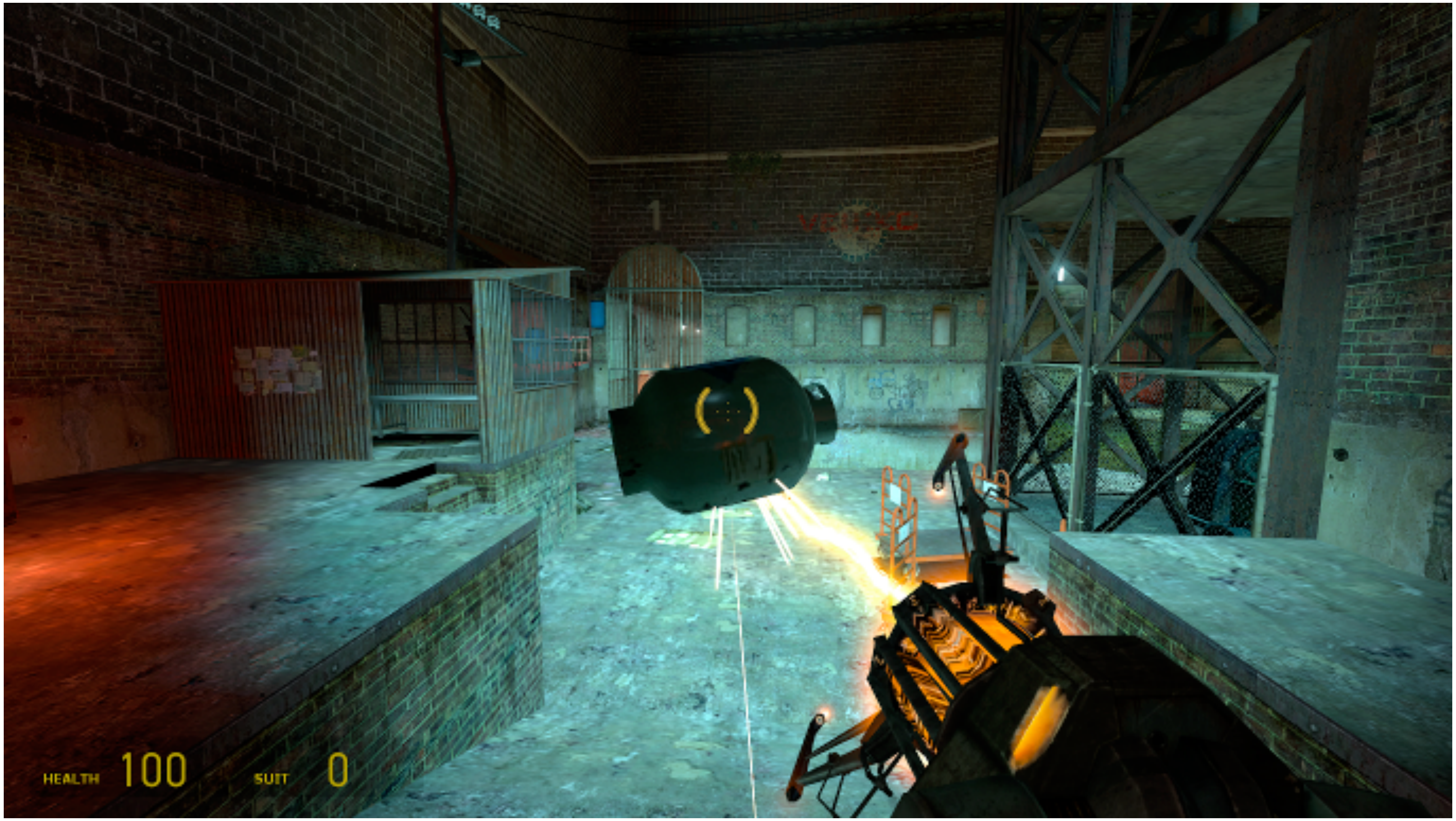
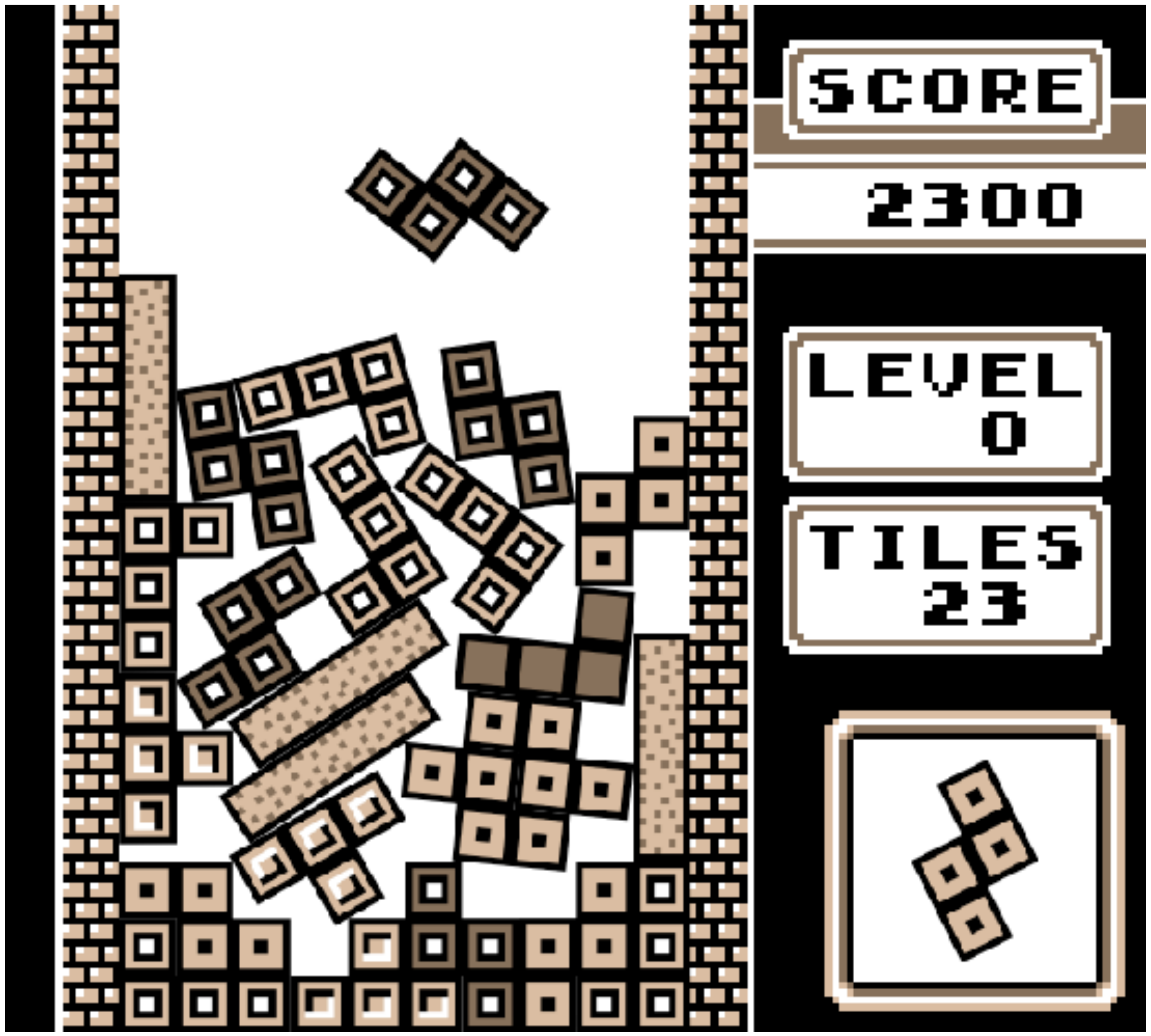
CSCI 4611

Physics engines

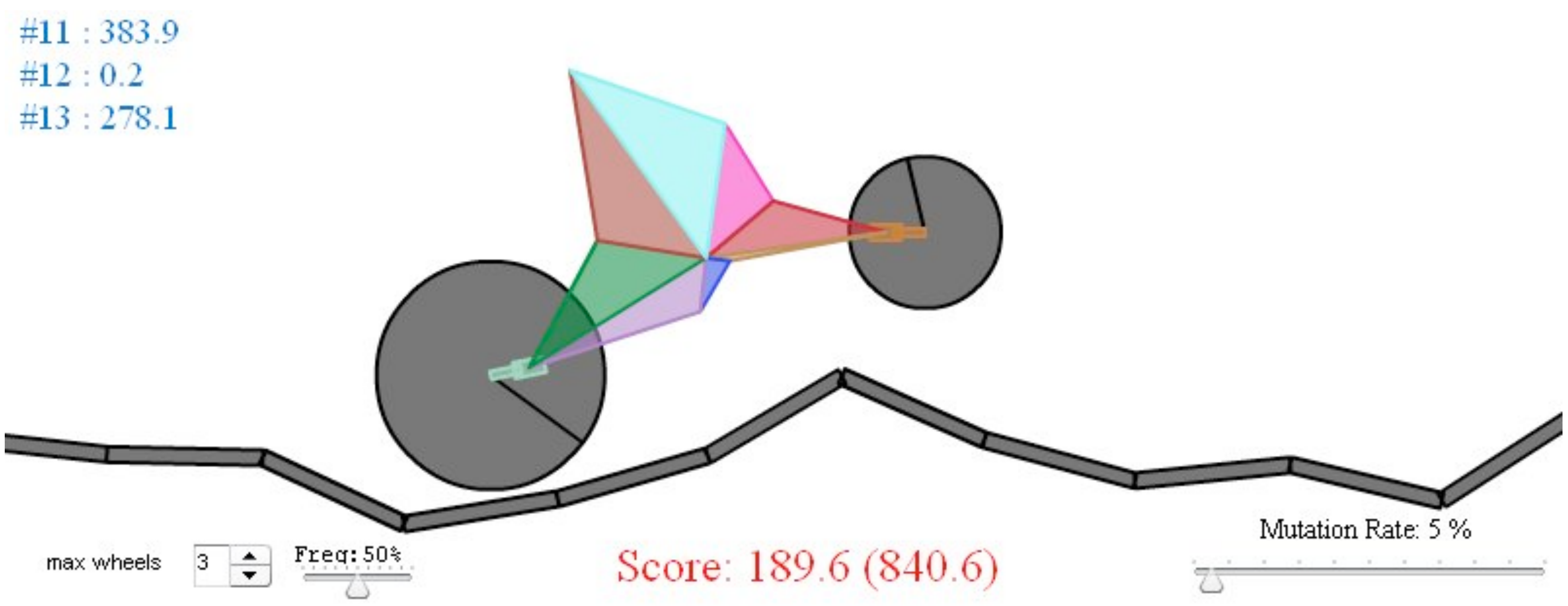
Interactive physics



Interactive physics



Interactive physics



#11 : 383.9
#12 : 0.2
#13 : 278.1

max wheels 3

Freq: 50%

Score: 189.6 (840.6)

Mutation Rate: 5 %

Physics engines

A physics engine only deals with the *physics* of the virtual world: the motion of objects, forces, collisions, etc.

Box2D

BULLET PHYSICS LIBRARY

havok

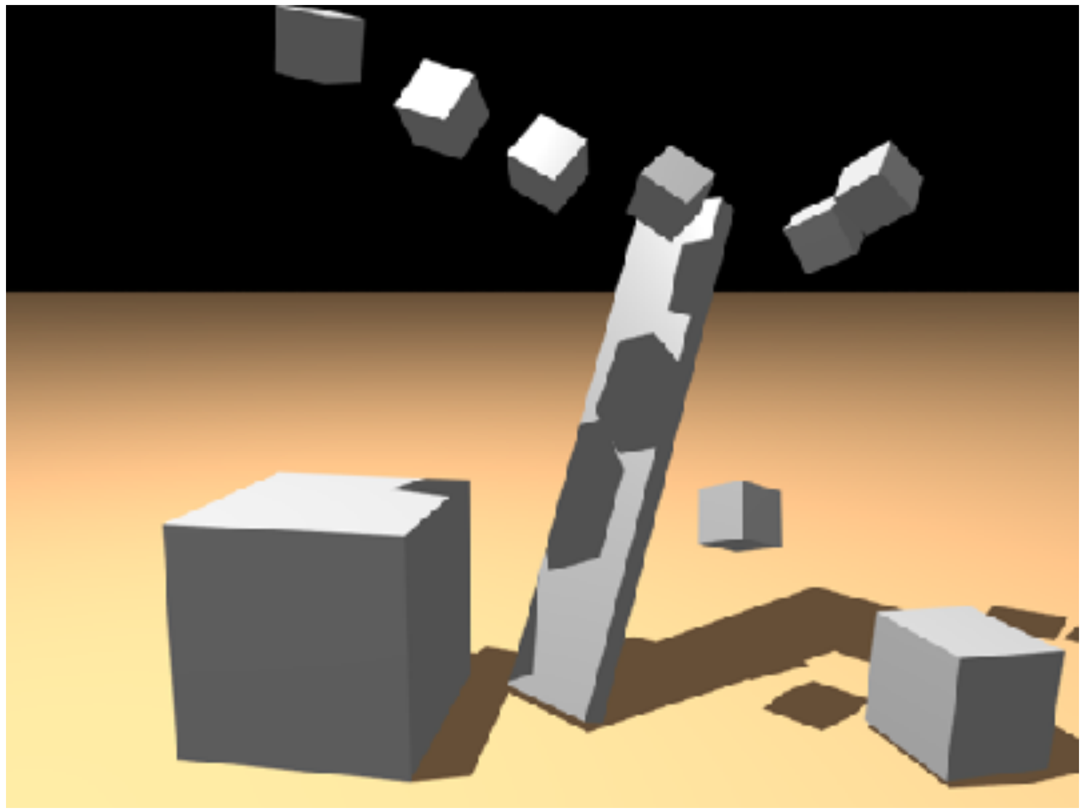
CHIPMUNK 7

OPEN DYNAMICS ENGINE™

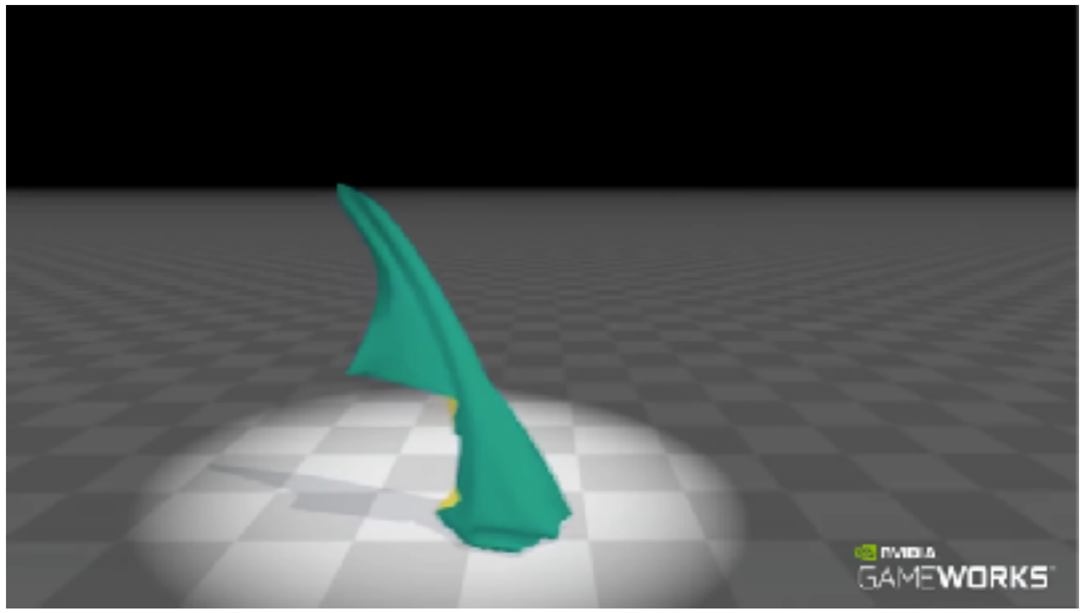
NVIDIA PHYSX

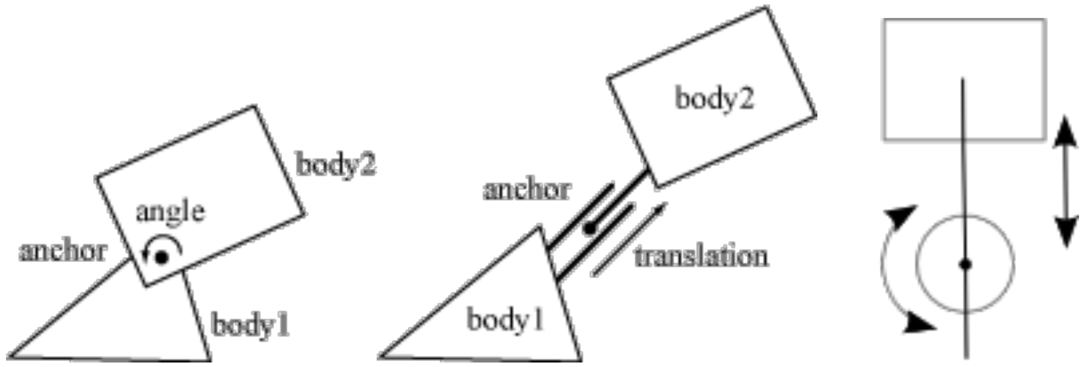
Physics

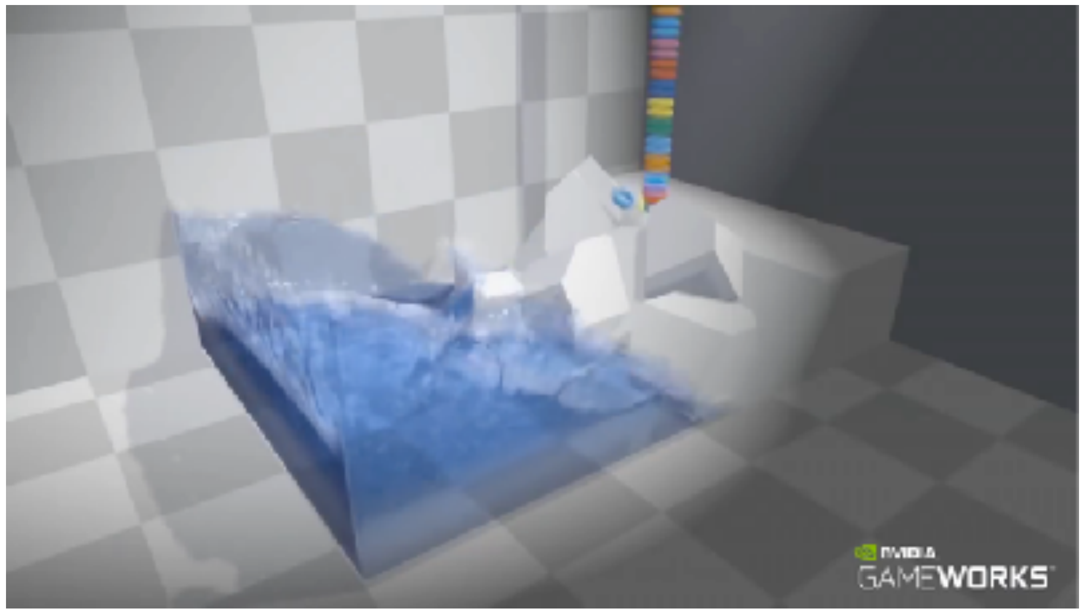
Deformable bodies



Rigid bodies







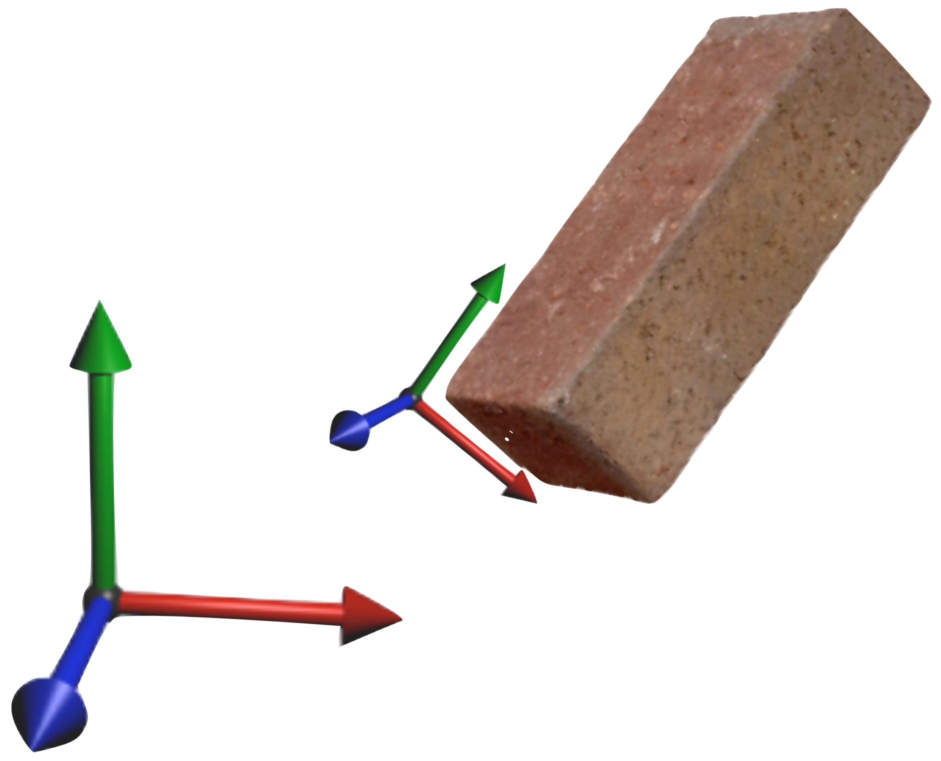
Fluids

Rigid bodies

A rigid body is an object that can only transform *rigidly*, i.e. only translation and rotation — no stretching, bending, twisting, etc.

- like a coordinate frame with a shape attached

- Physics engine calculates forces and updates pose on each simulation step



Using a physics engine

Initialization:

- Initial state (position, rotation, linear & angular velocity)
- Physical properties (density, friction, bounciness, etc.)
- Collision shape

Every frame:

- Step engine forward by Δt
- Get current state (position, rotation, etc.)

What a physics engine doesn’t do

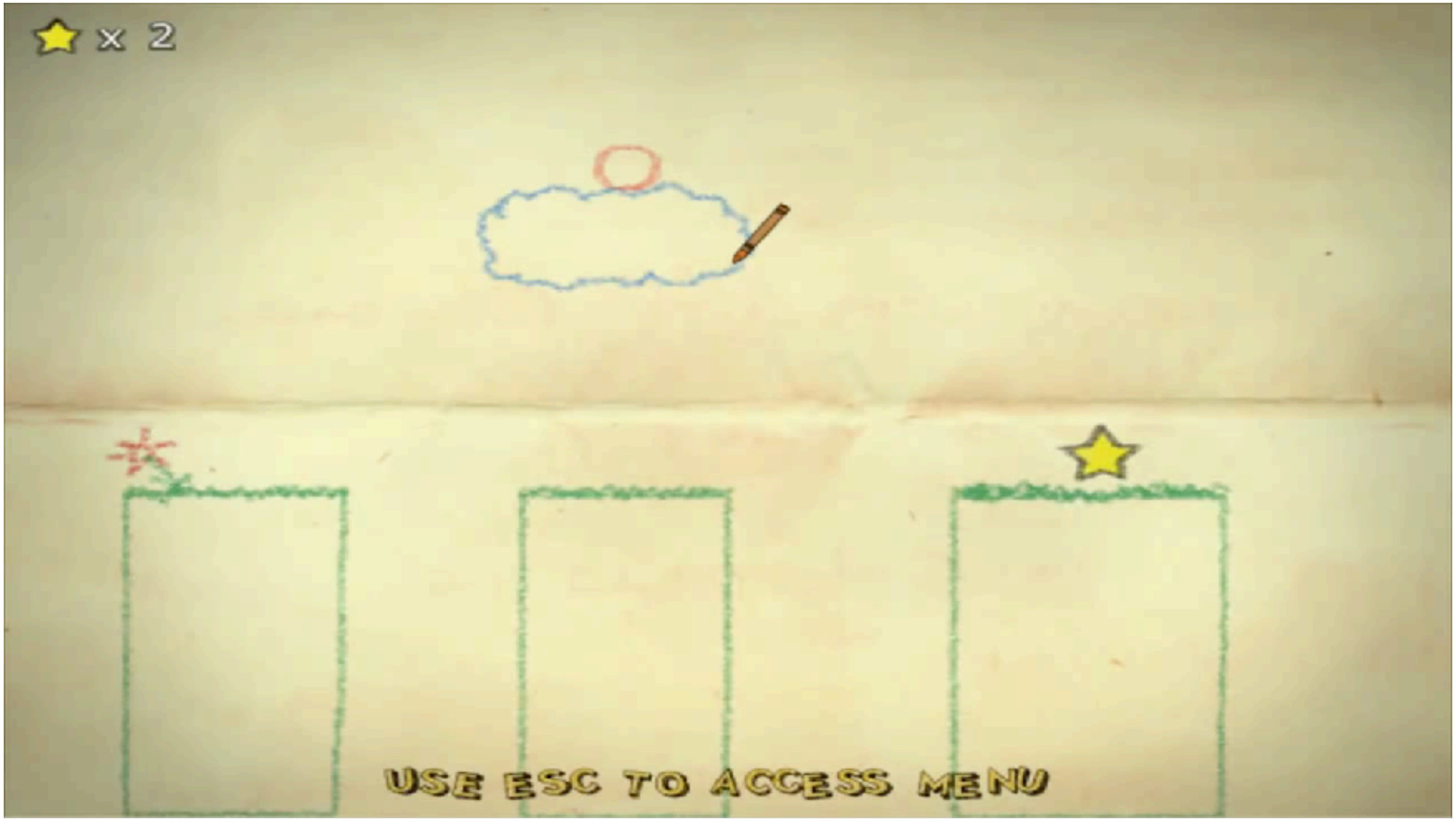
Rendering

- Take the computed state and draw your own shapes
- *“It is best to think of [the physics engine’s] bodies as moving billboards upon which you attach your artwork.”*

User interaction

- You have to add forces/impulses/constraints based on user input

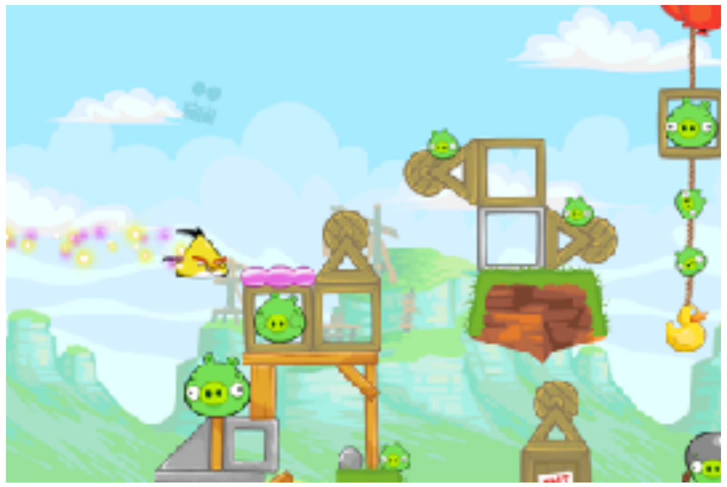
Assignment 6 (is inspired by)

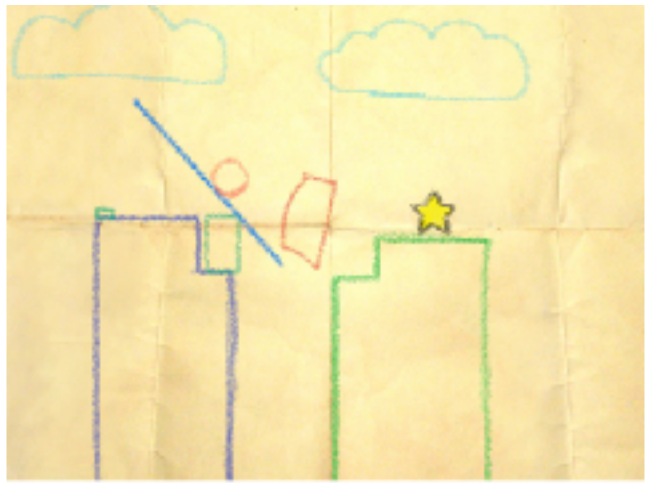


Assignment 6 objectives

- Connect a physics engine to an interactive graphics application
- Dynamically add and remove physics objects
- Manipulate physics objects with mouse interaction
- Listen for and react to collision events

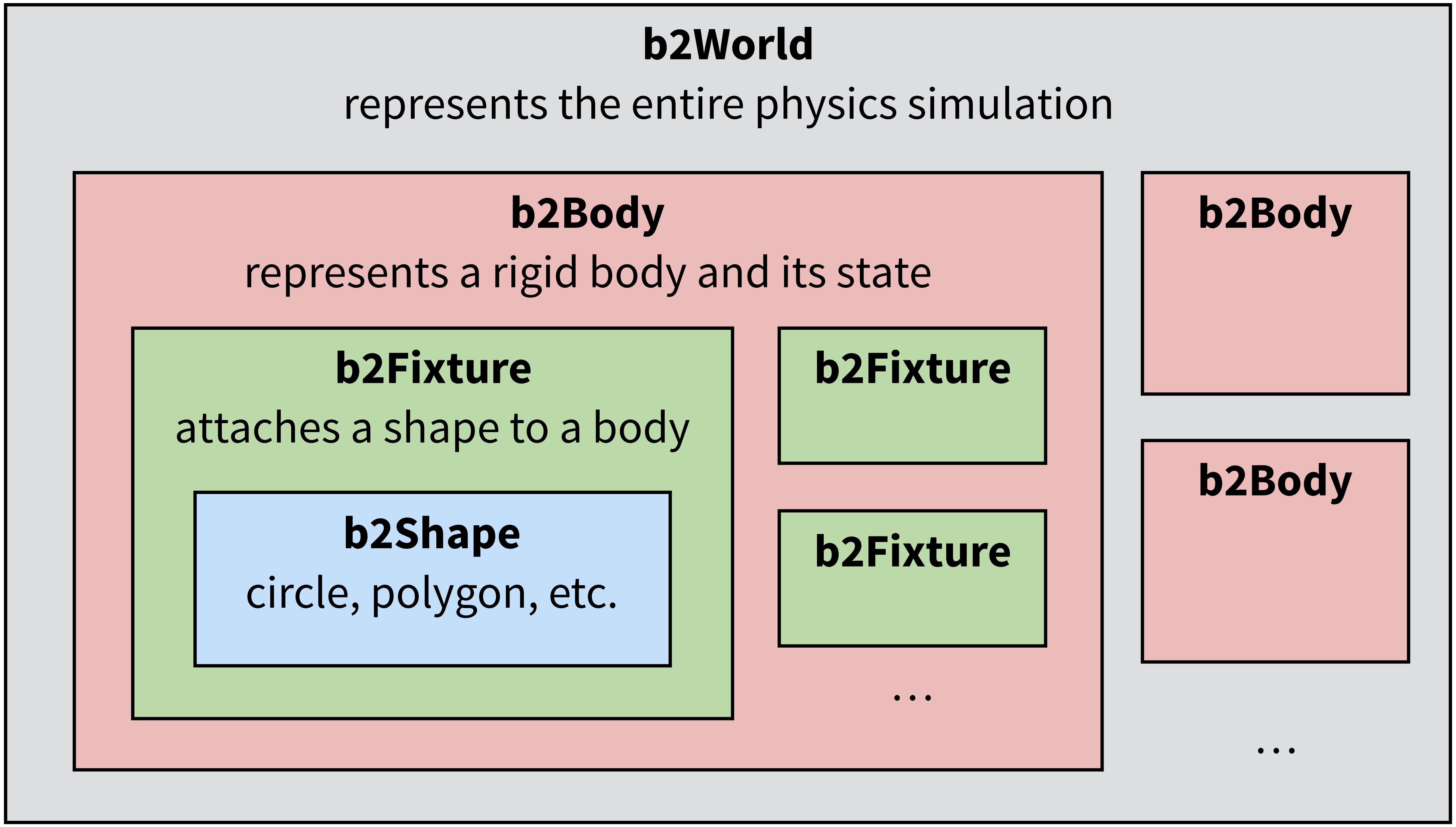
Box2D: http://box2d.org/



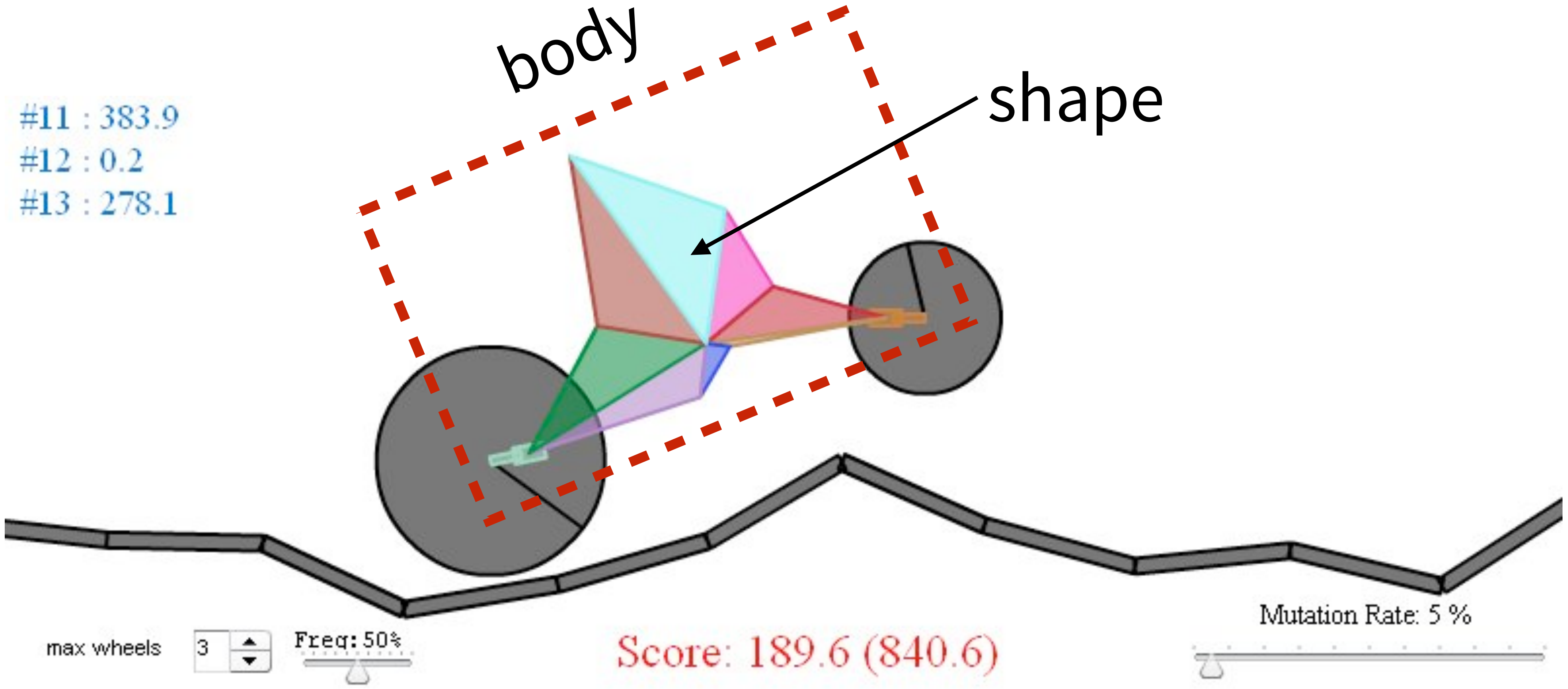


- Only rigid bodies in 2D
- Uses SI units (meters, kilograms, seconds)
- Very good and readable manual

Box2D overview


b2World
represents the entire physics simulation
b2Body
represents a rigid body and its state
b2Fixture
attaches a shape to a body
b2Shape
circle, polygon, etc.
b2Fixture
b2Fixture
...
b2Body
b2Body
...

Bodies, fixtures, and shapes



Getting started with Box2D

Step 1: Create the Box2D world.

```
// in initBox2D()
world = new b2World(b2Vec2(0., -9.8)); // gravity

// in advanceState(dt)
world->Step(dt, 8, 3);
```

should be kept fixed throughout the simulation

Step 2: Create a body (+ fixture + shape).

Box2D

```
// Create a new rigid body
b2BodyDef bodyDef;
bodyDef.type = b2_dynamicBody;
bodyDef.position.Set(0., 2.);
body = world->CreateBody(&bodyDef);
// Define a shape
b2PolygonShape polygon;
polygon.SetAsBox(0.2, 0.2);
// Use a fixture to connect the shape to the body
b2FixtureDef fixtureDef;
fixtureDef.shape = &polygon;
fixtureDef.density = 0.2;
fixtureDef.friction = 0.4;
fixtureDef.restitution = 0.4;
body->CreateFixture(&fixtureDef);
```

Bodies, fixtures, and shapes

- **`b2Body`**:
  tracks the pose, velocity, inertia of a rigid body
- **`b2Shape`**:
  a 2D shape, used for collision detection
- **`b2Fixture`**:
  attaches a shape to the rigid body’s frame, and sets some related properties (density, friction, etc.)

Bodies

- Bodies are static (immovable) by default unless set to be dynamic: `bodyDef.type = b2_dynamicBody`

- Set initial position in `bodyDef` before creating the body.
  Can also set `angle`, `linearVelocity`, `angularVelocity`

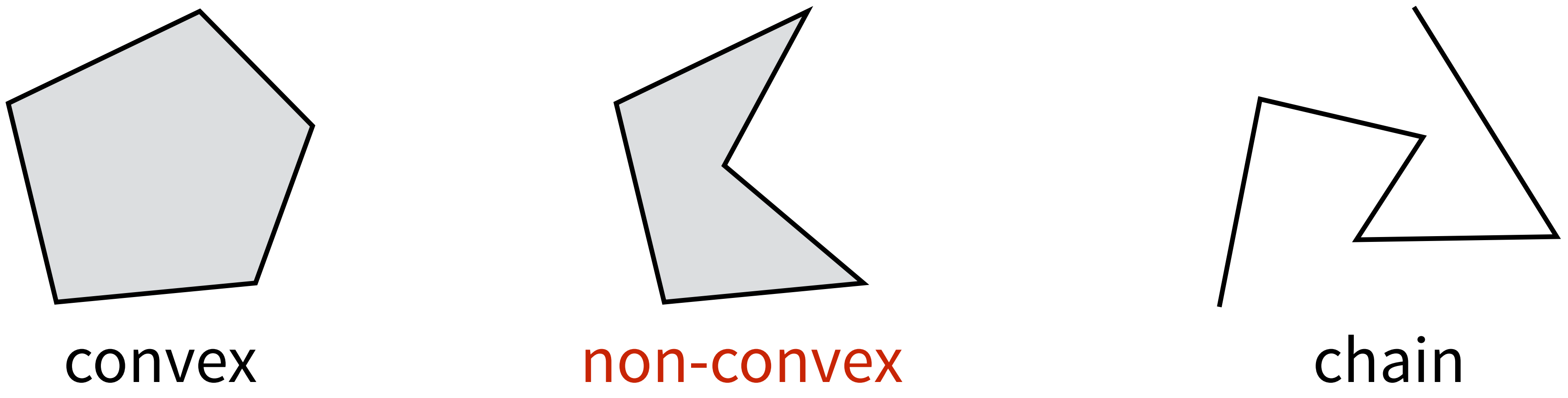
- `b2World` owns the pointer and will free the memory when the world is destroyed. Call
  `world->DestroyBody(bodyPointer)`
  to delete a body before that.

Fixtures

- **shape**
- **density**: mass per unit area (so Box2D can calculate a reasonable inertia for you)
- **friction**: between 0.0 and 1.0
- **restitution**: how bouncy the object is (also between 0.0 and 1.0)

Shapes

- `b2CircleShape`
- `b2PolygonShape` (must be convex)
- `b2ChainShape` (can’t be dynamic)



Why? Because fast collision detection is hard.

Getting started with Box2D

Step 3: Render the simulated world.

```
// in drawGraphics()
pushMatrix();
b2Vec2 pos = body->GetPosition();
float angle = body->GetAngle();
setMatrix(getMatrix()
          * glm::translate(…)
          * glm::rotate(…));
drawMesh(boxMesh, vec3(0.8,0.2,0.2));
popMatrix();
```

Getting started with Box2D

Let’s tell Box2D about the floor (a static 4m × 0.1m box) so that the body will bounce off it.

```
float width = 4., height = 0.1;
b2BodyDef floorDef;
floorDef.position.Set(0, -height/2);
b2Body *floorBody = world->CreateBody(&floorDef);
b2PolygonShape floorShape;
floorShape.SetAsBox(width/2, height/2);
floorBody->CreateFixture(&floorShape, 0.0f);
```

Exercise

Instead of creating a single box on initialization, allow the user to create arbitrarily many boxes.

1. Replace `b2Body* box` with a growable list of boxes (`vector<b2Body*> boxes`). Initially there should be no bodies. When the user presses a key, create a new box and add it to the list of boxes. In the rendering function, draw all the boxes in the list.

2. Try randomizing the initial linear and angular velocity.

Homework:

- Make sure Box2D works for you (on laptop or CSE Labs)
- Read first 2 chapters of the Box2D manual

Next class:

- Deformable objects using mass-spring systems