Ex. 1: **Plan deredundantization** (Implementation Challenge, 6(+6/ + 3) points)

Design and implement a plan improver in the Aquaplanning framework. The improver receives an input planning problem and an initial plan to the problem. It should then remove redundant actions from the given plan until a perfectly justified plan (a plan that has no redundant actions) is found. The plan improver is not allowed to reorder the actions or add new actions.

Use `aquaplanning.optimization.SimplePlanOptimizer` as a point of departure and change the implementation of the `improvePlan` method (which, currently, detects simple cycles in state space as a trivial example). Hand in your solution as a single java file `<Lastname>sPlanOptimizer.java`.

The best submission (which removes the most redundant actions in smallest time – highest removed actions per second) wins double the points, second best gets 3 extra points. See the `TestPlanOptimizer` class for testing and evaluating your solution.

Ex. 2: **SAT-based totally ordered HTN Planning** (Theory, 2 + 2 = 4 points)

a) Given a “finished” fully expanded hierarchical task network of depth \(d\) with \(c\) inner nodes (i.e. compound tasks) and \(p\) leaf nodes (i.e. primitive tasks / actions) leading to a total amount of \(t := p + c\) tasks, calculate the exact amount of encoding iterations that need to be done (i) for the Stack Machine Simulation (SMS) encoding and (ii) for the Tree-REX encoding until a plan corresponding to this task network can be found.

b) Find a shape of the task network for which (i) SMS is preferable / (ii) Tree-REX is preferable / (iii) the two encodings are even w.r.t. the amount of encoding iterations, or explain why this cannot happen.

Ex. 3: **Numeric RPG Domain** (Practice, 2 + 2 + 3 = 7 points)

Modify and enhance our RPG classical planning domain from the first exercise. Use numeric planning where it makes sense and test your model with Aquaplanning.

a) Our adventurer should be able to fight some monsters. Introduce a simple hitpoint mechanic and place some monsters in the dungeon. Obviously, the adventurer must not die and must fight on encountering a monster if he wants to proceed to the next room.

b) Place some health potions in the dungeon. They can be picked up by the hero given that there is still some space in his inventory, and they can be consumed at any time in order to heal some number of hitpoints.

c) Introduce offensive (e.g. sword) and defensive (e.g. shield) gear which can be picked up similar to potions. When carried, the damage the adventurer deals to monsters is multiplied by the strength of the weapon he uses, and the damage that is dealt to the adventurer is divided by the sum of the strength of his defensive items.

Ex. 4: **Temporal Planning** (Theory, 3 points)

Preconditions in durative actions may need to hold during the entire execution of the action (“over all” keyword). Sketch how these kinds of constraints may be properly incorporated into the temporal forward state-space search algorithm as presented in the lecture.