1.)

Activity	**Performance**	**Environment**	**Actuators**	**Sensors**
Playing a tennis match	Accurate and powerful hits, quick movement, rapid decision making, strategic plays, points won, games won, sets won, matches won, minimal injuries	Tennis court, net, ball, opponent, conditions	Sight, body control (wrist, arm, back, feet), speech, movement	Camera, computer vision system, servos and motors, gyroscope, accelerometer
Bidding on an item an auction	Item purchased, price minimized, no purchase over budget	Auction hall, other bidders, auctioneer, current price, timer	Bidding paddle or voice or hand, vision, basic math	Speaker or motor, camera or microphone, user input, math processor

1. Playing a tennis match:
   - Partially observable.
     - Agents can detect all details of ball motion and game state
     - Cannot tell what the opponent is thinking
   - Multiagent competitive
     - Two players minimizing the performance of the other
   - Nondeterministic
     - Cannot determine what opponent does at next hit
     - Cannot determine how wind or rain will affect game in future
     - Equipment malfunction could occur
     - No reliable way to determine probabilities of events
   - Sequential dynamic
     - Decision on where/ how hard/ how to hit ball influences next opponent turn and thus next player turn
     - Ball constantly moving, serve clock ticking
   - Continuous
     - No way to model all possible states
       - Wind, altitude, court condition, opponent position, agent position, point, set, game, match, and more are all factors
   - Unknown
     - Agent understands physics (ball movement, how to hit ball)
     - Agent does not know how the opponent plays

2. Bidding on an item at an auction
   - Partially observable
     - Can see current price, budget
     - Cannot see other bidders' intentions
   - Multiagent competitive
     - Only one person can win the auction

        - Many people at auctions
- Deterministic
    - Price is only affected by the actions of bidders
- Sequential
    - Bidding now increases the price later
- Dynamic
    - Other agents can bid at any time
- Continuous
    - Behavior of other bidders influences optimal performance
        - Cannot model all possible behaviors
- Unknown
    - Agent has to learn behavior of other agents

The performance of a tennis match can be measured fairly well without any modifications. Points scored, games won, sets won, and matches won all serve as metrics for how well the agent is performing relative to the opponent. Objective performance could be measured by measuring how difficult each shot the agent takes is to hit. This could be evaluated by speed, spin, impact direction and angle, and opponent position.

The performance of bidding on an item at an auction is harder to measure. One marker of success is the percentage of wanted items won. However, such a marker does not take into account if the agent is overpaying for the item. Thus, performance should also take into account what percentage of the budget the agent spent on items when they are won.

2.)
There are 9! or 362,880 possible states in the eight puzzle, assuming it is possible for all tiles to be in any position in any game state.

3.)
a.
No, because there are certain game states that are theoretically possible, but are nearly impossible to achieve, so there is not a way to calculate them. Accounting for what pieces are captured by what other pieces and when is incredibly challenging because chess is a sequential game.

b.
The state space of chess can be estimated with a branching factor formula. Assuming that at each turn a player can take 30 moves, and assuming an average chess game takes 80 turns, we can calculate that there are 30^{80} possible positions in chess. To do this, I asked ChatGPT “how can i approximate the state space of chess” and followed its recommendation to calculate using the branching factor. I also used its approximations for possible moves in a turn and total turns per game.

4.)

a.
There are x * y - (area of shapes) possible states. This is because the robot can exist in any spot not in the shapes. There are infinite paths to the goal because the robot can backtrack and loop around as many times as it wants before reaching the goal.

b.
The shortest path between any two polygon segments must consist of straight-line segments between vertices because the shortest path between any two points follows the tangents of any obstacles in the way. With polygons, the tangent points would be vertices, and there will be no curves because none of the shapes are curved. Using the branching factor formula again, I can estimate the state space of the new problem. The shortest path I could find hit 5 vertices, and the potential longest path would hit all 33 vertices, so we can assume the average path is (33 + 5) / 2 = 19 vertices. Assuming at each vertex there are on average 5 other vertices to move to, we can calculate a state space of 5^{19} = 1.9E13.

5, 6.)
Changes from HW1:

- Moved main function to EightPuzzle file
- Updated readme to be more descriptive
- Fixed scrambleState to make n valid moves starting from solved state
    - If scrambleState makes an invalid move, it tries again
    - Good enough efficiency and simple
- The move function returns whether the move was successful
- Output of move handled in cmd function, so scramble doesn't fill command line
- Added more tests for new functions
- Added test for invalid move direction

BFS design:

- I created a Node class so I could combine state, parent, and move direction in one structure
- I use a queue of Nodes to keep track of the frontier
- In my while loop condition I only have to check if the maxnodes count has been exceeded because without repeated state checking there are infinite states
- At the start of each loop I pop the queue and check if it is the goal state
    - If it is, I print the solution
    - If not, I add each possible move to the frontier in the specified order
- To recreate solutions, I start with the solution node and follow through parent nodes until I reach the start
    - This traces the solution in reverse order, so I put moves in a stack as I encounter them so I can print them forward order
- My design for adding nodes to the frontier is not ideal since I use 4 if statements. There is definitely a better implementation, but this homework is late enough already. Any implementation would ultimately use multiple if's, but I could use a loop.

DFS design:

- BFS but with a stack instead of a queue
- I should generalize this code so it isn’t duplicated for BFS and DFS

7.)

Added a print statement to print the line being read by the scanner before calling cmd function.