Middle East Technical University Northern Cyprus Campus

CNG 242 Programming Language Concepts – Assignment 4 - C++

Learning Outcomes: On successful completion of this assignment, a student will:

- Have utilized parts of the C++ library and classes provided
- Combine previously learned concepts to form a relatively larger scale project
- Appreciate clever design and reusability of functionality

(Art courtesy of D&D)

Dungeons, orcs, goblins and swords! A classic combination for a classic medieval themed game. In this assignment, you'll be implementing a simplified, text based dungeon crawler in console. It will have many components working together to form a big application, so please read through everything carefully.

The overall design will require you to generate a new scenario each time the program runs. The scenario will consist of special enemy encounters, strange paths leading to new areas, new items to use etc. Each time the program runs, different scenarios will be generated. Make sure to read everything carefully and connect links between **bold** words, as the design is tightly connected with many elements requiring each other.

The game has the following components, with no enforcement of your design whatsoever, so you'll have to come up with the best possible design to suit the needs of this game. There may be more than one way to implement this game.

1. Scenario

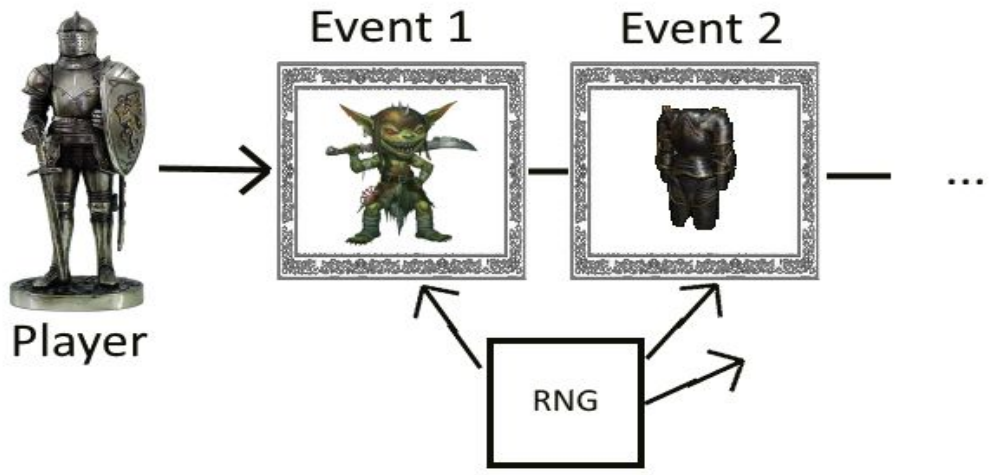
The game will consist of a series of **events**. These will dictate the encounters a **player** will have. Events are the core elements of the game. You'll generate a number of these in each game so that the program presents a series of adventures. You'll receive the total amount of events to generate as input.

```
Welcome to Dark Dungeons!
How many events will this adventure have? 100
What shall be the name of your hero? Doruk
Arise Doruk! Your adventure now begins...
```

In each event, you need to do the following:

1. Decide on the setting, or **area**. Use the file "Names.h" for this. You need to append some poetic presentation text to the setting, also found in the same file, and display this as a string to the user.
2. Decide whether it will contain an **enemy** or not. Assume it will contain an enemy **40% of the time**.
3. If an **event** doesn't contain an **enemy**, it will have 25% chance to have an **inventory** instead for the player to pick up or examine.
4.

A quick drawing to illustrate what the game is all about is as follows.



The player will be going through a series of events of which may contain an enemy, an inventory or simply nothing. The events and their contents are generated with the provided random number generated, which can be found under "RNG.h". Player goes through these until the last event.

The events will be presented, for example, as follows:

```
Turn 1
You stumble upon a treacherous crypt!
What shall you do?
        I - View Inventory
        P - Proceed to next area.
        U - Use item at slot #
        D - Drop item at slot #
        Q - Quit
Doruk
----------------
Health: 100 / 100
Level: 1
Experience: 0
```

You can see from the above that the **event** contains some presentation text that gives the feeling of atmosphere, and may present an **enemy**. You offer the following options for each event, where:

- If an **enemy** is present,
    - option to view details of it or attack it
    - If an **inventory** is dropped by an **enemy**, at the end of your battle with it to take it.
- If there is no **enemy,** the player can proceed
- If there is an **inventory** either dropped by an **enemy** or just laying around, the options to take it or examine it
- Regardless of the above, the options to view our **inventory**, use an item (this applies to **health potions** when player's health is low) or to quit

Remember that you need to keep the count of the events processed, as the game will end depending on the total number of events dealt with. Before the game begins, you'll take the total amount of events from the user. Refer to rules regarding enemies and inventories when it comes to deciding creation of those.

2. Actors

Actors are one of the key requirements for the game we are designing. An actor is anything that can interact in the game with the environment and/or another actor. To keep it simple, in this assignment, we have the following actors:

- Player
- Enemies
- Inventories

These actors are the key actors we'll require in order to have the game functional. Their definitions and requirements are as follows:

Player

The player is an actor and the main character of this game. In our game this will be the end user, deciding on the fate of the player character in each **event** with input from the console. For the sake of simplicity, we'll assume a single player type. (So **no options** for the user to choose a player character from a list like Knight, Mage, Assassin etc.)

Player will level up as they progress by killing enemies. This will be done by gaining **experience points** from kills. With each level, you'll add **+2 Maximum Health,** and every five levels, you'll add **+1 Defense and +2 Attack** to the player. Everytime the player levels up, restore their current health to be their maximum health. See below for how leveling up affects the player.

```
> A
You have killed the Orc!
You have leveled up! You are now level 2.

What shall you do?
        I - View Inventory
        P - Proceed to next area.
        U - Use item at slot #
        D - Drop item at slot #
        Q - Quit
Doruk
----------------
Health: 102 / 102
Level: 2
Experience: 15
```

You can assume maximum level a player can get is 25, and the experience requirement for levels will be calculated using the following formulae:

$L(x) = 108x^2 - 60x + 152$, raw experience for level **X**	(1)
$P(x) = 10^{log_{10}(L(x)) - 2.5}$, power to round **L(X)** with	(2)
$R(x) = int(int(\frac{L(x)}{P(x)})P(x))$, rounds the experience value to two digits. The **int()** function is just to cast it down to an integer from double.	(3)
$E(x) = int(R(x-1) - R(x-2))$, will calculate the actual experience limit of the given level **X**.	(4)

Table 1: Experience Formulae

Please note that the formulae given above is provided to make the level of experience needed as realistic as possible. At this stage, you are not required to understand how they are derived. Instead, you can directly adopt them in your design. The following example is provided to explain how to use them:

For example, let's assume you need to find the level of experience you need to reach level 3. Running the equation (4) for 3 will return 264, because **R(2) = 464**, **R(1) = 200**, which is the required experience points to reach level 3.

You'll assume the experience required to reach level 2 from level 1 is 100, which means **E(2) = 100**. If you notice **E(1)** isn't defined, as we assumed that the player starts from level 1. Beyond that, you'll be using the formula above to calculate your values, where **x** represents the level. Once player levels up, you'll reduce the excess experience points by subtracting the next level's required points, and keep the result as the current experience points. As a hint, the first 5 values from your formula should be the following:

100, 264, 480, 696, 912, ...

For example, assume player is level 3, and currently has 475 experience points. Once the player kills an enemy worth 100 experience points, they'll level up to 3 and have 575 - 480 = 95 experience points at level 4.

Player can also store **inventory** in a backpack. The backpack should be accessible in each **event** with the input **'B'**. Player will be able to select from the inventory to use in the **event** with a numeric input, like 9, which will return the inventory for use at index 9. It is assumed that the player can carry **at most 10 items**. You can store these in a list.

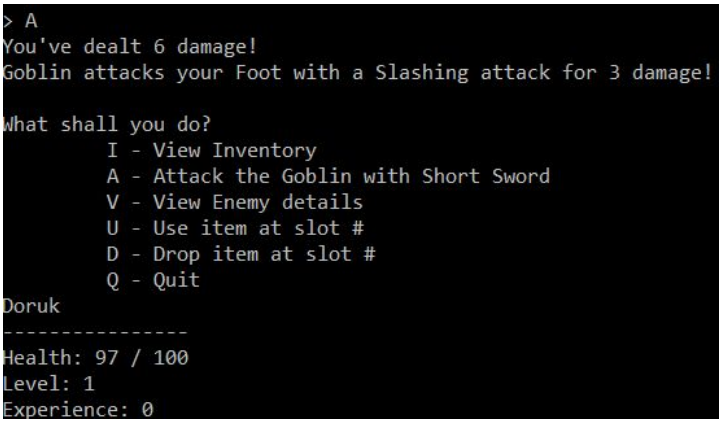
In our game, the **player** will have 100 maximum health and start with **2 health potions in his inventory and** a **short sword** (see "Names.h" for other names) that deals **5 S** damage. You can find more about these in the **inventory** section. Furthermore, the types of attacks and corresponding units are also explained below **Table 3**.

Receiving Damage: When player takes damage, you need to roll a chance to decide on which area they will be hit. These chances are:

Area	Chance %
Head	25
Body	60
Foot	15

Table 2: Damage area chances

You can think of the inventory system as in above. You'll take the corresponding **equipped inventory** of the player for damage calculations, and only consider defense of that alone! There are some small exceptions to this, and they're covered under the **inventory** section. After the calculations, damage taken will be subtracted from the current health of the player. The player can then choose to heal themselves using a health potion by 25% of their maximum health. Check the image below to see how damage is dealt.



Win Condition: The player must reach the end of the dungeon, which is specified before the game begins, by the dungeon master. If the player can kill the dungeon boss, they are the winner.

Enemies

Enemies will be from the standard medieval roster. They will contain at least the name, health, attack and defense **stats** as members.

A full, detailed list of enemies and their **stats** are as follows:

Name	Health	Attack	Defense	Drop %	Appear Weight	Exp Award	MPL
Goblin	15	3 **S**	0 **L**	10	75	30	1
Orc	25	5 **S**	1 **L**	15	60	60	1
Troll	35	8 **S**	0 **L**	20	50	85	3
Golem	60	6 **C**	1 **M - MR(3.0)**	30	30	125	6
Skeleton	35	6 **S**	2 **H**	25	45	100	6
Spider	20	4 **P**	1 **L**	20	60	55	4
Dragon	100	16 **P** - 10 **M**	5 **H - MR(1.0)**	60	20	550	10
Giant	125	12 **C**	1 **M**	50	25	500	10
Necromancer	100	16 **M**	1 **L - MR(5.0)**	60	20	600	10
Black Knight	250	16 **C** - 12 **M**	3 **H**	90	15**	2000	12
Dark Lord	500	20 **P** - 25 **M**	5 **H - MR(2.5)**	100	0*	5000	15

Table 3: Enemy data

* The Black Knight will have a weight of 30 in the last 15 **events**.
** The Dark Lord will **only appear at the last event.**

The definitions of the stats are as follows:

Name: Name of the creature in the game.

Health: The health of the creature. When it drops to zero or less, it will be defeated.

Attack: The attack type and damage of the enemy. Details can be found below.

- Types of Attack

S - Slashing

Light armor types take 20% extra damage.

P - Piercing

Light armor type takes 20% extra damage.

C - Crushing

Heavy armor types takes 35% extra damage.

M - Magic

Magical attacks can't be prevented with normal armor. They require **magic resistant armours** or **talismans** in order to be deflected.

Defense: The **armor class** and defense value. You have to make sure you factor in **armor class** weaknesses, and other **special** bonuses from inventories. Damage formula with defense factored in is as follows:

$$D = max(1, < damage\ of\ attacker > - < defense\ of\ target >)$$

- Classes of Armor

L - Light

Offers low protection. **Slash** and **Piercing** attack types do 20% more damage.

M - Medium

Has no weaknesses but offers no strengths.

H - Heavy

Crushing attack type does 35% more damage.

MR - Magic Resistance

This property is usually found in magic infused fabric or cloth. Works just like **defense** except only for magical attacks. This is a factor of the defense in which can participate in blocking magical attacks. For example, if someone takes 5 damage but they have 5 defense and 0.33 magic resistance, they'll take:

max(1, 5 - 5 * 0.33) = 3 damage (rounded down). This value has no cap, so it can go well over 1.0.

Note: The above definitions also apply to anything the player can use! So please pay close attention to that.

Drop %: The chance this enemy can drop an **inventory** item upon death.

Appear Weight: This is the particular weight of the monster to be used with the RNG. You'll classify the **enemy** to be rolled by using *bins* **[1]**, which is a procedure done by adding up all weights and figuring out which range a particular number falls in. You can find examples below.

Exp: The amount of experience points the enemy rewards to player after being killed. See the image below.

```
> A
You have killed the Goblin!

What shall you do?
        I - View Inventory
        P - Proceed to next area.
        U - Use item at slot #
        D - Drop item at slot #
        Q - Quit
Doruk
----------------
Health: 94 / 100
Level: 1
Experience: 25
```

MPL: The minimum-player-level required for the enemy to appear. If the player isn't at least this level, you'll reroll the enemy type again until this condition is satisfied.

After the classification, this enemy will be presented in the current **event.** Notice that the maximum weight possible is 400, so this is the maximum number to be rolled in the RNG. Check the image below to see how you can present an enemy. There are additional presentation text supplied in "Names.h". Pick one randomly.

```
> P
Turn 2
You managed to get to a terrifying temple!
Goblin lunges at you!

What shall you do?
        I - View Inventory
        A - Attack the Goblin with Short Sword
        V - View Enemy details
        U - Use item at slot #
        D - Drop item at slot #
        Q - Quit
Doruk
----------------
Health: 100 / 100
Level: 1
Experience: 0
```

Below you can find some examples to help you understand how to classify an enemy when rolling a random number.

***Example 1*:** If the RNG rolls 257, you'll classify this as a **Skeleton**, as it falls between 215 where a **Golem** ends, and 260 where a **Skeleton** ends (check the cumulative values).

***Example 2*:** If the RNG rolls 34, you'll classify this as a **Goblin**, as it falls between 1 and 75 where a **Goblin** ends. (Implying that **both 1 and 75 are included** in this range)

***Example 3*:** If the RNG rolls 135, you'll classify this as an **Orc**, as it's exactly at the beginning value for it. You won't classify this as a **Troll**!

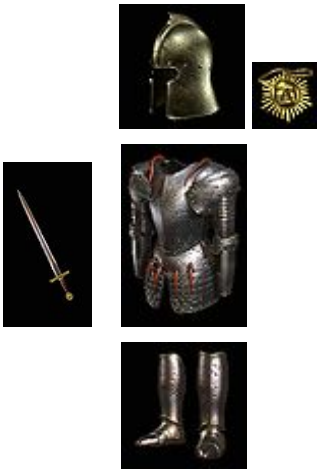
3. Areas

Areas by themselves aren't very interactable. Their labels are mostly required for storytelling purposes. However, they can contain some interesting things such as an **inventory** item. You'll simply present this area to the player when they come across an **event** and that's sufficient. See sample outputs for more information.

4. Inventory

An inventory is anything the **player** can equip or carry. Equipping implies the player can use the inventory for combat, or for other purposes like healing themselves. Carrying implies they can put the inventory in their backpack to save for later use. Inventory has subtypes. The **player** will have special slots for these to equip them in their inventory in addition to the 10 item slots for the player to carry. The subtypes that the player can **equip** are as follows:

1. **H**eadgear
2. **B**ody
3. **F**ootgear
4. **W**eapon
5. **T**alisman



(Art courtesy of Diablo II)

Headgear, Body and Footgear are your typical armor categories. The player can receive a hit from these locations, and their defense will be used when deciding how much damage they'll receive. Moreover, if the attack is magical, you'll consider the magic resist factor of the armor as well as the defense provided by the Talisman, if any. You'll add these up in your calculation.

The type the player can't equip is just **health potions**. These will be kept in the inventory of the player individually and consumed when the player consumes the potion to heal 25% of their life. The player will choose to equip for an inventory slot by their first letter. Refer to sample runs section for more info.

You're to roll random **stats** for these **inventories**, specifically the **H**eadgear, **B**ody, **F**ootgear, **W**eapon and **T**alisman types. Remember that the inventory will have at least the following: A name, a subtype, and a list of stats. Anything else is extra, but may be needed to help make your design simpler! As for finding names for your **inventory**, a name list is provided in "Names.h". You can roll a random name for each subtype there.

Rolling **stats** for **inventory** are as follows:

- 65% chance to roll only one stat.
- 35% chance to roll two stats.
- An inventory can roll a **special** only 10% of the time as a stat, and only one!

You can find the list of **stats** an inventory can have below:

Stat Name	Value Range	Which SubTypes
Health	random(1, 2 * **L**)	**H, B, F**
Defense	random(1, 1 + **L** / 3)	**H, B, F, T***
Magic Resistance	random(0, **L** / 4)	**T***
Damage	random(6, 1 + 3 * **L**)	**W, T****
Special	See **Table 6**	**All**

Table 4: Stat types

***** Talismans only provide defense against **magical** attacks, so they can only get **magic resistance** instead of defense on them! In other words, if defense is to be rolled on a Talisman, make it magic resistance instead.
****** Talismans only provide damage for **magical** attacks.

L label stands for **the player's level.** You'll increase the stat range depending on the player's level, so the player gets better items as they progress. The system is very simplistic as can be seen. There aren't many stats to be given. The only complication is on the **Special** stat type, which is a rare occurrence on an **inventory** that makes it unique. It has different chances to appear for each category, and certain special things will be attributed to the **inventory**. Refer to **Table 6**.

Inventory Subtype	Chance to Appear
Headgear	15%
Body	10%
Footgear	15%
Weapon	20%
Talisman	10%
Health Potion	30%

Table 5: Inventory Appear Percentages

Please note that an **inventory** is only allowed to have a **single special** on it. In other words, you're only required to roll the chance for this once. Refer to the table below to see what these specials can be.

To decide on the names to be given to the inventories, you can refer to "Names.h" file. You can pick a random name for the respective categories of items from there. The name itself will have no impact on the item. For the rest of the attributes on these, you can refer to **Table 4, 5 and 6.**

Special	Chance to Appear	Description	Which Inventory
Critical Strike	15%	Player's has a chance to do double damage	**W, T**
Evasion	15%	Player can mitigate attacks	**H, B, F, T**
Ignore Defense	30%	Player can ignore all defenses	**W, T**
Imbued Armor	25%	Player doesn't suffer from armor penalties when	**H, B, F**

		attacked.	
Life Leech	15%	Player's attacks drain life	**W**

Table 6: Special stat subtypes

***** You will apply critical strike damage bonus **BEFORE** defense reduction calculations. In other words, first you check if a potential critical strike can happen, then you reduce the multiplied damage by the **enemy's** defense.
****** In the case that the player chooses to use time shift in the last **event**, automatically declare player as victorious.

Critical Hits: Your attacks will deal double damage 25% of the time. Suppose you have critical strike special in your **W**eapon or **T**alisman you currently equipped. You'll roll a chance and if it's less than 15, you'll say this is a critical strike and double the player's damage. Pay close attention to the fact that this can only occur on **W**eapon and **T**alisman inventory types.

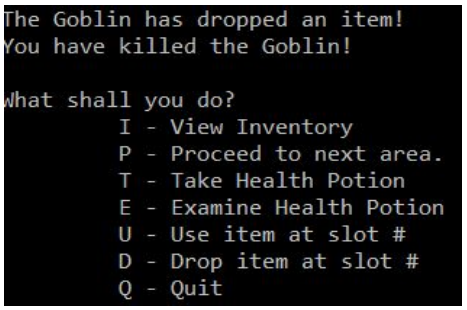
Evasion: You can completely ignore being attacked 15% of the time. Just like in critical strike, this time you'll roll a chance to completely negate an attack from an **enemy** instead. You'll assume the enemy skipped it's turn and the player can act after.

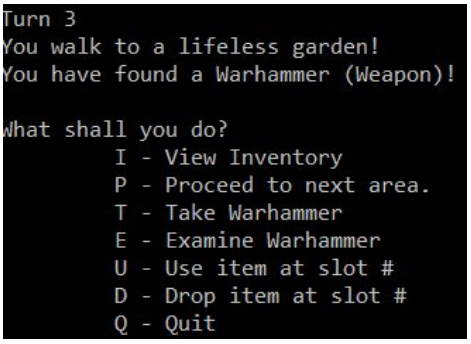
Ignore Defense: Any attack made from player will ignore any defense value of the enemy 25% of the time, treated as if the enemy has no defense at all. This will however also mean that the bonus damage from weakness of armor types will also not be applied.

Imbued Armor: When player receives an attack to the corresponding armor's location, no penalty will be applied for armor weakness against certain attacks. For example a light armor class won't suffer penalty from a slashing attack if imbued armor is present.

Life Leech: For each successful attack of player, heal them for 30% of the damage done.

In short, you have to pick which stat will be coming out with equal probability between Health, Damage, Defense, and only 10% chance for Special for **inventory subtypes** that can have them.. You will calculate a probability to drop an **inventory** once an **enemy** is defeated with respect to the **drop %** values provided in **Table 2**. After this, you'll offer the player the option to take the **inventory** or optionally save some space to pick it up by dropping another. Refer to sample runs section for how this works in more detail. Please remember that an **inventory** can appear either by **killing an enemy**, which requires you to roll a random number depending on the **drop %** value, or **if there's no enemy present in that event, 25% to just simply be laying around.** Examples for both cases can be seen below:





5. Gameplay

The gameplay is very simple. At its core, the following are needed:

1. Game loop, presents an **event** until the game has ended.
2. **Event** can have **enemies** or **inventories**, and the player is asked to overcome or use them.
3. If the max number of specified events have been processed, a new option to fight the **Dark Lord** will be shown.
4. If an **enemy** is there, player must defeat it somehow. Player will use their weapon to attack the **enemy** and proper damage calculations should be made.
5. If an **enemy** is killed by the player, ie. health of enemy drops to zero or below, player is then offered the option to proceed to next event.
6. If an **inventory** is there the player is then asked to pick it up or leave it. You should take the player's backpack space into consideration for space.
7. Repeat until the max amount of **events** have been processed, or the player has defeated the dungeon boss.

In order for this to be testable easily, use the "RNG.h" file for random number generation. This will create a random number generator that you can use and seed with a fixed number, so you can repeat your tests easily. The header file contains documentation on how you can use this class to generate random numbers.

SAMPLE RUN

You can find the entire sample run for a 100 turn game of which only 60 of is provided under the "sample_run.txt" file. Please check it and make sure to follow it as closely as possible.

GRADING POLICY:

Item	Marks (Total 20)
Actor Class	1
Player Class	3
Enemy Class	2
Inventory Class	3
Event Class	3
Game Logic	8

SUBMISSION RULES:

- You need to group your header and source files in a zip together with the Visual Studio Project files.
- This is a **group** project of at most 2 students! Please make sure both participants have put their names as comments in "Names.h"
- If you're working as a group, put comments to methods and classes to show who made which parts. Both members are still responsible from the whole project and are expected to know how each piece of code works!
- You need to submit this file to ODTU Class and only one submission per group!
- Make sure to properly use ***const*** and **virtual** where applicable!
- The structure of your code will also affect your grade! How good the design is, the formatting, memory management all matter.
- You are free to use any C++ feature you like, as long as you use it properly!

REFERENCES

[1]. Weighted Random Selection.
https://medium.com/@peterkellyonline/weighted-random-selection-3ff222917eb6