C++ project plan - Dungeon crawler

Project scope & features:

*Procedurally generated levels with randomized order of rooms. Room shapes will be pre-made, but the randomization of enemy and trap types inside the rooms and rotating and mirroring the rooms will keep them fresh.
*Permadeath and increasing difficulty as the player clears levels.
*Win condition – reach and clear the last level. The amount of levels in a game will depend on the results of playtesting and the final amount of different trap and enemy types.
*Turn-based gameplay where the player moves first, followed by allied characters and then hostiles.
*Tile-based movement of the player and enemies from a top-down perspective.
*The player character is a cube, and movement from a tile to another rolls the cube.
*Different sides of the cube are the inventory slots of the player.
*The item housed on the side of the cube that is left visible after moving will be automatically used if the conditions for using it apply, such as an enemy being in the attack range of a weapon. This promotes thought out approach to combat and makes every room like a small randomly generated tactical puzzle.
*The player can freely assign owned items on the sides of the cube they want and drop unwanted ones. Swapping items between slots is also possible. Inventory (the cube surface layout of items) can’t be accessed during combat.
*Enemies will look like cubes but will follow a simple pathfinding and set item usage pattern unique to their type instead of rolling from one side to another like the player. For example, a basic early-game enemy moves a single tile towards the player every turn and uses a sword on every other turn. Most enemies will have a simple pathfinding directly for the player, but there will be some enemies that will prefer to keep their distance or change their pathfinding based on the situation on the game board.
*Levels will contain the same amount of pre-set loot types between playthroughs, but the loot will be partially randomized. Each level will have loot of pre-set types, such as potions, weapons or spells, but the items inside those types are randomized. For example, a player will always find a melee weapon on the first level, but the weapon can be anything from a sword to a mace or a spear.
*The game is played with WASD / Arrow keys and mouse controls. Full mouse gameplay is an additional programming objective.
*Items can be picked up by moving over them, and they will be placed in the first free inventory slot available and will be on a short cooldown before they can be used after having been picked. Dropped items will have a short cooldown before becoming pickable again. If the player has full inventory, items on the floor are ignored. Several items can’t exist in the same tile, and dropping an item on top of another one is not allowed.
*After having been used, some items will have cooldowns of a few turns until they can be used again.
*Shops where it will be possible to purchase and upgrade some items (additional objective).
*No line of sight. Players can see everything in their current room. Enemies will be able to see the player after they step into the room until defeated.
*Various traps in some rooms. Some will be periodically triggering, others will trigger only once.
*Automated detection for softlocks (e.g. the player can’t move due to being completely surrounded by tiles that can’t be moved to due to enemies or obstacles occupying them).
*If the player enters a room with enemies, they will be locked in the room until all enemies have been defeated or the player has perished.
*Enemy, item and trap description windows that pop up when the cursor is hovered over these things on the screen.
*Music, sound effects and self-made pixel art graphics.

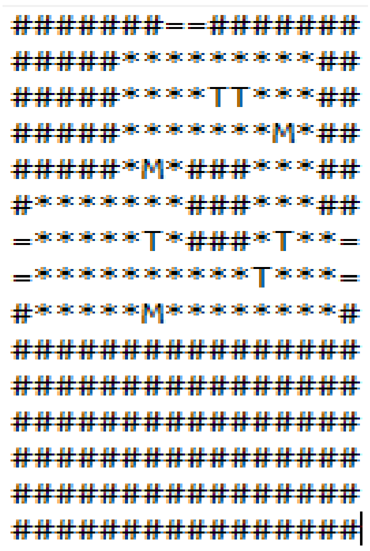
High-level structure and main classes:

Main file functions:

Main – Opens the main menu window for the game, and when a game run is started it holds a loop that tracks keypresses from the player and keeps the game going until the player character's death or victory. The main function calls other functions and classes appropriately based on player keypresses.

Other files:

RoomStorage – Holds the different room shapes for dungeon generation. Rooms are stored as ASCII art of 16 x 16 grid with different characters representing enemy spawn locations, trap locations, walls and the like, which will then be translated to level features and graphics when the file is read. Rooms will also be able to be rotated into any orientation and mirrored by the level generation algorithm to create more variety of a single pre-set room shape. In the example picture of what a room could look like 'M's represent monster spawn locations, 'T's represent trap locations if the level is generated on a floor that is deep enough for traps to spawn, '*'s represent ordinary floor tiles, '#'s represent walls and '='s represent doorways. These symbols may change as the project proceeds.
Rooms inside RoomStorage will be divided into five different room types based on the door amount and placement of a room.



```
#######==#######
#####*********##
#####****TT***##
#####*******M*##
#####*M*###***##
#*******###***##
=*****T*###*T**=
=**********T***=
#*****M********#
################
################
################
################
################
################
```

GraphicsStorage – Holds the sprites for the visual representation of the game.

SoundStorage – Holds the game sound effects and music.

Main classes:

Game – Holds information about the current level, the player and some statistics such as amount of killed enemies and turns taken, and passes the values to the graphics department for visual presentation.

Player – Holds information about the status of the player such as inventory and health points. The orientation of the player cube character and the inventory are stored in a vector of six InventorySlot objects, which will be moved inside the vector to change the orientation of the cube depending on player movement.

InventorySlot – Holds an instance of the Item class in it, empty by default. The item in an InventorySlot is used if it isn't on cooldown, the InventorySlot is on the top position of the player cube character and the conditions for using that particular item are met, for example an enemy is in range of a weapon or an enemy is attacking the player while a shield is on the top of the cube.

Item – An abstract class that all items inherit.

*Subclasses: various abstract classes for different item types such as melee & ranged weapons, shields, spells and potions, which will then be inherited by item classes inside those types. For example, a sword inherits the melee weapon class, which inherits the item class.

Character – An abstract class that all characters on the screen inherit.

*Subclasses: PlayerCube, Enemy, Ally.

PlayerCube – Representation of the cube nature of the player character on the game grid.

Enemy – An abstract class that all classes for hostile entities inherit.

*Subclasses: various enemies.

Ally – An abstract class that all classes for allied entities inherit.

*Subclasses: various allied entities that are spawned by some items.

Tile – A class that holds information about the contents of a tile on the game board, such as empty floor tiles, doorways, walls, traps and pits.

Room – A class that represents a room consisting of various tiles. Rooms will randomly select their layout from the aforementioned RoomStorage file, and will be connected to other rooms by doorways. When the player steps into a room for the first time, the room spawns enemies on enemy spawning tiles and locks all doorways until the player has defeated all of the enemies.

Level – A class that holds and connects nine rooms together on a 3x3 grid to create levels. Only one level exists at a time, and when a level is completed it is deleted and a new one is created.

MainMenuUI – Displays the main menu and allows the player to change settings and start a new game.

GameUI – Displays the status of the game, all tiles and the player inventory as a graphical window.

Settings - holds settings like volumes for SFX and music. (Optional objective)

Planned use of external libraries:

SFML will be used for implementing the graphics-related classes and functions of the project, as well as implementing game sounds.

Division of work and responsibilites between the group:

Elias Viro - Game and UI design, designing the main game mechanics, enemies, items, rooms and project structure, balancing the game, creating pixel art sprites and picking music and sound effects for the game and guiding the team to follow the vision. Various programming tasks.

Atte Aarnio – Implementation of the graphics and sounds, as well as interfaces with the graphics and sound libraries.

Selin Taskin – Programming enemies, items and other things.

Binh Nguyen – Various programming tasks, creating the level generation and room mirroring&orientation randomization algorithm.

Everyone – Testing, bugfixing, writing documentation.

Planned schedule and milestones before the final deadline of the project:

Weekly schedule will be flexible depending on the speed of progress.

Week 44: Laying the groundwork for further programming, such as implementing basic main function classes and creating the necessary project files.

Week 45: Implementing the dungeon and room generation algorithms and implementing a rollable player cube character and interaction with mouse and keyboard. Adding some basic rooms, melee weapons and melee type enemies with a basic pathfinding for testing purposes and ways for the program to access them from the files.

Week 46: Implementing functional and interactive player inventory and pathfinding algorithm for enemies. Adding more enemies, items (such as ranged weapons, potions and spells) and rooms, as well as traps and placeholder graphics for the game UI and entities on the game board.

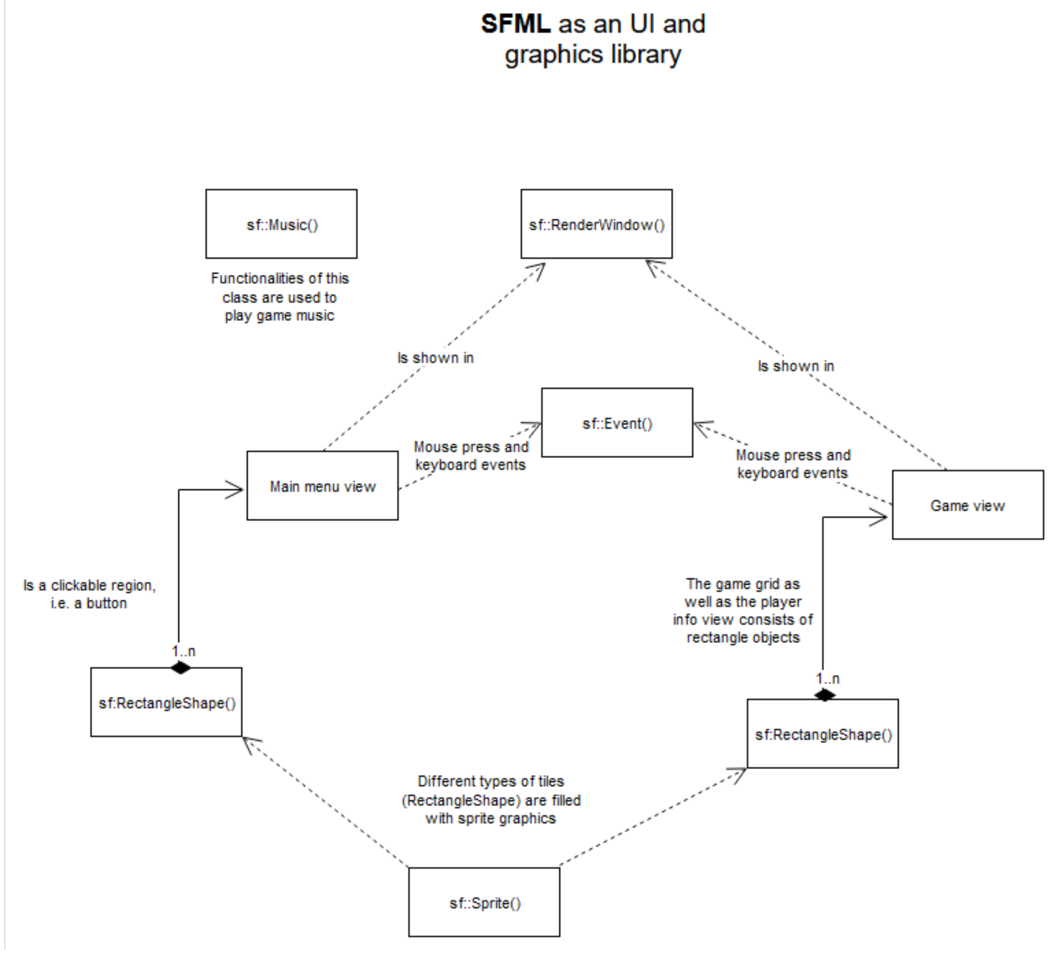
Week 47: Adding shopkeepers and more enemies, traps and items. Inspect functionality for enemies, items and tiles. Settings window for changing sound and music volume (optional). Improving the graphics by creating proper pixel art sprites for the game, including the main menu and settings.

Week 48: Adding more variety to the game with more room shapes, enemies and items. Playtesting and balancing the game to be a pleasant experience. Winning condition. Map of the current level.

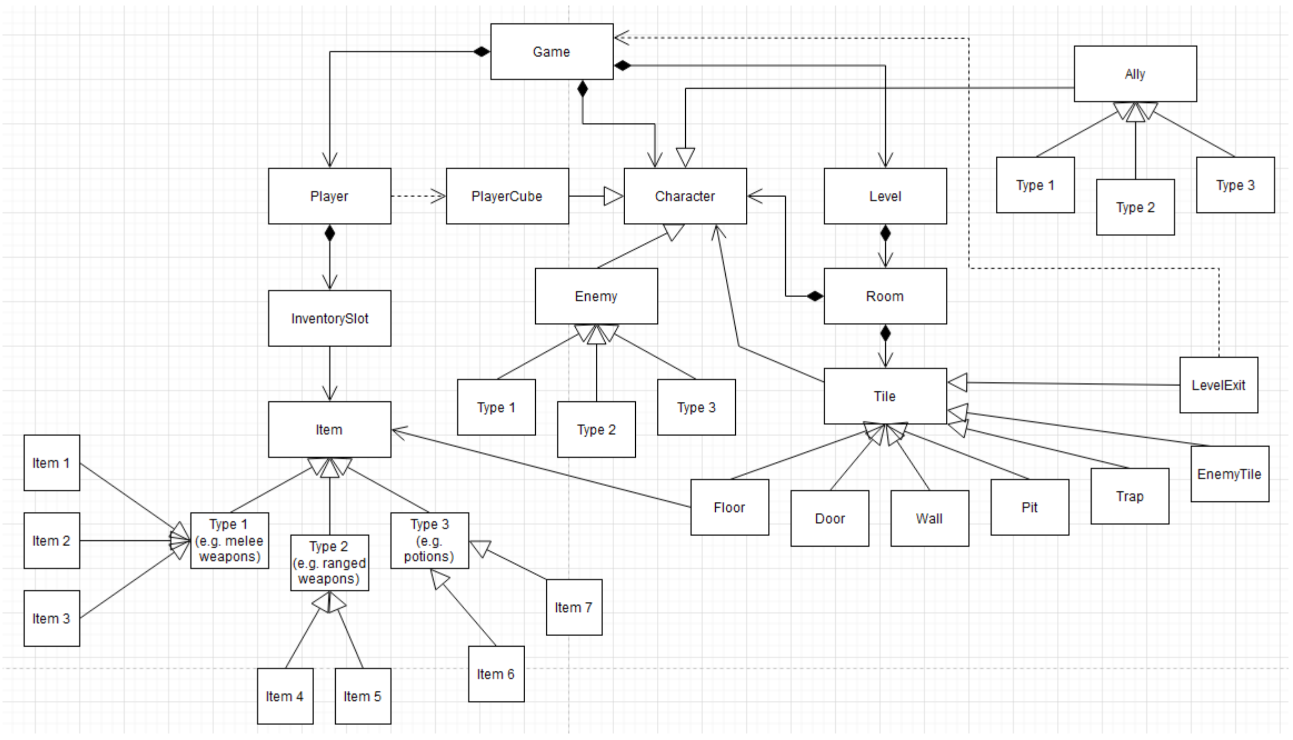
Week 49: Playtesting, balancing and bugfixing, getting the game to work without any bugs. Quality of life features if needed. Final polish. Finishing the documentation.

Project diagrams:

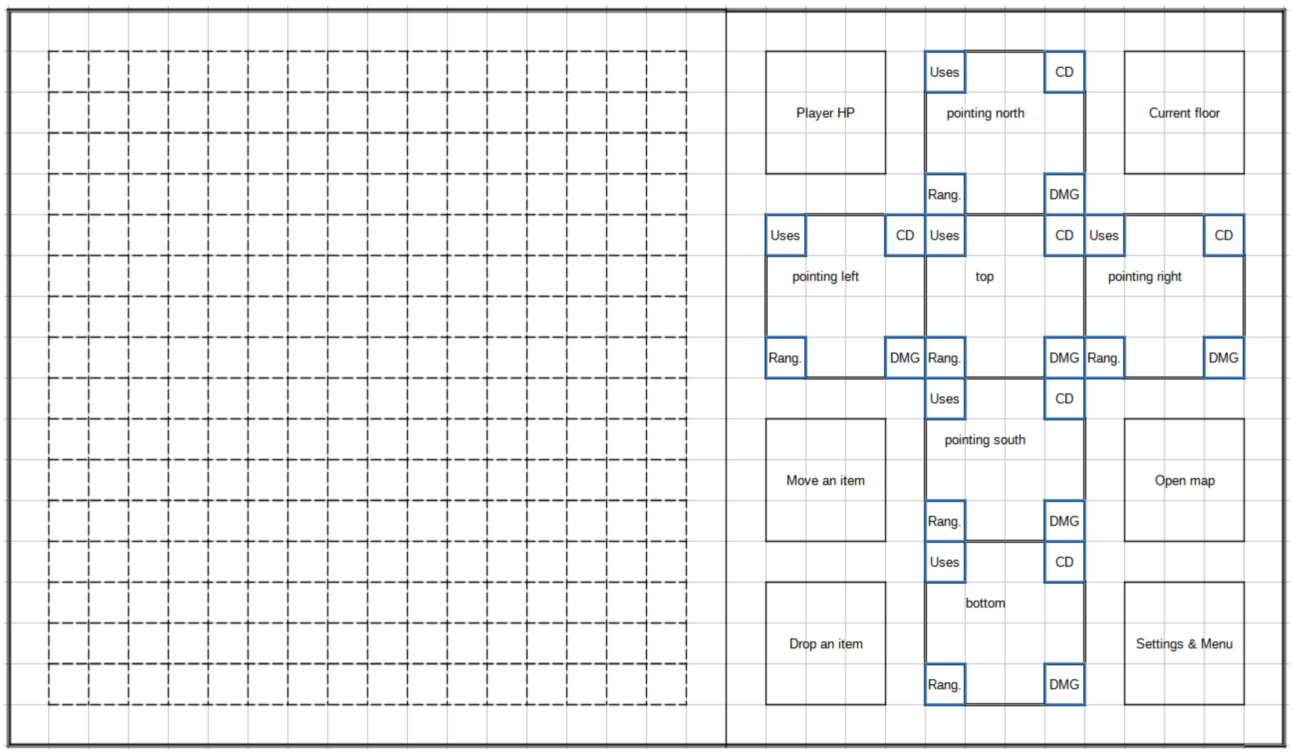
Graphics:


SFML as an UI and graphics library
sf::Music()
Functionalities of this class are used to play game music
sf::RenderWindow()
Is shown in
Is shown in
sf::Event()
Mouse press and keyboard events
Mouse press and keyboard events
Main menu view
Game view
Is a clickable region, i.e. a button
1..n
sf:RectangleShape()
The game grid as well as the player info view consists of rectangle objects
1..n
sf:RectangleShape()
Different types of tiles (RectangleShape) are filled with sprite graphics
sf::Sprite()

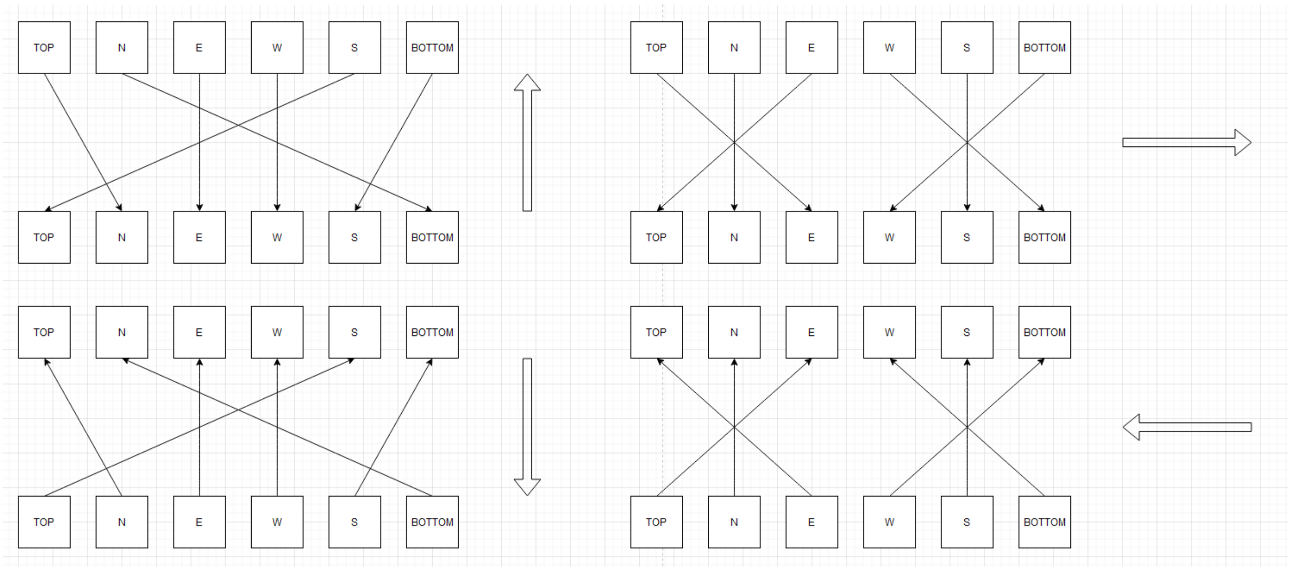
Game:



Game UI concept. The current room the player is in will be displayed on the left, and the player inventory and the orientation of the cube will be visible on the right at the same time by moving the items in the inventory slots depending on the directions where each item will end up pointing after moving. There will also be a few buttons that can be interacted with the mouse.



Concept for the mathematics behind moving the inventory slots on the surface of the player cube:


TOP
N
E
W
S
BOTTOM