ESPRIT: Explaining Solutions to Physical ReasonIng Tasks

Aadit Vyas

LILY Workshop Fall 2019

Dec 13, 2019

CPSC 490 Project Proposal

- Explainable AI is an important field
- PHYRE could present an interesting training ground

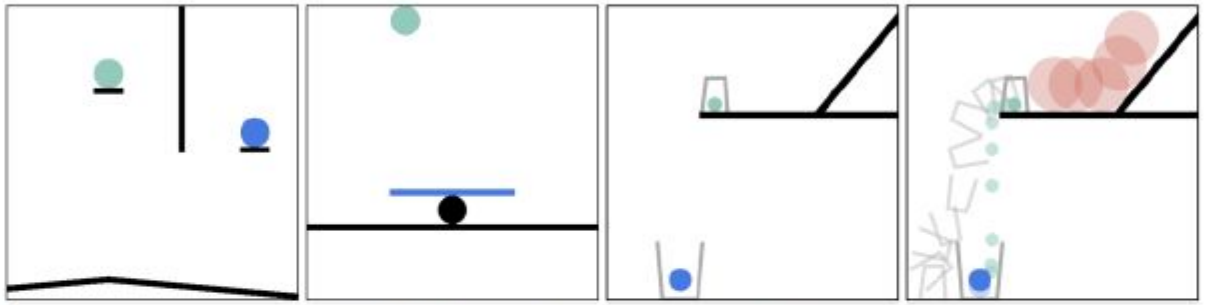
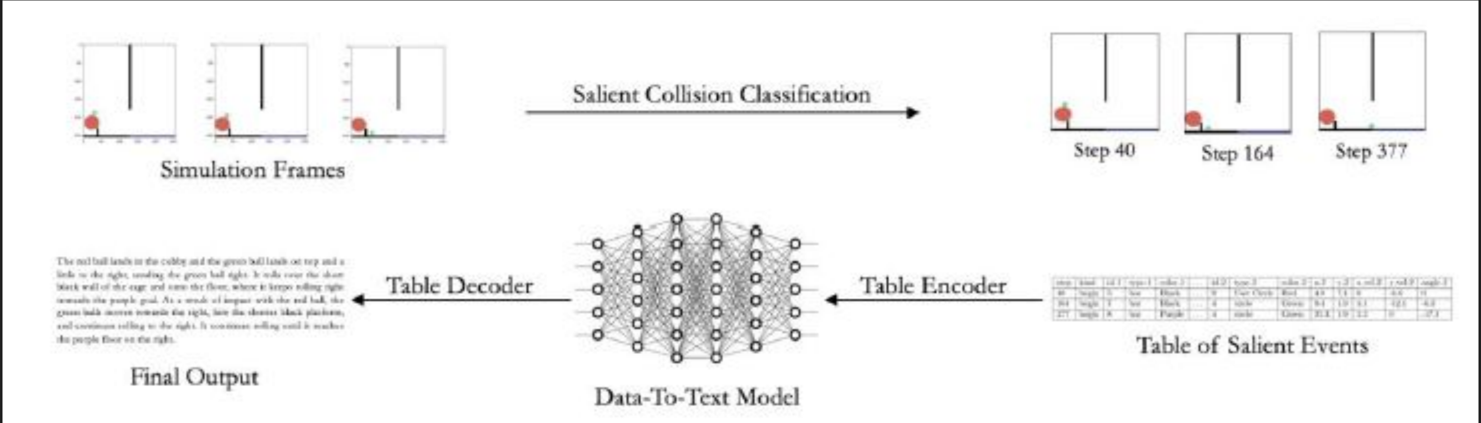


Figure 1: Three examples of PHYRE tasks (left) and one example solution (right). Black objects are static; objects with any other color are dynamic and subject to gravity. The tasks describe a terminal goal state that can be achieved by placing an additional object(s) in the world and running the simulator. The task in the left-most pane requires the placement of two balls to be solved, whereas the others can be solved with one ball. The right-most pane illustrates a solution (red ball) and the solution dynamics.

ESPRIT Final Project

- PHYRE dataset generation



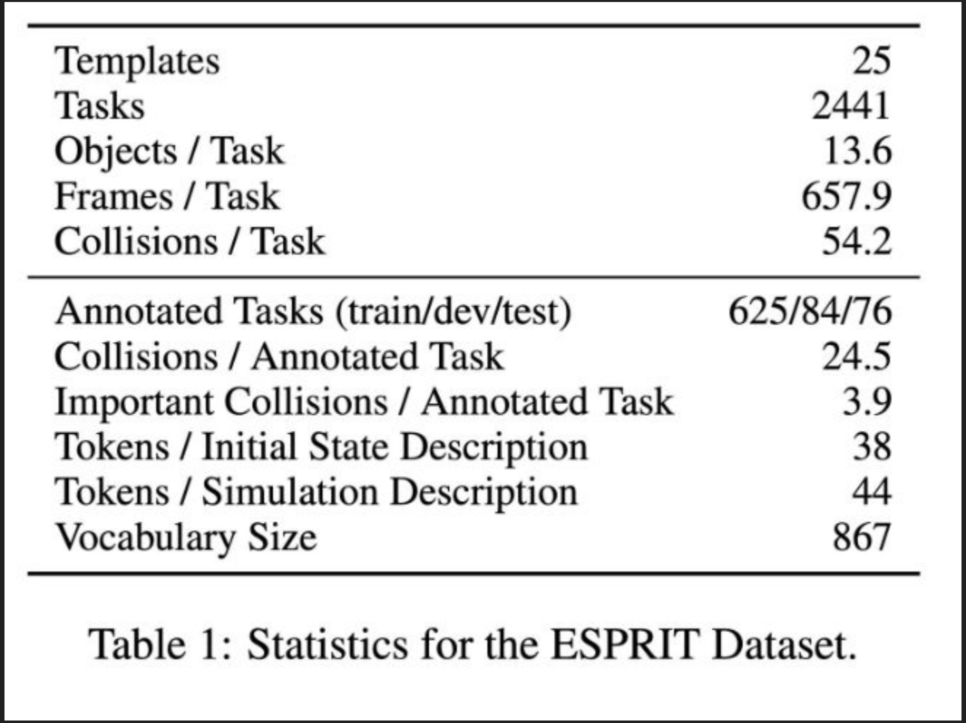
The red ball lands in the cubby and the green ball lands on top and a little to the right, sending the green ball right. It rolls over the short black wall of the cage and onto the floor, where it keeps rolling right towards the purple goal. As a result of impact with the red ball, the green balls moves towards the right, hits the shorter black platform, and continues rolling to the right. It continues rolling until it reaches the purple floor on the right.

step	kind	id 1	type 1	color 1	...	id 2	type 2	color 2	x 2	y 2	x vel 2	y vel 2	angle 2
40	Single	5	bar	Black	...	9	Cup Circle	Red	4.9	7.1	0	-0.6	0
164	Single	7	bar	Black	...	4	circle	Green	9.1	1.9	3.1	-12.1	-4.3
377	Single	8	bar	Purple	...	4	circle	Green	35.8	1.9	2.2	0	-27.1

Figure 2: The ESPRIT framework for identifying salient physical events and explaining solutions using natural language generation.

Building the ESPRIT Dataset

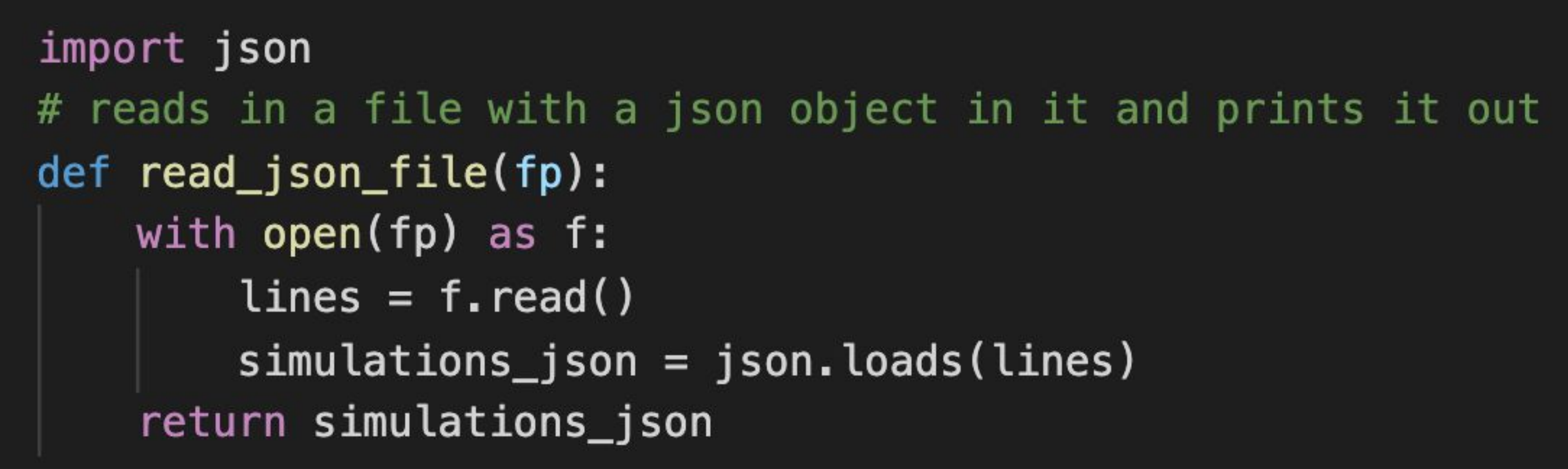
- Intercepting the PHYRE simulator
- Extracting information from Thrift objects
- Storing information in objects
- Windowing steps



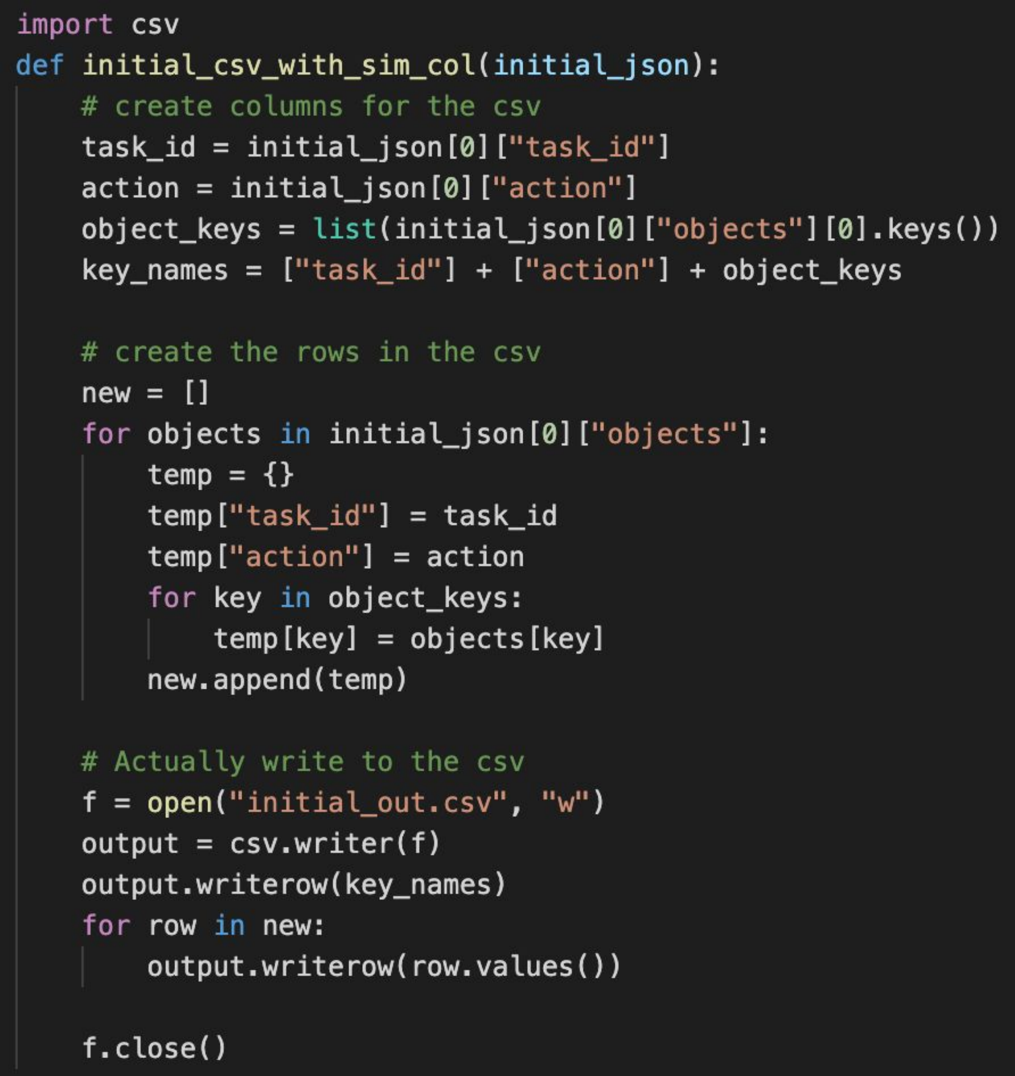
Templates	25
Tasks	2441
Objects / Task	13.6
Frames / Task	657.9
Collisions / Task	54.2
Annotated Tasks (train/dev/test)	625/84/76
Collisions / Annotated Task	24.5
Important Collisions / Annotated Task	3.9
Tokens / Initial State Description	38
Tokens / Simulation Description	44
Vocabulary Size	867

Table 1: Statistics for the ESPRIT Dataset.

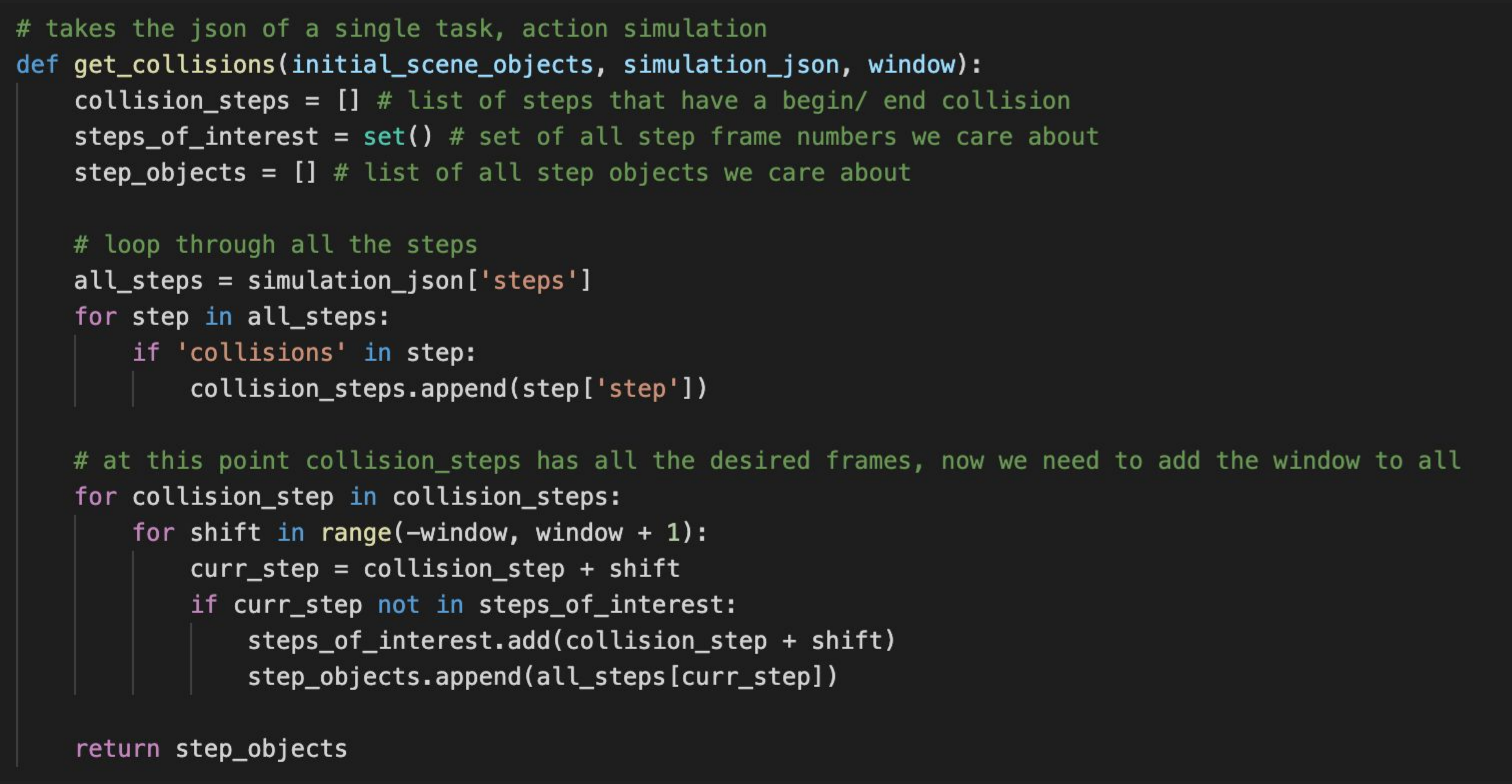
Magic of JSON



```python
import json
# reads in a file with a json object in it and prints it out
def read_json_file(fp):
    with open(fp) as f:
        lines = f.read()
        simulations_json = json.loads(lines)
    return simulations_json
```

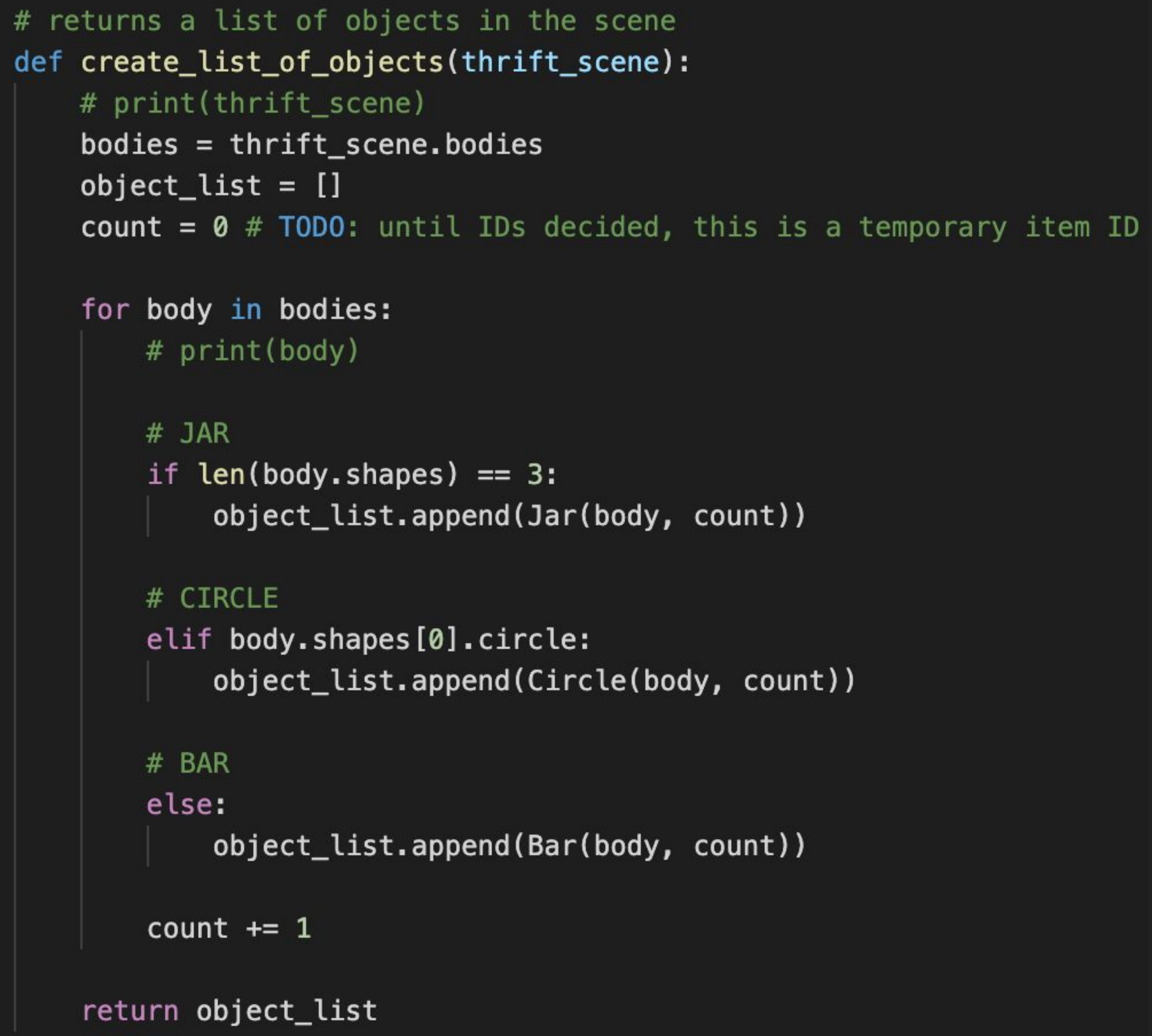


```python
import csv
def initial_csv_with_sim_col(initial_json):
    # create columns for the csv
    task_id = initial_json[0]["task_id"]
    action = initial_json[0]["action"]
    object_keys = list(initial_json[0]["objects"][0].keys())
    key_names = ["task_id"] + ["action"] + object_keys

    # create the rows in the csv
    new = []
    for objects in initial_json[0]["objects"]:
        temp = {}
        temp["task_id"] = task_id
        temp["action"] = action
        for key in object_keys:
            temp[key] = objects[key]
        new.append(temp)

    # Actually write to the csv
    f = open("initial_out.csv", "w")
    output = csv.writer(f)
    output.writerow(key_names)
    for row in new:
        output.writerow(row.values())

    f.close()
```

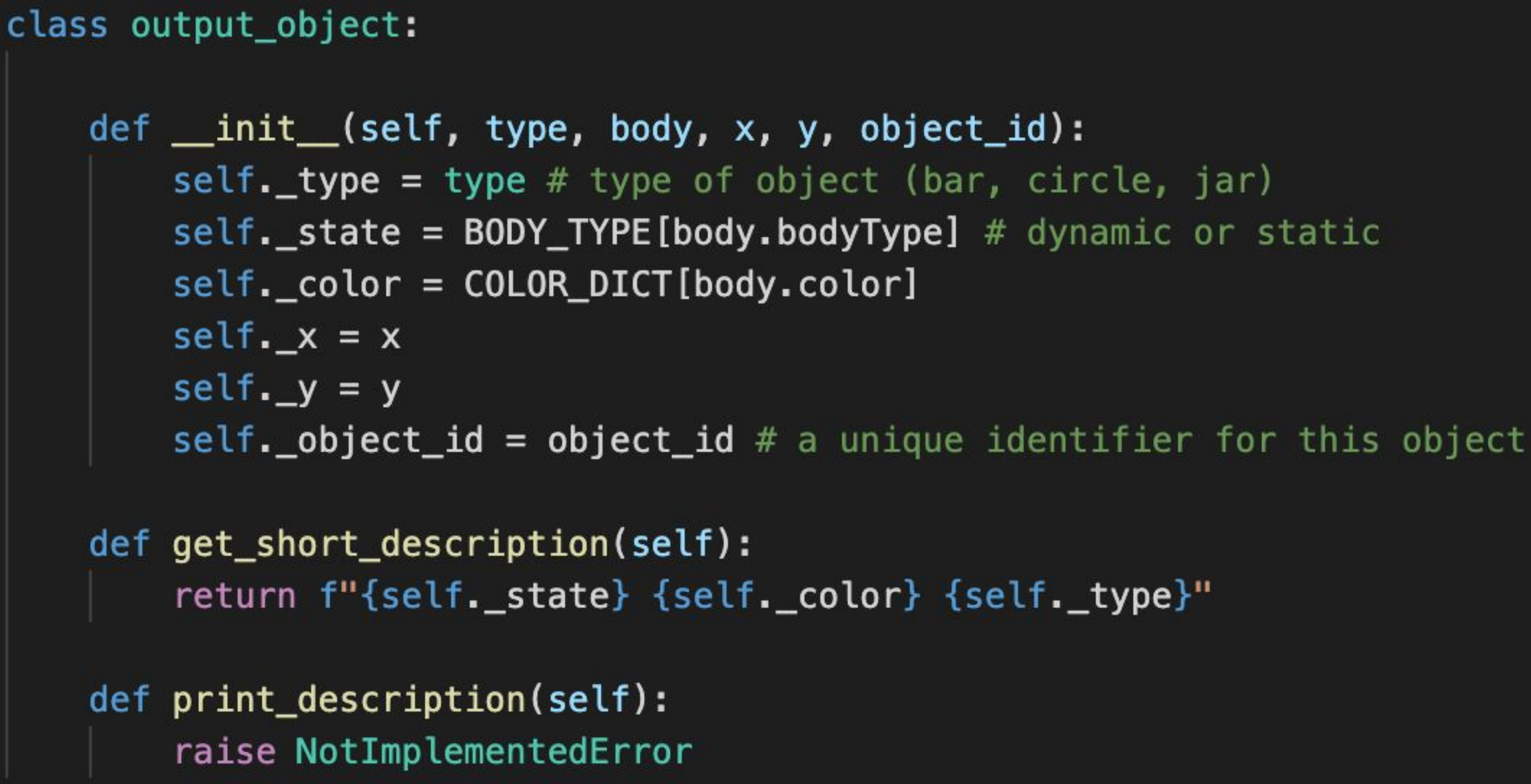


```python
# takes the json of a single task, action simulation
def get_collisions(initial_scene_objects, simulation_json, window):
    collision_steps = [] # list of steps that have a begin/ end collision
    steps_of_interest = set() # set of all step frame numbers we care about
    step_objects = [] # list of all step objects we care about

    # loop through all the steps
    all_steps = simulation_json['steps']
    for step in all_steps:
        if 'collisions' in step:
            collision_steps.append(step['step'])

    # at this point collision_steps has all the desired frames, now we need to add the window to all
    for collision_step in collision_steps:
        for shift in range(-window, window + 1):
            curr_step = collision_step + shift
            if curr_step not in steps_of_interest:
                steps_of_interest.add(collision_step + shift)
                step_objects.append(all_steps[curr_step])

    return step_objects
```

Creating Objects

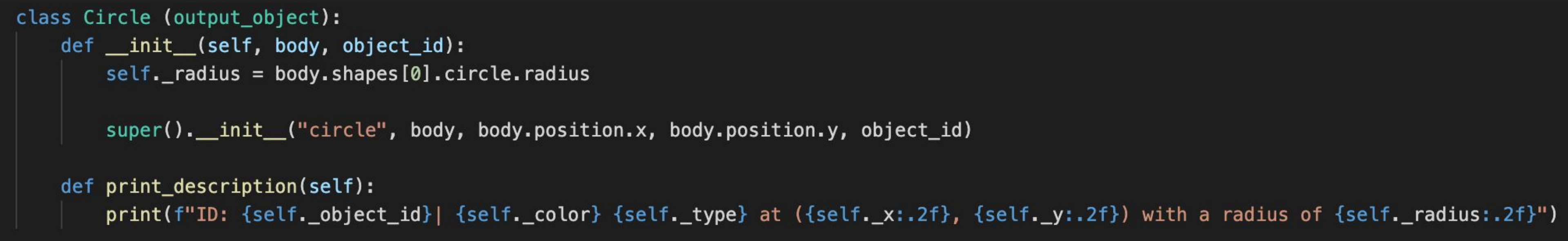


```python
# returns a list of objects in the scene
def create_list_of_objects(thrift_scene):
    # print(thrift_scene)
    bodies = thrift_scene.bodies
    object_list = []
    count = 0 # TODO: until IDs decided, this is a temporary item ID

    for body in bodies:
        # print(body)

        # JAR
        if len(body.shapes) == 3:
            object_list.append(Jar(body, count))

        # CIRCLE
        elif body.shapes[0].circle:
            object_list.append(Circle(body, count))

        # BAR
        else:
            object_list.append(Bar(body, count))

        count += 1

    return object_list
```



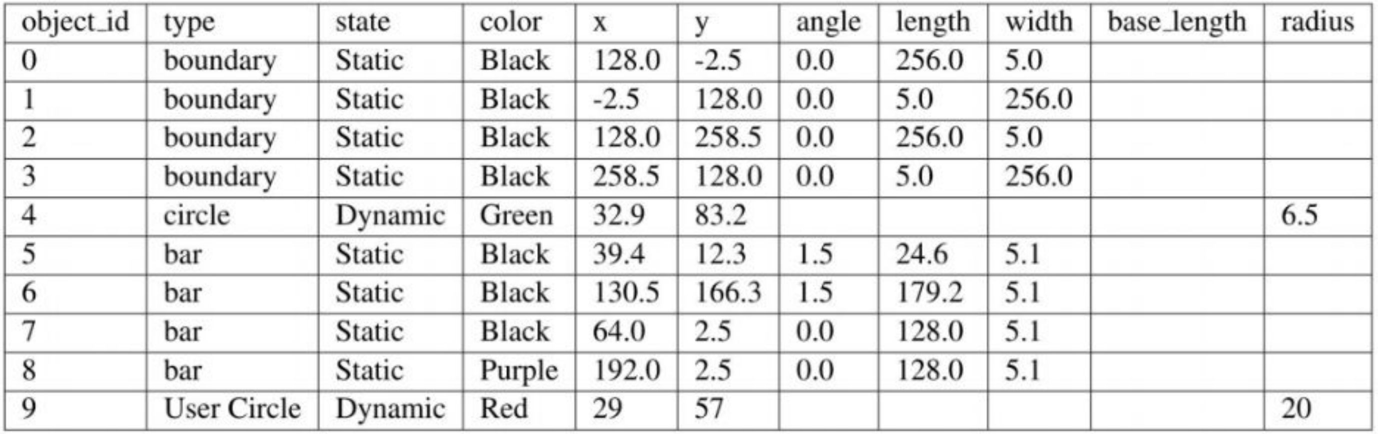
```python
class output_object:

    def __init__(self, type, body, x, y, object_id):
        self._type = type # type of object (bar, circle, jar)
        self._state = BODY_TYPE[body.bodyType] # dynamic or static
        self._color = COLOR_DICT[body.color]
        self._x = x
        self._y = y
        self._object_id = object_id # a unique identifier for this object

    def get_short_description(self):
        return f"{self._state} {self._color} {self._type}"

    def print_description(self):
        raise NotImplementedError
```

```python
class Jar (output_object):
    def __init__(self, body, object_id):
        x, y = body.position.x, body.position.y # this is the center of the base of the jar
        self._angle = body.angle # note this is in radians
        self._width = self.calculate_base_width(body) # assumption: that each bar has the same width

        self._baselen = self.calculate_base_length(body)
        self._sidelen = self.calculate_side_length(body)

        super().__init__("jar", body, x, y, object_id)

    # calculate the base length
    def calculate_base_length(self, body):
        shapes = body.shapes
        base = shapes[0]
        top_right = base.polygon.vertices[0].x
        return abs(top_right) * 2

    # calculate the width of a bar
    def calculate_base_width(self, body):
        shapes = body.shapes
        base = shapes[0]
        top_right = base.polygon.vertices[0].y
        return abs(top_right) * 2
```



```python
class Circle (output_object):
    def __init__(self, body, object_id):
        self._radius = body.shapes[0].circle.radius

        super().__init__("circle", body, body.position.x, body.position.y, object_id)

    def print_description(self):
        print(f"ID: {self._object_id}| {self._color} {self._type} at ({self._x:.2f}, {self._y:.2f}) with a radius of {self._radius:.2f}")
```

Representing the initial state as a structured table



object_id	type	state	color	x	y	angle	length	width	base_length	radius
0	boundary	Static	Black	128.0	-2.5	0.0	256.0	5.0		
1	boundary	Static	Black	-2.5	128.0	0.0	5.0	256.0		
2	boundary	Static	Black	128.0	258.5	0.0	256.0	5.0		
3	boundary	Static	Black	258.5	128.0	0.0	5.0	256.0		
4	circle	Dynamic	Green	32.9	83.2					6.5
5	bar	Static	Black	39.4	12.3	1.5	24.6	5.1		
6	bar	Static	Black	130.5	166.3	1.5	179.2	5.1		
7	bar	Static	Black	64.0	2.5	0.0	128.0	5.1		
8	bar	Static	Purple	192.0	2.5	0.0	128.0	5.1		
9	User Circle	Dynamic	Red	29	57					20

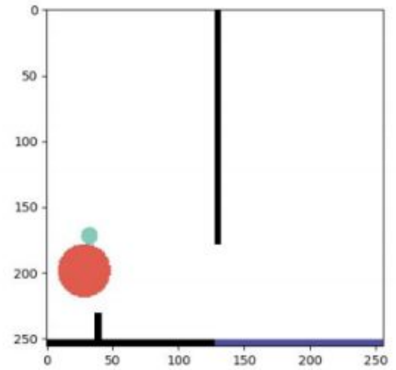


Table 1: Representing the initial state as a structured table. Example for task 00014:496.

Future Work

- Creating an agent that trains on the data-to-text output and plays PHYRE
- Expand the ESPRIT dataset