CS 162 – Final Project
Starship Simulator

Daniel Beyer
Dan.beyer@gmail.com
05/31/2016

Requirements

Assignment requirements: You will design and implement a text-based game or puzzle where the player moves through a series of rooms and gathers items to achieve some purpose. Each space will be a class with at least four pointer variables that link to other spaces. You must have at least 5 spaces of at least 3 different types. You will also have a space abstract class that will have a special pure virtual function, and each type of space will have its own special action. The player will also have a container to store items, one of which is required for the solution. You will also need to add and remove at least one space. There will be a time limit and the player must interact with parts of the structure and not simply collect things.

Overview/Game Design

In this game you play the role of captain of a starship that has just been attacked. Similar to the starship *Enterprise* from Star Trek, the player's starship is composed of different "decks" that represent the different departments. As captain, the player must go to these different "decks" and bargain to collect the correct power crystal to be used on the final deck, Engineering, to get the engines back up and running.

There will be an abstract Space class with six derived classes from Space: Bridge, Holodeck, 10forward, Sickbay, Backup Bridge and Engineering. Space will have a pure virtual function called **Special()**. Each derived class will have their own **Special()** definition:
Bridge: The player can assign command to their "Number One" first officer or to the new Ensign
Holodeck: The player can turn off the holodeck and take its green power crystal
10forward: The player can trade their green crystal from the holodeck for a blue crystal.
Sickbay: The player can trade the yellow crystal from the backup bridge for the red crystal.
Backup Bridge: The player can trade the blue crystal for the yellow crystal.
Engineering: Try to use the red power crystals to restart engines.

The player will have a backpack inventory to carry the crystals they collect from deck to deck. The limit will arbitrarily to set at something around 10. This will likely be an array to hold crystal items.
There will also be a timer set at the beginning of the game and the player will be notified of their progress after leaving each deck.

Program Input

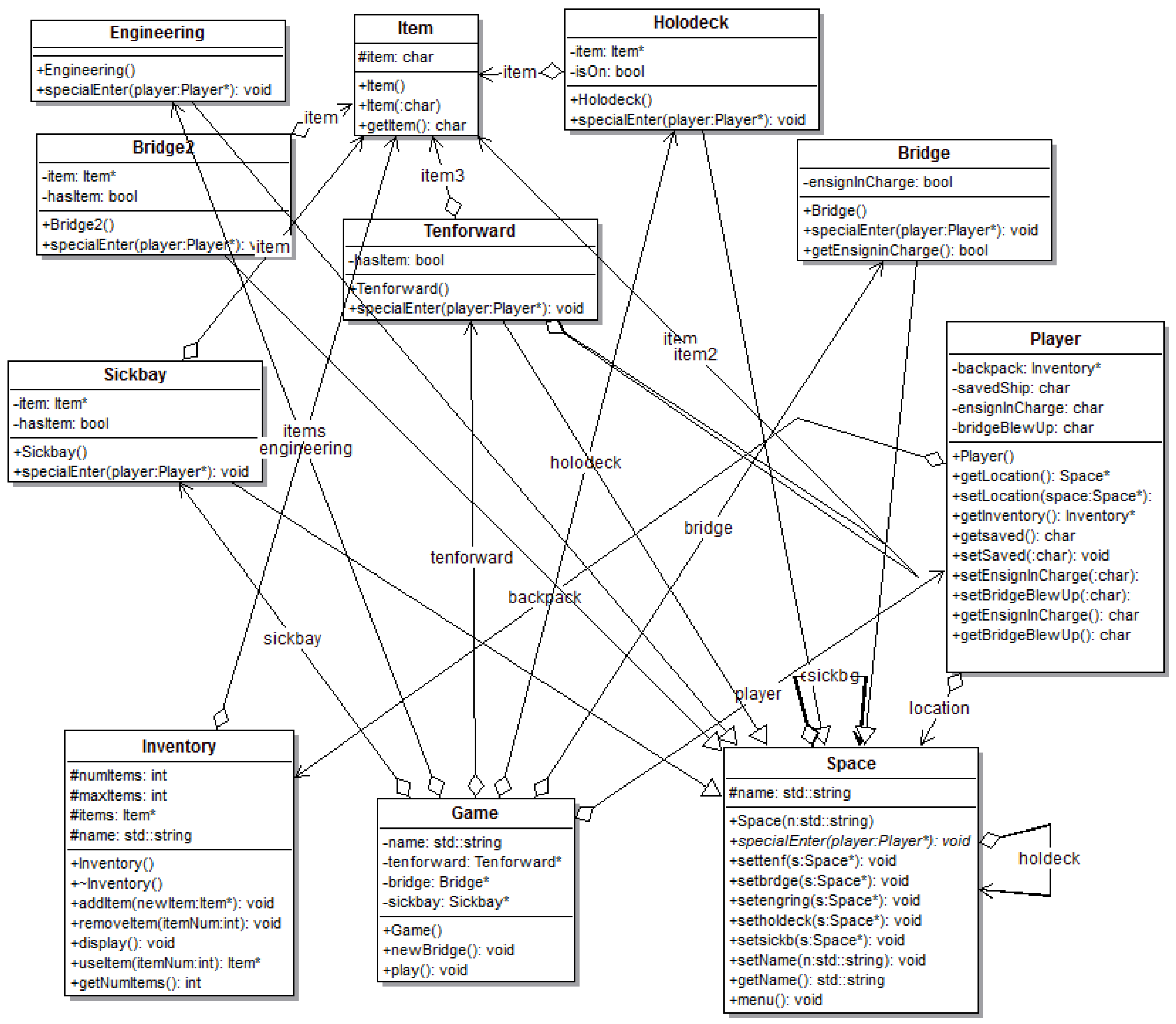
User enters:

- Primarily menus selections (select which deck to go to, select whether or not to perform special function)

Program Output

Program outputs messages in each deck: prompting user to trade crystals or which deck to progress to next. In addition, a timer is displayed after leaving each deck indicating how many moves the player has taken up so far. Messages are displayed when time has run out and if the player wins. The user is also prompted to see a solution at the beginning of the game.

Class Diagram



<u>**Test Plan and Results**</u>

I used incremental testing as I made each method in each child class of the Space class. Although this worked for the most part, some areas required me to finish the program completely and performing some tests once I was able to experiment with full playthroughs. The test results below are broken down by class ordered alphabetically.

Class	**Test**	**Expected Result**	**Result**
Bridge	Option 1: leave first officer in charge	"You opt to leave your trusted Number One first officer in charge	Pass
	Option 2: leave ensign in charge	"You opt to leave the new ensign in charge…"	Pass
	Option <1 or >2	"Invalid selection…"	Pass

Backup Bridge
Driver functions:
`player->getInventory()->useItem(num)->getItem();`
`player->getInventory()->removeItem(num);`
`player->getInventory()->addItem(item)`
	Enter bridge and already taken the yellow crystal	"There is nothing of interest here…"	Pass

	Select Item to trade that is not blue crystal	"That is not a blue crystal!"	Pass
	Select Item to trade that is blue crystal	"You give them the blue crystal…" Yellow crystal added to inventory	Pass
	View Inventory	Correct Inventory displayed	Pass
	Enter turbolift and select next destination -> correct destination arrived at	Correct destination arrived at	Pass

Engineering
Driver functions:
`player->getInventory()->useItem(num)->getItem();`
`player->getInventory()->removeItem(num);`
	Select Item to use that is not red crystal	"That is not the red crystal!"	Pass

	Use Red crystal		

`player->getInventory()->addItem(item)`		“Use you red crystal…You WIN!”	Pass
	No crystals in inventory	“You have no crystals”	Pass
	View Inventory	Correct Inventory displayed	Pass
	Enter turbolift and select next destination -> correct destination arrived at	Correct destination arrived at	Pass

Game
Driver Functions:
`player->getLocation()`
`specialEnter(player)`
`player->getBridgeBlewUp()`
	Player leaves bridge for first time	Bridge blows up, bridge space deleted, Bridge2 space added and all pointers to other spaces set appropriately	Pass

	Timer increment elapses max time limit	“Time ran out! Ship was destroyed!” Game ends	Pass

Holodeck
Driver Functions:
`player->getInventory()->useItem(num)->getItem();`
`player->getInventory()->removeItem(num);`
`player->getInventory()->addItem(item)`
	If (isOn == false)	“The holodeck is powered down…”	Pass

	Select any answer besides Y or y	“OK you will leave it alone for now.”	Pass
	Select Y or y	Green crystal added to inventory	Pass
	View Inventory	Correct Inventory displayed	Pass
	Enter turbolift and select next destination ->	Correct destination arrived at	Pass

	correct destination arrived at		

Inventory
Driver Functions:
`player->getInventory()->useItem(num)->getItem();`
`player->getInventory()->removeItem(num);`
`player->getInventory()->addItem(item)`
`player->getInventory()->removeItem(item)`
	See Space class tests.
	Inspection of Inventory Items
		Displays appropriate Inventory items.	Pass

	Inventory Items added	Items added to inventory and displayed appropriately	Pass
	Inventory Items removed	Item(s) removed	Pass
Main Driver Functions: Game.play()	If choice == 1	Game begins	Pass
	If choice == 2	Solution presented	Pass

Sickbay
Driver Functions:
`player->getInventory()->useItem(num)->getItem();`
`player->getInventory()->removeItem(num);`
`player->getInventory()->addItem(item)`
`player->getInventory()->removeItem(item)`
	If(hasItem == false)	“There is nothing of interest here…”	Pass

	Select crystal that is not the yellow crystal to trade	“That is not the yellow crystal!”	Pass
	Select yellow crystal to trade	“You give her your yellow crystal..” Yellow crystal removed, red crystal added to inventory	Pass
	There are not crystals in your inventory	“You have no crystals to trade!”	Pass
	View Inventory	Correct Inventory displayed	Pass
	Enter turbolift and select next destination -> correct destination arrived at	Correct destination arrived at	Pass

Tenforward
Driver Functions:
`player->getInventory()->useItem(num)->getItem();`
`player->getInventory()->removeItem(num);`
`player->getInventory()->addItem(item)`
`player->getInventory()->removeItem(item)`
	If(hasItem == false)
	
	Select crystal that is not the green crystal to trade
	
	Select Green crystal to trade
	
	View Inventory
	
	Enter turbolift and select next destination -> correct destination arrived at
		"There is nothing of interest here…"
		
		"That is not the green crystal!..."
		
		She gives you the blue crystal plus some other crystals added to your inventory
		
		Correct Inventory displayed
		
		Correct destination arrived at
			Pass
			
			Pass
			
			Pass
			
			Pass
			
			Pass

Reflection

My initial plan before I started actually designing this program was much more complex than the program I ended up with. Due to a combination of time constraints and my own technical limitations, I had to scale back my plan until it fit the design I presented here. This in itself was a tremendous learning experience and showed me how much work coding even a straightforward game like this can be. I initially wanted to have the player perform a small game on each deck, like a random number generator "highest number" game, but this proved to be beyond my capability right now.

I really struggled with some challenges while working on this project. One challenge I faced was the requirement to have the structure of the spaces change by adding and removing spaces. I knew from the beginning that I wanted to have the Bridge space be the space that was to be removed, and a new Bridge space added. But I had trouble finding a solution to the problem of telling my Game class when to create a new Bridge2 object. The solution I came up with was to have a variable in the player class that could be switched when the original bridge was to be deleted and the new Bridge2 object created. I created a simple if() statement that would check this player variable and if it had been switched, then the old Bridge is deleted and the new Bridge2 is created and all its pointers set to the other class objects. Once this is performed, another variable to switched in the player object indicating that the new bridge has been created.

Another challenge I faced initially was how to implement the Inventory class. I considered a vector first, but I remember Prof. Rooker mentioning we should focus instead on arrays for this course. As such, I implemented an array of pointers to items, with the items being created in their respective Space subclasses. Making the array of a set size made this implementation relatively straightforward.
Memory management proved to also be a challenge for me. I had to go back multiple times and make sure I had appropriate destructors for all my classes. The program now runs in in valgrind with the results:
“HEAP SUMMARY: in use at exit: 0 bytes in 0 blocks. Total heap usage: 20 allocs, 20 frees… All heap blocks were freed – no leaks are possible”
“ERROR SUMMARY: 0 errors from 0 contexts”.
This indicates to me that I was finally able to eliminate all memory leaks present in the program.
Another area of challenge for me was the order of my “#include” statements and forward declarations of other classes. I had to go back to my UML design graphic and plot out which classes were reliant on each other and avoid interdependencies.
This final project proved to be enjoyable but also quite a challenge. It gave me further opportunities to see the elegance of object oriented programming while also giving me an opportunity to hone my skills using inheritance, something I feel I needed more work on. I also came to realize how important the design process is when trying develop a program like this. The project seemed overwhelming at first, but after laying out a basic framework the process became much simpler as I could tackle one area at a time and see how I was making progress based on my design document.