Jell Kkaggi Development

Soft-body physics and RL model implementation for pseudo 3D games

SNU ECE 23
Sanghwa Lee | Soohyun Choi

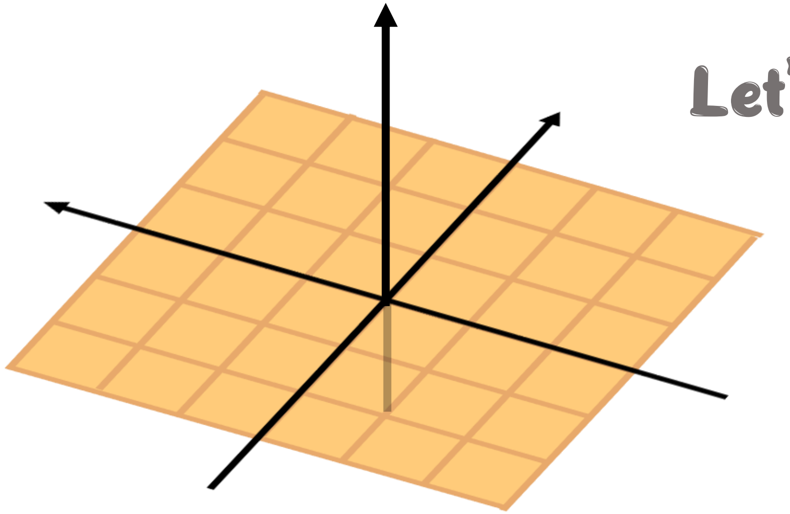
Overview

1 | ***Pseudo 3D Implementation***
Perspective of the player

2 | ***Soft-body Simulation***
Jelly & collision model

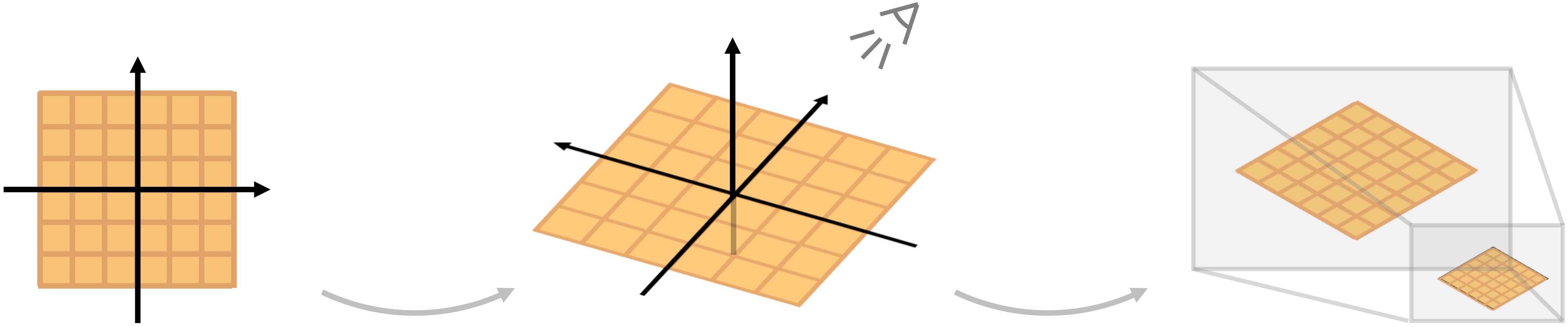
3 | ***Reinforcement Learning***
Training AI player

1 Pseudo 3D Implementation



Let's make 3D Alkkagi !

Pseudo 3D



2D to 3D

3D to 2D

2D plane

$\mathcal{B}_{xy} = \{\hat{e_1}, \hat{e_2}\}$

3D gameworld

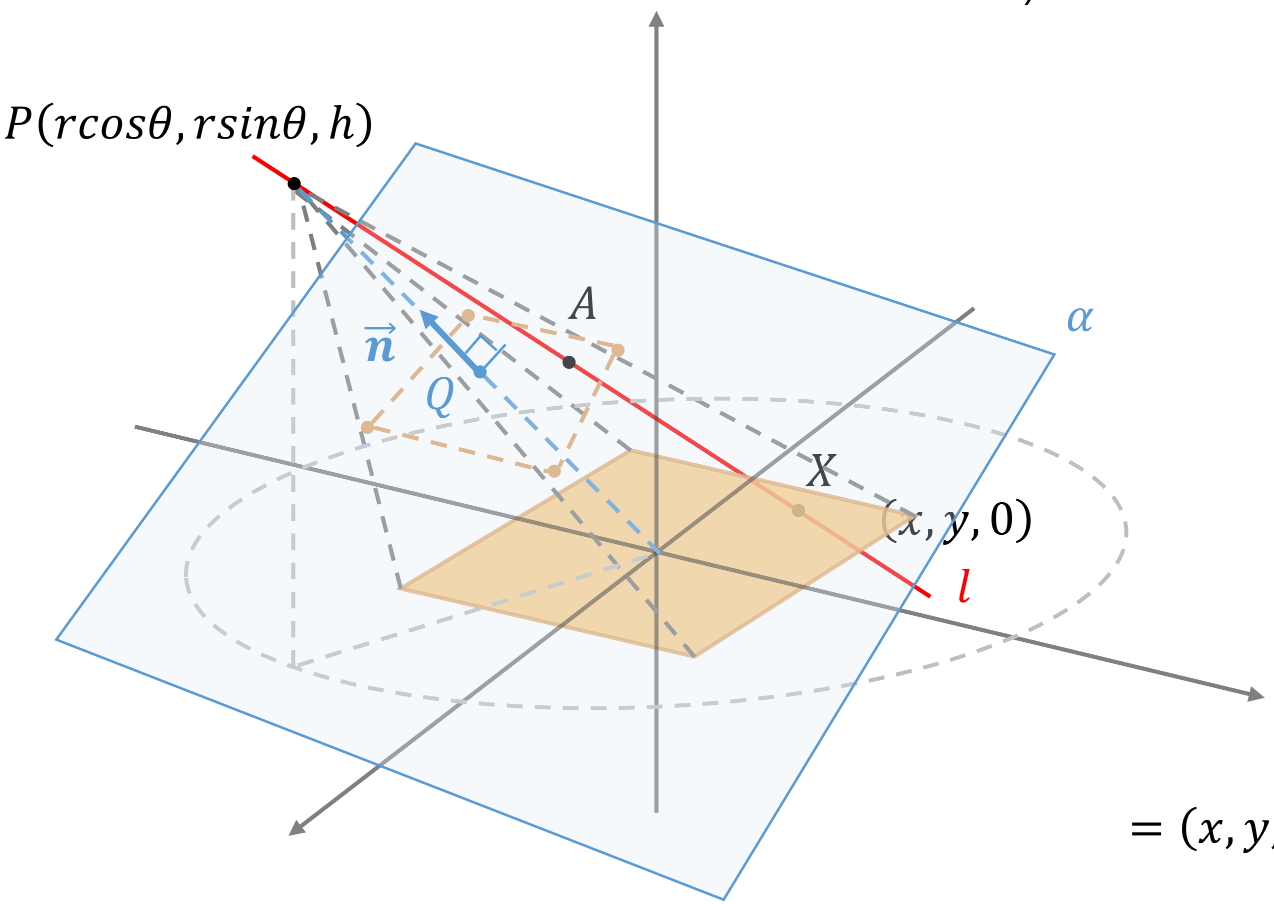
$\mathcal{B}_{xyz} = \{\hat{e_1}, \hat{e_2}, \hat{e_3}\}$

2D screen

$\mathcal{B}_{\alpha} = \{\vec{u}, \vec{v}\}$

Perspective Projection

1) 2D to 3D



$$Q\left(\frac{rcos\theta}{2}, \frac{rsin\theta}{2}, \frac{h}{2}\right) \qquad \vec{\boldsymbol{n}} = (rcos\theta, rsin\theta, h)$$

$$\alpha \ : \ rcos\theta + rsin\theta y + hz - \frac{h^2 + z^2}{2} = 0$$

$$l \ : (x, y, 0) + t(rcos\theta - x, rsin\theta - y, h) = 0$$

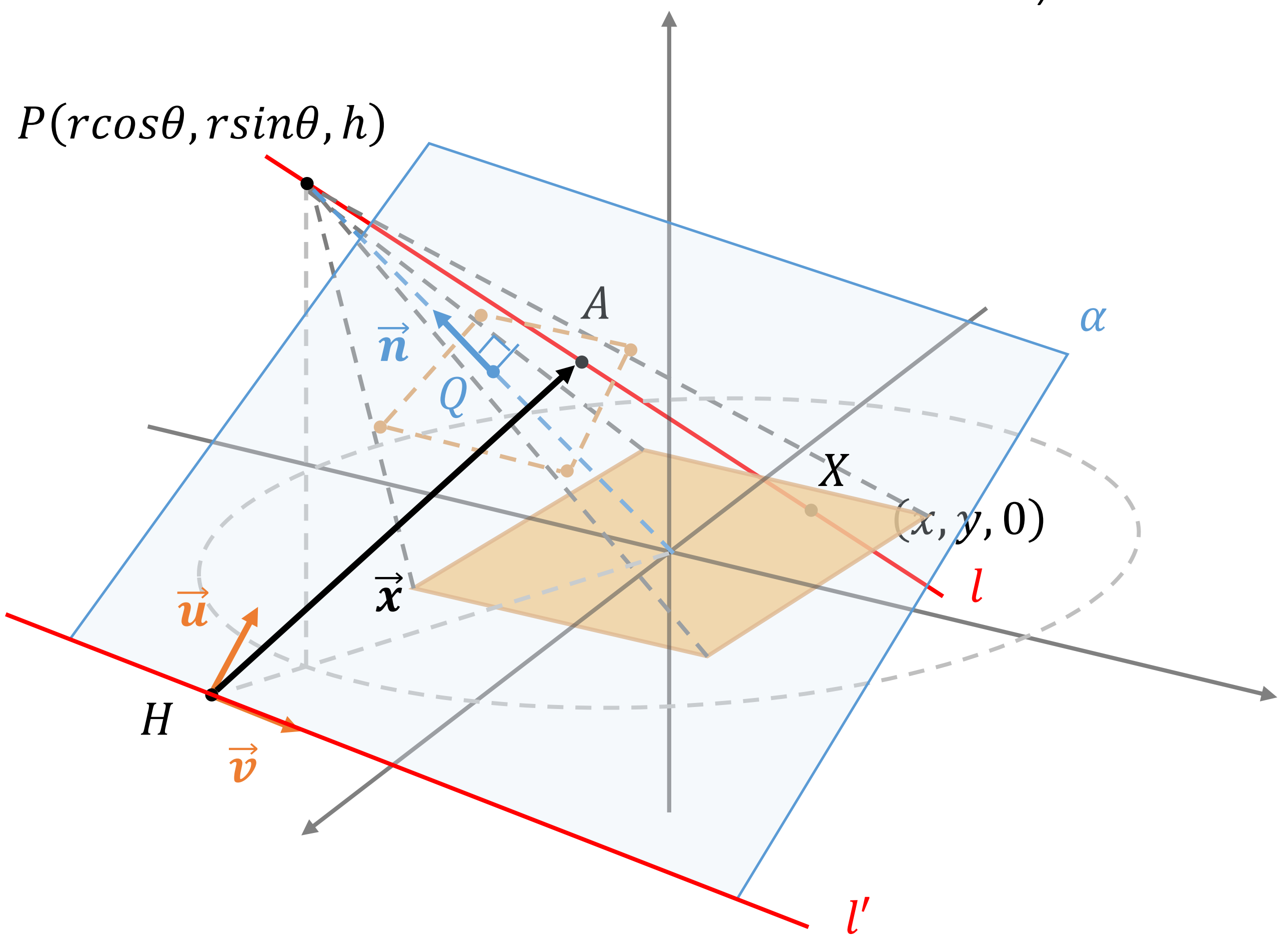
$$A : \ \alpha \cap l$$

$$A = proj_{\alpha}(x, y, 0)$$

$$= (x, y, 0) + \frac{\frac{r^2 + h^2}{2} - xrcos\theta - yrsin\theta}{r^2 - xrcos\theta - yrsin\theta + h^2}(rcos\theta - x, rsin\theta - y, h)$$

Perspective Projection

2) 3D to 2D



$$\mathcal{B}_\alpha = \{\vec{u}, \vec{v}\}$$

$$\vec{u} = \overrightarrow{QP} \qquad \vec{x} = \overrightarrow{PA} \qquad \vec{v} = \frac{d\vec{r}}{dt} = (-sin\theta, cos\theta, 0)$$

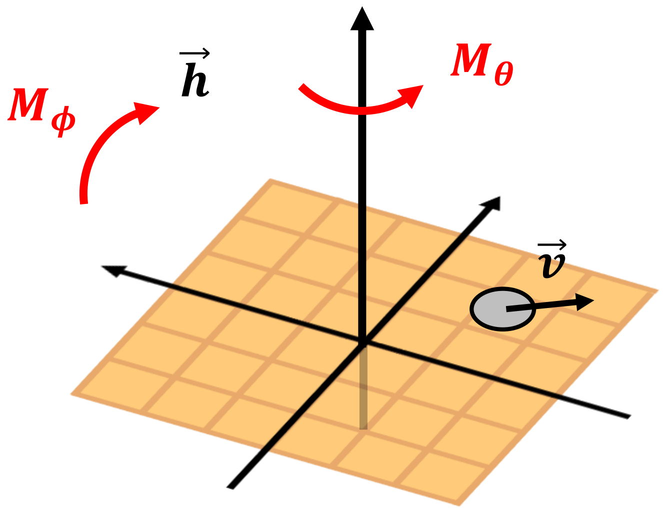
$$l' : \alpha \cap \mathbb{R}^2 \qquad xrcos\theta + yrsin\theta - \frac{h^2 + r^2}{2} = 0$$

$$H : l' \cap dir\,\vec{r} \qquad \left(\frac{r^2 + h^2}{4r} cos\theta, \frac{r^2 + h^2}{4r} sin\theta, 0\right)$$

$$conv\,\vec{x} = [T]_{\mathcal{B}_\alpha}^{\mathcal{B}_{xyz}}$$

$$= \left(\frac{\vec{u} \cdot \vec{x}}{\vec{u} \cdot \vec{u}} \vec{u}, \frac{\vec{v} \cdot \vec{x}}{\vec{v} \cdot \vec{v}} \vec{v}\right)$$

Perspective Rotation



$$M_\theta = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad M_\phi = \begin{pmatrix} 1 & tan\phi \\ 0 & 1 \end{pmatrix}$$

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} \qquad \vec{h} = \begin{pmatrix} h \\ 1 \end{pmatrix}$$

$$M_\theta \vec{v} = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix} = \begin{pmatrix} v_x cos\theta - v_y sin\theta \\ v_x sin\theta + v_y \cos\theta \\ 1 \end{pmatrix}$$

$$M_\phi \vec{h} = \begin{pmatrix} 1 & tan\phi \\ 0 & 1 \end{pmatrix} \begin{pmatrix} h \\ 1 \end{pmatrix} = \begin{pmatrix} h + tan\phi \\ 1 \end{pmatrix}$$

Implementation

```python
def conv3D(x,y):
    h_ = h[0]
    r_ = r[0]
    theta_ = theta[0]
```

→ *Get from global variable settings*

```python
    shift_y = (r_**2 + h_**2) / r_ * 0.25
```

→ *Shift origin (visual)*

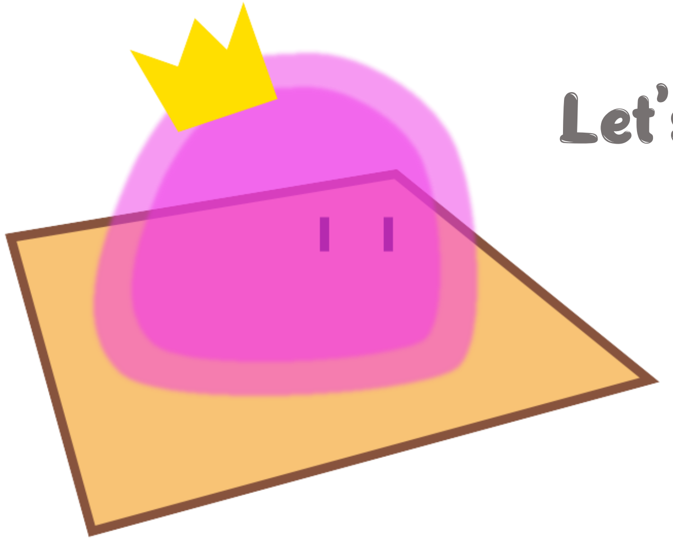
```python
    C = 1 - (h_**2/2 + r_**2/2) / (r_**2 + h_**2 - x*r_*cos(theta_) - y*r_*sin(theta_))
    proj_x = x + C*(r_*cos(theta_) - x)
    proj_y = y + C*(r_*sin(theta_) - y)
    proj_z = C*h_

    P_x = (r_**2 + h_**2) / (4*r_) * cos(theta_)
    P_y = (r_**2 + h_**2) / (4*r_) * sin(theta_)
    P_z = 0

    u = [r_*cos(theta_)/2-P_x, r_*sin(theta_)/2-P_y, h_/2]
    v = [-sin(theta_), cos(theta_), 0]
    PA = [proj_x - P_x, proj_y - P_y, proj_z - P_z]

    conv_x = np.dot(PA,v)
    conv_y = np.dot(PA,u) / sqrt(np.dot(u,u))
```

→ *Calculation*

```python
    return (15*conv_x + WIDTH//2, HEIGHT//2 - 15*(conv_y - shift_y))
```

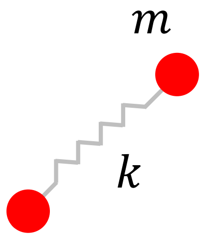
→ *Screen adjustment*

2 Soft-body Physics



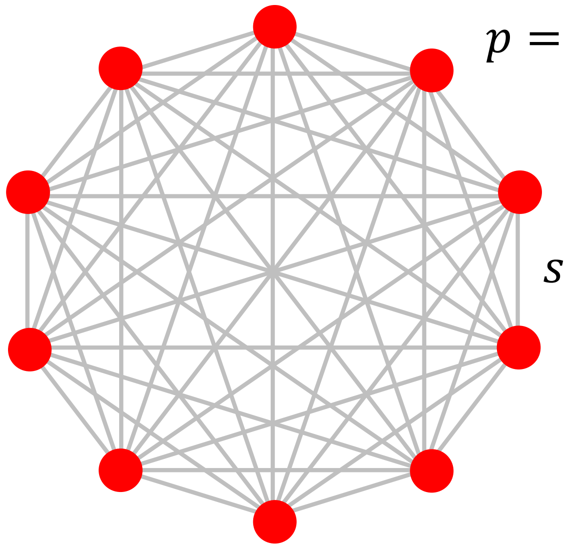
Let's make Jelly Alkkagi !

Soft-body Physics



$$\frac{d^2x}{dt^2} + b\frac{dx}{dt} + kx = 0$$

→ *Able to find analytic solution*

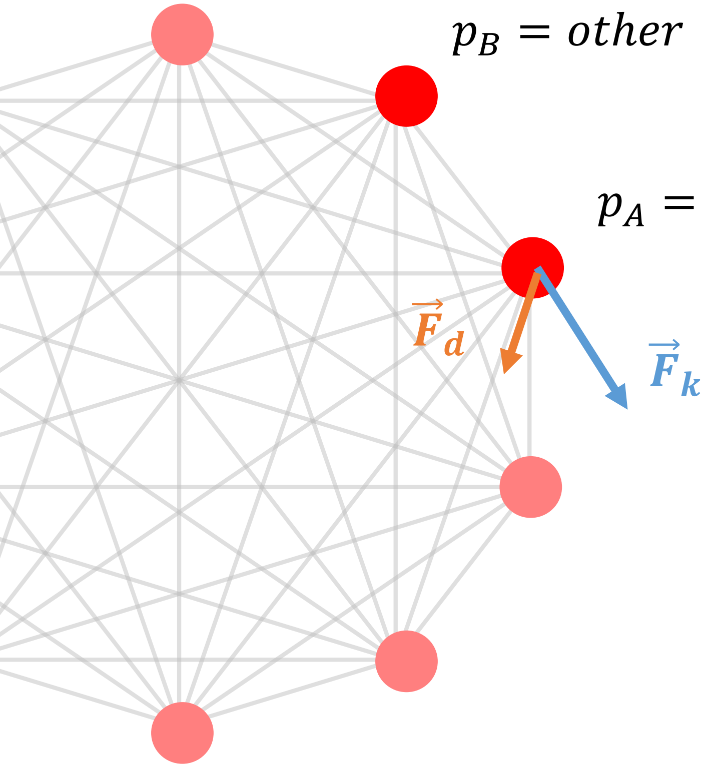


$$p = (x, y, v_x, v_y, m, F_x, F_y)$$

$$s = (p_A, p_B, L, k, b)$$

→ *N-body problem:*
Numerical method needed

Euler Integration



Pseudocode `stone.py/class Stone/update()`

$$self.connected = \{ (p_B, L), (p_C, L'), \cdots \}$$

$$for\ (other, L)\ in\ self.connected:$$

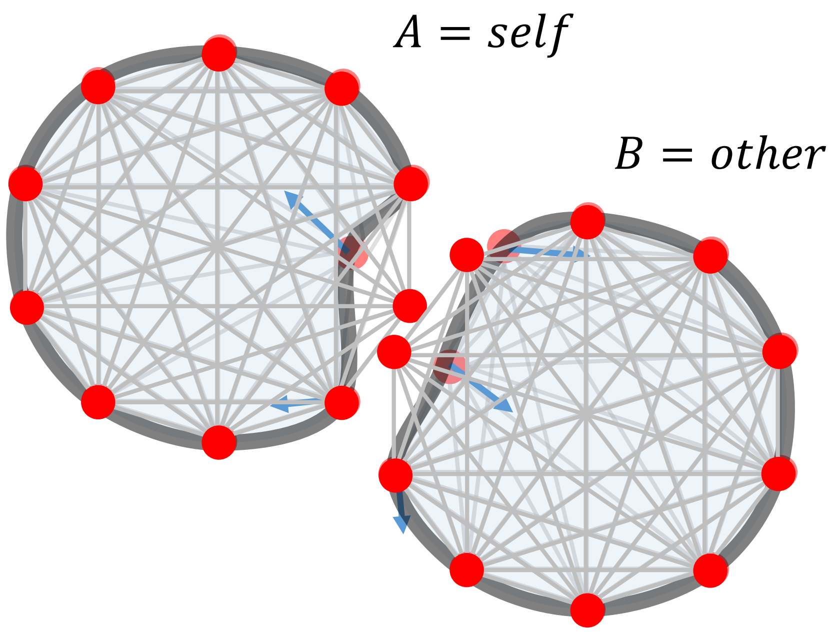
$$r = dist(self, other)$$

$$F = k(r - L)$$

$$\vec{\boldsymbol{F}} = F \cdot \left(\vec{\boldsymbol{x}}_{\boldsymbol{other}} - \vec{\boldsymbol{x}}_{\boldsymbol{self}}\right) - d \cdot \left(\vec{\boldsymbol{v}}_{\boldsymbol{other}} - \vec{\boldsymbol{v}}_{\boldsymbol{self}}\right)$$

$$\left.\begin{aligned} \vec{\boldsymbol{v}}_{\boldsymbol{self}} &= \frac{dt}{m} \cdot \vec{\boldsymbol{F}} \\ \vec{\boldsymbol{x}}_{\boldsymbol{self}} &= dt \cdot \vec{\boldsymbol{v}} \end{aligned}\right] \rightarrow \text{Euler Integration}$$

Collision Model



$$\boldsymbol{B}(t) = (1-t)\boldsymbol{B}_{\boldsymbol{P}_0\boldsymbol{P}_1\cdots\boldsymbol{P}_{n-1}}(t) + t\boldsymbol{B}_{\boldsymbol{P}_0\boldsymbol{P}_1\cdots\boldsymbol{P}_n}(t)$$

Pseudocode `stone.py/class Stone`

$def\ detectCollision(\,)$

$\quad stonesCollided = [\,]$

$\quad if\ dist(cen_A, cen_B) < r_A + r_B :\ \ add$

$def\ update(\,)$

$\quad for\ particles\ in\ self\ :$

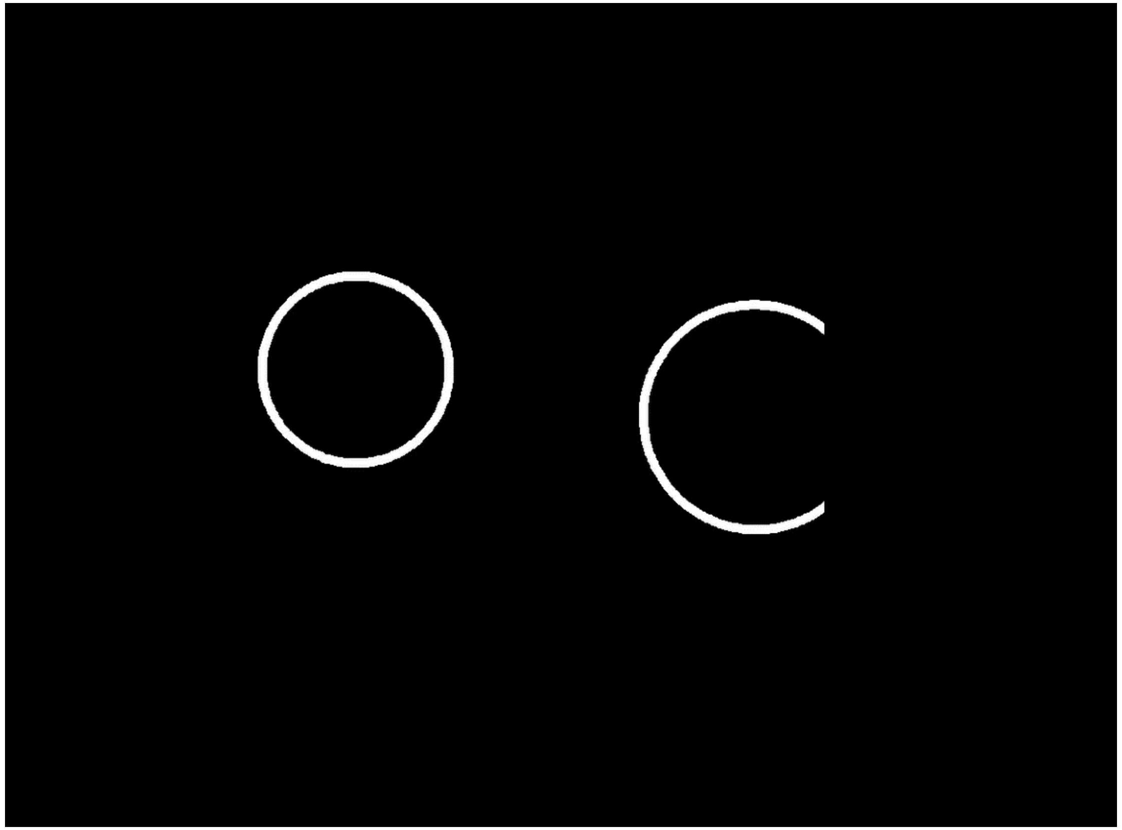
$\quad\quad if\ collided :\ \ change\ velocity$

$def\ draw(\,)$

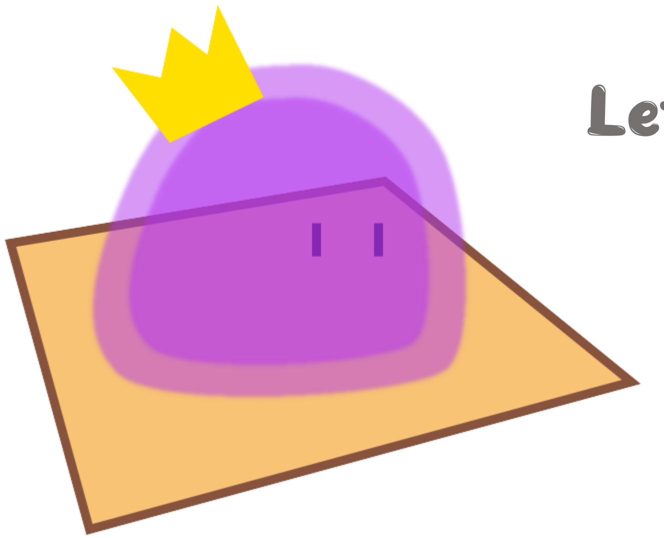
$\quad interpolate(particles)$ ⟶ *Bezier curve*

$\quad drawBorder(\,)$

Implementation

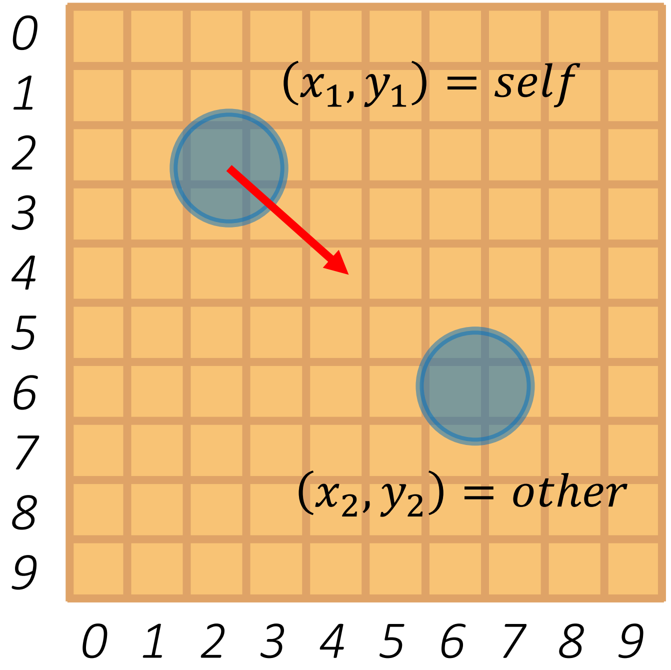


3 Reinforcement Learning



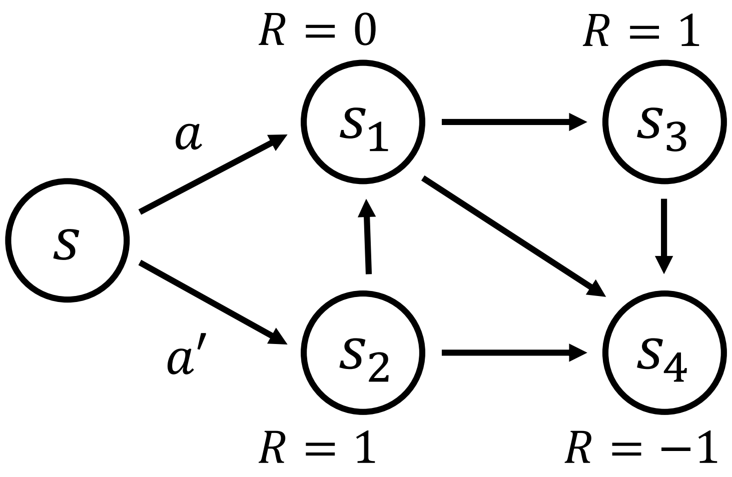
Let's make AI !

Training Model



State	$\mathcal{S} = \{0, \cdots, 9\}^2 \times \{0, \cdots, 9\}^2$	$\longrightarrow$ *Position*
Action	$\mathcal{A} = \{1,2,3,4,5\}$	$\longrightarrow$ *Shooting strength*
Probability	$\mathcal{P}(s, a) =$ *State after action a was done*	
Reward	$R(s,a) = \begin{cases} 1 & win \\ -1 & loose \\ 0 & tie \end{cases}$	$\longrightarrow$ *Other out of board* $\longrightarrow$ *Self out of board* $\longrightarrow$ *Otherwise*
Discount factor	$\gamma = 0.9$	

Q-Learning



Markov Decision Process

MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$

$\longrightarrow$ *Under policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$

State-value function

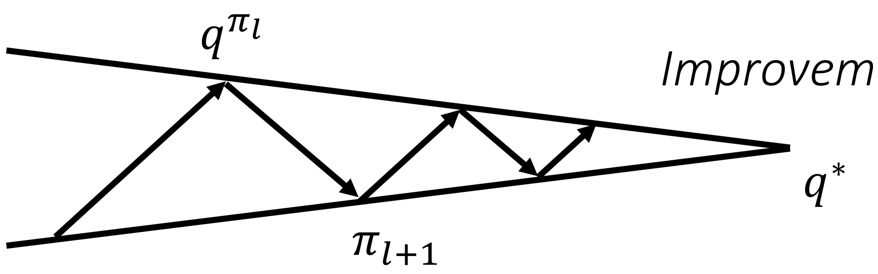
$$v^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R^{\pi}\left(N_k^{\pi}(s)\right)\right]$$

Action-value function

$$q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \left(\mathcal{P}(s,a,s') \cdot v^{\pi}(s')\right)$$

$\Big\downarrow$ *Bellman optimality theorem*

$$q^{*}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') \cdot \left(\max_{a' \in \mathcal{A}} q^{*}(s',a')\right)$$



Improvement every step

AI Model

Pseudocode `MDP.py/Q_learning()`

$q(s,a) = random(s \in \mathcal{S}, a \in \mathcal{A})$ $\longrightarrow$ *Start by random position*

$for\ each\ episode:$

$\quad s = initial\ state$

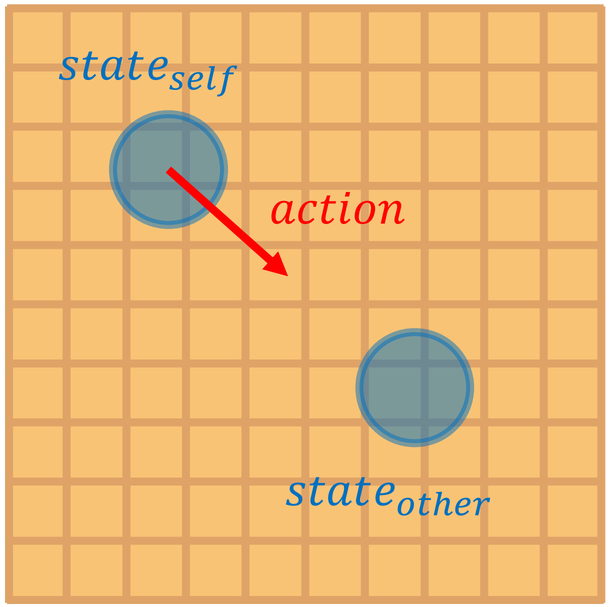
$\quad for\ each\ step:$

$\quad\quad a = \pi(s)\ in\ greedy$ $\longrightarrow$ *Predict the optimal action*

$\quad\quad r, s' = takeAction(a)$

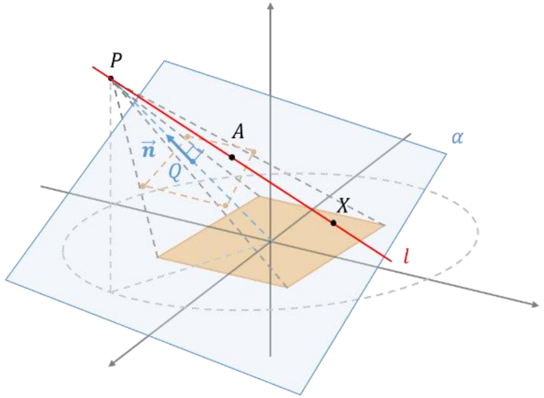
$\quad\quad q(s,a) \mathrel{+}= (1-\alpha)\cdot q(s,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} q(s',a')\right)$

$\quad\quad s = s'$ $\longrightarrow$ *Update the next state*

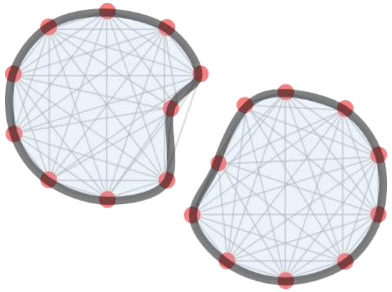


Summary

1 | ***Pseudo 3D Implementation***
Perspective of the player



2 | ***Soft-body Simulation***
Jelly & collision model



3 | ***Reinforcement Learning***
Training AI player

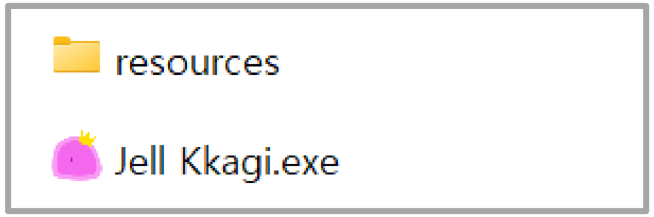
$$q^*(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \cdot \left(\max_{a'\in\mathcal{A}} q^*(s',a')\right)$$

Contact

2023-17519 이상화

010-4400-9422

torytony24@snu.ac.kr



github/torytony24