Programming 2 (2023) Project: "Interactive Fiction"

Originally by Christian Grévisse, modified by Aryobarzan Atashpendar
12 April 2023

Abstract

Text-based adventure games, often called *Interactive Fiction*, have been around for over 50 years. One of the popular games during the 80s was *Zork* (cf. Sheldon playing it in "The Irish Pub Formulation" (TBBT S04E06)). In this project, you will, in groups of 3, invent, design and implement such a text-based adventure game using the Swift programming language. The use of other programming languages is not permitted, unless needed to integrate a third-party API (approval required)!

Project Description

Create a Swift console application that allows a user to walk through a set of **inter-connected rooms**. Rooms do not necessarily need to be closed ones, it could also be a marketplace, a space station or an ocean. The game's setting is up to you! The player starts in a given room and makes their way through the game map using textual commands. There can be one or more final rooms, leading the player to win or lose the game. Whether there is a happy ending or not depends on you (think of a *Kobayashi Maru*-like experiment). Whether dying/losing is possible also depends on you. In any case, your choices should be well-defined and documented.

Connections between rooms are modeled as **exits** in certain directions. These connections do not need to be symmetric, so they can be one-way only. Also, exits might change over time, as they could disappear (e.g., a bridge could fall together such that the player can't go back to their previous location on that path; or the radiation after an explosion makes it unbearable to go back) or they could suddenly appear (e.g., a dark/foggy room only unveils its exits once the player has found a torch; or a door can be opened after finding its key or access code). Rooms could also be generated or removed dynamically, depending on the game flow.

The game should include at least 2 systems of **points** (health, money, magical powers, ammunition, ...) that the player can gain, spend or lose. There shall be different **player characters** from which the user can choose from at the beginning of the game. Characters might possibly be upgraded during the game (e.g., a novice wizard becomes an expert wizard). The characters define the initial values of the point systems.

Inside a room, the player might do other things than just travel to the nearest exit to another room. Rooms might propose **minigames** (you could also call them missions, quests, tasks, riddles, ...). For example, the player would need to go over a river of lava in a minesweeper-like way; or they would need to decipher an encrypted message bearing relevant information for the rest of the game. Other tasks could be based on some randomness/luck or even a timer.

The player can **collect objects** which can be found in rooms, and which may be required to fulfill a minigame in that same or another room. The player shall collect them in some kind of bag or inventory. For instance, a key could be necessary to open a door, or coordinates written on a piece of paper could be required for teleporting to a different room. Gaining or losing objects could also be the result of winning/losing a minigame.

A particular type of minigame that needs to be present is a **battle**. In a battle, the player needs to compete against an adversary (e.g., a monster, a witch, a cyborg, a pirate, ...). Entering a room containing an adversary could close some or all of its exits, requiring the player to confront them. A battle might require various actions (commands) using different weapons, items or tricks. The result of an action, possibly provoking a counter-attack, depends on the abilities of the player's character as well as the abilities of the adversary. The result of a battle can influence one or several point levels of the player, such as their health or gold. After a battle, the player may have won or lost one or several objects, or even have been placed into a different room.

A special kind of object that the player shall be able to collect in your game is a **compass** (or map, tricorder). If in possession of a compass, the player can 1) be shown the current game map and 2) find the location of as well as the path to a particular object that they want to find. For finding this path, you can rely on a breadth-first search algorithm.

Finally, an **automatic game mode** shall enable the game to be played without user interaction. Apart from character selection, this automatic mode shall discover the game world on-the-fly, without having access to the room map. The path shall by no means be hard-coded, but be generated in a trial-and-error approach. At all steps, necessary output shall be written to the console to understand what's happening. For this part, you may rely on recursion techniques (cf. Programming 1, Algorithms 1), such as *backtracking*.

Bonus Features

Any additional functionality is highly welcome and will be awarded bonus points (if functioning and well implemented, of course). Easter eggs, i.e., some hidden feature or goodies, are common elements of surprise in video games. For instance, the player might time travel back to a former state of the game, or they could be poisoned/drunk such that some commands become unavailable or non-functioning. You might require players of your game to win within a limited amount of commands. Or you could allow multiple players to play in a turn-based fashion. Finally, for the very motivated among you, you could enhance your game through ASCII art or, for macOS users, use *SpriteKit* to enable a graphical visualization (while still providing a textual command input).

Framework

Build your application on top of the framework "InteractiveFictionFramework". Your auto-generated repository will contain a starter Swift project alongside this framework which will be included in its `Sources` folder.

- **Warning**: Do not modify any of the files which are part of this framework.
- Instead, you are to create your own custom game which implements the `Game` protocol introduced by the framework.

You can also find a demo application* which uses the framework in the lab solutions repository.

Your first actions should be to:

- Read and understand the framework code and documentation
- Build and run the demo application*
- Play around with the code, make some changes and try them out

Possible Scenarios

Here are some ideas for possible scenarios your game could realize:

- Borgs aboard the USS Voyager
- Robinson Crusoe on Monkey Island
- Dragons in caverns
- Getting off a delayed flight and trying to get to the boarding gate of a connecting flight on time
- ...

Provide a fascinating storytelling, be creative!

Requirements

- Implement the features described above, in particular:
  - rooms & exits
  - points system(s)
  - player characters & skillsets
  - min. 2 battles
  - min. 2 other types of minigames
  - collectible objects
  - compass
  - automatic game mode
- Demonstrate extensive use of Swift language features (classes/structs, optionals, closures, protocols, extensions, ...)
- Demonstrate good OOP design (encapsulation, polymorphism, error handling)
- Proper indentation and comments
- Output of all relevant information and storytelling on the console

Deliverables & Deadlines

The submission is handled through a Git repository on GitHub. After indicating your groups, your repository will be generated for you. Subject to grading are exclusively files part of the project repository at the deadline. The deliverables are:

- Group Composition Deadline (**16.04.2023 23:59**):
  - Indicate the full names and GitHub usernames of your 3 group members in this Google Sheet.
- Intermediary Deadline (**26.04.2023 23:59**):
  - To be included at the root of your project
  - Project Description (`Description.md`): A Markdown document describing:
    - the context and basic narrative of your adventure game
    - player characters
    - rooms & exits
    - collectible objects
    - interactions (commands) and their effects/consequences
    - minigames & battles
    - point systems
  - Task Distribution as *issues* on GitHub
- Final Deadline (**14.05.2023 23:59**):
  - the Swift code of your adventure game
  - a `README.md` file containing
    - build instructions (if third-party APIs have been used)

        - any changes wrt. the initial project descriptions (additions, modifications, removals)
        - a brief description of how you realized the automatic game mode
        - any bonus features/easter eggs
        - a description of special Swift language features that you used and want to emphasize
        - known issues/bugs (if any)
    - a console output of a walkthrough (either manual or in automatic mode)
    - a subfolder `doc/` containing documentation generated out of your commented code (use a tool like *jazzy*; cf. Markup Formatting Reference, Swift Documentation)
    - a nicely illustrated map of your game world (rely on what you learned in your graphics course!)

Evaluation

The grade of your project depends on the following aspects:

- coverage of the stated requirements
- architectural design
- use of language features
- creativity, ramification and linguistical correctness of your storytelling
- documentation, reports (`Description.md`, `README.md`) & game map
- use of Git
- live demo & discussion

Important Dates

Milestone	**Date**
Start	12.04.2023
Groups Deadline	**16.04.2023 23:59**
Intermediary Deadline	**26.04.2023 23:59**
Final Deadline	**14.05.2023 23:59**

Contact

For all project-related questions, please contact me at aryobarzan.atashpendar@uni.lu!