

# OXO Assignment Briefing

COMSM0103

Dr Simon Lock & Dr Sion Hannuna

# Assessed Exercise

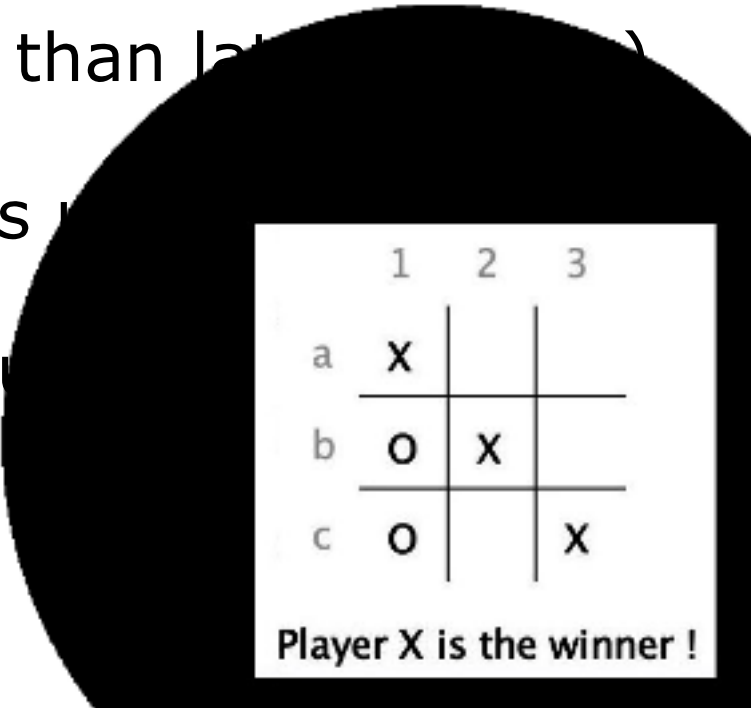
Next pair of workbooks are an ASSESSED exercise !

Aim is to build a noughts-and-crosses (OXO) game

It's a fairly easy activity (easier than last time)

Worth 20% of your mark for this part

Let's take a look at the key features



	1	2	3
a	X		
b	O	X	
c	O		X

Player X is the winner !

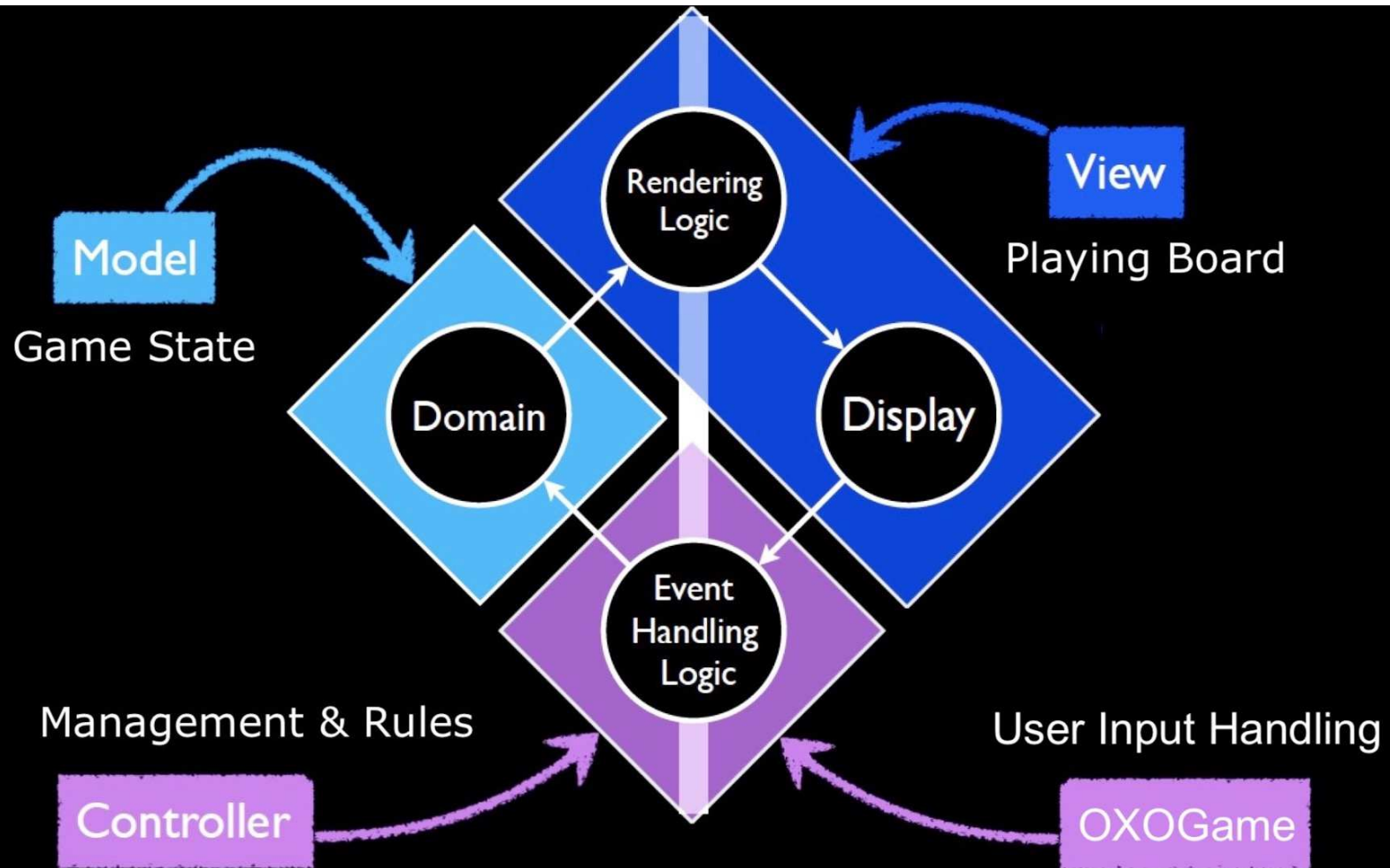
# Model View Controller

In this unit, we will talk a lot about "Design Patterns"  
First pattern we encounter is "Model-View-Controller"  
Useful for structuring interactive GUI applications  
Approach is to split application into 3 components:

- Model: the application core "state" data
- View: the bit that the user sees
- Controller: the bit that makes decisions

This separation makes change and evolution easier

# MVC for OXO



# Fill in the gaps

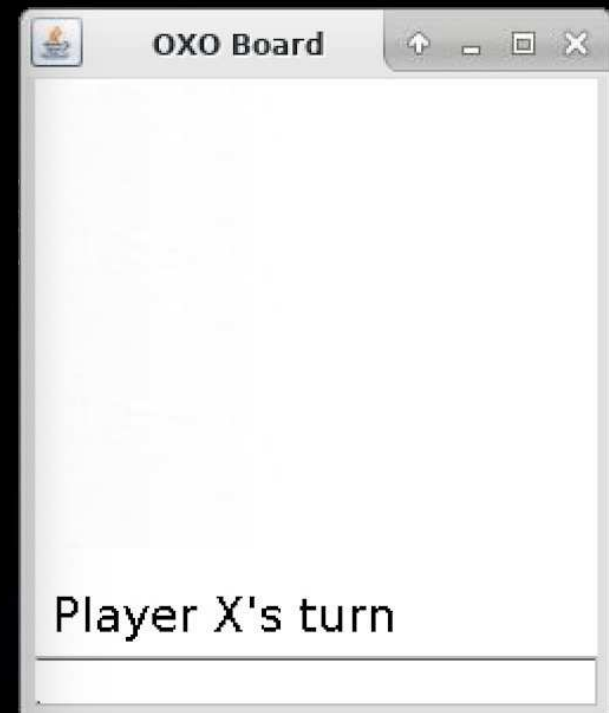
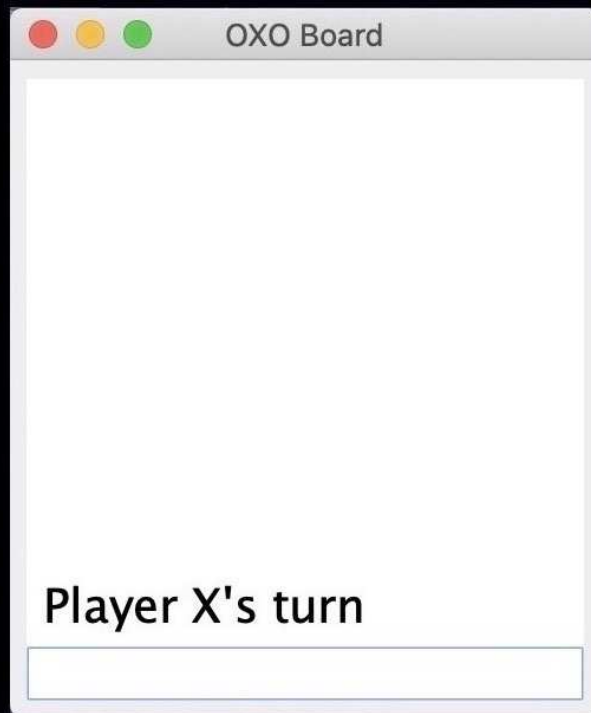
A Maven template project has been provided for you  
Already implemented are:

- OXOGame: Main window (also receives user input)
- OXOView: Provides "Rendering Logic" and "Display"
- OXOModel: Game State from the OXO domain

OXOController is empty: your task is to complete it !  
You must call OXOModel methods to change state  
You don't need to interface directly with OXOView  
(it monitors OXOModel and updates automatically)

# OXOGame

How the main window looks on different platforms



# OXOView

	1	2	3
a	x		
b		o	
c			x

# User Input

Players take it in turns to make a move

Enter cell identifier into the OXOGame GUI window

Consists of row letter and column number (e.g. b2)

Inputted identifier then passed OXOController via:

```
public void handleIncomingCommand(String command)
```

Your task: interpret identifier & update game state

run-demo



# What IS the Model Game State ?

- The number of cells in a row required to win
- The set of players currently playing the game
- The player whose turn it currently is
- The "owner" of each cell in the game grid
- The winner of the game (when the game ends)
- Whether or not the game has been drawn

# Some Key Features

# Dynamic Board Size

It would be nice to alter board size during a game

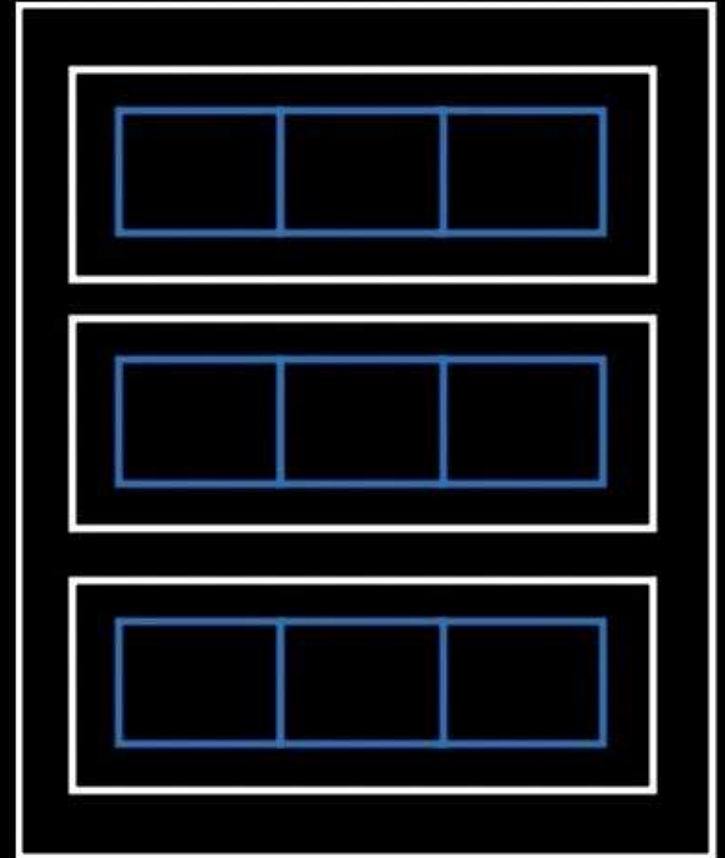
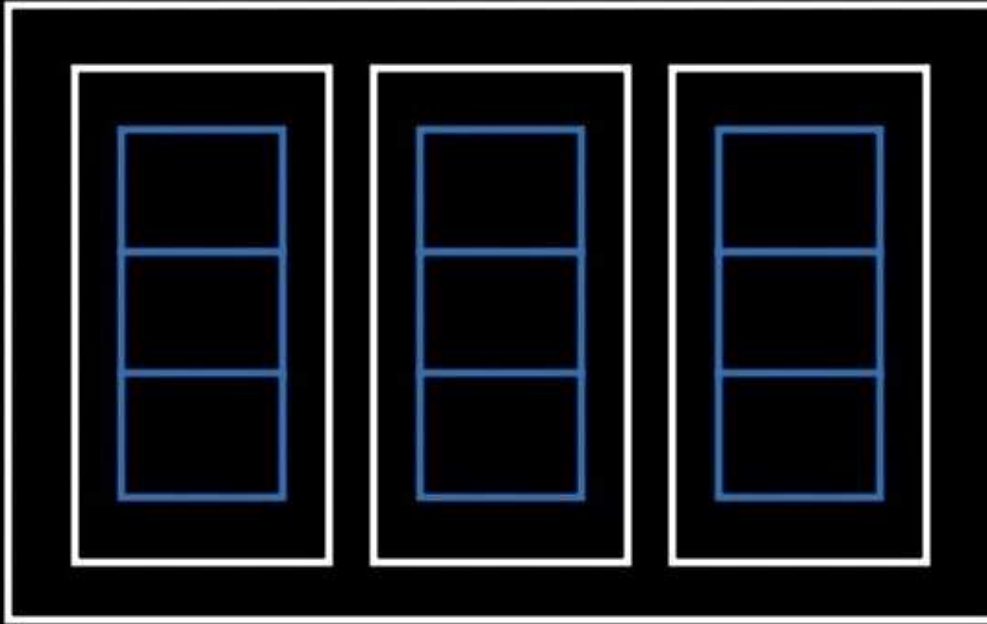
If it became clear a game was going to be a draw

(all blank cells are used but no one has won)

Would be nice to be able to increase the board size  
in order to allow play to continue

This influences data structures used in OXOModel...

# ArrayLists (from "Collections" package)



For a 3x3 game you must instantiating FOUR ArrayLists

# Win Detection

No point playing game if no one actually wins !  
It's Controller's job to detect when a win is achieved  
Must be able to check for wins in all directions  
(horizontally / vertically / diagonally)

Note: win detection must work on grids of ANY size  
(not just the standard 3x3 board !)

# Error Handling

It's likely users will make mistakes during gameplay  
Entering "invalid" cell identifiers into the GUI:

- Invalid Identifier Length: The string is not 2 chars
- Invalid Identifier Character: Row char not a letter or column char not a number digit
- Outside Range: The identifier values out of range (i.e. too big or too small)
- Already Taken: Cell has already been claimed

In Java we handle run-time errors using "Exceptions"  
See workbook for examples of how to use these

# Adjustable Win Threshold

Number of cells in a row required to win (normally 3)  
More interesting if we can change this threshold !

OXOGame allows user to enter desired value  
(by pressing the '+' and '-' keys)

Your controller needs to respond to the input:

- Update the game state in the model
- Apply the current threshold during win detection

# Greater Number of Players

Traditional number of players in an OXO game is 2  
Additional players makes game more interesting !

Add features to support any number of players

Number of players can't be changed using GUI  
This can only be done programatically

A good opportunity for automated testing...



# Automated Testing

Game complex enough to need automated testing  
As "developers" it is your job to write tests scripts  
Focus on testing OXOController (the bit you built)

You have been provided with a skeleton test script  
Populate this with a comprehensive set of test cases

Your tests cases will be assessed during marking !  
Aim is to fully test the functionality you have built

# Warning: There are Lambdas !

Example test script contains some unusual syntax:

```
assert(Exception.class, ()-> sendCommand(), comment);
```

Lambda operator "->" is something

we have not actually covered yet

Operates like an inline function

Allows us to pass CODE "into"  
an already existing method

More on this NEXT week !



# Details of Marking Process

# Choosing Test Cases

To mark your submissions, we need some test cases  
Selecting \*which\* test cases is however very tricky !

If we lecturers pick the set of marking test cases  
We would have to keep that test set really secret...  
It's too easy to write code to just pass specific tests

Also, having only a handful off test cases is risky...  
Could get "unlucky" and fail a few: with a big impact

How do we pick test cases to be fair to everyone ?

# The Solution ?

Our solution is to use EVERYBODY'S test cases !  
(the ones YOU submit with YOUR finished code)

So we aren't just checking a handful of input values  
(Just the few ones WE thought were a good idea)

Instead we check your code works in ALL situations  
(which is what REAL testing is SUPPOSED to do)

Much more democratic than US picking tests  
We get "quorate consensus on correctness"

# Things to consider

YOUR code will be assessed by YOUR test cases  
So you better make them good ones !

OTHER students will be assessed by your test cases  
So you better make them good ones !

You will be assessed on completeness of your tests  
So you better make them good ones !

Like other code, tests subject to plagiarism checks  
(so don't make your test scripts public !)

# Moderation

There is some unpredictability in this process  
We don't really know what tests you'll come up with

It is essential that we keep a "human in the loop"  
We (the teaching team) will apply our judgement

Identify appropriate thresholds levels to convert:  
"number of tests passed" into a "percentage grade"

Overall aim: to appropriately reward achievement



# What about "rogue" students ?

There are always going to be some "rogue" students  
Who see things in a different way to everyone else  
(and who thus write some very "unusual" test cases)

As programmers, we must consider all eventualities  
Good practice to think about alternative perspectives

These have marginal impact on body of test cases  
We reserve the right to remove individual test cases  
(if we think they are just plain crazy !)

**Deadline: Thursday 23rd Feb at 13:00**

Online Q&A: Tuesday 21st at 12 noon

Questions ?