GOLD (Game Oriented Language for DnD)
Language - Project Proposal
COMSW4115

Language Guru: Timothy E. Chung (tec2123)
System Architect: Aidan Rivera (ar3441)
Manager: Zeke Reyna (eer2138)
Tester: Dennis Guzman (drg2156)

1. Introduction and Motivation:
Our team decided that the creation of a game language would be the most enjoyable for all involved, and that to reasonably do such, a turn-based interactive game on a 2 dimensional matrix would be doable. Essentially, our language contains common constructs in games that allow developers to quickly prototype their game’s environment. This way, they can focus on assembling the business logic of the game rather than worrying about writing a backend that keeps track of the states of the game like Characters and location of the Characters.

2. Language Overview:

   **Primitive Data Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Regular integer that is 64bit</td>
</tr>
<tr>
<td>bool</td>
<td>Boolean, true/false</td>
</tr>
<tr>
<td>string</td>
<td>String, “”</td>
</tr>
<tr>
<td>amount</td>
<td>(ie. 2gold) integer followed by string</td>
</tr>
<tr>
<td>typeStruct</td>
<td>(ie. Item, Character) Definition with fields and their types; all our default typeStructs are denoted by capitalized letter and camel casing like in “Item” and “Character”</td>
</tr>
<tr>
<td>null</td>
<td>Null is the uninitiated state</td>
</tr>
</tbody>
</table>

   **Supported Data Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Array of any primitive.</td>
</tr>
</tbody>
</table>

Basic Keywords and default typeStruct (typeStruct is capitalized while actions start with lowercase)
### Declaration

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type()</td>
<td>Creates an object of Type that is a typeStruct</td>
</tr>
<tr>
<td>Type{()},()</td>
<td>Create an array of Type is a typeStruct</td>
</tr>
</tbody>
</table>

### Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-,+,*,/,-=,+=</td>
<td>Operator and shorthand operators for multiplication, subtraction, addition, and division.</td>
</tr>
<tr>
<td>&gt;, &lt;, ==, !=</td>
<td>Boolean logical operator</td>
</tr>
<tr>
<td>=</td>
<td>Assignment operator</td>
</tr>
</tbody>
</table>

3. **Examples and Sample Program**

Our goal is to create a language that allows developers to quickly build a gaming backend and the game’s business logic using abstractions most games have like Characters and Barriers. Our language aims to provide a convention and framework to abstract all the recurring logic and constructs in most common games while leaving space for customization.
let add2health = func(myCharacter typeStruct) {
  if (myCharacter.health < myCharacter.maxHealth - 2) {
    myCharacter.health += 2;
  } else {
    myCharacter.health = myCharacter.maxHealth;
  }
}

let add8health = func(myCharacter typeStruct) {
  if (myCharacter.health < myCharacter.maxHealth - 8) {
    my_character.health += 8;
  } else {
    myCharacter.health = myCharacter.maxHealth;
  }
}

// Program must have dim specified
dim(7, 7); //Back end prevents out of bounds movement, spot declaration

/* Item defined by tuple of name, type, characteristic, x-position, y-position ... */
Item("sword", "weapon", add2health, Character, 3, 4);
Item{("green_potion", "potion", add8health, Character, 3, 3),
  ("hummus", "potion", 2, 1, 6)};

/* Char defined by tuple of name, type, location, health, other stats... */
Character("Zekius Penius", "Warrior", 100, 0, 0);
Character{("Aldario", "Mage", 80, 0, 1),
  ("Dennis le Menace", "Rogue", 90, 1, 0)};

/* Barrier defined by tuple of type, starting spot, ending spot CANNOT BE DIAGONAL */
Barrier("wall", 4, 6, 7, 6);

/* Enemy defined by tuple of name, boss?, location, health, other stats, reward */
Enemy{("Goblin", false, 2, 4, 30, 20g),
  ("Kilgore", true, 5, 6, 200, 2green_potion)};

/* ---------- (Backend definitions: defaults, types, etc.) -----*/
Character {
    name = string;
    type = string;
    location = int;
    health = int;
    otherStat = int;
}

Enemy {
    name = string;
    isBoss = bool;
    location = int;
    health = int;
    otherStat = int;
    reward = int // this is the worth of the Enemy when captured
}

Barrier {
    type = string;
    x1 = int; /* if not specified or default, randomized */
    y1 = int;
    x2 = int;
    y2 = int;
    permeability = bool false; // this means default false
}

Item { /*can be held in inventory*/
    name = string;
    type = string;
    effect = func; // the effect func must accept a typeStruct that is
    target = typeStruct;
    x = int;
    y = int;
}