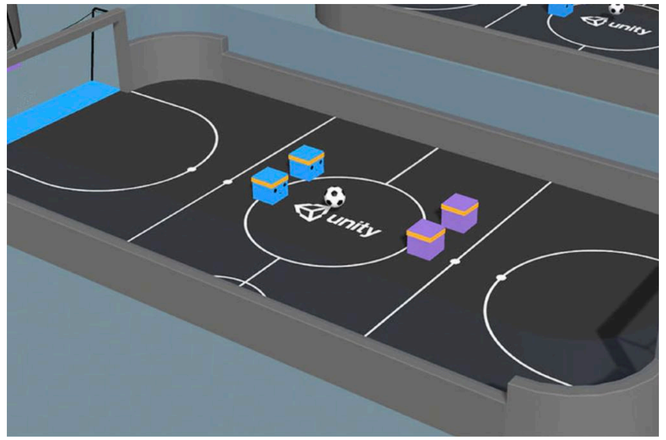
Lecture 11:

High-Throughput World Simulation for Agent Training

Visual Computing Systems
Stanford CS348K, Spring 2025

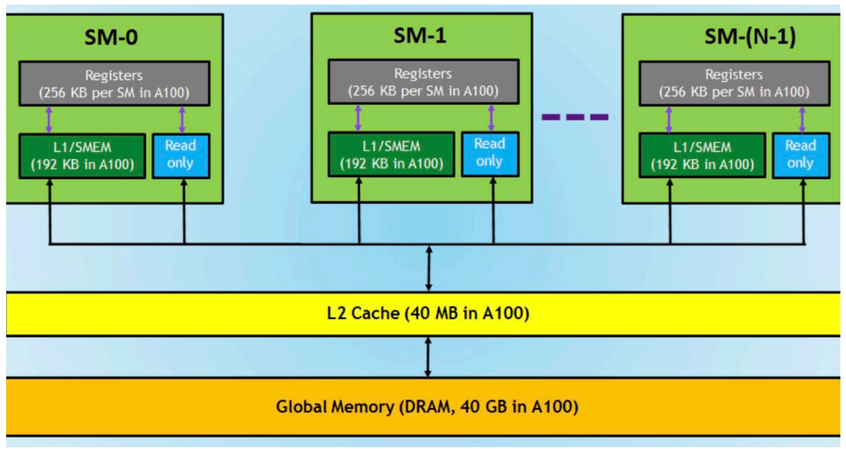
Tons of engineering knowledge and effort needed to build new GPU-accelerated batch simulator from scratch!

Task Knowledge



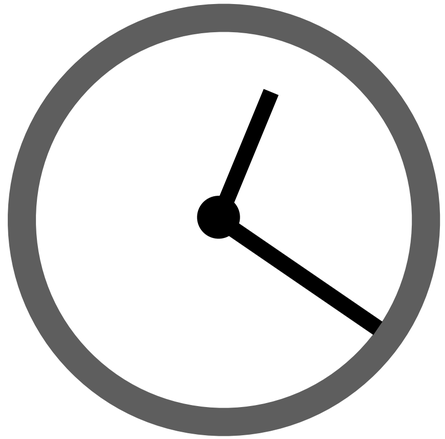
+

GPU Programming Skill

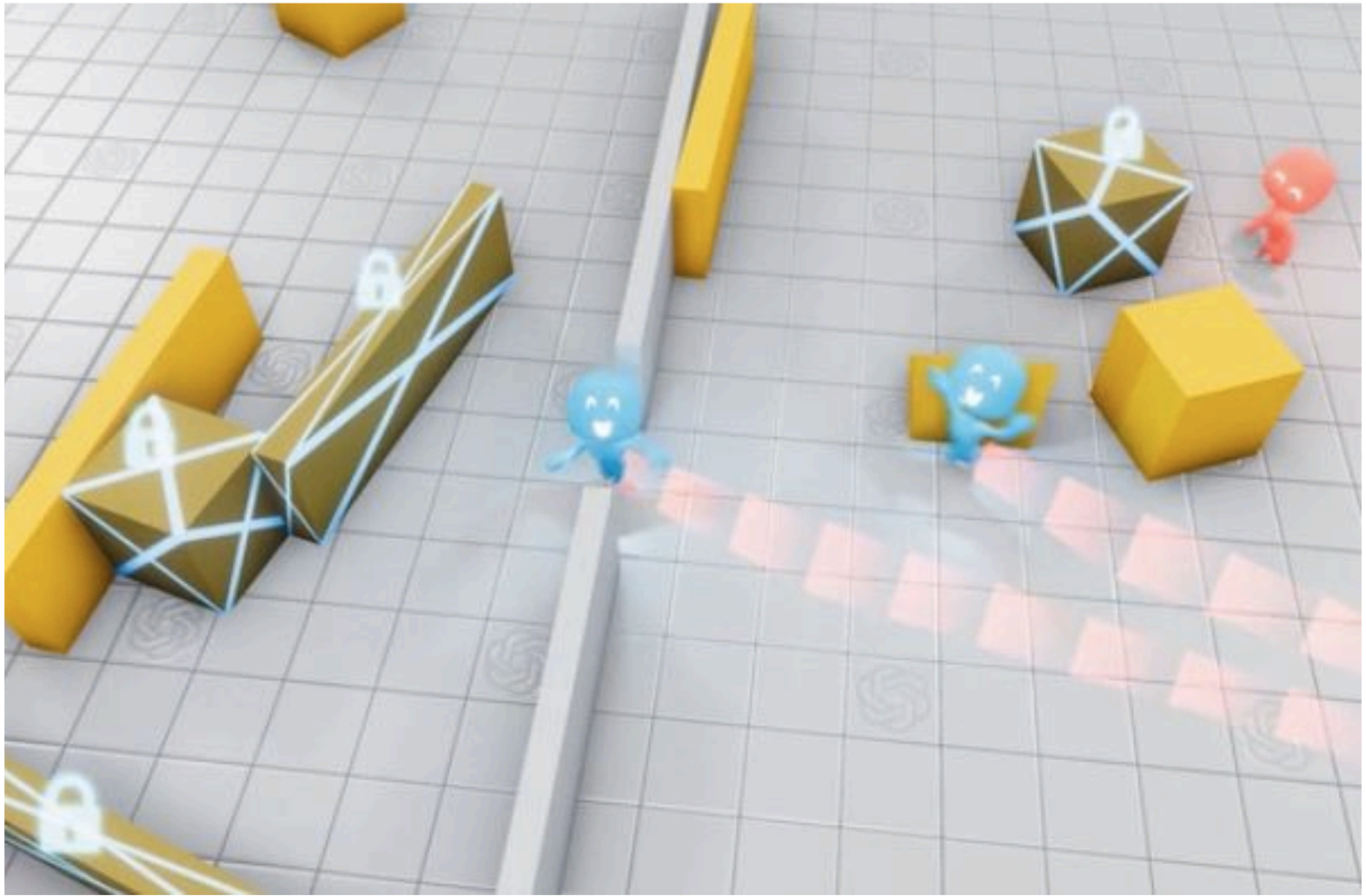


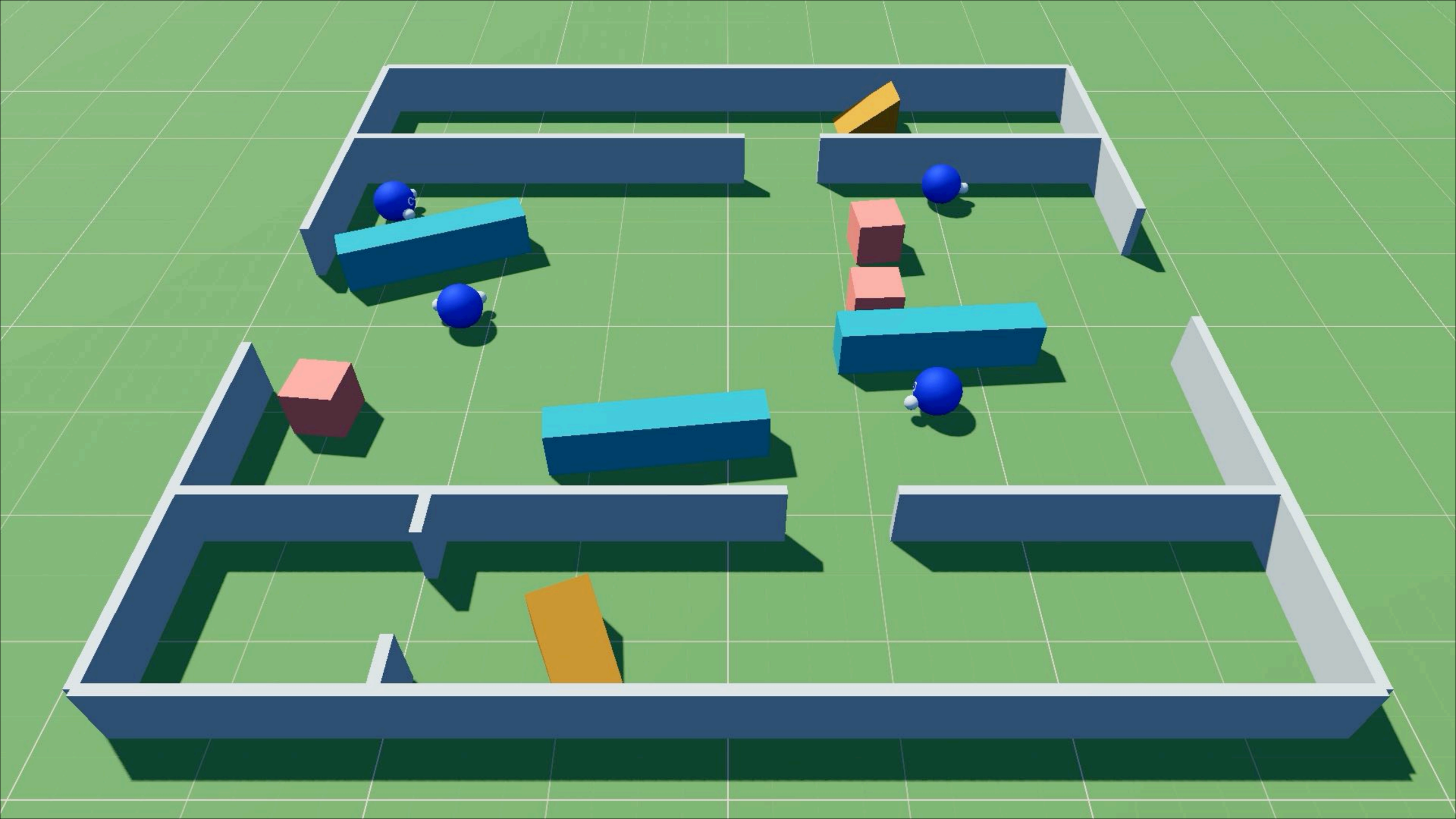
+

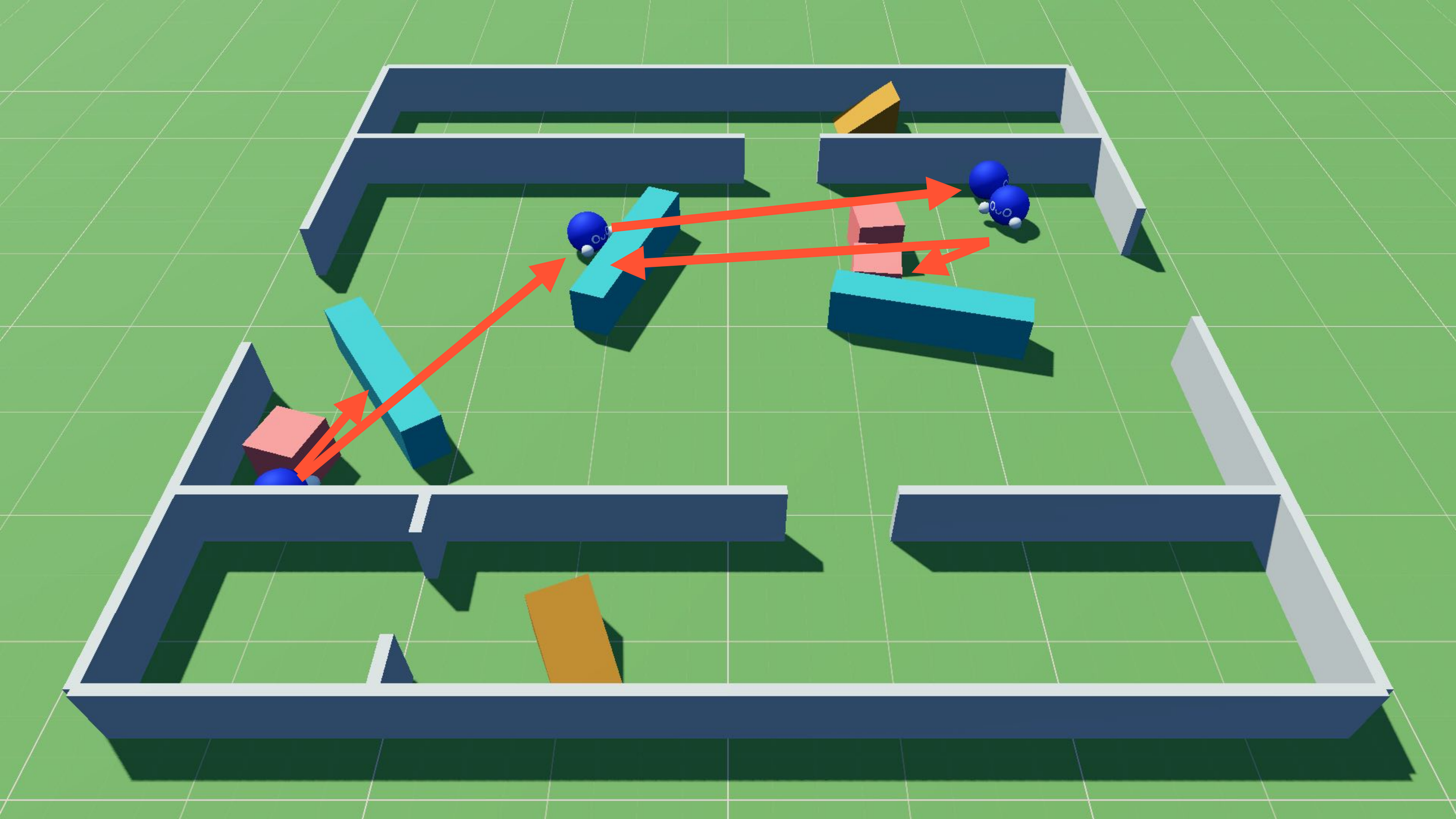
Engineering Time

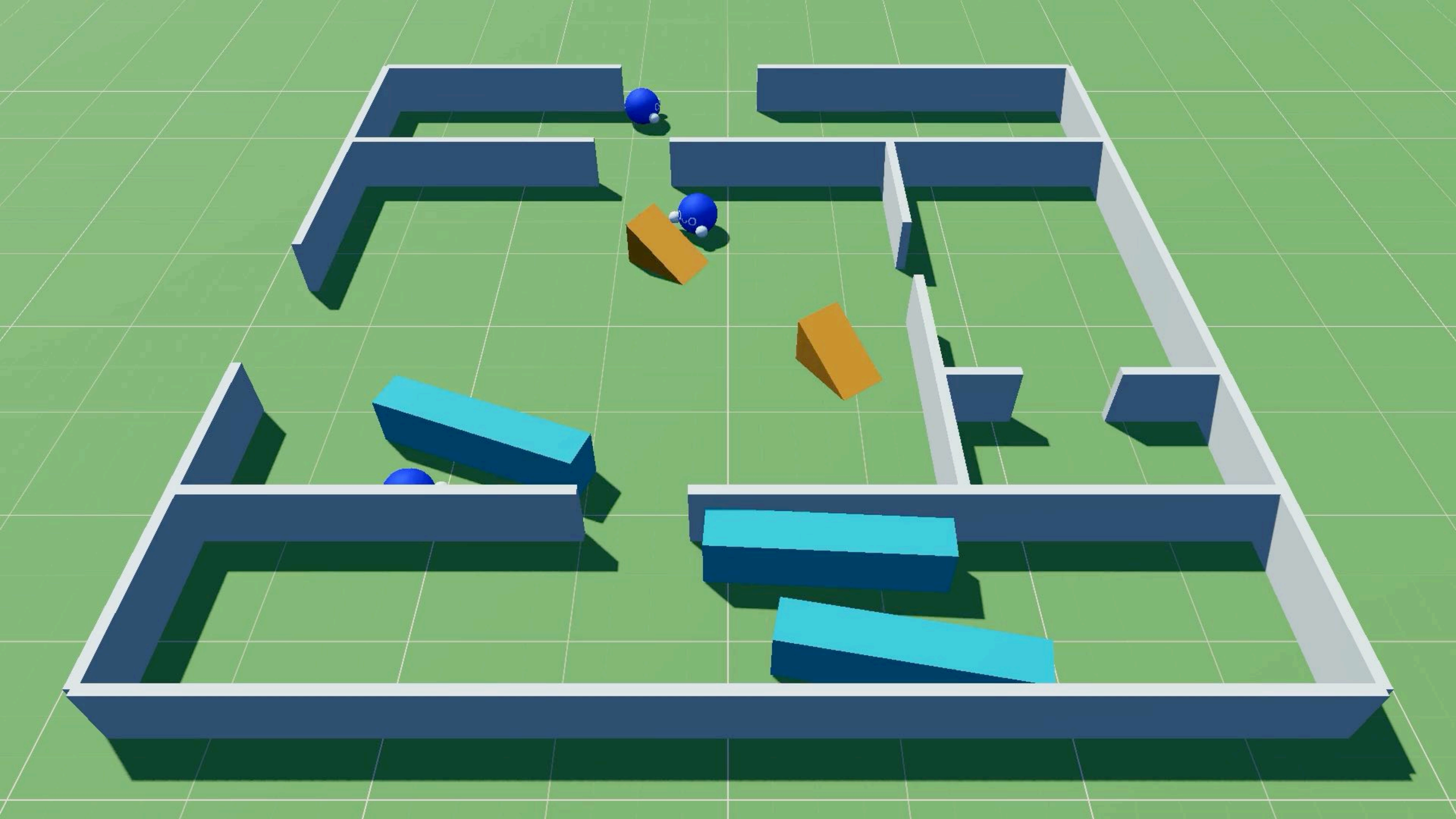


Running example: Simulating OpenAI's 3D "Hide and Seek" learning environment



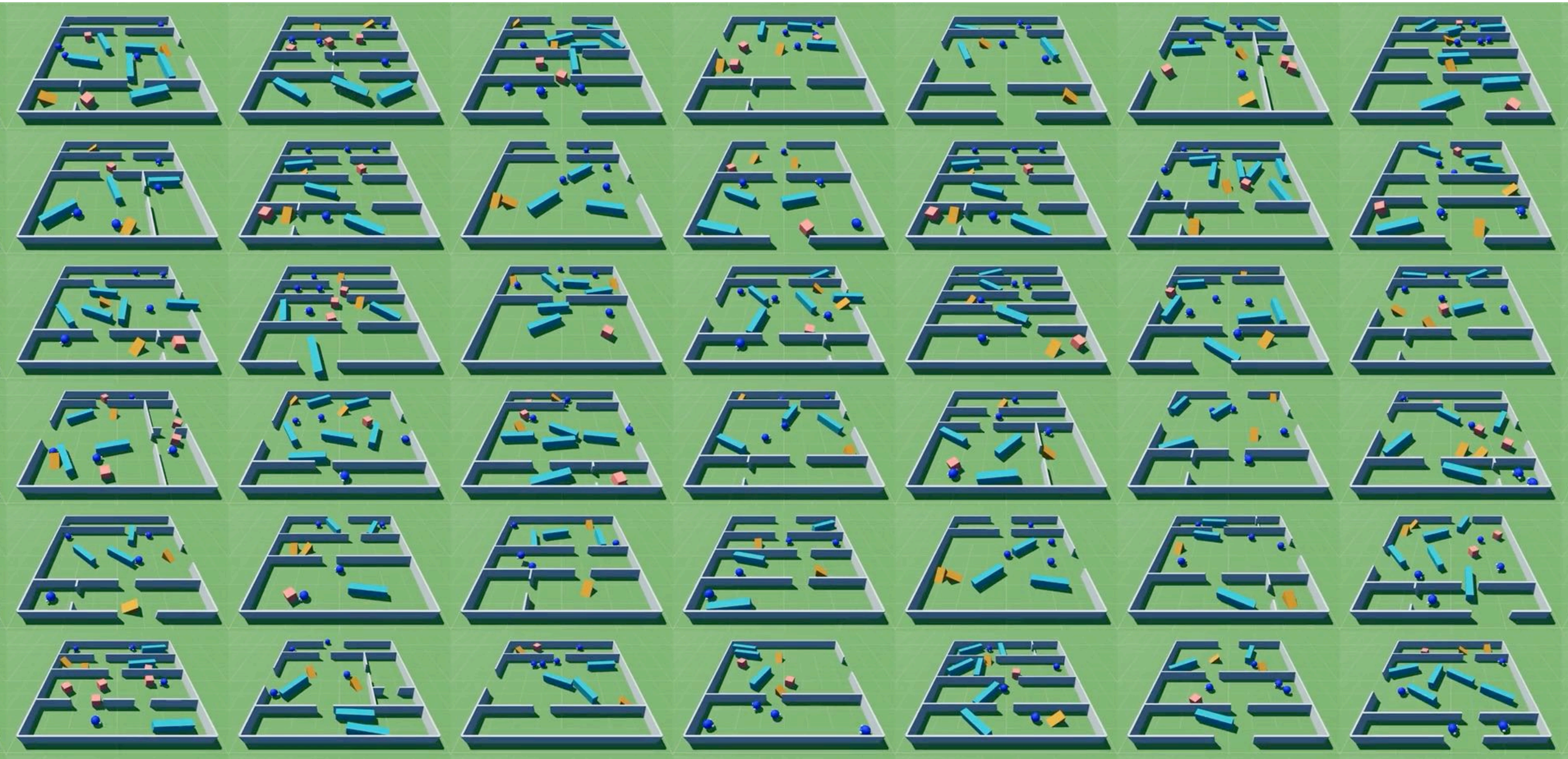




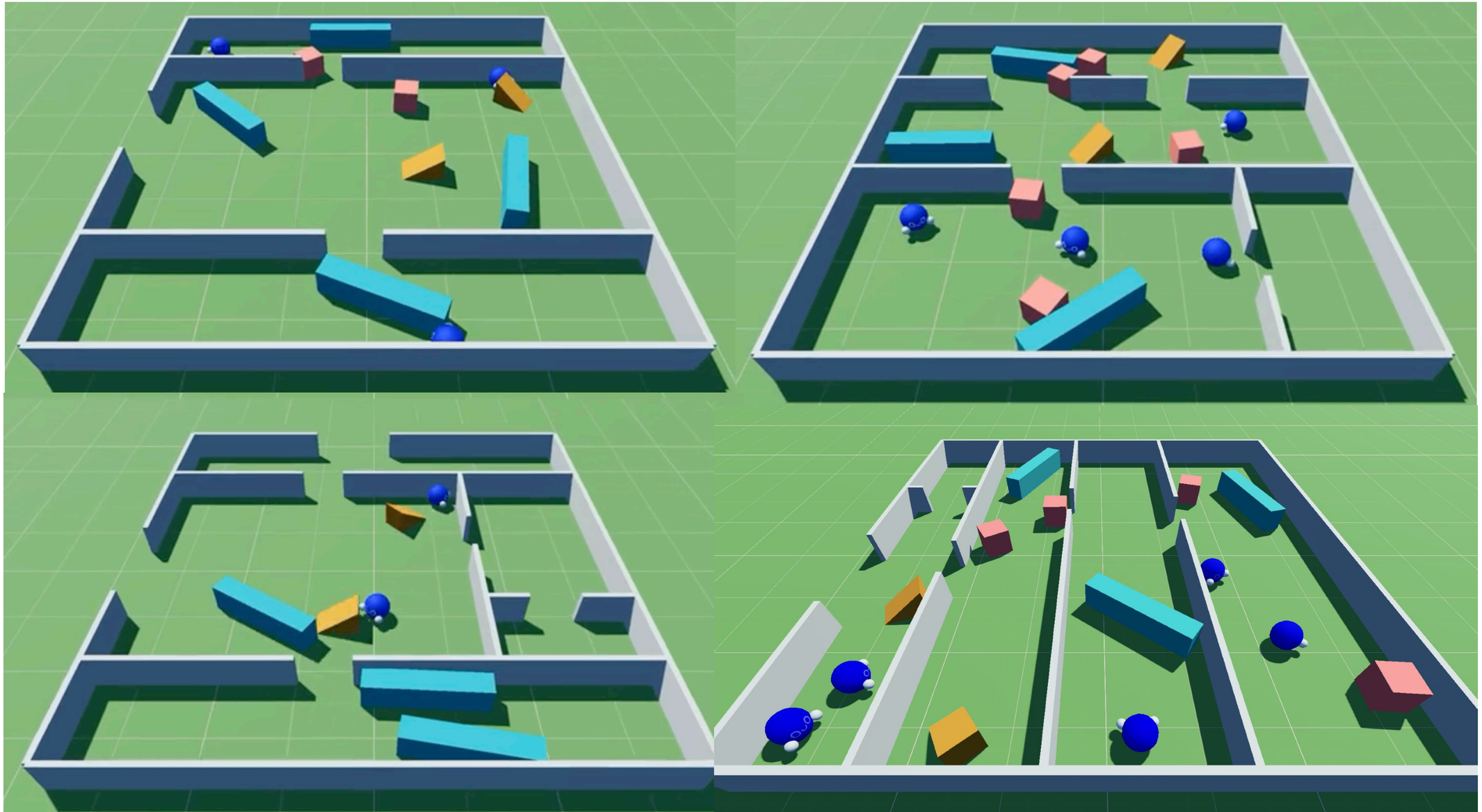


Parallel Systems Programming Requirements

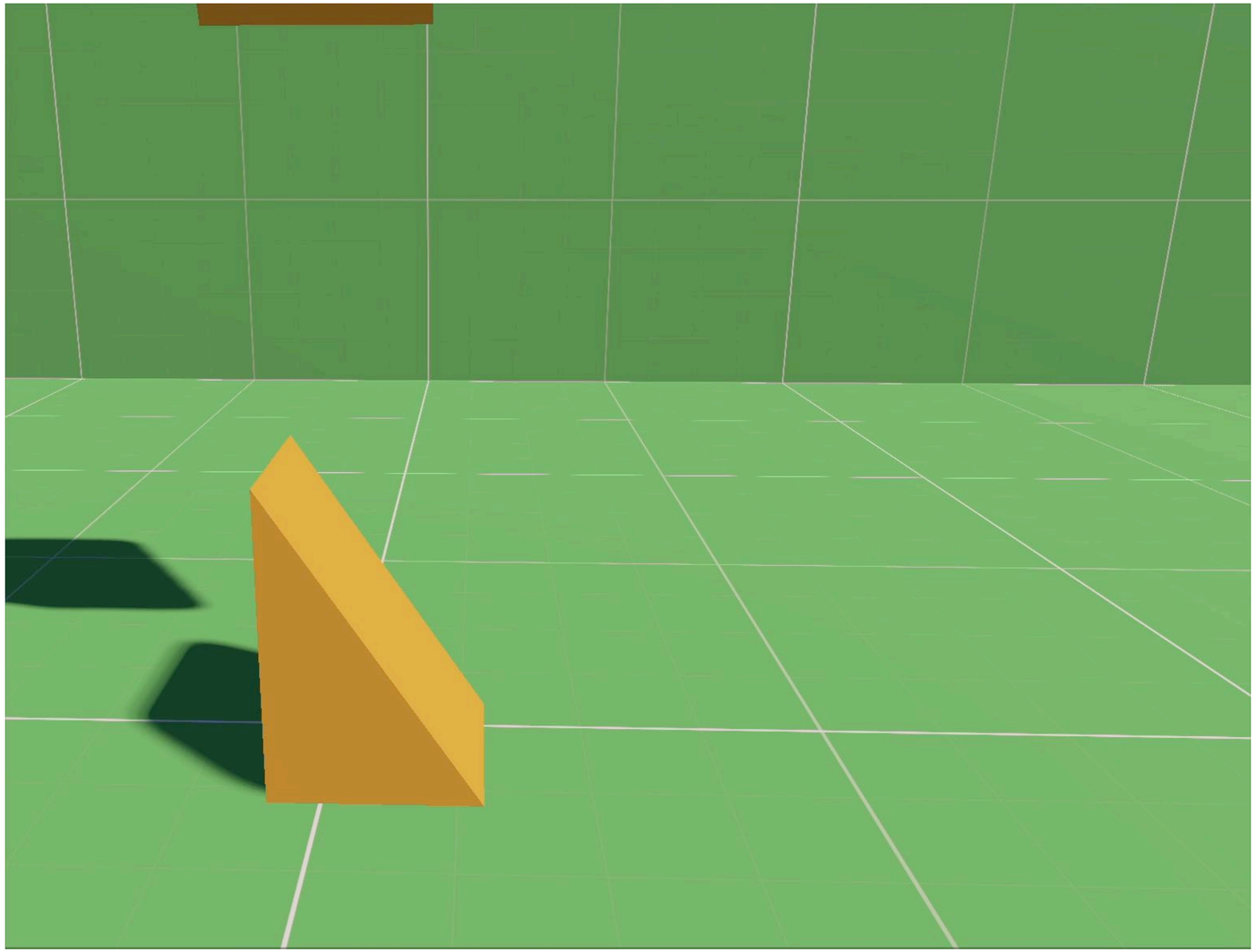
1. Nested Parallelism: Task logic for each object in each world



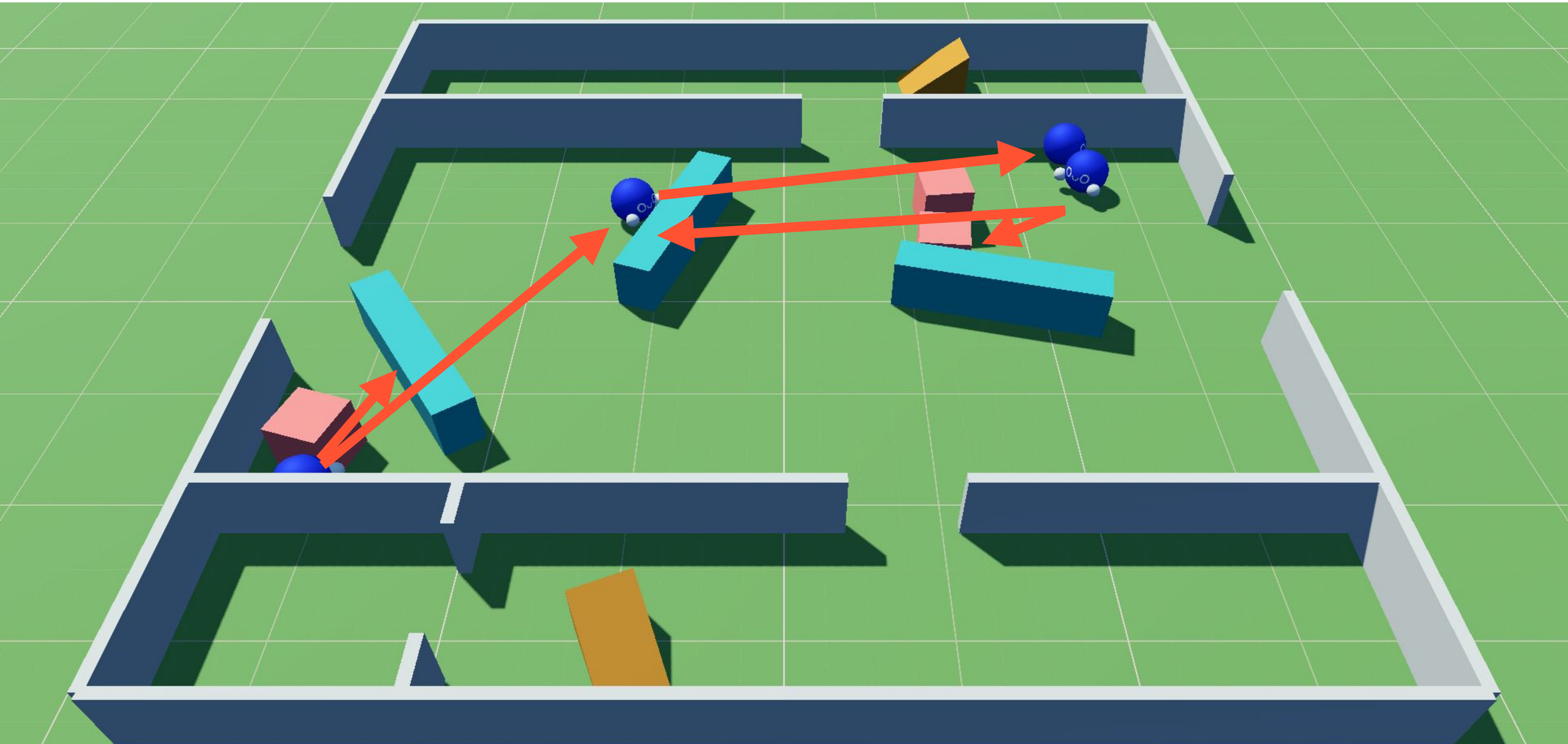
2. Irregular Nested Parallelism & Irregular Collection Sizes: Worlds with varying numbers of objects



3. Dynamic GPU-controlled memory allocation & parallelism: Physics contacts, sparse events



4. Complex Spatial Joins: Ray casting, 3D collisions



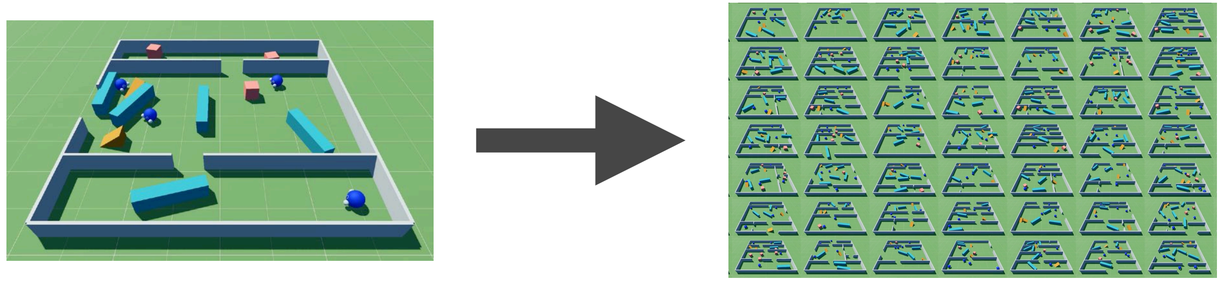
Usability: Easy scripting of task-specific logic

```
def process_action(agent_position, action):
  if action.type == MOVE_CHARACTER:
      force = computeMovementForce(action.dir)
   if action.type == LOCK_OBJECT_IN_PLACE:
      hit_obj = raycastForward(agent_position)
      if hit_obj:
        lockObject(hit_obj)
 ...
```

Usability: Easy scripting of task-specific logic

```
def process_action(agent_position, action):
  if action.type == MOVE_CHARACTER:
      force = computeMovementForce(action.dir)
   if action.type == LOCK_OBJECT_IN_PLACE:
      hit_obj = raycastForward(agent_position)
      if hit_obj:
        lockObject(hit_obj)
...
```

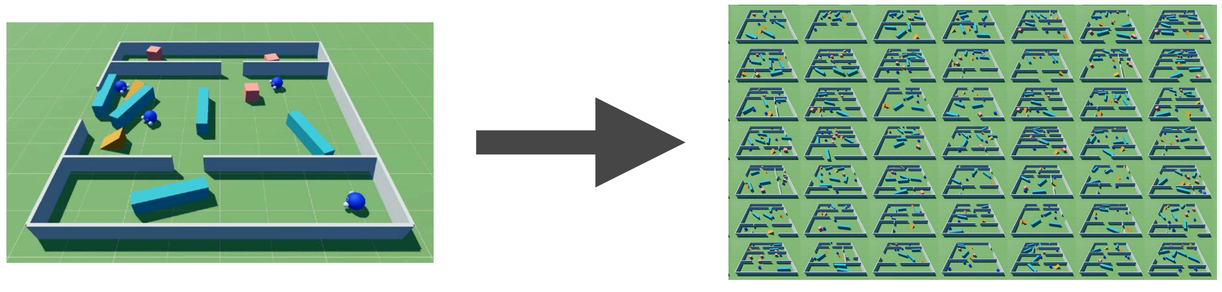
5. Implicit Parallelism: Logic written in terms of 1 environment, automatically batched



Usability: Easy scripting of task-specific logic

```
def process_action(agent_position, action):
  if action.type == MOVE_CHARACTER:
      force = computeMovementForce(action.dir)
   if action.type == LOCK_OBJECT_IN_PLACE:
      hit_obj = raycastForward(agent_position)
      if hit_obj:
        lockObject(hit_obj)
...
```

6. Convenience of standard SPMD control flow



Madrona research project claim:

We need to create a "game engine" for building batch simulators that meets these requirements!

1. Nested Parallelism
2. Irregular Parallelism & Collection Sizes
3. Dynamic GPU-Controlled Allocation
4. Complex Spatial Joins
5. Implicit Parallelism
6. SPMD-Style Control Flow

What About Using Existing Systems / Frameworks to Help to Build Complex Batch Simulators?

Lowest-level option: Where does CUDA C++ fall short for our needs?

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Writing a batch simulator in raw CUDA requires custom parallel memory management & scheduling

	CUDA C++
Nested Parallelism	✓
Irregular Parallelism & Collection Sizes	X
Dynamic GPU-Controlled Allocation	X
Complex Spatial Joins	X
Implicit Parallelism	X
SPMD-style control flow	✓

Higher-level GPU kernel languages: friendlier syntax, but same abstraction as CUDA C++

NVIDIA Warp


```python
import warp as wp

@wp.kernel
def integrate(p: wp.array(dtype=wp.vec3),
              v: wp.array(dtype=wp.vec3),
              f: wp.array(dtype=wp.vec3),
              m: wp.array(dtype=float)):

    # thread id
    tid = wp.tid()

    # Semi-implicit Euler step
    v[tid] = v[tid] + (f[tid] * m[tid] + wp.vec3(0.0, -9.8, 0.0)) * dt
    x[tid] = x[tid] + v[tid] * dt

# kernel launch
wp.launch(integrate, dim=1024, inputs=[x, v, f, …], device="cuda:0")
```

Array-based programming: Describe simulation in terms of bulk operations on fixed-size tensors

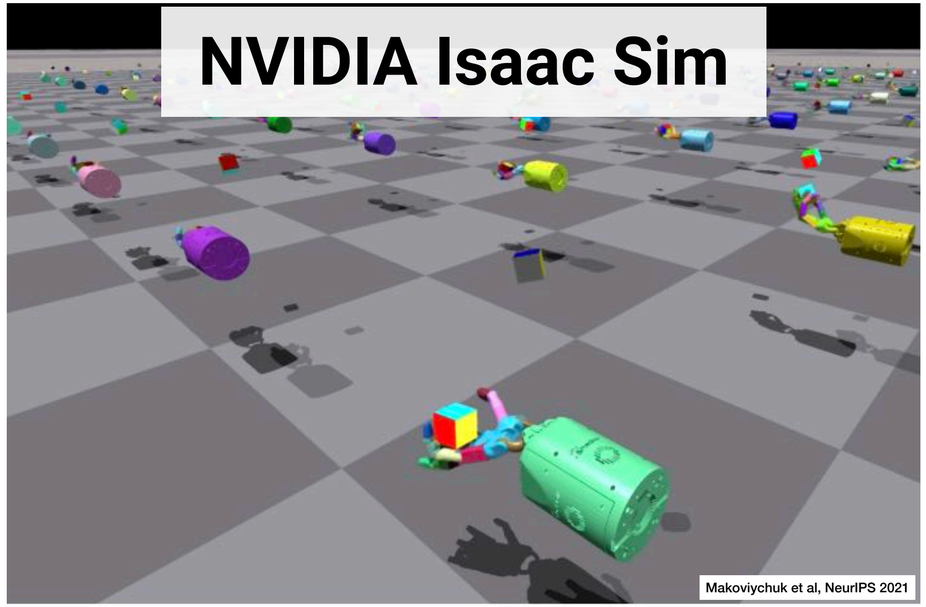
PyTorch

Array-based programming: Variable environment structures, procedural generation are challenging

	PyTorch	
Nested Parallelism	✓	✓
Irregular Parallelism & Collection Sizes	~	X
Dynamic GPU-Controlled Allocation	X	X
Complex Spatial Joins	X	X
Implicit Parallelism	~	✓
SPMD-style control flow	X	X

What about building batch simulators by reusing existing GPU simulation libraries?

NVIDIA PHYSX



Existing GPU simulation libraries (PhysX) are designed to accelerate a CPU-controlled simulation

	GPU PhysX
Nested Parallelism	✓
Irregular Parallelism & Collection Sizes	~
Dynamic GPU-Controlled Allocation	X
Complex Spatial Joins	✓
Implicit Parallelism	X
SPMD-style control flow	X

Existing Systems Do Not Meet the Requirements!

	PyTorch	JAX	CUDA C++	Warp / NUMBA	GPU PhysX
Nested Parallelism	✓	✓	✓	✓	✓
Irregular Parallelism & Collection Sizes	~	X	X	X	~
Dynamic GPU-Controlled Allocation	X	X	X	X	X
Complex Spatial Joins	X	X	X	X	✓
Implicit Parallelism	~	✓	X	X	X
SPMD-style control flow	X	X	✓	✓	X

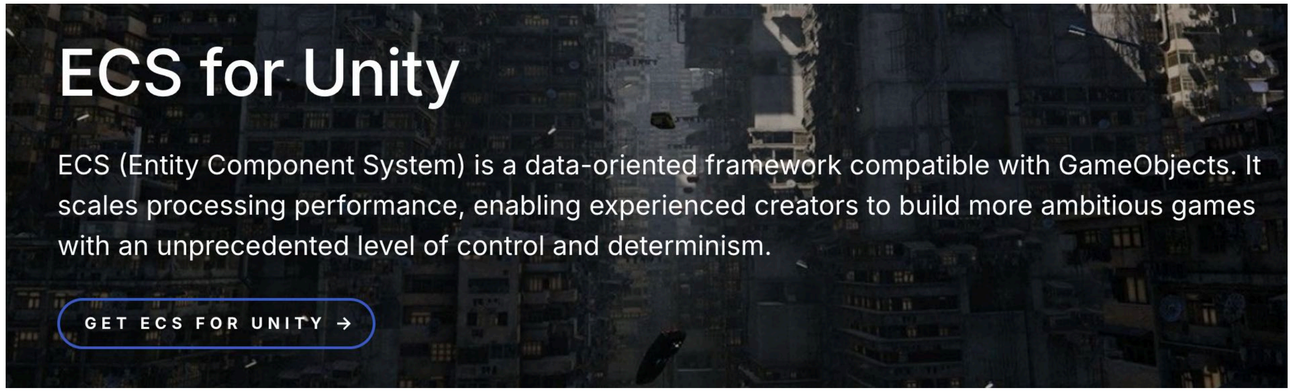
Madrona simulation engine project

- Madrona engine project (Stanford project)

- A framework for building GPU batch simulators using entity component system (ECS) design patterns

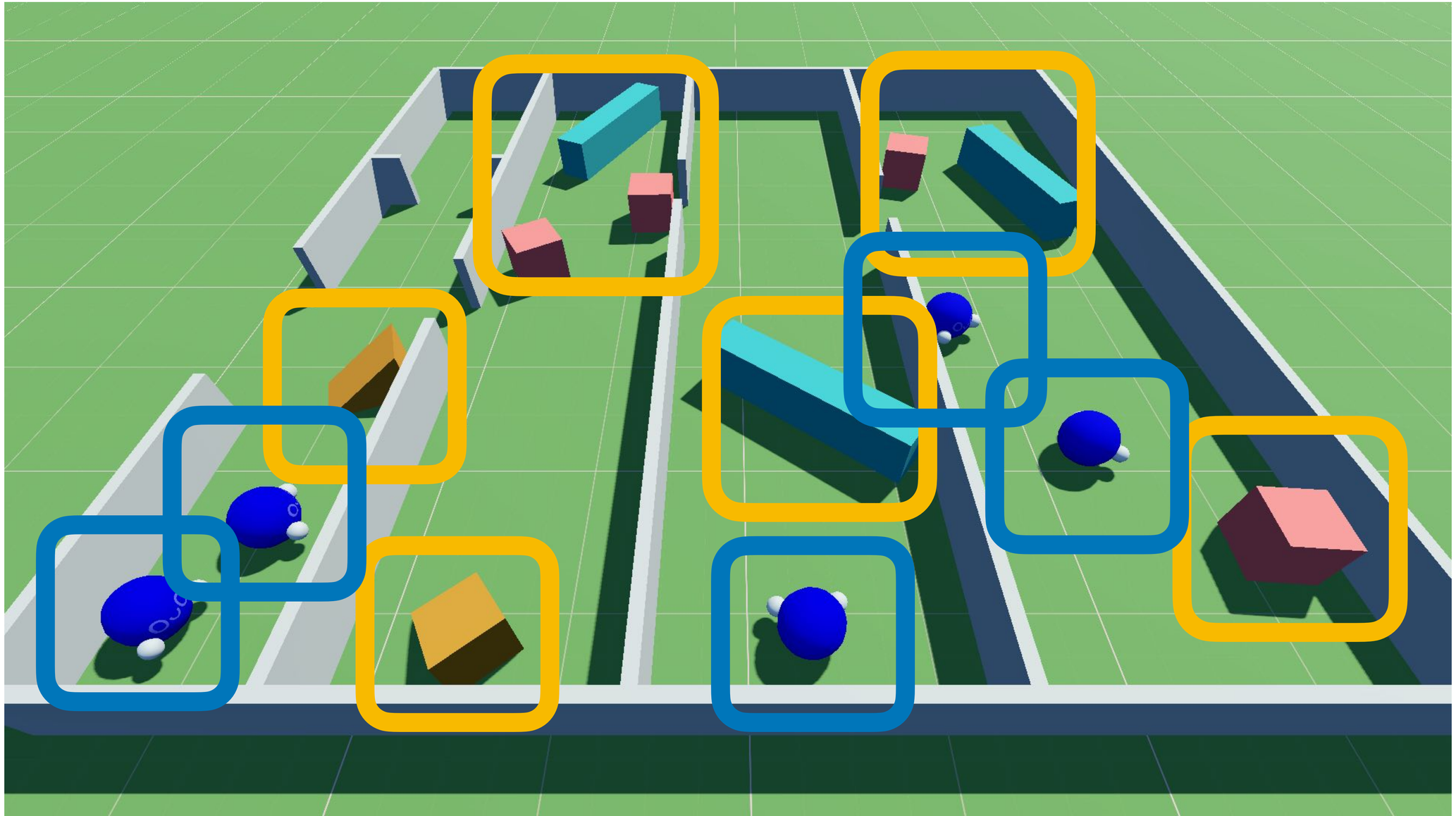
Idea #1

- **The well-known Entity Components System* design pattern is a good solution for structuring data and communication in a batch simulator**

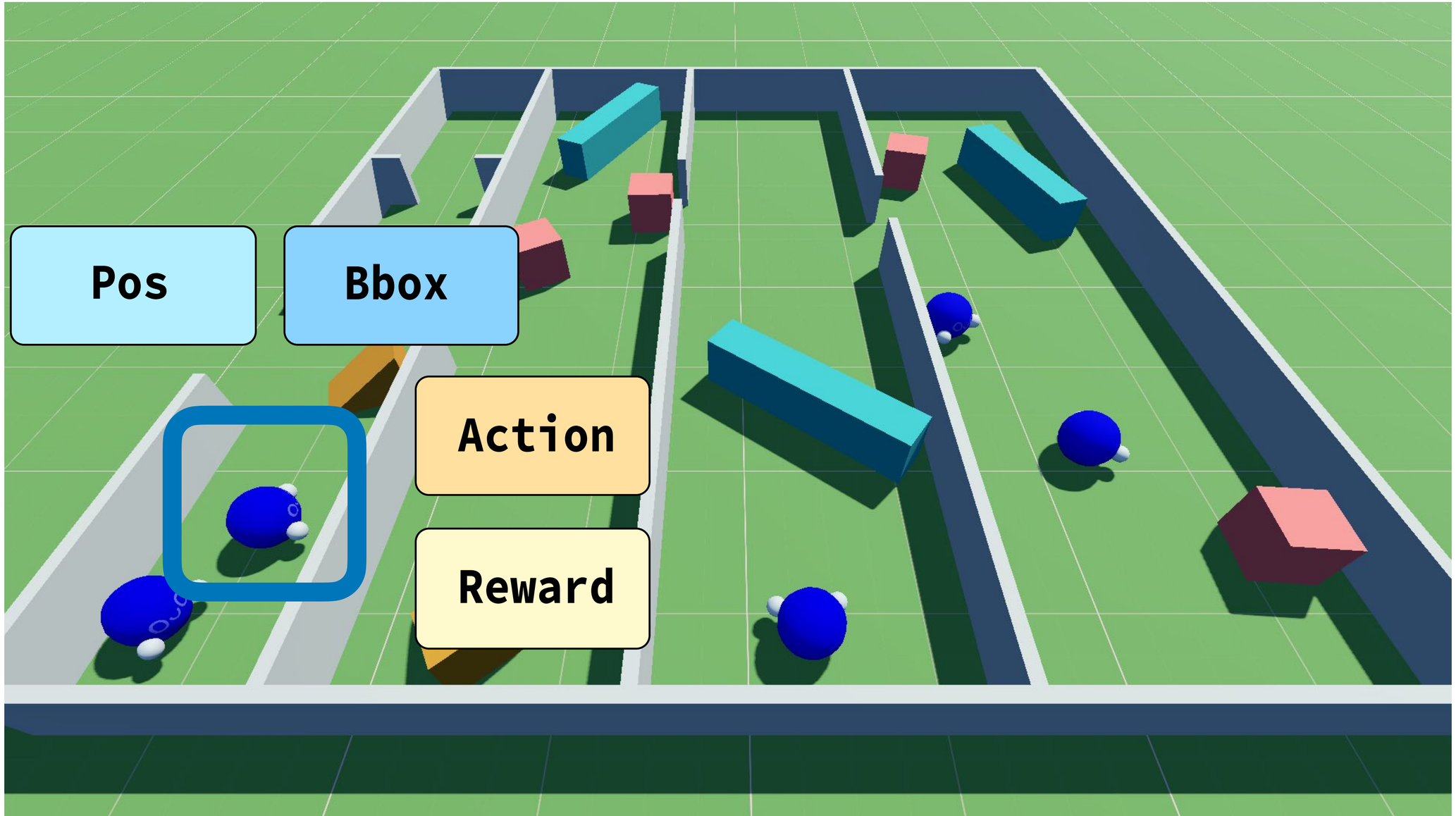


Tutorial:
Entity Component System (ECS) Design Patterns on the GPU

Concept 1: Entities (ECS)



Concept 2: Components (ECS)



Concept 2: Components (ECS)

Agents

Id	EnvID	Pos	Bbox	Action	Reward	
12	0	[0,0,.5]	{min...	LEFT	0.1	...
32	0	[2,1,0]	{min...	FWD	-0.1	...
51	0	[1.5,0,1]	{min...	FWD	-2.5	...

Obstacles

Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	0	[0,1,3]	{min...	...
61	0	[1,1,2]	{min...	...

Batch simulation ECS: Store data across all environments in unified tables in GPU memory

Agents

Id	EnvID	Pos	Bbox	Action	Reward	
12	0	[0,0,.5]	{min...	LEFT	0.1	...
32	0	[2,1,0]	{min...	FWD	-0.1	...
51	0	[1.5,0,1]	{min...	FWD	-2.5	...
62	1	[1.5,0.5,1]	{min...	RIGHT	0.3	...
65	1	[-0.5,0,0]	{min...	RIGHT	0.5	...
20	2	[0.5,1,1]	{min...	LEFT	1.5	...
23	2	[-1.5,0,0]	{min...	LEFT	10.5	...
41	2	[0.5,1,0]	{min...	BACK	-10	...

Obstacles

Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	0	[0,1,3]	{min...	...
61	0	[1,1,2]	{min...	...
49	1	[0.5,1,2]	{min...	...
70	1	[-0.5,1,0]	{min...	...
33	2	[1.5,0,2.5]	{min...	...
81	3	[2.5,0.5,2]	{min...	...
11	3	[1.5,0.5,2]	{min...	...

Unified table storage also enables throughput-oriented dynamic memory allocation

Agents

Id	EnvID	Pos	Bbox	Action	Reward	
12	0	[0,0,.5]	{min...	LEFT	0.1	...
32	0	[2,1,0]	{min...	FWD	-0.1	...
~~51~~	~~X~~	~~[1.5,0,1]~~	~~{min...~~	~~FWD~~	~~-2.5~~	~~...~~
62	1	[1.5,0.5,1]	{min...	RIGHT	0.3	...
65	1	[-0.5,0,0]	{min...	RIGHT	0.5	...
20	2	[0.5,1,1]	{min...	LEFT	1.5	...
23	2	[-1.5,0,0]	{min...	LEFT	10.5	...
41	2	[0.5,1,0]	{min...	BACK	-10	...
51	1	[0.5,1,1]	{min...	BACK	-10	...

Obstacles

Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	0	[0,1,3]	{min...	...
61	0	[1,1,2]	{min...	...
~~49~~	~~X~~	~~[0.5,1,2]~~	~~{min...~~	~~...~~
70	1	[-0.5,1,0]	{min...	...
33	2	[1.5,0,2.5]	{min...	...
81	3	[2.5,0.5,2]	{min...	...
~~11~~	~~X~~	~~[1.5,0.5,2]~~	~~{min...~~	~~...~~
15	2	[1.5,1.5,2]	{min...	...
12	0	[2.5,0.5,3]	{min...	...

Concept 3: Systems (ECS)

Agents

Id	EnvID	Pos	Bbox	Action	Reward	
12	0	[0,0,.5]	{min...	LEFT	0.1	...
32	1	[2,1,0]	{min...	FWD	-0.1	...
51	2	[1.5,0,1]	{min...	FWD	2.5	...
22	2	[2.5,0,1.5]	{min...	RIGHT	1.1	...

Obstacles

Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	1	[0,1,3]	{min...	...
61	1	[1,1,2]	{min...	...
25	2	[1.5,1,2.5]	{min...	...

ProcessActions

Pos, Action

Collisions

Id, Pos, Bbox

ComputeRewards

Pos, Reward

Concept 3: Systems (ECS)

Agents

Id	EnvID	Pos	Bbox	Action	Reward	
12	0	[0,0,.5]	{min...	LEFT	0.1	...
32	1	[2,1,0]	{min...	FWD	-0.1	...
51	2	[1.5,0,1]	{min...	FWD	2.5	...
22	2	[2.5,0,1.5]	{min...	RIGHT	1.1	...

Obstacles

Id	EnvID	Pos	Bbox	
13	0	[0.5,0,.5]	{min...	...
72	1	[0,1,3]	{min...	...
61	1	[1,1,2]	{min...	...
25	2	[1.5,1,2.5]	{min...	...

ProcessActions

Pos, Action

Collisions

Id, Pos, Bbox

ComputeRewards

Pos, Reward

Systems written as straight-line, per-entity logic

Agents

Id	EnvID	Pos	Bbox	Action	Reward
12	0	[0,0,.5]	{min...	LEFT	0.1
32	1	[2,1,0]	{min...	FWD	-0.1
51	2	[1.5,0,1]	{min...	FWD	2.5
22	2	[2.5,0,1.5]	{min...	RIGHT	1.1

```
def process_action(agent_position, action):
  if action.type == MOVE:
      force = computeMovementForce(action.dir)
   if action.type == LOCK:
      hit_obj = raycastForward(agent_position)
      if hit_obj:
        lockObject(hit_obj)

...
```


ProcessActions

Pos, Action

Collisions

Id, Pos, Bbox

ComputeRewards

Pos, Reward

Parallel GPU threads execute system logic over each table row

Agents

Id	EnvID	Pos	Bbox	Action	Reward
GPU Thread	0	[0,0,.5]	{min...	LEFT	0.1
GPU Thread	1	[2,1,0]	{min...	FWD	-0.1
GPU Thread	2	[1.5,0,1]	{min...	FWD	2.5
GPU Thread	2	[2.5,0,1.5]	{min...	RIGHT	1.1

```
def process_action(agent_position, action):
  if action.type == MOVE:
      force = computeMovementForce(action.dir)
   if action.type == LOCK:
      hit_obj = raycastForward(agent_position)
      if hit_obj:
        lockObject(hit_obj)

...
```


ProcessActions

Pos, Action

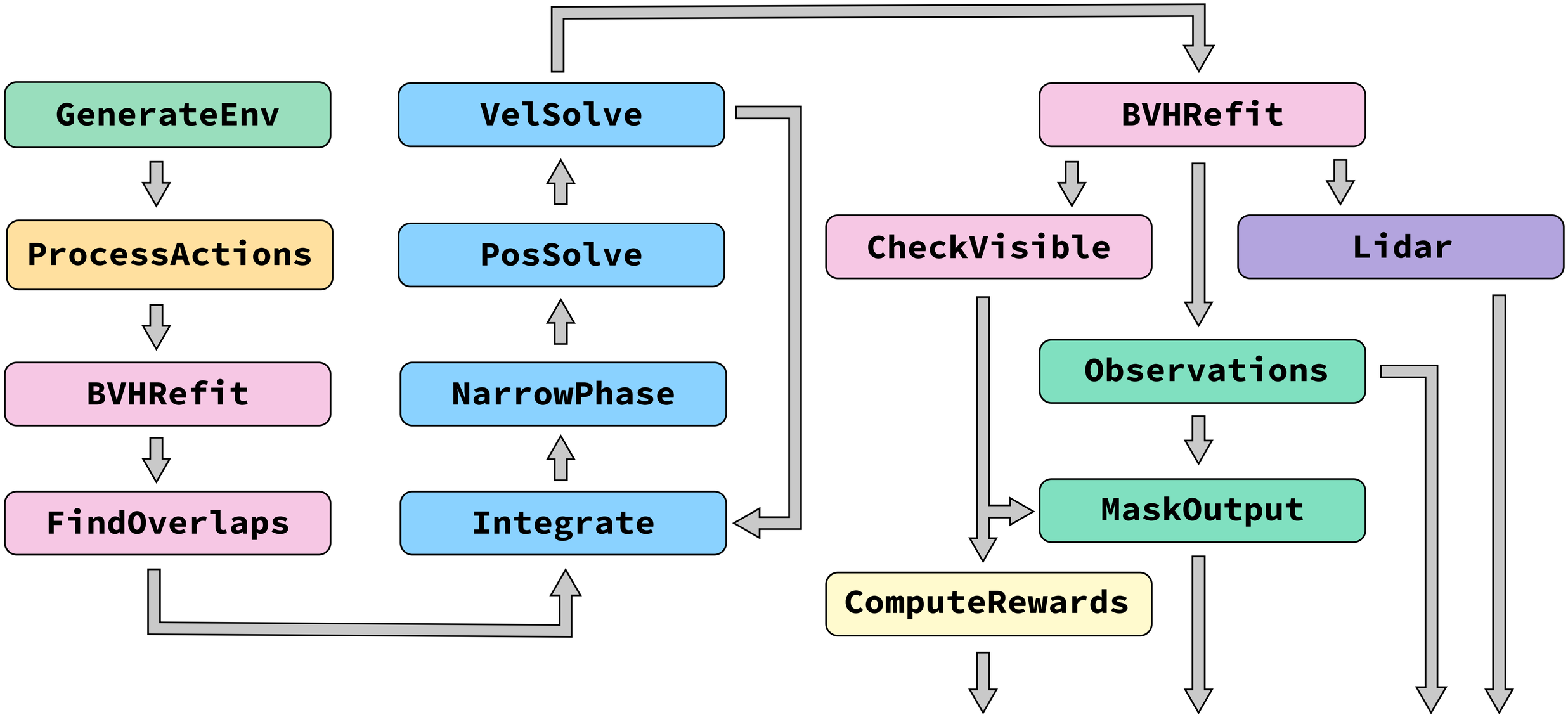
Collisions

Id, Pos, Bbox

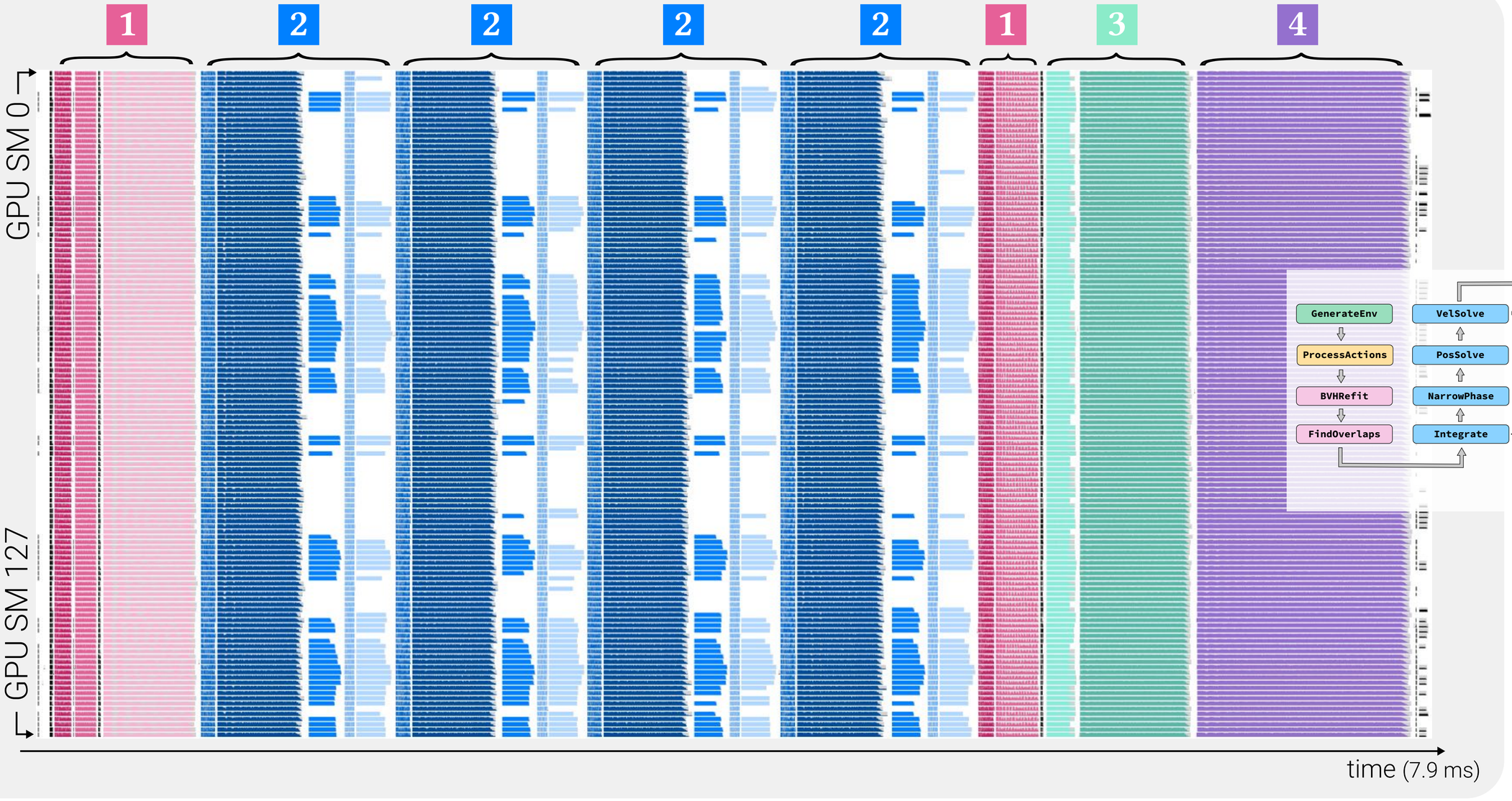
ComputeRewards

Pos, Reward

ECS systems combined into task graph and executed in parallel on the GPU



Scheduling batch of worlds onto GPU



1 BVH & Broad Phase 2 Physics Sub-Step 3 Agent Observations & Rewards 4 LIDAR

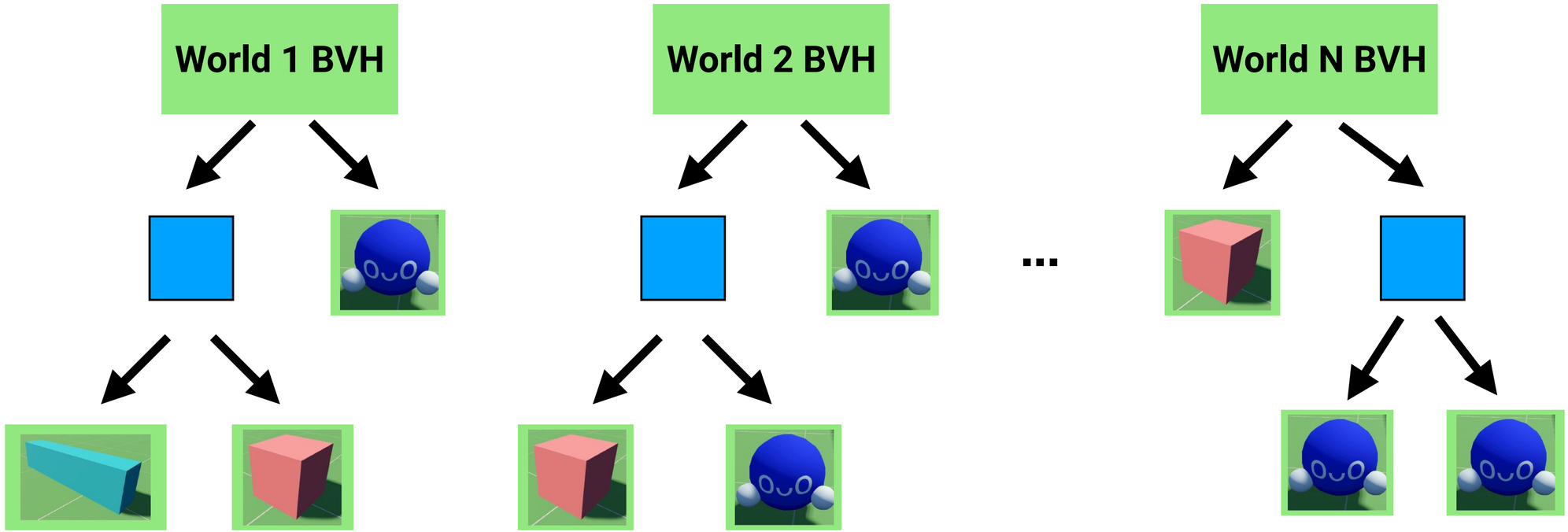
What about spatial queries?

```
def process_action(agent_position, action):
  if action.type == MOVE:
        force = computeMovementForce(action.dir)
    if action.type == LOCK:
        hit_obj = raycastForward(agent_position)
        if hit_obj:
          lockObject(hit_obj)

  ...
```

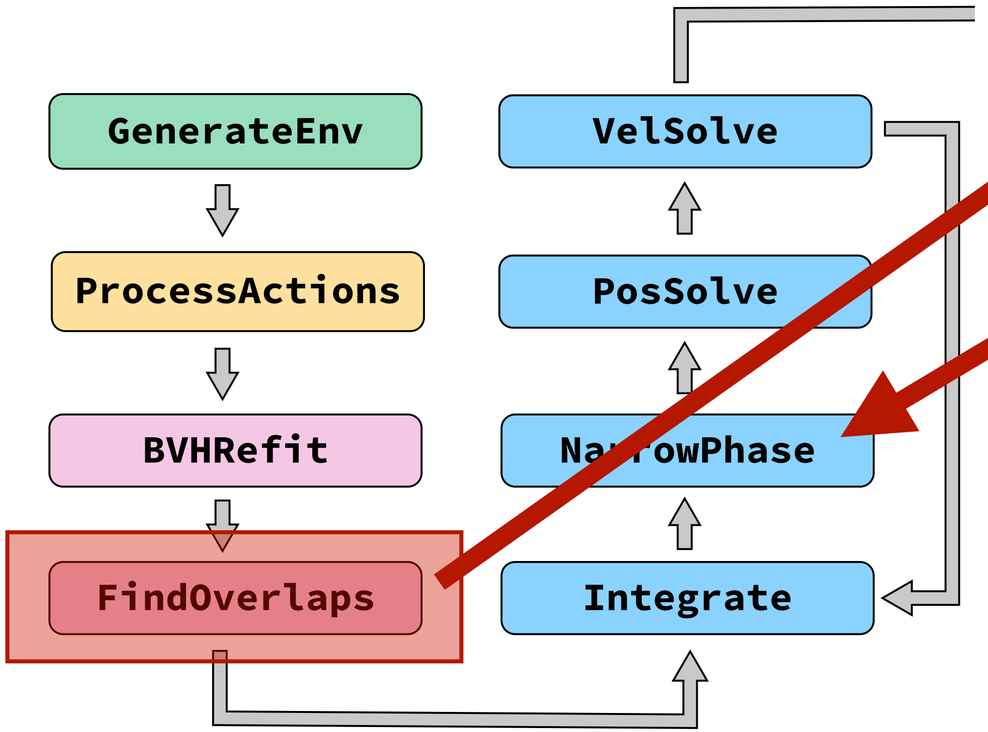
Agents

Id	EnvID	Pos	Bbox	Action	Reward	
12	0	[0,0,.5]	{min...	LEFT	0.1	...
32	1	[2,1,0]	{min...	FWD	-0.1	...
51	2	[1.5,0,1]	{min...	FWD	2.5	...
22	2	[2.5,0,1.5]	{min...	RIGHT	1.1	...

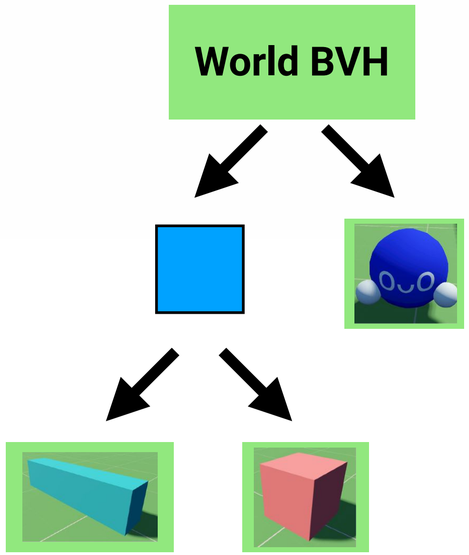
Madrona standard library provides high-performance per-world 3D acceleration structure (BVH) as an index to make queries fast



BVH Standard Library Calls Allow ECS Systems to Make Spatial Queries

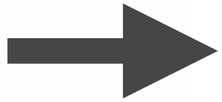



```
def find_overlaps(world, my_id, position, bbox):
  for bvh_node in world.bvh.find_overlapping(bbox)
    world.createEntity(CollisionPair(my_id, bvh_node.id))
```



Madrona needs a straightforward imperative language for authoring ECS systems

```
def process_action(world,
                   agent_position,
                   action):
 if action.type == MOVE:
       force = computeMovementForce(action.dir)
   if action.type == LOCK:
       hit_obj =
         raycastForward(world, agent_position)
       if hit_obj:
         lockObject(hit_obj)

 ...
```



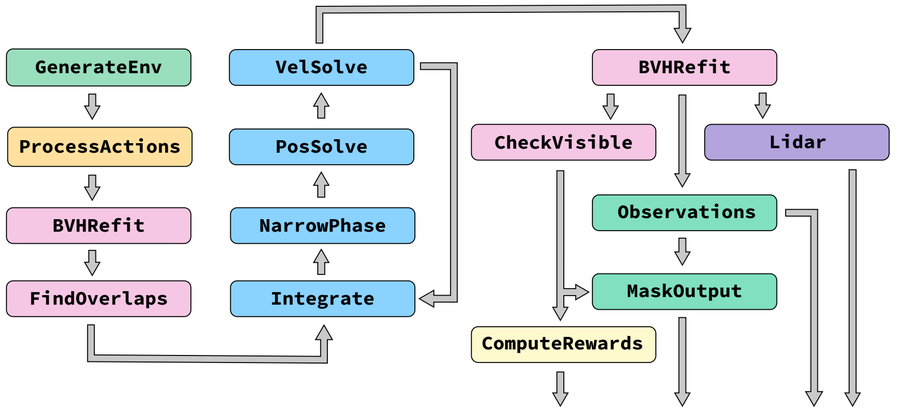
```
void process_action(World &world,
                    Position &position,
                    AgentAction &action) {
  if (action.type == MOVE) {
        Vector3 force =
            computeMovementForce(action.dir);
    }
    if (action.type == LOCK) {
        Entity hit_obj =
          raycastForward(world, agent_position);
        if (hit_obj) {
          lockObject(hit_obj);
        }
    }

  ...
}
```

Madrona Framework Summary

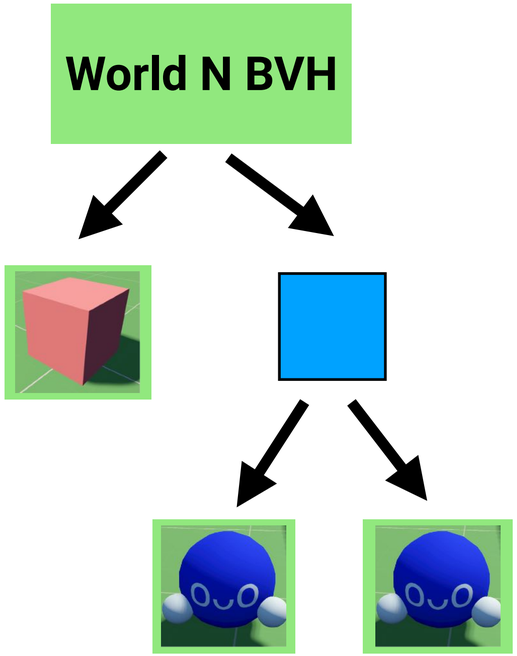
ECS Storage

Agents						
Id	EnvID	Pos	Bbox	Action	Reward	
12	0	[0,0,.5]	{min...	LEFT	0.1	...
32	1	[2,1,0]	{min...	FWD	-0.1	...
51	2	[1.5,0,1]	{min...	FWD	2.5	...
22	2	[2.5,0,1.5]	{min...	RIGHT	1.1	...

Parallel ECS Scheduler for GPU



ECS-Integrated Standard Library (BVH, Physics, etc)



Summary: Madrona meets core requirements for building wide range of batch simulators

		Madrona
Irregularly Sized Collections	Multi-World ECS Tables	✓
Dynamic GPU-Controlled Allocation		✓
Irregular Nested Parallelism	ECS Systems & Parallel GPU Task Graph Scheduling	✓
Dynamic GPU-Controlled Parallelism		✓
Implicit Parallelism		✓
Complex Spatial Joins	Built on General-Purpose Programming Language (CUDA C++)	✓
SPMD-Style Control Flow		✓

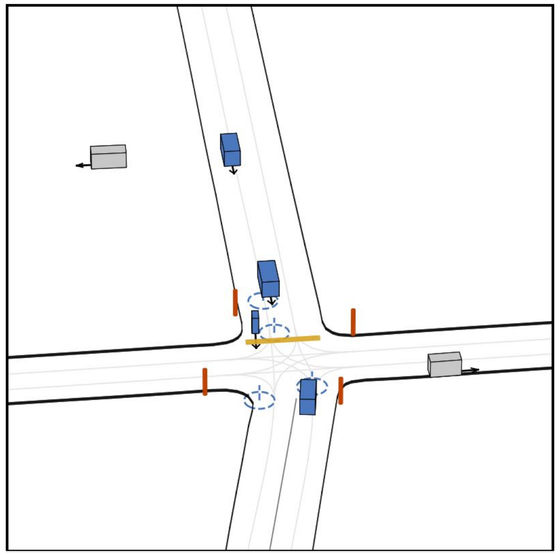
Case Studies:
(Not in the paper)

End-To-End Reinforcement Learning Experiments Using Madrona

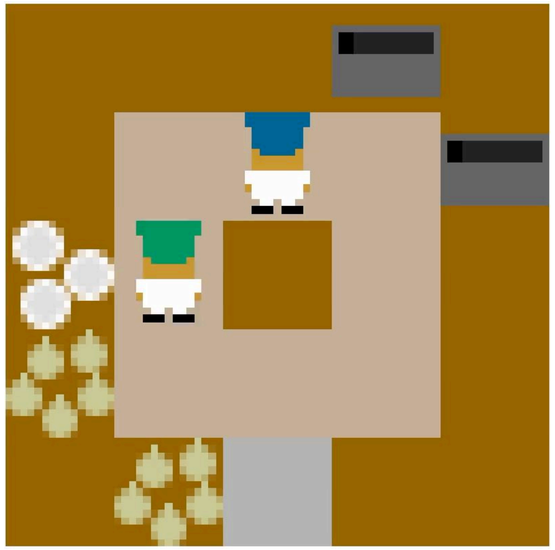
Case Study Questions:

1. Can ML researchers / engineers actually use the system?
2. Is Madrona helpful for speeding up end-to-end training loops, not just simulation speed?
3. What are the implications & use cases for very large simulation batches in end-to-end training?
4. Are agents trained in Madrona useful beyond Madrona (Sim-to-Sim, Sim-to-Real)?

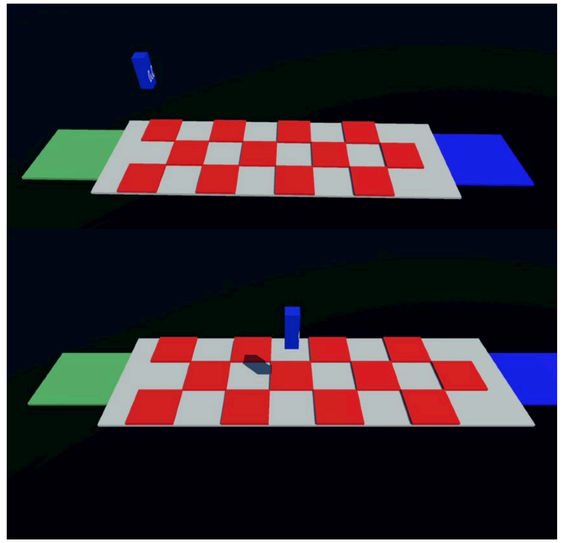
Case Studies Overview



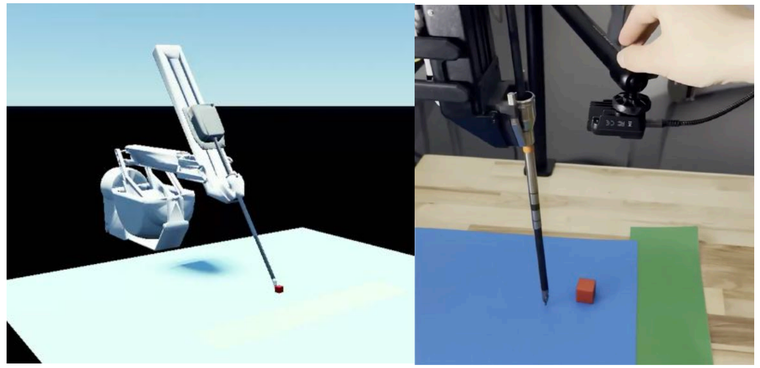
Autonomous vehicle planning & coordination



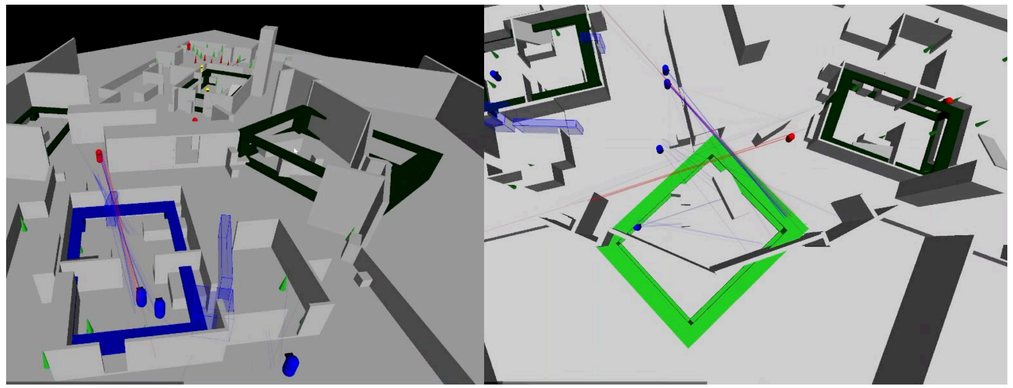
Human-compatible cooperative agents



Assessing 3D game difficulty

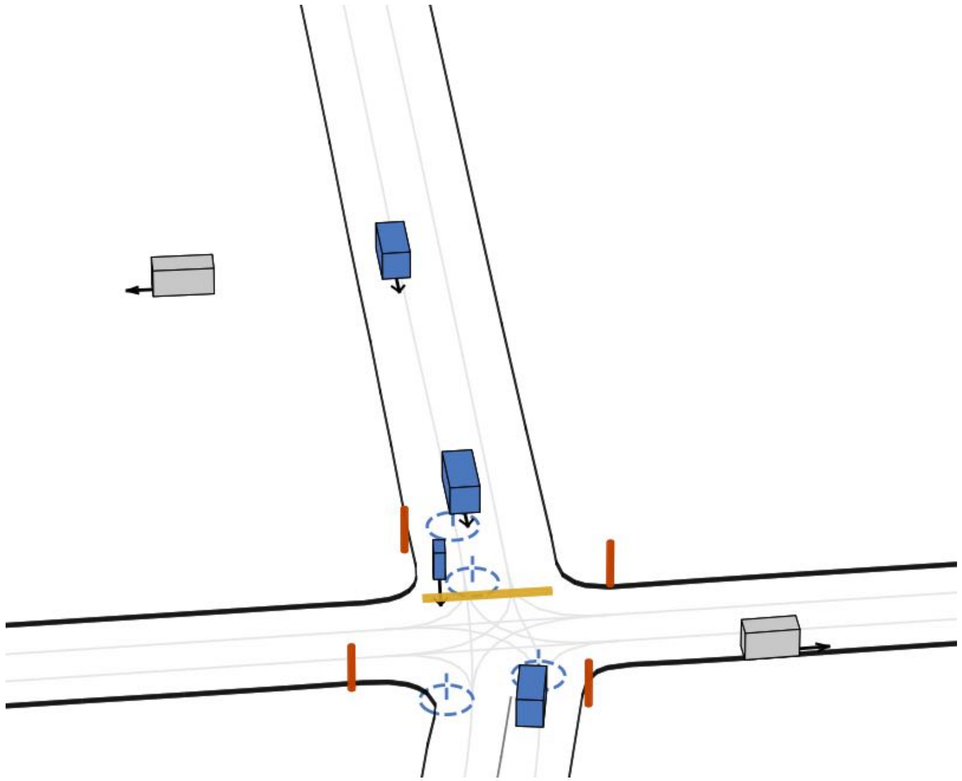


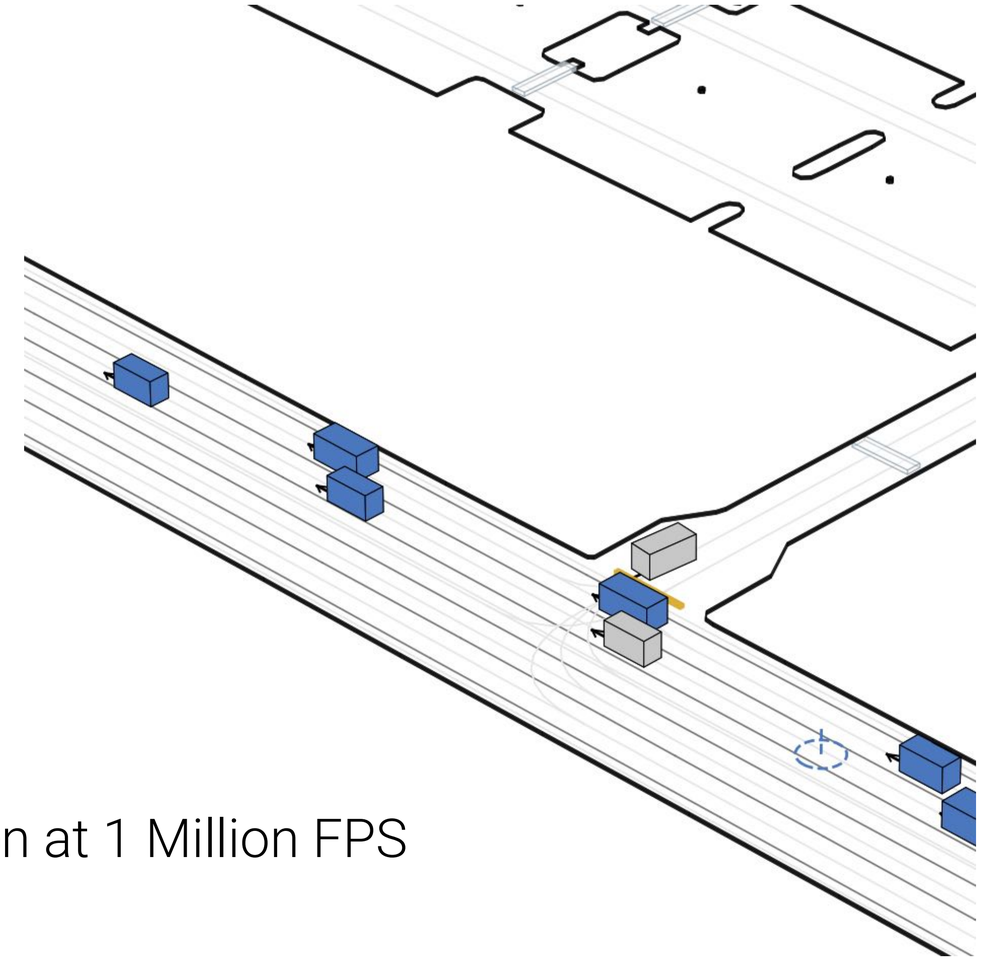
Pixels to actions training with Mujoco MJX



Learning competitive strategies in 6v6 video game

GPUDrive: Multi-Agent High-Level Autonomous Driving Batch Simulator



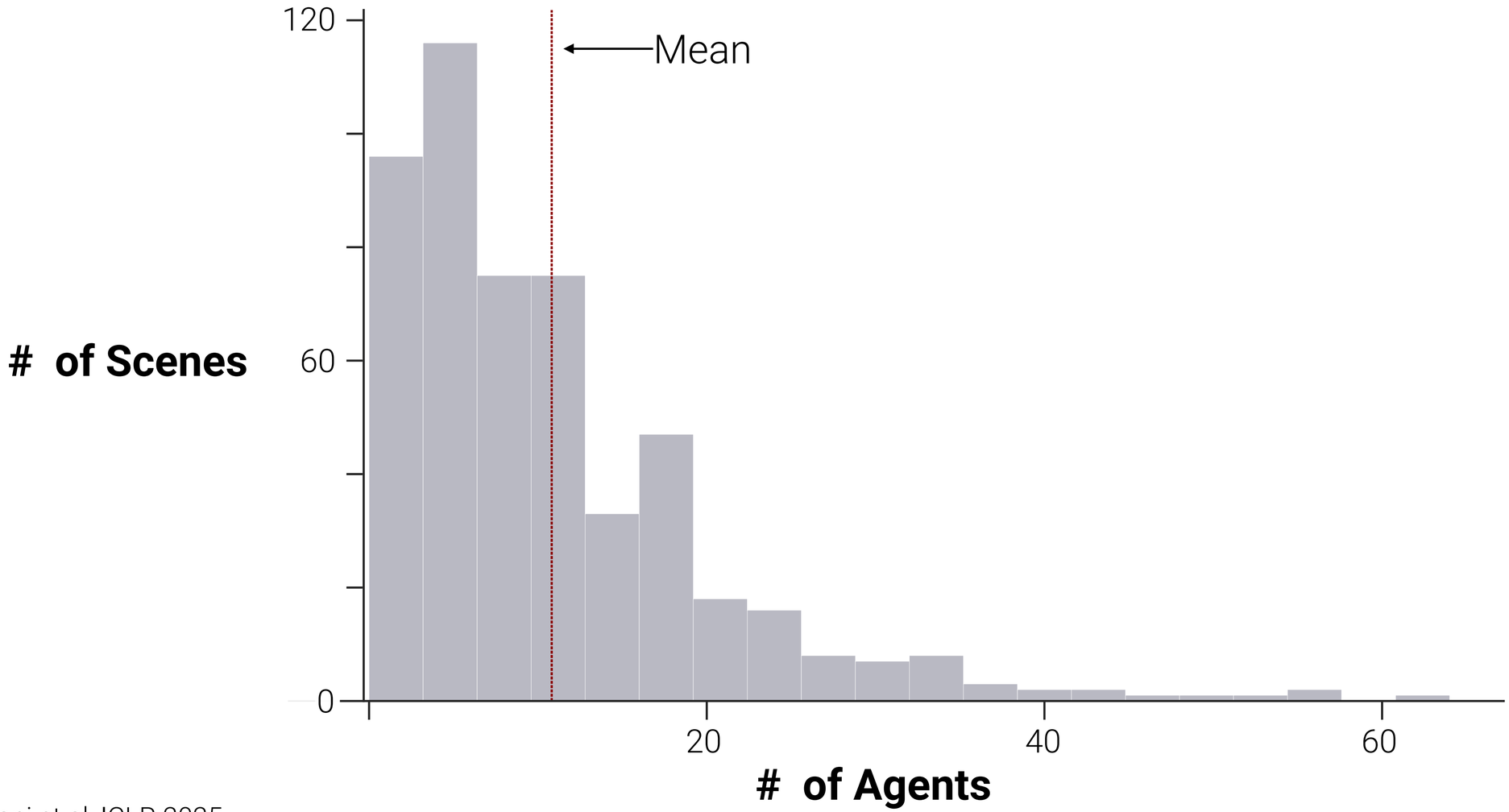


GPUDrive: Data-Driven, Multi-Agent Driving Simulation at 1 Million FPS

Saman Kazemkhani, Aarav Pandya, Daphne Cornelisse,
Brennan Shacklett, Eugene Vinitsky

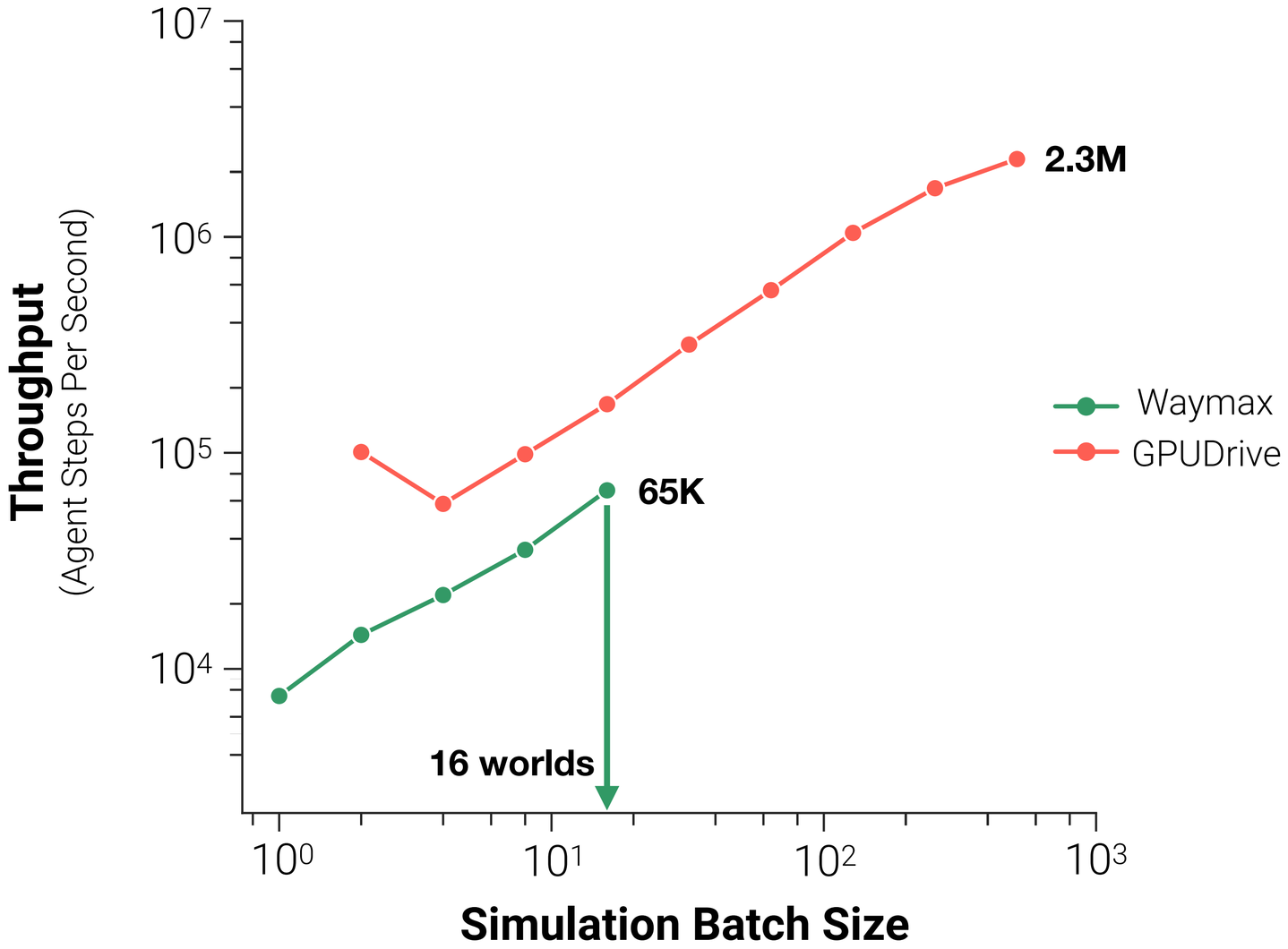
ICLR 2025

Irregular parallelism: Waymo Open Dataset contains wide range of agents counts per scene



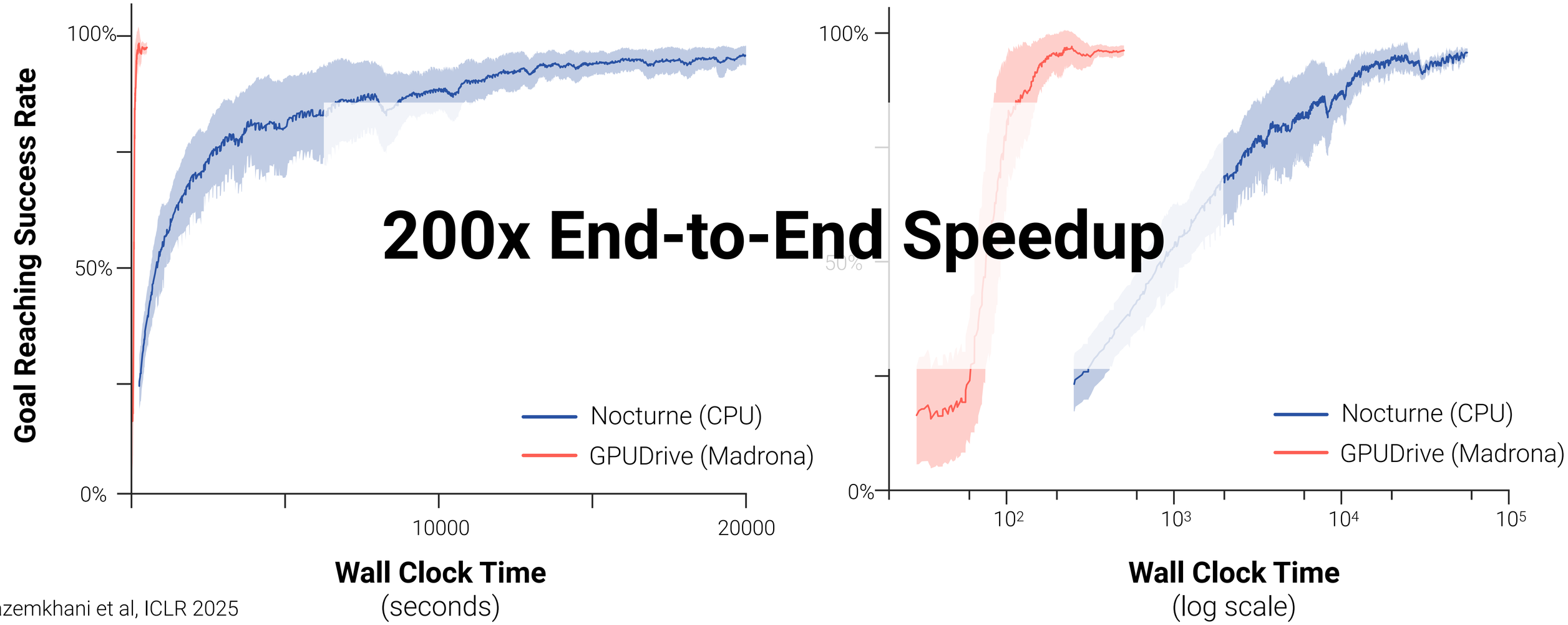
Kazemkhani et al, ICLR 2025

Overallocation in JAX-based Baseline (Waymax) Severely Limits Peak Parallelism Relative to GPUDrive



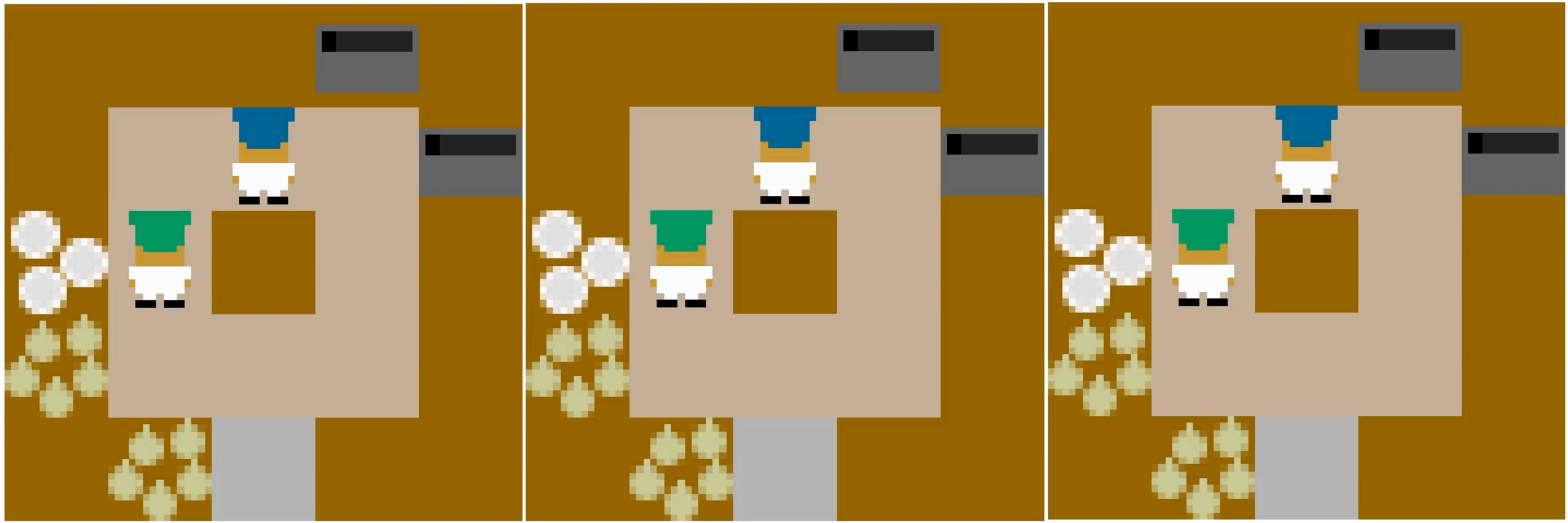
Kazemkhani et al, ICLR 2025

GPUDrive training reaches >95% Success in <5 minutes, vs >10 hours with CPU baseline



Kazemkhani et al, ICLR 2025

Learning Human-Compatible Agents in Overcooked-AI

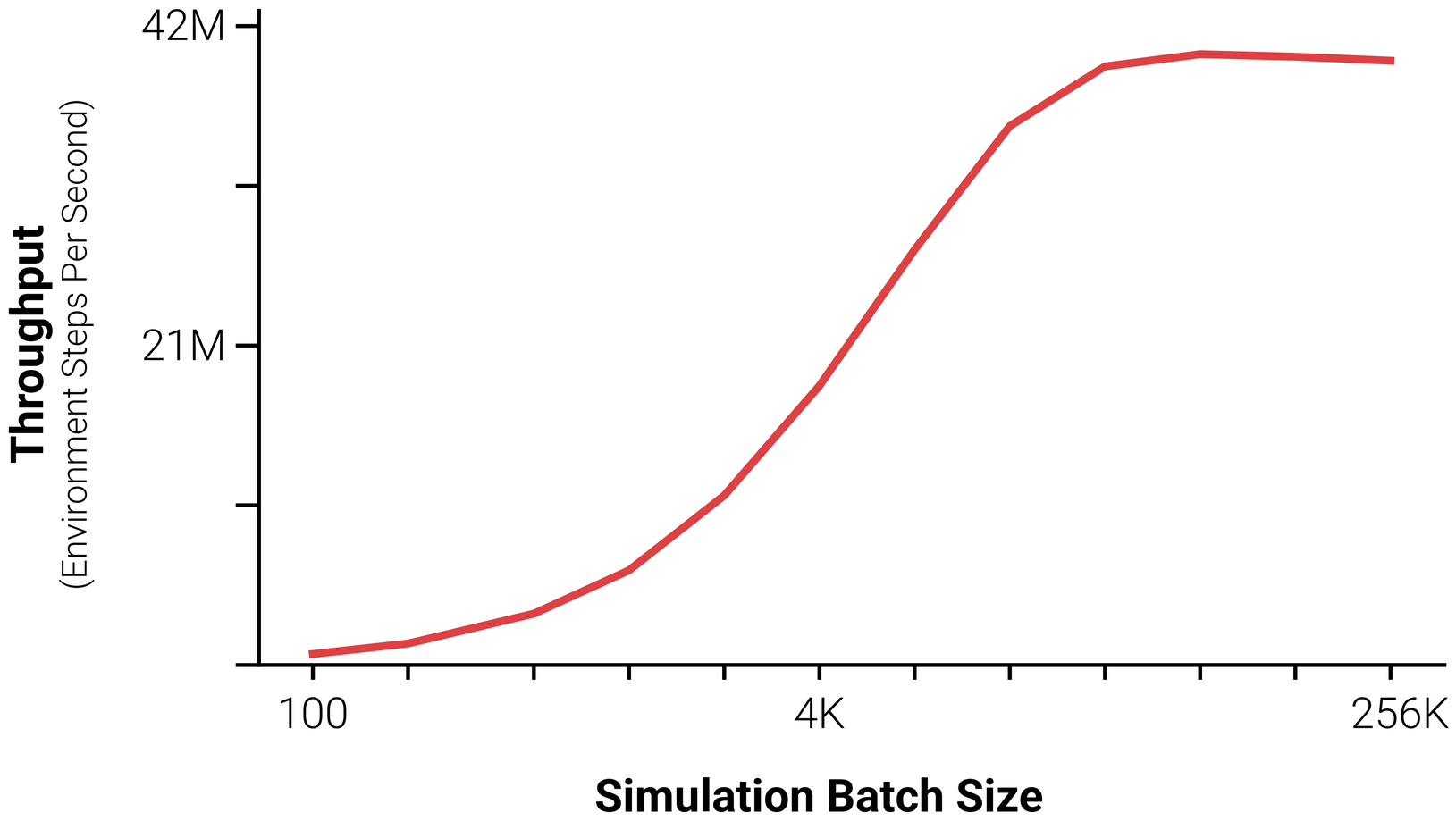


Diverse Conventions for Human-AI Collaboration

Bidipta Sarkar, Andy Shih, Dorsa Sadigh

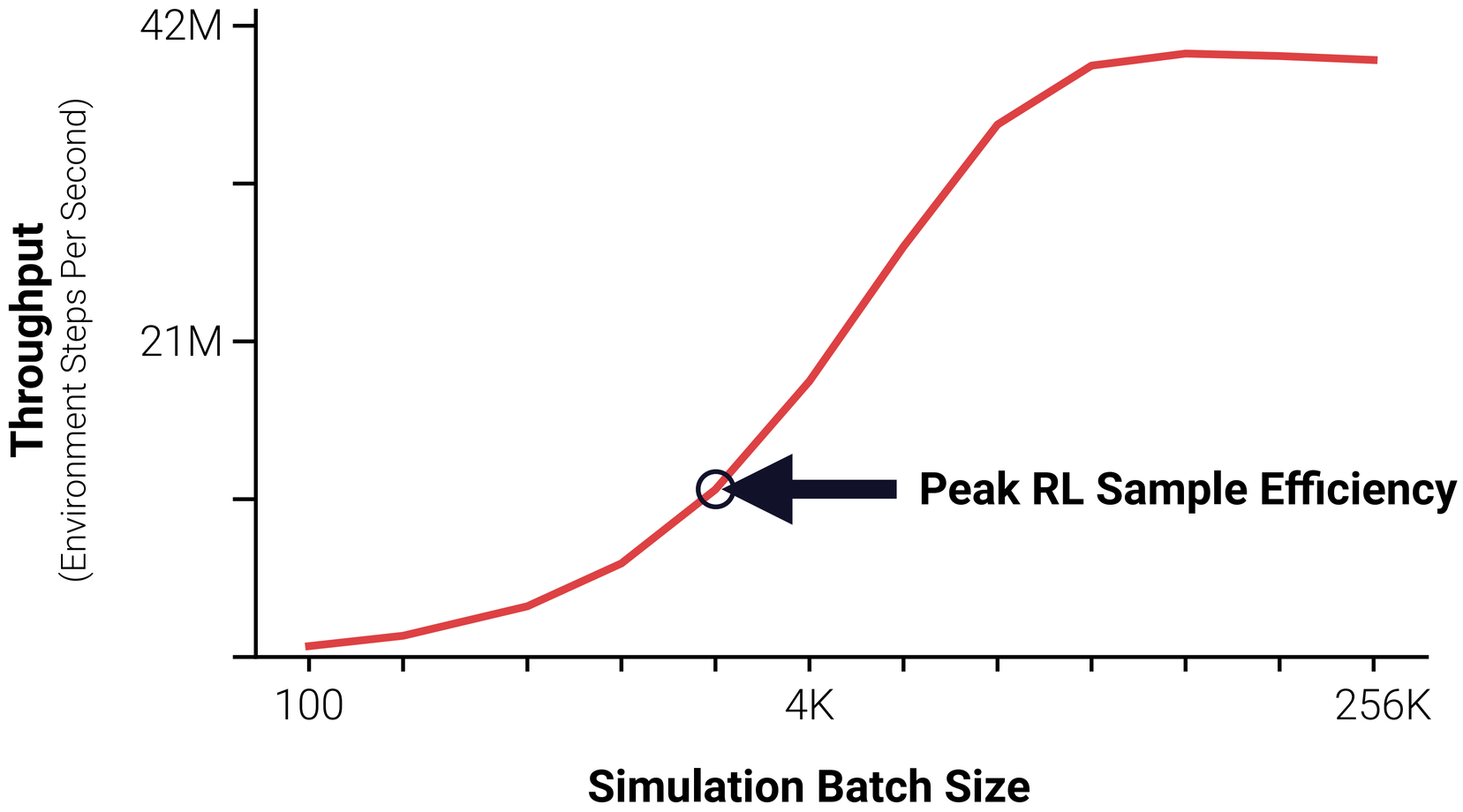
NeurIPS 2023

Batch simulator throughput scales with number of parallel environments



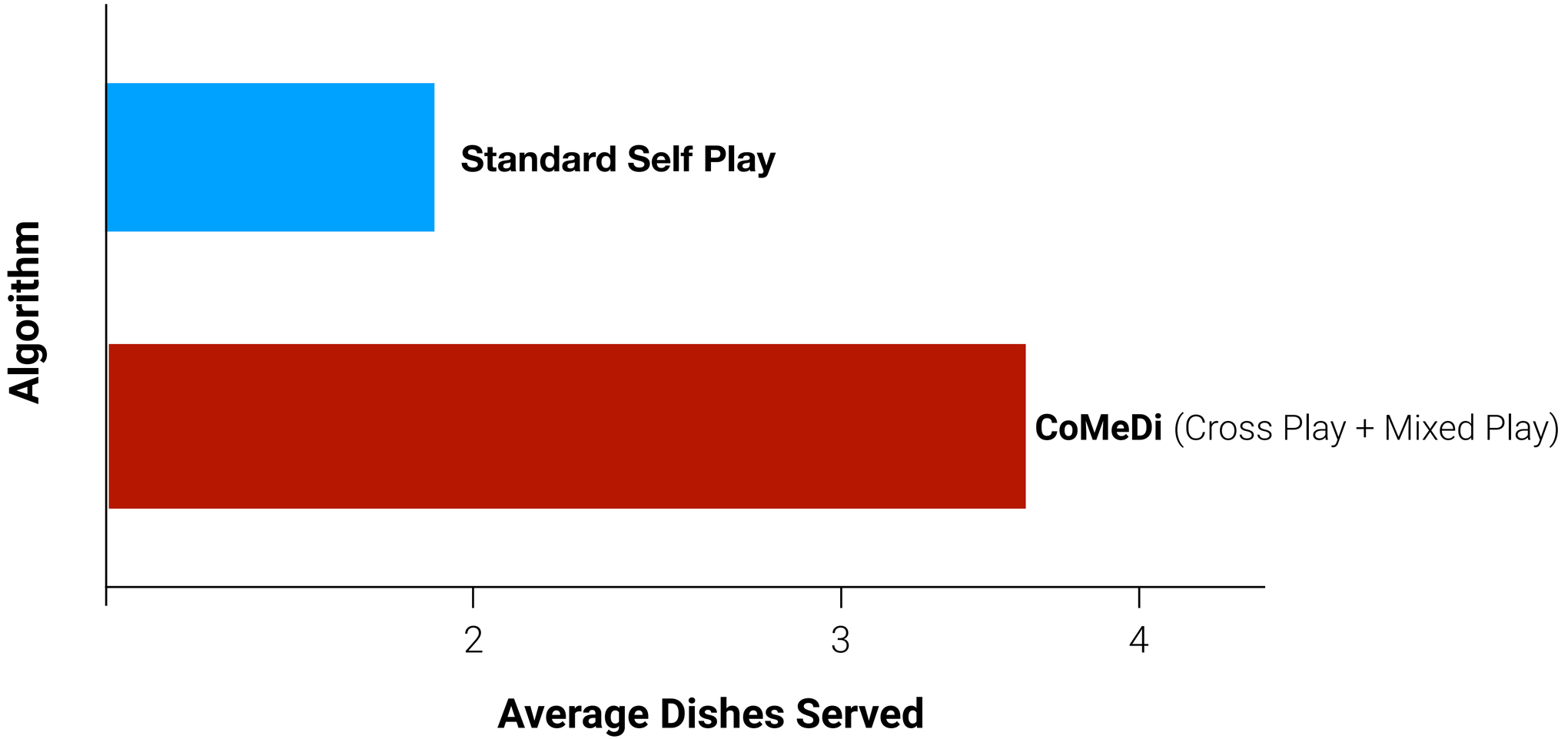
Sarkar et al, NeurIPS 2023

Training sample efficiency hits diminishing returns far below peak simulation throughput



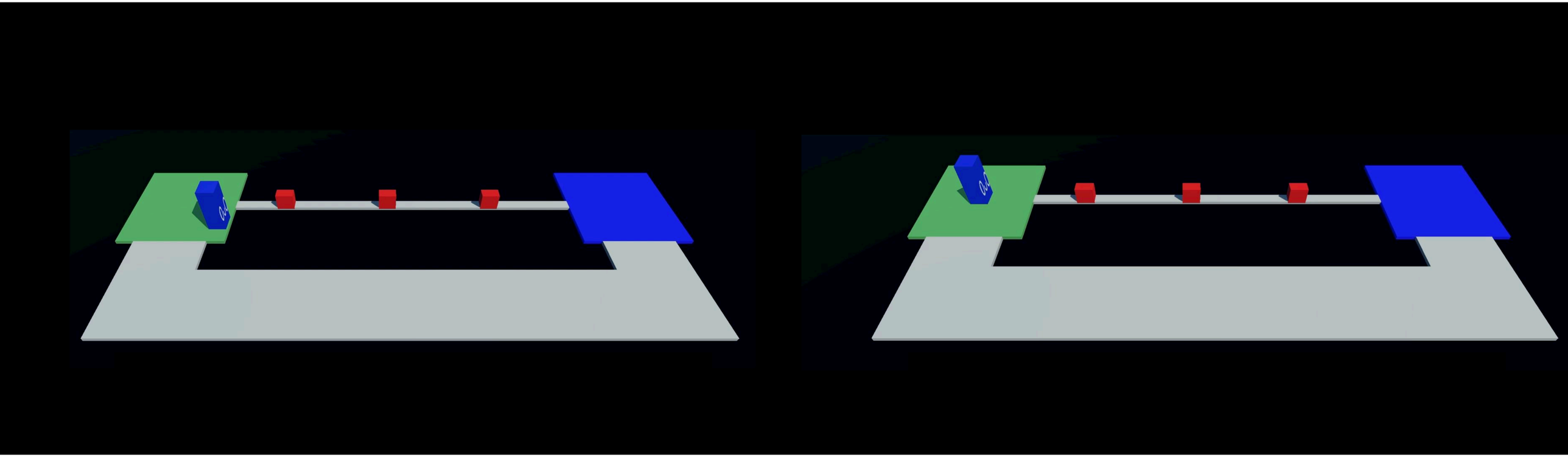
Sarkar et al, NeurIPS 2023

Idea: Use extra simulation throughput to train robust agents that can cooperate with other policies



Sarkar et al, NeurIPS 2023

Learning Large Population of Agents with Varying Skill Levels to Assess Difficulty in Obstacle Course Games

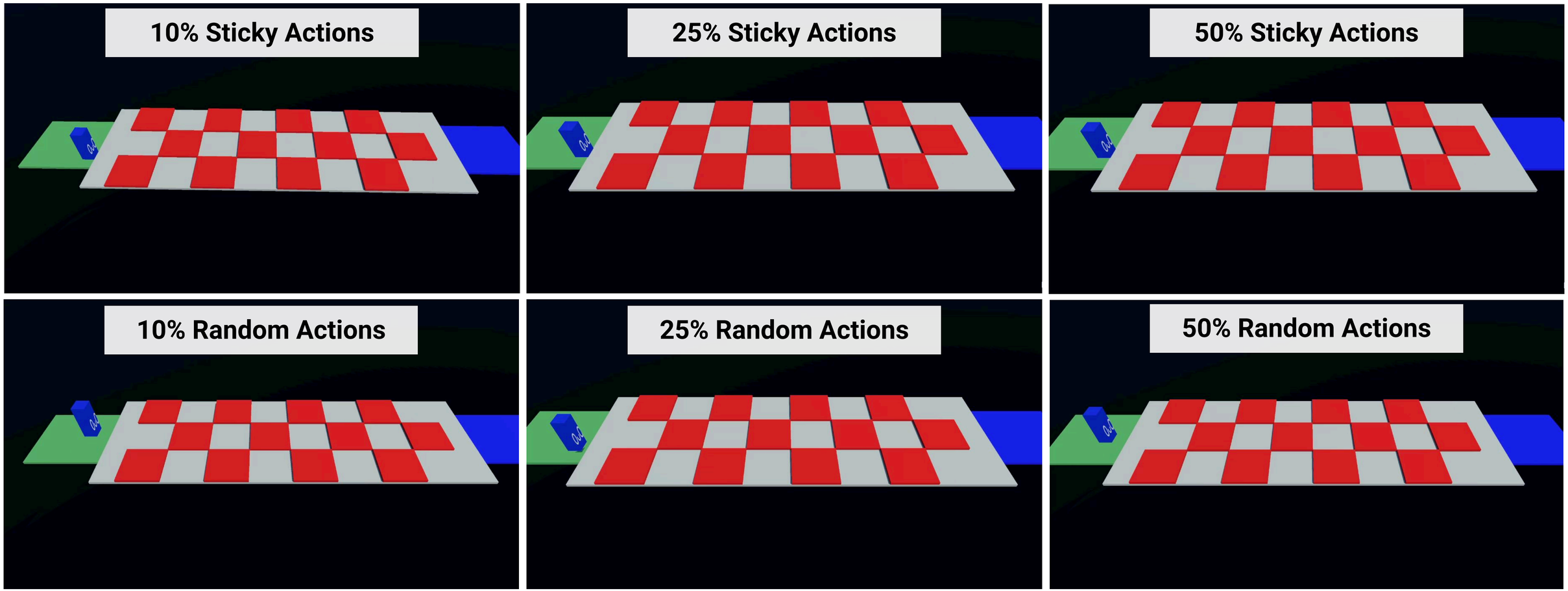


Modeling Human Limitations in RL Agents For Difficulty Assessment in Obstacle Course Games

Zander Majercik, Sharon Zhang, Will Wang, Brennan Shacklett, Maneesh Agrawala, Kayvon Fatahalian

Under Submission SIGGRAPH 2025

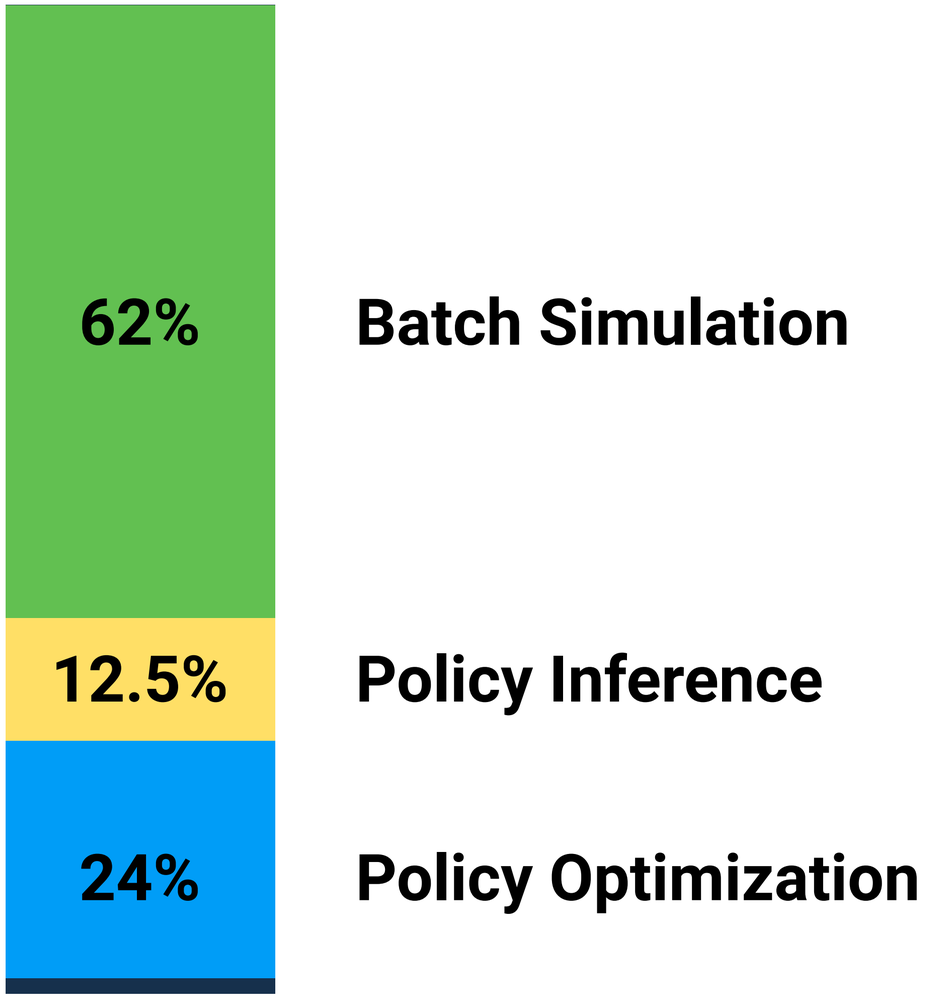
26 years of game-play experience in 13 hours allows training many agents with wide range of skill levels



Majercik et al, Under submission

Lightweight policy network + optimized PPO implementation: Simulator still dominant cost

1.2M FPS end-to-end training speed (RTX 4090)

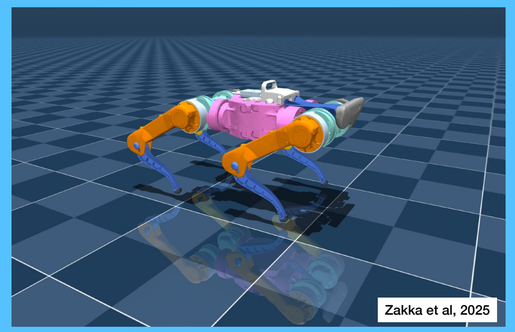


Sim-to-Sim: Policies trained in Madrona can transfer directly to unseen obstacles in Roblox

Maiercik et al.

Integrating Madrona Batch Renderer with Mujoco MJX

Mujoco MJX



World States

Madrona Batch Renderer

Observations

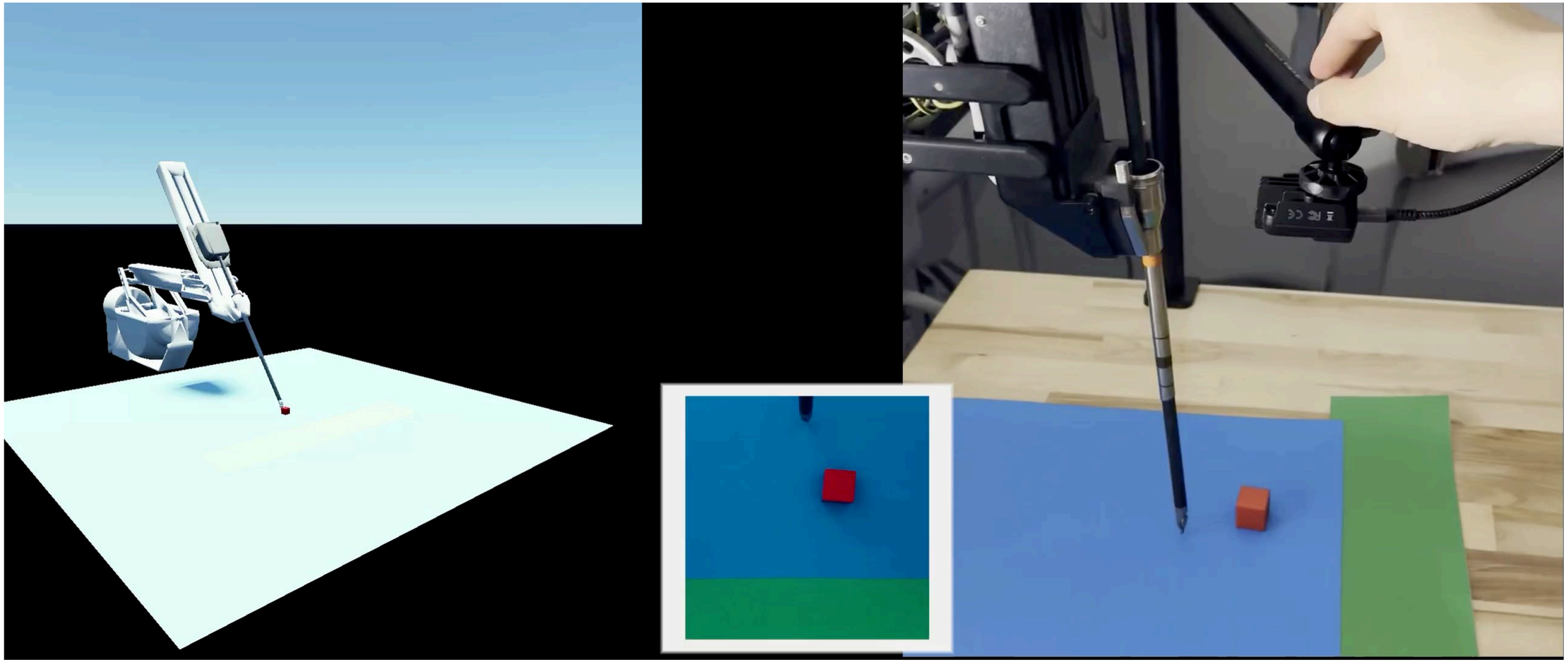
Policy Inference
(DNN forward pass)

Agent Actions

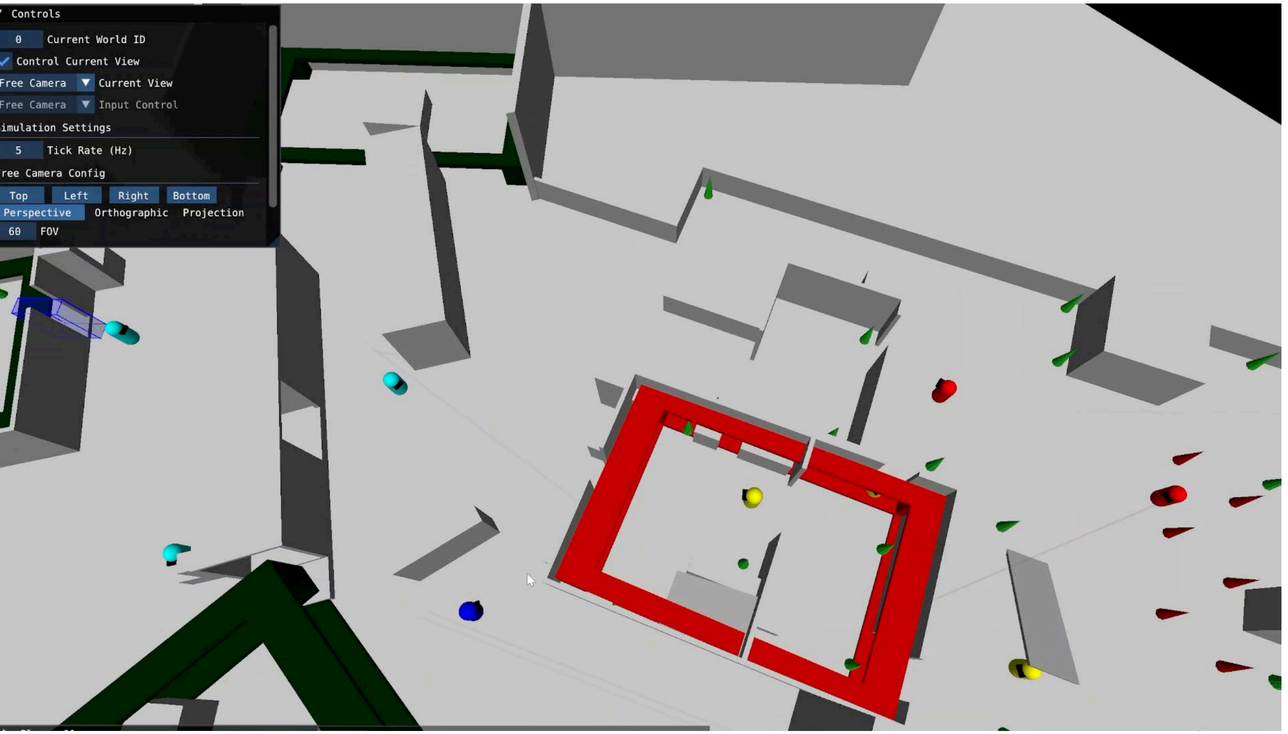
Policy Weights

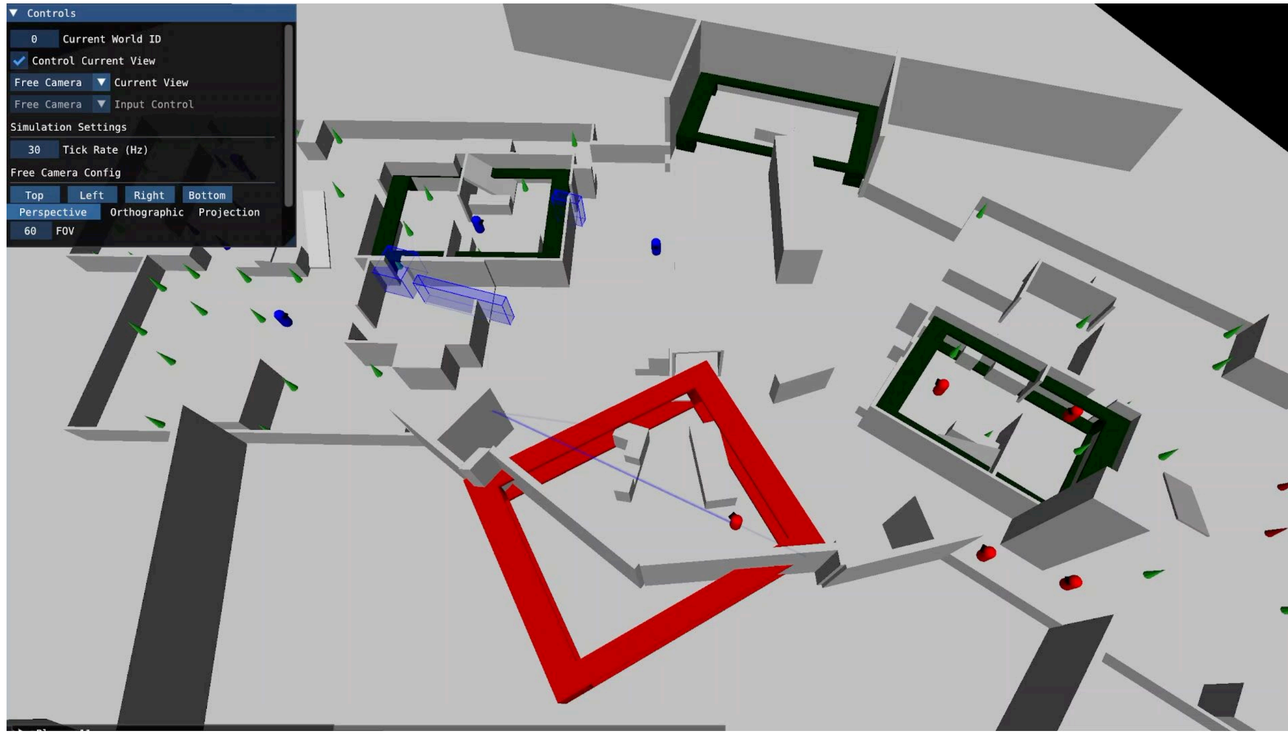
Policy Optimization
(DNN forward & backward)

Simple sim-to-real transfer with domain randomization

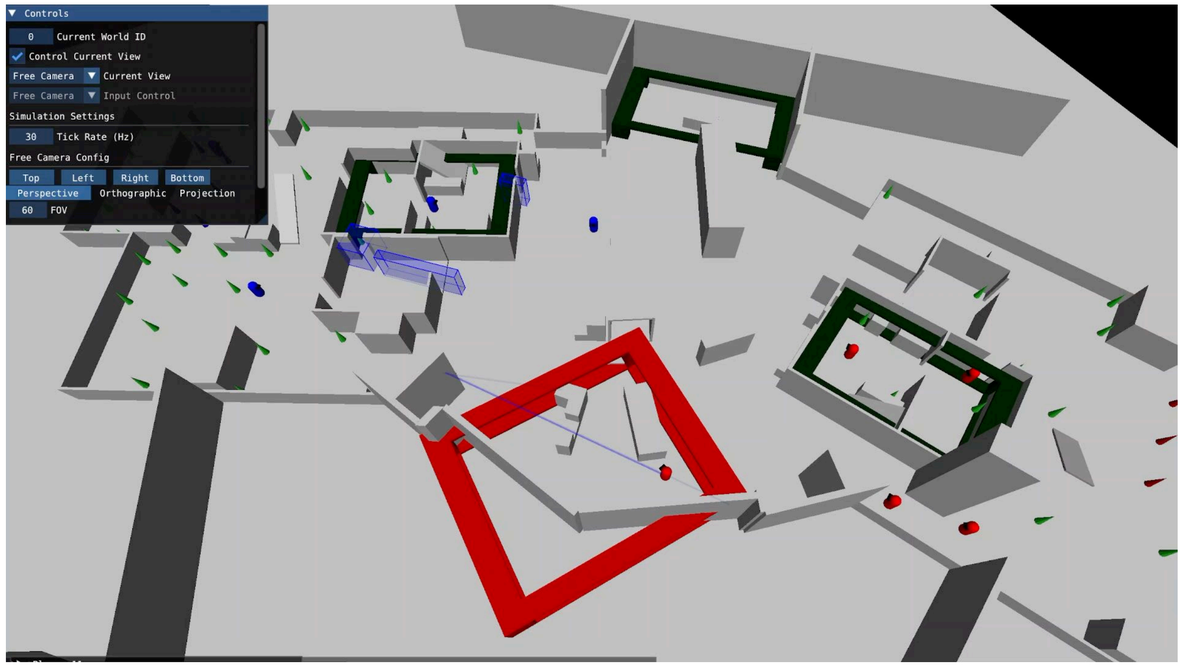


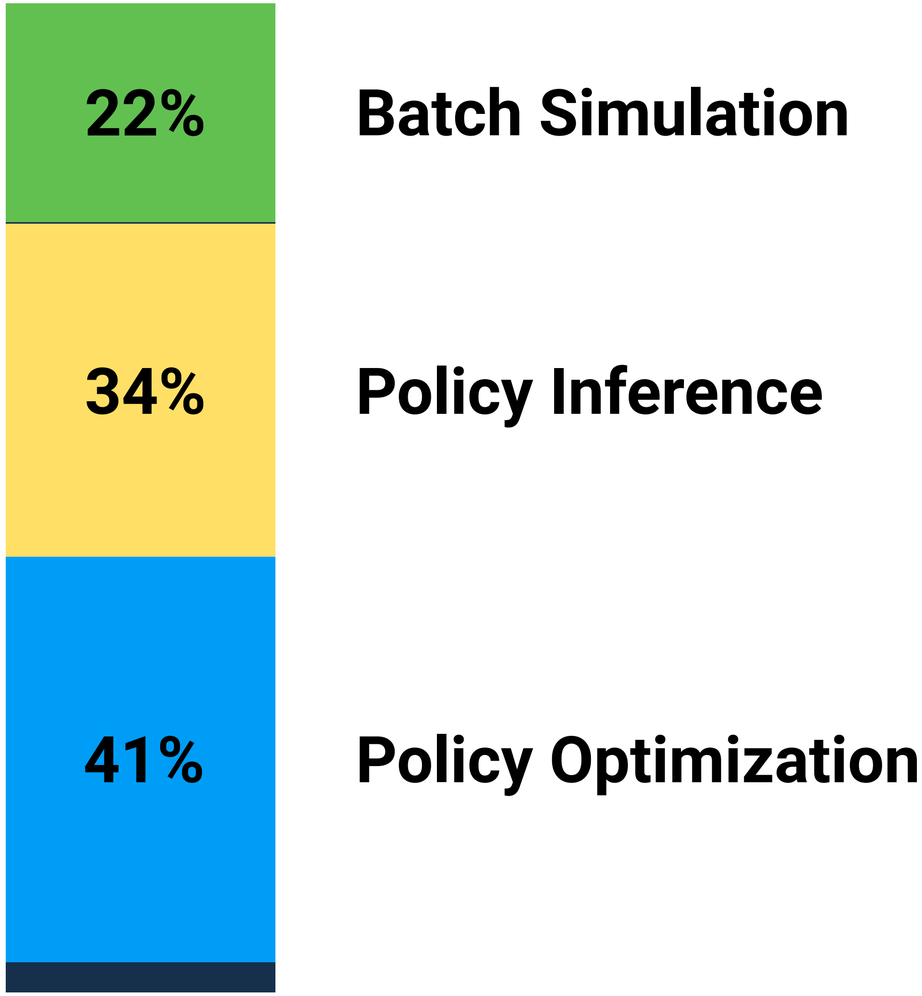
Learning competitive strategies in 6 vs 6 video game





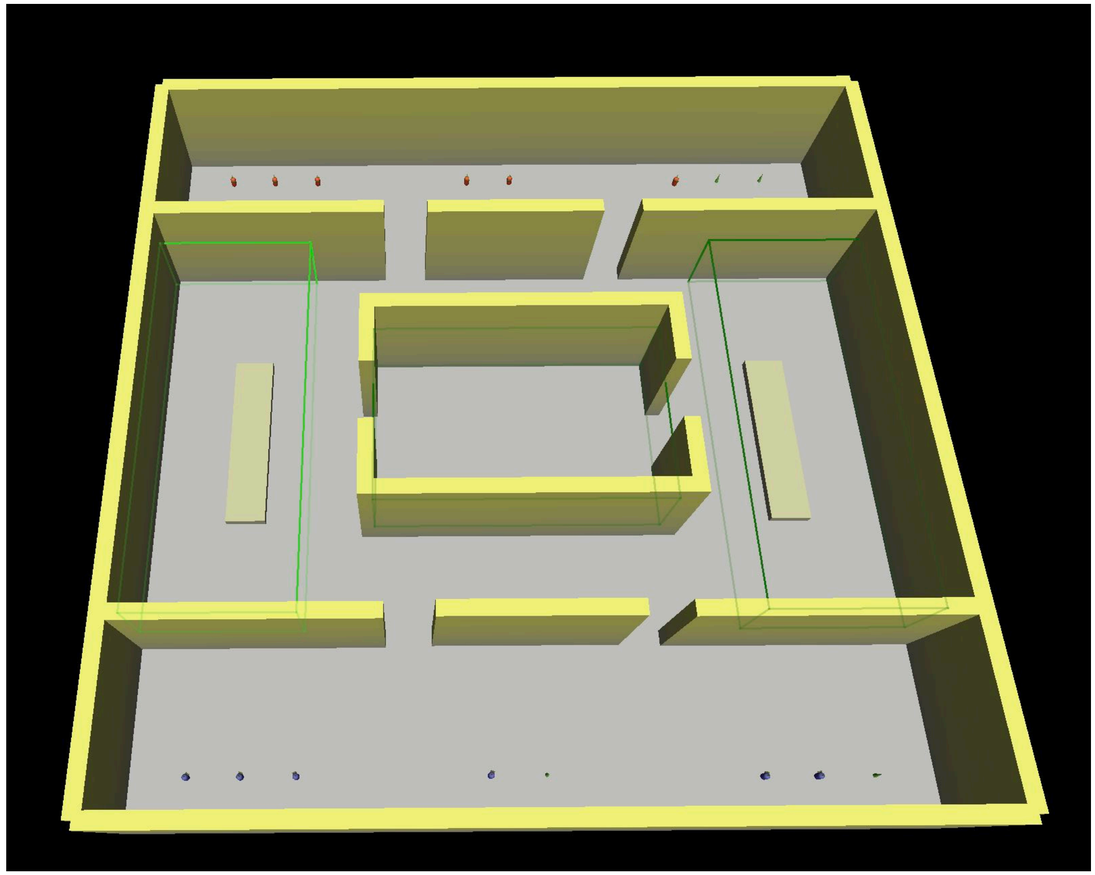
12 independently controlled agents makes policy inference & optimization majority of end-to-end time





Population-based training: multiple policy DNNs learning & competing simultaneously on one GPU

- **Training with population of 16 DNNs**
- **JAX-based training codebase supports training multiple policies on single GPU**
  - Training opponents chosen dynamically based on skill level



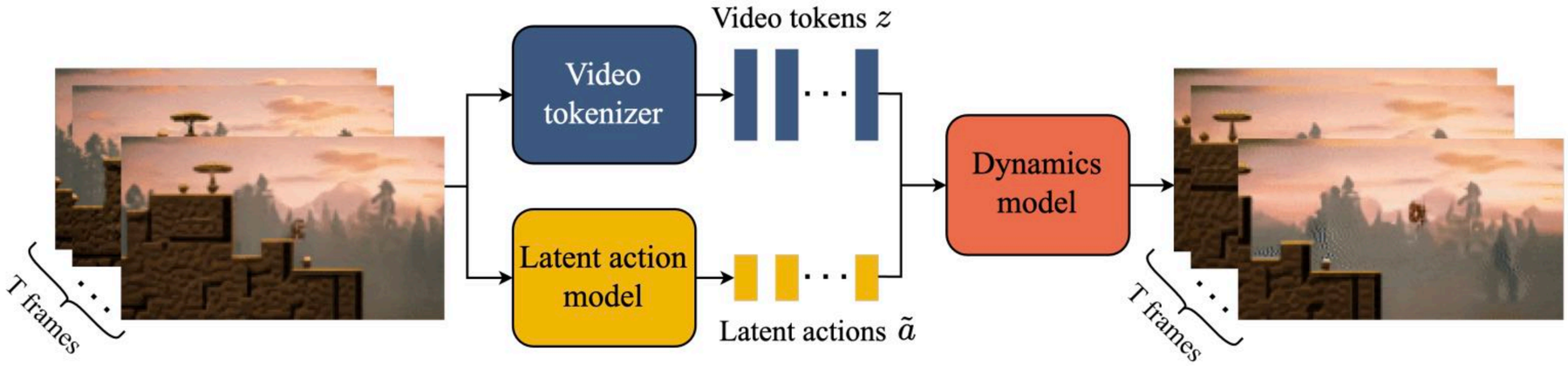
Tonight's reading topic:
An alternative approach to simulating worlds:
Generative AI as a means to generate world simulation output

Enhancing CG images to look like real-world images using image-to-image transfer

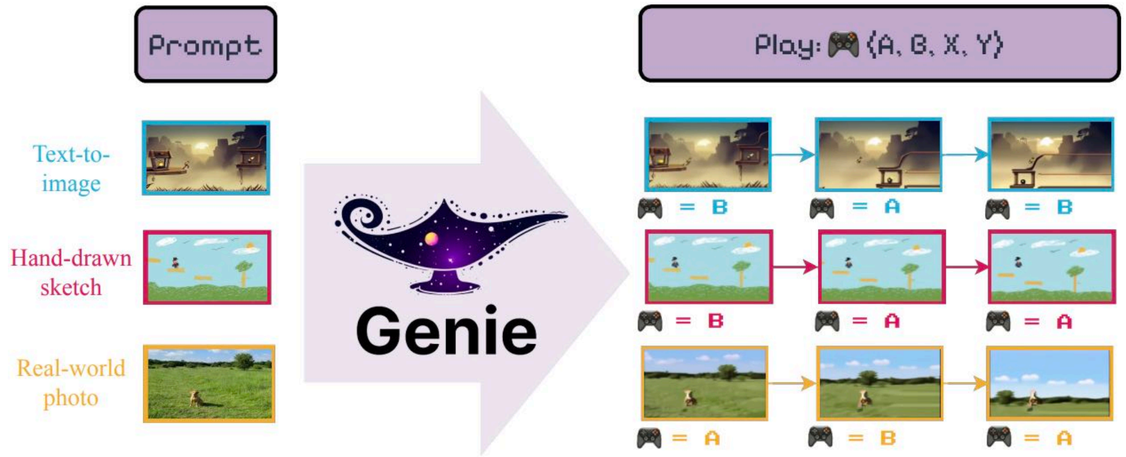
Modifying real-world images to create novel situations

Genie

- Key idea: learn a world simulator from videos of video game play
  - From video, learn latent user actions, and dynamics model that steps work given (current state, action)

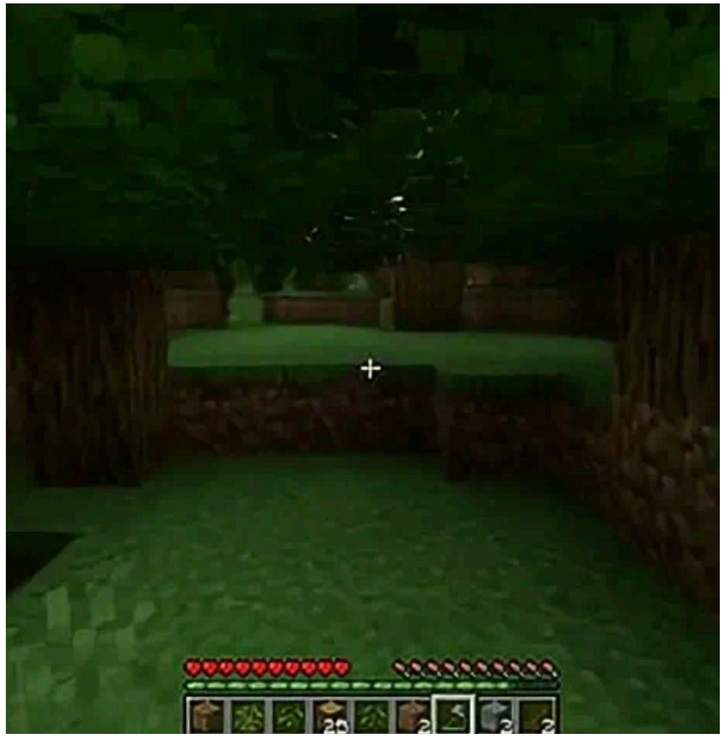


  - Then at "test time" given a novel world state (perhaps one generated from a prompt), and given user input, time step the novel world forward in time



Other recent world model examples

Oasis



Diamond

Next class we'll have a "debate"

- After reading the required reading and skimming through suggested readings for lectures 11 and 12, I want you to make a bet:

- If in 10 years you were to bet on a simulator implementation approach for producing the virtual environments for agent training, do you think that implementation will look more like:
  - Game engines today, where aspects of the world situation (physics/rendering/ programmable logic) continue to me explicitly modeled
  - Emerging "world models" where a learned network is responsible for producing the next state of the world given the agent's agent.