Gema Almoguera
(UNI: GA2347)
Project Proposal

Description of the language

The proposed project is an interactive fiction language to program text-based adventure games.

This is a type of text game that mixes writing and programming. The writer creates the code so that the player becomes the main character of the game. The player interacts with the elements of the game by using a command line to type actions. These actions can be movements (north, south, etc.), interactions with the environment (take sword, play flute) or game commands (inventory, look). The story progresses as the player performs the right actions, uses the right items, and explores the correct rooms.

The purpose of this project is to write a reduced version of one of the most used programming languages for interactive fiction, called TADS3 (Text Adventure Development System version 3), and so the syntax and semantics used will look similar to the original language.

How it should be used

The writer of the story is the programmer. Once the writer has a story, they will create the rooms, items, and actions in the story, defining how the game must behave depending on the commands typed by the user.

More concretely, the programmer will be able to create item and location objects. These objects can have properties (similar to attributes in Java) and methods.

The programmer will use these methods to attach certain actions to objects and the results of such actions. For example, if there is an object in our game called ‘Flute’, we can write a method to explain how the story progresses when the player types ‘play flute’ and what happens if a different action is performed with the flute. Actions that are applied to objects are composed of a verb in the imperative mood and a noun.

The writer can use if/then/else operations for the methods and variable assignments. The main data types used are strings and booleans, but other types might be added if deemed necessary to write basic games.

Once the code is finished, the writer can compile the algorithm into bytecode. The player uses an interpreter on the bytecode to play the game. Please note that the details on how to compile and interpret the game will be decided once the scope of the project is better defined.

Code examples

The game will start by defining a game object, which includes an introduction that can be customized by the writer:

```
gameMain: GameMainDef
showIntro()
{
"Welcome to your new adventure! You are a new castle guard looking for the queen's lost ring. If you find it, the queen will allow you to become her personal chef – Your childhood dream! Armed only with your sword and and a lamp that's running out of oil, you find yourself at the entrance of a cave. It's getting dark and you can hear the wolves howling close. What are you going to do now?"
}
```

This is an example of the definition of a room object, with an in-game name, description, and an exit as properties. As seen below, comments are allowed and their syntax is similar to Java's.

```
bedroom: Room
    {   roomName = 'Bedroom'
        desc = "This is a luxury room that you found inside the cave, with a full bathroom
                and a nice view of the Caribbean."
        east=TerrifyingDangerousCave    // Where the player will go if they type 'east' from here.
    }
```

Items follow a similar format, but can have methods that describe actions related to the item:

```
ring: Thing
    {   vocabWords = 'small silver ring'    // These are the words that the player can use.
        name = "Ring"
        location = bedroom          //  This is where the ring will be found.
        desc = "It's a small silver ring with a a black flower"

        action (action)
         { if (Action == "Take")
           then say "This is not the ring you are looking for. ";
    }
```

If the player meets certain conditions, the writer can finish the game by calling a function:

```
finishGameMsg ('Sorry! The giant squid ate the ring and the game is over.')
```

This function will display the ending message, wait for the player to press a key, and finish the game.

Standard libraries

Even though the programmer is responsible for any programming that is relevant to his or her story, the programming language provides some functions already incorporated for the writer, for example, the following basic commands:

- Look/examine
- Inventory
- Basic direction movements (north, south, east, west, northwest, northeast, southwest, southeast and their abbreviations).
- Error messages when the actions are not recognized

The game will keep track of other elements like the current location of the player, their inventory, and which items and locations have been discovered so far.

Even though the more complicated commands would be programmed by the writer, any other basic commands necessary to play a simple game would be included with the programming language.