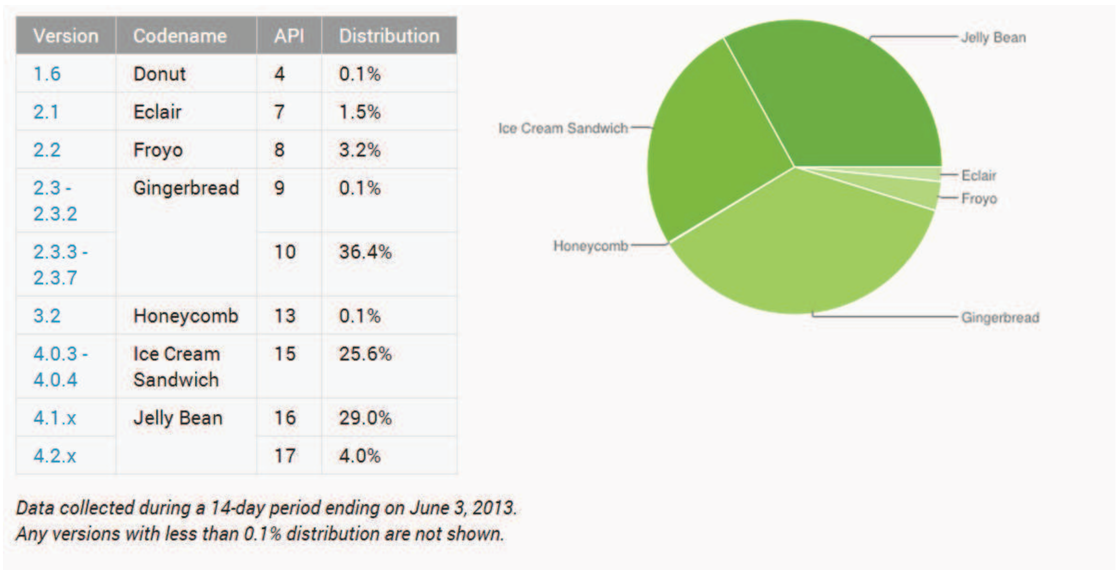
1. Overview

The objective of this document is to describe the design implementation validated for the development of the original game Minesweeper on the mobile platform Android.

Multi Device compatibility

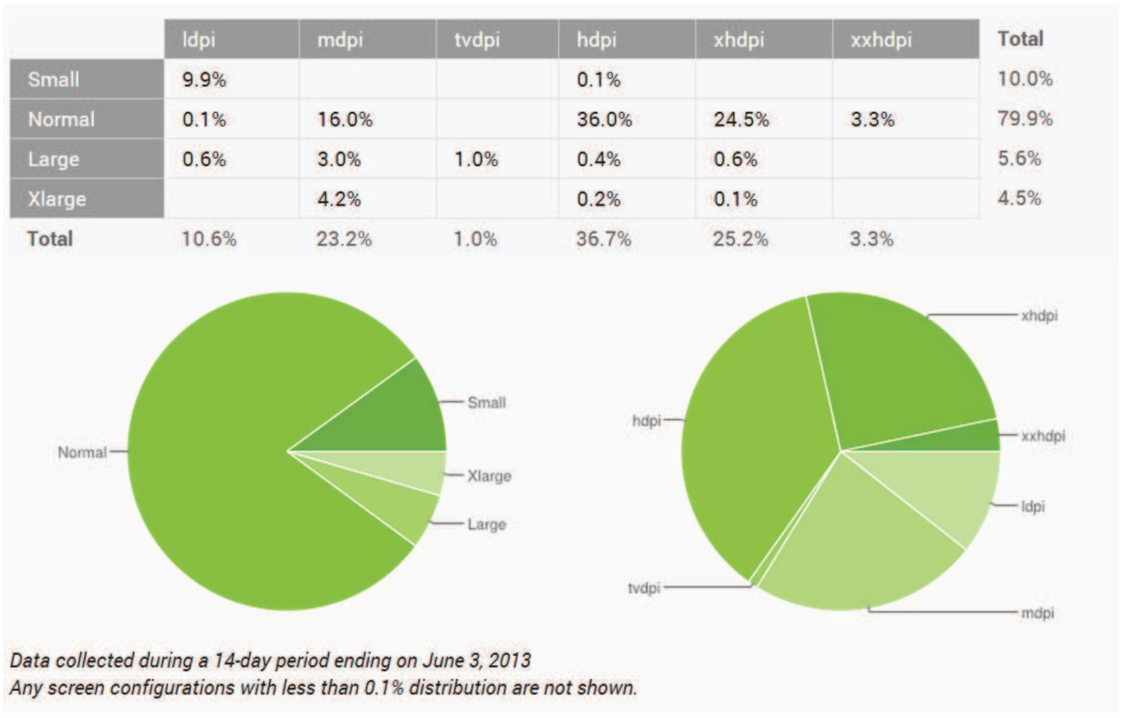
The implementation should be developed with the objective to be a maximum compatible with the different platform versions of Android and cover a good percentage of the market.



Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.5%
2.2	Froyo	8	3.2%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	36.4%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	25.6%
4.1.x	Jelly Bean	16	29.0%
4.2.x		17	4.0%

Data collected during a 14-day period ending on June 3, 2013.
Any versions with less than 0.1% distribution are not shown.

Moreover since Android has a large distribution and it is supported by lots of devices, the final application should be adaptive to the different screen sizes and formats.



	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	9.9%			0.1%			10.0%
Normal	0.1%	16.0%		36.0%	24.5%	3.3%	79.9%
Large	0.6%	3.0%	1.0%	0.4%	0.6%		5.6%
Xlarge		4.2%		0.2%	0.1%		4.5%
Total	10.6%	23.2%	1.0%	36.7%	25.2%	3.3%	

Data collected during a 14-day period ending on June 3, 2013
Any screen configurations with less than 0.1% distribution are not shown.

Environment consideration

Programming for mobile platforms implies taking into account the different issues and restrictions relative to the devices. Hence the developer has to manage with the limited resources and lifecycle activity process by adopting a specific programming strategy accordingly.

General guidance

- The development of the minesweeper game should follow the KISS principle (Keep it simple, stupid).
- The application should be close to the android design philosophy.
- A good responsiveness of the game is required to ensure a good end-user experience.

Game rules implementation:

- The game should start when the player uncovers the first square of the board.

- A square is uncovered basically by simple click (we assume no flag tool) .The player can uncover a square only if this square is not flagged.

- To flag a square the player has to switch to the flag tool and can add/remove flag by simple click.

- The game finishes either in game over situation or if the player wins.

- It is game over if a square hiding a mine is uncovered.

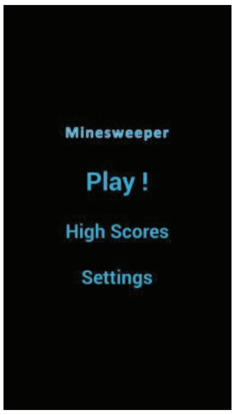
- The player wins if all squares except mined square are uncovered.

-In normal game mode if the player wins his score is the number of moves made. Adding a flag counts as a move (in order to add challenge). In timed game mode the best score is the shortest time for completing the game.

2. Interface Requirements

The application will be drawn entirely thanks to the default canvas and views. For a basic storyboard game it is not necessary to implement an external graphic library as it would be wise to do for more complex scenario games. In order to have a simple and efficient way to display each component, the basics of different layouts and views will be enough. Each screen has to be designed with relative dimension in order to accommodate a variety of phones screen sizes.

There will be several screens which permit the users to have the different interactions mandatory by the defined features.



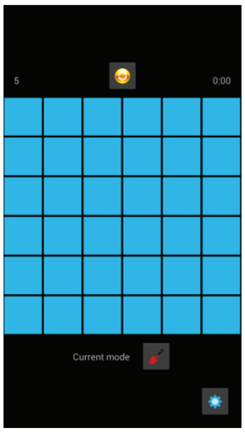
The menu screen: Will be the first screen when the application is launched. The three actions required are:

- Play: As explicitly mentioned to start a new game of minesweeper.

- High Scores: To show the best scores in the two different game modes (normal and timed mode).

- Settings: To allow the player to change the options relative to the game and the application is general.

The game screen: Will be the screen most of the time displayed and where the player will play the game. The following elements are needed:



- A label displaying originally the number of hidden mines. Like the classic game, this counter should be decremented every time a flag is placed (can become negative).

- A label displaying the current score of the player.

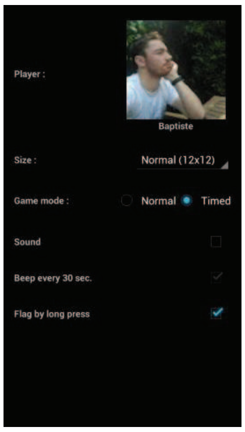
- The player should be able to quit the game by pressing the conventional back button (respects the android guideline recommendations). http://developer.android.com/design/patterns/navigation.html

- The game could be restarted by pressing the restart button.

- A dialog for quit and restart confirmation is mandatory if the game is started in order to avoid accidental end of game.

- Settings should also be accessible from the game in order to change the configuration.

<u>The settings screen:</u> Could be accessed from the menu screen or the game screen. This screen permits to change the game configuration and other details:


Player : Baptiste

Size : Normal (12x12)

Game mode : ○ Normal ● Timed

Sound ☐

Beep every 30 sec. ✓

Flag by long press ✓

-An element allowing the player to choose his name and photo from the contact list of the device. This change should be effective instantly if a user is actually playing.

-A picker to choose the size of the minefield. The large one should be accessible only for tablet (large screen). This change should be postponed to the next game.

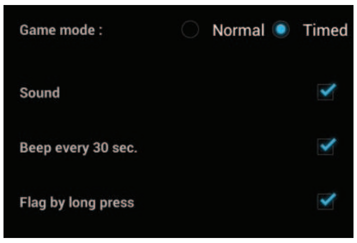
-A radio button to select the game mode (normal or timed). This change should be effective instantly if a user is actually playing (and appropriately defined).

- A checkbox to allow to flag by long press gesture. This change should be effective instantly if a user is actually playing.

-A checkbox for sound effects. This change should be effective instantly if a user is actually playing.

- A checkbox to activate the beep sound. This change should be effective instantly if a user is actually playing.

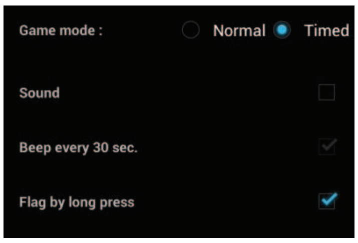
- If the checkbox corresponding to the sound is on, and the game is set on timed mode, it is possible to activate a beep sound for every 30 seconds of playing time. If any of these conditions is not respected, the beep checkbox is disabled.


Game mode : ○ Normal ● Timed

Sound ✓

Beep every 30 sec. ✓

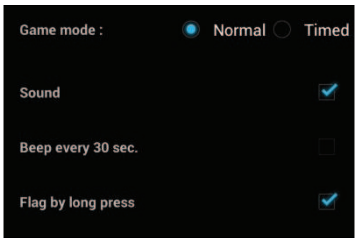
Flag by long press ✓


Game mode : ○ Normal ● Timed

Sound ☐

Beep every 30 sec. ✓

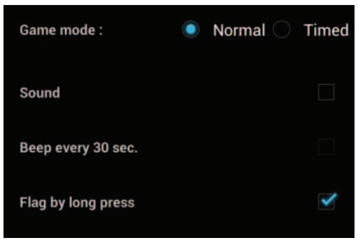
Flag by long press ✓


Game mode : ● Normal ○ Timed

Sound ✓

Beep every 30 sec. ☐

Flag by long press ✓

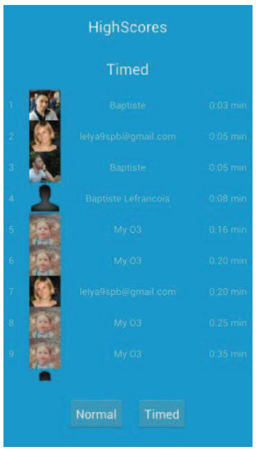

Game mode : ● Normal ○ Timed

Sound ☐

Beep every 30 sec. ☐

Flag by long press ✓

<u>The high scores screen:</u> Has to store the 10 best scores played for each game mode since the installation of the game. The requirements are:



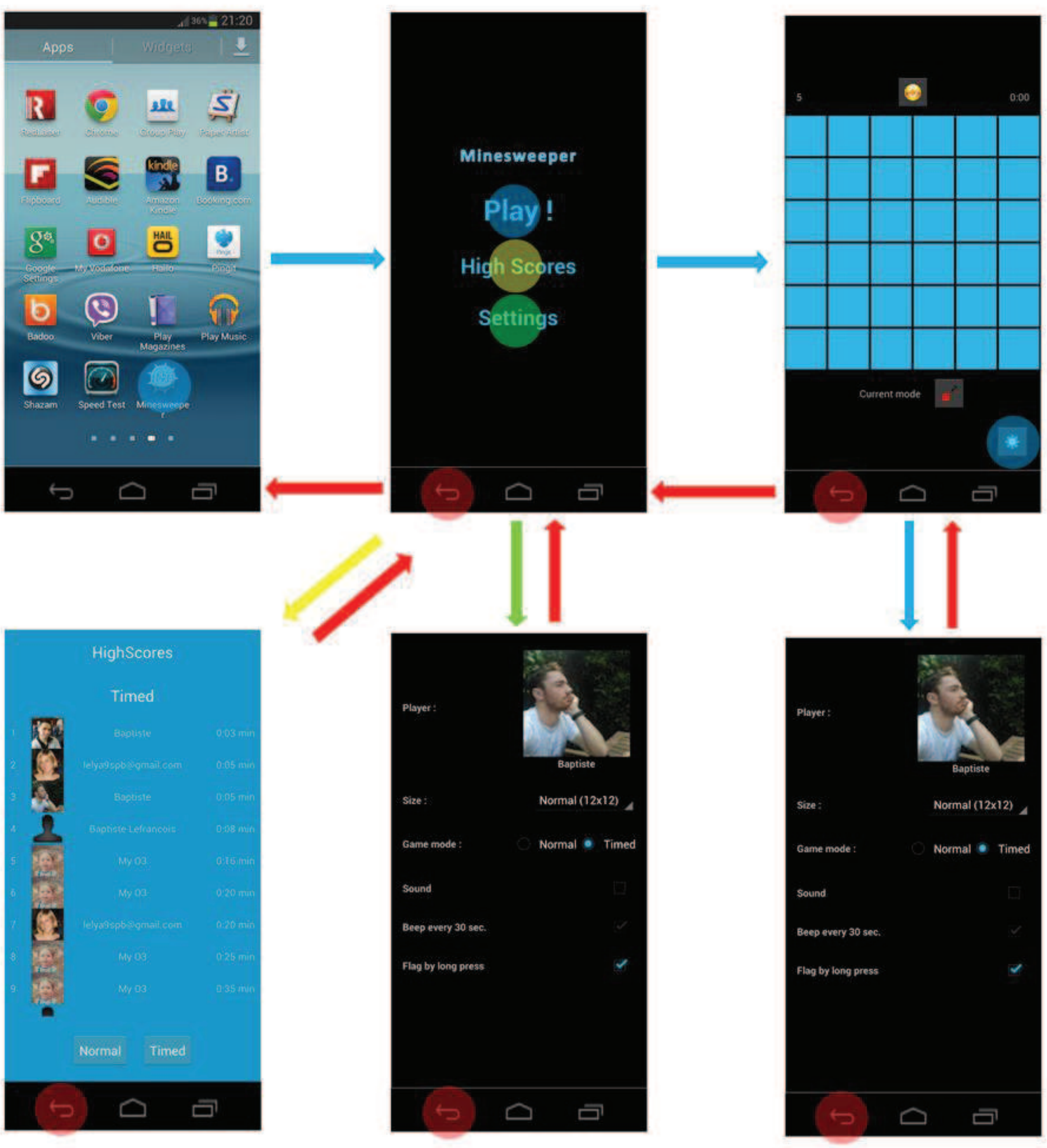
1	Baptiste	0:03 min
2	lelya9spb@gmail.com	0:05 min
3	Baptiste	0:05 min
4	Baptiste Lefrancois	0:08 min
5	My O3	0:16 min
6	My O3	0:20 min
7	lelya9spb@gmail.com	0:20 min
8	My O3	0:25 min
9	My O3	0:35 min

- Two buttons should permit to choose which high scores list should be displayed (normal or timed mode).

- The score has to be recorded in a database (SQLite). The name and a reference of the contact have to be saved too. Keeping a reference of the contact has the advantage to display dynamically the actual name and photo of the contact and manage future changes. A default icon is displayed if no photo found. If a contact is deleted the last option is to display the backup name associated to the score with a default icon.

Each screen presented is an activity. Their technical implication will be more detailed in the next part.

3. The activities navigation



HighScores

Timed

1		Baptiste	0:03 min
2		lelya9spb@gmail.com	0:05 min
3		Baptiste	0:05 min
4		Baptiste Lefrancois	0:08 min
5		My 03	0:16 min
6		My 03	0:20 min
7		lelya9spb@gmail.com	0:20 min
8		My 03	0:25 min
9		My 03	0:35 min

Normal Timed

Player : Baptiste

Size : Normal (12x12)

Game mode : Normal [✓ Timed]

Sound ☐

Beep every 30 sec. ✓

Flag by long press ✓

Player : Baptiste

Size : Normal (12x12)

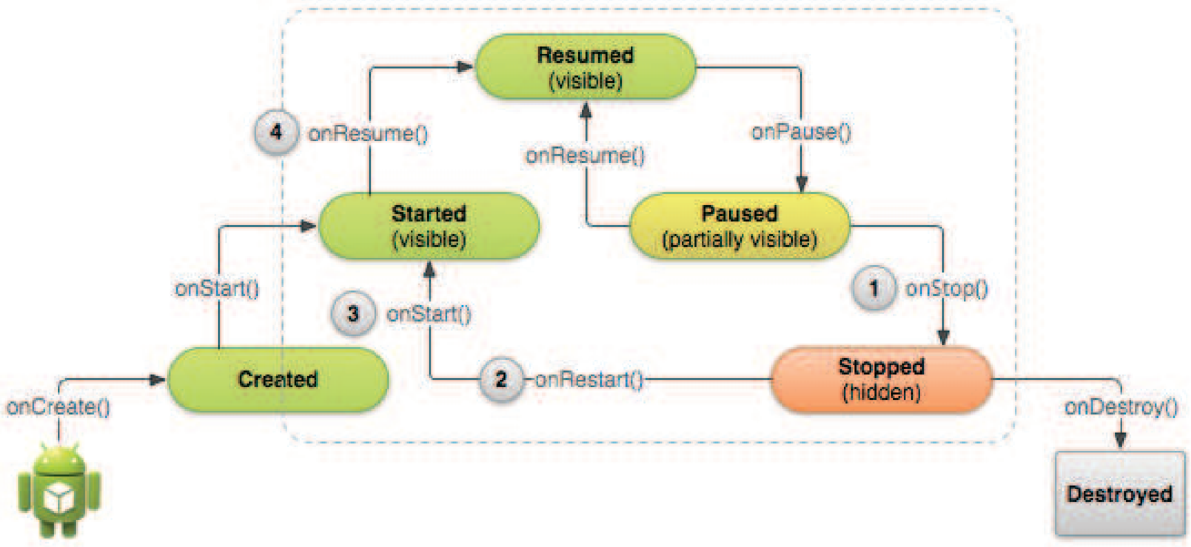
Game mode : Normal [✓ Timed]

Sound ☐

Beep every 30 sec. ✓

Flag by long press ✓

4. Life cycle management

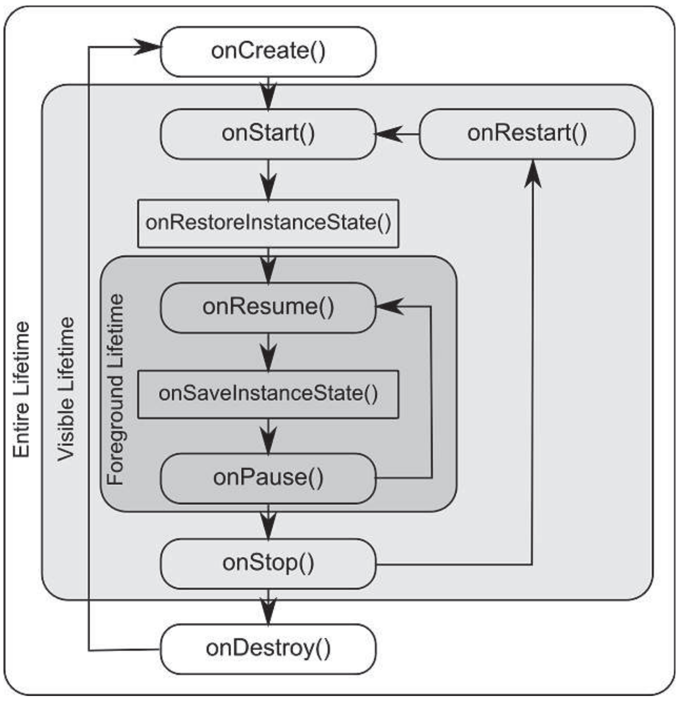


Source: http://stuff.mit.edu/afs/sipb/project/android/docs/training/basics/activity-lifecycle/stopping.html

The Minesweeper game has to take in consideration any external event which will occur during the application execution.

The previous schema show how a live cycle is managed for each activity. If a new activity is created, the parent activity goes into paused mode.

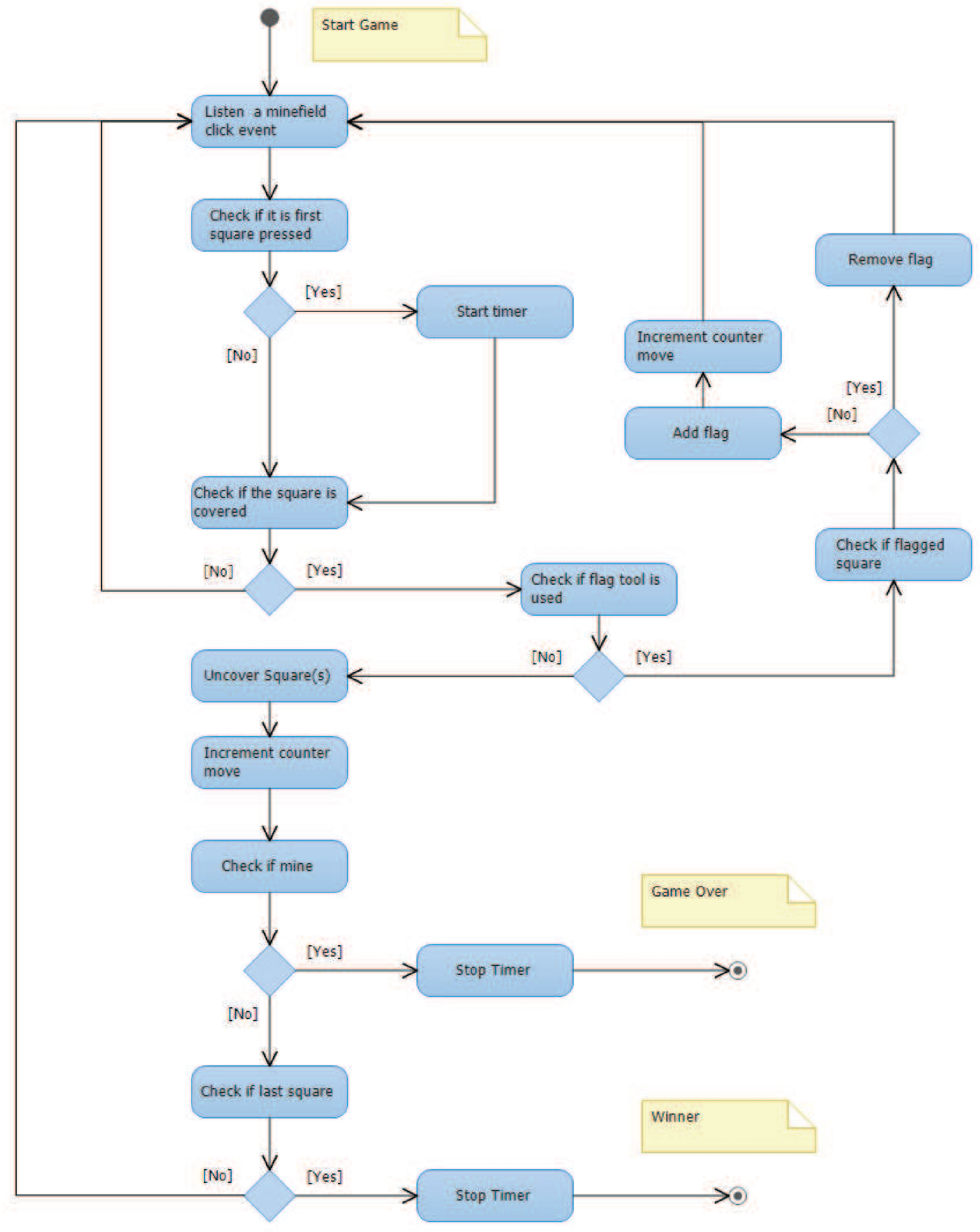
This first schema can be completed by the next one which shows more precisely how the activity is managed when the platform force the activity to quit.



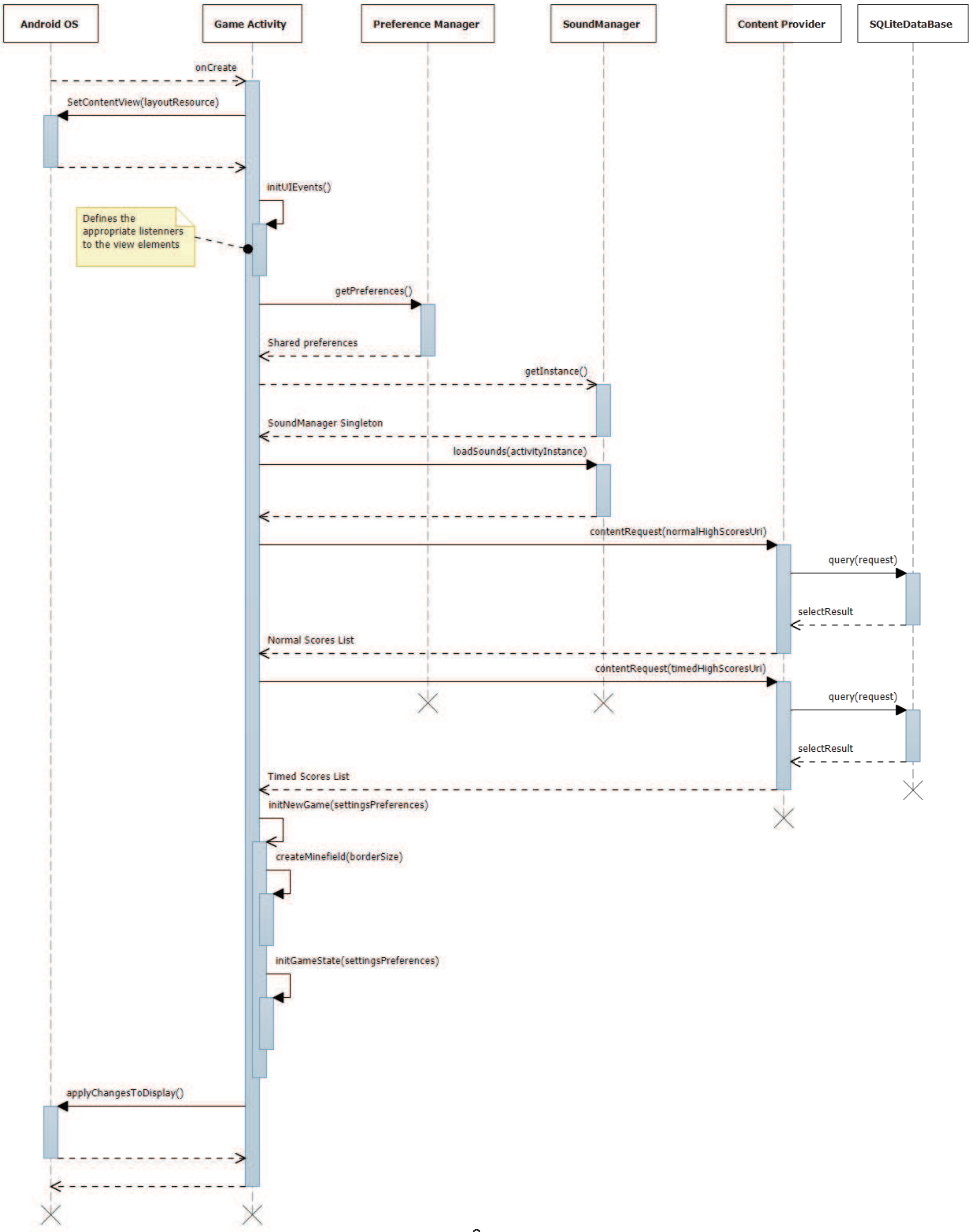
If order to safeguard the system, an activity can be killed at any time if resources are needed. To ensure that an activity can be restarted later (when it got again the priority) the method onSaveInstanceState(Bundle instance) is called. Thanks the bundle parameters the activity can save the information necessary for its recovery. The methods onCreate(Bundle savedInstance) and onRestoreInstance(Bundle savedInstance) permits this recover operation.

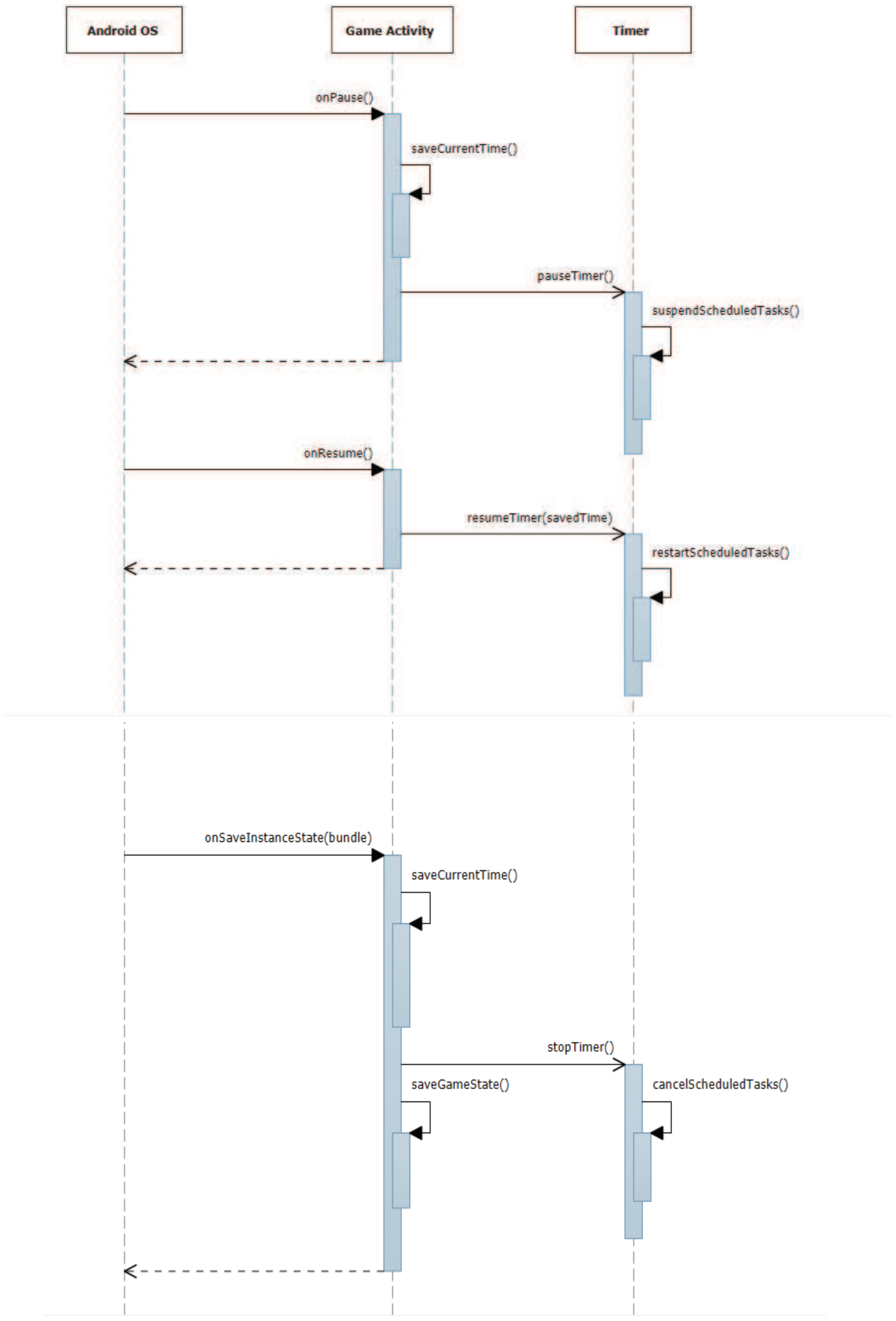
As already explicitly mentioned, the minesweeper game has to take in consideration the operating system limitations and must manage the activities interruptions introduced by external events (e.g. a phone call) by overriding the methods activities presented in this chapter when it is appropriate.

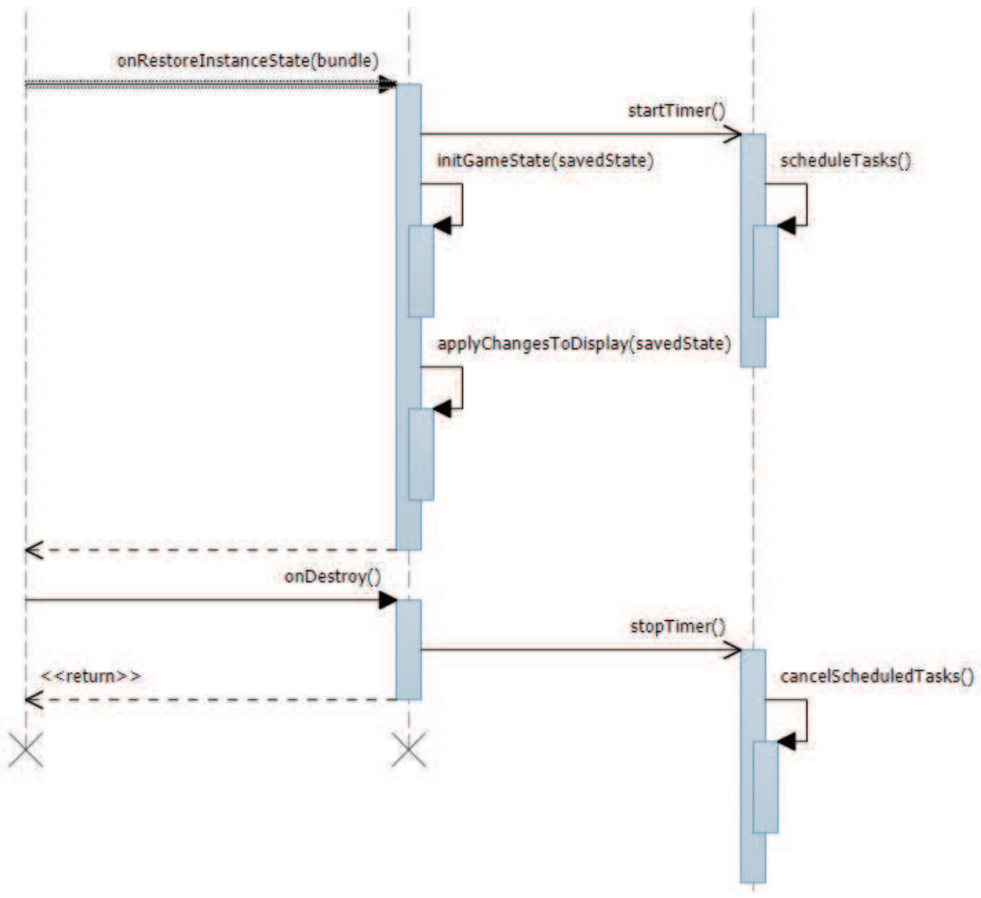
5. The logic associated to the game process.



6. Sequence diagram and implementation of the game activity




Android OS
Game Activity
Timer
onPause()
saveCurrentTime()
pauseTimer()
suspendScheduledTasks()
onResume()
resumeTimer(savedTime)
restartScheduledTasks()
onSaveInstanceState(bundle)
saveCurrentTime()
stopTimer()
saveGameState()
cancelScheduledTasks()


onRestoreInstanceState(bundle)
startTimer()
initGameState(savedState)
scheduleTasks()
applyChangesToDisplay(savedState)
onDestroy()
stopTimer()
<<return>>
cancelScheduledTasks()

These previous diagrams show how could be implemented the management of the game activity lifecycle. Several methods from the super class Activity have to be overridden in order to ensure a complete recovery mechanism.

The onPause() method manages classic interruptions when the activity is not destroyed, hence only the timer start/pause should be needed, the activity instance is still the same after the onResume() call and the others intern variables relative to the state of the game will not require a special attention.

A more complex save mechanism would be mandatory if the activity is forced to quit (not naturally by end-user action like the back button press). The minesweeper application should store a copy (like a cache version) in the bundle with the essential information required for restoring the same game state later.

Note : onSaveInstanceState and onRestoreInstanceState are mandatory methods if the minesweeper game needs to manage properly the screen orientation changes (implies destroy/recreate activity process).

7. Application development guidance

In order to have better performance some basic rules have to be followed as mentioned in the android developer documentation.

- The simple first rule is to avoid creating unnecessary objects. It is recommended for example to not instantiate a class directly at each iteration in for loops but create one reusable object before.

- If a method returns a string which will be used systematically in the same way (StringBuffer), it is better to pass the StringBuffer in parameters and append directly the value in the method.

- In internal code, it is better to use parallel arrays to store a list of tuples than using an ArrayList of key-value pairs.

- If a method does not need to access to object's fields, defining the method static improves the performance around 15%-20%.

- Add in the constants instantiation the keyword final (for example: static String msg = "Bonjour" becomes => static final String msg = "Bonjour").

- Do not use getters/setters in internal use and access the field directly (around seven times faster with JIT).

- Use the enhanced for loop by default for iterable object except for ArrayList where it is better to use a hand-written code loop.

- Choose to use existing libraries than writing your own. More than the common advantages of using a library, this one is often already the must optimised way to do an action (because may be generated in low level language).

To conclude this section, it should be highlighted that optimisation should not be done when you can, but when you have too.

"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified" — Donald Knuth