Bilkent University

Department of Computer Engineering

CS319 – Object Oriented Software Project

Project short-name: An Object Oriented Approach to Zork-Like (Text Based) Games

Analysis Report

Project Group 1

Fatih Karaoğlanoğlu
Erdoğan Ege Tokdemir
Furkan Emrehan Kılıç
İpek Lale

Analysis Report
Oct 15, 2014

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

Analysis Report

Project short-name: An Object Oriented Approach to Zork-Like (Text Based) Games

1 Introduction

Text-based games are computer games that are displayed not with computer generated images, but with plain text. The user is, thus, free to use his/her imagination alongside a storyline expressed through text. Popular in the 1970s and 1980s, text-based games have now been largely replaced by video games with graphics; one can still program a text-based game to gain familiarity with programming due to their being easy to write[1]. This project was inspired by and aims to recreate the classical text-based game "Zork" (1977) using an object-oriented approach; the aforementioned being easier to write makes it possible to emphasize the object-oriented software engineering aspect of the project by rebuilding an old game using modern tools and paradigms.

2 Current System

Zork was released in 1977 for the DEC PDP-10 mainframe computer family, making it one of the earliest text-based games. It is an interactive fiction computer game that was written using the MDL Programming Language [2]. Even though text-based games have largely fallen out of the market since the advancements in computer graphics, some are still being written for current platforms. Zork is played in a command-line interface, where the user is supposed to type in commands to move his/her character. The game begins near a house in a forest inside what the developers call the "Great Underground Empire" from which the player must return with wealth and victory [1].

3 Proposed System

The main goal is to implement a text-based game using an object-oriented programming language, unlike the original Zork, written in MDL. The proposed system will resemble Zork in terms of setting and game plot, but will differ largely in terms of software structure. Objects will be used to represent the character, the map and even the parser. Several other classes such as "User" will be implemented to further realize the game in an object-oriented fashion. The "User" class will also bring along a user management system; each user will have a username and a password, which when entered will maintain an individual account for saved game files and previous scores. This system may go as far as enabling one user to challenge other users online for a higher score. The parser will in fact have a class of its own and an instance of it will be initiated every time the player hits "new game". The entire system will reside in a class called 'GameSystem' which will manage instances of classes such as 'User', 'Parser' and 'Game'. Unlike the original Zork, this game will randomly generate the environment. Thus, the player will experience a unique gameplay each time he/she starts over.

Additionally, there will be features such as ignoring some typos in commands. For example, the original Zork parser would not recognize the command "taek", a mistyped version of the word "take". A sophisticated parser that will also recognize prepositions and conjunctions is ideal.

1.1 Overview

The system to be developed is a text-based game. As the name of the game's genre implies, the most important components of the game are user and game interactions since visual graphics do not exist. Written commands are the basic way to interact with the game. Unlike Zork, the game will provide flexibility for typos and case sensitivity.

The game plot will be based on survival. The game challenges a player to survive by displaying obstacles in his way such as hunger, heat and several hostile creatures. A player determines his/her score based on how well he/she can handle such situations. In addition, the player has a health rate which when drops to zero, ends the game. When the game ends, a high score table appears including the player's name, score and total number of moves.

Points and health will be calculated based on randomly generated values for some commands such as "eat" and "drink", thus enabling a chance factor in the game in addition to a skill factor.

1.2 Functional Requirements

- In the game there must be an instructions screen which includes the description of the game and a list of commands that the parser understands.
- The game is controlled via mouse and keyboard.
- The game must have a score screen that shows the player's score.
- The game should tolerate typos and must not be case-sensitive.
- The game should include a "help" section.
- The game should recognize the following commands: look, search, go, take, attack, kill, use, examine.
- The user should be able to login to the game.
- The user should be able to save his/her progress in the game.
- The user should be able to load a previously saved progress.
- The user should be able to logout.
- The user should be able to restart at any point in the game.
- The user should be able to save only one instance of his playing.
- The user should be able to create a new account with a username and password.
- The game should display a high score table.
- The high score table should have dates, names and scores.
- The system should show the date in this format: dd / mm / yyyy.
- Multiple players should be able to play the game, although not simultaneously.

1.3 Non-functional Requirements

- The response time for commands should be less than one second.
- The game should be able to run on Windows, Mac and Linux systems.
- The content of the game should be easy to understand.
- There should be separate classes for each construction (locations, items, characters etc.) in the game to ease testability and increase flexibility of the system (working with separate classes make it easier to add features to the system).

- The level of expertise of the user shall be basic.
- The system should store a maximum of 10MB of data.
- The user should have JRE installed in his system in order to run the program.
- The user's system should have at least 128MB of installed memory.

1.4 Pseudo Requirements

1. The project should be completed within three months.
2. Java should be used as a programming language.
3. The system must be a desktop application.
4. The system must be distributable.

1.5 System Models

1.1.1 Scenarios

Overall usage

Lucy, a student at Bilkent University, has a lot of assignments and exams. She often gets bored and feels the need to play a computer game to relax; she runs the game, creates an account and starts playing. After a while, she gets bored, saves the game and quits. Later, she comes back and runs the game again. She loads her save file, and continues to play where she had left off. Eventually, she dies in the game. Then high score table appears and her score, name and time of end game appears along with other players. She then quits the game and continues with her studies.

Controls

During the game, she uses the "help" command to learn how to play the game. Then, using the commands that appear in the "help" section she moves the character and tries to survive the obstacles.

1.1.2 Use-Case Models

This diagram (Figure 1.1) shows the Use Case Model for logging in. The login operation can either return a success or a fail. The "load game" and "new game" operations follow "login".

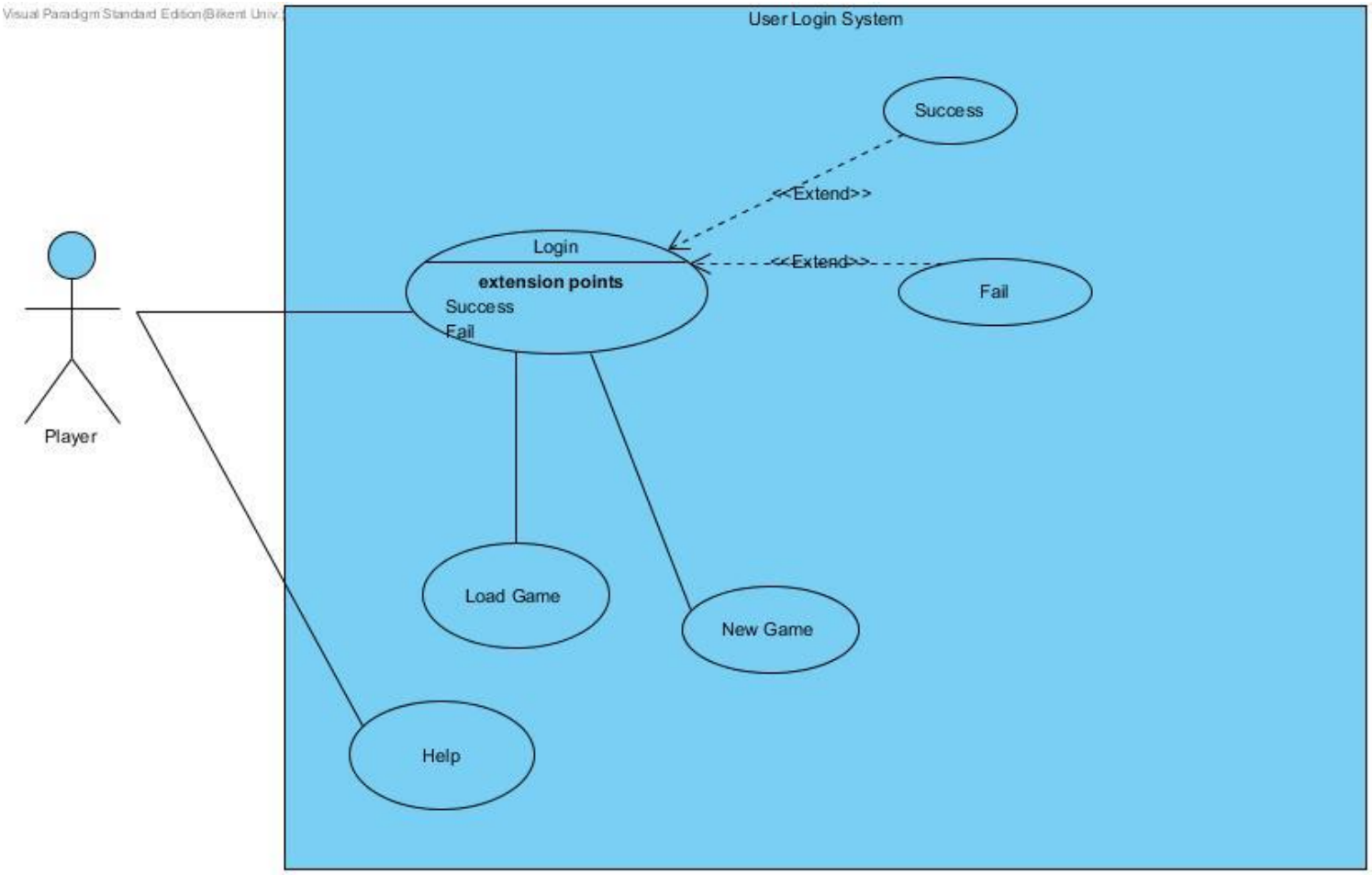

Figure 1.1

This diagram (Figure 1.2) shows the Use Case Model for a gameplay. The list of commands is as below. The entire gameplay will consist of typing commands into the command-line, some of which extend other commands.

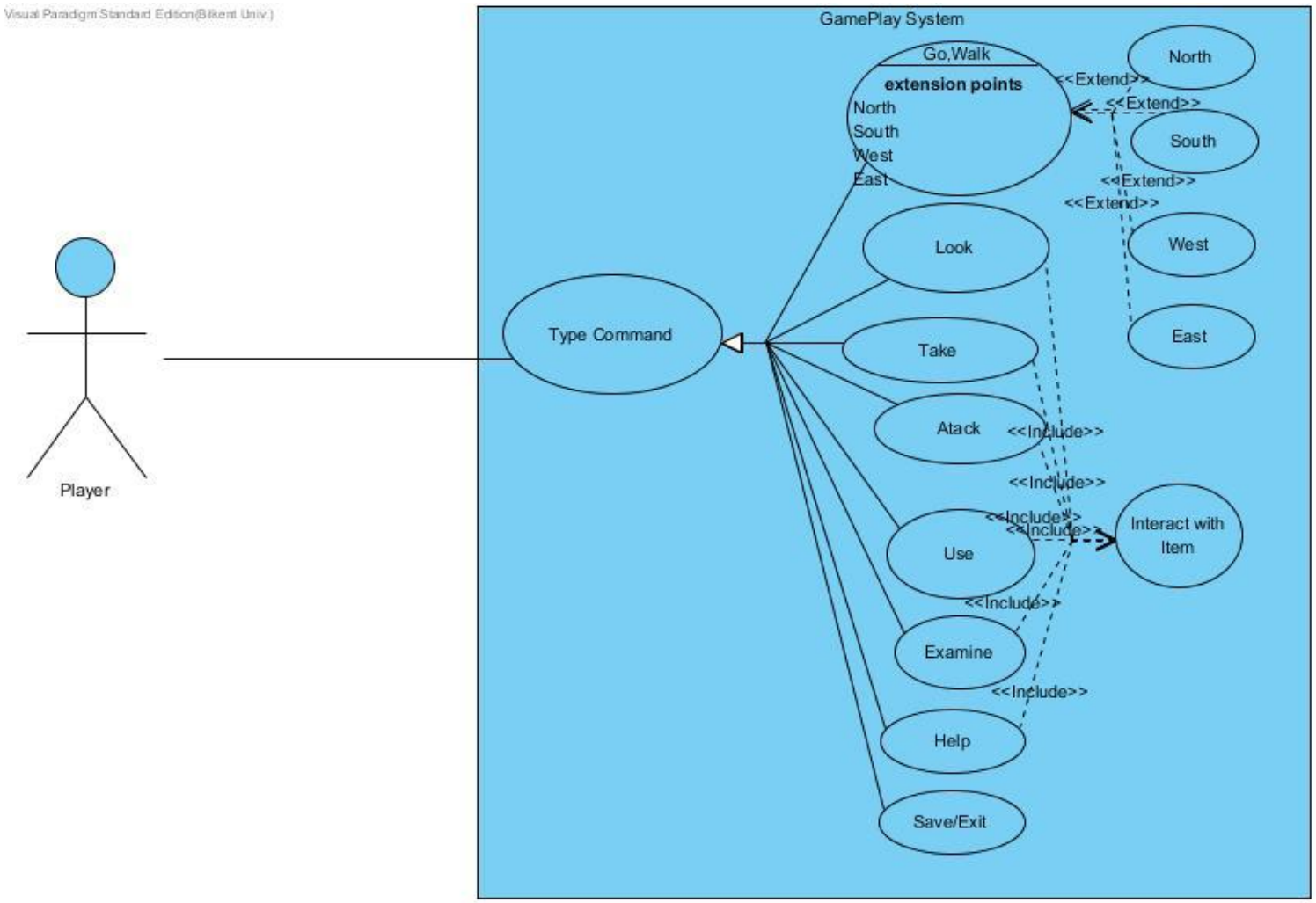

Figure 1.2

Object and Class Model

This diagram (Figure 2.1) shows the entire structure of classes and their relations. The GameSystem class, as mentioned earlier, is the class that manages all operations related to individual users. It holds an instance of "Parser" which in turn receives a Game instance as parameter. When a new game is to be started, a new instance of Game is generated inside GameSystem.

The Parser class can manage one Game object at a time. Every command entered into the terminal is delivered to the Parser object as a string, which is then interpreted in the form of a modification to the Game object. The Game object is the next class in the hierarchy and it can hold instances of several other classes such as Map and Character. Character is further divided into the Player and Non-Player Characters, which are dungeon trolls, ogres and other similar creatures the player can combat. As stated earlier, the map of the game will be randomly generated according to a seed. The map will consist of Location instances that can be thought of as coordinates linked to one another. An additional detail here is that these Location instances will represent a two dimensional plane, while some Location instances will be linked to another Location instance that resides on a completely different plane of Locations which can be accessed by "climbing up" or "climbing down" a certain structure, or Thing, inside that particular Location instance, which ultimately imitates a three dimensional environment.

Lastly, the aforementioned Thing class represents all objects available in the game, i.e items that can be acquired by the player or other stationary structures such as walls, doors, tables etc. The classes that inherit the Thing class are appropriately titled Item, which is divided further into edibles and inedibles (weapons, treasure, pamphlets etc.) via class attributes; and Furniture, both respectively modeling the previously mentioned objects in the environment.

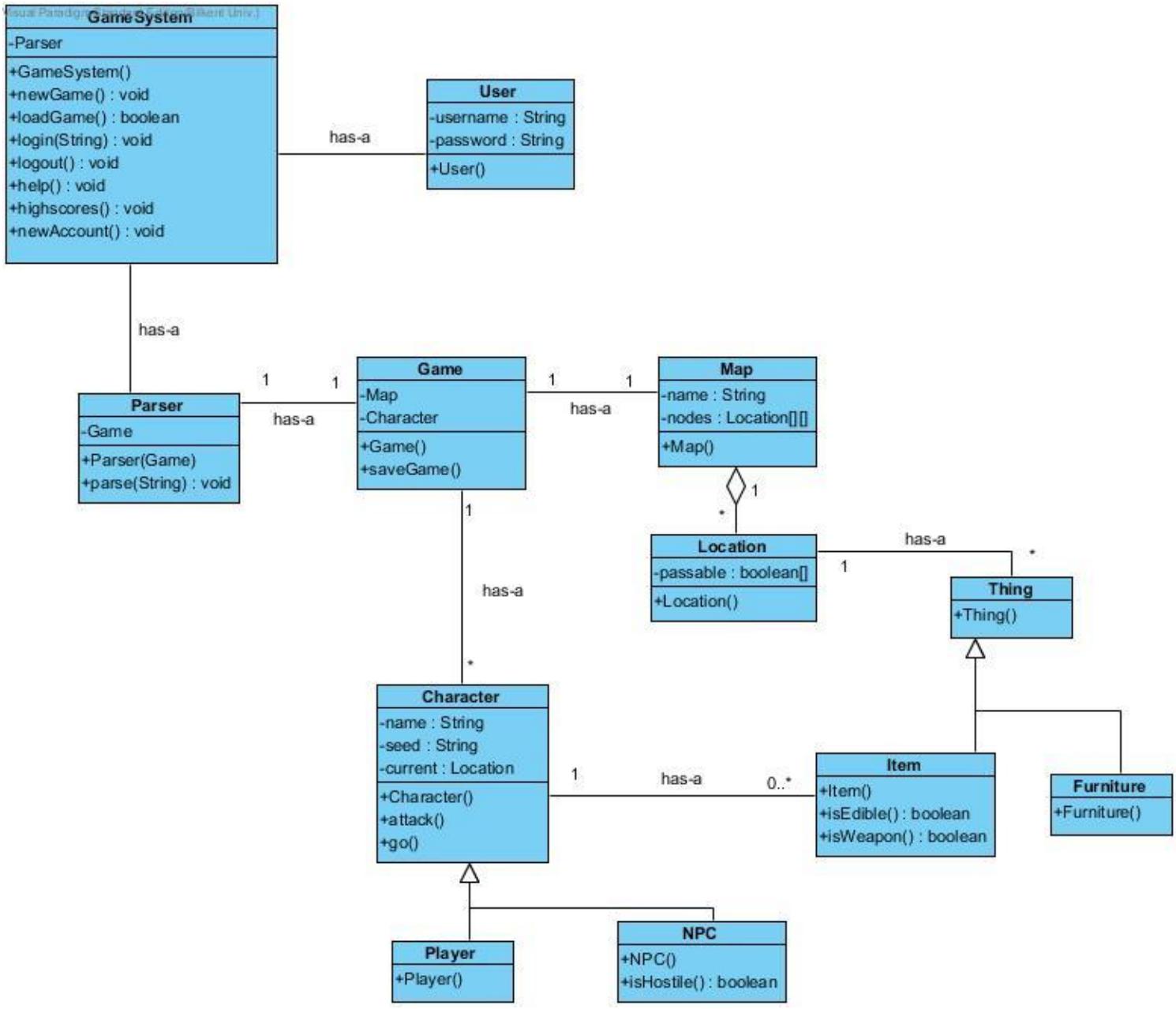

Figure 2.1

1.1.3 Dynamic Models

This diagram (Figure 3.1) shows a State Machine Diagram that realizes the entire system. The "Parse" state models actual gameplay and the rest model the non-gameplay system.

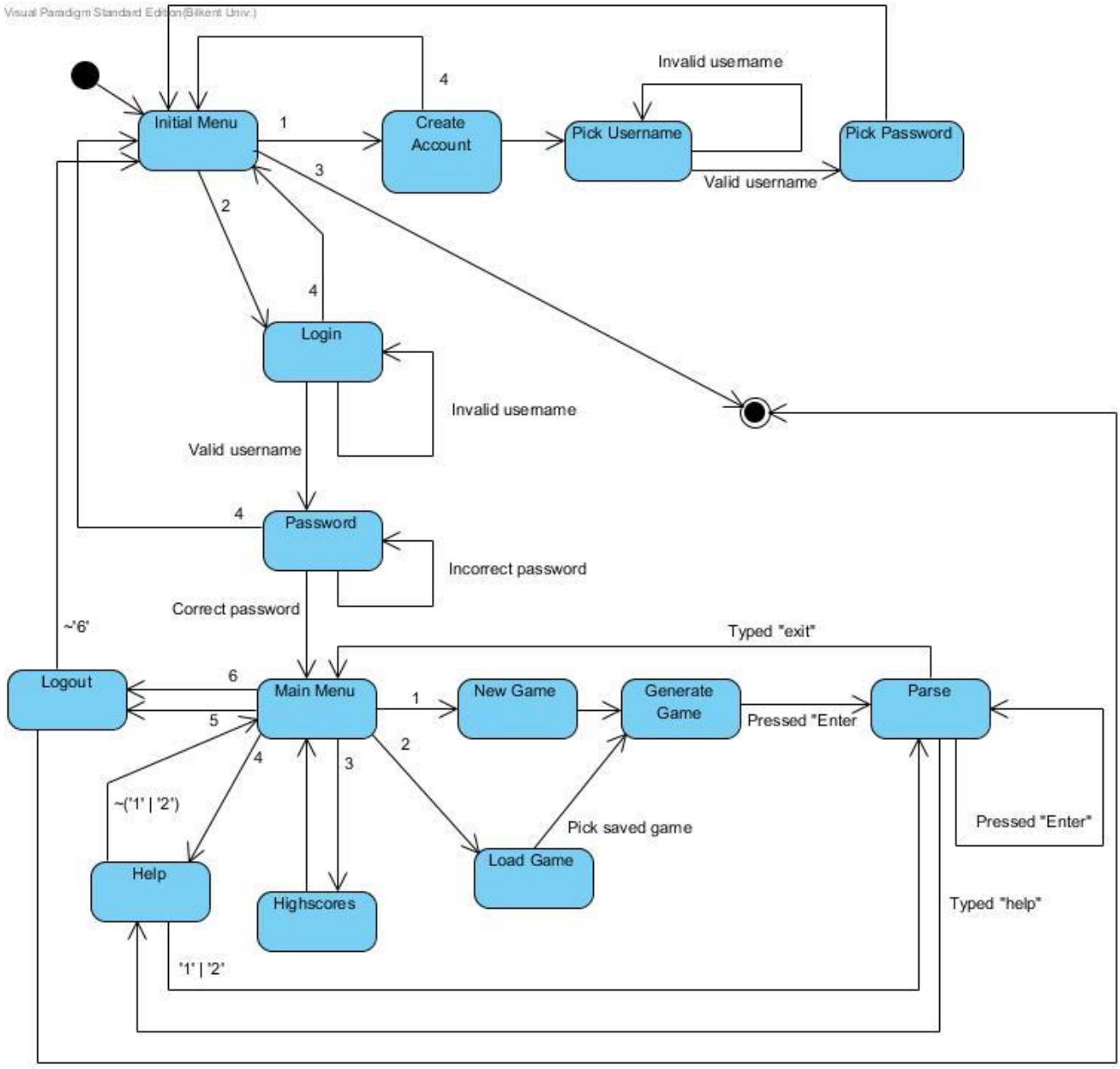

Figure 3.1

This is an additional table (Table 1) that describes the meanings of the numbers in Figure 3.1.

Table 1

Initial Menu	Main Menu
1. Create account	1. New game
2. Login	2. Load game
3. Quit	3. View high scores
4. Back (only used after 1 or 2)	4. Help
	5. Logout
	6. Quit

This diagram (Figure 3.2) shows a Sequence Diagram for the saveGame() function.

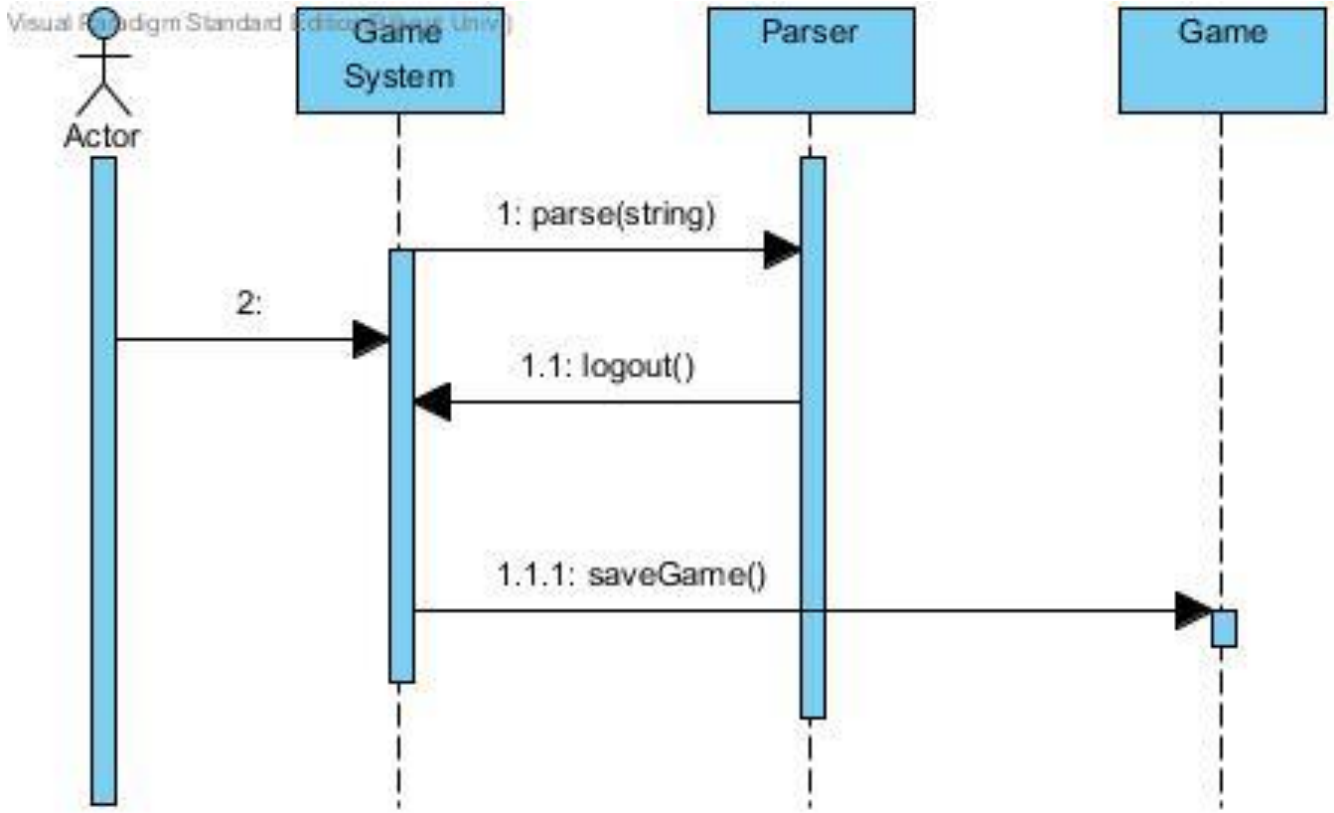

Figure 3.2

This diagram (Figure 3.3) shows a Sequence Diagram for the function startGame() which initiates a new game.

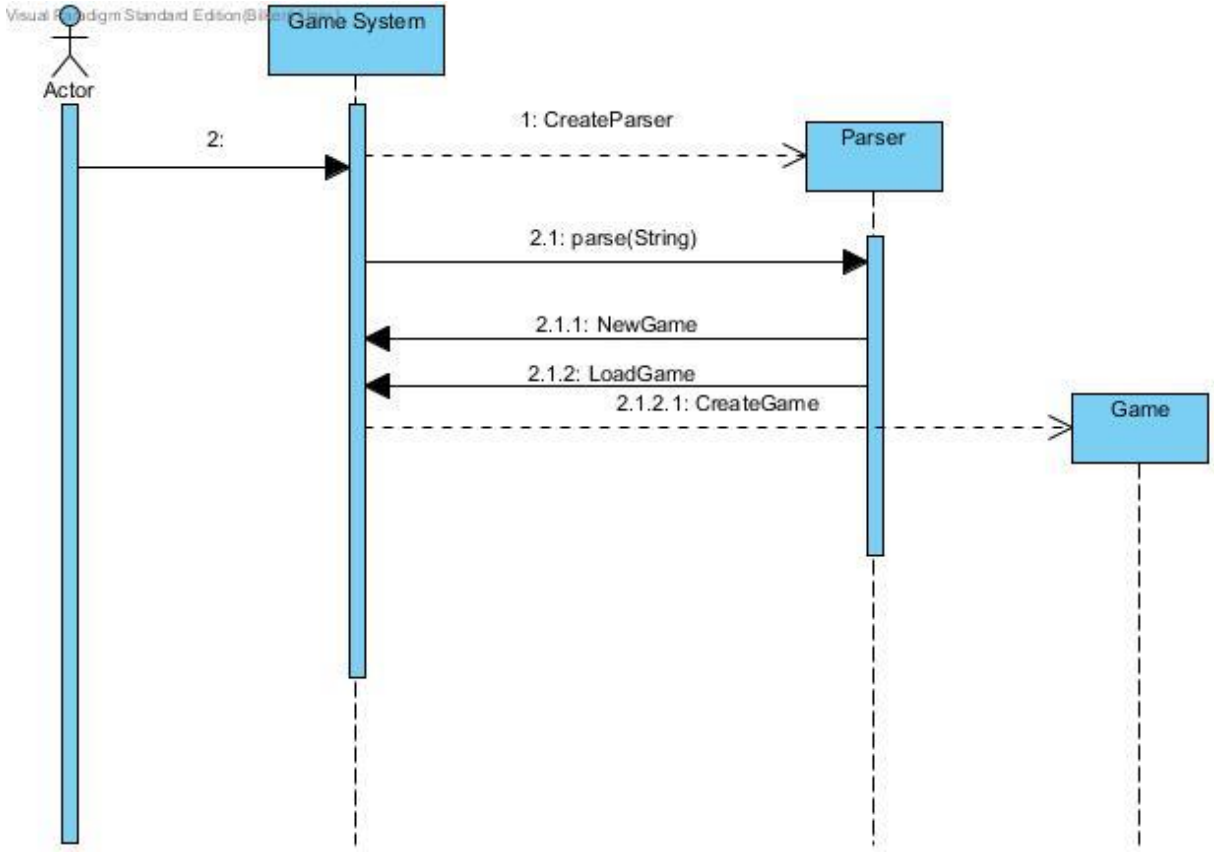

Figure 3.3

1.1.4 User Interface

The user interface is simply a command-line interface.

4 Glossary

MDL: MIT Design Language, a descendant of the Lisp programming language.

5 References

[1] http://en.wikipedia.org/wiki/Zork.

[2] http://en.wikipedia.org/wiki/Text-based_game