Proposal for Game Environment

September 29, 2015

Goal: Our goal is to build a game-like environment in which to train neural networks. Specifically, we would like to have a flexible 3d rendered environment in which a neural-network "agent" can operate and learn. The axes on which we're trying to design the environment are: (i) rendering quality, (ii) physics realism, (iii) interaction complexity, (iv) procedural generation, (v) tunability/extensibility, and (vi) speed.

Inspiration: We're inspired by thinking about what would engage babies. Ideally, our environment would be something that, if a baby weer in this environment, they'd find a lot of things that were entertaining for them to do. As a result, we're not constrained by making the kinds of interactions and assets that would needed to engage the attention / believability of an adult.

Requirements

Rendering Quality

We have a significant need for the graphics to be fairly realistic. The basic reason for this is that we would like the neural networks that we train using this environment to be able to generalize to images from the real world. As a result, the graphics in Catlateral Damage (for example) are a good start but probably not realistic enough. This said, we probably don't have to be totally overboard here. I think it would be nice if we could have transparent objects that scatter light in a somewhat believable way. However, we probably don't need perfectly realistic light interactions, so we could compromise a bit here.

Physics Realism

We would certainly like to have a pretty reasonable basic rigid-body physics as part of this environment. This is probably easy given the existing tools. There are a couple of questions, however, about what to do here that we don't totally know the answer to yet. Specifically:

- Staying within the realm of rigid-body physics, how detailed are the interaction and collision volumes that we'll be able to use? For example, we have a bunch of high-quality 3d face objects at our disposal. It would be cool if we could have a scenario in which the agent is touching these face and learning their contours. Will the 3d collision objects be able to be detailed enough to capture the structure of the 3d face mesh, rather just (e.g.) a sphere or cylinder? What is the right for us to be able to introspect / control the nature of the collision objects and their level of detail?

- Will we be able to have any softbody physics? How hard would it be? We're not really sure we need it.
- It would be nice if there were a connection between bumpiness of textures in images and physics friction coefficients. Is this hard? Again, it might not be necessary.

Interaction Realism / Complexity

Acting on the Environment: We like to enable the agent to interact with the environment without having to resort to "magical" control. Specifically, it would be nice if we had a clear way to map a variables describing the way the agent exerts force onto the environment. Specifically:

- The *easiest* non-trivial scenario that we'd like to enable is something like an agent push an object along a table until it falls off. This is a bit like the interactions in Catlateral Damage except that in Catlateral Damage the interactions felt less "real" that we'd like – somehow, the nature of the interface between the controller and 3-D objects wasn't tuned exactly the way we'd want. We'd probably have to go into more detail about this in person.
- A "medium" complexity scenario that we'd like to be able to support eventually is the putting of one object into another one — e.g. placing a ball into a box. It would be especially nice if this box could be realistically transparent (see previous section) but probably not necessary.
- The *most complex* physical interaction scenario that we'd like to enable is something like: the ability to open a door by turning a round doorknob.
- How hard would it be for us to enable new kinds of interactions? I guess this may relate to the types of (player) assets that we use.

Feedback From the Environment: We'd like to be able to get force feedback from the environment to use it in training the agents. This is probably as simple as being able to introspect the collision objects for their parameters — however, we just don't know enough about the details of this question to know exactly what the standard physics engines give access to, or how hard it would be to map variables to deliver simple force feedback.

Procedural Generation

We'd like to be have environments generated in a procedural fashion so that we could easily configure a large number of different environments. This is a bit like how Catlateral Damage already works. We also really like the work being done in No Man's Sky by Hello Games in this regard. However unlike these really good examples, we're actually not constrained by the requirement that the environments seem "realistic" to an adult human. Having weirdly assembled environments with piles of stuff that aren't quite "right" is actually fine for us.

Tunability, Extensibility & Communicability

We'd like the environment to be tunable and extensible. Specifically:

- Tunability for realism. It would be great if there were software settings that would allow us to easily tune up and down the amount of realism along several axes, e.g. rendering realism, physical interaction, realism, etc.
- Tunability for hardness of tasks in the environment. Not sure how this would work, but it would be nice if we could specify something in the procedure generation of the environment that would make the environments harder or easier to navigate or understand (e.g. by "understand" I mean, be able to predict the future states of the objects under the physical dynamics).
- Extensibility in terms of assets. It would be nice if there were a simple for us to programmatically add new assets and still have the game environment work well and integrate these assets properly. For example,it would be nice if the assets were accepted by the game environment in a standard enough format that we could go out onto some market (like the Unity store) and buy new assets that we could integrate, without too much effort.
- Extensibility in terms of general coding. We need to integrate this coding environment into other programs that we right. It's important that software engineering choices be made so that the code can be most easily integrate. E.g. we'd like to be able easily run it in a headless setup, in parallel, with some kind of fairly simple human-readable input-output format.
- Communicability: It needs to be really easy and fast for us to instrument the game environment code to communicate with other processes running separately, e.g. via sockets. Basically, we need to be able to easily intervene in various places in the game code to get data in and out without it being a major project and without breaking everything.

Speed

The environment game loop needs to be a pretty fast, because we need to be able to integrate it into our inner training loop. Ideally we could get several hundred frames per second. Thus, we'd be willing to compromise on realism in any of the above areas to achieve speed. In a way the key art of this project will be knowing which comprises to make and which things we can "cheat on" to make something that like pretty good while still maintaining the kind of performance we want.

Practically Getting There

I think we'll probably want to do the project in a set of simple stages, progressing from having a very simple version of the above delivered first, to having the more complex aspects of it solved later. We'd like to get the project started pretty fast, even if that means not having all the features described above completely finished (as long they could potentially be added later).