Amy Suzuki
CS 162_400 Final Project

Game Design:

Necessary Functionality:

- Have Spaces (at least 3 Derived)
- Have Items
- Have a step limit
- Interaction between player and space using items
- Player can move through spaces.
- Player can win game.

Game Class

- Load Spaces based off file information. Connect spaces based off file information
- Set User to starting location
- Show introduction/instructions
- While under max steps and win conditions not met
    - Show adjacent space options
        - Check to see if spaces can be moved to
        - Move to available location
    - Check for new Items
        - Add item to bag if User choses
        - Replace item if bag is full
    - Increment steps
    - Check to see if win Conditions have been met
        - Check Items that player is carrying

Space Class

- Abstract class to hold space pointers
- Derived class that have the following restrictions
    - Cannot be passed through
    - Can hold Items
    - Have conditions upon entering the space (based on items)

Player

- Store Items and ItemList

Items

- String with name that can be checked by Spaces and Game

ItemList

- Linked list to store Items

Testing

Test	Purpose	Expected Result	Obtained Result
Game Win	See if game ends if win conditions met	Game ends when user brings back the correct painting to "Window" space	Game ends when user brings back the correct painting to "Window" space
Game Lost	See if game ends if they run out of steps	Game ends when maximum allowed steps are made. If user makes multiple erroneous steps, each one deducts from allowed steps but does not crash game.	Test 1: Memory error because int steps was not initiated to zero Test 2: Game ends when maximum allowed steps are made. If user makes multiple erroneous steps, each one deducts from allowed steps but does not crash game.
New Item from Space	See if space can "hand off" an item to user and user can store item in "bag" (Item linked list)	Space will return the new item memory. User will add item to linked list. Space item will be set to nullptr. No memory leak errors upon the end of the game.	Test 1: Memory error; double delete because user and space both have copy of Item. Test 2: Space will return the new item memory. User will add item to linked list. Space item will be set to nullptr. No memory leak errors upon the end of the game.
Replace item when bag is full	See if user can properly reallocate node to a new item and return old item. See if space can take old item and store item	User replaces object in linked list. Returns the Item being replaced. Space receives new item. If Space is Painting, space recognizes if item is or is not a painting. No Memory leaks	Test 1: Memory error, User not correctly reassigning object within linked list node. Test 2: Items properly replaced. No memory Leaks

Space Wall	Test canEnter()	User can never enter a wall	User can never enter a wall
Locked Room	Test canEnter()	User cannot enter without a key. Can enter with a key. Can re-enter without a key	User cannot enter without a key. Can enter with a key. Can re-enter without a key
Painting Space	Test Item Handling	User cannot obtain painting without proper item. User can leave other objects at Painting Space. Painting Space recognizes handles both items that are and are not Paintings	User cannot obtain painting without proper item. User can leave other objects at Painting Space. Painting Space recognizes handles both items that are and are not Paintings
Dark Room Space	Test canEnter()	User cannot enter with flashlight.	User cannot enter with flashlight.
hiddenObject Space	Test Item handling	User can obtain item or replace item with object from bag. Space can receive items.	User can obtain item or replace item with object from bag. Space can receive items.
Game Move() function	Check to see if user can move through spaces appropriately	User can choose between multiple spaces that are connected to their current location. Menu created dynamically to only show available options. Function prohibits user from entering room when conditions are not met.	User can choose between multiple spaces that are connected to their current location. Menu created dynamically to only show available options. Function prohibits user from entering room when conditions are not met.
Introduction()	Check to see if story is properly uploaded from file and buffered	Story is displayed to user at the beginning. Story has a “press enter to continue” function	Test 1: Error with first buffer (skips the first press enter to continue) Test 2: Story is displayed to user at the beginning. Story

			has a “press enter to continue” function
Player Replace Item	Check to see if player can handle new item when maximum number of items is met	Function offers player to replace item with another item in bag. No memory leaks. Returns old item to space.	Function offers player to replace item with another item in bag. No memory leaks. Returns old item to space.
Find Item (for Space)	Spaces that have required items can search a linked list and return if the item is found.	Spaces can handle receiving a linked list. Spaces traverse list and return false if item not found	Test 1: Memory error. Trying to reach memory outside of bounds. Test 2: Space can handle linked list and find items. Return false if item not found

Major Changes, Problems, and Solution

- Item Handling
    - Problem #1: Items stored in vector were difficult to remove once the bag was “full” and causing memory issues.
        - Items storage changed from vector to a linked list. If an item was being replaced, the stored object at a node would be replaced with the new item and the old item would be returned. Nullptr would be returned if the bag still had space.
    - Problem #2: Items that were replaced in bag would be “lost” and cause memory errors
        - Space that stored and handed items were redesigned to “receive” new items that were replaced by user and returned to space.
        - Items were always stored at a space or with user and the space and user deconstructor would delete items. No items were stored in two locations to avoid double deletes.
- Space Storage
    - Problem #3: Spaces were stored in a dynamically stored array which was difficult to resize and often had memory leaks
        - Space storage changed to a vector to allow new spaces to be easily created and added to the game. Since the order of the spaces stored in the game did not matter, new spaces could easily be pushed onto the vector

Extra Notes: Design Stages

Prototype 1: Player can move through spaces in Game

- Player Class
  - Variables:
    - Int steps
    - Space *location
  - Move(Space *Location)
    - Receive new location and increment steps
- Space Class
  - Variables:
    - Space *front
    - Space *back
    - Space *right
  - string prompt()
    - return prompt for space
- Game Class
  - Variables:
    - Player
    - Vector
  - Game()
    - Create Player
    - Set first
  - Move()
    - Check Player's current Location
    - Check available next locations and store in an array
    - Create a menu for choice with move choices
    - Allow user to select new location
    - Player::Move(location)

Changes:

- Added Game Deconstructor
- Added int choice to track how many options each space had to move
- Changes Space* array to track rooms into a vector

Additions:

- Add *left, *up, and *down Spaces to Space Class and adjust as necessary

Adjustment 1:

Prototype 1: Player can move through spaces in Game

- Player Class
  - Variables:
    - Int steps
    - Space *location

        - Item Array
    - Move(Space *Location)
        - Receive new location and increment steps
    - addItem()
    - removeItem(int choice)
        - re-adjust array
- Space Class
    - Variables:
        - Space *front
        - Space *back
        - Space *right
        - Space *left
        - Space *up
        - Space *down
        - Item*
        - leftItem*
    - string prompt()
        - return prompt for space
    - bool prereq(Item**)
        - get a list of items and see if item should be removed or not
- Game Class
    - Variables:
        - Player
        - Vector
    - Game()
        - Create Player
        - Set first
    - Move()
        - Check Player's current Location
        - Check available next locations and store in an array
        - Create a menu for choice with move choices
        - Allow user to select new location
        - Player::Move(location)
- Item Class
    - String name

Changes:

- Created a new menu for yes/no questions
- Realized an error that items that are removed cannot be recovered.
    - addItem() changed to return an Item address or nullptr to be "caught" by the space

Adjustment 2:

- Player Class
  - Variables:
    - Int steps
    - Space *location
    - Item Array
  - Move(Space *Location)
    - Receive new location and increment steps
  - addItem()
  - removeItem(int choice)
    - re-adjust array
- Space Class
  - Variables:
    - Space *front
    - Space *back
    - Space *right
    - Space *left
    - Space *up
    - Space *down
    - Item*
    - leftItem*
  - string prompt()
    - return prompt for space
  - bool prereq(Item**)
    - get a list of items and see if item should be removed or not
- Game Class
  - Variables:
    - Player
    - Vector
  - Game()
    - Set up spaces via text file
    - Create Player
    - Set first
  - Move()
    - Show Dialogue based off player's location, item, and time limit
    - Check Player's current Location
    - Check available next locations and store in an array
    - Create a menu for choice with move choices
    - Allow user to select new location
    - Player::Move(location)
- Item Class
  - String name

Note: Reading Spaces from file

- Create Temp Space via new

- Number
- Location Name
- Name
- Prompt
- Requirement
- Item Name else NONE
    - Create temp item via new
    - Assign name and assign to setObject()

Note: Connect Spaces from File

Major Change
Bag changed to linked list