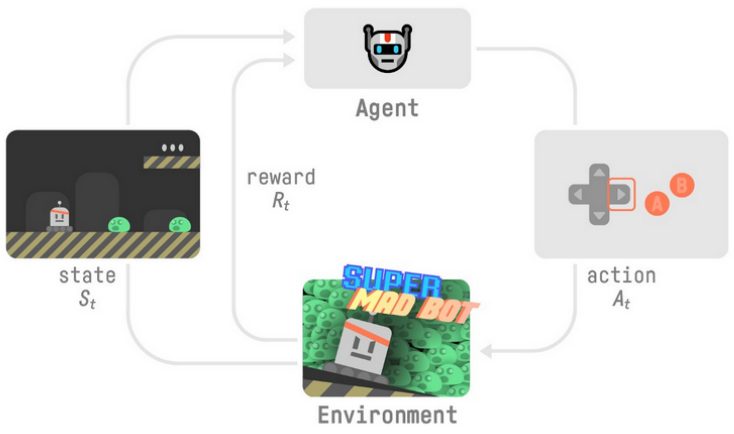
Establih ML Modeling Baseline

3D bin-packing team: Yueqi Peng, Bryan Boyett, Pano Evangeliou

The 3D bin packing problem is an np-hard variable-size permutation-invariant combinatorial optimization problem. To model it we train a Reinforcement Learning (RL) agent in a simulated environment. The idea is that the RL-agent will learn from the environment by **interacting with it** (through trial and error) and **receiving rewards** (negative or positive) as feedback for performing actions.

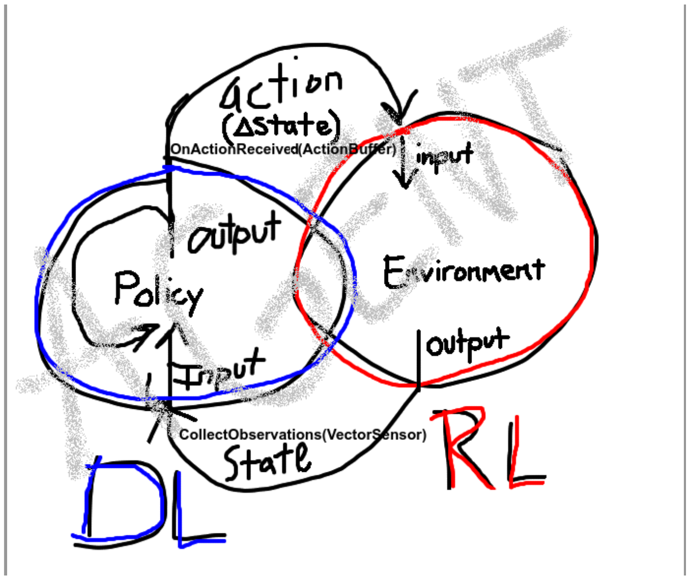


- Our Agent receives state S_0 from the Environment — we receive the first frame of our game (Environment).
- Based on that state S_0, the Agent takes action A_0 — our Agent will move to the right.
- Environment goes to a new state S_1 — new frame.
- The environment gives some reward R_1 to the Agent — we're not dead (Positive Reward +1).

After this gentle intro let's explain our RL model.

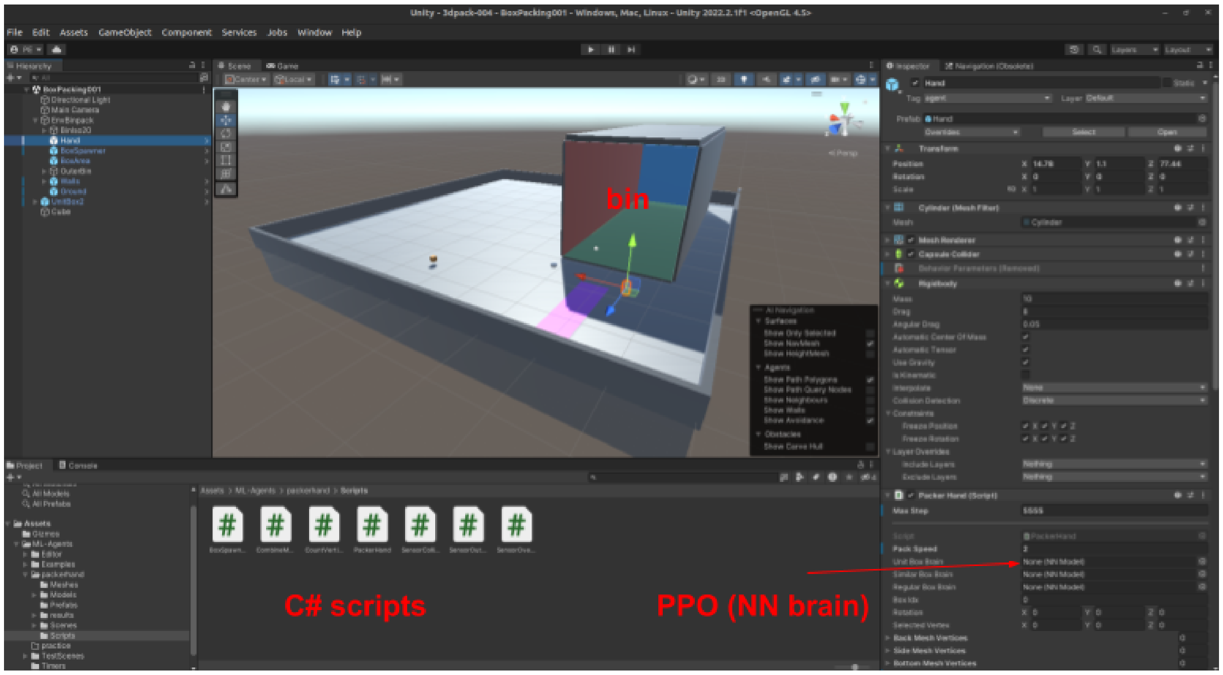
1. Establish baseline model

In summary, we train a Deep Reinforcement Learning (DRL) agent using a Proximal Policy Optimization (PPO) inside a Unity environment. The reward policy is dense including also penalties. The dedicated Unity enviroment is custom-made through C# scripting. The goal is to teach the agent to pack as many boxes into the bin as possible.

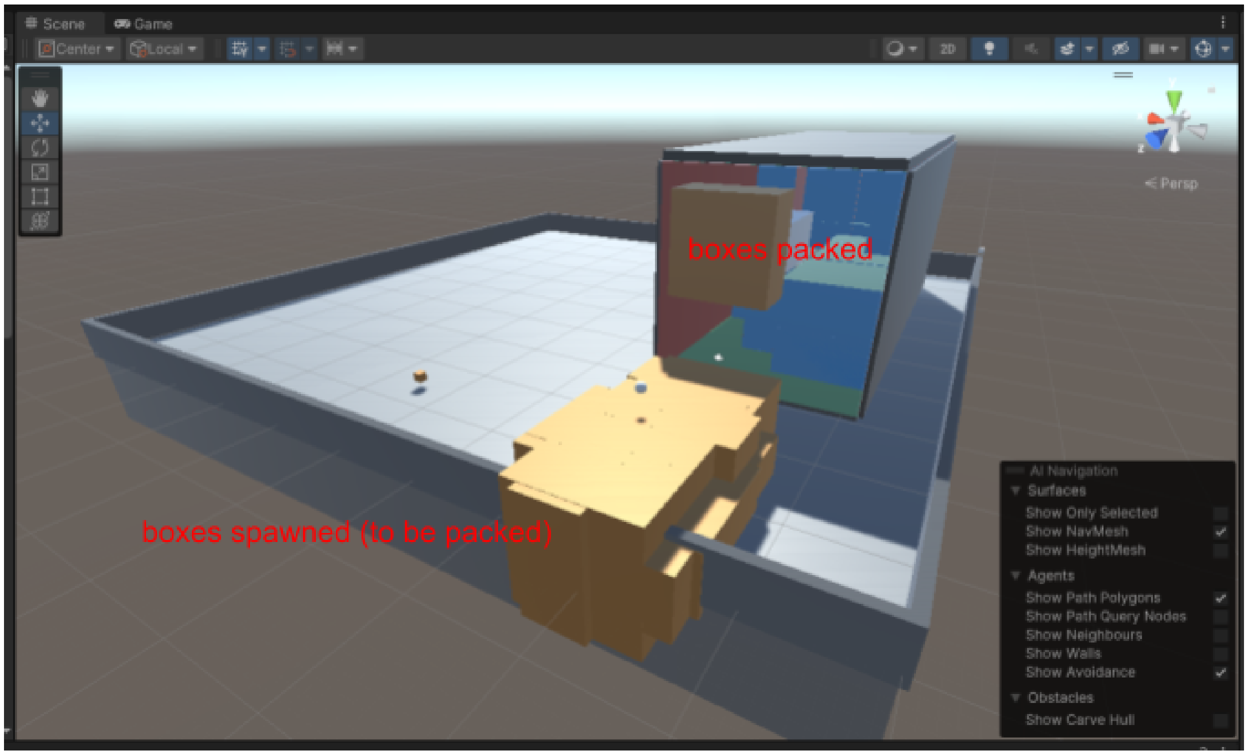


(we loved this scribble from the 1st moment)

We design our evironment in the **Unity** game engine. The physics simulation and the scripting environment provide a lot of possibilities to create a realistic box-packing environment. The specific encific environment is made using C# scripting.



We use **Unity ML-Agents** to model our agent. Our agent uses a policy-gradient deep neural netrowk to model the probability of taking an action given a corresponding state in the environment. The specific policy used is the Proximal Policy Optimization (PPO). The deep neural network will be called *Brain* from now on.



The workflow is the following:

I. State S_0
   A. Boxes are spawned in the spawning area outside the bin
   B. The initial *Observations Vector* is formed. It contains the possible positions to place the next box, the boxes to be placed and their rotations.

II. Agent
   A. Collects the *Observations Vector* from the state S_0.
   B. The PPO initializes the weights of the DNN brain.

III. Action
   A. The agent selects a box from the *Observations Vector*
   B. The agent selects a rotation from the *Observations Vector*
   C. The agent selects a position from the *Observations Vector*

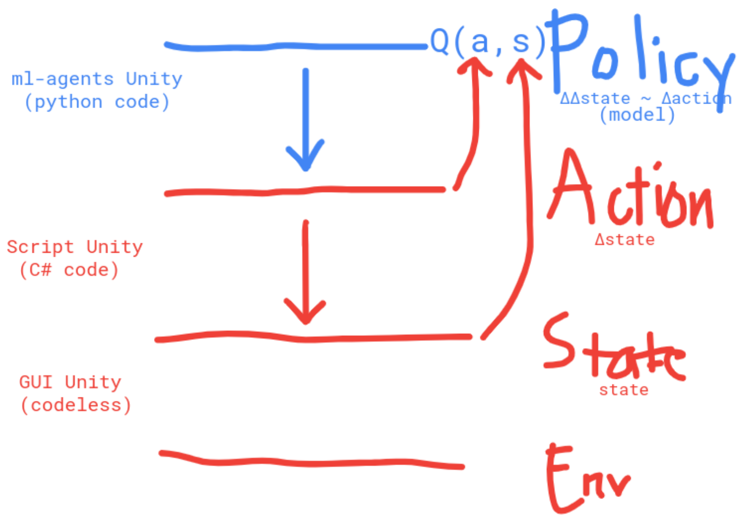
IV. The selected box is placed at the selected position inside the bin, with the selected rotation.

V. State S1
   A. The placed box is merged with the bin mesh
   B. The topology inside the bin is updated. The box is now placed. The *Observations Vector* is modified accordingly (new available positions, one less box, 6 rotations)
   C. A reward is given based on the new state in the environment.

VI. Agent
   A. Based on the reward, the agent updates the weights of the Brain.
   B. The agent collects the new *Observations Vector*

VII. Repeat steps (III) - (VI)

VIII.



Essentially, the features used for training the *Brain* of the ML-Agent are contained in the *Observation Vector* and consist of:

- The available positions vector

This is a variable size vector that contains the vertices of the topology inside the bin. The topology changes after placing each box inside the bin. Thus, the vector size changes as well. This vertices act

- The spawned boxes vector

  This vector contains all the boxes to be fitted to the bin. Its size is also reduced after is box is placed inside the bin.
- The rotations vector

  Always 6 available rotations since we are only fitting cubic boxes for our baseline model

Reward policy: We currently a ”Dense” (instead of “Sparse”) reward system. This means that we reward the agent frequently, ie:

- When a box is placed inside the bin (positive reward)
- When a box is adjacent to as many surfaces as possible to ensure tight packing (positive reward)
- When a box is misplaced (negative reward)

2. Organize code in scripts

The code is organized in the following script:

- Unity C# scripts

  These are custom-made scripts that create and control the environment and feed the data to the *Observations Vector.* Then, they connect the *Brain* to the *ML-Agent* scripts that carry out the training.

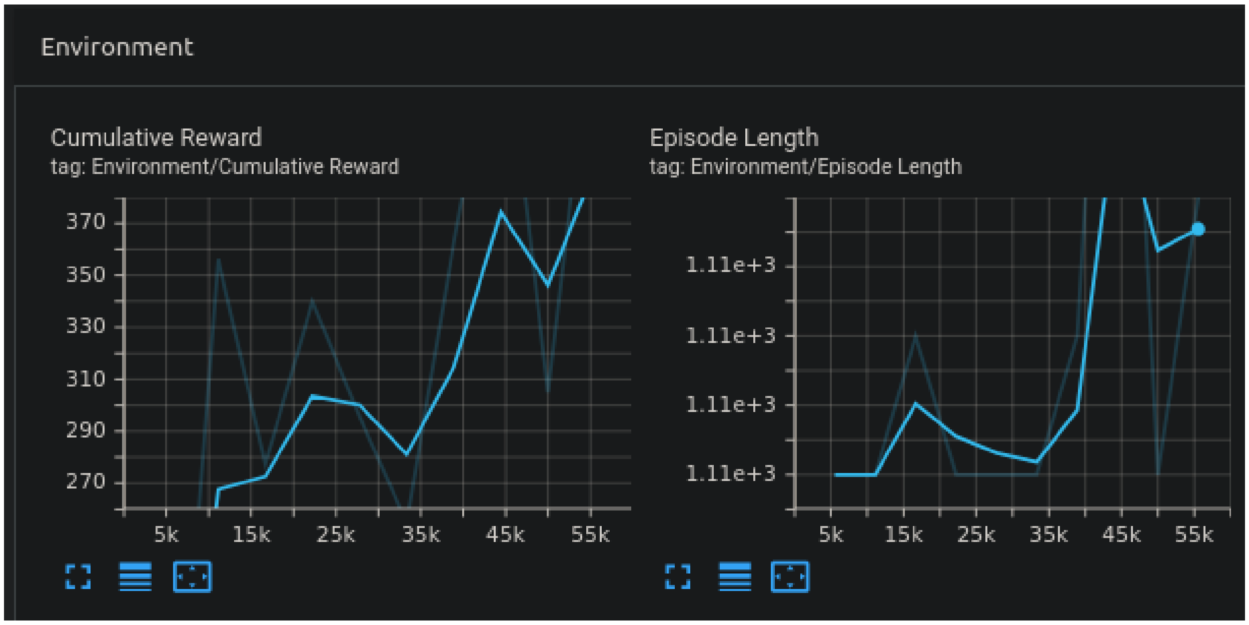
  The C# scripts can be found here.
- ML-Agents scripts

  These are the existing scripts of the ML-Agents implementation.

  The currently implemented *Brain* PPO can be found here.

3. Document performance report in markdown

After initial training of our baseline model, the following results are acquired in a *Tensorboard* format.



The cumulative reward graph shows that hte agent is actually training. THe discontnuities in the training curve are caused because of the variable length feature vector and because of the complexity of the problem. The PPO - though a state-of-the-art benchmark policy - is ideal to handle variable-sized feature vectors. For this reason we will probably adopt to variable-sized permutation invariant Transformers. Moreover, an added complexity is the variable size of the boxes and the numerous actions available to the agent to decide on.

Still, the cumulative reward graph clearly shows our agent being improved during the training epochs, selecting constantly higher rewards and learning to pack boxes more efficiently.