System Design

Joshua Froggatt | frojy009@mymail.unisa.edu.au | 110375072

- **Item** inherits from the **abstract item** interface, which houses the basic variables and functions that all items share.
- **Item** has different multipliers that it adds to certain simulation parameters, item can have 1 or many different multipliers, this is why a hashmap is used to keep track of the multipliers.
- Abstract item has a hashmap called **system_multiplier.** This hashmap keeps track of the multipliers that the **Item** hash
    - The scrap value multipliers are all added to the multi map under the **value** key, this key keeps track of all of the value multipliers, which are then applied to scrap value during the landed phase.
    - The keys for the **system_multiplier** multi map are:
        - value
        - explorer
        - operator
        - save
        - loot
- The simulation parameter multipliers are constant as they cannot and will not change

- **Inherited Moon** is made up of an inheritance of abstract moon,
- **Abstract Item** has a single Set() function which is used to set whether the item is bought or not, which means whether it can affect the simulation parameters.
- **Item Manager's** addMulti() function was removed, to keep encapsulation strong.
- **Item Manager** instead has a **function** called applyMulti() which takes in a float and the name of the system parameter, and applies that multiplier to the float
- The **Multi** object was removed, as it made no sense to have the multiplier as an object instead of a float.
- **Moon Manager** and **Item Manager** have an insertion_order vector, this vector keeps track of the order that the moons and items were created, this is done for printing them out in their respective show commands (store, moons, etc)
- **Travel Cost** was removed because it is more efficient for travel cost to be stored as an integer in the **Inherited Moon** instead of an object
- **send()** and **sell()** methods were moved from **Moon Manager** to **Abstract Moon** so that the system fits the design constraints.
- **onNavigate()** method decreases the player balance when routing to a paid moon
- **getWeatherCondition()** method returns a string that is the weather condition name in brackets, this method is used only for printing.
- **Game** contains all of the variables that are used throughout the game, as well as references to all of the classes and objects
- **Game's** increase and decrease methods are called by **Abstract Moon** when certain conditions are met (e.g. when selling cargo, when a crew member dies)

- **generateNum()** is a method that holds the random number generator, it takes in a minimum value and a maximum, but has default values of minimum 0 and maximum 1. The method returns the generated number is a floating point value.
- **defineItems()** and **defineMoons()** creates the moons and items, and adds them to their respective managers
- **initialiseNewGame()** sets the initial variable values, the current_orbiting_moon to **Corporation** as well as displaying the initial startup text and the games logo.
- **newDay(), inOrbit()** and **landedMoon()** handle the actual game, **newDay()** resets specific variables to their default, **inOrbit()** handles the pregame inputs such as: moon routing, buying items, and viewing the store, moons and the players inventory, while **landedMoon()** handles all of the moon gameplay, with sending players and selling items.
- All of the items and moons were made as their own c++ files, as it made it easier to implement them.
-