1 REFLECTION

1.1 DESIGN DESCRIPTION

"Space Debris" is a garbage collection game – in space! You are a Private employed by a space station to help remove this space garbage from orbit around earth. On your first day, you are thrust out into the field and left to learn on the fly.

Just as you're getting started, there's an emergency – the old ISS is on a collision course with your space station. It's up to you to gather the necessary parts and intercept it before time runs out.

The game is laid out on a 5x5 grid with 16 available "tiles" for the player to walk around on. There are 4 different events that can happen on a tile:

1. The player has the option to pick up debris
    a. This costs them extra time, but increases the final score by 15 pts.
2. The player collects a required item to get into the ISS
3. The player collects extra jetpack fuel – you'll need these to win the game!
4. The player must enter a series of letters in the correct order to dodge incoming debris. If they fail, it costs them extra time.
5. The player uses their collected items to get into the ISS and save the day! But beware, without all the required items, you'll waste valuable time traveling to the rendezvous point.

The player is equipped with a jetpack. At any time, the player can use 'j' for their movement to move 2 spaces for the cost of only 1 movement turn. This is tracked on the player HUD.

When the player enters the event (the ISS collision), the players HUD updates their "Oxygen" countdown to "Minutes", since that is the pressing factor.

Inventory items are added the top of the player's HUD. Required items are marked with a "*". Other items are randomly given a name from a predetermined list. The items are actual items listed on the Wikipedia page for space debris. There's a limit of 7 items, including the 3 required items to win the game.

It is necessary to save the station to win the game. However, it is possible to beat the high score of saving the station by only collecting 4 debris items. Each debris item is 15 points and saving the station is 50 points.

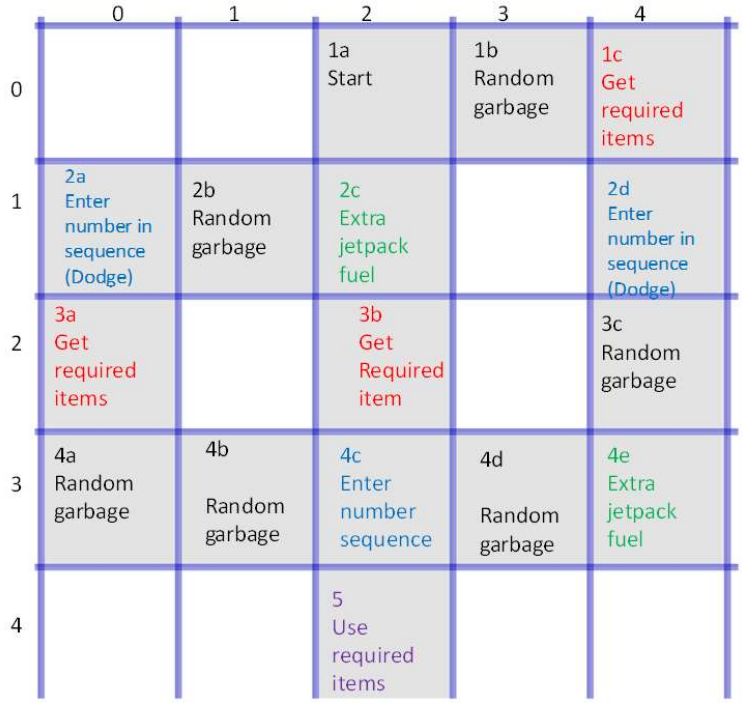
The final score is outputted to the player, along with the score board that tracks all time high scores in a text file.

1.2 ORIGINAL PLAN OVERVIEW

My original plan in its entirety is below for reference.

My plan initially was a 20x20 board – at the point of the original design, I didn't quite understand how I was going to link together my spaces. Once I realized how I was going to go about it (by

using a function to set each link), I decided writing out 400 links manually might be a bit much and scaled the board way back to the 16 tile board it appears as now.:



The original design was also going to have a sort of RPG like aspect – where you could respond to your supervisor with various positive and negative responses, and the game would remember how you responded. At the end, you'd get an end credits summary describing your adventure which would use your responses to paint the narrative. It is still something I would like to add in the future, but for the purpose of this assignment, it got put on hold.

1.3 Changes in Design

My original design involved a more straightforward 2D board (like one created by a 2D array) and I wanted the space ship to move across the board so that the player would have to time it to end up in the correct spot. The idea was to treat it like relative position vectors. It was also going to be 20x20, which would have been way too much.

However, when I switched to 16 linked "tiles' for the board, I had to change how the user would go about winning the game. This required a fairly major overhaul to my Game class (which in my original design was actually in the Menu – when it go too large, I made it it's own Game class).

1.4 Problems Encountered

The biggest problem I encountered was scope creep. My plan for the game was a lot larger than what I finally implemented. I didn't realize just how much code and time would be required to implement everything I wanted, so I was redesigning as I went to scale it back to something manageable. Another example of this was the idea of weight. I wanted the users movement to

be proportional to how much mass they had on them (F=ma!), but I didn't get the chance to implement that functionality either.

The biggest issue I had with this code was finding out the "owner" of my pointers in my debris inventory. I had an issue when I tried to delete them in the Player class where either I had an invalid new-delete match or I tried to delete something twice. As I looked into it, I was introduced into this concept of the "owner" of the pointer.

I thought that owner had to be the player class, since that was where the pointers were stored. However, it ended up being the *game* class, because I suppose the game owned the player – I'm still not entirely confident why it worked out that way.

1.5 TEST TABLE

Test	Note	Outcome
Menu		
displayMenu	Check menu properly apepars	All 5 options shown
runProgram	Check each menu item runs as expected	Each option runs correctly
runProgram	Check asks again when done	Menu is redisplayed
aboutGame	Check displays contents of about.txt	Text file is displayed
Game		
setAllLinks	Verify spaces work correctly by moving player around	As expected
setGameMap	Verify game map matches expected movements	Map displays correctly
displayHUD	HUD displays	As expected
moveOrJetpack	Verify both jetpack and non jetpack moves work	As expected
movePlayer	Player moves in direction indicated	Player moves, is not allowed to move out of bounds
testGameWon	Game returns win when required	Win screen plays when appropriate
checkValidMove	Check if move is valid	Returns appropriate response
makeMove	Player position updates	As expected
updateBoard	Board updates to reflect move	As expected
calculateScore	Check score is correct	Score correct for multiple scenarios
getLeaderboard	Check output matches fil	As expected
showLeaderboard	output matches files and displays correctly	As expected

addToLeaderboard	New score is added or not added	If score is high enough, is added in order. Otherwise, it does not appear
sortBoard	Board is in correct order by score	As expected
displayEventMessage	Triggers after first move	displays event
startUpGame	Triggers if option 1	As expected
gameInstruction	Triggers if option 1	As expected
finalMessage	Displays correct message on win or lose	As expected
runGame	Verify all options	all items run as expected
Player		
Verify Inv Limit	Game does not allow more than 7 items	As expected
All items created	Player members created succesfully	As expected
Debris		
pickName	Random debris is chosen	Random names from list appear on HUD
pickRequiredName	One of each req'd is chosen	Each of required options appears as gathered.
Display		
printSpaceStation	Displays correctly when called	As expected
displayMissionControl	Displays correctly when called	As expected
printOpening	Displays correctly when called	As expected
displaySatellite	Displays correctly when called	As expected
displayLoseGame	Displays correctly when called	As expected
Leaderboard		
Members are saved	Can access members in function as needed	Members display and update correctly
Space Class & Child Classes		
interactSpace - StartSpace	Nothing happens	As expected
interactSpace - UseDebris	Gives user detail on saving station	Dispays not enough and saving as expected
interactSpace - PickUpDebris	Displays all scenarios for inventory correctly	Works for req'd and not items, and for full inventory
interactSpace - EnterCode	Generates and matches code	Creates a random code and can test if user entered same code
interactSpace - JetpackFuel	Increases jetpack fuel number	player jetpack uses increases

2 ORIGINAL DESIGN:

"Health" is your oxygen

To start: "You only have an hour of oxygen left, try to get as much SPACE DEBRIS as you can before your oxygen is up

Halfway through - the old ISS is hurtling towards an incoming ship and will destroy it! Use the debris you've gathered to alter it from it's collision course!

Must have at least 1 of: crowbar, old battery, wires

Change countdown from oxygen to timer

Possible to have already gathered these things, increasing your chance of succeeding
Give X turns (10) before the message for them to start collecting garbage

Wow, you saved the space ship! And managed to gather up X amount of garbage while doing it! Nice job! Who knew you were such an excellent engineer - maybe it's time for a promotion?

1. Yes, get me out of garbage collection!
2. No thanks, this job is too important.

You couldn't stop the SPACE DEBRIS in time. Thankfully the passengers all escaped in the pods, but the collision generated even more space debris. Looks like we have a lot more work ahead of us!

1. Yes, I can't wait to get started
2. No, I quit!

An ending screen printing results (happy to be in this job? Saved the ship? Continued to be a garbage collector?)

Spaces to interact - must have three different actions the user can do besides move

1. Use jetpack (# of spaces moved effected by # of debris collected)
2. Unload some debris

    a. Exit

Randomly generate ship location in upper part of one quadrant

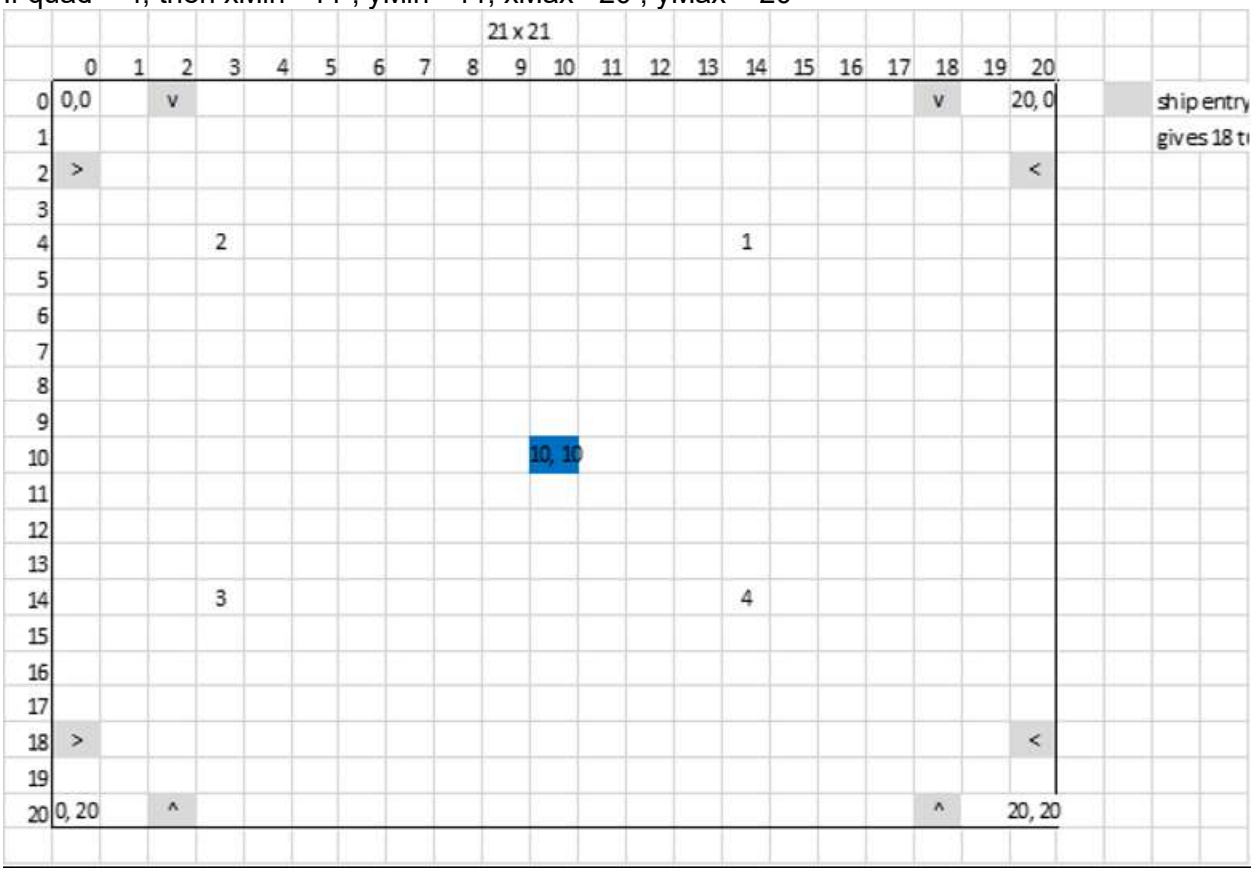
Choose quad 1, 2, 3, 4
Within quadrant, choose x or y
Then find starting pos

Randomly generate the position of the three needed items

Find quadrant user is in
Pick random location in each of the other three quadrants for the three debris

Randomly add other misc debris

Quadrants:
If quad = 1, then xMin =11 , yMin =9, xMax =20 , yMax = 20
If quad = 2, then xMin =0 , yMin =0, xMax =9 , yMax = 9
If quad = 3, then xMin =0 , yMin =11, xMax =9 , yMax = 20
If quad = 4, then xMin =11 , yMin =11, xMax =20 , yMax = 20



Debris:

3 categories, Heavy, medium, light

Heavy worth 5 points, 5 weight multiplier
Medium worth 3 points, 3 weight multiplier
Light worth 1 point, 1 weight multiplier

Jetpack

- This affects the jet pack. On empty, the jet pack can move 5 spaces. For every 5 pounds, that is reduced by 1 square. So at 5 pounds it moves 4 squares, 10 pounds is 3 squares, etc.
- Jet pack starts out with (3? 5?) uses, can find (2? 3?) extra jet pack fuel

Updated Board:

1. **Space class**
   a. The game requires a **Space** class, which represents the space the player can be in. The Space class must be an **abstract class** that will have pure virtual functions (you can add not pure virtual functions, too).

   b. Inside the **Space** class, there must be at least **4 Space pointers**: top, right, left, and bottom.

   c. Use the class to create a game with the structure of linked space. (You are free to add more Space pointers to the Space class, but must have at least 4 Space pointers)

2. The game must have at least **3 derived classes** that are derived from the **Space**

   Each representing a different type of space, and need to have a special action for the player to interact with. It can be opening the door to another space, or maybe attack the monster, or turn on the light switch, or sing a song to please the king.

3. Inventory
    a. Member variables:
        i. itemName
        ii. itemWeight
        iii. requiredItem
        iv. xCoord
        v. yCoord
        vi. maxWeight
    b. Functions
        i. Add item to inventory
4. Player
    a. Linked list with inventory
    b. Players inventory weight
    c. Players jet pack uses
5. Board - see space class?
6. Ship
7. Game
8. Story - used to output all text, save player choices, etc.