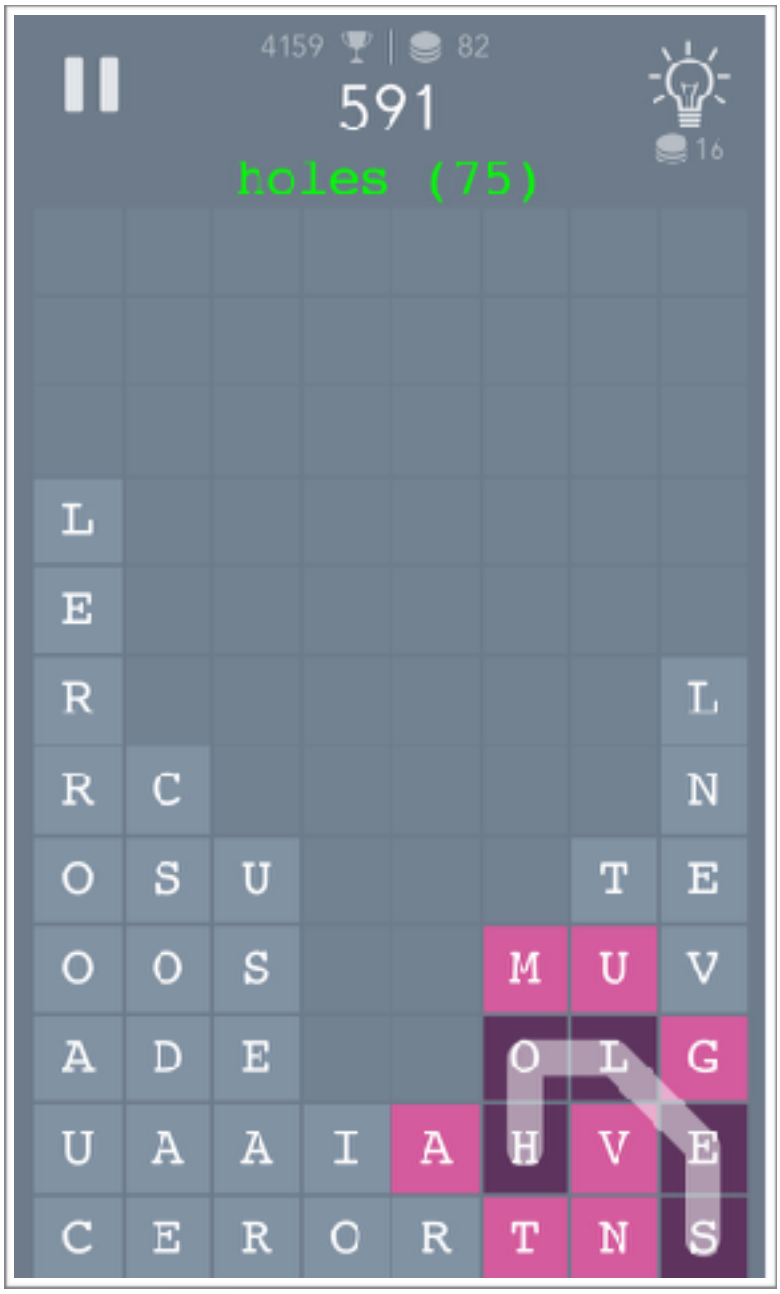
For my father's iOS game Hangman Dojo, I implemented a new mode — the recently popular word tower, where players find and eliminate words from a grid of random characters to prevent the grid from growing too high. As I observed that most similar games in the market generate their grid randomly, sometimes resulting that only short boring words like "it" or "is" exist in the grid. In order to add fun to the game, I designed the grid generation procedure to ensure that pre-selected long words will be hidden inside the structure for the players to find.

I made use of a word list of around of 300,000 English words. To improve word searching efficiency as well as memory use, I implemented a raw trie data structure with C++, where data stored in binary form is directly accessed. This structure hints the program which letter combinations, "abma" for example, are impossible word prefixes, and thus reduce the amount of letter combinations I need to look for when finding possible locations to hide new words. In this way, I managed to reduce the time usage of generating a new line of words (and hide random long words inside) down to 0.1 seconds, from a 3-4 seconds' time of a brutal-force searching algorithm, while at the same time compressing the data structure's memory use by more than 20 times (by using raw binary storage), from a huge 200MB usage of a similar data structure built from standard methods.

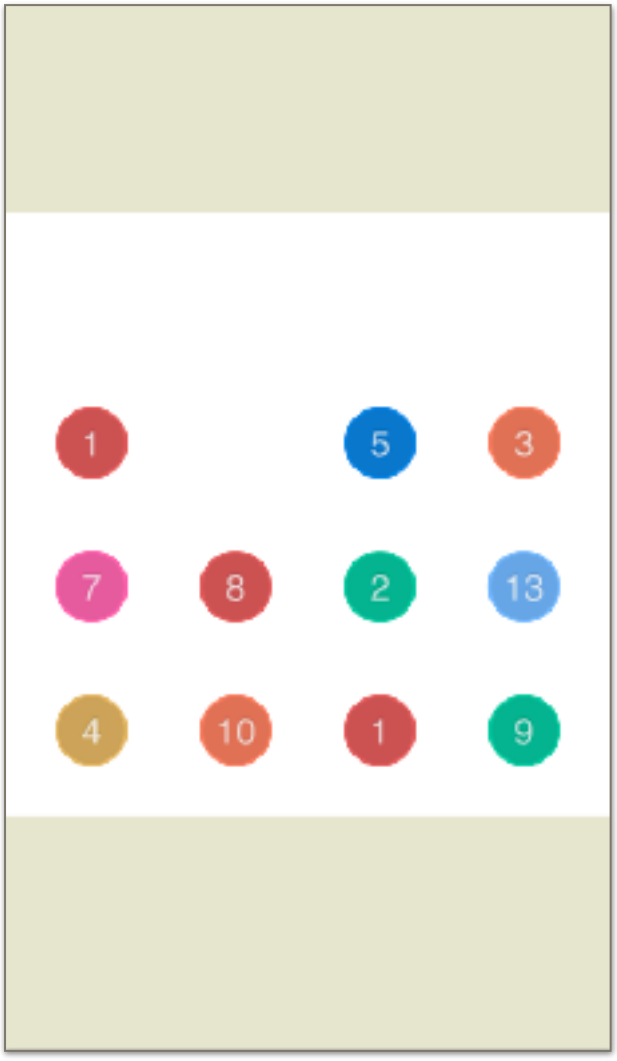


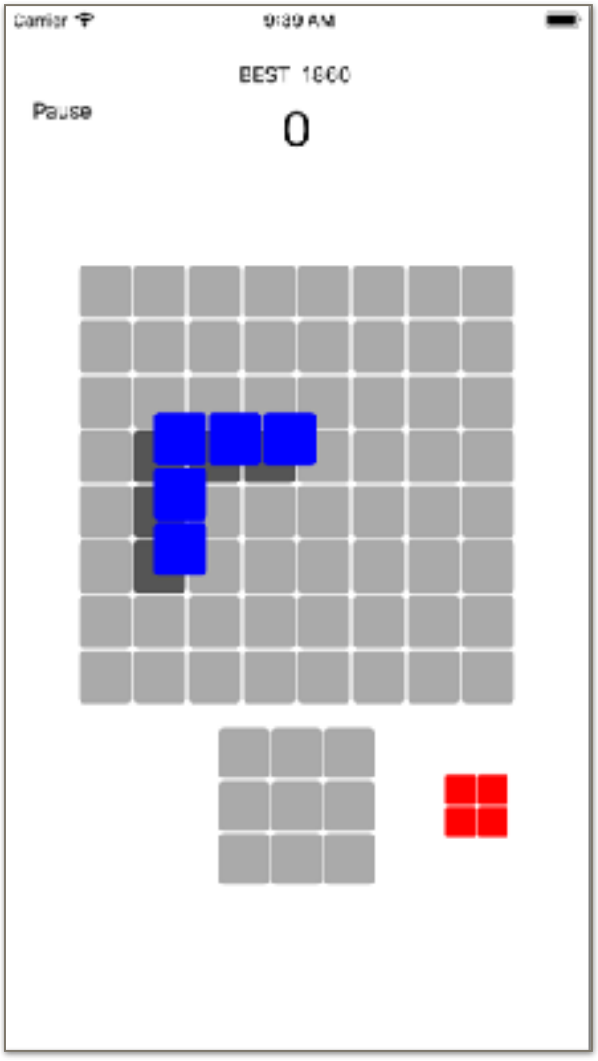
The result is a fun word searching game that instantly generate grids with lots of long words to find. It has been submitted onto the iOS app store for around 1 month, and has captured more than 1,000 players.

After completing the Hangman Dojo update, I devoted much of my free time into developing a game engine that allows for quick development of casual arcade, board or trivia games reusing many code components like game storage functionality (which allows a person to pause in a game and come back to continue it sometime later), leaderboard and friends functionality and social media sharing functionality.

By dividing the code into six clear modules (game layer, widget layer and data layer for handling of game logic, and controllers, renderers and displays for handling of the rendering of game graphics), I created a first game using the engine — “Color Me! Number You!” (image on the left), being a brightly-colored collection of many number games, like the classic “2048” number-elimination game, requires you to swipe, tap and drag to eliminate as many numbered cells as possible under different rules. The engine made it as easy as 1-2 hours’ work and a bit over 100 lines of code to implement any one extra game for the collection, where one only need to state the game logics literally, like “all numbers shall now be swiped to the left”.

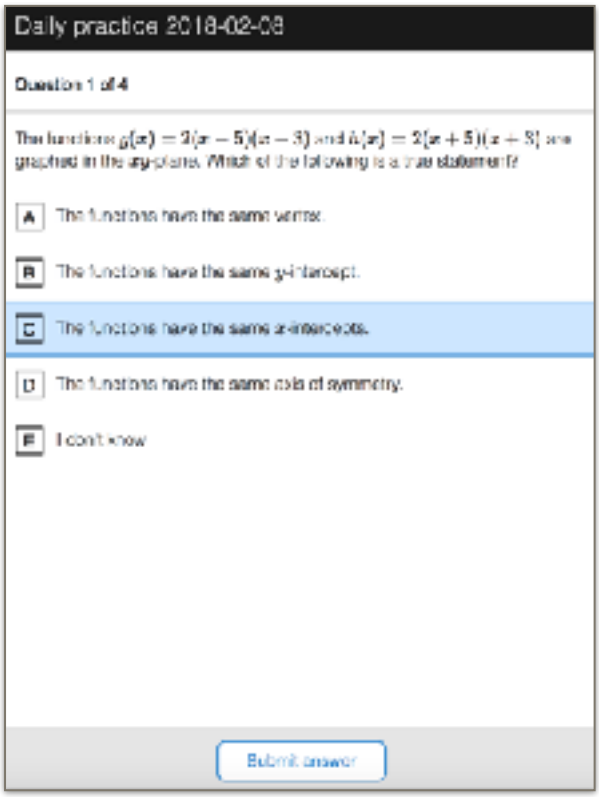
The game engine’s design was further tested as I created a second game, “Shape Master” (image on the right), one of a completely different genre. With just two days of work on new widgets that handle the display and movements of the shapes, the game engine did all the dirty work and presented me with a complete game with a fluent experience navigating across the game and various dialogs.

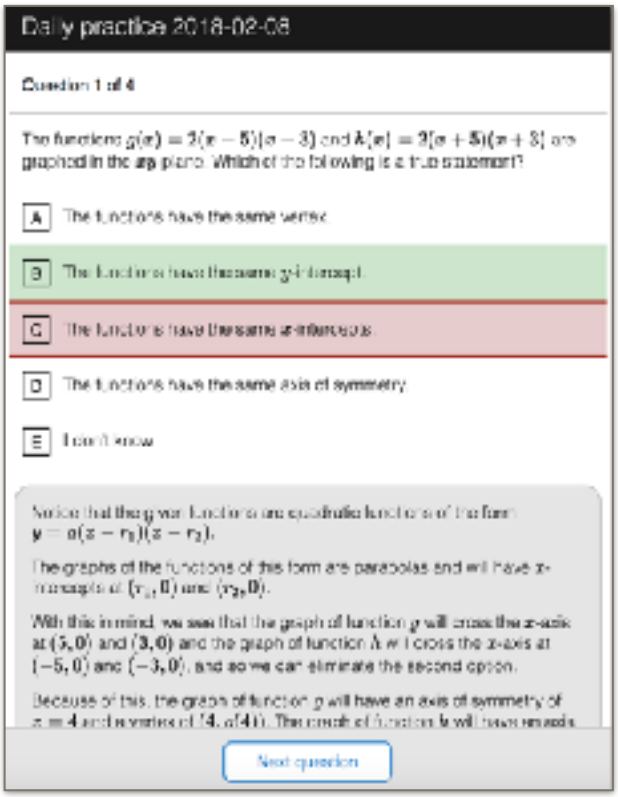




I also volunteered to continue my work for Weijun Technology to create more educational utilities. Taking the idea from my previous Math research on student performance predicting, I created another web application for the practice of SAT questions.

By evaluating the students' accuracy of each difficulty and category of problems, I managed to rate the students' performance on topics like geometry and algebra, and give suggestions on whether a student tends to make mistakes out of carelessness or a misunderstanding of concepts as my program tries to differentiate between "calculation problems" and "concept problems". Thus, I gave customized daily practices to these students, training them on their weaknesses, in order to accelerate their improvements. (The sample problems in the images are borrowed from Khan Academy's SAT practice tests)





I have extended on my computer science research in 2016 as well, where I used machine learning to order a sequence of Chinese words into grammatically correct sentences, as I now have some extra free time. By reading on the construction of word vectors and long short-term memory recurrence neuron networks, which I already slightly touched in 2016, I tested their functionalities on analyzing essays, and is currently attempting to implement automatic student essay graders as well as article keyword analyzers, so that I can integrate the reading and writing sections of SAT into my auto-practice-generating application.