TBAG: The Text-Based Adventure Game Language

Project Proposal

Gregory Luan Chen <glc2121>, Yu Chun (Julie) Chien <yc2937>,
Maria Van Keulen <mv2482>, Brian Slakter <bjs2135>, Iris Zhang <iz2140>

September 30, 2015

Describe the Language you Plan to Implement

Text-based adventure games were first introduced in the late 1970's, and while gameplay technology and graphics have evolved considerably since then, these games still remain popular and new ones are continuously being developed. These games may differ on some details, but many of them operate on similar principles - there is a map that a user travels through, this map contains different rooms, and the rooms contain different monsters and items that the user can interact with. Because of these commonalities among different games, there is no doubt similarity in the development process. We propose developing TBAG, a language that makes building these game elements a simple process, where typical components (map, rooms, non-player-characters (NPCs), items) will be built in as data types.

Explain what sorts of programs are meant to be written in your language

With TBAG, developers will be able to write programs that create an interactive world for gameplay. More specifically, some components of this program would encompass:

- creating a room with a description and various NPCs or items
- creation of a larger world composed of rooms with links between them
- a character generation process with I/O
- defining interactions between characters and NPCs
- loading json files written by a non-programmer to prevent hard-coding

As a more concrete example, a developer may be able to create a fantasy world with a hero whose quest is to find a hidden treasure, defeating any foes he may encounter along the way. In such a game, the hero may navigate through a dungeon, which is composed of smaller rooms. In some of the rooms, there will be powerups the hero can collect to boost his HP or armor. The developer can use the language to specify the types of items that exist in each room, and how the user can interact with these items to gain their associated attributes. In other rooms, there may be monsters such as trolls or dragons that the hero must battle to continue on his quest. In this case, the developer will specify attributes of the NPC such as strength or attack damage, as well as the interaction between hero and NPC. Once fully developed, the program will allow a player to function as the hero in a playable, interactive game.

Language Basics

Primitives:

Primitive	Examples
`int`	-100, 0, 42
`boolean`	TRUE, FALSE
`char`	‘t’, ‘b’, ‘a’, ‘g’
`string`	“Hi!”

Operators:

Operator	Description	Associativity
`* / %`	Multiplication, division, modulo	Left-to-right
`+ -`	Addition, subtraction	
`< <=`	Inequality Operators: Less Than, Less Than Or Equal	
`> >=`	Inequality Operators: Greater Than, Greater Than Or Equal	
`== !=`	Equal, Non-Equal	
`&&`	Logical AND	
`||`	Logical OR	

Keywords:

Keyword	Description
`if, elif, else`	Control flow statements
`while`	Loop statements

Unique Globals

- Objects that are automatically initialized in each program
- Each program contains exactly one of each Unique Global
- Cannot be created or destroyed by the user

RoomTable

A hash table containing the game's various Rooms (array implementation).

Field Summary:

Type	Field and Description
`Room[]`	**`theArray`** The array of Rooms
`int`	**`currentSize`** The number of occupied cells

Interface Summary:

Type	Method and Description
`void`	**`insert`**`(String x)` Inserts x into the hash table.
`void`	**`remove`**`(String x)` Removes x from the hash table.
`int`	**`hash`**`(String x)` Returns the hashval of x.

ItemTable

A hash table containing the game's various items (array implementation).

Field Summary:

Type	Field and Description
`Item[]`	**`theArray`** The array of Items
`int`	**`currentSize`** The number of occupied cells

Interface Summary:

Type	Method and Description
`void`	**`insert`**`(String x)` Inserts x into the hash table.
`void`	**`remove`**`(String x)` Removes x from the hash table.
`int`	**`hash`**`(String x)` Returns the hashval of x.

NPCTable

A hash table containing the game's various NPC's (array implementation).

Field Summary:

Type	Field and Description
`NPC[]`	**`theArray`** The array of NPC's
`int`	**`currentSize`** The number of occupied cells

Interface Summary:

Type	Method and Description
`void`	**`insert`**`(String x)` Inserts x into the hash table.
`void`	**`remove`**`(String x)` Removes x from the hash table.
`int`	**`hash`**`(String x)` Returns the hashval of x.

Player

Contains player attributes

Field Summary:

Type	Field and Description
`int`	**`health`** the Player's health stat
`int`	**`attack`** the Player's attack stat
`String[]`	**`items`** the Player's array of items
`String`	**`location`** the Player's current location

Interface Summary:

Type	Method and Description
`void`	**`addItem`**`(String item)` Adds item to Player.items

ActionMenu

A class used to display menu of possible actions to the player, then execute the player's choice.

Field Summary:

Type	Field and Description
`String[]`	**`theArray`** The array of possible actions

Interface Summary:

Type	Method and Description
`void`	**`refresh`**`()`

	Finds all possible Player actions (by using information about Player, World, etc.) and fills the theArray with them. Any previous data in theArray is erased.
`String`	**`toString`**`()` Returns a String representation of all entries in theArray. Used for printing to screen.
`void`	**`execute`**`(int userSelect)` Executes the chosen action corresponding to theArray[userSelect]

Built-in Classes

- The essential building blocks for any game
- To be used by the user

Room

Each instance of **Room** represents a location in **World**.

Field Summary:

Type	Field and Description
`String`	**`name`** The name of the Room
`String[]`	**`items`** The keys (names) to the actual items
`String[]`	**`NPCs`** The keys (names) to the actual NPCs
`String[]`	**`adjList`** The keys (names) to the adjacent Rooms

Interface Summary:

Type	Method and Description
`void`	**`connectTo`**`(String targetRoom)` Adds targetRoom to this Room's adjList, and this Room to targetRoom's adjList
`void`	**`addNPCFoe`**`(String name, int hp, int atkDmg)` Creates an NPC foe in a room with the specified HP and attack damage
`void`	**`addNPCFriend`**`(int powerup)` Creates a friendly NPC in a room with the specified powerup
`void`	**`addItem`**`(String name, String statModified, int powerup)` Creates an item that can be picked up

NPC

Non-playable Characters

Field Summary:

Type	Field and Description
`String`	**`name`** The name of the NPC.
`boolean`	**`friendly`** TRUE if NPC is a friend, FALSE if NPC is a foe
`int`	**`health`** the NPC's health stat
`int`	**`attack`** the NPC's attack stat
`String`	**`location`** the NPC's location

Item

Can be picked up by the player to modify the player's statistics

Field Summary:

Type	Field and Description
`String`	**`name`** The name of the Item.
`int`	**`value`** The magnitude of stat change that the Item causes.
`String`	**`statModified`** The stat that the Item modifies.

```
sampleGame1.tbag

setupWorld() {
	Room home = new Room();	// under the hood, a room is
				// created and added to RoomTable
	home.name = “Home”;
	dungeon = new Room();
	dungeon.name = “Dungeon”;
	final = new Room();
	final.name(“Final”);
	home.connectTo(“dungeon”);
	dungeon.connectTo(“final”);
	dungeon.addNPCFoe(“Minotaur”, 100, 10);
}

setupItems() {
	Item wand = new Item();
	wand.name = “Wand”;
	wand.value = 40;
	wand.statModified = “attack”
	Item sword = new Item();
	wand.name = “Sword”;
	wand.value = 40;
	wand.statModified = “attack”
}

setupPlayer() {
	int userClass = -1;
	while(userClass != 1 && userClass != 2){
		tprint( “Please enter 1 for Mage or 2 for Warrior class:\n” );
		userClass = in.next();
	}
	if (userClass == 1) {
		Player.health = 80;
		Player.attack = 20;
		Player.addItem(“Wand”);
	} elif (userClass == 2) {
		Player.health = 100;
		Player.attack = 10;
		Player.addItem(“Sword”);
	}
	Player.location = “Home”;
}

/* main to run the game */
main {
	setupWorld();
	setupPlayer();
	int userSelect = -1;
```

```
	/* Play until either the player or minotaur dies */
	while (Player.health > 0 && minotaur.health > 0) {
		actionMenu.refresh;
		tprint(Player.location);
		userSelect = in.next();
		actionMenu.execute(userSelect);  // execute the user's choice
	}
	if(Player.health <= 0) {
		tprint("You died. Game over.");
	}
	else {
		tprint("You won!");
	}
}
```