The Urban Jewel Heist

Due: Nov 22, 2024 11:59 PM CST 100 points.

- **This final programming project counts toward 10% of the course grade.**
- You are allowed to work with at most one other person on this final project. Your group mate CAN be from a **different section** than yours. Make sure to put both of your names, user names (e.g. hyogcn), and section info in the header of all the C++ files. Only one of you needs to submit to GitLab.
- The due date for different sections may vary. Please verify the exact due date with your instructor to ensure timely submission.

Documentation Header Reminder

Before you start your assignment, you will need to add documentation similar to what we demonstrated in the first few lectures.

```
// Programmer: San Yeung
// Date: 9/4/1002
// File: fahr2celc.cpp
// Assignment: HW2
// Purpose: this file contains the main function of the program which
//    will input Fahrenheit temps from the user, then convert and
//    output the corresponding Celcius temperature.
```

Function Documentation Reminder

Having proper documentation for C++ functions is important. Make sure that each function includes the following:

- General Description: A brief description of what the function does.
- Precondition: Describe any requirements that must be met before calling the function.
- Postcondition: Explain what the function accomplishes and any changes it makes to variables or data.

Multiple File Compilation

For this assignment (and onward), you will submit **multiple** C++ compilable files containing a program written in C++. Name your files a meaningful name and give it a proper extension (.h, .hpp, .cpp).

To compile multiple files in the same directory, simply execute "fg++ *.cpp -o your_executable_file_name" on the CS Linux virtual machine's command line environment. The syntax *.cpp will grab all the .cpp files to be compiled (header files need NOT to be included). Remember, fg++ is just a shortcut for invoking the g++

command with some useful flags turned on, i.e. g++ -Wall -W -s -pedantic-errors, to ensure the quality of the program follows the C++ standard.

Background

While you are working hard to try to get as many dates as possible in the big city, a group of skillful robbers is also working hard, hoping to add more funds to their retirement accounts by deciding to rob a rich bank together[1]. However, during their attempt, an incident occurred and caused all of the valuable jewels in the bank to end up scattering all over the city streets. The robbers are now rushing to recollect their prize, but of course the bank alarms sounded and cops are now just showing up too. You know that an epic chase is about to happen in the city grid, and you just couldn't resist watching it through your apartment window even though your love interest has been calling your mobile more than ten times now…

Specifications

For each class defined below, you do have the flexibility to add additional class members as needed. This can include more variables and proper member functions, as long as they serve a purpose and improve the overall organization and functionality of the class. Furthermore, you are welcome to define new classes too as long as there is a clear reason behind it.

1. The **City** Class

This class should have the following attributes:

- A 2D array of type character (char) with size 17 x 10 representing the **city grid**, where the jewels, robbers, and police officers are positioned. It serves as the playing field for our simulation.
- A **current count** of the number of jewels scattered in the city grid.
    - The number of jewels in the city grid is subject to change as police and robbers move during the chase.

It should have the following member functions:

- An overloaded **default constructor** (with no parameters) to initialize the grid and set the required number of jewels scattered around the city.

2. The **Jewel** Class

This class should have the following attributes:

- The **value** (or the worth) of the jewel.
- The **coordinate** that it was originally scattered at in the city grid.

A jewel should have the following member functions:

- `Jewel(const int value, const int xPos, const int yPos):` A parameterized constructor that takes a value and a position upon creation.
- **Overloading the `*` (Multiplication) Operator**
    - It MUST be overloaded as a **non-member function**, but can be a *friend* function.
    - The multiplication operator is used to return a new jewel object with a new value that is doubled of the original. For instance, if a `Jewel` object has a value of 100, using this operator (`aJewel * JEWEL_FACTOR`) would result in a new `Jewel` object with a value of 200, where `JEWEL_FACTOR` is a constant integer set to 2.
- **Overloading the `*=` Operator**
    - It MUST be overloaded as a **member function.**
    - The fast multiplication assignment operator is used to modify the current, existing `Jewel` object with a new value that is doubled of the original. Using `aJewel *= JEWEL_FACTOR` will change the *calling object* (i.e., `aJewel`) with an initial value of 100 directly to 200.

3. The **Robber** Class

This class is required to be implemented as a *template class*. This will allow robbers to collect and manage various types of loot items. While this assignment is using jewels, other different loot items could later be involved such as cash, artifacts, cryptocurrencies, and puppies. By templating the Robber class, the simulation would be able to adapt to different loot types without modifying the underlying logic for robber behavior.

The class should have the following attributes:

- An unique **id**
- The current **coordinate** for the robber's position in the city grid
- A **bag** to hold different types of loot items (i.e. jewels, cash, artifacts, or cryptos)
    - The maximum capacity for the bag should be set to 20.
- Another current **count** of the total worth of the loot collected by ALL the robbers so far. This should be a *static* class member variable.
- A variable indicating whether the robber is still **active** (inactive corresponds to being arrested by the cop)
- A variable representing the type of the robber: *ordinary* or *greedy.* Each type of a robber *moves* differently in the program.

A robber should have the following member functions:

- `Robber(const Robber& other):` **(Copy Constructor Overload) (INCORRECT)**
- **Corrected:** `Robber(Robber& other):` **(Copy Constructor Overload)**
    - This constructor will be used to create a new `Robber` instance as a copy of an existing one. It should copy all the attributes of the existing one.
    - *Loot Bag Duplication:* The loot items in the original robber's bag are to be randomly redistributed between the original and the newly copied robber.
- The ***pickUpLoot()*** function
    - Simply insert the loot into the robber's bag if it is not full. If the bag is already full, then perform nothing.
- **Fast Decrement Operator Overload** (`--`)
    - This operator should simulate the robber losing one loot item from their bag.
    - You may choose to implement either the **prefix** (e.g., `--robber`) or the **postfix** (e.g., `robber--`) version.
    - The operator should also check if the bag is not empty before attempting to remove a loot item.

- The ***move()*** function
    - This function should be used to move the robber one step in a specific direction. The direction should be determined by a random number between 0 and 7.

0 (NW)	1 (N)	2 (NE)
3 (W)	Robber	4 (E)
5 (SW)	6 (S)	7 (SE)

    - Only **valid** movements are allowed. No one is allowed to move outside of the city grid border.
    - **Tripping Mechanism:** The robber might trip during the move with a 60% chance. If a trip occurs, the robber will drop one piece of loot from their bag, calling the decrement operator overload to simulate this loss. Place the dropped loot in the *nearest available cell.*

Possible scenarios after moving to the new spot:

- You ran into your friend, another robber:
    - Nothing bad would happen. You happily greet your friend, cheerio!
    - However, if you are a greedy robber, when you bump into another active robber, your overexcitement will cause *half* of your already collected loot to fall out of the bag! You'll need to redistribute the fallen loot back to their original coordinates in the city grid. If the grid is already occupied by any entity, then choose another random location to place it at.
        - Only the **moving** greedy robber will lose half of the jewels, even if the robber being bumped into is another greedy robber.
        - The suggestion is to **round down** if the number of loot in the bag turns out to be an odd number.
- You found a loot at the new spot:
    - Put it into the bag by using the *pickUpLoot()* function. You should be properly recording its coordinate and estimated value:
        - The estimated value of the loot should be computed as the summation of the grid coordinate's x and y values *squared*. For instance, the value for the loot would be $196 if it's found at coordinate (7, 7) in the city grid ((7+7)^2 = 196).

    - Moreover, if you are a greedy robber, you'll need to check this: if the estimated value of the loot turns out to be an *even* value, then the greedy robber gets to move again!
        - The greedy robber can only move 3 times consecutively though.
    - **Extra Robber Spawning:**
        - When a robber's total loot value (*not* the collective worth) exceeds a threshold during their turn, spawn (instantiate) a **new** robber using the *copy constructor* of the `Robber` class. This robber would start fresh at a random location.
        - Each robber is allowed to spawn at most ONCE in the simulation.
- You ran into a cop, a police:
    - Too bad, you get caught by the police and the *arrest()* function from the Police class will be called. Your status should then be set to inactive and can no longer participate in the chase.
- At the end of the move() function, if the robber has been moving 5 turns (including the current one) without picking up a jewel, then the robber will become immobilized for the next two turns.

- [**Bonus** (10 points)] This bonus implementation only applies to the greedy robber type. Modify the move() function for greedy robbers to follow these additional rules:
    - [5 points] When getting the random direction, the resulting direction needs to be guaranteed that it contains at least one jewel in its path (to the border of the city grid). If such a direction does not exist (i.e. the current jewel count in the city is zero) , then simply move in a random direction.
    - [5 points] If a greedy robber has a full bag (maximum capacity reached), they will try to find the closest police officer and attempt to bribe them. The greedy robber will move in the direction of the closest police officer, even if there's no jewel in that direction. If there are multiple police officers at the same distance, the robber will move towards the one with the lowest ID. Once reached, the greedy robber will give up one of their collected jewels from the bag, and the police officer will release one of the arrested

robbers. The released robber will continue the case. If the police officer didn't arrest any robbers during the chase, the officer would keep the bribe as part of their confiscated loot. Otherwise, it should NOT be counted toward the total confiscated loot at the end of the program.

4. The **Police** Class
   - This class should have the following attributes:
     - An unique police **id**
     - The current **coordinate** for the police's position in the city grid
     - A current **count** of the total worth of the loot confiscated so far
     - The total number of robbers caught by the police
   - A police officer should have the following function(s):
     - The ***arrest()*** function
       - This function is invoked whenever a robber gets caught by the police. During the arrest, the police should diligently record the worth of the confiscated loot and update the number of robbers detained by incrementing one.
     - The ***move()*** function
       - This function should be used to move the police one step in a specific direction, identical to that of an ordinary robber.

Possible scenarios after moving to the new spot:

- You ran into an active robber:
  - Arrest the robber immediately!
  - **Single Robber Arrest:**
    - When a single active robber is caught, the police will use the standard multiplication operator (`*`) to increase the value of the confiscated loot, i.e., each piece of jewel's value is doubled in the robber's bag.
    - *Example:* `aJewel = aJewel * 2;`
  - **Multiple Robbers Arrest:**
    - If there is a group of robbers at the spot, then the police should arrest all of them. Use the fast multiplication assignment (`*=`) operator to increase the value of the confiscated loot.
    - *Example:* `aJewel *= 2;`
- You found a loot at the new spot:
  - Remove it from the city grid (not counting it toward the total confiscation) because you're for sure going to return it to the bank after the chase is over.

- You ran into another officer:
    - Evenly distribute the current confiscated loot between this officer and the other one. Then, the current officer should attempt to move again until they do not encounter another officer.

5. The **AI Police Assistant (Drone)** Class
    - This class should have the following attributes:
        - A unique **ID** for the assistant.
        - The current **coordinate** for the assistant's position in the city grid.
    - A police assistant should have the following function(s):
        - The ***move()*** function
            - This function should be used to move the police assistant one step in a specific direction, identical to that of a police officer.
        - The ***scan()*** function
            - This function should be used to allow the drone to scan the nearby area for robbers, and report back their positions.
            - The scan is performed in four cardinal directions (north, east, south, west) in this fixed order, each scan extending all the way to the edge of the city grid or until a robber is detected.
            - Upon detecting a robber during its scan, the drone will immediately report the exact location of the first robber it identifies to ALL the police. During the police's next turn following the receipt of the drone's report (if not a NULL detection), the police officers will prioritize moving towards the reported location/direction.
    - Given that the drones are airborne, they can move freely across the city grid without worrying about collision.

Additional Requirements

1. You are required to implement at least ONE static member function in your project. They should provide meaningful utility to the simulation and demonstrate effective use of static functionality within an object-oriented context.

Overall Program Flow

1. The simulation of the chase should begin by first creating the city grid.
2. Randomly scatter 70 jewels in the city grid by notating them with 'J' in the grid.
    a. Each cell in the grid can be occupied by only one jewel.
3. Create three police, three AI drones, two ordinary robbers, and two greedy robbers. Randomize their initial locations in the grid and notate them with 'p', 'd' and 'r' correspondingly.
    a. Each cell in the grid can be occupied by only one entity in the beginning.
    b. To effectively prevent uncontrolled spawning of new robbers that could lead to social chaos, place a **limit** on the total number of robbers objects created throughout the game to be 25.
4. Print out the initial state of the city grid nicely (you might want to create a function to handle this specifically).
5. Let the chase begin! Each turn consists of having the ordinary robbers move first, followed by the greedy robbers, and then the AI drones and police officers.
    a. Remember, a robber stops moving (and becomes inactive) once they get caught. If a robber gets arrested, then the remaining robbers can still continue the chase.
    b. Use 1000 for the loot value threshold in the Robber Spawning logic.
6. At the end of each turn, print out the state of the city grid nicely.
    a. Notate the robbers who are inactive/arrested with 'A".
    b. If multiple robbers occupy the same cell (regardless of their status), then notate the cell value with 'R'.
    c. If multiple officers occupy the same cell, then notate the cell value with 'P'.
    d. If a robber and a police officer occupy the same cell, then notate the cell value with 'Q'.
    e. Collisions with AI drones will have no effect on the printing logic.
7. We will do a maximum of 100 turns. If the police fail to catch all the robbers in the final chase, then all the robbers get to run away freely, but the confiscated loot stays with the police.
    a. The robbers could also win if they managed to collectively pick up enough jewels with a net worth of $5000 at any point during the chase.
        i. If this happens, then the police will release all the arrested robbers as well since the bribe is too tempting. Also, the police get to keep all the confiscated loot too.
8. When the chase terminates, print out a summary of the chase outcome. For example, the format could be:

```
Summary of the chase:
    The robbers wins the chase because maximum turns
```

```
(30) have been reached
          Police id: 1
               Confiscated jewels amount: $512
               Final number of robbers caught: 0
          Ordinary Robber id: 1
               Final number of jewels picked up: 5
               Total jewel worth: $91
          Ordinary Robber id: 2
               Final number of jewels picked up: 0
               Total jewel worth: $0
          Greedy Robber id: 3
               Final number of jewels picked up: 22
               Total jewel worth: $75
          Greedy Robber id: 4
               Final number of jewels picked up: 8
               Total jewel worth: $240
```

9. At the end of the simulation, produce a file ("simulation_with_drones.txt") containing all the outputs.
10. Run the simulation another time without any AI drones, and produce a file ("simulation_without_drones.txt") containing all the outputs.

Additional Notes

1. As a reminder, trying to get rich by robbing a bank is ALWAYS a bad idea. It is MUCH safer to just get a job as a programmer.
2. Set the random seed to be 210 for this assignment.
3. It is recommended that each class definition goes in their own header file.
4. For all the data member variables and functions in the class definitions, it is up to you to determine their proper access levels (i.e. public vs private).
5. **Clarification on Unique ID Requirement:** For this project, each class that requires a unique ID should maintain the uniqueness *internally* within the class. The IDs do not need to be unique across different classes.
6. It is also your sole responsibility to determine the best return types and parameters for the functions specified in this assignment.
7. You may create additional member attributes in the classes if proven to be helpful.
8. You may define additional functions to be used to improve the program's organization. This includes global functions and member functions for the classes as well.

9. You may use functions from existing libraries if 1) they are explicitly stated in the assignment or 2) the usage of the library functions has been thoroughly introduced in class. Otherwise, the usage of functions from existing libraries is prohibited without the permission of your instructor.
10. Test the program with different inputs. This will help you ensure that the program is functioning as expected and will also help you identify any bugs or errors.
11. Read the error messages carefully if the program is not working as expected. Understanding the error messages will help you to identify and fix the problem.
12. Don’t forget to use an adequate amount of comments to explain your code and make it easy to understand.
13. You must use proper indentation (two whitespaces) to make the code more readable and easy to understand.
14. Finally, don't hesitate to ask for help from the instructor or TA if you are having trouble with the assignment. We’re here to support you! Also, remember that campus tutors and LEAD tutors are available to assist you as well.