Introduction to Software Engineering

Basic Game Programming

Rami Sayar @ramisayar

GitHub Talks If you find a mistake, don't hesitate to fork the repository, clone it to your local machine, edit the mistake, `git commit -a -m` and then press the pull request button to notify me.

Basic Game Programming Concepts

Game development is one of the most exciting and complicated projects you can undertake in software engineering. Game development is a complex art and involves programming topics ranging from mathematics and linear algebra all the way to artificial intelligence. In this class, you are not expected to develop a state of the art game but rather to develop a very simple game and focus on the concepts of software engineering. These notes are consequently **not** exhaustive (not even close, maybe 0.0000001%, there is so much literature on game programming and so much to know, we will unfortunately not be able to coverit all in one tutorial). These notes should serve as a reasonable starting point for your asteroids project and will contain at least some of the terminology you will undoubtedly encounter online and elsewhere.

Game Loop

The most basic element of any game is the **game loop**. The game loop is the core loop in your game that drives the logic, graphics, sound, etc..

At its most basic form, the game loop is composed of four states: an initialization or startup state, a shutdown state, an update logic state and a draw/sound state. In pseudocode, it might look like this:

```
initialize()

while not quit:
    update()
    draw()

shutdown()
```

It is fairly easy to implement the above in Java. However, this game loop in its current state is very problematic and suffers a fatal slow; there is no control over the speed at which this loop operates, therefore the game will be executed at different speeds for users with faster computers versus users with slower computers.

The above game loop needs to be modified and an addition of a control condition is required. A simple way to control this loop is to have a target frame rate and force the execution to be stalled if we are executing much faster. In other words, we want to add a sleep condition (calling the sleep method to stall the loop) to the game loop and skip handling the rest of the operation if the sleep condition is met. This addition will allow you set a target frame rate (updates/second) and standardize the frame rate across all types of hardware. In pseudocode

```
initialize()

while not quit:
    sleep = calculate_sleep_time()
    if sleep > 0:
        Thread.sleep(sleep)
    else:
        update()
        draw()

shutdown()
```

There is just two more things we can improve about the above game loop. Firstly, the calls to both update() and draw() happen sequentially but if we get stuck in update() than the draw() will not take place often enough. Furthermore, if the update() is slow, user input will not be handled fast enough to offer a fun experience. Thus a slow update() can cause significant problems, so you will have to modify your loop to draw every once in a while.

Graphics Rendering

Graphics rendering (drawing a 3D or 2D game scene onto the screen) is often considered the most glorious aspect of game development. The graphics often makes or breaks a game for the classic hardcore gamer, however, they are less important for the more casual gamer. There are several ways to draw graphics in Java. As the purpose of this project is to build a 2D game, it is highly recommended that you stay away from the more complicated rendering pipelines such as OpenGL and stick with the Java 2D API.

There are several resources available on the Java 2D API, among the classic Java Tutorials which you can look at. Lesson: Overview of the Java 2D API Concepts

There are a couple of topics I would like to address before moving to the next topic. Firstly, the coordinate system that you use is extremely important and may notbe as straightforward as you

would expect. The Java 2D API actually simplifies this topic for you by forcing all geometrics passed to the API to be specified in a user-space coordinate system as opposed to a device specific coordinate system. Furthermore, the location of (0, 0) is now where you woul traditional expect it to be, in fact, its in the top-left corner as opposed to the bottom-left corner.

Attached is sample code that demonstrates how to draw geometric primities, and load and draw geometric primities.

Threading

Concurrency and parallalism are extremely important topics in software engineering. Threading will often be mentioned in the context of game programming. We will have a separate tutorial on threading but for now, I will define some terms and ask you to stay patient until the next tutorial.

- **Thread:** A thread is a seperate path of code execution.
- **Semaphore:** A semaphore is a method of controlling access to data. (See Mutexes and Locks))

Resource Management

Resource management is an interesting and non-trivial topic. In most games, you will often find that there are more artistic resources than can fit in memory all at once. As a result, resource loading (such as textures and music) needs to be done in a more sophisticated fashion. The approach of naively loading things all at once or when you need them does not work. Furthermore, you will note that file I/O will be slower and your code will have to take into account using a seperate thread to load files at runtime to prevent stalling the game.

In general, a good resource manager would have the following features:

1. You can reference a resource by an easy to remember string as opposed to its file path. This gives you the added benefit that the user can install the game anywhere on their system and you can change the organization of the files without affecting the resource usage.
2. The resource manager prevents duplicate resources from being loaded from the file system and taking up unnecessary memory. You will often reuse the same image/texture so loading it again each time will reduce your performance.
3. The resource manager can handle resources that are dependent on each other.
4. The resource manager might do some type of reference counting and unload resources from memory when they are no longer used or needed.
5. The resource manager takes care of loading from files in the background.

Animation and Sprites

Animation consists of drawing an image and moving the location of the image and drawing it again to simulate movement or some other effect. You will be anmiating several elements in

your game to create a dynamic environment. You should pay attention to ensure flickering is not a major concern. The movement of a sequence of pregenerated frame images is called *frame-based animation*.

A more interesting animiation technique is *cast-based animation* or **sprite animation**. Sprite animiation is the movement of the sequence of pregenerated frame images independent of the background. Think of the asteroid in your game and how it could rotate in the same position while slowly moving elswhere, this type of animation would be considered sprite animation or cast-based animation where the asteroid is the "cast member".

There are interesting challenges, particularly, with how you will control when you switch per frame to ensure a consistent animation independent of the speed of your game.

Events - User Input

A game would be pretty boring if there isn't an event that takes place. Events can be user-generated or triggers based on the story of the game or perhaps triggered by artificial intelligence. As a result, there is a certain class of software architecture that is generally used to handle situations such as events; they are called event-driven architecture or callback architecture.

User input in Java is event-driven. An event will be generated when a user presses a key or moves the mouse. This event will be trapped and sent to be handled by a method of your choosing. Typically, you will handle user input immediately when it occurs but you will delay the effect of the user's input to the next update() in the cycle.

There are different ways to handle user input depending on the framework you are using to host the game. If you are using awt as your UI framework, you will most likely have to override methods such as keyDown() and keyUp() to handle the event.

```
public boolean keyDown(Event evt, int key)
public boolean keyUp(Event evt, int key)
```

It is highly recommended you consult the literature of your game framework to figure out the optimal way to handled user input. This online book contains a chapter on user input handling in Java.

References

Internet Game Programming with Java

Entropy Interactive - Game Engine Design

Java 2D games tutorial

Animation