# Floating Polygons

## Crowdsourcing touch screen inputs for a contextual data model

Kasun Buddhika Dissanayake
Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka
kasundissanayake.12@cse.mrt.ac.lk

***Abstract*— Since the first iPhone introduced in 2007, smart phones have being changing the way we communicate, work, shop, and more. Today there are billions of apps and millions of app developers. The UX design of these apps should be carefully designed in order to attract the targeted audience. The purpose of this project is to create a data model about the user interactions with the touch screen which can be referred when designing new apps and games. The Android game "Floating polygons" was developed as a demonstration and it captures all the data about user interactions with the touch screen while it is being played. In addition, an online data model is being created with data of the game which can be later referred by the developers when they need. This data model is categorized to three gesture types, three screen sizes and two screen orientations.**

***Keywords*— *Touch screen; Crowdsource; Android app; Touch inputs***

## I. Introduction

The number of mobile applications and games in Apple App Store and Google play is increasing exponentially day by day. There are billions of apps and millions of app developers. Touch screens have become the main input device for these applications. Users can interact with the app by various gestures such as translating, pinching and rotating. The developers have to more careful when it is come to the UX or user experience engineering because if it fails, the whole system will be ineffectual.

By the end of March in 2015, there were nearly 600 thousands of mobile apps which had been rejected by the users in the Google play [1]. According to the experts of the field the poor quality and the poor UX designing are the major reason to these failures. The apps and games would be more user friendly if there exists a data model to refer when UX engineering.

The proposed project is to develop an Android game which is capable of fetching categorized data about the user interactions with the touch screen and dispatch them to an online server. The data which are being uploaded to the server will be the screen size and orientation of the device and arrays of touch counters with respect to the different gestures. Since all the data have been categorized according to the screen size and screen orientation a complete data model can be created and the developers can refer that model when they are taking UX related decisions.

The implemented solution was "Floating Polygons" which is an Android game where players have to use all different type of gestures (i.e. translate, pinch and rotate) to continue with the game. The game is not manipulating player to go to specific positions in the screen. Player can pan and zoom the camera by translating and pinching while he can use rotate gesture to rotate game objects. For an example, if one player is more comfortable with bottom-right side of the screen nothing prevents him to stay at the bottom-right. If game objects are trying to go away from the bottom-right he can easily pan the camera again to the bottom-right. In that way we can find out what are the most comfortable areas of a touch screen to the user.

The paper is structured as follows: Section II describes the related literature and Section III explains the system model including requirements and design aspects of the Android app. The implementation details and the application GUI are presented in Section IV. Section V specifies the testing and analysis of the system and Section VI concludes the paper.

## II. Literature Review

Researches focusing on effective and efficient input with the touchscreen on mobile devices have always been important after it began to increase of the usage of mobile devices exponentially. Today researches related to touch input analysis are going in mainly three areas. Those are describing and understanding touch input, HCI input analysis and crowdsourcing user studies while this research project is focused on HCI input analysis and Crowdsourcing [2].

During recent past, crowdsourcing has become popular among the researchers. There are games with a purpose of studying user behaviors, as presented by Law et al. [3]. They present the concept of using games that rely on input agreement to collect labeled data and same technique is used in this project. While these platforms were first mainly used for a lot of human computation tasks, recently some research groups have started utilizing the potential of crowdsourcing HCI user studies [4, 5].

Recently Niels Henze et al. present a paper with the analysis of 100,000,000 taps on the touch screen [6]. The Android game they presented is "Hit it". Their intention was find error rate when touching the screen. "Hit it" can help developers to understand errors of user inputs. A self-learning keypad app is an example app for dealing with user input errors. Developers can rely on the results of the above mentioned research since it has an analysis of 100,000,000 user inputs. The same method, crowdsourcing, is used in "Floating polygons" project in order to get a reliable analysis of user inputs.

Tao Feng et al. have published a data model about gestures in an uncontrolled environment [7]. They have created a model with different users on how their touch movement went when they using the launcher. This enables developers to get an abstract idea about the user interactions with the touchscreen. Hence developers were able to decide the majority of users from left handed and right handed users which will helps to develop a better UX design.

But the mobile app development industry travelled so far and it now needs more parameters to decide UX design. It cannot survive with only an abstract view of user interactions. The data should be more specific and categorized such as screen sizes, screen orientation, country and localization. Currently developers are getting these data from doing experiments with their own apps. If a UX designer cannot decide where a button should be placed on left side or right side, he releases two versions of app with the integration of an analytic tool and comes to a final decision with the results. Since the above method is a slow process to find out the required data, "Floating Polygons" will be carried out as a solution by creating a data model which has been categorized into different types of gestures, screen sizes and screen orientation with crowdsourcing.

## III. System Models

### A. *System Requirement*

System was developed in order to fulfill four main functional requirements. Developing a playable android game which uses different types of gestures to control the game. Capturing user gestures details accurately is a functional requirement since the data model relies on the user data. The captured data must be dispatched to the remote server without disturbing the player. The data is needed to be stored locally before dispatching them to server because frequent connections with the server will increase the server traffic and decrease the performance of the game.

Non-functional requirements are challenging game levels and attractive graphics in order to get a maximum audience. Apart from that frame rate of the game should be at a higher value because users do not want to experience lags in the game. Heat maps are used to represent the data model as it helps to analyze the data productively.

As shown in the usecase given in Fig. 1, **Game manager** is responsible for activities and processes in the game. It initiates the game when user wants to play the game. If the level is a bonus level, **Game manager** will organize the game board according to the bonus. **Game manager** can enable or disable the input handler. Disabling input handler should be done when player wants to pause the game. **Game manger** can pass a message to the data fetcher to dispatch data to the server.

**Input handler** can capture user gestures and control game objects according to the gestures. It can send captured gestures to the data fetcher in order to store them a local file until it is being dispatched to the server.

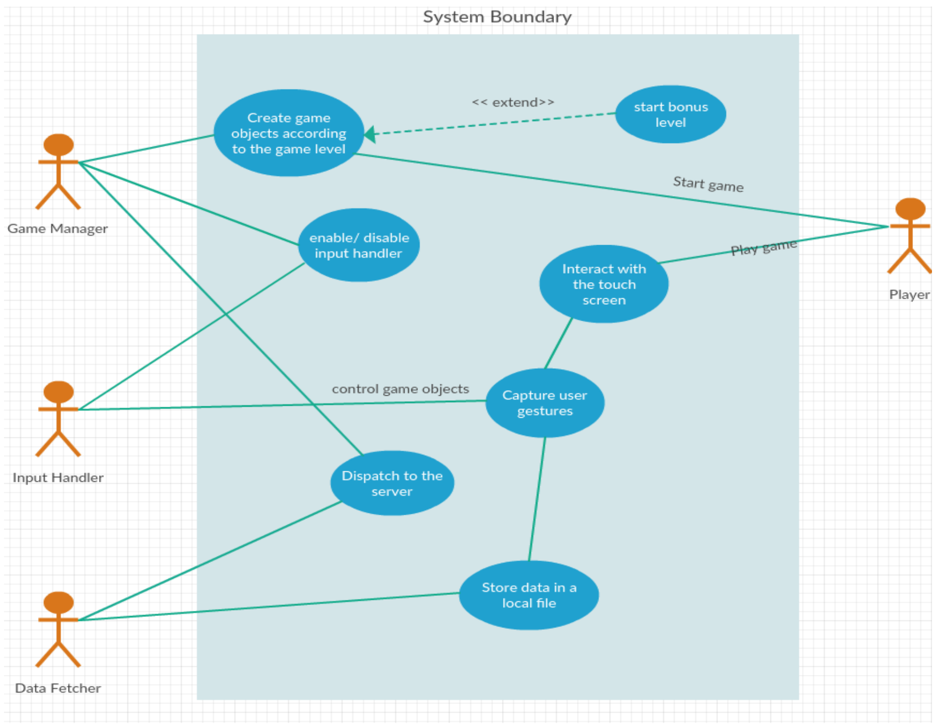
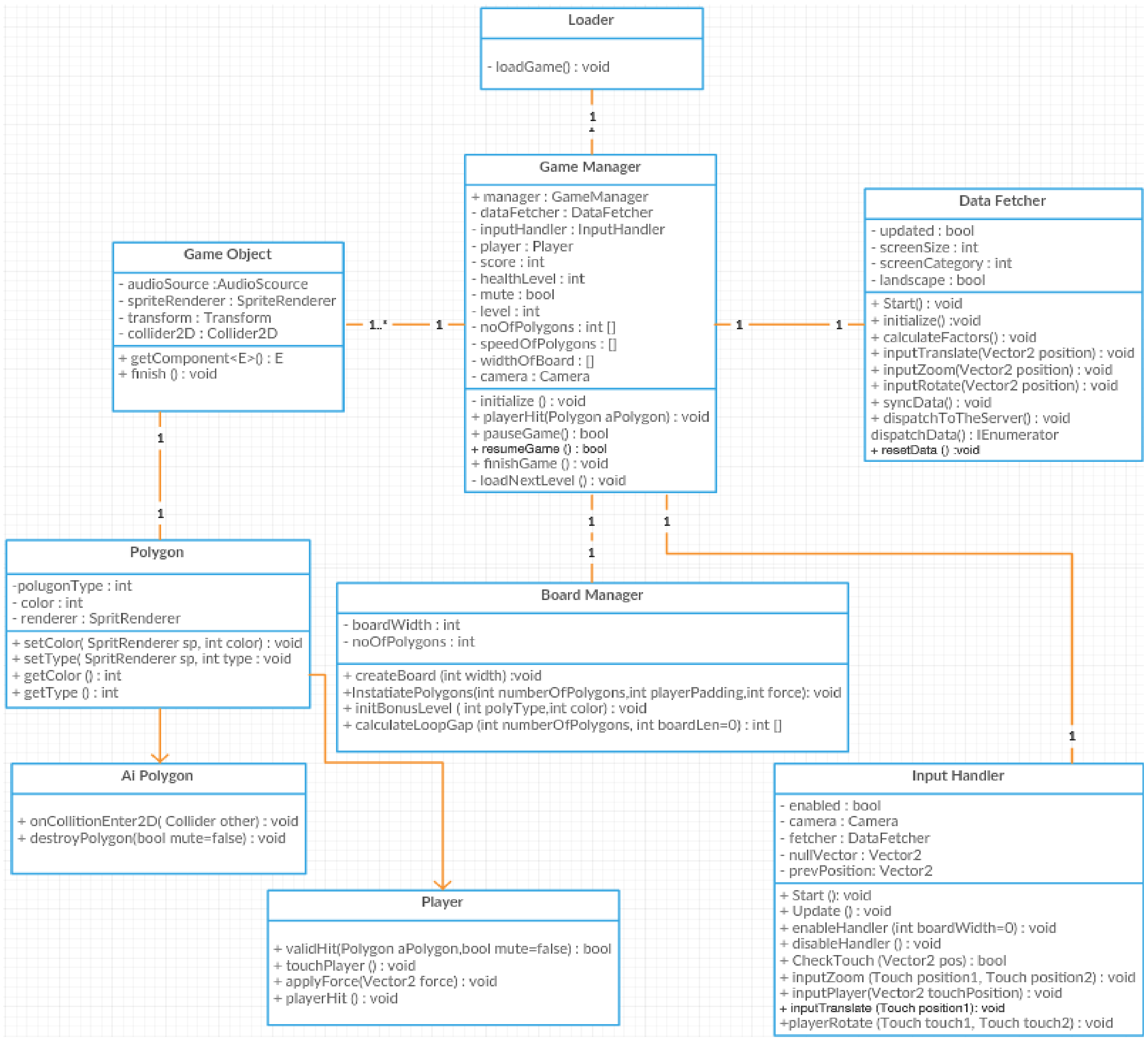


***Fig. 4. Use Cases of the System***



Loader
- loadGame() : void
Game Manager
+ manager : GameManager
- dataFetcher : DataFetcher
- inputHandler : InputHandler
- player : Player
- score : int
- healthLevel : int
- mute : bool
- level : int
- noOfPolygons : int []
- speedOfPolygons : []
- widthOfBoard : []
- camera : Camera
- initialize () : void
+ playerHit(Polygon aPolygon) : void
+ pauseGame() : bool
+ resumeGame () : bool
+ finishGame () : void
- loadNextLevel () : void
Data Fetcher
- updated : bool
- screenSize : int
- screenCategory : int
- landscape : bool
+ Start() : void
+ initialize() :void
+ calculateFactors() : void
+ inputTranslate(Vector2 position) : void
+ inputZoom(Vector2 position) : void
+ inputRotate(Vector2 position) : void
+ syncData() : void
+ dispatchToTheServer() : void
dispatchData() : IEnumerator
+ resetData () :void
Game Object
- audioSource :AudioScource
- spriteRenderer : SpriteRenderer
- transform : Transform
- collider2D : Collider2D
+ getComponent<E>() : E
+ finish () : void
Polygon
-polugonType : int
- color : int
- renderer : SpritRenderer
+ setColor( SpritRenderer sp, int color) : void
+ setType( SpritRenderer sp, int type : void
+ getColor () : int
+ getType () : int
Board Manager
- boardWidth : int
- noOfPolygons : int
+ createBoard (int width) :void
+InstatiatePolygons(int numberOfPolygons,int playerPadding,int force): void
+ initBonusLevel ( int polyType,int color) : void
+ calculateLoopGap (int numberOfPolygons, int boardLen=0) : int []
Ai Polygon
+ onCollitionEnter2D( Collider other) : void
+ destroyPolygon(bool mute=false) : void
Player
+ validHit(Polygon aPolygon,bool mute=false) : bool
+ touchPlayer () : void
+ applyForce(Vector2 force) : void
+ playerHit () : void
Input Handler
- enabled : bool
- camera : Camera
- fetcher : DataFetcher
- nullVector : Vector2
- prevPosition: Vector2
+ Start (): void
+ Update () : void
+ enableHandler (int boardWidth=0) : void
+ disableHandler () : void
+ CheckTouch (Vector2 pos) : bool
+ inputZoom (Touch position1, Touch position2) : void
+ inputPlayer(Vector2 touchPosition) : void
+ inputTranslate (Touch position1): void
+playerRotate (Touch touch1, Touch touch2) : void
1
1
1..*
1
1
1
1
1
1
1
1
1

Fig. 5. Class diagram of the system

## B. *System Design*

The system is designed according to the MVC architecture. The model component of the system is the **Game manager** which is responsible for managing resources and the Controller component is the **Input Handler** which controls the game objects according to the input gestures. View component is the Unity Renderer which renders the objects in the game.

Fig. 2, shows the class diagram of the system. The **Loader** class is only for loading the **Game manager** and the **Game manger** is managing all the other resources. It enables the **Input handler** after the game has started. This procedure assures the user inputs are captured only when the game is being played. Therefore it does not take any user input when user is in the game menu. **Game manager** sends a message to the data fetcher to dispatch data after the game ends.

**Polygons** are the main game objects in the game. There are three types of polygons with four colors. Whenever a **polygon** hit by the player it checks the validity of the hit via the game manager. If it is a valid hit **game manage**r updates the score and destroy the **polygon**. If not **game manager** decrease the health level. If health level is zero, the game should be finished. **Ai Polygons** has a method called **onCollitionEnter2d** which will be executed whenever it hit with another object.

Player has method to check a hit is valid or not. A hit should be valid only if player's polygon type is equal to other the polygon or the color of the player is equal to the other polygon. If it is a valid hit player' will transform its color and polygon type to the properties of the polygon which hit player.

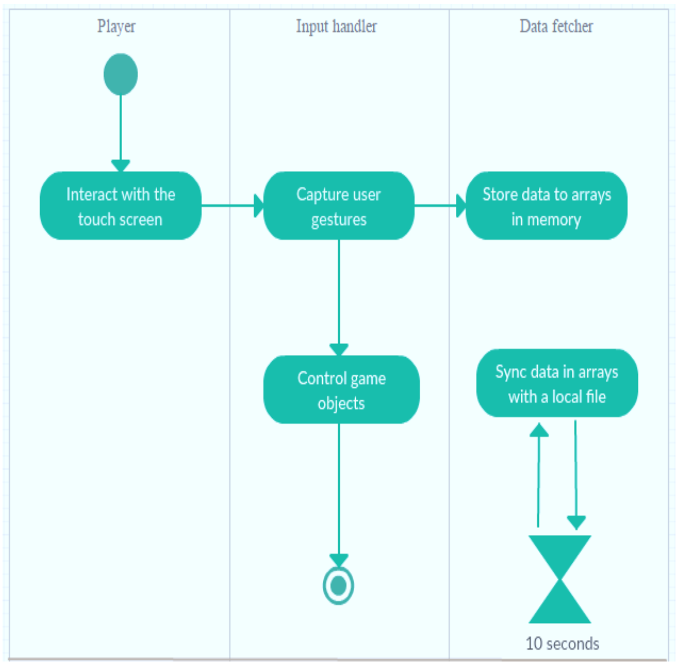

Player
Input handler
Data fetcher
Interact with the touch screen
Capture user gestures
Store data to arrays in memory
Control game objects
Sync data in arrays with a local file
10 seconds

*Fig. 6. Main activity diagram of the system*

Main activity of the system is capturing user gestures and the diagram is shown in Fig. 3. **Input handler** controls the game objects after it identifies the gesture and it sends a message to the **Data Fetcher** about the gesture. The data fetcher stores them in an array in the memory. Data fetcher has a method to sync the data in the RAM and the data file which is stored in the local storage. This method will run once in 10 seconds. This strategy is used to increase performance by not writing the file every time and store all the data by writing them once in 10 seconds.

## C. *Database Design*

An entity relationship diagram is not available since there are no relationships between entities. Database is only used for store the touch counts with respect to the gesture and orientation. There are 6 tables named according the orientation of the screen and the gesture (i.e. landscape_translate, landscape_zoom).

Each table has 25 columns. 24 of them are big integers while other is a tiny integer. Tiny integer is for three categories of the screen. Big integers are used in order to avoid integer overflow. Each touch point is transformed in to 4*6 grid in the **Data Fetcher**. Therefore it only needs to manage a grid of 24 for one table. The reason behind transforming a grid is reduce the bandwidth at the server and the database. Network traffic and bandwidth must be controlled because crowdsourcing deals with large number of data.

The script at the server increments the counts of the database according to the category when it receives data from a mobile device. Category 0 is for screen sizes lower than 5, category 1 for screens between 5 and 10 and 2 is for large screens. This enables to select

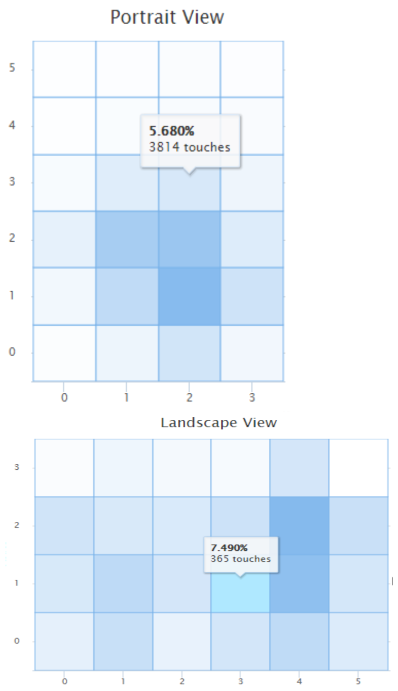


*Fig. 7. Data model*

data from the database using short and simple queries.

Example to select zoom gesture data of mobile phones in landscape orientation.

Table: landscape_zoom category: 0

```
SELECT * FROM `landscape_zoom` WHERE `cat` =0
```

## IV. System Implementation

## A. *Implementation Procedure*

The game was developed with unity3D [8] and Unity Test were used for testing. PHP scripts are used to upload data to the database and to show data from the database. Any additional material have not used when implementing this system. GitHub was used as a Version Control System.

Gesture controlling is an essential part of the system because it controls the game objects while capturing gestures to store gesture data. The **update** method is called once per frame by the unity engine. This method can be used to update data of the variables. The update method has been used to get touch count and identify the gesture by analyzing sequence of touch data.

It looks for the number of touch points and detect the gesture because translate gesture needs only one touch and the previous touch position. But if it finds two touch points, it can fetch the gesture by previous touch point and the delta position of the current touch point. Rotation and pinch gestures need two touches. Delta position which is an attribute of **Touch** class in Unity 3D game engine is used to detect the direction of the touch movement.

## B. *Main Interfaces*

The Fig. 5, is the main menu of the game. Player can start the game, view high scores, mute the music or can exit from the game with these buttons. The Fig. 6, represents the gameplay and game ending iterface where user can restart or exit from the game.

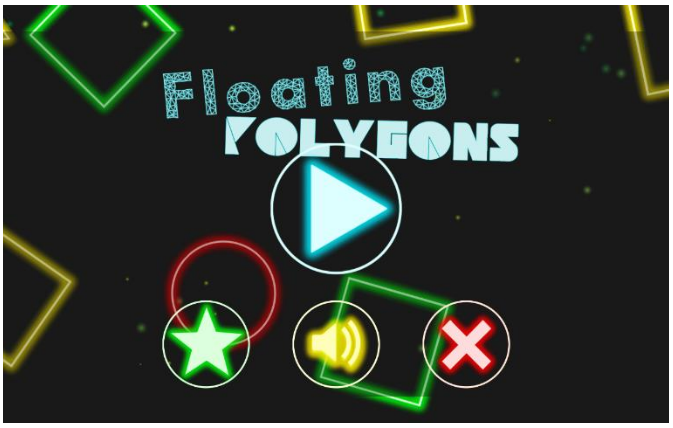


*Fig. 8. Main menu of the game*

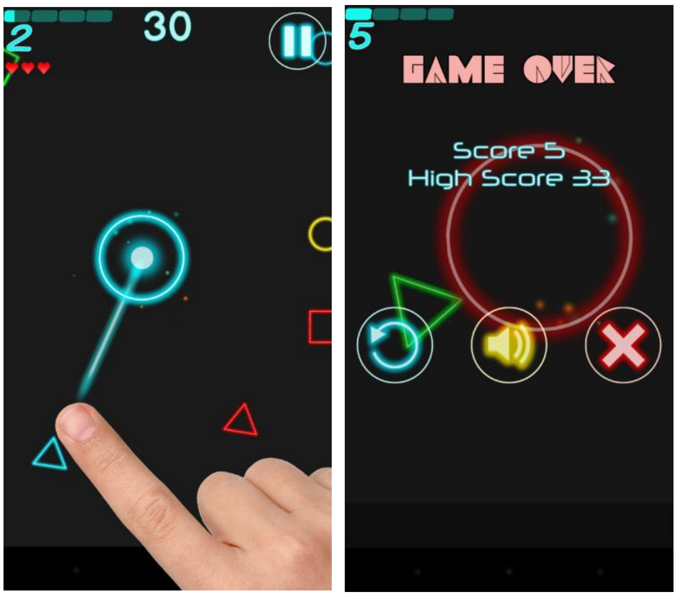
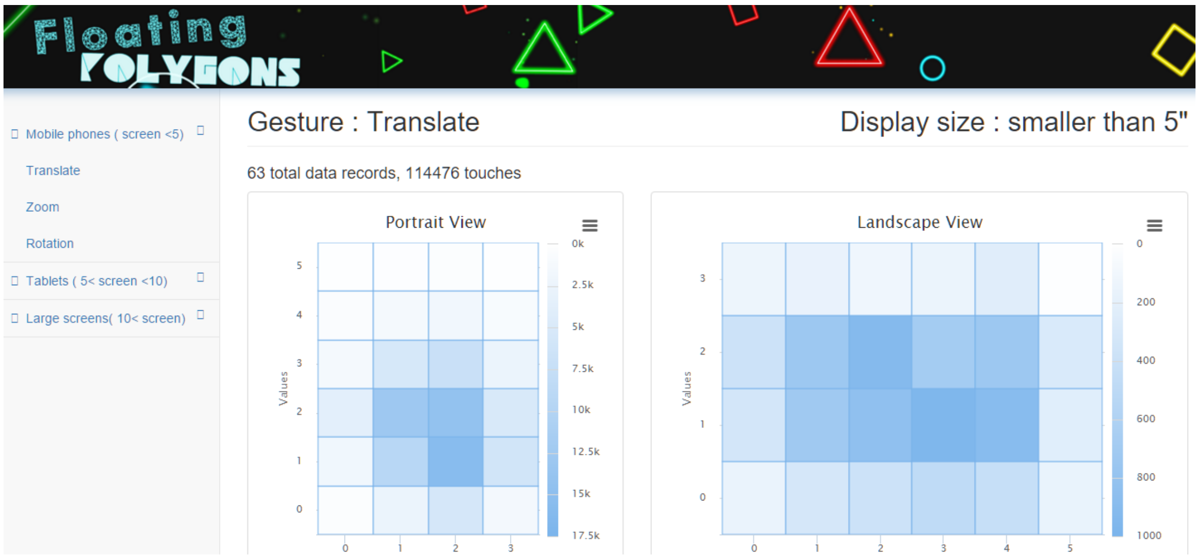


*Fig. 9.:Game play*



Gesture : Translate

Display size : smaller than 5"

63 total data records, 114476 touches

*Fig. 10.. Interface of the data model*

In gameplay, a player gets a large polygon and he has to touch on the polygon and hit other small polygons with same color or same shape in order to clear the level. If player has a red circle he can hit any polygon in red or any circle. The player's polygon trasforme in realtime when it hits with a small polygon. If the player hits a red tringle, his polygon transformes into a red triangle. Player can zoom in and out the screen using pinch gesture and rotate the player's polygon using rotating gesture.

Developers can refer data model with the above interface. There is a navigation menu at left side for select screen size and the gesture. User can view *portrait view* and *landscape view* heat maps in the middle. He can see actual no. of touches when he move mouse pointer to a grid.

## V. System Testing and Analysis

System testing was done with the unity test tools while implementing the system. Unit tests were written with NUnit [9] in order to confirm functionality of the mandatory methods. Apart from that assertion tests were done in order to confirm the functionality of the player game object. With the assertion tests runtime behavior of the player could be easily tested and identified if there is an unexpected behavior.

There were five unit tests. Three of them were to test the data fetching module and others is to test the game manager and the board manager.

Assertion test were conducted to test the functionality of the player game object. Player have a box collider, a circle collider and a polygon collider. Other than the relevant collider to the type of the polygon in the player, rest of the colliders must be disabled while the game is being played. For this assertion components were added to the player and those are meant to print errors when player hits a polygon.

The tests which cannot be tested with Unit Test and assertion test, were tested by manually debugging. For an example, opening the internet connection and dispatching data to the server cannot be test in unit test in the unity3d environment because it does not allow to return value from IEnumerator methods. Animations in the game menu were manually tested with debugging since it tends to be slow at the beginning in some low end mobile phones. Security of the data also manually tested by debugging PHP script.

## VI. Conclusion and Future Work

This paper describes the design and implementation details of an android game that mainly facilitates the ability to capture user data of the interactions with the touch screen. The game can be downloaded by anyone and the user data will be uploaded to the server. Currently it has received about 200,000 touches counts.

Future works are to update the game with more levels and to promote the game among people since the data model will be more accurate when there is more user data. The project is available at the GitHub and also it can be developed with the collaboration of the developers and researchers who are interested in this area. As well as the project can be extended to get categorized data according to the country, localization and age groups since those parameters are not considered in this project.

Game download link:
https://play.google.com/store/apps/details?id=com.kasun.floating_polygons
The data model : http://floatingpolygons.esy.es/index.php

## VII. References

[1] Marcello Lins. (2015). *Data Mined from the Google Play Store* [Online]. Available FTP: s3.amazonaws.com Directory:GooglePlayStore File: Google Play Stats_2015_02.xlsx

[2] Sarah Martina Kolly et al, “A Personal Touch - Recognizing Users Based on Touch Screen Behavior”, 2012

[3] E. Law and L. von Ahn, “Input-agreement: a new mechanism for collecting data using human computation games”, CHI, 2009.

[4] A. J. Quinn and B. B. Bederson, “Human computation: a survey and taxonomy of a growing field”, CHI, 2011.

[5] B. Tidball and P. Stappers, “Crowdsourcing contextual user insights for ucd. In Workshop on Crowdsourcing and Human Computation”, CHI, 2011.

[6] Niels Henze et al., “100,000,000 Taps: Analysis and Improvement of Touch Performance in the Large”, 2011

[7] Tao Feng et al., “TIPS: Context-Aware Implicit User Identification using Touch Screen in Uncontrolled Environments”, 2014

[8] Unity 3D, Avialable : http://unity3d.com/get-unity

[9] NUnit, Avialable : http://www.nunit.org/