Final Project: Design, Test Table & Reflection

Nicole Reynoldson

8/13/19

CONCEPT: The player is stranded on a desert island and must escape. To do so, they must collect a tarp, rope and 3 logs and bring them to the beach to assemble them. To retrieve logs, the player must find an axe to chop wood in the forest. The rest of the items are scattered on the map for the player to find.

DESIGN PLAN

Space Class - Base

Member Variables

- spaceType
- Pointers to top, bottom, left and right
- String that holds the name of an item if one is available to find on that space
- String to hold an item needed to do a specific action (ie. chop wood)
- String to hold the description of the special action square
- Pointer to a Player
- Pointer to an Inventory

Member functions

- Constructor
    - Takes a Player pointer as a parameter
- enterSpace()
    - Pure virtual must be defined by each derived class
    - Intended to give a description of each space and trigger any random events
- randomEvent()
    - Can be overridden by each derived class
- spaceAction()
    - Intended for any special space actions (ie. chopping wood, fishing etc.)
    - Void in base class and can be overridden by derived
- explore()
    - Prints a generic prompt if the derived Space does not contain an item to be found
    - Else will print the special prompt for finding an item for that space, will add the item to the bag and update the variable to be void.
- doAction()
    - Compares the player inventory to the specific item required to do the unique space action. If it is in the inventory, returns true, else returns false.
    - Virtual so that the function can be overridden in the case that a unique space action is always available (no item required)
- Getter functions for all pointers
- setLinks - Takes space pointers as parameters to connect the space to others
- getItem() -Returns the string name of any item stored in the space for finding
- getSpaceType() - Returns the name of the space
- getActionDesc() - Returns a string containing a description of the unique space action (ie.chopping)

Beach Class - Inherited from Space Class

Member Variables

- fishProb

Member functions

- Constructor
    - Takes value between 1 and 100 and initializes the chances of catching a fish on that particular beach
    - Takes a string and initializes the item if there is one on that particular beach
    - Initialize base class derived member variables: item, actionItem, actionDesc, explorePrompt and spaceType
- enterSpace()
    - Override from space class with a unique descriptor
- spaceAction()
    - Updates the player health for completing the action
    - If player has not died
        - Generates a random number between 1 and 100
        - If the value is less than or equal to the fishProb add a fish to the players inventory
        - Else provide a prompt
    - Else let the player know how they died

DenseForest Class - Inherited from Space Class

Member functions

- Constructor
    - Set the spaceType, actionItem and actionDesc
- enterSpace()
    - Overridden from base class to give unique descriptor
    - Call the random event function
- chopWood()
    - Update the player health
    - If the player is not dead
        - Add logs to the bag (will prompt if no space is available)
    - If the player dies
        - Display prompt that they have died
- randomEvent()
    - Generate a random number between 1 and 100
    - Compare against the probability of being bitten by a snake
        - Update the health accordinging
        - If the player is not dead
            - Let them know they have been bitten
        - Else let player know how they died

Sparse Forest - Inherited from Space Class

Member functions

- Constructor
    - Set the spaceType and the action description
- enterSpace()
    - Overridden from base class. Gives a unique scene descriptor
- randomEvent()
    - Generate a random number between 1 and 100
    - If the value is less than or equal to the probability of finding aloe
        - Add to health
    - Else do nothing
- spaceAction()
    - Update the player health for foraging
    - If the player dies
        - Inform the user how they died
    - Else add the berries to the inventory (will give prompt if bag is full)
- doAction()
    - Overridden from base class
    - Always returns true since no items are required to complete the action

Cave Class - Inherited from Space Class

Member Variables

Member functions

- Constructor
    - Set spaceType and item
    - Initialize options for the fight menu
- enterSpace()
    - Gives a unique descriptor of the space
- explore()
    - If the player already has the rope, let them know there is nothing else in the cave
    - Else launch the bat fight sequence
- batFight
    - If player has the axe, set their attack higher than if they did not
    - Display the menu and allow user to choose to fight or run
- combat()
    - Player attacks first
    - Use the rollDie function to determine the attack and subtract that from bat health
    - If the bat has not died
        - rollDie function using the bats stats
        - Subtract the attack from the player health
        - If the player has died, let them know
            - Else tell the player their current health

Hut Class - Inherited from Space Class

Member Variables

Member functions

- Constructor
    - Initializes the spaceType, item to be found and the explorePrompt

Inventory Class

Member Variables

- Capacity
- Vector of strings

Member functions

- Constructor
    - Takes an int for the capacity as a parameter
- addItem
    - If the bag is not full
        - Add the item to the end of the vector
    - Else let the user know their bag is full
- hasItem
    - Iterate through the bag, if the item at that location has the item
        - Return true
        - Else return false
- removeItem
    - If the bag has the item requested to be removed use an iterator to point to the item to be removed
    - Erase the item from the vector and return true
    - Else return false
- Print inventory
    - If the bag is full let the user know they have no items
    - Else print the contents of the bag
- isFull
    - Checks that the vectors size is equal to the capacity passed in
        - If so, return true
        - Else return false
- getItemQty
    - Used to check that the user has 3 logs in their inventory
    - Same as the hasItem function but returns the quantity kept as count

Player Class
Member Variables
<ul><li>Health</li><li>Inventory</li><li>Dead</li><li>Vector to hold the food options for eating</li><li>Vector to hold tips that Wilson can possibly provide</li></ul>
Member functions
<ul><li>Constructor<ul><li>Sets health to 100</li><li>Allocate the inventory</li><li>Set the dead status to false</li></ul></li><li>updateHealth<ul><li>Takes an int as the amount to increase or decrease health</li><li>If the health is below 0, set it to 0</li><li>If the health is above 100 set the health to 100</li></ul></li><li>catalogFood<ul><li>Checks if the player has food items in their bag</li><li>Will dynamically change depending on what the user has at that moment</li><li>If there is no food in the inventory the function will return false</li><li>If the player does not want to eat food, the function will return false</li></ul></li><li>eatFood<ul><li>If catalogFood returned true foodMenu will display options for the user to choose from</li><li>If the player chooses berries<ul><li>Remove the berries from the bag</li></ul></li><li>there chance to be poisoned is calculated<ul><li>If poisonous health is updated<ul><li>If player dies, let them know how</li><li>Else let them know they have been poisoned</li></ul></li><li>If regular berries update health</li></ul></li><li>If the player eats fish, update health, remove item from bag.</li></ul></li><li>canBuild()<ul><li>Checks if the user has all of the items for a raft and returns a boolean variable</li></ul></li><li>setWilson()<ul><li>Checks what items the user has in their inventory and provides hints based on what still needs to foundor done</li></ul></li><li>Wilson()<ul><li>If the player has wilson</li><li>Cycle through possible options for hints based on what items the player is missing</li></ul></li><li>isDead()<ul><li>Returns the value of dead</li></ul></li><li>getHealth()<ul><li>Returns the current health points</li></ul></li><li>getInventory()<ul><li>Returns a pointer to the users bag</li></ul></li></ul>

Island Class
Member Variables
• Pointer to Player • Pointer to Inventory • Pointer to current location • Pointer to each specific square • Character status • gameState variable • Vectors for holding main menu options • Vector for holding change location menu options
Member functions
• Constructor   ○ Create a new player, call the createBoard function   ○ Set the gamestate to in progress   ○ Have the bag pointer point to the player inventory • createBoard()   ○ Instantiate and link all of the spaces together   ○ Set the currentSpace pointer to initialize • Destructor   ○ Free memory used to create the board • Displayboard()   ○ Prints the map of the board   ○ Usings the current space pointer to place the player on the map • catalogMoves()   ○ Checks that a move is possible in each direction by looking at whether the pointer is null or not   ○ If a move is possible, add it to the vector of Option objects • changeLocation()   ○ Calls the catalog function and displays a list of the possible options   ○ Use switch statement   ■ Change location to new square   ■ Decrease health from character   ○ If the player has not died   ■ Call enteringSpace() for specific space   ○ Else prompt the user they have died • generateMenu()   ○ Push Option object (string and a unique number identifier) into the menu if that action is possible • Turn   ○ Display menu   ○ Get the user choice and return the key   ○ Switch statement to make choice   ○ Clear the menu vector for next turn   ○ If the player has died, update the gameState

TEST TABLE

Test	Input	Expected Outcome	Actual Outcome
Inventory Class			
addItem()	Instantiate an inventory object with a capacity of 2 for testing. Try to add 3 strings to the inventory.	Should give prompt that an item has been added to the bag if possible On the third try, prompt should be displayed indicating that the bag is full.	Says bag is full when no items are present ● Changed a relational operator
hasItem()	Use the previous test for addItem() Check that the 2 strings that were added to the inventory return a true value and the one that could not be added does not.		As expected
remItem()	Use the previous test input for hasItem() Remove the two strings that were successfully added and attempt removing the string that could not be added.	Should remove both items that can be removed and return true. Should return false for the item that was not successfully added to the inventory.	As expected
isFull()	Have inventory filled to capacity and print the result. Remove an item from the inventory and print the result	Should remove both items that can be removed and return true. Should return false for the item that was not successfully added to the inventory.	As expected
printInventory()	Use above tests to print out the results in the inventory between each test.	Should print in the correct order and shift elements as some are removed.	As expected Does not display anything if the inventory is empty - update to add prompt
getItemQty()	Enter varying levels of duplicates. Ex. 2 of one item, 3 of another.	Should print in the correct number of specific items in the inventory	As expected

Player Class			
updateHealth()	Pass in different values negative and positive values for changing the heath Inputs: 10, -10, -5, -100	Health variable should update accordingly If health dips below 0, should change the status of the player to dead Health should never exceed 100	Adding a value when health is full will allow user to exceed 100 ● Add an if statement to prevent values above 100 ● Also want to include setting updating health to 0 if it becomes negative for displaying purposes
catalogFood() && eatFood()	Create an inventory with just fish, just berries a combination of both and no food items. Choose from each food option and exit.	If only berries exist as food in the inventory, it should be the only option with exit. Same for fish. If no food is present, the menu should not display. Item eaten should be removed from the inventory	Listing all food options instead of just one when necessary ● No break statement in switch Items not removed from inventory - missing call to remItem in inventory
canBuild()	Test any combination of items that will not allow a raft. Test all specific items needed for a raft Input tarp, rope and 1 log Input tarp and 3 logs no rope etc.	Expect that the function will only return true when exactly 3 logs, a tarp and rope are in the inventory.	As expected
wilson()	In full program, pick up Wilson as the first item. Run through all of his prompts Pick up another item and see which events run	Expect tips to disappear as more game progress is made Events should be randomly displayed if there is more than 1 possible	As expected
Beach Class			
spaceAction() **Fishing - requires fishing pole**	Pass in a player to beach. Pass in a fishing probability of 100, 50, and 0.	Should always catch fish when probability is 100, no fish when probability is 0 and equal probability when value is 50.	As expected

	Repeat fishing action until the player dies from exhaustion.	If the player is weak enough, they should die of exhaustion. Expect the fish to be added to the players inventory	
explore()	Call the function twice.	Should report that an item has been found. Once the object is found, the prompt should not appear again. Item should be added to inventory and not added a second time.	As expected
Sparse Forest Class			
spaceAction() **Foraging - no items required**	Call the spaceAction function several times.	Berries should always be found and added to the inventory	As expected
explore()	Call the function twice.	No items to be found on this square so should report that nothing can be found.	As expected
randomEvent()	Call the function several times.	Based on aloe probability variable. Expect the event to trigger based on its value. Expect health to update accordingly.	As Expected
Dense Forest Class			
spaceAction() **Chop wood - requires axe**	Call the spaceAction function several times.	If the player is weak enough, they should die of exhaustion. Logs should always be added to inventory unless it is full.	As expected

explore()	Call the function twice.	No items to be found on this square so should report that nothing can be found.	As expected
randomEvent()	Call the function several times.	Based on snake probability variable. Expect the event to trigger based on its value. Expect health to update accordingly.	Health not being updated - Failed to call the health update function
Cave Class			

explore()
batFight()
&&
combat()
	Instantiate a cave space. Pass in a player that has the axe and another instance that does not have an axe.
	
	Test fighting and running
	
	Test running in the middle of a fight and then fighting again
	
	Try triggering the explore() event when the player already had rope in their inventory
		Player with the axe should do more damage than the player without the axe.
		
		If the player chooses to fight it should trigger one round of combat. If the player runs it should exit the menu
		
		If the player runs in the middle of the fight, their health should not regenerate, but the bats should
		
		If the player already has the rope it means the bat has been defeated and combat should not be triggered again.
			Turning to run does not exit out of the menu
			- Have to change logical operators (issues with && and ||)
			- Also switch form while to do while

Hut Class			
Hut instantiation	Create a hut object and use the explore function twice.	Correct prompt should be displayed for exploring. Tarp should be added to the inventory.	Turning to run does not exit out of the menu - Have to change logical operators (issues with && and ||) - Also switch form while to do while

Island Class			
Island constructor and createBoard()	Test moving the player to each possible space	Correct prompts should display. No segmentation faults.	As expected
catalogMoves && change location	Test moving the player to each possible space	Should give the correct options for movement if player chooses to move locations Movement should accurately update the board and health The second set of movement should clear the menu and generate it again (no repeat responses) Should display the correct next square description.	As Expected
turn()	Test each possible branch of the switch statement	Should trigger the correct action for each branch of the menu Test on several different squares to get each unique space action and with varying inventory contents	As expected
displayMap()	Move to each space on the board and display map at each turn	Should accurately update the users position on the map	As expected

Reflection
This was not necessarily the hardest project we had this term (Langton's ant was such a learning curve), but it was certainly the longest most comprehensive piece I have coded. The biggest roadblock I had was coming up with a theme for my program. I had a lot of unique ideas, but struggled to think of ways to implement them. Eventually I settled on the deserted island idea. I thought I would have to cut a lot of the features I originally planned, but was pleasantly surprised that I was able to incorporate nearly all of them (I'm a bit disappointed I didn't get to implement any ASCII art though).

The goal for the player is to collect a tarp, rope and 3 logs to craft a raft. Many of the action options available to the player are dependent on whether they are on a specific square and whether they have a specific item in their inventory. I chose this set up without really thinking about what a mess it would be for me to organize, it really increased the work necessary to get everything running, but I am very happy with the result.

Some issues I had along the way:
When I first started the design document I naively thought that coming up with the menu would be one of the easiest parts of the code to write, but it was actually the trickiest for me. I wanted to have a dynamic menu that would display only certain options to the user depending on specific circumstances in the game (ex. The player is in the woods and they have an axe then they can chop and collect wood). I could not for the life of me figure out how to provide the correct options and then to process the user choice to get the correct action. My TA gave me some great advice (thank you!), but I was not able to implement it due to time constraints as I already had much of the program coded. I decided to create a separate class that could link a string object with a unique numerical identifier. Then my menu could iterate through in descending order and instead of returning the value from the list, the identifier would be returned.

As I mentioned, I tend to struggle with organizing the code and determining which functions belong where. When coding the Cave class, I wasn't sure where to include the bat fight sequence. I considered randomEvent() and explore(). I decided that using randomEvent() would cause more trouble since I was letting the user choose between rounds whether to continue fighting or to run. If the player turned to run, I would have to move them to a new square and figure out how I would decide which Space they should move to. All of this seemed far too complicated for the time limit that I had so I put it in the explore() function which I think works wonderfully.

Another issue was displaying my map. I originally went with a more traditional island shape (ie. not a square) and probably with a lot more thought could have figured out a way to print the menu with updated player positioning, but with the time constraints settled on making a square map for simplicity. Even then the map ended up taking much more time to figure out than I anticipated, but I'm not dissatisfied with the end result.

I decided to change the mechanics of the game when it came to poison berries. If the bag was full, I was originally going to have the user be forced to eat them at that moment. Again, due to time constraints and some opinions from friends, I decided against this feature. It was easier to simply tell the user the bag was full and to calculate the probability of a poison berry at the time of eating rather than having to repeat the same code twice in different locations.

I learned that figuring out the balance of the game is hard. This is such a simple game, but I'm still not sure if the level of difficulty is adequate or too hard (I am biased because I know where all the items are hidden, my husband is biased because he watched me code this whole game and tell him every little detail so he's not the best guinea pig). I am glad I included all of the constants in a single file as it made it very easy to make adjustments when I felt necessary.

If I was going to use parts of this program for another game, I would redesign the constructors so that all classes could take strings as in the Beach class which would allow me to add items to every square to find. I could also assign the items to random squares each time, which could be fun and increase the difficulty (like I said, I always know where the axe and fishing rod are buried).

Overall, I found this a very rewarding (though painfully long) project. I am very happy with the end product, but even still, feel like there is more I could add (and probably will even after turning in the assignment).