1 Architecture

In our architecture we solve the issue of running a distributed drone simulation program.

We have made a number of decisions:

- There is a separate engine which models all physics and hitbox-detection. There are a number of engines able to already simulate this. The complexity of the project would otherwise be enormous. We could also have chosen to implement the physics distributively. This would complicate any kind of movement or hitbox-detection as everyone would have to talk to everyone in order if a certain movement would create a collision.

  Another reason for using a separate engine is that it results in a realistic approach. The engine will decide the ”real” movement while the drone can only change its physical attributes indirectly through actuators.

- The engine is the leader of information. It decides where a drone is and will publicize this in the data-store. The drone can then use this information to decide its next action. However, in certain situations it is necessary to change the reality as given by the engine.

  An example: When an asset is carried by a drone, it should have the same position as the drone. In the engine, it might have a position near the drone.

  Another example: A position scrambler is used by a drone. It will publicize a ”wrong” position while it will save the real position for the drone itself and friendly team-members.

  We have chosen to split the log, created by the engine, and the reality of the data / the state into separate data-sources. Each component can then subscribe on the generated log and write the correct data to the state.

- We shall use the components to transform engine log messages into state. Each component will have the ability to both publicize information about itself through etcd and also save the information itself.

- We have chosen to split the gamestate into a separate service. We will be able to implement rules through states and the gamestate service will be able to kill or spawn processes through the instance manager while also being able to manipulate the data in etcd. This separate the concerns between game concepts and environment concepts.

- A consequence of using the components to transform engine log messages to state in etcd is that only a single component will be able to write about a specified piece of information. If there are multiple writers, they will have to communicate with each other to decide to decide on the common answer. Otherwise, it would be considered a race condition as values from one writer will be overwritten by the other.

2 Risks

- Hacking is out of scope - Hacking of drones between drones is out-of-scope as this would change the reality for the underlying components. A possibility to implement this would be through a second step of processing from log engines messages to malicious messages which then in turn could be transformed to realstate by the targeted drone components.
- Bottleneck in feedback loop - This architecture contains a feedback loop where drones will change the results of the engine based on the results from the engine. So far there is no central synchronization as this would defeat the purpose of the realistic realtimeness of the simulation. However, if the loop is too slow, the drones would not be able to compensate for any reactions within the engine. For instance: If the drone is moving towards another, he would not be able to correct course if it takes half a second to update the position of the drone.

For our architecture we have a number of implementation choices: