Alice O’Herin  
CS162  
Final Project Design & Reflections  
6/5/2016

**DESIGN DESCRIPTION**

How did you approach the design? How did you understand the problem? How did you plan to solve the problem? Did you design on paper? Did you research information on the internet? Did you write out the solution on paper before entering it in the computer? For each step you didn’t take, please explain why.

Due to the larger scope of the Game, I spent a lot of time mapping out the design in the following process:

1. Listed out requirements. (See “Interpreting the Requirements.”)
2. Created a theme and created goals in keeping with the theme. (See Appendix C: Game Objectives.)
3. Brainstormed a space that could be interacted with in order to reach the items/goals. (See Appendix D: Map and “Rooms and Items.”)
4. Brainstormed items that could plausibly cause events that would add Items or remove obstacles to help get to the goal. (See “Rooms and Items” again.)
5. Broke all of the desired actions/causes into action categories (that would go on the Menu) and thought about how/what data would need to flow between Game, Room, and Item classes in order to achieve them. Since each Item had slightly different actions, I listed them out in order to make sure they were all possible. (See “Game Architecture.”)
6. Made adjustments to the design as necessary to reduce complexity, reuse code and reuse variables. (See “Additional Reflections.”)

***Interpreting the Requirements***

* *Theme/story* – Resentful of his new baby sister hogging all the attention, Tommy Tucker plans the best Mother’s Day ever. See game or Appendix A for more details.
* *Text-based puzzle where player moves through series of rooms/compartments, gathering items for some purpose* – You move through the house the house and yard, gathering ingredients from different spaces to make a perfect Mother’s Day surprise.
* *Abstract space class* – Room class
  + *4 pointer variables to other spaces* – north, east, south, west
  + *Data members* – Items\* LinkedList (objects in the Room), bool tracking whether: you can enter the room, the room is lit, the door or object is opened, etc.
  + *5 spaces of at least 3 different types* – There are 17 Room classes
  + *Special pure virtual function/action for space class* – use(Item\*) is a pure virtual function in Room implemented in each derived class
* *Track player’s location* – Game class has Room\* curr pointer.
* *Container to carry items* – Using the “dinosaur backpack” increases Item limit from 3 to 25.
  + *Has some limit*
  + *One or more items required for solution* – Yes, see Walkthrough below.
* *Change structure of space*
  + *Add a space* – When you use the “yellow ring,” a secret passageway is created/added north of the Nursery.
  + *Remove a space* – When you leave the Crib space, it is removed, since you climbed/fell out of the Crib and cannot get back in. (Conceptually, it is “inside of” or part of the Tommy’s Room space, but structurally it is represented as a separate Room in the game.)
* *Interact with parts of structure, not simply collect things* – You can open drawers and doors, pacify a dog or your crying baby sister, move aside a hen, climb up to and fill up a cup of orange juice, etc.
* *Have a goal* – Gather all required items (see Appendix C: Game Objectives or the game itself) and (optional) bonus item for your Mother’s Day surprise.
* *All actions taken through menu (no free-form input)* – Yes, see Appendix B: Menu Tutorial or game itself.
  + *Option to reveal goal* – Yes, option #7 from the Main Menu
* *“Time Limit”* – There are two time limits, both of which are dependent on the difficulty level you choose at the beginning. One limit is for the game overall – this is checked each time you finish taking an action from the main menu. The other limit is a challenge that starts up during the game: you have to get your sister to stop crying before she wakes your parents up.

(I ended up incorporating ***The Story***, ***Menu Tutorial****,* ***Game Objectives***, and ***Map*** into the game itself, and moved those sections to the very end of the Design Document as Appendices just so they are easy to reference.)

***Walkthrough***

* Getting Started / Getting Out
  + You begin the game “stuck” in the **Crib** space. If you try to move go west into Tommy’s Room, it will tell you that you need something soft to land on if you climb out.
  + **Pick up** and **use** your **stuffed animals**. They will create a cushion for your landing. **Move** west into Tommy’s Room. You will not be able to go back into the Crib.
  + **Pick up** your **backpack** and **use** it. You’ll want it for carrying around your stuff, and you can’t leave the room without it.
  + **Pick up** the **toy knife** and **use it** to pry off the child safety lock. **Move** west.
* Card
  + **Go** west to enter the Study. **Get** the **markers** on the desk. **Open** the **desk (drawer)** and **get** the **construction paper**. You will also want the **scissors** for the roses.
* Flowers
  + You should already have gotten the scissors from the Study.
  + **Go** to the nursery and **get** the **nightlight owl**. (If you try to Look or Get anything now, the room is dark.) **Use** the nightlight, so you can see in the outdoor spaces (backyard, etc.) in spite of the dark.
  + **Go** to the kitchen and **open** the **trash can**. When it falls over, pick up the **large ham bone**. **Open** the door in the second/south Living Room space so you can go into the Backyard. Go east and **use** the large bone before the dog scares you off!
  + **Use** the scissors to cut the flowers and **get** the **flowers**. You will also want to **get** the **gardening spade** in order to get the eggs later.
* (Quieting the Baby)
  + Getting the flowers triggers your baby sister to start crying (because she heard the barking). Depending on difficulty level, you will have 6, 4, or 3 minutes to quiet her before the game ends, and there will be warnings outputted as you get closer to the deadline.
  + **Get** the pacifier from the north/first Living Room space, and **use** it to comfort her, or dad will come and catch you.
* Orange juice
  + **Get** the painted pail from the first/north Backyard space.
  + **Get** the discarded sippy cup in the living room.
  + **Go** to the kitchen and **use** the pail. You will now be able to reach the orange juice, which appears as an Item in the room, but you will be unable to pick it up – it will suggest you use a container instead.
  + **Use** your sippy cup, and you will place it on the floor to pour the orange juice in. **Get** the **cup of orange juice**.
* Toast
  + **Go** to the Dining Room. **Open** the sideboard.
  + **Get** the **toast** you stowed away at dinner.
* Eggs
  + You should have already gotten the gardening spade from the Roses.
  + **Use** it in the chicken coop to parry the pecky chicken and **get** the **eggs**!
* Turkish delight
  + **Get** the yellow ring from the second/south Backyard space.
  + **Use** the yellow ring anywhere, and the **wardrobe** in the Nursery will become a passage.
  + **Open** the wardrobe in the Nursery, and **go** north.
  + **Get** the wool coat and **use** it. Otherwise, you will be forced to leave when you try to go north because it is too cold.
  + **Go** north and **get** the **Turkish delight**.

***Game Architecture***

main() initializes an instance of the class Game and calls its function play().

When Game is initialized, it dynamically allocates all of the Rooms in the game (minus the secret passage) and links them to one another. Game contains a pointer to the current Room, a LinkedList of items in the inventory, and other member variables that impact the entire game, such as:

* whether the player has (used) a light
* current number of, and maximum number of, items in the inventory
* time limits for the game and the baby crying
* boolean indicators of player’s progress in terms of collecting needed/bonus items

The main menu in Game has the following functions. As I was designing the game, I mapped through each of these actions to be sure that the information needed by each action was stored on / accessible by the appropriate object/function. I also checked Item by Item to make sure that the desired action by the Item could be taken.

* *Move*
  + Game calls the current Room’s showDirections() function, which returns the player’s selection as a pointer to the next Room or NULL (a direction with no exit, or door closed).
  + Game calls next Room’s enter() function, but if the return value is false (e.g., angry dog at the Roses), it calls its own showUseMenu() function, and if Room’s getStay() is still false (e.g., the used Item did not deter the dog), keeps current at the previous Room and re-enters() the Room (to display description and items again).
* *Get*
  + Item capacity is checked on the Game level.
  + If there is inventory capacity, Game calls the current Room’s showGetMenu() function, which returns the player’s selection as a pointer to the Item and removes the Item from the Room’s LinkedList of items. The Item returned will be NULL if the player decides to leave the menu without picking anything up, or fails to pick it up (only certain “special” item). Game class passes information on whether or not the player has a light so the Room knows what to display.
  + Game checks:
    - If the Item picked up is Roses, the baby crying start time / timer is triggered in its member variables.
    - If this Item completes all of the required items to win
* *Use*
  + Game calls its own showUseMenu function to display Items available for use in the player’s inventory. Game class passes information on whether or not the player has a light so the Room knows what to display.
  + Game checks Item’s activeRoom (-1 to be used anywhere or specific Room ID) and current Room ID to see if the Item can be used.
  + If the Item can be used, Game checks targetRoom.
    - If targetRoom is -1, then the Item affects a game-wide variable (such as total number of items that can be carried), so Game calls its own use(Item \*i) function to take the appropriate action depending on Item ID.
    - Otherwise, if targetRoom is not currentRoom, then it searches through all Rooms until it finds and returns a pointer to the correct one (or NULL if not found). The use(Item \*i) function of the targetRoom is called to take the appropriate action depending on Item ID.
* *Open* – Calls the current Room’s showOpenMenu() function. Game class passes information on whether or not the player has a light so the Room knows what to display. The room takes all necessary action within the Room’s function.
* *Look* – Calls the current Room’s print() function. Game class passes information on whether or not the player has a light so the Room knows what to display.
* *Map* – Simple text output of the map.
* *Goal* – Simple text output of goals. Checks Game private member variables to display if an item has been acquired, and outputs time limits on game or baby crying calculated from Game member variables.
* *Quit* – Simple variable change to break out of while loop.

***Rooms and Items***

In order to track each Room and its functions, I created the following table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Links | ID | Room name | Get? | Use? | Open? | Needs light to see? |
| 1 | 1 | Woods | Yes | No | No | No |
| 2 | 2 | Wardrobe | Yes | No | No | Yes |
| 2 | 3 | Hallway1 | No | No | No (Parents’ room not a real exit) | No |
| 1 -> 2 | 4 | Nursery | Yes | Yes | Yes (north – wardrobe) | No -> Yes |
| 1 | 5 | Study | Yes | No | Yes (desk) | No |
| 4 | 6 | Hallway2 | No | No | No | No |
| 2 -> 1 | 7 | Tommy’s | Yes | Yes | Yes (west – locked door) | No |
| 1 | 8 | Crib | Yes | Yes | Yes (exit is “closed” until you provide padding) | No |
| 2 | 9 | Kitchen | No -> Yes | Yes | Yes (trash) | No |
| 3 | 10 | Living1 | Yes | No | No | No |
| 2 | 11 | Dining | No -> Yes | No | Yes (sideboard) | No |
| 3 | 12 | Living2 | Yes | No | Yes (east – glass doors) | No |
| 3 | 13 | Backyard1 | Yes | No | No | Yes |
| 2 | 14 | Hedges | No | No | No | Yes |
| 1 | 15 | Roses | Yes | Yes | No | Yes |
| 2 | 16 | Backyard2 | Yes | No | Yes (south – chicken coop) | Yes |
| 1 | 17 | Chicken Coop | No -> Yes | Yes | No | Yes |

In order to make sure that each Item’s desired actions were in fact possible, I created the following table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Item Short | Item Long | Found Room | Active Room | Use |
| 10 | Turkish delight | A round box of Turkish delight spills out onto the snow. | Woods | None (0) | N/A - required object |
| 80 | a pile of stuffed animals | A ratty stuffed animal collection lies here in a pile. | Crib & Tommy’s | Crib | Triggers open = true so you’re willing to climb out of crib |
| 70 | dinosaur backpack | A half-zipped dinosaur backpack lies on the ground. | Tommy’s | Any (-1) | Grows inventory capacity to 25 |
| 71 | toy knife | A toy knife is wedged underneath the toy box. | Tommy’s | Tommy’s | Allows you to exit the room |
| 50 | two markers | A black and a red permanent marker are tucked under stray papers. | Study | None (0) | N/A - required object |
| 51 | construction paper | The edge of a slightly crumpled piece of blue paper peeks out of the drawer. | Study (open drawer) | None (0) | N/A – required object |
| 52 | scissors | A pair of rubber-grip scissors beckons enticingly. | Study (open drawer) | Roses | Cuts the roses so you can pick them up. |
| 160 | yellow ring | A very bright yellow ring hums with power. | Backyard2 | Nursery | Creates and connects the secret passageway in the nursery wardrobe |
| 40 | owl nightlight | An adorable owl-shaped nightlight glows pink. | Nursery | Any (-1) | Allows you to see in unlit places (e.g. outdoors) |
| 90 | large bone | A large ham bone lies amidst other trash. | Kitchen (push trash) | Roses | Allows you to distract the dog and stay in the room. |
| 151 | roses | Mrs. Spencer’s ruthlessly hewn off roses have fallen to the ground. | Roses | None (0) | N/A – required object. |
| 100 | pacifier | A plastic green pacifier is stuck between the sofa cushions. | Living1 | Nursery | Stops the baby from crying. |
| 120 | sippy cup | A plastic sippy cup lies in a small wet spot. | Living2 | Kitchen | Allows you to get the orange juice |
| 121 | sippy cup with orange juice | A sippy cup filled with delightful citrus goodness is on the floor. | Kitchen (after use sippy cup) | None (0) | N/A – required object |
| 122 | a carton of orange juice | A slightly dented carton of pulpy orange juice lies on its side. | Kitchen (after use pail) | None (0) | You cannot actually pick it up, so you cannot actually use it. But it is created as an Item so it shows up in the Get menu, since you would naturally try to pick it up. |
| 110 | leftover toast | The dried piece of toast you so cleverly hid is still in the sideboard. | Dining (open sideboard) | None (0) | N/A – required object |
| 130 | a painted metal pail | A bright metal pail is upended here, serving as a stool. | Backyard1 | Kitchen | You use it as a stool in order to reach the orange juice in the refrigerator. |
| 150 | gardening spade | A rusted gardening spade is stuck into the dirt. | Roses | Chicken Coop | Fends off the chicken while you get the eggs (already in the room). |
| 170 | eggs | Three speckled brown eggs are nestled in the nest. | Coop | None (0) | N/A – required object |
| 20 | wool coat | A wool coat hangs here, smelling of mothballs. | Wardrobe | Any (-1) | Allows you to stay in the Woods and not freeze. |

* ID numbers are by room found. So since Woods has ID 1 and Turkish delight is found there, Turkish delight is 10. The next item found in Woods would be 11.
* Active room of -1 means it is active anywhere. Active room of 0 means it can’t be used.  
  Target room of -1 means action is taken in Game class. Active room of 0 means no action is taken.

**TEST PLAN & RESULTS**

How did you approach testing your program? What data did you use? How did you ensure complete or good test coverage?

I re-used code for menus/input validation that had been extensively tested in the past, so I did not spend a lot of time testing those areas out.

I also did not extensively test functions such as the Game class’ displayIntro() and showMap() that were really just a bunch of couts, other than making sure there wasn’t anything strange going on with line wrapping. I did not include these simple text output functions in the testing chart below.

It was difficult to isolate down to a single function and I did not generally try to unless there were errors occurring. Many had to be tested together because in order to take certain actions, it involved using the menu, then using an item, to see using look/print() whether or not a room became light or dark or using move/showDirections() whether a new passageway was added. Then I tried to use other functions that were impacted by the room’s light/darkness or entrances/exits to make sure it all worked properly. The Look menu option / print() is not included in the below table since it was tested with each change (e.g. did “opening the drawer” create the items properly? Look. Did the hasLight variable work properly? Look to see. Did get work properly? Look.)

I built and tested incrementally in this order:

* Created all rooms and made sure I could walk through them. (Main Menu “Move” selection, Room’s showDirections(), enter() and print())
* Created Item class and placed Items in the correct Rooms. (I placed Items that would later be added when you “opened” a drawer, etc., directly into the Room.) Walked through to make sure they appeared correctly. (Room and Items’ print() functions.)
* Built out showGetMenu() and attempted to pick up Items.
* Built out showOpenMenu() and attempted to walk through before/after opening.
* Built out showUseMenu() and attempted to use Items. I tested each item individually – for example, I tested visiting/getting/using in dark rooms before and after removing/using the nightlight. I tested going into the Wardrobe before and after using the yellow ring (which creates the passage). And so on.
* Built out the rooms that “expel” you (Roses and Woods): stay boolean, enter() function, and use(Item \*i) to change stay bool to true.
* Winning: Added a check in Game for each of the needed items, allowing user to continue if the bonus item was not yet found.
* Losing: Timers, option to choose difficulty level (varied time limits), and warning messages
* Intro/exit displays, storyline, etc.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Tested Input / Scenario | Expected result | Actual result | Comments/Questions |
| **Input validation** | | | | |
| Menu for selecting difficulty level | Random characters, invalid ints, RETURN key | Returns error message or (for Return) simply outputs a line break. Continues to wait for input. | As expected | Moved a line break for better readability |
| play() / main menu | Random characters, invalid ints, RETURN key | Returns error message or (for Return) simply outputs a line break. Continues to wait for input. | As expected |  |
| displayAsMenu() used by both showUseMenu() in Room and showGetMenu() in Game | Random characters, non-menu ints, RETURN key | Returns error message or (for Return) simply outputs a line break. Continues to wait for input. | As expected |  |
| Select showOpenMenu() in a Room with something to open (e.g. Tommy’s Room) | Random characters, non-menu ints, RETURN key | Returns error message or (for Return) simply outputs a line break. Continues to wait for input. | As expected |  |
| getYesNo used for quitting and for keep playing | Random characters (not n or y), ints, RETURN key | Returns error. Waits for input. | As expected |  |
| **Move** | | | | |
| Moving normally in open spaces. Game play(); Room showDirections(), print() | Move to test each of four directions in each room | Able to move back and forth among all open spaces (including secret passage after adding using ring) in the Game, not able to walk in NULL directions (error message). | As expected, except... (see comments) | Seg fault when I tried to go South (invalid direction) from Nursery. This was because of a special condition that attempted to check the ID of a null pointer in the showDirections() override. Added a check for != NULL. |
| Crib space successfully deleted after moving to Tommy’s Room. Game play(); Room showDirections(), Crib enter() | Move west from (open) Crib to Tommy’s room | Crib correctly removed. showDirections() does not show East as a valid exit, and attempting to move East results in error message and staying in same space. (No memory leaks.) | As expected |  |
| Attempt to leave Tommy’s room before/after picking up backpack. Game play(); Tommys showDirections() | Move west from (open) Tommy’s room. Test this after having tested open works correctly (below). | Outputs error message (“You feel like you’re forgetting something...”) if try to leave without it. You are still in same room (can confirm by Look). Once you get it, proceeds normally. | As expected |  |
| **Open** | | | | |
| Walking into closed doors. Room showDirections(), bool open | Tommys (go west), Living2 (east), Backyard2 (south), Crib (west), Nursery (north after using ring), Hallway1 (west) | Display special message to explain why you can’t enter / what you need to do. Stay in the same room. | As expected |  |
| Open doors using showOpenMenu() (versus opening items like drawers). Room showOpenMenu(), bool open | Tommys (go west), Living2 (east), Backyard2 (south), Crib (west), Nursery (north after using ring), Hallway1 (west) | Displays message explaining/describing how you open the structure (or in case of Hallway1, how you do not open the structure). | As expected |  |
| Walking past/through a closed door once it has been opened by showUseMenu (using Item) or showOpenMenu (opening door/object). Game play(), showUseMenu(); Room showDirections(), showOpenMenu(), bool open | Tommys (go west), Living2 (east), Backyard2 (south), Crib (west), Nursery (north after using ring), Hallway1 (west) | Enter/exit as normal with no errors/messages. Some rooms require Item to be used to open. | At first, doors were staying closed. | I had forgotten to remove the derived class private member variable open after I moved to declare it in the main Room class, so the local copy was still false. |
| Call showOpenMenu() when open already is true for entrance/exit (both in Rooms where this changes and a Room where this is the starting value). | Call showOpenMenu() in same spaces as above. | Show error message (“There is nothing to open here.”) | As expected, except Tommy’s Room. | This room had a “locked” door such that you could try to open it in the menu, but you would not succeed. I was trying to setting the open boolean to closed even when it was already open. Instead I needed to be checking its previous value and restoring that. |
| **Stay** | | | | |
| Being expelled from a room. Tests Game play(), Room enter(), bool stay. | Enter Rose space or Woods space without using ham bone or wool coat respectively. | Should give the user one chance to use an Item, and if the Item cannot be used (wrong Item), forced to retreat to previous Room | As expected |  |
| Entering a Room that should expel you, but using the correct Item when prompted (dog) or beforehand (wool coat). Tests Game play(), Room enter(), bool stay | Enter Rose space or Woods space and use ham bone (in Rose space only) or wool coat (anywhere) respectively. Try re-entering. | Success message printed, user able to stay in the room. Future entry into the room is without incident. | As expected |  |
| **Get** | | | | |
| Get with light (default). Game play(), Room showGetMenu(), showUseMenu(), print() | Get any/all items in lit Rooms (can be used light or room light). | Success message printed, item added to your inventory (when you select Use menu option). Item is not there when you try to get() again or look. | As expected |  |
| Try to pick up orange juice. Game play(), showGetMenu(), showUseMenu(), print() | Get orange juice in Kitchen (after using pail). | Fail message printed, item not added to inventory, item is still in Room when you try to get() or look. | As expected |  |
| Get with light for Items added after open. Game play(), showOpenMenu(), showGetMenu(),  showUseMenu(), print() | Get items added after open: scissors and paper in Study, toast in Dining. | Item is there when you try to look or get and shows up successfully after the latter in the inventory for use. | As expected |  |
| showOpenMenu() when open already is true (for Rooms that add items). | Select Open in Study, Dining a second time. | Tells us there is nothing to open, no objects added. | Was adding the same objects again whenever showOpenMenu was called (after the desk/sideboard was already opened). | I wasn’t tracking change in the open variable before/after getting user input, just whether or not it was open. So I changed this to save the previous value and check for change. |
| showOpenMenu() – cancel | Select Open and Return to Main Menu | Returns us to the main menu without taking any action | As expected |  |
| **Light and Max Items (variables in Game)** | | | | |
| Get nightlight to test Room’s hasLight variable by going from lit -> dark in Nursery. Room showGetMenu(), showOpenMenu(), print(); Game showUseMenu. | Attempt to get, open, look, and use from main menu while in the Nursery both before and after getting the nightlight. | Before getting the nightlight, you can see normally. After you get the nightlight, you get messages about it being dark and (except in the case of use) take the action. | As expected |  |
| Use nightlight to test Game’s hasLight variable by going from dark -> lit. Room showGetMenu(), showOpenMenu(), print(); Game showUseMenu | Attempt to get, open, look, and use from main menu both before and after using the nightlight. | Before using the nightlight, you get messages about it being dark and (except in the case of use) cannot take the action. Afterwards, you can see and take action normally. | As expected |  |
| Max items. Game play() | Attempt to pick up > 3 items before using backpack container. | Get works normally (see above tests) until you try to pick up a 4th item. | As expected | It worked exactly as expected, but the player could get stuck with hands full before picking up the container – they’d be stuck in the game. So I created logic to force the user to pick up the backpack before leaving Tommy’s room. It MUST be one of the first 3. |
| Change in max items. Game play(), showUseMenu(), use(Item \*i) | Attempt to pick up > 3 items after using backpack. | Get continues to work normally after the 3rd item. | As expected |  |
| **Use** | | | | |
| *Stay boolean. Able to enter and stay in room normally after using ham bone, wool coat – This was already tested in the “Move” section.* | | | | |
| *Use an Item that can be used anywhere – This was already tested in the “Light” section (using nightlight and testing other actions) and “Move” section (using ring and testing passageway).* | | | | |
| *Use an Item whose targetRoom is not the Game and not the current room. Game showUseMenu(), findRoom(), resetSearch(); Room use(Item \*i) – This was already tested in the “Move” section (using ring and testing passageway).* | | | | |
| Use an Item that cannot be used. Game play(), showUseMenu() | Attempt to use construction paper, markers, toast, eggs, orange juice in sippy cup, flowers, and Turkish delight (goal items) | Returns error message, Items still in inventory | As expected |  |
| Use an Item in a Room other than its activeRoom. Game play(), showUseMenu() | Use gardening spade outside Coop, use pacifier outside Nursery, use sippy cup outside Kitchen, etc. | Returns error message, Items still in inventory. | As expected |  |
| Use an Item that creates a new Item | Use pail and sippy cup in Kitchen, scissors in Roses, gardening spade in Coop | Can Look and see, and Get (except orange juice) the Items added upon use | As expected |  |
| *Use an Item that creates Rooms (e.g. ring). Room use(Item \*i), showDirections(), open(), enter() -- This was already tested in the “Move” section (using ring and testing passageway).* | | | | |
| Kitchen use() | Use sippy cup in Kitchen before using the pail | Error message, sippy cup not used, no Items added | As expected |  |
| Nursery use(), babyCrying | Use pacifier in Nursery when baby is not crying | Error message, does not use item, which is still in inventory | As expected |  |
| babyCrying variable in Game. Use pacifier: baby stops crying, cry counter stops. Game showGoals(), Nursery print(). | After testing baby cry timer start, use the pacifier in the Nursery | Looking in baby room does not show baby crying in description. No more crying warnings, and after time limit runs out, game does not end. showGoals() no longer displays time remaining. | As expected |  |
| **Ending the Game: goals, timers, winning/losing** | | | | |
| Get all items but not Turkish delight, end game | Get the last required item before getting Turkish delight. You are asked if you want to continue the game. Select N for no. | Ends the game, displays winning ending story. | As expected |  |
| Get all items but not Turkish delight, keep playing, then get Turkish delight | Get the last required item before getting Turkish delight (bonus). You are asked if you want to continue the game. Select Y for yes. Get the Turkish delight. | Allows you to keep playing, ends the game when you get Turkish delight, displays (winning) ending story that includes Mom’s comment on Turkish delight. | As expected |  |
| Get all items but not Turkish delight, keep playing, then run out of time in Game | Get the last required item before getting Turkish delight. You are asked if you want to continue the game. Select Y for Yes but then run out of time. | Allows you to keep playing, ends the game when time runs out, displays winning ending story. | As expected |  |
| Get all items but not Turkish delight, keep playing, then run out of time with cryTime | Get all required items well before game time limit, picking up the Rose last. Select Y to keep playing, but don’t pacify the baby. Wait for baby’s crying timer to run out. | Allows you to keep playing, ends the game when cry time runs out, displays transition and winning ending story. | As expected |  |
| Already have Turkish delight when get the final required Item | Get the Turkish delight before getting the last required item. | No prompt to continue, goes straight to winning ending story (with comment on Turkish delight). | As expected, except... | The last Item that I got was the Roses, whose “getting” triggers the baby to start crying. So the game displayed all of the happy story and ending... and then the message on the baby crying. I added a check for !complete before the cout. |
| Lose by quitting | Select 8 on the main menu. Select Y for Yes to confirm. | Goes to losing (quitting) story. Ends game. | As expected |  |
| Game timer / lose by running out of time in the game | Play until the timer runs out without getting the required Items. | Use showGoals() and set external timer to check game timer (starting when you enter the first room). Gives you the correct amount of time to play depending on the level you selected, displays warning messages when expected (each msg only once), and prints the correct losing story | As expected |  |
| Baby crying timer / lose by running out of time while baby crying (not all required Items collected) | Without all other required items collected, pick up the Roses to trigger the baby crying / timer | Use showGoals() and set external timer to check babyCrying timer. Gives you the correct amount of time to quiet the baby depending on the level you selected, displays warning messages when expected, and prints the correct losing story | As expected |  |
| Room showGoals() function | Select Goals from the main menu | Correctly displays the items, whether or not you have gotten the Item, and the amount of time left in the Game and/or baby crying (also tested while testing timers) | As expected |  |
| ~Room() after searching | Quit the game after using ring, running on valgrind. | No memory leaks or errors (search and reset functions correctly implemented) | As expected |  |
| ~Room() and valgrind/memcheck | Run program using valgrind on FLIP | No memory leaks or errors | As expected | memcheck returned Invalid read of size 1 during the deallocation process, but all Rooms were still freed successfully and program ran to end. Unable to figure out this error after Googling and TA sessions. |

**ADDITIONAL REFLECTIONS**

***Circular References between Rooms and Items***

I knew that Rooms had to contain references to Items so it could know which Items to display or allow a player to get. At the same time, I wanted Items to contain a reference to the Room so it could manipulate the room as necessary when used – for example, add a rose object because you used a pair of scissors to clip the rose bushes.

But if I tried to use the simplistic solution of forward declaring (which seemed like bad style/design to me anyway), I would have had an incomplete type in the class that used the forward declaration rather than #include, which seem like it would cause its own set of problems.

In addition, adding pointers to Rooms within the Item objects would have made the initialization process more complicated.

To resolve this, I decided to have Items contain two different member variables: activeRoom and targetRoom. activeRoom was the Room ID in which an Item could be used (or -1 for anywhere, 0 for cannot be used) and targetRoom reflected the Room ID that the Item action had to be taken from (or -1 for action happening at Game level, such as expanding the inventory capacity, or 0 for cannot be used). Then use(Item \*i) would be called by the target Room (or by Game in special cases).

What this meant was that actions for a specific item were encapsulated in a Room rather than in the Item itself. I struggled with this idea at first, because in my head I thought, “The knife is what opens the door, not the Room itself.” But when I thought about it some more, I was actually very happy with this design, both because it made it much easier to implement actions like adding a secret passageway or unlocking a door directly from within the Room class, but also because it makes sense to encapsulate the action in the Room. Actually, being able to “clip roses” with scissors depends as much on being in the room “The Rose Garden” as it matters that you use a pair of scissors. In fact, one could design a system where the scissors can be used on multiple rooms, and then the action truly is Room-specific. Additionally, I think this was the original design of the assignment, which tells us to have a “special action” within the different Room classes.

This design decision around circular dependencies also influenced my decision to make Item a single class, rather than creating a separate “OrangeJuice” or “Roses” class. There was less or no need for Items to each have a set of their own functions because most of the relevant action was taking place within Room. In the end this simplified the coding by a lot, since the alternative would have meant 20 additional classes.

***Ambitions and Simplicity***

The danger of having an open-ended assignment such as the final project was scope creep: where the ambitions of the project balloon endlessly. This happens all of the time in real work environments, and is part of why technology consultants/professionals often spend a long time listing out exactly what is or isn’t required in a project.

As soon as I heard of this room / adventure-based text-based game, I thought in my head of the [MUDs](https://en.wikipedia.org/wiki/MUD) (Multi-User Dungeons) that I used to play, which were all text based. These were online worlds of hundreds or thousands of rooms, creatures, and objects. You could wear armor, cast spells, build up a wide variety of stats like Strength, Armor, HP, Wisdom, Intelligence, Dexterity, etc. Obviously, my game was not going to be anything like these games, but they still anchored my imagination when I thought about what my game was going to be.

In the end, I had to actively discard a lot of unnecessary functionality. For example, I thought about making it so that the player could “wear/hold” things – such as the light or the ring. (These would be saved as equipment the player was utilizing, and could be viewed/removed later.) I also planned to allow users to close (as well as open), drop (as well as pick up), remove (after wearing), and reuse (sometimes, after using once). All of these were functions available in the games I had played. Soon I realized that for the purposes of this game, many of these functions were not necessary. In my game, “using” an item successfully always deletes it, and I easily wrote this into the storyline: the scissors got stuck in the rosebushes, for example, or the pacifier is taken by the baby.

I originally had two separate functions for “opening” and “pushing.” But later I discarded “pushing” because I realized it was a function employed only in one room. Instead I adjusted the story so that when trying to “open” the trash can, you accidentally tipped it over, and the item “fell out.”

Additionally, all Room instances contain a single “open” member variable. This means that you couldn’t have 4 different closed doors in one room, for example, and in the generic base class code I could not automatically adjust descriptions based on the variable. (The base class would not know if “open” referred to the desk drawer, or the door to the east, or the wardrobe, etc.) This was another change that I made as I was programming, and again required slight changes in some of my descriptions, but I truly believe that simplicity is part of good design. Only one room would have used more than one “open” variable, and I saw value in a lightweight design.

Overall, without the “extra” actions, I think this makes the main menu much easier to navigate, as well.

***Using Linked Structures***

I used linked structures both to hold my Rooms and to hold my Items within the Rooms/Inventory.

For Rooms, this was less an active design choice and more a response to the requirements, which required a pointer in at least four directions. Sometimes an Item would be used in a Room that was not the Room that needed to take action based on its use. Hoping to avoid implementing a search function, I actually talked to a TA about storing the Room pointers in an array in addition to or instead of independent member variables in the Room class, but was discouraged from doing so based on Assignment requirements. This linked structure presented obstacles for searching and deleting.

Because the linked structure was not sorted, I wasn’t able to use tail recursion in the way that one might do for a binary tree; there was no way of knowing which cardinal direction would lead to / yield the result. I settled on introducing a “searched” member variable and sequentially calling the recursive search function on each cardinal direction that was not NULL and had not already been searched.

The destructor for Rooms required similar code, but I ran into a hitch in that there is no way to check if a particular Room is deleted; it does not automatically get set to NULL, so a Room pointing to a deleted Room would have some kind of undefined value there. Eventually I realized that I would have to set all pointers TO a particular Room to NULL before deleting the Room.

For Items, I used a LinkedList class based on previous code. This was helpful in that the objects within the inventory or any Room were constantly changing, and with LinkedLists it was easy to add and remove without having to shift every item over. Also, it was not necessary to copy/create a new array when the array got to a certain size – though its current size, the game could easily function with Room static arrays preset to size 5 and an inventory set to size 25 (this would have to be dynamic since using a container changes the inventory size).

For the purposes of creating a menu, though, an array would have been easier to use with its built-in indices. In the end I considered that both classes had their pros and cons, and chose to re-use the LinkedList code that I had already written for previous assignments.

However, I did have to write/modify some additional functions in order to display and select the LinkedList items based by number, which were somewhat inefficient since the List had to be iterated through each time. In retrospect I would probably implement the assignment with arrays.

**REFERENCES**

[Self Sufficient Me]. (2014, Apr. 9) *How to Collect Eggs From a Pecky or Aggressive Hen.* [Video File]. Retrieved from <https://www.youtube.com/watch?v=63Ef_c0q8yw>.

The “bonus item” portion of the storyline was inspired by C.S. Lewis’ well-known children’s book, *The Lion, the Witch, and the Wardrobe*,and its prequel *The Magician’s Nephew*. In the former, Lucy first discovers the magical world of Narnia while hiding in a wardrobe which leads her to a clearing with a lamppost. Later, the White Witch reigning over Narnia corrupts Edmund with a box of magical Turkish delight. In the prequel, the evil magician creates yellow and green rings which can draw its wearers/touchers to and away from alternate dimensions such as the magical world of Narnia.

**APPENDIX A: THE STORY**

The peaceful idyll of the home has been shattered for you, 3-year-old Tommy Tucker, with the arrival of The Creature, a very loud, very annoying wrinkled thing that seems to live in Mommy’s arms and lap. Last night, when It started sobbing during bedtime story time, was the last straw!

But you’ve come up with the perfect plan to win the Favorite Child blue ribbon: you’re going to make tomorrow the PERFECT Mother’s Day. Whether it’s outsmarting child safety locks, stealing construction paper from the study, or clipping your neighbor’s roses (guarded by her dog), you’ll stop at nothing. But careful, you’ll be in big trouble if you’re caught (by the dog or your parents), and The Creature has sharp ears (and a big mouth), too!

**APPENDIX B: MAIN MENU “TUTORIAL”**

1: Move -- Shows what is in each adjacent space, including if there is no exit, and gives you option to move there. It does not show whether a space is locked/closed.

2: Get -- Shows the objects in the room that you can pick up. You will not be able to see anything if it is a “dark” Room (like the Backyard or Chicken Coop) and you don’t have a light.

3: Use -- Shows the items in your inventory that you can use. Most items have a specific place that they can be used, but a few can be used anywhere on the map.

4: Open -- Shows the objects (doors, drawers, etc.) you can open. There is only one thing or direction to open in each Room.

5: Look -- Shows the room description and objects, which may be clues. Actions you have taken may reveal new objects in the room, and you can Look to see.

6: Map -- Displays a map of the world.

7: Goals -- Shows your goals, progress, and remaining time in the game (in seconds). If you have triggered the baby crying, then it will also show seconds until the baby’s crying wakes your parents up and ends the game.

8: Quit – If you confirm, ends the game without saving. Mom will be so disappointed!

**APPENDIX C: GAME OBJECTIVES**

Gather the ingredients for the perfect Mother’s Day before someone wakes up:

* Materials to make a card
  + Paper
  + Markers
* Ingredients for breakfast
  + Cup of orange juice
  + Toast
  + Eggs
* Fresh-cut flowers
* BONUS: Turkish delight

**APPENDIX D: MAP**

The Woods

|

Wardrobe

X

(Parents) X Hallway1 – Nursery  
 |

Study – Hallway2 X Tommy’s Rm - (Crib)

|

Kitchen – Living1

| |

Dining – Living2 X Backyard1 – Hedges – Roses (Dog)

|

Backyard2

|

Bird Coop