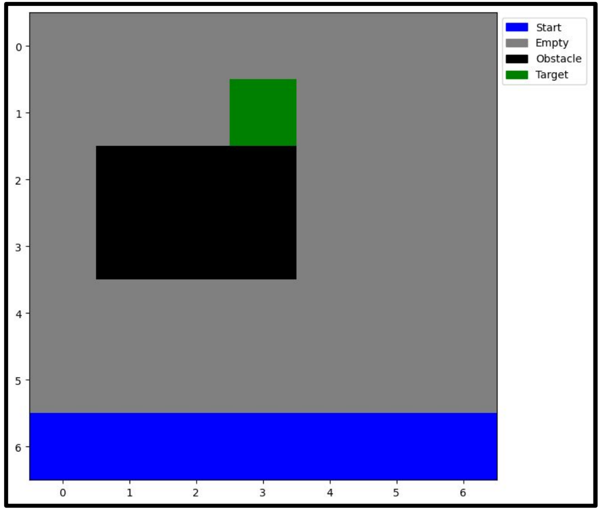
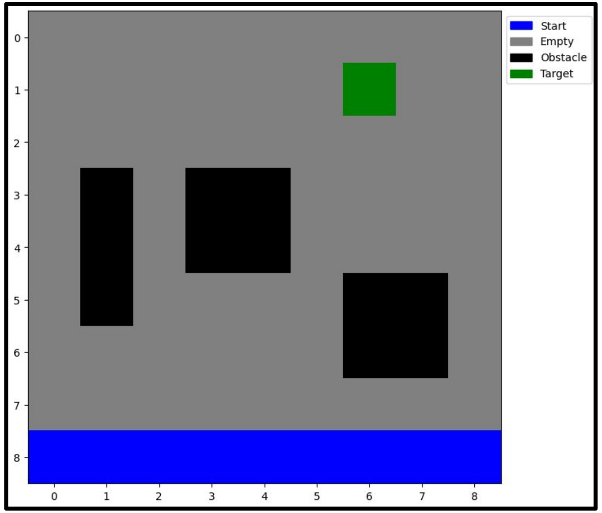
TECHNISCHE UNIVERSITÄT WIEN

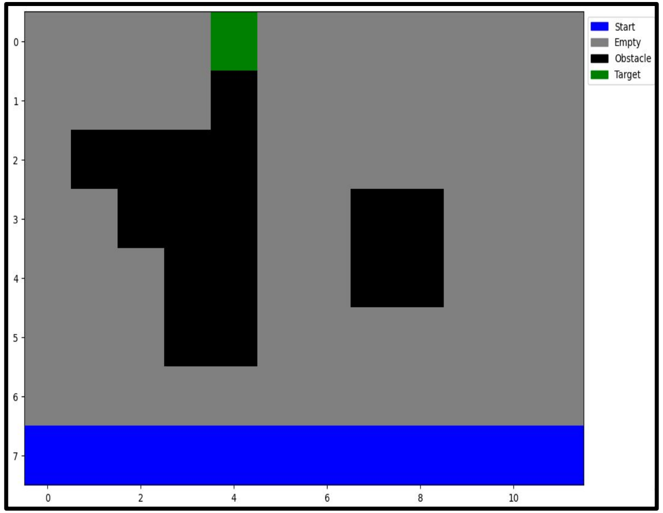
Exercise 3: Applying Reinforcement Learning for path finding

Selenge Tuvshin, Iftikhar Fakhar, Haider Tobias Abraham
Group 43

Grids







Easy - Grid **Medium - Grid** **Hard - Grid**

Agent initialization

- Search for the field set as the target and save it in a variable
- Define the allowed actions based on the max velocity value
- Initialize all potential states (fields within the grid that are no obstacles)
- Initialize all states with some values

```
def initialize_grid_variables(game_grid, target_field, obstacle_field, max_velocity):

    target = tuple(np.argwhere(game_grid == target_field)[0])

    actions = [(i, j) for i in range(-max_velocity, max_velocity + 1) for j in range(-max_velocity, max_velocity + 1) if (i, j) != (0, 0)]

    states = [(i, j) for i in range(game_grid.shape[0]) for j in range(game_grid.shape[1]) if game_grid[i, j] != obstacle_field]

    state_values = {state: 0 for state in states}
    state_visits = {state: 1 for state in states}  # to prevent division by zero

    return target, actions, states, state_values, state_visits
```

Episode generation

- An episode represents one path of a robot which leads to the target field. This function calculates infinitely many random actions put together in a sequence until the sum of all actions are the coordinates of the target field.
- Following rules are added to reduce runtime and errors:
  - No action can result in the robot leaving the grid
  - The robot can not move to an obstacle
- In case of the robot having no option to move (crashing into an obstacle because of too high velocity), the robot is put back in the starting position. The sequence continues from there, leading to a high error for the actions in this episode.

Episode generation

```python
def generate_episode(grid, actions, start):
    episode = []

    state = start
    action = [0, 0]
    episode.append((state, action))

    while grid[state] != target_field:
        action_options = [
            next_action for next_action in actions
            if abs(action[0] - next_action[0]) <= 1 # not crossing left border
            and abs(action[1] - next_action[1]) <= 1 # not crossing top border
            and 0 <= state[0] + next_action[0] < grid.shape[0] # not crossing right border
            and 0 <= state[1] + next_action[1] < grid.shape[1] # not crossing bottom border
            and grid[tuple(np.add(state, next_action))] != obstacle_field # no obstacel
        ]
        if (len(action_options) > 0):
            # pick any available action
            action = action_options[np.random.randint(len(action_options))]
            next_state = tuple(np.add(state, action))
        else:
            # start from starting point
            action = [0, 0]
            next_state = start

        episode.append((state, action))
        state = next_state

    return episode
```

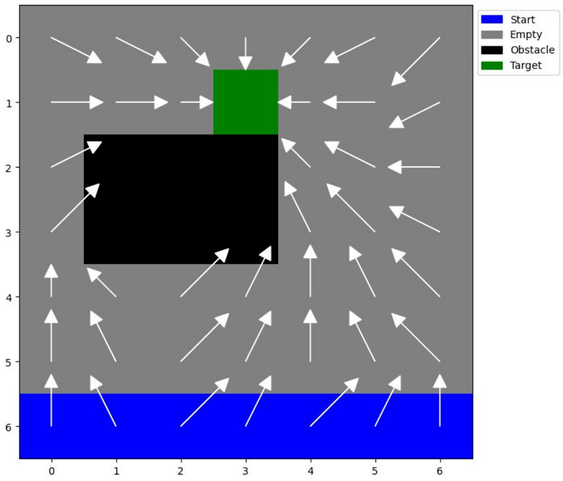
Training: Monte Carlo Control

- Generate episodes and determine state values
- Setup policy following leading to paths of highest state values

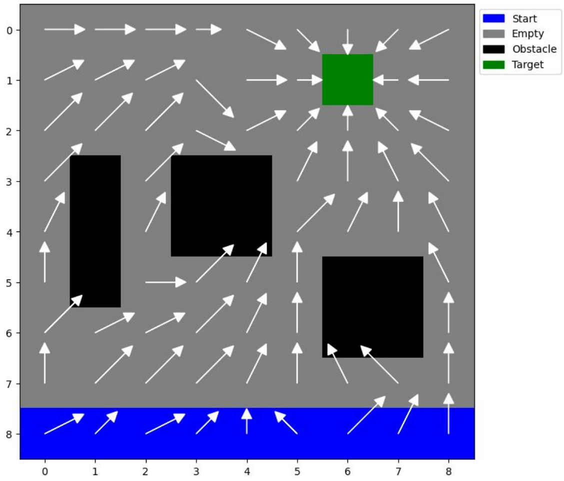
```python
def monte_carlo_control(game_grid, actions, states, state_values, state_visits, start_field, target, obstacle_field, gamma, n_episodes, generate_episode):
    for _ in range(n_episodes):
        start_options = np.argwhere(game_grid == start_field)
        start = tuple(start_options[np.random.randint(start_options.shape[0])])

        episode = generate_episode(game_grid, actions, start)

        total_return = 0
        for state, action in reversed(episode):
            total_return = gamma * total_return - 1
            state_values[state] += total_return
            state_visits[state] += 1

    policy = {}
    for state in state_values.keys():
        if state != target:
            possible_actions = [a for a in actions if tuple(np.add(state, a)) in states and game_grid[tuple(np.add(state, a))] != obstacle_field]

            if possible_actions:  # Check if there are any possible actions
                best_action_value = max(state_values.get(tuple(np.add(state, a)), float('-inf')) / state_visits.get(tuple(np.add(state, a)), 1) for a in possible_actions)
                best_actions = [a for a in possible_actions if state_values.get(tuple(np.add(state, a)), float('-inf')) / state_visits.get(tuple(np.add(state, a)), 1) == best_action_value]
                policy[state] = best_actions[np.random.randint(len(best_actions))]  # Break ties randomly

    return policy, state_values, state_visits
```

Results & Conclusion

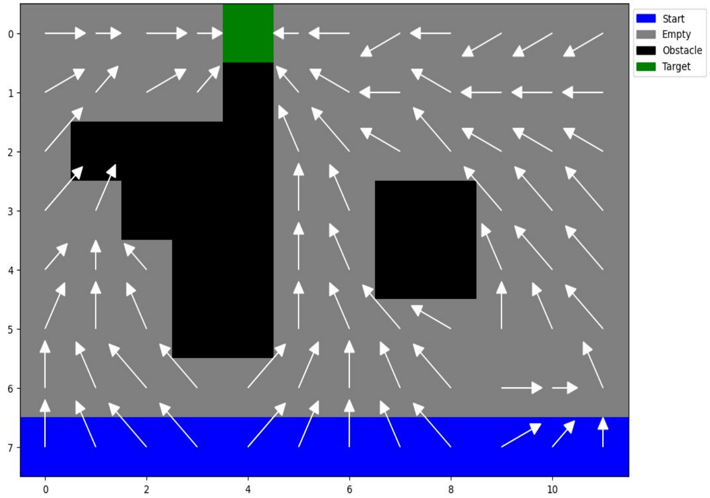
Policy Visualization



Easy - Grid

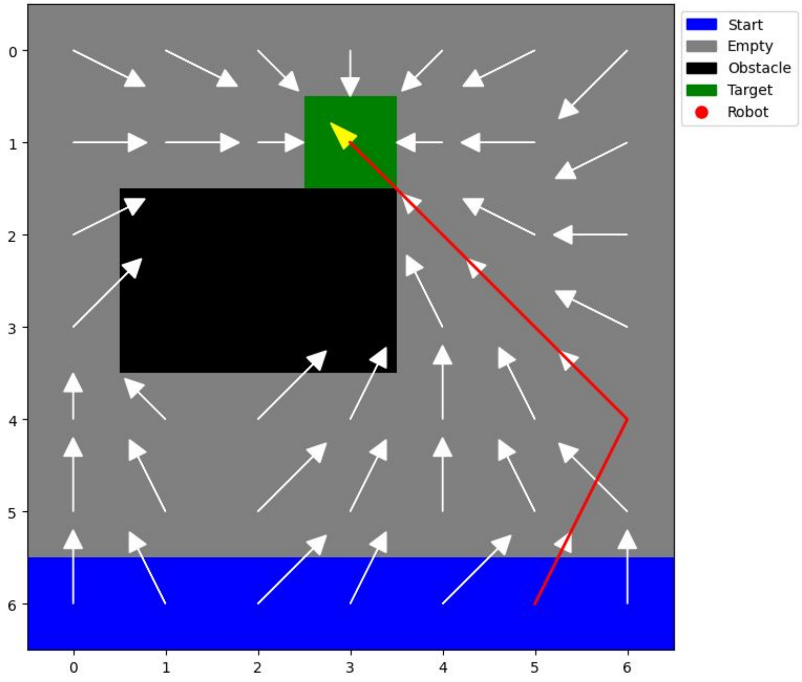


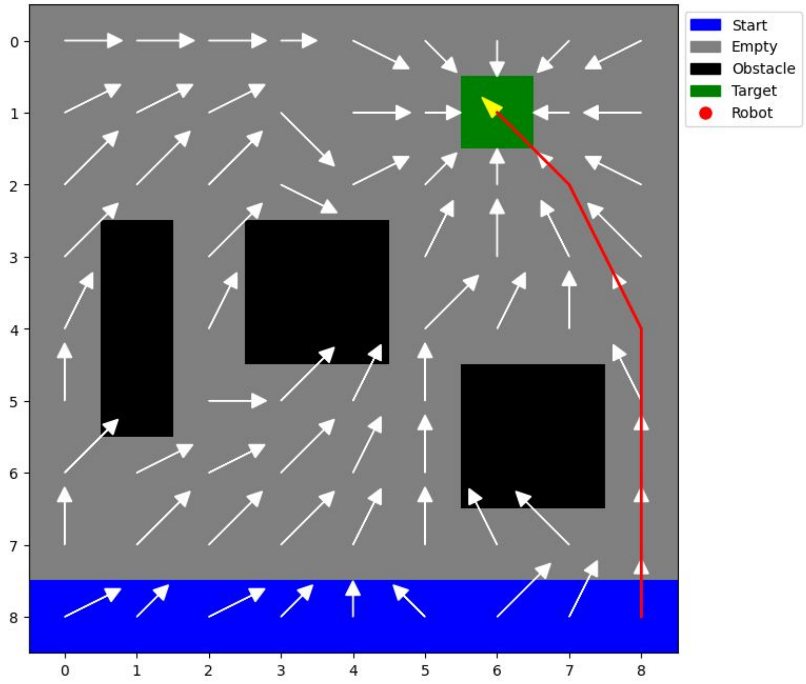
Medium - Grid



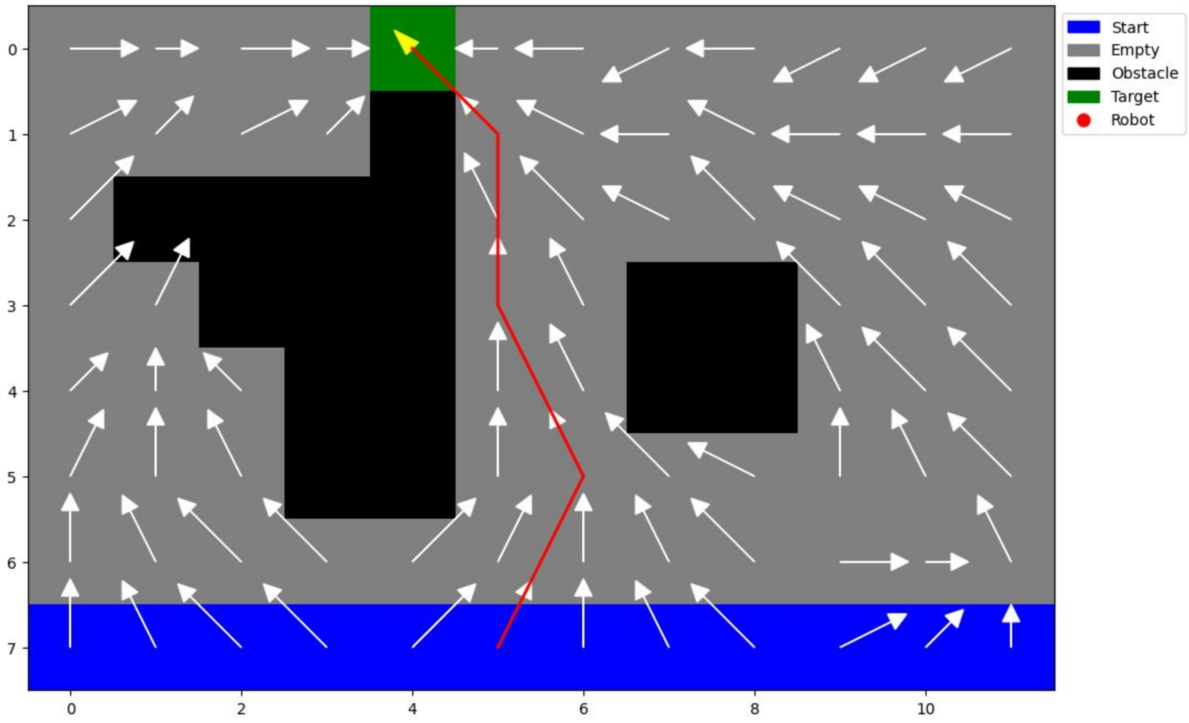
Hard - Grid

Optimal path for random initial start





Optimal path for random initial start



Thank you for your attention!