Game Loops (Lecture)

Last week we developed an HTML version of hangman.

[Slide with a hangman game]

How did our games advance?

Poll the audience and write their answers on the board

Let's develop a flowchart building from this idea.

Draw a flowchart with the primary phases (setup, play, end) and any expansions the students offer.

[Slide with three phases - setup, play, end]

In fact, we could argue that *all* games involve a similar structure - a kind of loop.

[Slide of a game loop]

This is known as the *game loop* and it is a programming pattern we'll see again and again. Let's break down some of our favorite games and see how this structure applies to them.

Draw loops for board games, adventure games, real-time strategy games, networked games, etc.

Now, as we look through these examples, can we identify any major considerations?

[Turn-based vs. Real-time game]

What do we mean by a turn-based game loop? A real-time game loop?

Open for questions - we are looking for the concept of blocking.

[Turn-based games]

This is one of my favorite turn-based games, Civilizations 2. Gameplay is divided into turns, and each player gets to move every one of his pieces before the next player gets their turn, and play cycles around to all the players. Thus, while one player is taking his turn, the other players, and the simulation itself are *blocked*.

[Real-time games]

In contrast, in a real-time game the simulation advances regardless of the player taking action. The goombas keep patrolling - they don't stop and wait for Mario to move.

How would you implement a turn-based game?

Allow for answer

A real-time game?

Allow for answer

[Fixed-timestep vs. Variable timestep]

One of the more important questions in considering a real-time game is the timestep - should it be fixed or variable? Any thoughts on what I mean by that?

Allow for answers

[Example - Ballistic Motion]

Let's work out an example. Say we're creating a pocket-tanks like game, which involves calculating trajectories of missile weapons. Let's write a function to move each projectile.

Work out examples on the board. Show how variable timesteps require more information.

[Hybrid Game Loop]

In fact, most physics simulation simulation strategies work best with a fixed timestep, and behave unpredictably when faced with a variable time delta. One strategy for mitigating this issue is a hybrid game loop, where the physics is advanced multiple timesteps based on how much time has passed.

Draw table on board

[Variable update/render game loop]

Also, we often see strategies for managing computational demands by triggering updates and renders at different rates as the game loop progresses.

Draw table on board.

[Coding Game Loops]

Now let's take a look at how this could be implemented in code.

Work through a basic example of calling update and render at different rates in a language of the student's choice. Apply a fixed time step within a variable-rate loop.

[Javascript setTimeout game loop]

In JavaScript, we might use the setTimout function to set up a game loop. This function calls another function at some future point, which we express in milliseconds.

[JavaScript setInterval game loop]

Another approach would be to use setInterval, which works similarly to setTimeout but calls the function at the provided interval rate. We need to keep a handle to the interval object, so that we can stop it in the future.

[JavaScript recursive game loop]

You might be wondering why we didn't use a game loop like this one. What problem does this game loop pose for JavaScript programs that doesn't exist for other languages?

This game loop is blocking; as JavaScript runs in a browser, this will be interpreted as a non-responsive script. While this approach might work running as an interpreted script within a game engine or within Node.js, it violates the asynchronous assumptions of JavaScript coding.

[Why so Asynchronous?]

JavaScript is an asynchronous language by design - it performs much of its work through context switching, much like an operating system does. An asynchronous function, like setTimeout or setInterval, asks the browser/operating system to create an event at a set point in the future, and add it to the JavaScript event queue. This allows our code to keep running without any blocking.

Asynchronous programming is one of the core ideas behind JavaScript, but it can also be one of the tougher ones to get your head around. If this is the case for you, read some of the other course resources on asynchronous functions or find some resources online. And practice, practice, practice with asynchronous function calls.