Afan Rasool ; Richmond Frimpong ;Syed

Group #:

Abstract

Implementation of the game Zork in Eiffel Programming Language

eecs 3311

Project : ZORK

Contents

[BRIEF DESCRIPTION 3](#_Toc457949064)

[<How Scoring works> 3](#_Toc457949065)

[<How Health works> 3](#_Toc457949066)

[USER DOCUMENTATION 🡪 FOR PLAYERS 3](#_Toc457949067)

[CLASS DOCUMENTATION 5](#_Toc457949068)

[INVENTORY CLASS 5](#_Toc457949069)

[DESCRIPTION OF FEATURES: 5](#_Toc457949070)

[DESIGN CONTRACTS 5](#_Toc457949071)

[CONTROLLER CLASS 6](#_Toc457949072)

[DESCRIPTION OF FEATURES: 6](#_Toc457949073)

[VIEW CLASS 7](#_Toc457949074)

[MODEL CLASS 7](#_Toc457949075)

[PLAYER CLASS 7](#_Toc457949076)

[DESCRIPTION OF FEATURES: 7](#_Toc457949077)

[DESIGN CONTRACTS 8](#_Toc457949078)

[POINTS CLASS 9](#_Toc457949079)

[DESCRIPTION OF FEATURES: 9](#_Toc457949080)

[DESIGN CONTRACTS 10](#_Toc457949081)

[DRAGON CLASS 10](#_Toc457949082)

[GOLDCHEST CLASS 10](#_Toc457949083)

[Design Pattern 12](#_Toc457949084)

[SINGLETON 12](#_Toc457949085)

[MVC 12](#_Toc457949086)

# BRIEF DESCRIPTION

This is an implementation of the game Zork. This is a text game in which a player ventures through different mazes in search of wealth and treasure (anything valuable the player can find while exploring the maze). The maze is considered to be dangerous and the ultimate goal of this game is to come out of the maze alive with all the treasures needed to complete the maze. Scoring depends on the number of moves you make before collecting all the treasures from the maze.

The game is played using various text commands consisting of, although not limited to, various verb and noun words such as “NORTH”, “SOUTH”, “OPEN DOOR” etc. Complete description of the commands available is explained later in this report.

## <How Scoring works>

## <How Health works>

# USER DOCUMENTATION 🡪 FOR PLAYERS

COMMANDS

Help:

Typing “Help” gives the options available to the player.

To Navigate through rooms:

NORTH: Navigates to the North

SOUTH: Navigates to the South

EAST: Navigates to the East

WEST: Navigates to the West

To Interact with Chests and Door:

Goto Gold Chest: Navigates to the Gold Chest

Goto Silver Chest: Navigates to the Silver Chest

Open Gold Chest: Opens the Gold Chest

Open Silver Chest: Opens the Silver Chest

Open Door: Opens the door

To Pickup Elements:

Take Key : Picks up the treasure

To Interact with Dragon:

'Hi' - Greetings

'Enigma' - to ask for Riddles

'Attack' - to attack the dragon

'Key' - To ask the dragon for key

# CLASS DOCUMENTATION

## INVENTORY CLASS

### DESCRIPTION OF FEATURES:

>list: ARRAYED\_LIST[ITEM[T]]

-Holds the list of treasures available

>count: INTEGER

-Number of items in the list.

>add\_inventory(item:T):BOOLEAN

- Adds an item to the list of treasures (inventory)

>get\_inventory:ARRAYED\_LIST[T]

-Returns a list of inventory items

>remove\_inventory\_item(item:T):BOOLEAN

-Removes item:T from the list of inventory

### DESIGN CONTRACTS

Class Invariant: list/=void & count >= 0

>add\_inventory(item:T):BOOLEAN

- require: item /= void

- ensure: count = old count + 1 & list.count = count

Explanation: The pre-condition we chose makes sure that item being added to the list is not void. The post-condition we chose makes sure that the list has been incremented by one making sure that a new item has been added.

>get\_inventory:ARRAYED\_LIST[T]

require: <no precondition required>

ensure: result /= void

Explanation: There is no pre condition because our design allows the players to have no inventory (an empty list). The post condition we chose makes sure that this function we have a valid result and not void.

> remove\_inventory\_item(item:T):BOOLEAN

require: item /= void

ensure: old count - 1 & list.count = count

Explanation: The pre-condition we have chosen ensures that the item we are trying to remove exits in the list and isn’t void. The post-condition we chose makes sure that after the function has been performed, our list has one lesser item.

## CONTROLLER CLASS

### DESCRIPTION OF FEATURES:

> controller (loc:STRING ; i: INTEGER)

- User Input is being handled here. This method takes input from the user and acts accordingly.

> updateView (output: STRING)

- Updates the view. This method communicates between the Controller Class and the View Class.

## VIEW CLASS

* This class outputs the result, following the user input handled by Controller Class, on to the Command line. This is basically the interface between the game and the player.

## MODEL CLASS

## PLAYER CLASS

Player Class stores the various attributes of a player.

### DESCRIPTION OF FEATURES:

Attributes:

* Name: STRING
* Score: INTEGER
* inventory: INVENTORY[STRING]
* points: POINTS
* health: HEALTH

make(n: STRING;s:INTEGER)

* This is the constructor of a player instance which takes inputs n : STRING and s:INTEGER and sets the name of the player as n and sets the score of the player as s.

get\_name: STRING

* Returns the name of the player

get\_score: INTEGER

* Returns the score of the player.

add\_inventory\_object(item:STRING)

* Adds item: STRING to the player’s inventory.

print\_inventory\_using\_iterator(iter: INV\_ITERATOR[LINKED\_LIST[STRING]])

* Prints player’s inventory.

set\_name (n: STRING)

* Changes the name of the current player to n.

set\_score(s: INTEGER)

* Changes the score of the current player to s.

get\_points:INTEGER

* Gets the points of the current player.

increase\_points

* Increases the the points of the current player by one.

decrease\_points

* Decreases the points of the current player by one.

increase\_player\_health

* Increases the health of the current player

decrease\_player\_health

* Decreases the health of the current player

### DESIGN CONTRACTS

make(n: STRING;s:INTEGER)

require: n /= void & s >= 0

ensure: name =n & score =s

- get\_name: STRING

ensure: name /= void

- get\_score :INTEGER

ensure: Result >= 0

- set\_name (n: STRING)

require : n/= void

- set\_score(s: INTEGER)

require s /= void

Explanation:

All the design contracts in this class make sure that no invalid parameter is passed on and no attribute has a value which is outside the allowed range.

## POINTS CLASS

This class holds the points of a player and can be used to manipulate the player’s points by using methods that are part of this class.

### DESCRIPTION OF FEATURES:

get\_points:INTEGER

* Returns the points of the current player.

increase\_points

* Increases the the points of the current player by one.

decrease\_points

* Decreases the points of the current player by one.

reward\_player

* Rewards the current player by 10 points.

### DESIGN CONTRACTS

decrease\_points

ensure: points >= 0

increase\_points

ensure: points <= max\_points

## DRAGON CLASS

This class contains the commands which can be given to a dragon and updates the view by first notifying the controller accordingly. The various commands available to be used on a dragon can be seen by typing in “Help” command.

## GOLDCHEST CLASS

open\_gold\_chest: STRING

* Method to open the Gold Chest once it has been found.

get\_goldchest\_instance: STRING

* Updates the view by using the controller to indicate that a Gold Chest is present on this location.

# Design Pattern

1. Singleton
2. MVC (Model, View, Controller)
3. Iterator Pattern

## SINGLETON

< description OF SINGLETON COMES HERE>

## MVC

MVC design pattern used to implement the divide and conquer approach. Different tasks being handled by each one of the Model, View and Controller simplifies communication problems between tasks.

Model:

Model does data manipulation and processing if needed. <in our case I think model isn’t being used>

Controller:

Controller is the “middleman” between model and view. View and Model cannot interact directly. View mostly interacts with the controller, reacts to the actions performed by view and instructing it to get changed accordingly. In the context of Zork , it receives user input collected from view and updates the view accordingly depending on what action is performed by the user.

View:

This is the graphical part of the program, the screen seen by the User. In our context this is the command line and game text visible to the user is handled here in View.e class of our program.

Figure

User enters “NORTH”

Updates View with what can be seen in the NORTH

Figure 1 shows an example of the communication between the controller.e and view.e classes of our Zork prototype.

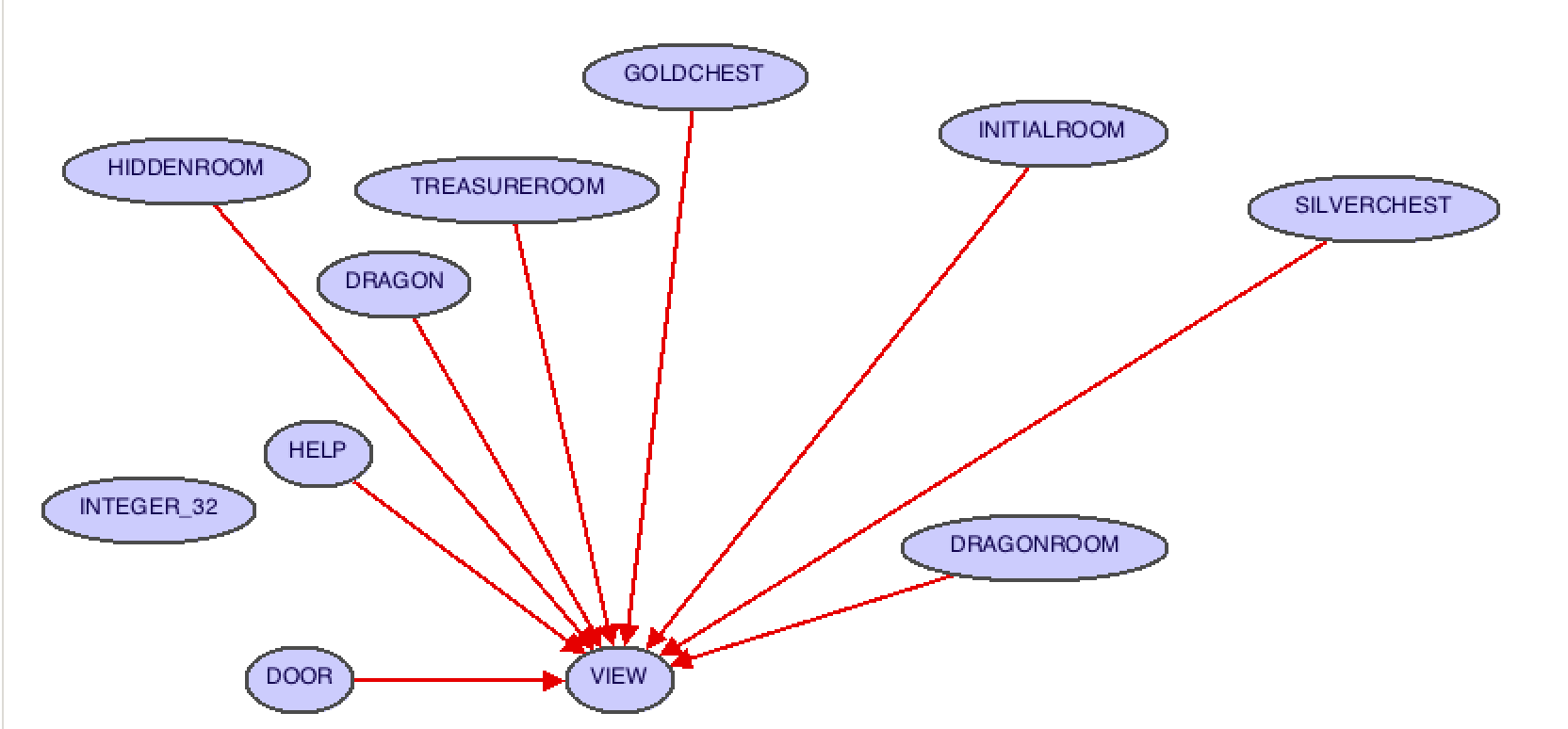


Figure shows the classes which inherit the class view and in turn have something to output on the command line once a user commands soemthing relative to these classes.