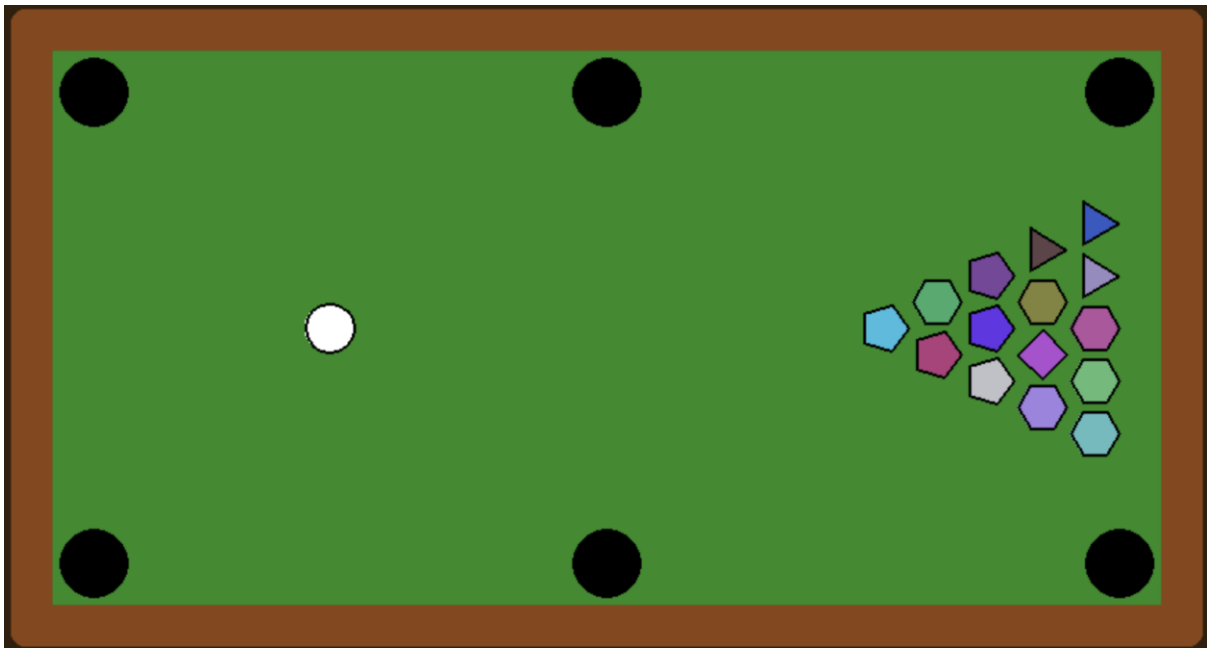
Project #3 Develop Your Own Physics Engine

2024105455 박민기

프로젝트 개요

프로젝트 명 : Polygon Billiards (다각형 당구 시뮬레이션)



Physics (물리)

- SAT (Separating Axis Theorem): 원이 아닌 임의의 볼록 다각형(Convex Polygon) 간의 정밀 충돌 감지
- Rotational Dynamics : 충돌 지점과 무게 중심의 차이에 의한 토크 발생 및 회전 구현
- Friction & Damping : 당구대의 현실감을 위한 선형/각속도 마찰력 시뮬레이션

Game Logic

- Drag & Shoot : 마우스 드래그 벡터를 힘(Force)으로 변환하여 공 발사
- Pocket System: 당구대 포켓 구현 및 승리 조건 처리

제작 목표

Physics : 다각형의 정밀 충돌 처리 및 회전 역학 구현

- 기존의 당구는 단순한 원 간의 충돌은 거리 계산만으로 충분하지만, 삼각형, 사각형, 육각형 등 다양한 형태의 물체가 상호작용하는 물리 엔진을 제작하는 것을 목표로 했다.
- SAT 알고리즘을 구현하여 오차 없는 충돌 감지를 수행했다.
- 물체의 중심이 아닌 곳에 충돌이 발생했을 때 회전이 발생하는 물리 현상을 구현하기 위해 관성 모멘트와 충격량(Impulse)공식을 적용했다.

디자인 & 구조

- 수구 (Cue Ball): 16각형(거의 원형)의 흰색 물체. 플레이어가 마우스로 조작한다
- 적구 (target Balls): 랜덤한 색상과 꼭짓점(3~6각형)을 가진 다각형 물체들.
- 당구대 (Environment): 마찰력이 작용하는 바닥과, 탄성 충돌을 일으키는 벽, 그리고 공을 제거하는 6개의 포켓으로 구성됨
- UI: 우측 상단 메뉴 버튼을 통해 언제든 게임을 재시작하거나 종료할 수 있음.

특징 설명 (Feature Description with Code)

RigidBody 클래스 및 상태 업데이트

물체의 물리적 속성(위치, 속도, 회전, 질량, 관성 모멘트)을 정의하고, 당구대의 물리적

특성을 반영하여 선형 및 회전 마찰력을 적용했습니다.

```
class RigidBody:
    def __init__(self, x, y, mass, color, shape_type="poly", size=30):
        self.pos = np.array([float(x), float(y)])
        self.vel = np.array([0.0, 0.0])
        self.angle = 0.0
        self.ang_vel = 0.0

        self.mass = mass
        self.inv_mass = 1.0 / mass if mass > 0 else 0.0
        self.color = color
        self.size = size
        self.is_cue = (shape_type == "cue")

        radius = size / 2.0
        self.inertia = 0.5 * mass * (radius ** 2)
        self.inv_inertia = 1.0 / self.inertia if self.inertia > 0 else 0.0
```

Physics Update : 오일러 적분법을 사용하여 위치와 각도를 갱신하고, 마찰력(Friction)을 적용하여 자연스러운 감속을 구현했습니다.

```
def update(self, dt):
    self.pos += self.vel * dt
    self.angle += self.ang_vel * dt

    self.vel *= 0.985
    self.ang_vel *= 0.98
```

-> 이 구조를 통해 모든 공이 독립적인 강체로 동작하며, 위치/속도/회전/질량 등 물리적 속성을 엔진에서 일관되게 처리할 수 있습니다.

SAT (Separating Axis Theorem) 충돌 감지

원이 아닌 다각형 간의 충돌을 감지하기 위해 SAT 알고리즘을 사용했습니다. 두 다각형의 모든 변의 법선을 분리 축으로 사용하여 투영하고 겹침을 검사합니다.

```python
def check_collision_sat(body_a, body_b):
    verts_a = body_a.get_world_vertices()
    verts_b = body_b.get_world_vertices()

    axes = get_axes(verts_a) + get_axes(verts_b)

    min_overlap = float('inf')
    collision_normal = None

    for axis in axes:
        min_a, max_a = project(verts_a, axis)
        min_b, max_b = project(verts_b, axis)

        if max_a < min_b or max_b < min_a:
            return False, None, 0

        overlap = min(max_a, max_b) - max(min_a, min_b)
        if overlap < min_overlap:
            min_overlap = overlap
            collision_normal = axis

    if np.dot(body_b.pos - body_a.pos, collision_normal) < 0:
        collision_normal = -collision_normal

    return True, collision_normal, min_overlap
```

-> 원형 충돌보다 더 일반적인 다각형 충돌을 정확하게 처리할 수 있어, 보다 복잡한 형태의 물체를 포함하는 물리 시뮬레이션이 가능합니다.

충격량 기반 충돌 반응 (Impulse Response)

충돌 시 물체가 튕겨 나가고 회전하도록 충격량(J)을 계산하여 적용합니다. 특히 충돌 지

점이 중심과 일치하지 않을때 발생하는 토크를 반영하여 각속도를 변화시킵니다.

```
def resolve_collision(body_a, body_b, normal, depth):
    total_inv = body_a.inv_mass + body_b.inv_mass
    if total_inv == 0: return

    move = normal * (depth * 0.5)
    body_a.pos -= move
    body_b.pos += move

    contact_a = get_support(body_a.get_world_vertices(), normal)
    contact_b = get_support(body_b.get_world_vertices(), -normal)

    vals_a = np.dot(contact_a - body_b.pos, -normal)
    vals_b = np.dot(contact_b - body_a.pos, normal)
    if vals_a > vals_b:
        contact_point = contact_a
    else:
        contact_point = contact_b

    r_a = contact_point - body_a.pos
    r_b = contact_point - body_b.pos

    def cross_2d(v1, v2): return v1[0]*v2[1] - v1[1]*v2[0]

    vel_a = body_a.vel + np.array([-body_a.ang_vel * r_a[1], body_a.ang_vel * r_a[0]])
    vel_b = body_b.vel + np.array([-body_b.ang_vel * r_b[1], body_b.ang_vel * r_b[0]])
    rel_vel = vel_b - vel_a
    vel_normal = np.dot(rel_vel, normal)

    if vel_normal > 0: return
```

```
e = 0.9
rn_a = cross_2d(r_a, normal)
rn_b = cross_2d(r_b, normal)

denom = (total_inv + rn_a**2 * body_a.inv_inertia + rn_b**2 * body_b.inv_inertia)
j = -(1 + e) * vel_normal / denom

impulse = j * normal

body_a.vel -= impulse * body_a.inv_mass
body_a.ang_vel -= cross_2d(r_a, impulse) * body_a.inv_inertia

body_b.vel += impulse * body_b.inv_mass
body_b.ang_vel += cross_2d(r_b, impulse) * body_b.inv_inertia
```

-> 단순 속도 반전 방식이 아닌 실제 회전·관성 기반 충돌을 반영하므로 엔진의 현실성이 크게 향상됩니다.

게임 규칙 및 상호작용

사용자 입력(Drag & Shoot)을 힘으로 변환하고, 공이 포켓에 들어갔을 때의 게임 규칙(제거 또는 리셋)을 처리합니다.

```
elif event.type == pygame.MOUSEBUTTONUP:
    if event.button == 1 and is_dragging:
        end_drag = mouse_pos
        force = np.array(start_drag) - np.array(end_drag)
        if np.linalg.norm(force) > 300:
            force = force / np.linalg.norm(force) * 300
        cue_ball.vel = force * 5.0
        is_dragging = False
```

```
if check_pocket_fall(ball, pockets):
    if ball.is_cue:
        ball.pos = np.array([TABLE_X + 200, TABLE_Y + TABLE_HEIGHT/2])
        ball.vel[:] = 0
        ball.ang_vel = 0
    else:
        balls_to_remove.append(ball)
```

-> 사용자 입력이 실제 힘(force)으로 변환되어 엔진의 물리 연산과 자연스럽게 연결되도록 설계했습니다.

기술적 구현 및 기여 (Technical Implementation & Own Contributions)

1. Physics Engine 구조

- 각 물체는 RigidBody클래스로 관리되며, 위치, 속도, 각도, 질량, 관성모멘트 등을

포함한 강체 물리 모델을 사용한다

- 오일러 방식으로 업데이트하고, 마찰을 단순화한 감쇠(damping)를 적용해 자연스러운 움직임을 구현하였다.

2. Collision Detection

- 각 다각형의 엣지에서 법선 축 생성
- 축에 대한 projection 비교 -> overlap 여부 계산
- 모든 축에서 겹치면 충돌, 최소 overlap 축을 침투 방향으로선택

-> 이를 통해 원형이 아닌 임의의 다각형 간 충돌을 처리할 수 있다.

3. Collision Response (Impulse + Rotation)

- 접촉점 (contact point) 계산
- 상대 속도 및 각속도 기여 계산
- 질량 + 관성모멘트를 모두 고려한 충돌 임펄스 적용
- 침투 보정으로 안정성 향상

-> 이를 통해 회전이 포함된 자연스러운 반응을 구현하였다.

4. Engine Gameplay Features

- 모든 공을 원 대신 랜덤 볼록다각형 rigid body로 생성
- 드래그 기반 큐 힘 시스템
- 벽 반사 및 포켓 낙하 처리
- 메뉴/리셋 UI 포함한 전체 게임 루프 구성

5. Own Contributions

- 다각형 기반 rigid body 구조 구현
- SAT 충돌 감지 알고리즘 구현
- 회전 포함 Impulse 충돌 반응 구현

- 침투 보정, damping 등 엔진 안정화 기능 추가
- 큐 force 시스템, 포켓 감지, UI 포함 전체 게임 구조 설계

시연 영상 :
https://drive.google.com/file/d/1kIgjtxg_xjeNEmadqpcmlTWOo5dM0ZLn/view?usp=share_link