Homework 5

Wumpus Console Design Document

Student:

Lillith Chute

Teacher:

Clark Freifeld

Course:

CS 5010

Design Summary

This application will use the dungeon builder from homework 3 as its base. That is to say, the dungeon will be configured and laid out in the same fashion. What will be different is that now the rooms will be built with the idea of being occupied by new items. The rules are as follows:

1. Exactly 1 cave has the Wumpus. The Wumpus doesn't move but can be smelled in adjacent caves.
2. There are n bottomless pits that provide a draft that can be felt in adjacent caves. this is done by supplying a percentage of caves with pits (a number between 0 and 1).
3. There are m caves with superbats. Entering a cave with superbats means that there is a 50% chance that the superbats will pick up the player and drop them in another place in the maze. Again, the user will provide a number between 0 and 1 for the amount of bats.
4. If a cave has both a bottomless pit and the superbats, the bats have a 50% chance of picking up the player before they fall into the bottomless pit.
5. A player can attempt to slay the Wumpus by specifying a direction and distance in which to shoot their crooked arrow. Distance is defined as the number of caves (but not tunnels) that an arrow travels. Arrows travel freely down tunnels (even crooked ones) but only travel a straight line through a cave.
6. Distance must be exact. For example, if you shoot an arrow 3 rooms to the east and the Wumpus was only 2 rooms to the east, you miss the Wumpus.
7. You win by killing the Wumpus.
8. You lose by being eaten, falling in a pit, or running out of arrows.

Code Design

1. This application uses the MVC design.
2. There is an interface for the controller, adventurer, room, and dungeon.
3. The model remains essentially the same as previous, with dungeon, room, and adventurer. These models are adjusted to include bats, pits, firing arrows, and the wumpus; removing the thief and gold.
4. The major difference is in the controller handling the functionality previously given to the driver program.

Test Plan

Do the basics:

1. Make sure the tests are not in the same package as the code.
2. Test that the constructors for the various classes work to included error checking and validations.
3. Testing getter and setter methods are generally straightforward.
4. Check code coverage to make sure as many lines of code have been checked as possible.
5. As a note, I test private methods by making them public if they are complex and then setting them back to private. There is probably a better way, but that's the quick and dirty way for me at the moment.

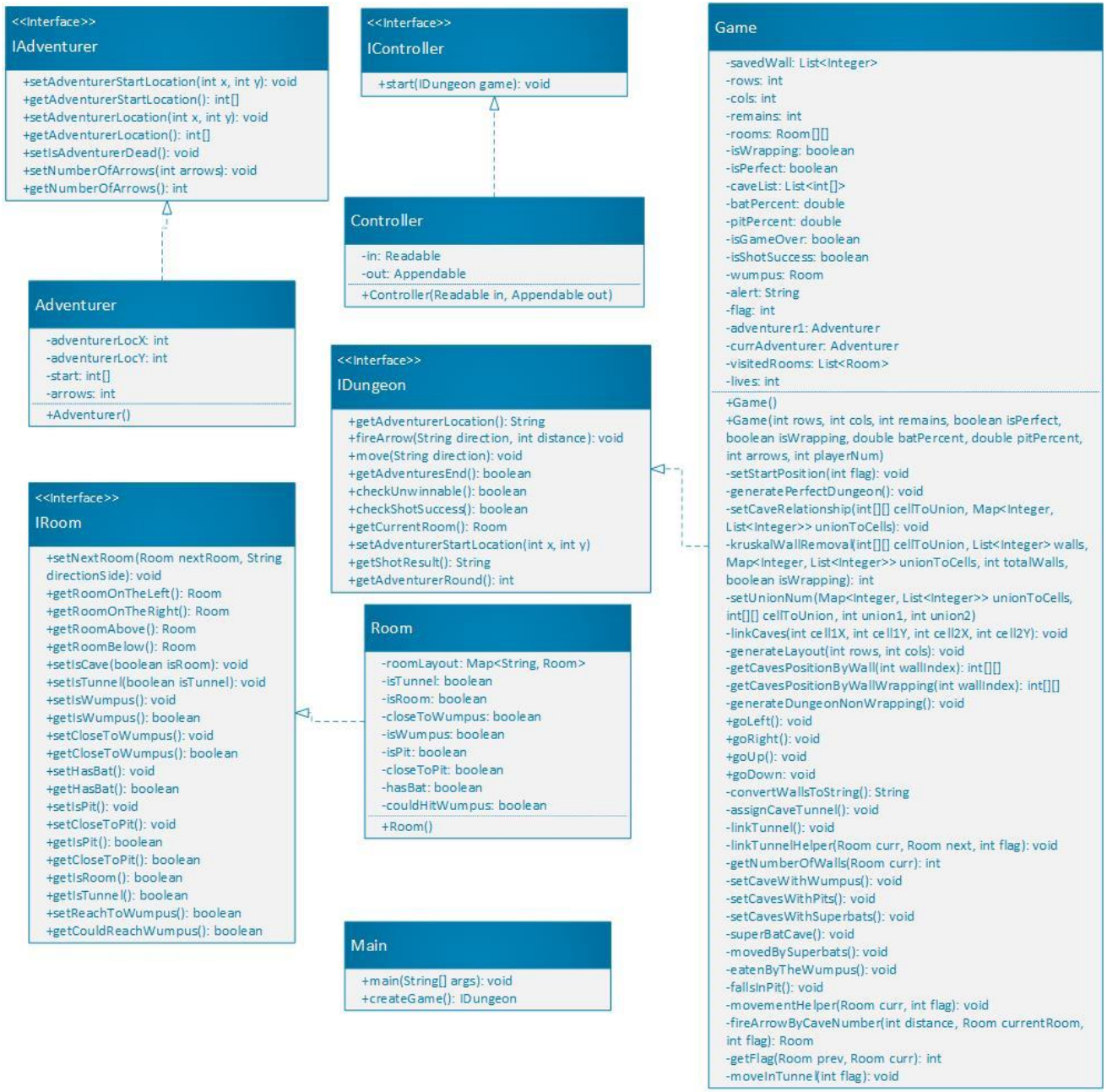
Difficulties

My primary difficulty was the time limit based on being out for three weeks and making the most of crunched time. There are some things I would have liked to spend more time on, particularly the text based interface. There is some error handling that could be improved as well. However, it seems functional and correct.

Notes

N/A.

UML Diagram


<<Interface>>
IAdventurer
+setAdventurerStartLocation(int x, int y): void
+getAdventurerStartLocation(): int[]
+setAdventurerLocation(int x, int y): void
+getAdventurerLocation(): int[]
+setIsAdventurerDead(): void
+setNumberOfArrows(int arrows): void
+getNumberOfArrows(): int
<<Interface>>
IController
+start(IDungeon game): void
Game
-savedWall: List<Integer>
-rows: int
-cols: int
-remains: int
-rooms: Room[][]
-isWrapping: boolean
-isPerfect: boolean
-caveList: List<int[]>
-batPercent: double
-pitPercent: double
-isGameOver: boolean
-isShotSuccess: boolean
-wumpus: Room
-alert: String
-flag: int
-adventurer1: Adventurer
-currAdventurer: Adventurer
-visitedRooms: List<Room>
-lives: int
+Game()
+Game(int rows, int cols, int remains, boolean isPerfect, boolean isWrapping, double batPercent, double pitPercent, int arrows, int playerNum)
-setStartPosition(int flag): void
-generatePerfectDungeon(): void
-setCaveRelationship(int[][] cellToUnion, Map<Integer, List<Integer>> unionToCells): void
-kruskalWallRemoval(int[][] cellToUnion, List<Integer> walls, Map<Integer, List<Integer>> unionToCells, int totalWalls, boolean isWrapping): int
-setUnionNum(Map<Integer, List<Integer>> unionToCells, int[][] cellToUnion, int union1, int union2)
-linkCaves(int cell1X, int cell1Y, int cell2X, int cell2Y): void
-generateLayout(int rows, int cols): void
-getCavesPositionByWall(int wallIndex): int[][]
-getCavesPositionByWallWrapping(int wallIndex): int[][]
-generateDungeonNonWrapping(): void
+goLeft(): void
+goRight(): void
+goUp(): void
+goDown: void
-convertWallsToString(): String
-assignCaveTunnel(): void
-linkTunnel(): void
-linkTunnelHelper(Room curr, Room next, int flag): void
-getNumberOfWalls(Room curr): int
-setCaveWithWumpus(): void
-setCavesWithPits(): void
-setCavesWithSuperbats(): void
-superBatCave(): void
-movedBySuperbats(): void
-eatenByTheWumpus(): void
-fallsInPit(): void
-movementHelper(Room curr, int flag): void
-fireArrowByCaveNumber(int distance, Room currentRoom, int flag): Room
-getFlag(Room prev, Room curr): int
-moveInTunnel(int flag): void
Controller
-in: Readable
-out: Appendable
+Controller(Readable in, Appendable out)
Adventurer
-adventurerLocX: int
-adventurerLocY: int
-start: int[]
-arrows: int
+Adventurer()
<<Interface>>
IDungeon
+getAdventurerLocation(): String
+fireArrow(String direction, int distance): void
+move(String direction): void
+getAdventuresEnd(): boolean
+checkUnwinnable(): boolean
+checkShotSuccess(): boolean
+getCurrentRoom(): Room
+setAdventurerStartLocation(int x, int y)
+getShotResult(): String
+getAdventurerRound(): int
<<Interface>>
IRoom
+setNextRoom(Room nextRoom, String directionSide): void
+getRoomOnTheLeft(): Room
+getRoomOnTheRight(): Room
+getRoomAbove(): Room
+getRoomBelow(): Room
+setIsCave(boolean isRoom): void
+setIsTunnel(boolean isTunnel): void
+setIsWumpus(): void
+getIsWumpus(): boolean
+setCloseToWumpus(): void
+getCloseToWumpus(): boolean
+setHasBat(): void
+getHasBat(): boolean
+setIsPit(): void
+setCloseToPit(): void
+getIsPit(): boolean
+getCloseToPit(): boolean
+getIsRoom(): boolean
+getIsTunnel(): boolean
+setReachToWumpus(): boolean
+getCouldReachWumpus(): boolean
Room
-roomLayout: Map<String, Room>
-isTunnel: boolean
-isRoom: boolean
-closeToWumpus: boolean
-isWumpus: boolean
-isPit: boolean
-closeToPit: boolean
-hasBat: boolean
-couldHitWumpus: boolean
+Room()
Main
+main(String[] args): void
+createGame(): IDungeon