Manager: Morgan Zee (mbz2112)
Language Guru: Shirley Ye (sy2650)
System Architects: Saumya Agarwal (sa3656) and Xinye Jiang (xj2253)
Tester: Janelle Ponnor (jp4024)
COMS 4115, Spring 2021

Racontr Language Proposal

Introduction - Description of Language & Motivation

The programming language we plan to implement allows users to design text adventure games. The language will provide the user with multiple predefined storylines to choose from, and depending on what option the user selects, the story will evolve in different ways. The language will account for the various decisions users make as they interact with other characters and objects in the adventure world. In terms of the motivation behind our language, we were inspired by the music languages students built in previous semesters; as music has many predefined rules (i.e. key signature, notes, etc.), we found that in a similar way, stories have several components (i.e. characters, settings, possible actions, etc.) we could use. Building on this idea, we found an existing interactive storytelling language called Inform7 (http://inform7.com/), a system and programming language used to produce interactive stories, which we plan to base our language off of. Similarly, the Artificial Intelligence-driven Dungeons and Dragons game (https://play.aidungeon.io/main/landing), which allows users to input any action, storyline, or dialogue and the AI seamlessly incorporates it. Netflix's Black Mirror: Bandersnatch also takes on this interactive fiction model, which allows the user to select the fate of the characters throughout the episode. We hope to draw on elements of these existing languages and interactive fiction experiences we found in the language we build. Through this language, we hope users can write interactive fiction game programs.

Language Features

1. Lexical and Syntactical Conventions

There are five kinds of tokens: identifiers, keywords, strings, constants, expression operators, and other separators. Semicolons are required as separators at the end of every line of code. A tab is required after if and while statements.

The fundamental block of the language is a phrase in the form:

(<descriptive>)<noun><actions>

2. Identifiers and Keywords

An Identifier is a sequence of letter and digits, where the first letter is alphabetic. It defines the type of a variable. These are the reserved identifiers: int, things, descriptive, action, rule, succeeds, fails, create, initialize, set, match, auto, static, goto, return, sizeof, break, continue, if, else, for, do, while, case, default, alive, dead.

3. Loops and Conditionals

Loops: *Racontr* supports for loops like those in Python. There would be a counter variable “i” which is initialized outside of the for loop. The range will be specified at the start of the for loop (i.e. “for i in range (1, len(arr)))”. The range can be specified in terms of an iterable object. Inside the for loop, the counter variable will have to be incremented. An example of the use of the for loop is to iterate through a data structure that contains a character’s possessions and then print its contents.

Conditionals: *Racontr* supports if, elif, and else statements as in Python. If statements start with the conditional predicate and end with {. The contents of the conditional are indented and describe actions to do if the predicate is met. After the last action in the body of the if conditional, a single } will be on the next line and it will be aligned with the if condition. If there is an additional special condition that the user wants to define for the same variable that was tested in the if statement, the same syntax will be followed, except the keyword will be elif. If the user wants to define an action that should be taken if neither the if and elif conditions are met, then the same syntax will be used, except this time the keyword is else.

4. Classes, Types, and Objects

Racontr supports two declarable storage classes: automatic and static; it has six types: boolean, thing, descriptive, noun, action, integer.

Racontr comes with two fundamental types of objects: characters and integers, which leads to lists, functions, and structures of various object types.

Characters: Racontr allows users to define characters as objects. They can specify information about the characters including their gender, age, character name, and possessions for example. An example scenario would be a game where a user has a magic pouch, where they can define the items that go inside of that pouch. It might look like this: struct Magic Pouch capacity = 10, which might be a stack that stores a list of items.

Rules: Our language allows users to define their own rules in their programs. They can do this by writing a series of If and Else statements. These statements would translate into boolean values that would indicate the effects on the game player. For instance, a sample scenario would be when Player 1 meets Player 2, if Player 1 loses the challenge, something will happen to that player. The programmer is free to define the rules as they like with these conditionals.

5. Operators:

We follow Python’s guidelines for definitions of most operators in addition to some of Java’s definitions for convenience.

-Arithmetic: +, -, *, /, %, //, **, ++, --

-Comparison: <, >, ==, !=,<=, >=

-Logical: and, or, not
-Assignment: +=, -=, *=, /=, %/, //=, **=

6. Additional Features
We would like to design *Racontr* to have automatically managed garbage collection reminiscent of Java to avoid functions like free() in C.
Racontr will have a static type scope (like Python) so that variable assignments are dependent on the opening and closing curly brackets {}.
Like Python, *Racontr* is also weakly typed in order to allow different attributes of the character class to be contained in the same variable or data structure if needed. *Racontr* will also be dynamically typed.

Source Code

Source Code For Hypothetical Game

```
**Create Character:
Import lib.coc;
def character_set():
    initialize Character[protagonist] p1;
    set p1.name = input();
    initialize Character[monster] m_main = lib.coc.azathose;
    initialize Character[protagonist] p2;
    set p2.name = “Lisa”;

def plate():
    initialize Plate[2by2] base[unlimited];
    initialize random Adventure ad[15];
    set base.adventure = ad;
    match base.adventure random_odd;

**Create Rules:
def rule(default):
    if default enters scene 3:
        set default.visibility = “invisible”;
    if default is in scene 6 and sizeof(scene) is 1:
        succeed default;
```