Game Based Learning

of

Sorting Algorithms

Software Engineering Lab (CS243)

January – April (2017)

Software Requirements Specification Document

[First Draft]

<u>Instructor</u>

Dr. Samit Bhattacharya

CSE Department, IITG

Group Number: 2

<u>Project Team Members</u>

150101003 : Abhishek Kumar

150101045 : Patoliya Meetkumar Krushnadas

150101054 : Saket Sanjay Agrawal

Table of Contents

1] INTRODUCTION:

In this virtual game based learning system, students learn through a gaming environment. The application is a fully mobile based application. The system provides touch based user interface which will guide student to learn different sorting algorithms. Unity which is used for creating simple graphics is used to assist us in creating the application

1.1] PURPOSE:

The main purpose of this document is to serve as a binding agreement regarding the requirements of the project. The system provides touch based user interface which will guide student to learn different sorting algorithms. The student can play and learn during the lecture or outside of the college hours. A teacher also can use this application as a supplementary for teaching sorting algorithm.

1.2] SCOPE:

This document contains a complete description of the functionality of the project. It consists of use cases, functional requirements and nonfunctional requirements, which, taken together form a complete description of the software.

1.3] SYSTEM OVERVIEW:

- A 3D game engine, Unity, shall be used to create 3D cubes on which numbers are written.
- Initial interface will contain menu like play game, restart, quit, user helps etc. Rest part of the screen should contain the rules for playing the game.
- Under the **play the sorting game** player will select any sorting from the list (The app should contain at least 3 options in the menu for 3 different shorting algorithms).

- Based on the selection of the sorting algorithm from the menu, the algorithm in pseudo code should be displayed first. Student should learn the working principle of the algorithm from there.

- After pressing the "Start Playing" button, some 3D cubes labeled with some 2-digit numbers in unsorted order will appear on the screen. The cubes on the screen are swappable. The student should swap the cubes correctly as per his/her understanding of the algorithm.

- Every correct swap adds a score of +10 and every wrong swap -10. If a student performs three conjugative incorrect swaps, a message will be displayed requesting to restart the game.

- If a player successfully performs all swaps, it will report the score as per total correct and incorrect swaps and time taken to complete the sorting.

- The system also generates game statistic report that include name of the player, max score (without mistake), number of restarting the game etc.

2] FUNCTIONAL REQUIREMENTS:

2.1] SORTING:

2.1.1) Bubble sort:

Input: Unsorted array of numbers.

Output: Sorted array.

Output array description:

Smallest → Biggest from Left → Right

Description: Repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order.

Sub-functions used:

2.1.1.1) Comparison:

Input: Two numbers.

Output: Smallest number among two.

Description: Compare numbers and return smallest or biggest number according to our need.

2.1.1.2) Swap:

Input: Two numbers in particular order.

Output: Two numbers in opposite order.

Description: Initial order (a → b).

Order after swapping (b → a).

2.1.2) Insertion sort:

Input: Unsorted array of numbers.

Output: Sorted array.

Output array description:

Smallest → Biggest from Left → Right

Description: Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list.

Each iteration, insertion sort removes one element from the input data, **finds the location** it belongs within the sorted list, and inserts it there. It repeats until no input elements remain**.**

All elements up to that position **shifts** right.

Sub-functions used:

2.1.2.1) Finding location:

Input: Array and index.

Output: Index of the location where the element of input index can be inserted.

Description: Finds such a location for input index that resulting subarray is sorted when element at input index can be inserted at located index.

2.1.2.2) Shifting right:

Input: Array and index.

Output: Array with all entries shifted to right by one place after that index.

2.1.3) Selection sort:

Input: Unsorted array of numbers.

Output: Sorted array.

Output array description:

Smallest → Biggest from Left → Right

Description: The algorithm proceeds by **finding the smallest** (or largest, depending on sorting order) element in the unsorted sub-list, exchanging (**swapping**) it with the leftmost unsorted element (putting it in sorted order), and moving the sub-list boundaries one element to the right.

2.1.3.1) Linear search:

Input: Array of numbers.

Output: Target value within that array.

Description: In this case target value is smallest value and can be found by checking the list sequentially.

2.2] SCOREBOARD:

Input: Move by user.

Output: Change in score.

+ 10 if move is allowed by algorithm.

- 10 if move is not allowed by algorithm.

2.3] PAUSE AND RESUME:

Input: Signal by user (Pause/resume).

Output: Timeframe will stop during pause period and start while resuming.

2.4] RESTART:

Input: Signal by user.

Output: Fresh game will start with timeframe starting from zero and zero scores as well.

2.5] LIFELINE:

Input: Signal by user.

Output: Same game will continue without disturbing score and timeframe.

Description: Available after user plays three wrong moves.

2.6] HELP:

Input: Signal from user.

Output: Set of instructions to help user to play the game.

2.7] QUIT:

Input: Signal from user.

Output: Terminate the game.

2.8] MUSIC:

Input: Signal from user.

Output: Audio will be enabled or disabled based on input.

2.9] LEADER BOARD:

Input: Game scores.

Output: Top ten best scores.

Description: Different levels will have different leader boards.

3] NON-FUNCTIONAL REQUIREMENTS:

3.1] USABILITY:

3.1.1] Contextual enquiry:

3.1.1.1] The user segment targeted app prefer to learn this three sorting algorithms.

3.1.1.2] The music option is favored by both the segments of users who prefer or do not prefer music in the app.

3.1.1.3] Leaderboard tempts user to improve their performance making them more fluent in the sorting algorithm.

3.1.1.4] This app is made for android/ ios/ windows platform targeting all the smartphone users.

3.1.1.5] Levels inclusion helps target users learn from best case to worst case performance of each sorting algorithm.

3.1.2] Simplicity: Simple to use. Launch the app → Select one of three sort → Start solving and learning.

3.1.3] Accessibility: For android users, this app will be available at play store and IOS app store for IOS users.

3.1.4] Relevancy:

This sorting based learning app **can be used on various operating system** as this app is developed in unity. Some of the popular OS on which this app can be run are:

1) Android
2) IOS
3) Microsoft

3.2] SOFTWARE REQUIREMENTS:

3.2.1] Unity Game Engine.

3.2.2] Android studio.

3.2.3] Visual Studio

3.3] EXTENSIBILITY:

This app can further be modified to include various other sorting algorithms.

3.3.1] Merge Sort
3.3.2] Quick Sort
3.3.3] Bucket Sort
3.3.4] Radix sort

3.4] MODIFICATION:

3.4.1] Number of levels can be increased.

3.4.2] Complexity of hardness can be modified at each level.