Matherator Extreme!

The Feckless Ellipses

2014

Team

- Talus Baddley — *Team Leader*
- Taz Chapman
- Tanner Floisand
- Aprillemæ Hanley

Contents

Project Concept

The Matherator Extreme is a system of arithmetic-practice computer games for elementary school students in the first and second grades, designed to reïnforce the skills of addition and subtraction. It will run as a Java-based desktop application for all modern computer platforms.

It's not easy to fool kids into willingly participating in any education-based media—in fact, when done poorly, such computer games and TV shows feel transparently deceitful and disrespectful. But the good ones can even be addicting. Take *The Oregon Trail* game: Despite being moderately educational, it has been wildly popular for decades, by making the pioneer's adventure the student's own.

For The Matherator Extreme to succeed, it needs to be engaging like the best of the computer games that came before it. That may mean that a little bit of game skill will be necessary in addition to pure math skill.

Components

See also Figure 4.1 on page 8

a. **The Student Component**

Students, arriving at a computer, will launch The Matherator and log in with their name and their class (these will be set up in advance).

The games they can play will be divided up into a sequence of Units, corresponding to the areas of learning to cover in the student's class—for instance, singles addition, single-double addition, long subtraction, etc. The exact breakdown of these units is still to be determined.

Within each unit, there is a small handful of games, roughly increasing in difficulty from one to the next. The student starts playing at the first game they haven't completed within the unit, and move up through the games at their own pace. Some of the games in a unit can include (if applicable):

- Kitten Toss, using math to toss kittens a precise distance to safety…or into grave peril.

- Maze of Monty Halls, using math to guide the student through a maze of doors and tigers.
- Blackjack—because…why not, it's math-based!
- Battleship, using math to find the opponent's ships.
- Racecars, where students race against each other, and need to steer and do mental math at the same time.

More games can be added in the future.

b. **The Teacher Component**

Using a separate program, a teacher can administer the gameplay of their class(es). They see a list of their students in each class, and can add and remove students and classes. They can also get a report of how much each student has completed, and how well each is doing.

There is also a Unit Manager, which presents all the installed Units in a sensible default order. The teacher can re-arrange these according to their own class structure, and can enable and disable individual units as presented to the class, to keep the class on the same track.

If the student has completed all of the games in all of the currently-available units, they are permitted to re-play any game they wish.

c. **The Server Component**

Coördinating the teacher and student components is the Server software. It should be very simple to install and to use to add teachers to the system.

Ideally, it will provide an auto-discover mechanism on the network, to keep the students from having to enter the server address. This feature may need to be added later, though.

Mockups

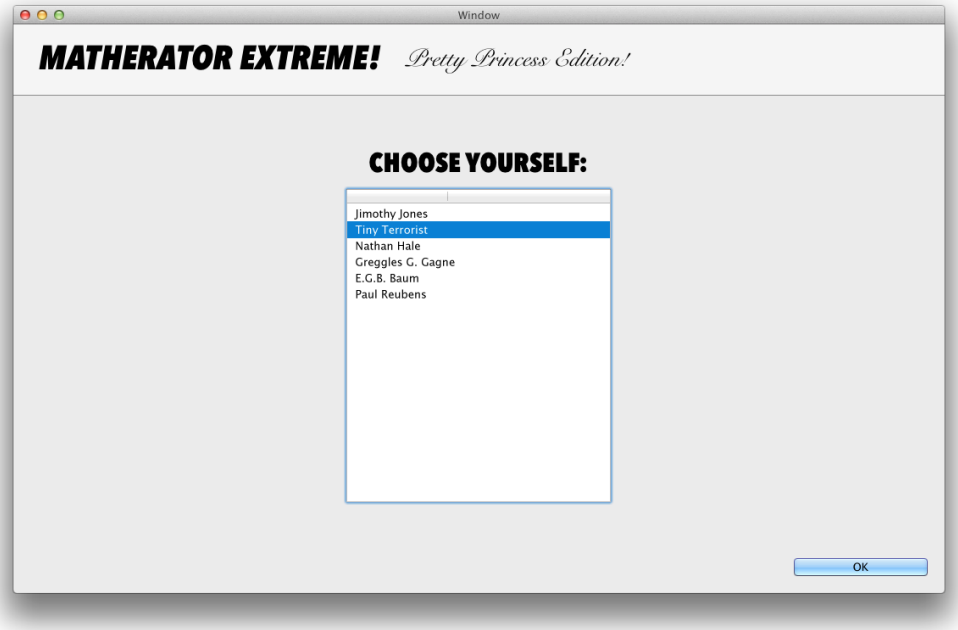

Figure 3.1: The student log-in screen is shown only once; subsequently Matherator remembers the student.

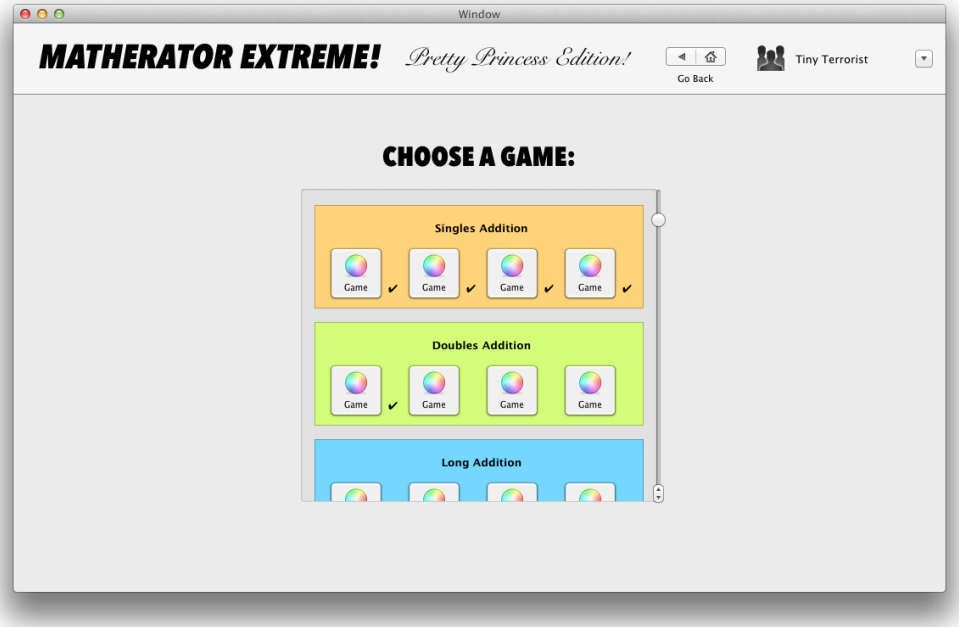

Figure 3.2: Student then chooses the game they want to play.

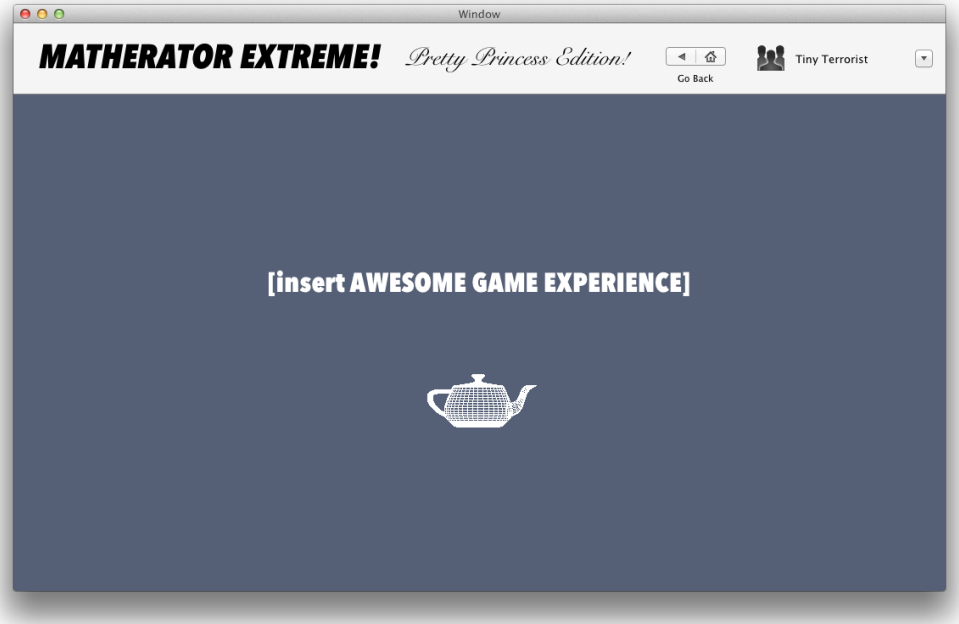

Figure 3.3: Super-exciting placeholder gameplay.

Project Architecture

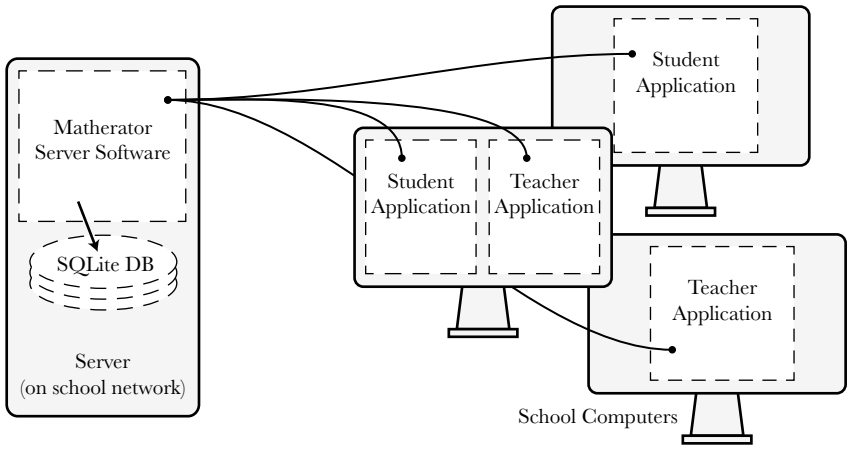

Figure 4.1: Overarching project architecture.

To maintain maximum interoperability between platforms, nearly all components of The Matherator will be written in Java.

The Student Component is a GUI-based desktop application built from a single-window philosophy. Standard Swing widgets will be used around the edges, but most of each game will be programmed using basic mouse- and key-listeners and basic drawing instructions.

For all the human interfaces we will be using a Model-View-Controller architecture, especially for the games:

- The model will contain all of the logic for gameplay, and will be just modular enough so that parts of it can be reüsed between the different games.
- The views consist of the objects and sprites that draw themselves to the screen within the game.
- The controller (by definition) connects the views to the model, delivering digested and meaningful messages to the model in response to raw events from

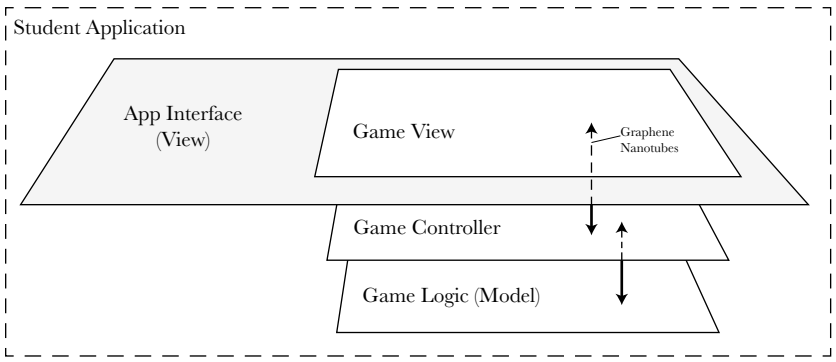

Figure 4.2: It's sensible MVC. No, seriously.

the views. It also receives update messages from the model and acts as a data supplier for view objects.

This is to facilitate sane implementation and maintenance, particularly in areas where rudimentary drawing instructions must be used.

The Teacher Component is also a GUI-based desktop application; in this case, only standard Swing widgets will be needed (still reasonably MVC-ish, of course).

The Server Component is a headless piece of software which maintains all its known teacher and student records in an SQLite database, which it controls directly.

The Student and Teacher software will communicate with the server in simple, record-based messages (encoded in YAML) over standard TCP connections—so it should go without saying that the computers running them need to be networked together.

At this point, on the client side, we are only providing for desktop-based applications. However, there is no reason that, in the future, these client components could not be adapted to mobile platforms.

Project Requirements and Tasks

Because of the length and relative dynamism of the project requirements document, it is housed on the Matherator project Wiki, in the form of a list of user stories.

Requirements will be added there as they are discovered, and shuffled between releases.

Similarly, the list of tasks, as they relate to the development of the Matherator, are also maintained on the project Wiki, so that their status can be updated all the time.