ИНСТИТУТ ЗА МАТЕМАТИКУ И ИНФОРМАТИКУ
ПРИРОДНО-МАТЕМАТИЧКИ ФАКУЛТЕТ
УНИВЕРЗИТЕТ У КРАГУЈЕВЦУ

СЕМИНАРСКИ РАД

РАЗВОЈ ТЕКСТУАЛНЕ RPG ИГРЕ У HASKELL-У

Ментор
Татјана Стојановић

Студент
Драган Миљковић

Август 2024.

Садржај

Увод

У свету програмских језика, *Haskell* се издваја због свог чисто функционалног карактера, нудећи јединствен приступ решавању сложених проблема. Овај семинарски рад истражује развој текстуалне *RPG* игре смештене у Париз тридесетих година 20. века, где играч преузима улогу детектива који је увучен у мистериозан случај. Игра користи *ASCII* уметност за приказивање сцена и омогућава играчима да доносе одлуке које утичу на наратив. Поред тога, у раду се детаљно описује имплементација кључних компоненти игре у *Haskell*-у и разматра потенцијал овог пројекта да служи као основа за развој *game engine*-а који омогућава креирање сличних *RPG* искустава.

Слика 1: Почети екран игре

1. Избор програмског језика Haskell

Haskell је чисто функционални програмски језик који нуди јасноћу и елеганцију у коду, као и снажне типске системе који помажу у откривању грешака у раним фазама развоја. Употреба *Haskell*-а у развоју текстуалне *RPG* игре омогућава коришћење апстракција и моћних алата за управљање стањем и улазом/излазом, што значајно олакшава рад на оваквом пројекту.

2. Управљање пројектом помоћу Cabal-а

Cabal је алат за изградњу и управљање *Haskell* пројектима. Он омогућава једноставну конфигурацију, управљање зависностима и компилацију кода. У овом пројекту, *Cabal* се користи за конфигурисање пројекта и осигуравање да су све потребне библиотеке правилно укључене.

2.1. Иницијација Cabal пројета

Први корак у раду са овим пројектом је иницијација новог *Cabal* пројекта:

```
cabal init
```

Током овог процеса, потребно је дефинисати основне информације о пројекту и укључити све релевантне зависности.

2.2. Конфигурација *.cabal датотеке

`*.cabal` датотека садржи све информације потребне за компилацију пројекта. Важно је осигурати да су све потребне библиотеке укључене у `build-depends` секцију:

```
1  build-depends:
2            base >=4.13 && <4.14
3          , ansi-terminal >= 0.6
4          , directory >= 1.3
5          , text >= 1.2
6          , filepath >= 1.4
7          , split
```

*Део *.cabal конфигурационог фајла*

2.3. Компилација и покретање пројекта

Након конфигурације, пројекат се може компајлирати и покренути помоћу следећих команди:

```
cabal build

cabal run
```

2.4. Додавање нових модула

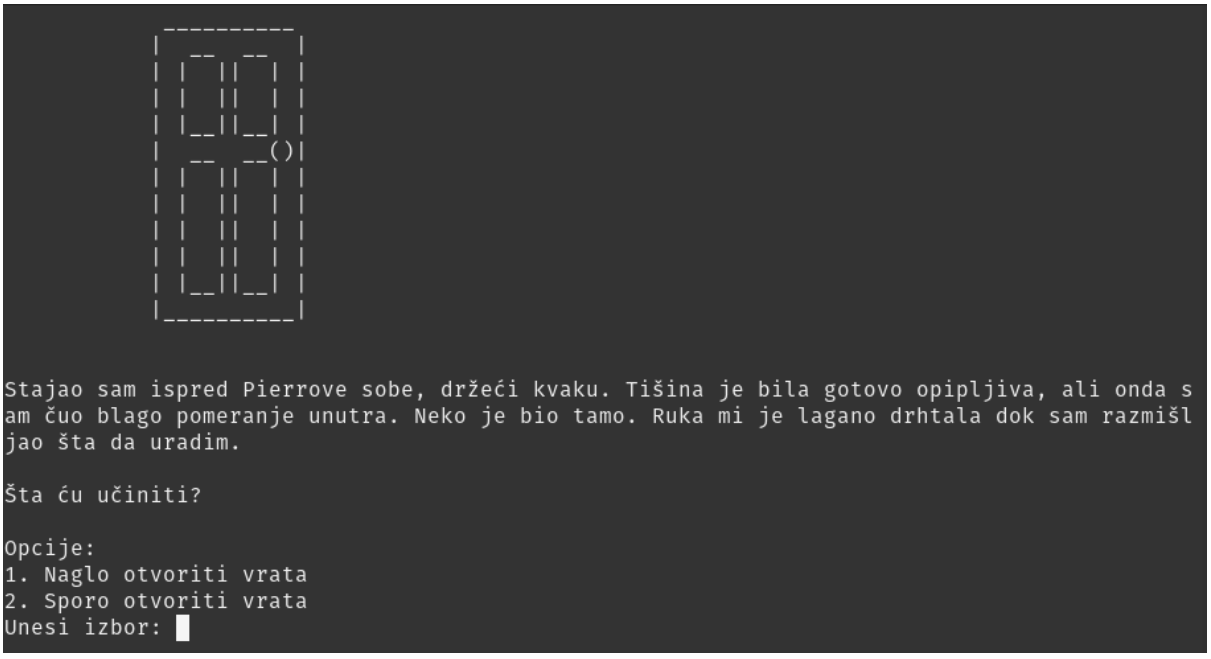
Сваки нови модул који се додаје у пројекат потребно је укључити у `library` или `executable` секцију `*.cabal` датотеке, како би био правилно компајлиран и доступан у пројекту.

3. Прича игре

Прича игре је смештена у Париз током 1930-их година, где играч преузима улогу детектива који покушава да реши мистериозни случај. Игра почиње у скромном стану на Монмартру, где детектив добија писмо које га води у низ различитих истрага и авантура. Како се прича одвија, играч се суочава са низом сложених одлука које ће обликовати ток догађаја и на крају довести до различитих завршетака игре.

Једна од најзанимљивијих карактеристика игре је управо та могућност да играч утиче на крај приче. У зависности од избора који се праве током игре, детектив може доћи до различитих решења случаја, односно до различитих крајева игре. Постоје добри завршетци, где детектив успешно решава мистерију и излази као херој, али и лоши завршетци, где погрешне одлуке могу довести до фаталних последица, било за детектива, било за невине људе укључене у причу.

Различити завршеци игре се постижу путем сложене мреже избора и последица. На пример, одлука да се пође у одређену истрагу може открити важан траг или, супротно томе, довести детектива у замку. Постоје тајни путеви и скривени трагови који могу отворити нове сцене или омогућити играчу да избегне опасност. Ово подстиче играча да истражује различите опције у више пролазака кроз игру како би открио све могуће завршетке и открио све детаље мистерије.



Слика 2: Избор у игри

Такође, структура игре омогућава више различитих путева који воде ка завршетку приче. Нека решења су очигледнија и лакша, док су друга сакривена и захтевају пажљиво разматрање сваке опције и сваког трагова. Ова мултипликативност избора и њихових исхода даје игри изузетну дубину и реиграбилност, јер сваки нови пролазак кроз игру може донети нова искуства и открића.

Слика 3: Избор опције за нагло отварање врата

4. Формат сцена

Сцене у игри су дефинисане у текстуалним датотекама и свака сцена садржи следеће елементе:

- ***ASCII* арт** - Графички приказ који визуелно представља сцену. Ово је обично први део у датотеци.
- **Опис сцене** - Текстуални опис који даје контекст и детаље о сцени коју играч тренутно истражује.
- **Опције** - Листа могућих избора који играч може направити. Свака опција води до нове сцене.

Ево примера структуре датотеке која представља једну сцену:

```
    _______
   /       \
  /   O O   \
 |     >     |
  \  \___/  /
   \_______/

Nalazim se u mračnoj uličici Pariza. Svetla trepere dok čujem korake u daljini.

Prati korake -> scene2
Sakrij se iza kontejnera -> scene3
```

Пример фајла који представља једну сцену у игри

5. Имплементација

Игра је имплементирана кроз неколико *Haskell* модула који су организовани на следећи начин:

5.1. Main.hs

`Main.hs` је главни улазни фајл који покреће игру и управља њеним током. У овом фајлу се иницира прва сцена коју играч види. Након тога, игра приказује *ASCII* уметност која је саставни део сваке сцене, описује шта се дешава у тој сцени, и приказује доступне опције које играч може да изабере. Функција `displayScene` је одговорна за приказивање тренутне сцене на екрану и обраду играчевог избора. Када играч направи избор, функција прелази на следећу сцену на основу датотеке која је повезана са изабраном опцијом. Функција `getChoice` се користи за читање и валидацију играчевог уноса, осигуравајући да је унесен исправан број који одговара једној од понуђених опција.

```
1  module Main where
2
3  import Scenes
4  import Types (Scene, asciiArt, description, options)
5  import System.Console.ANSI
6  import System.IO
7
8  main :: IO ()
9  main = do
10     hSetBuffering stdout NoBuffering
11
12     startScene <- readScene "start"
13     displayScene startScene
14
15 displayScene :: Scene -> IO ()
16 displayScene scene = do
17     clearScreen
18     setCursorPosition 0 0
19     putStrLn (asciiArt scene)
20     putStrLn ""
21     putStrLn (description scene)
22     putStrLn ""
23     putStrLn "Opcije:"
24     mapM_ (\(idx, (opt, _)) -> putStrLn (show idx ++ ". " ++ opt)) (zip
   [1..] (options scene))
25     choice <- getChoice (length (options scene))
26     let nextSceneFile = snd (options scene !! (choice - 1))
27     nextScene <- readScene nextSceneFile
28     displayScene nextScene
29
30 getChoice :: Int -> IO Int
31 getChoice numOptions = do
32     putStr "Unesi izbor: "
33     choiceStr <- getLine
34     if null choiceStr
```

```
35          then do
36              putStrLn "Nevažeći izbor. Molimo pokušajte ponovo."
37              getChoice numOptions
38          else case reads choiceStr of
39              [(choice, "")] | choice >= 1 && choice <= numOptions -> return
    choice
40              _ -> do
41                  putStrLn " Nevažeći izbor. Molimo pokušajte ponovo."
42                  getChoice numOptions
```

Main.hs

5.2. Scenes.hs

`Scenes.hs` је модул који садржи функције за учитавање и парсирање сцена из текстуалних датотека. Свака сцена у игри је сачувана у засебној датотеци, која садржи три кључне компоненте: *ASCII* уметност (која визуализује сцену), наративни опис (који пружа контекст и детаље), и листу опција (које воде до других сцена). Функција `readScene` је одговорна за читање садржаја датотеке, њено парсирање у три дела, и креирање објекта типа `Scene`. Овај објекат се затим прослеђује у `Main.hs` како би се приказао играчу. Такође, модул садржи функцију `parseOption`, која парсира опције у формату "`<назив опције> -> <датотека наредне сцене>`", што омогућава динамичко навођење путање до наредне сцене у зависности од избора који играч направи.

```
1   module Scenes where
2
3   import Types
4   import Data.Char (isSpace)
5   import Data.List.Split (splitOn)
6
7   readScene :: FilePath -> IO Scene
8   readScene filePath = do
9       let realPath = "src/scenes/"++filePath++".txt"
10      contents <- readFile realPath
11      let [art, desc, optsStr] = splitOn "\n\n" contents
12      let optionsList = map parseOption (lines optsStr)
13      return Scene { asciiArt = art, description = desc, options =
    optionsList }
14
15  parseOption :: String -> (String, String)
16  parseOption optionStr =
17      let [option, sceneFile] = splitOn " -> " optionStr
18      in (trim option, trim sceneFile)
19
20  trim :: String -> String
21  trim = f . f
22    where f = reverse . dropWhile isSpace
```

Scenes.hs

5.3. Types.hs

`Types.hs` дефинише основне структуре података које се користе у игри. Главни тип података је `Scene`, који представља једну сцену у игри. Свака сцена је моделована са три поља: `asciiArt` (за чување *ASCII* уметности која визуелно представља сцену), `description` (за чување текста који описује шта се дешава у сцени), и `options` (листа

опција које играч може да изабере, при чему свака опција води до одређене наредне сцене). Ова структура омогућава лако проширење игре додавањем нових сцена у засебним текстуалним датотекама, без потребе за изменама кода у другим деловима програма.

```
1  module Types where
2
3  data Scene = Scene
4      { asciiArt :: String
5      , description :: String
6      , options :: [(String, String)]
7      } deriving (Show)
```

Types.hs

Закључак

Закључак ове игре показује како текстуално засноване авантуре, у комбинацији са једноставним али ефектним *ASCII* уметничким приказима, могу створити дубоко импресивно искуство које подстиче машту играча. Иако наизглед минималистичка у свом приказу, игра пружа богато наративно искуство које је у стању да ангажује играче кроз своје изборе и сложене приче.

Коришћење *Haskell*-а за имплементацију ове игре показало је колико моћан и флексибилан овај функционални језик може бити, чак и за развој игара. Структура кода, која је чиста и интуитивна, омогућава лако проширивање игре и додавање нових сцена, чиме се ствара могућност за будуће надоградње и модификације. Са својом једноставном, али ефективном употребом уноса и излаза, систематским парсирањем текстуалних фајлова и креирањем динамичких прича, ова игра показује како је могуће створити богата искуства уз коришћење функционалног програмирања.

Као последњи део, могућност креирања сопствених сцена и прилагођавања приче чини ову игру више од обичне авантуре; она постаје платформа за стварање и истраживање нових прича. На крају, овај пројекат представља успешну синтезу програмирања, приповедања и уметности, показујући да чак и једноставна графика и текст могу створити дубоко и незаборавно искуство за играче.