Shay Snyder

Honors 1260 – Spring 2020

**Zork Design Document**

Problem: 1260 Honors Project 5

**List of Inputs, Outputs, and Processing Required**

Inputs

* Allow the user to control the game by entering various commands
  + - * “go north” – move the player upward
      * “go south” – move the player downward
      * “go east” – move the player to the right
      * “go west” – move the player to the left
    - The user will be able to advance and return to randomly generated cells
    - The user will be able to fight randomly generated monsters
    - The user will be able to escape the dungeon

Outputs

* A visually appealing user interface that display various aspects of the game
* A map of the entire Dungeon
* A display of each room including monsters and weapons
* The player’s health and progress
* A detailed prompt of each fight
* A success screen when the user exits the dungeon

Processing

* Randomly generate a two-dimensional dungeon with each cell potentially containing a monster and weapon
* Use the random class to determine the outcome of a fight between the player and the monster
* Use regular expressions to validate user input
* Determine if the user’s desired move is possible
* Move the user to the next room
* Move the user to the last room
* Decrease the player’s and monster’s health as each fight progresses
* Account the variance of weapons and monsters in each fight
* Determine if the player is still alive
* Allow the player to attack and harm the monster
* Allow the monster to attack and harm the player

**Identification of Classes and Their Responsibilities**

**Class name: ZorkDriver**

Requirements

* Drive the Zork game
* Control all input and output

**Class name: Zork**

Requirements

* Manage all other dependencies of the Zork game
* Hold the global random object
* Display the current status of the game
* Be an intermediary for the user’s desired move

**Class name: Dungeon**

Requirements

* Represent the game’s Dungeon by holding all cells
* Display the dungeon
* Manage the user’s desired move
* Determine the outcome of all fights
* Manage the game’s progression

**Class name: Cell**

Requirements

* Represent each individual cell
* Display the cell’s contents
* Hold a weapon for player pickup
* Determine if a given move is available
* Hold the cell’s monster
* Determine if a cell contains a monster and if so, return said monster

**Class name: Monster**

Requirements

* Represent a monster
* Hold the monster’s health
* Hold the monster’s type
* Hold the monster’s damage capability
* Manage attack by the monster
* Determine if monster is alive
* Manage attacks to the monster

**Class name: Skeleton**

Requirements

* Extend the Monster class
* Represent a skeleton
* Set the skeleton’s damage

**Class name: Zombie**

Requirements

* Extend the Monster class
* Represent a zombie
* Set the zombie’s damage

**Class name: Creeper**

Requirements

* Extend the Monster class
* Represent a creeper
* Set the creeper’s damage

**Class name: Player**

Requirements

* Represent a player
* Hold the player’s health
* Hold the player’s type
* Hold the player’s damage capability
* Manage attack by the player
* Determine if player is alive
* Manage attacks to the player

**Class name: Weapon**

Requirements

* Represent a weapon
* Hold the weapon’s damage
* Hold the weapon’s type
* Manage the weapon’s attack
* Set the weapon’s damage to 5

**Class name: Hand**

Requirements

* Extend the Weapon class
* Represent a hand
* Set the weapon’s damage to 3

**Class name: Stick**

Requirements

* Extend the Weapon class
* Represent a stick
* Set the weapon’s damage to 4

**Class name: Sword**

Requirements

* Extend the Weapon class
* Represent a sword
* Set the weapon’s damage to 6

**Class name: Knife**

Requirements

* Extend the Weapon class
* Represent a knife
* Set the weapon’s damage to 5

**Class name: Laser**

Requirements

* Extend the Weapon class
* Represent a laser
* Set the weapon’s damage to 8

**Class name: Gun**

Requirements

* Extend the Weapon class
* Represent a gun
* Set the weapon’s damage to 7

**Class name: Participant**

Requirements

* Represent the basis of all game participants
* Create the methods and attributes that apply to all participants
* Set the weapon’s damage to 7

**UML Class Diagram**

A close up of a map

Description automatically generated

**Algorithms**

**Class:** ZorkDriver

**Method:** main(args : String[]) : void

**Description:** The main method in the ZorkDriver class serves as the backbone for the rest of the program. From this point, the intro screen, main game, and outro screen will be called

**Class:** ZorkDriver

**Method:** intro() : void

**Description:** This method will use JOptionPane to introduce the game to the user.

**Class:** ZorkDriver

**Method:** startgame() : void

**Description:** This method will initialize the actual game and control all interaction between the game and the user via JOptionPane. The user will have four commands to control the game: “go east”, “go west”. These commands will be used to move the player throughout the dungeon in search for the exit. If a user enters a cell and a monster lies within, a prompt will be displayed detailing various aspects of the fight. If the user enters an invalid command, a prompt will be displayed informing them of the error. Once the user reaches the end of the dungeon, a prompt will be displayed congratulating them on their success.

**Class:** ZorkDriver

**Method:** outro()

**Description:** This method will thank the user for playing the game and wish them a good day.

**Class:** ZorkDriver

**Method:** confirmQuit()

**Description:** This method will use JOptionPane to ask the user if they are sure they would like to quit the game. If they select ‘yes’, the game will terminate; otherwise, the game will continue.

**Class:** ZorkDriver

**Method:** confirmNewGame()

**Description:** This method will use JOptionPane to ask the user if they are sure they would like to start a new game. If they select ‘yes’, a new game will be generated; otherwise, the game will continue.

**Class:** ZorkDriver

**Method:** submitMove()

**Description:** This method will relay the user’s desired move to the game itself. In response, the game will return a String detailing the resulting events of said move. JOptionPane will be used to display said information

**Class:** Zork

**Method:** Zork()

**Description:** This is arg-constructor for the Zork class. Its purpose is to initialize the game by calling the Dungeon() no-arg constructor to generate the dungeon.

**Class:** Zork

**Method:** display : String

**Description:** this method will be used to inform the user of the game’s current status. This method will return the most relevant data to the user. For example, the string could contain the map of the dungeon.

**Class:** Zork

**Method:** move(move : String)

**Description:**  This method will be called whenever the player decides to make a move. If they make an invalid move, the string will contain “INVALID MOVE! Please try again.” If they simply move to a new cell, the string will contain “SUCCESS! You have advanced to the next cell.” If they move to a previous cell, the string will contain “WARNING! You have returned to the previous cell.” If the user enters a room with a weapon, the string will contain “CONGRATS! You have advanced to the next room and found a new <some weapon>.” If the user enters a room with a monster, the string will contain “You have advanced to the next room and awaken the beast!” and then display various aspects of the fight. If the user exits the dungeon, the string will contain “CONGRATULATIONS! YOU HAVE SUCCESSFULLY ESCAPED”.

**Class:** Zork

**Method:** isOver()

**Description:** This method will return a boolean attribute that represents the completion status of the game.

**Class:** Dungeon

**Method:** Dungeon()

**Description:** This method will randomly generate a 1D dungeon. The first cell will always be in the left-uppermost cell. This algorithm will repeat until the desired number of cells are generated.  One of the middle cells will be randomly chosen to generate a weapon. Here, the user can pickup said weapon to increase their fighting power throughout the rest of the game. Furthermore, each inner cell will also have a 50% chance of generating a monster.

**Class:** Dungeon

**Method:** showDungeon() : String

**Description:** This method will return a string that contains the layout of the dungeon along with details of each room, the player’s health, the player’s current weapon, and the current cell.

**Class:** Dungeon

**Method:** move(move : String) : String

**Description:** This method will handle the user’s desired movement. The method will determine if the desired move is valid. If said move is valid, the method will continue; If said move is not valid, the method returns a String containing the error status. If the user decides to return to the previous room, the returnToPreviousCell() method will be called. If the user decided to advance to the next room, the advanceToNextCell() method will be called. If the player advanced to the next room and they reach the final cell, the gameOver Boolean attribute will be toggled to true and a String congratulating the user will be returned.

**Class:** Dungeon

**Method:** isGameOver()

**Description:** return the gameOver Boolean attribute

**Class:** Dungeon

**Method:** fight(monster : Monster) : String

**Description:** This method is responsible for calculating the result of each encounter between the player and monster. The player has to kill the monster in order to progress through the dungeon.  In every round of the fight, the player will attack first. The player has a 10% chance of missing the match. Every time the player lands a hit, the appropriate amount of damage will be taken from the monster’s health. There is a 20% chance that the monster will miss the player. Every time the monster lands a hit, the player’s health will be reduced by 4 points. This process will be repeated until someone dies. If the player wins, they will be able to advance. If the monster wins, the game is over. The returned string will detail every round of the fight and who won.

**Class:** Dungeon

**Method:** advanceToNextRoom() : void

**Description:** Before doing any exchanging occurs, this method will make sure the player isn’t in the last room. If the user happens to be in the last room, the method will do nothing. Assuming the player isn’t in the last cell, the method will do the following…  The old cell (current cell) and the next cell (current cell + 1) will be extracted from the ArrayList. The play will be extracted from the previous cell and added to the new cell. The player will be removed from the old cell. Finally, the currentCell integer attribute will be incremented by one.

**Class:** Dungeon

**Method:** returnToPreviousRoom() : void

**Description:** Th Before doing any exchanging occurs, this method will make sure the player isn’t in the fisr room. If the user happens to be in the first room, the method will do nothing. Assuming the player isn’t in the first cell, the method will do the following…  The old cell (current cell) and the new cell (current cell - 1) will be extracted from the ArrayList. The play will be extracted from the previous cell and added to the new cell. The player will be removed from the old cell. Finally, the currentCell integer attribute will be decremented by one.

**Class:** Dungeon

**Method:** equipPlayer()

**Description:** This method will check if the new cell has a weapon. If the cell doesn’t have a weapon, the method will do nothing but return and empty String. If the cell does have a weapon, the player will be equipped with said weapon and the method will return a String specifying the type of weapon that was found.

**Class:** Cell

**Method:** Cell()

**Description:** This is the main constructor for the Cell class. The argued cell type will be used to specify if the cell is the starting point, exit, or a normal cell. The argued player will be used to add the player of Zork to a given cell. If the player isn’t currently in a cell, the player object will be set to null. The argued Random object will be used for weapon and monster generation. The generateWeapon Boolean attribute is used to determine if a weapon should be generated in a given cell. The generateBoolean attribute is used to determine if a weapon should be generated in a given cell.

**Class:** Cell

**Method:** showCell() : String

**Description:** This method will generate a string that contains a visual representation of a given cell and the items within. The layout of the String will be as such: |\_\_( 1 )\_\_( 2 )\_\_(3)\_\_|. The (1) zone will be used to show the location of the player: “\_P” if the player is in the cell and “\_\_” if the player isn’t in the cell. The (2) zone will be used to display the presence of a weapon within a given cell: “\_\_” if no weapon is in the cell, “St” if a Stick is in the cell, “Kn” if a Knife is in the cell, “Sw” if a Sword is in the cell, “Gu” if a gun is in the cell, and “La” if a Laser is in the cell. The (3) zone will be used to display the presence of a monster: “\_\_” if no monster is in the cell, “Zo” if a Zombie is in the cell, “Cr” if a Creeper is in the cell, and “Sk” if a Skeleton is in the room.

**Class:** Cell

**Method:** getPlayer()

**Description:** This method will return the player within a given cell. It will return null if no player is in the cell.

**Class:** Cell

**Method:** setPlayer()

**Description:** This method is used to add and remove the player from any given cell

**Class:** Cell

**Method:** getMonster() : Monster

**Description:** This method returns a Monster object containing the monster within a given cell.

**Class:** Cell

**Method:** isMoveAvailable(move : String)

**Description:** This method returns a Boolean value specifying the validity of the user’s desired move.

**Class:** Cell

**Method:** setMonster()

**Description:** This method is used to add and remove monsters from a given cell.

**Class:** Cell

**Method:** getWeapon()

**Description:** This method returns the weapon object contained within a given cell. This method will return null

**Class:** Cell

**Method:** removeWeapon()

**Description:** The method removes a cell’s weapon by setting its value to null

**Class:** Cell

**Method:** createWeapon()

**Description:** This method uses a random number generate to determine which type of weapon will be added to a given cell. Once the random number is generated, the weapon at said number will be added.

**Class:** Cell

**Method:** createMonster()

**Description:** **Description:** This method uses a random number generate to determine which type of monster will be added to a given cell. Once the random number is generated, the monster at said number will be added.

**Class:** Cell

**Method:** showMonster()

**Description:** This method takes the cell’s monster object and converts it to a String that will be used during visualization. The following values with be returned: “\_\_” when no monster is in the cell, “Cr” when a Creeper is in the cell, “Sk” when a Skeleton is in the cell, and “Zo” when a Zombie is in the cell

**Class:** Cell

**Method:** showPlayer()

**Description:** This method takes the cell’s player object and converts it to a String that will be used during visualization. The following values will be returned: “\_\_” when the player is not present, and “\_P” when the player is present

**Class:** Cell

**Method:** showWeapon()

**Description:** This method take’s the cell’s weapon object and converts it to a String that will be used during visualization. The following values will be returned: “\_\_” when the cell doesn’t have a weapon, “St” when the cell has a Stick, “Kn” when the cell has a Knife, “Sw” when the cell has a sword, “Gu” when the cell contains a Gun, and “La” when the cell contains a laser.

**Class:** Skelton

**Method:** Skelton()

**Description:** This constructor is used to initialize the Skeleton class. Since this class extends the Monster class, the constructor sets the Skeleton’s damage to 4.

**Class:** Zombie

**Method:** Zombie()

**Description:** This constructor is used to initialize the Zombie class. Since this class extends the Monster class, the constructor sets the Zombie’s damage to 5.

**Class:** Creeper

**Method:** Creeper()

**Description:** This constructor is used to initialize the Creeper class. Since this class extends the Monster class, the constructor sets the Creeper’s damage to 6.

**Class:** Monster

**Method:** Monster()

**Description:** This constructor is used to initialize the Monster class. Since this class extends the Participant class, the constructor sets the Monster’s damage to 3.

**Class:** Monster

**Method:** attack()

**Description:** This method returns the integer value representing the monster’s damage. This value is used to reduce the health of players.

**Class:** Participant

**Method:** getHealth()

**Description:** return the participant’s health attribute as an integer

**Class:** Participant

**Method:** isAlive()

**Description:** return a Boolean attribute that represents the living status of said participant; if their health is greater than 0, return true; otherwise, return false

**Class:** Participant

**Method:** decreaseHealth()

**Description:** decrease the participant’s health by the argued integer value

**Class:** Participant

**Method:** attack()

**Description:** abstract method that could be defined in subsequent classes

**Class:** Player

**Method:** Player()

**Description:** This is the default constructor for the Player class. The player’s health attribute is set to 100 and the weapon attributed is set to hand.

**Class:** Player

**Method:** getWeaponTitle()

**Description:** return the title of the player’s current weapon

**Class:** Player

**Method:** attack() : int

**Description:** This method returns an integer value that represents the damage dealt by a player’s attack.

**Class:** Weapon

**Method:** Weapon()

**Description:** This is the default constructor for the Weapon class. The damage attribute is set to 5. The type attribute is set to “General”.

**Class:** Weapon

**Method:** attack()

**Description:** This method returns the  integer value representing the weapon’s damage.

**Class:** Weapon

**Method:** getType()

**Description:** return the String attribute that represents the weapon’s type

**Class:** Hand

**Method:** Hand()

**Description:** This is the default constructor for the Hand class which extends the Weapon class. The damage attribute is set to 3. The type attribute is set to “Hand”.

**Class:** Stick

**Method:** Stick()

**Description:** This is the default constructor for the Stick class which extends the Weapon class. The damage attribute is set to 4. The type attribute is set to “Stick”.

**Class:** Sword

**Method:** Sword()

**Description:** This is the default constructor for the Sword class which extends the Weapon class. The damage attribute is set to 6. The type attribute is set to “Sword”.

**Class:** Knife

**Method:** Knife()

**Description:** This is the default constructor for the Knife class which extends the Weapon class. The damage attribute is set to 5. The type attribute is set to “Knife”.

**Class:** Laser

**Method:** Laser()

**Description:** This is the default constructor for the Laser class which extends the Weapon class. The damage attribute is set to 8. The type attribute is set to “Laser”.

**Class:** Gun

**Method:** Gun()

**Description:** This is the default constructor for the Gun class which extends the Weapon class. The damage is set to 7. The type attribute is set to “Gun”.

**Test Cases**

These are the test cases that were used to test our project. The various inputs and outputs are detailed below.

**Test Case #1**

 Input: the user starts the game

Output: the into screen should be displayed

**Test Case #2**

Input: the user types “GO EAST”, “GO NORTH”, ”GO WEST”, “GO SOUTH”

Output: attempt move the player to the right, north, west, or south

**Test Case #3**

Input: the user types “go east” , “go north”, “go west”, “go south”

Output: attempt to move the player to the right, north, west, or south

**Test Case #4**

Input: the user types “east”, “north”, “west”, “south”

Output: attempt to move the player to the right, north, west, or south

**Test Case #5**

Input: the user types anything but the aforementioned inputs

Output: display a prompt informing the user of the error

**Test Case #6**

Input: The user exits the dungeon

Output: Congratulate the player for completing the game

**Test Case #7**

Input: The user prematurely terminates the program

Output: Ask the user if they are sure they would like to exit the program.

**Test Case #8**

Input: The user dies during a fight

Output: Ask the user if they would like to play again.

**Test Case #9**

Input: The user kills the monster

Output: The details of the fight will be displayed to the user and the game will proceed.