*Dunes of the Farlands: A text-based adventure game written in C++.*

*Group One*

Project Portfolio

*November 13th, 2023*

[Introduction 2](#_Toc125988975)

[The GROUP ONE Team 3](#_Toc125988976)

[System Requirements 4](#_Toc125988977)

[Requirements 4](#_Toc125988978)

[User Stories 4](#_Toc125988981)

[User Story #1 4](#_Toc125988982)

[User Story #2 4](#_Toc125988982)

[Project Management 5](#_Toc125988983)

[Continuity of Operations Plan (COOP) 5](#_Toc125988984)

[Project Plan 0](#_Toc125988985)

[System Architecture Design and Development 0](#_Toc125988986)

[System Implementation 0](#_Toc125988987)

[Project Postmortem <Postmortem> 0](#_Toc125988988)

[Project Wins 0](#_Toc125988989)

[Root Cause Analysis 0](#_Toc125988990)

[Lessons Learned 0](#_Toc125988991)

[System Design 1](#_Toc125988992)

[System Architecture 1](#_Toc125988993)

[Component Design 1](#_Toc125988994)

[Data Flow 1](#_Toc125988995)

[System Components 1](#_Toc125988996)

[Component [Main Loop] 1](#_Toc125988997)

[Component [User Interface] 1](#_Toc125988998)

[Component [Input Parser] 1](#_Toc125988999)

[Component [Game Objects] 1](#_Toc125988999)

[Component [Object Actions] 1](#_Toc125988999)

[Component [Player Info] 1](#_Toc125988999)

[Design Pattern 1](#_Toc125989000)

[Design Pattern 2](#_Toc125989001)

[System Implementation 3](#_Toc125989002)

# Introduction

The main goal for our project is to create a terminal-based interactive game with the C++ programming language. Our plan is to make an adventure game in the same vein as classic text-based adventure games, such as Zork I: The Great Underground Empire. This will require the program to take in user input, parse text strings, output story events, as well as save and update the current game state so that the game can respond to the player’s actions. In order to implement these systems, we will be utilizing the C++ programming language and the C++ standard template library (STD).



Core Features:

* Read & Parse User Input
* Read & Parse Story Elements
* Output Game Events
* Update Game State

Viable Features:

* Inventory System
* Chapter Select System

Stretch Features:

* Basic Item Combination System
* Combat System

# The Group One Team

Project GitHub link: <https://github.com/hkaiserteaching/csc3380-fall-2023-project-group1>

Group One is made up of six people:

* Connor Morris
  + Designer of the Input Parser & Main Game Loop
* Lane Durst
  + Designer of the User Interface & Main Game Loop
* Shawn Russell
  + Story Lead & Gameplay Designer
* Logan Remondet
  + Story Lead & Gameplay Designer
* Maureen Sanchez
  + Story Implementer & General Designer
* Yu Joo
  + Story Implementer & General Designer

The leader for all milestones is/was Connor Morris.

# System Requirements

## Requirements

Any computer running an OS which can compile and run terminal programs made with C++ Standard 20.

## User Stories

### User Story #1

*As a Gamer, I want to make decisions in the game, so I can feel a sense of agency.*

### User Story #2

*As a Developer, I want to access different story objects and events quickly, so I can easily update and make changes to the game’s story.*

# Project Management

## Continuity of Operations Plan (COOP)

Group One plans on communicating and coordinating via Discord, with meet-ups in person if required. If someone suddenly becomes unavailable for a period of time, the workload they were assigned will be evenly divided among the rest of the group members. Said person will also be kept informed about the project’s status while they are unavailable so that they can rejoin the group quickly once they’re available again. If someone becomes permanently unavailable, via illness or dropping the class, the same steps of spreading their workload out among the rest of the team will be taken.

## Project Plan

### System Architecture Design and Development

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Activity** | **Pre #** | **Estimated**  **Effort** | **Actual**  **Effort** | **Estimated**  **Start Date** | **Estimated**  **Finish Date** | **Actual**  **Start Date** | **Actual**  **Finish Date** |
| 1 | Working File System |  | Medium | Medium | Sep. 19th | Sep. 20th | Oct. 6th | Oct. 8th |
| 2 | Visual Output |  | Low | Low | Sep. 22nd | Sep. 24th | Oct. 5th | Oct. 6th |
| 3 | User Input | 2 | High | Medium | Sep. 24th | Sep. 30th | Oct. 5th | Oct. 6th |
| 3.1 | Take and Store User Input |  |  | Low |  |  | Oct. 5th | Oct. 6th |
| 3.2 | Parse User Input | 3.1 |  | Low |  |  | Oct. 7th | Oct. 9th |
| 3.3 | Update Game State | 3.2 |  | High |  |  | Oct. 6th | Oct. 9th |

### System Implementation

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Activity** | **Pre #** | **Estimated**  **Effort** | **Actual**  **Effort** | **Estimated**  **Start Date** | **Estimated**  **Finish Date** | **Actual**  **Start Date** | **Actual**  **Finish Date** |
| 1 | Clean up & beautify the look of the program |  |  | High |  |  | Oct. 26th | Nov. 10th |
| 1.1 | Output game to a separate program window |  | Medium | N/A\* | Oct. 13th | Oct. 16th | Oct. 26th | N/A\* |
| 1.2 | Support for multiple separate game menus | 1.1 | Low | N/A\* | Oct. 16th | Oct. 20th | Oct. 26th | N/A\* |
| 2 | Complete implementation of dedicated classes |  | Low | Medium | Oct. 14th | Oct. 21st | Oct. 13th | Nov. 11th |
| 3 | Add more thorough text parsing |  |  |  |  |  |  |  |
| 3.1 | Implement using objects on other objects via text |  | Medium | N/A\*\* | Oct. 21st | Oct. 28th | Nov. 7th | N/A\*\* |
| 3.2 | Recognize multiple predicates and synonyms | 3.1 | Medium | High | Oct. 28th | Nov. 5th | Nov. 7th | Nov. 11th |
| 4 | Finish writing story |  | Medium | Low | Oct. 11th | Oct. 25th | Oct. 13th | Nov. 1st |
| 5 | Complete implementation of all story objects and events | 4 | High | High | Oct. 14th | Oct. 12th | Oct. 11th | Nov. 12th |

\*We went through multiple GUI and TUIs before deciding to keep it simple and stay terminal-based, so separate windows and ‘menus’ weren’t implemented as such. We did, however, cleanup the current UI and added more features + nicer formatting & the like.

\*\*Ended up splitting the implementation of predicates & synonyms due to the effort required being higher than expected – as a result, we didn’t get to implement recognizing multiple game objects in a sentence.

## Project Postmortem <Postmortem>

### Project Wins

[Provide a bulleted list of at least 3 positive aspects of the project.]

### Root Cause Analysis

[Provide a bulleted list of at least 3 negative aspects of the project. For each negative, provide the answer to the three successive “Why” questions. ]

### Lessons Learned

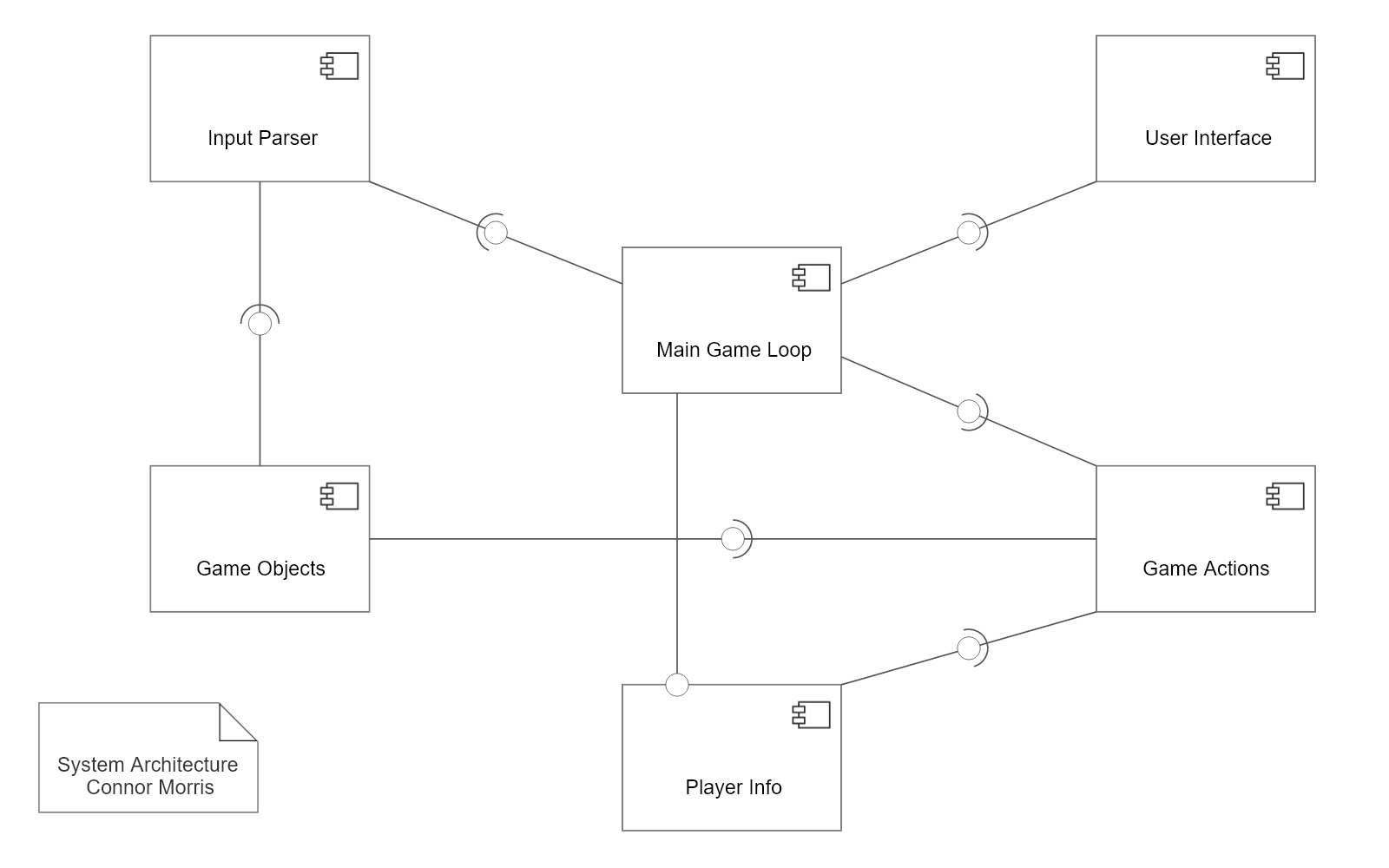
[For each negative aspect identified in the Root Cause Analysis, provide a mitigation strategy (i.e., what process should be introduced) to ensure that the problem is not repeated in subsequent projects.]

# System Design

## System Architecture

### The System Architecture consists of the Main Game Loop, which on each iteration calls the User Interface to prompt the user for text. This text is then fed back through the Main Game Loop into the Input Parser, which either rejects invalid input or attempts to assign (based on the input) an action and game object from Game Objects. The pair of action and game object is then sent back to the Main Game Loop and then into Game Actions, which determines the results of the player’s action based on the Game Objects and Player Info. Once this has finished, the result is returned back to the Main Game Loop, and is then sent to the terminal via the User Interface. The Main Game Loop continues until either the player dies and inputs that they do not want to restart, inputs a specific exit command, or completes the game.

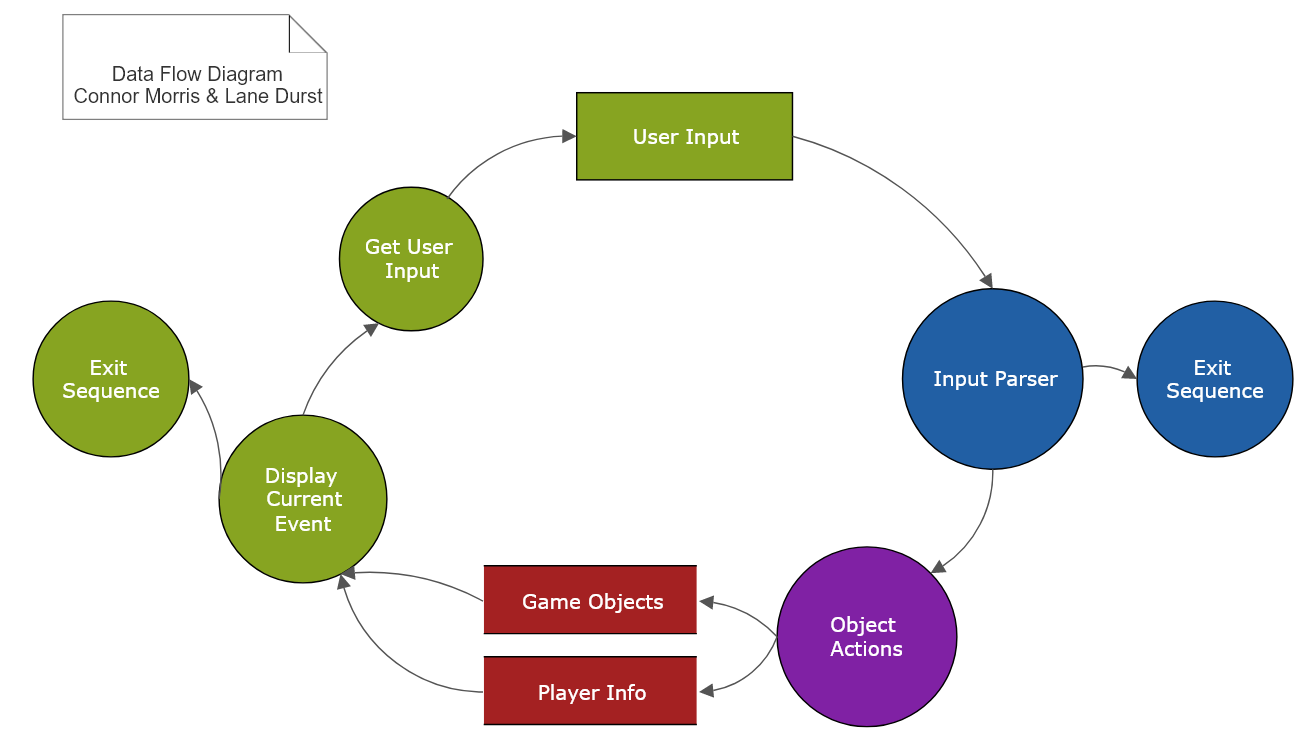
### Component Design



The major system components are:

1. The Main Game Loop which, as its name implies, handles the main loop of the game for the program.
2. The User Interface, which uses standard I/O to send and receive text from the terminal.
3. The Input Parser, which interprets the meaning of the user input and either returns a valid combination of action and game object to the main loop or tells the main loop that the input is invalid.
4. Game Objects, which contains information about the ‘game\_object’ class, the methods to access and modify said game objects, and all of the stored game objects that the program will use for the story.
5. Game Actions, which determines which actions can be performed on which items as well as what those actions entail.
6. Player Info, which contains information about the ‘player\_info’ class and the methods to access/modify an instance of ‘player\_info’.

### Data Flow

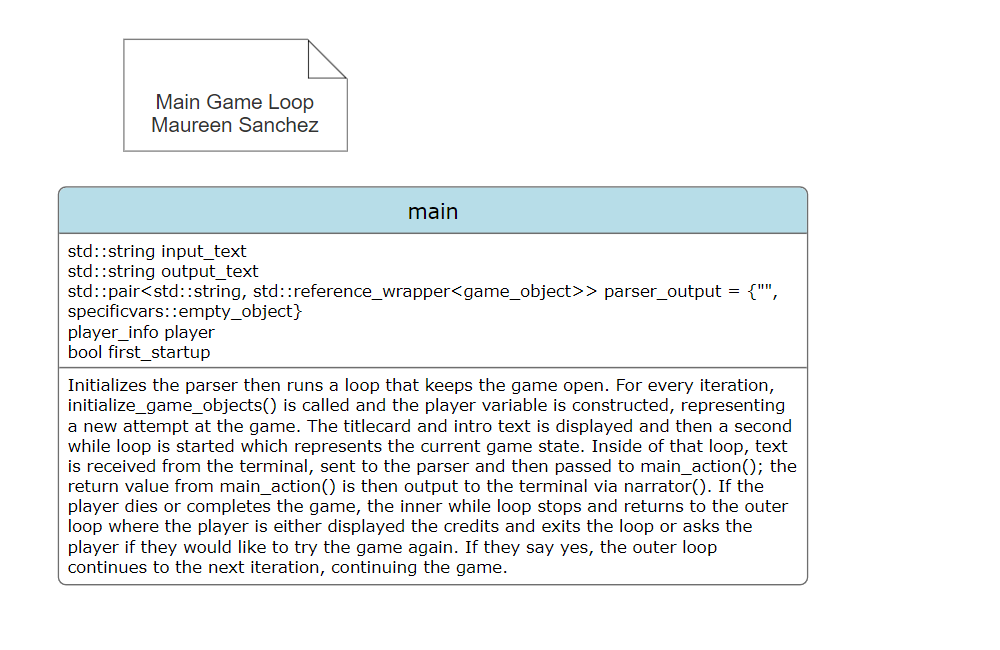


The data flow of the program consists mainly of the user input gathered by the User Interface, which is then sent to the input parser to determine what should be sent to object actions (if anything) or if the program should stop. Object actions then calls upon game objects and player info to determine what the current story event is, followed by the UI displaying said event (or exiting if the game is finished) and then cyclically prompting the user for input again.

## System Components

### Component [Main Game Loop]

The Main Game Loop is the main file of the program and handles all tasks regarding how the game is ran, what is occurring at each point in the game, and what functions from other parts of the program need to be called to achieve a successful gameplay loop.



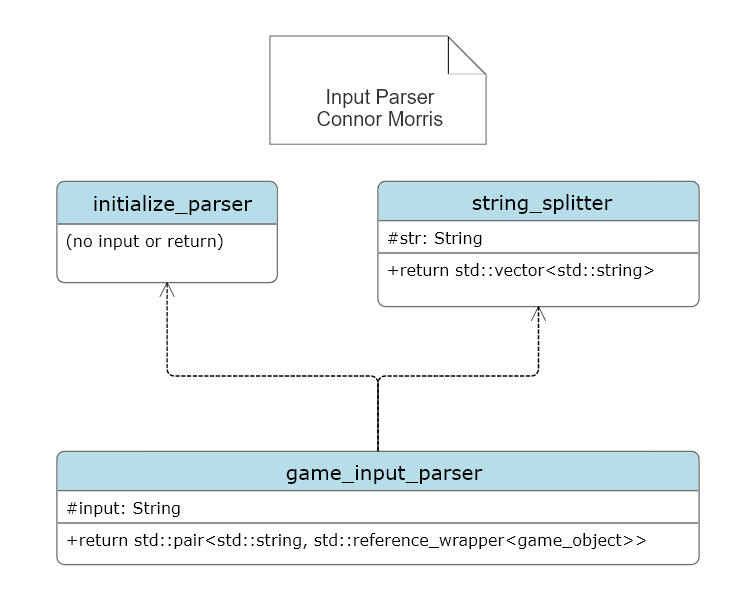
### Component [User Interface]

User Interface contains various functions related to the user interface of the game, including commands to receive input from the terminal, commands that output formatted text to the terminal, as well as other functions which handle the formatting of said text after starting with basic std::strings.

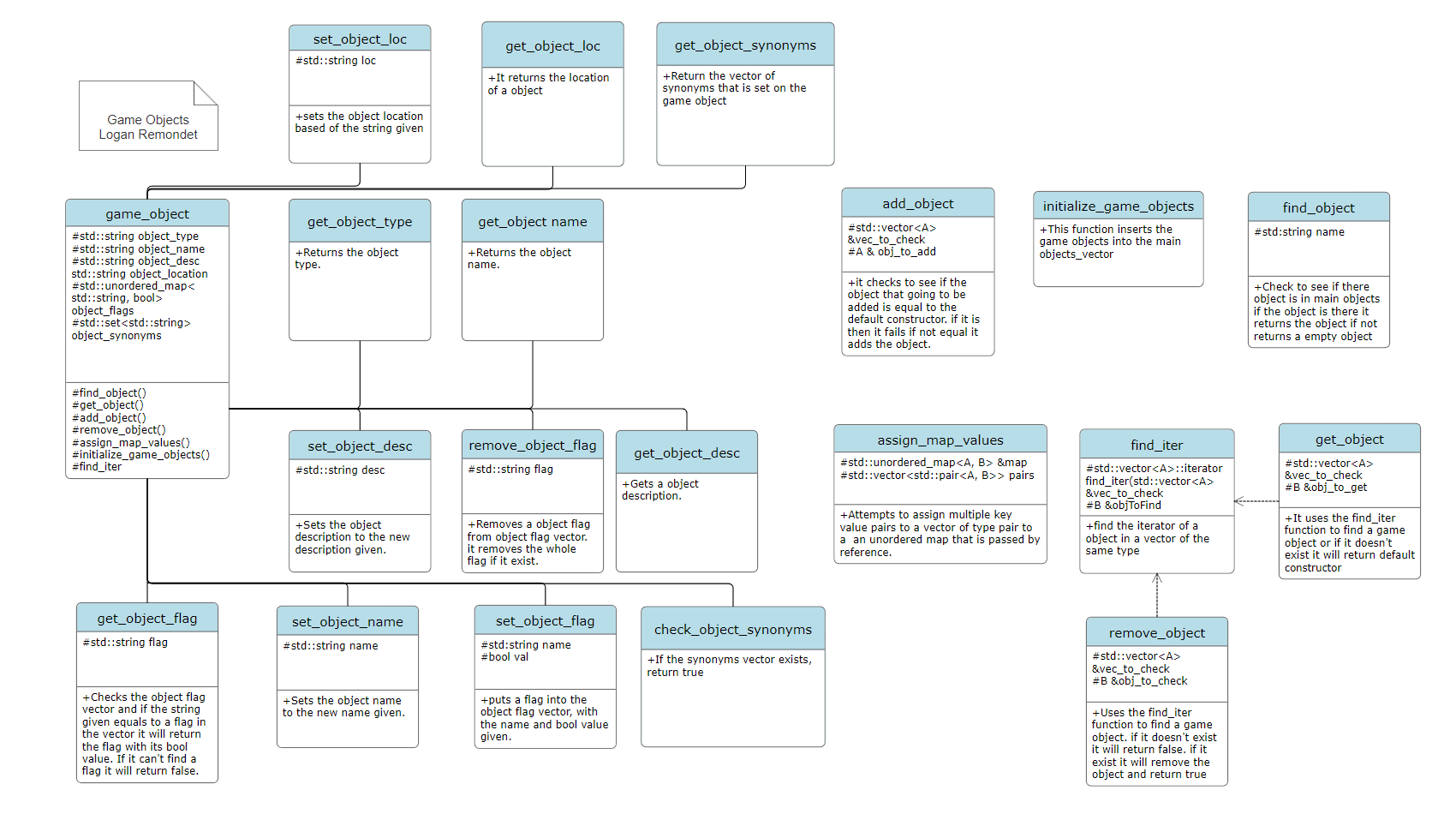
### 

### Component [Input Parser]

Input Parser contains the function ‘game\_input\_parser’ and its related functions, all of which are used to determine first if the input from the player (gathered by the User Interface) is valid, and then determine what the various parts of the input are referencing (a.k.a., which action and game\_object to send back to the main game loop).

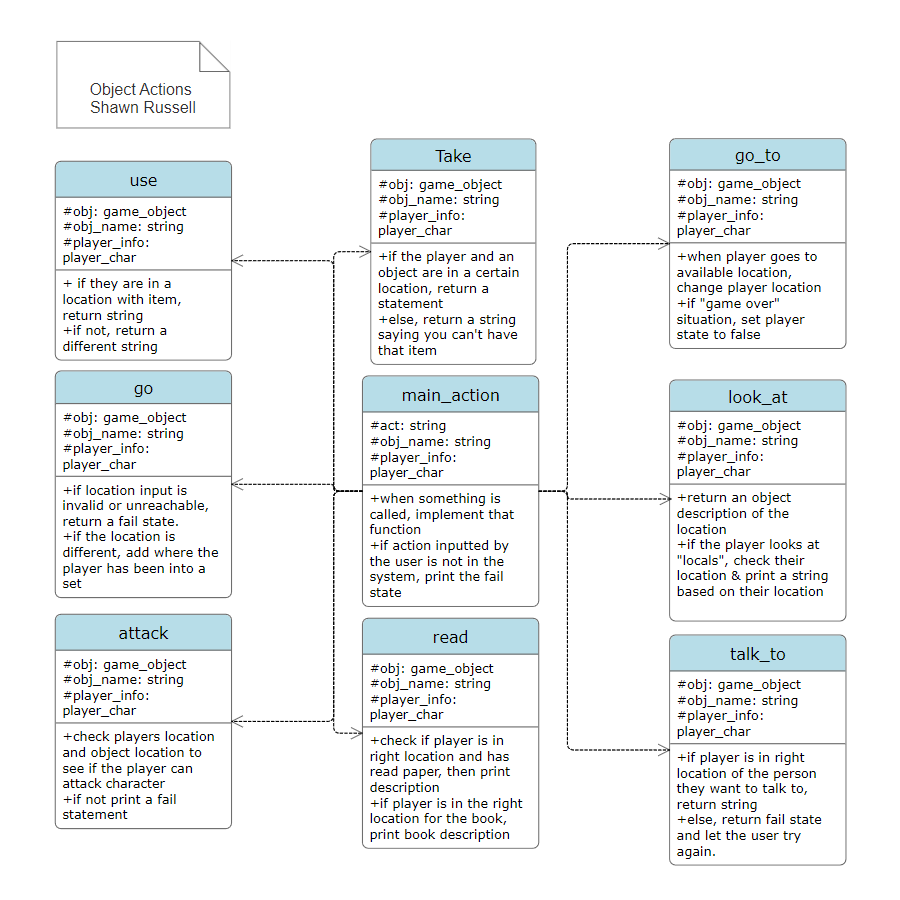


### Component [Game Objects]

Game Objects contains the class ‘game\_object’, its methods, related functions, and a store of all game objects used by the game for story events. The class ‘game\_object’ itself is used to store information about a location, item, or character within the game world.

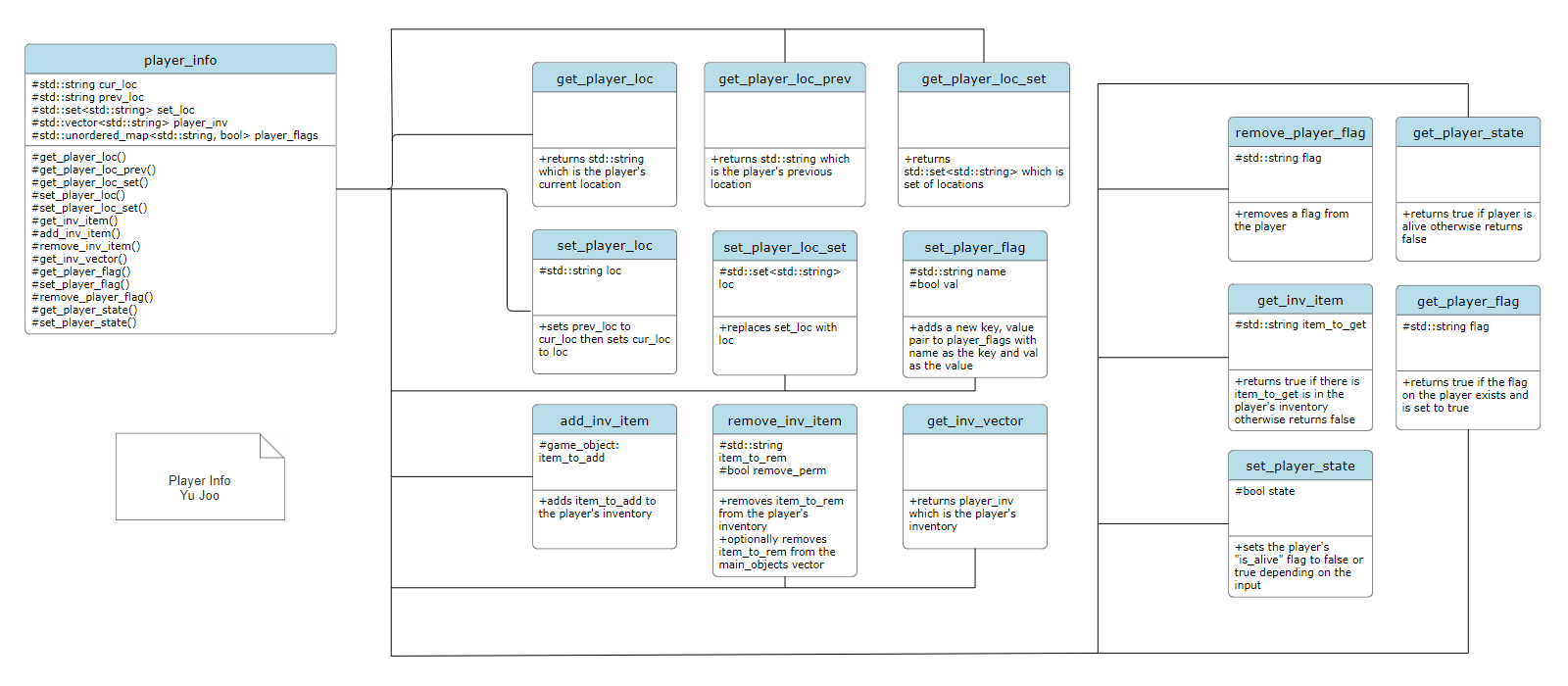
### Component [Object Actions]

Object Actions contains the function ‘main\_action’ and its related functions, all of which determine the outcome of the player after they input a combination of action & game\_object (ex: go to the town). Said outcome is then relayed back to the main game loop so it can be outputted to the terminal.

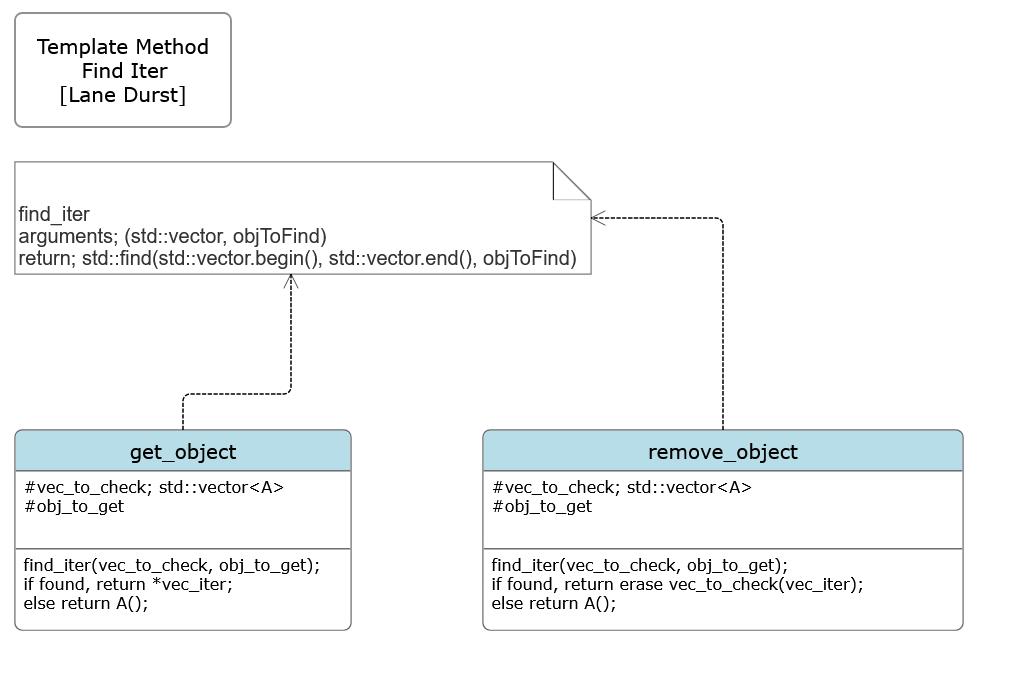


### Component [Player Info]

Player Info contains the class ‘player\_info’ and its methods – this class is used to store information about the player character during program runtime.



## Design Pattern



# System Implementation

|  |  |  |
| --- | --- | --- |
| **Architectural Component** | **Programming Language(s) %** | **Team Member(s) %** |
| Main Loop | C++ 100% | Lane Durst (50%)  Connor Morris (50%) |
| User Interface | C++ 100% | Lane Durst (100%) |
| Input Parser | C++ 100% | Connor Morris (70%)  Maureen Sanchez (15%)  Yu Joo (15%) |
| Game Objects | C++ 100% | Logan Remondet (40%)  Shawn Russell (20%)  Maureen Sanchez (15%)  Yu Joo (15%)  Lane Durst (5%)  Connor Morris (5%) |
| Object Actions | C++ 100% | Shawn Russell (65%)  Logan Remondet (30%)  Connor Morris (5%) |
| Player Info | C++ 100% | Connor Morris (90%)  Maureen Sanchez (5%)  Yu Joo (5%) |