Team Impending Doom
Jeffrey Lin, Gabi Newman
Period 3

Gabi and Jeffrey's Tower Defense

Introduction

For our final project, our team would like to create a tower defense game. This would allow us to gain experience with both game development and graphics design, as well as explore the practical applications of data structures.

Tower defense is a subcategory of strategy video games. In a tower defense game, the player attempts to prevent the enemy from crossing a threshold. This is done by adding defensive structures (i.e. "towers") near the path of attack, which can shoot projectiles to eliminate enemies and prevent them from advancing. The player must strategize in order to determine the optimal number and position of towers, given a certain amount of "money" to spend. Levels consist of "waves" of enemies, which vary in number and strength depending on difficulty. When enough of the enemy cross the threshold, the game is lost.

Implementation

We can create an abstract Enemy class to define what an enemy must have and can do, and use multiple subclasses to represent different types of enemies, which usually have different abilities.
Similarly, we can create an abstract Tower class to define what a tower must have and can do.

When generating a level, we would start by populating a Queue (possibly a java.util.concurrent.DelayQueue) with Enemies that will be released as the level progresses. The level is completed when there are no Enemies on the board, or in the Queue.

As Towers are placed, their general position on the board will be mapped onto a (quad) tree to decrease the time it takes to check if an Enemy is within range of a Tower. For each Tower, a circle is drawn to represent its range of attack. Using the tree, we check for collisions between a given Enemy and all of the circles. If they collide, the Enemy is within range of a Tower and will be fired upon.

The Enemies on the board could be represented as items in an ArrayList. Iterating through the ArrayList and sequentially mapping each Enemy onto the tree would ensure that the Tower targets the Enemy closest to the end of the trail. Later on, we could add filters that would cause the Towers to target other specific attributes, such as health. This would require fast sorting of the ArrayList, which can be done using either QuickSort or HeapSort.

Major topics from this term:

~~sorting [merge, quick]~~
state-based searching
List [linked]
iterators
~~nested classes~~
~~deque [stack, queue, priority queue]~~
~~tree [searching, heap]~~