PhysiGym: Reinforcement learning on PhysiCell agent-based models.

Elmar Bucher, Ph.D. Student

Intelligent Systems Engineering
Indiana University

Society of Mathematical Biology, 2025-07-17

LUDDY
SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Macklin Lab
@MathCancer.bsky.social
MathCancer.org

Acknowledgement

MathCancer Lab:

- Aneequa Sundus
- Heber Rocha
- Randy Heiland
- Paul Macklin

Pancaldi and Rachelson Lab:

- Alexandre Bertin
- Marcelo Hurtado
- Owen Griere
- Vincent François
- Emmanuel Rachelson
- Vera Pancaldi

Founding:

- Chateaubriand Fellowship, Embassy of France in the United States
- Occitanie Region Toulouse, France
- Inserm, France

Talk outlines

1. PhysiCell agent-based modeling basics.
2. Gymnasium and reinforcement learning basics.
3. PhysiGym implementation.
4. Reinforcement learning on the tumor_immune_basic model.
5. Result and Discussion.

PhysiCell & Gymnasium Basics

PhysiCell agent base modeling software

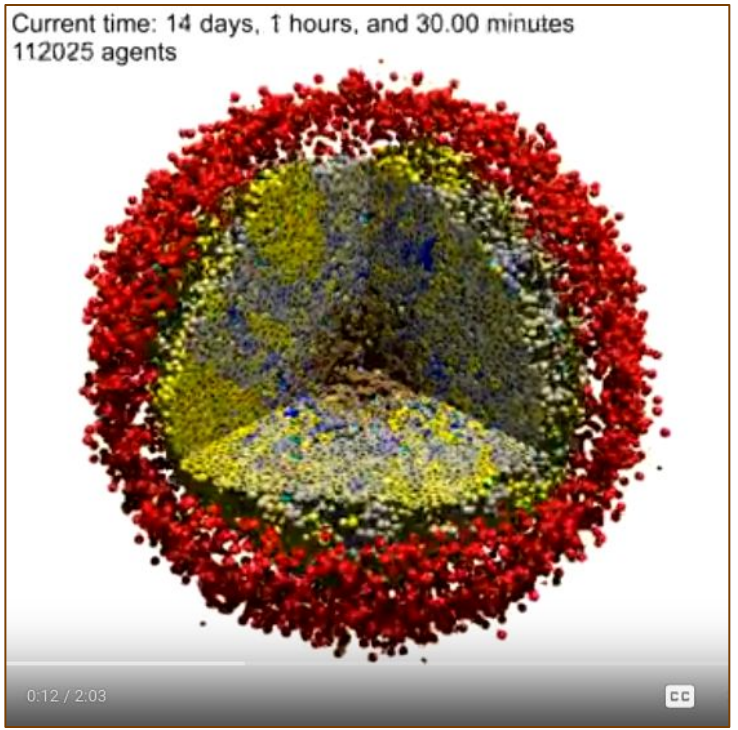
Agent-based modeling:

- ★ mathematical, dynamical system modeling approach.
- ★ concept: the parts of the system (agents) act autonomously, according to agent-type specific rules.

PhysiCell:

- ★ AB modeling framework written in C++.
- ★ implemented to model multicellular systems based on Newtonian physics.
- ★ Cells: agents.
- ★ Cell_type: specifies the rule set the agents apply.
- ★ Tissue structure: emerges from the cell interactions.
- ★ Substrates: can be modeled with the integrated BioFVM diffusive transport solver.
- ★ Intracellular models: can be integrated into cell agents.

The resulting AB models are 2 or 3-dimensional, off-lattice, center-based, and multiscale in space and time.



The tumor immune basic agent-based model

Cell_type and substrates:

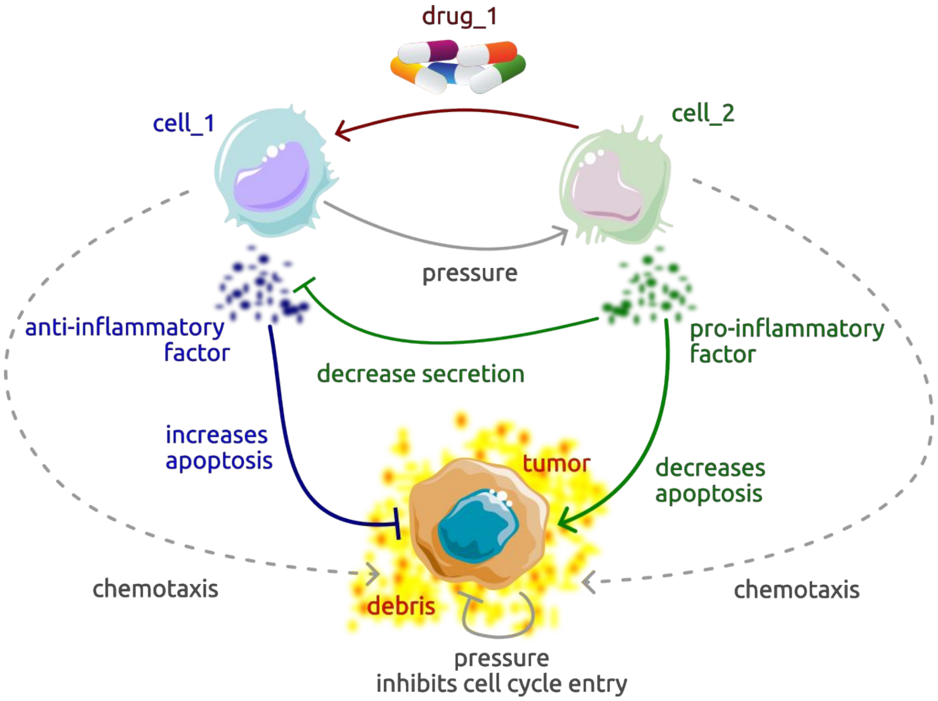
- ★ cell_1 -> anti-inflammatory factor
- ★ cell_2 -> pro-inflammatory factor
- ★ tumor -[*apoptosis*]-> debris

Rules:

- ★ **drug** increases cell_2 to cell_1 transformation!
- ★ [*pressure*] increases cell_1 to cell_2 transformation.
- ★ tumor: anti-tumoral factor increases [*apoptosis*].
- ★ tumor: pro-inflammatory factor decreases [*apoptosis*].
- ★ cell_1: pro-inflammatory factor decreases anti-inflammatory factor secretion.

Others:

- ★ cell_1 are motile and chemo taxi towards debris.
- ★ cell_2 are motile and chemo taxi towards debris.
- ★ tumor are immotile.



Reinforcement learning

Aim: learning a (drugging) policy to control the environment.

Algorithm:

1. Observe the environment.
2. Learn policy: based on reward and observation.
3. Apply policy: calculate action based on reward and observation.
4. Apply action to the environment.
5. Receive reward.
6. Goto 1.

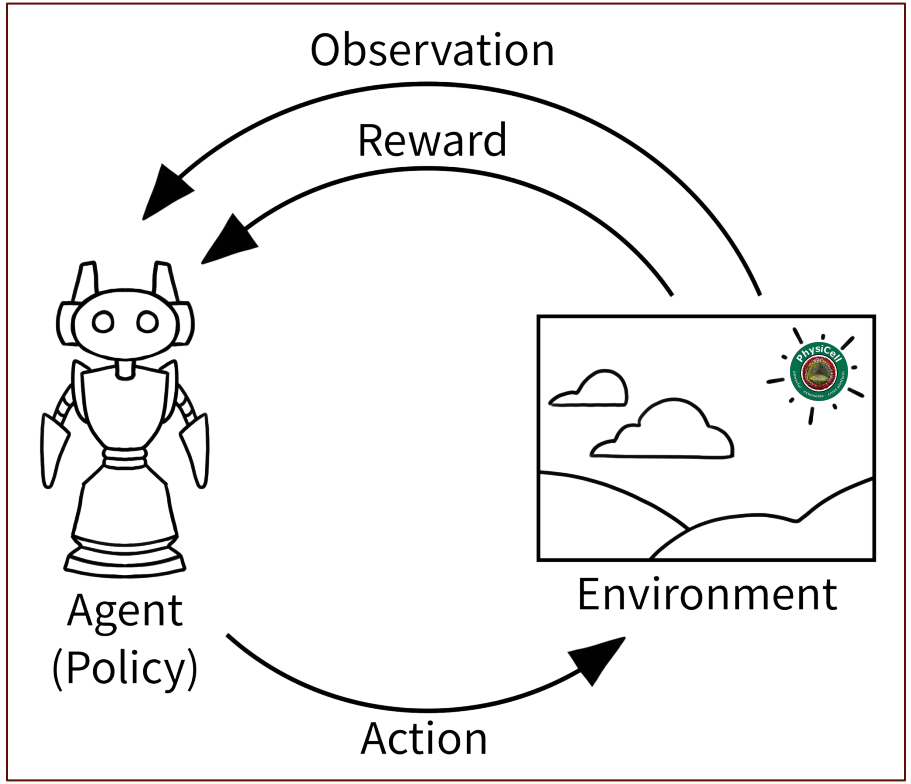
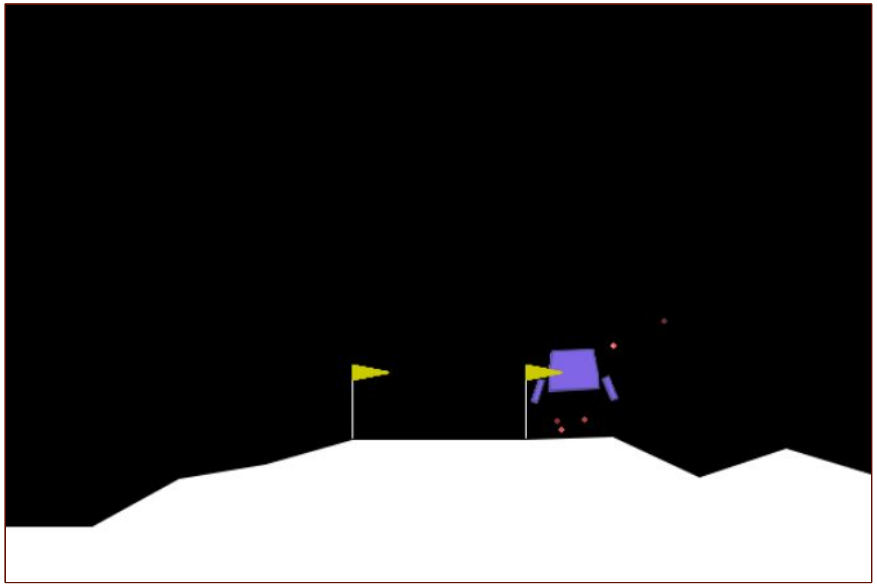


Figure adapted from: https://gymnasium.farama.org/main/introduction/basic_usage/

Reinforcement learning and Gymnasium

```python
# Library
import gymnasium

# Initialize the environment
env = gymnasium.make('LunarLander-v3', render_mode='human')

# Reset the environment to generate the first observation
o_observation, d_info = env.reset()

# Running episodes
for _ in range(1024):

    # Apply policy: calculate action based on reward and observation.
    o_action = env.action_space.sample()

    # Apply action to environment.
    # Receive next observation, reward, and if the episode finished.
    o_observation, r_reward, b_terminated, b_truncated, d_info = \
    env.step(o_action)

    # If the episode ended, reset to start a new episode.
    if b_terminated or b_truncated:
        o_observation, d_info = env.reset()

# Shutdown environment
env.close()
```

python powered



LUDDY SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

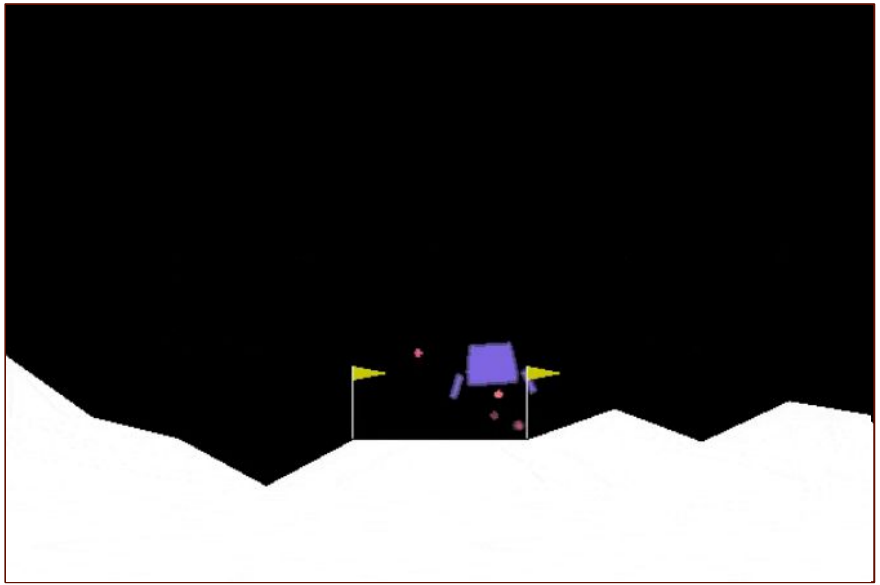
Code and Figure adapted from: https://gymnasium.farama.org/main/introduction/basic_usage/

Macklin Lab
@MathCancer.bsky.social
MathCancer.org

Reinforcement learning and Gymnasium

```python
# Library
import gymnasium

# Initialize the environment
env = gymnasium.make('LunarLander-v3', render_mode='human')

# Reset the environment to generate the first observation
o_observation, d_info = env.reset()

# Running episodes
for _ in range(1024):

    # Apply policy: calculate action based on reward and observation.
    o_action = env.action_space.sample()

    # Apply action to environment.
    # Receive next observation, reward, and if the episode finished.
    o_observation, r_reward, b_terminated, b_truncated, d_info = \
    env.step(o_action)

    # If the episode ended, reset to start a new episode.
    if b_terminated or b_truncated:
        o_observation, d_info = env.reset()

# Shutdown environment
env.close()
```

python powered



LUDDY SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Code adapted from: https://gymnasium.farama.org/main/introduction/basic_usage/
Figure adapted from: https://stable-baselines3.readthedocs.io/en/master/guide/examples.html

Macklin Lab
@MathCancer.bsky.social
MathCancer.org

Reinforcement learning and Gymnasium

```python
# Library
import gymnasium

# Initialize the environment
env = gymnasium.make('LunarLander-v3', render_mode='human')

# Reset the environment to generate the first observation
o_observation, d_info = env.reset()

# Running episodes
for _ in range(1024):

    # Apply policy: calculate action based on reward and observation.
    o_action = env.action_space.sample()

    # Apply action to environment.
    # Receive next observation, reward, and if the episode finished.
    o_observation, r_reward, b_terminated, b_truncated, d_info = \
    env.step(o_action)

    # If the episode ended, reset to start a new episode.
    if b_terminated or b_truncated:
        o_observation, d_info = env.reset()

# Shutdown environment
env.close()
```

python powered

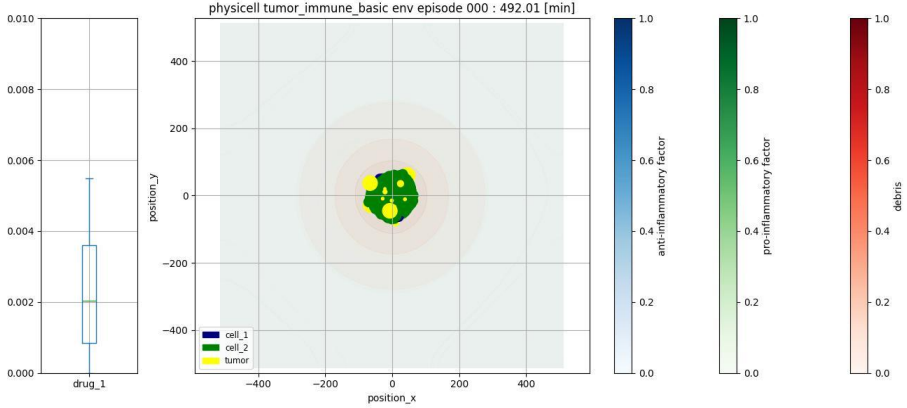
gymnasium ships with a diverse collection of reference environments used in the RL community for benchmarking.

developing control policies, observation strategies, reward functions, and learning algorithms is an open research field.

PhysiCell and Gymnasium

a computer engineering challenge

```python
# Library
import gymnasium

# Initialize the environment
env = gymnasium.make('ModelPhysiCellEnv-v0', render_mode='human')

# Reset the environment to generate the first observation
o_observation, d_info = env.reset()

# Running episodes
for _ in range(1024):

    # Apply policy: calculate action based on reward and observation.
    o_action = env.action_space.sample()

    # Apply action to environment.
    # Receive next observation, reward, and if the episode finished.
    o_observation, r_reward, b_terminated, b_truncated, d_info = \
    env.step(o_action)

    # If the episode ended, reset to start a new episode.
    if b_terminated or b_truncated:
        o_observation, d_info = env.reset()

# Shutdown environment
env.close()
```



Reinforcement learning and oncology

a PhD challenge

```python
# Library
import gymnasium

# Initialize the environment
env = gymnasium.make('ModelPhysiCellEnv-v0', render_mode='human')

# Reset the environment to generate the first observation
o_observation, d_info = env.reset()

# Running episodes
for _ in range(1024):

    # Apply policy: calculate action based on reward and observation.
    o_action = env.action_space.sample()

    # Apply action to environment.
    # Receive next observation, reward, and if the episode finished.
    o_observation, r_reward, b_terminated, b_truncated, d_info = \
    env.step(o_action)

    # If the episode ended, reset to start a new episode.
    if b_terminated or b_truncated:
        o_observation, d_info = env.reset()

# Shutdown environment
env.close()
```

★ writing agent-based models that make biological sense.

★ developing control policies based on observation and reward functions that make practical sense.

LUDDY SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Code adapted from: https://gymnasium.farama.org/main/introduction/basic_usage/

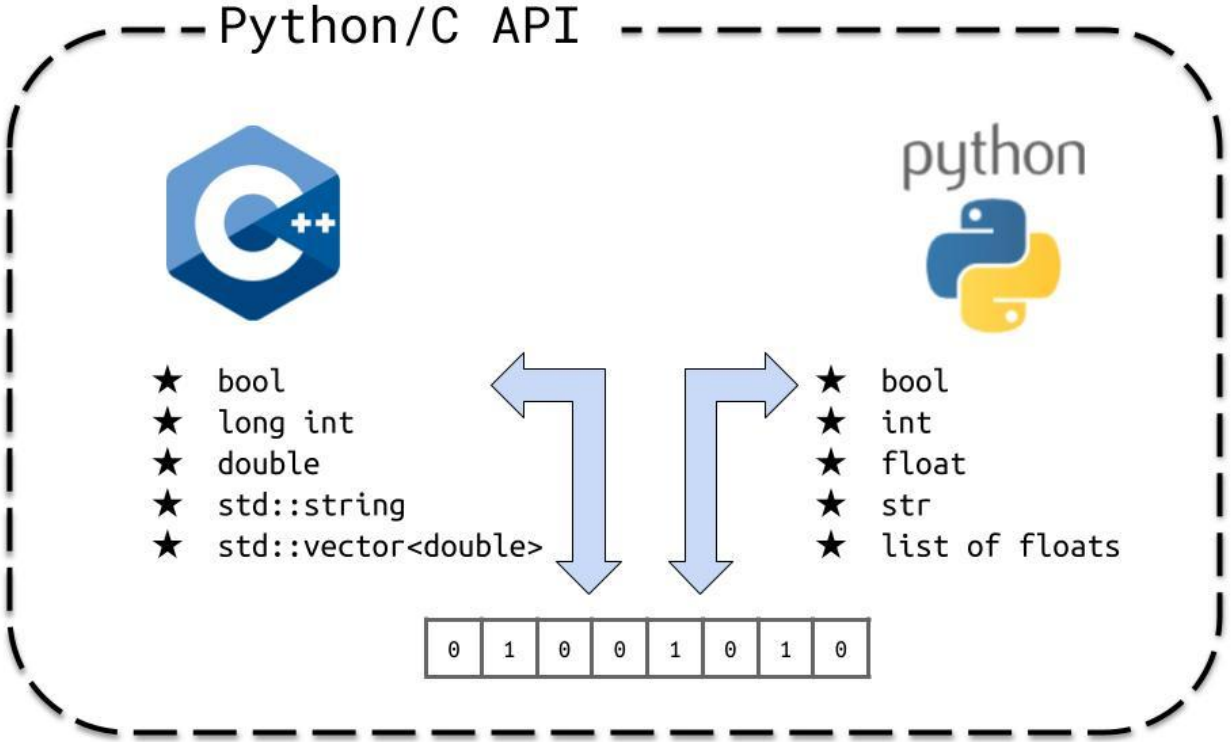
PhysiGym Implementation

Challenge I: Run the Main Loop


```
g++ main.cpp -o project
./project
```

```
from extending import physicell

physicell.start()
physicell.step()
physicell.stop()

exit()
```

Challenge II: Python/C++ Interface



Challenge II: C++/Python Interface

Python/PhysiCell API

- Parameter<bool>
- Parameter<int>
- Parameter<double>
- Parameter<std::string>

- pCell->custom_data.variables<double>
- pCell->custom_data.vector_variables<double>

- pCell->position
- microenviroment.mesh and substrate

- from extending import physicell

- physicell.set_parameter()
- physicell.set_variable()
- physicell.set_vector()

- physicell.get_parameter()
- physicell.get_variable()
- physicell.get_vector()

- physicell.get_cell()
- physicell.get_microenv()

Python/C API

Challenge III: Run Consecutive Episodes

Learning is not possible if you have to destroy the runtime after each episode to reset to the initial condition because you will at the same time erase everything you learned from this episode.

Challenge III: Run Consecutive Episodes

The cell_defaults struct can not be deleted!

- ★ all cell_types and substrates later on used have to be defined in this loaded environment.

How to reset PhysiCell within a run time?

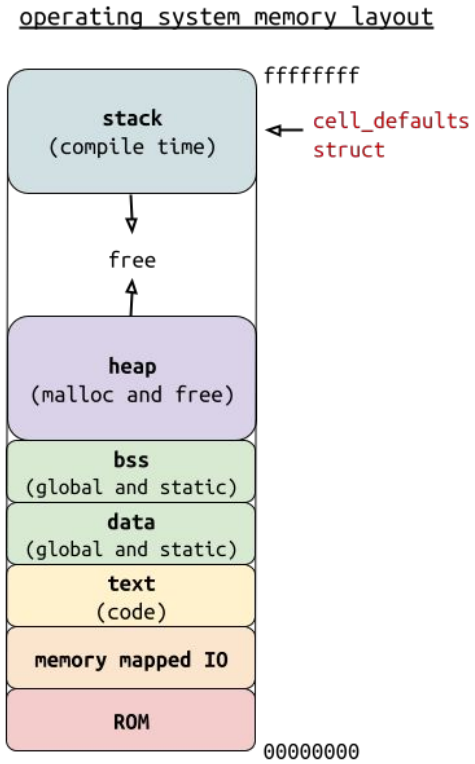
- ★ delete all cells.
- ★ reset global variables.
- ★ reset microenvironment.
- ★ setup tissue.

PhysiCell memory layout caused PhysiGym limitations:

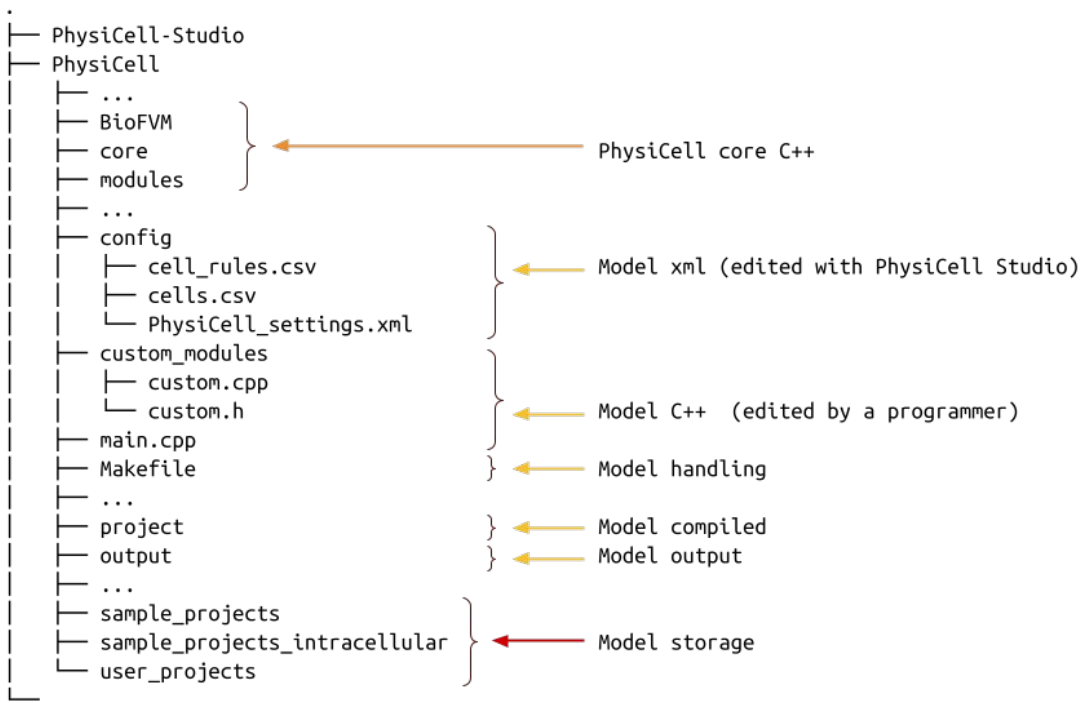
- ★ per runtime only one Gymnasium PhysiCellModel env instance can be loaded and run!

PhysiCell filesystem layout caused PhysiGym limitations:

- ★ Python has to be run at the PhysiCell root folder.



Filesystem: PhysiCell Model



Filesystem: PhysiCell Model

```
.
├── PhysiCell-Studio
├── PhysiCell
│   ├── ...
│   ├── BioFVM
│   ├── core
│   ├── modules
│   ├── ...
│   ├── config
│   │   ├── cell_rules.csv
│   │   ├── cells.csv
│   │   └── PhysiCell_settings.xml
│   ├── custom_modules
│   │   ├── custom.cpp
│   │   └── custom.h
│   ├── main.cpp
│   ├── Makefile
│   ├── ...
│   ├── project
│   ├── output
│   ├── ...
│   ├── sample_projects
│   ├── sample_projects_intracellular
│   └── user_projects
└── ...
```

PhysiCell core C++

Model xml (edited with PhysiCell Studio)

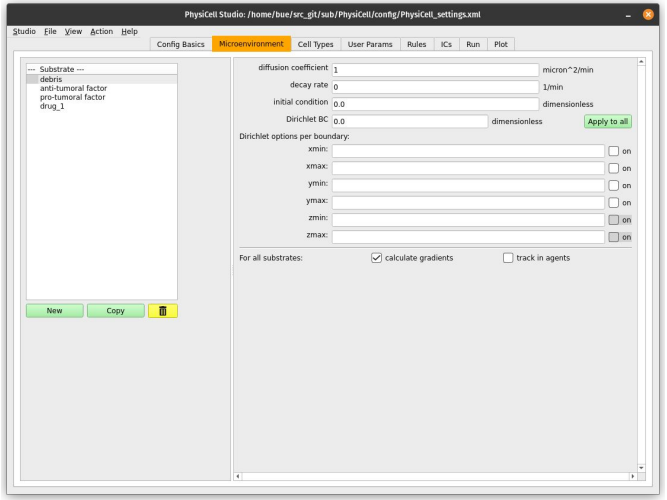
Model C++ (edited by a programmer)

Model handling

Model compiled

Model output

Model storage



Filesystem: PhysiCell Model custom_modules

```
.
├── PhysiCell-Studio
├── PhysiCell
│   ├── ...
│   ├── BioFVM
│   ├── core
│   ├── modules
│   ├── ...
│   ├── config
│   │   ├── cell_rules.csv
│   │   ├── cells.csv
│   │   └── PhysiCell_settings.xml
│   ├── custom_modules
│   │   ├── custom.cpp
│   │   └── custom.h
│   ├── main.cpp
│   ├── Makefile
│   ├── ...
│   ├── project
│   ├── output
│   ├── ...
│   ├── sample_projects
│   ├── sample_projects_intracellular
│   └── user_projects
└── ...
```

PhysiCell core C++

Model xml (edited with PhysiCell Studio)

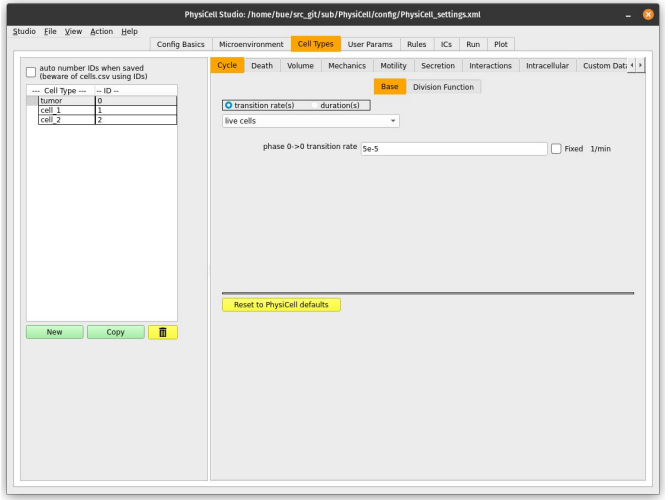
Model C++ (edited by a programmer)

Model handling

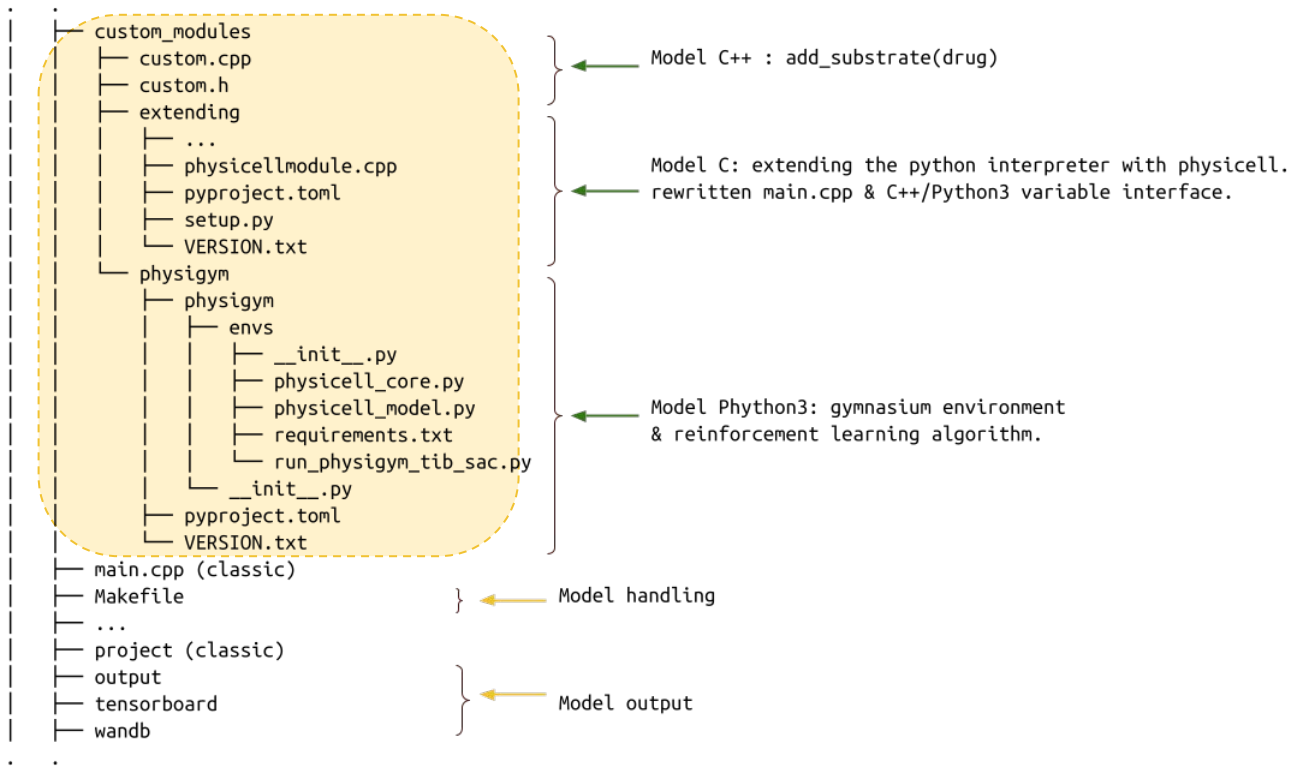
Model compiled

Model output

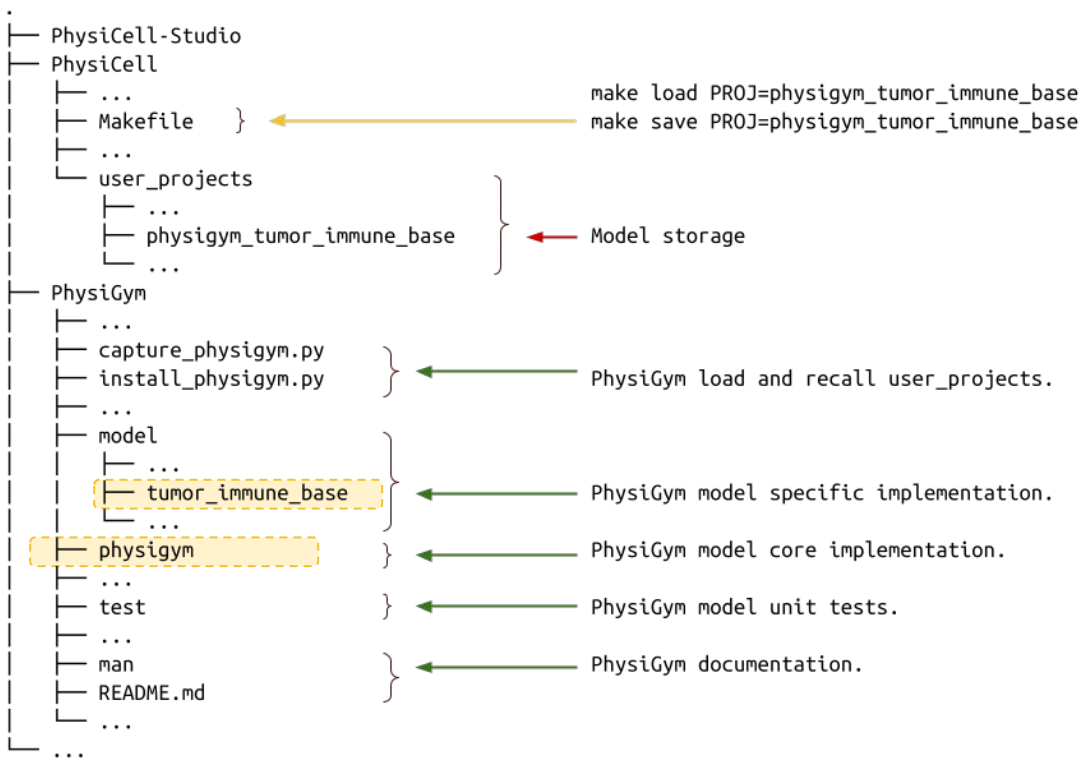
Model storage



Filesystem: PhysiGym



Source Code Repository: PhysiGym



Demonstration

- ★ tree PhysiGym: *REAME.md, man, model, physigym, test*
- ★ studio: *substrate, cell_type, rules, custom_data, user_parameters*
- ★ python3 install_physigym.py tibbue
- ★ make data-cleanup clean reset
- ★ make load PROJ=physigym_tibbue
- ★ make classic -j 8
- ★ ./project
- ★ make
- ★ vim custom_modules/physigym/physigym/envs/run_physigym_tibbue.py
- ★ python3 custom_modules/physigym/physigym/envs/run_physigym_tibbue.py
- ★ vim custom_modules/physigym/physigym/envs/run_physigym_tibbue_sac.py

PhysiGym related PhysiCell core changes

- ★ `custom_modules` deep folder structure (v1.14.0)
- ★ `episode sample_project` (v1.14.2)

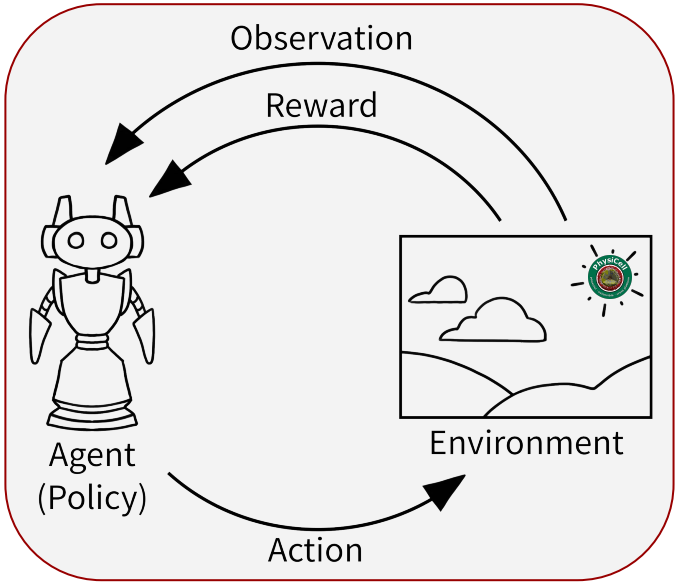
Take-home message: PhysiGym is lightweight!

- ★ PhysiGym models are PhysiCell user_projects.
- ★ The Python/PhysiCell interface is built on top of:
  - PhysiCell's user_parameters, custom_variables, and custom_vectors.
  - core Python's Python/C API.
- ★ Implementation in: C, C++, Python3, and PhysiCell Studio.
- ★ Compiles on: Windows Subsystem Linux, MacOSX, and Linux.
- ★ Runs from the command line.
- ★ Well documented.
- ★ PhysiGym can handle any PhysiCell model.

Reinforcement learning

RL: Characteristic

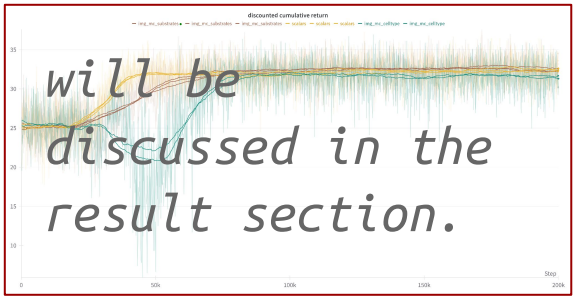
- ★ MDP = S,A,T,R
- ★ MDP: Markov Decision Process
- ★ S: state space (observation space)
- ★ s: state (observation)
- ★ A: action space
- ★ a: action
- ★ T: transition $s_{t+1} = T(s_t, a)$
- ★ R: reward is determined by $R \subseteq S_t \times A \times S_{t+1}$



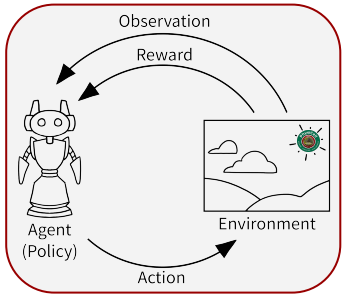
RL: Objectives

Maximize the expected discounted cumulative reward (return) over time.

$$\arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t \mid s_0 = s, \pi\right]$$



- ★ γ: discount factor (short versus long-them)
- ★ r_t: reward function
- ★ s_0: initial state (observation)
- ★ s: a state from the state space.
- ★ π: policy

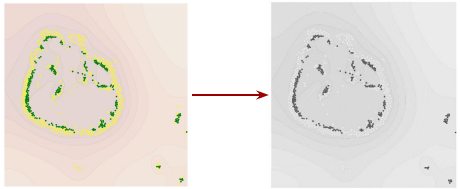


RL: Observation Space

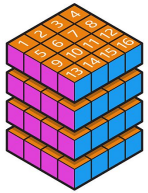
Parameter

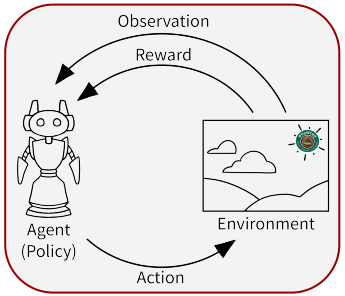
★ scalar:
- cell count from cell_types. [0, ..]
- possibly max, mean, std from substrates. [0.0, ..]

★ image: RGB to grayscale.



★ image: multichannel.
- one channel per cell_type.
- one channel per substrate.





RL: Reward Function

equation

- ★ r: reward at time t [-1..1]
- ★ α: balance factor = 0.5
- ★ C: tumor cell count
- ★ D: drug dose [0..1]
- ★ t: time step

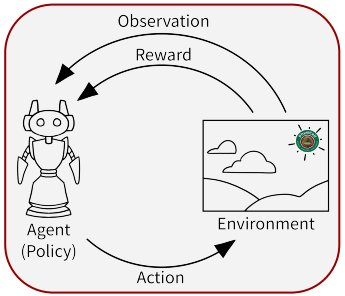
$$r_t = \alpha \cdot \frac{C_{t-1} - C_t}{C_{t-1} \cdot e^{\mu \cdot dt} - C_{t-1}} + (1-\alpha) \cdot (-D_{t-1})$$

normalized by exponential growth

$$C_t = C_{t-1} \cdot e^{\mu \cdot dt}$$

- ★ e: Euler number
- ★ μ: growth rate
- ★ dt: delta time

the tumor burden term is clipped by [-1..1]



RL: Reward Function

equation

- ★ r: reward at time t [-1..1]
- ★ α: balance factor = 0.5
- ★ C: tumor cell count
- ★ D: drug dose [0..1]
- ★ t: time step

alpha

drug burden

$$r_t = \alpha \cdot \frac{C_{t-1} - C_t}{C_{t-1} \cdot e^{\mu \cdot dt} - C_{t-1}} + (1-\alpha) \cdot (-D_{t-1})$$

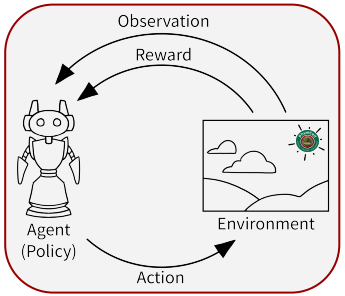
tumor burden

normalized by exponential growth

$$C_t = C_{t-1} \cdot e^{\mu \cdot dt}$$

- ★ e: Euler number
- ★ μ: growth rate
- ★ dt: delta time

the tumor burden term is clipped by [-1..1]



RL: Reward Function

equation

- ★ r: reward at time t [-1..1]
- ★ α: balance factor = 0.5
- ★ C: tumor cell count
- ★ D: drug dose [0..1]
- ★ t: time step

$$r_t = \alpha \cdot \frac{C_{t-1} - C_t}{C_{t-1} \cdot e^{\mu \cdot dt} - C_{t-1}} + (1-\alpha) \cdot (-D_{t-1})$$

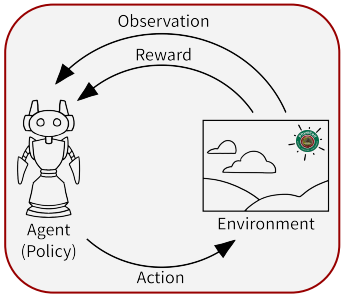
tumor burden

normalized by exponential growth

$$C_t = C_{t-1} \cdot e^{\mu \cdot dt}$$

- ★ e: Euler number
- ★ μ: growth rate
- ★ dt: delta time

terms are clipped by [-1..1]



RL: Reward Function

equation

- ★ r: reward at time t [-1..1]
- ★ α: balance factor = 0.5
- ★ C: tumor cell count
- ★ D: drug dose [0..1]
- ★ t: time step

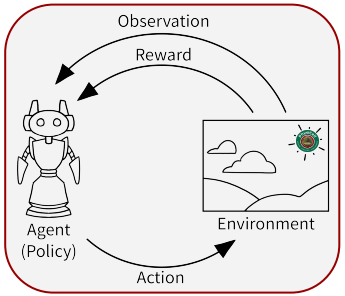
$$r_t = \alpha \cdot \frac{C_{t-1} - C_t}{C_{t-1} \cdot e^{\mu \cdot dt} - C_{t-1}} + (1-\alpha) \cdot (-D_{t-1})$$

normalized by exponential growth

$$C_t = C_{t-1} \cdot e^{\mu \cdot dt}$$

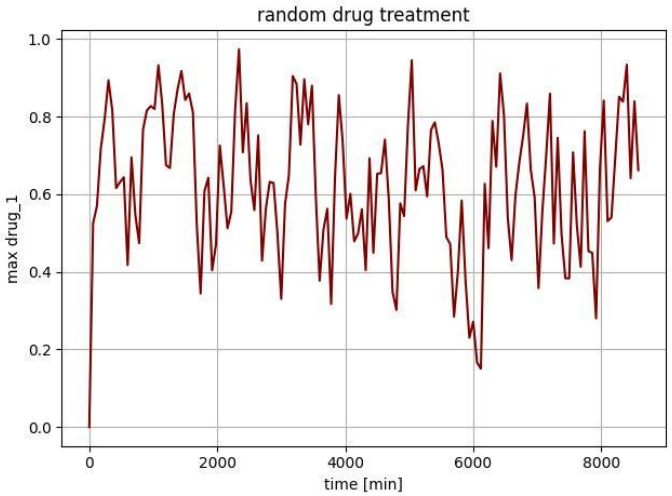
- ★ e: Euler number
- ★ μ: growth rate
- ★ dt: delta time

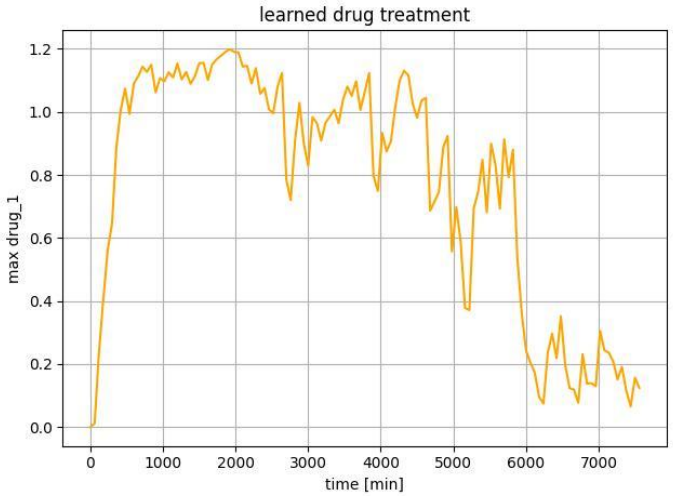
terms are clipped by [-1..1]

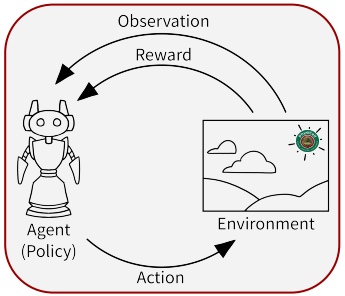


RL: Action Space

★ D: drug_1 [0..1]

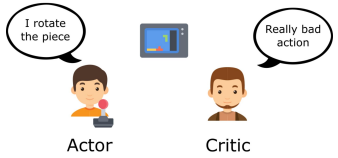






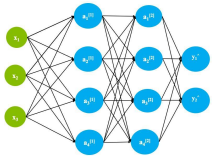
Reinforcement deep learning algorithm

★ algorithm: SAC (Soft Actor Critic)
- soft: continuous action space.
- actor: function (neural network) to control the policy.
- critic: function (neural network) to control the actor.
- entropy based: explore new states versus exploit known states.
- objective: maximize expected cumulative reward & entropy.

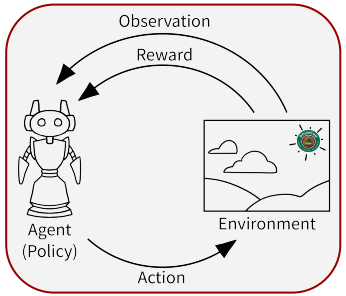


$$H = -\sum_{s=0}^{n} p_s \log_2 p_s$$

★ policy to learn:
- scalars: multilayer perceptrons neural network (pytorch).
- image: convolutional neural network (pytorch).



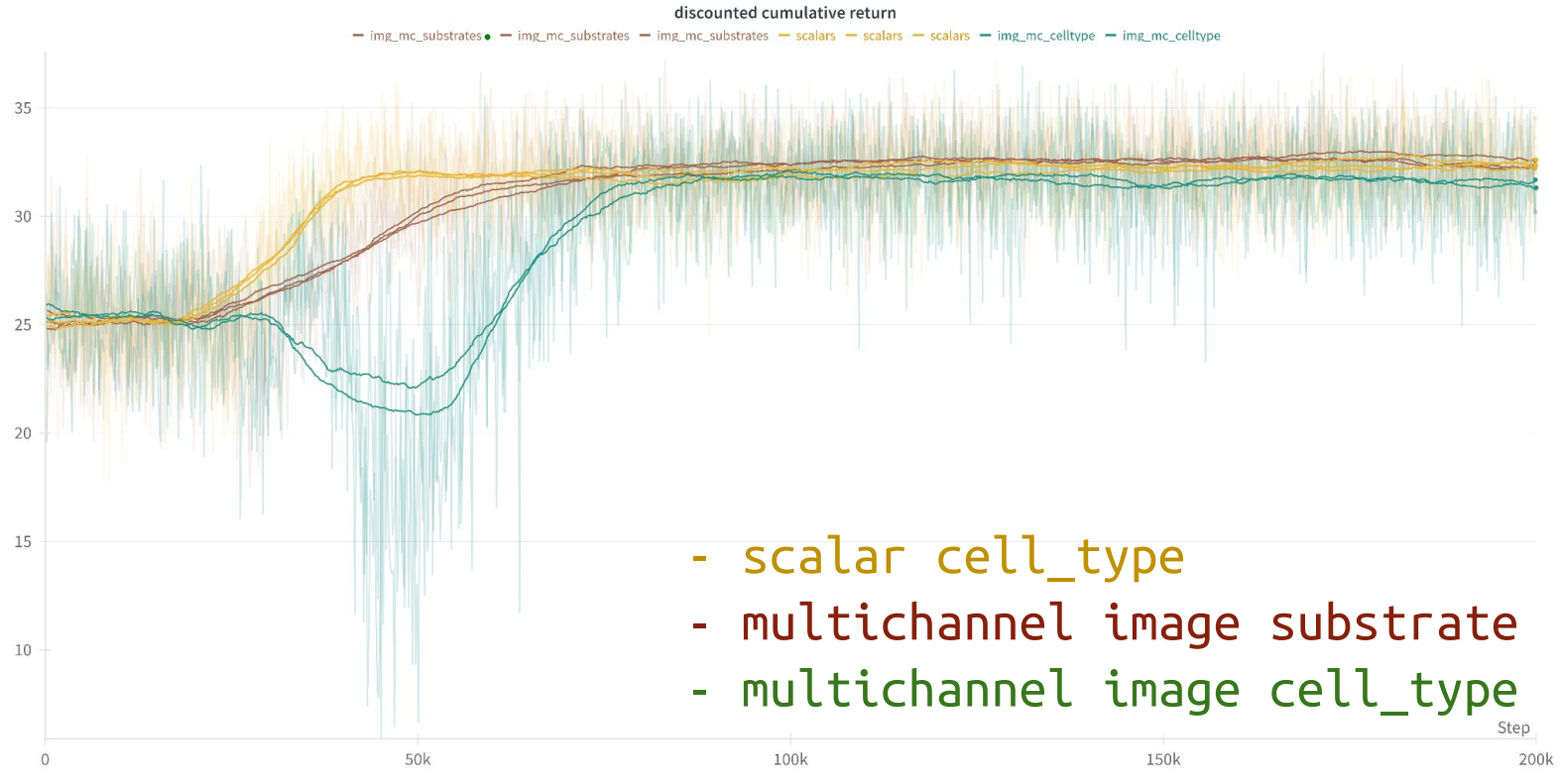
★ reply buffer (memory):
- record:
  - observation, reward, termination, truncation, action, next observation.
- remember the last 8192 runs.
- random sample 256 runs from the buffer.



LUDDY
SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING
Actor critic figure adapted from: https://huggingface.co/learn/deep-rl-course/unit6/advantage-actor-critic
Deep neural network figure adapted from: https://developer.ibm.com/learningpaths/get-started-with-deep-learning/an-introduction-to-deep-learning/
Macklin Lab
@MathCancer.bsky.social
MathCancer.org

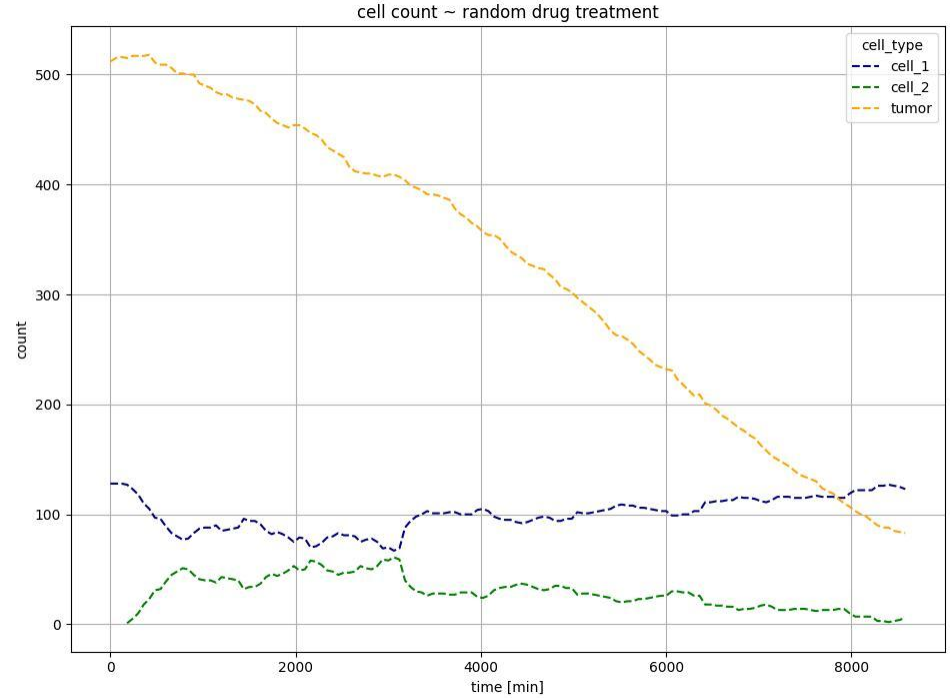
Results

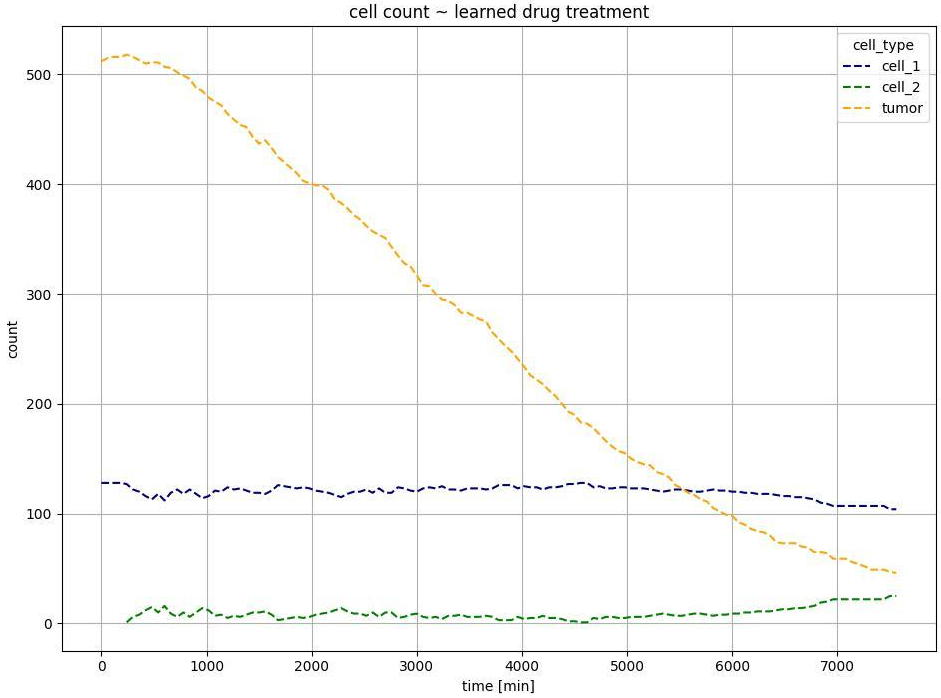
Discounted cumulative return



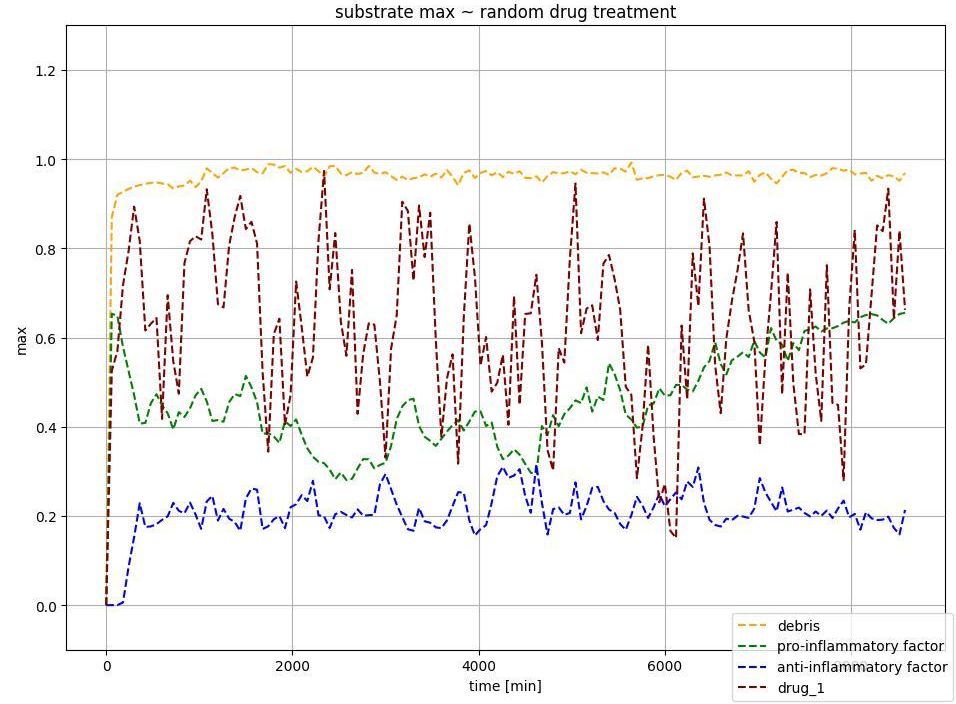
- scalar cell_type
- multichannel image substrate
- multichannel image cell_type

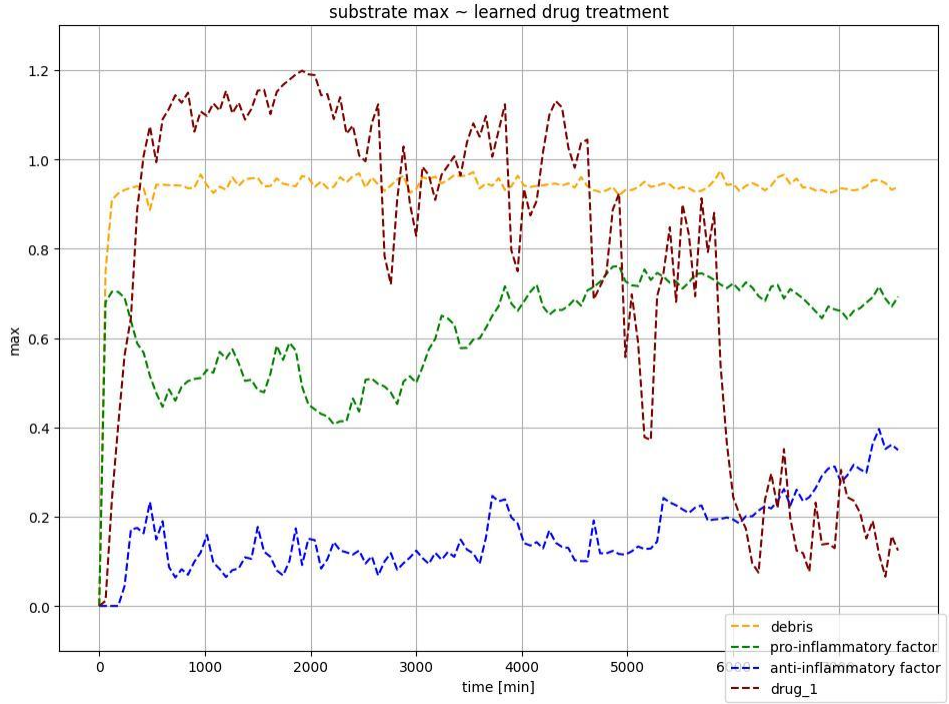
Observation: cell_type



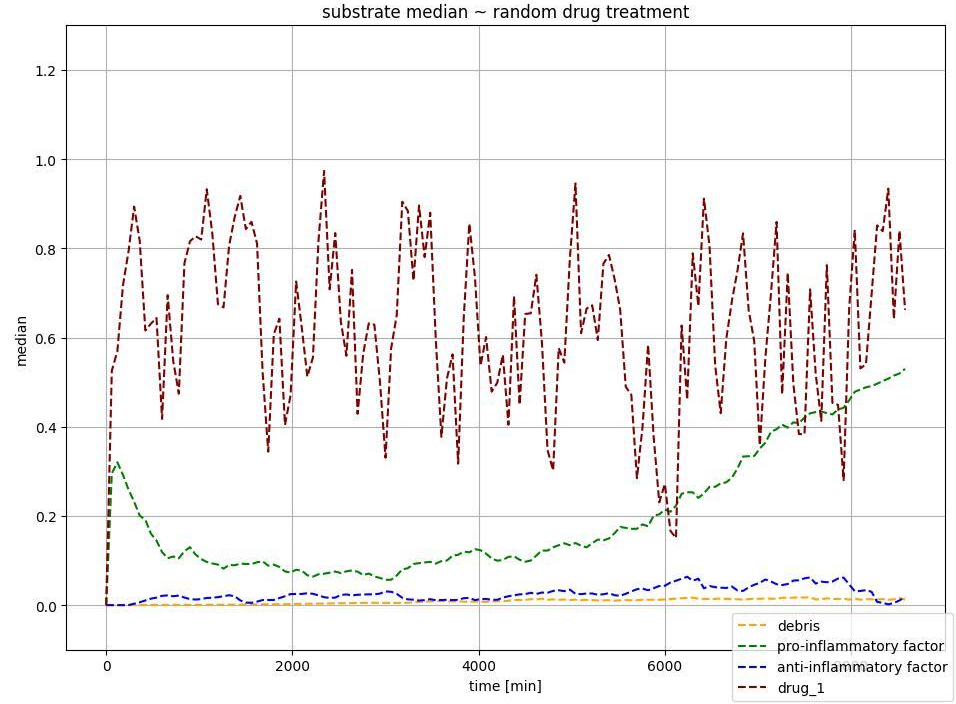


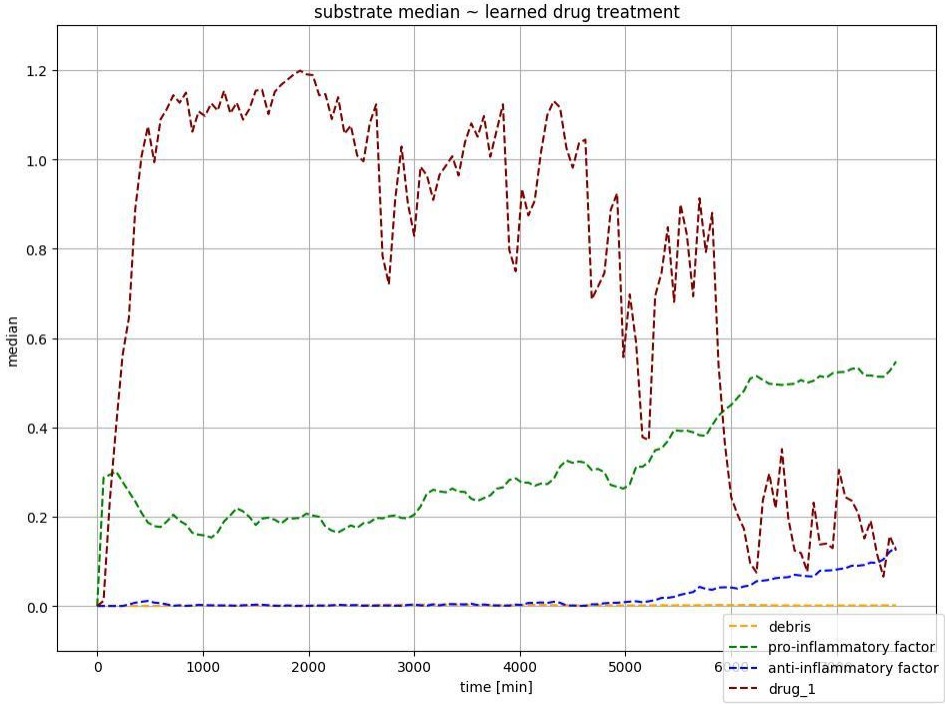
Observation and action: substrate maximum



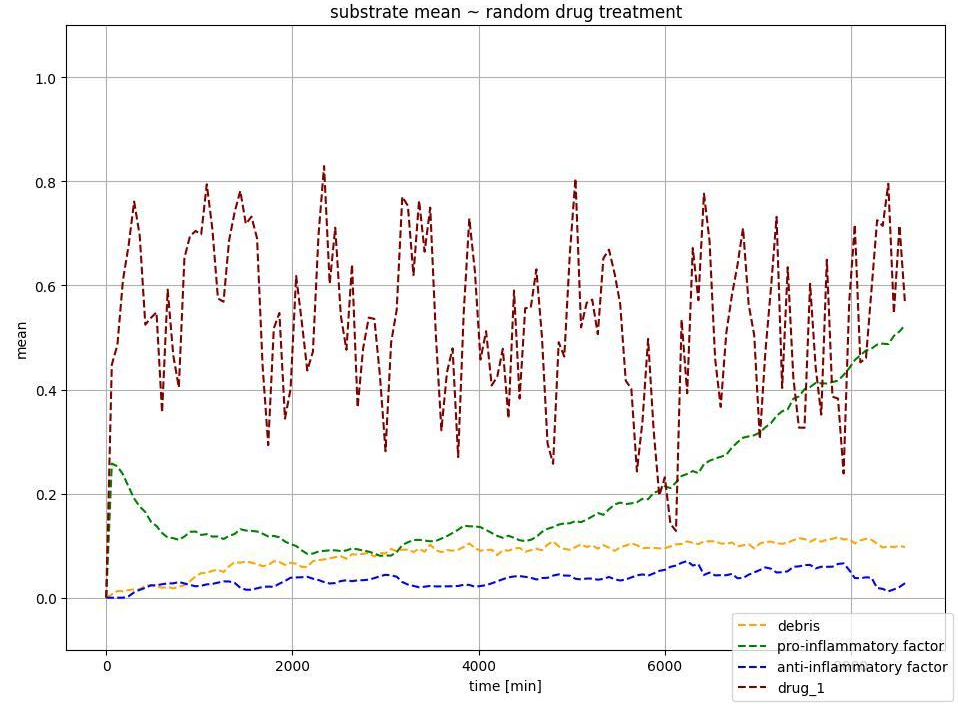


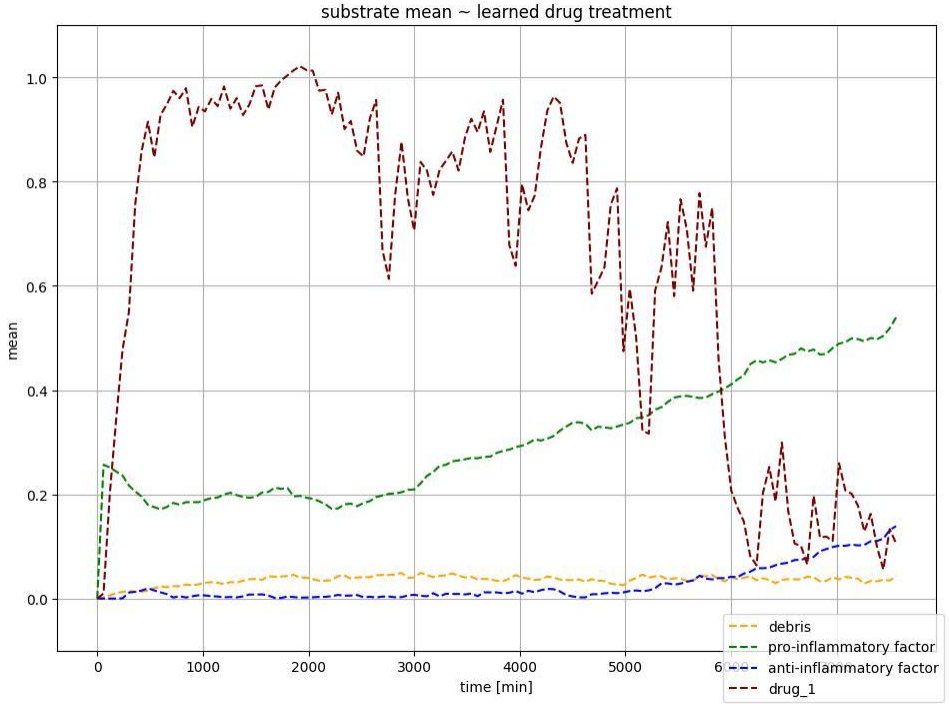
Observation and action: substrate median



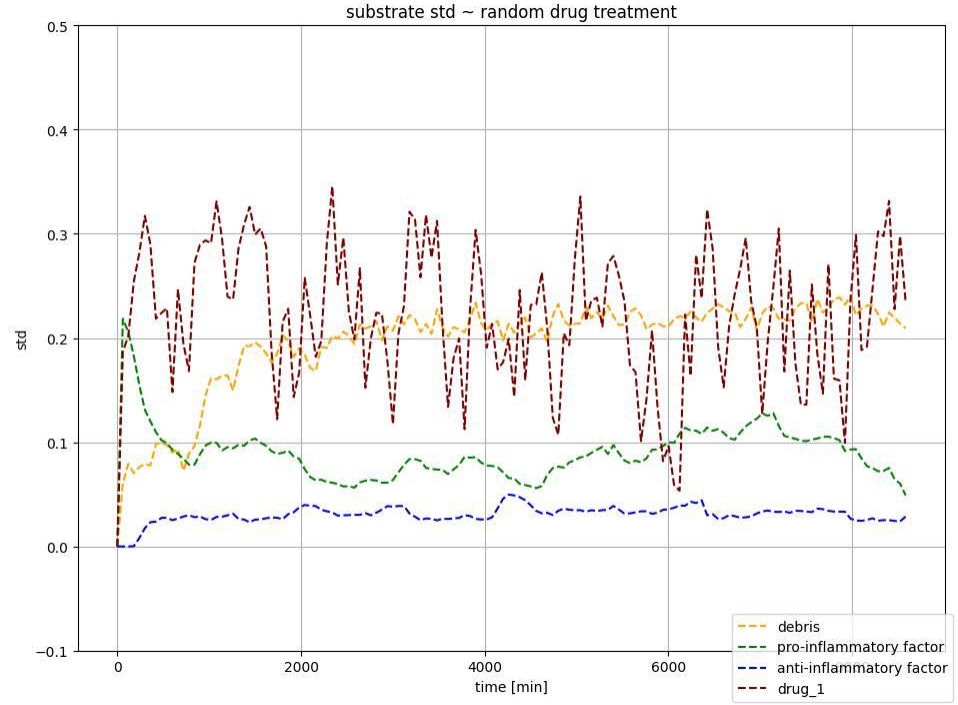


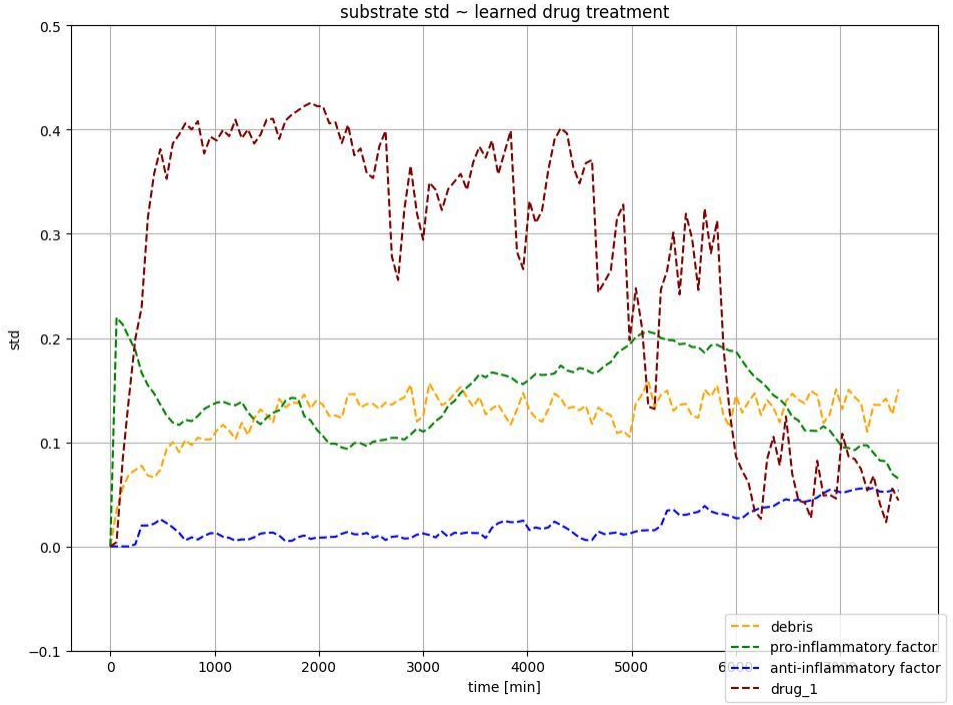
Observation and action: substrate mean





Observation and action: substrate standard deviation





Discussion

Related Work

Reinforcement learning for optimal scheduling of Glioblastoma treatment with Temozolomide

Amir Ebrahimi Zade[a], Seyedhamidreza Shahabi Haghighi[a,*], Madjid Soltani[b,c,d,e]

[a] Faculty of Industrial Engineering and Systems Management, Amirkabir University of Technology, Tehran, Iran
[b] Faculty of Mechanical Engineering, K.N. Toosi University of Technology, Tehran 1969764499, Iran
[c] Advanced Bioengineering Initiative Center, Computational Medicine Center, K. N. Toosi University of Technology, Tehran, Iran
[d] Centre for Biotechnology and Bioengineering (CBB), University of Waterloo, Waterloo, ON, Canada
[e] Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada

ARTICLE INFO

Article history:
Received 18 September 2019
Revised 17 February 2020
Accepted 8 March 2020

ABSTRACT

Background: : Glioblastoma multiforme (GBM) is the most frequent primary brain tumor in adults and Temozolomide (TMZ) is an effective chemotherapeutic agent for its treatment. In Silico models of GBM growth provide an appropriate foundation for analysis and comparison of different regimens. We propose a mathematical frame for patient specific design of optimal chemotherapy regimens for GBM patients.

Related Work

GRAPEVINE: A Reinforcement Failing Framework for AI-Guided Discovery of Hidden Spatial Dynamics in Adaptive Tumor Therapy

Serhii Aif[1,2], Maximillian Eiche[1,2], Nico Appold[1,2], Elias Fischer[1,2], Timon Citak[1,2], and Jona Kayser[1,2,*]

[1]Max-Planck-Zentrum für Physik und Medizin & Max Planck Institute for the Science of Light, 91058 Erlangen, Germany
[2]Department of Physics, Friedrich-Alexander-University Erlangen-Nürnberg, 91054 Erlangen, Germany
[*]E-mail: jona.kayser@mpzpm.mpg.de

March 31, 2025

Future steps

Alex:

- ★ Write PhysiCell models that are meaningful in praxis.
- ★ Write reinforcement learning code that is meaningful in practice.
- ★ Vectorize the environment.
- ★ Explore PhysiGym.

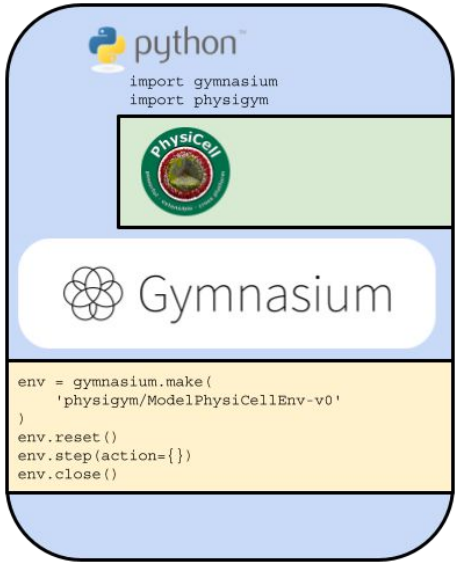
Elmar:

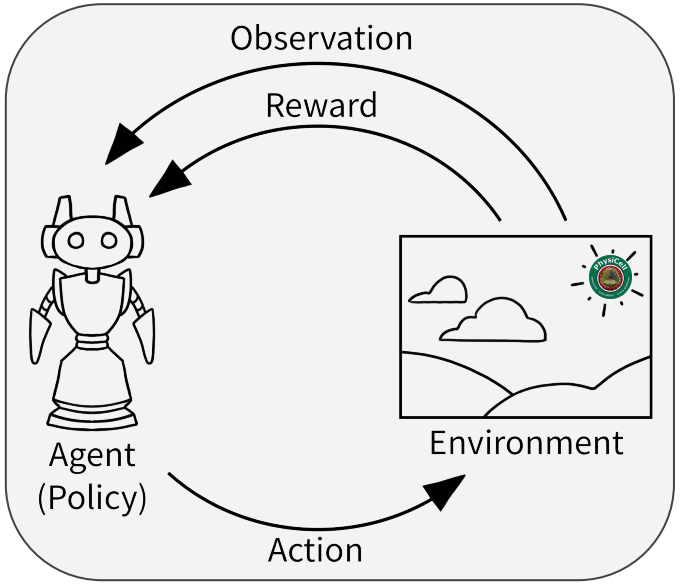
- ★ Explore the physicell embedding library (Python/C++ interface).
- ★ Focus on my PhD work.

Thank you for your attention!

I am happy to take your questions.

PhysiGym resources:





- https://physicell.org/
- https://gymnasium.farama.org/
- https://github.com/Dante-Berth/PhysiGym