CSC310 Software Engineering
Presented to: Prof. Lin Jensen

Group Project Report:

Line 'Em Up

Prepared by:
Celine Bensoussan
Mario Marchal-Goulet
Jill Perreault

April 22, 2013

Contents

Overview

Our **Line ‘Em Up** application is a digital version of the classic board game released under the *Connect Four* trademark by Milton Bradley in 1974. Our version runs on Google’s Android operating system for mobile touchscreen devices such as smartphones and tablets.

Connect Four is a two-player game in which the players take turns dropping coloured discs into a seven-column, six-row vertically-suspended grid. The pieces fall straight down, occupying the next available space within the column. The object of the game is to connect four of one's own discs of the same color next to each other vertically, horizontally, or diagonally before the opponent (Wikipedia).

Our version of the game stays true to the original in that it is setup to be played by 2 human players who will take turns using a single platform (an Android smartphone or tablet) to play a head to head game of connect four by physically passing the device back and forth between themselves.

Line 'Em Up
User Documentation

2.1 Android Device Requirements:

- The minimum Android platform version required to install and run the application is Android 3.0 (Honeycomb).
- According to the Android developer website this accounts for approximately 46% of Android devices in circulation as of March 4th 2013. (http://developer.android.com/about/dashboards/index.html)

2.2 Application Installation

- The application is not currently available for download on any Android app marketplace (e.g. Google Play), nor from any independent website because it was not considered to have the appeal to justify making it available to a mass audience. Such an option would be relatively simple to implement (however the Google Play app marketplace does charge a one-time fee to set up a developer account) and would render our app globally accessible.
- For the time being, a request for the app can be sent to the developers by email (Note: please see the developer contact information section at the end of the documentation).

Installation Requirements:

1. A Gmail account.
2. The official Gmail application.
3. Installation from unknown sources must be enabled.

Installation Procedure:

1. Send a request for the application to the developers by email. You will receive an email with the application (apk file) as attachment.
2. On the mobile device, open the email with the Gmail application.
3. Click on the install button (Gmail automatically recognizes apk attachments).

2.3 How to Play:

- Although this may seem obvious, touching the “Let’s Play!” button from the main menu that is displayed when the application is started allows the user to start playing the game.
- An “About” menu is accessible from the main menu which displays information relevant to the app.
- The user can always touch the “How to play” button to access the menu which displays instructions on how to play while in-game or from the main menu.
- The “How to play” section displays the following information:
  - Player’s take turns dropping colored discs into a seven-column, six-row grid.
  - When the player touches somewhere on a column which still has space available, the disc is dropped into the specified column.
  - The disc falls straight down, occupying the next available space within the column.
  - The objective of the game is to connect four of one's own discs of the same color next to each other vertically, horizontally, or diagonally before the opponent.
- The “Reset” button will start a new game and also reset the player scores to 0.
- The “New Game” button will only start a new game and not reset the scores.
- Pressing the “Back” button while in game will return to the main menu and the game in progress along with the current scores will be lost.

2.4 List of Features (design/non-technical):

- Aesthetically pleasing design (e.g. a lot of effort went into the design of the graphical elements).
- Score keeping of rounds won by each player.
- In-game accessible help menu.
- Manual reset of the grid following a win to allow players to see what happened…
- The app can be interrupted by other Android processes (e.g. an incoming call) and will return to its current state when awakened.

Line 'Em Up Programmer Documentation

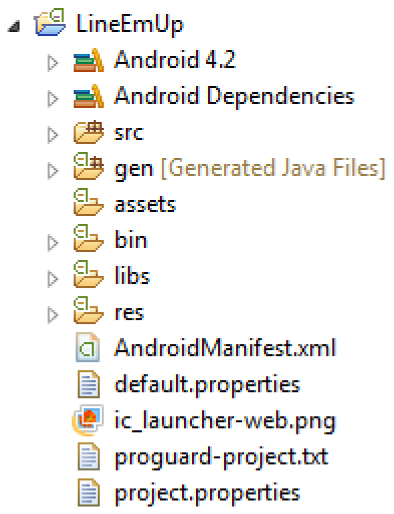
3.1 System Requirements:

The Eclipse IDE with the Android Developer Tools (ADT) plugin installed is the recommended platform for Android development, and was also the integrated development environment used by the project members during this app's creation. Eclipse provides features to help build, test, debug, and package Android apps and is available for the major operating systems (Windows, Mac OS, and Linux). In depth installation instructions can be found at http://developer.android.com/tools/index.html.

3.2 Getting the Source Code:

Although there is no desire to keep the project private, a request for the source code must be made to one of the project members directly because the project has not been made available to the public (Note: please see the developer contact information section at the end of the documentation). Due to its relatively small size the entire project folder can easily be transmitted as an email attachment.

3.3 Project Folder Hierarchy:

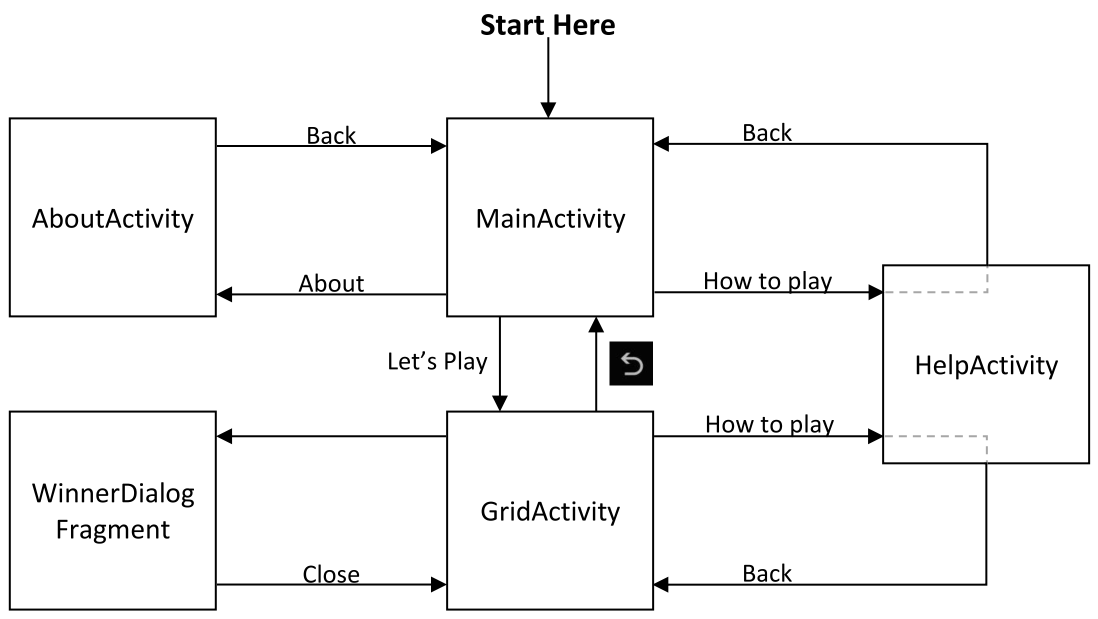


```
LineEmUp
  Android 4.2
  Android Dependencies
  src
  gen [Generated Java Files]
  assets
  bin
  libs
  res
  AndroidManifest.xml
  default.properties
  ic_launcher-web.png
  proguard-project.txt
  project.properties
```

The project folder (e.g. as shown on the left) contains many files/folders that are automatically created when using the Eclipse IDE with the Android development tools plugin installed and that are required to build the actual Android application. The following folders/files which contain the developer created content are of highest interest to future maintainers:

- **src**: contains all developer created java files (e.g. the various classes that make up the project).
- **res**: contains the resources (e.g. images, text and .xml layout files) used in the project.
- AndroidManifest.xml: contains essential information about the application to the Android system (e.g. information the system must have before it can run any of the application's code).

3.4 Class Interaction



3.5 Class Descriptions & Diagrams:

I. Brief Background on the Android Activity Class:

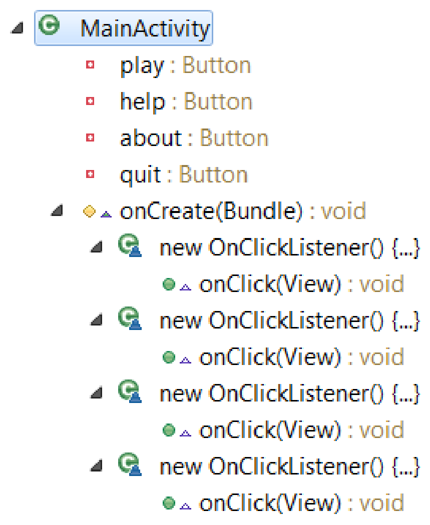
An Android Activity (Android.app.Activity) is an application component that provides a screen with which users can interact in order to do something. Each activity is given a window in which to draw its user interface. An application usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions. More information on the Activity class and other android specific classes/functions that will be mentioned can easily be found on developer.android.com.

Initialization of an Activity:
```
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_main);
}
```

To modify the layout of an activity, refer to the resource section (Section 3.6).

II. MainActivity

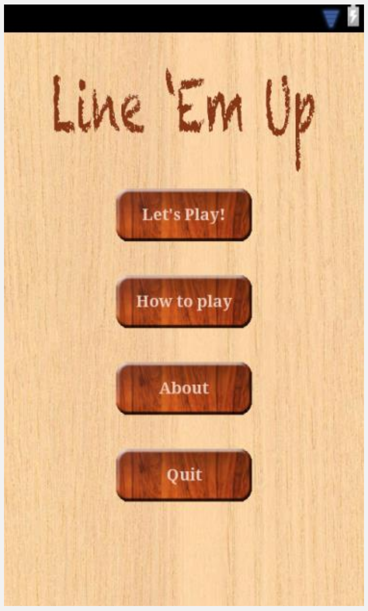
Class Outline



```
MainActivity
    play : Button
    help : Button
    about : Button
    quit : Button
  onCreate(Bundle) : void
    new OnClickListener() {...}   ← for play button
        onClick(View) : void
    new OnClickListener() {...}   ← for help button
        onClick(View) : void
    new OnClickListener() {...}   ← for about button
        onClick(View) : void
    new OnClickListener() {...}   ← for quit button
        onClick(View) : void
```

* See code for full descriptions of all class functions and variables.

What You Can See…



← play

← help

← about

← quit

This is the activity of the home page. It instantiates the four button and implements their setOnClickListener function.

Instantiate a button variable:

```
private Button play = null;
```

Connect it to the button in the view:

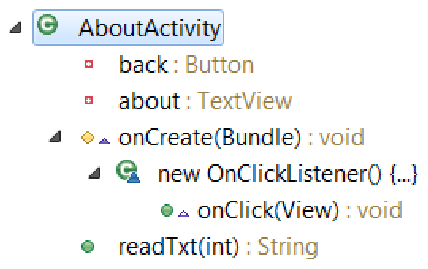
```
play = (Button)findViewById(R.id.play);
```

Implementation of its setOnClickListener function. When clicking the button, the GridActivity will be the current activity.

```
play.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View v) {;
      Intent intent = new Intent(MainActivity.this.GridActivity.class);
      startActivity(intent);
      }
});
```

III. AboutActivity

Class Outline



```
AboutActivity
    back : Button
    about : TextView
  onCreate(Bundle) : void
    new OnClickListener() {...}
      onClick(View) : void
  readTxt(int) : String
```

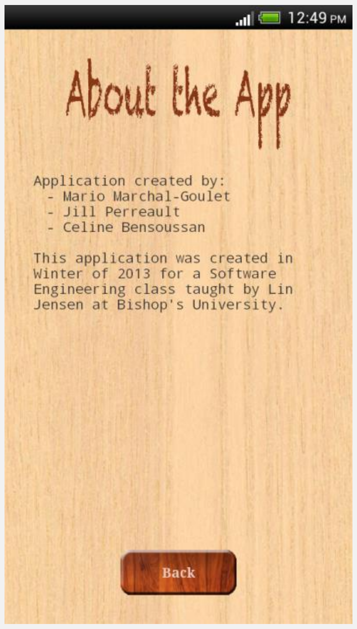
*** See code for descriptions of all class functions and variables.**

The AboutActivity is created with an empty TextView. The text that it then displays is read from a text file located in the folder `res/raw/about.txt`.

To do that, it uses an `InputStream` and a `BufferedReader`.

It contains one button used to go back to the home page.

What You Can See…

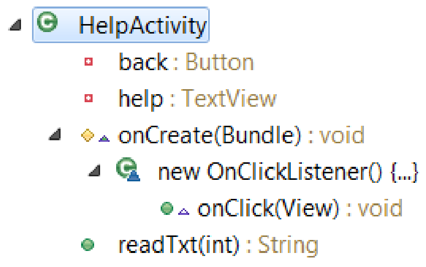


About the App

```
Application created by:
 - Mario Marchal-Goulet
 - Jill Perreault
 - Celine Bensoussan

This application was created in
Winter of 2013 for a Software
Engineering class taught by Lin
Jensen at Bishop's University.
```

IV. HelpActivity

Class Outline



```
HelpActivity
    back : Button
    help : TextView
  onCreate(Bundle) : void
    new OnClickListener() {...}
      onClick(View) : void
  readTxt(int) : String
```

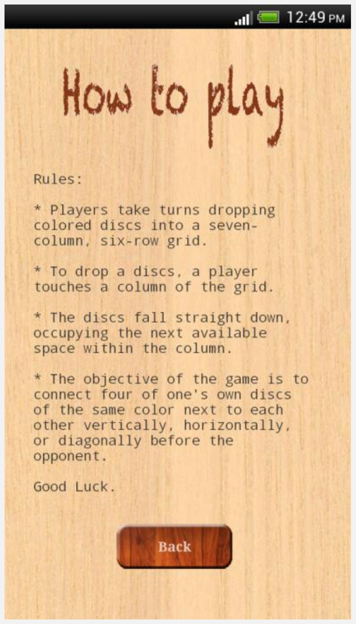
*** See code for descriptions of all class functions and variables.**

The HelpActivity is created with an empty TextView. The text that it then displays is read from a text file located in the folder `res/raw/help.txt`

To do that, it uses an `InputStream` and a `BufferedReader`.

It contains one button used to go back to the home page.

What You Can See…

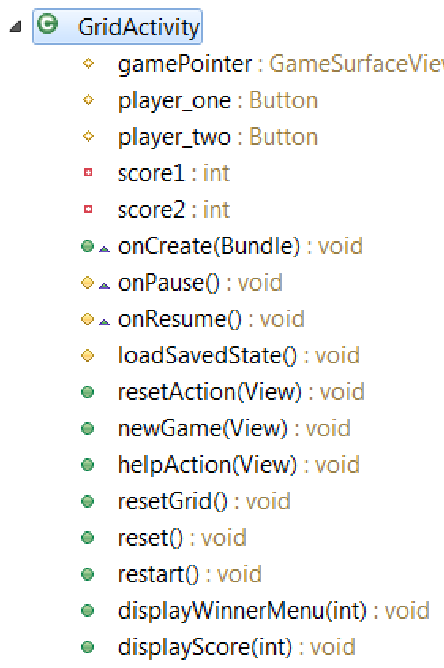


How to play

```
Rules:

* Players take turns dropping
colored discs into a seven-
column, six-row grid.

* To drop a discs, a player
touches a column of the grid.

* The discs fall straight down,
occupying the next available
space within the column.

* The objective of the game is to
connect four of one's own discs
of the same color next to each
other vertically, horizontally,
or diagonally before the
opponent.

Good Luck.
```

V. GridActivity

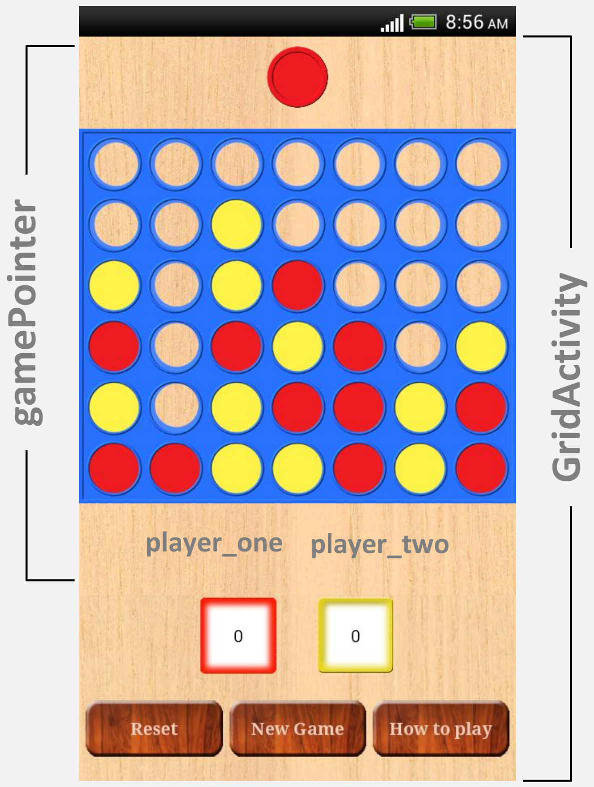
Class Outline



```
GridActivity
    gamePointer : GameSurfaceView
    player_one : Button
    player_two : Button
    score1 : int
    score2 : int
    onCreate(Bundle) : void
    onPause() : void
    onResume() : void
    loadSavedState() : void
    resetAction(View) : void
    newGame(View) : void
    helpAction(View) : void
    resetGrid() : void
    reset() : void
    restart() : void
    displayWinnerMenu(int) : void
    displayScore(int) : void
```

* Some of the class member functions and variables were left out of the outline due to their limited importance in the understanding of the class. **See code for descriptions of all class functions and variables.**

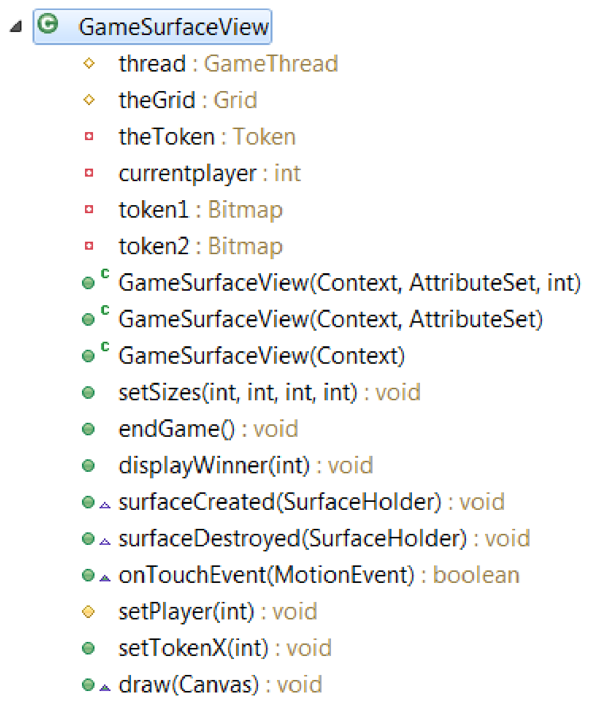
What You Can See…



- **GridActivity** extends **Activity**.
- Initiated when the **play** button is clicked in the **MainActivity**.
- Implements the **activity_grid** layout.
- Displays the **GameSurfaceView** object, two score buttons (**player_one** and **player_two**), and a “Reset”, “New Game” and “How to play” button.
- Buttons “Reset”, “New Game” and “How to play” are fully defined in the **activity_grid** layout and therefore will not appear in the code. As the layout states, functions **restAction(View)**, **newGame(View)** and **helpAction(View)** will be called when the “Reset”, “New Game” and “How to play” buttons are clicked respectively.
- Application data is saved when the **onPause()** function of this class is called.
- Application data is reloaded when the **loadSavedState()** function is called.
- The **reset()** function resets both the current game and the scores
- The **restart()** function resets only the current game.
- The **displayWinnerMenu(int)** function creates and displays the **WinnerDialogFragment**.

VI. GameSurfaceView

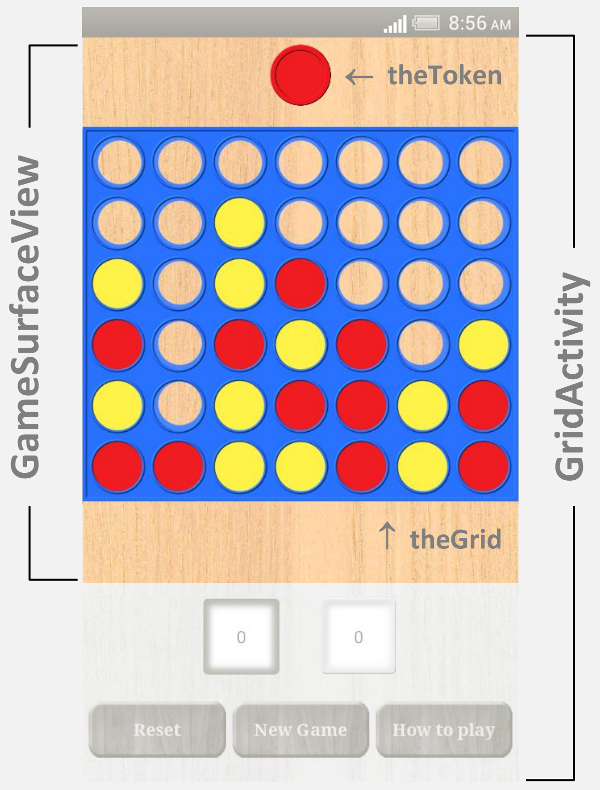
Class Outline



```
GameSurfaceView
    thread : GameThread
    theGrid : Grid
    theToken : Token
    currentplayer : int
    token1 : Bitmap
    token2 : Bitmap
    GameSurfaceView(Context, AttributeSet, int)
    GameSurfaceView(Context, AttributeSet)
    GameSurfaceView(Context)
    setSizes(int, int, int, int) : void
    endGame() : void
    displayWinner(int) : void
    surfaceCreated(SurfaceHolder) : void
    surfaceDestroyed(SurfaceHolder) : void
    onTouchEvent(MotionEvent) : boolean
    setPlayer(int) : void
    setTokenX(int) : void
    draw(Canvas) : void
```

* Some of the class member functions and variables were left out of the outline due to their limited importance in the understanding of the class. **See code for descriptions of all class functions and variables.**

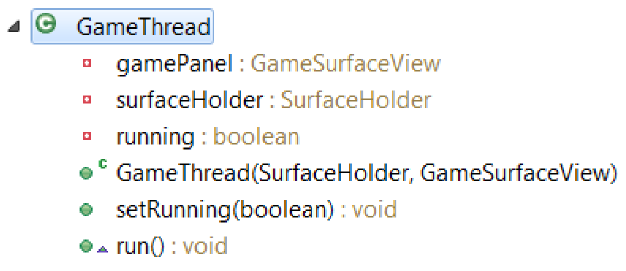
What You Can See…



- **GameSurfaceView** extends **SurfaceView**.
- An object of this class is instantiated when the **GridActivity** object is created.
- **GameSurfaceView** does not implement a layout, instead it uses a canvas.
- Called when **GameSurfaceView** is created, **surfaceCreated(SurfaceHolder)** starts **thread**.
- Called when **GameSurfaceView** is destroyed, **surfaceDestroyed(SurfaceHolder)** tells **thread** to finish and waits for it to end.
- **setPlayer(2)** sets **currentplayer** to 2 and **theToken**'s Bitmap (image) to **token2.**
- **setTokenX(int)** updates **theToken**'s x-coordinate.
- **setSizes(…)** sets the size of the grid (board) and all Bitmaps involved.
- **endgame()** stops **thread** from updating the canvas.
- **onTouchEvent(MotionEvent)**, called whenever a player touches the **GameSurfaceView**. Handles actions such as `MoveEvent.ACTION_DOWN` (on click), `MoveEvent.ACTION_MOVE` (on drag) and `MoveEvent.ACTION_UP` (on release).
- **draw(Canvas)**, adds **theToken** and **theGrid** to the canvas by calling their respective draw functions.

VII. GameThread

Class Outline



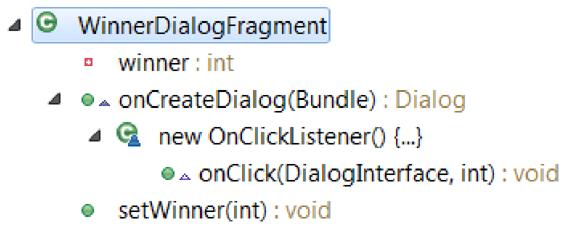
- GameThread
  - gamePanel : GameSurfaceView
  - surfaceHolder : SurfaceHolder
  - running : boolean
  - GameThread(SurfaceHolder, GameSurfaceView)
  - setRunning(boolean) : void
  - run() : void

* **See code for descriptions of all class functions and variables.**

- **GameThread** extends **Thread.**
- While **running** is true **GameThread** continuously re-draws the canvas upon any change.
- A change occurs when a player touches the grid.
- **surfaceHolder** allows **GameThread** to edit the pixels in the **GameSurfaceView**, and monitor changes to the **GameSurfaceView**.

VIII. WinnerDialogFragment

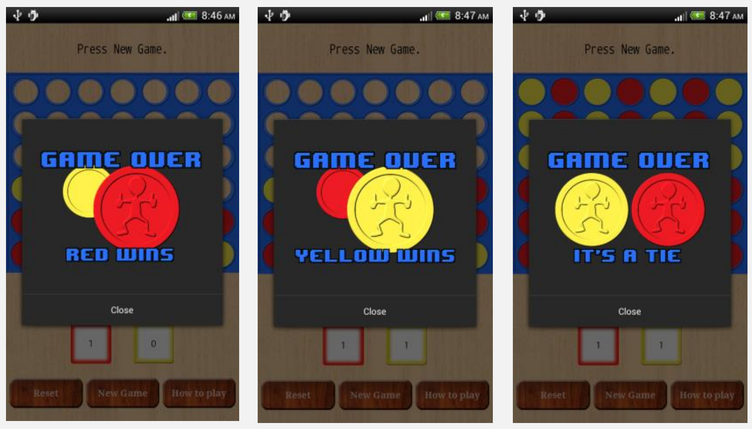
Class Outline



- WinnerDialogFragment
  - winner : int
  - onCreateDialog(Bundle) : Dialog
    - new OnClickListener() {...}
      - onClick(DialogInterface, int) : void
  - setWinner(int) : void

* Some of the class member functions and variables may have been left out of the outline due to their limited importance in the understanding of the class. **See code for descriptions of all class functions and variables.**

What You Can See...

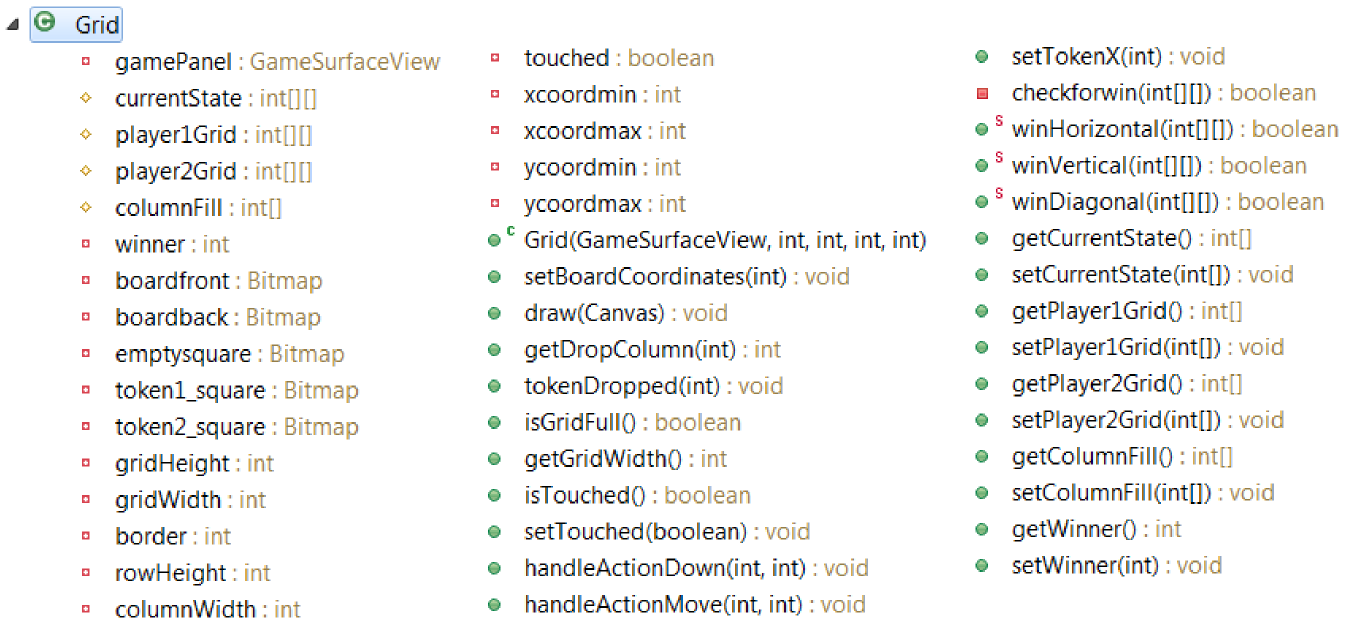


winner = 1 winner = 2 winner = 3

- **WinnerDialogFragment** extends **DialogFragment**.
- An object of this class is created when a player wins or there is a tie (the **Grid** is full).
- Depending on the value of **winner**, **WinnerDialogFragment** implements the following layouts:
  - If **winner** = 1 -> implements the **token1_winner** layout
  - If **winner** = 2 -> implements the **token2_winner** layout
  - If **winner** = 3 -> implements the **tie_winner** layout
- The **new OnClickListener**(){... **onClick(DialogInterface,int)**...} is created when making the **Close** button. The **Close** button simply closes the **WinnerDialogFragment** (pop-up), returning players to the **GridActivity**.

IX. Grid

Class Outline

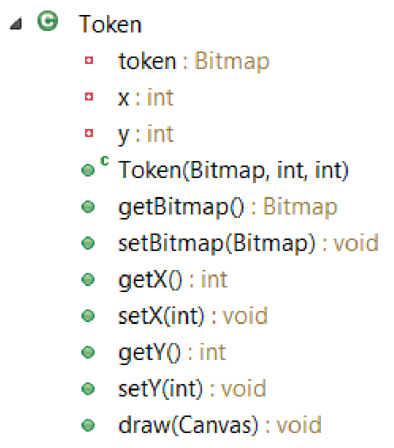


```
Grid
  gamePanel : GameSurfaceView
  currentState : int[][]
  player1Grid : int[][]
  player2Grid : int[][]
  columnFill : int[]
  winner : int
  boardfront : Bitmap
  boardback : Bitmap
  emptysquare : Bitmap
  token1_square : Bitmap
  token2_square : Bitmap
  gridHeight : int
  gridWidth : int
  border : int
  rowHeight : int
  columnWidth : int
  touched : boolean
  xcoordmin : int
  xcoordmax : int
  ycoordmin : int
  ycoordmax : int
  Grid(GameSurfaceView, int, int, int, int)
  setBoardCoordinates(int) : void
  draw(Canvas) : void
  getDropColumn(int) : int
  tokenDropped(int) : void
  isGridFull() : boolean
  getGridWidth() : int
  isTouched() : boolean
  setTouched(boolean) : void
  handleActionDown(int, int) : void
  handleActionMove(int, int) : void
  setTokenX(int) : void
  checkforwin(int[][]) : boolean
  winHorizontal(int[][]) : boolean
  winVertical(int[][]) : boolean
  winDiagonal(int[][]) : boolean
  getCurrentState() : int[]
  setCurrentState(int[]) : void
  getPlayer1Grid() : int[]
  setPlayer1Grid(int[]) : void
  getPlayer2Grid() : int[]
  setPlayer2Grid(int[]) : void
  getColumnFill() : int[]
  setColumnFill(int[]) : void
  getWinner() : int
  setWinner(int) : void
```

- Grid takes care of the actual playing.
- **handleActionDown()** is called when the player first touches the **gamePanel** (**GameSurfaceView**). **Grid** then checks if the player is actually touching within the **Grid**'s coordinates. If so, **gamePanel**'s **theToken**'s x-coordinate is updated.
- Similarly, **handleActionMove()** updates **theToken**'s x-coordinate, as the player drags their finger across the board.
- Functions **setCurrentState(…)**, **setPlayer1Grid(…)**, **setPlayer2Grid(…)**, **setColumnFill(…)** and **setWinner(…)** are called within **GridActivity**'s **loadSavedState()**.
- Functions **getCurrentState()**, **getPlayer1Grid()**, **getPlayer2Grid**, **getColumnFill(), and getWinner()** are called when saving **GridActivity**'s current state.
- **draw(Canvas)** uses **currentState** to draw the grid onto the canvas.

X. Token

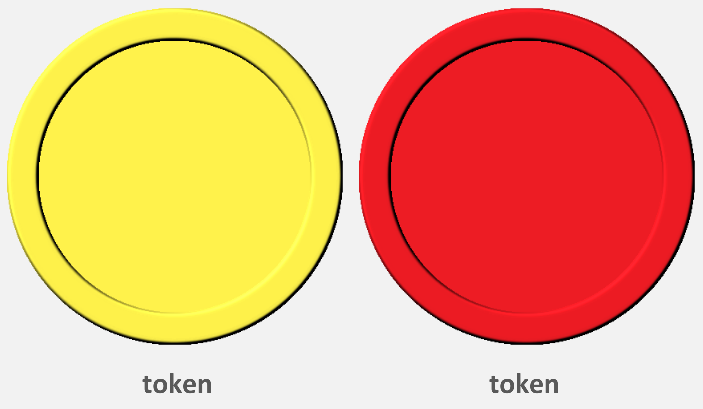
Class Outline



```
Token
    token : Bitmap
    x : int
    y : int
    Token(Bitmap, int, int)
    getBitmap() : Bitmap
    setBitmap(Bitmap) : void
    getX() : int
    setX(int) : void
    getY() : int
    setY(int) : void
    draw(Canvas) : void
```

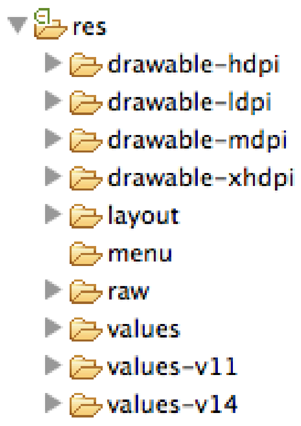
* **See code for descriptions of all class functions and variables.**

What You Can See…



- An object of this class is instantiated when the **GameSurfaceView** object is created.
- A **Token** is what is seen above the **Grid** in the **GameSurfaceView**.
- **token** (image) is either yellow or red depending on the current player.
- **draw(Canvas)** adds the token to the canvas at position (x, y).

3.6 Resources:

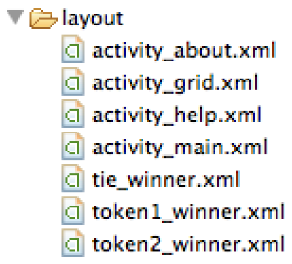

res
drawable-hdpi
drawable-ldpi
drawable-mdpi
drawable-xhdpi
layout
menu
raw
values
values-v11
values-v14

The resource folder contains the folders shown.

The drawable folders contain the images to be used in the application. The difference between the four folders is the screen resolution of the android device used:

- ldpi: around 120dpi
- mdpi: around 160 dpi
- hdpi: around 240 dpi
- xhdpi: around 320 dpi

Layout


layout
activity_about.xml
activity_grid.xml
activity_help.xml
activity_main.xml
tie_winner.xml
token1_winner.xml
token2_winner.xml

The Layout folder contains the XML files which determine the layout of the various activities in the application.

They can either be handled using the Graphical Layout and the Palette or they can be written directly from the XML file.

Raw

Raw is another folder than contains resources other than images such as text files.

Values

This folder contains two xml files: strings.xml and styles.xml

Strings

All the strings used in the application need to be declared in this file before being used. The string name is then used as its id to be referenced somewhere else in the app.

```
<resources>
   <string name="app_name">Line \'Em Up</string>
   <string name="button_OK">Let\'s Play!</string>
   ...
</resources>
```

Styles

A style should be declared anytime there is repetitive code (or xml attributes) in the layout.

Our file contains two styles:

- MyTheme: setting the image of the background on all main activities.
- button_style: setting the style of all buttons

Once a style has been created, it can be used as follows in the xml code of an activity's layout:

```
<Button
        style="@style/button_style" />
```

3.7 Potentially Reusable Components:

- Although very little of the actual code would be reusable as is, many of the patterns (e.g. state saving, setting sizes of content, and detecting four elements in a row in a matrix) could potentially be useful in different situations with slight code modification.

3.8 References

Reference material (e.g. websites/textbooks) used by the developers in the creation of the application and the production of this documentation which could be of interest to future developers:

- developer.android.com
- javacodegeeks.com
- *Professional Android™ 4 Application Development*, by Reto Meier

Developers Contact Information

Please do not hesitate to contact us with requests for the source code or the app itself. We will also happily answer any questions regarding the application to the best of our abilities. We can be reached at the following email addresses:

mario_marchal@hotmail.com
jperreault10@ubishops.ca
cbensoussan09@ubishops.ca

We thank you for your interest in our Android application project and we hope you enjoy the game and that you can benefit from our source code.