NovaJS Design

October 15, 2018

1 Purpose of this Document

This design document aims to specify every design choice in the NovaJS project and define all interfaces between pieces of the project.

2 Accessing Game Data

There is a universal interface, used on both the client and the server, for accessing game data.

2.1 Game Data Interface

All game data is accessed from a single object hereafter referred to as *gameData*. *gameData* has the following data types as fields: **outfits**, **picts**, **planets**, **spriteSheets**, **systems**, and **weapons**. Each data type field has a **.get** method which takes a string ID of a resource in the field and returns a promise that resolves to the resource corresponding to the ID.

The format of each resource is described below. Resources do not have circular references and can be JSONified.

2.1.1 Common Fields

Almost every resource has the following fields. The two exceptions are **pict** and **spriteSheetImage** (due to how PIXI.js loads images):

- **id**: The unique global ID of the resource. Same as the ID used to **.get** it.
- **name**: The name corresponding to the resource. Only used for player interaction.

2.1.2 Outfit

An **outfit** is anything that can be attached to a ship. It has the following fields in addition to the default ones:

- **functions**: An object whose keys are the general functions an outfit performs and whose values are parameters for those functions. See 3 for details.
- **weapon**: An object specifying the weapon (if any) that the outfit has. Specifies the global ID under **id** and the quantity of the corresponding weapon under **count**.
- **pictID**: The global ID of the picture to use in the outfitter.
- **desc**: The text description of the outfit.
- **mass**: The mass of the outfit (see 4).
- **price**: The price of the outfit (see 4).
- **displayWeight**: Determines the order in which the outfit is shown in the outfitter. See 4
- **max**: The maximum number that can be on a single ship.

2.1.3 Pict

A **pict** stores a single picture. It is a buffer containing the data of the picture in PNG format.

2.1.4 Planet

A **planet** is a stationary stellar object. Space stations fall under this category. A **planet** has the following fields in addition to the default ones:

- **type**: A field containing the value **"planet"** specifying that this object is a planet (maybe change this).
- **position**: A list specifying the $[x, y]$ position of the planet.
- **landingPictID**: The global ID of the picture to show when a player lands.
- **landingDesc**: The text description to show when the player lands.
- **animation**: An object specifying the spritesheet to use for the planet (and how to use it? Revise this)

2.1.5 Ship

A **ship** is anything a player can fly around in. Ships have the following fields:

- **pictID**: The global ID of the *pict* to show in the shipyard.
- **desc**: The text description of the ship.

- **animation**: An object with the following fields containing instructions on how to animate the ship:
  - **images**: Most ships are comprised of multiple spriteSheets that serve differe purposes:
    * **baseImage**: Required. The ship itself. Does nothing fancy. Moves and turns as the ship does.
    * **altImage**: An alternate set of images to show when some condition is met (e.g. hyperjumping)(IS THIS CORRECT???)
    * **glowImage**: The engine glow. Appears on top of the baseImage with ADD blending.
    * **lightImage**: Running lights. Appear on top of baseImage with ADD blending. Blink or fade in and out according to (TODO).
    * **weapImage**: Flashes or glows when certain weapons are fired. ADD blending.

    images is an object whose keys are a subset of the above types of images. Each key's corresponding value is an object with **id** as the global ID of the spriteSheet to use and **imagePurposes** as an object describing what ranges of images in the spriteSheet correspond to what actions taken by the ship as follows:
    * **normal**: An object specifying the start and length of the range of the **spriteSheet**'s frames used for the ship's normal movement.
    * **left**: Same as **normal** but for when the ship is banking left. If not present, frames from **normal** are used instead.
    * **right**: Same as **left** but for when the ship is banking right.
  - **exitPoints**: An object specifying where projectiles and beams exit the ship from. Has a field for each type of weapon: **gun**, **turret**, **guided**, and **beam**, each of which has a list of four 3-dimensional positions for the exitpoints. Also has fields called **upCompress** and **downCompress** to account for the ship's non-vertical rendering angle. See the EV Nova Bible page 14 for more information.
- **shield**: The maximum (before outfits) shield of the ship.
- **shieldRecharge**: The rate at which the shield recharges in points per second (TODO).
- **armor**: The maximum (before outfits) armor of the ship.
- **armorRecharge**: The rate at which the ship's armor recharges in points per second (TODO).
- **energy**: The maximum (before outfits) energy / fuel of the ship.
- **energyRechargs**: The rate at which the energy recharges in points per second (TODO).

- **ionization**: The maximum ionization value of the ship. Ships become ionized when their ionization value is above half their ionization value.
- **deionize**: The rate at which ionization disappates from the ship in points per second (TODO).
- **speed**: The ship's maximum speed in (WHAT UNIT? TODO)
- **acceleration**: The ship's acceleration while the engine is on.
- **turnRate**: The ship's turn rate in radians per second.
- **mass**: The ship's mass in tons. Used for collision knockback.
- **outfits**: An object whose keys are the global IDs of outfits and whose values are the number of that type of outfit on the ship. To add weapons to a ship, put the outfit corresponding to them in this object.
- **freeMass**: The total mass available for installing outfits. This includes the mass of outfits listed in **outfits**, so it will show up as less if the outfits in **outfits** are massive.
- **initialExplosion**: The initial explosion to use when the ship dies. (TODO: explain more)
- **finalExplosion**: The final explosion to use when the ship dies.
- **deathDelay**: The delay in milliseconds between when the ship hits zero armor and when the ship explodes, during which the initial explosion plays repeatedly.
- **displayWeight**: Specifies the order in which the ship is displayed. See Shipyard.

2.1.6 SpriteSheetImage

A **spriteSheetImage** resource is identical to a **pict** resource, however, it exists within a different id scope (**spriteSheetImages** instead of **picts**).

2.1.7 SpriteSheetFrame

A **spriteSheetFrame** resource describes how to pull textures out of a **spriteSheetImage**. It uses the TexturePacker format (because PIXI.js requires it).

2.1.8 SpriteSheet (TODO)

A **spriteSheet** resource encodes data about the contents of a **spriteSheetImage** including the global ID of the **spriteSheetImage**, frame boundaries, and collision convex hulls for each image.

2.1.9 Systems

A **system** resource describes the contents of a system. It includes the following fields:

- **position**: The $[x, y]$ position of the system on the star map.
- **links**: A list of the global IDs of **systems** that this system has a link to.
- **planets** A list of the global IDs of **planets** contained in the system.

2.1.10 Weapon

A **weapon** resource describes how a weapon behaves.

- **type**: A string specifying the type of weapon. See 5 for details on the types. (THIS CONTRADICTS THE USE OF TYPE IN PLANET!!!)
- **animation**: (TODO: REFACTOR ANIMATION WITH SPRITESHEET). (This might turn out not to be an animation.)
- **shieldDamage**: The amount of shield damage to do per shot. In the case of a beam weapon, the amount of shield damage to do per second of contact. (TODO: FIX BEAM WEAPONS)
- **armorDamage**: Same as shieldDamage but for armor.
- **ionizationDamage**: Same but for ionization.
- **ionizationColor**: The color to tint the ionized ship with.
- **reload**: The time in seconds it takes to reload the weapon after firing. A weapon can not fire more than 60 times per second. (TODO: FIX THE UNITS IN THE CODE)
- **duration**: The time in seconds that a projectile stays on the screen. Ignored for beams (WAIT. Maybe this is how multibeam works? Check that plugin)
- **speed**: The speed of projectiles as they leave the weapon. Frame of reference depends on the weapon type.
- **turnRate**: The turn rate of the projectile if it is guided.
- **fireGroup**: "primary" or "secondary". Whether the weapon fires with primary weapons or is a secondary weapon. Ignored for point defense (TODO: MAKE SURE THIS IS TRUE).
- **exitType**: Which exitpoints (2.1.5) of the ship to fire the weapon from. "gun", "turret", "guided", "beam", or "center".

- **inaccuracy**: The maximum angle to either side of the desired firing angle that the shot may take. Radians. (TODO: MAKE IT RADIANS, CALL IT INACCURACY)
- **impact**: The change in momentum to impart to the ship it hits. (Tons distance / second)
- **burstCount**: The number of shots the weapon may fire at it's standard reload period before performing an additional burst reload. Zero means infinity.
- **burstReload**: The time in seconds the weapon must spend reloading after firing off a burst (TODO: MAKE IT SECONDS)
- **shield**: The shield of the weapon. Used if it's vulnerable to point defense.
- **armor**: The armor of the weapon. Used if it's vulnerable to point defense. (TODO: Refactor the fact that shield damage deals 1/2 to armor for weapons into giving weapons shields? Maybe not. I'm not sure.)
- **trailParticles**: The number of particles the weapon emits per second while in flight (TODO: Make it seconds)
- **hitParticles**: The number of particles the weapon emits when it hits something.
- **maxAmmo**: (TODO: Decide how maxAmmo will work)
- **useWeapImage**: Whether or not to use the weapImage (2.1.5) animation of the parent ship when firing. (TODO: RENAME FROM useFiringAnimation)
- **energyCost**: The energy to use per shot. If it's a beam weapon, the energy to use per second while firing.
- **ammoType**: (TODO: Decide how ammo is tracked)
- **fireSimultaneously**: Whether multiple weapons of the same type should all fire at once or fire one after another.
- **explosion**: The explosion to use for the weapon. (TODO: Explain explosion)
- **secondaryExplosion**: A secondary explosion to use when the weapon explodes. About 16 are placed randomly around the main explosion.
- **vulnerableTo**: A list of weapon types the weapon's shots are vulnerable to. Including "point defense" will make point defense weapons fire on this weapons projectiles.
- **submunitions**: A list of objects with the following fields describing the submunitions to fire and information about how to fire them:

- **id**: The global ID of the weapon to sub into.
- **count**: The number of submunitions of this type to fire.
- **theta**: The largest angle offset from the parent projectile's angle at which a sub may fire in radians (TODO: Make this radians)
- **limit**: The recursion depth limit when a submunition is the same as its parent projectile. Doesn't handle mutual recursion (behavior of the game engine is undefined in that case).
- **uniformSpacing**: Whether the submunitions should be uniformly spaced at angles of **theta** or whether they should appear randomly within a cone of angle 2***theta** in front of the parent projectile.

3 Outfit Functions

4 Outfitter

5 Weapons

The types of weapons are as follows:

- **unguided**
- **turret**
- **guided**
- **freefall bomb**
- **rocket**
- **front quadrant**
- **rear quadrant**
- **point defense**
- **beam**
- **beam turret**
- **point defense beam**