Swift Project "Interactive Fiction"

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 (see Sheldon plays 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 you 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 **interconnected rooms**. Rooms do not necessarily need to be closed ones, it could also be a marketplace, a space station, or an ocean. The player starts in a given room and makes her way through the game map using textual commands. It can be one or more final rooms, leading the player to win or lose the game. Whether there is a happy end or not depends on you (check out Kobayashi Maru experiment). If 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. Exits might change over time: they could disappear (a bridge could fall together such that the player can't go back to his previous location on that path) or they could suddenly appear (a dark room only unveils its exits once the player has found a torch). 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, etc.) that the player can gain, invest or lose. There shall be different **player characters** from which the user can choose at the beginning of the game. Such characters might be upgraded during the game (e.g., a student wizard becoming voldemortesque). 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 can call them missions, quests, tasks, riddles, etc.). For example, the player would need to go over a river of lava in a minesweeper-like way; or she would need to decipher a cryptic 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** that can be found in rooms, and that may be required to win a minigame in that same or another room. She shall collect them in some sort of bag. For instance, a key might be necessary to open a door, or coordinates written on a piece of paper could be required for teletransportation. Gaining or losing objects could also be the result of winning 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 (monster, witch, cyborg, pirate, etc.). Entering a room containing an adversary could close some or all of its exits, requiring the player to confront him. A battle might require various actions (commands), using different arms 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. After a battle, the player may have won or lost one or several objects, or even have been set 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 as well as the path to a particular object that he wants 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, the necessary output shall be written to the console to understand what's happening. For this part, you may rely on recursion techniques, 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 and enjoyment in video games. For instance, the player might time travel back to a former state of the game, or he 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 provided here. The documentation can be found here (use the access credentials from the server you got the virtual machine from, see Moodle). An example application using this framework is available in the with lab solutions. You may use this example application as a starting point for your own adventure game.

Your first steps are:

- Read and understand the framework code and documentation
- Build and run the demo application (its dependency on the framework is already configured in the Package.swift file)
- Play around with the code, make some changes and try them out

Sample Scenarios

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

- Borgs aboard the USS Voyager Robinson
- Curse of Monkey Island
- Harry Potter at Campus Belval
- Dragons in caverns
- Getting off a delayed flight and trying to get to the boarding gate of a connection flight on time

Provide fascinating storytelling, have some fun!

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, options, closures, protocols, extensions, etc.)
- Demonstrate good OOP design (encapsulation, polymorphism, error handling) Properindentation and comments
- Output of all relevant information and storytelling on the console

Deliverables & Deadlines

The submission is handled through your group repository on GitHub. Check if your group (Team #) is already created when accepting the invitation. If you are the 1st one – create it yourself. Team table is published on Moodle. Subject to grading are exclusive files part of the project repository at the deadline.

The deliverables are:

- Intermediary Deadline: **30.05.2021, 23:59**
    - 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 via issues on GitHub

- Final Deadline: **20.06.2021, 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)
  - 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 final grade will depend on

- 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 (Luxembourg time)

Start	20.05.2021, 11:16
Intermediary Deadline	30.05.2021, 23:59
Final Deadline	**20.06.2021, 23:59**