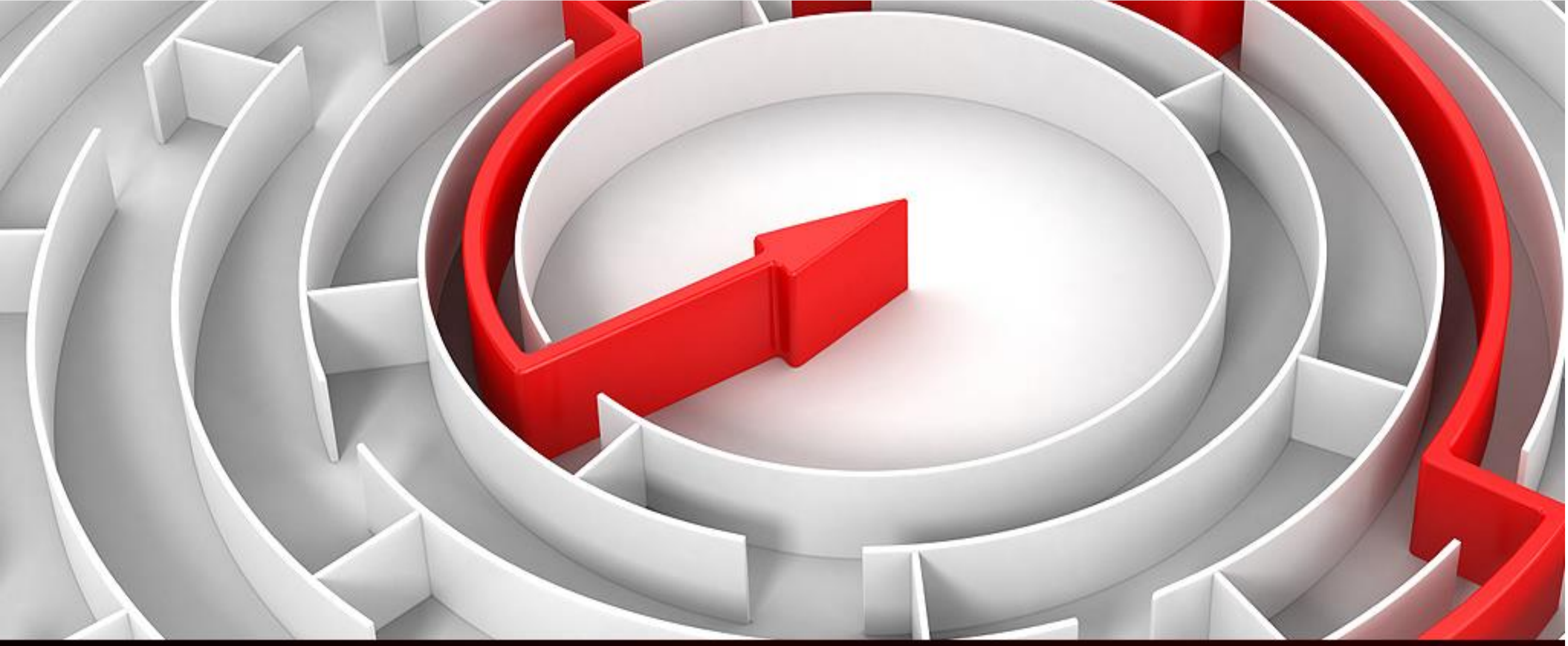


Dungeons & Dragons

Chapter 1. Introduction

1.1. Project description

The project is a game similar to the famous "Dungeons and Dragons".
The main objective of the game is for your character to move around a playing field - a map, collecting treasures and defeating monsters. The idea of the project is to make a role-playing game where the character can choose his race, his items and what type of attack to attack with (power attack or spell), when fighting a monster.

1.2. Purpose and tasks of the development

The goal of the project is to make an object-oriented game that will contain a map (maze), characters of different races, and items, including different types of weapons, spells, and armor. The developer's job is to create a character of the player's chosen race, create different types of mazes (both in size and content), and create different types of items that the player character will be able to choose whether to put in their inventory in consequence.
The game is considered complete if the player's character manages to pass through all the mazes, starting from the upper left corner and ending at the lower right corner of the maze. While traversing the maze, the player will encounter monsters to battle and find items of various types.

1.3 Structure of the documentation

The documentation is structured in five chapters:

- Introduction – project description, objectives and structure.
- Overview of the subject area - definitions, concepts and methods.
- Design - system architecture and diagrams.
- Implementation and testing - description of the implementation and tests.
- Conclusion - summary and directions for future development.

Chapter 2. Overview of the subject area

2.1. Basic definitions, concepts and algorithms

- Labyrinth Map: A playing field on which the player character will move, with (.) marking a free square and (#) marking a wall. Also, the field contains a (T) field and a (M) field, which are essentially about finding treasure and fighting a monster.
- Hero: The player character who has vitals of strength, mana and health and who belongs to one of three races – human, mage and warrior.
- Treasures and Monsters: Areas on the map that the player character interacts with positively and negatively, respectively.
- Battle: The process of interaction between the hero and a maze monster, which involves the exchange of attacks and spells between the warring parties.
- Treasure Finding: The process in which the player character chooses whether or not to replace the found item with one of their own.

2.2. Defining problems and complexity of the task at hand.

Some of the main issues include:

- The creation of a well-structured maze with well-generated randomly type (T) and (M) objects.
- Creating a well-balanced battle system between the hero and the monster.
- Good management of the different types of items in the character's inventory.
- Correct distribution of bonus points upon successful completion of the level.
- Correctly increasing the health of monsters in the maze after successfully completing a level.

2.3. Approaches and methods for solving problems.

Using well-structured object-oriented code that will allow the breakdown of a complex task into smaller, easier-to-solve problems. This will avoid generating long and complex code that is prone to problems.

2.4. User requirements and quality requirements

User Requirements: The player must be able to control his character by choosing what treasures to pick up for himself and what attacks to defeat monsters with. In addition, the player must be able to choose for himself how to distribute the bonus points upon successful completion of each level.
Quality requirements: Game scalability with increasing levels, game support and easy data update via external files.

Chapter 3. Design

3.1. General architecture – OOP design

The project architecture includes the following main classes:

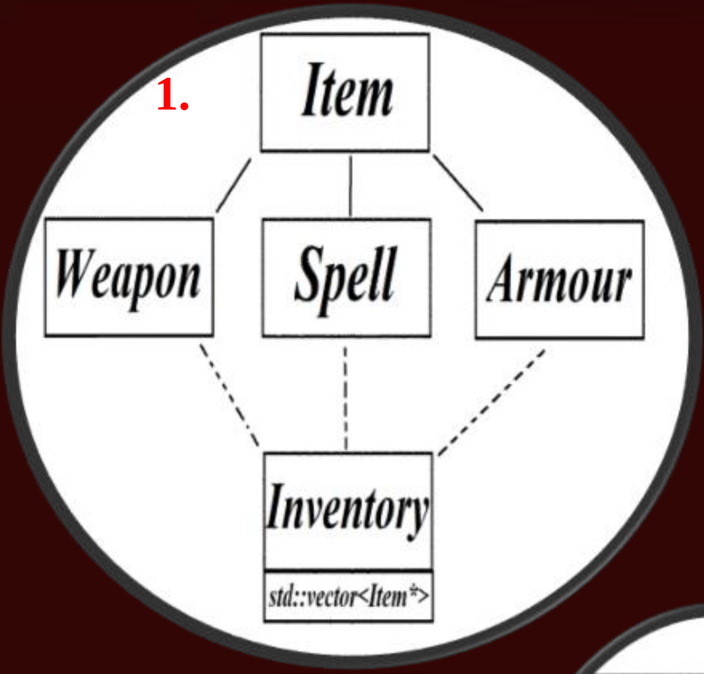
- ***Item:***[1] **Represents an item that can be found if the hook lands on a space (T). The item itself has a name and a parameter that contributes to the character's vitals.**
- ***Inventory:***[1]**Represents an inventory that contains a vector of pointers to items. It serves to easily manage the items in the character's inventory.**
- ***Character:***[2]**Represents the player's character and has attributes such as strength, mana, and health, as well as an inventory that contains the character's corresponding items.**
- ***Game:***[2] **This is the class responsible for the hero's battles with the monsters in the maze. Also responsible for the memory of the hero and monsters. It is here that it is decided whether the hero has defeated the monster or lost, which would mean the end of the game.**
- ***Maze:***[3]**This is the most important and complex class, since here the character goes through the maze recursively, collecting treasures and fighting monsters, trying to reach the exit of the maze.**
- ***Play:***[3]**Serves to provide the different levels that Maze uses.**

3.2. Inheritance charts

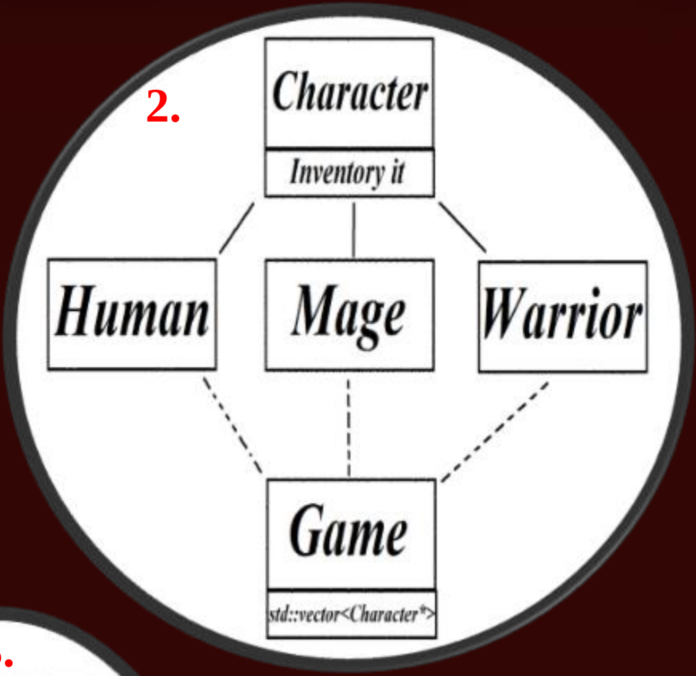
Item class polymorphism

Character class polymorphism

1.



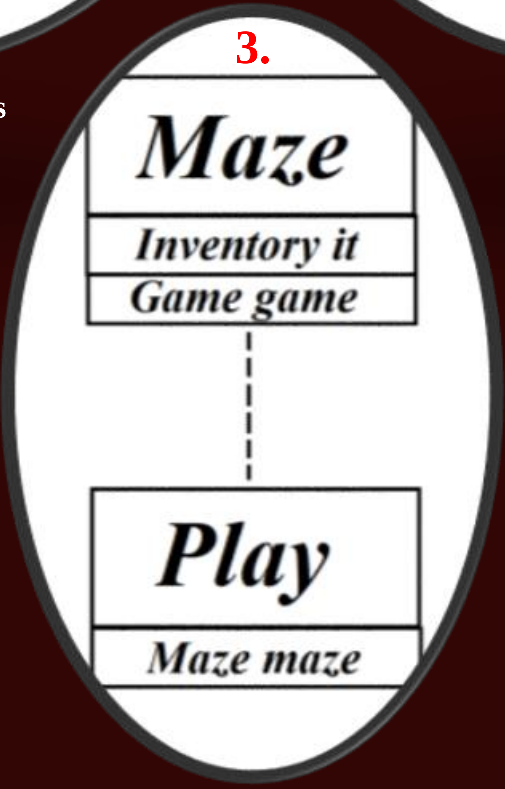
2.



Construction of Inventory class

Construction of Game class

3.

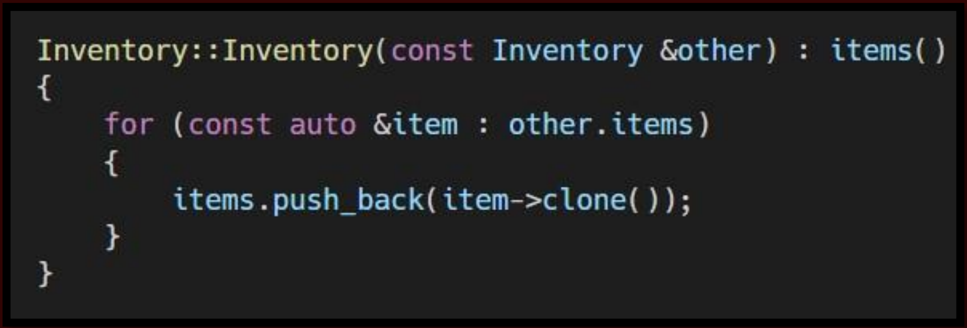


Construction of the Play class

Chapter 4. Implementation and tests

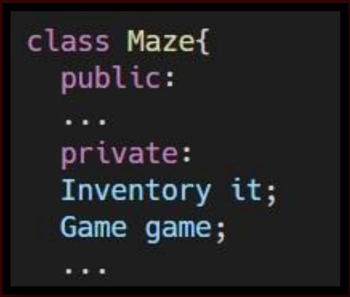
4.1. Implementation of classes

The Item and Character classes use polymorphism, to be able to provide the Inventory classes[1]and Game[2]an easy way to create a heterogeneous container (in this case std::vector<[pointers]>). In turn, the same ones above do deep copy, to avoid memory sharing. Example with the Inventory class:

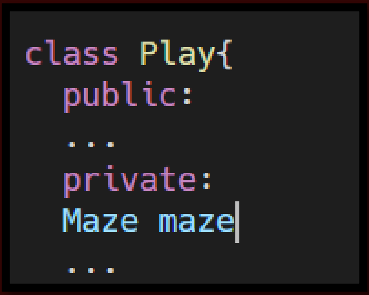


```
Inventory::Inventory(const Inventory &other) : items()
{
    for (const auto &item : other.items)
    {
        items.push_back(item->clone());
    }
}
```

The Maze and Play classes, on the other hand, use the functionality of already written classes as attributes in their "private" variables. Example, for Maze and Play classes respectively:



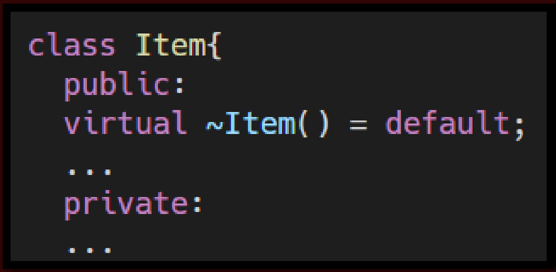
```
class Maze{
  public:
  ...
  private:
  Inventory it;
  Game game;
  ...
```



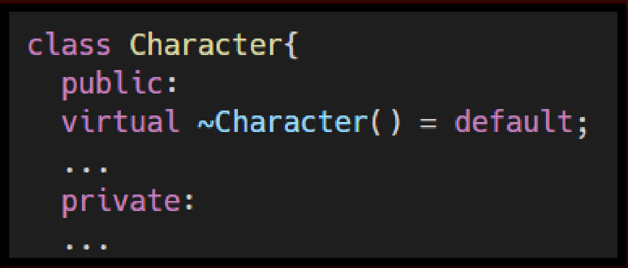
```
class Play{
  public:
  ...
  private:
  Maze maze
  ...
```

4.2. Memory management

The memory of the respective classes is managed by their respective destructors. Since two of the base classes have polymorphism, one of the things we need to consider is the virtual destructor of the base class. An example for the Item and Character classes, respectively:



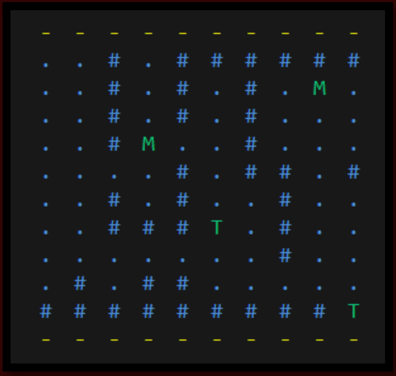
```
class Item{
  public:
  virtual ~Item() = default;
  ...
  private:
  ...
```



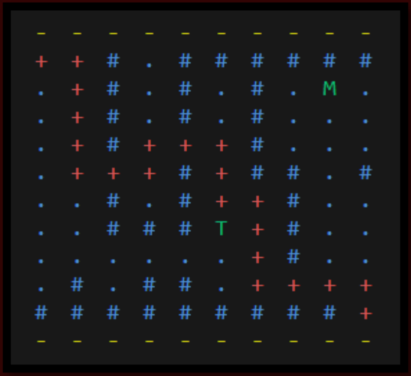
```
class Character{
  public:
  virtual ~Character() = default;
  ...
  private:
  ...
```

4.3. Planning, describing and creating test scenarios

The test scenario must meet all the specified conditions, that is, check whether the maze has an exit, whether the number of monsters and the number of treasures meet the specified condition, and whether the player character moves correctly on the map. An example, starting and passing maze respectively:



```
-  -  -  -  -  -  -  -  -  -
.  .  #  .  #  #  #  #  #  #
.  .  #  .  #  .  #  .  M  .
.  .  #  .  #  .  #  .  .  .
.  .  #  M  .  .  #  .  .  .
.  .  .  .  #  .  #  #  .  #
.  .  #  .  #  .  .  #  .  .
.  .  #  #  #  T  .  #  .  .
.  .  .  .  .  .  .  #  .  .
.  #  .  #  #  .  .  .  .  .
#  #  #  #  #  #  #  #  #  T
-  -  -  -  -  -  -  -  -  -
```



```
-  -  -  -  -  -  -  -  -  -
+  +  #  .  #  #  #  #  #  #
.  +  #  .  #  .  #  .  M  .
.  +  #  .  #  .  #  .  .  .
.  +  #  +  +  +  #  .  .  .
.  +  +  +  #  +  #  #  .  #
.  .  #  .  #  +  +  #  .  .
.  .  #  #  #  T  +  #  .  .
.  .  .  .  .  .  +  #  .  .
.  #  .  #  #  .  +  +  +  +
#  #  #  #  #  #  #  #  #  +
-  -  -  -  -  -  -  -  -  -
```

Chapter 5. Conclusion

5.1. Summary of the implementation of the initial objectives

The developed prototype of the Dungeons and Dragons game successfully realizes the main ideas of moving the character through a maze, collecting treasures, fighting monsters and successfully distributing the bonus of vital indicators when leveling up.

5.2. Directions for future development and improvement

Many new levels and new functionalities will be added in the near future such as:

- Monster items with level up.
- Changes to character races to provide more options for the player.
- Added functionality such that the player can choose to run away from the battle and seek another exit.
- Added fields (H) in the maze, such that if the player's character lands on such a field, they are healed by a certain percentage of their health.
- Adding functionality for auto-generated mazes that will meet all the requirements above.
- Added functionality that will allow an item to be upgraded if an item of the same type is found.