the point at which the most recent request was made. Shields use 2 units of energy per turn.
- Ships have low power collision-shields that automatically raise when they are about to collide with another ship, hit the outside barriers, or get hit by a weapon. Collision shields on space ships are circles with a radius of 40 units. Collision shields on drones have a radius of 15 units. Collision shields do not protect against bullets or pulse weapons.
- When two ships collide with each other, the momentum of both will be affected. If one is a drone and the other is not, the drone will absorb most of the impact. Collisions are inelastic.
- Ships can beam (transfer) 5 units of energy per turn to another ship when they are within 200 units distance. Space ships can also beam energy to their home planet to gain points. Energy can be beamed "through" other obstacles. View the JavaDoc for the beam() methods for details.
- If thrust or rotation thrust are not set during a turn, ships will continue moving with the previous settings. Firing weapons, beaming, and shield commands only act for a single turn (and can each be called only once per turn).
- Ships have thrusters which are powered by energy. Main (forward and reverse) thrusters cost 0.002 units of energy per turn per unit of thrust. At maximum thrust, 0.2 units of energy is used per turn. Emergency thrust can be used to accelerate even more quickly. Emergency thrust is equivalent to a thrust setting of 250 units and uses 1.5 units of energy per turn.
- Rotational thrusters use 0.02 units of energy per turn per unit of thrust. At maximum rotational thrust, 0.2 units of energy is used per turn.
- The main thrusters (forward and reverse) accelerate the ship at 0.0075 units per unit of thrust. The rotation thrusters accelerate rotation at 0.0015 degrees per unit of thrust.
- Ships with less than 1 unit of energy are considered "immobile" and opponents will not receive any points for hitting them with weapons. The move() method of these ships will continue to be called.
- Due to interstellar gas, there is 2% friction per turn on movement and rotation.

Weapons

There are two weapons available to ships. Ships can fire only one of the two weapons each turn and must wait for a certain period of time (called reload time) before firing again. Weapons that hit your own ships do not cause any damage or change in points.

The first type of weapon fires bullet directly out of the front of the ship. Bullets travel at 12 units per turn and have a range of 720 units. If you hit a ship with a bullet when its shields are down, you earn points and the opponent's ship will lose 50 units of energy and be thrown slightly off course. Firing a bullet requires 2 units of energy and requires the ship to reload for 6 turns.

The second type of weapon activates an electromagnetic pulse outward in all directions from the ship. Pulses expand at 15 units per turn and have a range of 250 units. They disrupt opponent ships' courses and push them away from the firing ship. If you hit a ship when its shields are down, you earn points and the opponent's ship will lose 25% of its energy. Firing a pulse requires 50 units of energy and requires the ship to reload for 50 turns.

Scoring

Points are earned according to the following table. Remember that it is total points earned that determines which teams advance during elimination rounds.

Action	Points Earned
Successfully hitting an opponent's space ship with a bullet	5
Successfully hitting an opponent's drone with a bullet	2
Successfully hitting an opponent's space ship with a pulse	10
Successfully hitting an opponent's drone with a pulse	5
For each 5 units of energy beamed to your home planet	1
For each 10 units of energy remaining in the space ship at end of match	1
For each 20 units of energy remaining in each drone at end of match	1