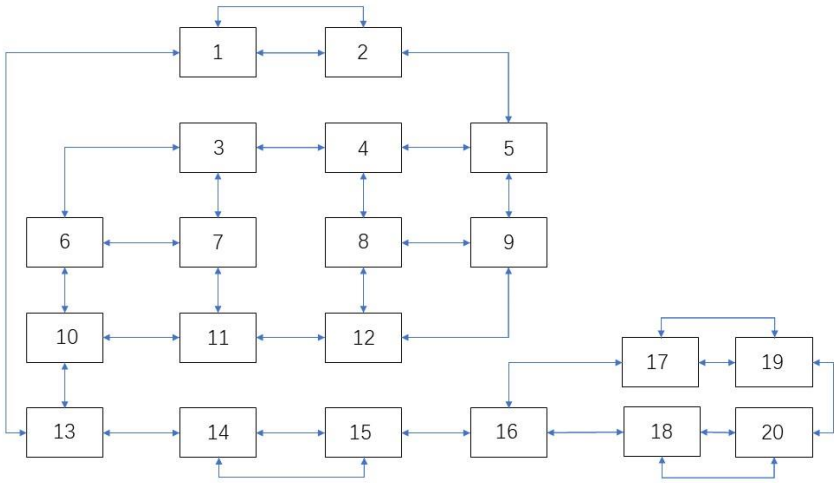
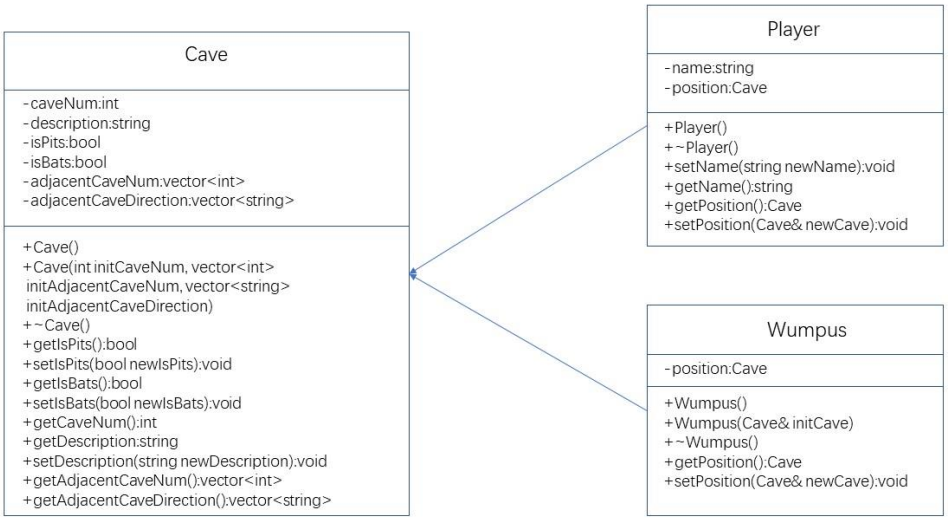
Name: Yunkai Liu ID:29616425

MAP:



UML diagram:



Additional function:

1. Player have two lives. Once player is killed by pits or Wumpus, player will lose one life and then be reset to a new cave that does not contain any hazards. If player loses both lives, player will lose the game.
To achieve this functionality:
i. I create a global variable called numLife and assign 2 to it.
ii. If player encounter hazards, numLife--. If numLife==0, isGameEnd variable will be set to true. The program will jump out of the loop and the game is over. Otherwise, go to step iii.
iii. The program will call a function named resetPosition. This function will use a do while loop to generate a new location for player.

```
do
{
    //generate a random value from 0 to 19 using rand() % 20
    //and set player's position randomly by using this value
} while (player's new position is same as one of hazards' position);
```

2. There will be a map item in the game environment. If player get the map, player can use [MAP] instruction to see the map of the cavern.
To achieve this functionality:
i. I create two global variables, bool hasMap = false and int mapPosition
ii. I generate a random value and assign it to mapPosition
iii. If the mapPosition and the caveNum of cave that player is in now are the same, hasMap will be set to true. A message will be displayed to player for notification. Then mapPosition will be set to a negative value to ensure that message will not be displayed twice.
iv. When player enter MAP, if hasMap is true, display the map to player, otherwise show an error message.

Program Reflection:

Motivation of the program design:

I design the program starting with completing all classes I might need. Because this program only has two instructions and they are all related to one or more classes like Cave or Wumpus. As a result, I think after finishing all classes, remaining part will be easy to complete.

For the class design, first I analysis all the things I might need for this program : Cave, Player, Bats, Pits, Wumpus. But bats and pits can be an attribute of cave because they do not have many behaviors. All they need to do is influencing player's state after player encounter them. So, I think they can be a bool attribute of Cave. Then I can check them to know whether I need to change player's state or not. As a result, I only have three classes as shown in the UML diagram above.

For the driver cpp of this program, the main structure will be a while loop. If player do not lose all lives or kill the Wumpus, this while loop will continue asking for new input and use this input to find the correct response.

Implementation of design:

When I tried to implement my design, I didn't have issues with writing Cave, Player and Wumpus classes but I got several problems while coding the driver class.

First, when I wanted to write move function for MOVE instruction, I noticed that move function could not be invoked at all. I thought that this issue might be caused by errors in the body of move function so I commented out those code to check the location of problem. However, even I commented all the body out, the function still did not work. Then I found that the problem is the name of function. Move cannot be used to name the function in cpp.

Second, I had a problem for initializing all caves. Because each cave will have three adjacent caves according to the map and I cannot find a regular pattern here. That means I had to create all objects one by one. But that is unfriendly for code extension. Then I found a better way to solve this problem. I created an array to store all 60 adjacent caves and went through a for loop 20 times. Each time I could take 3 adjacent caves to initialize one cave. Thus, this code becomes extensible. If I add more caves, all I need to do is adding elements to the array and go through loop for more times.

Easier or more efficient solution:

If I could rewrite the code, I'll do two changes.

First, I'll wrap all code in the driver class into functions. At start, I did not have a clear understanding of what code I might need for this program. Thus, after I finished the code, I found that there are many duplicates. So I wrapped those duplicates code into functions. That wastes me a lot of time. Next time, I'll wrap everything first. At last, if I found that code is not used more than once, I can copy the code from the function body, replace function with that part of code and delete function. That will speed up my coding.

Second, I did not take care of the sequence of code in if and else at start. Then I noticed that sometimes by swapping the statement in if and else, code can run much more efficient. So next time I will be careful about this problem. Put simple code in if block and put complex code in else block.