1. AI for the game of chess
MDP:

- Set of **States *S***: $S = \{0, ..., 12\}^{64}$, where each entry s_i of $s \in S$ represents a tile ($8 \cdot 8 = 64$ tiles) and its value defines what kind of piece occupies it. Values of 0 indicate an empty field, 1 to 6 the white chess pieces and 7 to 12 the black chess pieces. For example, if $s_5 = 1$, there is a white pawn on tile A5 and if $s_{13} = 8$, there is a black knight on tile B5 $(13 = 5 + 8)$.
- Set of **Actions *A***: $A = \{(i, j) \mid i, j \in \{1, ..., 64\}, i \neq j\}$ - an action is a pair of board coordinates i and j, that describes the action of moving any piece from position i to position j.
- **Probabilistic State Dynamics** p(s'|s,a, policy of the opponent):
    - Any **illegal move** (e.g. selecting a tile without a chess piece on it, selecting a tile with a chess piece of the opposing color, moving a pawn sideways etc.) is mapped back onto the same state s with a probability of 1 and the agent needs to pick a action again.
    - Any **legal move** results deterministically in a board configuration $\hat{s} \in S$. Depending on the opponent's policy $\pi^o(\cdot \mid \hat{s})$, we obtain a new state $s' \in S$ with probability $\pi^o(s' \mid \hat{s})$.
- **Reward Dynamics** $p(R_{t+1} \mid s, a)$:
    - Any **illegal move** will be penalized by e.g. -10
    - Any action resulting in **losing a chess piece** due to the opponent's action will give a negative reward depending on the role of the taken piece (queen gives higher penalty than a pawn) e.g. between -5 and -20
    - Any action that **takes an opponents chess piece** will get a reward dependent on the role of the taken piece (queen gives higher reward than a pawn) e.g. between +5 and +20
    - **Taking** the **opponent's king** or setting the opponent **checkmate** will win the game and yields a high reward, e.g. +100.
    - Checking the opponent will yield a positive reward e.g +10
    - By defining the rewards as above the agent is encouraged to learn what moves are actually allowed by himself, additionally to learning a good strategy.
- **Policy** $\pi(a|s)$: $\pi(a|s) = (p_{a1}, p_{a2}, ..., p_{a4032})$, where $\sum_{over\ all\ p} = 1, \forall s \in S, a \in A$

2. LunarLander MDP & Policy

MDP:

- Set of **States *S***: S = $\{ (x,\ y,\ rot,\ v_x,\ v_y,\ v_{rot},\ leftLegContact,\ rightLegContact) \mid x\ \in \mathbb{Z},\ y\ \in \mathbb{Z},\ rot\ \in [0,\ 2 * \pi),$ $v_x,\ v_y,\ v_{rot}\ \in\ \mathbb{R},\ leftLegContact,\ rightLegContact\ \in\ \{0, 1\} \}$
- Set of **Actions *A:*** (do nothing, fire left orientation engine, fire main engine, fire right orientation engine)
- **Probabilistic State Dynamics** p(s'|s,a) = Since we have infinite fuel, no action will fail. Therefore, all actions will lead to a follow up state s' directly following the physics of the environment with p(s'|s,a)=1 and p(t|s,a)=0 for all other states t.
- **Reward Dynamics** r(s,a) = [moving down ~ +100-140, zero speed ~ +100-140, moving away from landing pad = losing reward, coming to rest = +100, crashing = -100, firing engine = -0.3 per frame, solved = +200, leg ground contact = +10 per leg]
- **Policy** $\pi(a|s)$:
  $\pi(a|s)\ = (p_{do-nothing},\ p_{fire-left-engine},\ p_{fire-right-engine},\ p_{main-engine}),$
  where $\sum_{over\ all\ p}\ = 1, s\ \in\ S,\ a\ \in\ A$

3. Discuss the Policy Evaluation and Policy Iteration algorithms

Explain what the environment dynamics (i.e. reward function and state transition function) are and give at least two examples.

- State transition function: Provides the probability of arriving in state s' when performing action a in state s $p(s'|s,\ a)$.
  - E.g. in a self-coded grid world the probability of getting to the next state can be coded as 1, so there is no uncertainty about s', i.e. when the agent wants to move up from (0,0) it will always get to position (1,0).
  - E.g. in a 2-player game like go or chess, where the opponent's moves cannot be predicted, the future game state is probabilistic, therefore the state-transition-function is probabilistic and usually unknown.

- Reward function r(s,a): Provides the expected reward given the state a and action s
  - Again, in the self-coded grid world, the developer defines which states give which rewards, e.g. state (4,4) is a goal state and transitioning into it yields a reward of +100 and all other transitions might get penalized by a small amount of -1 to encourage taking the shortest route as possible.
  - In a 2-player game like go or chess, where the opponent's moves cannot be predicted, an action may result in a better or worse outcome depending on the opponent's reaction. Therefore, reward is probabilistic and the reward function returns the expected (mean) reward.

Discuss: Are the environment dynamics generally known and can practically be used to solve a problem with RL?

- If we have a controlled environment that is e.g. self-coded (like Gridworld), then probably yes.
- In an uncontrolled environment (e.g. letting a robot interact with the real world), having these is unlikely and can only be observed through interaction with the environment, i.e. an estimate can be obtained.