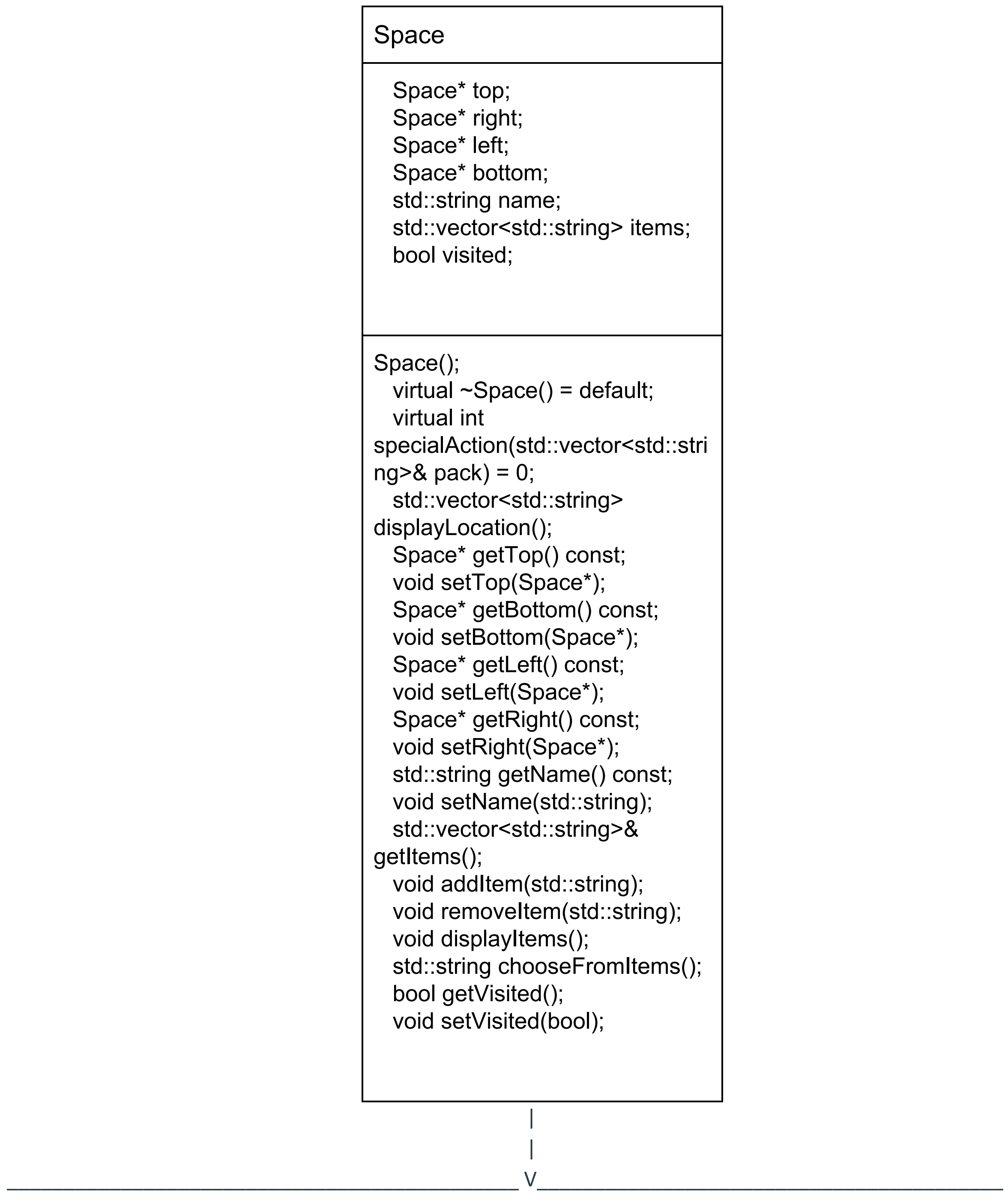
Rebecca Lashua
CS 162: Intro to CS II
Final Project Design and Reflection

DESIGN

CLASS HIERARCHY DIAGRAM



```
Space

   Space* top;
   Space* right;
   Space* left;
   Space* bottom;
   std::string name;
   std::vector<std::string> items;
   bool visited;

Space();
   virtual ~Space() = default;
   virtual int
specialAction(std::vector<std::stri
ng>& pack) = 0;
   std::vector<std::string>
displayLocation();
   Space* getTop() const;
   void setTop(Space*);
   Space* getBottom() const;
   void setBottom(Space*);
   Space* getLeft() const;
   void setLeft(Space*);
   Space* getRight() const;
   void setRight(Space*);
   std::string getName() const;
   void setName(std::string);
   std::vector<std::string>&
getItems();
   void addItem(std::string);
   void removeItem(std::string);
   void displayItems();
   std::string chooseFromItems();
   bool getVisited();
   void setVisited(bool);
```

```
|                  |            |        |         |            |            |
|                  |            |        |         |            |            |
V                  V            V        V         V            V            V
```

Cage	LabFloor	Closet	Vent	BreakRoom	Office	Outside
Cage() Int specialAction() Bool makeJump()	LabFloor() Int specialAction() Bool brokenIntoBox()	Closet() Int specialAction() Void rollDie()	Vent() Int specialAction() Bool dodgeSpideR()	BreakRoom() Int specialAction() Bool breakIntoSafe()	Office() Int specialAction() bool outrun()	Outside() Int specialAction()

DESIGN AND FLOW:

The game starts in the cage, where a rat pack is retrieved (or not if the user chooses). Then the player needs to jump up to the water bottle and drink before leaving. Then once in the labroom floor, the player needs to guess a number in order to open a box of items. An important item, “scalpel” is in there. That will be useful in the vent when fighting the spider (player doesn’t know this). Then on to the closet, where they will play a roll Die challenge with a guinea pig. Then they can move to the vent, where they will fight a radioactive spider. If the player has the scalpel, they can use it. Otherwise, they can try and dodge the spider, risking instant death. Then they can either go to the breakroom, where they will need to break into a safe and possibly pick up cat nip, or they can go to the office, where they will need to either throw cat nip at to escape or will need to chance outrunning, once again chancing instant death. Once they reach outside, without going over the step limit, they win. If the step limit is reached before getting outside, they have been caught by the silly lab assistant.

Steps are incremented every time a space is entered or reentered and also are incremented when the player needs to replay a challenge game.

Everything flows from Game::simulate().

- First the space is set up Game::setUpSpace(), with all spaces pointing at the right spaces.
- Then the introduction message is displayed. Game::displayIntro()

- Then a while loop is entered until the location of the player is outside.
    - If it is the first time in the rat cage, a rat pack is offered to player.
    - If the location has never been visited, the Space::specialAction() is called.
        - Returns any steps that might have accrued while fighting challenges.
        - -1 is returned to signal the player has died
    - The main menu is called after the special action (or immediately when space is re-entered more than a first time).
        - This menu can be used to pick up items, remove items, add to the rat pack, or change space
    - Space is changed if menu returns that option
    - Steps are incremented
    - If steps are too high, game is over.
- Once the player gets outside without using too many steps, they win the game.

TESTING PLAN.

- First test the Space class thoroughly, making sure the get and set methods work. And that the space is properly constructed.
- Test each individual derived class.
    - Test their helper method that Space::specialAction() calls.
    - Test their personalized Space::specialAction() method.
    - Test the Space::displayLocation() method.
    - Test Space::displayItems() method,
    - Test Space::chooseFromItems() method.
- Then test the Game class methods.
    - Test the get and set methods
    - Test the methods related to the main menu (Game::mainMenu(), Game::addToPack(), Game::removeFromPack, Game::displayPack(), Game::pickUpItem())
    - Test Game::changeSpace()
    - Test Game::simulate()
- Test the entire game from main.cpp

Once I was able to pass all of these tests, I was able to submit my project.

REFLECTIONS

So right now I am struggling with the flow of my game. I was going to handle all cases of action within the specialAction() method within Space and then update the currentLocation to the next space within my game, but I realized that this will not work because sometimes the player might want to trace their steps backwards to retrieve an item that they didn't pick up beforehand but now need. And since I need to know what is in the rat pack before I add anything in my specialAction() method, I need to reconfigure how and where the game flow happens.

I would like to start out my game with the player deciding whether or not they want to take a rat pack with them. The rat pack is their container to carry items during their escape. And it is required in order to finish the game, because you can't escape certain rooms without having specific items. For example, I am planning on making it impossible to leave the office (the last room) to the outside unless you have catnip to distract the cat with. That catnip is picked up in the closet. If the player gets to the office without catnip, they will have to go all the way back to the beginning, wasting 6 steps, in order to get their rat pack. And then make it all the way back. It is possible that this causes them to lose because they run out of steps.

But here is my problem. If they don't take their rat pack, how do I store the rat pack until then? For example, if I were to not take it with me, and then I traced back to go get it, how would I then recapture it? I can't do that in my specialAbilities function because that function does not track the hasPack boolean value. Perhaps in my simulate() while loop I can first check to make sure the boolean "visited" Space attribute is true, and if it is true, and hasPack is false, then I offer the rat pack again.

I decided to solve this problem by creating a new Game method called retrieveRatPack() that checks to see if the cage has already been visited (meaning the player has moved on to the next space at least once already), and if not, it offers the first statement about the cagemate offering it to them, and if so, it has a welcoming back statement and offers it again. This is much cleaner, and doesn't detract from the special abilities method that is called on cage.

I figured out a way to handle the instant death scenarios. Beforehand, I had made the Space::specialAction() methods boolean methods. And would then loop over them in the Game::simulate() method until the player beats the challenge. But this was ineffective and unclear, as well as messy. So I instead changed that method to be an int method, that returns a step count that then can be added to the step count in Game::simulate(). And if the player dies within one of the space's methods (the office where the cat is and the vent where the radioactive spider is), then it returns -1. And I just check for that in the Game::simulate() method. If it doesn't return -1, then that number is added to the steps count. This is a much more elegant solution. I am sure I could think of a solution that is even better one, but I think this suffices for now until I can think of something better.

Now I am stuck on the issue of how to handle the cage actions. Because I would like the player to have the option of not taking the rat pack with them, to later find out that they need to return to the cage to retrieve it. But the way I wrote my Game::simulate() method, I check for the condition of a room being called, and then call Space::specialAction(), since I don't want the action to happen when they have already visited the room. (For example, when the rat has already fought or dodged the spider, the spider won't appear in the vent again). I need to play with it more and see what happens.

I ultimately decided to go ahead and change my while loop in simulate() to separately check for the case of whether or not cage has been visited and to not call retrieveRatPack before entering the while loop, as I was doing before.