A Software Design Specification for Typing to the Cross

by Jennifer Weeks, Sam Mills, Drew Heaton, and Anuj Shah

Table of Contents

1. Introduction

1.1. Document Outline

- Introduction
- System Overview
- Design Considerations
    - Assumptions and Dependencies
    - General Constraints
    - Goals and Guidelines
    - Development Methods
- Architectural Strategies
- System Architecture
- Policies and Tactics
- Detailed System Design
    - Classification
    - Definition
    - Responsibilities
    - Constraints
    - Composition
    - Uses/Interactions
    - Resources
    - Processing
- Glossary
- Bibliography

1.2. Document Description

1.2.1. Introduction

The purpose of this document is to describe the specifications for *Typing to the Cross*. This document encompasses all necessary functions for Typing to the Cross. This document will allow the programmers to implement the functionality of Typing to the Cross without having to recontact the customer.

1.2.2. System Overview

- Small database of verses for each course
    - Random verses generated for each encounter
    - Verses are "tagged" for different topics

Player will be able to select in the start menu the different tags, from '1' to 'all' being available for selection, before pressing start. During the game only verses with those tags will generate for their encounters. Each verse may have multiple tags. Each single tag must be supplied with enough verses to complete the journey.

- Main Menu
  - Selectable font sizes
    - For those who have trouble reading smaller fonts
  - Typing Speed Goal
    - This will dictate the difficulty of the encounter. The higher the goal the faster your enemies will move during the encounter.
    - Optional: adjustment of typing speed goal during game in case user goal is unrealistic for their abilities
  - Length of Journey
    - See game length below
  - Verse Topics
    - Selected as check boxes, players will have the option of selecting as many as they want, but must select at least one.
    - A select all button will be available.
  - Start
- Turn sequence
  - Hit start
  - Static background (leading to hill with cross)
  - Character automatically walks along path until encounter
  - Encounter
    - Find and opponent and a verse appears at top or bottom of screen
    - Begin typing verse as quickly as possible
    - Opponent advances at slow set pace
    - Correct character input pushes opponent back
    - Incorrect character input allows opponent to advance
    - Branch: victory or defeat
  - Branch victory – resume automatic walk
  - Branch defeat – slowed/injured
    - Injury is removed by redoing battle (no penalty for loss here)
- Game Length
  - Demo: 2 encounters (Roman Road)
  - Quick Game: 3-5 encounters
  - Regular Game: 10-12 encounters
  - Long Game: 24-26 encounters
- Typing stats
  - Speed
  - Accuracy

2. Design Considerations

2.1. Assumptions and Dependencies

Desktop running Windows 8.1 (possibly working on Mac OS X as well). No mobile versions planned. Optional web version compatible with all operating systems. Internet connections are not required but will enhance the experience.

2.2. General Constraints

It will need to conform to the standards of current Microsoft Windows operating systems. It will be optimized for desktops and laptops as opposed to a mobile experience. It will need to communicate with a server over the Internet for optimal experience. No encryption is required since the data being sent over the network is not sensitive. Data will be sent using TCP to ensure its integrity. By connecting to the Internet, the user will have access to a wider variety of verses and translations.

2.3. Goals and Guidelines

- Keep the project as simple as possible while still producing a quality product.
- Emphasis would be on speed to give the user a fluid experience.
- Create a unique product.

2.4. Development Methods

The development method for this project will be the spiral method. This will allow us to identify risks and problems early on and have frequent testing of the prototypes throughout the process.

3. Architectural Strategies

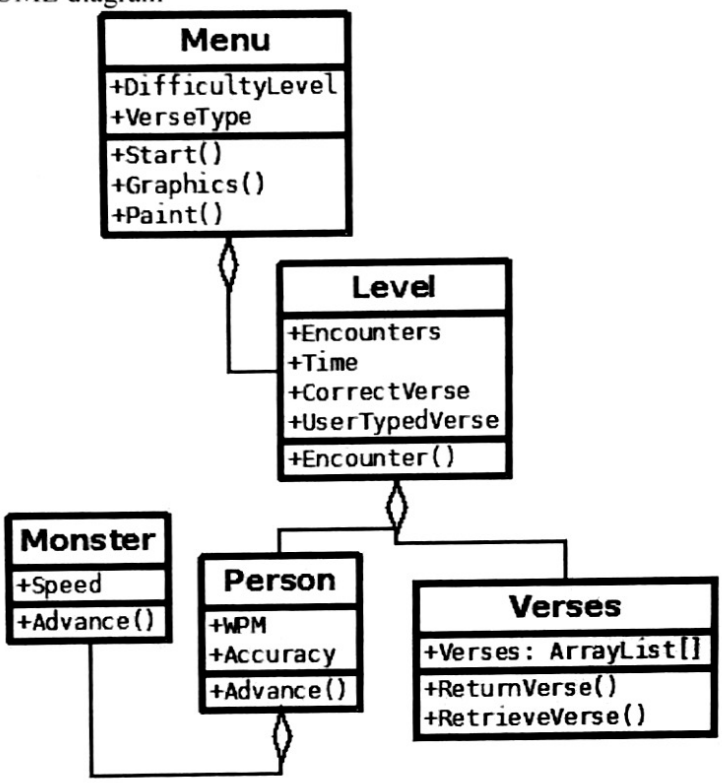
- The application will be programmed in Java. The database will be written in MySQL (or .Net).
- Possible changes to allow it to run on Apple and mobile devices.
- Takes information from the keyboard and mouse and outputs to the display.
- Use Java's built in exception handling.
- Will communicate with an off-site database.

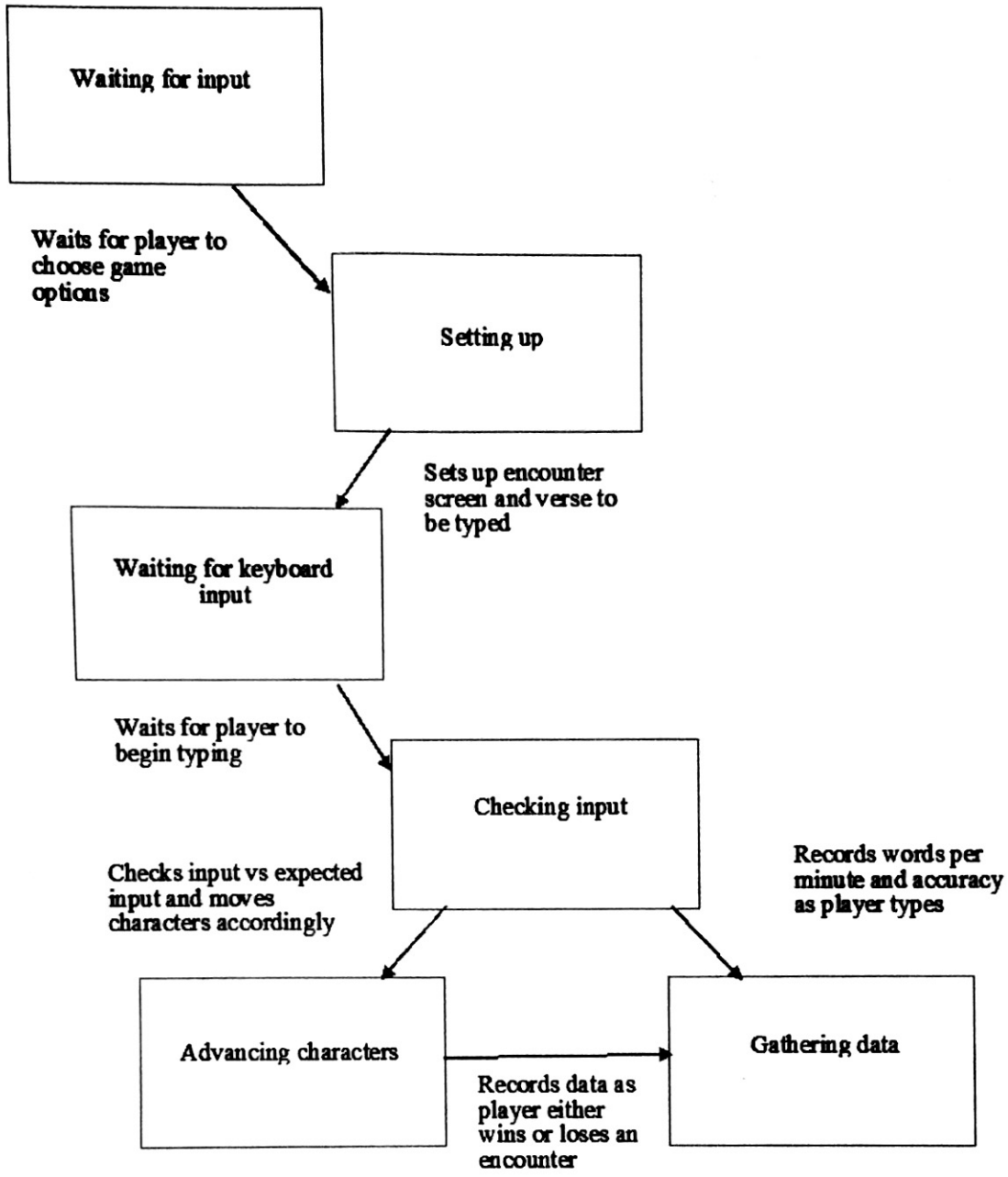
- Will sync a small number of verses from the database to the end user's device to allow offline mode.
- Use TCP over a network.

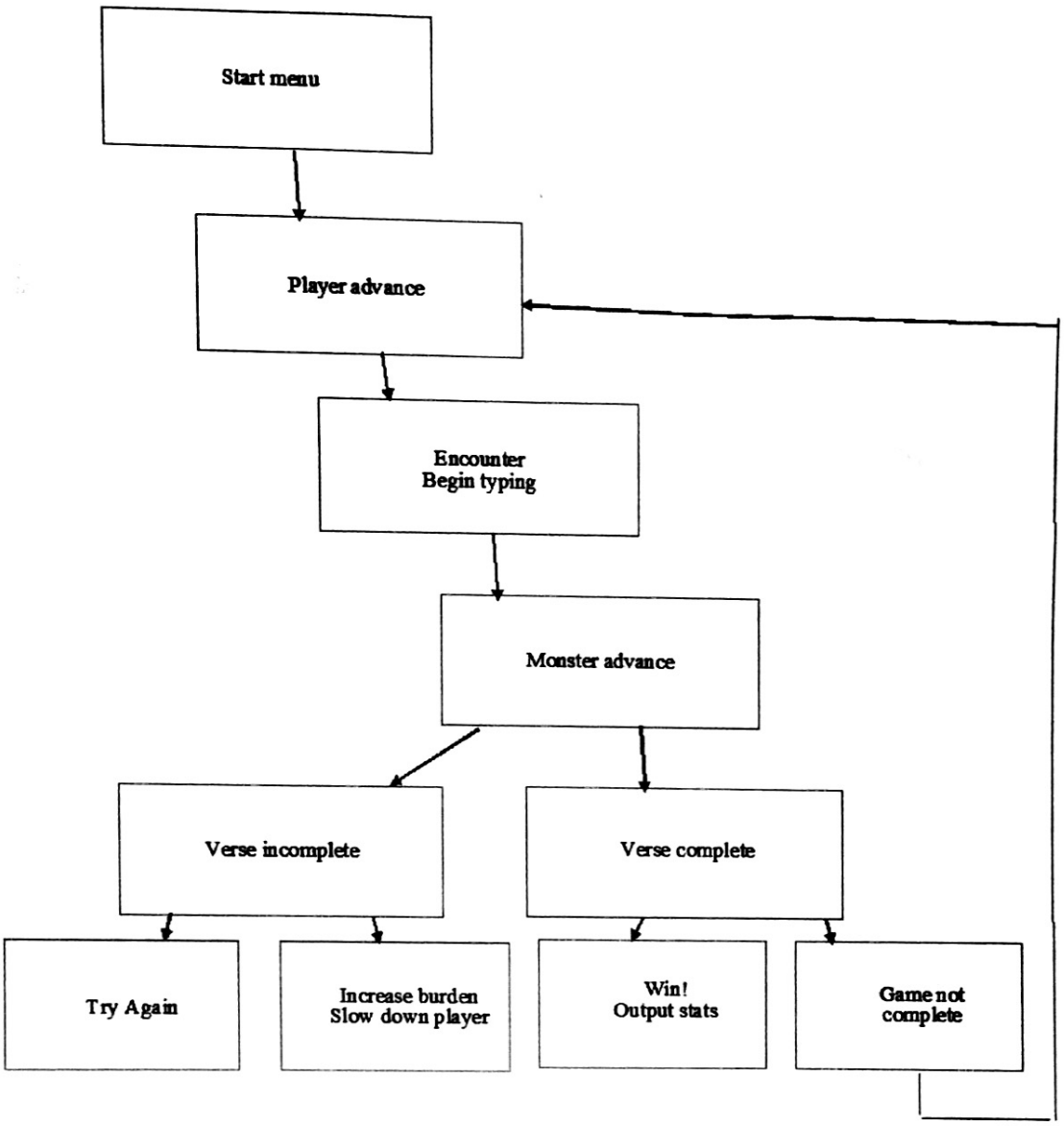
4. System Architecture

This program will follow the architecture of event systems.

UML diagram




Waiting for input
Waits for player to choose game options
Setting up
Sets up encounter screen and verse to be typed
Waiting for keyboard input
Waits for player to begin typing
Checking input
Checks input vs expected input and moves characters accordingly
Records words per minute and accuracy as player types
Advancing characters
Gathering data
Records data as player either wins or loses an encounter



4.1. Subsystem Architecture

The level class described in the UML diagram above will be the main game screen that the user interacts with. This class will be responsible for the graphics, receiving and processing the user input, displaying output to the screen, and displaying a timer to the user to see how quickly they are completing verses. This class will be the main way for the user to interact with the program.

5. Policies and Tactics

- Using the Java compiler and the built in Java libraries.

- Comments will be used throughout the code to improve readability and describe important or complex parts of the code.
- Sections of the code that fulfill requirements will be marked by comments.
- Refer to error messages and built in IDE debugger.
- The program will only need maintenance if obscure and unforeseen errors occur, or if the program needs to be updated to work on future operating systems. Making the code readable will show where exactly the code would need to be changed.
- Organized into Java classes.
- Use the command line to call the Java compiler.

Function to handle battles:

```
function battle(player, monster) {

        verse = getVerse();
        new Label(verse);
        new TextBox(no backspace);
        new timer;
        timer.start();
        userInput = TextBox.getInput();
        timer.stop();

        // will use Levenshtein algorithm
        accuracy = checkAccuracy(verse, userInput);

        if(accuracy > monster.accuracy && timer < monster.time)
                print("You did it!");
                print(accuracy);
        else
                print("Monster advances!");
                monster.Advance();

        player.WPM = getWPM(time, verse);
        player.accuracy = accuracy;
}
```

6. Detailed System Design

6.1. Classification

- **Player Class**
- **Monster Class**
- **Menu Class**
- **Level Class**

6.2. Definition

- **Player Class** - This class represents the player's character and will keep track of important information and data related to the character.
- **Monster Class** - This class will keep track of all information related to the monsters that appear to challenge the player and what requirements that need to be met before it can be defeated.
- **Menu Class** - This class represents the main menu used by the player to start and configure the game.
- **Level Class** - This class represents the actual game instance that the player will play in a given session.

6.3. Responsibilities

- **Player Class** - This class is responsible for keeping track of the player's movements onscreen and contains stats about the player relating to how quickly and how accurately they are able to retype the verses.
- **Monster Class** - This class is responsible for keeping track of the location of the monster and stats that the player needs to match or exceed in order to successfully defeat it.
- **Menu Class** - This class is responsible for giving the user a graphical interface they can use to select options and preferences for the game, as well as starting the game with the settings that they have chosen applied.
- **Level Class** - This class is responsible for displaying the game graphics to the player. It will also keep track of the input from the player during battles, display the text that the user types onscreen, keep track of the time passed in the current battle, and keep track of the accuracy in the current battle.

6.4. Constraints

Stats will be kept track of as integers. All functions and other classes used will have to be classes that work for the Java programming language.

6.5. Composition

These classes will be composed of simple built in methods and classes provided by Java. These will include the battle function described in section 5 and a function created with the Levenshtein algorithm.

6.6. Uses/Interactions

The level class will make use of all the other classes in some way. Settings data chosen in the menu class will be used by the level class to set up the game. Stats and position data will be traded between the level class and the monster/player classes very often.

6.7. Resources

All components will require the Java Runtime Environment to execute. This game can probably run on one processor with an inconsequential amount of RAM. The game will at most take up a few MB on the user's hard drive.

6.8. Processing

The menu class by default will display the start menu, but it will also be responsible for loading and displaying the level class, or reloading the main menu if the player quits the game.

7. Glossary

Algorithm
- a self-contained step-by-step set of operations to be performed.

Class
- a class is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior

Compiler
- A compiler is a computer program (or set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language, often having a binary form known as object code).

Exception
- An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

IDE
- An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger.

Interface
- An interface is a description of the actions that an object can do... for example when you flip a light switch, the light goes on, you don't care how, just that it

does. In Object Oriented Programming, an Interface is a description of all functions that an object must have in order to be an "X".

Levenshtein Algorithm:

- An algorithm that compares two strings and returns a number indicating how many single character edits are needed to make one string identical to the other. Perfect for a typing game, returning the number of mistakes made.

Metaclass

- a metaclass is a class whose instances are classes. Just as an ordinary class defines the behavior of certain objects, a metaclass defines the behavior of certain classes and their instances.

Module

- Programs are composed of one or more independently developed modules that are not combined until the program is linked. A single module can contain one or several routines.

Source Code

- source code is any collection of computer instructions (possibly with comments) written using some human-readable computer language, usually as text.

Spiral method

- (A method for developing software) The spiral model is a risk-driven process model generator for software projects. Based on the unique risk patterns of a given project, the spiral model guides a team to adopt elements of one or more process models, such as incremental, waterfall, or evolutionary prototyping.

Subclass

- A class that is derived from another class

Subroutine

- a set of instructions designed to perform a frequently used operation within a program

Superclass

- A class that has a derivative class

Tag

- A word or phrase used to describe/find a piece of data. Useful for organizing and searching data

TCP

- Transmission Control Protocol, a protocol for sending data over a network or the internet in a reliable manner with guaranteed accurate delivery.

UML

- Unified Modeling Language
- A general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.