**Embedded System Software CSE4116**

**HW#4 Project :**

**Andriod Puzzle App with**

**android service**

**컴퓨터공학과**

**20131579 윤기영**

1. **개발 환경**

* **Windows 10 x64**
* **Android Studio 3.4.1**
* **Gradle 5.1**

(대부분의 프로그래밍을 개인 노트북의 Andriod studio와 AVD를 사용하였고, 완성 후, Target Board에서 작동하는지 테스트해보았다.)

1. **개발 목표**

보드에서 실행 가능한 간단한 Puzzle java application 프로그램을 작성한다.

1. **개발 내용 및 필요 배경지식**

* **보드에서 실행시킬 수 있는 Application 구현. 모든 버튼은 Linear Layout으로 구현.**

1. App 실행 시 초기화면

* 퍼즐 게임 버튼 : 퍼즐 게임 페이지로 넘어간다.
* 현재 화면에서 TextView를 통해 자신의 학번을 확인할 수 있게 한다.

1. 퍼즐 게임

* 퍼즐 개수는 EditText를 통해 [row col]형식으로 입력 받아, java file에서 dynamically 버튼을 만들어준다. 이 때 버튼의 숫자는 random하게 할당하고 버튼 중 한 개의 버튼은 빈 공간으로 다른 색의 버튼을 사용한다. 버튼의 개수는 최대 5X5까지로 한다. 빈 공간의 위, 아래, 왼쪽, 오른쪽에 있는 퍼즐 중 하나를 눌렀을 경우, 해당 퍼즐과 빈 공간의 위치가 바뀌게 한다. 모든 숫자가 정렬되고 빈 공간이 우측 하단에 위치하면 퍼즐 게임이 끝나고 초기화면으로 돌아간다.

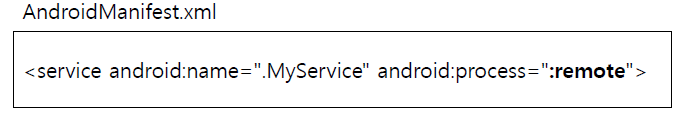
1. 추가 구현 (Android Service)

* Make Button을 누르기 시작한 이후의 퍼즐게임 플레이 시간을 나타내는 타이머를 Service로 구현한다. 이 시간은 매 게임마다 초기화 되어야 하며, 분과 초로 각각 2자리씩 출력한다.
* **Android Service**

Service는 어떤 task를 수행하기 위해 background에서 실행되는 프로세스이다. Android Framework에서는 여러가지 System Service를 제공하고 Application Service를 프로그래머가 Service class를 상속하여 구현할 수 있다. Application Service는 2가지로 분류된다.

**Local service :** 서비스를 요청하는 Service client 내에서 구현된다. Service를 만든 application 내에서만 사용되며, application이 종료될 시, Local service도 종료된다.

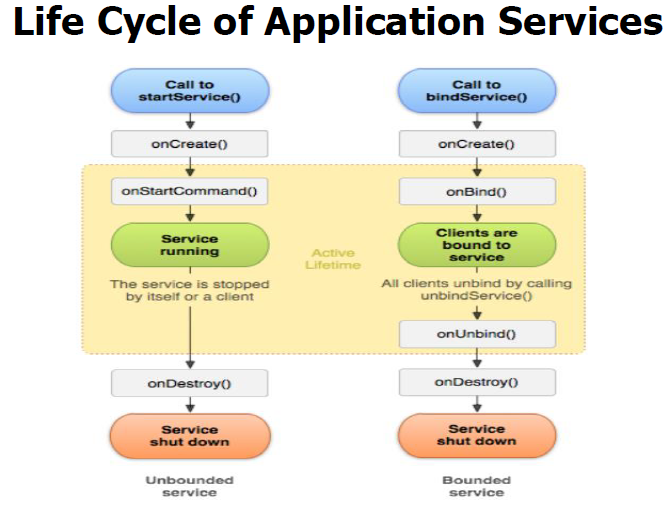
**Remote service :** 만들어진 Service와 Service client가 서로 다른 프로세스이다. Service client가 종료되더라도 Service는 계속 실행된다. AndroidManifest.xml 파일에서 "android:process" 필드에 ":remote"라는 값을 넣어주면 된다.



Application service는 started(unbounded) service와 bounded service 이 두 가지로 분류된다.

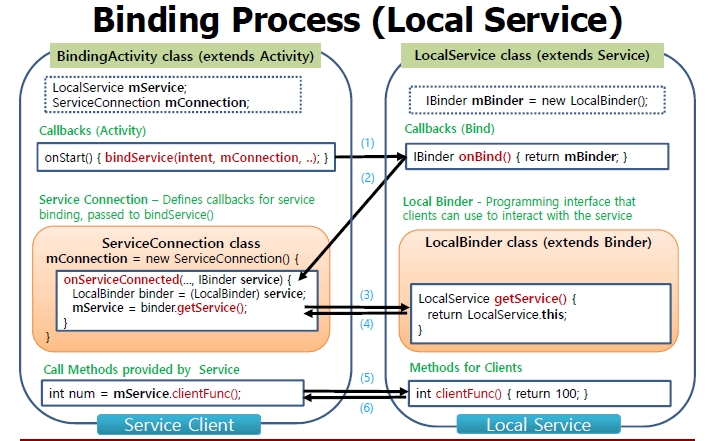
**Started (Unbounded) Service :** startService()로 시작하며 application이 system에게 어떤 것을 background에서 처리해달라고 요청하는 기능이다. 유저가 application과 직접 상호작용하지 않아도 된다.

**Bounded Service :** bindService()로 시작하며 어떤 application이 다른 application에게 자신의 기능을 제공할 수 있도록 하는 기능이다. service와의 상호작용을 위해 긴 connection을 허용한다.



* **Binder**

Application과 service는 다른 process에서 실행되더라도 서로 communicate하거나 data를 share해야 한다. 일부 IPC는 오버헤드가 크고 security적인 문제가 있기 때문에 Binder를 사용한다. Binder는 IPC 기능을 해주는 driver이며 RPC(Remote Procedure Call) programming paradigm과 유사하다. Binder는 모든 프로세스가 공유하는 커널 메모리(Shared memory)를 참조하게 함으로써 메모리 복사 오버헤드를 최소화한다.

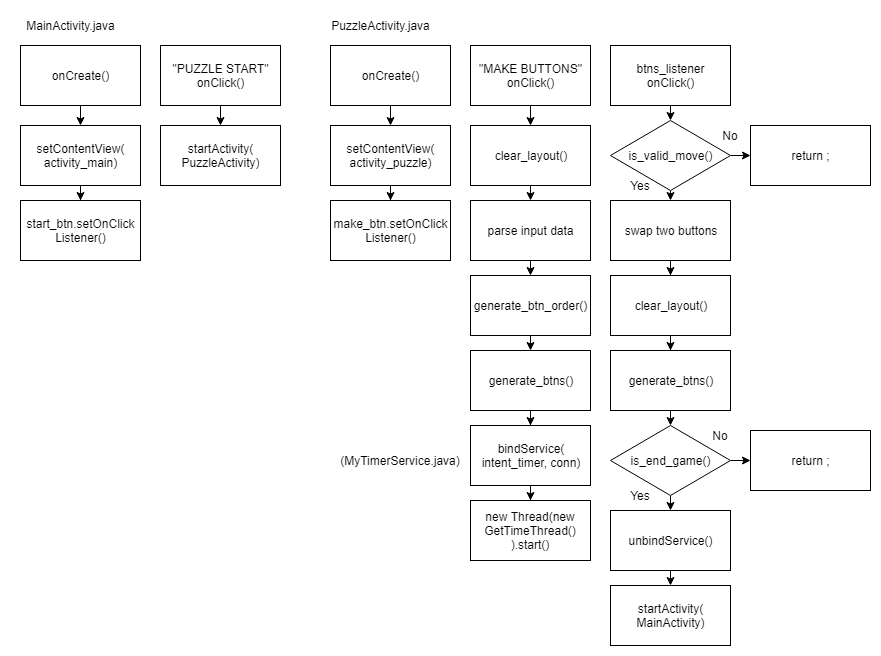


1. **개발 일정**

|  |  |
| --- | --- |
| 날짜 | 개발 내용 |
| 2019.06.07 | - Puzzle app의 기본화면과 퍼즐게임 activity 작성 |
| 2019.06.08 | - service를 이용한 timer 구현  - 섞여진 버튼이 항상 solvable 하도록 변경 |
| 2019.06.14 | - Service 부분을 AIDL을 사용하는 remote service에서 AIDL을 사용을 안 하도록 Local service로 변경  - 보고서 작성 |

1. **개발 결과**

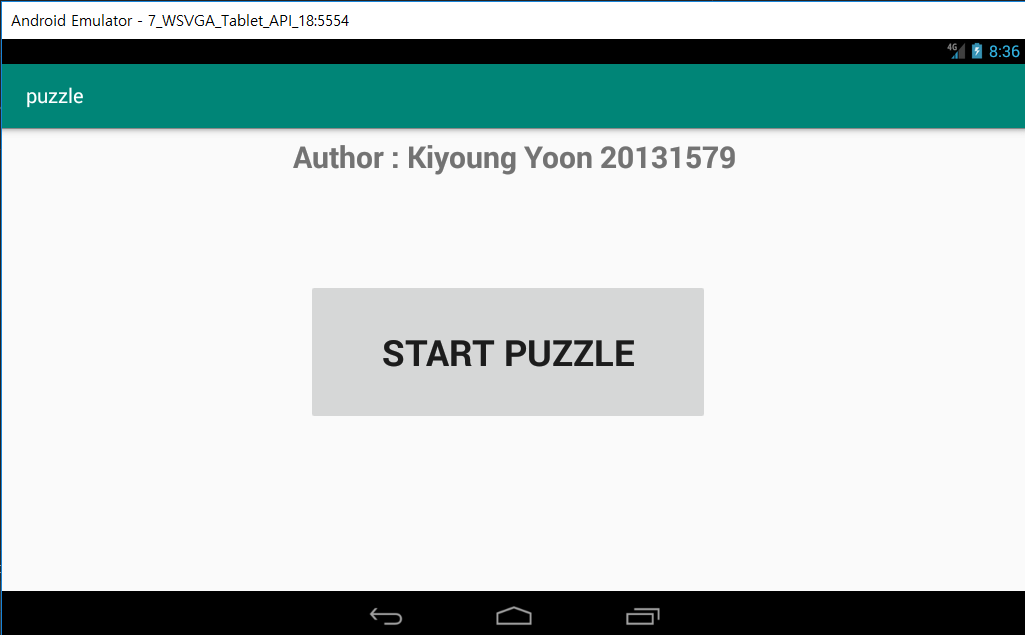
* **Flow chart**



* **보드에서 실행시킬 수 있는 Application 구현. 모든 버튼은 Linear Layout으로 구현.**

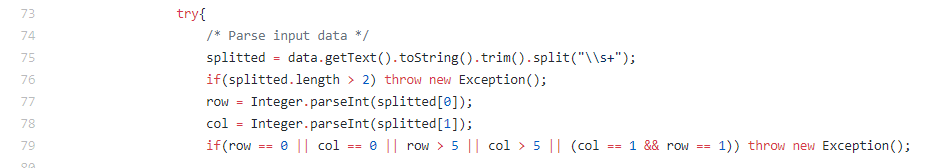
1. App 실행 시 초기화면

먼저 중앙 상단에 textView를 통해 이름과 학번을 출력하였다. 그리고 중앙 하단에 "START PUZZLE"이라는 버튼을 만들고 해당 버튼에 onClickListener를 만들어주었다. 이 listener는 버튼이 클릭 시, PuzzleActivity로 전환해주는 intent를 만들고 해당 intent를 이용하여 PuzzleActivity로 화면을 전환한다.



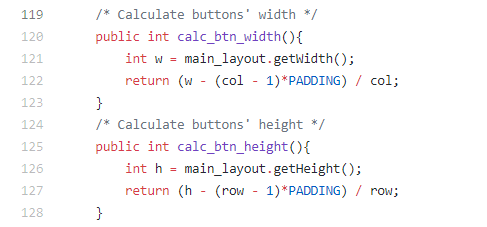
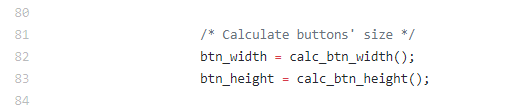
1. 퍼즐 게임

화면 상단에 EditText를 통해 User로부터 Puzzle의 size를 입력 받는다. 입력 받은 데이터는 String으로 접근하므로, row와 col에 대한 정보를 얻기 위해 parsing을 한다. 만약 parsing한 row, col 값이 invalid하다면 try-catch문을 통해 예외처리를 해준다.

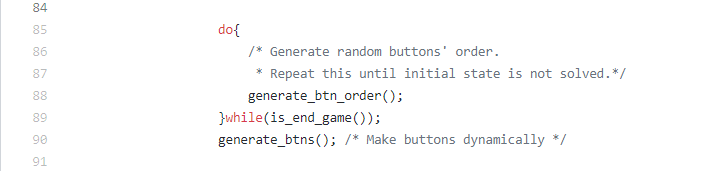


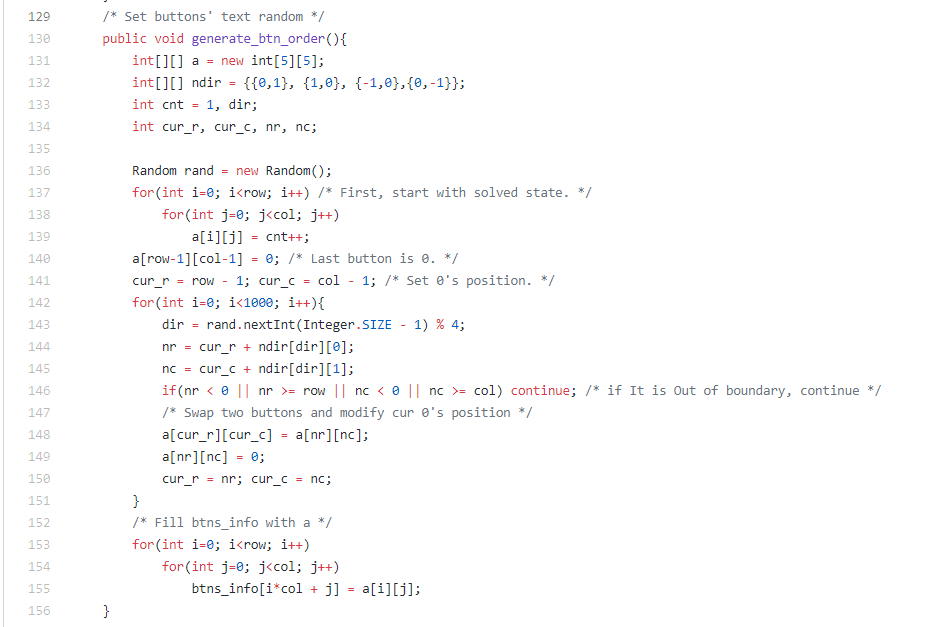
"Make Buttons"버튼의 onClickListener listener에서는 Puzzle을 시작하기 위해 버튼을 생성하고 타이머를 작동시키는 기능을 한다. 내용은 다음과 같다.

입력 받은 row, col 값만큼 버튼을 생성하기 위해서 버튼을 넣을 main\_layout을 미리 생성해 두었다. 그리고 버튼을 화면 가득히 채우기 위해서 각 버튼의 width, height를 계산한다. 이 때 각 버튼이 너무 붙지 않도록 PADDING값도 고려하여 계산한다. (PADDING = 16)

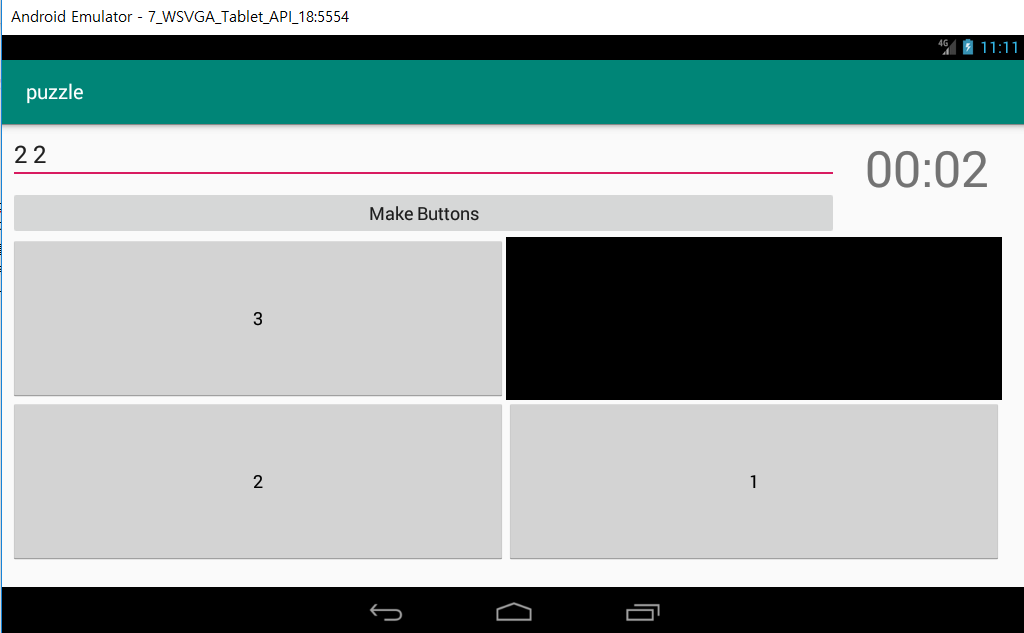


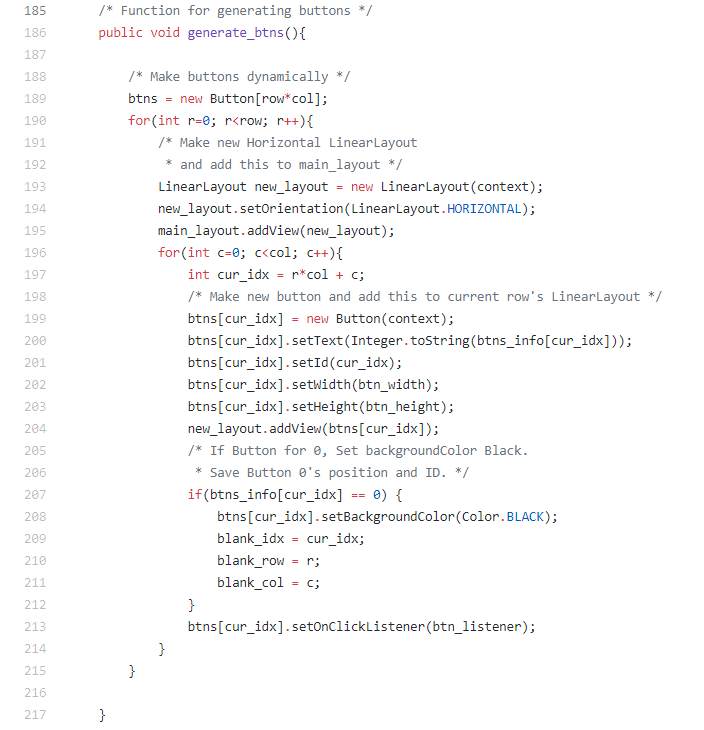
그리고 랜덤하게 버튼을 섞는 과정을 수행한다. 이 때, 항상 Puzzle이 풀리도록 완성된 형태에서 랜덤하게 버튼을 섞는다. 랜덤하게 섞는 과정에서 다시 정답인 형태로 버튼이 만들어질 수도 있다. 이러한 경우를 없애기 위해서 섞은 후에 정답 체크를 해서 만약 정답이면 다시 섞는다. 그리고 Dynamically 버튼을 생성하는 함수인 generate\_btns()를 호출한다.



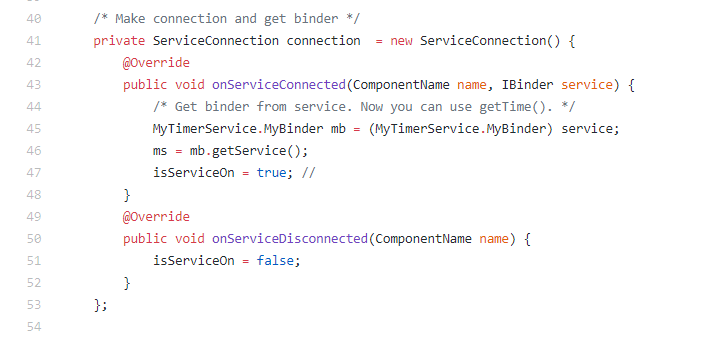


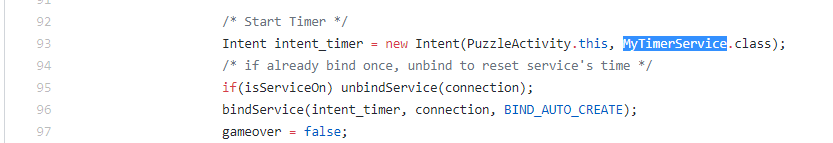
버튼을 생성할 때 미리 생성해둔 main\_layout안에 버튼을 생성한다. main\_layout은 Orientation이 VERTICAL로 설정하여 추가한 View가 수직으로 더해지게 된다. Grid 형태를 유지하기 위해서, LinearLayout을 HORIZONTAL로 설정하여 생성한 다음, 이 layout에 col개의 버튼을 생성하여 추가해준다. 이렇게 생성한 한 개의 row를 main\_layout에 추가함으로써 한 줄을 완성시킨다. 그리고 이 과정을 row번 만큼 반복하면 Grid형태의 버튼들이 만들어진다. 버튼의 id값이 0이면 빈 공간을 의미하므로, 배경을 검은색으로 칠해준다.



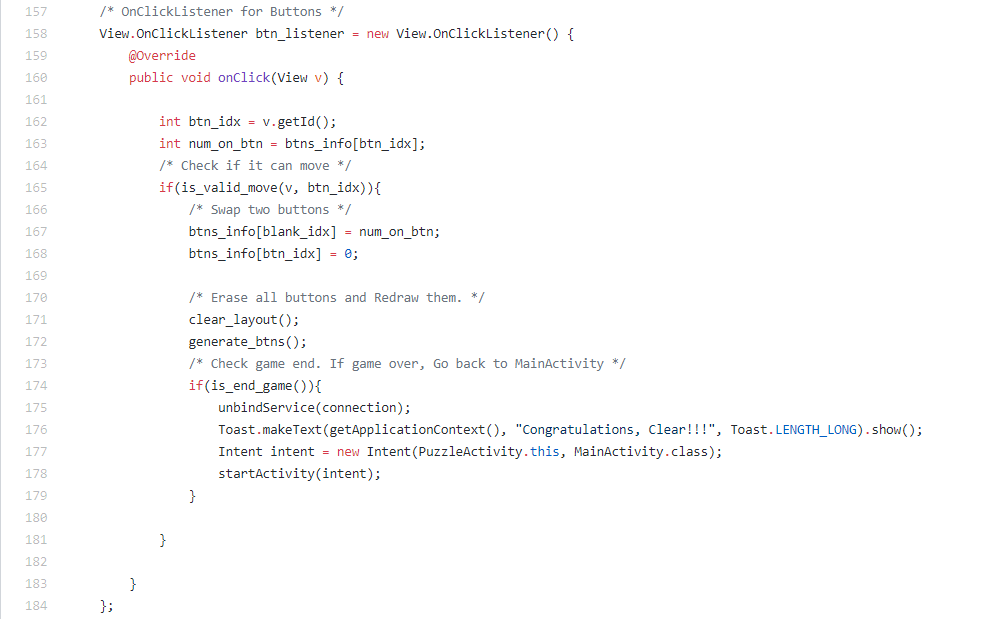


버튼을 생성한 이후 타이머 기능을 구현하기 위해서 MyTimerService class의 정보를 담고 있는 intent를 만든 후, 이 서비스를 bind한다. bind가 되었을 때, 호출되는 callback함수 ServiceConnection 객체를 만들어 두었다. 이 객체는 LocalBinder를 넘겨받고 service에 대한 레퍼런스를 얻는다. 이 과정을 통해서 PuzzleActivity에서 service에 있는 함수를 호출하여 time값을 가져올 수 있다.

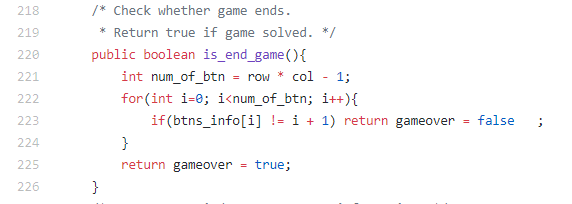




버튼이 생성되면 onClickListener를 만들어 빈 공간에 인접해 있는 버튼을 눌렀을 때 빈 공간으로 버튼이 이동되게 구현해야 한다. 눌린 버튼의 id를 이용해 눌린 버튼의 row, col을 알아낸 후, 미리 저장해 두었던 빈 공간의 row, col과 비교하여 인접해 있으면 두 버튼을 바꾼다. 이 때 실제로 버튼의 위치를 바꾸는 것이 아니라, btns\_info[] 배열에서만 바꾼다. 그리고 main\_layout에 있는 버튼들을 모두 지운 후, 다시 그려주는 작업을 한다.



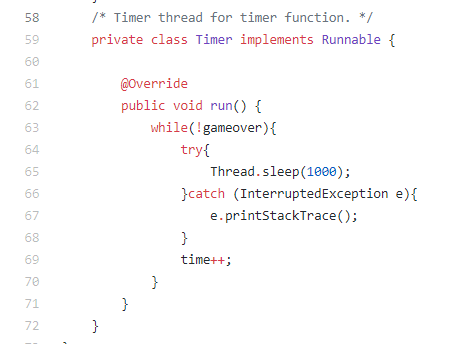
게임이 끝났는지는 버튼들의 정보가 담겨있는 btns\_info[]를 이용하여 쉽게 확인할 수 있다.



1. 추가 구현 (Android Service)

"Make Buttons" 버튼을 누른 후 실행되는 GetTimeThread class의 객체는 gameover Flag가 true가 될 때까지 계속 service의 getTime()함수를 호출하여 time\_sec값을 받아온다. 그리고 해당 time\_sec값을 MM:SS형식으로 바꾸어 textView의 객체인 clock에 출력해준다. MyTimeService에서 onCreate()함수 호출 시 시작되는 Timer 객체는 1000ms마다 sleep하며 시간을 잰다. GetTimeThread 객체는 service로부터 그 값을 100ms마다 읽어 오기 때문에 IPC의 오버헤드를 고려해도 0.1초의 오차만 허용한다.





1. **기타**

* **소감**

마지막 개인 프로젝트가 끝이 났다. 기존에 Kotlin을 배우기 위해서 Android Studio를 사용해 어플리케이션을 만들어 본 경험이 많은 도움이 되었다. 다행히 Target Board에 있는 LCD와 동일한 태블릿의 Android Virtual Device가 있어서 집에서 개발을 완료할 수 있었다. 실제 디바이스에서 실행했을 때, 문제가 생기지 않을까 걱정했지만 다행히 AVD와 동일하게 작동하였다.

프로젝트의 완성도를 높이기 위해서, Timer의 시간 출력도 눈에 확인하기 쉬운 부분에 위치시켰고, Puzzle도 아무렇게나 랜덤하게 숫자를 생성하는 것이 아닌, 정답이 될 수 있도록 완성된 형태에서 섞는 방법을 채택하였다. 이보다 더 중요한 팀 프로젝트가 남아있어서, 문제없이 빨리 이번 프로젝트를 완료하여 다행이었다.

[추가구현 – Service 구현 부분 Code]

<MyTimerService.java>

package com.example.puzzle;  
  
import android.app.Service;  
import android.content.Intent;  
import android.os.Binder;  
import android.os.IBinder;  
  
  
/\* Service for timer.  
 \* PuzzleActivity uses this Service and get time from binder.  
 \* \*/  
public class MyTimerService extends Service {  
  
 private int time = 0;  
 private boolean gameover = false;  
 private Thread timer;  
  
 public MyTimerService() {  
 }  
 IBinder mBinder = new MyBinder();  
  
 class MyBinder extends Binder {  
 MyTimerService getService(){  
 return MyTimerService.this;  
 }  
 }  
 int getTime(){  
 return time;  
 }  
 @Override  
 public IBinder onBind(Intent intent) {  
 System.*out*.println("onBind");  
 time = 0;  
 gameover = false;  
 return mBinder; // binder  
 }  
  
 @Override  
 public void onCreate(){  
 System.*out*.println("onCreate");  
 super.onCreate();  
 timer = new Thread(new Timer()); /\* Start timer thread. \*/  
 timer.start();  
 }  
  
 @Override  
 public void onDestroy(){  
 System.*out*.println("onDestroy");  
 super.onDestroy();  
 }  
  
 @Override  
 public boolean onUnbind(Intent intent) {  
 System.*out*.println("onUnbind");  
 gameover = true;  
 return super.onUnbind(intent);  
 }  
 /\* Timer thread for timer function. \*/  
 private class Timer implements Runnable {  
  
 @Override  
 public void run() {  
 while(!gameover){  
 try{  
 Thread.*sleep*(1000);  
 }catch (InterruptedException e){  
 e.printStackTrace();  
 }  
 time++;  
 }  
 }  
 }  
}

<PuzzleActivity 중, Service 관련 코드>

MyTimerService ms;  
boolean isServiceOn = false;  
  
/\* Make connection and get binder \*/  
private ServiceConnection connection = new ServiceConnection() {  
 @Override  
 public void onServiceConnected(ComponentName name, IBinder service) {  
 /\* Get binder from service. Now you can use getTime(). \*/  
 MyTimerService.MyBinder mb = (MyTimerService.MyBinder) service;  
 ms = mb.getService();  
 isServiceOn = true; //  
 }  
 @Override  
 public void onServiceDisconnected(ComponentName name) {  
 isServiceOn = false;  
 }  
};

/\* onClickListener for make\_btn. ("Make Buttons") \*/  
View.OnClickListener listener = new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
  
 /\* Clear all views(buttons) \*/  
 clear\_layout();  
 try{  
 /\* Parse input data \*/  
 splitted = data.getText().toString().trim().split("\\s+");  
 if(splitted.length > 2) throw new Exception();  
 row = Integer.*parseInt*(splitted[0]);  
 col = Integer.*parseInt*(splitted[1]);  
 if(row == 0 || col == 0 || row > 5 || col > 5 || (col == 1 && row == 1)) throw new Exception();  
  
 /\* Calculate buttons' size \*/  
 btn\_width = calc\_btn\_width();  
 btn\_height = calc\_btn\_height();  
  
 do{  
 /\* Generate random buttons' order.  
 \* Repeat this until initial state is not solved.\*/  
 generate\_btn\_order();  
 }while(is\_end\_game());  
 generate\_btns(); /\* Make buttons dynamically \*/  
  
 /\* Start Timer \*/  
 Intent intent\_timer = new Intent(PuzzleActivity.this, MyTimerService.class);  
 /\* if already bind once, unbind to reset service's time \*/  
 if(isServiceOn) unbindService(connection);  
 bindService(intent\_timer, connection, *BIND\_AUTO\_CREATE*);  
 gameover = false;  
 new Thread(new GetTimeThread()).start();  
  
 } catch (Exception e){  
 /\* If splitted data's length > 2, or row == 0, or col == 0 \*/  
 Toast.*makeText*(getApplicationContext(), "Invalid input data [row(1~5) col(1~5)]", Toast.*LENGTH\_LONG*).show();  
 if(isServiceOn) {  
 unbindService(connection); /\* Disconnect binder \*/  
 isServiceOn = false;  
 }  
 gameover = true; // To finish GetTimeThread  
 clock.setText("00:00"); // Init clock TextView  
 System.*out*.println("Make\_btn listener Exception occurred! : " + e);  
 }  
 }  
};  
make\_btn.setOnClickListener(listener);

private class GetTimeThread implements Runnable {  
 int time\_sec;  
 /\* Need handler to get time from binder. \*/  
 Handler handler = new Handler();  
 @Override  
 public void run() {  
 while(!gameover){  
 if(!isServiceOn) continue; /\* If disconnected. \*/  
 handler.post(new Runnable() {  
 @Override  
 public void run() {  
 try{  
 /\* Get time from binder.  
 \* Set MM:SS data in clock textView \*/  
 time\_sec = ms.getTime();  
 System.*out*.println("time\_sec :" + time\_sec);  
 String str = String.*format*("%02d:%02d", time\_sec/60, time\_sec%60);  
 clock.setText(str);  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 });  
 try{ /\* Check interval \*/  
 Thread.*sleep*(100);  
 } catch (InterruptedException e){  
 e.printStackTrace();  
 }  
 }  
 }  
}