



# Task 09 - Spike Summary Report



**Spike:** Task\_09

**Title:** Game Data Management

**Author:** Thomas Horsley, 103071494

## Goals & Deliverables

**Aim:** Develop an understanding around various data collection methods in relation to games programming. Specifically when to use them, how to implement them and best practices surrounding the data structures in question. From here, choose the most applicable data structure and implement an inventory system for Zorkish.

### Deliverables:

- Functioning Code
- Spike Report
- Short Report on Data Structures
- Git Commit History

## Technology, Tools and Resources

### Tech and Tools



The project was scripted in C++ 20 using Visual Studio Community 2022.

UML's and charts are made with [www.Lucidchart.com](http://www.Lucidchart.com)

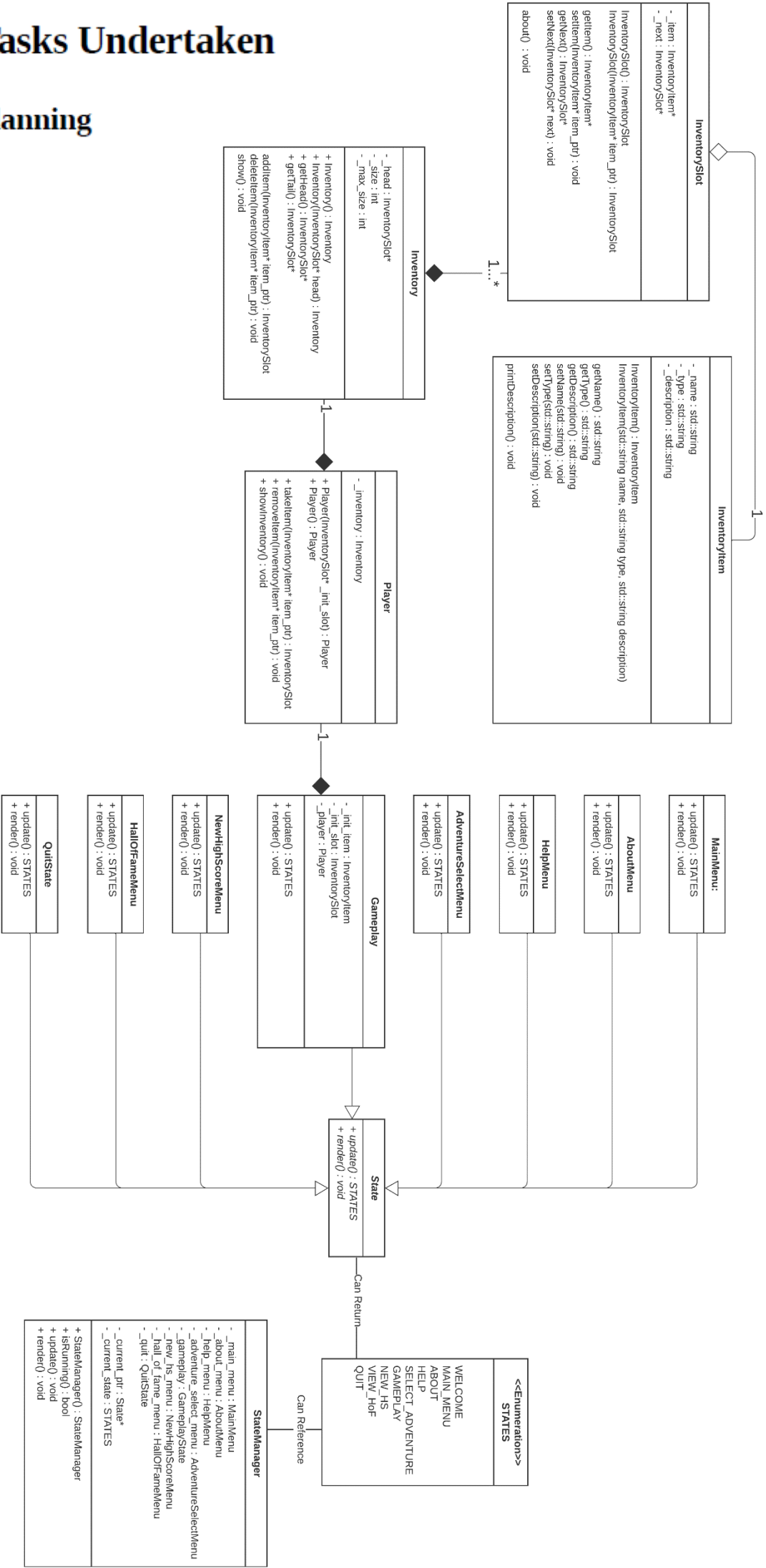
Source control is handled using Git.

### Resources

- Echo360 Lecture “Data Structures”
- Vectors vs Arrays, What’s the performance gap? <https://stackoverflow.com/questions/381621/using-arrays-or-stdvectors-in-c-whats-the-performance-gap>
- Comparison between Array, Linked Lists and Vectors: [https://vlsiuniverse.blogspot.com/2016/04/array-linked-list-vector.html#:~:text=Vector %3A Vector is a data,size let us say N.](https://vlsiuniverse.blogspot.com/2016/04/array-linked-list-vector.html#:~:text=Vector%3A%20Vector%20is%20a%20data,size%20let%20us%20say%20N.)
- Vector Performance Management: <https://www.acodersjourney.com/6-tips-supercharge-cpp-11-vector-performance/>

# Tasks Undertaken

## Planning





# Task 09 - Spike Summary Report

---

## Class Descriptions and Notes

Before discussion regarding the planning phase of the Zork(ish) inventory system it's important to note a change regarding the UML and final solution. The only notable mention in this regard is that addItem (and implicitly it's wrapper functions) no longer return an InventorySlot but rather construct a new InventorySlot on the Heap before assigning a reference to said InventorySlot to the lists tail. This lovely UML mistake cost 2 days of headache as the InventorySlot was falling out of scope when the function returned resulting the slot being lost and the data unreadable, a true emotional rollercoaster.

InventoryItems are simple placeholder classes for the moment containing three string variables for name, type and description, an InventoryItem has the ability to write these values. The Inventory itself (discussed below) is a collection of InventorySlot items each containing a reference to an item, the next slot in the list, a set of properties for the Inventory (manager class) to assign and reference and the ability to print it's own values.

The Inventory class is the Node manager for the Forward Linked List data structure which has been implemented to manage the inventory. The Inventory holds the functionality to add new InventorySlot items to the list, traverse the list and find Nodes containing InventoryItems and remove those items from the list with query methods. All of this functionality is encapsulated within a Player object which contains one reference to an Inventory and wrapper functionality for that Inventory item.

---





# Task 09 - Spike Summary Report


## Implementation

### Git Commit History



**Commits**


 **main** ▾


 Commits on Aug 24, 2023

Finished Task 09 - Spike 'Game Data Structures' 



unknown committed 18 hours ago


 2fef4ec 


 Commits on Aug 22, 2023

Fixed bug which allowed handling of incorrect input data leading to i... 



unknown committed 3 days ago


 9f6565d 

 Commits on Aug 21, 2023



Bug fix time: 


unknown committed 3 days ago

 f2b2c9e 



[Implemented the Player and Inventory systems into the Zorkish Gamepla...](#) 


unknown committed 3 days ago

 45c0f5b 



Refactored the working inventory code 


unknown committed 3 days ago

 80df400 



Inventory system complete. Next step is Zorkish implementation. 


unknown committed 3 days ago

 3f79250 



Merge branch 'main' of <https://github.com/KingSchlock/COS30031-2023-1...> 


unknown committed 4 days ago

 d815108 


Added InventorySlot (Node) class.class 



unknown committed 4 days ago


 4c471dd 

 Commits on Aug 20, 2023



Started refactoring Task 04 - Spike 'Gridworld Multithreaded'


 KingSchlock committed 4 days ago

 c37d3cb 

Started transition from Vectors to Linked Lists. 



unknown committed 4 days ago

 dfc4074 

 Commits on Aug 19, 2023



Started Task 09 - Spike 'Game Data Management'.

unknown committed last week

 c9204ed 

Started Task 09 - Spike 'Game Data Management'.

unknown committed last week

 d274568 



# Task 09 - Spike Summary Report

## Code

```
136 //Gameplay state definitions
137 STATES GameplayState::update() {
138     std::string action;
139     std::string object;
140
141     std::cout << ">> Action: ";
142     std::cin >> action;
143     std::cout << ">> Object: ";
144     std::cin >> object;
145
146     std::cout << std::endl;
147
148     // Using this big chunk of if-else statements until command processor is build
149     if (action == "quit" && object == "game") { return STATES::QUIT; }
150     else if (action == "show" && object == "highscores") { return STATES::VIEW_HoF; }
151     else if (action == "show" && object == "inventory") {
152         _player.showInventory();
153         return STATES::GAMEPLAY;
154     }
155     else if (action == "take" && object == "twigs") {
156         _player.takeItem(_bundle_of_twigs_ptr);
157         return STATES::GAMEPLAY;
158     }
159     else if (action == "take" && object == "petrol") {
160         _player.takeItem(_petrol_and_lighter_ptr);
161         return STATES::GAMEPLAY;
162     }
163     else if (action == "drop" && object == "twigs") {
164         _player.removeItem(_bundle_of_twigs_ptr);
165         return STATES::GAMEPLAY;
166     }
167     else if (action == "drop" && object == "petrol") {
168         _player.removeItem(_petrol_and_lighter_ptr);
169         return STATES::GAMEPLAY;
170     }
171
172     else { return STATES::GAMEPLAY; }
173 }
```

The update Gameplay state containing a placeholder command system which is extremely limited. However the implementation of taking and remove items from the inventory is abstracted nicely.

```
Main.cpp  Inventory.h  X
GameDataStructures  -  Inventory
1  #pragma once
2  #include "InventorySlot.h"
3  #include "InventoryItem.h"
4
5  class Inventory {
6  private:
7      InventorySlot* _head = nullptr;
8      int _size = 0;
9      int _max_size = 64;
10
11  public:
12      Inventory();
13      Inventory(InventorySlot* head);
14
15      InventorySlot* getHead();
16      InventorySlot* getTail();
17
18      void addItem(InventoryItem* item_ptr);
19      void deleteItem(InventoryItem* item_ptr);
20      void show();
21  };
22
23
```

The Inventory class is the linked list manager with functionality to find the tail, add and remove items

```
Main.cpp  Inventory.h  X
GameDataStructures  -  Inventory
31
32 void Inventory::deleteItem(InventoryItem* item_ptr) {
33     InventorySlot* previous_ptr = nullptr;
34     InventorySlot* traversal_ptr = _head;
35
36     while (traversal_ptr->getItem() != item_ptr) {
37         previous_ptr = traversal_ptr;
38         traversal_ptr = traversal_ptr->getNext();
39     }
40
41     previous_ptr->setNext(traversal_ptr->getNext());
42     traversal_ptr->setNext(nullptr);
43     _size--;
44 }
45
46 void Inventory::show() {
47     std::cout << "Zork(ish) :: Your Inventory " << std::endl;
48     std::cout << "-----" << std::endl;
49     std::cout << std::endl;
50
51     InventorySlot* traversal_ptr = _head;
52     while (traversal_ptr != nullptr) {
53         traversal_ptr->getItem()->printDescription();
54         std::cout << std::endl;
55         traversal_ptr = traversal_ptr->getNext();
56     }
57 }
```

```

InventorySlot.cpp  InventorySlot.h
GameDataStructures (Global Scope)
1  #pragma once
2  #include "InventoryItem.h"
3
4  class InventorySlot {
5  private:
6      InventoryItem* _item;
7      InventorySlot* _next;
8
9  public:
10     InventorySlot();
11     InventorySlot(InventoryItem* item_ptr);
12
13     InventoryItem* getItem();
14     void setItem(InventoryItem* item_ptr);
15     InventorySlot* getNext();
16     void setNext(InventorySlot* next);
17
18     void about();
19 };
20
21

```

The Node is represented through the InventorySlot object. A very basic object whose whole purpose is to hold an object and a reference to another slot.

```

InventorySlot.cpp  InventorySlot.h
GameDataStructures (Global Scope)
1  #include <iostream>
2  #include "InventorySlot.h"
3  #include "InventoryItem.h"
4
5  InventorySlot::InventorySlot() {
6      _item = nullptr;
7      _next = nullptr;
8  }
9
10 InventorySlot::InventorySlot(InventoryItem* item_ptr) {
11     _item = item_ptr;
12     _next = nullptr;
13 }
14
15 InventoryItem* InventorySlot::getItem() { return _item; }
16
17 InventorySlot* InventorySlot::getNext() {
18     if (_next != nullptr) { return _next; }
19     else { return nullptr; }
20 }
21
22 void InventorySlot::setItem(InventoryItem* item_ptr) { _item = item_ptr; }
23 void InventorySlot::setNext(InventorySlot* next_ptr) { _next = next_ptr; }
24
25 void InventorySlot::about() {
26     std::cout << "data: " << _item << std::endl;
27     std::cout << "next: " << _next << std::endl;
28 }
29
30

```

There's still a few things left to make this Inventory system work in a game setting which will be implemented for the next stage of Zorkish. This includes garbage collection on the inventory, the destructor and node deletion wasn't implemented as the game loop is so short the inventory closes before anything too bad happens...

## What was Learned?



This Spike revolved around understanding different methods of data encapsulation within a game, with emphasis on the decision making process regarding choosing the most correct container under a given circumstance.

Additionally, the task allowed me to

- Refresh my UML knowledge
- Understand pointers and references in greater depth
- Improve debugging skills
- Expand the scope of my code performance knowledge to include container types
- Make an Inventory for a game :)