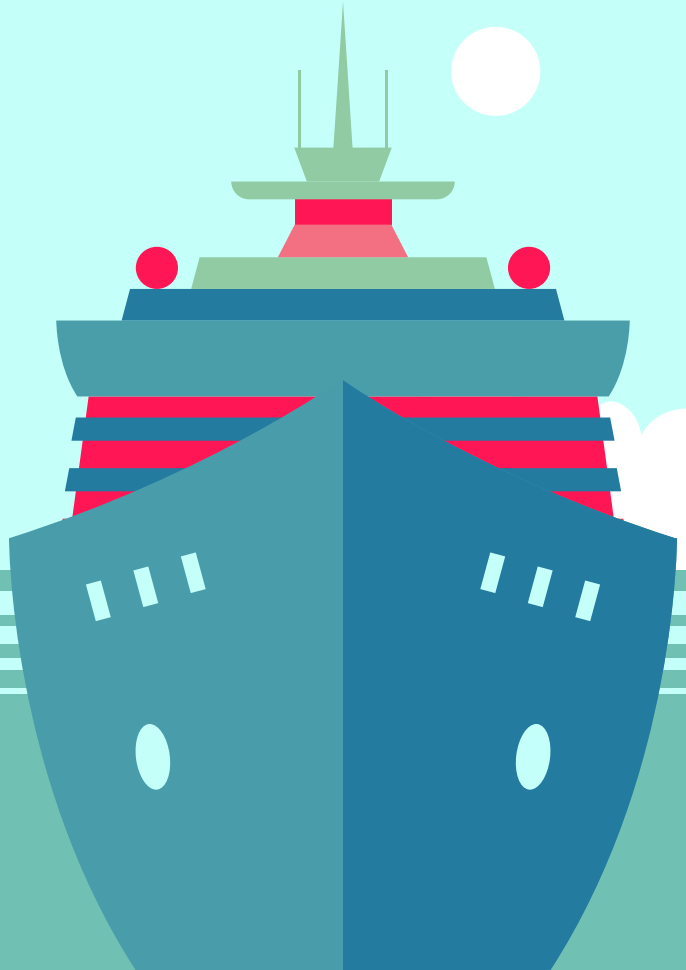


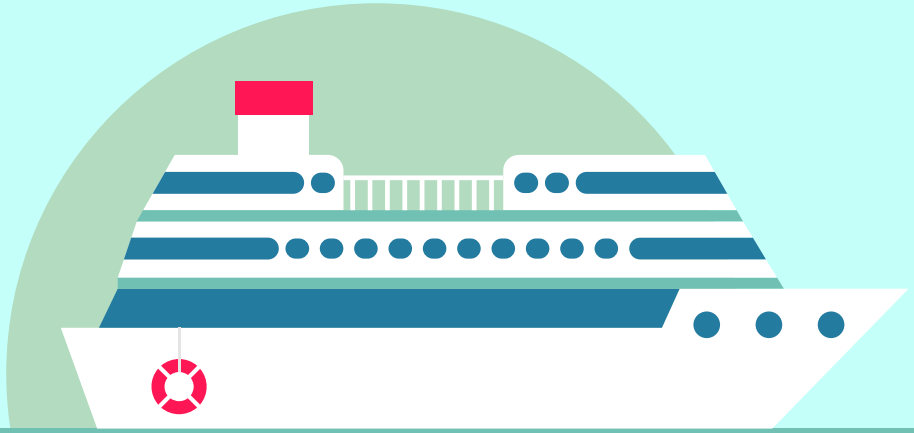
Packing 3D container with
Deep Reinforcement Learning

Team 6: Seal is Cute

牟展佑
王子文
蔣立元

翁玉芯
黃雅筠

TABLE OF CONTENTS



01
ABOUT US

OUR TEAM

Team Members:

蔣立元
清大資工23級

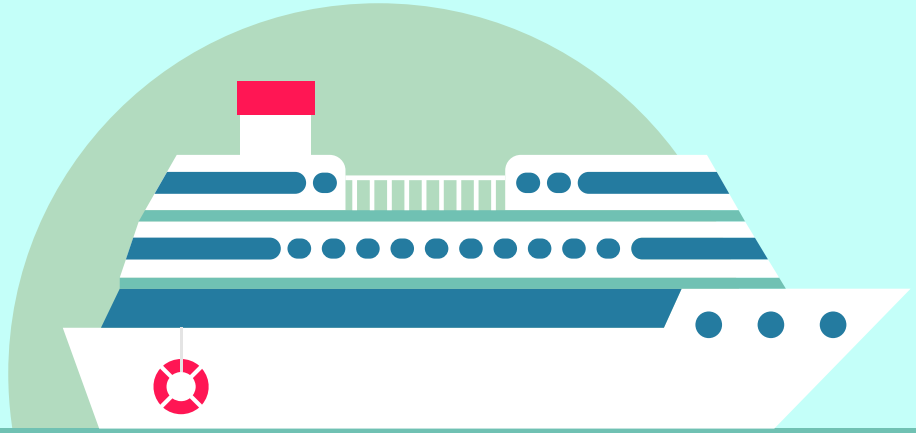
牟展佑
清大資工23級

王子文
清大電資22級

Seal is Cute

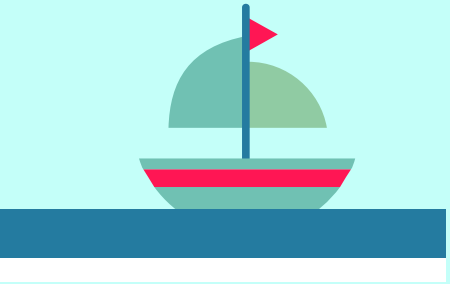
黃雅筠
清大資工24級

翁玉芯
清大資工23級



02
PIPELINE

OUR EVOLUTION



Input

Generate Box Data with JSON

01

Select

Choose the Corresponding Model

02

Preprocessing

Sorting Box According Priority

03

OUR EVOLUTION

AI

Pack Bin
Container with RL

04

Output

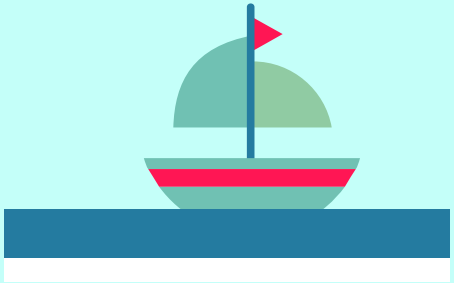
Output Layer
with Json

05

Visualize

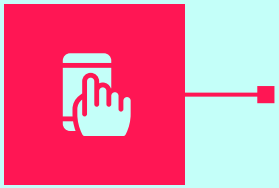
Visualize with
three.js Render

06



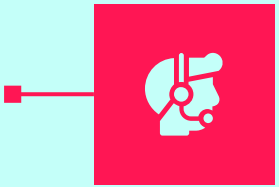
What we Achieved

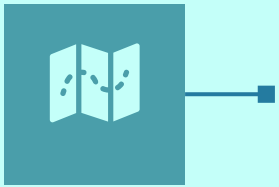
Realtime
AI Model



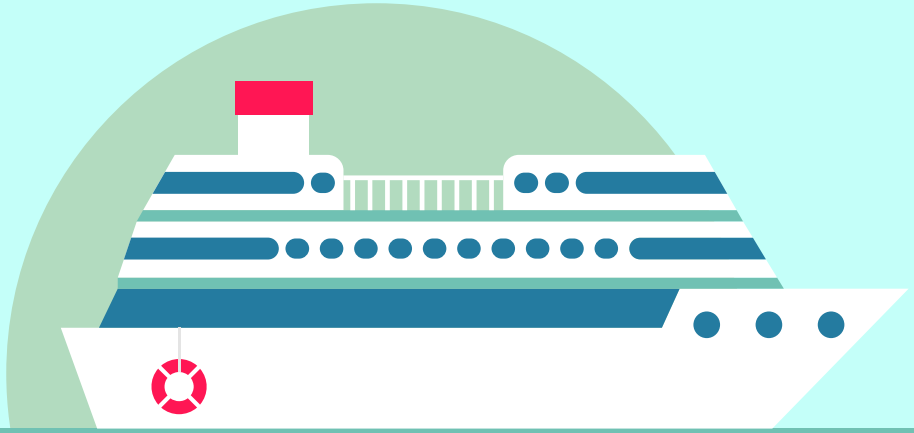
Avoid
floating object

GPU
Acceleration





Develop
Rendering Page



03
BACKEND

Online 3D Bin Packing with Constrained Deep Reinforcement Learning

Hang Zhao[1], **Qijin She**[1], **Chenyang Zhu**[1], **Yin Yang**[2], **Kai Xu**[1,3*]
[1]**National University of Defense Technology,** [2]**Clemson University,** [3]**SpeedBot Robotics Ltd.**

Abstract

We solve a challenging yet practically useful variant of 3D Bin Packing Problem (3D-BPP). In our problem, the agent has limited information about the items to be packed into a single bin, and an item must be packed immediately after its arrival without buffering or readjusting. The item's placement also subjects to the constraints of order dependence and physical stability. We formulate this *online 3D-BPP* as a constrained Markov decision process (CMDP). To solve the problem, we propose an effective and easy-to-implement constrained deep reinforcement learning (DRL) method under the actor-critic framework. In particular, we introduce a *prediction-and-projection* scheme: The agent first predicts a feasibility mask for the placement actions as an auxiliary task and then uses the mask to modulate the action probabilities output by the actor during training. Such supervision and projection facilitate the agent to learn feasible policies very efficiently. Our method can be easily extended to handle lookahead items, multi-bin packing, and item re-orienting. We have conducted extensive evaluation showing that the learned policy significantly outperforms the state-of-the-art methods. A preliminary user study even suggests that our method might attain a human-level performance.

1 Introduction

As a classic NP-hard problem, the bin packing problem (1D-BPP) seeks for an assignment of a collection of items with various weights to bins. The optimal assignment houses all the items with the fewest bins such that the total weight of items in a bin is below the bin's capacity c (Korte and Vygen 2012). In its 3D version i.e., 3D-BPP (Martello, Pisinger, and Vigo 2000), an item i has a 3D "weight" corresponding to its length, l_i, width w_i, and height h_i. Similarly, c is also in 3D including $L \geq l_i, W \geq w_i$, and $H \geq h_i$. It is assumed that $l_i, w_i, h_i, L, W, H \in Z^+$ are positive integers. Given the set of items $\mathcal{I}$, we would like to pack all the items into as few bins as possible. Clearly, 1D-BPP is a special case of its three dimensional counter part – as long as we constrain $h_i = H$ and $w_i = W$ for all $i \in \mathcal{I}$, a 3D-BPP instance can be relaxed to a 1D-BPP. Therefore, 3D-BPP is also highly NP-hard (Man Jr, Garey, and Johnson 1996).

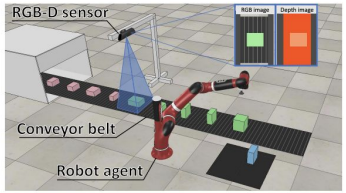


Figure 1: Online 3D-BPP, where the agent observes only a limited numbers of lookahead items (shaded in green), is widely useful in logistics, manufacture, warehousing etc.

problem) as many real-world challenges could be much more efficiently handled if we have a good solution to it. A good example is large-scale parcel packaging in modern logistics systems (Figure. 1), where parcels are mostly in regular cuboid shapes, and we would like to collectively pack them into rectangular bins of the standard dimension. Maximizing the storage use of bins effectively reduces the cost of inventorying, wrapping, transportation, and warehousing. While being strongly NP-hard, 1D-BPP has been extensively studied. With the state-of-the-art computing hardware, big 1D-BPP instances (with about $1,000$ items) can be exactly solved within tens of minutes (Delorme, Iori, and Martello 2016) using e.g., integer linear programming (ILP) (Schrijver 1998), and good approximations can be obtained within milliseconds. On the other hand 3D-BPP, due to the extra complexity imposed, is relatively less explored. Solving a 3D-BPP of moderate size exactly (either using ILP or branch-and-bound) is much more involved, and we still have to resort to heuristic algorithms (Crainic, Perboli, and Tadei 2008; Karabulut and İnceoğlu 2004).

Most existing 3D-BPP literature assumes that the information of all items is known while does not take physical stability into consideration, and the packing strategies allow backtracking i.e., one can always repack an item from the bin in order to improve the current solution (Martello, Pisinger, and Vigo 2000). In practice however, we do not

Online 3D Bin Packing with Constrained Deep Reinforcement Learning

https://arxiv.org/pdf/2006.14978.pdf

Input/Output

Input

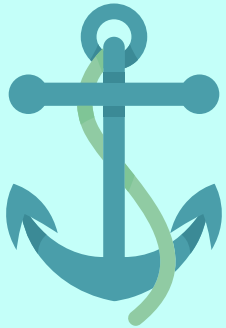
- List of Boxes:
  [[B1w, B1l, B1h], [B2w, B2l, B2h], ...]
- Container Size:
  (Cw, Cl, Ch)

Output

- Procedure of placing boxes
- Each state is a w*l matrix
- There are # boxes states in total

```
Place box [4, 4, 2]
[[2 2 2 2 0 0 0 0 0 0]
 [2 2 2 2 0 0 0 0 0 0]
 [2 2 2 2 0 0 0 0 0 0]
 [2 2 2 2 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
Place box [4, 4, 4]
[[6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
Place box [4, 4, 4]
[[6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [4 4 4 4 0 0 0 0 0 0]
 [4 4 4 4 0 0 0 0 0 0]
 [4 4 4 4 0 0 0 0 0 0]
 [4 4 4 4 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

```
Place box [3, 3, 3]
[[9 9 9 6 0 0 0 0 0 0]
 [9 9 9 6 0 0 0 0 0 0]
 [9 9 9 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [4 4 4 4 0 0 0 0 0 0]
 [4 4 4 4 0 0 0 0 0 0]
 [4 4 4 4 0 0 0 0 0 0]
 [4 4 4 4 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
Place box [4, 4, 2]
[[9 9 9 6 0 0 0 0 0 0]
 [9 9 9 6 0 0 0 0 0 0]
 [9 9 9 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
Place box [5, 3, 3]
[[9 9 9 6 3 3 3 0 0 0]
 [9 9 9 6 3 3 3 0 0 0]
 [9 9 9 6 3 3 3 0 0 0]
 [6 6 6 6 3 3 3 0 0 0]
 [6 6 6 6 3 3 3 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [6 6 6 6 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

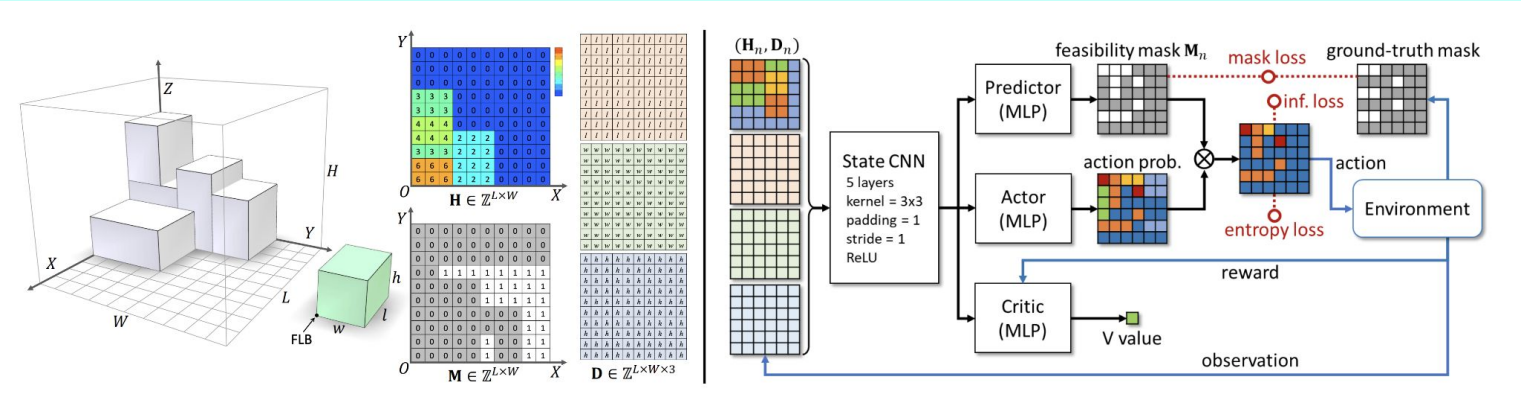


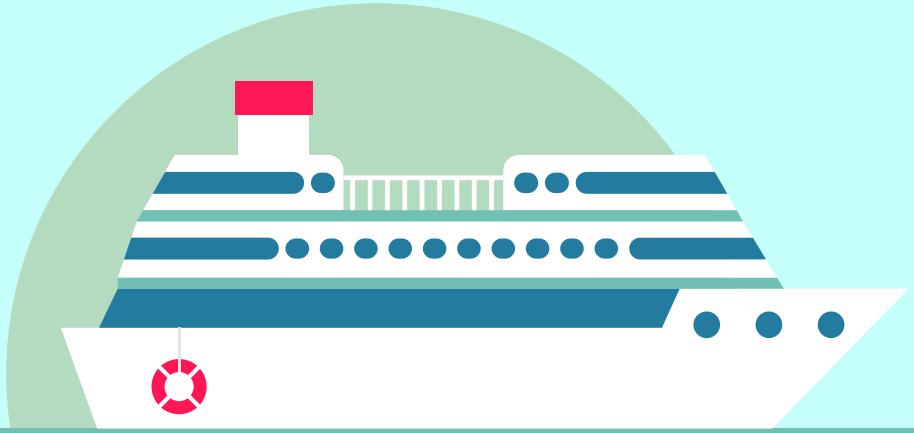
10s

For predicting a 10x10 Container with 5 step lookahead

Model Training

- actor-critic framework with Kronecker-Factored Trust Region (ACKTR)
- Input current state and the coming boxes
- Take longer to predict if number of lookahead boxes increase, but with better quality

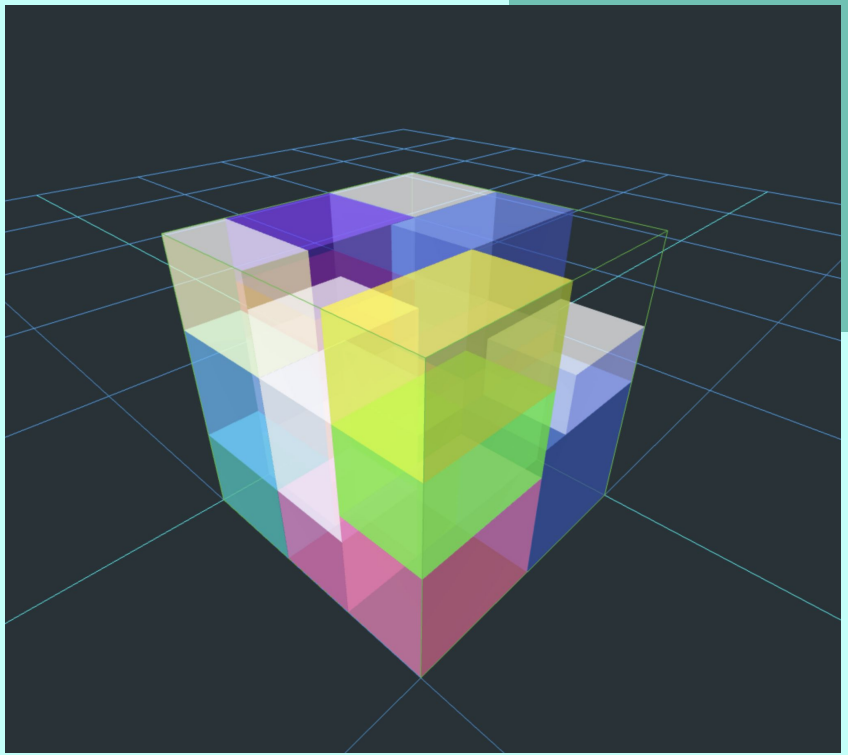




04
FRONTEND

Develop Rendering Page

Three.js rendering of container with boxes.



Transform Layer to Placement

- Create Placement Class
- Transform layer data structure to placement date structure
- Easy to operatore

```python
class Placement:
    def __init__(self, step) -> None:
        self.step = step
        self.x = None
        self.y = None
        self.z = None
        self.dx = None
        self.dy = None
        self.dz = None


def transform() -> Placement:
    placements = []
    containers = np.load("obs_list.npy")
    containers = np.insert(containers, 0, np.zeros((10, 10)), 0)

    for each_layer in range(len(containers)-3):
        layer = containers[each_layer+1] - containers[each_layer]
        placement = Placement(each_layer + 1)
        for i in range(N-1, -1, -1):
            for j in range(0, N):
                if layer[i][j] != 0:
                    placement.x = i
                    placement.y = j
                    placement.z = containers[each_layer][i][j]
                    break
        placements.append(placement)

    boxs = np.load("box_list.npy")[:-1]
    for i, placement in enumerate(placements):
        box_xyz = boxs[i]
        placement.dx = box_xyz[0]
        placement.dy = box_xyz[1]
        placement.dz = box_xyz[2]
```

Export Placement to Json

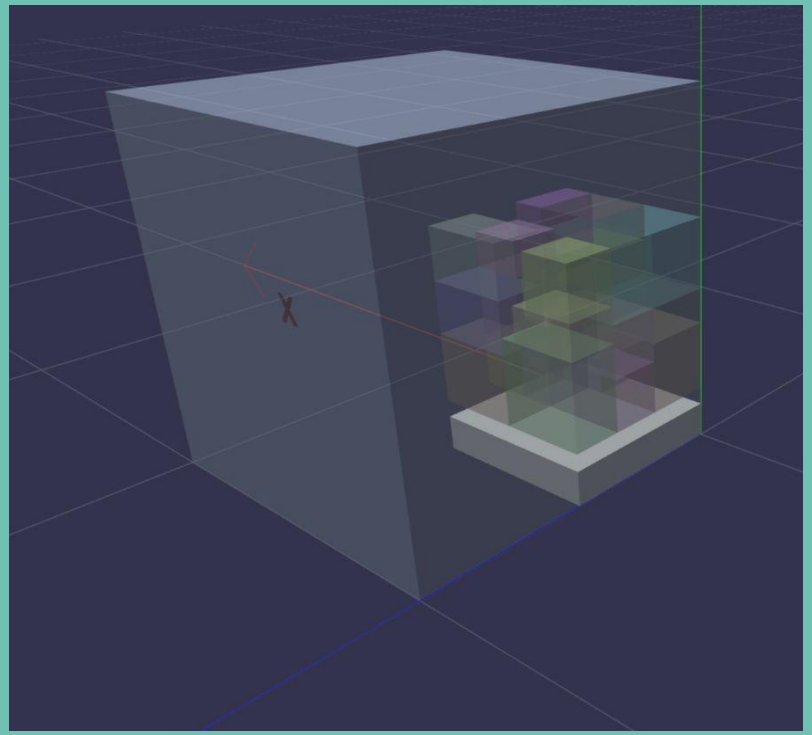
- Make a tmp dictionary
- Insert it to sample.json
- Output render.json

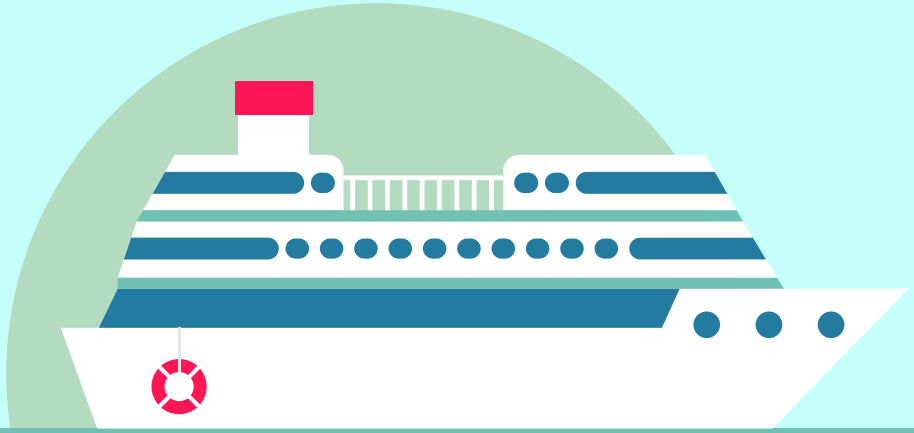
```python
def save_to_json(placements) -> None:
    sample_place = {
        "step": 2,
        "id": 0,
        "name": 0,
        "plugins": [],
        "x": 0,
        "y": 0,
        "z": 0,
        "stackable": {
            "step": 2,
            "id": "4",
            "name": "4",
            "plugins": [],
            "dx": 4,
            "dy": 3,
            "dz": 1,
            "type": "box"
        }
    }

    with open('container_sample.json', 'r') as f:
        data = json.load(f)
        data_place = data["containers"][0]["stack"]["placements"]
        for i, placement in enumerate(placements):
            this_place = copy.deepcopy(sample_place)
            this_place["step"] = int(placement.step + 1)
            this_place["x"] = int(placement.x)
            this_place["y"] = int(placement.y)
            this_place["z"] = int(placement.z)
            this_place["stackable"]["dx"] = int(placement.dx)
            this_place["stackable"]["dy"] = int(placement.dy)
            this_place["stackable"]["dz"] = int(placement.dz)
            this_place["stackable"]["step"] = int(placement.step + 1)
            this_place["stackable"]["id"] = str(int(placement.step + 1))
            this_place["stackable"]["name"] = str(int(placement.step + 1))
            data_place.append(this_place)
```

Better than Original

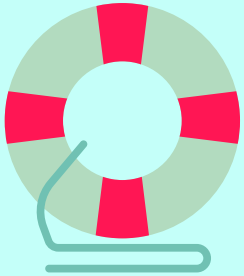
- More efficient and smoother
- Simplify some parameters for implementation
- Keep the same type, same color and other important functions





05
Future

WHAT SETS US APART?

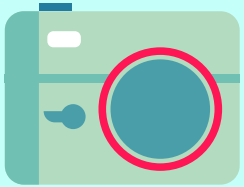


Completeness

Packaged into Docker container for easy use

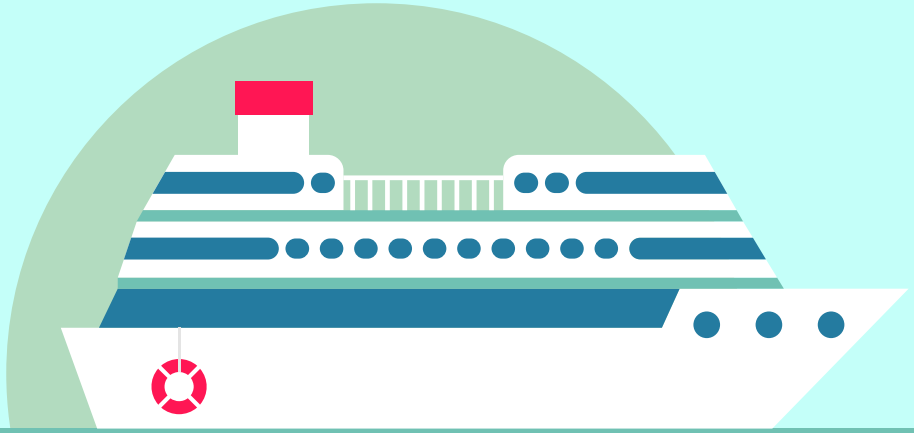
Visualize

Get better user interface.



Weight balance

The weight difference between the two ends of the pallet does not exceed 15%



06

Code & Ref

BUYER JOURNEY

01

Our GitHub Link

https://github.com/engine210/3D-Packing_DRL

02

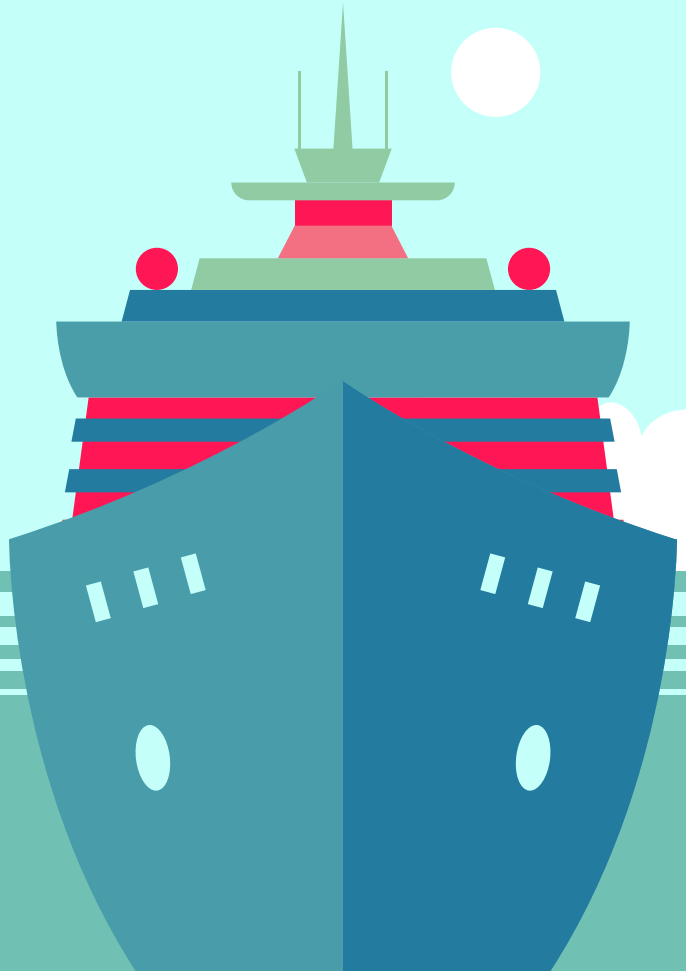
DRL Paper Reference

https://arxiv.org/pdf/2006.14978.pdf

03

Visualization Viewer

https://github.com/skjolber/3d-bin-container-packing/tree/master/visualization/viewer



Thank You

Team 6: Seal is Cute

牟展佑
王子文
蔣立元

翁玉芯
黃雅筠