#Design Report

##TicTacToe   
The TicTacToe challenge is a simple game for two players, X and O, who take turns marking the spaces in a 3x3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. The following example is won by the second player, O:

| O | X | O |

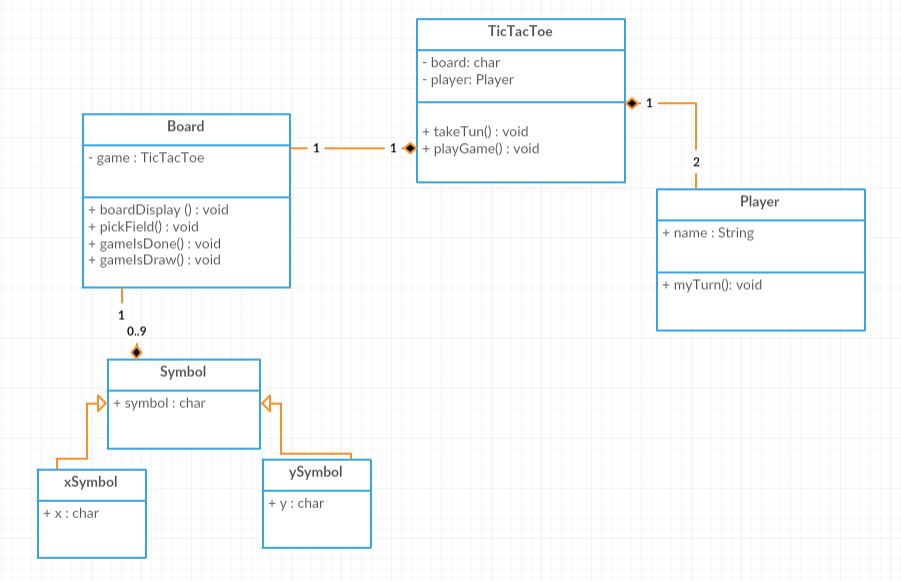
-------------

| X | O | X |

-------------

| X | O | O |

##Initial Design   
We made class diagram for the initial design we had in mind. The class diagram gives a good overview over the game's classes, their attributes, operations and/or methods. It also shows the relationships between the classes and how we decided on the branches to use. When programming the game, we used the class diagram although we knew the diagram would probably change in the process.

[](https://github.com/KontrolAltDelete/TicTacToe/blob/master/docs/images/Tictactoeclassdiagram.JPG)

##Coding Rules   
When deciding on programming rules to follow, we looked for inspiration on Google. We designed our rules so that the code would be easier to read and understand. The following rules are the ones we decided upon and few examples to explain them a little better:

###Brackets  
All brackets should be opened at the end of a statement (class/function) and closed in the beginning of a line.

public int myFunction(){

...

}

###Classes   
Classes should always follow the rule of\_Pascal casing\_ where the first letter of each word is capitalized. Use noun when naming classes.

public class MyClass(){

...

}

###Functions   
Functions should always follow the rule of *Camel casing* where the first letter is in lowercase. Use descriptive names for functions and not too long.

public int myFunction(){

...

}

###Variables and constants   
Variable names start with a lower case and always follow the rule of *Camel casing*. Use nouns when naming variables.

int iCount = 0;

To name constants use uppercase for each word and seperate each pair of words with an underscore.

final int MAX\_COUNT = 10;

###Comments   
All one-line comments starts with "//". Use one-line comments to explain implementation details such as the purpose of specific variables and expressions.   
Explain local variable declaration with and end-line comment.

##Coding Method   
###TDD   
We use *Test Driven Development (TDD)* in our project. It is a software development process that relies on the repetition of a very short development cycle. Requirements are turned into very specific **test cases**, then the software is improved to oass the new tests only. When the code is succesful we can commit the code and then refactor it if neccessary.

###Branches   
We decided to use feature branches for every class in our project and one branch for all documentation.  
Branches make it easier for us to work on seperate parts of the project and keeps us from getting merge conflicts. Branches also make the project easier to read if you go over it on GitHub.

###Markdown syntax and pair programming <br/ > We use *markdown syntax* in our documentation. We have three reports to make our code more clear. They are all stored in a folder called docs on the root of the repository. We mostly use *pair programming* to work on our project and think it is the most efficient way to work in big groups.