ELEC A7151 Project Plan

Returned: 30.10.2023

Members:

Matias Veikkola

Henri Palin

Lauri Palin

Daniil Parniukov

Aalto-yliopisto

ELEC-A7151 Object oriented programming with C++

Table of Contents

1. Scope of work

1.1 Description of the game

We are creating a classic rogue-like game. The game is going to be real-time. Since we are not sure that we'll be able to create proper 3d graphics, we think the game is going to have top-down 2d graphics (example: Hotline miami). The game advancing is simple: player descends deeper into the dungeon by finding stairs (or elevators, or something similar). The game ends with the player's win if they reach n-th level and with player's loss when they die (hp equals 0). The movement and controls are intuitive: 4 possible directions for movement controlled with wasd keys. Some interactions will be made using keyboard (e.g. "E to interact"), some - with mouse (e.g. aiming and attack). In the dungeons player can find some items to help them later, like potions or better weapons. We think there will probably be two main types of weapons (and combat) - melee and firearms. There will be several classes of monsters and at least one boss (possibly several).

1.2 Basic features

- Scope of the work: what features and functionalities will be implemented, how is the program used, and how does it work
    - Simple 2D graphics
    - Moving through corridors and rooms.
    - Combat between the player and different types of monsters
    - Collectibles which can be used later (weapons, potions, etc.)
    - Some sort of progression (winning & losing conditions)

1.3 Additional features

If we have time left after implementing basic functionality (we estimate we will), then there is a list of additional features to be implemented:

- **Minimap**, where player can see their location relative to the current floor.
- **Sound effects** - game sounds and possibly music
- **Randomly generated dungeons**, so that the game is different every time

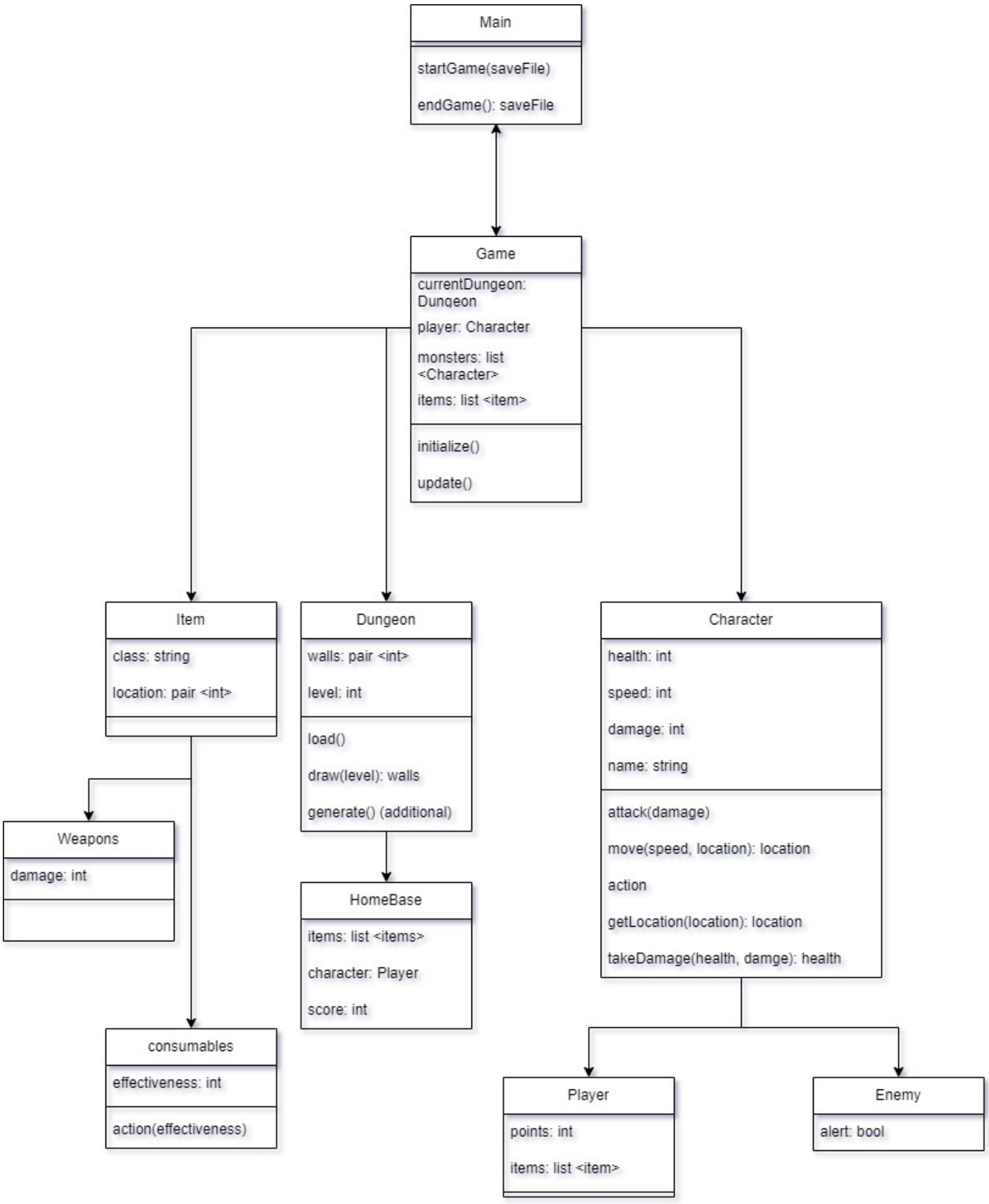
Those three features are our highest priority. We think without them game looks a bit pale.Next features are more advanced or not that essential. They include:

- **Home base**, where player rests between dungeon raids
- **Character leveling**, new skills and abilities

- **Ability to craft / modify items**, self explanatory, but probably done after home base (the base will be the place for crafting).

The list is not exhaustive, maybe there will be some other additional features that we would like to implement.

2 UML


Main
startGame(saveFile)
endGame(): saveFile
Game
currentDungeon: Dungeon
player: Character
monsters: list <Character>
items: list <item>
initialize()
update()
Item
class: string
location: pair <int>
Dungeon
walls: pair <int>
level: int
load()
draw(level): walls
generate() (additional)
Character
health: int
speed: int
damage: int
name: string
attack(damage)
move(speed, location): location
action
getLocation(location): location
takeDamage(health, damge): health
Weapons
damage: int
HomeBase
items: list <items>
character: Player
score: int
consumables
effectiveness: int
action(effectiveness)
Player
points: int
items: list <item>
Enemy
alert: bool

3.Description of classes

- Main
  - The main class of the application. Starts the execution of the game.
- Game
  - Coordinates the overall game flow, including loading levels, handling game states, and managing transitions between different parts of the game.
- Item
  - Parent class for the collectible items in the game. Contains the name and the location where its found on the map.
- Weapons
  - A subclass of Item, representing weapons that characters can use to attack. Contains properties like damage, range, and attack speed.
- Consumables
  - A subclass of Item, representing consumable items that provide various temporary or permanent benefits when used. Examples include health and damage potions .
- Dungeon
  - Contains the functionality for the dungeons of the game. Is used to generate and draw the play area.
- HomeBase
  - Represents a safe location within the game, such as a home base or hub area where the player can rest, craft or perform upgrades.
- Character
  - Contains the common logic that apply to all characters. Every character has for example health, speed, damage and name.
- Player
  - Extends the Character class, but includes functionality specific to the character. Keeps track of the inventory of the player.
- Enemy
  - Extends the Character class and includes functionality specific to enemies. Manages enemy behaviors, such as AI, pathfinding and combat abilities.

4.External libraries

4.1 SFML

SFML (Simple and Fast Multimedia Library) is a C++ library that provides a way to create 2D games, multimedia applications and graphical user interfaces. It offers modules useful for the project in handling graphics, graphical windows, audio, and networking. SFML is going to be the main external library for graphics, user interface, and audio.

Graphics: SFML provides modules to handle graphics to create and render game elements.

- sf::RenderWindow
- Sf::Sprite
- sf::Texture

User Inputs: SFML allows the program to handle user input events for both keyboard and mouse inputs.

- sf::Keyboard
- sf::Mouse

4.2 Box2D

If the project requires the implementation of a physics engine, we will utilize Box2D.

Box2D is a C++ physics engine that can be used to simulate 2D physics in games and simulations. While it is more commonly used for handling the physics simulation of objects, it can be integrated into a game engine like SFML to provide physics to a dungeon crawler game.

5. Division of work and responsibilities between the group members

- **Henri**, as the group member with the most experience, researches how the game should be structured and implements core features. Core features may include such parts of the program as the event loop, type hierarchy and user interface.
- **Daniel** handles character related features. Those features include player and monster class.
- **Matias** handles dungeon related features, such as dungeon loading / generation (later, when dungeons are randomly generated), filling the dungeons with monsters, transition between floors and so on.
- **Lauri** does graphics for the game and writes the code for collectibles and items (for example, code for potions, weapons, etc.)

6. Description of features

- Simple 2D graphics:
    - Using SFML to render a window that draws the the world, the characters and other world assets
    - This also includes an UI, which is controlled using mouse. The game will have a start menu, a pause menu, and maybe options?

- Moving through corridors and rooms:
    - The player moves through a series of interconnected rooms and corridors. These rooms can contain monsters, items, traps, and other interactive elements.
        - Create a data structure (i.e.2D array) to represent the dungeon map. Define room layouts, corridors, and connections between rooms.
        - Implement a system to handle player movement. The player can move in cardinal directions (up, down, left, right) or to stairs or ladders to access different levels of the dungeon.

        - Use collision detection to ensure the player does not walk through walls.
        - Implement algorithms for room generation and placement of obstacles and items.
- Combat between the player and different types of monsters: Henri
    - When the player presses mouse button, (or some keyboard key) the player attacks
        - The attack which is executed depends on what type of a weapon the player has currently equipped.
            - Sword -> the player swings the sword towards the enemy. If the sword hits, then the enemy takes damage
            - Bow (did we talk about this?) the player shoots an arrow towards the enemy
- Collectibles which can be used later (weapons, potions, etc.):
    - The player can pick up or otherwise receive items which then get added to their inventory.
        - Create a system that generates a random assortment of collectibles in each randomly generated dungeon.
        - When consumables are used, they are removed from inventory.
- Some sort of progression (winning & losing conditions): Henri
    - Winning condition
        - The player proceeds through enough of the dungeons. In other words the player makes its way to the end of the map. Then a winning screen/text is shown.
    - Losing condition
        - When the player has lost all of its health, the game is lost and a losing screen is displayed.
- Mini-map:
    - There is a mini-map display on one of the screen's corners.
    - The mini-map highlights important things, for example enemies as red dots.
    - 1st idea: This mini-map shows a much larger area of the dungeon than the normal view.
        - The mini-map follows the player as the center of the mini-map.
    - 2nd idea: The mini-map starts as a black screen and its view doesn't move. As the player enters a room or a corridor, that area gets added to the mini-map.
        - The mini-map covers the entire play area.

- Sounds:
    - Play sounds accordingly when notable actions occur in the game for a more immersive experience. For example, hit sound, item pick-up sound, death sound, winning and losing sound.
- Home base:
    - The home base serves as a safe place for the player, allowing them to rest, store items, and prepare for future dungeon playthroughs. It may also include NPCs for interaction.
        - Create an inventory system to allow players to manage their items.
        - Provide a user interface to allow players to access home base features.
        - (optional) Implement interactions with NPCs or vendors who can provide information, quests, or item trading.
- Randomly generated dungeons:
    - Randomly generated dungeons provide replayability and variety to the game.
        - Implement an algorythm to create random dungeon layouts.
        - Ensure that generated dungeons are logically connected.
        - Define criteria for difficulty level that influence dungeon generation
- Character leveling (skills & abilities): Henri
    - Player can level up when it has killed x amount of enemies
        - Leveling up makes the player stronger (more hp or more damage)
    - The player can gather abilities, which can be used against the enemies
- Ability to craft / modify items:
    - In the inventory tab there are two item input slots and one item output slot. When two compatible items are placed in their respective item input slots, the crafted or modified output appears on the output slot.
        - Clicking on the output item removes the two input items and adds the output item to the inventory.

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

- **Project plan submission (commit) to git (deadline):** Tuesday, October 31, 2023 at 23:59
- **Project plan review with your advisor:** Week 44 (starting Monday, October 30)
- **Week 45-46**: The initial core functionality of the game is done, meaning that there is a “game loop” of some sorts and a window that can draw some basic game assets. Started progress on dungeons, characters and items.
- **Week 46-47**: Initial dungeon character and item implementations
- **Week 48**: Base game done. Start implementing additional features until the deadline.
- **Project demo to your advisor:** Weeks 49 and 50 (starting Monday, December 4)
- **Weeks 49 and 50** Documentation and finalizations
- **Project final commit to git (deadline):** Sunday, December 10, 2023 at 23:59
- **Project peer evaluation (deadline):** Friday, December 15, 2023 at 23:59