CS4332 Introduction to Programming Video Games:

Assignment 1 - Hunt the Wumpus

This is an individual assignment. You are expected to work on this assignment by yourself, not with your project team member (s).

**Due Time:** 11:59pm, Sept. 23rd, 2016.

**Late Policy:**

* If you miss the due time above, the assignment will still be accepted until 11:59pm, Sept. 30th, 2016, with a 10% penalty per day.
* After 11:59pm, Sept. 30th, 2016, if not submitted, you will receive a grade of zero for Assignment #1.

The goal of this homework assignment is to develop proficiency with the C# programming language by creating a simple text-based game, Hunt the Wumpus. Since this game is text-based, it does not use any XNA capabilities, and hence is easier to debug.

In Hunt the Wumpus (<http://en.wikipedia.org/wiki/Hunt_the_Wumpus>) the player hunts a creature called the Wumpus in a dodecahedral dungeon. In each turn, the player can move or shoot a crooked arrow on a path containing up to 5 rooms. Hazards to the player include superbats (randomly move the player), pits (kill the player), and the Wumpus (kills the player once it awakes). The game gives clues to the location of hazards: if the player is in a neighboring room, the game says, "Bats nearby", "I feel a draft", and "I smell a Wumpus". Gameplay takes the form of exploration and deduction. You can play Hunt the Wumpus online at the site:

<http://www.ifiction.org/games/play.phpz?cat=&game=249&mode=html>.

Note that this game has a few more options (different map types) than the game described below.

**Game Rules:**

The game map has 20 rooms, each connected to three other rooms, with the topology of a dodecahedron.

Before the start of the game, one Wumpus, two groups of superbats, and two bottomless pits are randomly placed in the map.

On each turn, the game first moves the Wumpus (if awake), then checks neighboring rooms for hazards, to see if a warning message should be displayed. The player is then asked for their action, which can be move or shoot.

**Moving:**

For moving, the game displays a list of the room numbers of connected rooms, then asks for the room number of the destination room. If the entered room is not connected to the current room, the game displays, "Not possible" and re-asks for the destination room.

On a move, the game moves the player to the new room, then checks for whether the player has hit a hazard in that room.

**Hazards:**

A superbat attack causes the player to be randomly placed in a new room. When this happens, the game displays "Zap--Super Bat snatch! Elsewhereville for you!" This is the equivalent to a normal move, in that hazard checks are performed after determining the new room.

A bottomless pit kills the player, and ends the game. When this happens, the game displays, "YYYIIIIEEEE . . . fell in a pit"

The Wumpus begins the game asleep. The first time the player enters the room containing the Wumpus, the Wumpus wakes up, and the game displays, "... Ooops! Bumped a Wumpus" If the player shoots an arrow anywhere in the map, the Wumpus wakes up. After the Wumpus is awake, on each turn (including the turn he was awoken) he either (25% chance) stays in place or (75% chance) moves to a new adjacent room. If an awake Wumpus ends the turn in your room, the player dies.

**Warnings:**

If superbats are in an adjoining room, display the warning message, “Bats nearby”.

If a bottomless pit is in an adjoining room, display the warning message, "I feel a draft".

If the Wumpus is in an adjoining room, display the warning message, "I smell a Wumpus".

**Shooting**

If the player chooses to shoot, the game then asks for up to five rooms the arrow should visit. The game should not check that the player has entered a valid set of rooms during user input (the game does perform this check while the arrow is flying). If the player enters a path where the next room is not connected to the current room, the game picks a random connected next room. The game does, however, check to ensure the arrow does not go from room A to B and back to A again. If the player attempts to enter an A-B-A path, the game replies, "Arrows aren't that crooked - try another room" and the player must re-enter the next room.

Once the player has entered all rooms of the arrow's path, the game then checks each room in the path. If the arrow enters the room containing the Wumpus, the game displays, "Aha! You got the Wumpus!" (and the player wins the game). If the arrow enters the room containing the player, the game displays, "Ouch! Arrow got you!" If the arrow goes through its entire path and does not hit the Wumpus (or the player), display "Missed!"

The player only has 5 arrows, and loses one each time an arrow is shot. The game ends when the player runs out of arrows.

**Winning the game:**

When the player wins the game, display, "Hee hee hee - the Wumpus'll getcha next time!!"

The player is given the option to re-play the game using the same randomly chosen locations for the Wumpus, bats, and pits, or to have new random locations chosen for them.

In the event of questions concerning the behavior of the game, the final arbiter is the BASIC source code of the game. A description of the origins of the game, a gameplay session, and BASIC source code can be found in Gregory Yob’s Wumpus 1:

<http://www.atariarchives.org/morebasicgames/showpage.php?page=178>

**Submission Instructions:**

For this assignment, you must submit the source code in a ZIP/RAR file on eLearning.

**Evaluation:**

Assessment of the first assignment focuses on two areas: correctness (2/3) and design (1/3).

For correctness, assignments will be examined to ensure they fully implement the rules of the game. In particular, we will focus on whether arrow firing is handled correctly, whether the Wumpus logic is correct, and whether the game gives the player the choice to restart with hazards in the same place, or in new random locations. These three functionalities are the most complex aspects of the game.

For design, we will examine your source code to ensure it follows reasonable object-oriented design technique. In particular, we expect to see a class for the game map, a class for the player, and a class for the Wumpus. Classes and variables should have descriptive names. Classes for "nouns" in the game are expected to have class variables, as well as methods for manipulating these class variables. In addition, proper code documentation should be included (act as though you are working for a game development company where multiple teammates will be accessing your code).