Evey Watson

evey2.watson@live.uwe.ac.uk

Technical Design Document

troubled towers

THE LONG-AWAITED MODERN PORT OF AN ADVENTURE GAME CLASSIC

Table of Contents

[System Overview 4](#_Toc22917397)

[User Loop 4](#_Toc22917398)

[Processing user command 4](#_Toc22917399)

[Class Diagram 5](#_Toc22917400)

[Command processing – Flow Diagram 6](#_Toc22917401)

[Class Overview 6](#_Toc22917402)

[Game 7](#_Toc22917403)

[InputParser 8](#_Toc22917404)

[ScreenDisplay 8](#_Toc22917405)

[CommandManager 9](#_Toc22917406)

[Word 10](#_Toc22917407)

[Object 10](#_Toc22917408)

[Item 10](#_Toc22917409)

[Location 10](#_Toc22917410)

[Verb 11](#_Toc22917411)

[Player 11](#_Toc22917412)

[Structs 12](#_Toc22917413)

[Action 12](#_Toc22917414)

[Reaction 12](#_Toc22917415)

[RequiredItem 12](#_Toc22917416)

[Enums 12](#_Toc22917417)

[ReturnAction 12](#_Toc22917418)

[Direction 12](#_Toc22917419)

# System Overview

## User Loop

## Processing user command

Reject unrecognised

Reject inappropriate target

Cannot complete command

Return result of command

## Class Diagram

A close up of text on a white background

Description automatically generated

## Command processing – Flow Diagram

Command input

Find first whitespace

Is whitespace present?

Split command at space

Split command at space

Word1 = Verb

Word1 = verb  
Word2 = target

Valid one-word command?

Reject

Execute as basic command

Is verb recognised?

Does verb have defined response for target?

Is target object present?

Other conditions met?

Perform action prescribed by verb/target mapping

# 

# Class Overview

## Game

Members:

* Player
* Locations
* Verbs
* Items
* CommandManager
* InputParser
* ScreenDisplay

Holds universal data regarding game’s state and access to processor classes

Functions:

* Init
  + Sets up all objects and data
  + Calls ReadFromJSON
* ReadFromJSON
  + Attempts to locate JSON files for game data
  + Reads data of all locations, items, verbs into appropriate vector containers
  + Returns false if failure to locate or open files and prevents game execution
* KeyHandler
  + Identifies current key being pressed
* Update
  + Checks if key is pressed and sends it to InputParser for processing
  + Fetches InputParser full current string of player input and feeds into ScreenDisplay
  + If enter key is pressed calls InputParser for full string of player’s command then sends to CommandManager to generate response.
  + Feeds CommandManager response into ScreenDisplay
* Render
  + Renders text for all members in ScreenDisplay
    - Command response text
    - Location info text
    - Player current input
    - Any error message

## InputParser

Members:

* ValidSymbols

Verifies player input as it’s received and discards invalid symbols (numerical or non-alphanumeric characters)

Functions:

* checkValidInput
  + Uses a regex to ascertain if a new character matches an accepted symbol

## ScreenDisplay

Members:

* Screen width
* Screen height
* Command response text
* Command input text
* Location info text
* Error text
* Scanlines
* ScrollingBar

This is mostly to hold data for display, though it also holds additional objects that may be desired to render, in this case the scan lines and a scan bar to emulate a CRT monitor.

Functions:

* UpdateScrollingBar
  + Moves the scrolling bar up the screen and manages it returning to the bottom of the screen when it moves off the top

## CommandManager

Members:

* Locations
* Verbs
* Player

The bulk of processing occurs in this class; here is where the player’s command is fed into, split into its component parts & used to make decisions on how to process game data.

The Game class enters via the HandleInput function which returns a string containing the output text to be displayed to the player. All game logic is handled invisibly within the CommandManager class.

Functions:

* HandlePlayerInput
  + The ‘switchboard’ passing on the original input to the series of functions that manage processing the command and updating game logic.
* CheckValidCommand
  + Initial checks regarding whether the command is valid; whether the verb is recognised, if it can be attributed to a suitable object, if it is an appropriate single word command (e.g. “HELP”)
* ExecuteBasicCommand
  + If the command does not have a target object, such as a movement command or a single-word command such as ‘HELP’ or ‘INVENTORY’ it can be executed here.
* ExecuteItemCommand
  + This is a more complex series of logic which requires accessing the command’s Verb object, which will contain a map of suitable targets for the command against a Tuple which dictates:
    - The action to perform (pick up item, update item state, set player flag)
    - The item to perform this action upon
    - Any modifiers to the command (which state to change it to, which flag to set on the player)
  + Some verbs may be mapped to a null object, which is treated as accepting any target for the command (E.g. any object can be examined, so the ‘EXAMINE’ verb will return whatever target is fed into it)

## Word

Members:

* Word

A simple object representing a ‘Word’ input by the player; an temporary instance is created from the player’s inputs and compared to known instances stored by the game

Functions:

(Getter and Setter)

## Object

Members:

* ID
* Name
* Description

A virtual class from which Item and Location are built off; describes any object present in the world that may be targeted by a command.

Functions:

(VIRTUAL Getters and Setters)

## Item

Members:

* ID
* Name
* State
* Descriptions
* Pickupable

Represents an item present in a room/location. ‘State’ is an integer that points to an index in the descriptions array, indicating the item has changed appearance or state in the world.

Functions:

(Getters and Setters)

## Location

Members:

* ID
* Name
* Description
* Routes
* Items\_here

A location in the game. ‘Routes’ is a map of up to 6 directions – N,S,E,W,U,D

Functions:

* AddNewItem
* AddRoute
* CheckHasRoute
* CheckItemPresent
* RemoveItem

## Verb

Members:

* Name
* Valid\_Combinations

Holds a map of valid objects that may be targeted by the action, against the Action to be performed.

Functions:

* CheckValidCombination

## Player

Members:

* Current\_location
* Inventory
* Flags

The player character, tracks their current location, carried objects, and objectives achieved

Functions:

* AddItemToInventory
* CheckItemInInventory
  + Returns true if player possesses current item

# Structs

## Action

Members:

* Success\_response
* Failure\_response
* Reactions

Holds a list of reactions to execute for a valid command, and the strings to return if all requirements for them are met, or not.

## Reaction

Members:

* ReturnAction
* Target\_Object
* Required\_objects
* Parameters

Holds information on how to process game logic for a reaction

## RequiredItem

Members:

* Item\_required
* State\_required

Sets a condition under which a reaction is executed, i.e. an item that must be present and what state it should be in

# Enums

## ReturnAction

Represents actions that should be performed on game logic in a reaction:

1. GET\_ITEM\_DESCRIPTION
2. PICK\_UP\_ITEM
3. DROP\_ITEM
4. SET\_FLAG\_TRUE
5. CHANGE\_OBJECT\_STATE
6. CHANGE\_OBJECT\_NAME
7. MOVE\_ITEM\_TO\_LOCATION
8. REMOVE\_ITEM\_FROM\_LOCATION
9. ADD\_ROUTE\_TO\_LOCATION

## Direction

Represents each of the four cardinal directions & up/down

1. NORTH
2. SOUTH
3. EAST
4. WEST
5. UP
6. DOWN