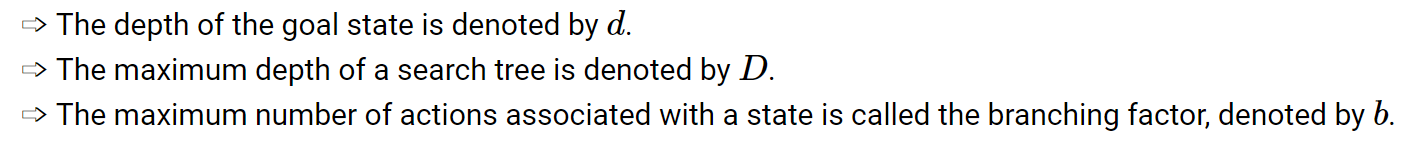
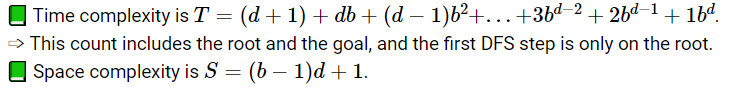
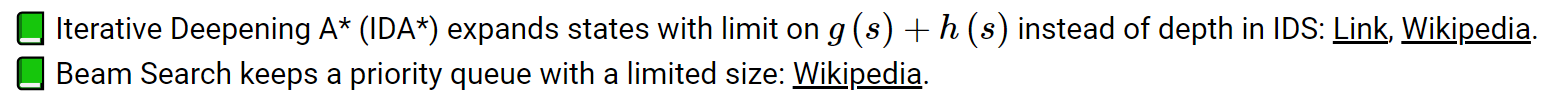
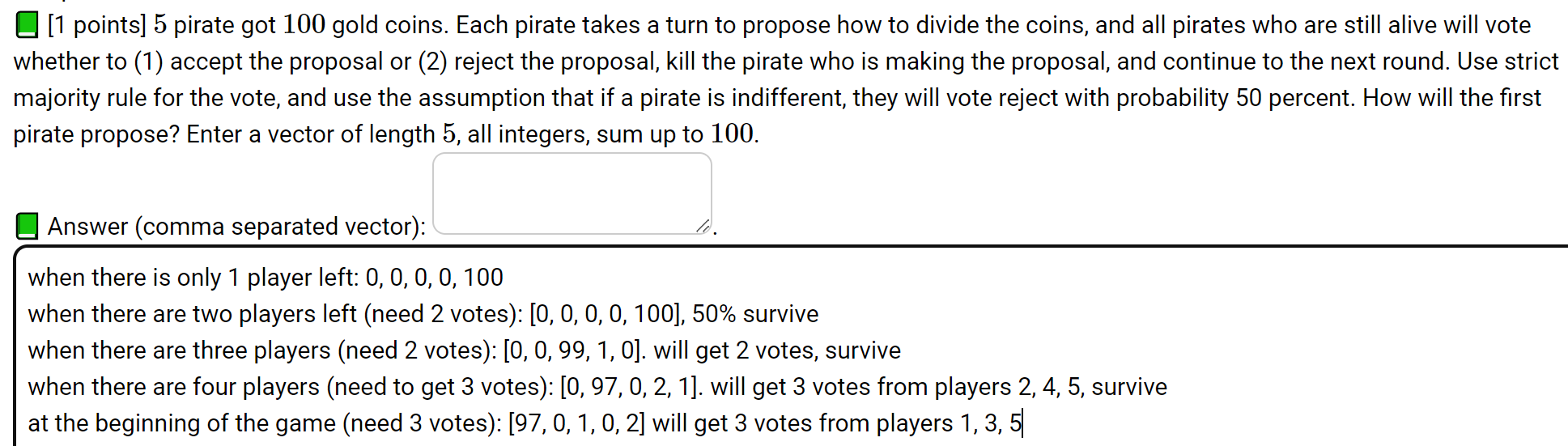
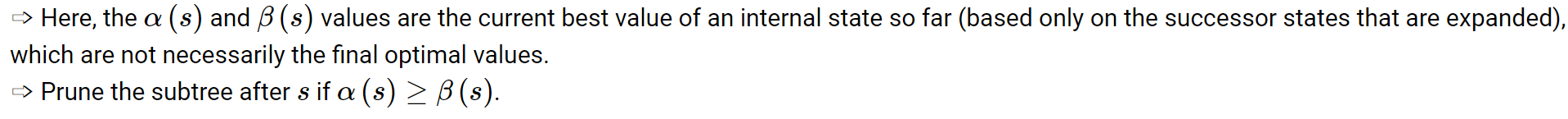
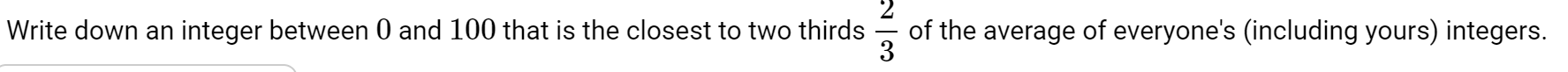
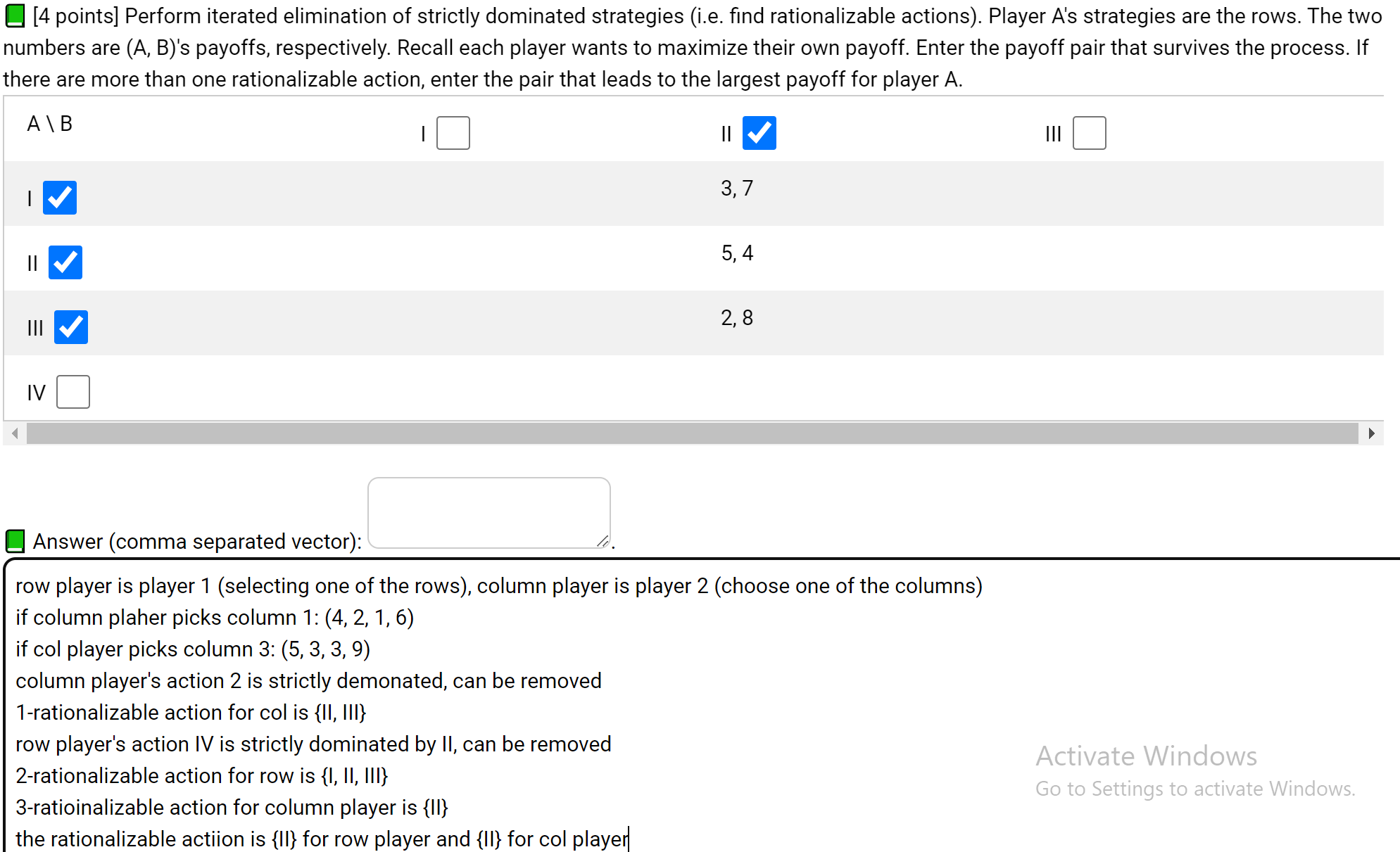
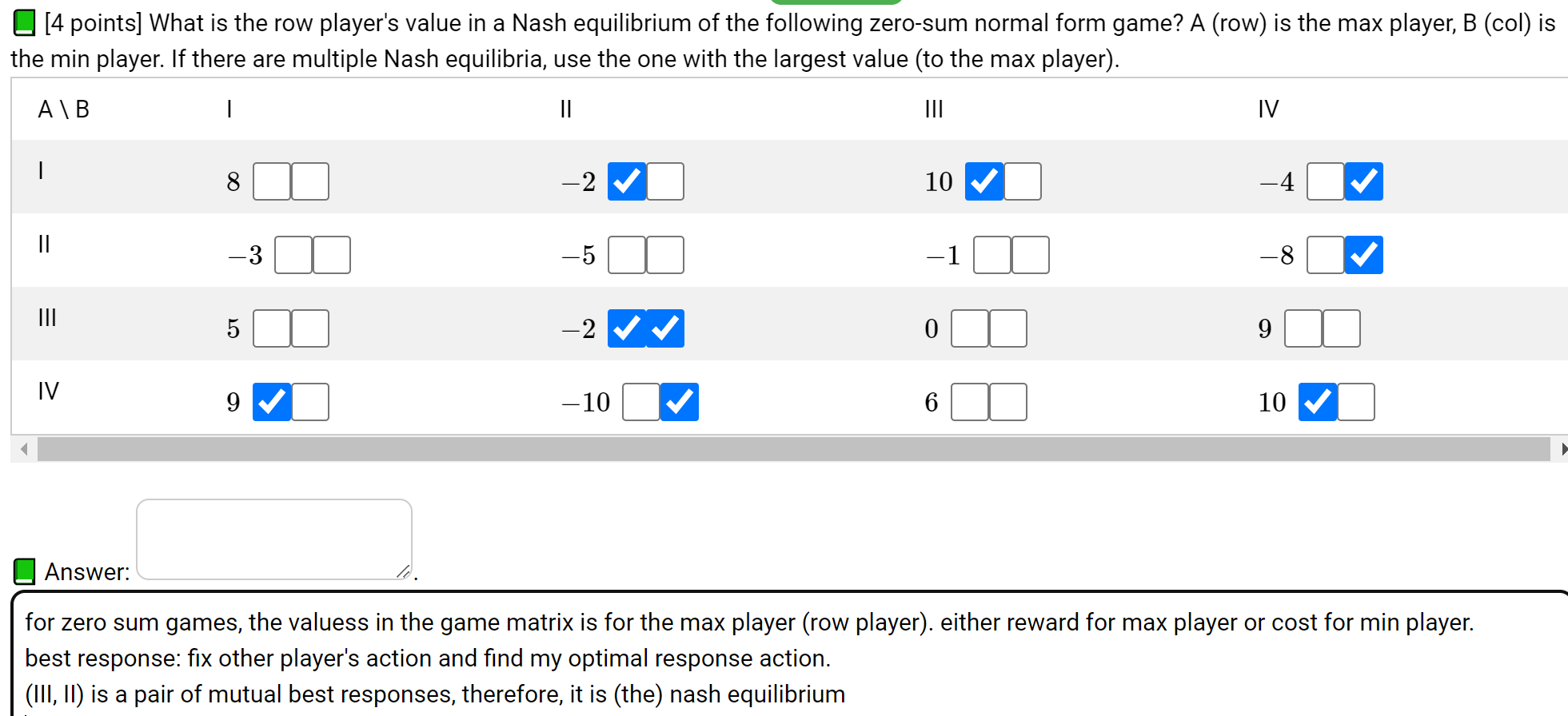
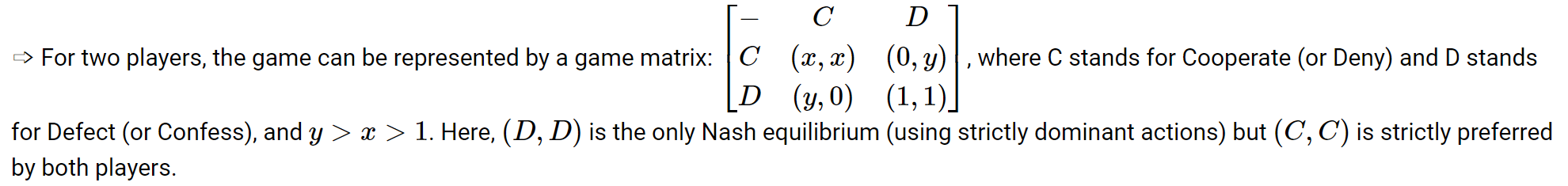
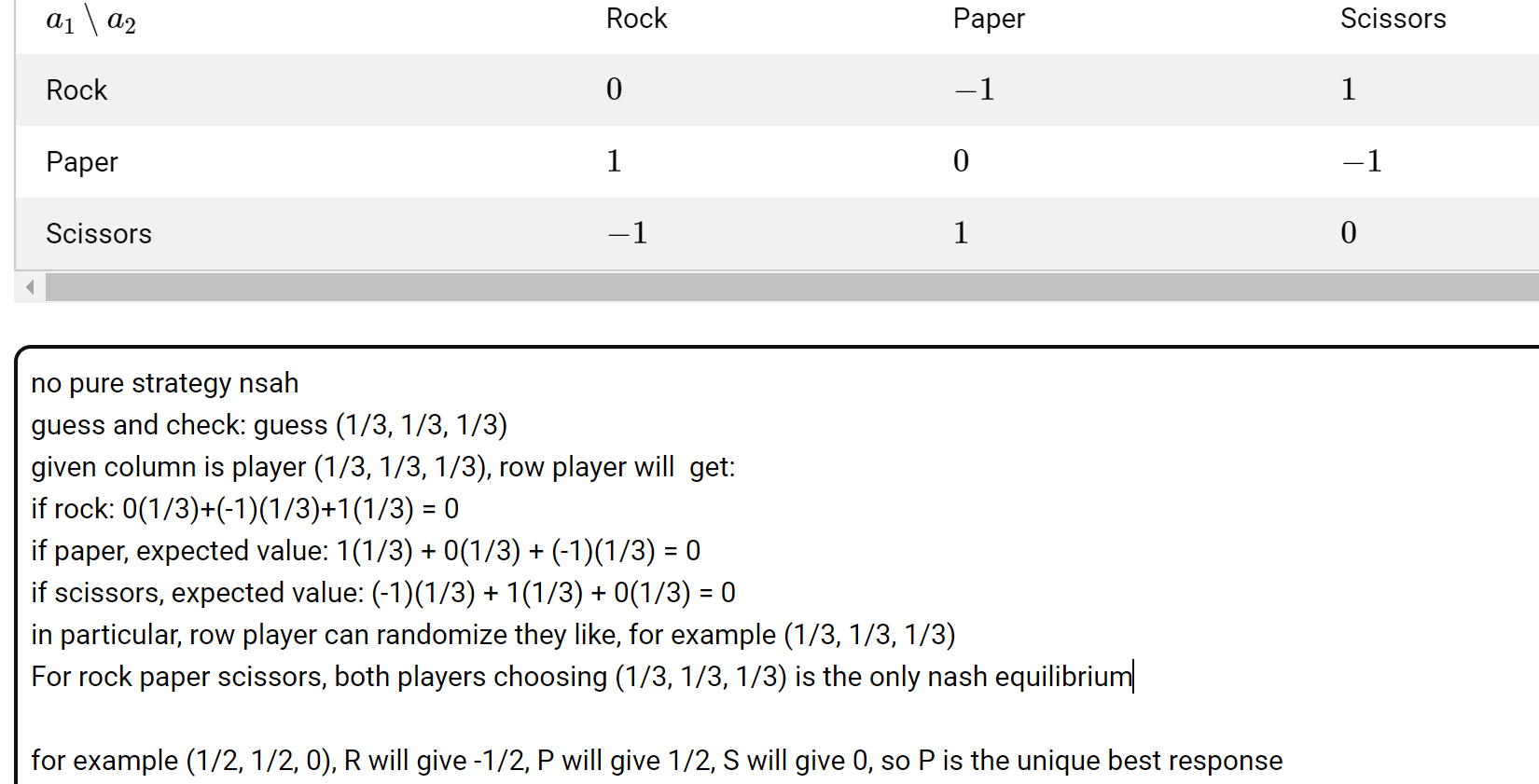
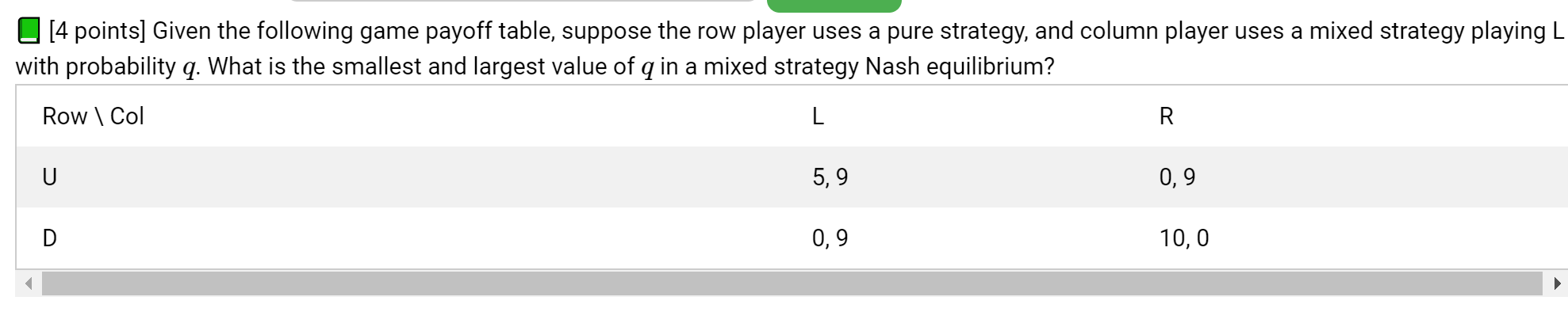
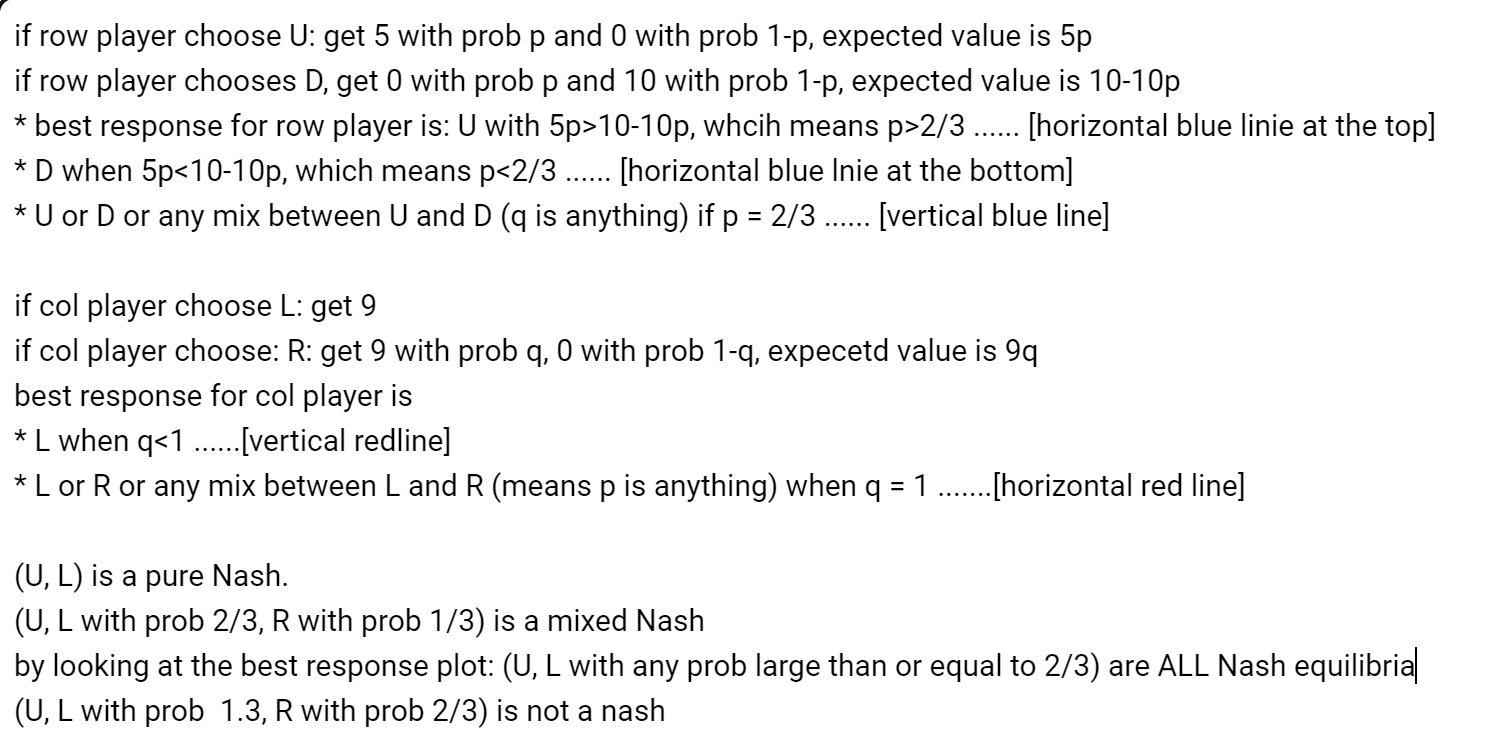
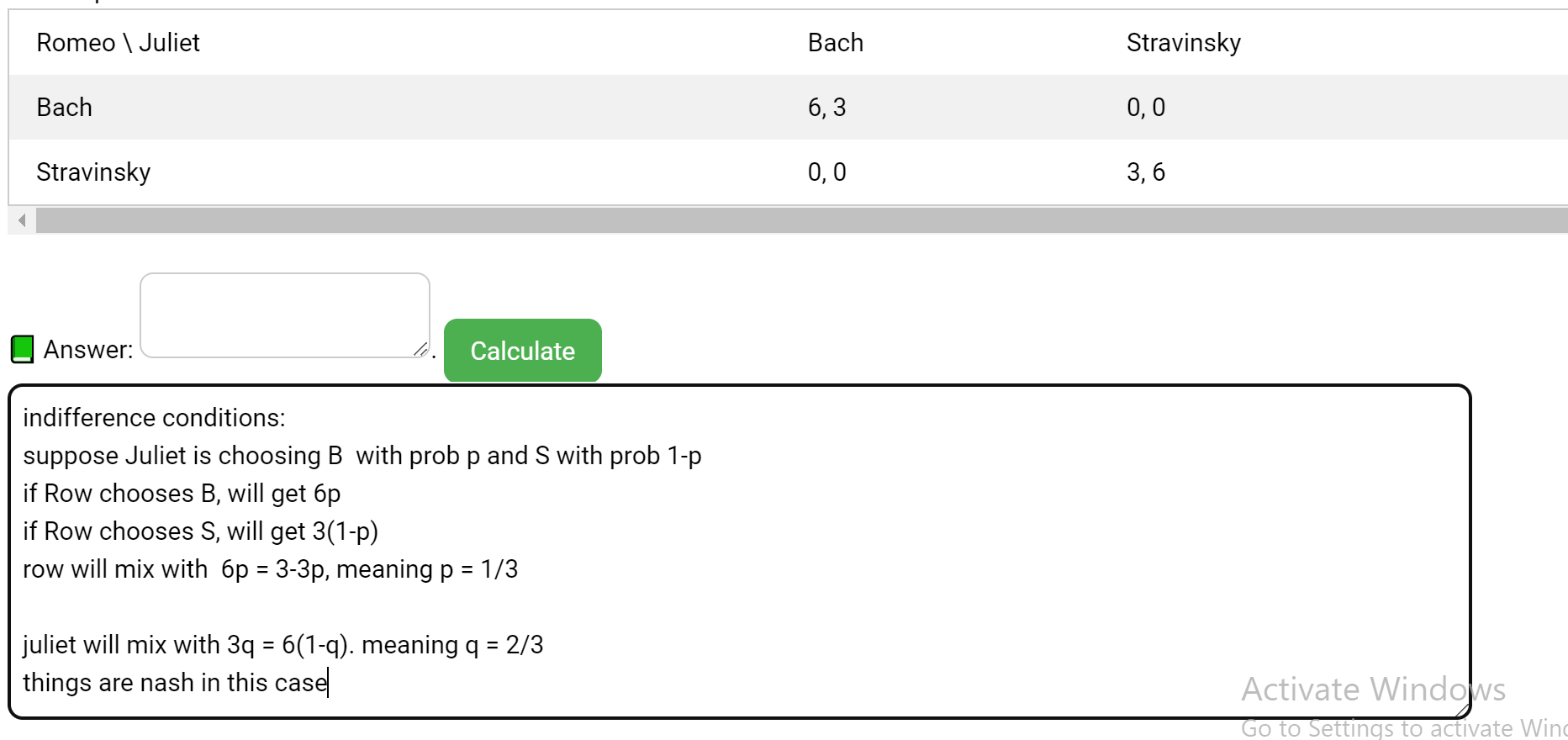
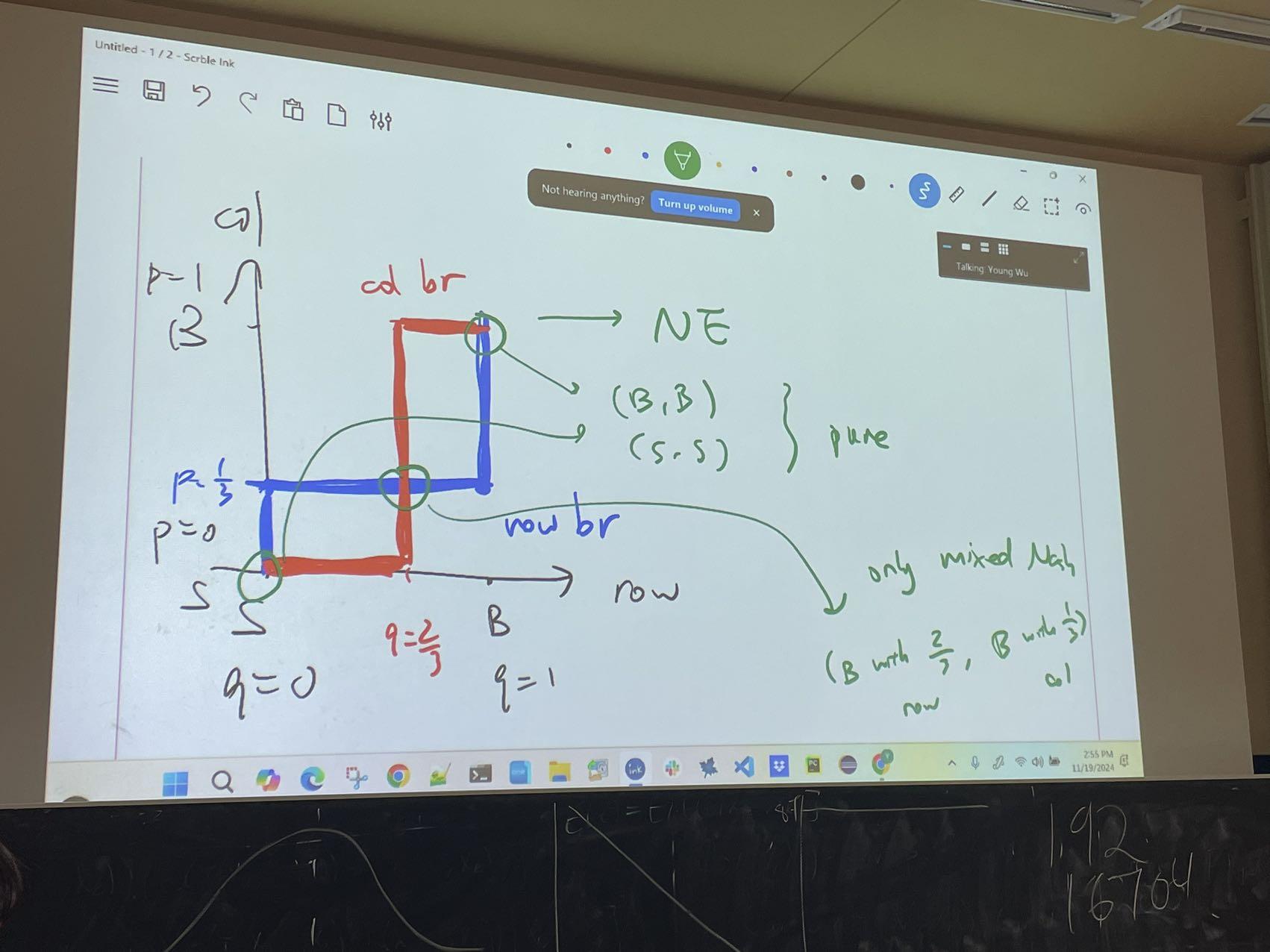
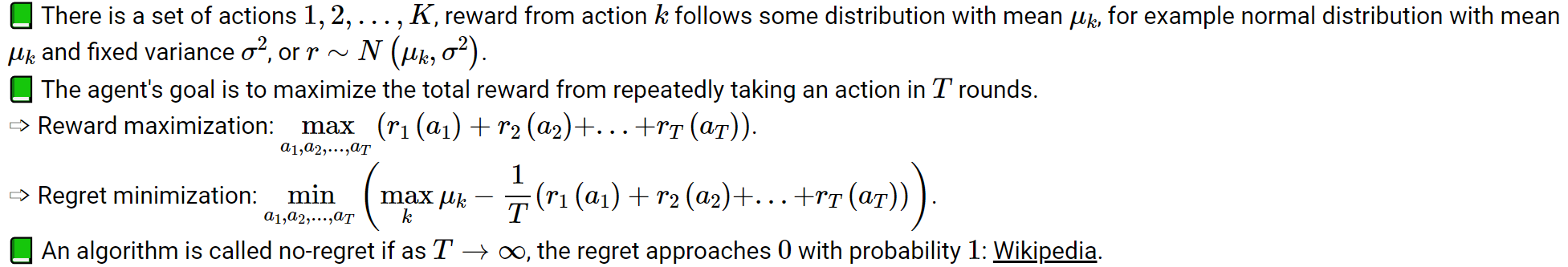
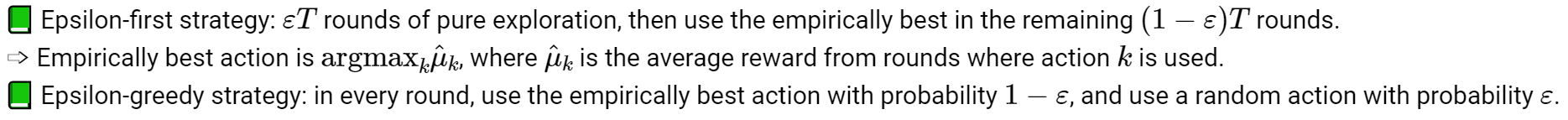
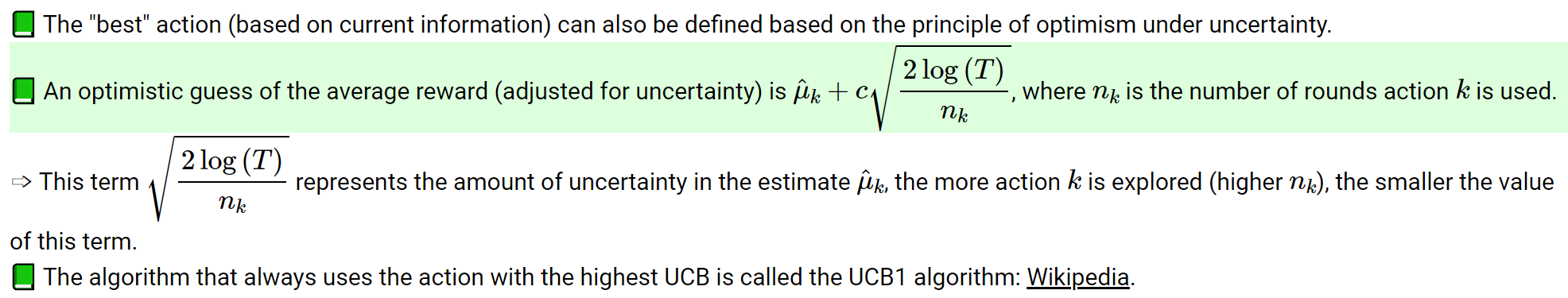
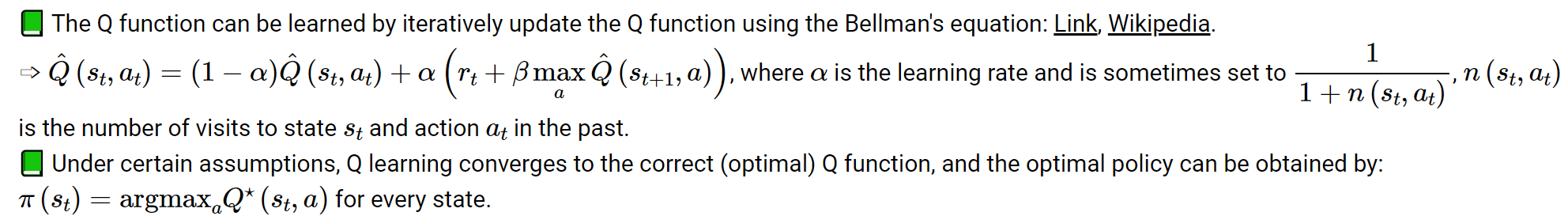
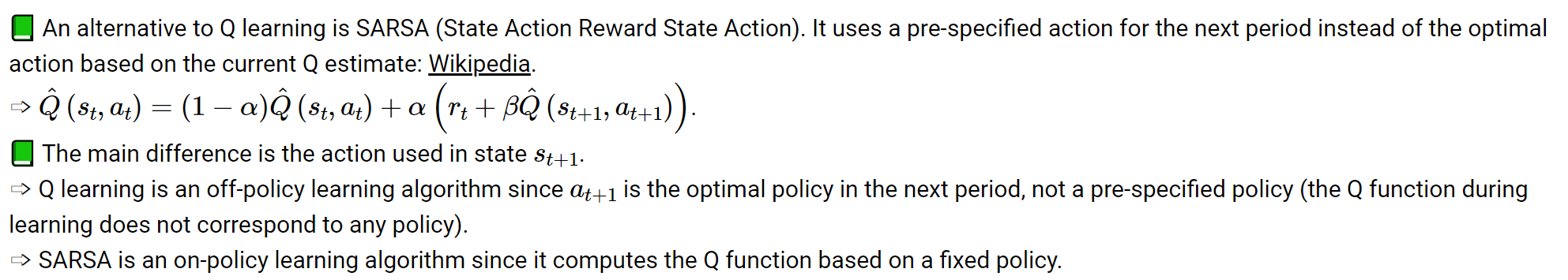
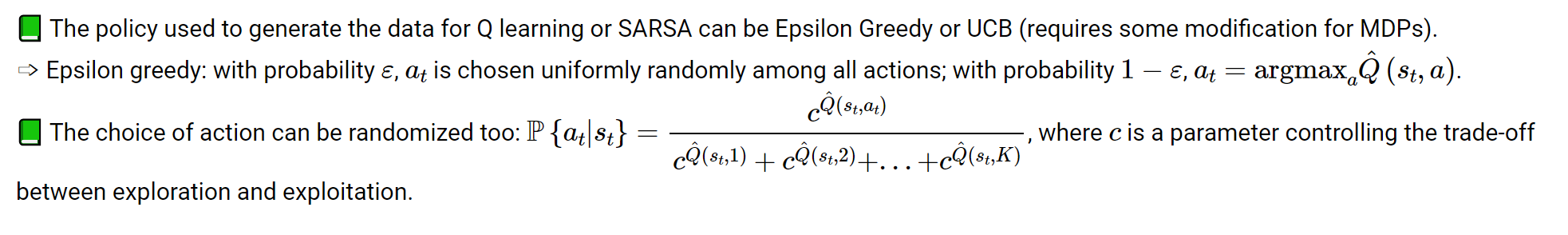
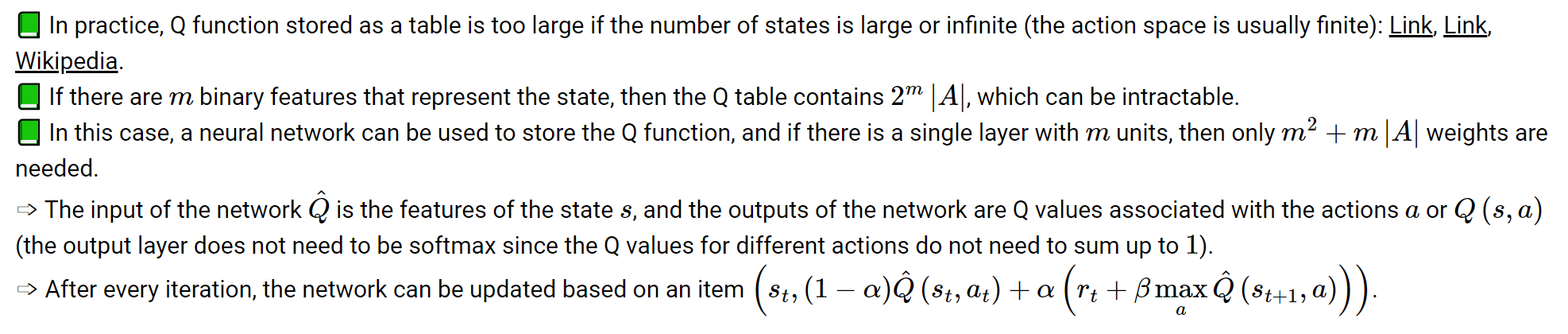
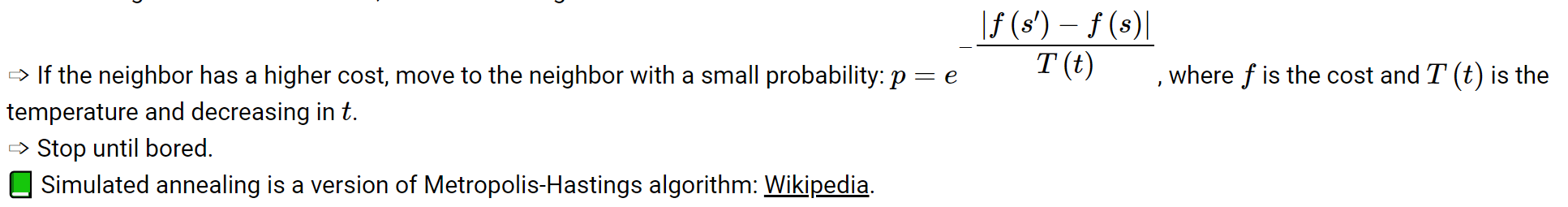
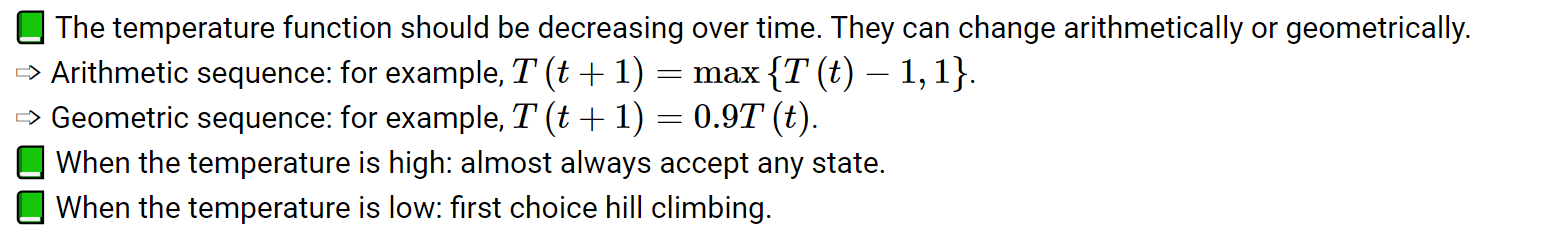
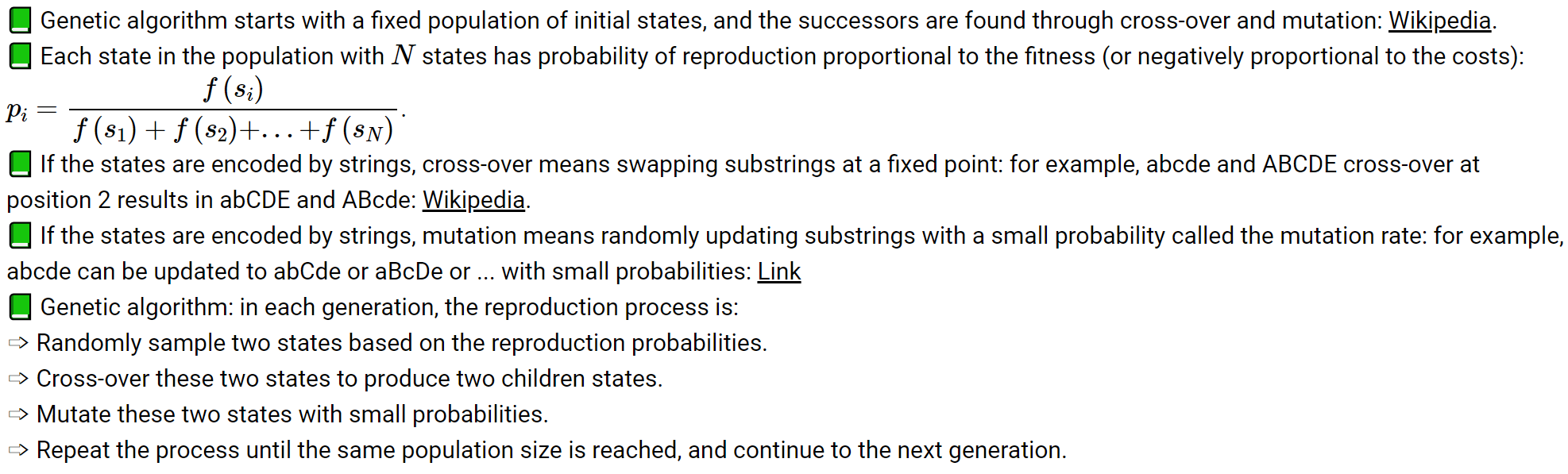
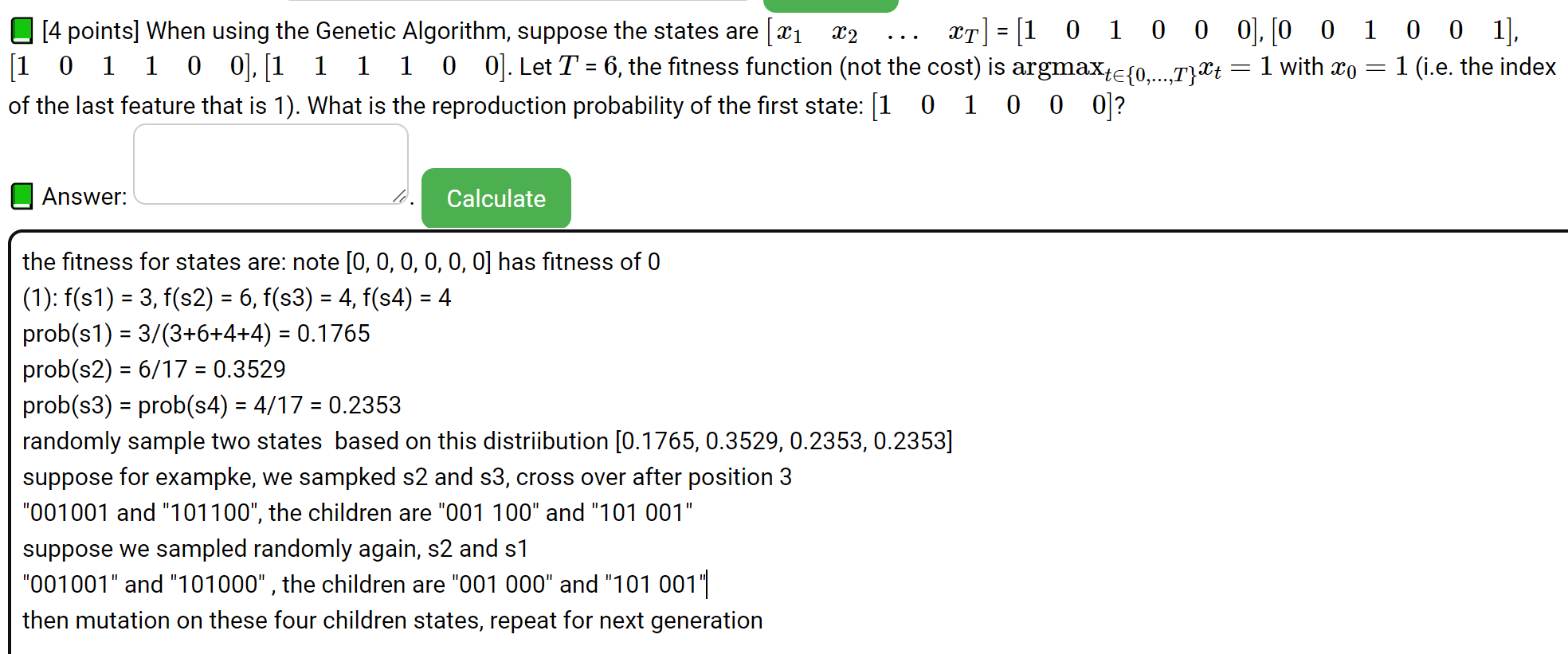
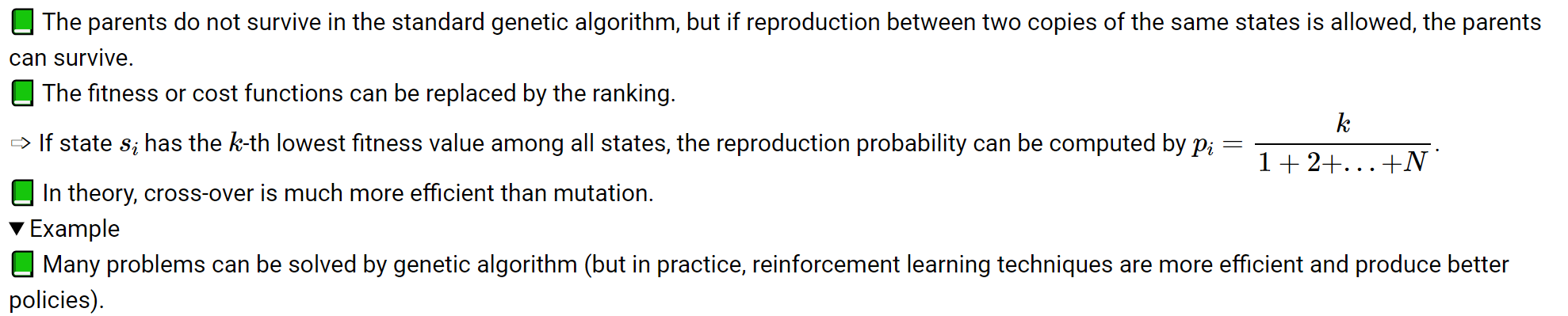
* States
  + Initial state
  + Successor state - state that can be reached by taking one action
  + Expanding the state - listing all successor states of some state s
* Search problem
  + Search problems solve for a sequence of optimal actions to minimize the total cost of these actions
  + Reinforcement learning is similar but the cost is unknown so the agent has learn the cost (usually in the form of reward) while finding the optimal actions
  + State spaces are usually huge so we do not want to search every state
* State space search
  + V - set of nodes - represents the states
  + E - set of edges - possible action at each state
  + c - cost/weights associated with each edge, assumed to be positive
  + Search strategy is the order in which the states are expanded
  + Frontier - leafs of a search tree
* Search performance
  + Search strategy is complete if it finds at least one solution
  + Search strategy is optimal if it finds the optimal solution
  + Compare time complexity (worst case)
  + Space complexity (worst case) maximum number of states stored in the frontier at a single time
  + 
* BFS
  + Convention - stop the search when the goal state is expanded, not when the goal is put in the frontier
  + T = 1+ b + b^2 + b^3 + … + b^d
  + S = b^(d+1)
  + Bidirectional search does BFS from the initial and goal states at the same time and stops when they meet
  + Complete
  + Optimal when the cost of every action is 1
* DFS
  + Incomplete if D = infinity
  + DFS is not optimal
  + Time complexity is T = 1 + b + b^2 + … + b^D - (b + b^2 + … + b^(D-d)) = 1+ b^(D-d+1) + b^(D-d+2) + … + b^D
  + S = (b-1) \* D + 1
  + What we want: optimal like BFS but low space complexity like DFS
* Iterative deepen search
  + IDS
  + Performs multiple DFS with increasing depth limits
  + DFS and stops if path length is larger than 1, larger than 2, …
  + IDS is complete
  + IDS is optimal when the cost of all action is 1
  + 
* Heuristic
  + Additional information can be given in the form of a heuristic cost from any state to the goal state: this is an estimate or guess of the minimum cost from s to a goal state
  + The cost from the initial state to a state s is denoted by g(s)
  + The cost from the state s to the goal state is denoted by h\*(s), which is unknown during the search
  + The estimated value of h\*(s) is called the heuristic cost, denoted by h(s)
* Uniform cost search
  + Which is Dijkstra’s algorithm
  + Expand the state with the lowest current cost g(s)
  + It is BFS with a priority queue based on g(s), and it is equivalent to BFS if the cost of every action is 1
  + Complete
  + Optimal
  + For this course, we remove repeated state with lower priority (higher cost)
    - But multiple expanding of one node is ok
  + Stop when the goal is expanded, not when it is in the queue
* Best first greedy search
  + Same as dijkstra except we use h(s) instead of g(s)
  + Expand the state with lowest heuristic cost h(s)
  + BFGS is not an abbreviation of greedy search, since it usually stands for Broyden Fletcher Goldfarb Shanno algorithm (a version of a gradient descent algorithm)
  + Greedy uses a priority queue based on h(s)
  + Greedy is incomplete
  + Greedy is not optimal
* A search
  + A search expands the state with the lowest total cost g(s) + h(s)
  + A search uses a priority queue on g(s) + h(s)
  + A is complete
  + A is not optimal
* Admissive heuristic
  + A heuristic is admissible if it never overestimates the true cost: 0<=h(s)<=h\*(s)
  + A search with an admissible heuristic is called A\* search
  + A\* is complete
  + A\* is optimal
  + Heuristic can be defined according to the scenario to be admissive
* Dominated heuristic
  + h2 dominates h1 (or h1 dominated by h2) if h1(s) <= h2(s) <= h\*(s) for every state s
  + A\* with a dominated heuristic is less informed and worse for A\*
* Some variants
  + 
* Adversarial search
  + The search problem for games (one or more players searching at the same time) is adversarial search
  + 
  + Indifferent means that if the rewards for each choice are the same, choose any of them
  + The initial state is the beginning of the game.
  + Each successor represents an action or a move in the game.
  + The goal states are called terminal states, the search problem tries to find the terminal state with the lowest cost (highest reward).
  + The opponents search at the same time and try to minimize their costs (or maximize their rewards).
    - For zero-sum games, the sum of the rewards and costs for the two players is 0 at every terminal state. The opponent minimizes the reward (or maximizes the cost) of the first agent
* Minimax algorithms
  + Dfs on the game tree
  + The game is solved backwards starting from the terminal states
  + Each player chooses the best action given the optimal action of all players in the subtrees
  + For zero-sum games, the optimal value at an internal state for the max player is called alpha(s) and for the min player is called beta(s)
  + The time and space complexity is the same as DFS with D = d
* Non-deterministic games
  + Some interval states represent movies by chance (can be viewed as another player), for example, dice roll or coin flip. For those states, expect castor reward are used instead of max or min
  + Dfs on games with chance is called expectiminimax
* Pruning
  + When a branch will not lead the current agent to win, the branch can be pruned (both players will not need to search the branch)
  + Alpha-beta pruning - dfs with pruning
  + 
  + If we know that the branch will choose the current option (which is worse than an existing good option) or another one that is even worse, prune the branch
  + Alpha is for the max player and beta is for the min player
* Static evaluation function
  + The heuristic to estimate the value at an internal state for games is called a static (board) evaluation (SBE) function
    - For zero-sum games, SBE for one player should be the negative of the SBE for the other player
    - At terminal states, SBE should agree with the cost or reward at that state
  + For chess, the SBE can be computed by a neural network based some features such as material, mobility, king center control, or a convolutional neural network treating the board as an image
  + IDS can be used with SBE
    - In iteration d, the depth s limited to d and SBE of the internal states at depth d are used as their cost or reward
* Monte carlo tree search
  + Random subgames ca be simulated by selecting random moves for both players
  + Win rate is calculated by large number of simulation, and among them calculate win/total games
  + The move corresponding to the highest expected reward (win rates) can be picked
    - The move corresponding to the highest optimistic estimate of the reward can be also picked
* Rationalizability
  + Unlike sequential games, for simultaneous move games, one player (agent) does not know the action taken by the other player.
  + Given the actions of the other players, the optimal action is called the best response.
  + An action is dominated if it is worse than another action given all actions of the other players.
    - For finite games (finite number players and finite number of actions), an action is dominated if and only if it is never the best response.
    - An action is strictly dominated if it is strictly worse than another action given all actions of the other players. A dominated action is weakly dominated if it is not strictly dominated.
  + Rationalizability (IESDS, Iterative Elimination of Strictly Dominated Strategies): iteratively remove the actions are that dominated (or never best responses for finite games)
  + 
    - 1-rationalizable: (actions that may be optimal, given other players are choosing valid actions)
      * 68, 69, 70, … 100 are not optimal no matter what the other players are choosing
      * 1-rationalizable actions: {0, 1, 2,.., 67}
    - 2-rationalizable: actions that may be optimal, given other players are choosing 1-rationalizable actions
      * 2-rationalizable actions: {0, 1, 2, …, 43, 44}
    - 3-rationalizable actions: {0, 1, 2, …, 30}
    - Infinity-rationalizable actions are called rationalizable actions: {0, 1}
  + 
* Nash equilibrium
  + If the actions are mutual best responses, the action forms a nash equilibrium
  + 
* Prisoner’s dilemma
  + A symmetric simultaneous move game is a prisoner's dilemma game if the Nash equilibrium (using strictly dominant actions) is strictly worse for all players than another outcome
  + 
  + (C, C) is preferred for both players, but (D, D) is the only Nash (rationalizable) actions
* Mixed strategy
  + When a player randomized between multiple actions
  + A pure strategy is when a player only use one action with probability 1
  + A mixed strategy nash equilibrium is a nash equilibrium for the game in which mixed strategies are allowed (called mixed extension of the original game)
  + 
  + 
  + 
  + 
  + If col is choosing B: expected reward is 3q + 0(1-q) = 3q
  + If col is choosing S: expected reward is 0q + 6(1-q) = 6-6q
  + Best response for col player is:
  + If 3q>6-6q or q>⅔, then best response is B
  + If q = ⅔, then best response is B or S or any mix
  + If q = ⅔, then best response is S
  + There are two pure Nash: (B, B), (S, S), there is only one mixed Nash, (B with prob ⅔ for row, B with prob ⅓ for col)
  + 
* Nash theorem - Every finite game has a (possibly mixed) Nash equilibrium
* Reinforcement learning
  + An agent interacts with an environment and receives a reward (or incurs a cost) based on the state of the environment and the agent's action
  + The goal of reinforcement learning is to maximize the cumulative reward by learning the optimal action in every state
  + Unlike search problems, the agent needs to learn the reward or cost function.
* Multi armed bandit problem
  + A simple reinforcement learning problem where the state does not change is called the multi-armed bandit
  + 
* Exploration vs exploitation
  + There is a trade-off between taking actions for exploration vs exploitation
    - Exploration: taking actions to get more information (e.g. figure out the expected reward from each action).
    - Exploitation: taking actions to get the highest rewards based on existing information (e.g. take the best action based on the current estimates of rewards).
  + 
* Upper confidence bound
  + 
* Markov decision process
  + States: s1, s2, … st (in the car example, it represents which square the car is on)
  + Action: a1, a2, …, at (in the car example, U, D, L, R, S)
  + Reward: R(st, at) is the reward you get from taking action at when you are in state st sometimes it depends on st+1 as well, realized random rewards are r1, … rt
  + Transition: s’ = st+1 = T(st, at) is which state you will go to in next period if you start at st and used action at
  + A markov decision process is a set of states, actions, reward and transition function
  + Goal: we are trying to find a policy function: pi(s) it takes in state s and output an optimal action a
* Value function
  + Optimal means we want to policy that leads to the highest (total, discounted, expected) value
  + V(s) = E[r1 + beta r2 + beta2 r3 + …], where beta is the discount factor
  + If the rewards are bounded by 1, then value is bounded by 1+beta+beta2+... = 1/(1-beta), assumes beta<1
  + In the 2 state 2 action example, beta = 0.8
    - States: A, B
    - Actions: stay, move
    - Rewards: R(A, stay) = 1, R(B, stay) = 2, R(A, move) = R(B, move) = 0
    - Transition function: T(A, stay) = A, T(B, stay) = B, T(A, move) = B, T(B, move) = A
    - Guess optimal policy is pi(A) = move, pi(B) = stay
    - Vpi(A) = 0 + 0.8\*2 + 0.82\*2 + 0.83\*2 + … = 2(1 + 0.8 + 0.82 + …) - 2 = 2/(1-0.8) - 2
    - Vpi(B) = 2+0.8\*2 + 0.82\*2 + … = 2/(1-0.8) = 10
    - Try another policy pi’(A) = pi’(B) = stay
    - Vpi’(A) = 1 + 0.8\*1 + 0.82\*1 + … = 1/(1-0.8) = 5 < 8, this policy is worse at state A compared to the previous policy
    - pi’’(A) = stay, pi’’(B) = move
* Q function
  + Value function given a specific action in the current period (and follows pi in future periods).
  + Qpi(s, a) = R(s, a) + beta \* Vpi(s’) = R(st, at) + beta\*R(st+1, pi\*(st+1)) + beta2R(st+2, pi\*(st+2)) + …
  + Q function is value you get from taking action a in the current period, and the action specified by pi starting from the next period
  + Then pi \* (s) = argmaxa Qpi(s, a)
* Q learning
  + 
  + Q learning is not learning the value function Q of our epsilon greedy policy, it is just converging to the optimal policy, therefore it is an off-policy RL algorithm
  + SARSA is an on-policy RL algorithm, it actually learns the Q of our epsilon greedy policy
* SARSA
  + 
* Exploration vs exploitation
  + 
* Deep Q learning
  + 
* Local search
  + For some problems, every state is a solution, only some states are better than other states specified by a cost function (sometimes score or reward)
  + The search strategy will go from state to state, but the path between states is not important
  + Local search assumes similar states have similar costs, and search through the state space by iteratively improving the cost to find an optimal state
  + The successor states are called neighbors (or move sets)
* Hill climbing
  + Discrete version of gradient descent
    - Starts at random state
    - Move to the best neighbor (successor) state
    - Stop when all neighbors are no better than the current state (local minimum)
  + Random restarts can be used to pick multiple random initial states and find the best local minimum (similar to neural network training)
  + If there are too many neighbors, first choice hill climbing randomly generates neighbors until a better neighbor is found
* Simulated annealing
  + Simulated annealing uses a process similar to heating solids (heating and slow cooling to toughen and reduce brittleness)
    - Each time, a random neighbor is generated
    - If the neighbor has a lower cost, move to the neighbor
  + 
* Temperature
  + 
* Genetic algorithm
  + 
  + 
  + Variants
    - 
* State representation of neural networks
  + A neural network can be represented by a sequence of weights (a signal state)
  + Two neural networks can swap a subset of weights (cross-over).
  + One neural network can randomly update a subset of weights with small probability (mutation).
  + Genetic algorithms can be used to train neural networks to perform reinforcement learning tasks.