Abbildung 1: Connect 4 Gameplay

Abbildung 2: Space Invaders Gameplay

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme

Christoph Daniel Schulze, Sören Domrös, Prof. Dr. Reinhard von Hanxleden, Alexander Schulz-Rosengarten, Steven Smy

Institut für Informatik

1. Übung zur Vorlesung „Programmierung (Projekt)“

The One and Only Assignment Set
Abgabe am Sonntag, 01. März 2020 - 23:55

Aufgabe 1 - The One and Only Assignment 42 Punkte

The Games

In your project, you have a choice between two projects: Connect 4 and Space Invaders.

Connect 4

The game consists of a grid with six rows and seven columns. Two players alternate in dropping their pieces into a column which does not yet contain six pieces. The dropped piece will fall down the column until it either hits the bottom or another piece. The game ends as soon as one player succeeds in building a horizontal, vertical, or diagonal line of four of their pieces, in which case the player wins. If the players manage to completely fill the board without one player winning, the game ends in a draw.

Connect 4 graphics by Silver Spoon. Own work, CC BY-SA 3.0.

Space Invaders

The game consists of alien ships, invading the screen from above, and a player ship at the bottom, trying to shoot at alien ships. The player wins when they succeed at destroying all alien ships by shooting them. The player loses when at least one alien ship touches the bottom of the screen. The player's ship can either be moved horizontally, or can move the angle of its cannon.

Space Invaders graphics. Fair use.

Requirements

We want you to implement one of the two games (implementing both is cool, but won't earn any bonus points). Since Space Invaders will be a lot harder to implement, you should probably default to Connect 4 if unsure.

We expect your implementation to meet the following requirements:

1. Have a proper user interface. Users must be able to play your game, either using the mouse or the keyboard (or both). How exactly that works is up to you. Pro tip: the lighthouse interface also works with game controllers; the Lighthouse API provides a way to work with controller input!
2. Use the Model-View-Controller (MVC) pattern. We expect each of its three parts to be represented by a separate class. It does not need to be perfect (people are fighting intellectual wars over how to apply the MVC pattern, anyway), but think about where each part will be implemented, and why.
3. Support the Lighthouse (more on how to do so below). Your game needs to be playable on your computer, but it must support adding in the Lighthouse as an additional screen which shows what is happening.

Going Beyond the Call of Duty

Here are a few ideas we've had for making the basic games even cooler. Feel free to think about and implement your own ideas. We suggest you only start with these once you have the basic game running, however.

- In general
  - Implement amazing effects. Basically, anything that would look cool on the Lighthouse, so go nuts with explosions and cool effects.
  - If the player is tired and wants to continue the game some other time, they would need a feature that will save and load games.
- Connect 4
  - Add a way to record/play back game sessions, perhaps also storing them in a file to have them available for later. This may be particularly useful if you plan (as you should) to show your game on the real lighthouse, see below. To give everybody a chance to show off their game, each turn will last just a few minutes, so you probably will not have enough time to play a full game manually.
  - Implement an AI player for humans to play against. This is an interesting algorithmic challenge.
  - Introduce options for players to customize how the game is displayed. This may also include changing the number of rows and columns the board consists of.
- Space Invaders
  - Killed invaders could drop loot with weapon upgrades.
  - Special levels could have the player fighting an evil alien boss with additional powers.
  - Have a computer player control the player's ship during menu screens.
  - How about replays...? These are harder to get right than they are for Connect 4.

This is by no means a comprehensive list, so use your imagination!

Implementation Notes

The easiest way for you to solve this assignment may be to keep using the ACM library for all the graphics and for the program structure. Using the library is completely fine. However, there are other, more sophisticated technologies that some of you may already be familiar with. Feel free to run off and do cool stuff with them.

The only hard requirement is that your game is implemented in Java, uses the **MVC pattern**, and supports the **Lighthouse API** (see below).

Whatever you do, use this opportunity to practice everything we've tried to teach you:

- Splitting your code into sensible classes and methods.
- Applying the MVC pattern.
- Properly formatting source code.
- Writing comments!

Getting Started

If you're unsure which project to choose, it's probably best to start with Connect 4. It's a cool game, and since it's event-driven, it is also way easier to implement. If you already feel comfortable as a Java programmer, feel free to try implementing Space Invaders. That, however, will require you to go a bit beyond what you learned in the lecture!

Connect 4 If you implement Connect 4, your code should be event-driven, as introduced in lecture 13. To start with, follow these steps:

1. Think about the MVC pattern. What should be part of the model? What should the views do? Where are the controllers?
2. Start by implementing a model of the board and its pieces.
3. Implement code that checks whether the final state has been reached. Also implement code that computes all movement options for a given piece.
4. Implement the user interface and the Lighthouse view.
5. Optionally think about cool features to improve your game.

Space Invaders If you implement Space Invaders, your code should not be event-driven, but game loop-driven (since aliens will move regardless of whether or not the player does anything). To start with, follow these steps:

1. Think about the MVC pattern. What should be part of the model? What should the views do? Where are the controllers?
2. Start by implementing a model of the player, aliens, and weapons.
3. Add a game loop. A game loop wich checks the state of all input keys, updates the world, and tells the views to redraw themselves.
4. Implement how the world should be updated depending on the input state.
5. Implement the user interface and the Lighthouse view.
6. Optionally think about cool features to improve your game.

The Oral Exam

To be allowed to take our final exam you will need to pass an oral exam in the project's second week (or in its first week if you're starting early; in that case, contact cds to make an appointment for the oral exam!). For that, be prepared to...

- ...explain your code

- ... explain how MVC applies to your code
- ... apply concepts you've learned during the semester to your code

It is okay if your game is not completely finished by the time of the oral exam, but you should have code and, ideally, a basic playable version.

Feel free to submit your project as a solution here in the iLearn system. That, however, is not required to be allowed to take the final exam.

Note: the successful completion of the project is part of the requirements for being admitted to the final exam. Now, we don't want to kick anybody out this late in the semester. Thus, if you make a serious effort in the project and get some reasonable results, you should have no trouble passing the oral exam. However, if your performance in the project and in the exam is so poor that you manage to fail the exam, we will give you a second chance, probably some time later in March. This will be too late to admit you to the first final exam, which takes place basically right after the project (March 3), but you could still be admitted to the second final exam (March 30).

The Lighthouse

Your game should of course provide a regular *high resolution view* to play it on your computer screen. However, your game must be designed such that it can be played on the Lighthouse as well - yes, that is the 15-story university highriser. Besides housing people, it can be thought of as a huge sceen where every physical window is a pixel. In terms of resolution, it is rather limited: only 14 pixels vertically and 28 pixels horizontally. Thus, your game must support a *low resolution view* of 28 x 14 pixels. Yes, that is quite a limitation - but gives you the option to play your game on an enormous display at the end of the project.

Whatever the high riser displays will be attributed to our university. So, be responsible: just show your game, nothing else. No bad language or symbols, no stupid jokes or you will ruin the whole experience for everyone.

Signing Up for an Account

To display stuff on the Lighthouse and, more importantly, to test your game beforehand, you need to sign up for a Lighthouse account. The account provides access to a web-based Lighthouse preview (for testing) as well as to the real thing (during the presentations). Here's what you need to do:

1. Go to this website.
2. Choose a user name and a password.
3. Enter the following course key, which *you are not to give to other people*: [redacted]
4. Click *Register*.

You will see a simulation of the Lighthouse.

Using the Lighthouse API

To be able to use the Lighthouse API, you first need to import a class and several libraries into your project:

1. Download this ZIP archive and extract it somewhere.
2. The archive contains a file named `LighthouseDisplay.java` as well as `ILighthouseInputListener.java`. Import them into your project and place them in the correct package.

3. Eclipse will complain that several classes used by `LighthouseDisplay` cannot be found. This is because you have not imported the necessary libraries yet. Start by adding a folder called `lib` to your project (right-click your project and choose *New*, *Folder*).
4. The archive contains eight JAR files. Import them into your new `lib` folder, either by dragging them from your file explorer to that folder in Eclipse, or by right-clicking your folder, choosing *Import* and using the *File System* import wizard that is part of the *General* category.
5. The required libraries are now part of your project. To get everything working we now need to tell Java that you intend to actually use them as well. Right-click your project and choose *Build Path*, *Configure Build Path*.
6. Similar to what we always did with the ACM library, switch to the *Libraries* tab, click the *Add JARs* button (not *Add External JARs*!) and look for the JAR files. Click *OK* and *Apply and Close*.

Eclipse should now be able to compile `LighthouseDisplay` and `ILighthouseInputListener` properly. To actually send data to the Lighthouse, you can use the following code as an example:

```
LighthouseDisplay display = new LighthouseDisplay("MyUsername", "MyToken");

// Try connecting to the display
try {
  display.connect();
    // Wait until connection is established
    while(!display.isConnected()) {
        Thread.sleep(100);
    }
} catch (Exception e) {
  System.out.println("Connection␣failed:␣" + e.getMessage());
  e.printStackTrace();
}

// Send data to the display
try {
  // This array contains for every window (14 rows, 28 columns) three
  // bytes that define the red, green, and blue component of the color
  // to be shown in that window. See documentation of LighthouseDisplay's
  // send(...) method.
  byte[] data = new byte[14 * 28 * 3];

  // Fill array

  display.send(data);
} catch (IOException e) {
  System.out.println("Connection␣failed:␣" + e.getMessage());
  e.printStackTrace();
}
```

The user name corresponds to the one you registered for your account. For the token you need to log in to your account and select *API-Token anzeigen*. The token is only valid for a certain period of time and you may need to generate a new token at some point.

The Lighthouse API offers another, more advanced feature: it can send keyboard and controller input from the browser to your application. This is advanced stuff that you should only consider if you have finished implementing your game. To receive input, you need to implement the `ILighthouseInputListener` interface and register the listener with the Lighthouse API by calling the `addButtonListener(...)` method. Note that this method is called from another thread, so you will have to ensure that it won't interfere with your main thread. Again, if you have no idea what that means, don't consider it.

The Lighthouse Project Presentation

You will have the chance (but are not required) to demonstrate your game on Wednesday and Thursday evening of the second week of the project (February 26/27 2020, 19:30 - 21:45). The "control tower" will be set

up in the Audimax, on the stairs in front of *Hörsaal K*. This is also a nice way to show off what you have achieved during your first semester to your family and friends, so feel free to invite them over as well. The more, the merrier!

A few impressions from previous years:

- Number 1
- Number 2