BooleanStorageYard

A high-performance container storage yard implementation using boolean mask operations for optimal spatial reasoning and container movement calculations.

Core Concept

The `BooleanStorageYard` represents a container terminal storage area using **boolean masks** for ultra-fast spatial operations. Instead of traditional object-based representations, it uses numpy boolean arrays to track container placement, special areas, and movement possibilities.

Key Innovation: Mask-Based Spatial Representation

```
Dynamic Yard Mask: True = Empty, False = Occupied
Special Area Masks: True = Valid for container type
Result = Special_Mask & Dynamic_Mask & Stacking_Rules
```

This approach enables **vectorized operations** on entire yard sections simultaneously, dramatically improving performance over iterative approaches.

Architecture

Coordinate System

- **4D coordinate space**: `(row, bay, split, tier)`
- **Split factor**: Sub-divides bays for precise container placement (default: 4)
- **Tier system**: Vertical stacking levels (ground = 0, up to max_tier_height)

Mask Types

- `dynamic_yard_mask`: Real-time occupancy state (True = available)
- `r_mask`: Reefer container designated areas
- `dg_mask`: Dangerous goods container areas
- `sb_t_mask`: Swap body and trailer areas
- `reg_mask`: Regular container areas (complement of special areas)
- `cldymc`: Container-length-specific dynamic masks for stacking rules

Container Length Handling

Different container types occupy different numbers of split positions:

- **TWEU (20ft)**: 2 splits
- **THEU (30ft)**: 3 splits
- **FEU (40ft)**: 4 splits
- **FFEU (45ft)**: 5 splits (cross-bay spanning)
- **Trailers/Swap Bodies**: 4 splits

Key Methods

Core Operations

```
__init__(n_rows, n_bays, n_tiers, coordinates, split_factor=4)
```

Initializes yard with specialized area configuration and boolean mask setup.

```
add_container(container, coordinates)
remove_container(coordinates)
```

Basic container placement/removal with automatic mask updates and tier unlocking.

Optimized Placement Finding

```
search_insertion_position(bay, goods, container_type, max_proximity)
```

High-performance placement finder using:

- Vectorized mask operations
- Bit manipulation for container fitting
- START/END positioning rules
- Cross-bay spanning support

Returns valid placements as `(row, bay, tier, start_split)` tuples.

Ultra-Fast Movement Calculation

```
return_possible_yard_moves(max_proximity=5)
```

Breakthrough optimization for finding all possible container moves:

Algorithm:

1. **Vectorized detection**: `coordinates[dynamic_yard_mask]` finds all empty positions
2. **Simple logic**: If `tier > 0`, movable container exists at `tier-1`
3. **Container grouping**: Group coordinates by container object
4. **Batch processing**: Call optimized search for each unique container

Performance: ~4,000 containers/second with linear scaling.

Utility Methods

```
get_container_coordinates_from_placement(placement, container_type)
```

Converts placement tuples to full coordinate lists for container operations.

```
_find_valid_container_placements(available_coordinates, container_type)
```

Bit manipulation core: Uses binary operations to find valid consecutive positions for containers.

Performance Characteristics

Scaling Performance

- **Linear time complexity** for movement calculations
- **O(1) mask operations** for area validation
- **Vectorized numpy operations** eliminate Python loops
- **~0.25 seconds** to process 1,000+ containers

Memory Efficiency

- **Boolean arrays** instead of object collections
- **Coordinate mapping** eliminates string operations
- **Cached computations** for repeated calculations

Integration Points

Container Factory Integration

Works seamlessly with existing `Container` and `ContainerFactory` classes:

```
container = ContainerFactory.create_container("ID", "FEU", "Import", "Reefer")
placement = yard.search_insertion_position(bay=10, goods='r',
container_type='FEU', max_proximity=5)
coordinates = yard.get_container_coordinates_from_placement(placement[0], 'FEU')
yard.add_container(container, coordinates)
```

Terminal Environment Integration

Designed for reinforcement learning environments requiring:

- **Fast action space generation**
- **Real-time movement validation**

- **Batch proximity calculations**
- **GPU-compatible tensor operations**

Usage Example

```python
# Initialize 5x58 yard with 5 tiers, specialized areas
yard = BooleanStorageYard(
    n_rows=5, n_bays=58, n_tiers=5,
    coordinates=[
        (1, 1, "r"), (58, 1, "r"),      # Reefer areas
        (28, 3, "dg"), (29, 3, "dg"),   # Dangerous goods
        (10, 1, "sb_t"), (15, 1, "sb_t") # Trailer area
    ],
    split_factor=4
)

# Find all possible container moves (ultra-fast)
possible_moves = yard.return_possible_yard_moves(max_proximity=5)

# Place container optimally
placements = yard.search_insertion_position(bay=20, goods='reg',
container_type='FEU', max_proximity=10)
if placements:
    coords = yard.get_container_coordinates_from_placement(placements[0], 'FEU')
    yard.add_container(container, coords)
```

Technical Advantages

1. **Vectorization**: Numpy boolean operations on entire yard regions
2. **Memory locality**: Contiguous arrays for cache efficiency
3. **Bit manipulation**: Binary operations for container placement validation
4. **Minimal string operations**: Coordinate-based instead of string-based lookups
5. **Batch processing**: Process multiple containers simultaneously
6. **Linear scaling**: Performance grows linearly with container count

This implementation achieves **order-of-magnitude improvements** over traditional object-oriented approaches while maintaining full compatibility with existing container terminal simulation frameworks.

Exemplary Configuration and Mask Output Examples

```
from simulation.terminal_components.BooleanStorage import BooleanStorageYard

new_yard = BooleanStorageYard(
    n_rows=5,
    n_bays=15,
    n_tiers=3,
    # coordinates are in form (bay, row, type = r,dg,sb_t)
    coordinates=[
        # Reefers on both ends
        (1, 2, "r"), (1, 3, "r"), (1, 4, "r"), (1, 5, "r"),
        (15, 1, "r"), (15, 2, "r"), (15, 3, "r"), (15, 4, "r"), (15, 5, "r"),

        # Row nearest to trucks is for swap bodies and trailers
        (1, 1, "sb_t"), (2, 1, "sb_t"), (3, 1, "sb_t"), (4, 1, "sb_t"), (5, 1,
"sb_t"),
        (6, 1, "sb_t"), (7, 1, "sb_t"), (8, 1, "sb_t"), (9, 1, "sb_t"), (10, 1,
"sb_t"),
        (11, 1, "sb_t"), (12, 1, "sb_t"), (13, 1, "sb_t"), (14, 1, "sb_t"), (15,
1, "sb_t"),

        # Pit in the middle for dangerous goods
        (7, 3, "dg"), (8, 3, "dg"), (9, 3, "dg"),
        (7, 4, "dg"), (8, 4, "dg"), (9, 4, "dg"),
        (7, 5, "dg"), (8, 5, "dg"), (9, 5, "dg"),
    ],
    split_factor=4,
    validate=True
)
```

Reefer Mask

```
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
[■ ■ ■ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ■ ■ ■]
```

Interpretation of Mask

- **True (■)**: Position is occupied by a reefer container
- ■ ■ ■ □ □ □: translates to (Bay1|Split1 & Row1) & (Bay2|Split2 & Row 2) with stacking height of 3