Aaron's Module Documentation

CoastAgent Class:

- **Overview**:
    - A CoastAgent object repeatedly divides itself into other CoastAgent objects, until a threshold defined by the calling function is reached. This army of agents then starts to roam a zero matrix, moving semi-randomly. They increase height values as they roam the matrix, creating a heightmap with one (mostly) unified landmass. Values are increased according to a perlin noise map at two scales (octaves) overlayed together. All altered points have their biomes labeled as 'coast'
- **Methods**:
    - **getSeed()**:
        - **Purpose**: Returns member field seed
    - **setSeed(newSeed)**:
        - **Purpose**: Change member field seed to specified value
        - **Unit Tests**:
            - **Equivalence Classes**:
                - newSeed is a number
                - newSeed is anything but a number
            - **Test Case 1**: newSeed is a number
                - Expected output: Object's seed field is set to newSeed
            - **Test Case 2**: newSeed is not a number
                - Expected output: Error
    - **getTokens()**:
        - **Purpose**: Returns member field tokens
    - **setTokens(newTokens)**:
        - **Purpose**: Change member field tokens to specified value
        - **Unit Tests**:
            - **Equivalence Classes**:
                - newTokens is a number
                - newTokens is anything but a number
            - **Test Case 1**: newTokens is a number
                - Expected output: Object's tokens field is set to newTokens
            - **Test Case 2**: newTokens is not a number
                - Expected output: Error
    - **getDirection()**:
        - **Purpose**: Returns member field direction
    - **setDirection(newDirection)**:
        - **Purpose**: Change member field direction to specified value
        - **Unit Tests**:
            - **Equivalence Classes**:

            - newDirection is legal direction String (‘south’, ‘north’, ‘west’, ‘east’)
            - newDirection is anything but a legal direction String
        - **Test Case 1**: newDirection is a legal direction String
            - Expected output: Object’s direction field is set to newDirection
        - **Test Case 2**: newDirection is not a legal direction String
            - Expected output: Error
- **randDirection(map)**:
    - **Purpose**: Sets member field direction to a random legal direction String
    - **Unit Tests**:
        - **Equivalence Classes**:
            - map is a Map type object
            - map is anything other than a map type object
        - **Test Case 1**: map is a Map type object
            - Expected output: Object’s direction field is set to a random legal direction String
        - **Test Case 2**: Map is not a Map type object
            - Expected output: Error
- **generate(map)**:
    - **Purpose**: Calls the recurCoast method with the current object as its first parameter
    - **Unit Tests**:
        - **Equivalence Classes**:
            - map is a Map type object
            - map is anything other than a map type object
        - **Test Case 1**: map is a Map type object
            - Expected output: Object’s direction field is set to a random legal direction String
        - **Test Case 2**: Map is not a Map type object
            - Expected output: Error
- **recurCoast(agent.tokens, agent.limit, agent.seed, map)**:
    - **Purpose**: Recursively divides agent into agents with fewer tokens, then procedurally raises points from the heightmap
    - **Unit Tests**:
        - **Equivalence Classes**:
            - agent.tokens is not a number
            - agent.limit is not a number
            - agent.tokens is less than or equal to agent.limit
            - agent.tokens is at or greater than agent.limit
            - map is a Map type object
            - map is anything other than a map type object
        - **Test Case 1**: agent.tokens, agent.limit, or map is the wrong type

- **Test Case 2**: agent.tokens is less than or equal to agent.limit
    - Expected output: One agent will create the entire landmass
- **Test Case 3**: agent.tokens is greater than agent.limit
    - Expected output: A number of agents proportional to the magnitude of this.tokens relative to this.limit will create the landmass

- **raisePoint(point)**:
    - **Purpose**: Change the value of a point's elevation according to a perlin noise map
    - **Unit Tests**:
        - **Equivalence Classes**:
            - point is anything but a Point type object
            - point is out of map boundaries
            - point is within map boundaries
        - **Test Case 1**: Point is anything but a Point type object
            - Expected output: Error
        - **Test Case 2**: Point is out of map boundaries
            - Expected output: point elevation is changed according to perlin noise, point biome is changed to 'coast'
        - **Test Case 3**: Point is within map boundaries
            - Expected output: point elevation is changed according to perlin noise, point biome is changed to 'coast'
- **assignBeacons(point, map)**:
    - **Purpose**: Creates and returns both a repulsor and attractor Point neighboring point on map
    - **Unit Tests**:
        - **Equivalence Classes**:
            - point is anything but a Point type object
            - point is within map boundaries
            - point is not within map boundaries
            - map is anything but a Map type object
            - map is a Map type object with > 2 points
            - map is a Map type object with < 3 points
        - **Test Case 1**: point or map are improper object types
            - Expected output: Error
        - **Test Case 2**: point is within map boundaries, map has > 2 points
            - Expected output: A list containing repulsor and attractor Points
        - **Test Case 3**: point is within map boundaries, map has < 3 points
            - Expected output: Infinite loop
        - **Test Case 4**: point is not within map boundaries, map has > 2 points
            - Expected output: Error

            - **Test Case 5**: point is not within map boundaries, map has < 3 points
                - Expected output: Error
    - **moveAgent(agent, map)**:
        - **Purpose**: Repeatedly change the seed of the agent according to its direction until it reaches an ocean or null point, at which point the agent either stops moving or self-destructs, respectively
        - **Unit Tests**:
            - **Equivalence Classes**:
                - agent is anything but Agent type object
                - agent has null direction
                - agent has null seed
                - agent has non-null direction and seed
                - map has no ocean points remaining in agent's direction
                - map has ocean points remaining in agent's direction
                - map is anything but Map type object
            - **Test Case 1**: agent or map are improper object type
                - Expected output: Error
            - **Test Case 2**: agent has null direction
                - Expected output: Error
            - **Test Case 3**: agent has null seed
                - Expected output: Agent self-destructs
            - **Test Case 4**: agent has non-null direction and seed, map has no ocean points remaining in agent's direction
                - Expected output: Agent self-destructs
            - **Test Case 5**: agent has non-null direction and seed, map has ocean points remaining in agent's direction
                - Expected output: Agent's seed will be set to the nearest ocean point in its direction
    - **score(point, beacons, map)**:
        - **Purpose**: point is scored relative to its distance to the beacons (repulsor and attractor), and distance to the edges of the map
        - **Unit Tests**:
            - **Equivalence Classes**:
                - point is not a Point type object
                - point is within map's boundaries
                - point is not within map's boundaries
                - beacons is not a list of two Points
                - beacons contains a Point not within map's boundaries
                - beacons contains Points within map's boundaries
                - map is not a Map type object
                - map is a Map type object
            - **Test Case 1**: Any input is the wrong type

        - Expected output: Error
    - **Test Case 2**: point is within map boundaries, beacons contains a point not within boundaries
        - Expected output: point is scored relative to its distance to the beacons and distance to the edges of the map
    - **Test Case 3**: point is within map boundaries, beacons contains points within map boundaries
        - Expected output: point is scored relative to its distance to the beacons and distance to the edges of the map
    - **Test Case 4**: point is not within map boundaries
        - Expected output: point is scored relative to its distance to the beacons and distance to the edges of the map, albeit with values larger or smaller than normal

BiomeAgent Class:

- **Overview**:
    - The BiomeAgent Class is a helper class, which prepares the map prior to the BeachAgent modifying it. After CoastAgent operates, all points of height 0 have the biome 'ocean'. The BiomeAgent uses an approximation algorithm to overestimate and label suspected lake points in the map (points of elevation 0 that are landlocked, as opposed to ocean points). Depth first search is used to investigate all suspected lake regions, then relabeling them as ocean points as necessary.
- **Methods**:
    - **generate(map)**:
        - **Purpose**: Identifies and labels suspected lake points, then relabels them as ocean points as necessary by using depth first search.
        - **Unit Tests**:
            - **Equivalence Classes**:
                - map is not Map object
                - map is Map object
    - **defineShore(map)**:
        - **Purpose**: Identifies and returns a list of all coast points adjacent to 'ocean' points.
        - **Unit Tests**:
            - **Equivalence Classes**:
                - map is not Map object
                - map is Map object
    - **approximateLakes(map)**:
        - **Purpose**: Labels the biomes of all points not directly accessible via the cardinal / intercardinal directions as 'lake'
        - **Unit Tests**:
            - **Equivalence Classes**:

            - map is not Map object
            - map is Map object
    - **_moveAlong(point, direction, map)**:
        - **Purpose**: From point, checks straight line path in direction within map. Labels all ocean points along line ‘lake’ and returns ‘lake’ if land is reached, else returns ‘ocean’.
        - **Unit Tests**:
            - **Equivalence Classes**:
                - point is not a Point object
                - point is not within map’s boundaries
                - point is within map’s boundaries and can access the ocean via the cardinal / intercardinal directions
                - point is within map’s boundaries and can’t access the ocean via the cardinal / intercardinal directions
                - direction is not a legal direction String
                - direction is a legal direction String
                - map is not a Map object
                - map is a Map object
            - **Test Case 1**: point, direction, or map is the wrong object type
                - Expected output: Error
            - **Test Case 2**: point is out of map boundaries
                - Expected output: Error
            - **Test Case 3**: point is within map’s boundaries and can access the ocean via the cardinal / intercardinal directions
                - Expected output: ‘ocean’ is returned
            - **Test Case 4**: point is within map’s boundaries and can’t access the ocean via the cardinal / intercardinal directions
                - Expected output: All ocean Points along line are labeled ‘lake’, ‘lake’ is returned
    - **findOcean(point, map, visitedLakes)**
        - **Purpose**: Uses depth first search to search points out from point on map. Every lake point discovered is added to visitedLakes, to reduce the amount of searching needed by the calling function. If an ocean point is found, it immediately returns true, otherwise false.
        - **Unit Tests**:
            - **Equivalence Classes**:
                - point is not a Point type object
                - point is within map boundaries and can be reached from the ocean
                - point is within map boundaries and cannot be reached from the ocean
                - point is not within map boundaries
                - map is not a Map type object

            - map is a Map type object
            - visitedLakes is an empty list
            - visitedLakes is a nonempty list
            - visitedLakes is not a list object
        - **Test Case 1**: point, map, or visitedLakes are the improper type of object
            - Expected output: Error
        - **Test Case 2**: point is within map boundaries and can be reached from the ocean, visitedLakes is an empty list
            - Expected output: returns true
        - **Test Case 3**: point is within map boundaries and can be reached from the ocean, visitedLakes is a nonempty list
            - Expected output: returns true
        - **Test Case 4**: point is not within map boundaries
            - Expected output: Error
        - **Test Case 5**: point is within map boundaries and cannot be reached from the ocean, visitedLakes is an empty list
            - Expected output: returns false
        - **Test Case 6**: point is within map boundaries and cannot be reached from the ocean, visitedLakes is a nonempty list
            - Expected output: returns false
- **assignOcean(point, map)**
    - **Purpose**: Uses depth first search to locate every lake point directly accessible from point in map, reassigning them all as ocean points (including point)
    - **Unit Tests**:
        - **Equivalence Classes**:
            - point is not a Point type object
            - point is within map boundaries and can be reached from the ocean
            - point is within map boundaries and cannot be reached from the ocean
            - point is not within map boundaries
            - map is not a Map type object
            - map is a Map type object
        - **Test Case 1**: point or map is the improper type of object
            - Expected output: Error
        - **Test Case 2**: point is within map boundaries and can be reached from the ocean
            - Expected output: point and all lake points directly accessible from point have biome set as ‘ocean’
        - **Test Case 3**: point is within map boundaries and cannot be reached from the ocean

        - Expected output: point and all lake points directly accessible from point have biome incorrectly set as ‘ocean’
    - **Test Case 4**: point is not within map boundaries
        - Expected output: Error

BeachAgent Class (beachAgent.js):

- **Overview**: A BeachAgent is used to traverse the edge of the landmass(es) generated by CoastAgents. It travels around the coast, identifying points that fall below a certain perlin noise level, labeling their biome as ‘beach’, and repeatedly reducing their height by small increments.
- **Methods**:
    - **generate(map, this.beachList, this.tokens)**
        - **Purpose**: Locates all coast points adjacent to the ocean, labelling them as ‘shore’. Proceeds to create increasingly more beach points around the landmass. Operates in waves, the number of which is determined by this.tokens.
        - **Unit Tests**:
            - **Equivalence Classes**:
                - tokens is anything but a number
                - tokens is below one
                - tokens is at or above one
                - this.beachList is anything
                - map is anything but a Map object
                - map has at least one coast point adjacent to the ocean
                - map has no coast points adjacent to the ocean
            - **Test Case 1**: tokens or map are the wrong type
                - Expected output: Error
            - **Test Case 2**: tokens is below one, map has at least one coast point adjacent to the ocean
                - Expected output: all coast points adjacent to the ocean are defined as shore or tallshore, this.beachList is defined as a list containing all shore points
            - **Test Case 3**: tokens is at or above one, map has at least one coast point adjacent to the ocean
                - Expected output: all coast points adjacent to the ocean are defined as shore or tallshore, this.beachList is defined as a list containing all shore points, shoreline is expanded and its height is decreased by an amount proportional to the number of tokens

            - **Test Case 4**: tokens is below one, map has no coast points adjacent to the ocean
                - Expected output: this.beachList is defined as an empty list
            - **Test Case 5**: tokens is at or above one, map has no coast points adjacent to the ocean
                - Expected output: this.beachList is defined as an empty list
    - **beachify(beachList, map, this.octave)**:
        - **Purpose**: For each point in beachList, decrease the elevation of the point by one and set a random terrestrial neighbor as a new beach point
            - **Unit Tests**:
                - **Equivalence Classes**:
                    - beachList, map, or this.octave are improper types
                    - beachList is an empty list
                    - beachList is a list of shore and beach type Points
                    - this.octave is anything but a number
                    - this.octave is a negative number
                    - this.octave is zero
                    - this.octave is a positive number
                - **Test Case 1**: Anything has the wrong type
                    - Expected output: Error
                - **Test Case 2**: beachList is an empty list and this.octave is anything
                    - Expected output: Nothing happens
                - **Test Case 3**: beachList is a list of shore and beach type Points and this.octave is a negative number
                    - Expected output: Beach will wrap around entire landmass
                - **Test Case 4**: beachList is a list of shore and beach type Points and this.octave is zero:
                    - Expected output: Error
                - **Test Case 5**: beachList is a list of shore and beach type Points and this.octave is a positive number
                    - Expected output: Beach coverage of the shoreline is proportional to the magnitude of this.octave
    - **defineShoreline(map, this.octave, this.beachNoiseMax)**:
        - **Purpose**: Looks around the edge of the entire landmass(s) and labels points as either shore or tallshore, dependent on their place in the perlin noise map
            - **Unit Tests**:
                - **Equivalence Classes**:
                    - map is anything but a Map object

        - map is a Map object
        - this.octave is anything but a number
        - this.octave is a negative number
        - this.octave is zero
        - this.octave is a positive number
        - this.beachNoiseMax is anything but a number
        - this.beachNoiseMax is a negative number
        - this.beachNoiseMax is zero
        - this.beachNoiseMax is a positive number
    - **Test Case 1**: Any of the parameters are the wrong type
        - Expected output: Error
    - **Test Case 2**: this.octave is negative and this.beachNoiseMax is negative
        - Expected output: Beach coverage of the shoreline is proportional to the magnitude of this.octave and this.beachNoiseMax
    - **Test Case 3**: this.octave is negative and this.beachNoiseMax is positive
        - Expected output: Shore will wrap around entire landmass (Beaches along whole edge)
    - **Test Case 4**: this.octave is positive and this.beachNoiseMax is positive
        - Expected output: Beach coverage of the shoreline is proportional to the magnitude of this.octave and this.beachNoiseMax
    - **Test Case 5**: this.octave is positive and this.beachNoiseMax is negative
        - Expected output: tallShore will wrap around entire landmass (No beaches)
    - **Test Case 6**: this.octave is positive and this.beachNoiseMax is zero
        - Expected output: tallShore will wrap around entire landmass (No beaches)
    - **Test Case 7**: this.octave is negative and this.beachNoiseMax is zero
        - Expected output: Shore will wrap around entire landmass (Beaches along whole edge)
    - **Test Case 8**: this.octave is zero and this.beachNoiseMax is anything
        - Expected output: Error