```
1  #include "../../zorkish/hdr/GameData.h"
2
3  #pragma once
4  enum CommandType {
5      INVALID,
6      MOVE,   // go somewhere
7      TAKE,   // transfer item
8      LOOK,   // display surrounding renderer info
9      SHOW,   // display attached component info
10     QUIT,   // change game_data is_running to false
11 };
12
13 class Command {
14 public:
15     virtual void triggerEvent() = 0;
16 };
17
```

```
1  #include <string>
2  #include <vector>
3
4  #pragma once
5  enum class ComponentFlag {
6      C_INVALID,
7      C_RENDER,
8      C_INVENTORY,
9      C_PORTAL,
10 };
11
12 /*  Future optimizations could include allowing the components to know which
13 *   Entity they're associated with. This could trigger flags if the entity has
14 *   been manipulated and only entites which have been flagged via command will
15 *   be updated and rendered to the user.
       */
16 class Component {
17 public:
18     virtual ComponentFlag getFlag() = 0;
19     virtual std::vector<std::string> getInfo() = 0;
20
21     virtual void onEvent() = 0;
22 };
23
24
```

```cpp
#include <map>

#include "../hdr/Entity.h"
#include "../../Components/hdr/C_Render.h"
#include "../../Components/hdr/C_Inventory.h"
#include "../../Components/hdr/C_Portal.h"


/*  This is the dataset which gets instantiated in the AdventureSelectMenu
*   before being passed to the GameplayState for play-time manipulation
    */

//  Try character maps floats are fucking shit
#pragma once
struct GameData {
    bool is_running = true;
    bool reinstance_local_entity_cache = true;
    bool discovered_area = false;

    std::string player;
    std::string current_location;
    std::map<std::string, Entity*> entities;

    std::map<std::string, C_Render*> c_renderers;
    std::vector<C_Render*> _local_renderers;

    std::map<std::string, C_Inventory*> c_inventories;
    std::vector<C_Inventory*> _local_inventories;

    std::map<std::string, C_Portal*> c_portals;
    std::vector<C_Portal*> _local_portals;
};


```

```
1  #include "../../commands/hdr/Command.h"
2
3  #pragma once
4  struct InputData {
5      CommandType c_type = CommandType::INVALID;
6      std::vector<std::string> args = {};
7
8      C_Inventory* src = nullptr;
9      C_Inventory* dest = nullptr;
10     Entity* item = nullptr;
11 };
```

```
#include "WorldLoader.h"
#include "EventDispatcher.h"
#include "InputHandler.h"

#pragma once
enum STATES {
    S_WELCOME,
    S_MAIN_MENU,
    S_ABOUT,
    S_HELP,
    S_SELECT_ADVENTURE,
    S_GAMEPLAY,
    S_NEW_HS,       //New Highscore
    S_VIEW_HoF, //View Hall of Fame
    S_QUIT,
};

class State {
public:
    GameData* _game_data = nullptr;

    virtual void setStateData(GameData* game_data = nullptr,
        std::vector<std::string> args = {}) = 0;

    virtual STATES update() = 0;
    virtual void render() = 0;
};


class MainMenu : public State {
public:
    void setStateData(GameData* game_data = nullptr,
        std::vector<std::string> args = {}) override;


    STATES update() override;
    void render() override;
};

class AboutMenu : public State {
public:
    void setStateData(GameData* game_data = nullptr,
        std::vector<std::string> args = {}) override;

    STATES update() override;
    void render() override;
};

class HelpMenu : public State {
```

```
50 public:
51     void setStateData(GameData* game_data = nullptr,
52         std::vector<std::string> args = {}) override;
53 
54     STATES update() override;
55     void render() override;
56 };
57 
58 class AdventureSelectMenu : public State {
59 private:
60     WorldLoader* _world_loader = nullptr;
61 
62 public:
63     AdventureSelectMenu();
64     ~AdventureSelectMenu();
65 
66     void setStateData(GameData* game_data = nullptr,
67         std::vector<std::string> args = {}) override;
68 
69     STATES update() override;
70     void render() override;
71 };
72 
73 class GameplayState : public State {
74 private:
75     bool first_pass = true;
76 
77     InputData* _input_data = new InputData();
78     InputHandler* _input_handler = nullptr;
79     EventDispatcher* _event_dispatcher = nullptr;
80 
81 public:
82     GameplayState();
83     ~GameplayState();
84 
85     void setStateData(GameData* game_data = nullptr,
86         std::vector<std::string> args = {}) override;
87 
88     STATES update() override;
89     void render() override;
90 
91 private:
92     std::string extractUEID(std::string ucid);
93     void resetRender();
94 };
95 
96 class NewHighScoreMenu : public State {
97 public:
98     void setStateData(GameData* game_data = nullptr,
```

```
 99          std::vector<std::string> args = {}) override;
100
101      STATES update() override;
102      void render() override;
103  };
104
105  class HallOfFameMenu : public State {
106  public:
107      void setStateData(GameData* game_data = nullptr,
108          std::vector<std::string> args = {}) override;
109
110      STATES update() override;
111      void render() override;
112  };
113
114  class QuitState : public State {
115  public:
116      void setStateData(GameData* game_data = nullptr,
117          std::vector<std::string> args = {}) override;
118
119      STATES update() override;
120      void render() override;
121  };
```

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include "../hdr/States.h"

/*****************************************************************************
*                                  MAIN MENU
*                                 DEFINITIONS
*
*****************************************************************************
*                              base class overrides
*****************************************************************************/
void MainMenu::setStateData(GameData* game_data,
    std::vector<std::string> args) {
}

STATES MainMenu::update() {
    int choice = 0;
    std::cin >> choice;

    if (!std::cin.fail()) {
        std::cout << std::endl;

        switch (choice) {
        case 1: return STATES::S_SELECT_ADVENTURE;
        case 2: return STATES::S_VIEW_HoF;
        case 3: return STATES::S_HELP;
        case 4: return STATES::S_ABOUT;
        case 5: return STATES::S_QUIT;
        default: return STATES::S_MAIN_MENU;
        }
    } else {
        std::cout << "Enter a value which correlates with a menu option."
          << std::endl << std::endl;

        std::cin.clear();
        std::cin.ignore();
        return STATES::S_MAIN_MENU;
    }
}

void MainMenu::render() {
    std::cout << "Zork(ish) :: Main Menu" << std::endl;
    std::cout <<
```

```
        "-------------------------------------------------------------" << std::endl;
44
45      std::cout << std::endl << "Welcome to Zorkish Adventures!" << std::endl;
46
47      std::cout << std::endl << "1. Select Adventure and Play" << std::endl;
48      std::cout << "2. View Hall of Fame" << std::endl;
49      std::cout << "3. Help" << std::endl;
50      std::cout << "4. About" << std::endl;
51      std::cout << "5. Quit" << std::endl;
52
53      std::cout << std::endl << "Selected 1-5: " << std::endl;
54  }
55
56
57
58  /*****************************************************************************
59  *                                  ABOUT MENU
60  *                                  DEFINITIONS
61  *
62  *****************************************************************************
63  *                              base class overrides
64  *****************************************************************************/
65  void AboutMenu::setStateData(GameData* game_data,
66      std::vector<std::string> args) { }
67
68  STATES AboutMenu::update() {
69      return STATES::S_MAIN_MENU;
70  }
71  void AboutMenu::render() {
72      std::cout << "Written by: Thomas Horsley (103071494)" << std::endl;
73      system("pause");
74      std::cout << std::endl;
75  }
76
77
78
79  /*****************************************************************************
80  *                                  HELP MENU
81  *                                 DEFINITIONS
82  *
```

```
83 /*****************************************************************************
84 *                              base class overrides
85 *****************************************************************************/
86 void HelpMenu::setStateData(GameData* game_data,
87     std::vector<std::string> args) { }
88
89 STATES HelpMenu::update() { return STATES::S_MAIN_MENU; }
90 void HelpMenu::render() {
91     std::cout << "The following commands are supported: " << std::endl;
92     std::cout << ">> quit" << std::endl << ">> highscore (for testing)" <<
       std::endl;
93     system("pause");
94     std::cout << std::endl;
95 }
96
97
98
99 /*****************************************************************************
100 *                            ADVENTURE SELECT MENU
101 *                                 DEFINITIONS
102 *
103 *****************************************************************************/
104 *                                De/Constructors
105 *****************************************************************************/
106 AdventureSelectMenu::AdventureSelectMenu() {
107     if (_world_loader == nullptr) { _world_loader = new WorldLoader(); }
108 }
109
110 AdventureSelectMenu::~AdventureSelectMenu() {
111     if (_world_loader != nullptr) {
112         delete _world_loader;
113         _world_loader = nullptr;
114     }
115 }
116
117
118
119 /*****************************************************************************
120 *                                Base Class Overrides
```

```
121 ************************************************************************
      ****/
122 void AdventureSelectMenu::setStateData(GameData* game_data,
      std::vector<std::string> args) { }
123
124 STATES AdventureSelectMenu::update() {
125     int choice;
126     std::cin >> choice;
127
128     if (!std::cin.fail()) {
129         std::cout << std::endl;
130
131         switch (choice) {
132         case 1:
133             _world_loader->setSaveFile("Zorkish/saves/test_save.txt");
134             _game_data = _world_loader->loadGameData();
135             return STATES::S_GAMEPLAY;
136         case 2:
137             std::cout << "Wow this worlds pretty cewl. Too bad it doesnt
                load"
138                 << std::endl;
139             return STATES::S_SELECT_ADVENTURE;
140         case 3:
141             std::cout << "This is the coolest non-functional world i've
                ever seen!"
142                 << std::endl;
143             return STATES::S_SELECT_ADVENTURE;
144         }
145     }
146
147     else {
148         std::cout << "Please enter a value correlating with the options"
            << std::endl << std::endl;
149
150         std::cin.clear();
151         std::cin.ignore();
152     };
153     return STATES::S_SELECT_ADVENTURE;
154 }
155
156 void AdventureSelectMenu::render() {
157     std::cout << std::endl << "Zork(ish) :: Select Adventure " <<
          std::endl;
158     std::cout <<
          "-------------------------------------------------------------" <<
           std::endl;
159
160     std::cout << std::endl << ">> 1. World (The loading one)" <<
          std::endl;
```

```
161     std::cout << ">> 2. Cool World" << std::endl;
162     std::cout << ">> 3. Even COOLER World" << std::endl;
163 }
164
165
166
167
168 /*****************************************************************************
169 *                                GAMPLAY STATE
170 *                                 DEFINITIONS
171 *
172 /*****************************************************************************
173 *                               De/Constructors
174 *****************************************************************************/
175 GameplayState::GameplayState() {
176     if (_input_handler != nullptr) {
177         delete _input_handler;
178         _input_handler = nullptr; }
179
180     if (_event_dispatcher != nullptr) {
181         delete _event_dispatcher;
182         _event_dispatcher = nullptr; }
183
184     _input_handler = new InputHandler();
185     _event_dispatcher = new EventDispatcher();
186 }
187
188 GameplayState::~GameplayState() {
189     if (_game_data != nullptr) {
190         delete _game_data;
191         _game_data = nullptr; }
192
193     if (_input_handler != nullptr) {
194         delete _input_handler;
195         _input_handler = nullptr; }
196
197     if (_event_dispatcher != nullptr) {
198         delete _event_dispatcher;
199         _event_dispatcher = nullptr; }
200 }
201
202
203
204
```

```
205 /*************************************************************************
206 *                                 Private
207 *************************************************************************/
208 void GameplayState::resetRender() {
209     std::map<std::string, C_Render*>::iterator it;
210 
211     for (it = _game_data->c_renderers.begin();
212         it != _game_data->c_renderers.end(); ++it) {
213         it->second->flagForRender(false);
214         it->second->doShallowRender(true);
215 
216         std::string UEID = extractUEID(it->second->getUCID());
217         if (UEID.front() == _game_data->current_location.front()) {
218             it->second->flagForRender(true);
219             it->second->doShallowRender(_game_data->discovered_area);
220         }
221     }
222 }
223 
224 std::string GameplayState::extractUEID(std::string ucid) {
225     // Check each character of the cuid until we hit an upper case letter
226     // take all the characters from before the upper case and that's our
        EUID
227     // to return.
228     std::string::iterator cuid_it;
229     std::string UEID;
230 
231     for (cuid_it = ucid.begin(); cuid_it != ucid.end(); cuid_it++) {
232         if (char(*cuid_it) > 96 && char(*cuid_it) < 123) {
233             std::string EUID_element(1, char(*cuid_it));
234             UEID.append(EUID_element);
235         }
236         else { return UEID; }
237     }
238 
239     return UEID;
240 }
241 
242 /*************************************************************************
243 *                             Base Class Overrides
244 *************************************************************************/
245 void GameplayState::setStateData(GameData* game_data,
      std::vector<std::string> args) {
```

```
246     if (game_data != nullptr) { _game_data = game_data; }
247     if (_game_data != nullptr) {
248         _event_dispatcher->setGameData(_game_data);
249         _event_dispatcher->filterLocalComponents();
250     }
251 }
252
253 STATES GameplayState::update() {
254     if (!_game_data->is_running) { return STATES::S_QUIT; }
255
256     std::string frame_start_location = _game_data->current_location;
257     std::string input;
258
259     getline(std::cin, input);
260     std::stringstream input_stream(input);
261
262     // The post office
263     std::queue<Command*> events = _event_dispatcher->processEvents
264         (_input_handler->handleInput(input_stream, _game_data));
265
266     // Consider this the mailing system
267     while (!events.empty()) {
268         events.front()->triggerEvent();
269         events.pop(); }
270
271     if (frame_start_location.front() != _game_data->current_location.front
      ()) {
272         _event_dispatcher->filterLocalComponents(); }
273
274     return STATES::S_GAMEPLAY;
275 }
276
277 void GameplayState::render() {
278     std::vector<C_Render*>::iterator traversal_it;
279
280     if (first_pass) {
281         for (C_Render* renderer : _game_data->_local_renderers) {
282             renderer->flagForRender(false); }
283
284         first_pass = !first_pass;
285     }
286
287     for (traversal_it = _game_data->_local_renderers.begin();
288         traversal_it != _game_data->_local_renderers.end(); +
          +traversal_it) {
289         if ((*traversal_it)->renderThis()) {
290             (*traversal_it)->onEvent(); }
291     }
292
```

```
293     resetRender();
294 }
295
296
297
298 /*************************************************************************
    ******
299 *                              NEW HIGHSCORE MENU
300 *                                  DEFINITIONS
301 *
302 *************************************************************************
    ******
303 *                              base class overrides
304 *************************************************************************
    ****/
305 void NewHighScoreMenu::setStateData(GameData* game_data,
306     std::vector<std::string> args) { }
307
308 STATES NewHighScoreMenu::update() {
309     std::string name;
310     std::cin >> name;
311
312     return STATES::S_MAIN_MENU;
313 }
314 void NewHighScoreMenu::render() {
315     std::cout << std::endl << "Zork(ish) :: Select Adventure " <<
        std::endl;
316     std::cout <<
        "-------------------------------------------------------------" <<
         std::endl << std::endl;
317
318     std::cout << "Holy... someone's jacked and cracked at zorkish, NEW
        HIGHSCORE!" << std::endl << std::endl;
319
320     std::cout << "World: [World here]" << std::endl;
321     std::cout << "Score: [Score here]" << std::endl;
322     std::cout << "Moves: [Move count here]" << std::endl << std::endl;
323
324     std::cout << "Please type your name and press 'Enter': " << std::endl;
325     std::cout << ">> ";
326 }
327
328
329
330 /*************************************************************************
    ******
```

```
331 *                                  HALL OF FAME
332 *                                   DEFINITIONS
333 *
334 /                                                                              ↵
      ****************************************************************************** ↵
      ******
335 *                                base class overrides
336 ****************************************************************************** ↵
      ****/
337 void HallOfFameMenu::setStateData(GameData* game_data,
338     std::vector<std::string> args) { }
339
340 STATES HallOfFameMenu::update() { return STATES::S_MAIN_MENU; }
341 void HallOfFameMenu::render() {
342     std::cout << std::endl << "Zork(ish) :: Select Adventure " <<          ↵
        std::endl;
343     std::cout <<                                                           ↵
        "-------------------------------------------------------------" << ↵
         std::endl << std::endl;
344
345     std::cout << "1. [Name], [World], [Score]" << std::endl;
346     std::cout << "2. [Name], [World], [Score]" << std::endl;
347     std::cout << "3. [Name], [World], [Score]" << std::endl;
348     std::cout << "4. [Name], [World], [Score]" << std::endl;
349     std::cout << "5. [Name], [World], [Score]" << std::endl;
350     std::cout << "6. [Name], [World], [Score]" << std::endl;
351     std::cout << "7. [Name], [World], [Score]" << std::endl;
352     std::cout << "8. [Name], [World], [Score]" << std::endl;
353     std::cout << "9. [Name], [World], [Score]" << std::endl;
354     std::cout << "10. [Name], [World], [Score]" << std::endl << std::endl;
355
356     system("pause");
357     std::cout << std::endl;
358 }
359
360
361
362 /                                                                              ↵
      ****************************************************************************** ↵
      ******
363 *                                  QUIT STATE
364 *                                  DEFINITIONS
365 *
366 /                                                                              ↵
      ****************************************************************************** ↵
      ******
367 *                                base class overrides
368 ****************************************************************************** ↵
      ****/
```

```
369 void QuitState::setStateData(GameData* game_data,
370     std::vector<std::string> args) { }
371 STATES QuitState::update() { return STATES::S_QUIT; }
372 void QuitState::render() { std::cout << "Quitting zorkish!" <<
      std::endl; }
```

```
#include <sstream>
#include "InputData.h"

#pragma once
class InputHandler {
private:
    InputData* _input_data = nullptr;

public:
    InputHandler(InputData* input_data = nullptr);
    ~InputHandler();

    InputData* handleInput(std::stringstream& raw_input, GameData*
      game_data);

private:
    void resetInputData();
    CommandType validateCommandType(std::string raw_c_type);

    std::vector<std::string> formatArgsForType(std::string raw_c_type,
        std::vector<std::string> args, GameData* game_data);

    std::string getItemName(std::vector<std::string> args, std::string
      safety_word = "");
    bool takeOrDrop(std::string take_modifier);
    std::string formatTakeOrDrop(std::string raw_c_arg);
    std::string extractUEID(std::string ucid);
};


```

```cpp
#include <algorithm>
#include "../hdr/InputHandler.h"

/**********************************************************************
******
*                              De/Constructors
**********************************************************************
****/
InputHandler::InputHandler(InputData* input_data) {
    if (input_data != nullptr) { _input_data = input_data; }
    else { _input_data = new InputData; }
}

InputHandler::~InputHandler() {
    if (_input_data != nullptr) {
        delete _input_data;
        _input_data = nullptr; }
}



InputData* InputHandler::handleInput(std::stringstream& entered_data,
  GameData* game_data) {
    std::vector<std::string> raw_args;
    resetInputData();

    while (entered_data.good()) {
        std::string command_arg;
        entered_data >> command_arg;
        raw_args.emplace_back(command_arg);
    }

    for (std::string& arg : raw_args) {
        std::transform(arg.begin(), arg.end(), arg.begin(),
          std::tolower); }

    std::string raw_c_type = raw_args[0];
    _input_data->c_type = validateCommandType(raw_args[0]);
    raw_args.erase(raw_args.begin());
    raw_args.shrink_to_fit();

    _input_data->args = formatArgsForType(raw_c_type, raw_args,
      game_data);
    return _input_data;
}

void InputHandler::resetInputData() {
    _input_data->c_type = CommandType::INVALID;
```

```
44      _input_data->args.clear();
45      _input_data->args.shrink_to_fit();
46  }
47
48  CommandType InputHandler::validateCommandType(std::string raw_c_type) {
49      if (!raw_c_type.empty()) {
50          std::transform(raw_c_type.begin(), raw_c_type.end(),
              raw_c_type.begin(), std::tolower);
51      }
52      //  Instantiating aliases is just refactoring this piece of code here
          to read from a map
53      //  containing a valid command and a string of aliases which can be
          added to, just got to
54      //  check for command collisions when adding.
55      if (raw_c_type == "move" || raw_c_type == "head" || raw_c_type == "go"
          ||
56          raw_c_type == "m" || raw_c_type == "g") {
57          return CommandType::MOVE;
58      }
59      else if (raw_c_type == "take" || raw_c_type == "grab" || raw_c_type ==
          "t" ||
60          raw_c_type == "drop" || raw_c_type == "d" || raw_c_type ==
              "discard") {
61          return CommandType::TAKE; }
62      else if (raw_c_type == "look" || raw_c_type == "l") {
63          return CommandType::LOOK; }
64      else if (raw_c_type == "show" || raw_c_type == "sh" || raw_c_type ==
          "s") {
65          return CommandType::SHOW; }
66      else if (raw_c_type == "quit" || raw_c_type == "q") {
67          return CommandType::QUIT; }
68      else { return CommandType::INVALID; }
69  }
70
71  std::vector<std::string> InputHandler::formatArgsForType(std::string
      raw_c_type,
72      std::vector<std::string> args, GameData* game_data) {
73      std::vector<std::string> fmt_args;
74
75      // Spit out the formatted args for whatever command type
76      switch (_input_data->c_type) {
77      case CommandType::LOOK: {
78          if (args.size() > 1) {
79              std::string modifier = args[0];
80              std::string item_name = getItemName(args);
81
82              fmt_args.emplace_back(modifier);
83              fmt_args.emplace_back(item_name);
84          }
```

```
85          else if (args.size() == 1) {
86              std::string modifier = "around";
87              fmt_args.emplace_back(modifier);
88          } return fmt_args;
89
90          // Format item name and figure out src address and dest address
91          // For now only care about player and current location
              interactions
92      case CommandType::TAKE: {
93          raw_c_type = formatTakeOrDrop(raw_c_type);
94
95          std::string src_UCID = game_data->
96              _local_inventories[takeOrDrop(raw_c_type)]->getUCID();
97          std::string dest_UCID = game_data->
98              _local_inventories[!takeOrDrop(raw_c_type)]->getUCID();
99          std::string item_name = getItemName(args);
100
101         _input_data->src = game_data->c_inventories[src_UCID];
102         _input_data->dest = game_data->c_inventories[dest_UCID];
103
104         for (C_Render* renderer : game_data->_local_renderers) {
105             if (renderer->getName() == item_name){
106                 args.clear();
107                 args.shrink_to_fit();
108                 std::string item_id = extractUEID(renderer->getUCID());
109                 _input_data->item = game_data->entities[item_id];
110                 return args;
111             }
112         }
113
114         break;
115     }
116
117     default:
118         return args; }
119
120     }
121
122     return args;
123 }
124
125 // Expects raw message format as given by user input - c_type;
126 std::string InputHandler::getItemName(std::vector<std::string> args,
127     std::string safety_word) {
128     std::string item_name;
129     std::vector<std::string> c_args = args;
130     std::vector<std::string>::iterator args_it;
131
132     if (c_args.size() > 0) {
```

```
133         if (c_args[0] == "at") {
134             //  Start at element 2 as first arg should be "at"
135             for (args_it = c_args.begin() + 1; args_it != c_args.end(); +
                  +args_it) {
136                 if (*args_it == safety_word) {
137                     item_name.pop_back();
138                     return item_name; }
139 
140                 std::string arg = *args_it;
141                 arg.append(" ");
142                 item_name.append(arg);
143             }
144 
145             if (!item_name.empty()) { item_name.pop_back(); }
146         } else {    //  Assume item name is the args passed
147             for (args_it = c_args.begin(); args_it != c_args.end(); +
                  +args_it) {
148                 if (*args_it == safety_word) {
149                     item_name.pop_back();
150                     return item_name;
151                 }
152 
153                 std::string name_part = *args_it;
154                 name_part.append(" ");
155                 item_name.append(name_part);
156             }
157 
158         }
159     }
160 
161     if (item_name.back() == ' ') { item_name.pop_back(); }
162     return item_name;
163 }
164 
165 bool InputHandler::takeOrDrop(std::string take_modifier) {
166     if (take_modifier == "take") { return true; }
167     return false; }
168 
169 std::string InputHandler::formatTakeOrDrop(std::string raw_c_arg) {
170     std::string raw_arg;
171 
172     if (raw_c_arg == "take" || raw_c_arg == "grab" || raw_c_arg == "t") {
173         raw_arg = "take";}
174     else if (raw_c_arg == "drop" || raw_c_arg == "d" || raw_c_arg ==
          "discard") {
175         raw_arg = "drop"; }
176 
177     return raw_arg;
178 }
```

```
179
180 std::string InputHandler::extractUEID(std::string ucid) {
181     // Check each character of the cuid until we hit an upper case letter
182     // take all the characters from before the upper case and that's our
      EUID
183     // to return.
184     std::string::iterator ucid_it;
185     std::string UEID;
186
187     for (ucid_it = ucid.begin(); ucid_it != ucid.end(); ucid_it++) {
188         if (char(*ucid_it) > 96 && char(*ucid_it) < 123) {
189             std::string EUID_element(1, char(*ucid_it));
190             UEID.append(EUID_element);
191         }
192         else { return UEID; }
193     }
194
195     return UEID;
196 }
197
```

```
#include <vector>
#include <queue>

#include "InputData.h"
#include "../../commands/hdr/LookCommand.h"
#include "../../commands/hdr/MoveCommand.h"
#include "../../commands/hdr/QuitCommand.h"
#include "../../commands/hdr/ShowCommand.h"
#include "../../commands/hdr/TakeCommand.h"

#pragma once

/*  This is the thing responsible for dispatching trigger messages to the
*   relevant entities and components.
*
*   This needs:
*       --> Ability to add entities and components to an address list
*               --> This system will be built such that only relevant Entities
*                   and components are registered (such as the location and it's
*                   contents).
*       --> Ability to unsubscribe entities for on event changes
*       --> Ability to take a message from an InputHandler and trigger the relevant
*           Command with the relevant arguments.
*               --> Needs to filter to the correct Entities / Components
*               --> Needs to be able to pass these Entities and Components to the
*                   relevant Commands onEvent(args)
*               --> notify
*
*   InputHandler needs:
*       --> Takes a stringstream input from the user and
*
*   GameData needs:
*       --> To hold a list of our Command* as well as our Entities and Components
*
*
*   The Entities and Components need:
*       --> Rebuild the inventory so that it works pls
*
*   The Commands need to be able to:
*       --> Take generic arguments (such as inventory1 and inventory2 for take/drop)
*               --> This will allow them to be notified by the eventDispatcher to do
*                   a thing given generic args.
```

```
43  *
44  *   ---- Commonalities between modifications ----
45  *   1.
46  */
47
48  class EventDispatcher {
49  private:
50      GameData* _game_data = nullptr;
51
52      MoveCommand* _move_command = nullptr;
53      TakeCommand* _take_command = nullptr;
54      LookCommand* _look_command = nullptr;
55      ShowCommand* _show_command = nullptr;
56      QuitCommand* _quit_command = nullptr;
57
58  public:
59      // Create all the commands within the _commands vector
60      EventDispatcher();
61      ~EventDispatcher(); // Delete commands
62
63      void setGameData(GameData* game_data);
64      void filterLocalComponents();
65      void resetComponents();
66
67      // Call the onEvent() method for the relevant commands parsing the
          relevant args
68      std::queue<Command*> processEvents(InputData* input_data);
69
70  private:
71      void getEntityComponents(const char UEID);
72      C_Render* getRenderer(const char UEID);
73      C_Inventory* getInventory(const char UEID);
74      std::vector<C_Portal*> getPortals(const char UEID);
75
76      std::string getExitUEIDFromDir(std::string direction);
77  };
```

```cpp
#include "../hdr/EventDispatcher.h"

/*****************************************************************************
*                                De/Constructors
*****************************************************************************/
EventDispatcher::EventDispatcher() {
    _move_command = new MoveCommand();
    _take_command = new TakeCommand();
    _look_command = new LookCommand();
    _show_command = new ShowCommand();
    _quit_command = new QuitCommand();
}

EventDispatcher::~EventDispatcher() {
    delete _move_command;
    delete _take_command;
    delete _look_command;
    delete _show_command;
    delete _quit_command;

    _move_command = nullptr;
    _take_command = nullptr;
    _look_command = nullptr;
    _show_command = nullptr;
    _quit_command = nullptr;
}



/*****************************************************************************
*                                   Public
*****************************************************************************/
void EventDispatcher::setGameData(GameData* game_data) {
    _game_data = game_data; }

void EventDispatcher::filterLocalComponents() {
    resetComponents();

    const char player = _game_data->player[0];
    const char current_loc = _game_data->current_location[0];

    getEntityComponents(player);
```

```
44      getEntityComponents(current_loc);
45  }
46
47  void EventDispatcher::resetComponents() {
48      if (!_game_data->_local_renderers.empty()) {
49          _game_data->_local_renderers.clear(); }
50      if (!_game_data->_local_inventories.empty()) {
51          _game_data->_local_inventories.clear(); }
52      if (!_game_data->_local_portals.empty()) {
53          _game_data->_local_portals.clear(); }
54  }
55
56  //  Using the input data, go through the commands and queue they're
      onEvent Methods()
57  /*  Commands are in order {move, take, look, show, quit} could us a map
      but meh     */
58  std::queue<Command*> EventDispatcher::processEvents(InputData* input_data)
     {
59      std::queue<Command*> events;
60      InputData* c_input = input_data;
61      CommandType c_type = c_input->c_type;
62
63      switch (c_type) {
64      case CommandType::MOVE:{
65          std::string exit_UEID = getExitUEIDFromDir(c_input->args[0]);
66
67          if (!exit_UEID.empty()) {
68              _move_command->setData(_game_data, exit_UEID);
69              events.push(_move_command);
70          } break;
71      }
72
73      case CommandType::TAKE :
74          _take_command->setData(c_input->src, c_input->dest, c_input-
              >item);
75          events.push(_take_command);
76          break;
77
78      // Expects input data args in form {<around> || <at, "x"> || <at, "x",
         in, "y">}
79      case CommandType::LOOK:
80          _look_command->setData(_game_data, c_input);
81          events.push(_look_command);
82          break;
83
84      case CommandType::SHOW :
85          _show_command->setData(_game_data, c_input->args[0]);
86          events.push(_show_command);
87          break;
```

```
88
89      case CommandType::QUIT :
90          _quit_command->setData(_game_data);
91          events.push(_quit_command);
92          break;
93      }
94
95      return events;
96  }
97
98
99  /*************************************************************************
    ******
100 *                                  Private
101 *************************************************************************
    ****/
102 void EventDispatcher::getEntityComponents(const char UEID)
103 {   // Get all our pointers
104     C_Render* renderer = getRenderer(UEID);
105     C_Inventory* inventory = getInventory(UEID);
106     std::vector<C_Portal*> portals = getPortals(UEID);
107
108     _game_data->_local_renderers.emplace_back(getRenderer(UEID));
109     _game_data->_local_inventories.emplace_back(getInventory(UEID));
110
111     for (C_Portal* portal : portals) {
112         _game_data->_local_portals.emplace_back(portal);
113     }   portals.clear();
114
115     // Cache all of our items and their renderers
116     for (std::string Item_UEID : inventory->held_item_UEIDs) {
117         C_Render* item_renderer = getRenderer(Item_UEID[0]);
118         _game_data->_local_renderers.emplace_back(item_renderer);
119     }
120 }
121
122
123 C_Render* EventDispatcher::getRenderer(const char UEID) {
124     // Check for an associated renderer
125     for (auto it = _game_data->c_renderers.begin();
126         it != _game_data->c_renderers.end(); ++it) {
127         if (it->first.front() == UEID) {
128             return it->second; }
129     }
130
131     return nullptr;
132 }
133
```

```
134 C_Inventory* EventDispatcher::getInventory(const char UEID) {
135     // Check for an associated inventory
136     for (auto it = _game_data->c_inventories.begin();
137         it != _game_data->c_inventories.end(); ++it) {
138         if (it->first.front() == UEID) {
139             return it->second; }
140     }
141 
142     return nullptr;
143 }
144 
145 std::vector<C_Portal*> EventDispatcher::getPortals(const char UEID) {
146     std::vector<C_Portal*> portals;
147 
148     // Check for an associated inventory
149     for (auto it = _game_data->c_portals.begin();
150         it != _game_data->c_portals.end(); ++it) {
151         if (it->first.front() == UEID) {
152             portals.emplace_back(it->second);
153         }
154     }
155 
156     return portals;
157 }
158 
159 std::string EventDispatcher::getExitUEIDFromDir(std::string direction) {
160     for (C_Portal* portal : _game_data->_local_portals) {
161         //  Returns {dir, UEID, dir, UEID, ...}
162         std::vector<std::string> portal_info = portal->getInfo();
163         std::vector<std::string>::iterator traversal_it;
164 
165         for (traversal_it = portal_info.begin();
166             traversal_it != portal_info.end(); ++traversal_it){
167             if (*traversal_it == direction) {
168                 return *(traversal_it + 1); // return the exits UEID
169             }
170         }
171     }
172 
173     return "";
174 }
175 
176 
```