Puzzle Up

Android Jigsaw Puzzle Game

Soroush Noursobhi

This thesis is presented as part of Degree of Bachelor of Science in Electrical Engineering department

15 ECTS

Blekinge Institue of Technology – B-right AB

June 2016

Blekinge Institute of Technology

Department of Applied Signal Processing
Supervisors: Anders Hultgren
Examiner: Sven Johansson

B-right AB
Johan Östling

Table of Contents

Abstract

We developed a jigsaw based puzzle game for Android called PuzzleUp. The idea is that you take a picture in real time and it breaks it into jigsaw pieces that you have to put together. It also features a multiplayer mode on local area network (LAN).

The main programming language used is Java and the main development environment is Android Studio (Based on JetBrains Idea) and AllJoyn has been used for inter-device communication to ensure maximum compatibility.

Introduction

Problem Statement

The basic idea is to take a picture in real time using your handheld device and then break it into jigsaw pieces to create a puzzle game.

After breaking into pieces, pieces are spread throughout the screen that you move with your fingers to join them. A timer records the time taken to solve the puzzle. There are different levels as well. You can create puzzles of size 3x3, 4x4, 5x5, 6x6, 7x7 and 8x8.

You can also play a multiplayer game with your peer if they are on the same network. For that, one user takes the picture and challenges the other to a game. If accepted, both play to see who completes the puzzle first.

A scoreboard records the lowest time for each level as well. The latest lowest score is shown on home screen as well.

Scope of Thesis Work

This is a project of software engineering. More specifically, of mobile app development, most important category of software engineering at the time of writing this paper.

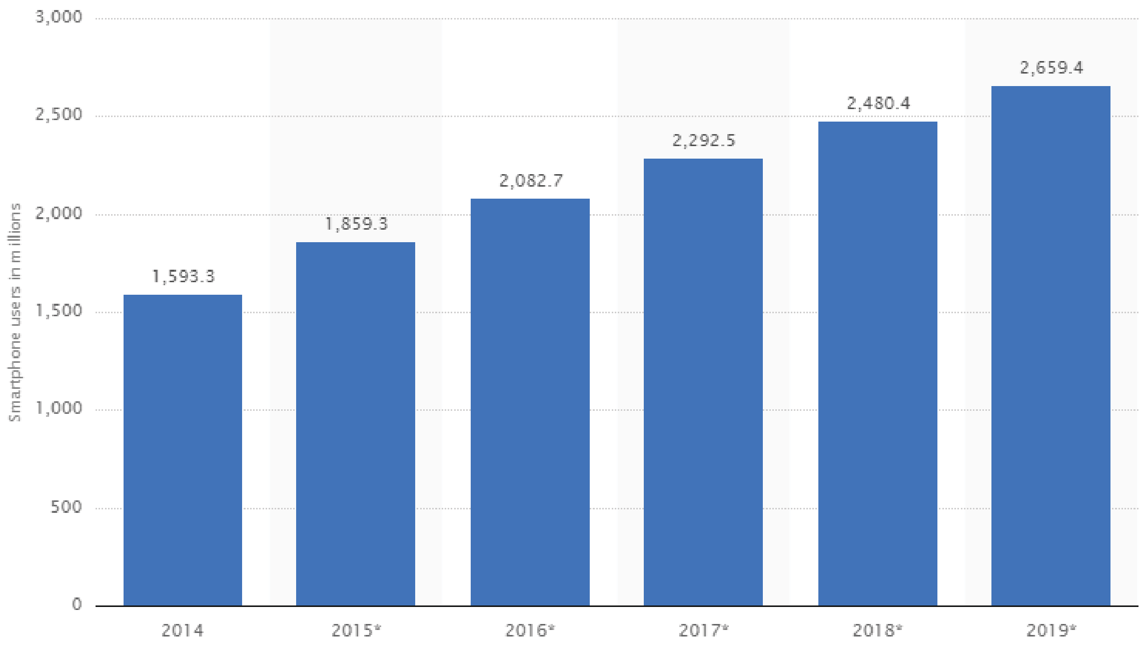


Figure 1 Number of smartphone users over the years

As you can see from the Figure 1, there are approximately 2 billion smartphone users amongst 7 billion people in the world. In another report, more than 80% people play games. For that matter, our project targets one of the biggest sectors in the world.

Design and Implementation

Requirements

As we are making an Android based puzzle game, we kept in mind several requirements before we started working on the project and these are here:

Taking a picture

As the basic idea was to create a puzzle in real time, we had to use phone camera in our app. We had two choices:

1. To use already built-in camera Intent
2. To integrate camera code in our own app

We went with the second option because in the first one, user experience would have varied from device to device.

Moreover, we also had to choose picture from either of front or back camera. While taking a picture, focusing is very important. So our app also required auto-focus. And for low lights, auto-flash was also required.

Selecting picture from gallery

Our app also required to use a pre-taken photo from the gallery. People all sort of data on their smartphones including variety of pictures. So we also had to consider importing a picture from the gallery.

User registration

Many of mobile apps require user registration. For our multiplayer game mode, this feature was needed so peers can find it easy to search for nearby people.

Note that our app is intranet based and not internet based. So we put the information on user’s phone rather than sending it over the cloud to decrease complexity.

We also thought to include the feature to change color theme so the user can tune it to his requirements.

In addition to all this, another requirement was an option to edit these preferences.

Game Levels

Nearly all the game feature difficulty levels so it was our requirement as well. We were supposed to generate varying number of puzzle pieces from the picture so it gets harder to play. We planned to include a user friendly control so difficulty level can be selected without entering it through keyboard.

The various difficulty levels that we included are: 3x3, 4x4, 5x5, 6x6, 7x7 and 8x8.

Scoreboard

Scoreboard is very crucial for games including ours. We had to keep track of the fastest puzzle solving time. But that was not it. Game is difficult to play on different difficulty levels. So we planned to include fastest time for each difficulty level.

Moreover, there was a separate scoreboard for multiplayer as well. That multiplayer score board also featured different difficulty levels.

The scoreboard was required to be accessible from the home screen. We also planned to include the time for last played time on the home screen itself.

The Game Challenge

As per requirement of a jigsaw puzzle, our game also required the puzzle pieces to be randomly placed on the screen. Not only were they supposed to placed randomly on the screen, we also thought of adding a rotation feature to make the game a little bit more fun. In short, the plan was to move pieces by dragging through finger and rotate them 90 degree by tapping on them. So if a piece is correctly rotated and comes next to piece where it fits, it gets snapped to that piece and from then on, both these pieces move together. If all the pieces are snapped together, the game finishes. It is exactly the same experience as real life jigsaw puzzle.

Multiplayer

We planned to let nearby devices play a multiplayer game with each other. The best solution was to incorporate Local Area Network. So if two users are on the same network (For example same WiFi network), they can find each other and play a multiplayer game. One user was supposed to take (or select) a picture and then it challenges the other user to a game. If he/she accepts it, the picture is transferred to the other user and the game is started. Whoever finishes first, wins.

Design

In design, there are two things:

1. The app flow
2. The puzzle algorithm

App Flow

You can see the basic flow that we decided in Figure 2:

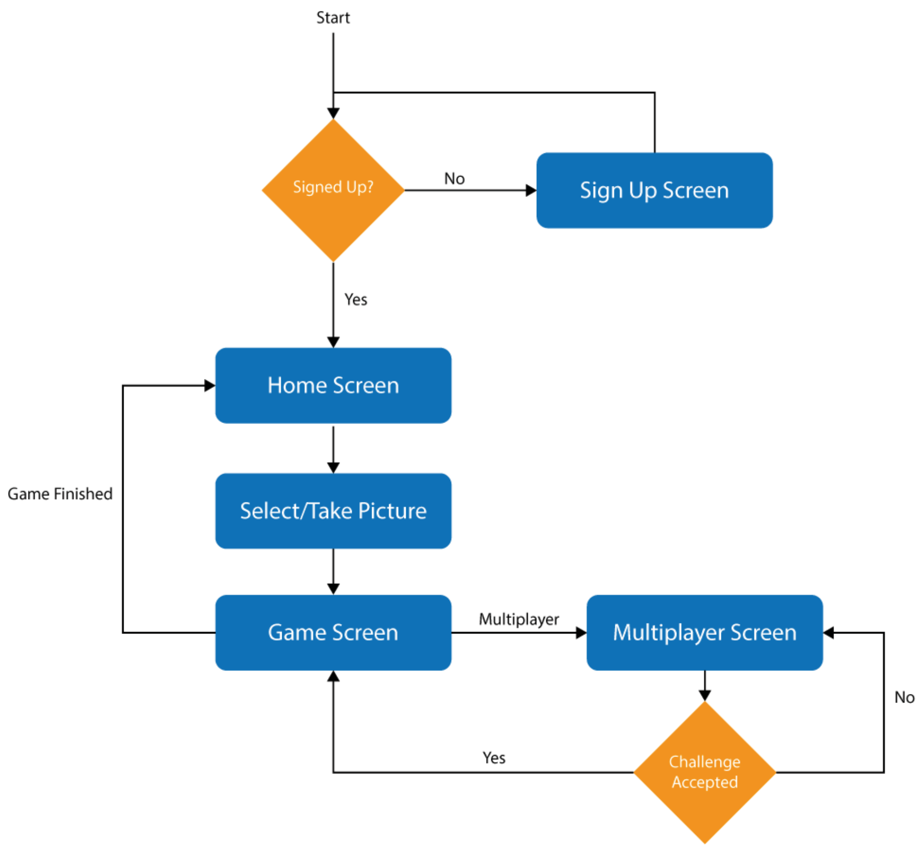


Figure 2 Flowchart of the game

When the user starts the app, it is checked whether he is registered or not. If not, he is taken to the sign up screen.

There, the user provides his name and a picture. After signing up, he is taken to the home screen. Home screen provides options to start game from the gallery or the camera. From the home screen, user can also view the scoreboard or change his name or profile picture.

When the user selects a picture from either gallery or take a new from camera, he is taken to the game screen where he is given option to choose the difficulty level. Then a tap or device shake breaks the picture into jigsaw puzzle pieces. Next the player chooses either to start single player or multiplayer game. If single player, the game immediately starts. If multiplayer, he is taken to the multiplayer deices screen where he chooses an opponent. If the opponent accepts the challenge, both players are returned to the game screen where they finish the puzzle. Whoever finishes first, wins.

Puzzle Algorithm

The most important part of the app is the algorithm used to break a picture into puzzle pieces. We had two choices while designing the algorithm:

1. Design a static algorithm with curves
2. Design a dynamic algorithm which breaks the puzzle from a raw data file

We went with the second option for the main reason that we did not have to hard code the algorithm and making it more dynamic and improvable.

The very first step is to calculate 4 dimension points of each puzzle piece which play crucial part later. The MATLAB code for the calculation is given by:

```
function [x1,y1,x2,y2,x3,y3,x4,y4]=calculatePieceSize(x, y, width, height,
poverflowW, poverflowH, numColumns, numRows)
% x,y are 1 indexed column and row numbers
% (x1,y1) starting point of inner rectangle
% (x2, y2) ending point of inner rectangle
% (x3, y3) starting point of outer rectangle
% (x4, y4) ending point of outer rectangle

x=x-1;
y=y-1;

w=width/numColumns;
h=height/numRows;

x1=x*w+1;
y1=y*h+1;

x2=(x+1)*w;
y2=(y+1)*h;

x3=x1;
y3=y1;

if x~=0
    x3=x3-poverflowW*w;
end
if y~=0
    y3=y3-poverflowH*h;
end

x4=x2;
```

```
y4=y2;

if x~=numColumns-1
    x4=x4+poverflowW*w;
end
if y~=numRows-1
    y4=y4+poverflowH*h;
end

x1=round(x1);
y1=round(y1);
x2=round(x2);
y2=round(y2);
x3=round(x3);
y3=round(y3);
x4=round(x4);
y4=round(y4);

end
```

The function takes 8 arguments:

1. x is the x-index of the piece. Will be 1 for first piece along the width
2. y is the y-index of the piece. Will be 1 for first piece along the height
3. width is width of the image
4. height is height of the image
5. poverflowW is a number between 0 and 1 which determines width of jigsaw boundary. We chose it be constant 0.1
6. poverflowH is a number between 0 and 1 which determines height of the jigsaw boundary. We chose it be constant 0.1

The algorithm is iterated for each piece. So if we require grid size 3x3, it will be iterated 9 times. For each piece, it will return 4 points:

1. Point 1 (x1,y1) denotes the ultimate start of the puzzle piece
2. Point 3 (x3,y3) denotes the start of the boundary after the circular part
3. Point 2 (x2,y2) denotes the end point of the boundary excluding circular part
4. Point 4 (x4,y4) denotes the ultimate end point of the puzzle piece

So if the puzzle piece is from top left corner of the picture, x1 will be same as x3 and y1 will be same as y3 because there is no circular part there. And if it is bottom right corner, x2 will be same as x4 and y2 will be same as y4 because there is no circular part over there. Similarly, the function also takes care of top pieces, bottom, left or right pieces accordingly.

After calculation of these dimension points, they are input to another algorithm which generates the puzzle pieces which means breaking down our original picture to our desire jigsaw shape:

```
function [imagePiece]=treat(image,breaker,x1,y1,x2,y2,x3,y3,x4,y4, LT,RB,L)

% Cut out the image from the original image
imagePiece=image(y3:y4,x3:x4,:);

% Convert from RGB to RGBA
```

```
imagePiece(:,:,4)=255;


% Keep flags to cut out the corners later
treatLeft=0;
treatRight=0;
treatTop=0;
treatBottom=0;




% Treat left
if x1~=x3
    % Set flag to 1 for later use
    treatLeft=1;
    % w represents the extra portion of the image where it can be cut
    w=x1-x3;
    h1=y1-y3+1;
    h2=y4-y2;
    % Separate out the treating part from the original image piece
    treatImage = imagePiece(h1:size(imagePiece,1)-h2,1:2*w,:);
    % Resize the breaker image according to that piece
    breakerB=imresize(breaker,[size(treatImage,1) size(treatImage,2)]);

    % Pull out the alpha matrix
    alphaMatrix=treatImage(:,:,4);
    % Set the respective alpha values
    alphaMatrix(breakerB==LT)=255;
    alphaMatrix(breakerB==RB)=0;

    % Put it back in the treat piece
    treatImage(:,:,4)=alphaMatrix;

    % Put the treat piece back in the image piece
    imagePiece(h1:size(imagePiece,1)-h2,1:2*w,:)=treatImage;
end

% Treat right
if x2~=x4
    treatRight=1;

    % w represents the extra portion of the image where it can be cut
    w=x4-x2;

    h1=y1-y3+1;
    h2=y4-y2;

    % Separate out the treating part from the original image piece
    treatImage = imagePiece(h1:size(imagePiece,1)-h2,size(imagePiece,2)-
2*w:size(imagePiece,2),:);
    % Resize the breaker image according to that piece
    breakerB=imresize(breaker,[size(treatImage,1) size(treatImage,2)]);

    % Pull out the alpha matrix
    alphaMatrix=treatImage(:,:,4);
    % Set the respective alpha values
    alphaMatrix(breakerB==LT)=0;
    alphaMatrix(breakerB==RB)=255;
    % Put it back in the treat piece
    treatImage(:,:,4)=alphaMatrix;

    % Put the treat piece back in the image piece
    % imagePiece(h:size(imagePiece,1)-h,size(imagePiece,2)-
2*w:size(imagePiece,2),:)=treatImage;

    imagePiece(h1:size(imagePiece,1)-h2,size(imagePiece,2)-
2*w:size(imagePiece,2),:)=treatImage;
end

% Treat top
if y1~=y3
    treatTop=1;

    % w represents the extra portion of the image where it can be cut
    w1=x1-x3+1;
    w2=x4-x2;
    h=y1-y3;
    % Separate out the treating part from the original image piece
    treatImage = imagePiece(1:2*h,w1:size(imagePiece,2)-w2,:);
    % Resize the breaker image according to that piece
    breakerB=imrotate(breaker,270);
    breakerB=imresize(breakerB,[size(treatImage,1) size(treatImage,2)]);

    % Pull out the alpha matrix
    alphaMatrix=treatImage(:,:,4);
    % Set the respective alpha values
    alphaMatrix(breakerB==LT)=255;
    alphaMatrix(breakerB==RB)=0;

    % Put it back in the treat piece
    treatImage(:,:,4)=alphaMatrix;

    % Put the treat piece back in the image piece
    imagePiece(1:2*h,w1:size(imagePiece,2)-w2,:)=treatImage;
end

% Treat bottom
if y2~=y4
    treatBottom=1;

    % w represents the extra portion of the image where it can be cut
    w=x1-x3;
    w1=x1-x3+1;
    w2=x4-x2;
    h=y4-y2;
    % Separate out the treating part from the original image piece
    treatImage = imagePiece(size(imagePiece,1)-
2*h:size(imagePiece,1),w1:size(imagePiece,2)-w2,:);
    % Resize the breaker image according to that piece
    breakerB=imrotate(breaker,270);
    breakerB=imresize(breakerB,[size(treatImage,1) size(treatImage,2)]);

    % Pull out the alpha matrix
    alphaMatrix=treatImage(:,:,4);
    % Set the respective alpha values
    alphaMatrix(breakerB==LT)=0;
    alphaMatrix(breakerB==RB)=255;
    % Put it back in the treat piece
    treatImage(:,:,4)=alphaMatrix;

    % Put the treat piece back in the image piece
    imagePiece(size(imagePiece,1)-
2*h:size(imagePiece,1),w1:size(imagePiece,2)-w2,:)=treatImage;
end

% Delete the top left corner
if treatLeft==1 && treatTop==1
    imagePiece(1:(y1-y3),1:(x1-x3),4)=0;
end

% Delete the top right corner
if treatRight==1 && treatTop==1
    w=x4-x2;
    imagePiece(1:(y1-y3),size(imagePiece,2)-w:size(imagePiece,2),4)=0;
end

% Delete the bottom left corner
if treatLeft==1 && treatBottom==1
    h=y4-y2;
    imagePiece(size(imagePiece,1)-h:size(imagePiece,1),1:(x1-x3),4)=0;
end

% Delete the bottom right corner
if treatRight==1 && treatBottom==1
    w=x4-x2;
    h=y4-y2;
    imagePiece(size(imagePiece,1)-h:size(imagePiece,1),size(imagePiece,2)-
w:size(imagePiece,2),4)=0;
end

end
```

The function “treat” uses following arguments as input:

1. image is the original untouched image
2. breaker is a raw file image consisting of three colors, red, green and blue:

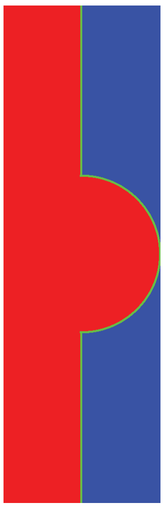


Figure 3 Breaker raw image

3. x1, y1, x2, y2, x3, y3, x4, y4 are the output of the previous algorithm
4. LT is a color from raw file which denotes the inner part of the puzzle piece. It is #FF0000 in the above image
5. RT is the color from raw file which denotes the outer part of the puzzle piece. It is #0000FF in the above image
6. L is the line color from raw file. It is #00FF00 in the above image

The algorithm overlays the breaker raw image on the boundaries of puzzle pieces in accord with four dimension points and then cuts out the blue and green part. (Or if you want, you can make the green part a line)

It also takes care if there will be no such boundary (for bottom, top pieces etc) and also cuts out the excess from corners.

It returns the cut out puzzle piece bitmap which can be manipulated around in the game.

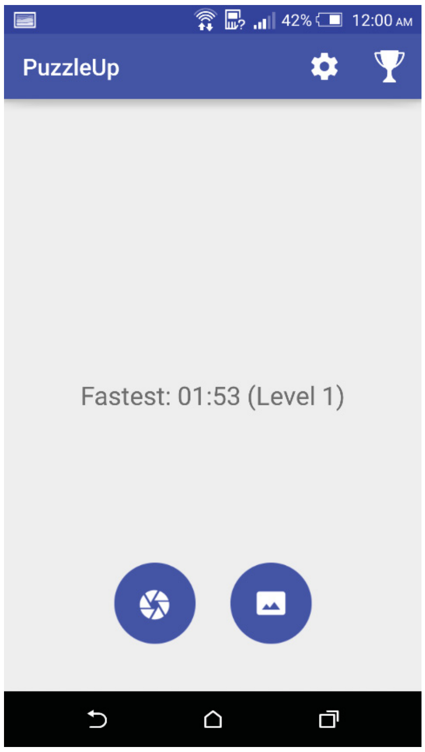
These two were the core algorithms for the game. There are also algorithms to move around pieces, detect if they snap to each other but they were too vague to develop here. For that reason, we completed that part straight in implementation.

Implementation and Results

Home Screen

As designed in app flow, home screen must provide option to start game from either gallery or camera. In addition, there should be option to view high scores and change your settings, see Figure 4.

As the most important buttons here were game starting buttons, hence we put them as floating action buttons [6]. The settings and high scores were made part of the action bar. As the space looked empty, hence we put high score from the last played level as well.



Fastest: 01:53 (Level 1)

Figure 4 Home Screen

Settings/Sign up Screen

In Settings/Sign up screen, you have the option to put in your profile picture, change your name or change the background color. Background color is the color which is displayed in background when you are in game.

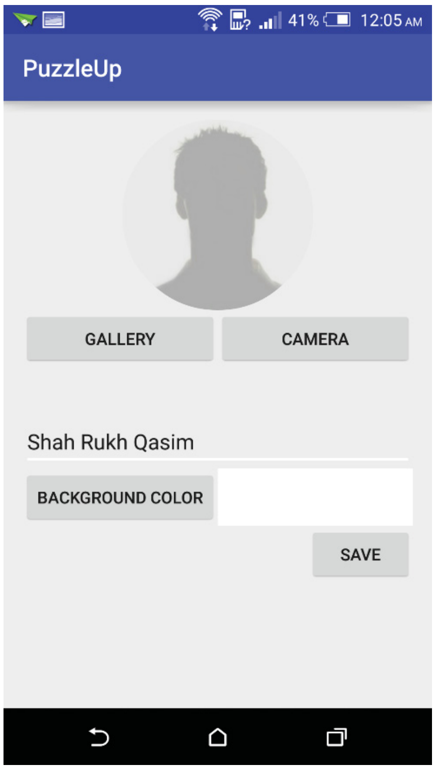


Figure 5 Registration screen

To display a circular picture, we used circular image view library [1]. For choosing color, we used a color picking library called Color Picker [2].

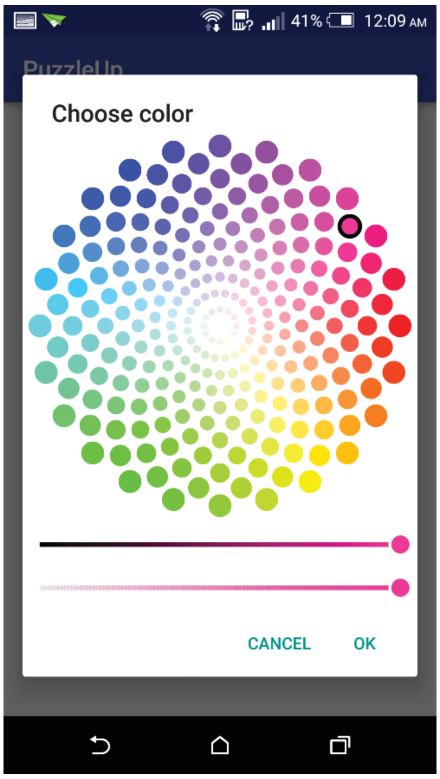
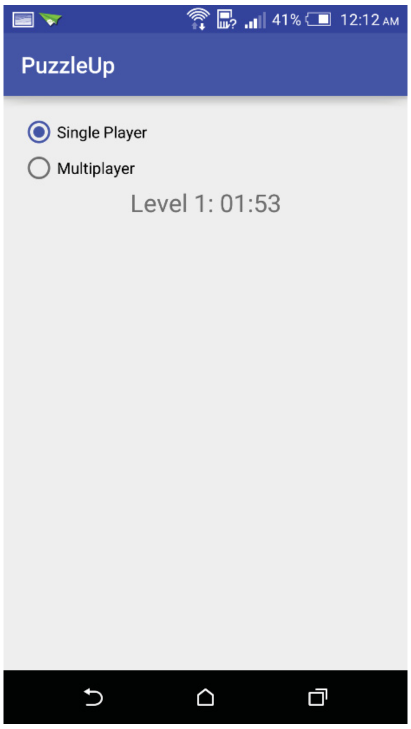


Figure 6 Color picker

High Scores Screen

There are two types of high scores, single player and multiplayer. You can choose what to see from here. All the levels played are then displayed.



[✓ Single Player]
Multiplayer
Level 1: 01:53

Figure 7 High scores screen

Choosing Image for Game

As discussed earlier, there are two options: gallery or camera. If you press the gallery button, you are given option to pick image with the help of gallery apps already installed on your device. If you chose camera, an in-app camera is opened. It features auto flash and auto focus. You can also change from front to back and vice versa from the settings button. This was achieved with the help of cwac-cam2 [3] library from github.

The image is loaded with the help of glide [4] library which helps in managing memory and cropping it to size.

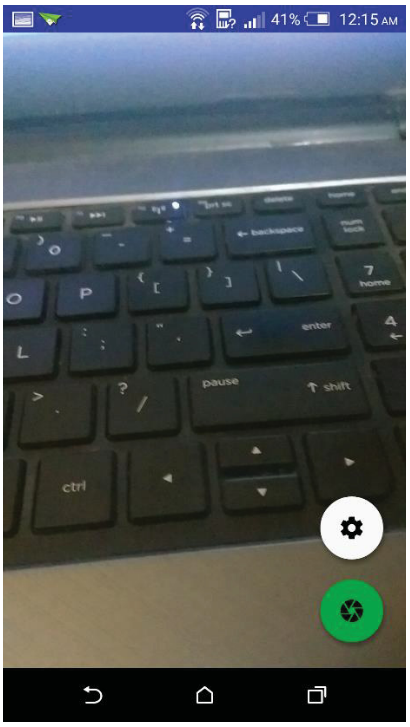


Figure 8 Camera screen

Game Screen

After selecting an image, you are taken to the game screen. Initially it gives you an option to select the game level. Which determines the grid size. You can go from 3x3 to 8x8.

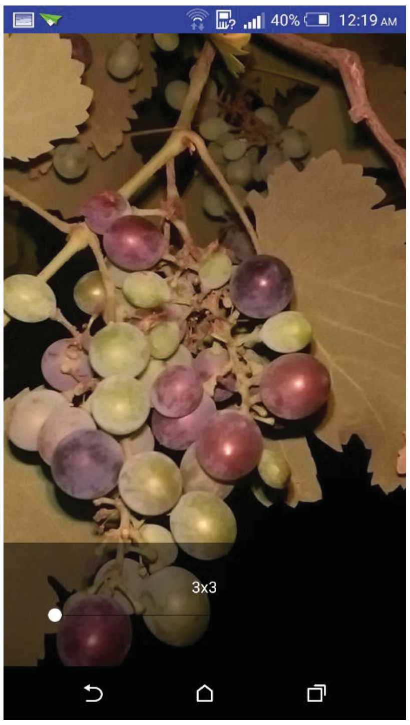


Figure 9 Difficulty level selection screen

After you have selected a level, you can either tap on screen or you can shake your device. The puzzle is broken into pieces. The experience is animated. Animation was achieved by a sin function. Animation happens both in rotation and translation. Following code demonstrates it:

```
int currentX = startOffset.x + (int) (Math.sin((((float) time) / totalTime) *
        Math.PI / 2) * (endingOffset.x - startOffset.x));
int currentY = startOffset.y + (int) (Math.sin((((float) time) / totalTime) *
        Math.PI / 2) * (endingOffset.y - startOffset.y));
double orientationCurrent = orientationStart + (Math.sin((((float) time) /
        totalTime) * Math.PI / 2) * (orientation - orientationStart));
```

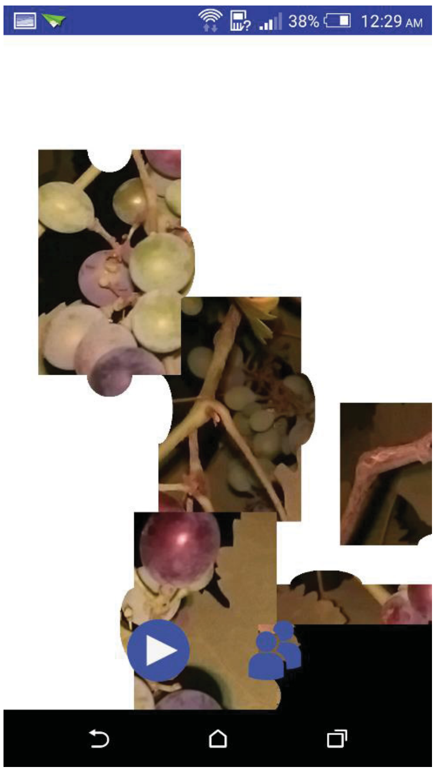


Figure 10 Game start menu

Now you can either start the game or go for multiplayer. As obvious by the buttons. At this point, you can shuffle the pieces again by either tapping the screen or shaking the device. Now that the game is started, you can see a timer and you may move around the pieces. To make the experience more user friendly, the selected piece gets transparent and the joined pieces go to back. You can also rotate the pieces by simple tap.

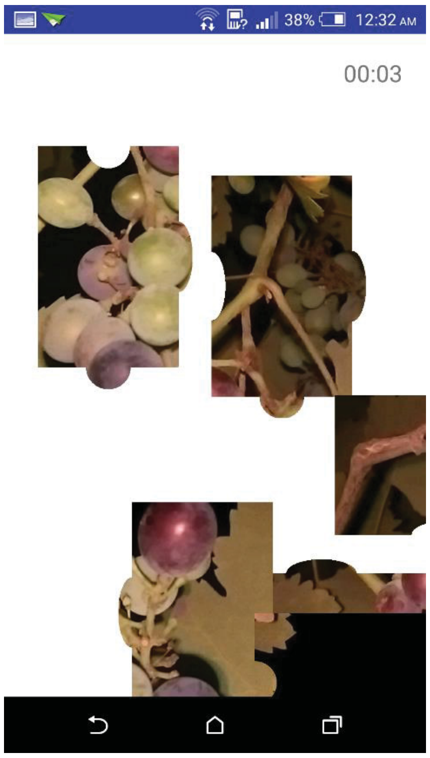
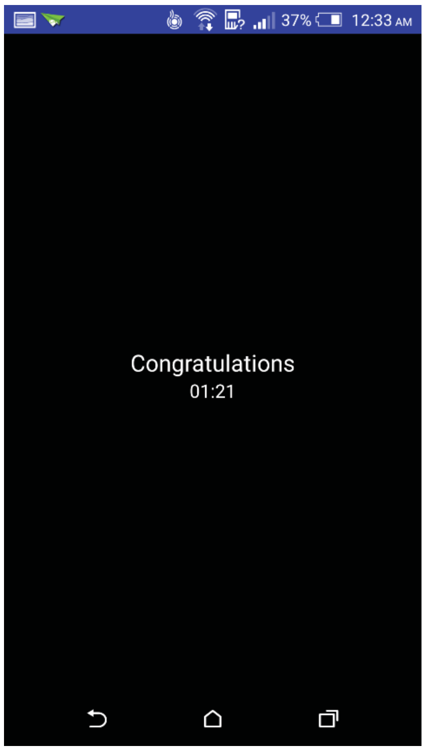


Figure 11 Game screen

When two pieces in correct orientation are near to each other, they get joined. Join all the pieces to win the game.



Congratulations
01:21

Figure 12 Game end screen

Vibration and Sound Feedback

To make the experience more interactive, we used sound and vibration feedback during game. When you break the image, you hear a sound and a vibration. When two pieces get joined, you hear a sound and when you win, you hear a sound and a vibration.

Multiplayer

There was also an option to start a multiplayer game. When you press the button, it takes you to a screen from where you can choose your opponent. Opponent's device must have to be on the same wireless network. Opponent receives a challenge notification. If he accepts it, both are taken to the game screen. Whoever completes first, wins.

For this multiplayer feature, we chose AllJoyn Framework to ease the job for us. AllJoyn is an open source framework maintained by Qualcomm. AllJoyn hides the network communication involving IP addresses, connections etc. from the developer and replaces it by simple function calls. AllJoyn gives you two options:

1. Peer to peer network
2. Client server architecture

We went with client server architecture in which every device hosts a server and broadcasts the name. The challenger in a game is the client which challenges the server to a game.

In AllJoyn, there exists a concepts of bus. Every communication entity must initiate a BusAttachment and communication is done via BusObject. BusObject relies on BusInterface which defines remote BusMethods or BusSignals. BusMethods are functions which can be remotely called. Ours is a simple interface consisting of five BusMethods:

```
@BusInterface(name = "com.example.puzzleup.CommunicationInterface")
public interface CommunicationInterface {
    @BusMethod
    void connect(String clientId, String serverId, String name);
    @BusMethod
    void disconnect(String clientId, String serverId);
    @BusMethod
    void messageToServer(String clientId, String serverId, byte[]message);
    @BusMethod
    void bigMessageToServer(String clientId, String serverId, int totalSize,
byte[]message, int index, int size, String messageId);
    @BusProperty
    String getName();
}
```

As the name implies, these functions are used to remotely do tasks such as connect, disconnect or send messages.

The server broadcasts its name is reverse domain notation to avoid interference. We used domain name:

`com.example.puzzleup.communication.UUID`

Here, UUID means Universally Unique IDentifier. It is generated using from util library in Java itself. Now when a device needs to search using AllJoyn, it can simply search for:

`com.example.puzzleup.communication`

And all the devices in the proximity come up. Then they can connect and communicate using simple bus methods. AllJoyn supports maximum transport size of 64 KiB. So we also had to fragment the data and rejoin on the other side while sending images.

Conclusion and Future Work

This paper discussed the app practically and how an idea was converted into a project. We started with a simple idea. Made the designs on paper and then worked its way to complete a basic product. Applying an incremental software methodology, we kept on improving and adding features. For the paper however we have divided it completely into design and implementation phases.

A product, especially a software related one, is never perfect. We can always make changes and improve it further. Same goes with our project as well. It is a release ready app but it can still use many changes.

Improvements

Bug Fixing

We have fixed all the bugs we came across but for a product, it is very hard to develop a bug free software. Therefore, whatever bugs users come across at later stage, they might need fixes.

Notifications

When a challenge is received, we should receive a notification in notification drawer of the device.

Better Scoreboard

So far the scoreboard only shows the best results of each stage. It would be very great if that can be improved to include number of games played, history of games etc.

Future Work

Over-the-Internet Multiplayer

So far, we have only done LAN multiplayer but most games nowadays feature internet connectivity in which people can play games over the Internet. We will require a server for this which will increase the complexity of the project but it can be treated as a future work.

Google Play Games Services

We can also include Google Play Games Services API. We will gain several advantages this way:

1. Save the game data to the cloud
2. Study analytics of the app usage

Socialize our app

Our game involves multiplayer and hence involving social platforms such as Facebook will not be a bad idea. People will love the game even more as reaching out to friends will become easier. Facebook provides API to achieve this task and so do other social networks.

Support more devices

So far our app only supports Android. Android only shares ~61% of the market at the time of writing this paper. iOS shares ~23% and Windows shares ~3%. If we can develop it for iOS and Windows, we can cover nearly all of the market. Even only for iOS will suffice to a good extent.

References

1. CircleImageView (Accessed January 2016)
   https://github.com/hdodenhof/CircleImageView
2. ColorPicker (Accessed January 2016)
   https://github.com/QuadFlask/colorpicker
3. Cwac-cam2 (Accessed February 2016)
   https://github.com/commonsguy/cwac-cam2
4. Glide (Accessed January 2016)
   https://github.com/bumptech/glide
5. AllJoyn (Accessed May 2016)
   https://allseenalliance.org/framework
6. Floating Action Buttons (Accessed January 2016)
   https://material.google.com/components/buttons-floating-action-button.html