Project Milestone 8

CS 4500 Software Development

Due: Friday, April 2, 11:59pm

Submission: Create a release on GitHub tagged `p8`. Add a `p8-exec.zip` to the release, containing:

1. The `localSnarl` executable for the testing task
2. A `README.md` file explaining how to run and interact with the local game
3. Any packages/modules necessary for the above executable to function

Here is a sketch of the `p8-exec.zip` layout:

```
|-- localSnarl
|-- README.md
...
```

Submit a ZIP with the following on Handins:

1. For the programming task, any new or updated *relevant* source files under `Snarl/src/` and the description of adversary strategies under `Snarl/planning/adversary-strategies.md`.
2. For the testing task, a folder `Snarl/local/` with the source code for the local SNARL runner.

Here is an example sketch of the Handins ZIP layout:

```
Snarl
|-- planning
|   `-- adversary-strategies.md
|-- local
|   |-- localSnarl.<ext>
|   |-- README.md
|   ...
`-- src
    ...
```

Programming Task: Adversaries

Implement two adversaries: a ghost and a zombie, based on the interface designed in the previous milestone.

Information available to adversaries:

- An adversary gets the full level information (comprised of rooms, hallways and objects) at the beginning of a level.
- An adversary gets an update on all player locations, but only when it's about to make a move.

Movement rules for zombies:

- A zombie can move at most one tile at a time in a cardinal direction.
- A zombie cannot skip a move, unless there is no valid move in any cardinal direction.
- A zombie cannot move onto a tile occupied by another adversary.
- A zombie cannot move onto a door tile. A consequence is, it cannot leave the room it spawned in.

Movement rules for ghosts:

- A ghost can move at most one tile at a time in a cardinal direction.
- A ghost cannot skip a move, unless there is no valid move in any cardinal direction.
- If the ghost enters a wall tile of a room, it triggers an interaction whereby it is transported to a randomly selected room.
- A ghost can enter a door tile and move through hallways.

The strategies for determining the next move of adversaries are up to you, but they should be sufficiently different: a ghost should clearly take advantage of the additional movement possibilities. Additionally, it has to be clear that the adversaries react to a player in the vicinity by moving towards it.

Describe the strategies in `Planning/adversary-strategies.md` and include some example situations and the choices the adversary will make.

Scope: We will look for good code design, readability, unit tests, and whether we can find the functionality we asked for above in the code. Adversary strategies should be particularly clear and easy to follow.

Testing Task: Local Snarl

This testing task does not involve writing a test harness as before. Instead, this is an exercise in integrating the components written so far to produce a playable demo.

Implement `localSnarl`, integrating the Game Manager, Player (User) and Adversary components (implemented above) to allow local gameplay for at least one player. The minimum functionality should be as follows:

1. The executable should take the following optional arguments:

a) `--levels FILENAME` where `FILENAME` is the name of a file containing JSON level specifications (see description below). Default is `snarl.levels`.

b) `--players N` where $1 \leq$ `N` ≤ 4 is the number of players. If your local implementation only sensibly supports a single player, and the given `N` is greater than 1, print an error message saying so and exit. Default is `1`.

c) `--start N` where `N` is the level to start from. If `N` is greater than the number of available levels, the behavior is undefined. Default is `1`.

d) `--observe`: by default, only the players' view should be presented. If this option is given, an observer view (the full level) should be presented in addition to or instead of the player view. This implies `--players 1`.

2. The game should prompt each player for a user name and register the players with the given names.

3. After player registration, load the starting level, and populate it with the player(s) and a number of adversaries. The initial locations for players and adversaries should be randomly chosen valid placements (any traversable room tile, not already occupied by a player or an adversary and not containing a key or exit). The number of adversaries is determined using the current level number $l \geq 1$:

   - number of zombies $= \left\lfloor \frac{l}{2} \right\rfloor + 1$, translated to Pythonish: `math.floor(l / 2) + 1`
   - number of ghosts $= \left\lfloor \frac{l-1}{2} \right\rfloor$, translated to Pythonish: `math.floor((l - 1) / 2)`

4. If a player finds a key, is expelled, or exits, display a message of the form `"Player <name> <event>."`, where `<name>` is the user name of the player and `<event>` is one of

   - `found the key`,
   - `was expelled`, or
   - `exited`.

   The message can be printed to stdout or it can be displayed in the players' view.

5. A level ends when all players are removed from the level. If at least one of the players exited, advance to the next level. If they all got expelled, the game ends and the game should inform the players which level they failed in.

6. If the last level ends successfully, inform the player(s) that they've won the game.

7. After the game ends, print the number of times a player successfully exited ~~or was expelled~~ and the number of keys they found. If there is more than one player, rank the players by the number of times they exited (most to least), followed by the number of keys (most to least).

Write a `README.md` with instructions on how to run and play the game. The game needs to be runnable and playable on Khoury Linux VMs. Include information about whether the game can be played in a terminal or requires an X session to display a GUI.

The `.levels` File

A `.levels` file contains a sequence of JSON values and is formatted as follows:

```
(natural)

(level)

(level)

...
```

The **`(natural)`** is the number of levels contained within the file. If the actual number of levels in the file is less than the number given, the behavior is undefined. Each **`(level)`** is a valid level specification as given in Milestone 4 (with the `objects` field unordered). Note, that each **`(level)`** value can be spread over multiple lines.

Scope: The local game must run on the Khoury Linux VMs, accept the given optional arguments and behave as specified above. The provided instructions must be clear and easy to follow.