

David Rider

March 21, 2017

CS 162 Final Project

### **Reflection**

This was a fun project for me. I feel like I could utilize most of the things I learned throughout this class. I initially planned to have a game based around finding items in the corrected order. Being a new dad, I decided to go with a crying baby theme. In my game, there are six rooms in a house. There are four items randomly placed among the rooms. Each room can only contain, at most, one item. Also, one room contains the baby. The object of the game is to locate the item which will make the baby stop crying. However, there are some limitations placed on the user. Firstly, the user can only carry one item at a time. Secondly, the user can only drop an item in an empty room (i.e. a room without another item). Thirdly, there is a time aspect. The user selects a difficulty in the beginning, and that determines how many moves the user will be able to make before their partner returns and the game is over. Each time the user moves into a new room the counter will progress. There is certainly some strategy involved. I elected not to print a live display of the locations of the player and the baby, so that the player would be forced to rely upon their memory and the simple map of the house. I had some problems with one of my early implementations. I initially wanted the player to have to collect all four items. The baby would only stop crying if all four items were collected in the right order. However, I found that this severely slowed down the gameplay. Since I was using a stack in the Player class to hold my items, I just reduced the size to only one element. The other stack functions- `pop()`, `push()`, `display()`- in my Player class allows the player to drop, pick up, or see their item, while the `top()` function allows the `play()` function in the game class to check and see if the player's item is the one which the baby wants. The items themselves are derived classes from a

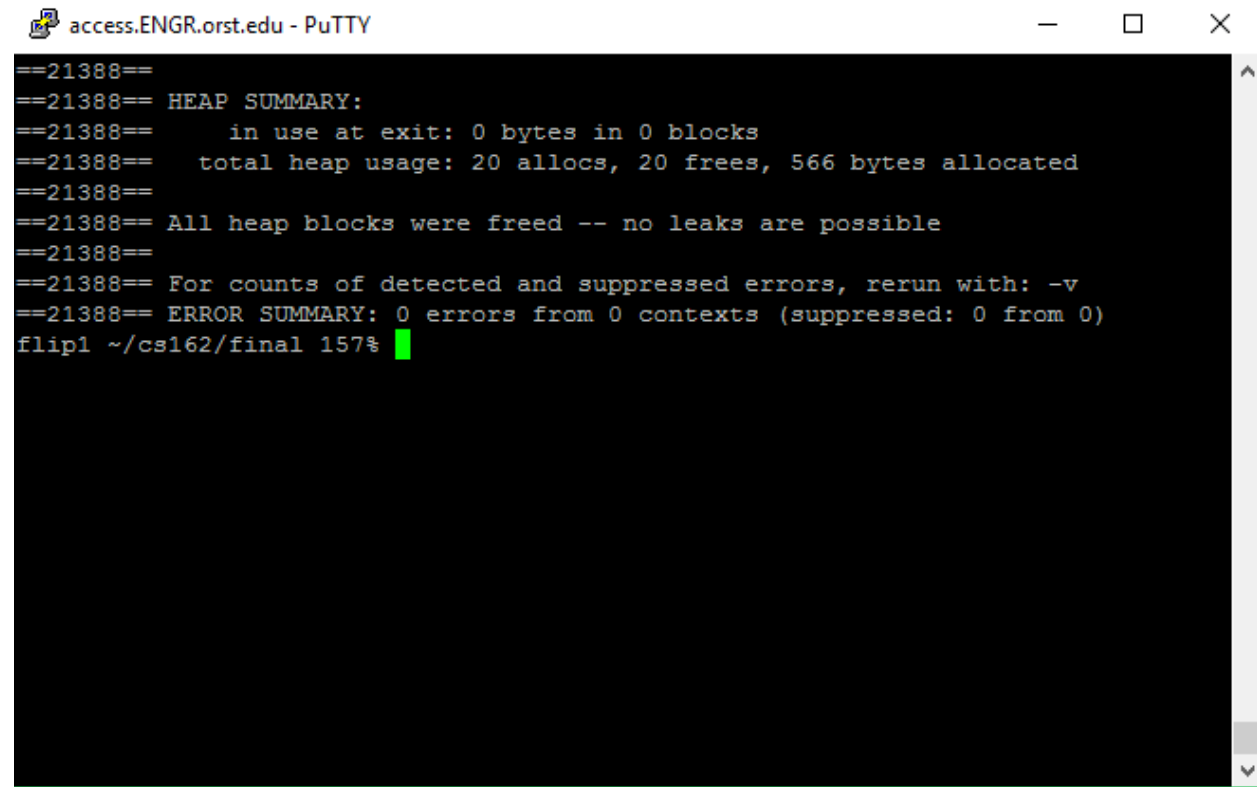
parent Item class. While the rooms of the house are derived from either one of three room types (bedRoom, passageWay, or commonRoom), which, in turn, are derived classes from the parent Space class. The creation of these parent and child classes were more or less straightforward. The real difficulty in this project was implementing my Game class, namely the Play( ) function. Given more time, I would decompose this function into smaller functions, as it is quite bulky. All in all, I think this project illustrates the concepts OOP, polymorphism, and pointers. If I had more time, I would likely expand the number of rooms, implement a difficulty option that requires more than one item to win, or add a second player to the game.

### Test Plan

| Test                | Input           | Expected Outcome                  | Observed Outcome                  |
|---------------------|-----------------|-----------------------------------|-----------------------------------|
| <b>Menu</b>         | Char value(a)   | Error message, prompt to re-enter | Error message, prompt to re-enter |
| "                   | Int value(3)    | Program quits                     | Program quits                     |
| "                   | Int value(2)    | Proceed to instructions           | Proceed to instructions           |
| <b>Difficulty</b>   | Char value(a)   | Error message, prompt to re-enter | Error message, prompt to re-enter |
| "                   | Int value(1)    | Difficulty set to amateur         | Difficulty set to amateur         |
| "                   | Int value(2)    | Difficulty set to medium          | Difficulty set to medium          |
| "                   | Int value(3)    | Difficulty set to hard            | Difficulty set to hard            |
| "                   | Int value( > 3) | Error message, prompt to re-enter | Error message, prompt to re-enter |
| <b>In game Menu</b> | Char value(j)   | Error message, prompt to re-enter | Error message, prompt to re-enter |

|                   |                |  |   |
|-------------------|----------------|--|---|
| "                 | Int value(0)   | Error message,<br>prompt to re-enter       | Error message,<br>prompt to re-enter    |
| "                 | Int value(1)   | Checks for item                            | Checks for item                         |
| "                 | Int value(2)   | Checks for baby                            | Checks for baby                         |
| "                 | Int value(3)   | Drops item                                 | Drops item                              |
| "                 | Int value(4)   | Leaves room                                | Leaves Room                             |
| "                 | Int value(5)   | Views item                                 | Views item                              |
| "                 | Int value(20)  | Proceed to round<br>input                  | Proceed to round<br>input               |
| <b>Take item</b>  | Char value(k)  | Error message,<br>prompt to re-enter       | Error message,<br>prompt to re-enter    |
| "                 | Char value(y)  | Item added to<br>stack or stack is<br>full | Item added to stack<br>or stack is full |
| "                 | Int value(100) | Proceed to<br>simulation                   | Proceed to<br>simulation                |
| "                 | Char value(n)  | Reprint menu                               | Reprint menu                            |
| <b>Leave Room</b> | Char value(k)  | Error message,<br>prompt to re-enter       | Error message,<br>prompt to re-enter    |
| "                 | Char value(y)  | Item added to<br>stack or stack is<br>full | Item added to stack<br>or stack is full |
| "                 | Int value(100) | Proceed to<br>simulation                   | Proceed to<br>simulation                |
| "                 | Char value(n)  | Reprint menu                               | Reprint menu                            |

## Memory Leak Check



The screenshot shows a PuTTY terminal window titled "access.ENGR.orst.edu - PuTTY". The terminal output displays the results of a memory leak check. The output is as follows:

```
==21388==  
==21388== HEAP SUMMARY:  
==21388==    in use at exit: 0 bytes in 0 blocks  
==21388==   total heap usage: 20 allocs, 20 frees, 566 bytes allocated  
==21388==  
==21388== All heap blocks were freed -- no leaks are possible  
==21388==  
==21388== For counts of detected and suppressed errors, rerun with: -v  
==21388== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)  
flip1 ~/cs162/final 157% █
```