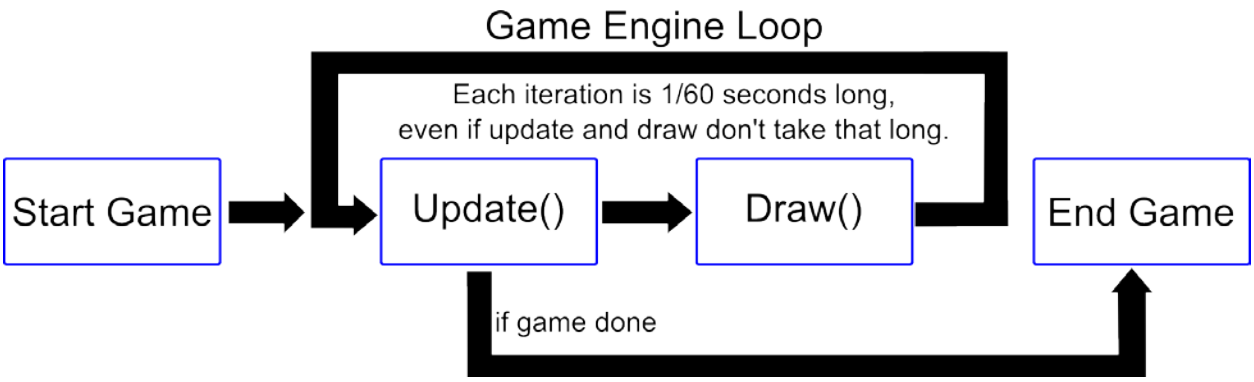
Simple Game Architecture

Note: This document is intended to be a simple introduction to some of the aspects of game architecture. It is by no means the final word on the game architecture discussed. Even further, concepts are discussed specifically in terms of the *Super Asteroids* game rather than generically. Using the **loadContent()** function for the *Super Asteroids* game engine does not imply you will see that function associated with all other game engines. For more information on the concepts discussed here, go to http://www.gameprogrammingpatterns.com .

Before tackling some of the finer details of the *Super Asteroids* game design, it is important to understand the heart of the game. The heart of the game is the game engine. For this project, it is only necessary to understand the following points concerning the game engine:

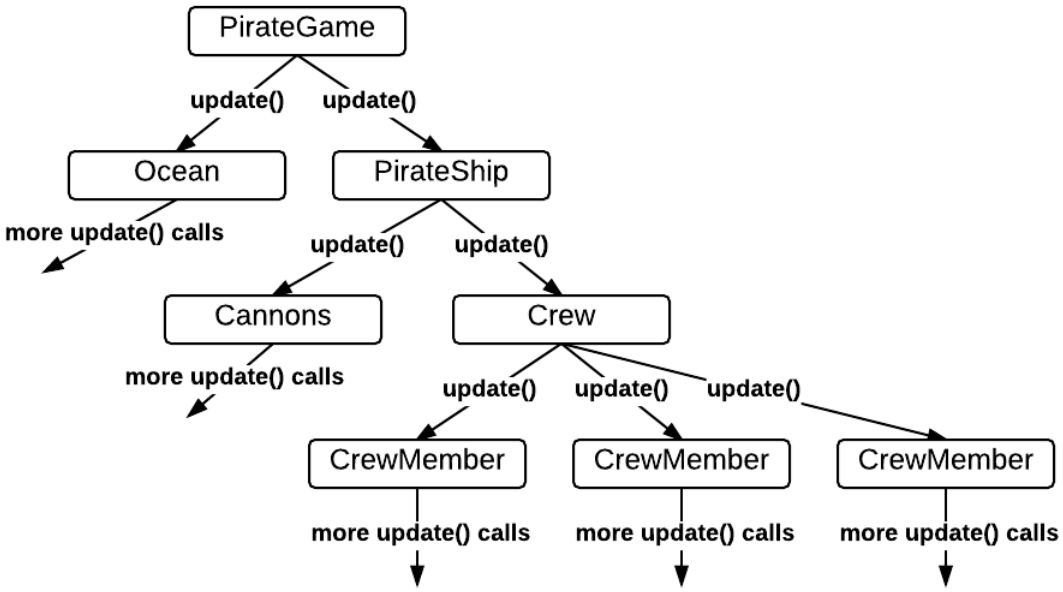
1. The game engine runs on a loop that iterates 60 times a second until the game ends.
2. The first part of the iteration loop calls the **update()** function.
3. The second part of the iteration loop calls the **draw()** function.

The flow of the game can be depicted by the image below.



The **update()** function is used to update all the elements of the game based on the elapsed time since the last update (in this case, elapsed time is always $1/60^{th}$ of a second). Updating a game element can include any number of actions including, but not limited to: changing its state, moving it, detecting and responding to collision, removing it from the game, calling **update()** on any sub elements, etc. Generally, the **update()** call starts with one object, then cascades through others.

Take for example a hypothetical pirate game. The game engine calls **update()** on a PirateGame object. As part of updating itself, the PirateGame calls **update()** on an Ocean object and the PirateShip object. The PirateShip updates itself and in so doing calls update on the Cannons and the Crew. Updating the Crew might result in **update()** being called on every CrewMember object. The image on the next page depicts this cascading effect.



The **draw()** function cascades through the game elements just as the **update()** function. When this function is called on a game element, the game element should draw itself to the screen and/or call **draw()** on any sub elements. Sometimes a game element will call **draw()** on its sub elements before drawing itself to the screen.

Load and Unload Content

The **loadContent()** and **unloadContent()** functions are also important calls made from the GameEngine and other key elements of the game. However, these functions are not invoked sixty times a second like the **update()** and **draw()** functions. Before going any further, it is important to understand the two functions.

loadContent() – Used to load content into main memory. Content to load can include sound and image files, as well as data from databases or other data files. Data loaded from databases or other data files can be used for a variety of things. In the case of the pirate game example, information loaded from data files could be used for deciding how many enemy pirate ships to generate for a particular level. This function has a cascading effect similar to the update cascade effect.

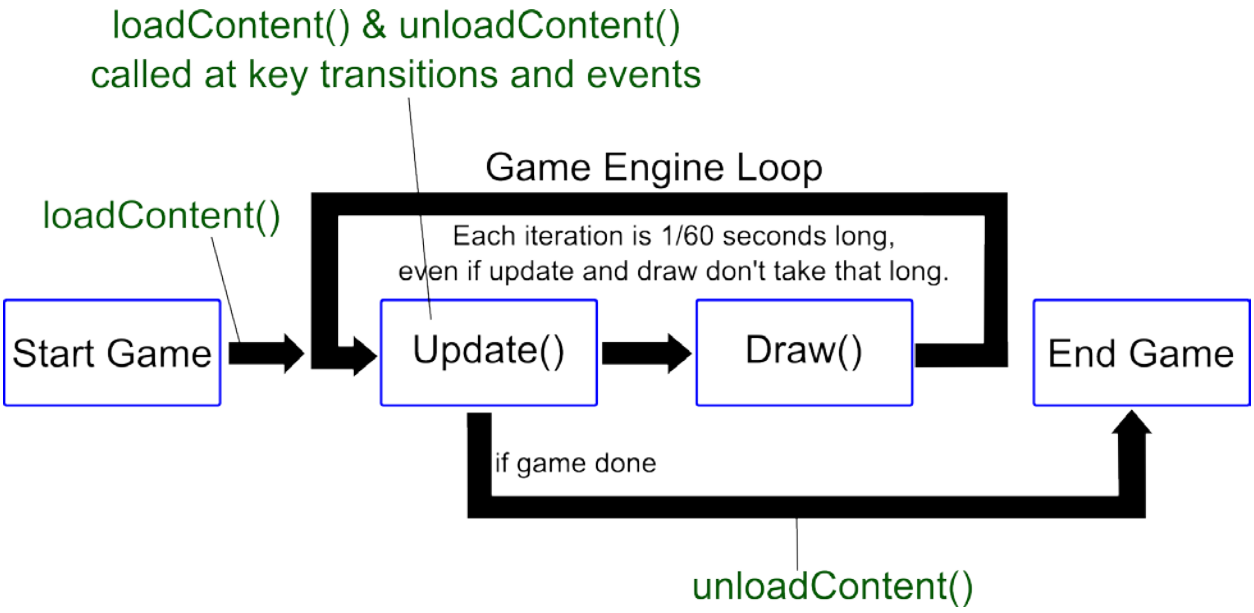
unloadContent() - This function is used to unload content from main memory. This function has a cascading effect similar to the update cascade effect.

Why have an unloadContent() function?

Take the pirate game for example. Let's say the game has 100 levels and each level loads 20MB of completely unique assets. If these unique assets are not freed from main memory after every level, main memory will become bloated with unnecessary media files. In fact, by the end of 100 levels, main memory will hold about 2GB of unused media files.

The **loadContent()** and **unloadContent()** functions are invoked during key game transitions and events. The best example of usage for these functions are seen when the game is first starting and when the game is ending. The *Super Asteroids* game engine will call **loadContent()** right before the game loop is entered. **unloadContent()** will be invoked right after the game loop has been exited. Another good example of a key transition in *Super Asteroids* is the level transition. When the game transitions from level *a* to level *b*, the

game should call **unloadContent()** on level *a* and **loadContent()** on level *b*. The figure below depicts the timing of these function calls. (You will be responsible for calling **unloadContent()** and **loadContent()** during level transitions, because the game engine does not know when level transitions occur, but you will.)



TODO Item: Connect to the *Super Asteroids* Game Engine

Connecting your game code to the game engine starts with a simple two-step process:

1. Create a class implementing the IGameDelegate interface found in the base package.
2. Complete the TODO item in the GameActivity **onCreate()** function.

If the above two steps are completed correctly, the game engine will:

1. Invoke **loadContent()** on your game delegate object right before the game loop is entered.
2. Call **update()** then **draw()** sixty times a second.
3. Invoke **unloadContent()** as the game is exiting.

Any other **loadContent()** or **unloadContent()** calls must be instigated by your code.