Sardines: A Networked Game

[Introduction: explain concept of game]

Architecture

Sardines is built on a straightforward client-server architecture... Reasons for choice... Though the current system does not have many variables to keep track of, the server also provides a 'master state' for clients to work from. Citation about networking [peer-reviewed]...

This may not remain true at a larger scale, however. While it ultimately proved too ambitious for this assignment, the original idea for . Since only navigators would be able to see the surrounding world, they could in some ways act as middle men in the hybrid architecture set out in Figure [FIGURE]. By treating each navigator as a 'local' server for their crew of 4, ... This would, of course, be subject to further testing, as [carries issues of P2P?]... [In paragraph/figure, introduce idea of navigator 'standardising' actions of the crew...]

[GRAPHICSX: Handdrawn diagram of interpolation and prediction interaction?]

Protocols

Transport Layer TCP: Reliable, etc... As much is set out in RFC 798 , where John Postel introduces the protocol:.... Provide reliability paragraph?...

Why not consider UDP?... Position updates every 0.1s...[1]

Application Layer Data sent with `Packet` struct... Break down serialisation...

While there isn't the space to break down every protocol in precise detail... [List packets used and what each

-

As an example, consider how a player might join the lobby...

API

Sardines is built with C# in the Godot engine. It uses System.Net.Sockets to handle networking, and System.Runtime.InteropServices to serialize/deserialize packet structs. This report notes that...

Integration

Asynchronous I/O... Connection class...

Discuss: offline vs. online updates to position!

Prediction

As discussed under 'Architecture', clients only send position updates every $0.1s$. What this report has so far failed to consider is how this appears to other clients - they experience what should be a smooth, continuous movement as discontinuous jumps over $0.1s$ intervals! Clearly, ... [introduce prediction - with reading?].

[1] Could simplify even further with event-based - have server calculate all positions from key presses...

When a player chooses to move forward, they do not jump to a constant speed but continuously accelerate from zero; naturally, *Sardines* uses second-order quadratic prediction to best approximate the second-order derivative of accelaration. Given a submarine's three most recent positions $\mathbf{r}_0$, $\mathbf{r}_1$,$\mathbf{r}_2$ (corresponding to times $t_0 > t_1 > t_2$), clients can average the velocities from $\mathbf{r}_1$ to $\mathbf{r}_0$, from $\mathbf{r}_2$ to $\mathbf{r}_1$, and the acceleration from $\mathbf{r}_1$ to $\mathbf{r}_0$ as

$$\mathbf{u}_0 = \frac{\mathbf{r}_0 - \mathbf{r}_1}{t_0 - t_1}, \quad \mathbf{u}_1 = \frac{\mathbf{r}_1 - \mathbf{r}_2}{t_1 - t_2}, \quad \mathbf{a}_0 = \frac{\mathbf{u}_0 - \mathbf{u}_1}{t_0 - t_1}, \quad \text{respectively.}$$

These estimates define the quadratic model

$$\tilde{\mathbf{r}}(t) = \mathbf{r}_0 + \mathbf{u}_0 t + \mathbf{a}_0 t^2.$$

In contrast,

If prediction is the act of waiting for data, then integration is how one 'catches up' on receiving it. On receiving a new `PositionPacket` at time t_0, a programmer might be inclined to start predicting under to a new quadratic model $\tilde{\mathbf{r}}_{\text{new}}(t)$ immediately, but if positions $\tilde{\mathbf{r}}_{\text{old}}(t_0)$ and $\tilde{\mathbf{r}}_{\text{new}}(t_0)$ are visibly far apart, then the player will see the corresponding submarine make an instantaneous jump across the screen.[2] Instead, one takes a set time T to linearly interpolate from the old trajectory to the new:

$$\tilde{\mathbf{r}} = \begin{cases} \tilde{\mathbf{r}}_{\text{old}}(t) & \text{if } t < t_0 \\ (1 - q(t))\tilde{\mathbf{r}}_{\text{old}}(t) + q(t)\tilde{\mathbf{r}}_{\text{new}}(t) & \text{if } t_0 \leq t < t_0 + T \\ \tilde{\mathbf{r}}_{\text{new}}(t) & \text{if } t \geq t_0 + T \end{cases}, \quad \text{where } q(t) = \frac{1}{T}(t - t_0).$$

In *Sardines*' particular implementation, `PositionPackets` are sent via TCP every $0.1s$; interpolation therefore takes place over a strictly shorter interval $T = 0.05s$.

[GRAPHICSX: Handdrawn diagram of interpolation and prediction interaction?]

To fully understand how *Sardines* uses it prediction techniques, this report must first introduce a core challenge of any networked game: conflict resolution.

In *Sardines*, the projectiles concerned are soundwaves. The visual language of the game, where soundwaves from external sources only become visible on collision with the player, provides a clear approach: the sender unequivocally takes precedence. Only when a player sees their soundwave hit another is a `MorsePacket` sent from their client (which will arrive with the usual delay). The sender knows with certainty who receives their message; the receiver, who cannot see the trajectory of the soundwave until it arrives, will have no sense of whether it "should" have hit them.

To further 'smooth over' the application's conflict resolution, the receiving client makes use of backward prediction. Since neither server nor client stores more than three of any submarine's past positions at a time, it is fortunate the above formulae can approximate the past as well as the future.[3]

Suppose a sender emits a soundwave from position $\mathbf{r}$ at time t_0, which they see reach a receiver at $t_0 + \Delta t$. On the arrival of the corresponding packet at t_1, then, the receiving client has to decide where the wave was emitted from *in its local view of the game*. The obvious choice would be the 'true origin' $\mathbf{r}$, but *Sardines* uses the backwards prediction $\tilde{\mathbf{r}}(t_1 - \Delta t)$. As [FIGURE] puts it in

[2] This might be regarded interpolation over $T = 0.0s$!

[3] *Sardines*' submarines are physics-based objects, and at one point in development, the drag they experience was factored into prediction. However, the differential equations for 2D motion with a quadratic drag were too complex to find an analytic solution - rather than being able to substitute a t-value into a given equation, the prediction would be calculated over incremental, irreversible forward time steps - so the application sacrifices this more realistic model for the ability to look backwards in time.

[REFERENCE], [QUOTE]; conflict resolution is the art of deciding which quantities are preserved across clients, and *Sardines* - a system designed around slow, real-time communications - is far less concerned with a shared view of geography than it is a shared view of delay.

Testing

[SORT THIS LAST THING - BUT PLAN THE TESTING OUT BY 15th?]