Danielle DuChene
Final Project Reflection: Beat The Beatles

Original Design

Space Class: Base class
- minigame
  - gives player item if completed correctly
- initial interaction
- bool if game has been completed
- getters/setters
- pointers: up, down, left, right

Derived Classes
- Ringo : tic tac toe
  - check array if winner is in a line/tied
- Paul : trivia
  - two arrays, with questions and answers
- John : hang man
  - two arrays, check array with word and fill in empty array
  - array – keeps track of wrong letters
- George: Sandwich
  - assign points to each topping
- Beginning: a place to return to after each turn
- End: checks if items are obtained, if so ends game

Item
-name and description
~~- used status bool~~

Item Bag
- a linked list of items
- add
- remove
- print

Player
-keep track of space
-keep track of number of turns

Game progression
-ask if user wants to play
-prompt with menu
  - list the boys they can choose from
  - they can look at their items
    -they can delete item
  - test to see if be completed
  - quit the game
- go back to menu after each turn

Design Reflection:

As I was going along, I realized that the Space pointers would need access to their respective member functions. This caused me a bit of trouble, as sometimes they needed to do different things. Originally, I wanted some of the same functions in different classes to return different data types. But apparently virtual functions do not like that, as I found out. I ended up being able to maneuver my way around, although it made things a bit more complicated.

My first thought regarding the items was to set a bool about whether or not the item had been 'used.' At that point, I still did not have a solid idea on my project. When I decided on the "Beat The Beatles" theme, I decided that having a simple collection would be better as the minigames would already be complicated enough. Thankfully, changing that aspect of the program was simple as I just commented out the related variables and functions and tested to see if it still ran as intended – and it did!

Another issue I ran into was the creation of items. I thought about giving each of The Beatles their own ItemBag that the player could get their own items from, but I just couldn't figure it out. So, I ended up with a function that creates 3 items every time it's called. I know that this will take up more processing power, but I will have to study up on this type of thing over the break.

When I was coding the program, I originally had the items only have a name. I decided to add in the description later, because I thought it was fun. However, it turned out to be a major hassle. I had to figure out how to make an overloaded member function to add either an Item or an Item pointer an ItemBag. In the end, the description doesn't really add or subtract from the basis of the game and it is more of an easter egg. If I decided not to have the description, maybe I could have spent more effort on cleaning up the code.

---

TEST CASES

| **What is being tested?** | **Input** | **Expected Output** | **Actual Output** |
|---|---|---|---|
| game ending after 10 turns | go into spaces 10 times | game ends | game ends |
| game ending after 10 turns, checking bag and end are not included | go into spaces 5 times<br>check bag<br>check end<br>go into spaces 5 times | game ends after 10 turns into spaces | game ends as expected |
| game ends after 10 turns with the 4 items | go into spaces 10 times, collect the items | game ends with a win | as expected |
| game ends after 10 turns without the 4 needed items | go into spaces 10 times, not enough items | game ends with a los | as expected |

| | | | |
|---|---|---|---|
| remembers if you already won the item for each Beatle | try playing a Beatle again after winning his game already | asks user if they want to face the Beatle again | as expected |
| quitting game at various times | try to quit the game at various spots in the game | quits the game | as expected |
| PAUL / TRIVIA | | | |
| If user is wrong firsts time, asks one more time | input wrong answer | asks again for a response | as expected |
| user gets it right the first time | input right answer | gives item | as expected |
| wrong first time, wrong second time | input 2 wrong answers | asks again then kicks user out | as expected |
| RINGO / TIC TAC TOE | | | |
| Win vertical | win 3 vertical ways | gives item | as expected |
| Win horizontal | win 3 horizontal ways | gives item | as expected |
| win diagonal | win the two different ways to win diagonally | gives item | as expected |
| tie | play the game and make it result in a tie | does not give item | as expected |
| loss | lose to ringo | does not give you item, does not set interact bool | as expected |
| JOHN / HANGMAN | | | |
| does not accept repeated letters | repeat A, A, A, A, A | reprompt 4 times, does not count in wrong letters more than once | as expected |
| game completes when all letters are guessed | complete the word | gives an item | as expected |
| game ends when 5 wrong letters are guessed | guess 5 wrong letters | does not give an item, prints out losing screen | as expected |
| GEORGE / SANDWICH | | | |
| can only win if you choose egg | choose egg | gives item | as expected |

| | | | |
|---|---|---|---|
| “ | choose ham or cheese | does not give item, george is disappointed | as expected |
| COMPLETING GAME | | | |
| can only win with the 4 instruments | choose to load van with the 4 needed items | ask user to make sure if they want to load van, if Y: print win screen | as expected |
| “ | choose to load van without the items | Tell user to go back and win more games | as expected |
| INVENTORY MANAGEMENT | | | |
| can only have 9 items | try to get more than 9 items | prompt user to get rid of an item | as expected |
| remove item | choose to remove an item | removes the item, moves the other items in the bag to appropriate spots | as expected |
| Read description | choose to read description and pick the item | displays description | as expected |