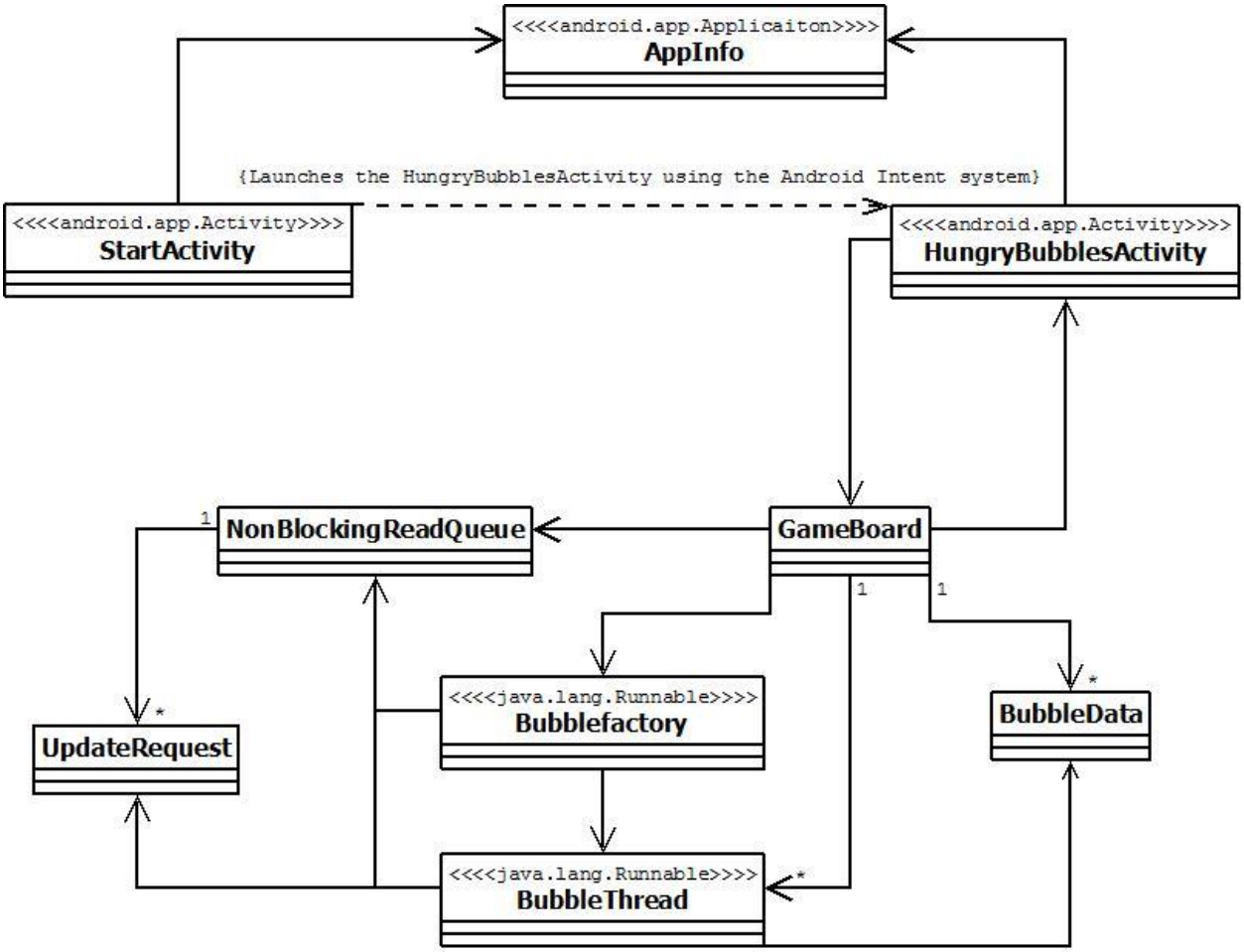
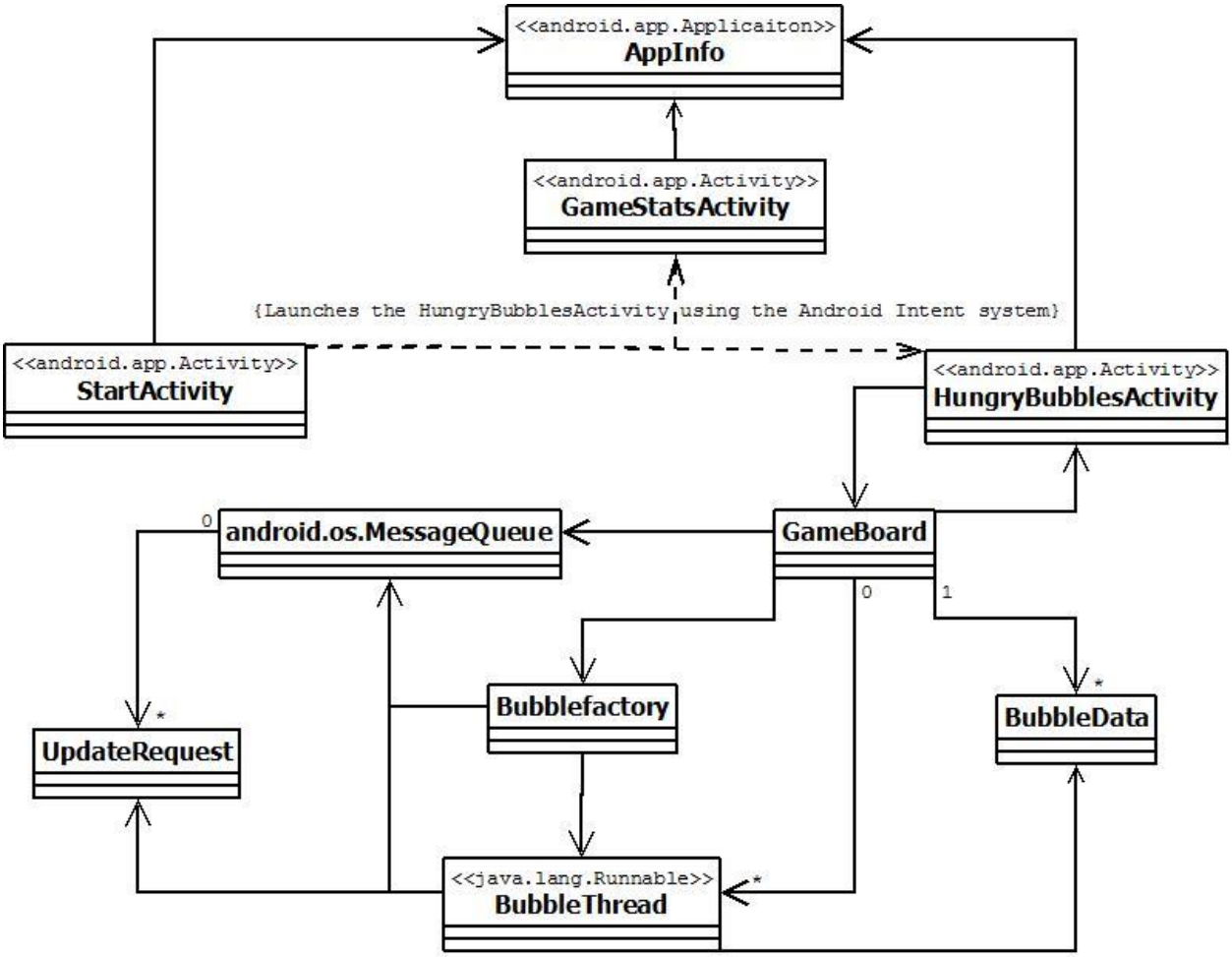
Hungry Bubbles Design

Initial Class Diagram:



Final Class Diagram:



*Note that all of the Activities (subclasses of android.app.Activity) have a corresponding XML file which is used to define the structure of the UI for that Activity (each Activity is roughly equivalent to a screen or page in the application). There is also an AndroidManifest.xml which declares general information about the application to the Android OS.

Class Descriptions:

AppInfo – This handles persistent data on startup and shutdown.

StartActivity – Launched when the application first starts up

GameStatsActivity – Used to display the current game statistics (the numbers of games won and the total number of games played) to the user

HungryBubblesActivity – Game screen

GameBoard – Maintains the positions of all of the bubbles using immutable BubbleData Objects.

BubbleData - Encapsulates data that defines a bubble

Bubblefactory – Used to construct bubbles with random attributes

BubbleThread – Controls the movement of a bubble; sends requests for the GameBoard to update the bubble's position

UpdateRequest - Encapsulates the position to move the bubble to as well as a reference to the BubbleThread making the request

NonBlockingReadQueue – Initially designed to serve as a shared data structure between the BubbleThreads and the GameBoard to implement a consumer-producer pattern; replaced by the thread-safe MessageQueue provided by the Android OS as explained below in the concurrency section

Concurrency:

Thread Types –

Main/UI Thread – The main thread which handles the primary application execution flow. This is also the only thread which Android allows to make direct changes and updates to the UI. This thread is also responsible for handling all updates to the player's position and information, with these updates being initiated through touch events which are received through a callback method in the GameBoard class (GameBoard.onTouch()).

BubbleThread – Controls the movement of a single opponent (non-player) bubble until either that bubble is eaten or the game ends or is suspended. New BubbleThreads are created by the main/UI thread using a BubbleFactory instance inside the GameBoard class using the startBubbleIfAppropriate() method.

Synchronization Points –

- Providing updated thread-driven bubble information to the game board so that updates to the opponent bubbles will be reflected in the UI
    - Handled using the producer-consumer pattern
    - BubbleThreads produce UpdateRequests which the GameBoard then consumes
    - Original plan was to use the custom NonBlockingReadQueue synchronized queue class as a shared data structure between the GameBoard and the BubbleThreads with the BubbleThreads adding UpdateRequests to the end of the queue whenever they want to move their bubble to a new position and the GameBoard reading the UpdateRequests out of the queue every time it starts a UI update

    - Decided to use the built in MessageQueue (android.os.MessageQueue) that Android provides for all threads for the purpose of sending messages between threads to replace the NonBlockingReadQueue
        - All of the BubbleThreads are given a reference to a Handler (android.os.Handler) for the main/UI thread's MessageQueue which they use to send Messages (android.os.Message) containing their UpdateRequest to the main/UI thread

- Shared access to bubble information
    - Created a fully immutable class (all fields are final static), BubbleData, to encapsulate all of the information which defines a bubble, including position, size, and color information to allow BubbleData objects to be safely shared between threads

- Notifying a BubbleThread that the bubble it controls has been eaten
    - Used Java synchronized blocks to synchronize access to the boolean field within the BubbleThread class which keeps track of whether or not the bubble controlled by the BubbleThread has been eaten
    - The GameBoard updates the value of this boolean field through a public BubbleThread setter method
        - Synchronizes on the BubbleThread (synchronized(this) { …}) before changing the value of the boolean field
        - The BubbleThread synchronizes on itself at the start of each iteration through its bubble update loop before checking to see if its bubble has been eaten

Android MessageQueue vs. NonBlockingReadQueue

The MessageQueue provided a number of advantages over using the NonBlockingReadQueue. One of the greatest advantages was that the NonBlockingReadQueue class was an untested, custom implementation while the Android MessageQueue class is built into the Android operating system which has already been proven to send communications between threads effectively and efficiently. Also, with the NonBlockingReadQueue approach the GameBoard would have been required to continuously poll the queue to see if UpdateRequests had been added whereas Android provides a callback method which is called whenever a message is sent to a thread through its MessageQueue, which means that UI updates will be initiated whenever an UpdateRequest message is sent to the main/UI thread and when there are no UpdateRequests to handle the main/UI thread will remain idle and be able to handle other tasks if necessary. This also fulfilled the requirement that the main/UI thread block for as little time as possible in order to maintain responsiveness and prevent the application from timing out and being shut down by the OS, which was the requirement the NonBlockingReadQueue was originally designed to handle. As such, the MessageQueue was able to take the place of the NonBlockingReadQueue in the system architecture with minimal accommodation and was ultimately much easier and cleaner to implement.