**Софийски университет „Св. Климент Охридски“**



Факултет по математика и информатика

“ИС”, Курс: 3

Дисциплина: “Мобилни приложения”

**Документация по проект „Tic Tac Toe“**



**Изготвил:**

Сава Станимиров Димитров (ФН:)

София

Летен семестър 2018/2019

***Съдържание***

[Увод – описание на приложението 3](#_Toc50134717)

[Какво е използвано при реализацията на проекта (IDE, език, проблеми при реализацията) 4](#_Toc50134718)

[Реализирани класове, функции и връзките между тях 5](#_Toc50134719)

[Потребителски интерфейс на играта 16](#_Toc50134720)

[“Screenshot” при победа на всеки от играчите 17](#_Toc50134721)

[“Screenshot” при игра без победител 18](#_Toc50134722)

# Увод – описание на приложението

Разработеното приложение е така наречената игра „Морски шах“ (“Tic Tac Toe”). Приложението е направено за двама играчи на едно устройство (телефон), играч срещу компютър (с трудности easy и medium), играч срещу играч чрез Bluetooth връзка.

Идеята на играта е че всеки играч има символ („Х“ или „О“) и трябва да го поставя в една от клетките на полето (матрица 3х3) на което се играе:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Този който започва първи е винаги с „Х“.

Играчите се редуват в слагането на своя символ на полето, докато някой не го сложи на 3 места едно до друго (хоризонтално, вертикално или диагонално). Т.е:

**Хоризонтално:**

|  |  |  |
| --- | --- | --- |
| Х | Х | Х |
|  |  |  |
|  |  |  |

**Вертикално:**

|  |  |  |
| --- | --- | --- |
| Х |  |  |
| Х |  |  |
| Х |  |  |

**Диагонално:**

|  |  |  |
| --- | --- | --- |
| Х |  |  |
|  | Х |  |
|  |  | Х |

(**И също по другия диагонал.)**

Ако никой не успее да направи 3 символа един до друг (т.е. да няма 3 еднакви символа хоризонтално, вертикално или диагонално), тогава играта се води “draw” или равен.

# Какво е използвано при реализацията на проекта (IDE, език, проблеми при реализацията)

Проекта е реализиран на езика Java, като използваното IDE е Android Studio. Без допълнителни настройки или програми. При създаването на проекта като минимално API ниво беше избрано: ***“API 15: Android 4.0.3 (IceCreamSandwich)”***. Това е нивото по подразбиране при създаването на нов проект, но беше оставено, тъй като показа, че с това минимално ниво приложението ще работи приблизително на 100% от устройствата, което може да ни осигури повече потребители.

**Дизайн:** За самата реализация е използван обектно ориентиран дизайн, като не е използвано наследяване, а композиция (проектът е малък и такъв, че трудно може да се измислят класове, които да се наследяват).

**Емулатор:** По време на реализацията приложението беше тествано на 2 емулатора:

1. Pixel XL API 27 – Phone (Pixel XL, 5.5”, 1440x2560, 560dpi); Api 27 (Oreo, 27, x86, Android 8.1).
2. 5.4 FWVGA API 28 – Phone (5.4” FWVGA, 5.4”, 480x854, mdpi); Api 28 (Pie, 28, x86, Android 9.0).

**Допълнителна информация:** Използван е version control (Git), за да може да се проследят промените правени в проекта през времето на реализация, но само на малка част от проекта (1/3 от него).

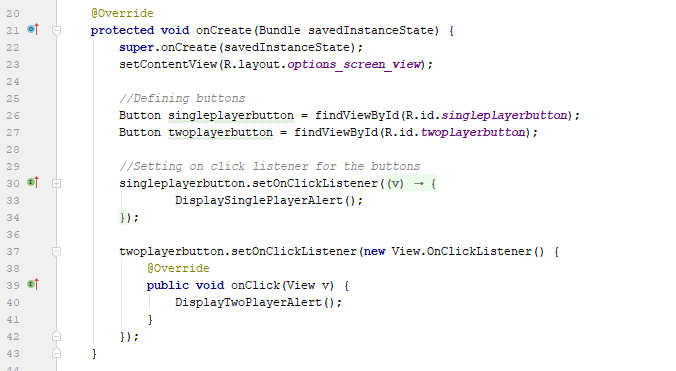
# Реализирани класове, функции и връзките между тях

Проектът е съставен от 5 класа:

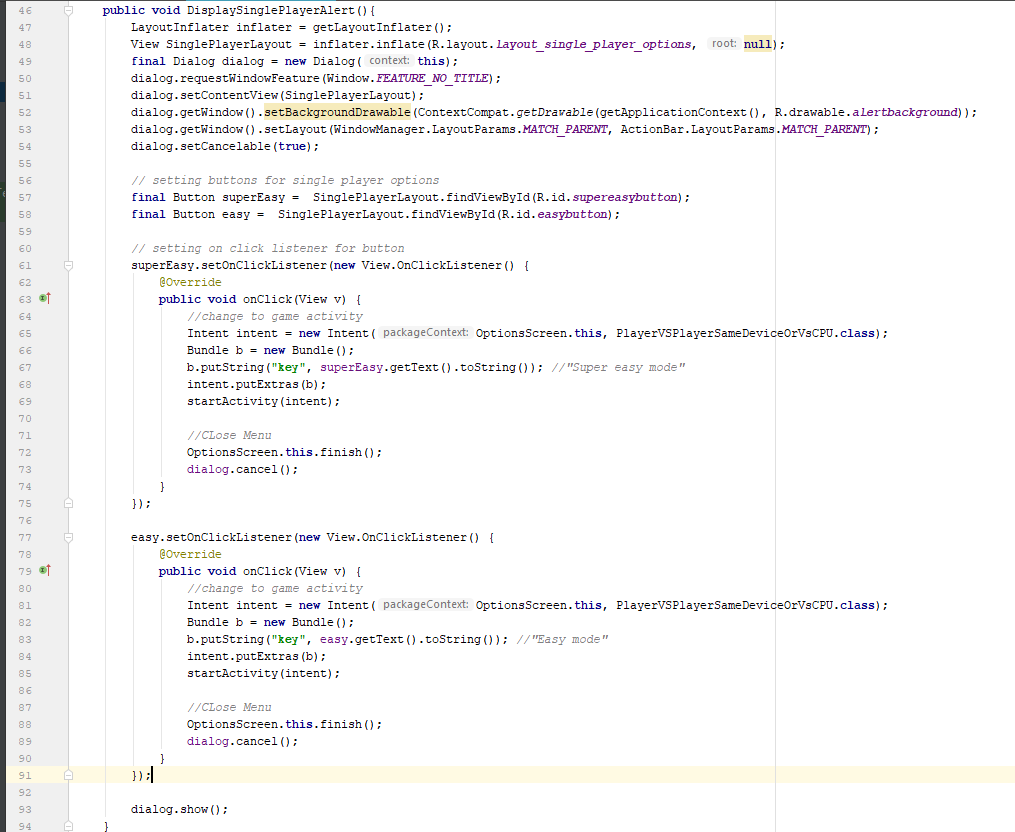
1. Основни: **OptionsScreen; PlayerVSPlayerSameDeviceOrVsCPU; BluetoothProcessHandling;**
2. Помощни: **GameBoard; Player;**

Класа **OptionsScreen** е съставен от един ***@******Override protected void onCreate(Bundle savedInstanceState)*** метод и две функции ***public void DisplaySinglePlayerAlert()*** и ***public void DisplayTwoPlayerAlert().*** Override метода създава два бутона за „single player“ и „multiplayer“ игра, като когато играча избере едното се показва нов диалогов прозорец съответстващ на една от двете функции. Ако играча избере „single player“ игра “onCreate” метода вика ***DisplaySinglePlayerAlert()*** функцията и съответно „multiplayer“ игра - ***DisplayTwoPlayerAlert()***. При „single player“ се вика класа **PlayerVSPlayerSameDeviceOrVsCPU**, като в този случай се прави игра срещу компютър. При „multiplayer“ се вика един от двата класа, ако е на едно устройство – **PlayerVSPlayerSameDeviceOrVsCPU**, като не се викат методите за игра срещу компютър, а ако е чрез „Bluetooth“, то тогава се вика **BluetoothProcessHandling** класа.

**onCreate()** метода:

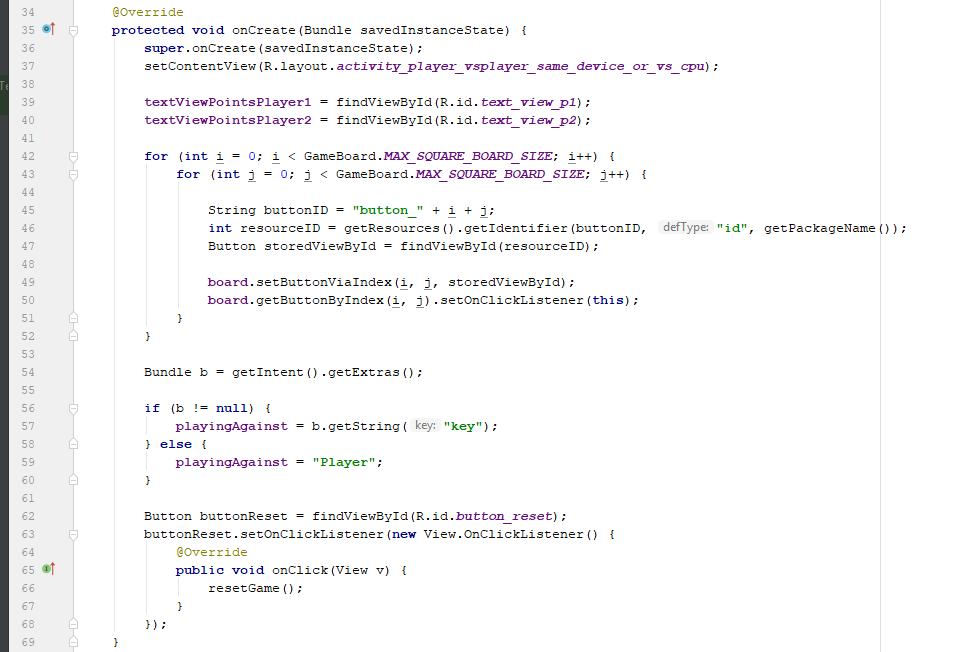


**DisplaySinglePlayerAlert()** функцията:



**DisplayTwoPlayerAlert()** функцията е аналогична на горната.

**PlayerVSPlayerSameDeviceOrVsCPU** класа се грижи за логиката при игрите „player vs player on same device“ и „player vs computer“. Логиката при първия вид игра е ясна, там трябва класа да се грижи само да поставя правилния символ на мястото на което е „кликнато“. Като първо се вика „onCreate()“ метода, който създава референции към всяко поле от „layout“-та за този тип игри, създава „reset“ бутона, за рестартиране на играта и „казва“ на приложението какъв тип игра е (срещу играч или компютър):



**onClick()** метода:



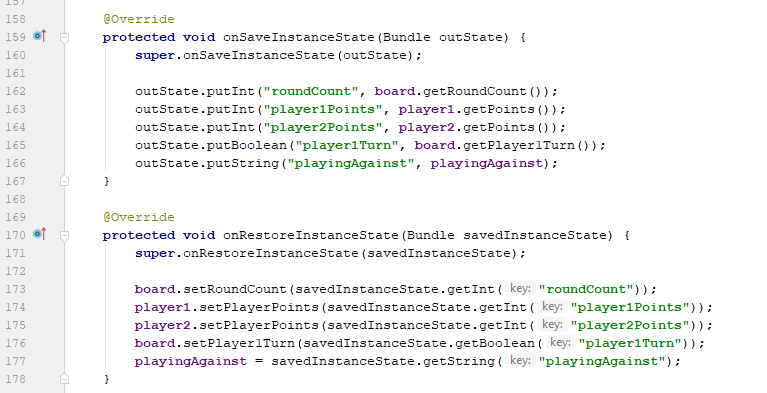
Самия метод (**onClick()**) отразява промените в матрицата и потребителския интерфейс при натискане на някой от 9-те бутона в UI-я, т.е. при натискане на една клетка слага „Х“ или „О“ в съответната клетка. Също така когато минат 5 кръга (минималния брой, за да е възможно да има победител) започва да проверява дали при натискането на бутон има победител.

Класът използва композиция, за да има в себе си променливите “board”, „player1“ и „player2“, т.е. той е свързан с класовете “GameBoard” и „Player“.

Има 7 променливи:

* **private GameBoard board = new GameBoard()** – създава матрица за изгралното поле
* **private Player player1 = new Player()** – създава играч 1 и му съхранява и променя точките
* **private Player player2 = new Player()** – създава играч 2 и му съхранява и променя точките
* **private TextView textViewPointsPlayer1** – изобразява на UI-я промяната в точките на играч 1, ако е победил
* **private TextView textViewPointsPlayer2** - изобразява на UI-я промяната в точките на играч 2, ако е победил
* **private String playingAgainst = “”** – показва какъв е “game mode”-а на играта (срещу компютър или друг играч на същото устройство)
* **private ArrayList<Pair<Integer, Integer>> AIIndexesMovedOver = new ArrayList<>()** – пази кои индекси е обходил компютъра при игра срещу компютър

Този клас също така се грижи да запазва систоянието на играта при завъртане на екрана. Това става чрез (@Override) методите **onSaveInstanceState(Bundle outState)** и **onRestoreInstanceState(Bundle savedInstanceState)** :

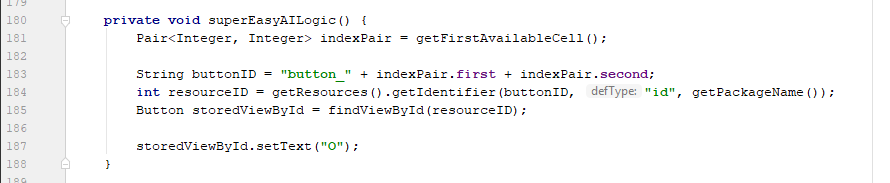


Кратко описание на останалите функции в класа:

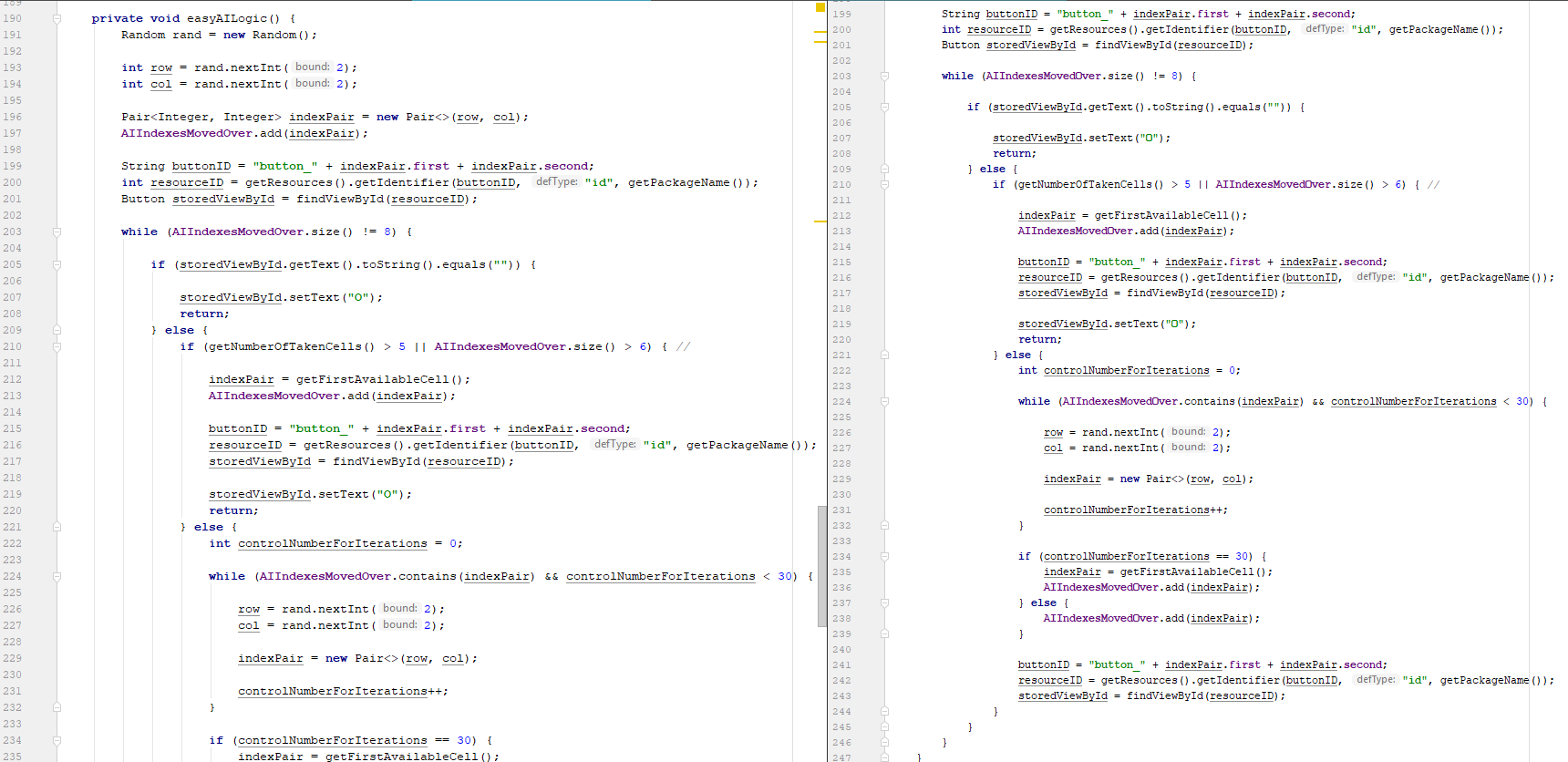
* **private void announceWinnerOrChangeTurn();** - ако има победител вика playerWon() функцията, която извежда името на победителя, ако няма е следващия играч на ход.
* **private void playerWon(int winningPlayerNumber);** - увеличава точките на победителя, извежда като текст на екрана, кой е той, като в същото време променя текства на точките на потребителския интерфейс и занулява матрицата.
* **private void draw();** - извежда съобщение, че няма победител и занулява матрицата.
* **private void resetGame();** - прави точките на играчите = 0, променя текства на точките на потребителския интерфейс и занулява матрицата
* **private void updatePointsText();** - променя текства на точките на потребителския интерфейс

За „AI“ логиката има два вида трудности „easy“ и „medium“, като използват две помощни функции:

**„easy“**

****

**„medium“**



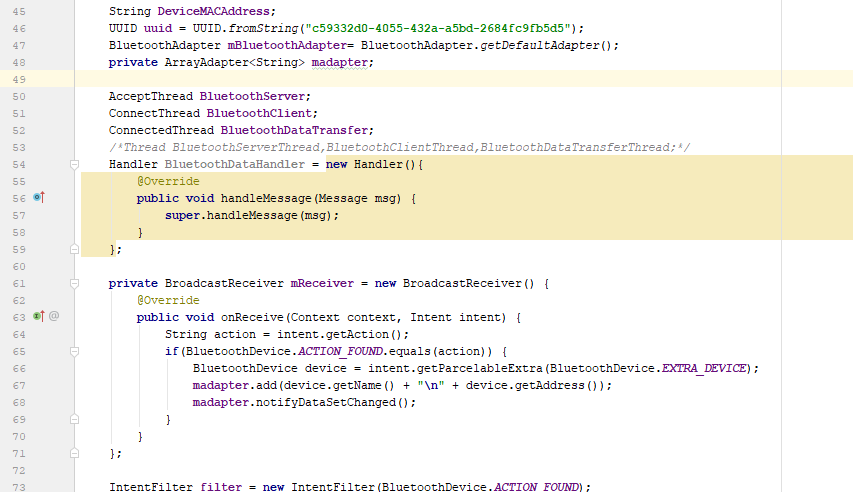
**Помощни функции**

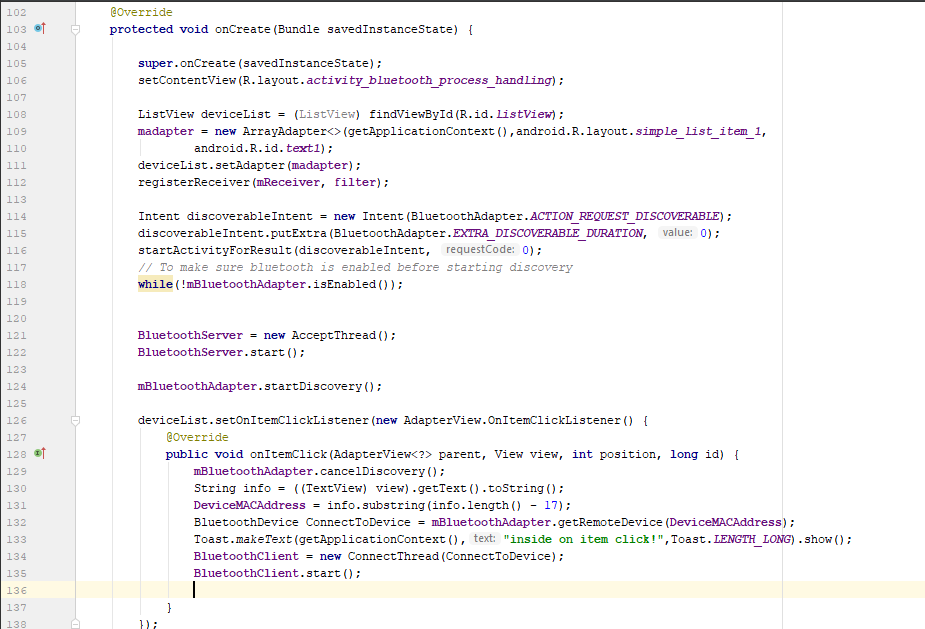
При „easy“ алгоритъма е доста прост, слага „О“ на първата свободна клетка.

При „medium“ алгоритъма също не е сложен, но е по трудно играча да предположи къде компютъра ще сложи „О“, тъй като в началото се генерират произволни индекси (i и j), на които компютъра слага „О“, а когато по-голямата част от полето е вече запълнена или обходените клетка станат половината от всички, компютъра слага на първата произволна клетка. Това е направено така, тъй като при тестването се забеляза, че има случаи в които при 5 запълнени клетки на приложението му отнена твърде дълго време да намери празна клетка на случаен принцип, което забавя приложението и то „crash“-ва, а това го предотвратихме, като зададохме на алгоритъма да търси на случаен принцип 30 пъти и ако до 30-я път не е намерило свободна клетка, то тогава да вземе първата свободна. Като броя пъти може да се промени и на повече, но достатъчно малко, за да не крашне приложението.

**BluetoothProcessHandling** класа се грижи за осъществяването на „Bluetooth“ връзка, за поддържането на тази връзка, както и за логиката при играене.

Следният код ще покаже променливите и „broadcast“ за управлението на Bluetooth свързването:

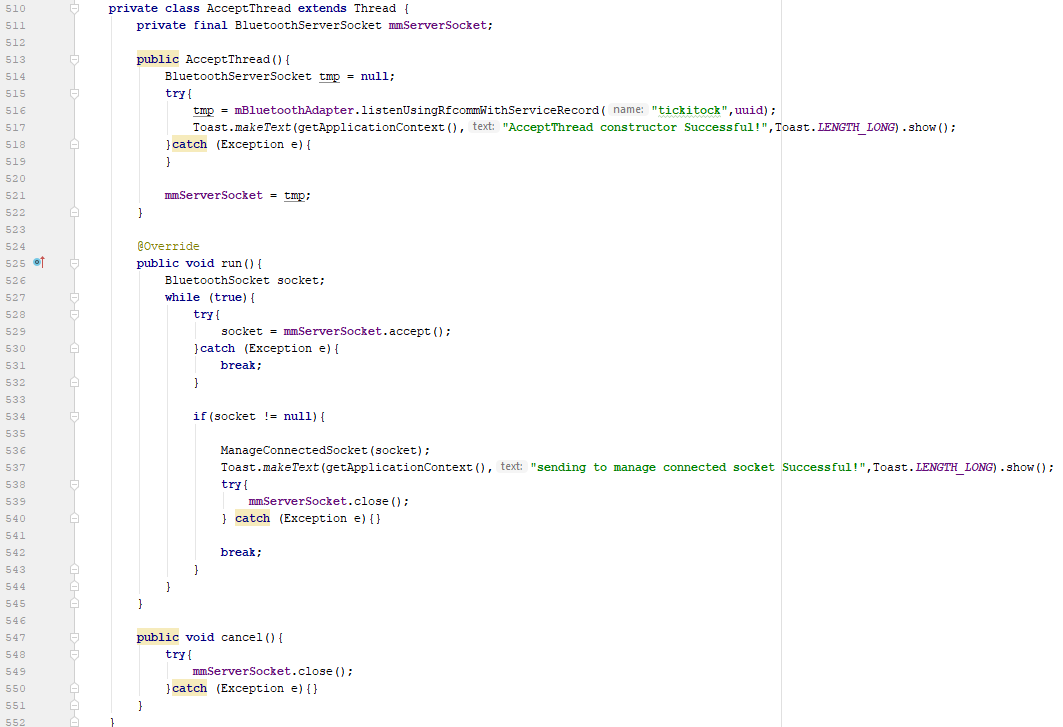


**onCreate()** метода осъществява Bluetooth връзка с устройство, което играча е избрал, като първо проверява дали Bluetooth на играча е пуснат: 

И също така запълва променливите за играта.

За осъществяването на „Bluetooth connection“ са използвани няколко допълнителни класа и функция:

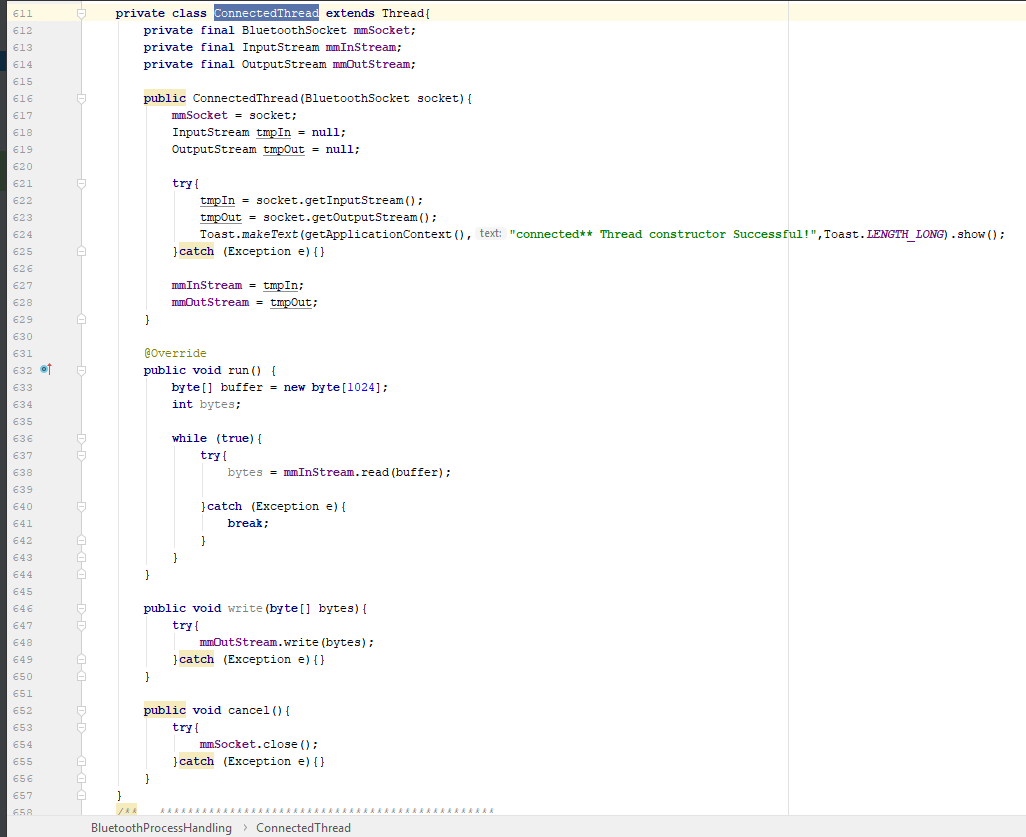
**AcceptThread:**

****

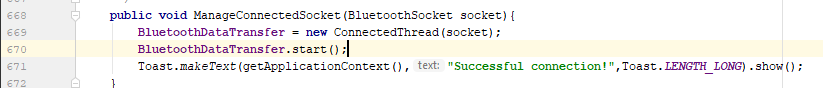
**ConnectThread:**

****

**ConnectedThread:**

****

**ManageConnectedSocket (function):**

****

***Описание на помощните класове:***

**GameBoard** класа има за цел да представи игралното поле (матрицата в която играча слага символите си, като тя е запълнена с референции към бутоните на потребителския интерфейс), да пази кой играч е на ход, колко кръга са изминали и да проверява за победител.

Променливите в класа:

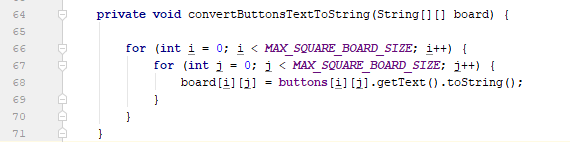
* **static final int MAX\_SQUARE\_BOARD\_SIZE = 3;** - константа представяща максималния брой колони и редове в матрицата.
* **private Button[][] buttons = new Button[MAX\_SQUARE\_BOARD\_SIZE][MAX\_SQUARE\_BOARD\_SIZE];** - матрица от бутони, които представляват референции към бутоните в потребителския интерфейс
* **private boolean player1Turn = true;** - представя дали играч 1 или играч 2 ще слага символа си в матрицата
* **private int roundCount = 0;** - отброява колко кръга са минали.

Всяка от променливите има селектор (getter) и мутатор (setter), като за матрицата селектора и мутатора са по индекс.

Функциите в класа:

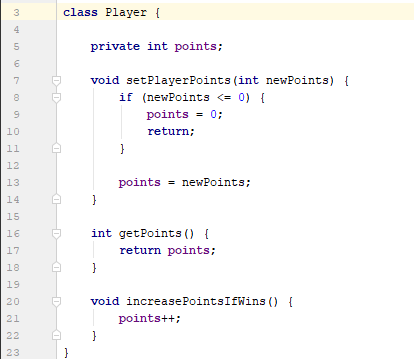
* **boolean checkForWin();** - функцията използва всички функции надолу, без resetBoard(). Функцията е обобщаваща, като ни казва дали имаме или нямаме победител.
* **private void convertButtonsTextToString(String[][] board);** - функцията приема двумерен масив от string-ве, който масив пълни с текста („Х“ или „О“) от бутоните в потребителския интерфейс, за да можем да направим проверката за победител.
* **private boolean checkRowForWin(String[][] board);** - проверява дали на някой от редовете има 3 еднакви символа.
* **private boolean checkColumnForWin(String[][] board);** - проверява дали на някоя от колоните има 3 еднакви символа.
* **private boolean checkMainDiagonalForWin(String[][] board);** - проверява дали има 3 еднакви символа на главния диагонал.
* **private boolean checkSecondaryDiagonalForWin(String[][] board);** - проверява дали има 3 еднакви символа на вторичния диагонал.
* **void resetBoard();** - занулява играта, прави текства на бутоните празен стринг, занулява рундовете и променя хода да е на играч 1

Ще покажем само функцията **convertButtonsTextToString**, тъй като е възможно да не се разбере точно какво прави в конкретния случай:



Идеята на функцията е да вземе текста от всяка клетка на игралното поле и да го преобразува в String и след което да можем да проверим за победител.

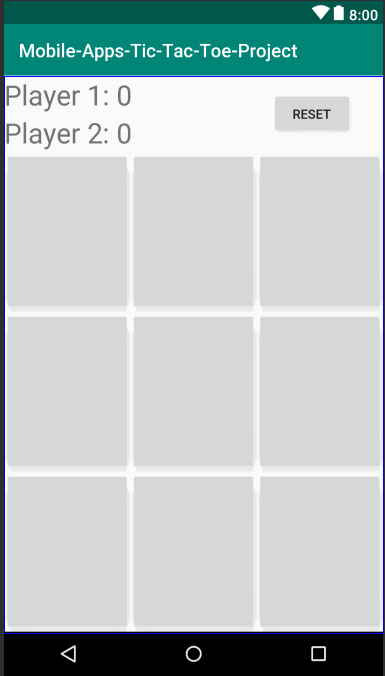
**Player** класа е най-простия, той има само една променлива **private int points**, която представя точките на играча. В него има и три функции **void setPlayerPoints(int newPoints) (**setter); **int getPoints() (**getter**);** **void increasePointsIfWins()** (увеличава точките на играча с 1)**;**



***Забележка: При всички класове там където няма модификатор за достъп се използва този по подразбиране, т.е. package-private.***

# Потребителски интерфейс на играта

**Интерфейс на модове „player vs player same device“ и „player vs computer“:**

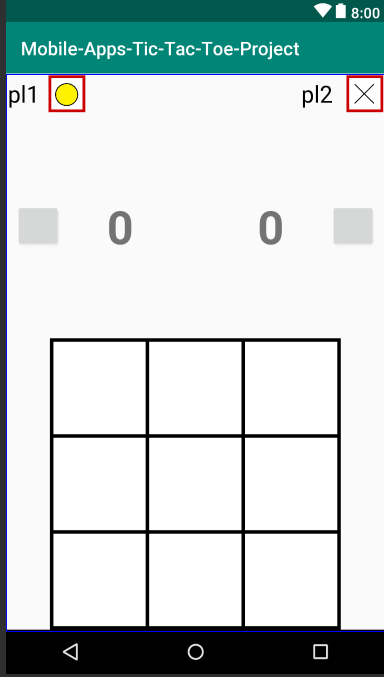
****

**Интерфейс на „player vs player via Bluetooth“:**

***Интерфейс при избиране на Интерфейс на игралното***

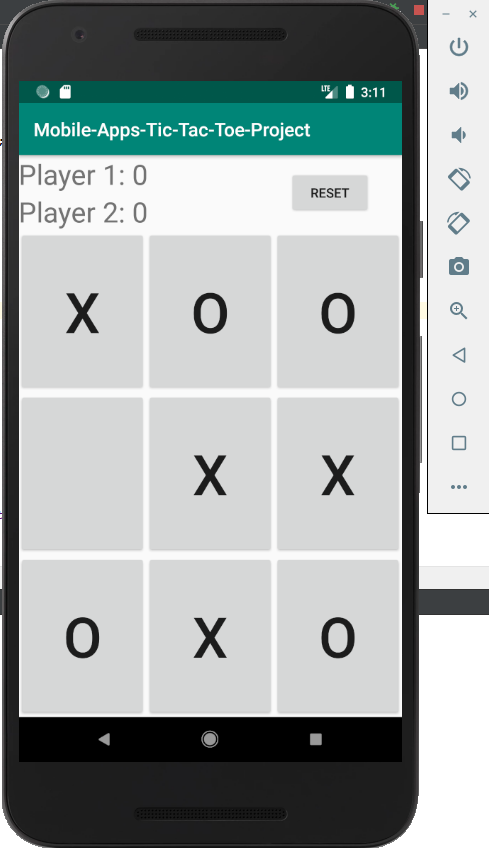
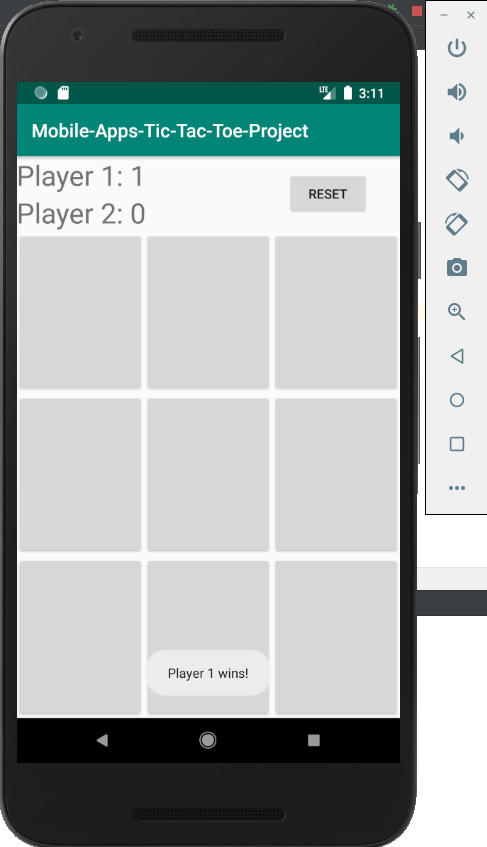
***устройство към което поле при осъществена връзка:***

***да се свърже:***

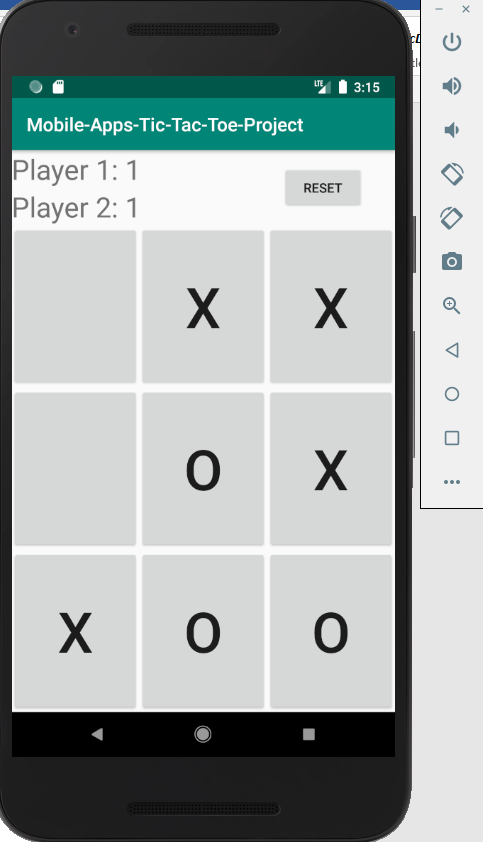
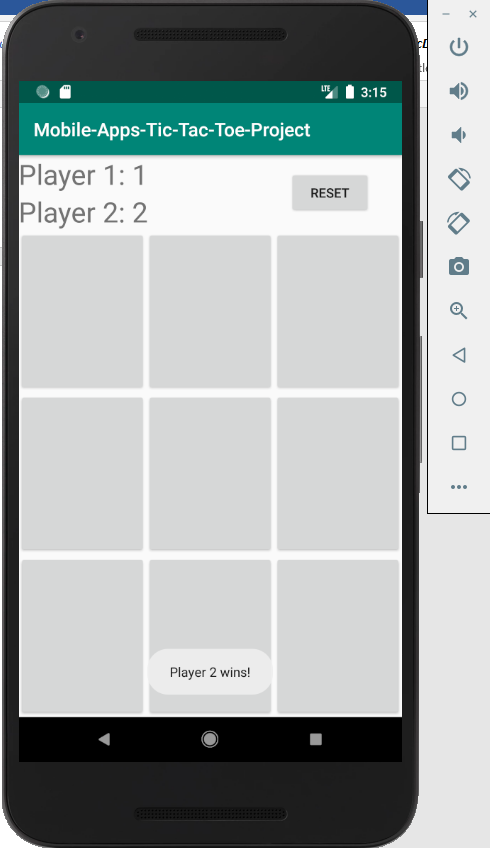
** **

# “Screenshot” при победа на всеки от играчите

**При победа на „Х“:**

**При победа на „О“:**

# “Screenshot” при игра без победител

**Игра при, която имаме „draw!“:**

