Umamah Aqeel
BSCS - VIII
Final Year Project

Dr. Onaiza Maqbool
Supervisor

Department of Computer Sciences
Quaid-i-Azam University, Islamabad

August 31, 2018

Snap Attack

Android based Word Game
(Urdu Version)

Table of Contents

CHAPTER 3: SOFTWARE DESIGN DESCRIPTION

CHAPTER 4: SOFTWARE IMPLEMENTATION

CHAPTER 5: SOFTWARE TESTING AND EVALUATION

Chapter 1

Software Product Management Plan

INTRODUCTION

Project Overview

Snap Attack is an Android based game in which player achieves highest score in two and a half minute round by building snaps on board with letters on tiles (see SRS for detailed gameplay).

This project provides a user with an intellectually stimulating experience alongside including enjoyment. It targets the community familiar with Urdu alphabet, the people who can read/ write Urdu, who can test and/ or enhance their Urdu vocabulary.

Project Deliverables

Project deliverables for this game are:

- Software Project Management Plan (SPMP)
- Software Requirement Specifications (SRS)
- Software Design Description (SDD)
- Software Implementation
- Software Test Documentation (STD)

PROJECT ORGANIZATION

This section encompasses description of software process model used for the project, roles and responsibilities of people involved in the project, and tools and techniques to be used.

Software Process Model

The process model followed for this project is *incremental model*. The project is organized to generate short-term wins by dividing the work into three iterations. Every iteration focuses on only one task/ feature added to the game. This model helps me to achieve:

- Higher quality
- Faster and more reliable results, or just to achieve results at all
- And reduce frustration and turnover

In **first iteration**, *Snap Attack* is developed as a single player game complete in itself. It is described in detail in SRS document. This is the minimum requirement for the project. All the chapters in the documentation ahead are detailed for this iteration.

In **second iteration**, the game is to be implemented on client-server architecture providing online connectivity. It allows user to play online against other users. Game flow will be same as in first increment. Although, ranking or some other criteria will be defined to match players to play against.

In **third iteration**, *play against computer,* is added to the game. It includes the implementation of Artificial Intelligence algorithms.

Roles and Responsibilities

There is no division of roles and responsibilities because I am a single developer/ designer/ manager of the project.

Tools and Techniques

Following are the tools used:

1. **Google Docs**: All the documentation is done with free, web-based software office suite offered by Google within Google Drive.
2. **SmartDraw**: Usecase diagram is made with the help of this licensed site.
3. **Google Drawings**: All of the diagrams (except usecase diagram) are made with this web-based diagramming software developed by Google.
4. **ProjectLibre 1.6.2**: used for making the project plan.
5. **Android Studio**: for the project development.

PROJECT MANAGEMENT PLAN

Tasks

There are two main phases of project plan other than implementation and Testing. First is *the requirement and analysis* phase and second is the *design* phase of this game.

In requirement and analysis phase, the major tasks are to identify requirements, define use cases, develop domain model, develop SRS and review SRS.

In second phase, the major tasks are to develop a design using Object Oriented Approach, develop models and evaluate design.

Description:

Task 1: Requirement and Analysis

Identify requirements: The main goal is to review case study and deine requirements by meeting stakeholder(s).

Define use cases: Define use cases and make a use case diagram.

Develop SRS: Define functional and nonfunctional requirements and develop software requirement specification document. It includes all other details of product like scope, purpose and introduction.

Review SRS: Review software requirement specification document.

Task 2: Design

Develop design: Develop architectural and interface design using Object Oriented Approach. Develop system sequence diagram and class diagram.

Evaluate design: Evaluate and verify design.

Task 3: Implementation and Testing

Implement: Implement the project corresponding to analysis and design phases

Document: Document the implementation properly.

Test: Test the deliverable product of first iteration and deploy it.

Deliverables and Milestones

See the *Timetable* section for deliverables and milestones.

Resources Needed

Name	Type
⊟**People**	**Work**
Umamah Aqeel	Work
Supervisor	Work
⊟**Software**	**Material**
Word processing tool: Google Docs	Material
Design Tool: Google Drawings, SmartDraw, ArgoUML	Material
Development Tool: Android Studio	Material
⊟**Hardware**	**Material**
PC	Material

Timetable

The timetable for first iteration can be seen in the following figures

	Name	Duration	Start	Finish	Resource Names
1	**⊟Analysis and Requirements**	21 days?	**11/2/17 8:00 AM**	**11/30/17 5:00 PM**	**Hardware;People;...**
2	**⊟Identify Requirements**	7.5 days?	**11/2/17 8:00 AM**	**11/13/17 1:00 PM**	
3	Poblem Definition	1 day?	11/2/17 8:00 AM	11/2/17 5:00 PM	
4	Define Requirements	6 days?	11/3/17 8:00 AM	11/10/17 5:00 PM	
5	Meet Stakeholder	0.5 days?	11/11/17 8:00 AM	11/13/17 1:00 PM	Supervisor;Umamah ...
6	**⊟Define Usecase**	4 days?	**11/17/17 8:00 AM**	**11/22/17 5:00 PM**	**Software**
7	Write Usecase	4 days?	11/17/17 8:00 AM	11/22/17 5:00 PM	
8	Draw usecase Diagram	0 days?	11/18/17 8:00 AM	11/20/17 5:00 PM	
9	Review	0 days?	11/19/17 8:00 AM	11/20/17 5:00 PM	
10	**⊟Develop SRS**	6 days?	**11/20/17 8:00 AM**	**11/27/17 5:00 PM**	**Software**
11	Define Functional and nonfunctional Requirements	3 days?	11/20/17 8:00 AM	11/22/17 5:00 PM	
12	Write Usecase Description	2 days?	11/23/17 8:00 AM	11/24/17 5:00 PM	
13	Finalize SRS	0 days?	11/25/17 8:00 AM	11/27/17 5:00 PM	
14	**⊟Review SRS**	1 day?	**11/30/17 8:00 AM**	**11/30/17 5:00 PM**	
15	Meet Stakeholder	1 day?	11/30/17 8:00 AM	11/30/17 5:00 PM	Supervisor;Umamah ...

	Name	Duration	Start	Finish	Predecessors	Resource Names
1	**⊟Snap Attack**	21 days?	**12/1/17 8:00 AM**	**12/29/17 5:00 PM**		**Tools;Hardware;W...**
2	Problem Understanding	1 day?	12/1/17 8:00 AM	12/1/17 5:00 PM		
3	**⊟Develop System Design**	12 days?	**12/1/17 8:00 AM**	**12/18/17 5:00 PM**		**Tools;Hardware;W...**
4	**⊟SYSTEM ARCHITECTURAL DESIGN**	5 days?	**12/1/17 8:00 AM**	**12/7/17 5:00 PM**		
5	Develop Architectural Design	5 days?	12/1/17 8:00 AM	12/7/17 5:00 PM		
6	Review Archutectural Design	1 day?	12/1/17 8:00 AM	12/1/17 5:00 PM		
7	**⊟Detail Design**	5 days?	**12/1/17 8:00 AM**	**12/7/17 5:00 PM**		
8	Create Sequence Diagram	3 days?	12/1/17 8:00 AM	12/5/17 5:00 PM		
9	Create Class Diagram	2 days?	12/6/17 8:00 AM	12/7/17 5:00 PM	8	
10	**⊟Interface Design**	7 days?	**12/8/17 8:00 AM**	**12/18/17 5:00 PM**	**7;4**	**People;Tools;Hard...**
11	Develope Interface Design	6 days?	12/8/17 8:00 AM	12/15/17 5:00 PM		
12	Review Interface Design	0 days?	12/16/17 8:00 AM	12/18/17 5:00 PM		
13	DSD Deliverable	1 day?	12/18/17 8:00 AM	12/18/17 5:00 PM		
14	**⊟Softtware Test Documentation**	6 days?	**12/19/17 8:00 AM**	**12/26/17 5:00 PM**	**3**	
15	System Overview	4 days?	12/19/17 8:00 AM	12/22/17 5:00 PM		
16	Test Plan	4 days?	12/19/17 8:00 AM	12/22/17 5:00 PM		
17	Test Cases	3 days?	12/22/17 8:00 AM	12/26/17 5:00 PM		
18	Review Test Documentation	3 days?	12/27/17 8:00 AM	12/29/17 5:00 PM		

Chapter 2

Software Requirement Specification

INTRODUCTION

This document details all the requirement specifications for android based word game. It has been written in conformance with IEEE standard document [1].

Purpose

This document describes all the requirements for *Snap Attack* (Urdu Version) game. It gives the complete description of tasks the system and the user has to perform. It includes all the functional and nonfunctional requirements. It will help the user to determine whether this product meets his/her needs or how the software must be modified according to the user needs. This document is intended for developer, and/ or project manager, tester, and documentation writer.

Scope

Snap Attack is a single player game based on Android in which player achieves highest score in two and a half minute round by building snaps on board with letters on tiles.

This project provides a user with an intellectually stimulating experience alongside indulging enjoyment. It targets the community familiar with Urdu alphabet, the people who can read/ write Urdu, who can test and/ or enhance their Urdu vocabulary.

Definitions, Acronyms, and Abbreviations

1. **Snap**: A word made with combination of letters on tiles in user's rack alongside the already made words on the board. It can be multiple words created at one time by interlacing them.
2. **Score**: Each individual tile has a score or point value on it which adds into the scoring of whole word and then the game. It is a measure of user performance.
3. **Achievement**: A set goal for the player to unlock/ achieve while playing the regular game.
4. **Round**: Pre-determined length of time in which player has to make maximum snaps/ score he can.
5. **Tray/ rack**: Framework/ container holding tiles for the user to make snaps with.
6. **Tile**: A slab with a letter from alphabet and its score

Overview

The rest of the SRS is organized in three sections: ***Overall description***, which rather includes more general information about the product than the specific requirement details to make the next section easier to understand; ***Specific requirements***, which contains all of the software requirements to a level of detail sufficient to enable designer to design a system to satisfy those requirements. These requirements include a description of every input into the system, every output from the system, and all functions performed by the system in response to an input or in support of an output; ***Supporting information****,* containing an index and an appendix.

OVERALL DESCRIPTION

This section of the SRS describes the general factors that affect the product and its requirements. It provides a background for specific requirements detailed in section three.

Product Perspective

The benefit of this application/ game is that it allows someone to play as a solitaire (stand alone) game without requiring another person to participate.

1. **User Interfaces**: All interaction with user will be via the touch screen interface of Android device. The screen resolution will be for portrait orientation only.
2. **Hardware Interfaces**: It includes the smartphone or tablet having touch screen UI. As this game is android based therefore all the android devices must meet memory requirements.
3. **Communication interfaces**: The game is standalone so no communication protocol is required. But for downloading this game from Android Play Store, internet is required. However, on later stages when initial game is complete, we may add multi-player feature which will require internet.
4. **Operations**: There are no various modes of user-initiated operations. No backup and recovery operations because of no involvement of storage information..

Product Functions

Main functions that the system is going to perform are:

1. Start the game
2. Read instructions on how to play
3. Handle game sounds
4. View locked and unlocked achievements
5. View statistics
6. Play the game
7. View snaps, score and achievements
8. Exit the game

User Characteristics

For *Snap Attack*, there is only one player who is playing. He must:

1. Know the Urdu alphabet.
2. Be familiar with Urdu language.
3. Be competent using a smartphone/ android device and in playing board games.

Constraints

Snap Attack is the Urdu version of game and people of a certain area and knowledge can play this game. Also, it is designed for android operating system only.

Assumptions and Dependencies

Initially it is assumed that it is a single player game complete in itself.

SPECIFIC REQUIREMENTS

External Interfaces

There is nothing coming in from the outside as input in this game. The dictionary is already in the app. The tiles are given to user and he forms words from those specific letters only. He can only add word from within the dictionary.

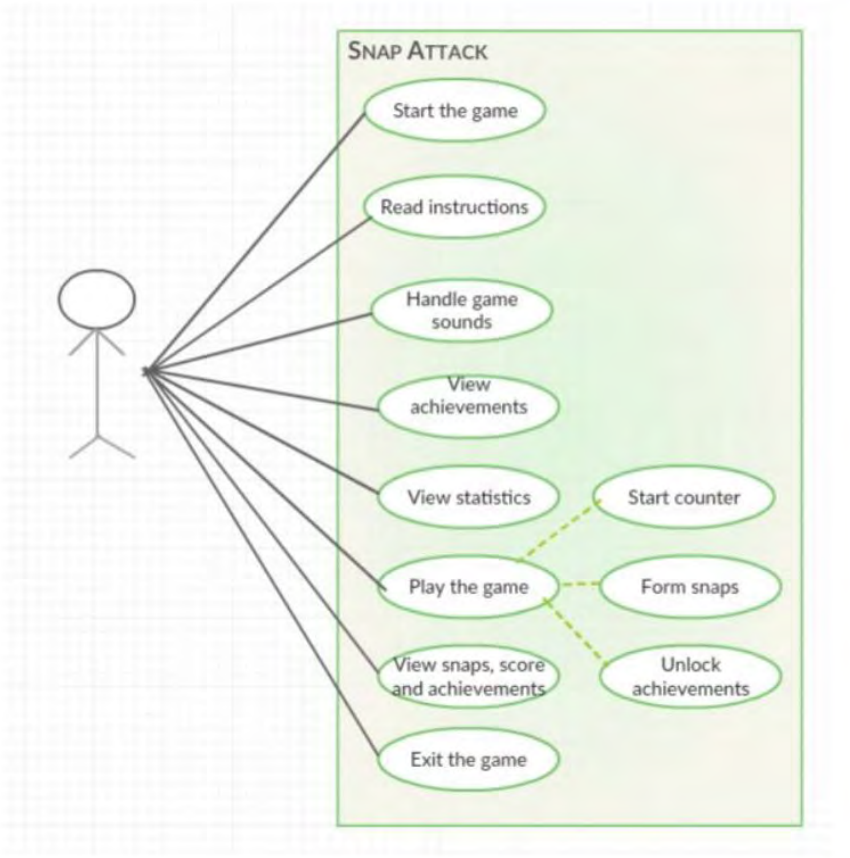
Similarly, the only output he has is the final score of the round. We can say that no as such external interfaces are involved.

Functions

Product Features Usecases, as mentioned in section 2.2 are described in detail here.

Stakeholders: The player of the game.

Usecase Diagram



UC Description:

UC1: Start the game

Primary Actor: User/ player

Pre-conditions: User has downloaded the game. His device is compatible with the features required.

Post-conditions: User has started the game. System has displayed the menu of game to the user.

Main Success Scenario:

1. User starts the game

2. System displays interface of the game that has play, statistics, achievements, help and sounds options.
3. User views the interface.

Alternate Flow:

1. At any time the system fails to respond
    - User restarts the game. The system recovers the previous state where the system failed to respond.
2. System fails to display menu on the screen
    - System displays a message to restart the game

Frequency: Whenever the user wants to play

UC2: Read instructions on how to play

Primary Actor: User/ player

Pre-conditions: User has started the game and viewed the interface.

Post-conditions: User has read the instructions on how to play the game.

Main Success Scenario:

1. User clicks help option
2. System displays instructions on screen
3. User reads the instructions

Frequency: Whenever the user wants to read instructions

UC3: Handle game sounds

Primary Actor: User/ player

Pre-conditions: User has started the game and viewed the interface.

Post-conditions: System has muted/ unmuted the sounds

Main Success Scenario:

1. User clicks sounds option
2. System displays options to mute/ unmute sounds

Frequency: Whenever the user wants to change sound option

UC4: View locked/ unlocked achievements

Primary Actor: User/ player

Pre-conditions: User has started the game and viewed the interface.

Post-conditions: System has displayed the achievements that user has achieved and those yet to achieve

Main Success Scenario:

1. User clicks achievements option
2. System displays achievements on screen
3. User views the achievements.

Frequency: Whenever the user wants to view achievements

UC5: View statistics

Primary Actor: User/ player

Pre-conditions: User has started the game and viewed the interface.

Post-conditions: System has displayed the statistics that user has reached

Main Success Scenario:

1. User clicks statistics option
2. System displays previously achieved statistics on screen
3. User views the statistics.

Frequency: Whenever the user wants to view statistics

UC6: Play the game

Primary Actor: User/ player

Pre-conditions: User has started the game and viewed the interface.

Post-conditions: System has displayed the board with a random word from dictionary on it. User has seven random letters from the alphabet to play the game. User has made a snap. System has marked the score for the snap and calculated total score of snaps formed until now. System has marked achievement unlocked (if any).

Main Success Scenario:

1. User clicks play option
2. System displays the fixed size board
3. System places a random word from dictionary on board to start the game
4. System fills user tray with seven random letters from alphabet
5. User views the word on board and the letters in tray/ rack
6. System starts time count for the round
7. User starts making snaps
8. System validates the word's existence in dictionary
9. System calculates the snap score
10. System adds the individual snap score to the total score of the game round
11. System checks if the snap made matches any of the pre-defined achievements to unlock
12. Steps mentioned above repeat until the time for the round is over

Alternate Flow:

1. System fails to place a word on board or letters in tray
    - System displays error message and asks user to replay the game
    - User clicks play option again
    - System recovers and repeat steps in main success scenario
2. User's snap is not a valid dictionary word and/ or user has made that snap already
    - System denies the snap
3. If player wants to play with computer **function 6.3**
    - User clicks *play against computer* option
    - User repeats steps from two to seven
4. If player wants to play against human player online **function 6.4**
    - User clicks *play online* option
    - User plays against other players in real-time by repeating steps two to seven

Frequency: Whenever the user wants to play the game

UC7: View snaps, score and achievements

Primary Actor: User/ player

Pre-conditions: User has played the game and finished the round

Post-conditions: System has displayed all the snaps user formed during game along with score.

Main Success Scenario:

1. User forms the last snap
2. System displays all the snaps formed in game round with scores and the total score
3. System shows any achievements that have been unlocked during the game round
4. User views the results

Alternate Flow:

1. If game was multiplayer function [function 7.1]
    - User also views best snaps made by other player(s)

Frequency: Once at the end of every game played

UC8: Exit the game

Primary Actor: User/ player

Pre-conditions: User has started the game. System shows the main interface of the game.

Post-conditions: User exited the game

Main Success Scenario:

1. User clicks the exit option
2. System allows user to exit the game

Alternate Flow: -

Frequency: Whenever the user wants to leave the game

Function Definitions: The complex system functions involved at the backend are detailed below with usecase number in which they appeared

F6.1: Placing initial word on the board/ grid: System chooses a random word from dictionary to place initially on board to start the game. This word must be smaller or equal to the board/ grid size. It can also place two words instead of one. Those words must have a connected letter between them. An algorithm is needed to match words.

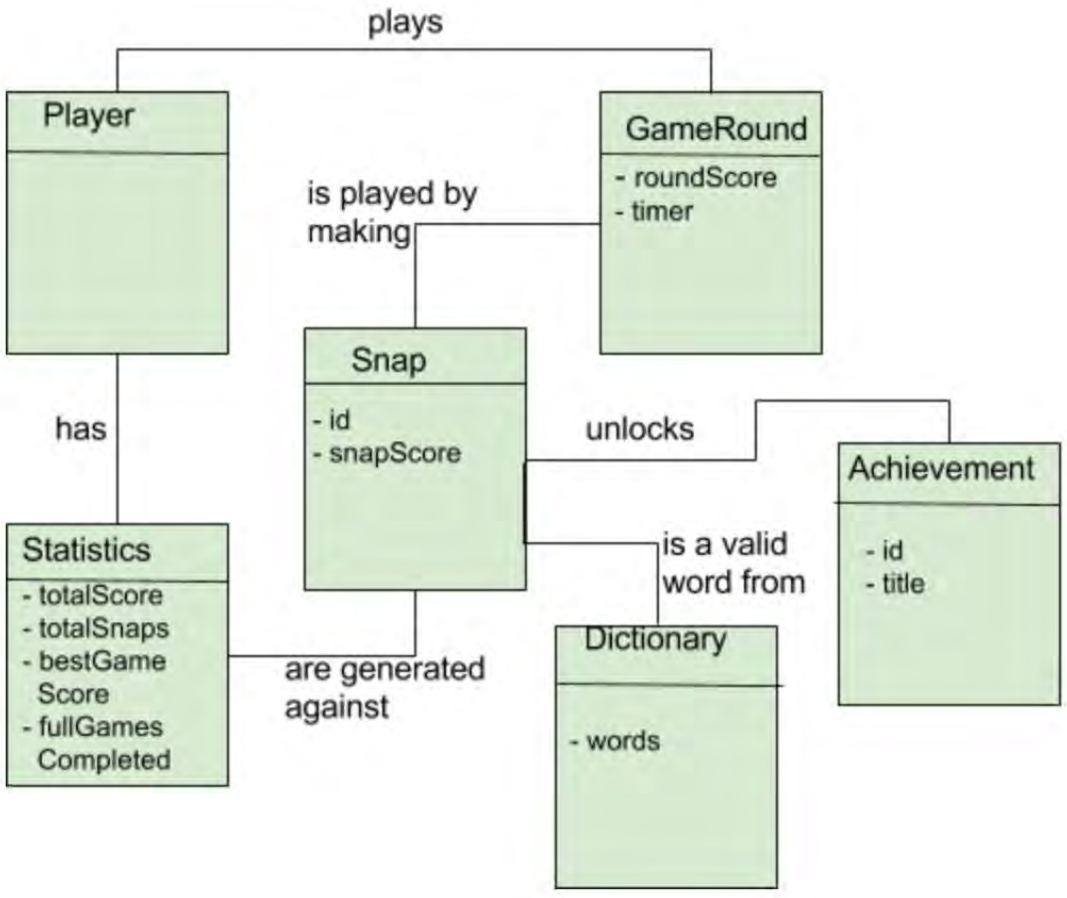
F6.2: Setting player's rack of tiles: System selects seven random letters of the alphabet for user to form snaps with. But, those letters must be selected carefully because unlike English letters, there are no vowels in Urdu and word formation is different too.

F6.3: Playing against computer: When a user is playing against computer/ system, AI algorithms are involved. Computer has to intelligently select words and put them on board.

F6.4: Playing against a human player online: This system feature is deployed on client-server architecture in second iteration.

F7.1: Viewing score in multiplayer game: User views end-scores and best snaps of other user he's been playing the round with.

Domain Model



Performance Requirements

Static numerical requirements include:

1. **Terminals**: Only one terminal device is required
2. **Simultaneous users**: Only one user is required to play
3. **Amount and type of information handled**: -

There are no dynamic numerical requirements.

Logical Database Requirements

There is no database involved in this project. Few information files are required such as:

1. File for Urdu dictionary
    - Types of information: It contains all the valid words of Urdu dictionary
    - Frequency of use: Every time a word is placed on board or user makes a snap
2. File for storing user statistics
    - Types of information: It maintains user stats like average snap score, total score so far, number of games played.
    - Frequency of use: Every time player plays the game
3. File for game achievements
    - Types of information: It contains all the locked and unlocked achievements
    - Frequency of use: Every time player plays the game

Software System Attributes

Following are some system attributes that may serve as requirements and can be verified objectively.

1. **Reliability**: As no user data is being stored or accessed, reliability has less or no say. Although, *Snap Attack* should never crash or hang.
2. **Availability**: The game must be available 24/7 until/ unless user uninstalls it. Other than that, there are no checkpoints, recovery and restart because no database or storage is involved.
3. **Security**: No user data shall be accessed. No system files shall be altered.
4. **Maintainability**: The code shall be kept modular permitting future modification. The interfaces should be kept separate from the logic so they can be changed without changing actual code.
5. **Portability**: *Snap Attack* shall be portable to any android system.

Chapter 3

Software Design Description

INTRODUCTION

This chapter describes software designs and establishes the information content and organization of a software design description (SDD). It has been written in conformance with IEEE standard document for design description [2].

Purpose

The purpose of this Software Design Document is to provide a description of the design of *Snap Attack* game to allow for software development to proceed with an understanding of what is to be built and how it is expected to build. It provides necessary information for the software to be built.

Design Overview

This document is intended for technical and managerial stakeholder(s) for an overall guidance to the software project. It guides a designer in the selection, organization, and presentation of design information. It helps to ensure that design descriptions are complete, concise, consistent, well organized, and easy to communicate.

Requirements Traceability Matrix

RTM is a document that connects requirements throughout the validation process. It is a tool for the validation team to ensure that requirements are not lost during the validation.

Project Name: Snap Attack								
Project Description: It is a word based android game in Urdu.								
TC\Req	**Req 1**	**Req 2**	**Req 3**	**Req 4**	**Req 5**	**Req 6**	**Req 7**	**Req 8**
Case 1	x					x	x	
Case 2	x	x	x					
Case 3	x					x	x	
Case 4	x			x	x		x	

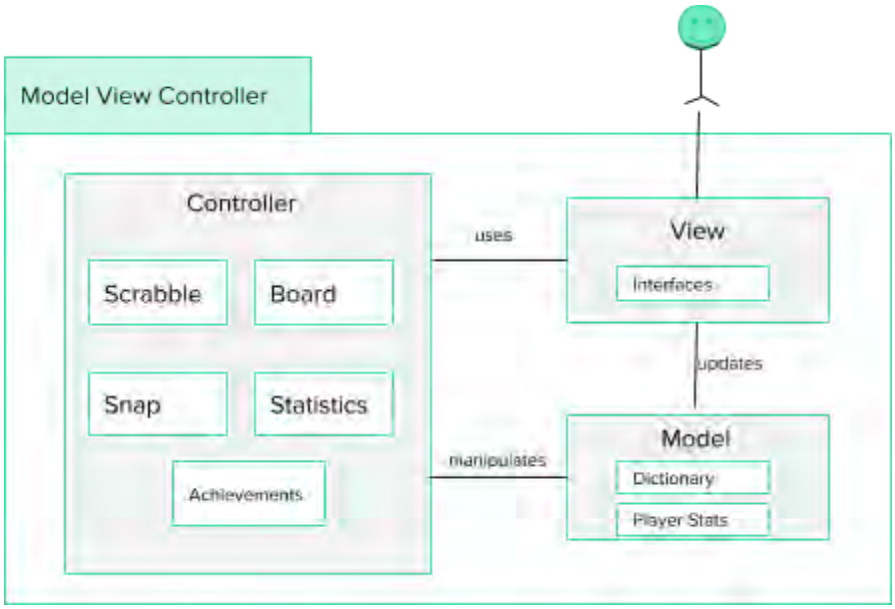
SYSTEM ARCHITECTURAL DESIGN

A system architecture is the conceptual model that defines the structure, behavior, and views of a system. It is a formal description and representation of a system components that will work together to implement the overall system.

Chosen System Architecture

The architecture I chose for my software is Model View Controller (MVC). The MVC pattern, in a nutshell, is this:

- The **model** represents the data, and does nothing else. The model does not depend on the controller or the view.
- The **view** displays the model data, and sends user actions (e.g. button clicks) to the controller. The view can:
    - be independent of both the model and the controller; or
    - actually be the controller, and therefore depend on the model.
- The **controller** provides model data to the view, and interprets user actions such as button clicks. The controller depends on the view and the model.



Discussion of Alternative Designs

The other alternatives present alongside MVC are:

- Presentation Abstraction Control
- Model View Presenter
- Model View ViewModel

These design patterns are not very different from MVC but are rather derived from MVC or are variations of it. They are not either/or choices in the first place. They have their differences in implementation on client/server sides. MVVM provides rich interactivity on both client and server sides. I only have the client side and another reason why I used MVC is that it is more general and it:

- makes *model classes* reusable without modification.
- makes the *view* reusable without modification.

Implementing new features and maintenance is easier in MVC.

DETAILED DESCRIPTION OF COMPONENTS

Here is the detailed description of components of system architecture.

Components

Player: The user of the game is the player. He plays the game by forming snaps to achieve highest score.

Achievements: The personal goals for player to achieve while playing the game.

Statistics: The data obtained from all the games user has played.

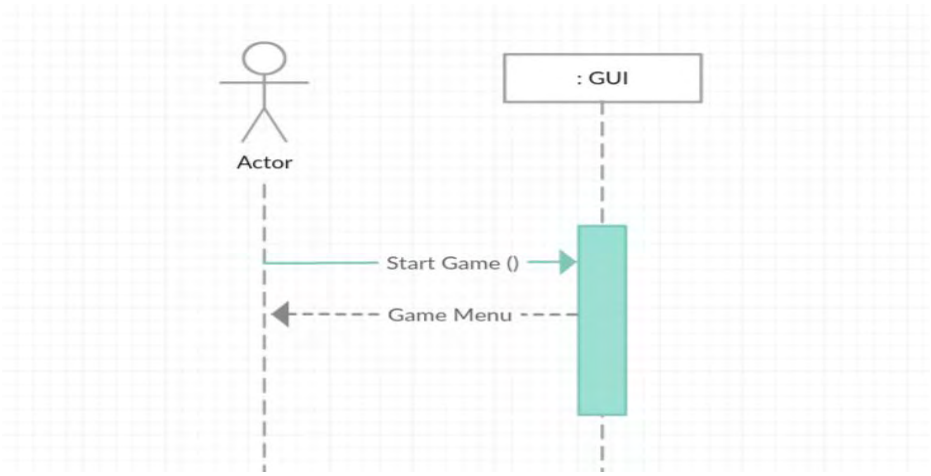
Game Round: One complete game is one round of the game. Player plays a round for one and a half minute and forms maximum snaps as he can.

Snaps: The words user makes with the given alphabet tiles are snaps. There can be multiple connected words in one snap.

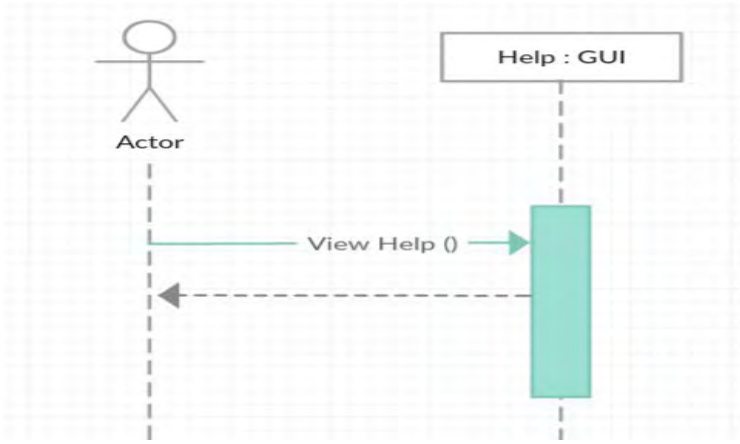
Sequence Diagrams

Following sequence diagrams depict the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the system.

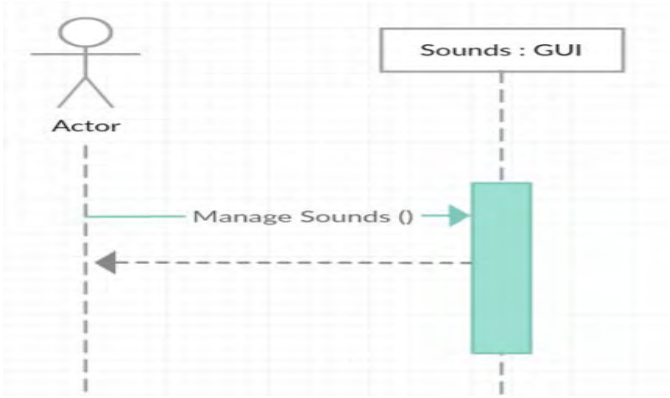
Start the game: Player clicks on the game icon to start the game. User Interface for main menu is displayed.



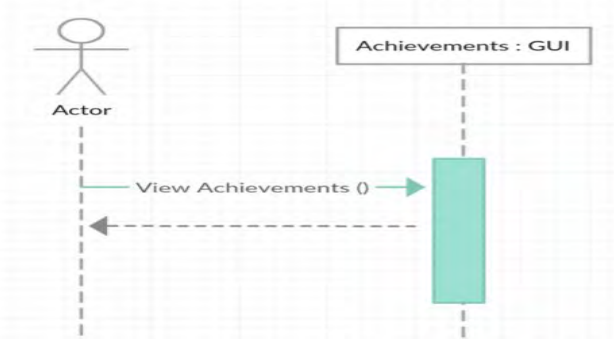
Read instructions on how to play: Player selects “Help” option from main menu. The pre-conditions and post-conditions remain same as described in usecase description.



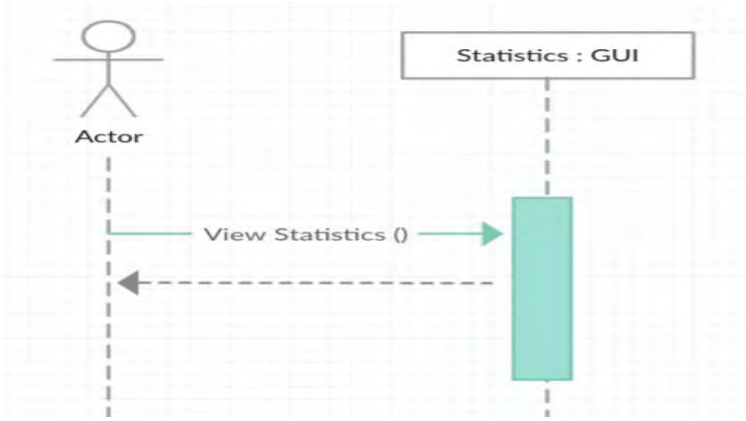
Handle game sounds: Player selects “Options” menu from main menu to on/off sounds.The pre-conditions and post-conditions remain same as described in usecase description.



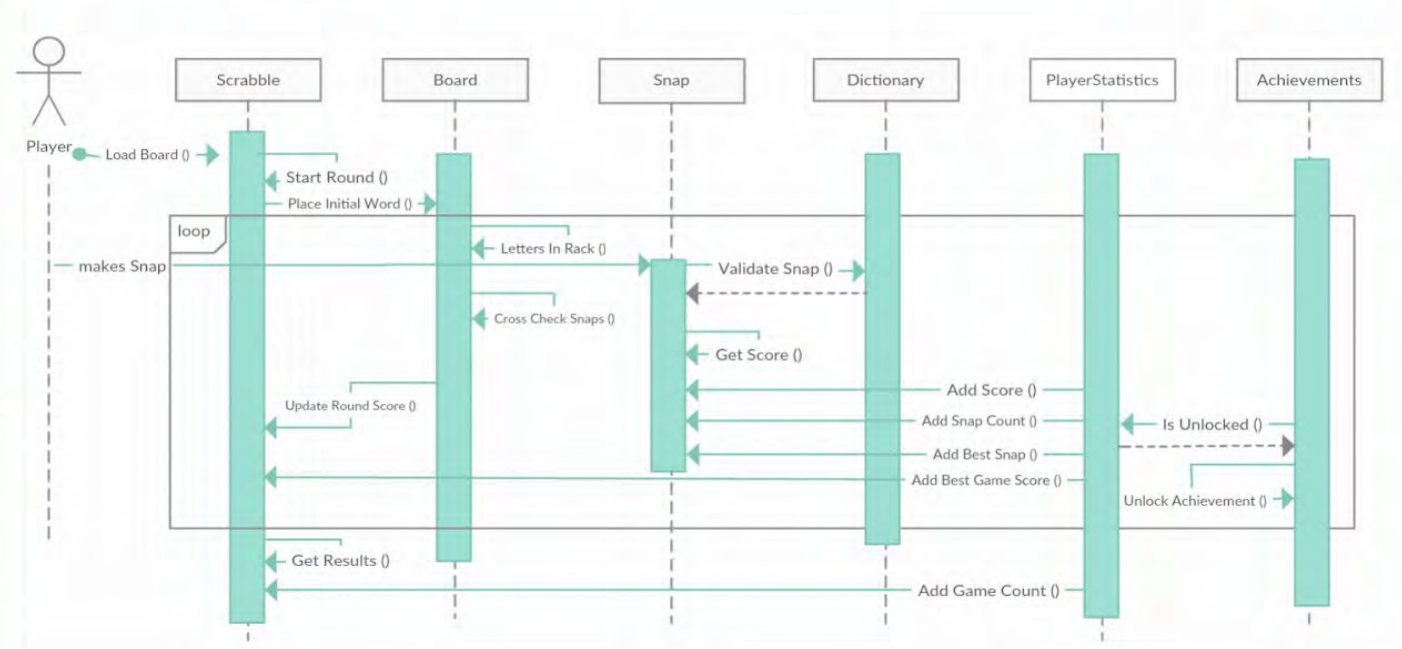
View locked and unlocked achievements: Player selects "Achievements" option from main menu to view all the locked and unlocked achievements. The pre-conditions and post-conditions remain same as described in usecase description.



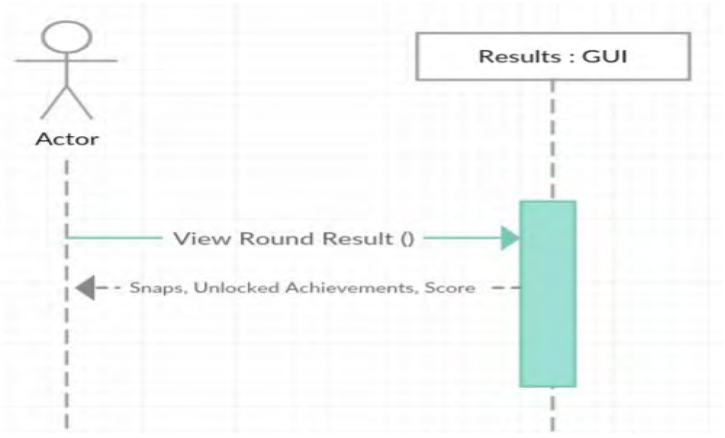
View statistics: Player selects "Statistics" option from main menu to view user statistics. The pre-conditions and post-conditions remain same as described in usecase description.



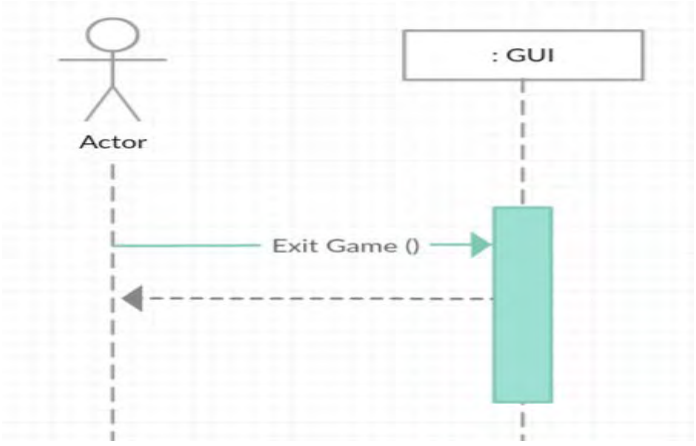
Play the game: Player starts the timed game round**.** The pre-conditions and post-conditions remain same as described in usecase description.



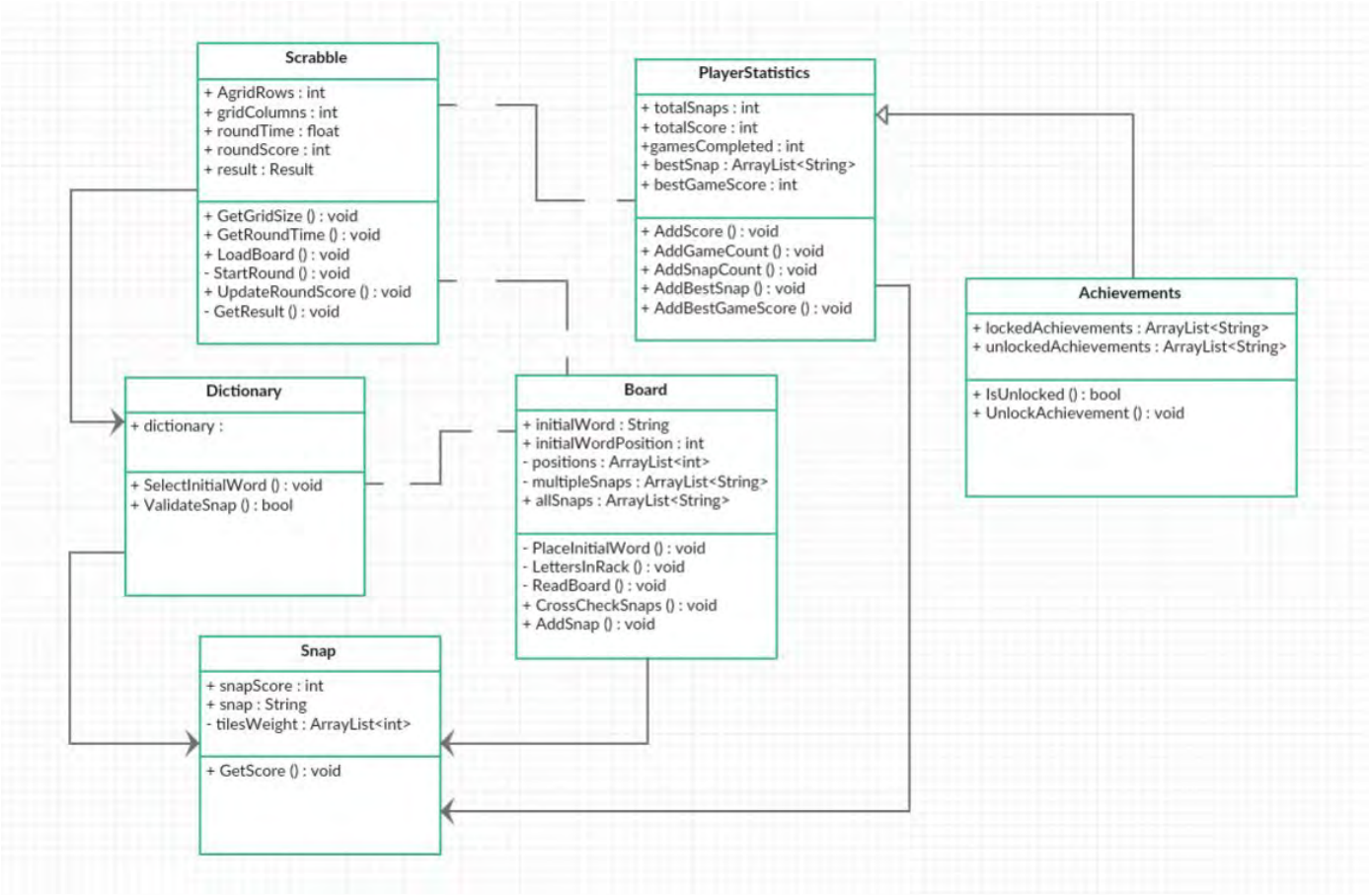
View Result (snaps, score and achievements) at the end of game round: Player played the game. His results have been calculated at the end of round (as shown in "Play the game" diagram). He views results at the end of the round for 30 seconds and returns to main menu of the game.



Exit the game: Player wants to quit the game. He selects “Exit” option from the main menu. The pre-conditions and post-conditions remain same as described in usecase description.



Class Diagram



USER INTERFACE DESIGN

Description of the User Interface

Screen Images

There are six screens (in horizontal sequence):

1. When the game just started after clicking the game icon. Main menu is loading
2. When the main menu loaded. There's play option and general menus displayed
3. Screen 3 & 4 show different menu option on the main menu
4. When the user clicked "play" and is in the middle of round
5. Last screen shows the results at the end of a game round

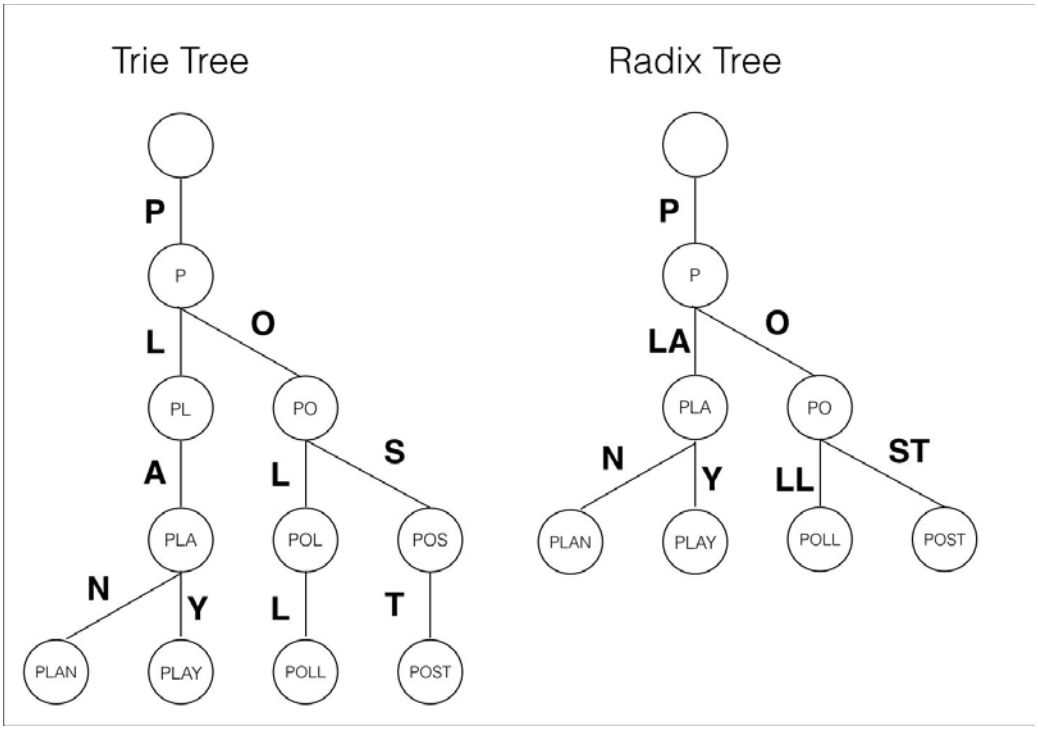

snap
ATTACK
کھیلیں
اعداد و شمار
کل سکور
12300
کل الفاظ
300
مدد + اختیارات
چیک ہوائیں
کچھ میرے متعلق
کھیلیں
اعداد و شمار
اوازیں
قوانین:
مدد + اختیارات
چیک ہوائیں
کچھ میرے متعلق
کھیلیں
Snap Attack
This game was developed as a final year project by a BS CS student. blah blah
اعداد و شمار
مدد + اختیارات
چیک ہوائیں
کچھ میرے متعلق
26 points
5 snaps
1 : 49
38 points
3 snaps
ایک لفظ
دو الفاظ

ADDITIONAL MATERIAL

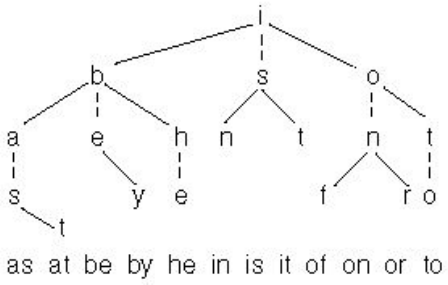
Data Structure for Dictionary

One of the main tasks being done in *Snap Attack* is referring to dictionary for validation of words that the player makes while playing the game. If a player makes 30 words in one round, we need to consider dictionary 30 times (or even more in case of cross words). An efficient data structure is needed. Few data structures that I found are hash, trie, radix tree, and ternary tree. I compared them mainly in terms of space/memory and then efficiency.

- **Hash** is simplest of them but is slow and occupies more space than other mentioned.
- **Trie** is used to store a dynamic set or associative array where the keys are usually strings.
    - It inserts/traverses in alphabetical order. Worst case is efficient than hash O(m).
    - It is space inefficient.
    - Traversal speed doesn't depend upon # of words but rather on # of letters in a word.
    - Needs encoding/decoding.
    - Good in representing dictionaries
- **Radix Tree** (compact prefix tree) represents a space-optimized trie in which each node that is the only child is merged with its parent.



- **Ternary Tree** (prefix tree) is a type of trie where nodes are arranged in a manner similar to a binary search tree, but with up to three children.
    - Space efficient than trie and hash
    - Good in spell checking and auto-correction



Urdu Letter Frequency

The letter tiles in *Snap Attack* need to be assigned some weights for scoring the words in game. Just like scrabble, or scrabble-like games, letter frequency is important in assigning weights to alphabet. The most frequent letters are less in points than those that appear rarely.

This is the work done on November 4 2012 by *Denny Vrandečić* (http://denny.vrandecic.de/) and his team at Google. They took the text of 262 language editions of Wikipedia and counted which letters appear how often.

6	ظ	5	ٹ	4	ف	3	ں	2	ہ	1	ا
6	ڑ	5	ط	4	ش	3	ج	2	ے	1	ی
6	غ	5	چ	4	ح	3	ع	2	ت	1	ر
6	ث	5	آ	4	ئ	3	پ	2	ل	1	ک
6	ذ	5	ڈ	4	ز	3	ھ	2	س	1	و
6	ي	5	ض	4	خ	3	ق	2	ب	1	م
10	ژ	5	ء	4	ص	3	گ	2	د	1	ن

Algorithm for the game

The game on the front end may not seem to be doing a lot but it surely does have much more going on at the back end. Following is the algorithm describing the back-end working of game when user selects the "Play" option from the main menu.

For loading the initial grid for the board

- Select a random word from dictionary of length less than or equal to number of cells in a row or column.
- Select a random direction (horizontal or vertical) position for the initial word to be placed on the board. It should be dependent upon the word length. For example, if word length is 5 and grid size is

User places word on the board without attaching it to the initial word

- After filling user rack with seven random letters, user is free to make as many words in game as he wants till the end of the round.
- He has to use at least one tile in his own word from the initial word.
- If he does not connect to the tile/tiles of the initial word, his word is not considered for validation and scoring. It is considered an illegal word.

User forms a legal snap

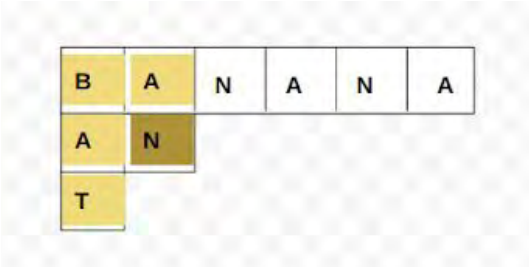
- Any word that is not validated from the dictionary or is not attached to the initial word or any word that is already placed on board, or has already been made previously is an illegal snap.
- That validated word is assigned with a score on the basis of the weights of the individual letters in the word.
- The information of the snap (word + score) is stored in data structure for current round.
- When user makes the same word in same game, he is prompted that this word already exists and score is not calculated again.

User forms a crossword (double word)

- In the game, if user makes one legal word and he is still left with some tiles in the rack, he can make another word with same conditions to be legal with those tiles. It is a crossword and its score is calculated as sum of all words made without removing any already placed tiles.
- The crossword's neighboring positions (right, left, up, down) are checked. If there are letters in the neighboring cells, another word is created with those neighbors and the program scans the grid for the neighbor's neighbors to reach the first letter of each new word formed. All

of those words go through same validation process as described above. If they're validated, mutual score for all the connected words is calculated.

For example, in the following figure, if BANANA is the initial word placed on grid and user makes BAT and AN, he gets points for BAT, AN and AN. AN is the crossword that is validated from both sides.



Chapter 4

Software Implementation

INTRODUCTION

At implementation stage, algorithms and the technical specifications are changed in to the program. Snippets from the code are shared to go through application development.

Framework Selection

Android Studio was used for development. It contains rich layout editor that allows you to drag-and-drop UI components, preview layouts on multiple screen configurations, and much more.

Unity was easier to use than Android Studio but the vast learning platform that Android Studio provides and make us meet the requirements can't be compared.

Language Selection

Two languages have been used to implement *Snap Attack*: Java and XML.

Android Studio allows the use of these two languages. The layouts have been implemented in xml and logic has been designed in Java.

APPLICATION DEVELOPMENT

Timer

```
//for timer
final TextView time = (TextView) findViewById(R.id.timer);
new CountDownTimer(15000, 1000) {

    public void onTick(long millisUntilFinished) {
        time.setText(millisUntilFinished / 1000 + " Seconds");
    }

    public void onFinish() {
        new AlertDialog.Builder(MainActivity.this)
                .setTitle("Time Over")
                .setMessage("Do you want to see the Result?")
                .setNegativeButton(android.R.string.no, (arg0, arg1) → {
                        MainActivity.super.finish();
                })
                .setPositiveButton(android.R.string.yes, (arg0, arg1) → {
                        Intent login = new Intent(MainActivity.this, ResultActivity.class);
                        startActivity(login);
                        finish();
                }).create().show();
    }
}.start();
```

Mapping Letters with HashMap

```
//Mapping Urdu Alphabet to keys for random alphabet selection
Map<Integer, String> m1 = new HashMap<Integer, String>();
m1.put(0, "ا") ;m1.put(1, "آ"); m1.put(2, "ب"); m1.put(3, "پ"); m1.put(4, "ت");
m1.put(5, "ٹ") ;m1.put(6, "ث"); m1.put(7, "ج"); m1.put(8, "چ"); m1.put(9, "ح");
m1.put(10, "خ");m1.put(11, "د");m1.put(12, "ڈ");m1.put(13, "ذ");m1.put(14, "ر");
m1.put(15, "ڑ");m1.put(16, "ز");m1.put(17, "ژ");m1.put(18, "س");m1.put(19, "ش");
m1.put(20, "ص");m1.put(21, "ض");m1.put(22, "ط");m1.put(23, "ظ");m1.put(24, "ع");
m1.put(25, "غ");m1.put(26, "ف");m1.put(27, "ق");m1.put(28, "ک");m1.put(29, "گ");
m1.put(30, "ل");m1.put(31, "م");m1.put(32, "ن");m1.put(33, "ں");m1.put(34, "و");
m1.put(35, "ہ");m1.put(36, "ھ");m1.put(37, "ی");m1.put(38, "ے");m1.put(39, "ئ");
m1.put(40, "ء");m1.put(41, "ي");
```

Random Words

```
Random rand = new Random();
int rand_idx;
String rand_word = null;
for (int i = 0; i < split.length; i++) {
    rand_idx = rand.nextInt(split.length);
    if (split[rand_idx].length() <= 8) {
        rand_word = new String(split[rand_idx]);
        break;
    }
}
System.out.println("String: " + rand_word);
```

Letter Score

```
public int letterScore(char c){
    if (c == 'ا' || c == 'ی' || c == 'ر' || c == 'ک' || c == 'و' || c == 'م' || c == 'ن')
        return 1;
    else if (c == 'ے' || c == 'ت' || c == 'ل' || c == 'ہ' || c == 'س' || c == 'ب' || c == 'د')
        return 2;
    else if (c == 'ں' || c == 'ج' || c == 'ع' || c == 'پ' || c == 'ھ' || c == 'ق' || c == 'گ')
        return 3;
    else if (c == 'ئ' || c == 'ف' || c == 'ش' || c == 'ح' || c == 'ز' || c == 'ص' || c == 'خ')
        return 4;
    else if (c == 'ٹ' || c == 'ط' || c == 'آ' || c == 'چ' || c == 'ڈ' || c == 'ض' || c == 'ء')
        return 5;
    else if (c == 'ي' || c == 'ث' || c == 'ذ' || c == 'غ' || c == 'ظ' || c == 'ڑ')
        return 6;
    else if (c == 'ژ')
        return 10;
    return 0; //default
}
```

Random Placement

```
int pos_x, pos_y;
boolean dir_h = rand.nextBoolean();
System.out.println("direction: " + dir_h);
int str_idx = 0;

if (dir_h) {//place horizontally
    pos_x = rand.nextInt((8 - rand_word.length()) + 1);
    pos_y = rand.nextInt(8);
    System.out.println("pos x: " + pos_x);
    System.out.println("pos y: " + pos_y);
    for (int y = pos_x; y < (pos_x + rand_word.length()); y++){
        grid[pos_x][y].setText((reverse.charAt(str_idx) + ""));
        str_idx++;
    }

}
else { //place vertically
    str_idx = 0;
    pos_y = rand.nextInt((8 - rand_word.length()) + 1);
    pos_x = rand.nextInt(8);
    System.out.println("pos x: " + pos_x);
    System.out.println("pos y: " + pos_y);
    for (int x = pos_y ; x < (pos_y + rand_word.length()); x++){
        grid[x][pos_y].setText((rand_word.charAt(str_idx) + ""));
        str_idx++;
    }
}
```

Result

Word Adapter

```
public class WordsAdapter extends ArrayAdapter<ResultWords> {

    private Context mContext;
    private List<ResultWords> wordsList = new ArrayList<>();

    public WordsAdapter(@NonNull Context context, ArrayList<ResultWords> list) {
        super(context, 0, list);
        mContext = context;
        wordsList = list;
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
        View listItem = convertView;
        if(listItem == null)
            listItem = LayoutInflater.from(mContext).inflate(R.layout.list_word,parent,false);

        ResultWords currentWord = wordsList.get(position);


        TextView word = (TextView) listItem.findViewById(R.id.word);
        word.setText(currentWord.getWord());

        TextView score = (TextView) listItem.findViewById(R.id.score);
        score.setText(currentWord.getScore());

        return listItem;
    }
}
```

Result Word

```
public class ResultWords {

    // Store the id of the  movie poster
    private int score;
    // Store the name of the movie
    private String word;


    // Constructor that is used to create an instance of the Movie object
    public ResultWords(String word, int score) {
        this.word = word;
        this.score = score;
    }


    public String getWord() { return word; }

    public void setWord(String word) { this.word = word; }
    public int getScore() { return score; }

    public void setScore(int score) { this.score = score; }

}
```

Chapter 5

Software Testing and Evaluation

INTRODUCTION

A test case is a set of conditions under which a tester will determine whether a software system or its features are working as it was originally established for it to do.

System Overview

When the game starts, a board/grid is presented with a valid dictionary word placed on it. User is given seven random tiles of letters of the Urdu alphabet with different score points. Player selects tiles and places them on board to form a new word. The individual score of letters combine to make word score. The player keep making words with those given seven tiles until the round is over. At the end of the gameplay, user views all the snaps he made and the total score he achieved.

Test Approach

Tests were done to measure *Correctness* and to detect any *Defects* that arise due to any input which causes product to fail.

Among the multiple types of testing, *Unit Testing* was done to ensure the correctness of code and whether the interfaces and the business logic was perfectly compatible with each other or not.

User acceptance testing (UAT) also called *beta testing* or *end user testing*, that consists of a process of verifying that a solution works for the user, was done by random players. They were given the phone with installed app and they played which also tested usability of the application. It was tested on multiple devices.

TEST PLAN

Features to be Tested

We test the functional and non-functional requirements using test cases. Some functional requirements are:

- Play the game
- Manage game sounds
- Form snap and calculate score
- View results

Features not to be Tested

Features not to be tested from a developer's point of view are:

- Rate at which the board is loaded and rack is filled
- Power used by processor
- Memory consumed by the game
- Maintainability of game

These features may be tested later on but are ignored at the moment.

Testing Tools and Environment

Testing was done manually. No specific tools and environment was required because code was tested by the programmer herself and the game play is a beta/user-end test. All a user need is an Android device with game installed.

TEST CASES

Following test cases list the functional requirements tested through beta testing. The *errors encountered* part covers the unit testing part.

Case 1

Purpose: Play the game

Setup: Install game on device

Instructions: In the main menu, select "play" option

Expected result: The timer will start. The board will load. A word will be placed on it already. The user tile rack will be filled with tiles.

Errors encountered: On testcase run, following errors were encountered:

- *CountdownTimer* started fine, but didn't end when the activity was interrupted/ended before the round ended. The timer went off only when countdown was completed.
- *Words* placed on board sometimes had spaces in them because of disjoining Urdu characters.
- *Tiles* were loaded with random letters but sometimes a letter appeared multiple times or the letters were so unique that user was unable to make words with them.

Final Verdict: True

Case 2

Purpose: Manage game sounds

Setup: Install game on device

Instructions: In the main menu, select “Help and Options” menu.

There will be a toggle icon in front of the sound option.

It is ON by default. Toggle it to OFF.

Expected Result: The sounds should stop right away.

Verdict: True

Case 3

Purpose: Form snap and calculate score

Setup: Open game. From the main menu select “Play” option. The game board will load.

Instructions: From the tiles given below, select the letters and place on the board to form new words.

Expected Result: The snap will be validated from dictionary. If it is a valid word, it will remain on the board as long as user doesn't remove any letters. Score for the word will be calculated. If it is an invalid word or it has been formed already, the tiles will go back to the rack.

Errors encountered: This is where the main algorithm worked. There were many options tested to make a word and verify it. First, user clicked okay after making a word to signal app to search it in dictionary and assign a score to it.. But it wasn’t efficient.

Final Verdict: True

Case 4

Purpose: View Results

Setup: Open game. From the main menu select play option.

Instructions: Play the game round. Form words with tiles for the specified time.

Expected Result: When the timer reaches the limit, the game round ends. It shows all the words/snaps formed during game along with their scores and the total game round score.

Verdict: True

CONCLUSION

If you read the document this far, you'd know all about how the game works. User is able to play the game, make words and achieve new personal target score everytime.

There are numerous future enhancements in the project to make application more interesting and fun to play. Few suggestions in that regard can be:

- Allow user to shuffle/reselect tiles during the game if he's unable to make any word with given letters.
- Choose *"word of the day"* from dictionary for user to learn new word everyday.
- The app suggests the maximum scoring word at the end of round.
- Allow multiple users to play it against each other online. Or app plays against user.
- Introduce difficulty levels (easy, medium, hard).

Similarly, a lot of things can be done to the UI. The target is to make user like the application environment and make it as easy or as challenging to play as the user wants.

REFERENCES

1. *IEEE Recommended Practice for Software Requirements Specifications*, IEEE Standard 830-1998 (Revision of IEEE Std 830-1993), reaffirmed 2009.
2. *IEEE Standard for Information Technology—Systems Design— Software Design Descriptions*, IEEE Standard 1016 ™ -2009.
3. Roger S. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 7th Edition, 2010.
4. Craig Larman, *Applying UML and Patterns: AN Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall, 2nd Edition, 2001.