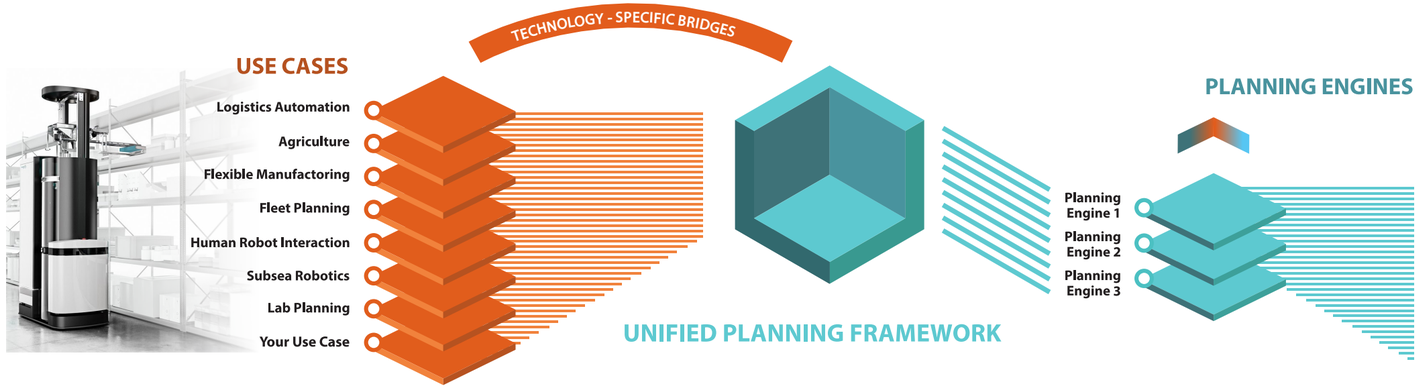
Unified Planning: A Python Library Making Planning Technology Accessible

AIPLAN 4EU



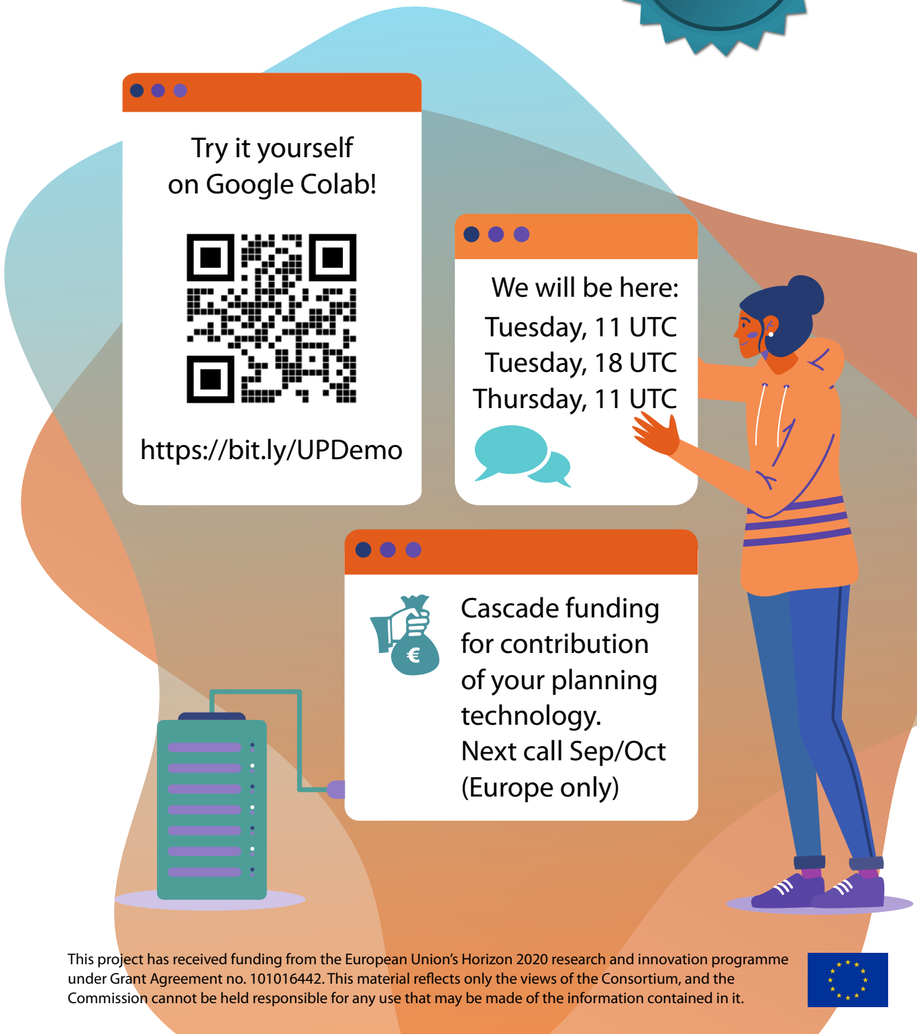
Apache License 2.0

programmatical modelling, transformation and solving of planning tasks in one framework

multi-paradigm: classical, temporal, numeric, multi-agent, hierarchical ...

multi-engine: e.g. Fast Downward, ENHSP, TAMER, skdecide, Tarski transformations, ...

```python
from unified_planning.shortcuts import *

# Suppose we have a graph of locations and we want to plan how to move from INIT to DEST...
# This is application-specific data
location_map, INIT, DEST = generate_networkx_topolgy(), 'start_node', 'goal_node'
locations = {str(l) : Object(str(l), Location) for l in location_map.nodes}

# Planning problems can be created entirely from code!
# Python code and libraries can be used to build problems and plans are data structures
Location = UserType('Location')
problem = Problem('robot')

robot_at = Fluent('robot_at', BoolType(), position=Location)
connected = Fluent('connected', BoolType(), l_from=Location, l_to=Location)
problem.add_fluent(robot_at, default_initial_value=False)
problem.add_fluent(connected, default_initial_value=False)

move = InstantaneousAction('move', l_from=Location, l_to=Location)
l_from, l_to = move.parameters()
move.add_precondition(And(robot_at(l_from), connected(l_from, l_to)))
move.add_effect(robot_at(l_from), False)
move.add_effect(robot_at(l_to), True)
problem.add_action(move)

problem.add_objects(locations.values())
problem.set_initial_value(robot_at(locations[INIT]), True)
for (f, t) in location_map.edges:
  problem.set_initial_value(connected(locations[str(f)], locations[str(t)]), True)
problem.add_goal(robot_at(locations[DEST]))

# We can now solve the problem with any planner installed supporting this problem kind!
# The library can detect the used syntactical features and can filter suitable planners
# 'OneshotPlanner' is just one of the "Operation Modes" supported by the library!
with OneshotPlanner(problem_kind=problem.kind) as planner:
    result = planner.solve(problem)
    if result.status == PlanGenerationResultStatus.SOLVED_SATISFICING:
        print(f'{planner.name} found a plan.\n The plan is: {result.plan}')
    else:
        print('No plan found.')
```



This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 101016442. This material reflects only the views of the Consortium, and the Commission cannot be held responsible for any use that may be made of the information contained in it.