VIA University College

Software Development with UML and Java 2

Course Assignment 1

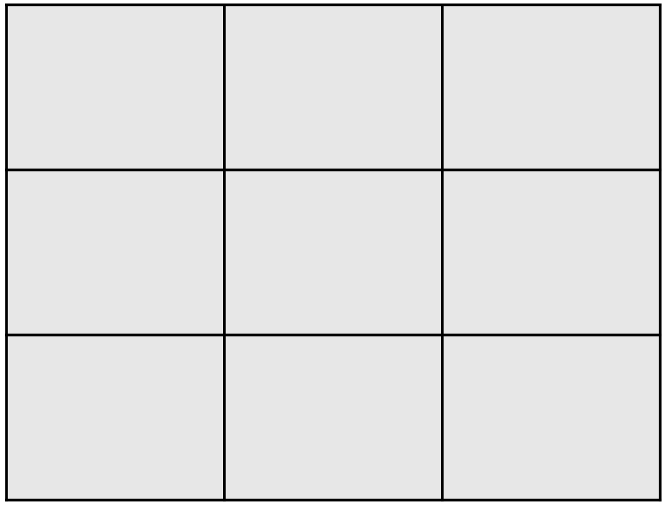
Topics covered

- RMI
- MVC
- Adapter
- Observer

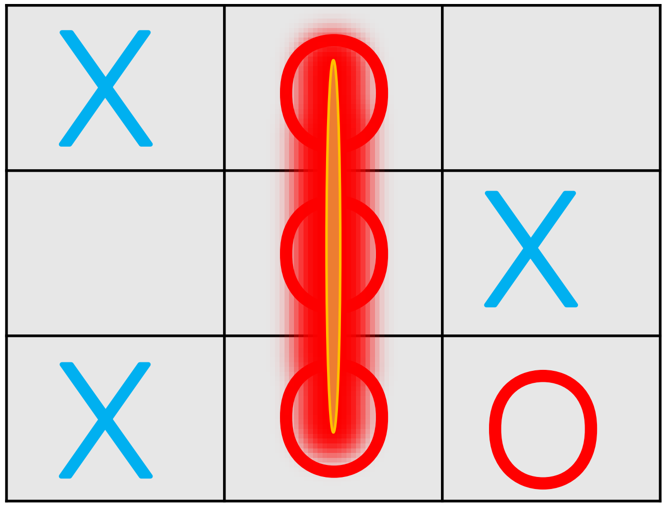
The idea

- This is a tic tac toe game, with spectators

Tic-Tac-Toe



Tic-Tac-Toe

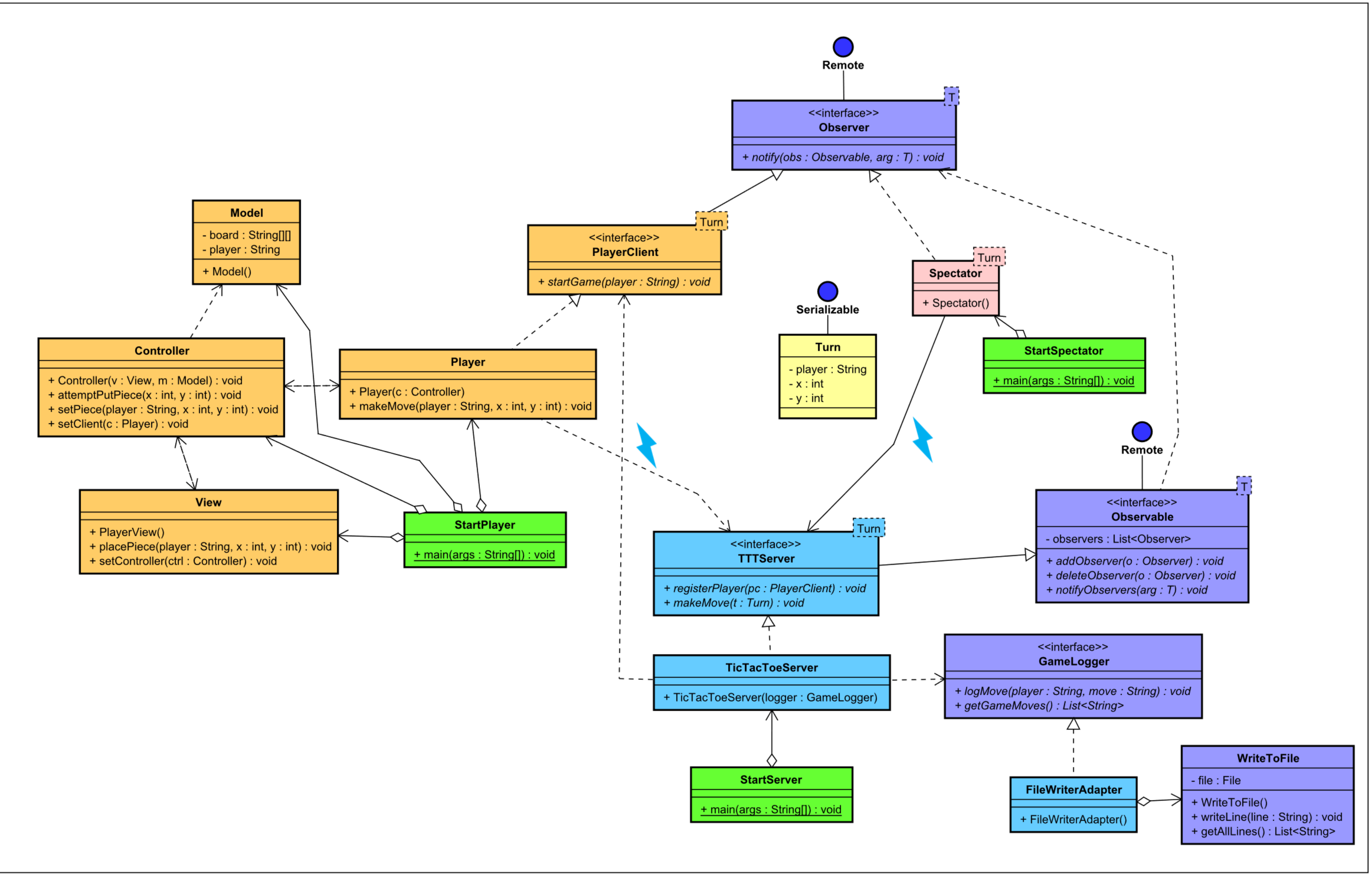


The idea

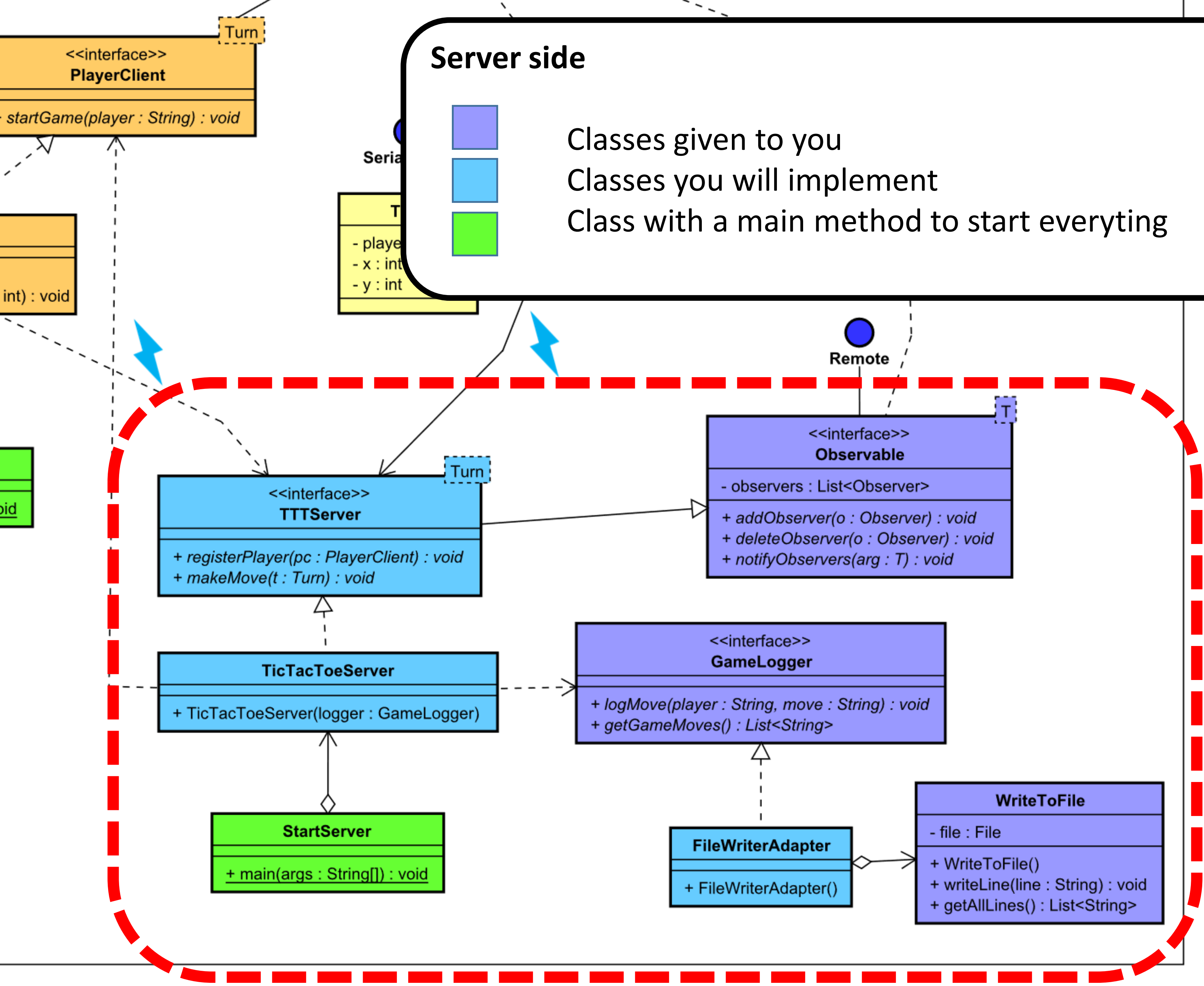
- Two types of client
  - One type of client is the “player” client.
    - There are two players to the game
  - Another type of client is a passive ‘spectator’
- The server
  - Handles multiple types of client
  - Broadcasts the game information to all interested

UML

- A guideline
- You must have the same classes/structure
- You may change methods/names
- You may add classes



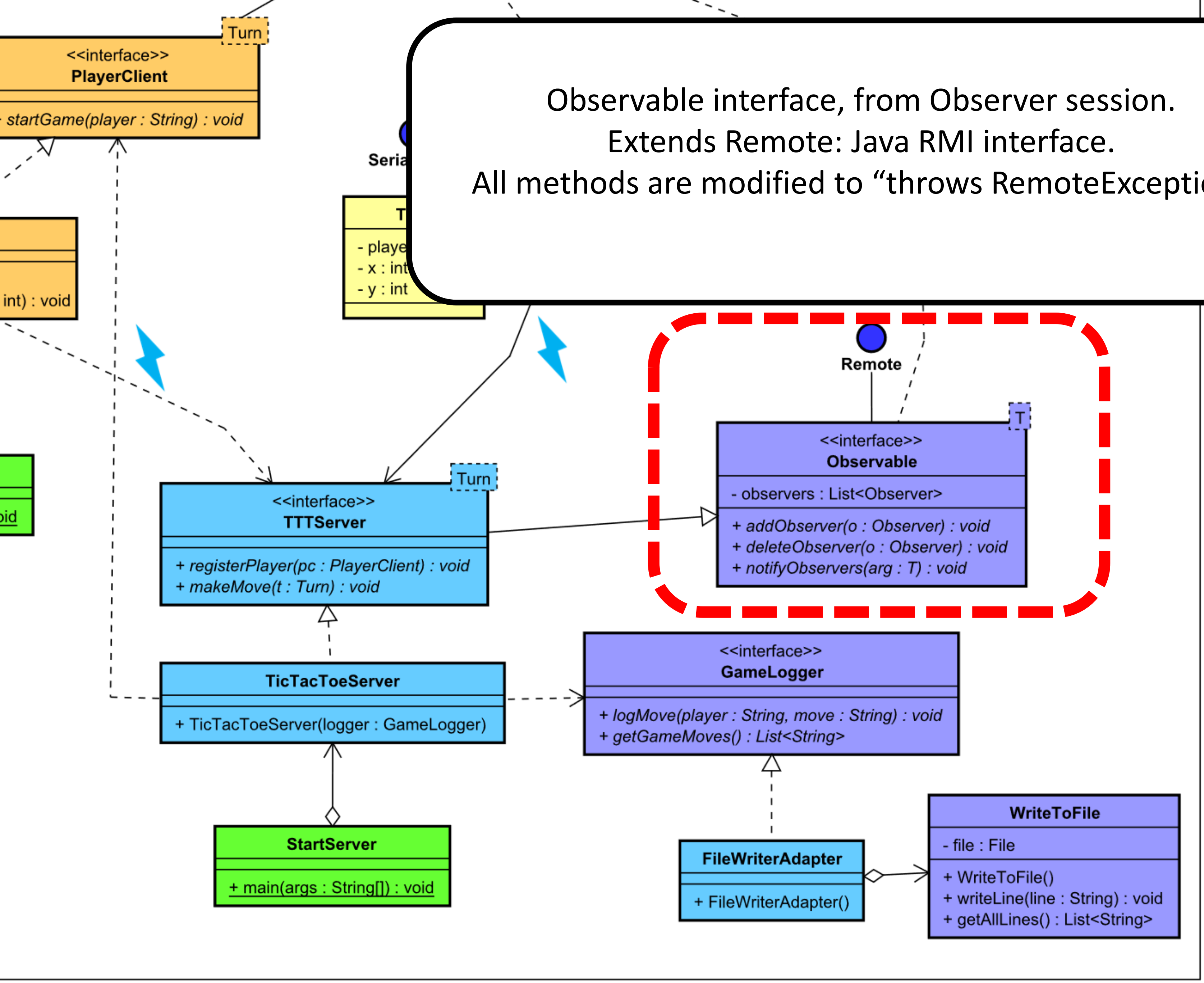
Remote
T
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : void
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Model
- board : String[][]
- player : String
+ Model()
Turn
Spectator
+ Spectator()
Serializable
Turn
- player : String
- x : int
- y : int
StartSpectator
+ main(args : String[]) : void
Controller
+ Controller(v : View, m : Model) : void
+ attemptPutPiece(x : int, y : int) : void
+ setPiece(player : String, x : int, y : int) : void
+ setClient(c : Player) : void
Player
+ Player(c : Controller)
+ makeMove(player : String, x : int, y : int) : void
Remote
T
<<interface>>
Observable
- observers : List<Observer>
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : void
+ notifyObservers(arg : T) : void
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
StartPlayer
+ main(args : String[]) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
TicTacToeServer
+ TicTacToeServer(logger : GameLogger)
<<interface>>
GameLogger
+ logMove(player : String, move : String) : void
+ getGameMoves() : List<String>
StartServer
+ main(args : String[]) : void
FileWriterAdapter
+ FileWriterAdapter()
WriteToFile
- file : File
+ WriteToFile()
+ writeLine(line : String) : void
+ getAllLines() : List<String>


Turn
<<interface>>
PlayerClient
startGame(player : String) : void
Seria
T
- playe
- x : int
- y : int
int) : void
oid
Remote
T
<<interface>>
TTTServer
Turn
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
<<interface>>
Observable
- observers : List<Observer>
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : void
+ notifyObservers(arg : T) : void
TicTacToeServer
+ TicTacToeServer(logger : GameLogger)
<<interface>>
GameLogger
+ logMove(player : String, move : String) : void
+ getGameMoves() : List<String>
StartServer
+ main(args : String[]) : void
FileWriterAdapter
+ FileWriterAdapter()
WriteToFile
- file : File
+ WriteToFile()
+ writeLine(line : String) : void
+ getAllLines() : List<String>

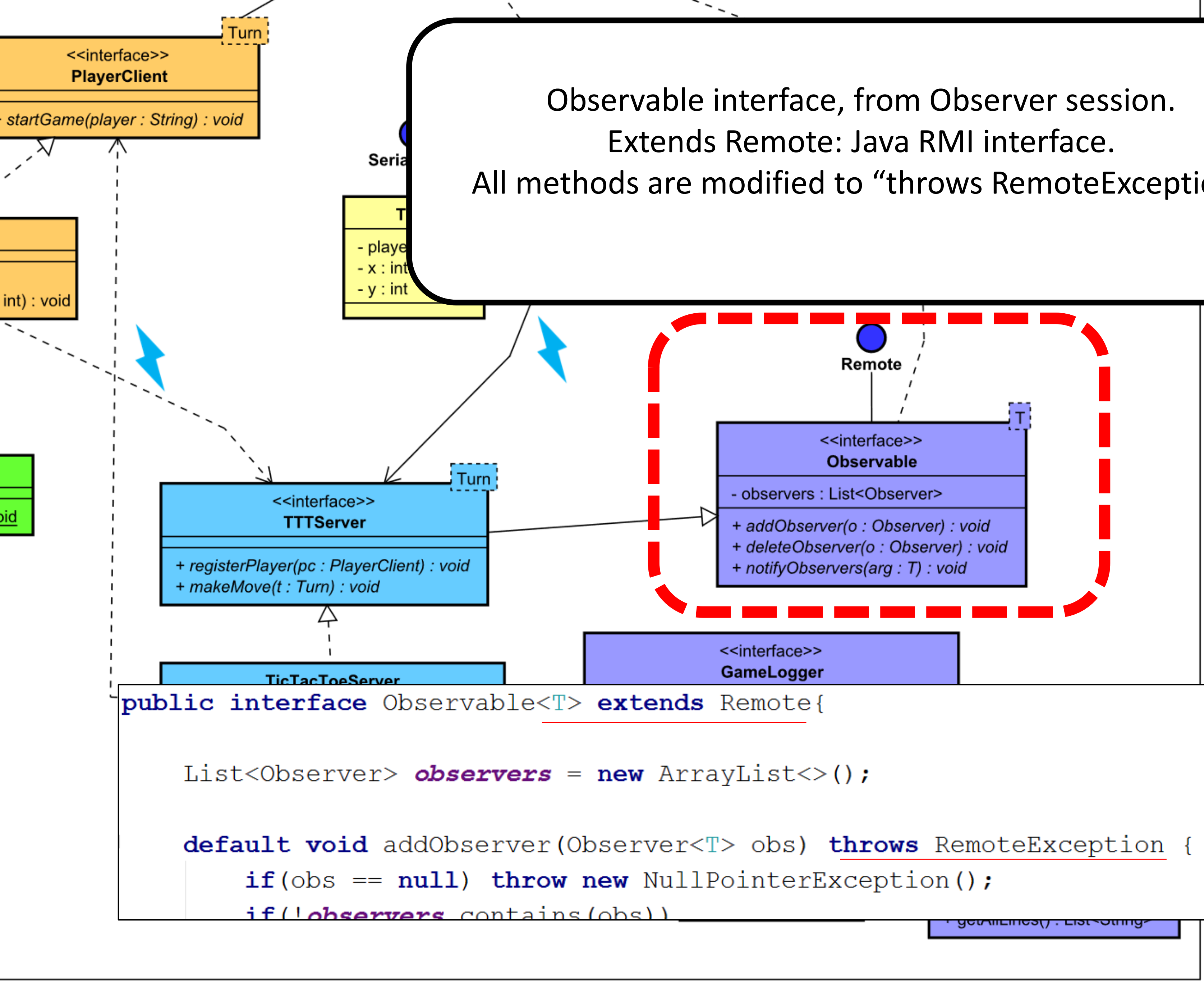
Server side

🟪	Classes given to you
🟦	Classes you will implement
🟩	Class with a main method to start everyting

Observable interface, from Observer session.
Extends Remote: Java RMI interface.
All methods are modified to "throws RemoteException"


Turn
<<interface>>
PlayerClient
startGame(player : String) : void
Seria
T
- playe
- x : int
- y : int
int) : void
oid
Remote
T
<<interface>>
Observable
- observers : List<Observer>
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : void
+ notifyObservers(arg : T) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
TicTacToeServer
+ TicTacToeServer(logger : GameLogger)
<<interface>>
GameLogger
+ logMove(player : String, move : String) : void
+ getGameMoves() : List<String>
StartServer
+ main(args : String[]) : void
FileWriterAdapter
+ FileWriterAdapter()
WriteToFile
- file : File
+ WriteToFile()
+ writeLine(line : String) : void
+ getAllLines() : List<String>

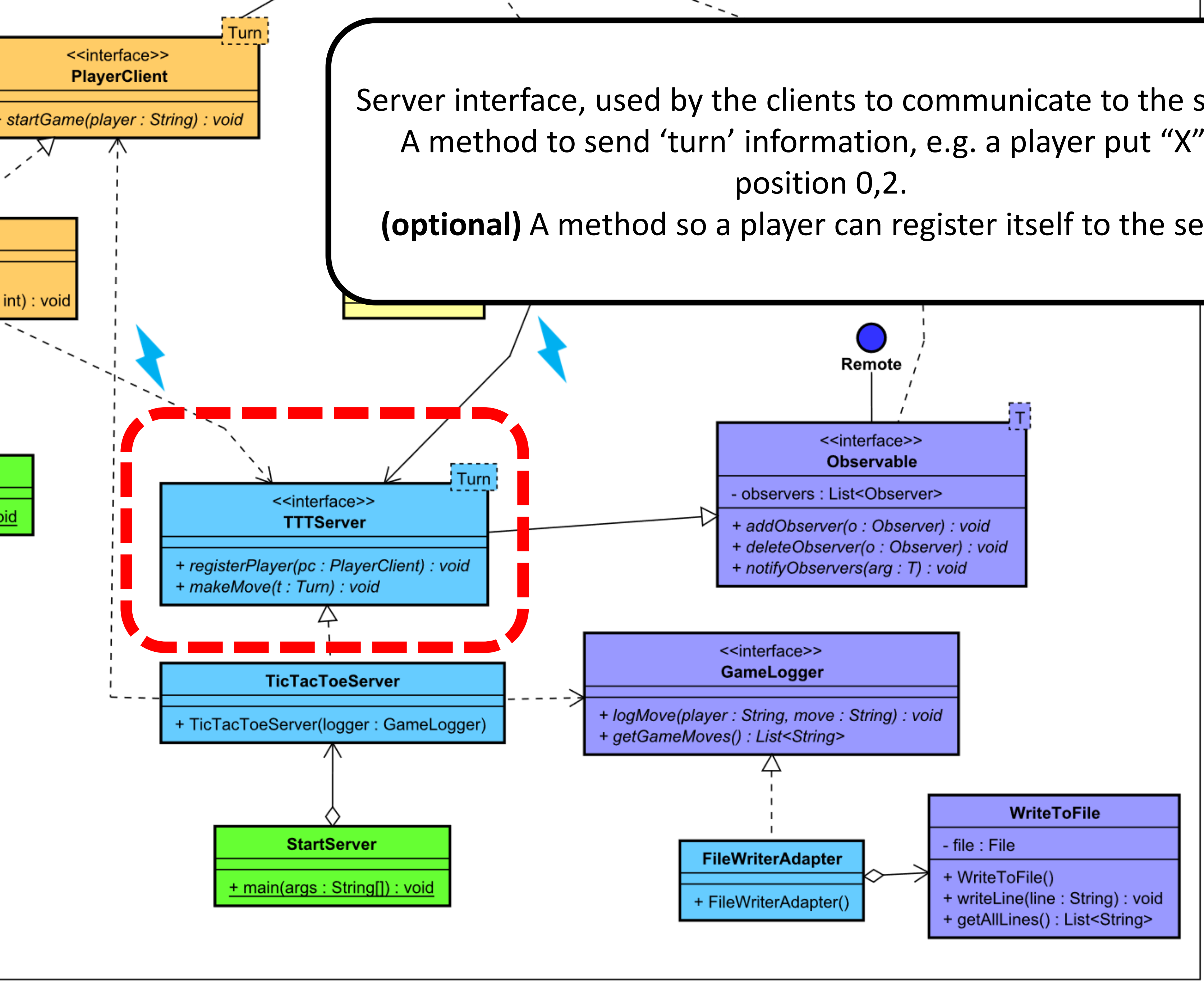
Observable interface, from Observer session.
Extends Remote: Java RMI interface.
All methods are modified to “throws RemoteException”

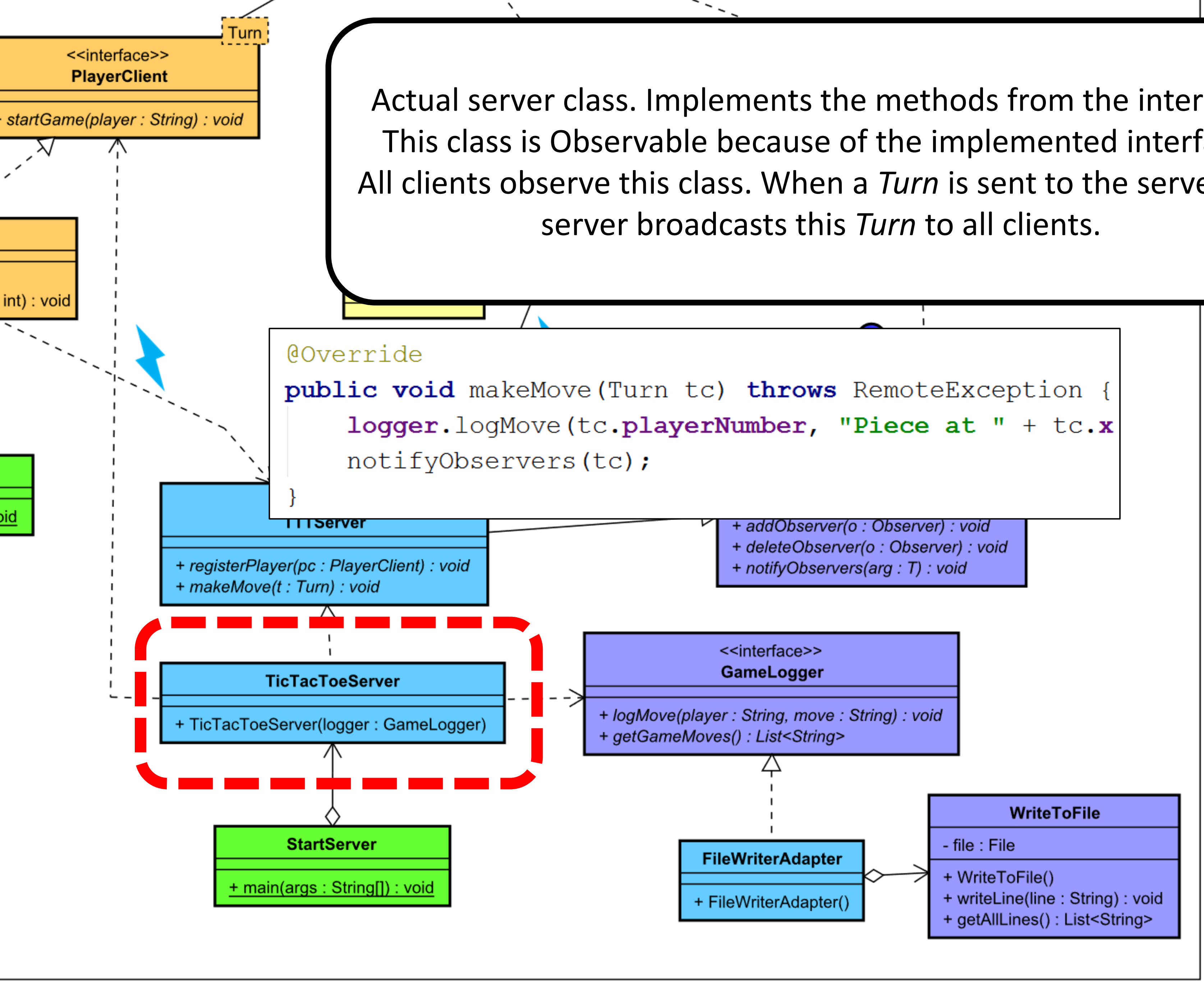


Turn
<<interface>>
PlayerClient
startGame(player : String) : void
Seria
int) : void
- playe
- x : int
- y : int
Remote
T
<<interface>>
Observable
- observers : List<Observer>
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : void
+ notifyObservers(arg : T) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
TicTacToeServer
<<interface>>
GameLogger

```
public interface Observable<T> extends Remote{

    List<Observer> observers = new ArrayList<>();

    default void addObserver(Observer<T> obs) throws RemoteException {
        if(obs == null) throw new NullPointerException();
        if(!observers.contains(obs))
```

Server interface, used by the clients to communicate to the server.
A method to send 'turn' information, e.g. a player put "X" at position 0,2.
(optional) A method so a player can register itself to the server

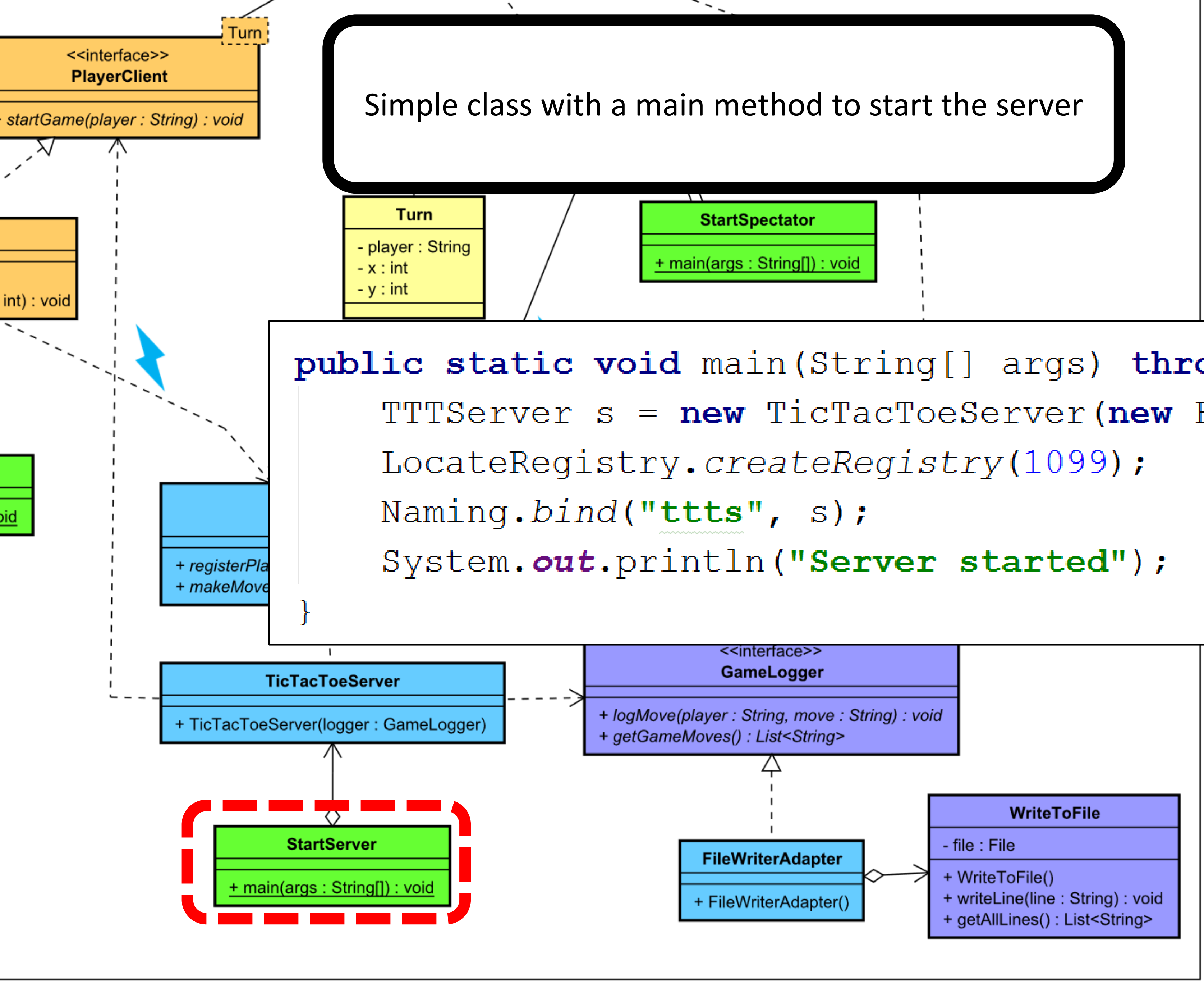

Turn
<<interface>>
PlayerClient
startGame(player : String) : void
int) : void
oid
Remote
T
<<interface>>
Observable
- observers : List<Observer>
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : void
+ notifyObservers(arg : T) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
TicTacToeServer
+ TicTacToeServer(logger : GameLogger)
<<interface>>
GameLogger
+ logMove(player : String, move : String) : void
+ getGameMoves() : List<String>
StartServer
+ main(args : String[]) : void
FileWriterAdapter
+ FileWriterAdapter()
WriteToFile
- file : File
+ WriteToFile()
+ writeLine(line : String) : void
+ getAllLines() : List<String>


Turn
<<interface>>
PlayerClient
startGame(player : String) : void
int) : void
oid
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : void
+ notifyObservers(arg : T) : void
TicTacToeServer
+ TicTacToeServer(logger : GameLogger)
<<interface>>
GameLogger
+ logMove(player : String, move : String) : void
+ getGameMoves() : List<String>
StartServer
+ main(args : String[]) : void
FileWriterAdapter
+ FileWriterAdapter()
WriteToFile
- file : File
+ WriteToFile()
+ writeLine(line : String) : void
+ getAllLines() : List<String>

Actual server class. Implements the methods from the interface. This class is Observable because of the implemented interface. All clients observe this class. When a *Turn* is sent to the server, the server broadcasts this *Turn* to all clients.

```
@Override
public void makeMove(Turn tc) throws RemoteException {
    logger.logMove(tc.playerNumber, "Piece at " + tc.x
    notifyObservers(tc);
}
```

Simple class with a main method to start the server


Turn
<<interface>>
PlayerClient
startGame(player : String) : void
int) : void
Turn
- player : String
- x : int
- y : int
StartSpectator
+ main(args : String[]) : void
oid
+ registerPla
+ makeMove
<<interface>>
GameLogger
+ logMove(player : String, move : String) : void
+ getGameMoves() : List<String>
TicTacToeServer
+ TicTacToeServer(logger : GameLogger)
StartServer
+ main(args : String[]) : void
FileWriterAdapter
+ FileWriterAdapter()
WriteToFile
- file : File
+ WriteToFile()
+ writeLine(line : String) : void
+ getAllLines() : List<String>

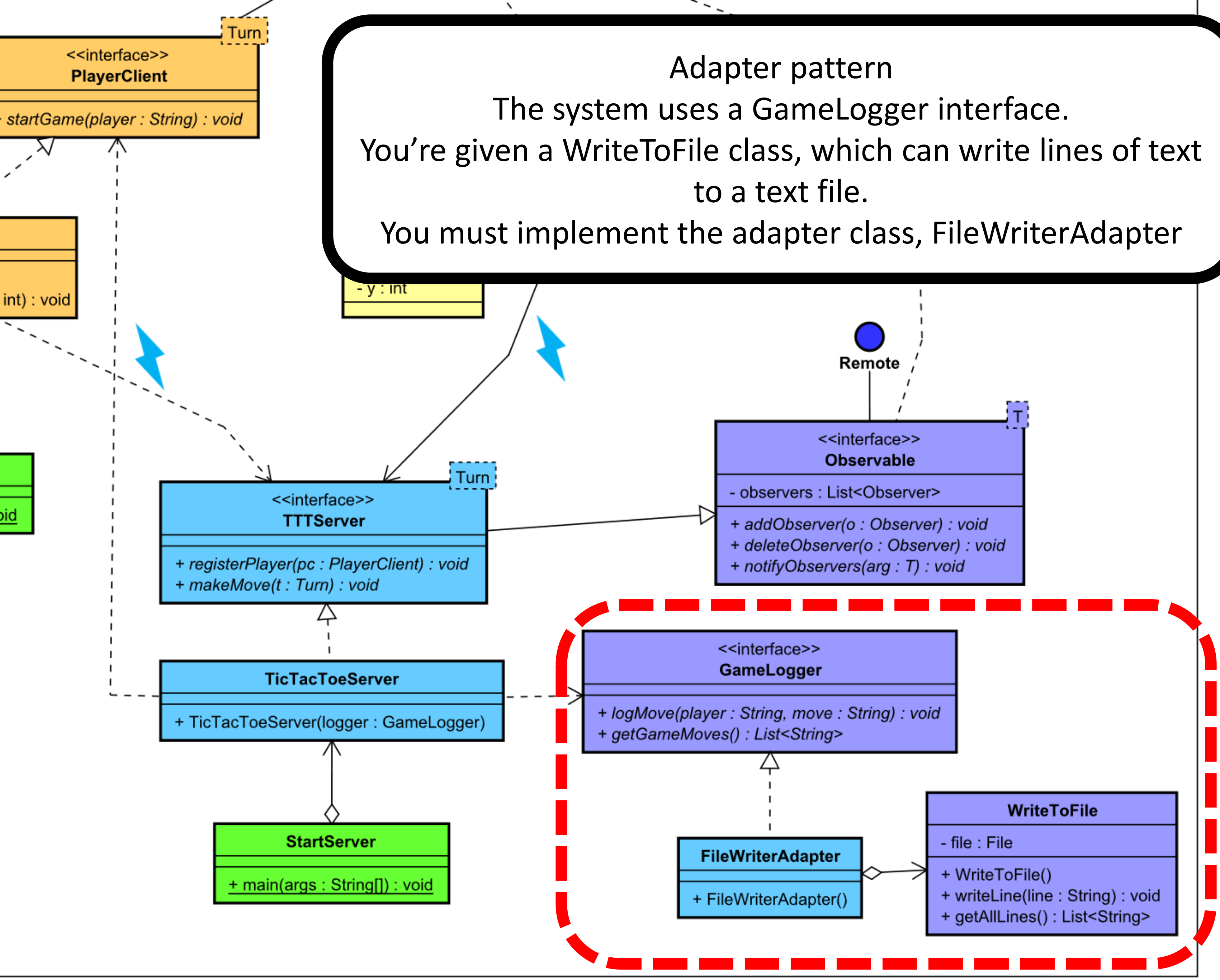
```
public static void main(String[] args) thro
    TTTServer s = new TicTacToeServer(new F
    LocateRegistry.createRegistry(1099);
    Naming.bind("ttts", s);
    System.out.println("Server started");
}
```

Adapter pattern

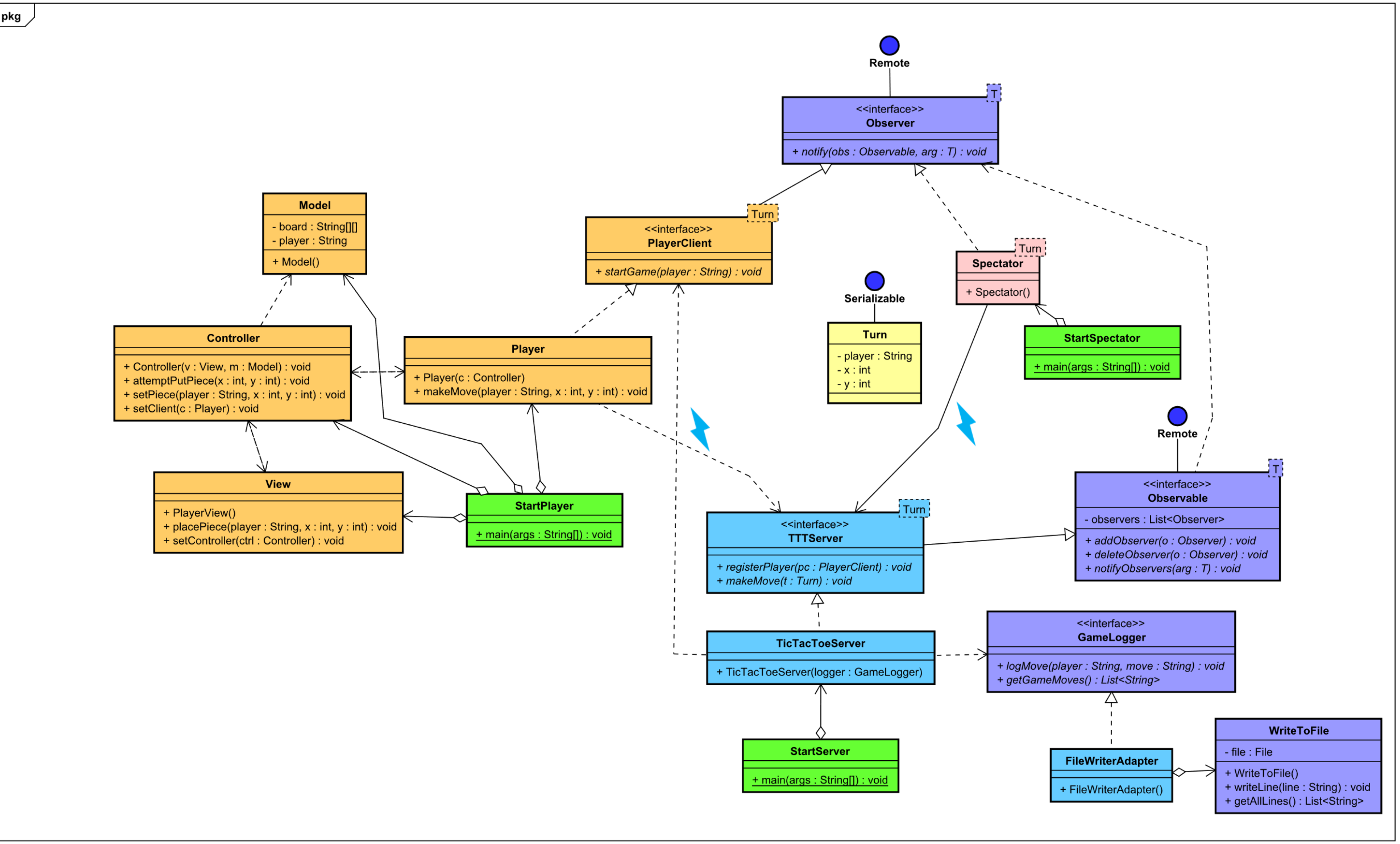
The system uses a GameLogger interface.

You're given a WriteToFile class, which can write lines of text to a text file.

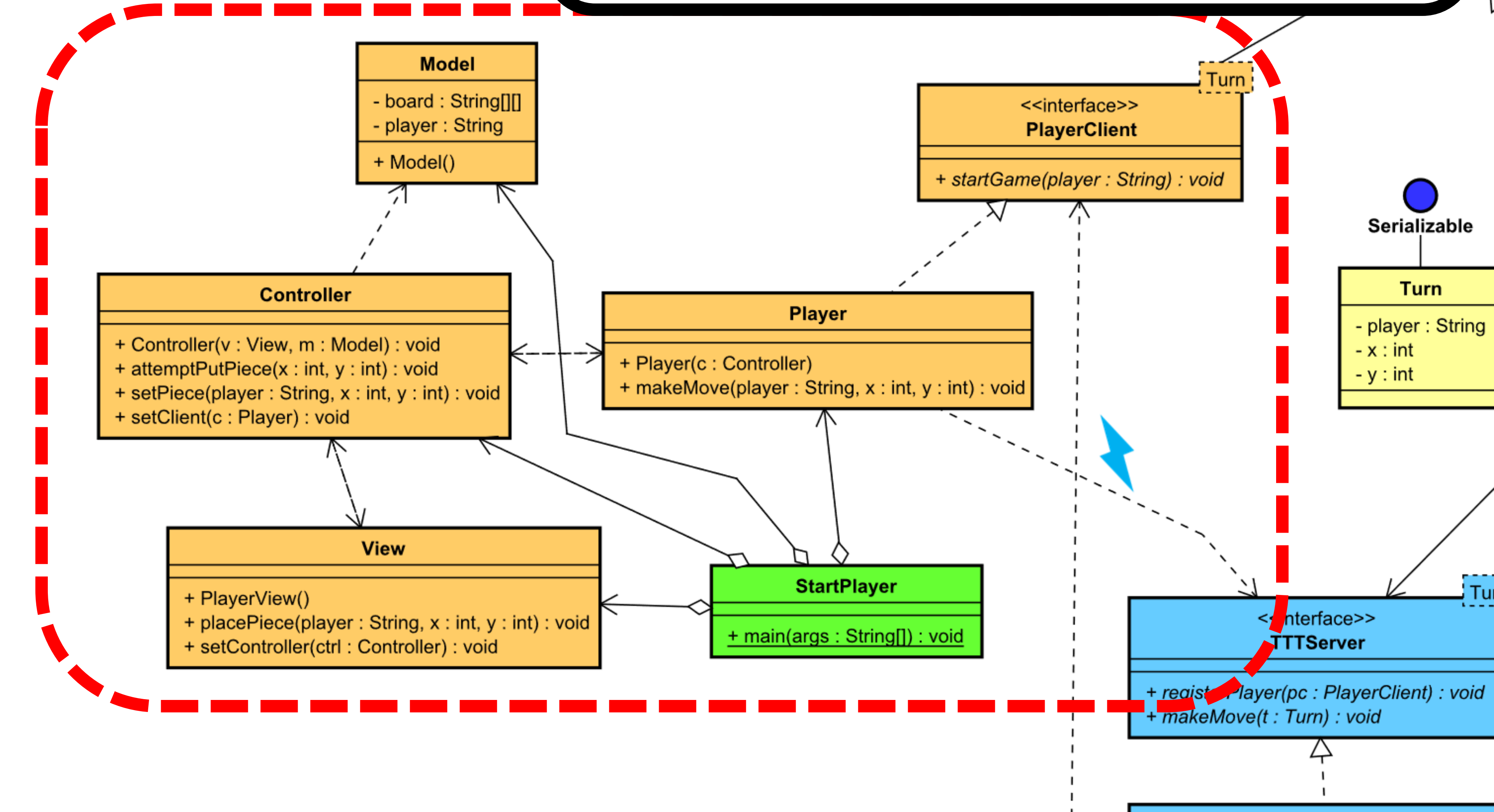
You must implement the adapter class, FileWriterAdapter



Turn
<<interface>>
PlayerClient
startGame(player : String) : void
int) : void
- y : int
id
Remote
T
<<interface>>
Observable
- observers : List<Observer>
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : void
+ notifyObservers(arg : T) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
TicTacToeServer
+ TicTacToeServer(logger : GameLogger)
StartServer
+ main(args : String[]) : void
<<interface>>
GameLogger
+ logMove(player : String, move : String) : void
+ getGameMoves() : List<String>
FileWriterAdapter
+ FileWriterAdapter()
WriteToFile
- file : File
+ WriteToFile()
+ writeLine(line : String) : void
+ getAllLines() : List<String>

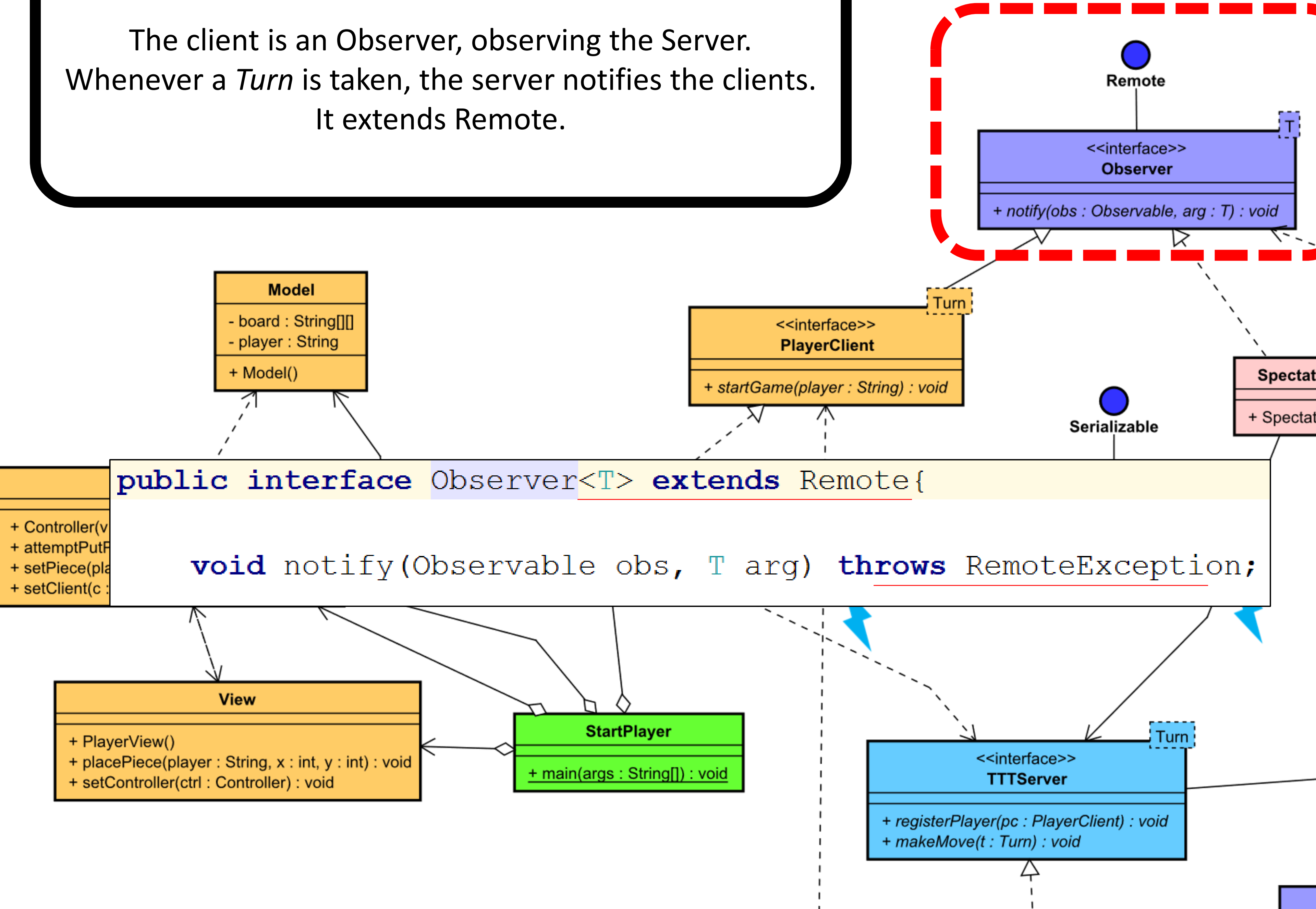

pkg
Remote
T
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : void
Model
- board : String[][]
- player : String
+ Model()
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Turn
Spectator
+ Spectator()
Serializable
Turn
- player : String
- x : int
- y : int
StartSpectator
+ main(args : String[]) : void
Controller
+ Controller(v : View, m : Model) : void
+ attemptPutPiece(x : int, y : int) : void
+ setPiece(player : String, x : int, y : int) : void
+ setClient(c : Player) : void
Player
+ Player(c : Controller)
+ makeMove(player : String, x : int, y : int) : void
Remote
T
<<interface>>
Observable
- observers : List<Observer>
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : void
+ notifyObservers(arg : T) : void
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
StartPlayer
+ main(args : String[]) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
TicTacToeServer
+ TicTacToeServer(logger : GameLogger)
<<interface>>
GameLogger
+ logMove(player : String, move : String) : void
+ getGameMoves() : List<String>
StartServer
+ main(args : String[]) : void
FileWriterAdapter
+ FileWriterAdapter()
WriteToFile
- file : File
+ WriteToFile()
+ writeLine(line : String) : void
+ getAllLines() : List<String>

Player client side. These are the classes for the client, which are used to play the game.
It is structured using MVC



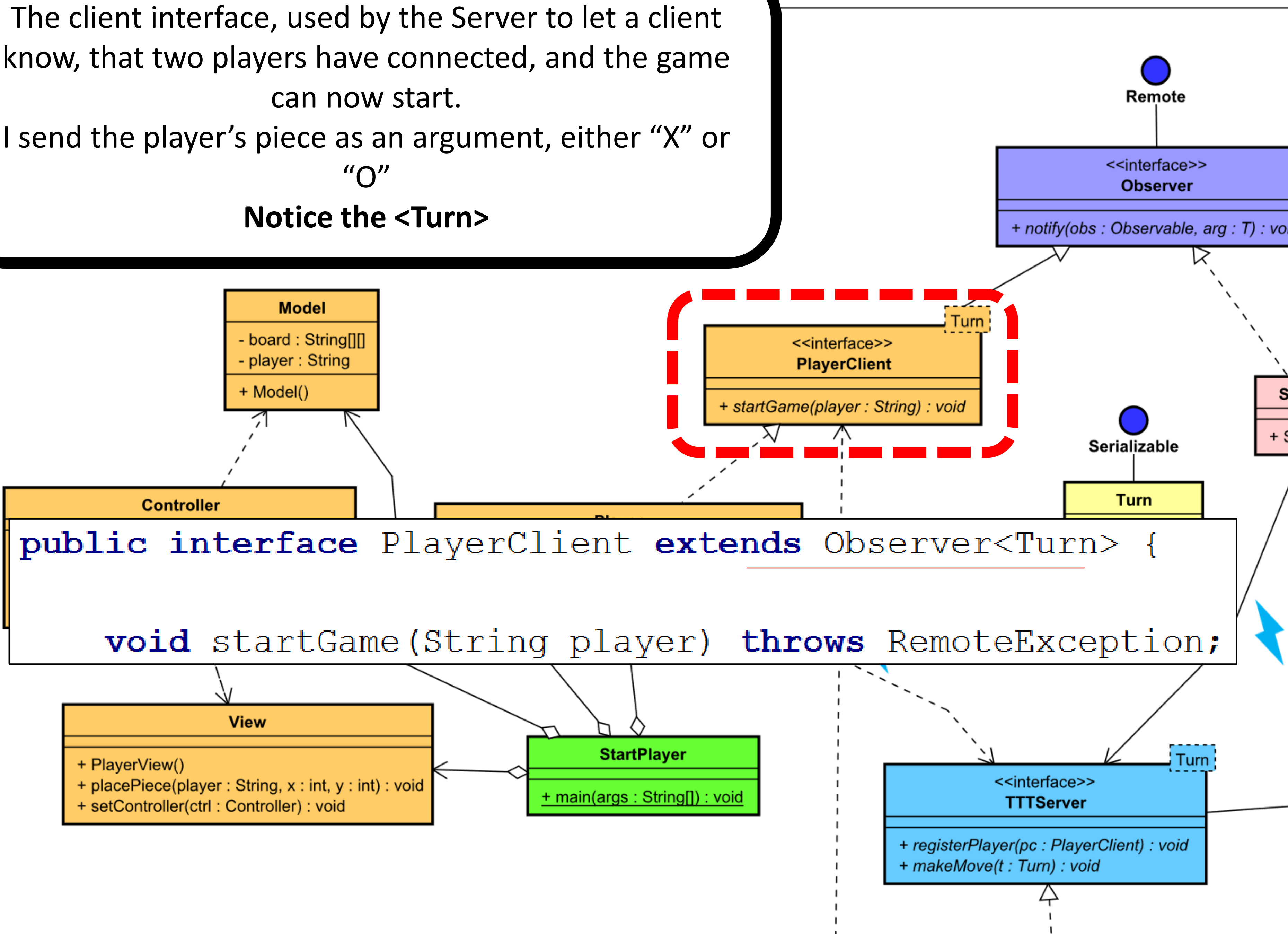
Model
- board : String[][]
- player : String
+ Model()
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Serializable
Turn
- player : String
- x : int
- y : int
Controller
+ Controller(v : View, m : Model) : void
+ attemptPutPiece(x : int, y : int) : void
+ setPiece(player : String, x : int, y : int) : void
+ setClient(c : Player) : void
Player
+ Player(c : Controller)
+ makeMove(player : String, x : int, y : int) : void
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
StartPlayer
+ main(args : String[]) : void
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void

The client is an Observer, observing the Server.
Whenever a *Turn* is taken, the server notifies the clients.
It extends Remote.


Remote
T
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : void
Model
- board : String[][]
- player : String
+ Model()
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Serializable
Spectat
+ Spectat
+ Controller(v
+ attemptPutP
+ setPiece(pla
+ setClient(c :
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
StartPlayer
+ main(args : String[]) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void

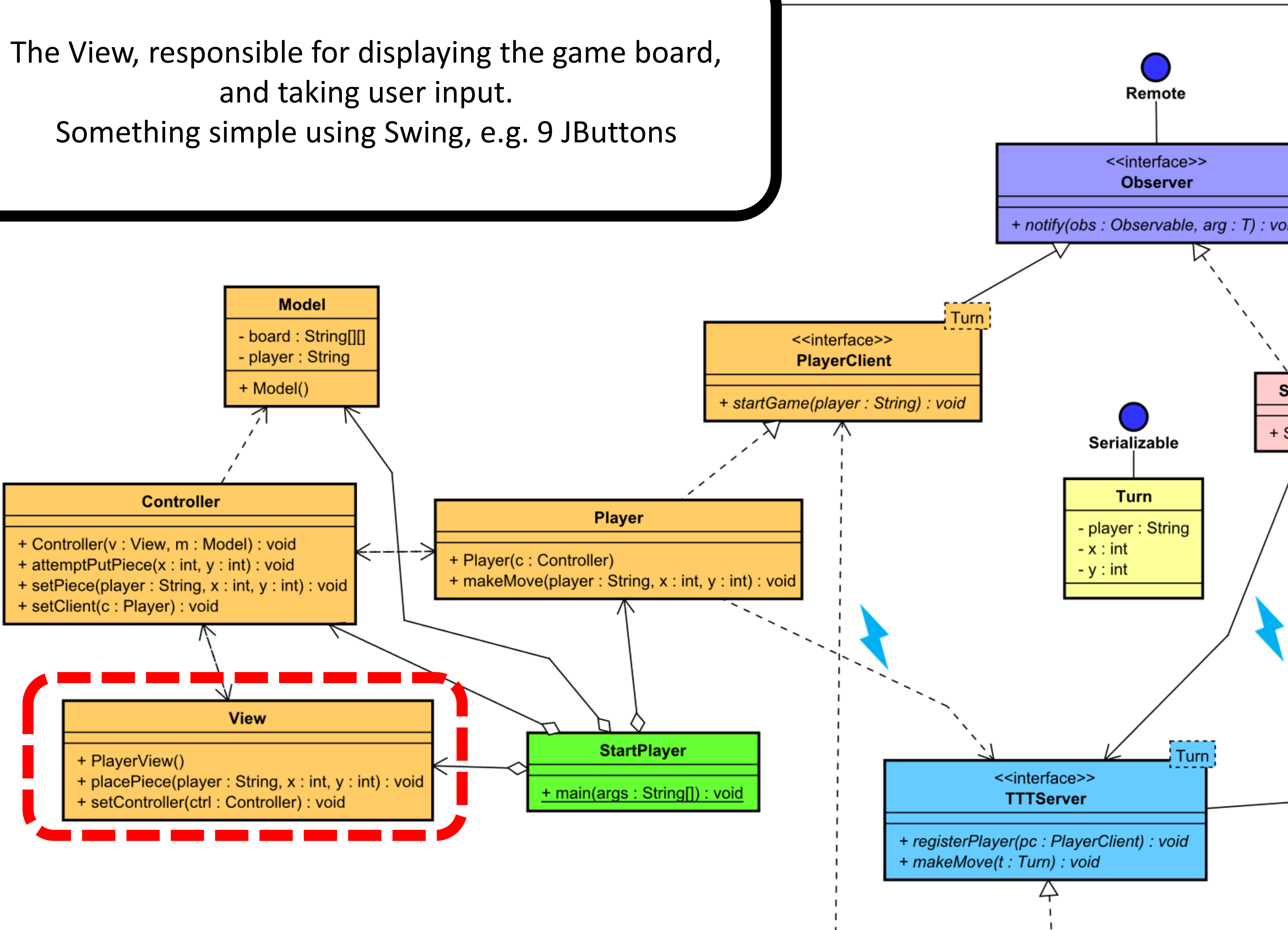
```
public interface Observer<T> extends Remote{

    void notify(Observable obs, T arg) throws RemoteException;
```

The client interface, used by the Server to let a client know, that two players have connected, and the game can now start.
I send the player's piece as an argument, either "X" or "O"
Notice the <Turn>

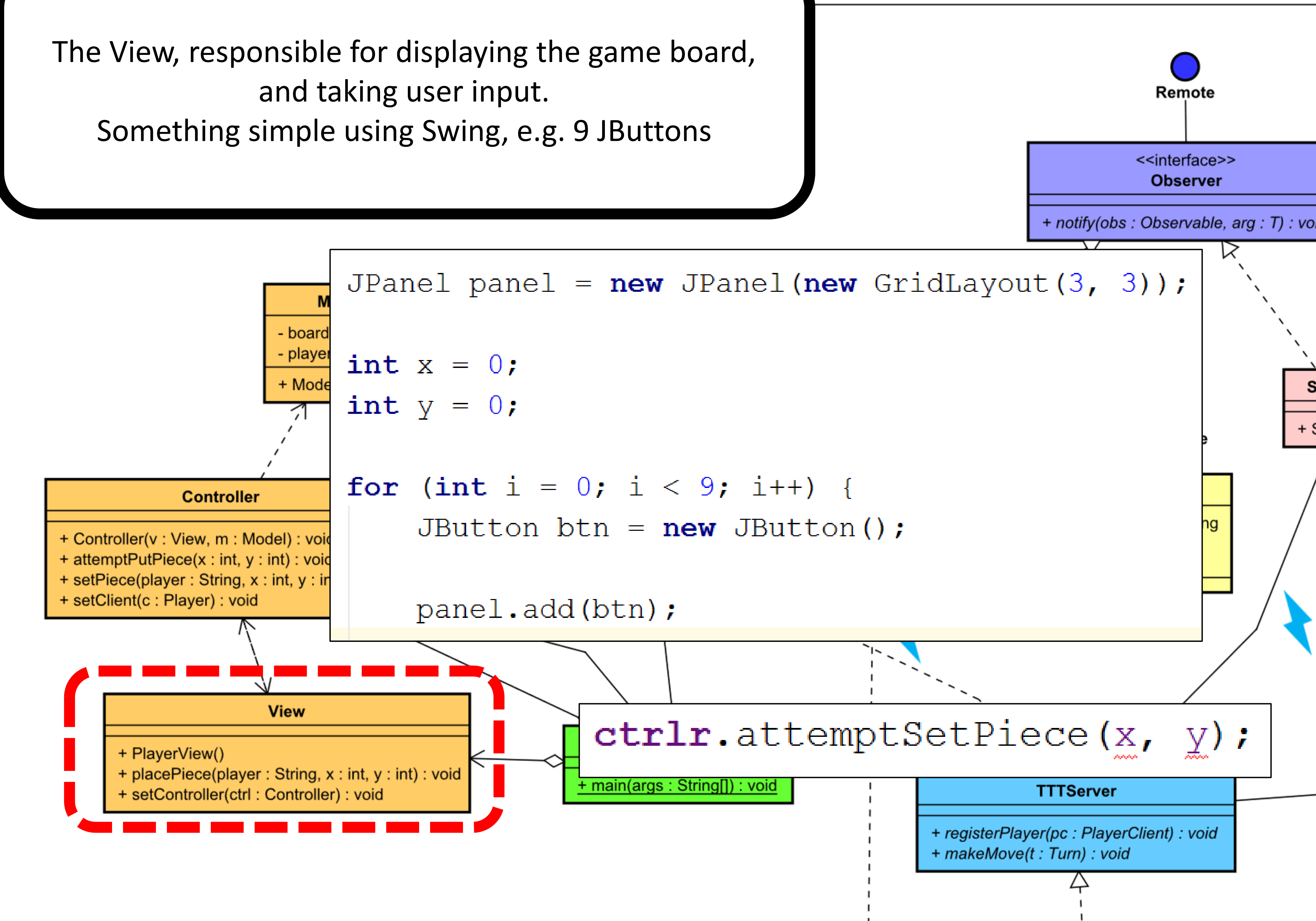

Remote
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : vo
Model
- board : String[][]
- player : String
+ Model()
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Serializable
Turn
Controller
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
StartPlayer
+ main(args : String[]) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void

```
public interface PlayerClient extends Observer<Turn> {

    void startGame(String player) throws RemoteException;
```

The View, responsible for displaying the game board, and taking user input.
Something simple using Swing, e.g. 9 JButtons


Remote
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : vo
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Model
- board : String[][]
- player : String
+ Model()
Controller
+ Controller(v : View, m : Model) : void
+ attemptPutPiece(x : int, y : int) : void
+ setPiece(player : String, x : int, y : int) : void
+ setClient(c : Player) : void
Player
+ Player(c : Controller)
+ makeMove(player : String, x : int, y : int) : void
Serializable
Turn
- player : String
- x : int
- y : int
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
StartPlayer
+ main(args : String[]) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void

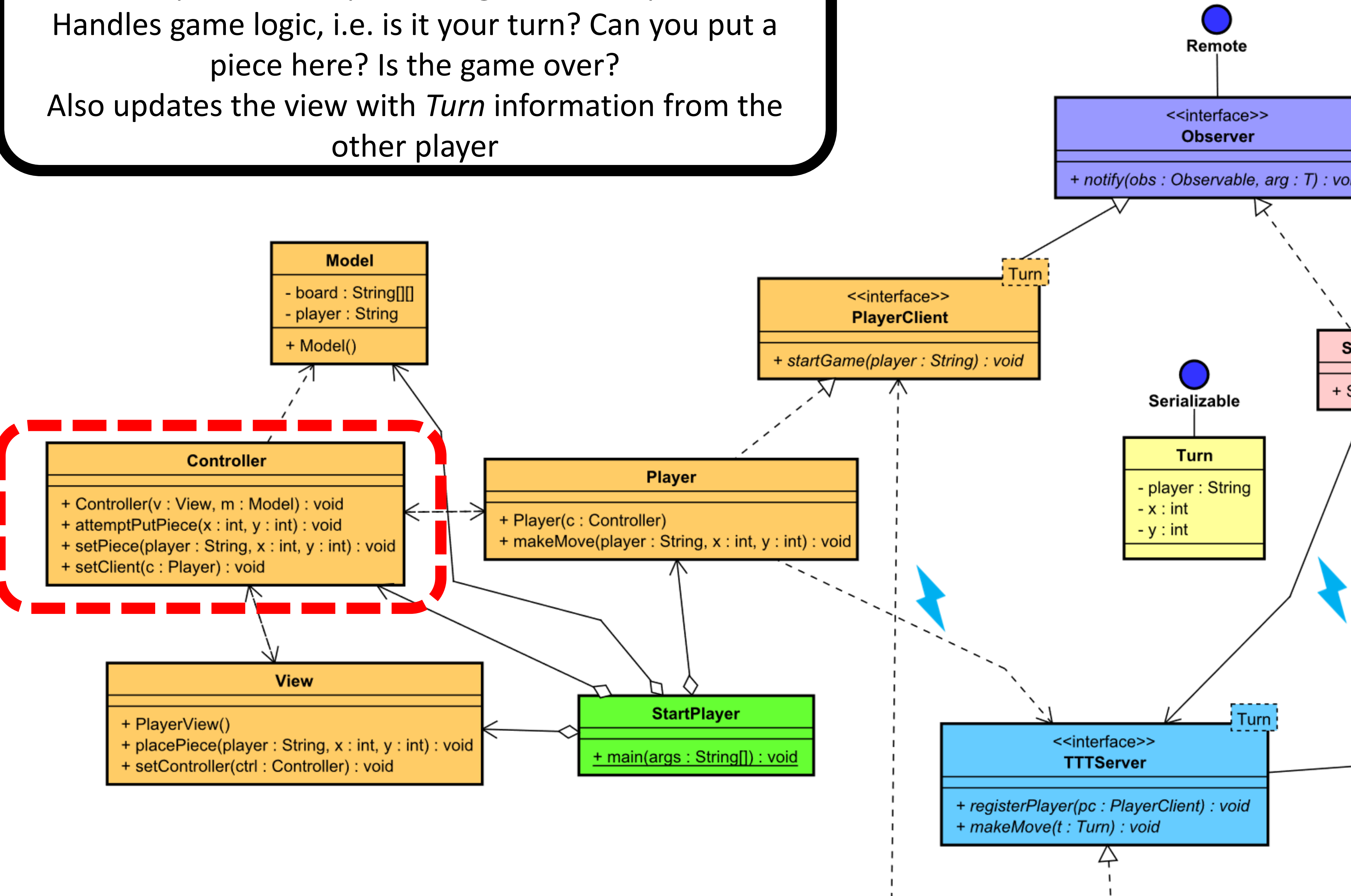
The View, responsible for displaying the game board, and taking user input.
Something simple using Swing, e.g. 9 JButtons


Remote
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : vo
- board
- player
+ Mode
Controller
+ Controller(v : View, m : Model) : voi
+ attemptPutPiece(x : int, y : int) : voi
+ setPiece(player : String, x : int, y : in
+ setClient(c : Player) : void
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
+ main(args : String[]) : void
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void

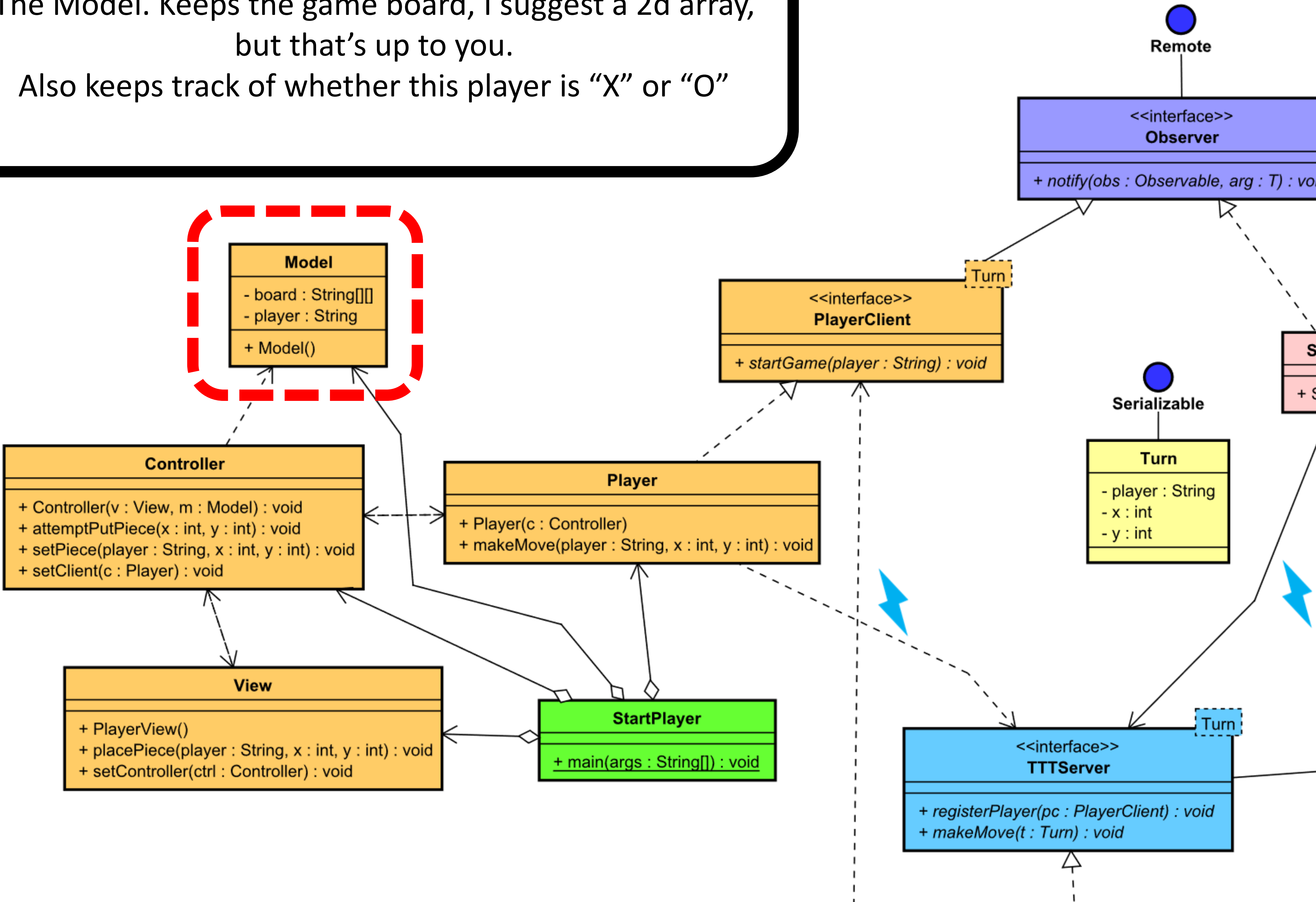
```
JPanel panel = new JPanel(new GridLayout(3, 3));

int x = 0;
int y = 0;

for (int i = 0; i < 9; i++) {
    JButton btn = new JButton();

    panel.add(btn);
```

```
ctrlr.attemptSetPiece(x, y);
```

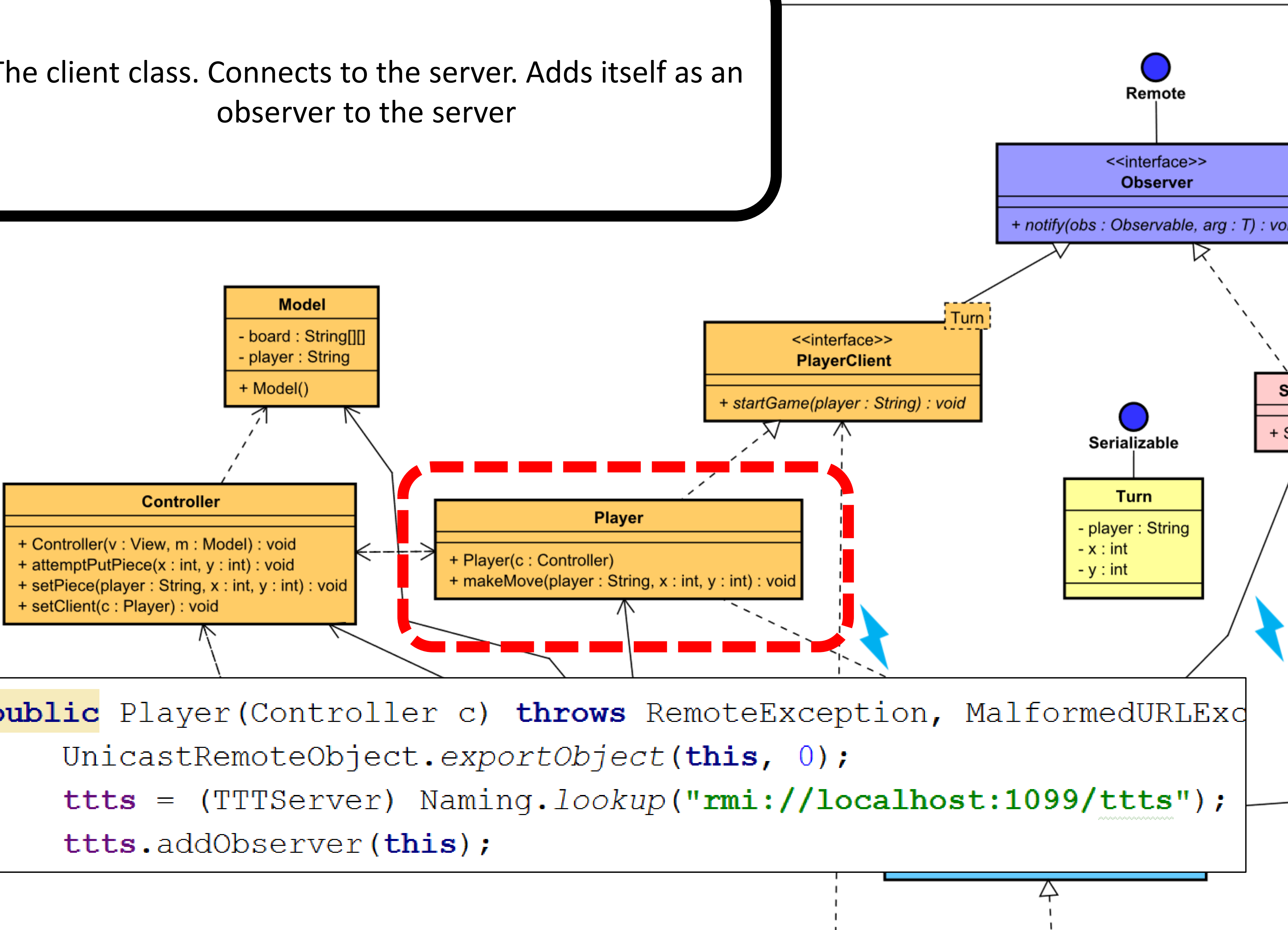
The controller.
Responsible for processing the user input.
Handles game logic, i.e. is it your turn? Can you put a piece here? Is the game over?
Also updates the view with *Turn* information from the other player


Remote
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : vo
Model
- board : String[][]
- player : String
+ Model()
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Serializable
Controller
+ Controller(v : View, m : Model) : void
+ attemptPutPiece(x : int, y : int) : void
+ setPiece(player : String, x : int, y : int) : void
+ setClient(c : Player) : void
Player
+ Player(c : Controller)
+ makeMove(player : String, x : int, y : int) : void
Turn
- player : String
- x : int
- y : int
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
StartPlayer
+ main(args : String[]) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void

The Model. Keeps the game board, I suggest a 2d array, but that's up to you.
Also keeps track of whether this player is "X" or "O"


Remote
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : vo
Model
- board : String[][]
- player : String
+ Model()
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Serializable
Turn
- player : String
- x : int
- y : int
Controller
+ Controller(v : View, m : Model) : void
+ attemptPutPiece(x : int, y : int) : void
+ setPiece(player : String, x : int, y : int) : void
+ setClient(c : Player) : void
Player
+ Player(c : Controller)
+ makeMove(player : String, x : int, y : int) : void
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
StartPlayer
+ main(args : String[]) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void

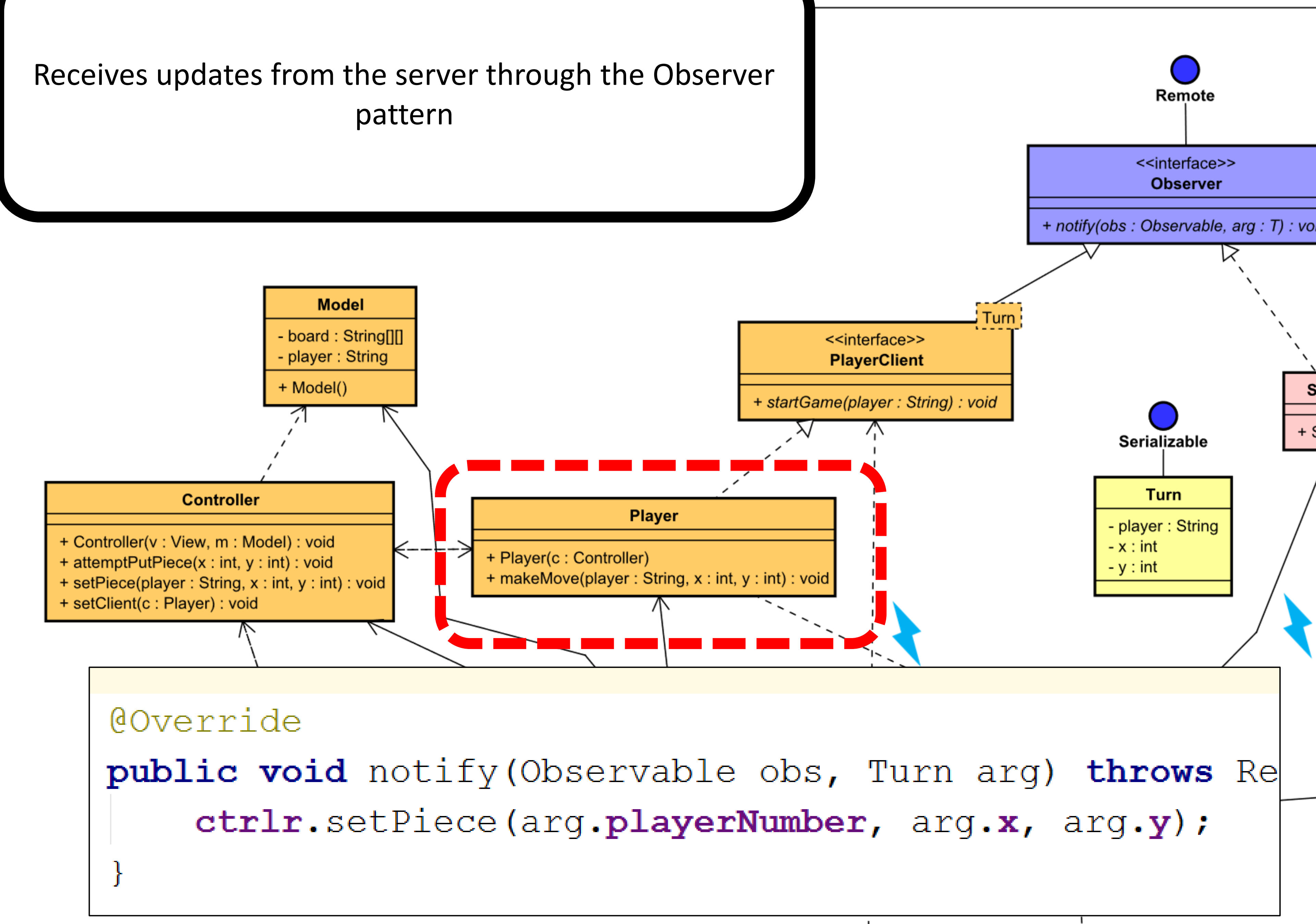
The client class. Connects to the server. Adds itself as an observer to the server



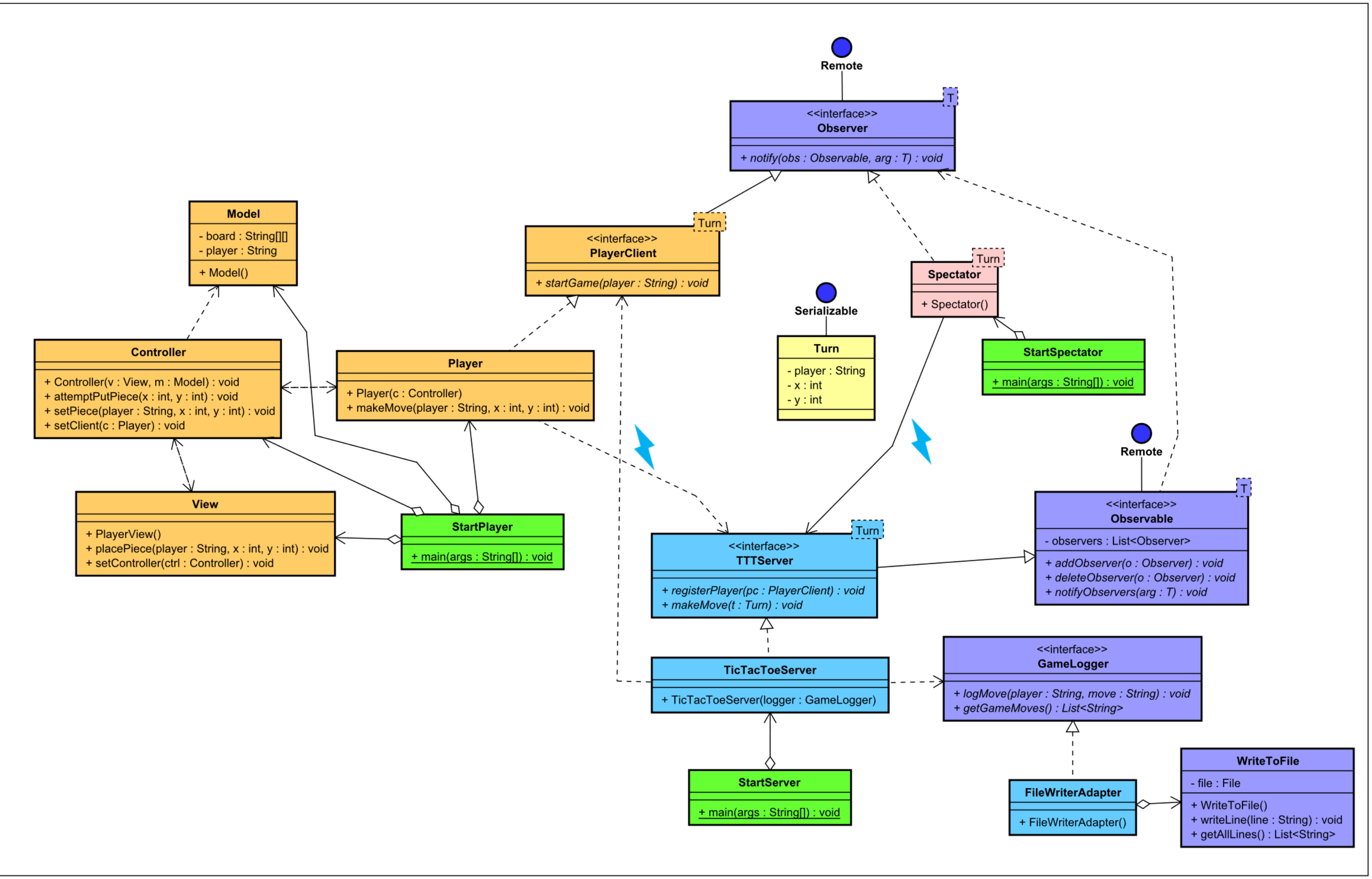
Remote
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : vo
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Model
- board : String[][]
- player : String
+ Model()
Serializable
Turn
- player : String
- x : int
- y : int
Controller
+ Controller(v : View, m : Model) : void
+ attemptPutPiece(x : int, y : int) : void
+ setPiece(player : String, x : int, y : int) : void
+ setClient(c : Player) : void
Player
+ Player(c : Controller)
+ makeMove(player : String, x : int, y : int) : void
S
+ S

```
public Player(Controller c) throws RemoteException, MalformedURLExc
    UnicastRemoteObject.exportObject(this, 0);
    ttts = (TTTServer) Naming.lookup("rmi://localhost:1099/ttts");
    ttts.addObserver(this);
```

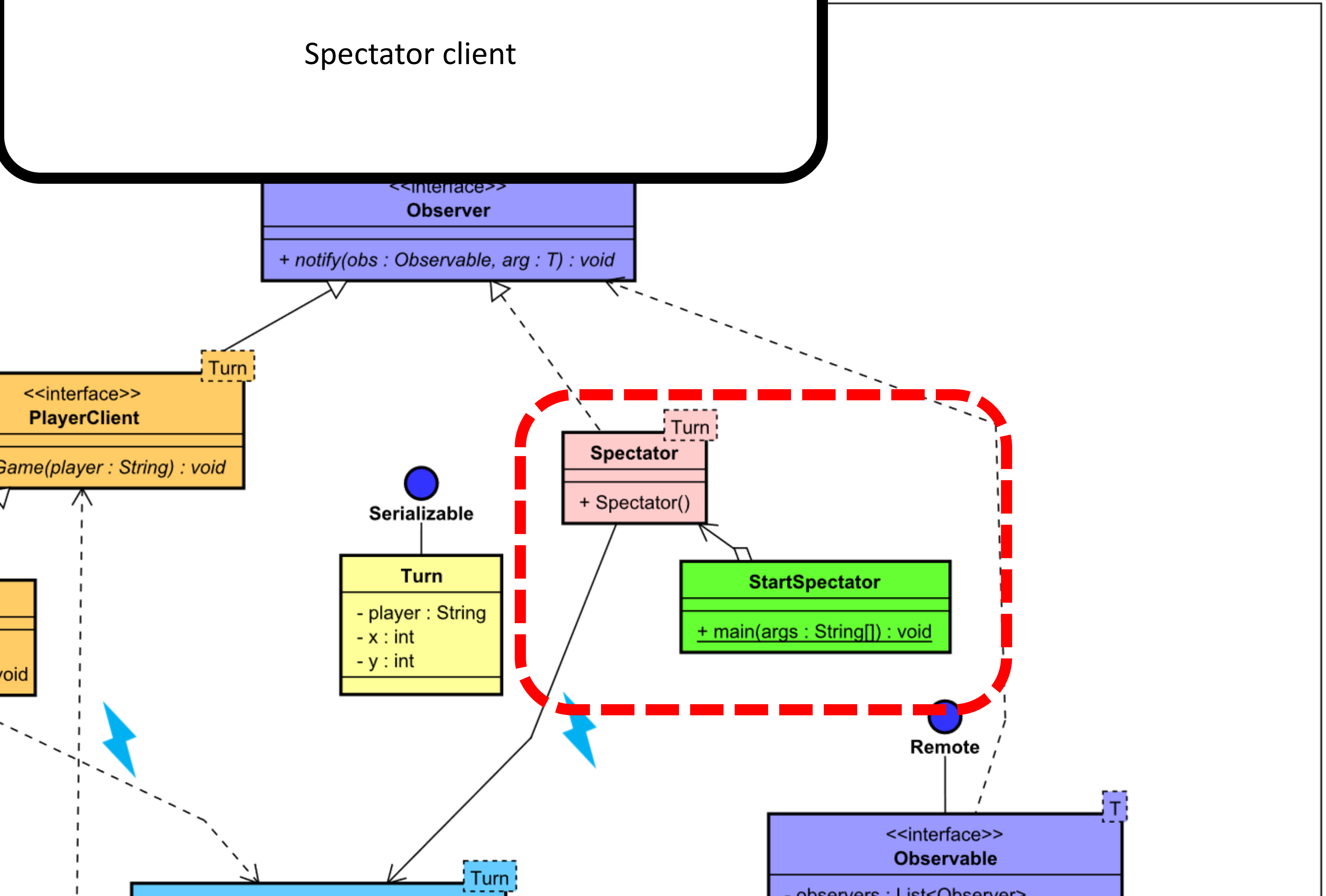
Receives updates from the server through the Observer pattern


Remote
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : vo
Model
- board : String[][]
- player : String
+ Model()
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Serializable
Turn
- player : String
- x : int
- y : int
Controller
+ Controller(v : View, m : Model) : void
+ attemptPutPiece(x : int, y : int) : void
+ setPiece(player : String, x : int, y : int) : void
+ setClient(c : Player) : void
Player
+ Player(c : Controller)
+ makeMove(player : String, x : int, y : int) : void

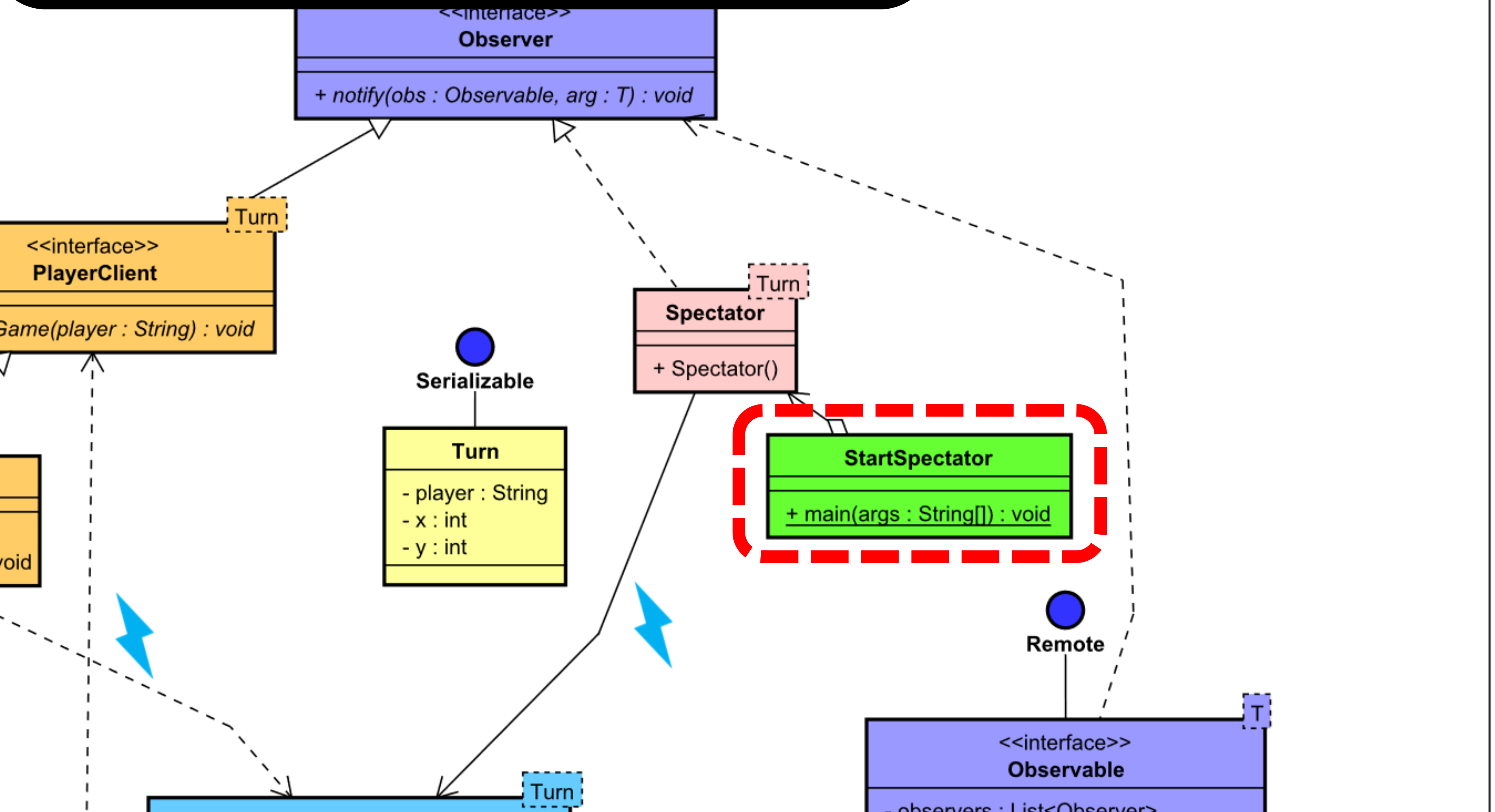
```
@Override
public void notify(Observable obs, Turn arg) throws Re
    ctrlr.setPiece(arg.playerNumber, arg.x, arg.y);
}
```


Remote
T
<<interface>>
Observer
+ notify(obs : Observable, arg : T) : void
Model
- board : String[][]
- player : String
+ Model()
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Turn
Spectator
+ Spectator()
Serializable
Turn
- player : String
- x : int
- y : int
Controller
+ Controller(v : View, m : Model) : void
+ attemptPutPiece(x : int, y : int) : void
+ setPiece(player : String, x : int, y : int) : void
+ setClient(c : Player) : void
Player
+ Player(c : Controller)
+ makeMove(player : String, x : int, y : int) : void
StartSpectator
+ main(args : String[]) : void
Remote
T
<<interface>>
Observable
- observers : List<Observer>
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : void
+ notifyObservers(arg : T) : void
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
StartPlayer
+ main(args : String[]) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
TicTacToeServer
+ TicTacToeServer(logger : GameLogger)
<<interface>>
GameLogger
+ logMove(player : String, move : String) : void
+ getGameMoves() : List<String>
StartServer
+ main(args : String[]) : void
WriteToFile
- file : File
+ WriteToFile()
+ writeLine(line : String) : void
+ getAllLines() : List<String>
FileWriterAdapter
+ FileWriterAdapter()

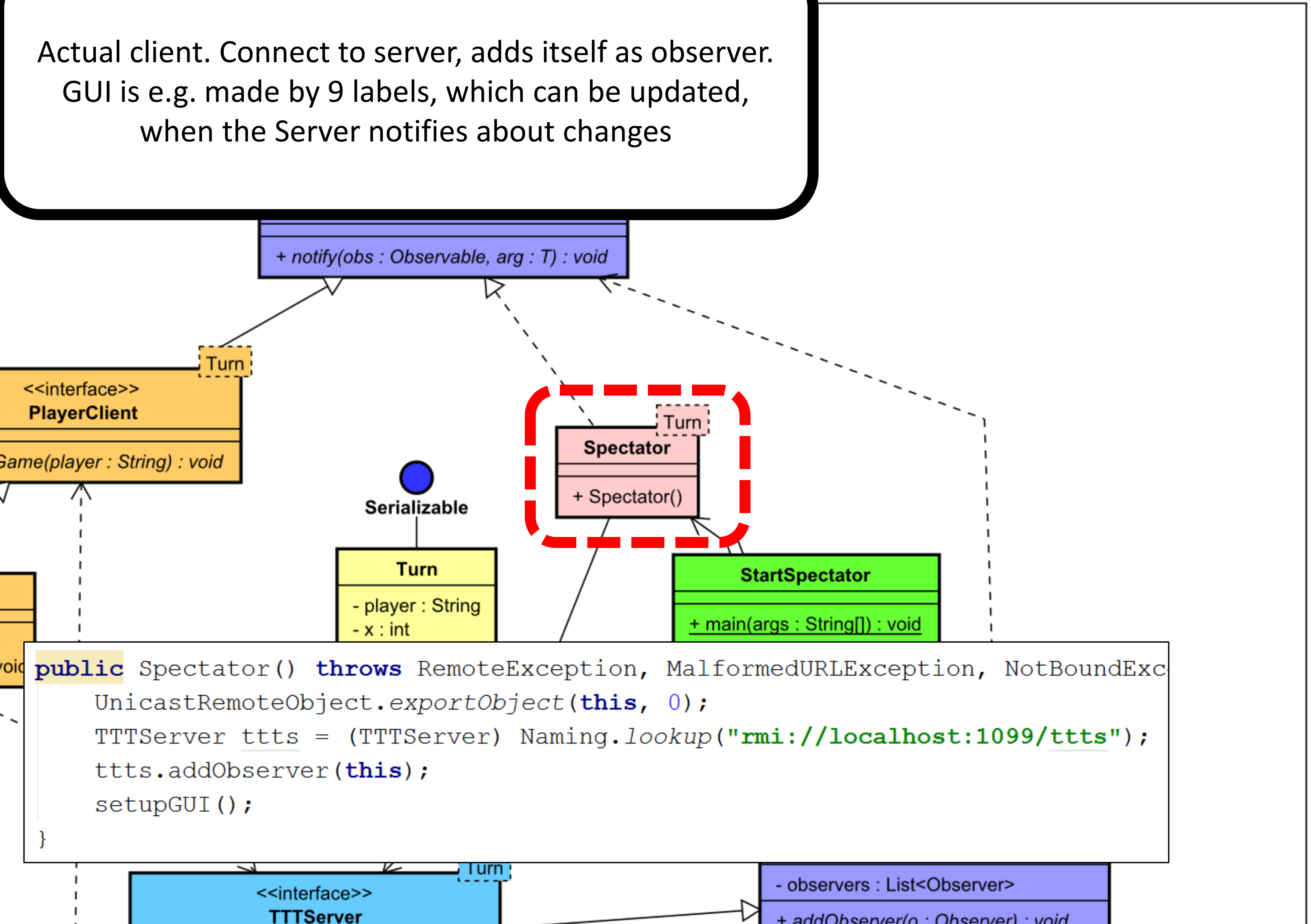
Spectator client



Class with main method to start the client



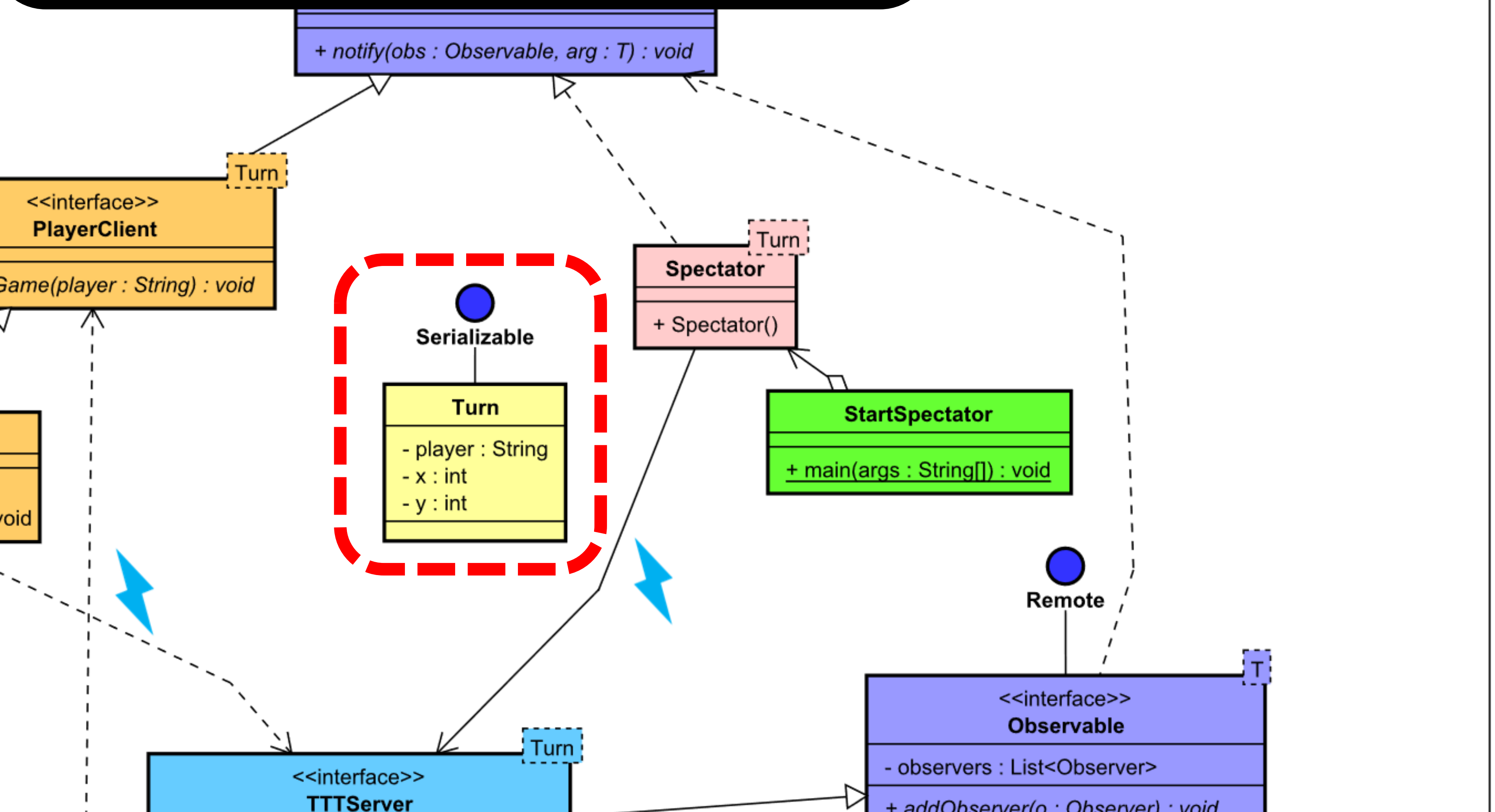
Actual client. Connect to server, adds itself as observer. GUI is e.g. made by 9 labels, which can be updated, when the Server notifies about changes



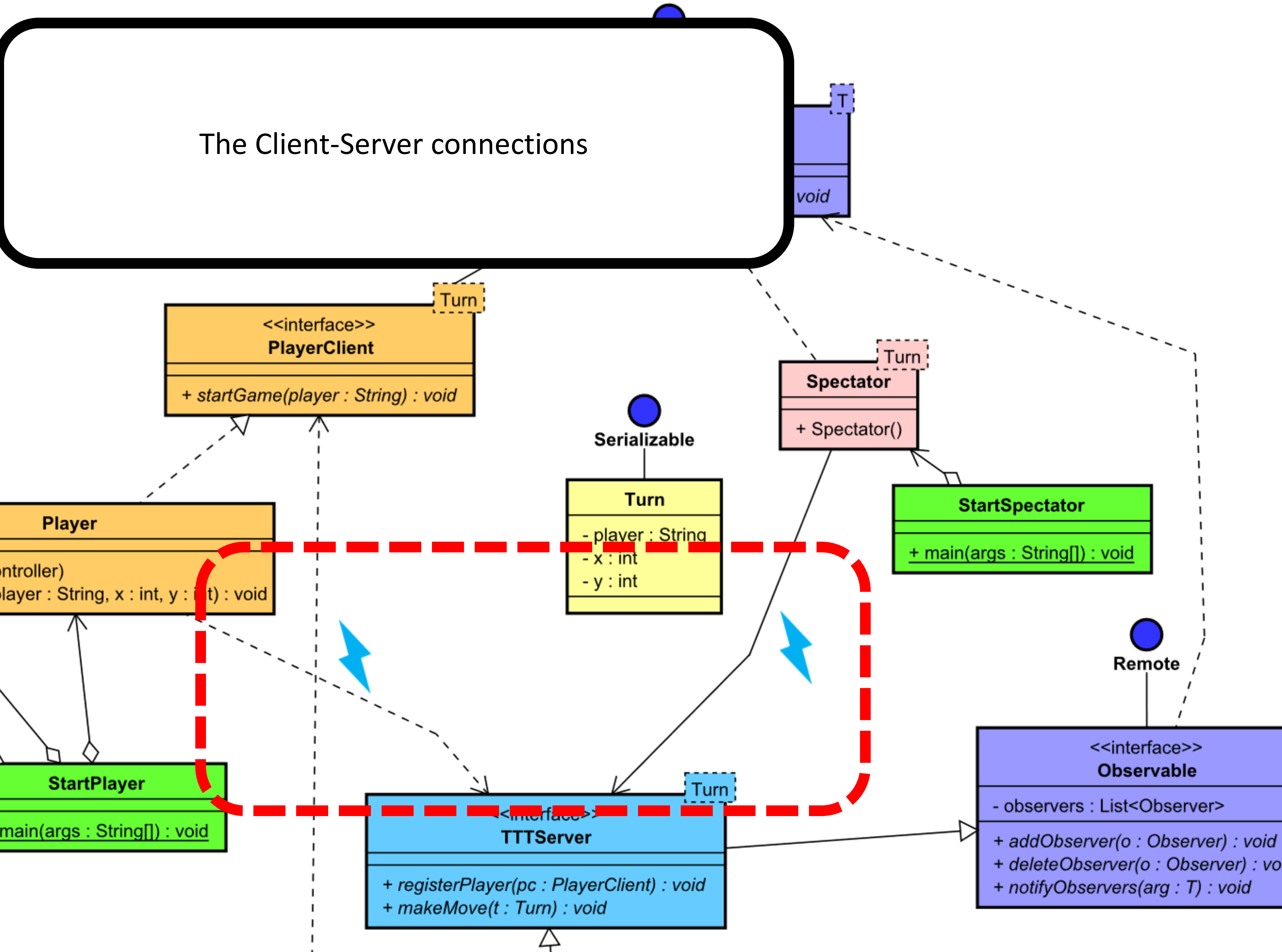
+ notify(obs : Observable, arg : T) : void
Turn
<<interface>>
PlayerClient
Game(player : String) : void
Serializable
Turn
- player : String
- x : int
Turn
Spectator
+ Spectator()
StartSpectator
+ main(args : String[]) : void
Turn
<<interface>>
TTTServer
- observers : List<Observer>
+ addObserver(o : Observer) : void

```
public Spectator() throws RemoteException, MalformedURLException, NotBoundExc
    UnicastRemoteObject.exportObject(this, 0);
    TTTServer ttts = (TTTServer) Naming.lookup("rmi://localhost:1099/ttts");
    ttts.addObserver(this);
    setupGUI();
}
```

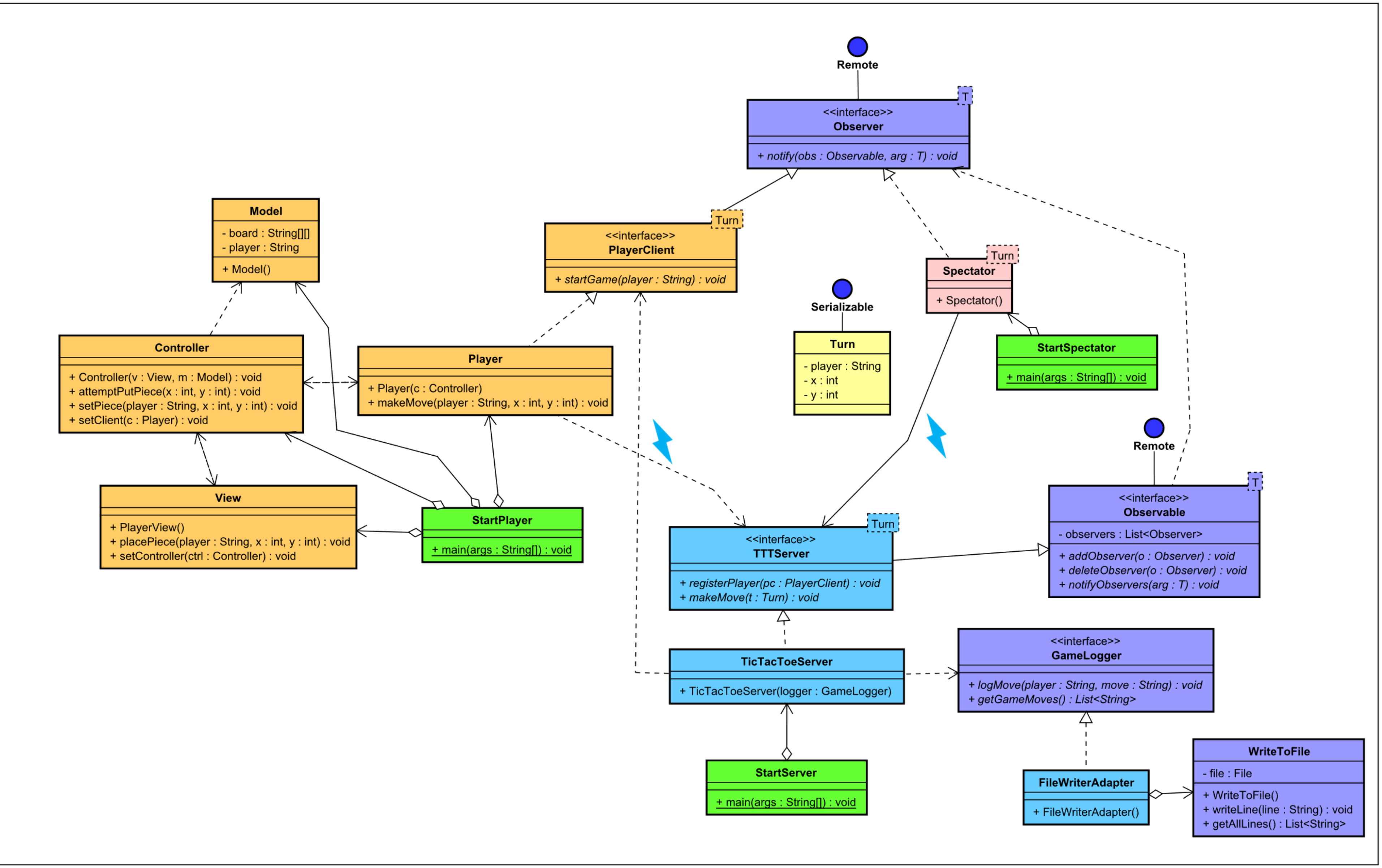
Turn class. Used to send information between Clients and Server.
Implements Serializable
Just contains three fields of information.



The Client-Server connections

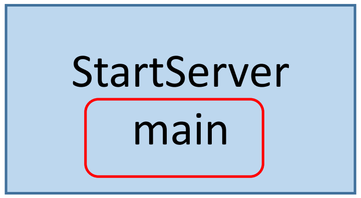


T
void
Turn
<<interface>>
PlayerClient
+ startGame(player : String) : void
Turn
Spectator
+ Spectator()
Serializable
Turn
- player : String
- x : int
- y : int
StartSpectator
+ main(args : String[]) : void
Player
ntroller)
layer : String, x : int, y : int) : void
Remote
StartPlayer
main(args : String[]) : void
Turn
<<interface>>
TTTServer
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
<<interface>>
Observable
- observers : List<Observer>
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : vo
+ notifyObservers(arg : T) : void

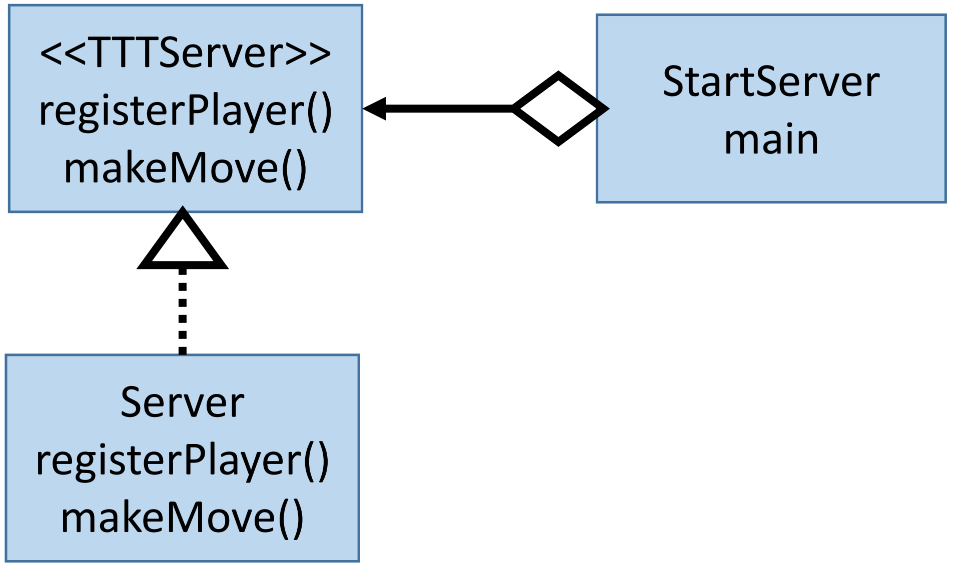

Remote
<<interface>>
Observer
T
+ notify(obs : Observable, arg : T) : void
Model
- board : String[][]
- player : String
+ Model()
<<interface>>
PlayerClient
Turn
+ startGame(player : String) : void
Spectator
Turn
+ Spectator()
Serializable
Turn
- player : String
- x : int
- y : int
StartSpectator
+ main(args : String[]) : void
Controller
+ Controller(v : View, m : Model) : void
+ attemptPutPiece(x : int, y : int) : void
+ setPiece(player : String, x : int, y : int) : void
+ setClient(c : Player) : void
Player
+ Player(c : Controller)
+ makeMove(player : String, x : int, y : int) : void
Remote
<<interface>>
Observable
T
- observers : List<Observer>
+ addObserver(o : Observer) : void
+ deleteObserver(o : Observer) : void
+ notifyObservers(arg : T) : void
View
+ PlayerView()
+ placePiece(player : String, x : int, y : int) : void
+ setController(ctrl : Controller) : void
StartPlayer
+ main(args : String[]) : void
<<interface>>
TTTServer
Turn
+ registerPlayer(pc : PlayerClient) : void
+ makeMove(t : Turn) : void
TicTacToeServer
+ TicTacToeServer(logger : GameLogger)
<<interface>>
GameLogger
+ logMove(player : String, move : String) : void
+ getGameMoves() : List<String>
StartServer
+ main(args : String[]) : void
FileWriterAdapter
+ FileWriterAdapter()
WriteToFile
- file : File
+ WriteToFile()
+ writeLine(line : String) : void
+ getAllLines() : List<String>

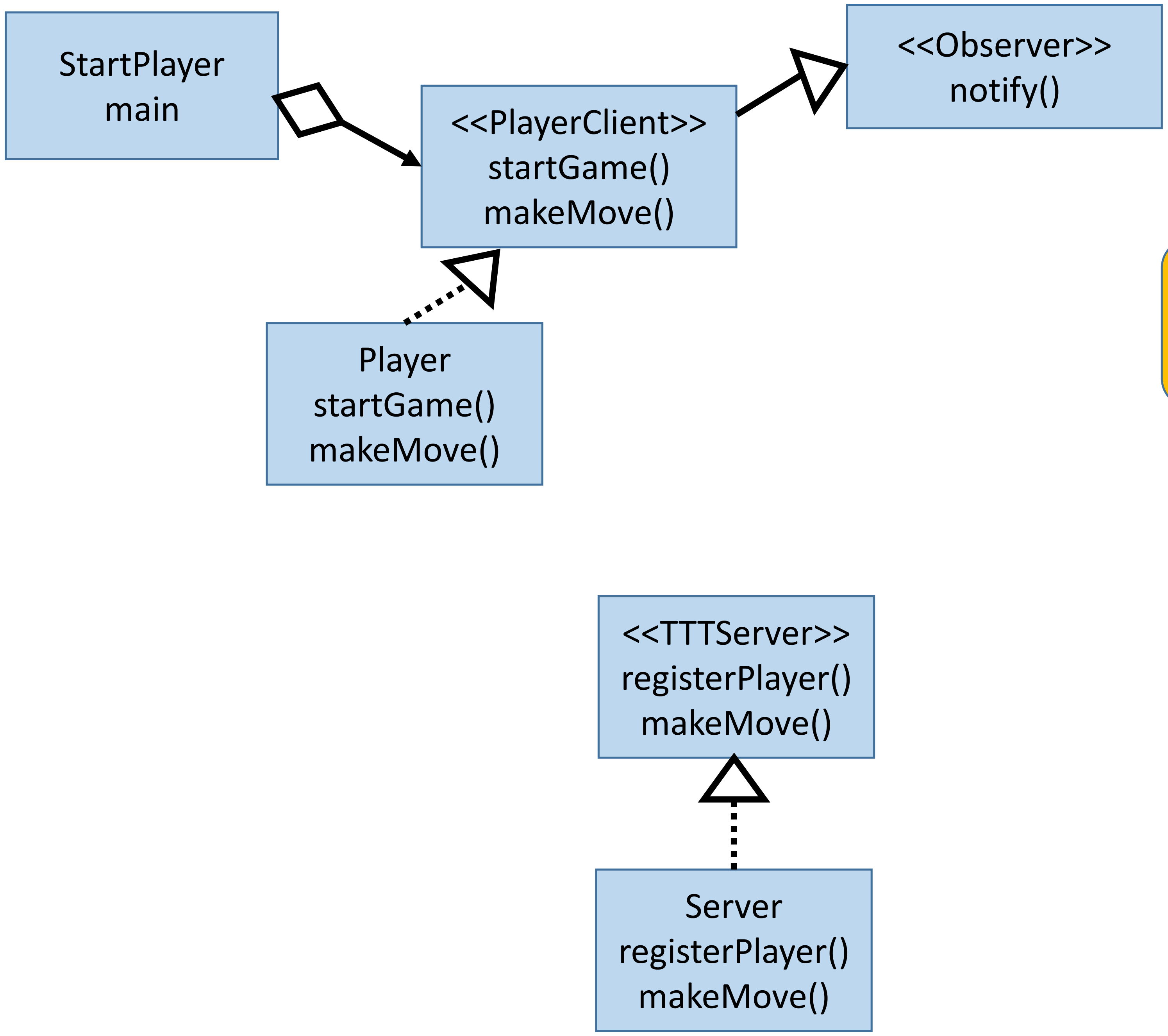
Sequence Diagram

Run main method to start server

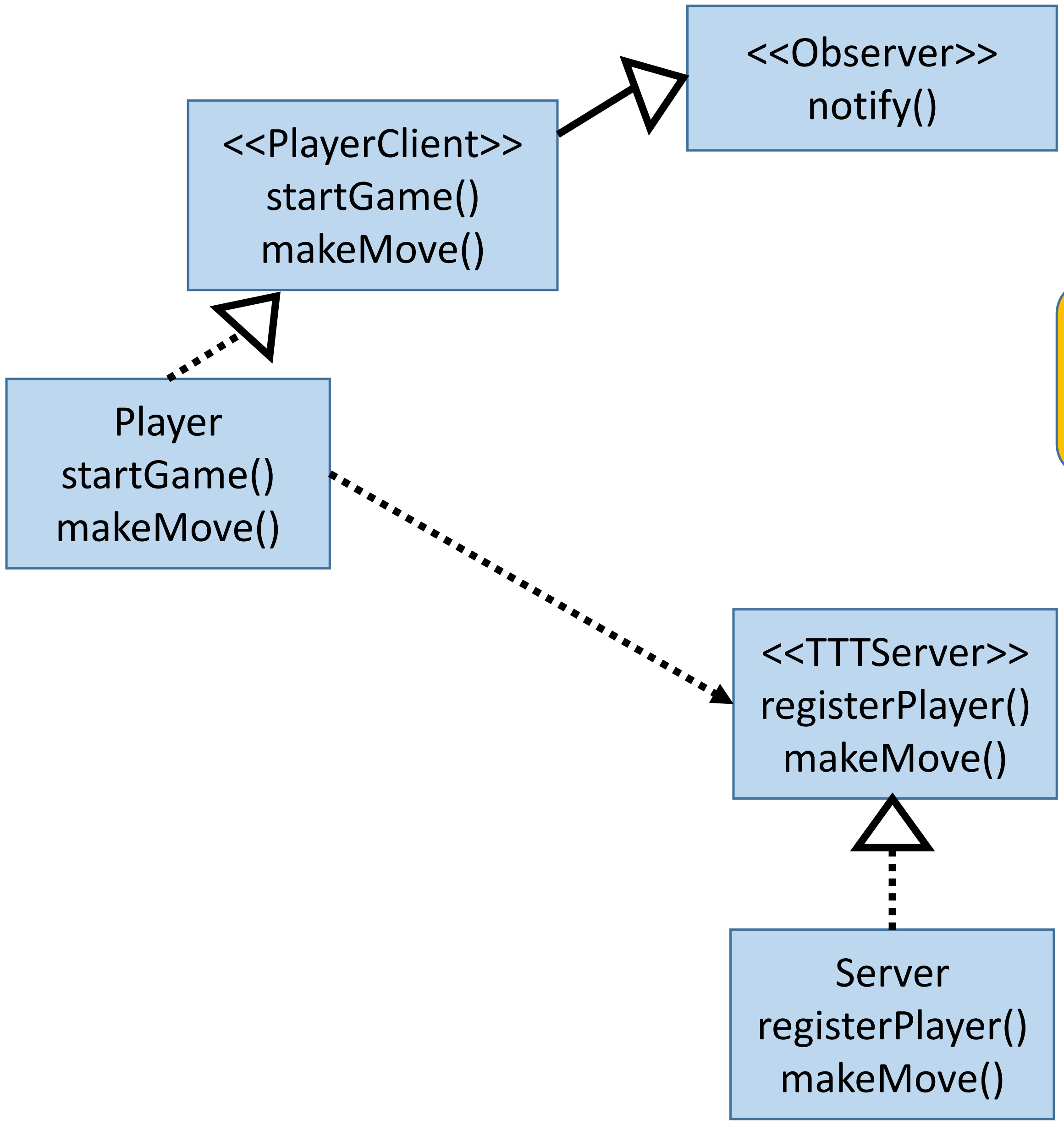


Server is created, put in registry

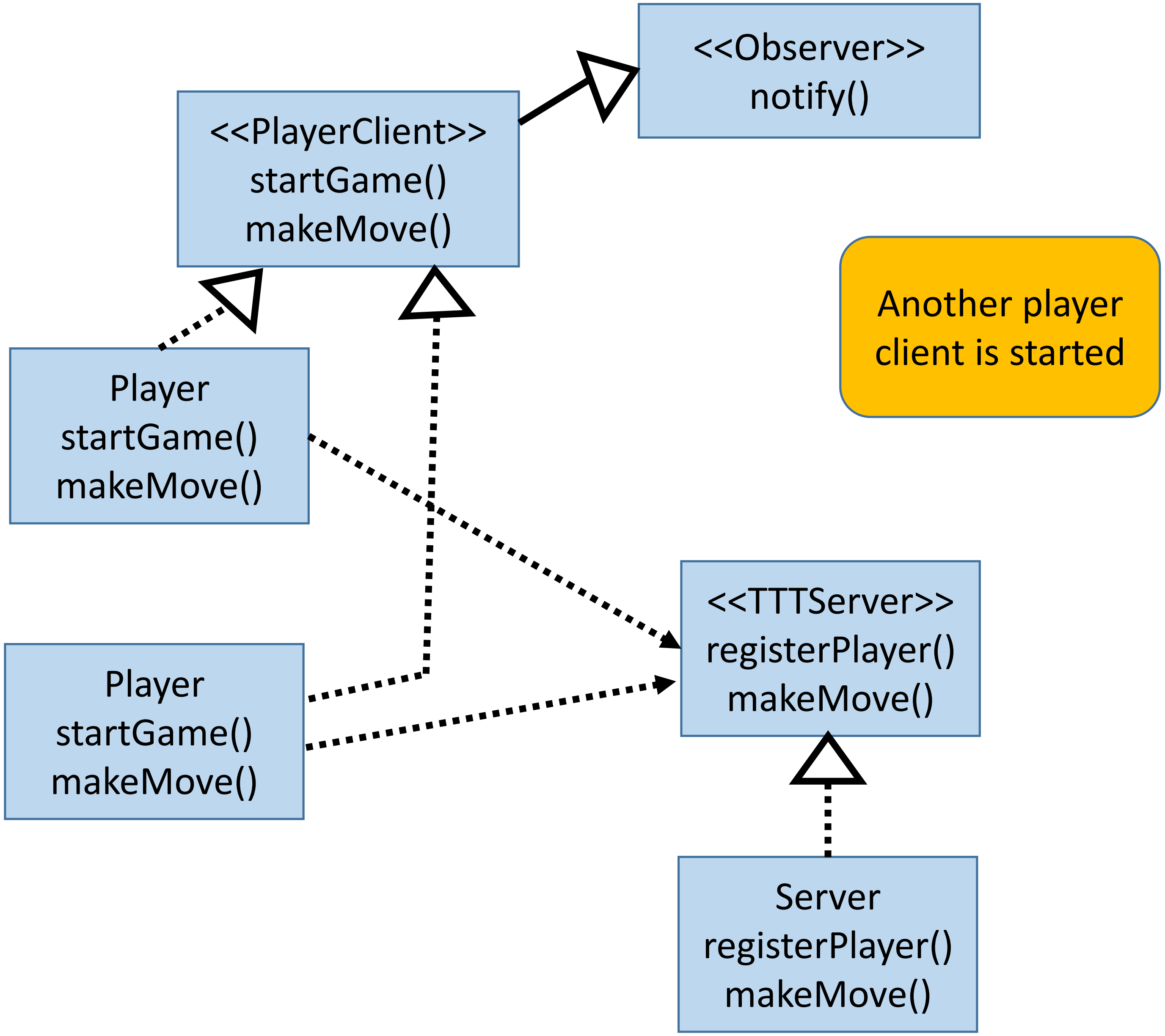




Run main method to start client

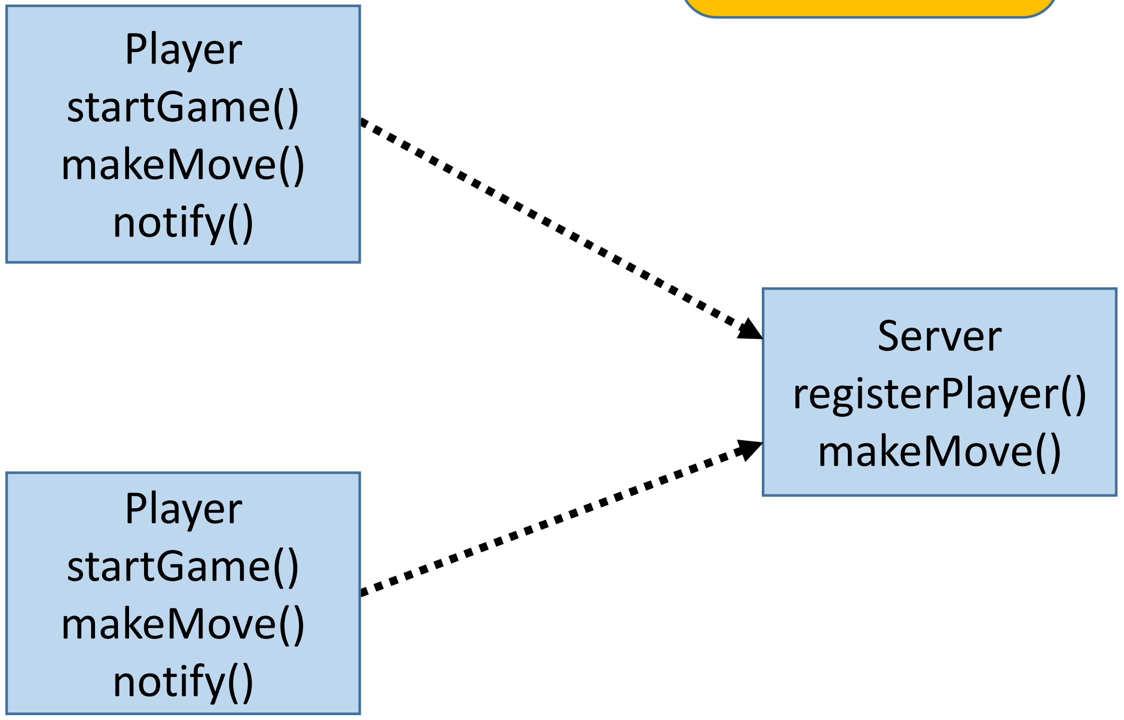


Client connects to server

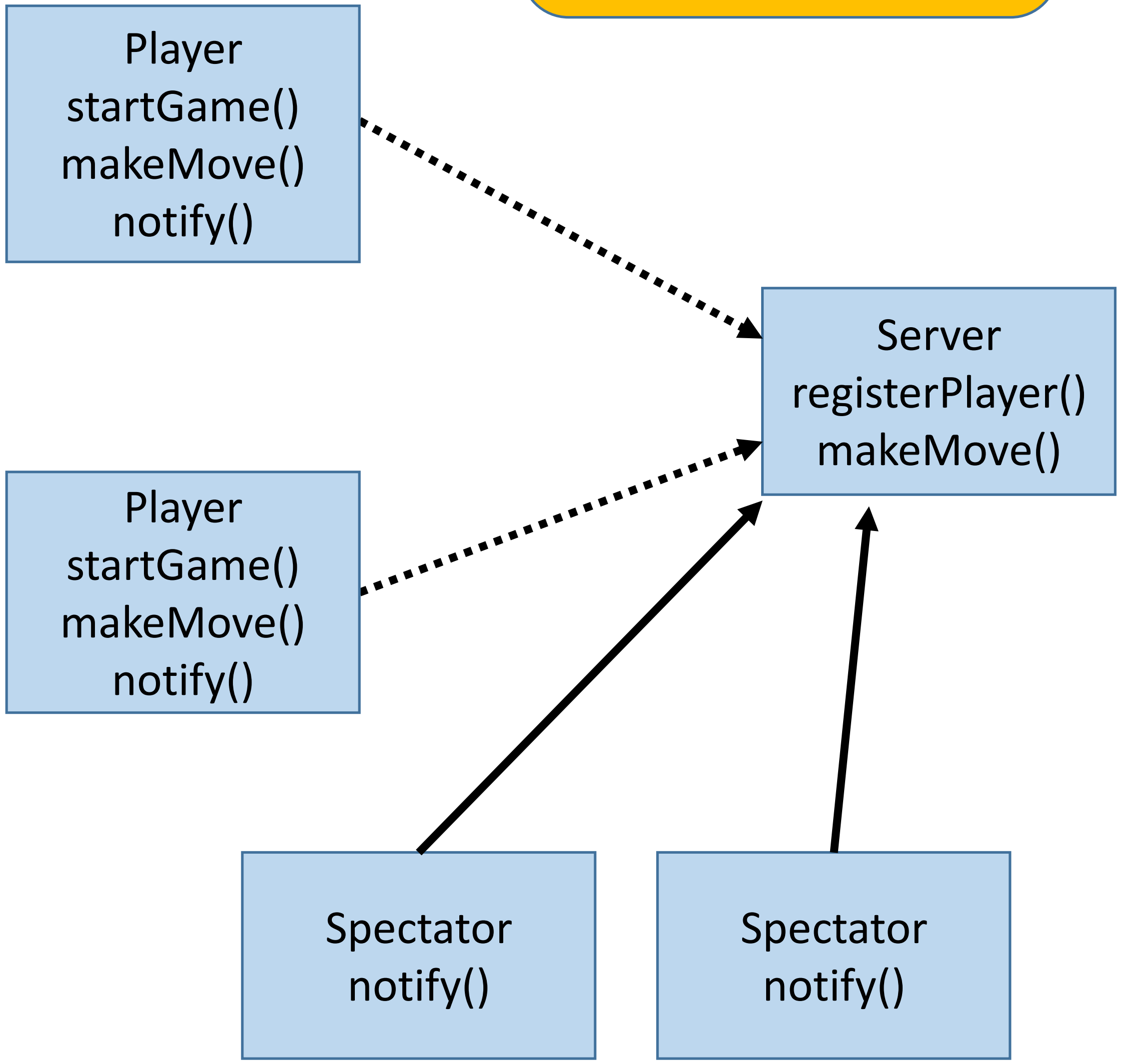


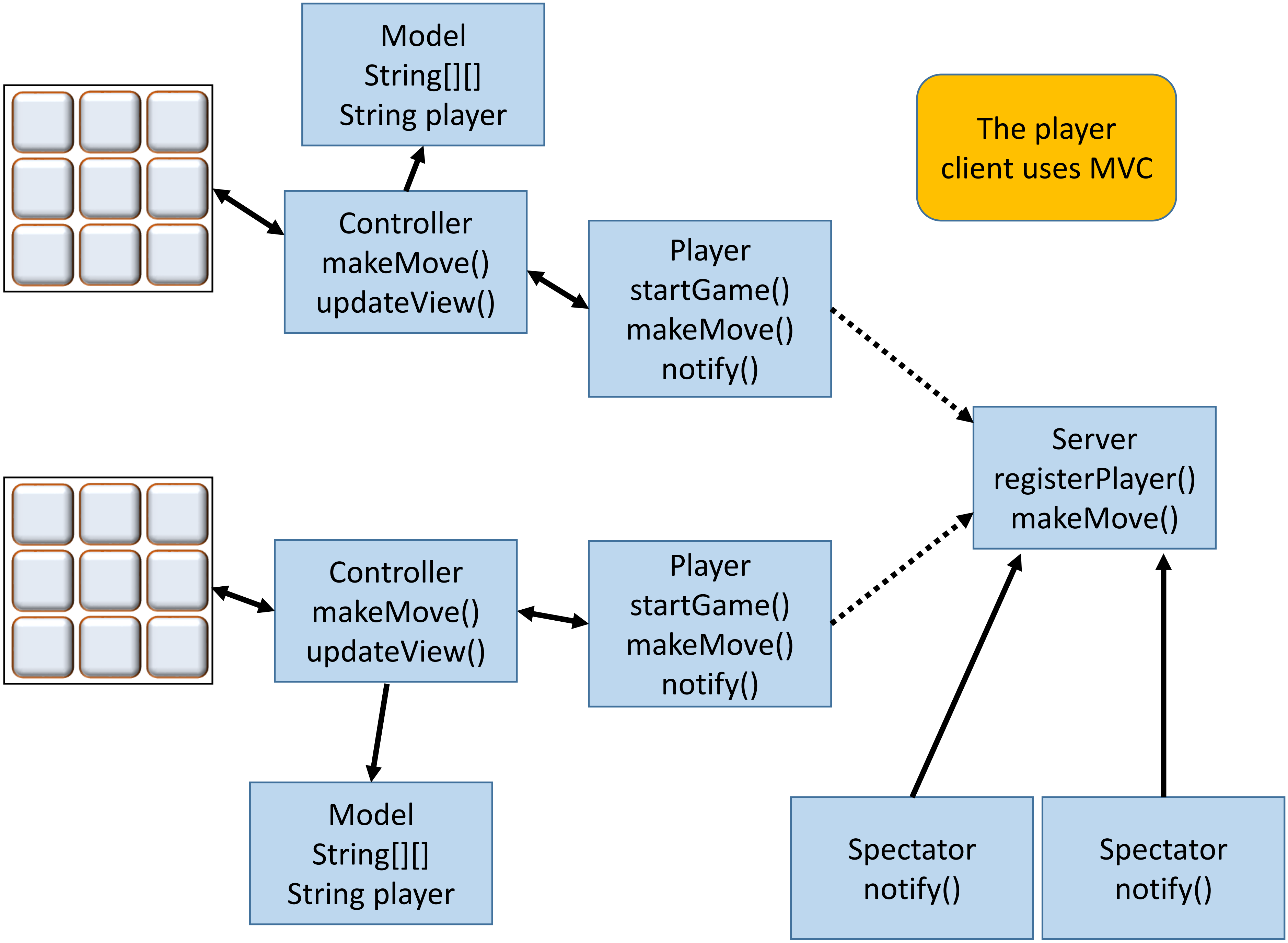
Another player client is started

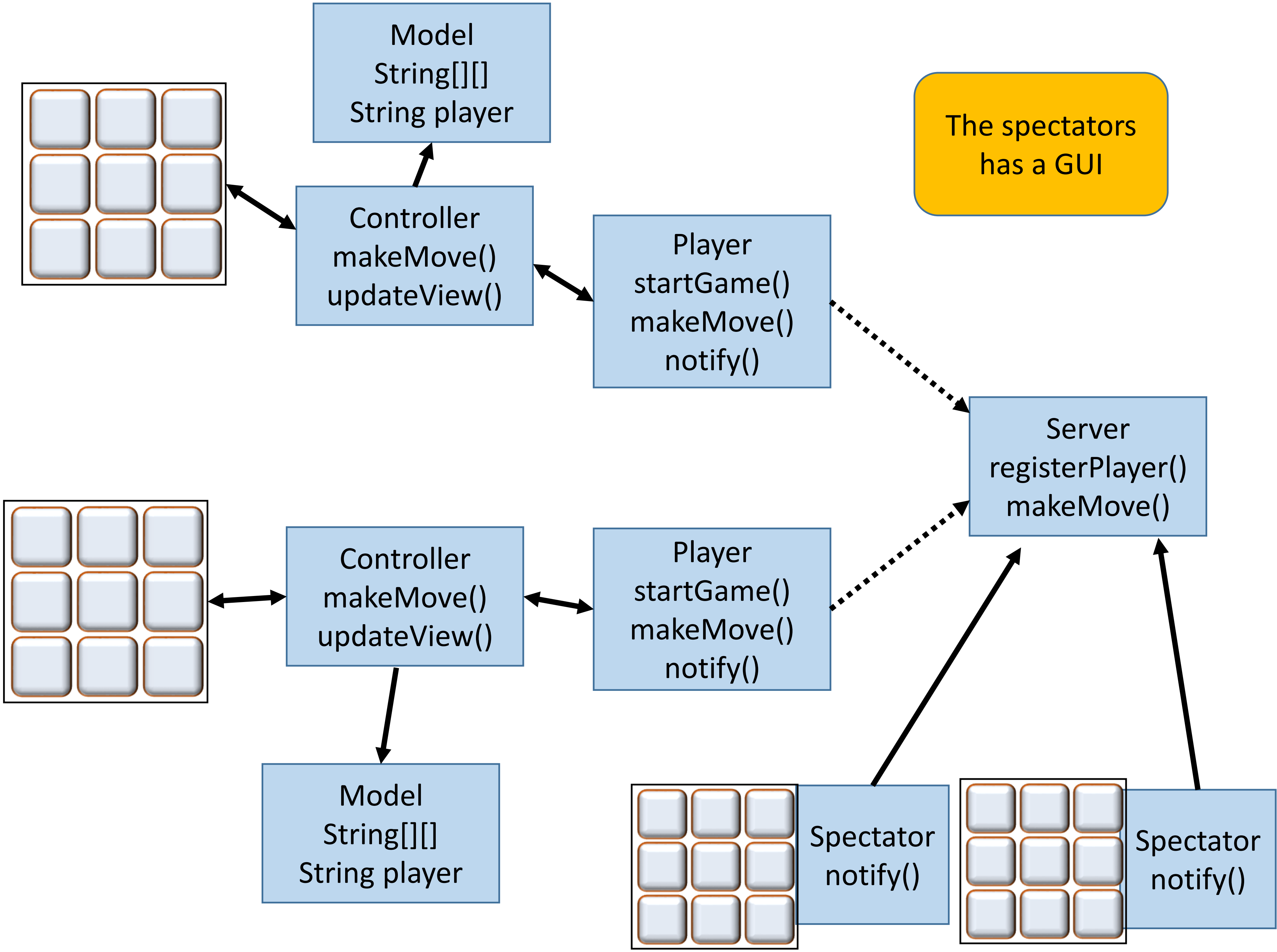
Just removing interfaces to clear up space

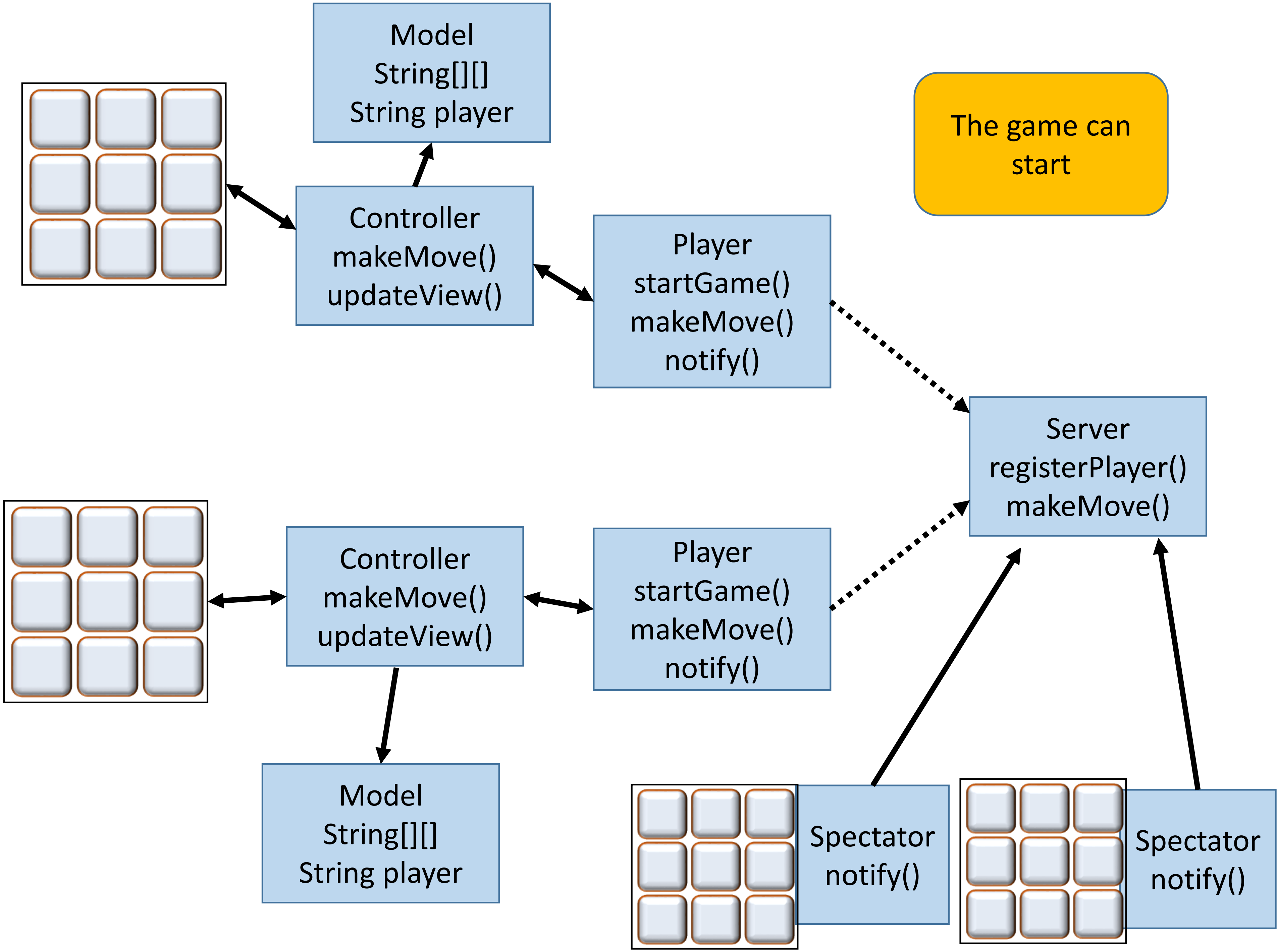


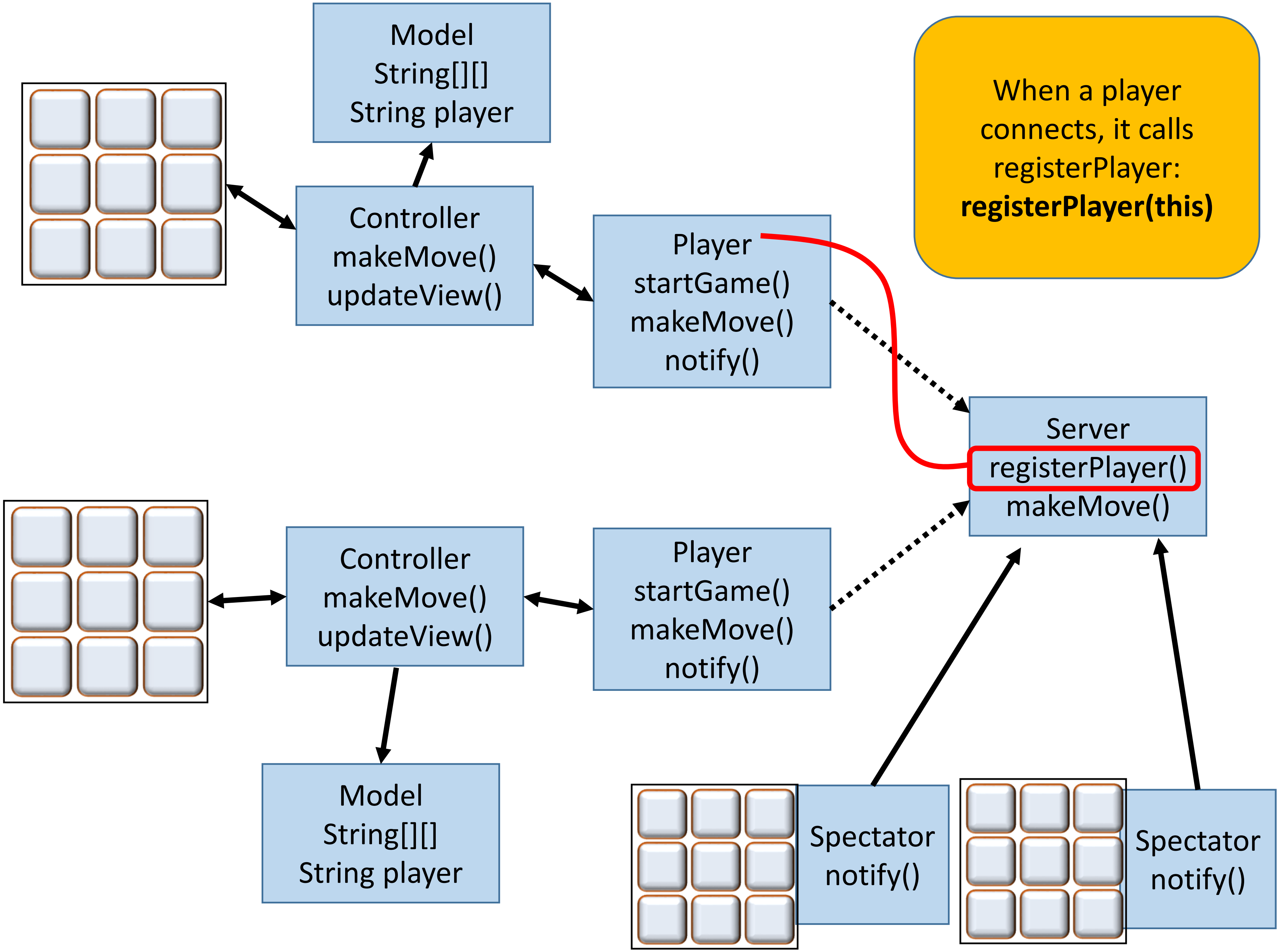
A number of observers connect, and also adds themselves as observers



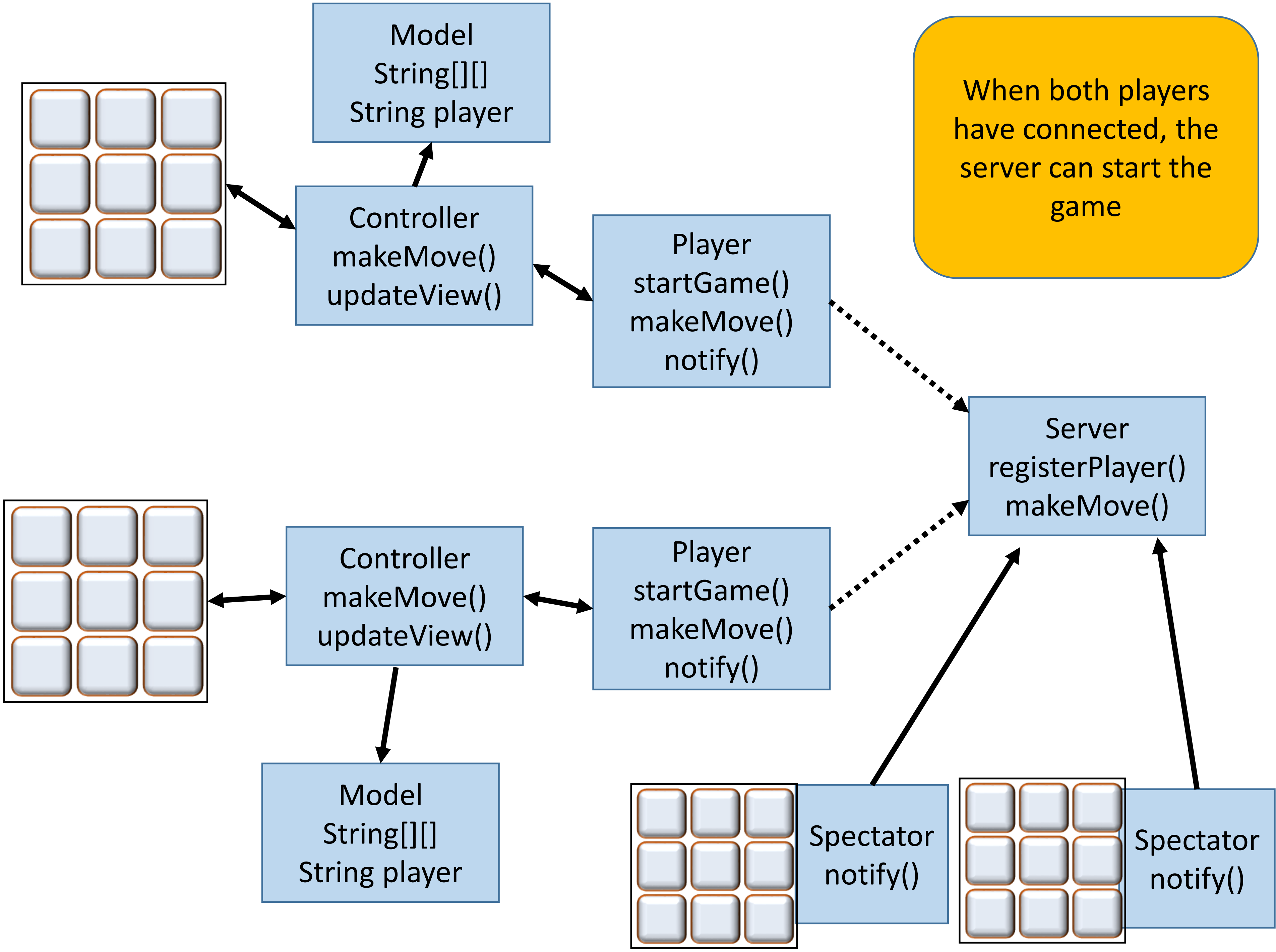

Model
String[][]
String player
The player client uses MVC
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Server
registerPlayer()
makeMove()
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Model
String[][]
String player
Spectator
notify()
Spectator
notify()


Model
String[][]
String player
The spectators has a GUI
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Server
registerPlayer()
makeMove()
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Model
String[][]
String player
Spectator
notify()
Spectator
notify()

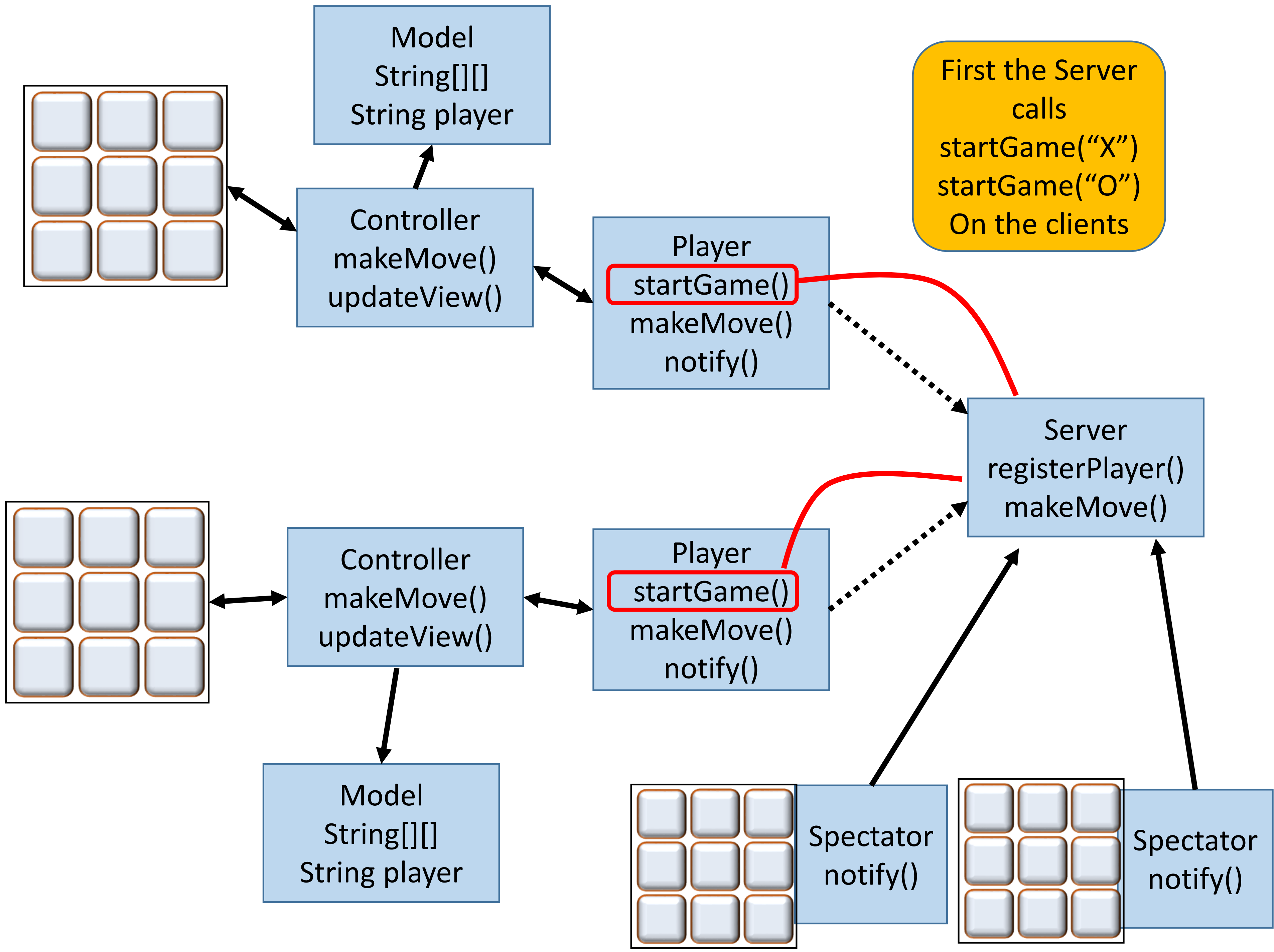

Model
String[][]
String player
The game can start
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Server
registerPlayer()
makeMove()
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Model
String[][]
String player
Spectator
notify()
Spectator
notify()

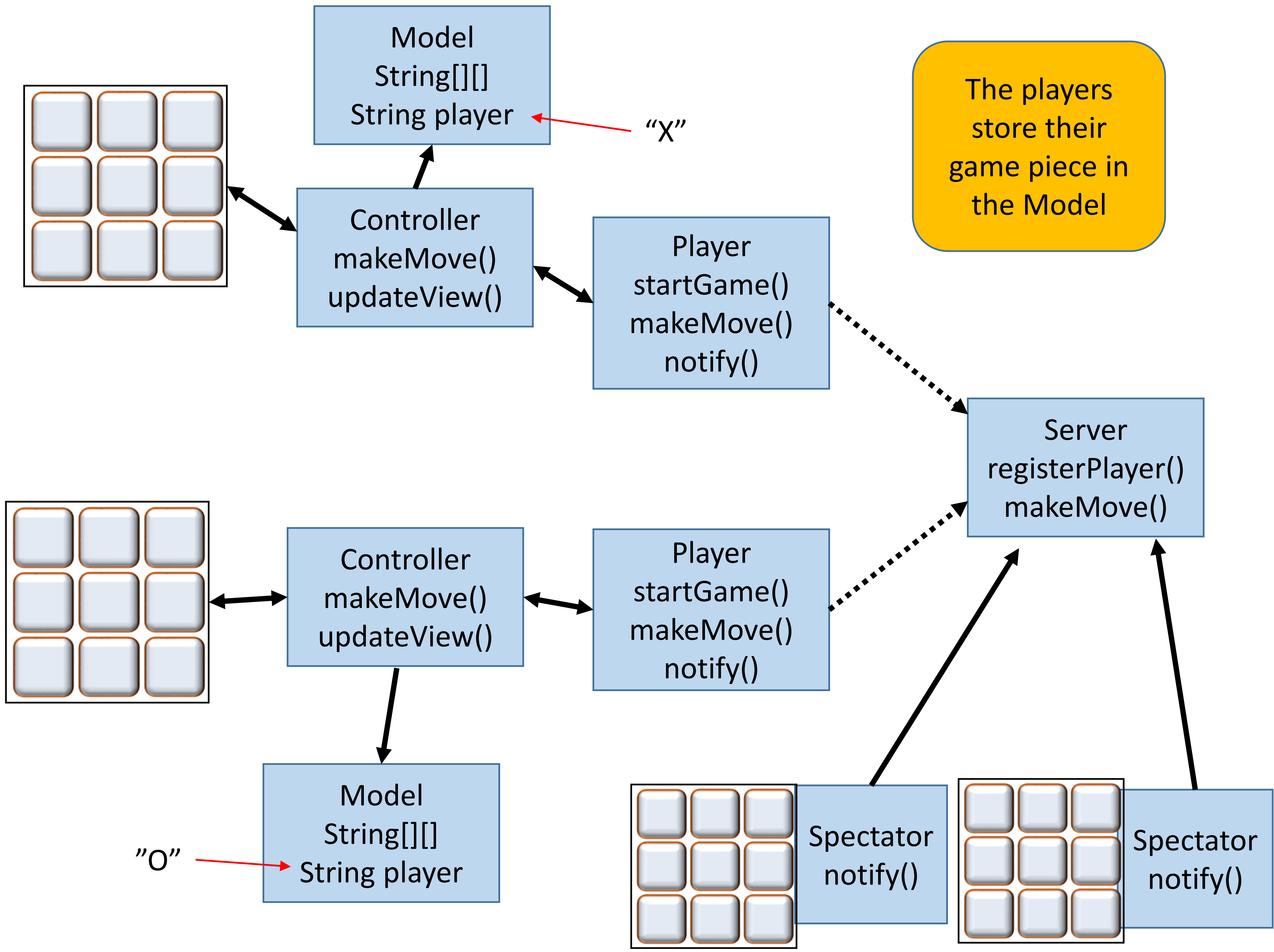


When a player connects, it calls registerPlayer:
registerPlayer(this)

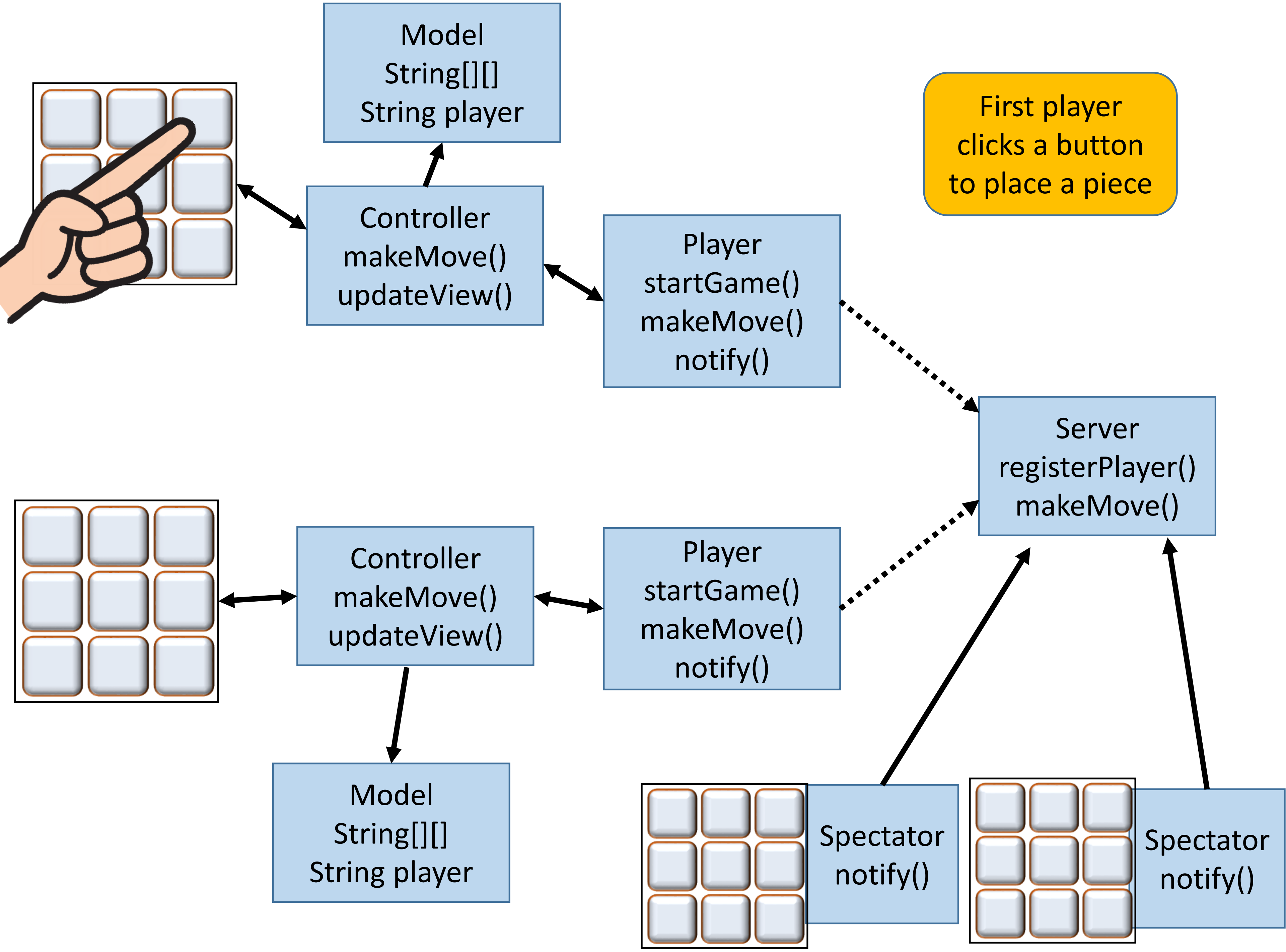


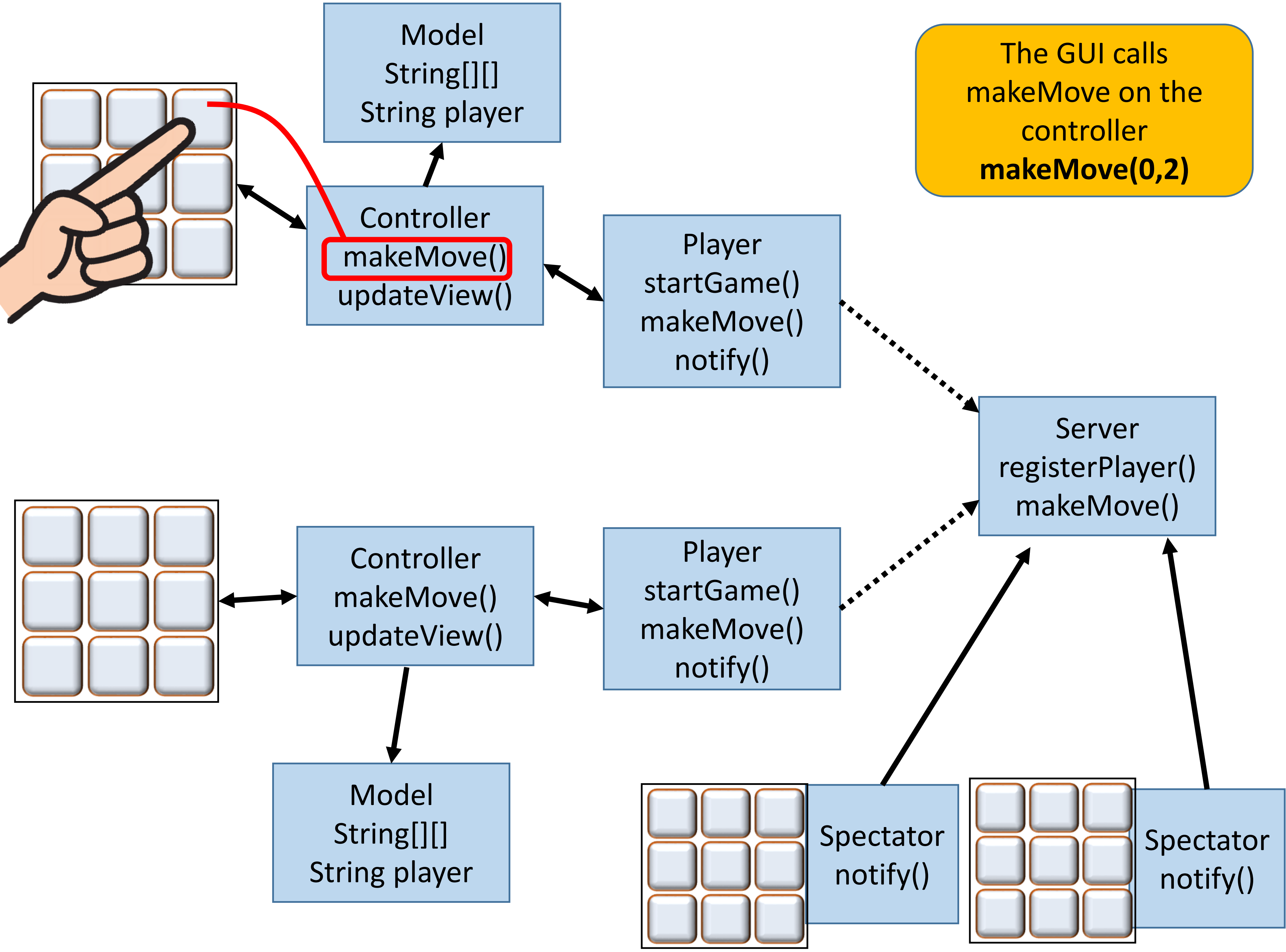
When both players have connected, the server can start the game


Model
String[][]
String player
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
First the Server calls startGame("X") startGame("O") On the clients
Server
registerPlayer()
makeMove()
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Model
String[][]
String player
Spectator
notify()
Spectator
notify()

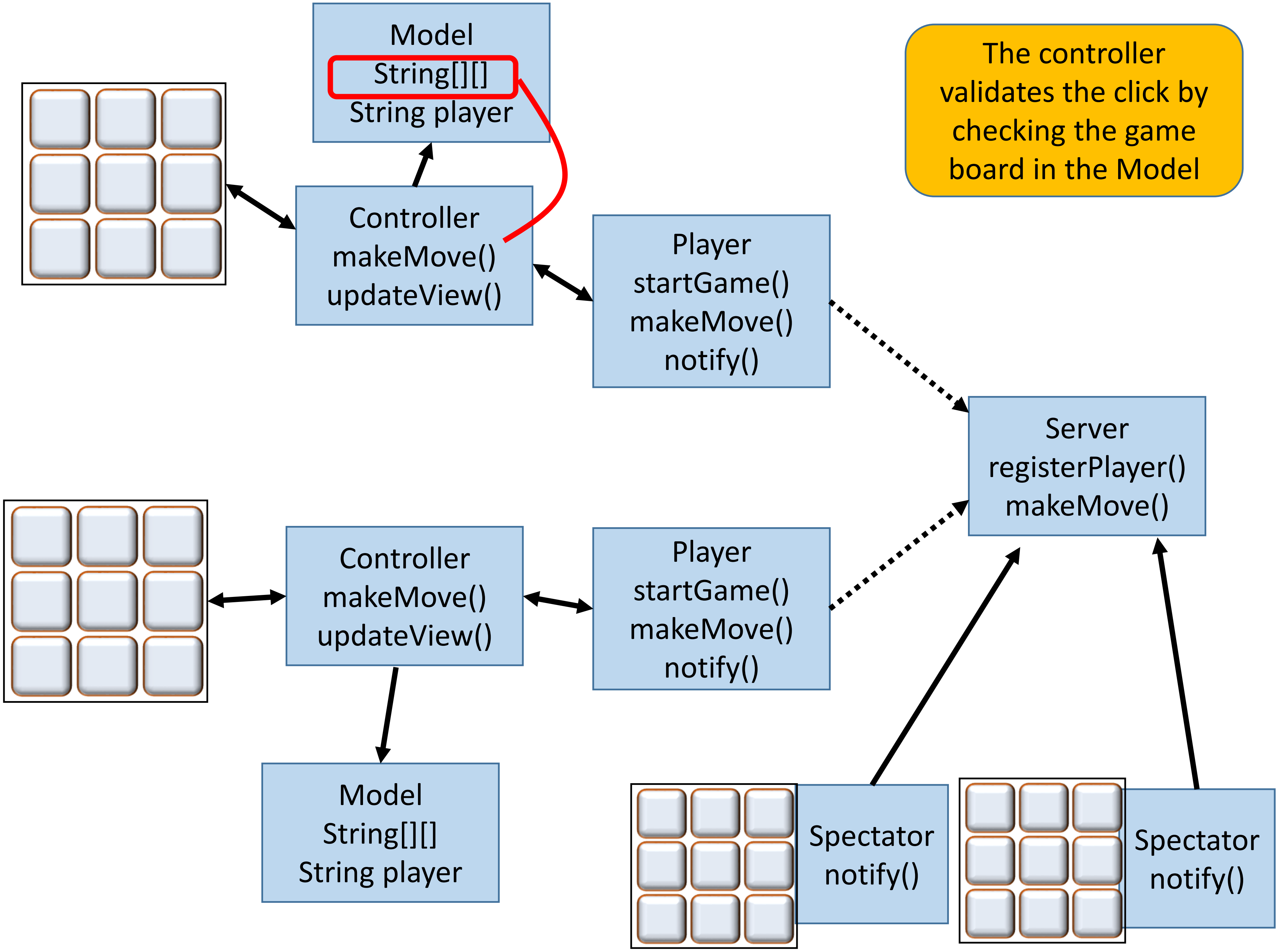


The players store their game piece in the Model

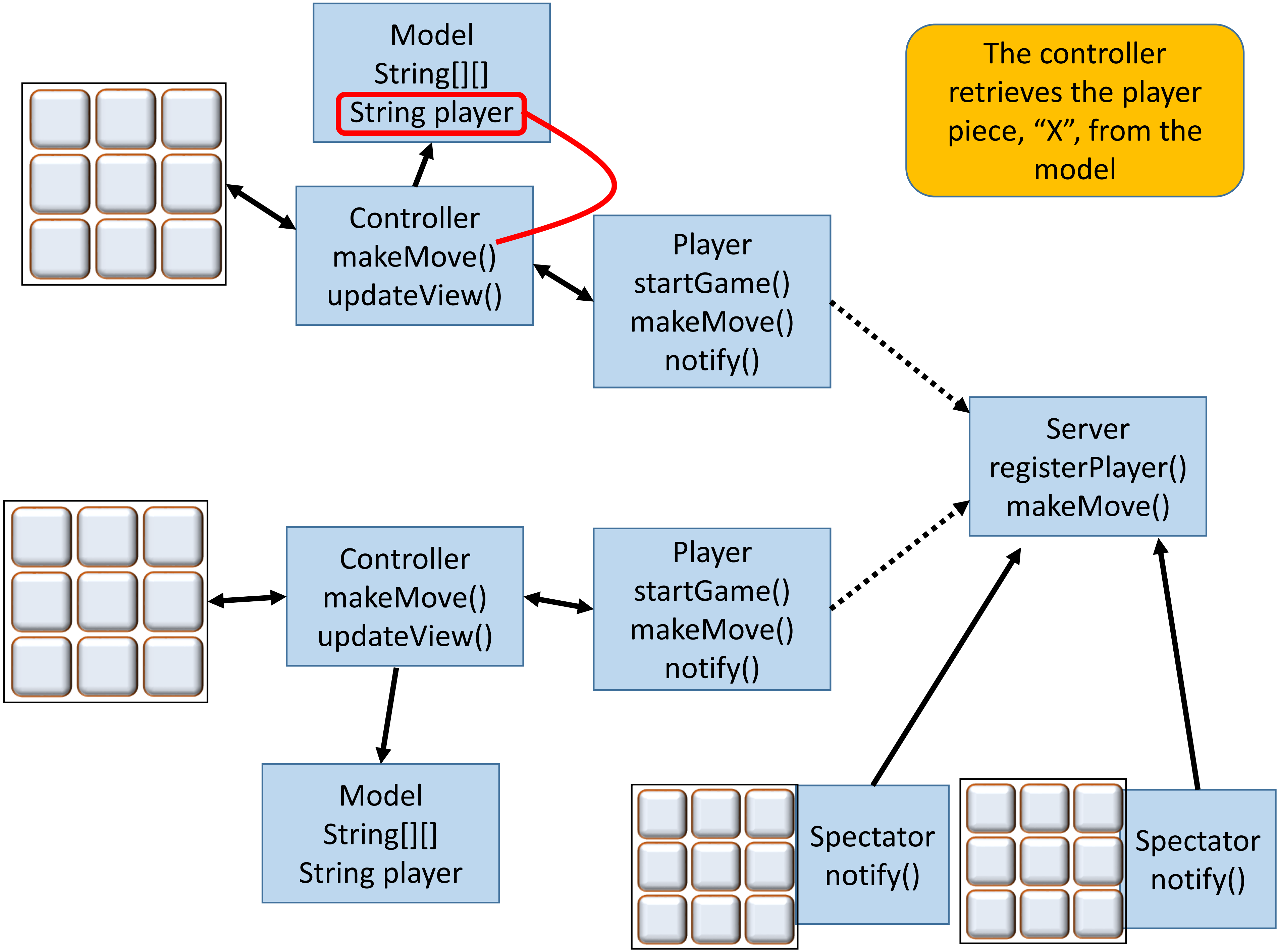

Model
String[][]
String player
First player clicks a button to place a piece
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Server
registerPlayer()
makeMove()
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Model
String[][]
String player
Spectator
notify()
Spectator
notify()



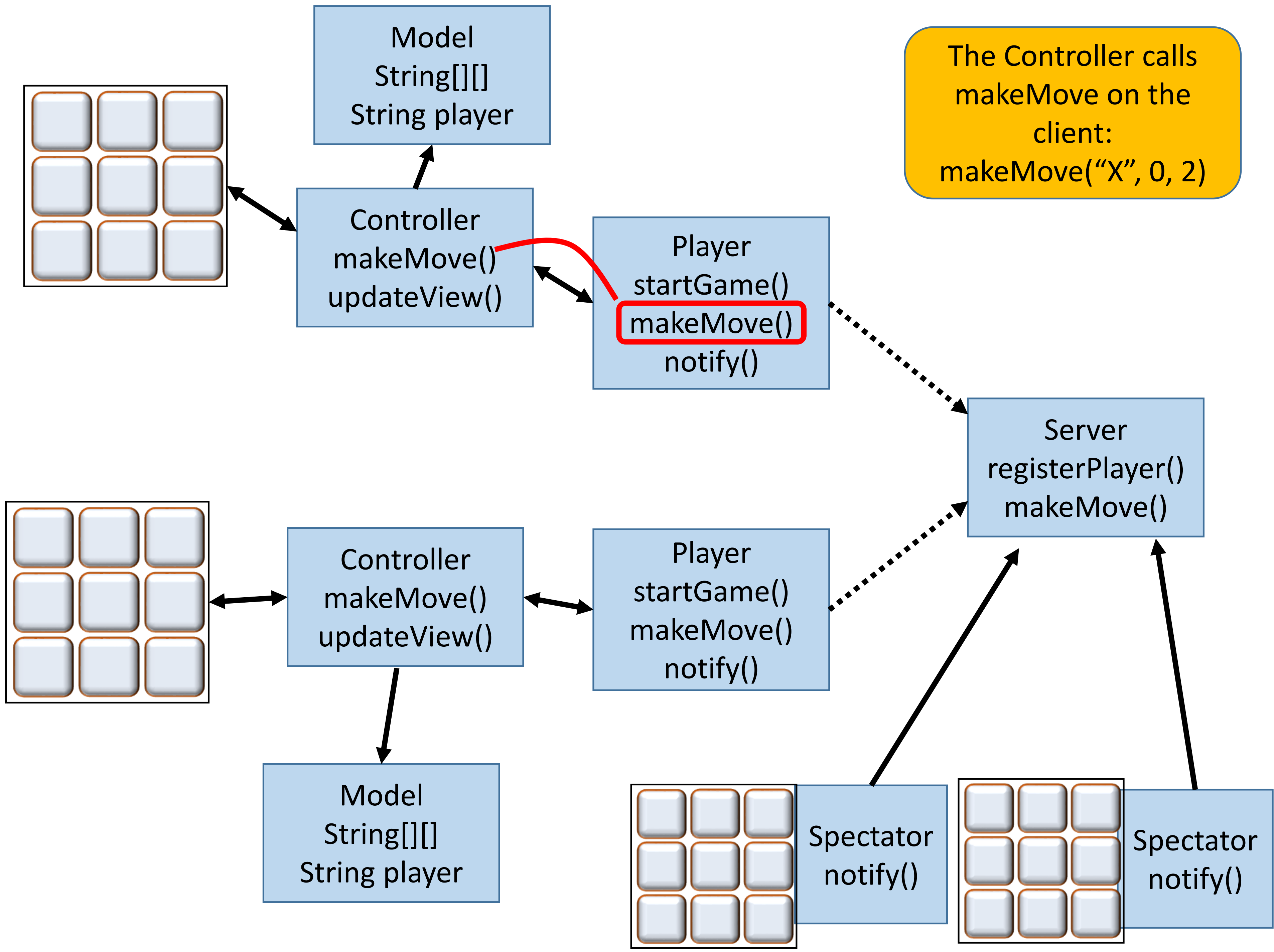
The GUI calls makeMove on the controller **makeMove(0,2)**



The controller validates the click by checking the game board in the Model

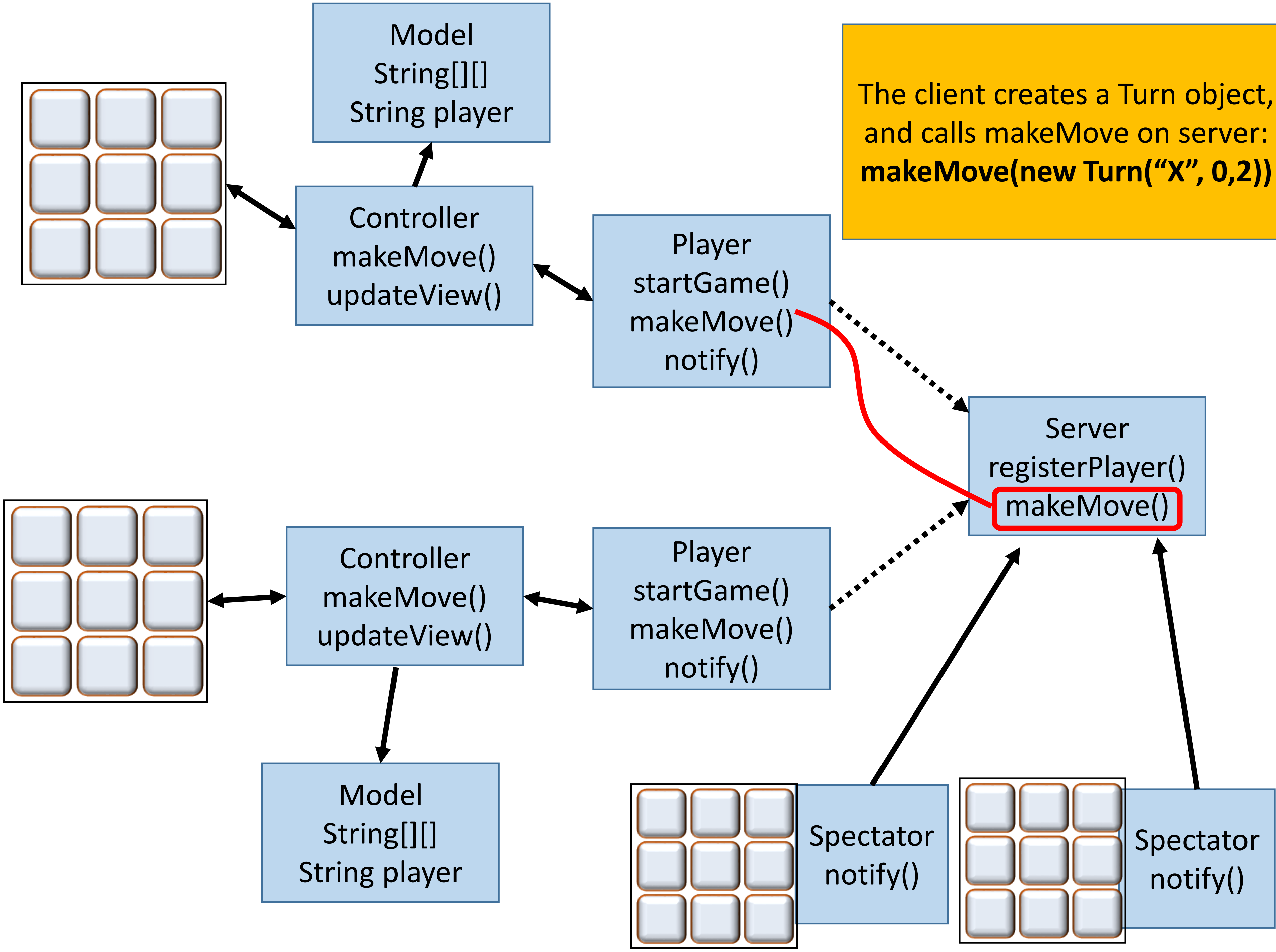


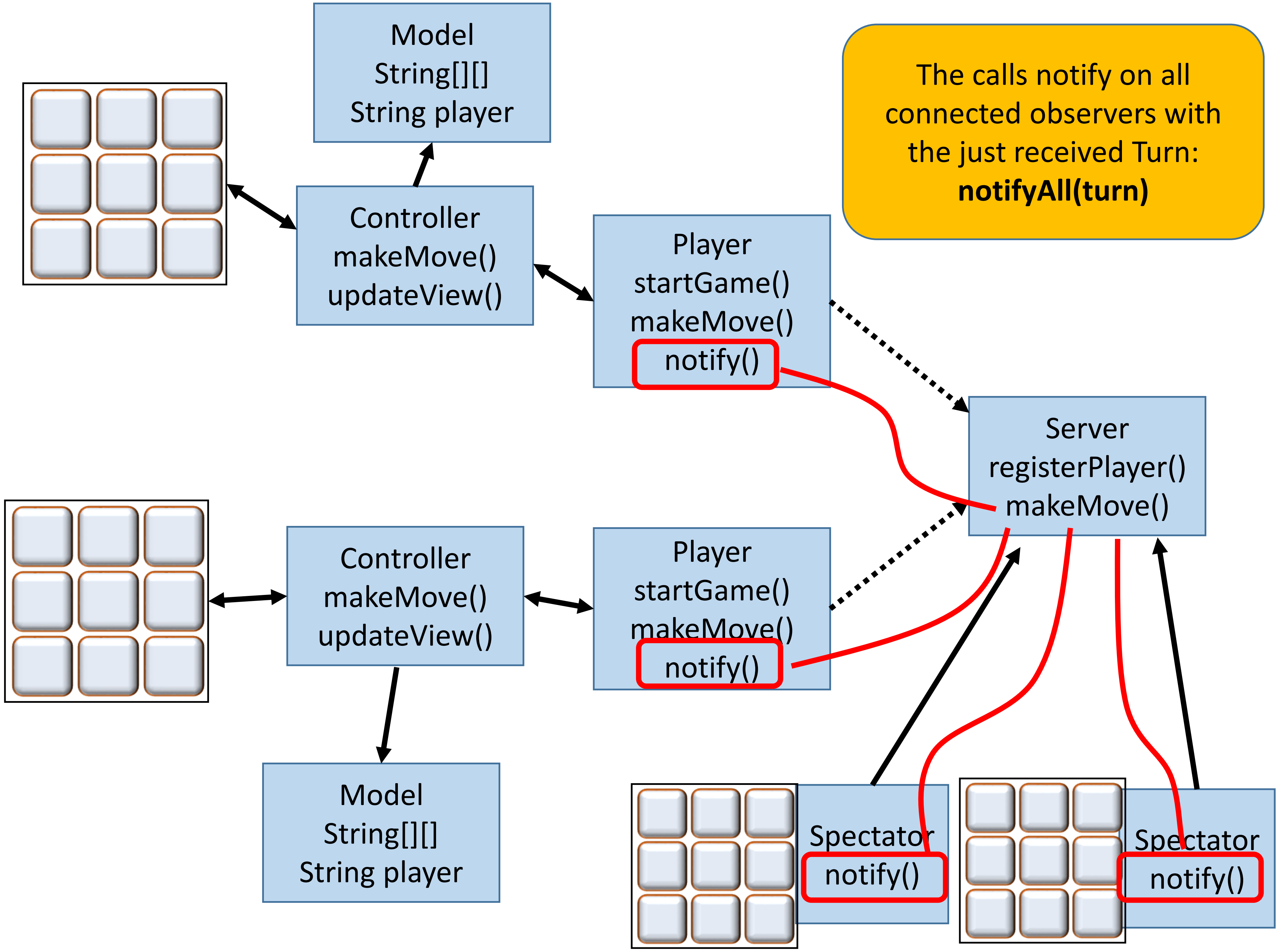
The controller retrieves the player piece, “X”, from the model



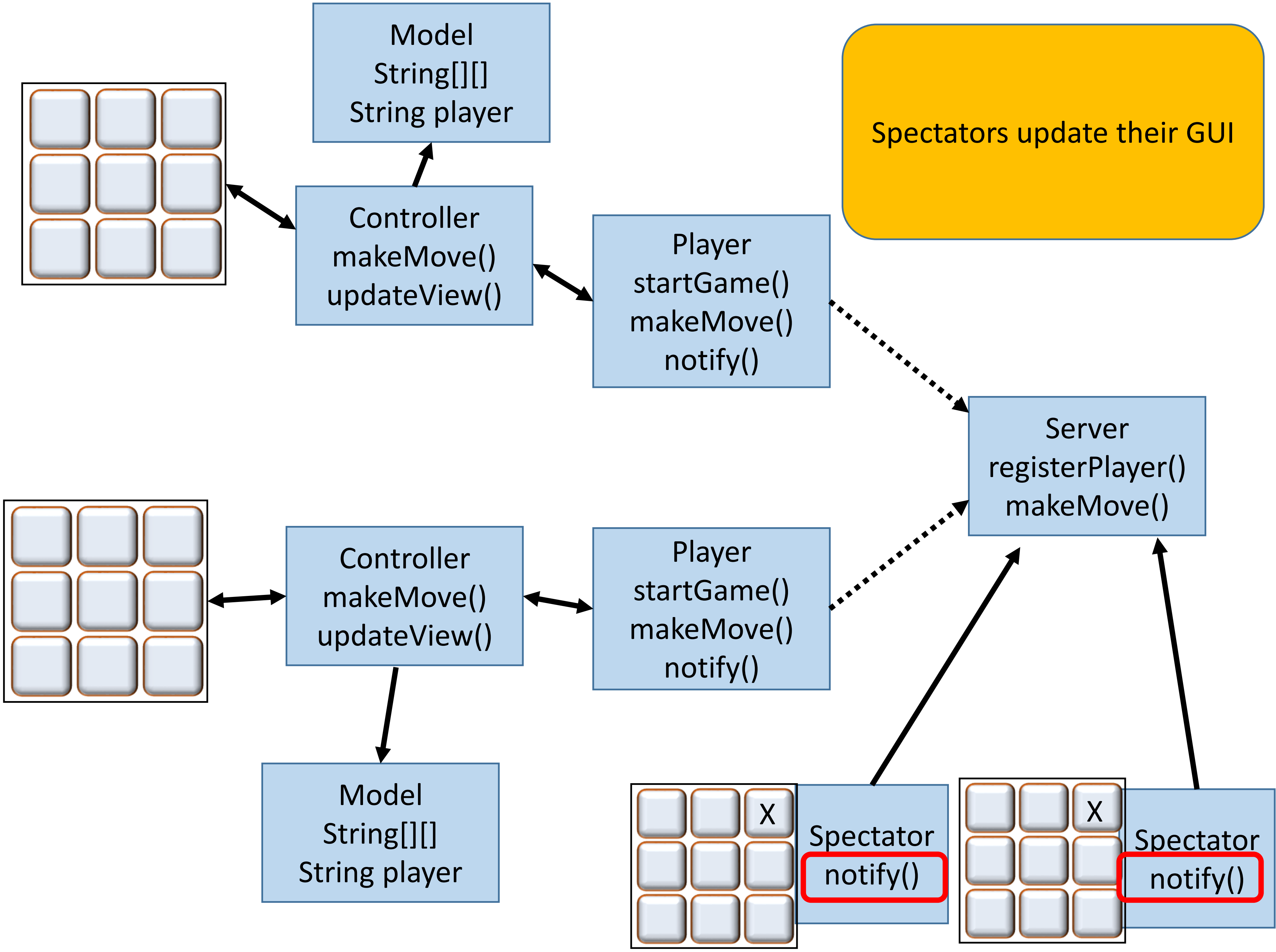
The Controller calls makeMove on the client: makeMove(“X”, 0, 2)

The client creates a Turn object, and calls makeMove on server:
makeMove(new Turn(“X”, 0,2))

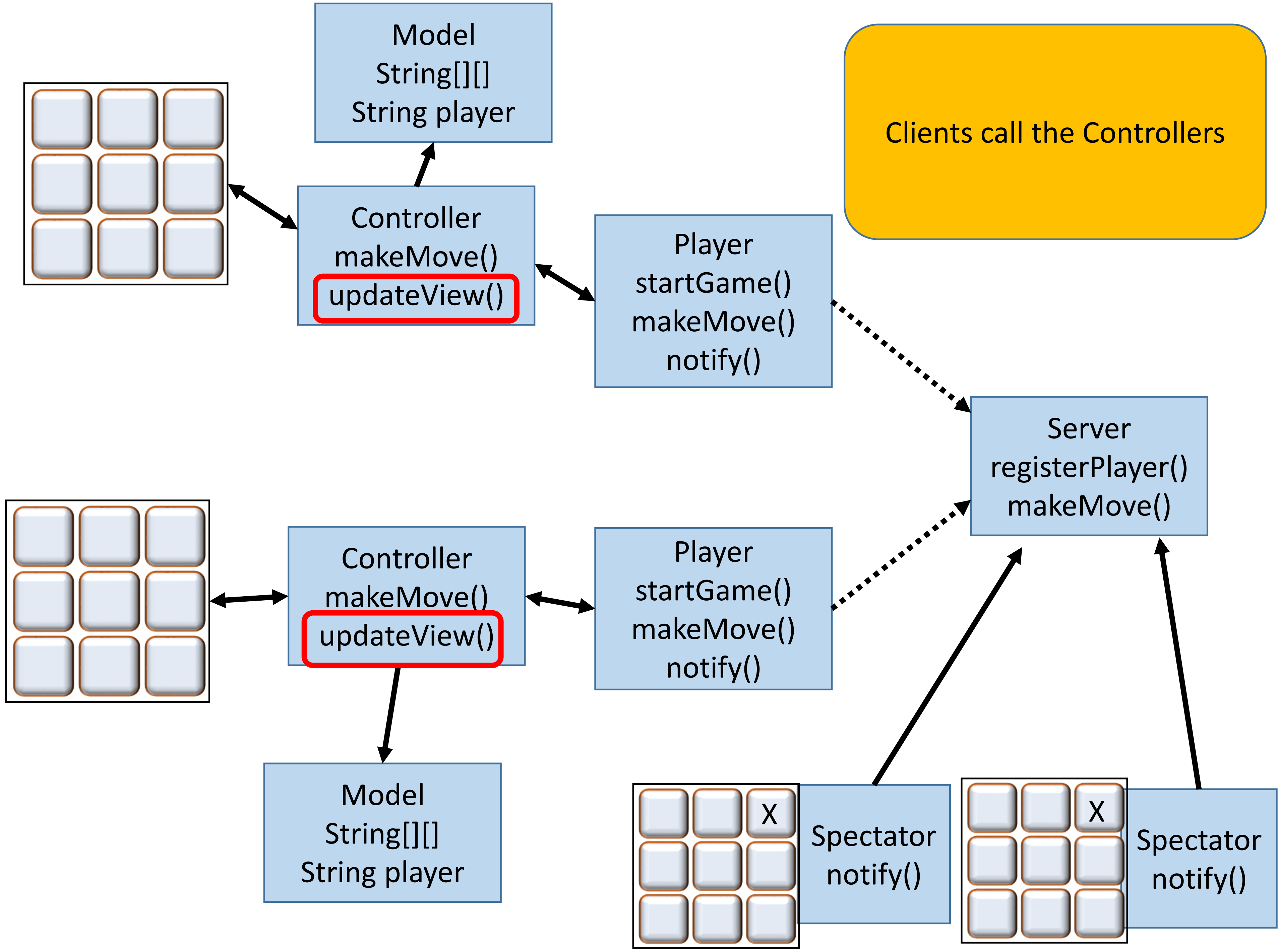


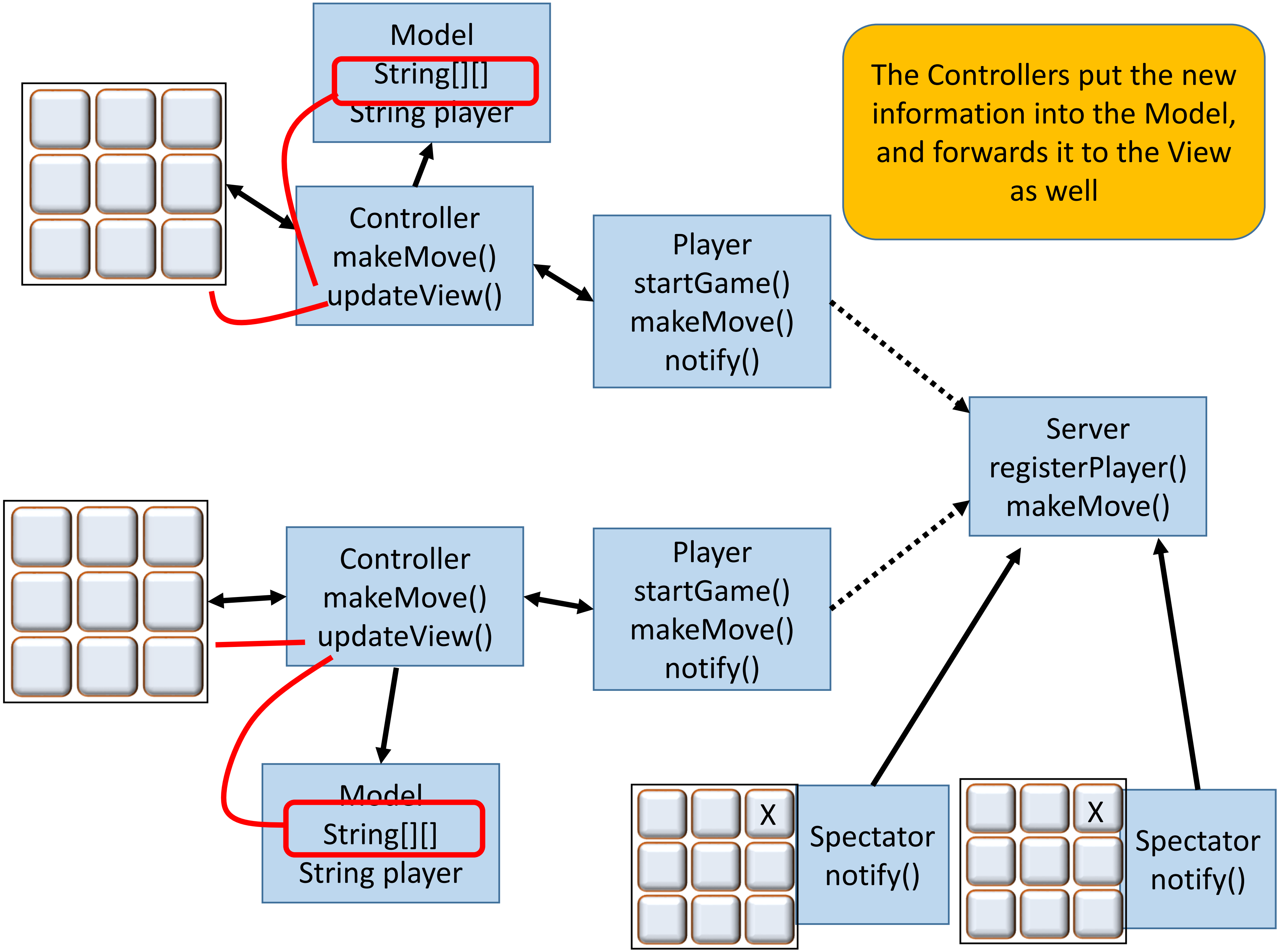


The calls notify on all connected observers with the just received Turn: **notifyAll(turn)**


Model
String[][]
String player
Spectators update their GUI
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Server
registerPlayer()
makeMove()
Controller
makeMove()
updateView()
Player
startGame()
makeMove()
notify()
Model
String[][]
String player
X
Spectator
notify()
X
Spectator
notify()

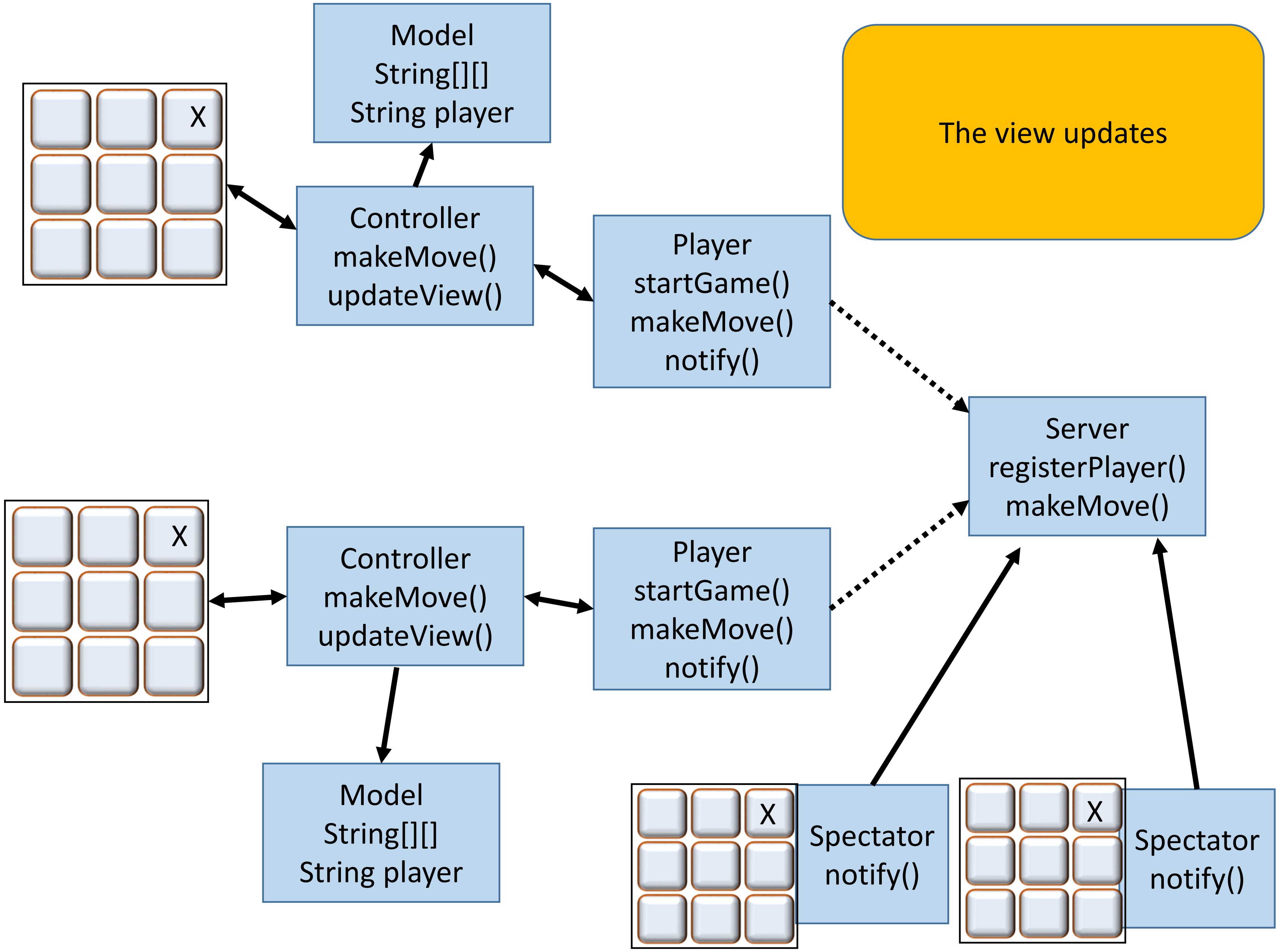
Clients call the Controllers

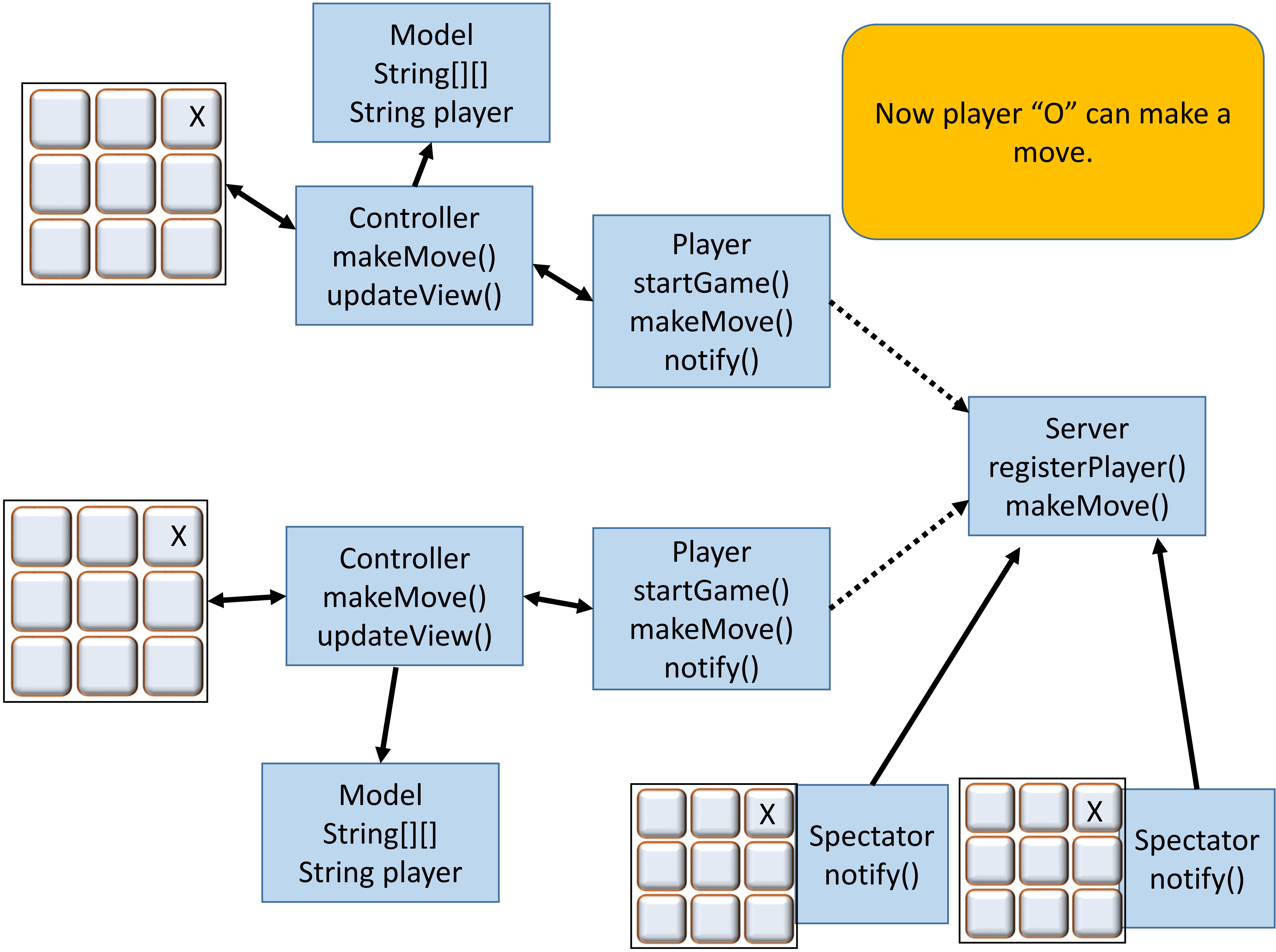




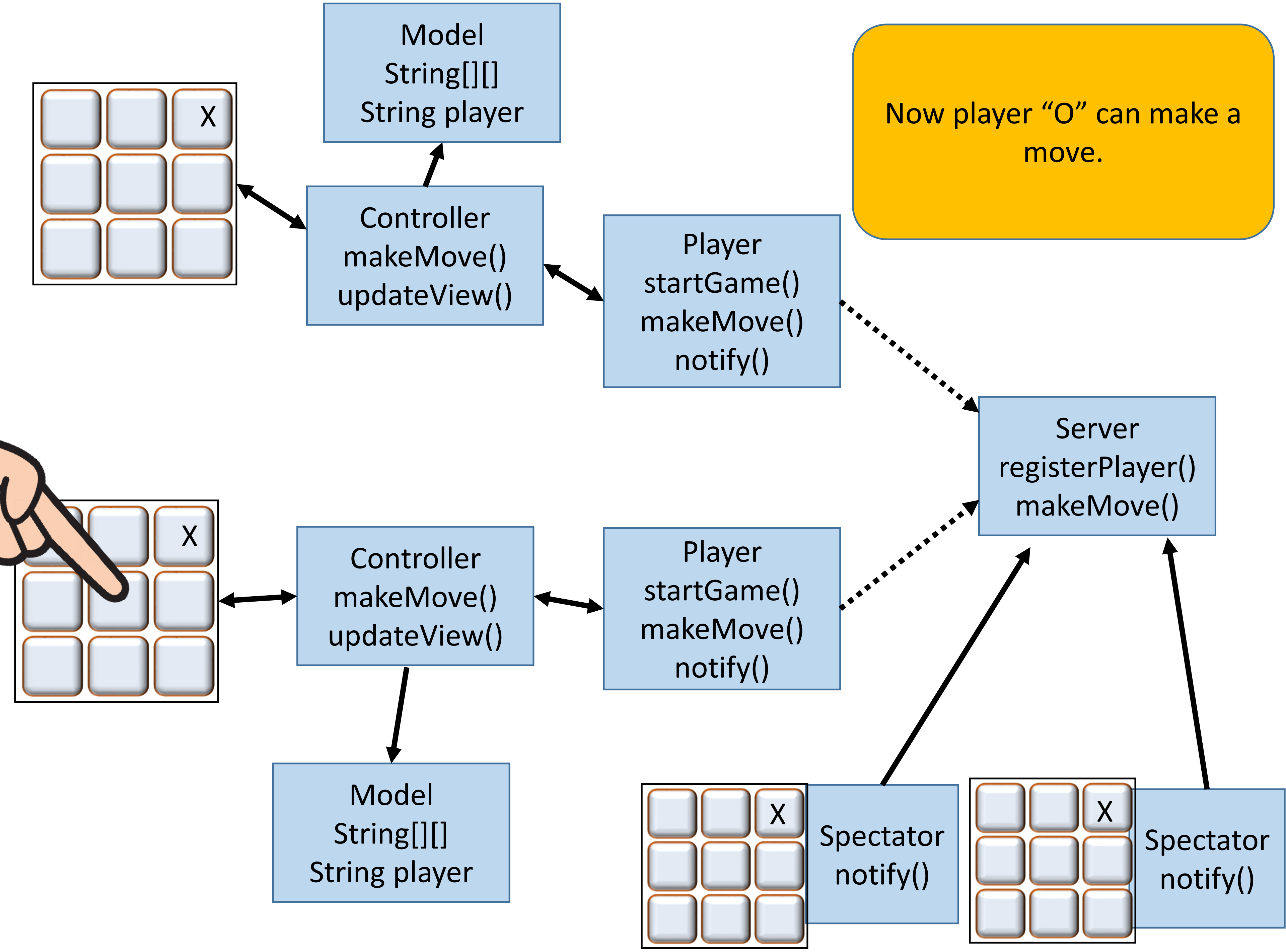
The Controllers put the new information into the Model, and forwards it to the View as well

The view updates

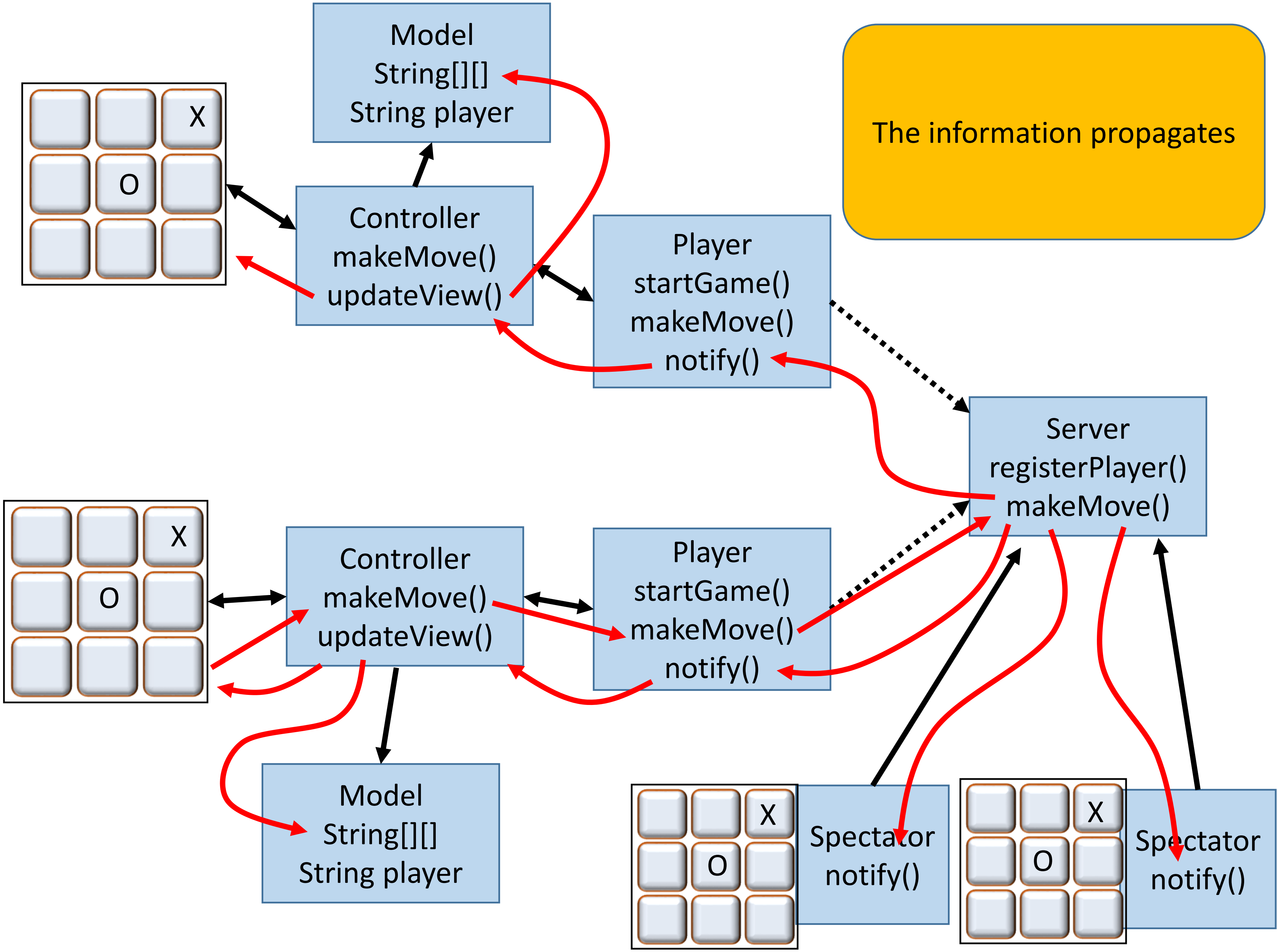




Now player “O” can make a move.



Now player “O” can make a move.



The information propagates

Deadline

- Friday the 30th of November
- Must be handed in and accepted to get access to the exam
- At the exam you will show and explain parts of your exercises