ELEC A7151 Project Documentation

Dungeon Crawler

Returned: 10.12.2023

Members:

Matias Veikkola

Henri Palin

Lauri Palin

Daniil Parniukov

Aalto-yliopisto ELEC-A7151 Object oriented programming with C++

Table of contents

Overview

This software is a basic dungeon crawler video game. It's programmed in C++ and uses SFML to draw the game.
Before the game starts, the software runs a series of tests which test the basic functionality of dungeon generation, creature creation and inventory handling. After all of the tests pass, the actual game starts.
The software uses SFML to draw a window and loads all used texture files from a set directory. A randomly generated dungeon is then created, consisting of rooms and corridors that connect them. The rooms' floors have a random distribution of different textures and the corridors have their own textures as well.
The player is spawned in the first room and its inventory is created. Currently it consists of a sword and collectible potions. The player can move with keyboard input, attack with mouse clicks and equip the sword or the potion(s) with key binds to their respective numbers to use them. Potions have a chance to spawn randomly in each room and they can be picked up and added to the inventory. Using potions restores health.
A single basic enemy is also spawned to the first room, and the player needs to kill the enemy to gain access to the corridor. Killing all of the enemies in each room is the basic game loop. The enemies can attack the player. If the player is hit, their health gets lower. If their health reaches zero, the game is lost. In this scenario, a losing screen is shown and there is a "Play again" button.
Clearing rooms of enemies becomes consecutively harder as there are more and more enemies to fight as the player progresses.
Finally, the last room contains only one enemy, the boss. The boss is much stronger than basic enemies and has its own texture. Killing the boss is the game's winning condition, where a congratulatory screen is shown. There is a "Play again" button.

Software structure

https://drive.google.com/file/d/1RYKd8ZrZ2QbKDOCPUWvmsN_6KGcSTBlT/view?usp=sharing

Instructions for building and using the software

Detailed compilation, building and running instructions can also be found from build/README.md of the git repository https://version.aalto.fi/gitlab/palinh1/cpp-course-project-repository-template/-/tree/master/build?ref_type=heads

How to compile the program

1. On your command prompt/terminal, clone the repository with
    a. git clone git@version.aalto.fi:palinh1/cpp-course-project-repository-template.git
2. Navigate to the root of the cloned git repository with
    a. cd cpp-course-project-repository-template
3. Before building the game SFML needs to be added as a submodule with the following command
    a. git submodule update --init --recursive –remote
4. SFML also needs some dependencies for it to compile. Download the required dependencies with the following apt-get command
    a. sudo apt-get install libgl1-mesa-dev libfreetype6-dev libopenal-dev libjpeg-dev libpng-dev libtiff5-dev libflac-dev libvorbis-dev libsfml-dev libudev-dev
    b. If you encounter problems with running the get command there are two additional commands you can try
        i. sudo hwclock --hctosys
        ii. sudo apt install pkg-config
    c. Now try running the initial apt-get command again
5. Compile, build and run the game
    a. Navigate to the build folder
        i. cd build
    b. Compile and build the application with the following commands
        i. cmake ..
        ii. make
    c. Then, you can run the application with the command
        i. ./DungeonCrawler

How to use the software

1. The player moves with “WASD” controls and looks to the mouse’s direction.
2. If the player is close to enemies, clicking “mouse1” deals damage to them.
3. The sword can be equipped by pressing “1” and potion by pressing “2”. A red circle indicates which item is equipped. The sword is equipped by default.

4. Pressing “e” while potion is equipped uses the potion and restores health. Potions can be picked up by pressing “f” near them.
5. After winning or losing, there is a button for playing again. Pressing “escape” closes the program.

Testing

The tests test the generation of a dungeon with multiple rooms and corridors.
Tests Conducted:
Checks if the correct number of rooms is generated.
Verifies if the correct number of corridors is generated based on the number of rooms.
Initializes a test window and a test creature.
Validates the initial position of the test creature.
Adds health potions to the test creature's inventory and verifies the inventory's size.
Purpose: Ensure the game and functions creates the expected number of rooms and corridors while checking basic functionality related to creature initialization and inventory handling.
The tests are done before actually running the game and will print out if any test is failed. If any test is fails it will also print out which tests failed.

Work log:

Description of division of work and everyone's responsibilities

1. Henri: Initial building instructions with cmake. Implementation of the game class, which is the central integration point for all of the other classes. Player and camera movement, user interface implementation. Keyboard and mouse input handling. Code reviews, assisting other team members with their tasks. Documentation.
2. Lauri: Creating all of the textures. Texture implementation in room generation. Fixed issues related to drawing and creature code. Performance improvements and game balance. Testing.
3. Matias: All code in generating the dungeon including random generation, tile system for placing textures and placement and coordinates for rooms, corridors and tiles. Initial room and inventory code and assisted in inventory rendering.
4. Daniil: Code related to behavior of all characters (player and monsters). Added some minor functions to other classes that were needed for character behavior. Refactored code in the end (added constants, splitted functions into declarations and implementations). Documented my part of code.

Weeks:

1. Week
    1. inital contact of the group.

2. Created project repository, teams for sharing files and work, and telegram channel for quick communication.
3. Project plan.
4. Henri: Setup CMake for the project, start with the core game logic
    i. Hours [4]
5. Matias: Set up basic start window, start coding the dungeon class
    i. Hours [4]
6. Daniel: Create basic skeleton for character class
    i. Hours [4]
7. Lauri: Setup structure for item class
    i. Hours [5]

2. Week
    8. Project setup, basic classes' functionality
    9. Henri: Setup submodules for the project, started implementing the game class
        i. Hours [10]
    10. Matias: Dungeon class. Started on random dungeon generation and tile system for dungeon rooms and corridors.
        i. Hours [15]
    11. Lauri: Basic Item class
        i. Hours [3]
    12. Daniil: Basic functionality for Creature class
        i. Hours [8]
3. Week
    13. Got everything ready to combine into a project
    14. Henri: initial game class implemented, code reviews
        i. Hours [8]
    15. Matias: Dungeon class. Fixed bugs with the dungeon generation, merged a basic version of the class to master, started on connecting rooms with two-sectioned corridors
        i. Hours [10]
    16. Lauri: Map textures
        i. Hours [10]
    17. Daniil: Helper functions for vector work
        i. Hours [5.5]
4. Week
    18. Henri: keyboard inputs, initial ui code
        i. Hours [8]
    19. Matias: Dungeon class. Reimplemented dungeon generation. Randomly generated dungeon consisting of rooms connected by corridors.
        i. Hours [22]
    20. Lauri: Graphics for player, enemy and a potion. Started working on code for adding potion to rooms.
        i. Hours [12]
    21. Daniil: Start of inroom character interaction.
        i. Hours [8]
5. Week

1. Henri: keyboard inputs, initial ui code
    i. Hours [8]
2. Matias: Dungeon class. Reimplemented dungeon generation. Randomly generated dungeon consisting of rooms connected by corridors.
    i. Hours [13]
3. Lauri: Optimised texture loading.
    i. Hours [15]
4. Daniil: Finished inroom character interaction without attack
    i. Hours [11]

6. Week
    1. Add doxygen to all code
    2. Matias: Sprites of the items in the top left corner of view. Top left of the sprite of item add info about it i.e. amount for potions and attack speed and damage for weapon. Added tests.
        i. Hours [7]
    3. Henri: Shift to run. Game start and end screen
        i. Hours [10]
    4. Lauri: Textures for another enemy, boss, inventory, items etc. Merged big code changes for creatures.
        i. Hours [13]
    5. Daniil: Transition between rooms. Boss in the last room. Attack. Accidentally breaking the inventory class and fixing it after myself.
        i. Hours [13]
7. Week
    1. Henri: Adjust window size. Winning and losing screen.
    2. Matias: Visible player and monster health bars. Indicator for what item is in use. More tests. Documentation.
        i. Hours [10]
    3. Lauri: Code reviews. Balancing. Textures for boss and HUD. Documentation
        i. Hours [15]
    4. Daniel: Item code (health potion and a sword). Item spawning and collecting code. Attack now depend on the item used. Refactoring the code, documenting Creature, Monster, Player, Inventory, Item classes. Adding constants.
        i. Hours [15]