Hex

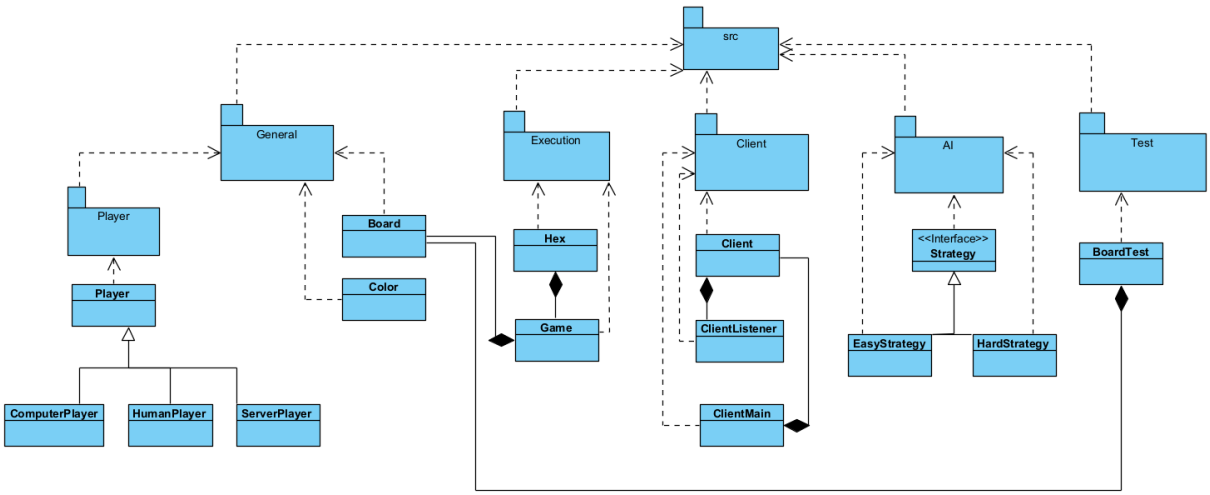
By Anna Smit (Control Group)
S2846012

Design

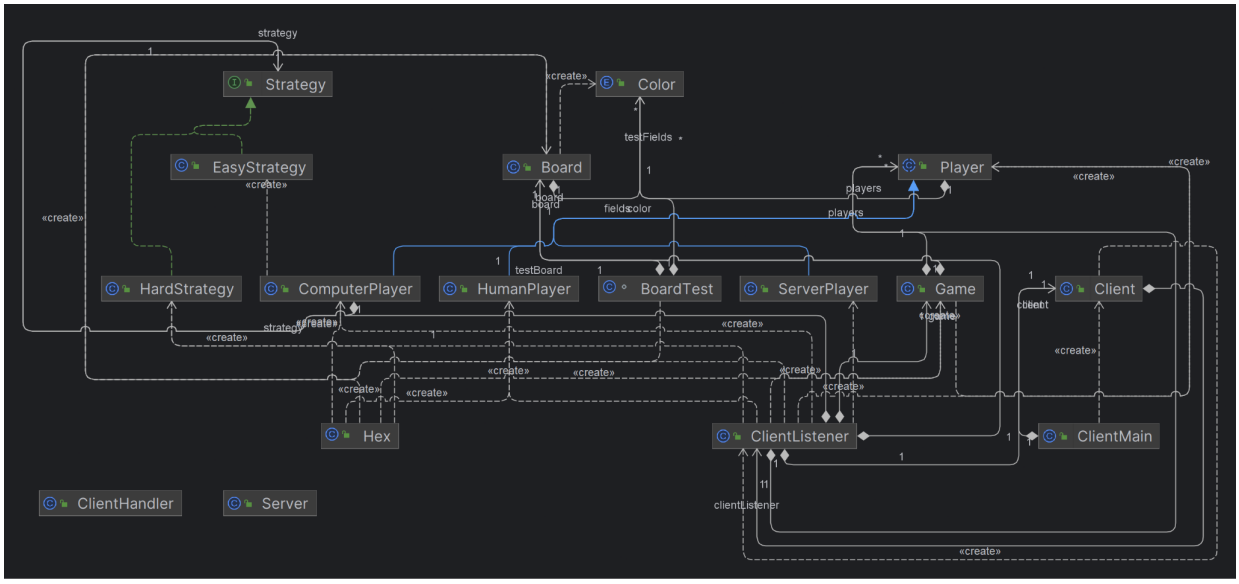
The class diagram below displays the design of this project. Every single row is a different package. I have a package that:

- contains an inner package that contains everything for the player and contains the color and board class. This package implements everything that has to do with the board and players itself. (general package)
- contains all AI strategies for the ComputerPlayer. (ai package)
- contains everything that has to do with playing the actual game locally. (execution package)
- contains everything client related that can connect to another server. (client package)
- contains a tester for the board functionalities.(test package)

The Class Diagram below describes the overall structure of the project and shows the general dependencies of the classes.



Of course, there are more dependencies then shown, the picture of the classes and all their dependencies is shown below.

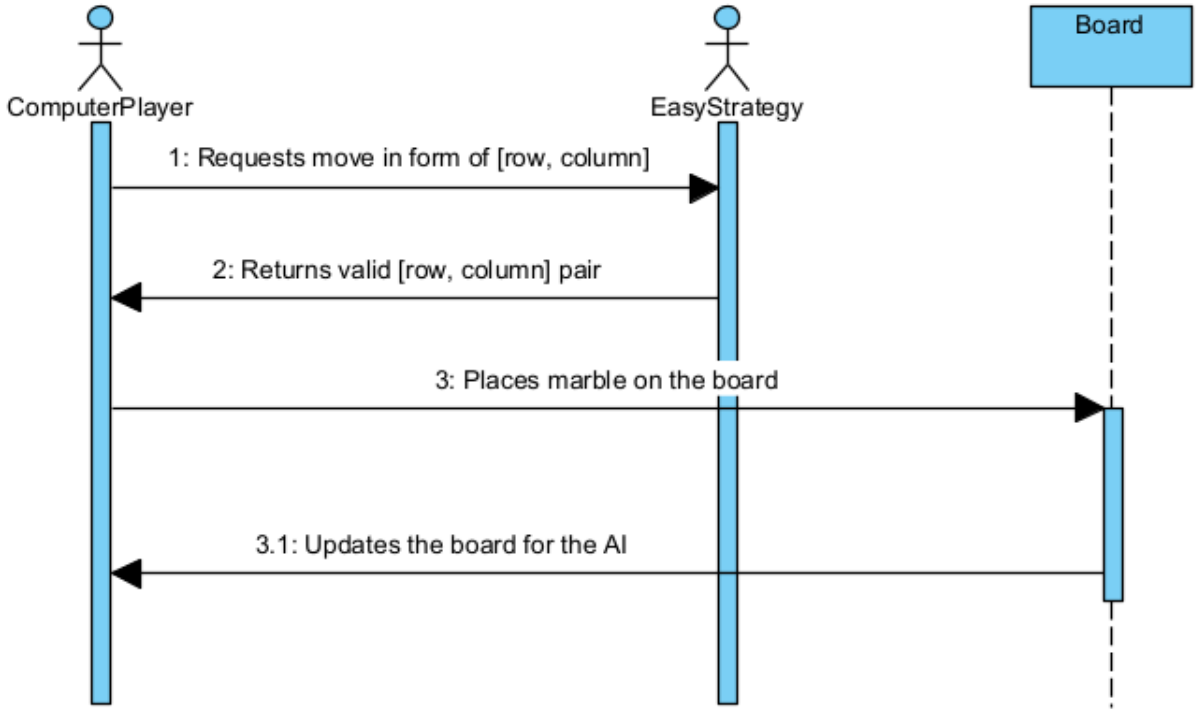


Now I will go over the functionalities of the packages.

The AI package:

- Manage the easy strategy
    - This class just determines the move based on a random integer. It never implements the swap rule.
- Manage the hard strategy class
    - This class has the same methods as the easy strategy, but it always implements the swap rule if allowed.
    -

The sequence diagram below displays how the ComputerPlayer interacts with the strategy and the board. In this case, the ComputerPlayers strategy is the easy strategy. It requests a move from the EasyStrategy, in the form of a [row, column] pair. The EasyStrategy returns a valid field on the board by checking. This means that this field is an existing field on the board and empty. Then the ComputerPlayer puts the marble on the board. The Board then updates and returns the updated version to the ComputerPlayer.



The General package:
This package has another package that contains everything player related.

<u>The player package:</u>

Responsibilities:

- Manage the Player
    - This class manages the general player functions regardless of the type of player. It is the foundation of the Human, Computer and Server player. Hence that these classes use the player class. In addition, since every player has different ways of determining a move, the class itself is abstract.
- Manage the Computer Player
    - Manages creating an AI with the correct strategy based on a certain difficulty. This class gets used when creating an AI in the local game, or using the another server.
- Manage the Server Player
    - This class manages the server player. The class itself looks quite empty, but this class serves mostly as a way to differentiate server moves and pass them correct onto a board when connected to a server.

<u>The other classes:</u>

The other classes in the general package are the Board and the Color class.

Firstly the Color Class.

- Manage the different colors on the board
    - Even though this isn't technically a class, it still plays an important role in the game. It makes sure a game can be filled with a mark of the correct color, and is vital in the algorithm to check if there is a winner.

Secondly, the Board class. This is quite a big class and has many responsibilities

- Manage the game board
- Manage making copies of the board
- Manage checking if there is a winner
- Manage updating the board

This class has a constructor to create a new, empty board of 81 fields. This class does everything board related. Most classes use the Board class either directly, by using everything related to a board, or indirectly, to determine a move on the board.

The Execution package

The execution package manages everything that has to do with playing a local game. The game class implements the final game rules, creates methods for starting or ending a game, and asks questions in the console. Furthermore, the Hex class contains a main, so the user can run the local game.

The responsibilities:

- Manage running the game
- Manage starting the game
- Manage asking questions to the client using the console
- Manage displaying a game over situation

The Hex class is only used when playing a local game, the game class is used to track current players when playing on a server.

The Client package

This class deals with everything related to connecting with an external server to play a game of hex.

- The Client class:
    - Responsibilities:
        - Manages connecting to the server
        - Manages sending messages to the server
        - Manages checking what the server sends back and taking appropriate action accordingly.

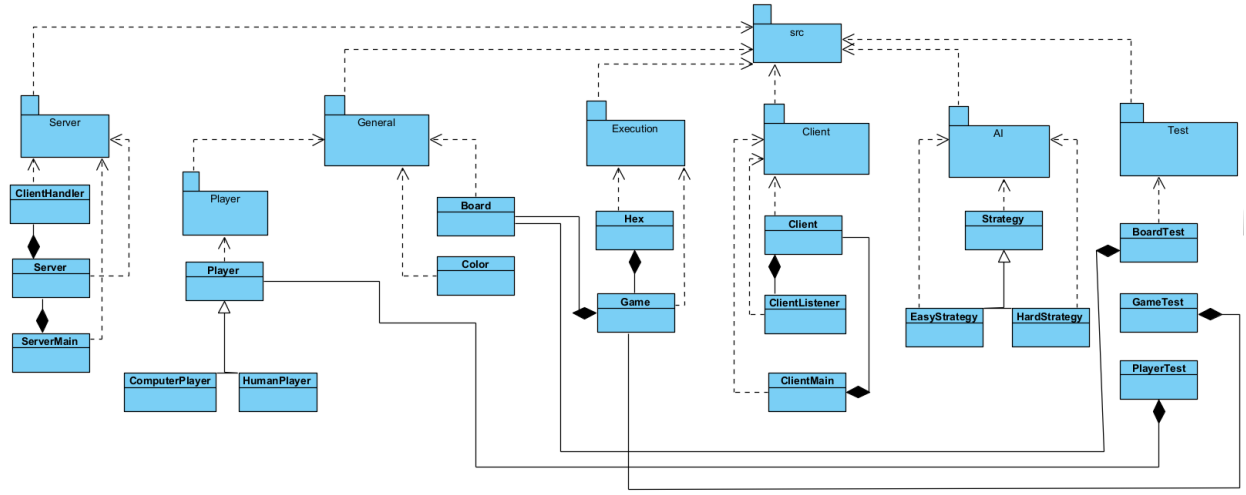
        - This class sends messages according to protocol to the server, reads messages back from the server, and then calls methods from the ClientListener that deals with the messages from the server. There are 2 threads running in this class, the thread of the Client itself and the thread that keeps up to date with what the server sends back.
        - This class is used by the ClientMain, since that creates a new client for every user that logs in. In addition, it is being used by the ClientListener class since for every Client we create a clientlistener.
- The ClientMain class:
    - Responsibility:
        - Manage running the client
    - This class makes sure that the client can input an IP address, port number to connect successfully to the server. This class is used whenever the user wants to play on a server.
- The ClientListener class:
    - Responsibilities:
        - Manages sending messages to the server (Only for the Hello command)
        - Manages the methods that are called based on the commands that the server sends.
    - This class is used only when playing on a server, since this class directly asks input from the user and displays everything sended by a server in the correct format to a user. This class makes sure that the user is always up to date about what is happening in the game.

The Test class
This class only contains one method, since I had no time to implement the others.

Responsibilities:
- Manage testing if the board class correctly can determine a winner.

Reflection

Initial Design



The Class Diagram above was my initial design. It is quite similar to the final design. I only did not manage to implement the entire Server package, and there are a couple test classes missing from the Test package. In addition, while busy creating the Client side of the project, I thought of the ServerPlayer class. In my initial design I did not think of making an entire class for the ServerPlayer, but whilst thinking of how to display the moves from the server on the board I came up with this class.

Summarized pro's and con's of the initial design:

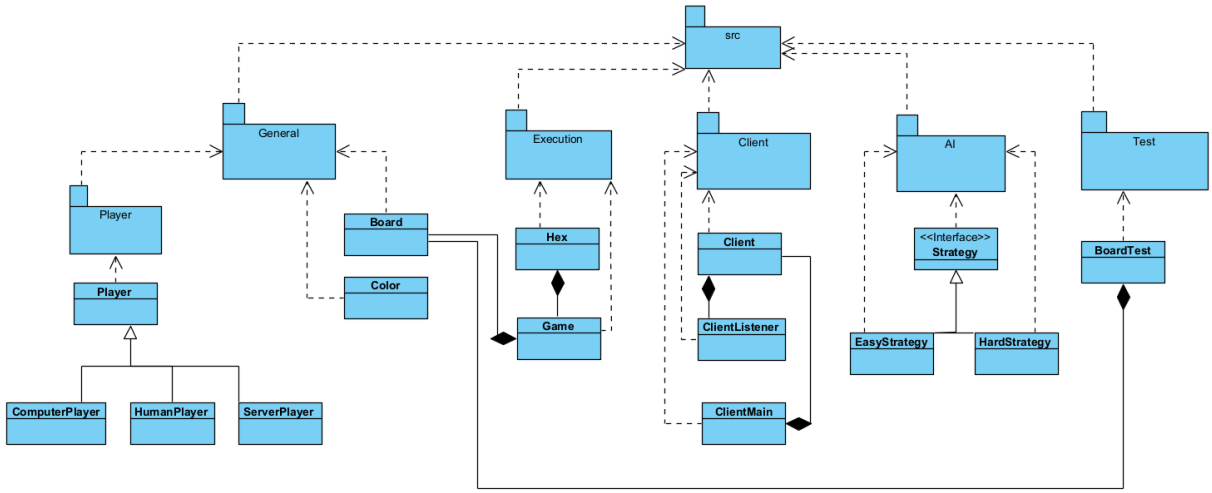
Pro's:

+ All packages and classes are implemented

Con's:

- There is no ServerPlayer meaning you have to work around either the HumanPlayer or the ComputerPlayer's methods for a player to place the move. Of course you could try to bypass it entirely and place it directly onto the board with no players involved. But I found the risk of messing up the entire game logic of the board to significant.

Final Design



Above you will find the class diagram displayed again for sake of ease.

As I mentioned in my initial design, the design is very close to the one I had in mind when I started working on this project. The pro's and con's are thereby also reversed with the one from the initial design:

Pro's:

+ It implements a ServerPlayer class which makes placing and tracking the server's moves on the board easier.

Con's:

- It does not implement the server package at all due to time constraint
- It does not implement all the tests I initially thought of.

Personal reflection

Of course this was all an experiment. I did notice that the coding for me went significantly easier than the first time I did this project. However, there are a lot of things I could not implement. I also do not think there was any place I could have cut time down to implement stuff that I left out. Maybe if I had written some more test classes I would have made debugging easier, thereby faster and thereby I could have implemented more. But that is the only thing I could think of.

I also would like to apologize for whoever is reading this report, since it is written quite fast and I am not sure how many grammatical errors or discrepancies there are. Normally I would have wrote it in a more professional manner but I had only 3 hours for this section of the project

Nevertheless, I made sure that the overall game is playable and I am able to play on a reference server, and in addition I have a semi decent report written down. Overall I am quite happy with what I manage to do.