# Rogue-like

Software Engineering Project Description

2016

## 1 General description

Dungeon crawls, or more specifically *rogue-likes*[1] are **turn-based** computer games in which a **single player** evolves through a procedurally generated dungeon, fighting creatures, finding various objects, acquiring experience. The final objective may be to reach the top (or bottom, if the dungeon is a cave) of the dungeon and come back, or to find a special object. Death is typically permanent, and plays can be very short, especially for beginners.

**Board** The game is played on a series of *dungeon levels* which are simple square grids. The grid cells may be empty, walls, or floor. Floor cells can contain objects, or a character. Characters can be computer-controlled creatures, or the player. The levels are procedurally generated, so that the dungeon never looks the same; different kinds of levels may be generated by different algorithms.

**Text-based graphics** The first rogue likes appeared before modern graphics existed, and many rogue-likes of today still implement text-based graphics (sometimes called ASCII graphics, though UTF-8 characters are sometimes used). The game is thus played in the terminal, and the view of the player and its surroundings looks like this:

```
. . . . . . . . . . #
. . . . . . . . . . # # #
. . @ . . . . . g . . . .
. . . . . . . . . . # # #
. . . . . . . . . . #
```

Here, the `@` sign represents the player. It is currently facing a goblin, represented by `b`, in a room whose floor cells are represented by `.` and walls by `#`. Of course, the game screen also shows stats about the player, its inventory, and perhaps some information about the room or what happened in the previous turn.

[1] `http://en.wikipedia.org/wiki/Roguelike`

**Keyboard control** Movements and actions are performed by simple key strokes. Keys `hjkl` are commonly used for directions, and combos such as `sl` may mean **s**hoot to the right (which is only possible if the player is suitably equipped).

**Artificial intelligence** Computer-controlled creatures may be hostile or friendly towards the player. They can interact with him, but also cooperate with each other. This means that several AIs have to be designed.

**Universe and quest** Rogue likes can depict universes that are not only dungeons, but may represent caves, forests, or even modern cities. Different game universes correspond to different sets of objects, characters, and different quests. The quest may consist of a single goal, a series of goals, or a more complex tree-like story. Besides this coarse-grained control of the game, more fine-grained control can be useful, e.g. to adjust the level of difficulty to keep the game exciting. *In the final project proposal, a fixed universe may be proposed or it may be proposed to implement several universes on top of a common game engine.* In any case, the game should feature different sorts of objects and characters, with several AIs, and the game engine should be as re-usable as possible.

# 2 Detailed objectives

Objectives for the first iteration are marked [$\alpha$], second iteration is marked [$\beta$] and optional objectives are marked [?].
The game should allow the following use cases:

- [$\alpha$] Playing.
- [$\alpha$] Saving the game state, exiting, resuming a game.
- [$\beta$] Choosing the species and characteristics of the player character before starting a game, through a series of configuration screens.
- [?] Store and remove objects from an inventory, inspect the inventory through a specific screen.
- [?] After the game is over, high scores may be shown.
- [?] A replay system of some sort may be designed.

Several procedural level generation algorithms will be created:

- [$\alpha$] Simple, fixed level for testing purposes.
- [$\alpha$] Read a level from a file.
- [$\alpha$] Simple random generation.
- [$\beta$] Algorithms for generating different kinds of levels.

Lighting:

[$\beta$] The game will feature a vision/lighting system, restricting the field of view of the player (and perhaps also of the creatures) based on various attributes (health, equipment, type of level, special events...).

Quest:

[$\alpha$] A quest/scenario mechanism will be designed.
At first, it may remain unused or used only in a basic way.

[$\beta$] The quest/scenario will be used in a richer way.

[?] A mini-game (e.g. sokoban) will be embedded in certain rooms.

Several kinds of creatures will be created:

[$\alpha$] Simple friendly creature following the player and attacker other characters.

[$\alpha$] Simple hostile creature attacking the player.

[$\beta$] More complex creatures featuring group behaviors.

# 3 Comments

## 3.1 Technological choices

This project is quite low tech. It can be tackled with any reasonable programming language, and only requires a library for accessing the low-level capabilities of terminals (e.g. curses). The backup facility may rely on an existing file format (e.g. JSON), database system (e.g. SQLite) or language facility for (de)serialization.

## 3.2 Possible extensions or variants

A modern graphical user interface, with bitmap or vectorial graphics, could be implemented. It can be a replacement for the ASCII interface proposed here; even better, two different interfaces could be realized on top of a unique backend.

The game may be made online in various ways. Without changing the game, it can be implemented as a client-server system, perhaps web-based, so that cheating is impossible, high scores are shared, and other's plays may be observed. Multiplayer versions may be considered as well but it requires a precise redefinition of the way the game is played.

## 3.3 Size of the team

This kind of project proposal works for a small team of students, but it should be possible to make it scale for the whole class, by opting for the more ambitious choices, and some of the extensions proposed in this document.