- How do you track the current direction and location of each drone spaceship?
    - Each Drone maintains its orientation and position internally as private variables orientation and position. Orientation is represented as a string, “n,” “e,” “w,” “s,” or a combination of two base orientations (e.g. “ne”) if the drone is orientated diagonally. Position is maintained using an internal Coordinates instance in each Drone which is updated whenever the Drone moves. If the Drone is destroyed, this is managed by SimulationSystem, which will delete the Drone from the collection of Drones, modeling the Drone being lost and unusable.
- How do you track the how much of the space region has been explored so far?
    - SimulationSystem keeps track of the space region explored so far with the integer spaceExplored. SpaceExplored is updated when a Drone performs the thrust action which calls the moveEntity() method in Region via the SimulationSystem. The returned MoveResult denotes that a new Space was explored by setting the newlyExplored flag to be true. In the performAction method, spaceExplored is iterated by 1 if newlyExplored is set in a returned MoveResult.
- How do you discover and track the locations of the suns?
    - The system discovers sun locations and tracks them using the Spaces, which contain coordinates and contents, some of which will be suns. The Drones use the SimulationSystem checkCoordinates() method, which models the collective Drone knowledge repository, to check what is known about a Space. In turn, this retrieves the Space from Region using getSpace(). The SimulationSystem will only return data to the Drone if the Space is known (explored or scanned) using the known flag in Space. If true, then the Drone will receive information about the Space, including if it is a sun or not. If false, the Drone will receive a Space with content “Unknown” and all flags set to false.
- How do you update the simulation state for thrust() and steer() actions?
    - Steer() is handled internally by the Drone, as it maintains its own orientation. When the Drone steers, it calls its steer() method with the orientation it has chosen, which update sthe internal orientation variable. Thrust takes in an integer between 1 and 3 and returns a DroneAction to the SimulationSystem via act(). The SimulationSystem calls moveEntity() in Region between 1 and 3 times, depending on the thrust strength, and resolves the consequences.
- How do you determine if a drone spaceship has crashed or if it’s still safe?
    - Between each moveEntity() call, the SimulationSystem checks the current Drone position against all other Drone positions for any collisions, against the newly arrived Space contents for a sun, and if the motion executed successfully, updates the Drone position using updatePosition(). This check is performed in the performAction() method. If a Drone has crashed, it is deleted from the Drones collection maintained by SimulationSystem.
- How do you determine the appropriate output for a scan() action?
    - When a Drone calls scan(), a DroneAction object is sent to SimulationSystem through act(), indicating the scanning position and the chosen action of “Scan”. In turn, the SimulationSystem calls the Region’s scanAroundCoordinates() method. ScanAroundCoordinates() takes in a scanning position (Coordinates) and returns a list of the contents surrounding the scanning position, starting from the North and going

clockwise. If any scanned squares fall outside of the region, “barrier” is added to the list. Region knows when a requested position is outside the region because height and/or width are greater than or equal to maxHeight and maxWidth, or height and/or width is negative. In these cases, a Space is not checked, “barrier” is immediately appended to the list, and the next position is scanned. Otherwise, the Space is checked for a Drone using containsDrone(). If true, “drone” is appended to the list. Finally, if the Space is still not processed, the contents of the Space are appended to the list using getContent().

- How do you determine which sections of the space region still need to be explored?
    - The knowledge of whether a Space has been explored or not is maintained by the explored flag on each Space. This is set to false by default when the simulation is started for each Space except for the ones that the Drones spawn on. When a Space is visited via the moveEntity() method, the flag is flipped to true, indicating that it has been explored. Thus, at any given time, all Spaces with explored set to false are unexplored and all Spaces with explored set to true have been explored. On a simulation level, there are still Spaces to be explored as long as spaceExplored is less than maxSpaceExplorable.
- How do you keep track of the “partial knowledge” star map collected by the drones?
    - Each Space has a Boolean flag named known. This is set to false when the Space is instantiated for all Spaces except for those that the Drones spawn on. When a Drone scans or visits a Space, the flag is flipped to true in scanAroundCoordinate(), which calls markAsKnown() internally. Drones have access to this partial knowledge through SimulationSystem’s checkCoordinates() method. When the Drone requests a coordinate check on a given set of coordinates, the SimulationSystem fetches the Space and checks if it is known or not. If it is known, SimulationSystem return the contents of that Space as a String. If the Space is not known, it will return “Unknown,” as the Drone star map does not have knowledge of the area. Thus, Drone communication is currently modeled as being perfect, with the SimulationSystem dispensing knowledge the Drones would know when asked for information, otherwise giving no information.
- How do you determine the next action for a drone?
    - Every turn, the Drone checks the Spaces around it, starting with the Space it is facing, to see if each Space is unknown, known, or explored. If any adjacent Spaces are unknown, the Drone performs a scan(). If all adjacent Spaces are known, and the Drone, the Drone will check if the Space it is facing is safe using isSpaceSafe(). This method checks that the input Space is known, and does not contain another Drone, a sun, or a barrier. If the Space is not safe per this definition, the method returns false and the Drone checks the next adjacent Space for safety. If the Space is safe, the Drone checks if the Space 1 and 2 deeper in the same direction as the safe Space are safe. If the Drone is facing the same direction as the safe Space(s), it thrusts 1, 2, or 3 Spaces, depending on how far the chain of Spaces is sage. If the Drone is not facing the safe Spaces, it will steer in the direction of the safe Spaces. If no Space is safe around the Drone, it will pass() until the next turn and the surrounding Space becomes safe to travel in.