

Requirements

Team 19

DAVIDE BRESSANI
ALEX DAVIS
HASSAN MAHMOOD
DANIEL PARRY
HARRY SWIFT
BENJAMIN WITHNELL

Backend Design

Class Structure

Class Object:

- **ID* (Int PrimaryKey)** - All objects have a unique ID, if all inherit from OBJECT then we won't have duplication, for example a ship and a duck having the same ID as they're different types!
- **type (String)** - The type of the object (useful for quick conditional referencing). For example, is this a ship, a floating item, a cannonball, etc.
- **Object {...}** - Child, the actual object. Object.Object will be the actual thing.

Class StaticObject ← Object: (<-- indicates inheritance)

- **dimensions (2d/3d Matrix of 1s and 0s)** - Matrix of 1s and 0s, the 1s representing 'part of this object' and the 0s 'not part of this object'
- **bLPosition ([x,y] coordinates - 2d static array)** - The bottom left of the object, the bottom left of the 'dimensions' matrix (keeps everything positive and doesn't matter if diameter is odd or even) for positioning on the main map. Doesn't need to be able to change, this is a static object.
- **danger (boolean)** - Is this object dangerous to collide with? For example, crashing a boat into land will cause damage to the boat.

Class Ship ← Object:

- **velocity ([x,y] coordinates - 2d non-static array)** - Movement across the x and y coordinates. Of course, needs to change as the ship changes direction / speed etc. (let's see if the library has anything helpful for velocity stuff)
- **arsenal ([<gun>, <gun>, ... , <gun>])** - the ship's arsenal - it's array of guns. Guns can fire with a cooldown period etc. and guns are the parents of cannonballs. Guns need to be their own objects, as you could fire one side as the other side cools down etc. It's important that we can access a ship's gun objects via the ship...
`ship.guns.where(side: port).fire()` (example of 'fire the ship's port-side guns')
- **HP (Int)** - HP as an integer

METHODS:

- **destroy()** - Destroys this ship object (or do we want to keep all objects but just have an 'alive or dead' boolean maybe? More like an archive? Or destroy so that we can re-use IDs and memory.
- **move(xChange, yChange)** - Causes a change in the ship's velocity attribute, based on the change in x and change in y arguments given. Causes this change every time the method is called. So, for example, for a gradual left turn, if the ship's velocity attributes are both positive (assume we're going straight up), this would be called with two negative values every given time interval (say - 20ms), rather than all at once, slowly reducing the x and y into negatives

Class UserShip ← Ship:

- **USER CONTROL METHODS FROM THE LIBRARY**

Class CPUShip ← Ship:

- **agent (Object)** - This is the 'team' that the ship fights for. We should store the actual object, so that we can use it for lookup. For example `downedShip.agent.reduceShipCount(1)` could be used to reduce the number of ships that a sunk ship's agent has by one, before destroying this object

METHODS

- **seek(coords)** - Essentially we need to implement a pathfinding algorithm here - this needs to let the ship 'seek' to a set of coordinates (perhaps a treasure, an enemy or user ship, etc.), avoiding obstacles such as land. Implement A*? :(
- **patrol(patrol)** - Sets the ship on a loop patrol of going up and down 'doing nothing' (when it is not aware of the user or 'guarding' a location, etc.). Basically will just take a patrol object and that'll have vectors to tell it where to go up and down. Will follow the path vector given.

Class Gun ← Object:

- **ammunition (Int)** - Ammunition count remaining

METHODS

- **restock(value)** - Increases the supply of ammunition by 'value'
- **fire()** - Fires the gun. Reduces the ammo stock by one, and creates a child object of 'cannonball'

Class Cannonball ← Object:

- **gun (Gun object)** - The gun object that fired this cannonball - parent - `thisCannonball.gun`
- **velocity ([x,y] coordinates - 2d non-static array)** - The velocity of the cannonball - works in the same way as any other object like a ship... just probably a little faster!
- **payload (Int)** - damage in HP that will be done to the target ship upon collision
- **range (int)** - The range of the shot across the map

METHODS:

- **hitTarget?()** - Use game engine hitboxing here?
- **destroy()** - Destroy itself if the range has been covered - fall into sea or the land, cannonballs don't have infinite range!

Class Patrol ← Object:

- **vectors** ($[[x,y], [x,y], \dots, [x,y]]$) - *A list of coordinates acting as vectors for a 'pre-defined route'. Can be passed as an argument to a ship to tell it where to go on a 'patrol loop' etc.*

Ship

