# Actor

Superclass of all the items in the world, including Rocks, Flowers, Bugs, and anything you create.

**move()**

Move one square into the space the actor is facing. This method will crash if the actor is on the edge of the world facing out, or if the actor is not in the world.

**turn\_left(), turn\_right(), turn\_around()**

Turn in the direction indicated.

**is\_in\_world()-> bool**

Return true if the Actor is in the world.

**remove\_self\_from\_world()**

Remove this actor from the world.

**get\_world()-> World**

Returns a reference to the world if the Actor is in the world; otherwise null;

**can\_move()-> bool**

Returns true if the actor can move forward:

The actor is in the world

The space in front of the actor is in the world,

There are no actors in the space in front of the world**.**

**get\_i()-> int, get\_j()-> int, get\_direction()-> int**

Get the i, j coordinate or direction of the actor

**get\_next\_i()-> int, get\_next\_j()-> int**

Get the I, j coordinate next to the actor in the direction the actor is facing

**get\_neighbor()-> Actor**

Get the actor in the next space the actor is facing. It will return null if there is no actor there.

**act()**

Called by the framework, once each step. Subclasses should override this method.

# Rock (extends Actor)

A type of actor that does nothing.

# Flower (extends Actor)

A type of actor that does nothing.

# Bug (extends Actor)

A type of actor that moves forward 1 space each timestep if it can move. If it cannot move, it turns left.

# World

Holds the actors. and manages the time stepping. Each timestep, the world ***act*** method is called. It calls ***act*** on every actor it contains.

**add(actor: Actor, i: int, j: int, direction: int)-> bool**

Add the given actor at the i, j, location, facing in the specified direction. Direction can take one of the following values: **NORTH, WEST, SOUTH**, or **EAST**. The return value is True if the add was successful.

**is\_available(i: int, j: int)-> bool**

Return true if the specified location is in the world and unoccupied

**get\_actor(i: int, j: int)-> Actor | None**

Return the actor at the specified location, or None if there is no actor there

**get\_all\_actors()-> Actor[]**

Return a list containing all the actors in the world

**remove(actor: Actor)-> bool**

Remove the actor from the world. Return true if the removal was successful.

**move\_actor\_to(actor: Actor, i\_new: int, j\_new: int)-> bool**

Move the specified actor to a new location. Return true if the move was successful.

**get\_imax()-> int, get\_jmax()-> int**

Return the maximum i and j values for the world. Zero-based counting is used, with 0, 0 at the top left.

# 

# How to Solve Bugz Problems

1. Do NOT change World, Actor, Rock, Flower, or Bug classes!
2. Create your own class that extends the Bug class.
3. Override the ***act*** method in your derived Bug class to solve the problem. Add instance variables and methods to your class as necessary.
4. Rewrite the add\_actors method in the BugWorld class to create actors including your own derived Bug class.
5. Run the code and verify that it works as expected.

# List of Bugs

1. **SpinnerBug** - spins around without moving.
2. **BouncerBug** - moves forward one space each timestep until it is blocked or has reached the edge of the world. Then it turns around.
3. **MoonWalkerBug** - moves to the space behind it every timestep. If it is blocked, it turns right.
4. **TiredBug** - acts like a bug 10 timesteps and stops. Try adding the number of timesteps to move as a constructor argument.
5. **SquareBug** - moves in a square pattern. Note: the bug can only move one space each timestep. Do this with nothing else in the world. Now modify it so that it turns left when it is blocked.
6. **FastBug** - moves two spaces each timestep unless it is blocked. If it is blocked on either step, it turns left. Hint: how does this relate to the Bug act method?
7. **RandomBug** - moves forward until it is blocked, then randomly turns left, right, or around.
8. **RandomerBug** - when not blocked, goes forward 80% of the time, turns left 10% of the time, and turns right 10% of the time. When blocked, it randomly turns left or right with equal probability.
9. **DyingBug** - acts like a regular bug, but only lives 5 timesteps. Then it disappears from the world. For a more advanced implementation, pass in the number of timesteps the bug can act.
10. **SlothBug** - Acts like a bug, but only every fifth timestep.
11. **ThanosBug** - removes all other bugs in the world in the first timestep. After the first timestep, it acts like a bug. Now modify it so it removes only half all the other bugs in the first timestep.
12. **ThanosWannabeBug** - removes one other bug in the world each timestep, then acts like a bug.
13. **BeeBug** - if the space in front contains a flower, this bug “eats” the flower (removes it from the world). Then it behaves like a normal bug. Hint: you will need to use the *isinstance* method to determine if an actor is a Flower. (isinstance(actor, Flower))
14. **PopeyeBug** - Acts like a BeeBug, but once it eats a flower it moves like a bug twice each timestep for five timesteps.
15. **MutatorBug** - acts like a bug, but every timestep its appearance changes. Hint: you will need to dig into the superclass methods to see how bug images are set.
16. **HomingBug** - moves towards the space at the origin (the 0, 0 space). If blocked, turn left.
17. **ProgrammedBug** - takes a string passed into its constructor that controls it. The behavior of the bug at each time step is determined by the character in the string, read one time step at a time. For example, the string “flra” will cause the bug to go forward (“f”) the first timestep, then left (“l”) the second timestep, then right(“r”), the third timestep, then turn around (“a”) on the final timestep. A space means that the bug does nothing on that time step. When the string has been read through completely, start back at its beginning. NOTE: if the command is “f” but the bug cannot move forward, turn left instead.
18. **AnnihilatorBug** - destroys (removes from the world) any actor in its way.
19. **PredatorBug** - moves like a bug but eats other bugs that it encounters.
20. **HungryBeeBug** - Acts like a BeeBug, except that it must eat a flower every 10 timesteps, or it will die (remove itself from the world).
21. **HopperBug** - If it encounters a filled space, it will hop over it to the next space if it is empty.
22. **MemoryBug** - It behaves like a bug, but if it encounters a space, it has already visited, it will not move into it, but will turn right.
23. **GhostBug** - It behaves like a bug, but it is invisible. Hint: You will need to look at how Bug and its superclasses work to figure this one out. Thought: how will you test this?
24. **WarpBug** - each time step, it randomly reappears in an unoccupied space in the world.
25. **MinerBug** - Moves towards the nearest rock if one exists in the world. When it is next to the rock, it stops. If it is blocked on the way, it turns left. Hint: you will need to get all actors in the world, and determine which is a rock, and which rock is the closest.
26. **StalkerBug** - Constantly moves towards the nearest other bug in the world.
27. **PoopingBug** - Acts like a bug, except that it leaves a rock everywhere it has been.
28. **CloneBug** - acts like a bug, but every 5 timesteps, it creates its clone behind itself if it has moved forward.
29. **SiameseBug** - is two bugs next to each other. They both move forward until either one is blocked. If either is blocked, they both turn left.
30. **BorgBug -** all bugs in the world are in communication. They behave like a normal bug, but if one bug turns, all the bugs turn.
31. **ZombieBug** - acts like DyingBug, but after one time step out of the world, it reappears. Note: this one is slightly impossible. Maybe you could have another bug remember that this bug existed.
32. **InfectedBug** - behaves like a bug, but when it is next to another bug, it turns that bug into an InfectedBug.
33. **ExplodingBug** - behaves like a bug, except every step it “explodes” every actor immediately surrounding it.
34. **WarpBug** - each step it randomly moves to another unoccupied location in the world. Try this without any other actors in the world first. Then add in logic to avoid other actors.
35. **RadioactiveRock** - this is not a bug, but a rock. When any bug is next to this rock, it dies.
36. **SwapperBug** - if there is a bug in front of it, this bug swaps places with it. Otherwise, it acts like a bug.
37. **PoisonFlower** - this is not a bug, but a flower. When a BeeBug is about to eat it, the BeeBug dies.
38. **SniperBug** - behaves like a bug, but “snipes” all other bugs in same row or column not behind another actor.
39. **MedusaBug** - turns all neighbor bugs into Rocks.
40. **SpiritBug** - passes through all other on-bug actors as it moves. If it hits a boundary, it turns.
41. **PollinatorBug** - eats flowers in front of it. After it has eaten a certain number of flowers, it spawns a new flower.
42. **SmartBeeBug** - Flowers that are in contiguous spaces are grouped into clusters. The SmartBeeBug visits each cluster from the largest to the smallest and eats all the flowers in the cluster.