Aiman Bickiya

UCLA ID: 105339123

1. Recursive Goblin move function- My function is not recursive and it operates with the same algorithm as Snakewoman and Boogeyman’s move functions. It is located in the Monster.cpp file

2/3. Design Description and Non-trivial algorithms

* Actor class:
  + - This is the base class for the Monster and Player classes, which derive a list of getter and setter functions mainly related to stats (Ex. getHP() and setDex() ). There is also a getter that returns a pointer to the actor’s weapon and a getter that returns a string of the actors name
    - Player
      * + Players are a derived class of Actor, and are initialized at construction with a position, a pointer to a dungeon object, as well as the default stats and weapons. The Player class’s responsibilities include setting the true/false conditions for the game to end (either dying or picking up the idol), as well as moving the player, displaying inventory, wielding weapons and reading scrolls. The non trivial algorithms are listed below:

wieldWeapon(char item)

This function takes in a char which represents the index of the player’s inventory

if index is ‘a’ {

*//pointing to already wielded weapon*

*Construct result string and push to dungeon class (to be printed after the turn is over*

*Exit function*

*Else, convert character to number using helper function*

*if it returns an invalid result, do nothing and exit*

*change pointer type from Object to Weapon using dynamic cast*

*If the newly created pointer is equal to null,*

*construct an error string and push it to the dungeon class*

*exit the function*

*else {*

*swap pointers of current weapon and selected weapon*

*Construct results string and push to dungeon class*

*exit function*

readScroll(char item)

This function takes in char which represents the index of the player’s inventory

if index is ‘a’ {

*//pointing to first slot which is always a weapon*

*Construct error string and push to dungeon class (to be printed after the turn is over*

*Exit function*

*Else, convert character to number using helper function*

*if it returns an invalid result, do nothing and exit*

*change pointer type from Object to Scroll using dynamic cast*

*If the newly created pointer is equal to null,*

*construct an error string and push it to the dungeon class*

*exit the function*

*else {*

*Call the effect function of the scroll*

*store the scroll pointer in a temp variable*

*erase the pointer from the player’s inventory*

*call delete on the scroll*

*exit*

move(int r, int c)

This function takes a set of coordinates which represent the spot the player wishes to move to.

*if player is asleep, do nothing and exit*

*if given coordinates is a valid spot (not a wall and not occupied by another actor), change player coordinates to given set*

*if given coordinates are occupied by a monster, call combat function (part of dungeon class)*

*use dungeon class function “findMonster” to locate the correct monster to fight*

grabItem()

This function is used to pick up an item from the floor of a dungeon and add them to the players inventory

*call function of dungeon class to check if item exists at player’s coordinates, it returns a pointer to the object (or nullptr if there’s nothing there)*

*if no item is found (nullptr)*

*do nothing and exit*

*if item is found and its the staircase*

*call the dungeon’s reset function and exit*

*if item is found and its the golden idol*

*set idolObtained to true and exit*

*if inventory is full (count is 26)*

*push result string to dungeon*

*exit*

*else add pointer the vector of player’s inventory*

*increment inventory counter*

*construct retrieval string and push to dungeon class*

*exit*

Constructor: initializes stats, position, dungeon pointer, and creates default weapon

Destructor: deletes equipped weapon, and deletes inventory

* + Monster
    - * Monster is a derived class of Actor, but Monster objects are never initialized because it is an abstract class. It provides the move() and drop() functions for inheritance and a getter for the monster’s display character
      * Snakewoman

An inherited class of monster. The constructor initializes Snakewomen with stats, a position, a dungeon pointer, display character, and creates a new weapon (Magic fangs of sleep) which is eventually deleted in the destructor. The Snakewoman class has the responsibilities of moving the Snakewoman and controlling potential item drops after a Snakewoman dies

move: the move function for Snakewoman first checks if the monster is asleep. If they are, it exits the function. Next, it checks if the player is adjacent, and if so, it calls the dungeon’s combat function with itself and the player as arguments and then exits the move function. If neither of those conditions are fulfilled, the function checks in a 6 x 6 box centered on the Snakewoman (3 spaces in each direction) to see if the players coordinates are contained in that box. If they aren’t, do nothing and exit the function, otherwise try to close the gap by moving one row or column closer. If the path is obstructed, do nothing and exit

Drop: the drop function first calls a function from the dungeon class with the monster’s coordinates to check if an item already exists at that spot. If there is already something there, exit the function. Otherwise, attempt the 1 in 3 chance. If it returns true, create a new Magic fangs of sleep on the monsters current coordinates.

* + - * Goblin

An inherited class of monster. The constructor initializes Goblin with stats, a position, a dungeon pointer, display character, and creates a new weapon (Short sword) which is eventually deleted in the destructor. The Goblin class has the responsibilities of moving the Goblin and controlling potential item drops after a Goblin dies.

Move: the move function operates the same way as Snakewoman but checks for 15 spaces in each direction instead of 3.

Drop: The same check for existing items is carried out as described in Snakewoman::drop, and if the spot is empty, the function attempts the 1 in 3 chance to drop an item. If it succeeds, it picks a random number between 1 and 2. For a 1, the dropped item will be magic fangs of sleep, and for 2 the dropped item will be a magic axe

* + - * Bogeyman

An inherited class of monster. The constructor initializes Bogeyman with stats, a position, a dungeon pointer, display character, and creates a new weapon (Short sword) which is eventually deleted in the destructor. The Bogeyman class has the responsibilities of moving the Bogeyman and controlling potential item drops after a Bogeyman dies.

Move: Operates as described in Snakewoman, but with a distance of 5 in each direction instead of 3.

Drop: Operates in the same way as Snakewoman::drop but with a 1 in 10 chance (instead of 1 in 3) and with the dropped item being a magic axe (instead of magic fangs of sleep)

* + - * Dragon

An inherited class of monster. The constructor initializes Dragon with stats, a position, a dungeon pointer, display character, and creates a new weapon (Long sword) which is eventually deleted in the destructor. The Dragon class has the responsibilities of moving the Dragon and controlling potential item drops after a Dragon dies.

Move: The dragon does not approach the player, so while it still borrows the sleep time check and adjacent player check from the Snakewoman function, it doesn’t search for the player across a set distance. Unlike the other three monsters however, the dragon checks every turn if its HP is below the max, and if it is, it has a 1 in 10 chance to recover 1 HP.

Drop: Like the other monsters, firstly there is a check to see if there is already an item on the space the dragon is occupying. If there’s nothing, the dragon has a 100% chance to drop a type of scroll. To determine the type of scroll, the function selects a random number from 1 to 5, with each number corresponding to a type of scroll (Teleportation, Enhance HP, Enhance Strength, Enhance Dexterity, Enhance Armor). It creates a new scroll of the determined type with the monster’s coordinates and pushes it to the dungeon.

* Objects
  + - Objects is a base class from which all weapons, scrolls, stairways and the golden idol are derived. It provides getters for the row and column, the display character of the object, the objects name, and if applicable, the string associated with picking it up.
      * Weapons

Weapons is a derived class of Object. It provides getters for the dexterity bonus and damage bonus of weapons as well as each weapon’s associated action string.

Mace

The Mace class has two constructors, one of which takes no arguments, and the other which takes two ints (for a row and column) as well as a pointer to a dungeon object. The constructor always initializes the weapon’s name, dexterity bonus, damage bonus, action string and display character. The constructor with no arguments exists so that it can be initialized as a weapon for a monster (although no monsters natively wield maces). The constructor that takes coordinates and the dungeon pointer is used when a Mace is initialized as a dropped item in a dungeon, so that the coordinates can be used when displaying the item on the map.

ShortSword

ShortSword has two constructors identical to the ones described in the Mace class. The constructor without arguments is used for both the player as well as goblin and bogeyman enemies.

LongSword

LongSword has constructors identical to Mace and ShortSword. The constructor without arguments is used when creating Dragon enemies, and the one with the additional parameters is used for creating dropped items.

Fang

Identical to above.

MagicAxe

Identical to the others, however the constructor without arguments isn’t called because no monsters natively wield magic axes.

* + - * + Scroll

Scroll is an abstract class which is derived from Object and which is the base class for the 5 different types of scrolls that are available. The Scroll class provides a getter for the scrolls’ flavor text (Ex. “Your armor glows blue”). It also has the declaration for the effect function, which takes a player pointer for its argument, but it is a pure virtual function so there is no implementation.

Teleport

The constructor for this scroll as well as the others takes in two ints and a pointer to a dungeon object. It initializes the name, flavor text string, display icon, retrieval string as well as the row, column and dungeon pointer.

Effect: Uses randInt to determine a random row and column and makes sure that its not already occupied by another actor or a wall, and then calls Player::move with those coordinates as the argument

EnhanceStr

See Teleport for constructor details

Effect: simply calls the setter for player’s strength with a random integer from 1 to 3

EnhanceDex

See Teleport for constructor details

Effect: Calls the setter for player’s dexterity with an argument of 1

EnhanceArmor

See Teleport for constructor details

Effect: Calls the setter for player’s armor with a random integer from 1 to

EnhanceHP

See Teleport for constructor details

Effect: simply calls the setter for player’s max HP with a random integer from 3 to 8

* + - * + Stairway

The constructor initializes the row, column and display icon as well as the dungeon pointer.

* + - * + Idol

The constructor initializes the row, column, display icon, name, retrieval string, and dungeon pointer

* Dungeon
  + The Dungeon class has several different tasks it carries out. Firstly is the task of initializing the dungeon, which involves generating rooms, corridors, monsters, and items. It also has the task of printing the display, and handling combat between attackers, as well as calling move and drop functions on monsters.
    - Constructor: sets player as nullptr, and creates initial rooms and columns
    - Destructor: iterates through vector and deletes each entry for monsters, dropped items, and rooms, and also deletes the player.
    - initializeMap: simple function that sets both the grid for display and the blank grid to be all ‘#’ (walls)
    - generateRoom: picks random coordinates for the center of a room, and determines a random width and height (constrained between 3 and 5). Returns false if the resulting room runs off the edges of the map or overlaps with another room. Otherwise, a new room is created and is ‘dug out’, with all the characters made into ‘ ’ in both the display grid and blank grid. The room is pushed to a vector of Room pointers.
    - addRoom: calls generateRoom to create between 4 and 6 rooms.
    - addCorridor: Takes a room pointer as an argument. Searches first in the east direction for an empty space starting from the rightmost column of the room. If a space is found (indicating another room), dig out that space (convert to ‘ ’) and exit. If a space is not found, search in the left direction. If that fails, then the north direction, and if that fails, the south
    - createDrops: Choses a random number between 2 and 3 for the amount of drops. Choses a random number between 1 and 7 with each number corresponding to a different object (3 types of weapons and 4 types of scrolls). Determine a random set of coordinates and make sure they’re empty. Initialize an object based on the number 1-7 and push it to the vector of Object pointers that represents the items on the dungeon floor. If the floor level is 3 or less, generate a staircase. If the floor level is 4, generate a golden idol (both with random coordinates). These are also pushed to the vector of items.
    - addMonsters: determines the total number of monsters for the floor with the provided formula. If the floor is 0 or 1, constrain p to be between 1 and 2. If the floor is 2, p is constrained between 1 and 3. If floor is 4, p is constrained between 1 and 4. Based on the value of p, create a new Snakewoman (1), Goblin (2), Bogeyman (3), or Dragon (4) with randomly determined coordinates. Push the new monster to a vector of monster pointers
    - Reset: iterates through vectors of monsters, items and rooms and calls delete on each entry, and also clears the vectors. Increments the floor value and calls initializeMap, addRoom, addCorridor, createDrops and addMonster to create an entirely new layout. Also calls move on player with random coordinates.
    - moveMonsters: iterate through the monster vector. For each entry, check if the monsters HP is less than or equal to 0. If it is, call that monster’s drop function, then erase from the vector and call delete. If HP is above 0, call that monster’s move function
    - combat: Takes two actor pointers as arguments. Calculates attacker points (dexterity + weapon’s dexterity bonus) and defender points (dexterity + armor). Push attacker and defender names and weapon’s action string to the turn results vector. Use randInt to determine if attacker hits defender. If hit, determine the damage dealt using attacker’s strength and weapon’s damage bonus and decrease defenders HP by that amount. If this causes defenders HP to reach 0 or lower, push a unique string to results. If the attacker is wielding magic fangs of sleep, attempt a 1 in 5 chance to put the defender to sleep for between 2 and 6 turns. If the defender is already asleep, only inflict the maximum of the new sleep value and the current sleep value. Push a unique results string. Otherwise if defender isn’t killed and is not put to sleep, push a generic result string. If attacker misses, push a result string indicating that. If player’s HP is 0 or lower, mark them as dead.
    - findMonster: Takes an actor pointer and a set of coordinates. Iterates through the monster vector to find one with matching coordinates. If one is found, call combat with the passed in Actor pointer as the attacker and the found monster as the defender. This function is called in the player class with the player representing the attacking actor.
    - findItem: takes in a set of coordinates and iterates through the items vector to determine if there is an item at that set of coordinates. If an item is found and it also matches the players coordinates, erase the item from the items vector and return that pointer. If an item with matching coordinates is found but it does not match the player’s coordinates, return that pointer without erasing it from the item vector. This function is called from the player class to pick up objects, which is why they are erased if the player is standing on them. However it is also called by monster drop functions to determine if there is already an item at that spot, in which case we would not want the item to be deleted.
    - display: Clears the screen, and copies each cell of blankGrid (only has walls and empty spaces) over to displayGrid. Iterate through the items vector, and for each entry, get the coordinates and copy the display character to that cell of displayGrid. Repeat the process for the monster vector, and add the player character. Print the grid. Assemble the statistics message and print it. Iterate through the vector of turn results and print the contents, then clear the vector for the next turn

4. Features not Implemented

* + - Recursive algorithm for goblins, as well optimal paths. They move in the same way as Snakewomen and Bogeymen, meaning they get blocked by objects instead of going around