Fill in the Blanks

A Touch Data Gathering Game to Improve UI Designs

Lochana Ranaweera
Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka
lochana.12@cse.mrt.ac.lk

***Abstract* — With the increasing number of utilities provided by touchscreen devices, the process of designing user interfaces is of utmost importance. In the search for UI design decisions, it is necessary to have access to current data related to user behaviour relating to touchscreen inputs. As such, the technique of crowdsourcing has been used to create a game to collect data related to drag and drop, rotate, pinch zoom in and out gestures in the project discussed in this paper. The major design decisions encountered in the project were creating an intriguing gameplay that involves all the touch gestures mentioned above and devising a mechanism to collect and store data related to such movements. The final outcome of the project is a game designed for Android devices that can be distributed via the Google Play Store, which will collect and transfer data to a remote location to be analyzed by UI designers.**

Keywords— User Interface Design; Crowdsourcing; User Interface Optimization; Touch Input; Unity 2D Development

I. INTRODUCTION

With the advent of touchscreens and large surfaces becoming a standard part of user interfaces and the growingly competitive world of User Experience (UX) and User Interface (UI) design, having the right data can make or break a great UI or UX. Therefore, it is of great importance that a careful study of standard touch gestures is carried out prior to UI design so that the designers can eliminate wastage of resources like time, money and the risk of designing for a product that nobody will use. This paper introduces a game that would serve as a computational model of user behaviour relating to touch screen inputs thereby enabling designers and engineers to improve usability of UI. This game is deployed in Android Play Store under the name "Fill in the Blanks" [1]. It allows for simultaneous data collection from multiple users whilst enabling users to engage in an intriguing gameplay that is fun with both visual and audio feedback. The paper discusses how Fill in the Blanks has been designed to collect touch data and the techniques that could be adopted to create a captivating gameplay for a "game with a purpose".

Given the need for collecting data on behaviour relating to touch inputs of many users including both novice and expert users, a technique of crowdsourcing was recommended [2]. When an attractive game is made publicly available for a large community, a model can be built to gather touch screen data to improve UI designs.

The system was created to be utilized as a computational model to collect data relating to touch gestures of users. Notwithstanding the above concern, the game was developed with the aim of enabling users to engage in an intriguing gameplay that is fun with both visual and audio feedback. A captivating gameplay will only make it possible to reach out to the masses in order to collect a considerable volume of touch data as such.

As for the basic development approach, Fill in the Blanks has different shapes of holes present in the surface of the game world. To score, the player has to scale, rotate, drag and drop objects that correspond to those voids, latching it there with an attractive animation. The gameplay allows logging of data from every position of the screen. Whenever the player touches the screen, the game logs the positions and duration of each touch on the screen differentiating by the input method. The data collected as such is stored in the persistent data path allocated for the app on the phone. This data is transmitted to remote server location once the user connects to a Wi-Fi network with the game being played at least once after installation and the application started.

II. LITERATURE REVIEW

Multiple research projects have been carried out all around the world with the aid of crowdsourcing. The interests of these projects span almost every aspect of human life such as healthcare, automobile industry, psychology, social media, environmental preservation and even space exploration. Today, there are millions of smartphone users all around the world spending millions of minutes per day playing games like Clash of Clans. Therefore, the potential of mobile technology to gather data presents researchers with an incredible opportunity to process far more data than ever before.

An interesting example of an existing crowdsourcing project that utilizes mobile technology to reach the masses is a game called "GeneGame" [3]. The data collected from this game goes to a cancer research project carried out by Cancer Research UK [4]. By playing the game the users had made a difference in the search for cures for cancer as the scientists have been able to bring down significantly the time spent on analyzing the data if

not for the game. Compared to GeneGame which is focused on gathering data for cancer research, Fill in the Blanks on the other hand, seeks to gather data that could be used by Big Data analysts with the aim of improving user interface design of mobile devices.

When it comes to systems similar to Fill in the Blanks, an interesting reference is the game "Hit It!"[5]. It is an Android game challenging the users to display quick reflexes. The game has over 100,000 downloads on the Play Store and is for Android devices with a version 1.6 or higher. The level design is as such the user has to touch each circle that appears on the screen as fast as possible to beat all levels. Faster the movement, higher the score will be. While users play the game where they hit the screen and how fast they are measured. By combining this information with the position and size of the circles it is estimated how easy each screen position is to touch. Based on this data user's performance with different button sizes and positions is predicted. According to the developers, the model thus produced is used to improve user interfaces of smartphones. While Fill in the Blanks focuses specifically on obtaining data relating to user gestures of scale, rotate, drag and drop, Hit It! simply is to collect data on how often a particular location in the screen is being touched.

Considering the potential of the technique of crowdsourcing, which gather large amounts of data for research purposes, the development of Fill in the Blanks was justified. The game would be utilized as a computational model to collect data relating to touch gestures of users which can be further analyzed by Big Data experts with the aim of improving user interface design of mobile devices.

III. System Models

A. System Requirement

As for specific requirements, Fill in the Blanks had to be created as a standalone application allowing for offline usage. The gameplay had to be enjoyable with the player being able to perform gestures as required to play the game without frustration. As of now, the player is able to pause the game whenever they wish and resume from that point on. The game automatically pauses when the user leaves the application to another from the android system. Upon completion, the player can view the score and see if there is further time remaining. The menus had to be designed in such a way that a player can navigate between screens without confusion also enabling them to quit the application properly as necessary. Also to learn how to play Fill in the Blanks, it was decided to include a tutorial level. The tutorial level will take the player through a series of tasks which will provide a quick understanding of how the game can be played. The player can go through the tutorial level even at a later time or can choose not go through it at all.

As shown in Fig. 1, the use cases were identified based on three users; the **Player, Game Engine** and **Remote Server.** Accordingly, the main use case of the system is when the **Player** interacts with the game. The precondition of this use case is that Fill in the Blanks has been launched by clicking on the application icon and the **Player** has selected play icon from the main menu. Assuming the **Player** has been able to load the game and it comprises of different shapes of holes and objects present on the surface of the game world, the **Player** should be able to scale, rotate, drag and drop objects corresponding to the rightly shaped hole in order to score. Once done so, the object snaps on to the hole if it has the same size and the alignment as the hole and both fades away with a whirling animation adding the remaining game time to the score. Alternatively, the **Player** is able to pause the game whenever he wishes to and resume from that point on.

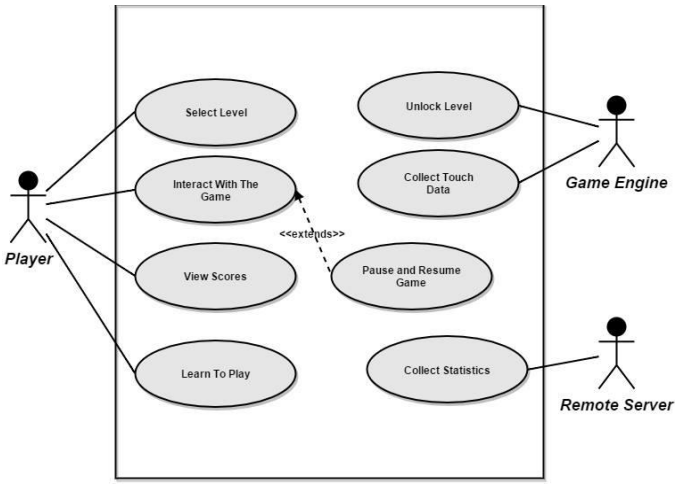

Fig. 1: Use Case Diagram

In terms of non-functional requirements adhered, the user can install the game straight off the play store as for any other android game and play the game without having to sign in to any accounts unnecessarily. Both the application and the level launches within 10 seconds. The game was to feature simple attractive user interfaces, which is captured using the unique "paper-like" theme adopted. To imply that the game is responsive to all of the players' gestures, the game objects comprise of colour changing animations. Regarding accuracy, the game had to detect gestures with maximum accuracy possible and log the touch inputs along with the gestures accurately. The logged data is only accessible by authorized personnel at the backend to ensure security. The device battery could not be drained at an unacceptable rate. Lagging could not be intolerable, holding a minimum of 30 frames per second.

B. System Design

Fill in the Blanks was developed using Unity 3D which is a powerful cross-platform game engine [6]. Therefore, the system architecture is a derivation of the domain-specific architecture adopted by the Unity3D game engine. As such, when explaining the architecture of the game, the top level functional components present can be identified as **Game Engine, Data Manager, Simulation** and the **Object System**. The component responsible for presenting the game to the player and receiving player inputs is the **Game Engine**. This includes the generation of graphics, and audio feedback to the player. The **Simulation** is responsible for updating the state of the virtual game world in response to player inputs and the rules of the game and virtual world. The **Data Manager** retrieves game data from the file system and manages storage and retrieval of game state to save/load game functionality. In Unity3D; a game is a collection of **Scenes** and a **Scene** is a collection of **GameObjects** [7]. A **GameObject**

could be a character, an environment, or a special effect [8]. An **Object System**, as such is responsible for maintaining the state information describing all objects in the game world.

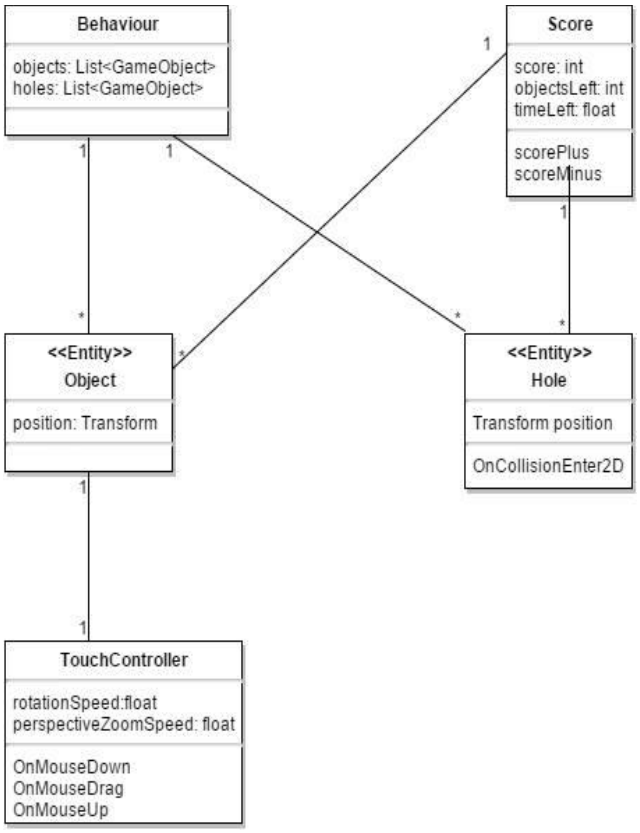

Fig. 2: Class Diagram

The Class diagram utilized during the development of Fill in the Blanks is shown in Fig. 2. The **Object** and **Hole** classes both extend the **MonoBehaviour** class, making them game objects according to Unity3D development. Therefore they have variables named **position** which stores the positions of the objects in the Unity game engine world. They both also have **Colliders** attached; these catch events such as coming into contact with other **Colliders**, when collisions end, etc. [9]. These events are handled according to game logic and update the relevant variables in **Score***(int score)* and **Behavior***(String log)* classes. The **TouchController** class handles all touch events and also handles the in game logging. The **TouchController** class also handles all the view controls such as **Pinch Zooming** and **Pinch Rotating**.

As shown in Fig. 3, the main sequence diagram of the game refers to the use case of interacting with the game. The **touchGesture( )** method could be any touch input made by the user at the touch screen. For example; it could be clicking the start button on the game menu which would cause the game engine to exit from that menu scene and enter the main game scene. It could be a touch input of scaling, rotating or pinch zooming performed on a **GameObject** in the game world. As such, independent of the **Scene** the game is in, the above sequence of inputs and outputs would be seen.

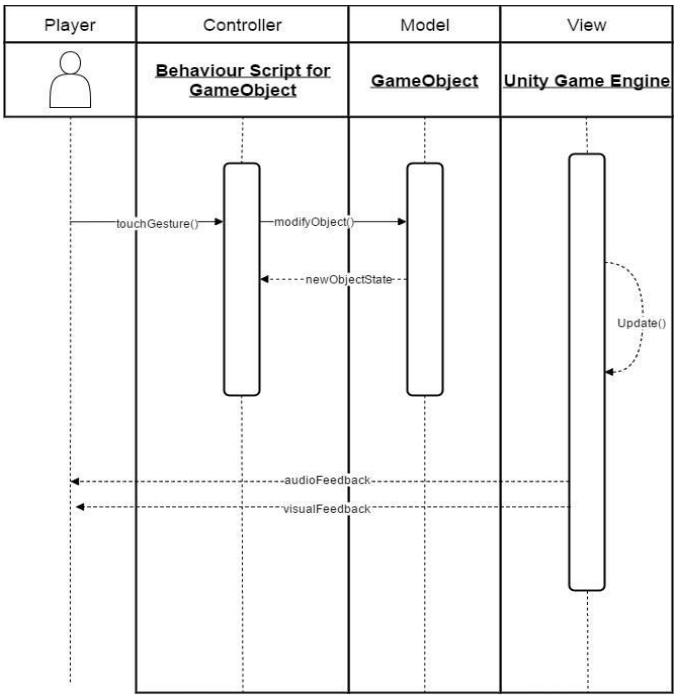

Fig. 3: Main Sequence Diagram

C. Database Design

The backend of the system hosted at a remote server location, includes a database to contain the login details related to administrators to enable secure login, and touch data sent by the mobile devices. Since the system requirements did not include analysis of data collected from users which is stored at the backend database, the database design was kept simple to consist of user login data and touch log to simply store the data being transferred from the mobile devices.

The database schema is supported by **MySQL** has two tables, namely '**login**'and '**touchlog**'. The **login** table includes three columns to have the auto generated identifier which is the primary key, the name of the administrator which of the type ***varchar*** and the corresponding hashed password. The **touchlog** table has four columns, the auto generated event identifier which is the primary key, the event time (***float***), the type of the event (***varchar***), X axis position on the screen (***float***) and the Y axis position of the screen (***float***). Since the database is simply for storing the data, and no foreign key constraints exist, a normalization of the schema was not considered necessary at the present requirements context.

IV. System Implementation

A. Implementation Procedure

Several technologies, tools, resources and strategies were applied in developing the system, Fill in the Blanks. The game was mainly developed as a standalone 2D game application using the Unity 3D editor, which is a free game engine/development tool, in conjunction with Mono Develop (the default IDE that is shipped with Unity) which has C# as the language of development. The backend of the system, which includes the functionality required by the administrators who

are authorized to access the touch data, was developed using NetBeans IDE as a PHP solution and Selenium Web Driver was used during testing. The Unity manual was referred to during development and also it was useful in learning about new features in Unity 5 as an upgrade guide. For the errors and anomalies encountered during development the unity question and answer forums were referred to. As for the relational database management system required at the backend MySQL was used. Apart from that, in building the view layer of the backend solution, the technologies HTML, CSS and JavaScript and the Bootstrap framework were used. For the graphic elements and other visual content required for the game, a photo editing software was used. In order to process the mp3 music files required to create the audio content of the game an audio trimming tool was made use of.

B. Algorithms

As for the algorithmic solutions generated during the development, the touch handling code can be stated as the most important of them. Since a user can make movements using his/her fingers that correspond to multiple different touch gestures, in the touch handling code the gestures had to be identified uniquely. In this section of the paper, the **pinch zoom** code, a section of the touch controller logic recognized as the toughest implementation is explained. The criticality of the algorithms used in the pinch zoom code arises due to the fact when the game is playing the user has to feel that scaling operations are responsive despite the fact that calculations and updates are performed frame-wise.

As shown in Fig. 4, the pinch zoom algorithm uses two ***Boolean*** variables. The variable **isSelected** which is used to check whether is object is about to be modified with the touch control and the variable **isGonnaDrag** which is used to know whether the object is eligible for drag and drop movement. The variable **isSelected** is set true whenever the object is touched and set false whenever the finger is removed. The variable **isGonnaDrag** is set true whenever the object is touched and set to false whenever two fingers are detected at the object. Every time the game engine updates the frame, **for** loop is executed. This is practically happening inside the **Update** function of the game engine [10]. The pinch zoom action starts if there are two fingers on the screen (obtained from the **Input.touches** array list) and the object has been touched, meaning **isSelected** has been set to true [11].

Since the object cannot be eligible for dragging when it is being scaled and rotated, **isGonnaDrag** will be set to false at this point. Thereafter both the touches (**touch_1** and **touch_2**) are stored using the array **Input.touches**. Next using 2D vector objects called **Vector2** the positions of each touch in the previous frame is calculated [12]. In this calculation **deltaPosition** is a 2D vector that represents the difference between the touch position recorded on the most recent update and that recorded in the previous update [13]. Next a ***float*** variable called **prevTouchDeltaMag** is declared. From that point, the magnitude of the vector (the distance) between the touches in previous frame, **prevTouchDeltaMag**, is calculated by obtaining the magnitude of the vector difference for **touch_1** and **touch_2**'s previous positions.

```
private bool isSelected = false;
private bool isGonnaDrag = false;

//If the user has touched the GUI element
        isSelected = true;
        isGonnaDrag = true;
        colour = yellow;
//If the user has touched GUI element and is
holding down the mouse
if (isGonnaDrag &&  time held > 1 second)
       colour = gold;
       isSelected = false;
//If the user has stopped touching the GUI element
       isSelected = false;
       isGonnaDrag = false;
       colour = white;

for each Frame_Update
if (number of fingers at the screen == 2 &&
isSelected)
{
isGonnaDrag = false;
colour = cyan;
touchZero = Input.GetTouch(0);
touchOne = Input.GetTouch(1);
Vector2 touchZeroPrevPos = touchZero.position -
touchZero.deltaPosition;
Vector2 touchOnePrevPos = touchOne.position -
touchOne.deltaPosition;
float prevTouchDeltaMag = (touchZeroPrevPos -
touchOnePrevPos).magnitude;
float touchDeltaMag = (touchZero.position -
touchOne.position).magnitude;
float deltaMagnitudeDiff = touchDeltaMag -
prevTouchDeltaMag;

//Gradully change scale of the object from its
original value in a linear interplative manner
during the time for a frame
}
```

Fig. 4: Pinch Zoom Algorigthm

Similarly, the magnitude of the vector difference for **touch_1** and **touch_2**'s current frame positions is calculated into a newly declared ***float*** variable called **touchDeltaMag**. Then a ***float*** variable called **deltaMagnitudeDiff** is found as to be the difference in the distances between each frame (two touches), which is **prevTouchDeltaMag** subtracted by **touchDeltaMag**. With this the scale of the object being touched can be adjusted by gradually changing the scale from its original value to the final scale which is clamped between 0.3 and 1.5 units (minimum and maximum size to which the object can be zoomed in or zoomed out) in a linear interpolative manner during the time for a frame. The algorithm adopts linear interpolation technique so that objects will not appear suddenly zoomed outside or zoomed inside in consecutive frames in a way that the player's eyes cannot notice the change [14].

In the algorithm for the ease of use of the gamer, several colour transitions are used. Whenever a shape is touched it will be changing the overlay colour to yellow, and whenever it is held for more than one second it will change to gold colour to indicate that the object is now eligible for dragging. Similarly once the object is ready to be scaled (pinch zoom) it will be having cyan as the overlay colour so that the player gets visual feedback on what he's doing at the moment.

C. Main Interfaces

1) Android Game

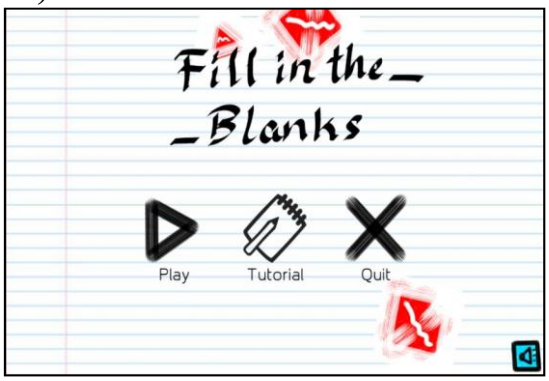

Fig 5: Main menu

As shown in Fig. 5, the main menu of the game comprises of a dynamic display of square and triangular shapes which later appears in the game world created in a random manner and made to float around. A separate audio track is too included for this scene, which can be turned off with by clicking the sound button at the bottom left corner.

Fig. 6: Screenshot from the Tutorial Level

Fig. 6, shows a screenshot of the tutorial level. It too includes its own audio track. The hints for the players appear for some time and fade away or the player can keep moving forward up to the hint on which he/she needs to be clarified.

Fig. 7, is a screenshot from the game arena once the player decides to move on to the actual game. The game arena consists of shapes and holes that have to be eliminated by placing the shapes on the correct holes to scores. Negative marks will be allocated for wrong matches. For the right matches, the shape and object will disappear with a swirling animation once done so, adding the remaining game time to the score. There is a separate audio track which can be muted, and a pause button which pause the game. Fig. 8, gives a snapshot of the game once the game time has come to end.

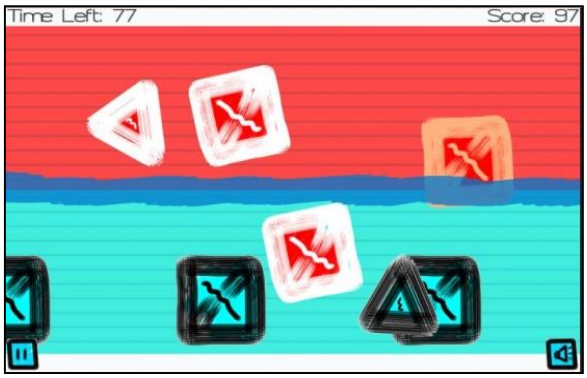

Fig. 7: Game Arena

Fig. 8: Game Over Panel

2) The Backend

At the backend of the system, once the administrator logs in to the system securely, the touch data log will be available to be viewed by the administrator as shown in Fig. 9.

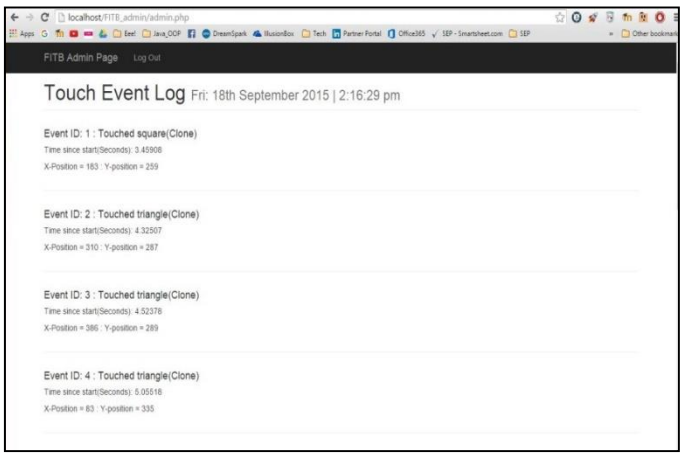

Fig. 9: Touch Event Log as shown in the Backend

V. System Testing and Analysis

The majority of tests were focused on UI functionality as this is a game. When the game was built using the Unity 3D

Editor, the game was played in an Android device per each build using Unity Remote application that can sends the visual output from Unity editor to a connected device's screen and live inputs are sent back to the running project in Unity [15]. Each UI components in this game was tested and ensured that the UI (Scene to Scene) navigation is seamless and UI is of acceptable scale. This was again tested during alpha testing stage, where a selected number of alpha testers were instructed to play the game and report any and all bugs discovered. Also they were recommended to use a log cat by setting the filter rule for log tag using the game's packaging bundle name [16]. During this process, the game was installed in different devices with different screen sizes to see if each UI component assembles in the right manner.

Performance testing aspect of the game was carried out using Unity Profiler tool. The tool when utilized along with Unity Remote helps record performance data for the device when the game is running on it [17]. The tests were realized using a smartphone with a comparatively less amount of RAM (512 MB), and no performance constraints were overridden during testing. The apk generated was less than 75 MB in size (nearing 32 MB actually) thereby the game does not overrule any memory constraints as 75MB is the maximum size for an external application. The game did not crash in all times it was launched during testing and development. However, when the player wants to quit a game, the quit button on the pause menu can be clicked. In this case, the game can be continued back from where it was, so the game has to be restarted. If the game is stopped in the middle of playing without the player doing so; for e.g.: battery drain, the touch log will not get updated as it is done only when the game has completely finished with a win or a game over situation.

VI. Conclusion and Future Work

In a time where many of the day today utilities are delivered to the human kind via touchscreen devices, it is significant to have access to the right data relating the User Interface (UI). Therefore, in order to generalize how majority of users prefer the UI elements to respond to their touch we need a large data set. This paper has presented an Android game named "Fill in the Blanks", with functionalities to collect gesture data and transfer them for crowdsourcing by UI designers. This game is provided with different shapes of holes where the player has to scale, rotate, drag and drop objects that correspond to those voids, in order to score. This Android game can be distributed among the vast community of Android users via the Google Play Store.

The research can be further expanded by implementing the system in multiple device platforms such as Windows, iOS and even online platforms such as Facebook. The touch data collected by the system can be analysed in an organized manner by getting the help of Big Data experts, to seek for new correlations that exist between the user behaviour by device screen position, timing and many other factors.

Although the game is developed with a scientific purpose, to reach out to the masses it is vital that the game has a captivating gameplay. As such, having an intriguing game play may seem to be a good enough reason for a game to become popular, having attractive graphic elements and especially catchy yet not frustrating tunes can also make a huge difference. This was one such observation derived out of implementing Fill in the Blanks.

The touch gesture data collected in this game is sufficient for obtaining UI design ideas for normal touchscreen device. However, at present with the concept of 3D touch introduced by Apple Inc. [18], the way a user interacts with a touchscreen device will change immensely. Therefore it is no longer simply tapping, swiping, or pinching that will make a device work but there'll be an entirely new field of touch gestures such as the concept of "pressure-touch". A game like Fill in The Blanks that has added functionality to support such touch gestures, will surely be an ideal solution.

References

[1] Play.google.com, 'Fill in the Blanks', 2016. [Online] Available: https://play.google.com/store/apps/details?id=com.lochanar.FITB

[2] Howe, J. and Robinson, M. (2006). The Rise of Crowdsourcing. *Wired*. [Online]. Available: http://www.wired.com/2006/06/crowds

[3] Wired.co.uk, 'Smartphone game 'GeneGame' to crowdsouce cancer research', 2013. [Online]. Available: http://www.wired.co.uk/news/archive/2013-07/19/genegame-app-cancer-research

[4] Cancer Research UK. (2002). *Nature Cell Biology*, 4(3), pp.E45-E45.

[5] Google Play Store, 'Hit It!', 2015.[Online]. Available: https://play.google.com/store/apps/details?id=net.nhenze.game.button2

[6] Unity 3D, 'Unity Pro and Unity Personal Software License Agreement 5.X', 2015. [Online]. Available: https://unity3d.com/legal/eula

[7] Unity 3D, 'Creating Scenes'', 2015. [Online]. Available: http://docs.unity3d.com/Manual/CreatingScenes.html

[8] Unity 3D, 'GameObject', 2015. [Online]. Available: http://docs.unity3d.com/ScriptReference/GameObject.html

[9] Unity 3D, 'Collider', 2015. [Online]. Available: http://docs.unity3d.com/ScriptReference/Collider.html

[10] Unity 3D, 'Monobehaviour.Update', 2015. [Online]. Available: http://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html

[11] Unity 3D, 'Input-touches', 2015. [Online]. Available: http://docs.unity3d.com/ScriptReference/Input-touches.html

[12] Unity 3D, 'Vector2', 2015. [Online]. Available: http://docs.unity3d.com/ScriptReference/Vector2.html

[13] Unity 3D, 'Touch-deltaPosition', 2015. [Online]. Available: http://docs.unity3d.com/ScriptReference/Touch-deltaPosition.html

[14] Unity 3D, 'Linear Interpolation', 2015. [Online]. Available: https://unity3d.com/learn/tutorials/modules/beginner/scripting/linear-interpolation

[15] Unity 3D, 'Unity Remote 4', 2015. [Online]. Available: http://docs.unity3d.com/Manual/UnityRemote4.html

[16] Android Developers, 'logcat', 2015. [Online]. Available: http://developer.android.com/tools/help/logcat.html

[17] Unity Documentation, 'Profiler', 2015. [Online]. Available: http://docs.unity3d.com/Manual/Profiler.html

[18] Apple, '3D Touch. The next generation of Multi-Touch', 2015. [Online]. Available: http://www.apple.com/iphone-6s/3d-touch/