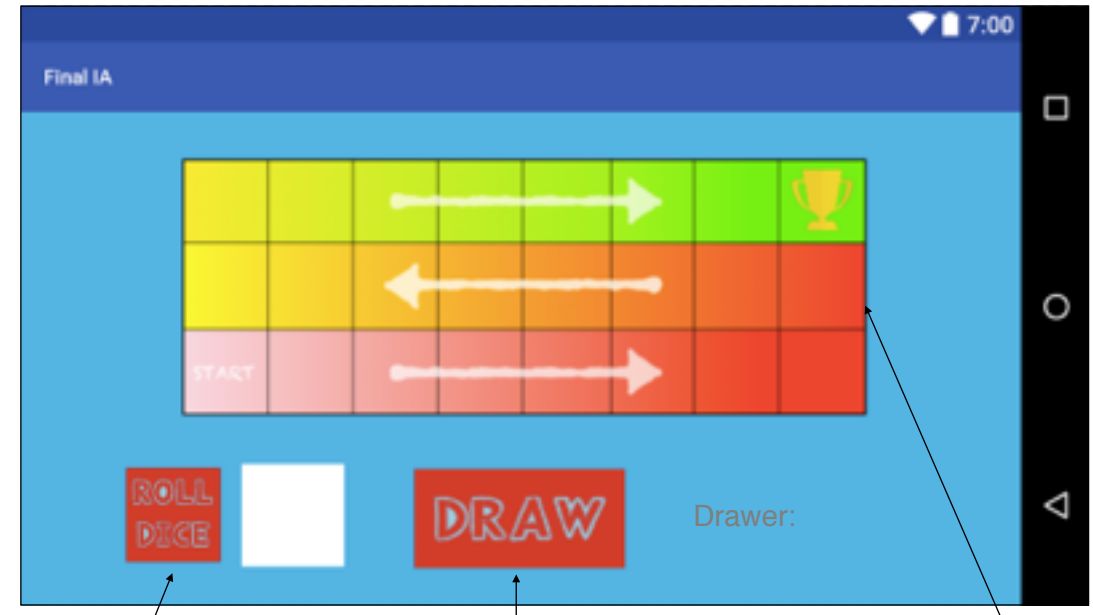
Criteria C – Development

Techniques

- Graphical User Interface
- Encapsulation
- Polymorphism
- Complex Data Structures
- Use of Timer
- Online Database
    - Writing queries in PHP
    - Communicating between PHP and Java files
- Modularity
    - Java Classes
    - PHP Files

Graphical User Interface

For the program to be easy to use and appealing, I used a graphical user interface. Making the application of android studio allowed me to accurately produce screens, as designed in the previous development stage, with ease and efficiency. Because of the ability to make the design as planned, the buttons are accessible and software is intuition to navigate through.



Button to roll dice

Button to begin drawing and guessing

Board to show team progress

Encapsulation

Encapsulation was used several times throughout the application to ensure that the value of a certain variable could be accessed by multiple classes without running the chance of changing the variable's value.

The two most significant examples in this program are (i) the encapsulation of the game pin, because it is vital that this value is never accidentally changed while the app is running, else all database table connections will not work, and (ii) the String holding username because the program will not be able to work if the device cannot identify the specific user.

```
public static String getGamePin(){
    return gamePin;
}
```

Function to return game pin

```
public String getName(){
    return name;
}
```

Function to return name of player

Polymorphism

Polymorphism is when multiple methods have the same name and only one of them is called depending on the situation it is called in. I used dynamic polymorphism because of the classes I was inheriting from.

Whenever I would make a new activity, launch a new screen, it was inherited from the pre-existing "AppCompatActivity" class. This activity had some methods I needed to change at run time, so I had to override them.

I had to override all the "onCreate" methods which were the constructors for each activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

Need to call super class while overriding default constructor

In some classes, I also had to override the onStart or onResume methods as shown in Fig. _.

```
@Override
protected void onStart(){
    super.onStart();
```

Need to call super class while overriding class that defines what happens after starting

I also had to use dynamic polymorphism in classes that helped me request connections to the database because they inherited from the "StringRequest" class. In all these classes I had to override "Map", from where the PHP file gets its values from.

```
@Override
public Map<String, String> getParams() {
    return params;
}
```

Returns all the data in the "Map" so that it can be used by the PHP file

Complex Data Structures

Different data structures were needed for storing data in the application. The most common was normal arrays to store things like the list of all the players in the game, but the more complex data structures used were stacks and array lists.

Array lists had to be used in the “canvasView” class to know the colour, opacity and thickness of every stroke drawn to satisfy criteria 10. I originally had a bit of trouble with changing these three properties for each line drawn because if they were changed for the “path” object, all the lines in the object would change. The array list works by storing the values for each line in the object so that when it is drawn, each line has its own set of values.

```
public static ArrayList<Path> paths = new ArrayList<>();
private static ArrayList<Integer> colours = new ArrayList<>();
private static ArrayList<Integer> thickness = new ArrayList<>();
```

Array lists to store essential info for each path

```
colours.remove(colours.size()-1);
colours.add(properties.colour);
colours.add(properties.colour);

thickness.remove(thickness.size()-1);
thickness.add(properties.thick);
thickness.add(properties.thick);
```

Need to do this so that the colour and thickness changes immediately, otherwise one more line needs to be drawn

Stacks had to be used for the undo and redo buttons. Anytime the undo button is pressed, the last values from the array lists discussed above, would be put on the stacks, and if the redo button is pressed, the value is popped from the stacks and put back in the array lists.

```
public static Stack<Path> rPaths = new Stack<>();
private static Stack<Integer> rColours = new Stack<>();
private static Stack<Integer> rThickness = new Stack<>();
```

```
rPaths.add(paths.get(paths.size()-1));
rColours.add(colours.get(colours.size()-1));
rThickness.add(thickness.get(thickness.size()-1));

paths.remove(paths.size() - 1);
colours.remove(colours.size() - 1);
thickness.remove(thickness.size() - 1);
```

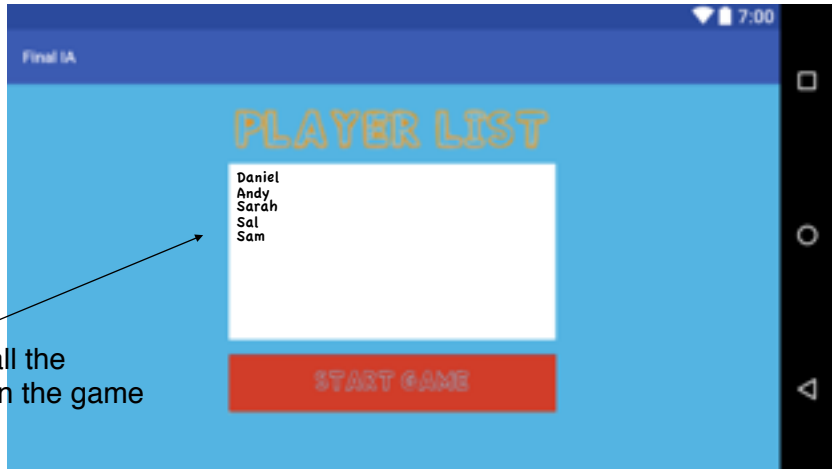
Removing information from the array lists and putting it in the stacks when the undo button is pressed

```
paths.add(rPaths.pop());
colours.add(rColours.pop());
thickness.add(rThickness.pop());
invalidate();
properties.change = true;
```

Removing information from he stack and putting it in the array lists when the redo button is pressed

Use of Timer

Timers are used in multiple classes. One of the uses of the timer is to keep refreshing the player list to show any new player that joins the game after the activity is created. The timer is used in the same way for drawing and making guesses, but the request made to the database is different.



Display showing all the players currently in the game

```
CountDownTimer countDownTimer = new CountDownTimer(1000 * 1000, 1000) {
    @Override
    public void onTick(long millisUntilFinished) {
        Response.Listener<String> responseListener = new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                String success = response;
                list.setText(success);
            }
        };
        showNameRequest join = new showNameRequest(GamePinEnter.getGamePin(), responseListener);
        RequestQueue queue = Volley.newRequestQueue(PrivateGameHost.this);
        queue.add(join);

    }

    @Override
    public void onFinish() {

    }
};
countDownTimer.start();
```

Countdown timer for 1000 seconds

Refreshing the list every second by making a request to the database for all the names in the table

The other implementation of the timer was to meet success criteria 13 and it was in the drawer and guesser classes so that each round of drawing and guessing lasted only 90 seconds

```
public void timer(){
    CountDownTimer countDownTimer = new CountDownTimer(90 * 1000, 1000) {
        @Override
        public void onTick(long millisUntilFinished) {
            timer.setText(String.valueOf((int)millisUntilFinished/1000));
        }

        @Override
        public void onFinish() {
            canvasView.cleanCanvas();
            finish();
        }
    };
    countDownTimer.start();
}
```

Countdown timer for 90 seconds

Displaying seconds remaining

End activity at end of timer

Online Database

The database was essential to this program because it is the only way for multiple devices to connect to the same game. I used a mysql database for this which uses “PHPMyAdmin”.

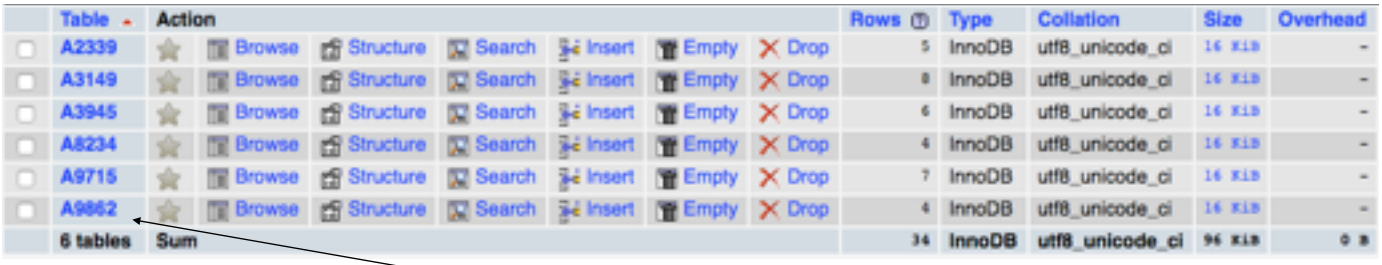


	Table	Action						Rows	Type	Collation	Size	Overhead
☐	A2339	☆	Browse	Structure	Search	Insert	Empty	Drop	5	InnoDB	utf8_unicode_ci	16 KiB	-
☐	A3149	☆	Browse	Structure	Search	Insert	Empty	Drop	8	InnoDB	utf8_unicode_ci	16 KiB	-
☐	A3945	☆	Browse	Structure	Search	Insert	Empty	Drop	6	InnoDB	utf8_unicode_ci	16 KiB	-
☐	A8234	☆	Browse	Structure	Search	Insert	Empty	Drop	4	InnoDB	utf8_unicode_ci	16 KiB	-
☐	A9715	☆	Browse	Structure	Search	Insert	Empty	Drop	7	InnoDB	utf8_unicode_ci	16 KiB	-
☐	A9862	☆	Browse	Structure	Search	Insert	Empty	Drop	4	InnoDB	utf8_unicode_ci	16 KiB	-
	6 tables	Sum							34	InnoDB	utf8_unicode_ci	96 KiB	0 B

All the active tables in the database

PHP Queries

Queries to the database were all made in PHP files. This required me to learn how to write PHP.

```php
<?php
    $con = mysqli_connect("localhost", "id3764634_sehejoberoi10", "abcd1234", "id3764634_csia");
    // Check connection
    if (mysqli_connect_errno()){
        echo "Failed to connect to MySQL: " . mysqli_connect_error();
    }

    $username = $_POST["username"];
    $table = $_POST["table"];

    mysqli_query($con, "INSERT INTO $table (username, team, start) VALUES ('$username','0','0')");
    echo(1);
    mysqli_close($con);
?>
```

Database login details

Received data

Query to add record to table with username as received

Returning value to java file to say that the job is done

```php
<?php
    $con = mysqli_connect("localhost", "id3764634_sehejoberoi10", "abcd1234", "id3764634_csia");
    // Check connection
    if (mysqli_connect_errno()){
        echo "Failed to connect to MySQL: " . mysqli_connect_error();
    }

    $pin = $_POST["pin"];

    $sql="SELECT `username`,`team` FROM `$pin`";
    $result=mysqli_query($con,$sql);

    while(list($username, $team)=$result->fetch_row()){
        echo "$username $team\n";
    }
    mysqli_close($con);
?>
```

Query to select username and team data from table defined by name in the variable $pin

Loop to echo/send all the data requested back to the java class

Commutation between Java and PHP

To read and write between the java files in the program and the PHP files saved on the online server, I needed to do a few things.

First, I had to use inheritance from the “StringRequest” class so that I can make a request to the internet.

```
public class privateGameJoinRequest extends StringRequest{
    private static final String Request_Join_URL="https://socsia.000webhostapp.com/privateGameGuest.php";

    private Map<String, String> params;

    public privateGameJoinRequest(String pin, Response.Listener<String> listener){
        super(Request.Method.POST, Request_Join_URL, listener, null);
        params=new HashMap<>();
        params.put("pin",pin);

    }

    @Override
    public Map<String, String> getParams() {
        return params;
    }
}
```

PHP file URL

Constructor for the object of type “privateGameJoin Request that needs the pin to be passed as a parameter

Map that stores all the data that is put into “params”

After this, I had to alter the default application manifest code to give the application internet access so that the requests could work.

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

permission to use the internet form the application

Finally I had to create the object from the aforementioned class and use the built-in “volley” library to communicate the data from the “Map” in the request class to the PHP query file.

```
Response.Listener<String> responseListener = (response) → {
            String success = response;
            if (success.equals("1")) {
                Intent enter;
                if (PrivateGameChoose.getIsHost()){
                    enter = new Intent (PrivateGameName.this, PrivateGameHost.class);
                }else{
                    enter = new Intent (PrivateGameName.this, PrivateGameGuest.class);
                }
                startActivity(enter);
            } else {
                AlertDialog.Builder builder = new AlertDialog.Builder(PrivateGameName.this);
                builder.setMessage("Login Failed. (Likely - Username is taken)")
                        .setNegativeButton("Retry", null)
                        .create()
                        .show();
            }
};
setName join = new setName(name, responseListener);
RequestQueue queue = Volley.newRequestQueue(PrivateGameName.this);
queue.add(join);
```

The response listener to get data and know when process is done and the task can continue

Alert dialogue box for any alert that may occur, this one is for not unique username

Uses the “volley” class to make an object and put it in a request queue so that the PHP file can be used for its queries

Modularity

Although all the files and classes in this program, have been designed for this application, a lot of the code can be slightly modified and reused in other applications.

Modularity of Java Classes

1) All the classes have response listeners that use an input, like a button click, to launch a new intent (The next screen in the GUI). The code from these listeners can be used in different classes by just changing the name of the new and current intent.

```
public void PrivateGame(View v) {
    Intent pGame = new Intent(this, PrivateGameChoose.class);
    startActivity(pGame);
}
```

By changing this class name, the code can be used for other things

2) The request classes as discussed in the database section are easy to reuse because all data has been stored in a variable rather than typing it everywhere it is needed. The reason it is so modular is because the only things that would need to be changed are the URL for the PHP file and data put in the "Map" that are actually passed as a parameter of the object.

```
public class privateGameJoinRequest extends StringRequest{
    private static final String Request_Join_URL="https://socsia.000webhostapp.com/privateGameGuest.php";

    private Map<String, String> params;

    public privateGameJoinRequest(String pin, Response.Listener<String> listener){
        super(Request.Method.POST, Request_Join_URL, listener, null);
        params=new HashMap<>();
        params.put("pin",pin);

    }

    @Override
    public Map<String, String> getParams() {
        return params;
    }
}
```

Changing Url and parameters are all that are needed to be modular

Modularity of PHP Files

All the PHP files use the data passed into them when using the "Volley" library and then keep the values as variables as much as possible. This makes all the queries in them modular because the only things that need to be changed to reuse the file is the login details for a database and variables being passed into the file

```
<?php
    $con = mysqli_connect("localhost", "id3764634_sehejoberoi10", "abcd1234", "id3764634_csia");
    // Check connection
    if (mysqli_connect_errno()){
        echo "Failed to connect to MySQL: " . mysqli_connect_error();
    }

    $username = $_POST["username"];
    $table = $_POST["table"];

    mysqli_query($con, "INSERT INTO $table (username, team, start) VALUES ('$username','0','0')");
    echo(1);
    mysqli_close($con);
?>
```

In the query, no real data is referred to, only the variables, which is changed in the java class, hence the code is modular