P

Project 2

Dungeons & Dragons Character Creator and Tester

By Kevin Kessler II



December 4, 2024

CIS-17C-47541

Prof. Mark Lehr

**Introduction:**

This program is a text-based RPG character creation and testing tool. It allows users to create characters of different classes (Warrior, Mage, Berserker), assign attributes, and test their abilities. The program is inspired by Dungeons & Dragons (D&D) and Baldur's Gate, incorporating concepts like character classes, spells, and combat actions. The program serves as a way to explore and experiment with different character builds and abilities in a text-based format.

**Reason for subject:**

I am a huge fan of D&D and games inspired by it, like Baldur’s Gate. With this program, I wanted to allow people to have small taste of what characters are like in those games, and maybe inspire them to get into the game.

**Skills Demonstrated:**

* The program uses a binary search tree to store character abilities, allowing for efficient retrieval and sorting.
* Inheritance
* Polymorphism
* data encapsulation
* Data Structures
* Error Handling via Exceptions and Try-Catch blocks
* Use of Standard Template Library (STL)
* Enums
* Operator Overloading

**Core Functionality:**

**Character Creation:**

Users can create characters of different classes, each with unique attributes and abilities.

Attributes like health, mana, strength, intelligence, etc., are assigned to characters based on their class.

Warriors have fighting styles and subclasses (e.g., Paladin, Ranger).

Mages have spell schools and subclasses (e.g., Wizard, Warlock).

Berserkers are a specialized warrior type with a frenzy state.

**Ability Testing:**

The program allows users to test character actions like attacking, defending, and using special abilities.

Mages can cast spells from their spellbook, which varies based on their spell school.

Berserkers can enter a frenzy state, enhancing their attacks.

**Error Handling:**

The program includes error handling for invalid input, such as incorrect attribute values or spell indices.

Custom exception classes are used to handle specific error scenarios.

**Code Structure:**

The code defines a framework for an RPG (Role-Playing Game) using object-oriented programming principles in C++. It establishes a base class Character and derives specialized classes like Warrior, Mage, and Berserker from it. Additionally, it includes a Spell class and various exception classes to handle errors and invalid operations.

**Class Descriptions:**

**Character**:

This abstract class serves as a blueprint for all characters in the game. It defines common attributes such as name, health, mana, and a tree-like structure to store abilities. The core actions attack, defend, and specialAbility are declared as pure virtual functions, ensuring that derived classes provide concrete implementations.

**Warrior:**

This class inherits from Character and represents warrior-type characters. It adds attributes like strength, defense, fightingStyle, and subClass to model different warrior archetypes. It overrides the base class actions to implement warrior-specific behaviors.

**Mage:**

This class also inherits from Character and represents magic users. It includes attributes like intelligence, spellSchool, subClass, and a spellbook to manage spells. It overrides the base class actions and adds a castSpell method for spellcasting.

**Berserker:**

This class further specializes the Warrior class to represent a berserker archetype. It adds attributes to manage a frenzy state, including isFrenziedState, frenzyDuration, exhaustion, and bloodlustActive. It refines the inherited actions to incorporate frenzy-related behaviors.

**Spell:**

This class represents a spell with properties like name, damage, manaCost, and element. It encapsulates the data and provides accessors (getters) for these properties.

**Pseudocode Snippet:**

// Character class

class Character {

// Attributes: name, health, mana, abilities (tree)

constructor(name, health, mana):

initialize attributes

// Methods:

getName()

getHealth()

getMana()

setName(name)

setHealth(health)

setMana(mana)

attack() - abstract method

defend() - abstract method

specialAbility() - abstract method

addAbility(ability) - adds ability to the tree

getAbilities() - returns the ability tree

}

// Warrior class (inherits from Character)

class Warrior {

// Attributes: strength, defense, fightingStyle, subClass

constructor(name, health, mana, strength, defense, fightingStyle, subClass):

initialize attributes (including inherited attributes)

// Methods:

getStrength()

getDefense()

getFightingStyle()

getSubClass()

setStrength(strength)

setDefense(defense)

setFightingStyle(fightingStyle)

setSubClass(subClass)

getFightingStyleAsString() - returns string representation of fighting style

getSubClassAsString() - returns string representation of subclass

attack()

defend()

specialAbility()

}

// Mage class (inherits from Character)

class Mage {

// Attributes: intelligence, spellSchool, subClass, spellbook

constructor(name, health, mana, intelligence, spellSchool, subClass):

initialize attributes (including inherited attributes)

assignSpellbook() - populates the spellbook

// Methods:

getIntelligence()

getSpellSchool()

getSubClass()

getSpellbook()

setIntelligence(intelligence)

setSpellSchool(spellSchool)

setSubClass(subClass)

attack()

defend()

castSpell(spellIndex) - casts a spell from the spellbook

specialAbility()

}

// Berserker class (inherits from Warrior)

class Berserker {

// Attributes: isFrenziedState, frenzyDuration, exhaustion, bloodlustActive

constructor(name, health, mana, strength, defense, fightingStyle, subClass):

initialize attributes (including inherited attributes)

// Methods:

isFrenzied()

setFrenzied(frenzied)

attack()

defend()

frenzy() - enters a frenzy state

specialAbility()

}

// Spell class

class Spell {

// Attributes: name, damage, manaCost, element

constructor(name, damage, manaCost, element):

initialize attributes

// Methods:

getName()

getDamage()

getManaCost()

getElement()

}

// Main function

function main():

display welcome message

loop:

display character type choices (Warrior, Mage, Berserker, Exit)

get user choice

if choice is Exit:

exit loop

try:

get character name

if name is empty:

throw InvalidInputException

switch (choice):

case Warrior:

get warrior attributes (hp, mp, strength, defense, fightingStyle, subClass)

create warrior object

add abilities to warrior

display warrior information

test warrior actions (attack, defend, specialAbility)

case Mage:

get mage attributes (hp, mp, intelligence, spellSchool, subClass)

create mage object

display mage spellbook

test mage actions (attack, defend, specialAbility, castSpell)

case Berserker:

get berserker attributes (hp, mp, strength, defense, fightingStyle, subClass)

create berserker object

test berserker actions (attack, defend, frenzy, attack, specialAbility)

catch exceptions (InvalidInputException, InvalidAttributeException, etc.):

display error message

display exit message

**Flowchart:**

I am not good at designing a flowcharts with images and squares, etc. Therefore, I will write the flowchart snippet similar to pseudocode for two key functions in the code.

**Character::addAbility**

[Start] --> Is abilities tree empty?

Yes --> Create root node with new ability --> [End]

No --> current = abilities (root)

--> Loop: Is current node NULL?

Yes --> Exit loop

No --> Is new ability < current->abilityName?

Yes --> Is current->left NULL?

Yes --> current->left = new node --> Exit loop

No --> current = current->left

No --> Is current->right NULL?

Yes --> current->right = new node --> Exit loop

No --> current = current->right

--> [End]

**Mage::castSpell**

[Start] --> Get spell iterator from spellbook

--> Advance iterator to spellIndex

--> Is iterator at the end of spellbook?

Yes --> Throw InvalidSpellIndexException --> [End]

No --> Get spell from iterator

--> Is spell mana cost > mage's mana?

Yes --> Throw "Not enough mana" exception --> [End]

No --> Deduct mana cost from mage's mana

--> Print spell casting message --> [End]

**Challenges:**

I actually faced many challenges writing this code, including drastically changing course from my original project to this one. My initial project was creating a simulation of the Pokemon Trading Card Game (TCG). You were to choose 1 of 3 decks, and the AI opponent would randomly select one of the remaining two to use against you and you battled it out. However, I cam across too many difficulties to detail here. I ended up changing course to this D&D Creator and Tester because I was already familiar with writing programs like this (I love making games and learning about how games are made), having written stuff like this on my own time (mostly in Python and game engines like GameMaker Studio and Godot4). Seeing as how I was already familiar with the inspiration and had a general idea of what I needed to do, this was light years easier. I did come across one major challenge with this version however but upon closer inspection, I found that it was a relatively simple fix. All I had to do was delete one line in the Warrior.h file and in main.cpp I had to reorder some of the catches. Once that was completed everything worked fine. Overall, the code took about 6 1/2hrs to write and check and it tops out at around 880 lines of code added up.

**Summary:**

The code demonstrates several fundamental programming concepts, including object-oriented programming (OOP) principles, data structures, and error handling.

**Object-Oriented Programming (OOP)**

**Inheritance**:

The Warrior and Mage classes inherit from the Character class.

**Polymorphism:**

The attack() method is declared in the Character class but implemented differently in Warrior, Mage, and Berserker.

**Encapsulation:**

The Spell class has private attributes (name, damage, manaCost, element) and provides public methods (getName(), getDamage(), etc.) to access them.

**Data Structures**

Binary Search Tree (BST): A tree-like data structure that allows for efficient searching, insertion, and deletion of elements.

Example: The Character class uses a BST (implemented with AbilityNode) to store abilities, enabling efficient retrieval and sorting.

**Error Handling**

**Exceptions**:

The code defines custom exception classes like InvalidInputException, InvalidAttributeException, and InvalidSpellIndexException to handle specific error scenarios.

**Try-Catch Blocks**:

The main function uses try-catch blocks to handle potential exceptions during character creation and action testing.

In addition to these core concepts, the code also demonstrates:

**Standard Template Library (STL):** The use of STL containers like std::list (for the Mage's spellbook) and algorithms like std::advance (for iterator manipulation).

**Enum:** The use of enums (FightingStyle, SubClass, SpellSchool) to define named constants, improving code readability.

**Operator Overloading**: Overloading the < operator in AbilityNode to enable direct comparison between AbilityNode objects and strings.

**Conclusion:**

The provided program implements a basic text-based RPG framework with character creation and action testing, inspired by D&D and Baldur's Gate. It demonstrates object-oriented programming principles, data structures, and error handling. The code can be further extended to include more complex RPG elements and potentially a graphical user interface.