Mobile App Engineering

Final Project

Brandon Dunlap, Jonathan Olcheski

Type Defense Force

Overview:

In the original proposal of Type Defense Force, we said we wanted to include a system that tracks high scores and typing performance. Due to the actual game portion of Type Defense Force being more difficult to implement then we thought; so the main game is all we were able to implement before the deadline.

Architecture

Overview: Our program uses a model view controller type architecture. A model view controller is a software design pattern for implementing user interfaces on computers. It divides a given software application into three interconnected parts, that work in a cycle to accept information from, and present information to the user. In basic terms, in model view controller, the program is spilt into; the model, the view, and the controller. The user users the controller, which in turn manipulates the model, which updates the view to be seen by the user.

Classes:

1. TextDefenseForce.java: Our initial activity that creates the controller, handles input from the keyboard, and forwards that input into the controller.
2. GameController.java: Is in charge of creating the game view, and updating the models (i.e asteroid, asteroidField, and ship) via a timer, as well as keeping score.
3. GameView.java: Updates what the user sees by looping through every asteroid in the asteroidField (i.e every asteroid on screen) as well as the ship, and draws them to the screen.
4. Asteroids
    a. Asteroid.java: the sprite class that represents the incoming asteroids, and the words that must be typed to destroy them. It holds the x and y coordinates of the sprite on the canvas.
    b. AsteroidField.java: Holds all the asteroids on screen…
5. Ship.java: the sprite class representing the ship.
6. Word List Service
    a. WordListService.java: A service that, when requested, will start a thread to get a word from the ‘word-list.txt’ file. It will return the word as a broadcast.
    b. Service Wrapper
        i. ServiceWrapper.java: requests words from WordListService, and listen’s to the service’s broadcasts. It funnels the new word to the ServiceWrapperListener.

ii. ServiceWrapperListener.java: To use the Service Wrapper, a service wrapper listener is required. This allows the service wrapper to call the necessary methods from the listener.

c. Cached Service Wrapper
    i. CachedServiceWrapper.java: used by the Game controller to cache up to X amount to words. These words are then ready to be added to the asteroid field when requested. The reason for caching to avoid waiting if more than 1 new asteroid is needed.
    ii. CachedServiceWrapperListener.java: To use the Cached service Wrapper, a cached service wrapper listener is required. This allows the cached service wrapper to call the necessary methods from the listener.

Work Division:

Jonathan Olcheski is completely responsible for implementing the Word List Service. The rest of the project was done evenly been Jonathan Olcheski and Brandon Dunlap.