

# Distributed Tetris

Programming workshop on Android applications for Google Phones  
Google EMEA's AndroidEDU

Authors:

Sergio Arcos Sebastián  
Jordi Castells Sala  
Josep Cugat Prieto

Project developed in the  
fall semester, course 2010-2011  
at the free-choice course

Programming workshop on Android applications for Google Phones (TPAAGP)  
offered by the  
Universitat Politècnica de Catalunya (UPC)  
registered in the  
Google EMEA's AndroidEDU Programme

Source code:

<https://github.com/m13/DisTetris>

## TABLE OF CONTENTS

1	SUMMARY.....	5
2	OVERVIEW.....	7
3	INTERFACE.....	8
3.1	NEW GAME.....	8
3.2	JOIN GAME.....	9
3.3	SINGLE (GAME).....	10
3.4	STATISTICS.....	10
3.5	CONFIGURE.....	11
4	ARCHITECTURE.....	12
4.1	NET.....	12
4.2	GAME.....	14
4.3	DATABASE.....	16
5	IMPLEMENTATION.....	17
5.1	GAME.....	18
6	CONCLUSIONS.....	20



## 1 Summary

Tetris is probably the most popular puzzle video game ever. It was originally designed and programmed in 1984 by Alexey Pajitnov in Moscow. In the standard version of the game, a random sequence of tetrominoes shapes (see Figure 1) fall down the playing field. The objective of the game is to manipulate these tetrominoes, by moving each one sideways and rotating it by 90 degree units, with the aim of creating a horizontal line of blocks without gaps. When such a line is created, it disappears, and any block above the deleted line will fall until it touches another block at the bottom of the playfield. This may provoke a chain-reaction effect and thus the disappearance of more lines. With every ten lines that are cleared, the game enters a new level. As the game progresses, each level causes the tetrominoes to fall faster. For each disappeared line, the player gets a score. The higher the playing level, the higher the score obtained per line. The game ends when the stack of tetrominoes reaches the top of the playing field and no new tetrominoes are able to enter. In certain versions, the game can also end if the player is able to get all the way to level 15.

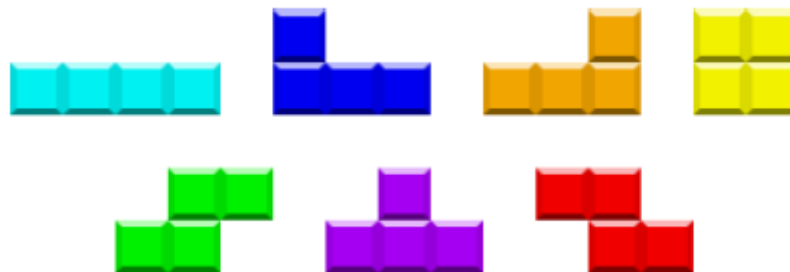


Figure 1: The seven tetrominoes used in tetris (top row, left to right: I, J, L, O; bottom row: S, T, Z).

We want to play tetris with our Google-phoner friends.

### Problems to be solved:

- Design a protocol that allows to play distributed tetris with Google phones that communicate among them via wireless. In such a version there is not only one user playing against the machine, but a set of players. All the players play the same game, and so they share the same board. Moreover, each player has full information about the game. The player play in turns. The turns are equally split among the players in a round-robin way. For example a turn could be just the handling of a tetromino or the handling of 10 tetrominoes. The points to be scored for each disappeared line go to the player who put the last piece in it.
- Design a protocol for starting the game and decide who are the players and establish the order among them.
- Extend the above protocol to include the possibility of playing in teams. In a team, only one of the players is allowed to use the team turn and move a tetromino. Thus, a team of players scores as such, not individually. Design a protocol for deciding which player in a team plays in every turn.
- Keep a data base of the players with their scores. Provide the possibility of showing statistics on the smartphone.

**Extensions:**

- Keep a data base of the players with their scores. Provide the possibility of showing statistics as a web service.
- Most Tetris games allow the player to press a button to make a piece descend to the bottom-most free position, rather than waiting for it to fall. Implement this option and modify the scoring in order to award also the height that the piece fell.

## 2 Overview

The original summary said that we had to develop the network with the Bluetooth protocol, but Bluetooth has some serious flaws for the purpose of this project. It cannot establish more than one connection at a time, and has a really high latency for a network game, so it would affect the usability of the game. We have implemented the game over Wireless, letting the player play versus other players in the same LAN, faster and more stable.

With this change, the application solves all the main problems, but only solves the second extension. We decided to not spend time with an external web-service because the focus of the project was the Android platform.

Figure 2 represents what is physically needed:



Figure 2. Physical global vision

We need at least a wireless access point and two phones. Although one phone always creates the server, it can play too. Also, we can have a large number of clients. They will be assigned as players and be split into different teams.

When the application is installed in the system, it can be launched from the android drawer searching for the corresponding icon.



Figure 3. DisTetris icon

The application needs the permissions of Internet and Wifi, and a phone with a platform version equal or higher than 1.5.

### 3 Interface

In the main view, there are five buttons in a row. It is a modified horizontal gallery. Each section uses a representative name to help understand its purpose, because the application does not have any kind of help. By default the selected option is “Join Game.”

Now, we are going to introduce all of them, showing a figure of his inner views and a brief description of all the options.

#### 3.1 New game

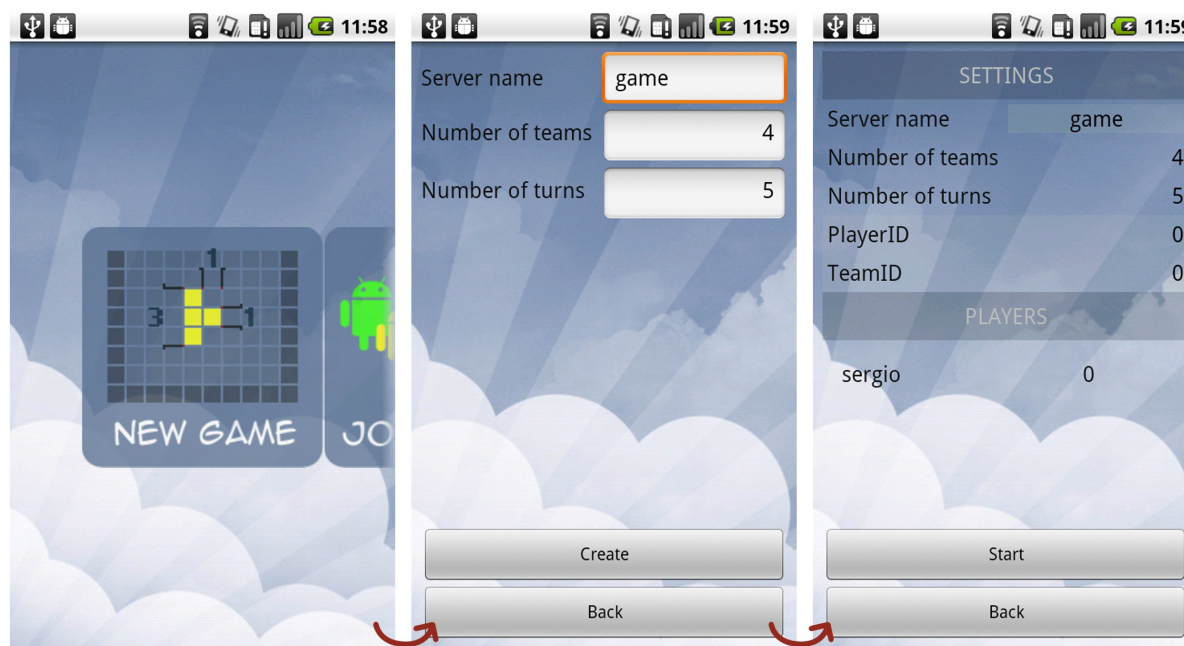


Figure 4. Views of New game

This section allows the user to create a new server.

There are three views. The first one is the main icon. The second one is where the user defines the name of the server (which will be shown to players searching for running servers), the number of teams and the number of turns each one plays. The teams are assigned with a round-robin. Finally the last screen shows the settings of the server and the players that have joined the server, refreshing in any update.

Minimum two players are required to start the game.



### 3.2 Join game

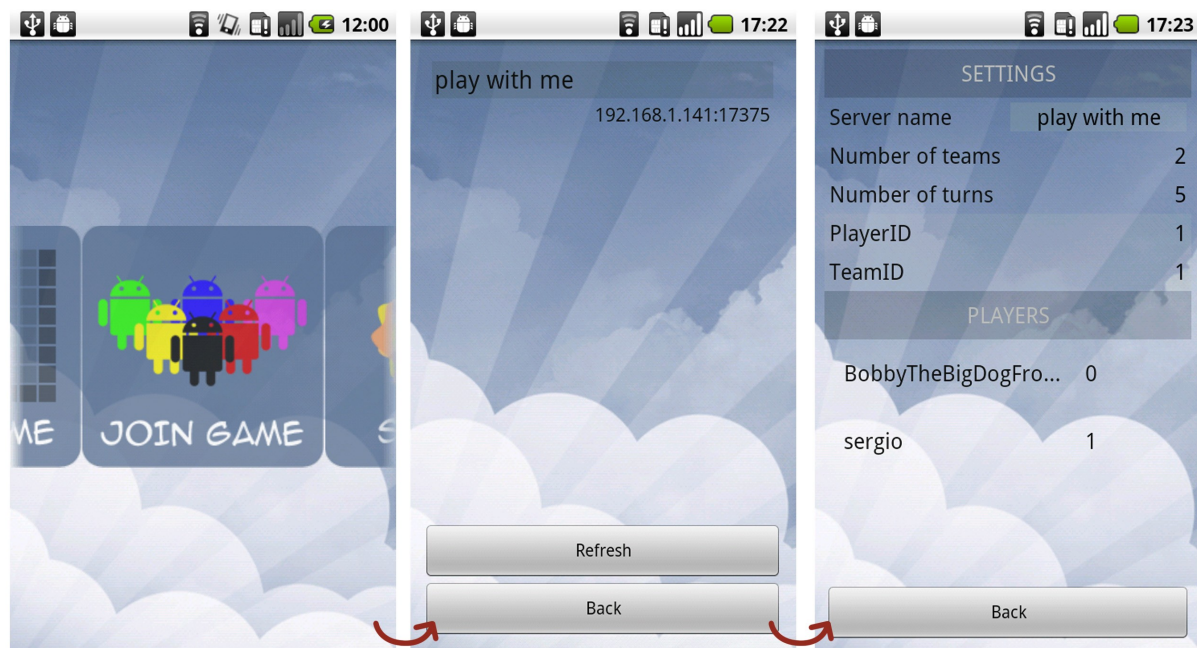


Figure 5. Views of Join game

This section shows the servers in the LAN so you can join them.

There are three views. The first one is the main icon. The second one shows all the servers in the LAN that are currently running and accepting new players. When you press one of them, it enters the last view. In this view you can only wait until the server starts the game, but you will be able to see the server settings at the top and the players at the bottom.

### 3.3 Single (game)

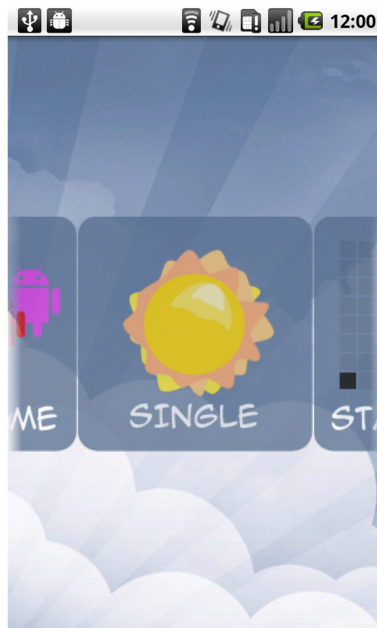


Figure 6. View of Single (game)

This section is a single screen, which allows you to play alone. The first view is the main icon.

### 3.4 Statistics



Figure 7. View of Statistics

This section shows the ranking with the saved scores.

There are two views. The first one is the main icon. The second is the table by teams and by single player. It is sorted by score in descending order.

### 3.5 Configure



Figure 8. View of Configure

This section lets configure the settings.

There are two views. The first one is the main icon. The second is a list of settings that can be modified by the user. Currently there is only one option, the username that is displayed to other players and in the scores view.

## 4 Architecture

The two big points to solve to get DisTetris working are the connections between devices and the logic of the game.

### 4.1 Net

There are two kind of protocols used in the application. First, the UDP protocol, that allows to find the devices in the connected LAN through broadcast, and second, the TCP protocol, that creates a link between the server and the client, where the data flows constantly.

Over the TCP layer, we have designed a custom protocol that allows to synchronize all the devices by commands. We can split this protocol in two phases.

#### First Phase:

In this phase, the game is stopped and the players can join it.

**Command:** WAITINGROOM

**Arguments:** Serialized Class WaitingRoom

**Who sends:** The server

**Who receives:** All the clients

**Description:** This message is sent to all the connected clients when a new client connects. It contains all the info about the game and the players.

**Command:** STARTGAME

**Arguments:** <none>

**Who sends:** The server

**Who receives:** All the clients

**Description:** Notifies the client that the game is going to start and the view must change to GameView.

#### Second Phase:

In this phase, the players are playing and they send and receive the information about the board.

**Command:** UPDATEDBOARD

**Arguments:** Serialized Class Board

**Who sends:** A client

**Who receives:** The server

**Description:** A client sends an updated board (because he is currently playing) and the server relays it to all the connected clients.

**Command:** UPDATEBOARD

**Arguments:** Serialized Class Board

**Who sends:** The server

**Who receives:** A client

**Description:** The player who receives the new board updates his view if he is not playing.

**Command:** UPDATEMYTURN

**Arguments:** Number of turns

**Who sends:** The server

**Who receives:** A client

**Description:** The server assigns a number of playable turns to the user who has to play.

**Command:** TURNFINISHED

**Arguments:** Serialized Class Board

**Who sends:** A client

**Who receives:** The server

**Description:** The client who was playing send this signal when he has finished his turns.

**Other signals:**

Any kind of commands that can be sent between the devices.

**Command:** SHUTDOWN

**Arguments:** <none>

**Who sends:** The server

**Who receives:** All the clients

**Description:** The servers detects an error with some connection and sends this command to all the clients to report it. The game is over but the score is not saved.

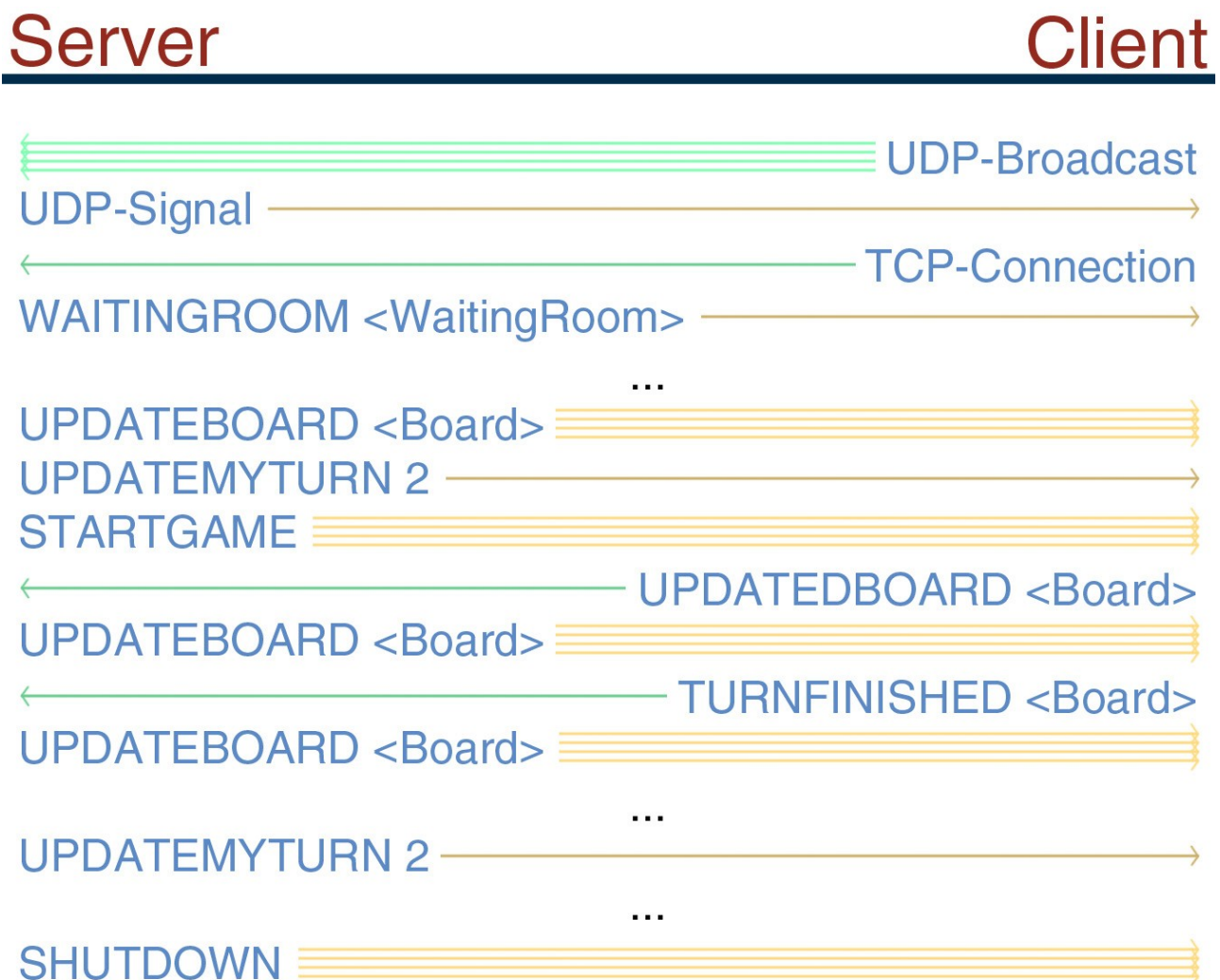


Figure 9. Communications brief



## 4.2 Game



Figure 10. Game View in a multi player

This part is composed by some different elements.

### Game and GameView:

Game is the main class when you are playing in single or multi player. Game contains the interaction with the user, all the game Logic is in GameController. When the user swipes the screen or push the buttons this is the part that manages it.

GameView draws the whole status of the game.

### The view:

The view is a representation of a concrete status of the game. The board, the current piece, the next piece and the scores have to be drawn.

For doing it in a efficient way, this uses a custom Android View with his own onDraw() function. This function is executed constantly in the same way as the gameLoop(), using a timer that invalidates constantly the view.

The view is independent from the game logic, it only uses the game structures to visualize a current point in the game.

### The game:

It is defined by the function gameLoop(). This function defines what is a step in the game, as it is showed in the next diagram:

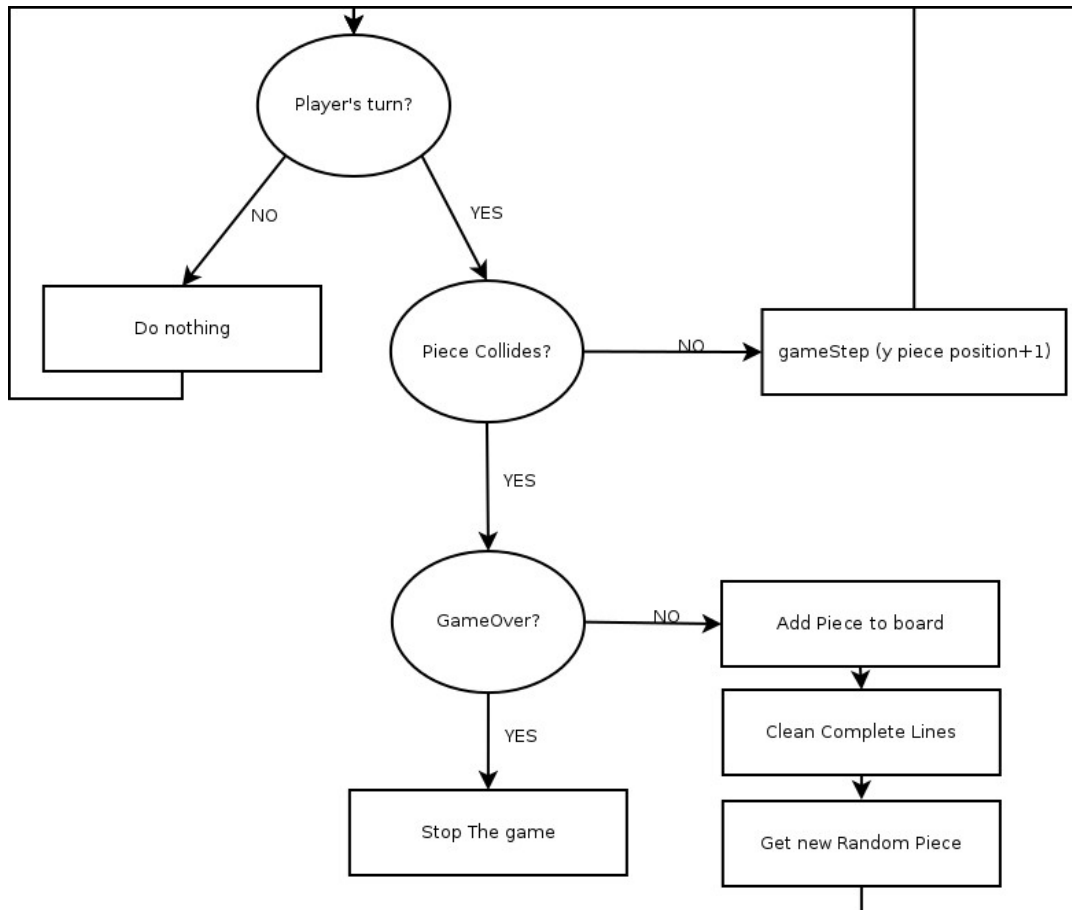


Figure 10. Diagram of the game

This is the logic of the game. The squares means actions and the circles means decisions. This is implemented with a timer with a associated task (GameLoop) executing it until the game is over.

### The movement:

The next table shows how you can interact with the piece when it is your turn.

	GestureDetector	Buttons
Move to the left	Swipe to the left	Left Button
Move to the right	Swipe to the right	Right Button
Fall block	Swipe to the bottom	-
Move to the bottom faster	-	Down Button
Rotate	Tap the screen	Up Button

Table 1. How to interact with the game

### The engine:

GameActivity is supplied with some of the functions of the class Board. It access through the domain controller to modify or check the collisions.

### The score:

The score of a successful line (you can delete between one and four lines) is calculated as:

point := (number of lines cleared) \* multiplier

multiplier := 1, normal fall  
or  
((number of line where the piece lands)-(number of line from you do a fall))/2

## 4.3 Database

The database contains three tables. Their schemes are the next:

config	
key (TEXT)	value (TEXT)
playername	Viciado
servername	Wifi
numteams	3
numturns	5

Table 2. config table

This table is like a hash map that saves the user config values. It is used in the configure screen and the new game screen.

individual		
name (TEXT)	score (INT)	date (INT)

Table 3. individual table

This table records all the scores with the name and the date, done by a single player game.

team		
name (TEXT)	score (INT)	date (INT)

Table 4. team table

This table records all the scores with the name and the date, done by a team player game.

The scores are saved when a game is finished with a Game Over message.



## 5 Implementation

The source code is implemented with a three layer design, naming conventions for the layers are: presentation, domain and storage.

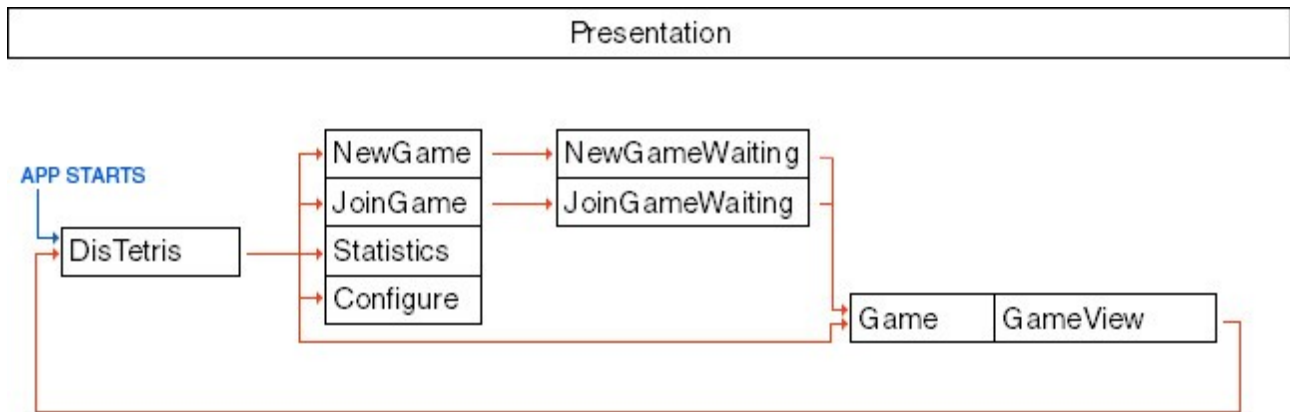


Figure 11. Presentation package

This package is started in the DisTetris class. All classes calls to the controller of the domain package.

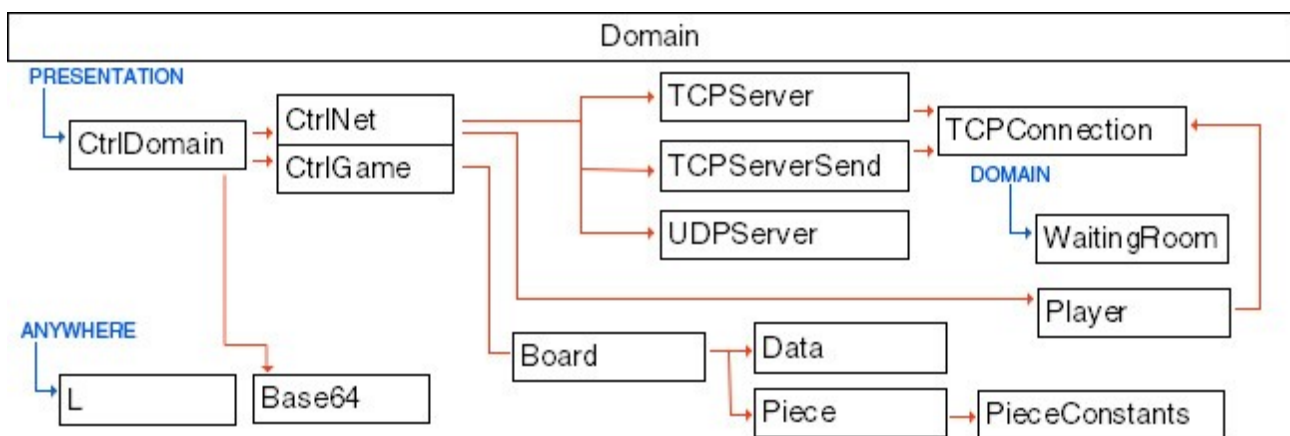


Figure 12. Domain package

Only the controller class CtrlGame calls to the package storage.

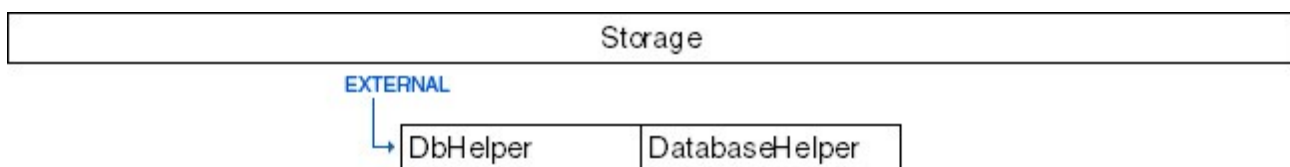
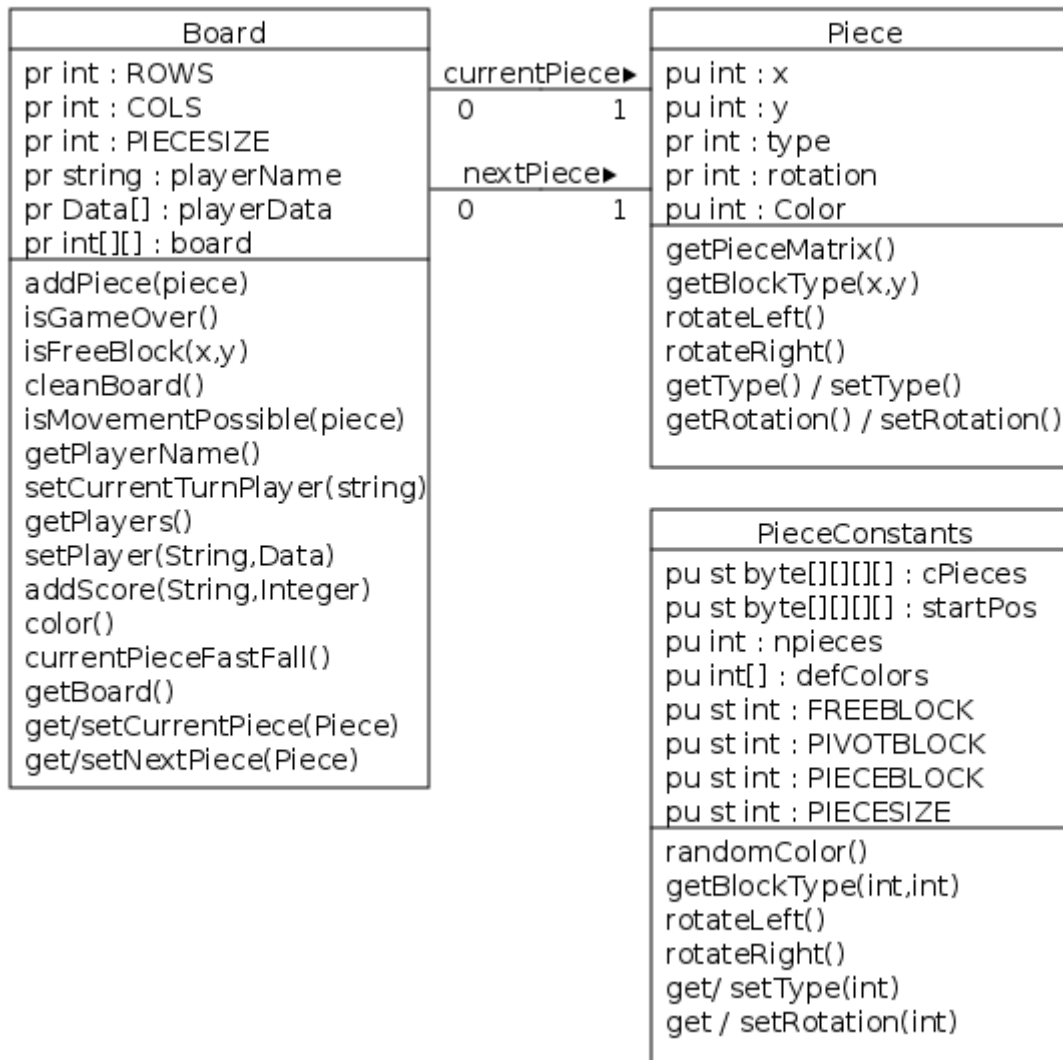


Figure 13. Storage package

## 5.1 Game

Due to the complexity of the game screen, we have attached a figure with all the basic structures and explained all his characteristics:



### BOARD:

The board is basically a colour matrix. If in a pixel there is a colour set, means that there is a block assigned. Otherwise if we find the value 'PieceConstants.FREEBLOCK' means that this is an empty block.

### PIECECONSTANTS:



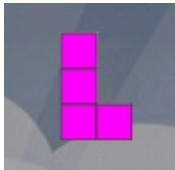
This class contains all the constants needed to do easier the painting and reference to Piece. This will be explained more clearly in the Piece class documentation.

### PIECE:

The pieces are matrix type of 5x5 set in 'PieceConstants.cPiece'. cPiece is a matrix type of four dimensions that allows to retrieve in a fast way the blocks assigned by a Piece with:

*cPiece[type][rotation][horizontal\_blocks][vertical\_blocks]*

Then, the type of a Piece is defined using only a number between 0 and 'PieceConstants.npieces'. The same happens with the rotation. The next Table shows an example:

Piece Definition	Matrix	Screenshot
Type 1 Rotation 0 Color "green" cPiece[1][0] → Matrix	{0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}, {0, 1, 2, 1, 1}, {0, 0, 0, 0, 0}, {0, 0, 0, 0, 0}	
Type 1 Rotation 1 Color "green" cPiece[1][0] → Matrix	{0, 0, 0, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 2, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 1, 0, 0}	
Type 2 Rotation 0 Color "pink" cPiece[2][0] → Matrix	{0, 0, 0, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 2, 0, 0}, {0, 0, 1, 1, 0}, {0, 0, 0, 0, 0}	

## 6 Conclusions

We split the work in three branches. The Standard view and menus definition, the UDP/TCP protocol to send the boards and all the Game logic and view. So the conclusions and comments are also somewhat split.

The design can always be better. There are a lot of custom designs and anyone teach you new tricks. For example, the background is the simplest element but it changes a lot the feeling of the application. Other elements of the application, as the database or the scores, had a complication by his relation with the game and the net.

The network structure was challenging because we haven't seen in the university how to design high-level network protocols. It was very hard to debug too because it involved multiple phones transmitting information with a lot of threads at the same time. We also found a bug in the Android implementation of the Java Sockets that we had to workaround for it to work correctly. You can find the Android issue here:

<http://code.google.com/p/android/issues/detail?id=7933>

The part related to the Game Logic and the views was not so difficult. When you read the documentation about how to use Canvas it's only a matter of thinking the design and spend some time in the implementation. We have not used the Android OpenGL library because we think it isn't necessary. Talking about the Game Logic, Tetris is a game with a lot of history (since 1984), so there's nothing new to implement, you can read a lot of examples on how to make it work, I'm sure that our implementation is not the best but one that works.