# Lab Assignment 3

Writing a text-based adventure

February 23, 2010

In this lab assignment, we are going to write an old-fashioned *adventure game*. Unfortunately, the first adventure games did not have fancy 3D graphics, but were totally text-based. So, all interaction between the game and player was through entering and printing text.

An example is *Zork* (see Figure 1) which was very popular in the beginning of the eighties. In this game, you are an adventurer that needs to collect several treasures, that are hidden throughout the world. All by just typing and reading text.

## The assignment

This assignment is two-fold. The first assignment is to create an *engine* for text-based adventures. Although the topic of game engines will be covered later in the course, this will not be a problem for this assignment. This engine does not need to be very complex. The second assignment is to create a text-based adventure using this engine. I will elaborate on the two assignments in the sections below.

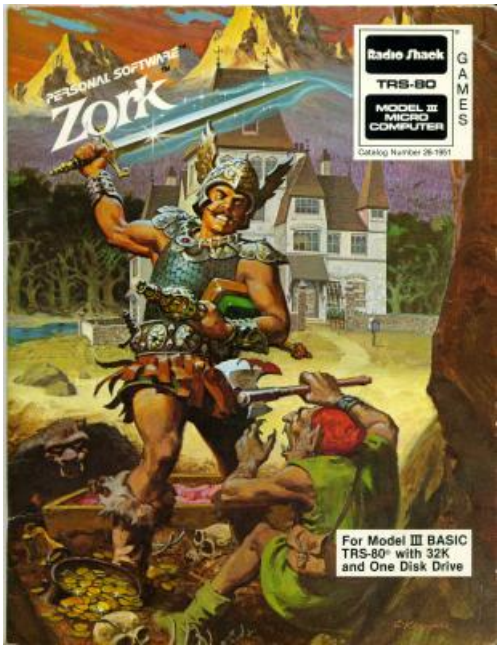
### Assignment 1: Text-based game engine

As said, the first assignment is to write a game engine for text-based adventures. In general, a game engine is a modular framework that allows for fast game development. It provides functional components for sound, 3D graphics, physics etc. Once you have developed such a game engine, in the ideal situation, you would only need to change the content of the game (such as media and scenarios) to create a new game. One would not need to write a new graphics engine or whatsoever.

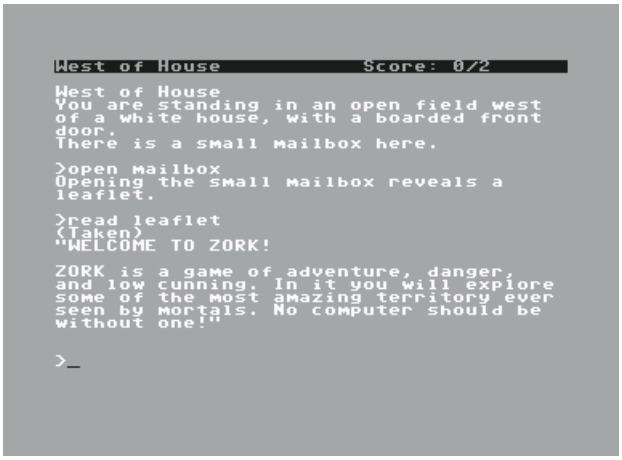
Our task is to write a game engine for text-based adventures. So, we do not need support for 3D graphics, yet we do need other components. For instance, we need a component for printing the current world state (room and object descriptions) to the screen and a component for reading and parsing input from the keyboard.

This game engine should be very modular so that specific components can be adapted without changing the entire engine. Second, the data and representation should be clearly separated. So, when necessary, we can swap the text representation for a 3D graphical representation.

The 3 main components of a text-based adventure are the *rooms*, *objects* and the *character*.




(a) The original Zork cover




(b) Zork on the Commodore 64

Figure 1: Zork was one of the first adventure games

### Rooms

The rooms (or areas) of the game should be read from a (text) file. An example:

```
Room: 1
Name: a dark cave
Description: You are standing in a dark cave. You hear water running
somewhere in the distance.
Exits: 2 3

Room: 2
Name: a waterfall
Description: You are standing in front of a beautiful waterfall. There
is an opening in the cave and you can see the moon shining brightly.
Exits: 1
Object: an old key
Object: a hammer

Room: 3
Name: a dark forest
Description: You are standing in a forest. Although the forest is dark,
you do feel at home. There is a cave behind that oak tree over there.
Exits: 1
```

When printing the text representation of a room to the screen, you should also print the exits to other rooms. Furthermore, rooms can contain *objects*. These should also be printed.

### Objects

The objects in the game can be written in code. Do keep in mind that you should implement them in a nice object-oriented fashion. For instance, you can have a *Key* object with variables such as *name* and *description*. Another type of object could be a *container* such as a garbage can. When opening or examining such an object, new objects appear while the container remains in place.

### Character

The character of the game, although not explicitly visible, has an *inventory* that contains several objects. Typing *inventory* should result in the list of the objects the character carries with him. Typing *pick up "object name"* should add objects from the room to your inventory. Typing *go to "room name"* should move the character to another room.

### Parsing

Most text-based adventures typically used a verb-noun parser to interact with the world. The objects in our engine should support at least 3 verbs, namely *examine*, *use ... with ...* and *open*. Note that applying verbs on an object can result into other new objects. So, the object class declarations could basically look like this:

```
class Object{

public:
    virtual ~Object(){}
    Object(string name, string description);

    virtual string examine() = 0;
    virtual string open() = 0;
    virtual string useWith(Object& object) = 0;

    //getters and setters for name etc.

private:
    //private members etc.
};

class Key : public Object{

public:
    Key(string name, string descriptions);
    ~Key();

    //implement interface Object
    string examine(){
```

```
        return string("This looks like it is a hundred years old");
    }
    string open(){
        return string("Nothing happens...");
    }
    string useWith(Object& object){
        if(object.getName() == "door"){
            object.setOpen(true);
        }
    }

};
```

A representation of the example room and key object could be:

```
---------------------------------------------------------------------
     a waterfall
---------------------------------------------------------------------

You are standing in front of a beautiful waterfall. There
is an opening in the cave and you can see the moon shining brightly.

You see: an old key and a hammer

From here you can go to: a dark cave

---------------------------------------------------------------------
```

The parser itself does not need to be very complex, it can be simply based on if-statements.

### Assignment 2: Text-based adventure

The second part of the assignment is to implement a game for your engine. The game does not need to be very complex or original but should just show that the engine works. It should just show that your game engine works properly. Implement at least 5 rooms, 5 objects (one object should be a *priceless Ming vase*) and some interaction between the objects. You should at least have one container and some verbs should result in new objects (so no renaming of existing objects) that are added to your inventory.

Make sure that all the verbs are implemented or return a message that nothing happens.

## Submitting your Work

The submission deadline is Monday, 8th of March, at midnight. Emailing your zipped *solution + level files* to basten@cs.uu.nl. Please include student numbers.

The assignment will be graded as 'satisfactory' when the engines provides all functionality described in this document, plus the following additional requirements:

- Separate the data from the representation (such that it is easy to swap it for a 3D engine, for example). Also write separate classes for reading room files.
- Your game should contain at least 5 rooms and 5 objects. Some verbs should result in additional new objects (so no renaming of existing objects). All verbs should be properly implemented for all objects.
- Provide a walkthrough of the game.
- Please include sufficient *comments* to the code so that at least the role of each function and class in your program is clear.
- A first requirement is that I can simply open the solution, compile and run the contained project, and check the output. If it does not compile and run in both debug and release, there will be no grading.
- In addition, please make sure to 'Clean' your solution, and delete the '.ncb'-file and '.ilk' files. Otherwise my mailbox will explode.