narratr

A language for creating text adventures

Team 2

Team Members

Shloka Kini—srk2169 (Project Manager)
Nivvedan Senthamil Selvan—ns2984 (Language Guru)
Yelin Hong—yh2617 (System Architect)
Jonah Smith—jes2258 (System Integrator)
Cecilia Watt—ciw2104 (Tester and Validator)

COMS W4115: Programming Languages and Translators
Spring 2015
Columbia University

Overview

narratr is a language to create text adventures, also known as interactive fiction. Text adventures involve users playing as a character in a virtual world, with with the user can interact only through text commands. The text adventures that narrator can create are games - with winning and losing conditions. Creating text adventures with a general-purpose programming languages is tedious because there are two crucial features of these games that can be more intuitively modelled with a domain-specific programming language:

1. Text adventures behave like state machines. They begin in a starting state, potentially move to other states based on input commands, and have final states - winning and losing.
2. Every state is executed in a Read-Eval-Print Loop (REPL) until an input command causes a transition to another state.

Text Adventure Games are great fun, and creating an easy, domain-specific programming language for these games could convince people with great ideas, who are otherwise put off by the tediousness of general programming languages, to create games more easily.

Users - Programmers and Consumers

Since the logic involved in text adventures is non-trivial, narratr will be a Turing complete language. narratr is intended at game developers who are interested in deveoping text adventure games. We expect that the users who would program in narratr have some prior experience in general programming. The syntax structure of narratr will however greatly simplify the control flow. Users, therefore, need not be expert programmers.

On the other hand, programs created with narratr can be used to play games by anyone who can type. They are the end users of our narratr.

Properties of narratr

The primary goal of narratr is to make creating text adventure games easier. It allows the user to program in a python-like language which is both text heavy and flexible and output a game which can give interesting feedbacks based on different interactions. narratr provides a concise, simple yet creative syntax for creating text adventure games which can be played via several platforms.

narratr’s design paradigm is a set of scenes (states) and transitions based on user input, closely modelling the execution of these games. It is also object-oriented, and all the items and characters in the adventure are modelled as objects.

Other languages/tools for creating text adventures

Languages

We are aware of two other programming languages for creating text adventures. The first is called TADS, which stands for Text Adventure Development System, and has been around since the late 1980s, though the language was overhauled to its modern form in 2006 with the release of TADS 3. The designers of TADS say the syntax of this overhauled release is based on C++ and JavaScript. The language has been used for several award-winning interactive fiction works. The newest version of the language can also serve the games over the internet.

The other language designed specifically for creating text adventures was another project for Professor Aho’s Programming Languages and Translators class. In Spring 2013, a team created a language called IFL, which stands for Interactive Fiction Language. It appears to have been an object-oriented language based around types with pre-programmed action sequences and gameplay options for the user.

It is also, of course, possible (albeit cumbersome) to program text adventures using general-purpose languages.

Tools

We are also aware of two tools for creating text adventures. The first is ADRIFT, a Windows application allowing developers to create text-based adventures using flow charts. (Gamers can use the accompanying ADRIFT Runner application to play games on Windows, Mac, or Linux computers, as well as a web browser.) The interface is drag-and-drop, and while initially overwhelming, the developers claim it “has a long standing reputation for being the easiest system to use for creating interactive fiction.”

Inform falls somewhere on the spectrum between a programming language and a tool. Rather than write code, programmers write natural language instructions that are then turned into a game. This code, however, is read and interpreted within a desktop application, from which it can be exported and run using a variety of different tools.

What makes narratr different?
First, a domain-specific language is clearly preferable to general-purpose programming languages for creating text adventures. The unique structure of text adventures (particularly related to flow-control and the maintenance of a ‘state’) makes implementation cumbersome in general-purpose languages.

TADS, while very flexible in its ability to make different types of text adventures, is a bloated language, requiring a complex setup for both developers and players and very strong programming background for developers. It also allows game developers to integrate multimedia elements, which, while interesting, is antithetical to the goal of text-based adventures. Compared to TADS, narratr will be lightweight and focused on serving purely textual games.

Additionally, narratr aims to improve some areas of weaknesses of IFL. For example, where IFL accepts only a pre-programmed selection of actions from the gamer, we imagine our gamers interacting with the game using arbitrary textual input. Though this will make our (and the programmer’s) work a bit harder, the payoff in gameplay will warrant the challenge.

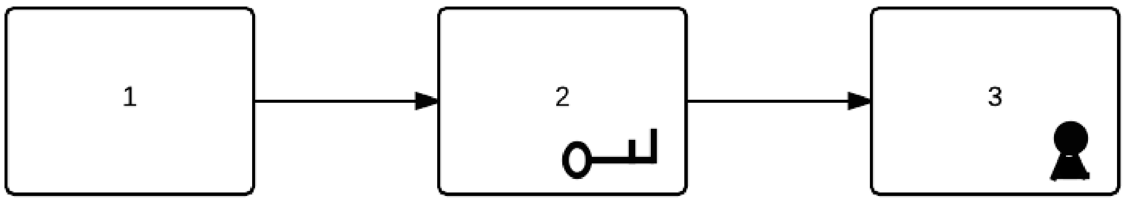
In contrast to tools like ADRIFT and Inform, narratr has tremendous flexibility for game construction and design. While both tools make writing interactive fiction easy, they are only as customizable as their frameworks allow. With narratr, games can be designed in infinitely many ways, including many we will not have thought of while designing the language. As such, we aim to strike a balance between the flexibility and power of a programming language and the ease-of-use and readability of these tools.

narratr in Action

Hello World

```
scene $1 {
      setup:
            say “Hello, World!”
      action:
            win
      cleanup:
}
start: $1  # Optional
```

A more involved example for creating a game with three rooms.



A player starts in room 1, walks into room 2 to pick up the key, and walks into room 3 to unlock the lock.

```
scene $1 {
     setup:
          exposition “You had a lot to drink last night. You don’t
     know where you are. Do you even know who you are? God, you’re a
     mess.”
          moves right($2) #action scope
     action:
          say “You can move right. What would you like to do?”
     cleanup:
}

scene $2 {
     setup:
          exposition “You’ve entered a strange room decorated with
     beautiful woven baskets. You can hear a wolf howling in the
     distance. You’re feeling slightly better, but the world is
     still spinning. On the table in the corner, there is a key. It
     seems to call to you.”
          moves left($1), right($3)
          has key(“the bronze key”, 1)
     action:
          say “You can move left or right. What would you like to
     do?”
          response.get()
          if (response is “pick up key”) AND (room has key(1))
               pocket.add key
     cleanup:
          say “‘Goooooodbye,’ howls the wolf.”
          if (key.pickedUp)
               exposition “You’ve entered a strange room decorated
     with beautiful woven baskets. You can hear a wolf howling in
     the distance. You’re feeling slightly better, but the world is
     still spinning. There is a table in the corner.”
}

scene $3 {
     setup:
          exposition “Now you’re in the last room flooded by
sunlight. But there are no windows. There’s a safe in the corner...
but it’s locked.”
          moves left($2)
          has lock(1)
     action:
          response.get()
          if response is “use key”
               lock.open
          if lock.opened
               win
     cleanup:
}

item key(n,i) {
     name is n
     id is i
     description is “A bronze key. It feels heavy in your hand. You
could probably whack someone with it and do some real damage.”
     pickedUp is FALSE
}

item lock(i) {
     id is i
}
```

A room of many ponies. Every time you enter the room, you receive a pony for each coin you have in your pocket.

(A pocket is a structure in our language which holds the user's objects. It is automatically initialized to hold nothing but can hold different objects at the start of the game if explicitly stated.)

```
scene $4 {
     setup:
          exposition "An infinitely large room. The gentle smell of
     manure and princess glitter."
          has ponies[]
     action:
          while(pocket.coins){
               ponies[].add new pony
               say "You got a pony!"
               pocket.coins[].remove_last()
          }
          say "You received " + i + " ponies."
     cleanup:
          say "The ponies don't fit in your pocket so now " +
     ponies[].length + " are gonna follow you around."
}

item pony {
}
```