Which of these statements is/are incorrect?

- `Player` – a class for human Connect Four players
  - `next_move` asks the human player where to move
- `RandomPlayer` – a class for an *un*intelligent computer player
  - all fields and most methods are inherited from `Player`
  - `next_move` chooses at random from the non-full columns
- `AIPlayer` – a class for an "intelligent" computer player
  - most fields and most methods are inherited from `Player`
  - new fields for aspects of its strategy
  - `next_move` tries to determine the best move!

A. `RandomPlayer` and `AIPlayer` are superclasses of `Player`.
B. `RandomPlayer` and `AIPlayer` override the inherited `next_move`.
C. `RandomPlayer` does not need its own constructor.
D. more than one of the above are incorrect.

Which of these statements is/are incorrect?

- `Player` – a class for human Connect Four players
  - `next_move` asks the human player where to move
- `RandomPlayer` – a class for an *un*intelligent computer player
  - all fields and most methods are inherited from `Player`
  - `next_move` chooses at random from the non-full columns
- `AIPlayer` – a class for an "intelligent" computer player
  - most fields and most methods are inherited from `Player`
  - new fields for aspects of its strategy
  - `next_move` tries to determine the best move!

A. `RandomPlayer` and `AIPlayer` are ~~`superclasses`~~ `subclasses` of `Player`.
B. `RandomPlayer` and `AIPlayer` override the inherited `next_move`.
C. `RandomPlayer` does not need its own constructor.
D. more than one of the above are incorrect.

Final Project

- Worth 150 points
- ***Cannot*** be replaced by the final exam
- More room for creativity than a usual assignment
- Pair-optional
  - pairs *must* work together (e.g., in a Zoom meeting)
  - they must share the work equally
  - see the collaboration policy

State

- The idea of *state* is a foundational CS concept.
- It captures the *current context* of a given computation.
- For example: the state of a Connect Four game

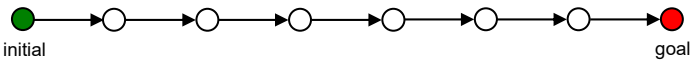
```
| | | | | | | |
| | | | | | | |
| | | |O| | | |
| | | |X| | | |
| | | |X| | | |
| | | |X|O| |O|
---------------
 0 1 2 3 4 5 6

Player X's turn
```

(plus more that isn't visible)

Solving Problems by Searching

- A wide range of problems can be formulated as *searches*.
- We search for a *sequence of actions* that will take us from an *initial state* to a *goal state.*



- Examples:
  - robot navigation
  - route finding
  - *many others*
- State space = all states reachable from the initial state by taking some sequence of actions.
- State-space search: search the state space for a solution!

Final Project: A Solver for Eight Puzzles

- A 3 x 3 grid with 8 sliding tiles and one “blank”
- Goal state:

	1	2
3	4	5
6	7	8

- Initial state: some other configuration of the tiles
  - example:

3	1	2
4		5
6	7	8

- Need to find a sequence of moves to get to the goal state.

3	1	2
4		5
6	7	8

⇨ slide the 4 right

3	1	2
	4	5
6	7	8

⇨ slide the 3 down

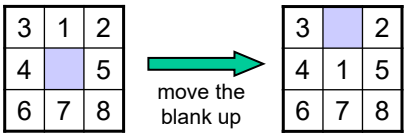
	1	2
3	4	5
6	7	8

Formulating a Search Problem

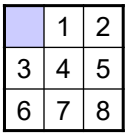
- In general, each search problem needs:
  1. an *initial state*
  2. the *moves:* actions that take you from one state to another
  3. a *goal test:* how to determine if a state is a goal state
  4. the *costs* associated with applying the moves
     - so we can find the lowest-cost solution

Eight-Puzzle Formulation

- *initial state:* some configuration of the tiles
- *moves*: it's easier if we focus on the blank
  - get only four moves
    - move the blank up
    - move the blank down
    - move the blank left
    - move the blank right



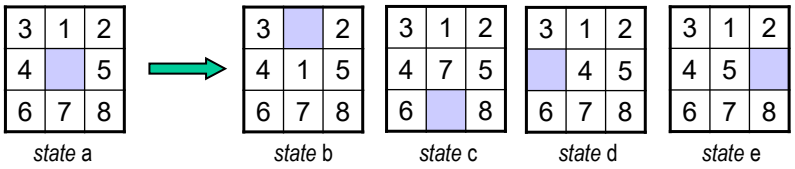
- *goal test*: simple equality test, because there's only one goal



- *costs:*
  - cost of each action/step = 1
  - lowest-cost solution = the one with the fewest moves

Relationships Between States

- A state's *successors* are all states that are one move away.



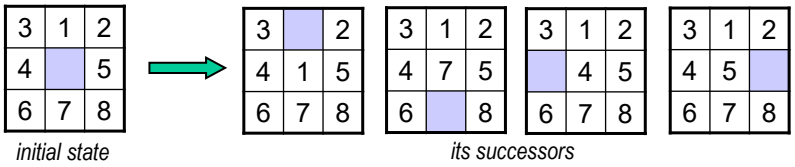
- In this example, state a's successors are states b, c, d, and e.
- Equivalently, we can say that state a is the *predecessor* of those other states.

Performing State-Space Search

- Basic idea:

  If the initial state is a goal state, return it.

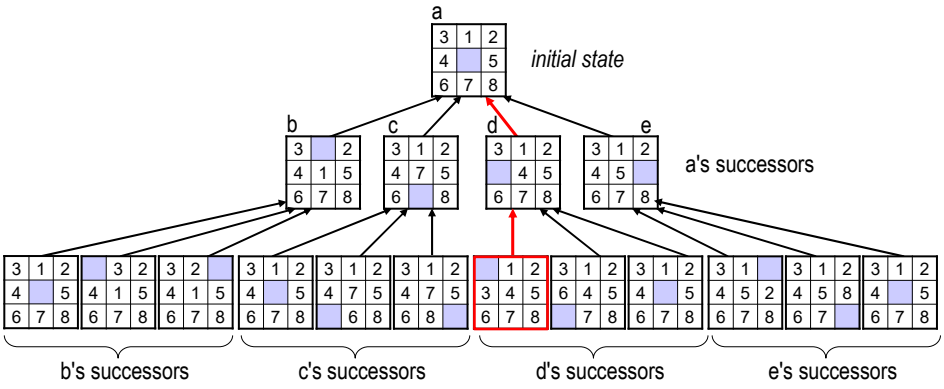
  If not, generate its successors.



  Consider the successors (and their successors…) until you find a goal state.

- Different search algorithms consider the states in different orders.

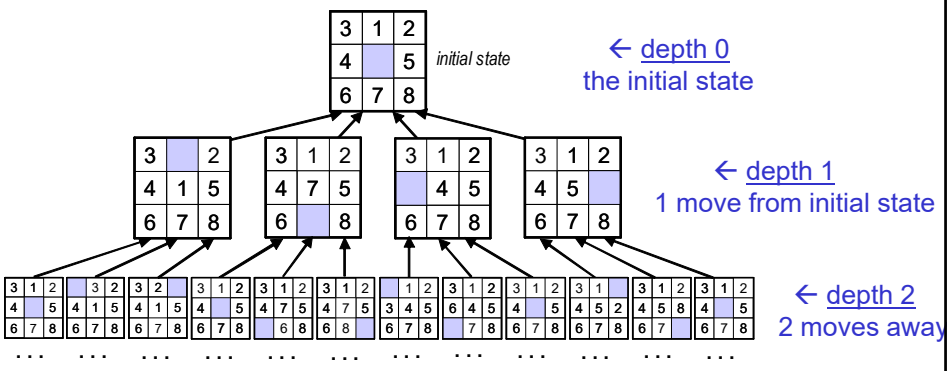
Performing State-Space Search (cont.)



- Once we find a goal state, what do we do?
- Each state maintains a reference to its predecessor.
- When we reach a goal, these references allow us to trace back and determine *the solution.*
  - the sequence of moves from the initial state to the goal

State-Space Search Tree

- The predecessor references connect the objects used to represent the states, creating a data structure known as a *tree*.
- The initial state is at the top of the tree.
- *depth* of a state S = the # of moves from the initial state to S



Search Algorithms

- The state-space search tree is built up gradually as we:
  - test states
  - generate their successor states
- Different search algorithms correspond to different ways of doing this.
- *More on this next time!*