Travis Langston
10/5/2016
CS 3 Period 2

Project Proposal
Zork

Create a text adventure game called Zork with the following features:

- An inventory system
- Objects to interact with (sword, food), items that can be used by the player. Have a superclass for item, than a subclass for each type of item.
- A world the game takes place in with buildings, forests, etc. Add different rooms to the game.
- Other creatures, like goblins, birds, a dragon. Have a class for creatures, then make objects of it. Have field for health.
- Player class and object, with a field for health, and collections to store the items in the inventory.
- Container objects, like a bottle or chest
- Trigger objects, like a key which is used to unlock a door.
- Game is controlled by typing in commands. (Go north, open door, hit troll with axe). Add new commands to the game.
- An endgame (conquering a dungeon and winning the treasure inside it)

Classes:
<u>RunGame:</u> creates an object of the game class and starts the game.
<u>Game:</u> contains the player object, and provides the structure for running the game. Creates room objects to create the map of the game. Uses method play() and processCommand() to allow the player to enter input commands using class parser, creates command objects, and then carries out operations based upon those commands. The game class has the methods goRoom(), playSound(), chooseTake(), takeItem(), chooseEat, eatItem(), open(), combat(), and endGame().
<u>Room:</u> Blueprint for creating room objects to create the map of the game. Each object can have a closing object, in case the room is blocked by a door, an HashMap inventory to store Item objects in the room, and creature objects for the player to fight in the room.
<u>Creature:</u> Provides blueprint for creating creature objects, used for creatures to fight in rooms and for the player. Has fields for health, power, maxHealth, maxPower, level, a HashMap inventory for storing item objects, and a class (warrior, archer, wizard) which determines which weapons show up in the game and what skills the creature can use.
<u>Parser:</u> Used to process commands by using stringTokenizer to split each command into two words so the command can be used by the game class.
<u>CommandWords:</u> Has a list of all valid commands, to help the game class know how to use them.

Command: blueprint for creating command objects, which are then used by the game class.
Item: Abstract class with methods for the items used by the player in the game, and stored in the rooms.
Weapon: Blueprint for creating weapon objects. Subclass of Item. Creates skill objects and has different skills depending on what type of weapon it is (sword, shield, etc.). These skills are called by the player during combat. Each skill has a low and high damage, which a value between is randomly chosen. Each weapon has a field for level, which is used to affect how much damage is done by the weapon and how much power is consumed - the higher the level, the higher the damage, and the more power consumed.
Skill: Blueprint for allowing weapon objects to create skill objects. Has fields for low and high damage, which a value between is randomly chosen, power consumed, the probability of missing, and how much health the skill can heal.
Food: Blueprint for creating food objects. Subclass of Item. Each object has fields for the health and power the object can restore. Food objects are found stored in rooms in the game.
Container: Blueprint for creating container objects, which are stored in rooms. Subclass of Item. Has a Stack inventory for item objects, which the player can retrieve from the container in the game.
Closing: Blueprint for creating closing objects, like doors and gates. Each are part of a room, and can be locked, forcing the player to find a specific trigger object to open it.
Trigger: Blueprint for trigger objects. Subclass of Item. Used by the player to open locked closing objects.

Paragraph: The game starts when the player calls the main method in the PlayGame class, which creates an object of Game. Game's constructor creates a Player object, and makes the player enter their name and choose a class. Their choice determines which weapons are available in the game and how much health they have. Game then calls its createRooms method, which creates the room objects the game will be played in, sets their exits, creates the Item objects in the rooms, creates the Creature objects in the rooms, and sets the current room in the jail. The main method of PlayGame then calls the play method in game, which uses a while loop and a Parser object to allow the player to enter commands to control the game. Using the processCommand method, the game executes certain actions based on the player's command, which include moving to a different room, taking an item from a room, opening a Container object in the room, which uses a Stack to store items in a container like a chest, eating food the player has which uses a Food object to restore the player's health or power, or quitting. When the player moves into a room where there is a creature, the combat method is called. The player chooses which weapons to use, and then the player and creature's health and a list of available skills are printed out, which are based upon the skills each weapon has, which are based upon the type of weapon. The player can choose to use these skills or eat some food to restore health and power. Each skill has a chance of missing. After the player attacks, the creature then attacks, by choosing a random skill from its weapons. Depending on the player's weapons, they may block the attack. This continues until either the player or the monster are dead. As the player progresses through the game, they will find Weapon objects with higher levels, which have Skill objects that do more damage and consume more power. They may also encounter a locked door to a room, created by a Closing object in the room, and need a specific trigger object to open it. Upon encountering the door, the player will be prompted to choose an item to open it. If this is the correct trigger object, then the door opens and the player can enter. Traveling to certain rooms will also play sound from a WAV file, and entering combat will play sounds from a Queue of WAV files.