Jake Brown (Leader)
Siming Chen
Jiang Jiang
Michael Lee
Brandon Vickrey

Design Document - Dungeon Drive (D:)

Purpose

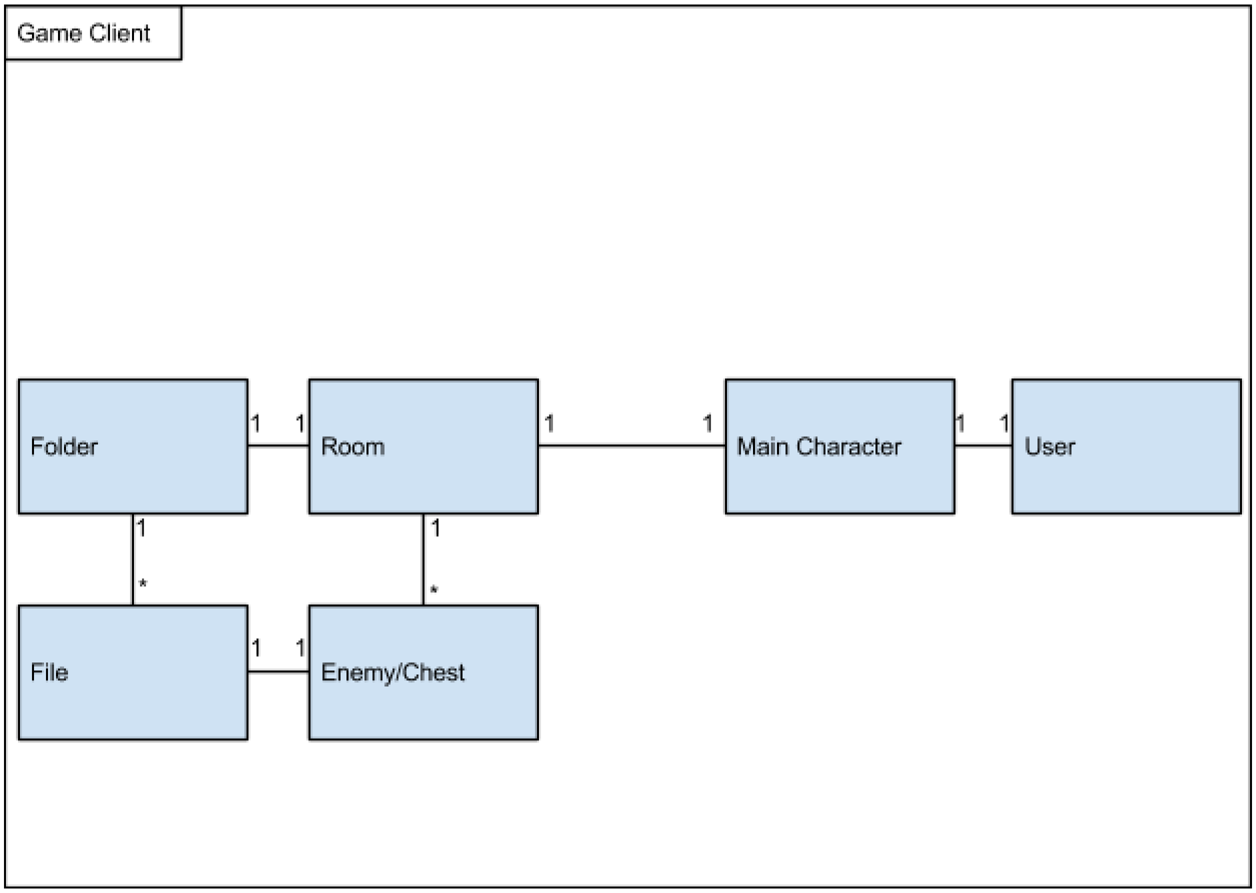
People need new and exciting ways to pass the time. We want to create a unique video game experience to do that by using the player's file system to generate the game world.

<u>Summary List of Functional Requirements</u>

● Interface to start the game that allows creating a new game or loading a previous game
● Automatically generated dynamic 2D maps based on the files and folders in user's filesystem.
● Use character 'w', 's', 'd', 'a' to move a character around the generated world.
● Combat system to fight enemies with varying types of artificial intelligence.
● Loot that can be found throughout the game.
● A pause menu where users can save and exit
● A "game over" screen if the player dies

Design Outline

(a) Outline your design decisions (for example client-server model), identify the components of your system, and describe the purpose of each component.



Client

1. Client will display the game, where users will be able to interact and receive information
2. Main features to be displayed include individual maps, enemies, the main character, etc.
3. Scans for multitude of files/folders through the hard drive of the user and uses that information to generate game assets (maps, enemies, items, etc.).

(b) Describe the interactions between individual system components.

System Components Interaction

1. A single user represents the main character displayed in the client. The user may interact with the game environment through this avatar, including movement, combat, inventory usage, etc.
2. There is only one main character for the user to control, and one room for the main character to explore and interact with at a time.
3. Each room is navigable and represented by a folder on the user's hard drive. A folder may contain multiple files, which may be represented as enemies, chests, or structures for the user to interact with.
4. As there can be multiple files in a folder, there can be multiple enemies/chests/structures in a single room.

(c) Include at least one UML diagram that clearly shows high-level structure of your system.

See Design Details

Design Issues

1. Efficiency

    When user input a large folder (i.e. a folder contains too many files and sub folders), the game will generate a high volume of content accordingly. This may result in a series of unexpected issues (rendering for instance).

    Solution:

    Option 1: Instead of loading all the files at once, we will scan the files on a floor-by-floor basis. For instance, the program will not scan anything other than the current floor-corresponding directory.

    Option 2: Instead of loading the current directory immediately upon start, the game will load to a certain capacity, and keep loading as the game character goes along.

2. Language to use

    Option 1: C#

    Option 2: Java

    Solution:

    We chose Option 1. Since our program is a game, which will use a lot of the visual content. C# provides a brush feature which make this process a lot easier. Also C# has a similar syntax with C, which is well known by most of our group members, therefore we chose C# instead of Java.

3. Difficulty

    Option 1: Translate all files into their corresponding game content, even if an unfair number of enemies appear.

    Option 2: Put a cap on the number of translated files to insure that gameplay is fair.

    Solution:

    We chose Option 2. This is because it allows for the game to be more robust in handling a larger variety of file systems. If we went with option 1, then a particular file system could be unusably difficult. By adding the cap, we will have more control over the game’s progression and can allow for a more fair and consistent experience.

4. Handle exceptions
   When the user modify (i.e delete, rename a file) a corresponding file while the game is running.
   Option 1: We can insert a file protection mechanism to prevent the folder from being changed.
   Option 2: We can set up an exception that whenever a file is being deleted, a easter egg would be spawned.
   Option 3: We re-load the entire dungeon floor every time any changes if made.

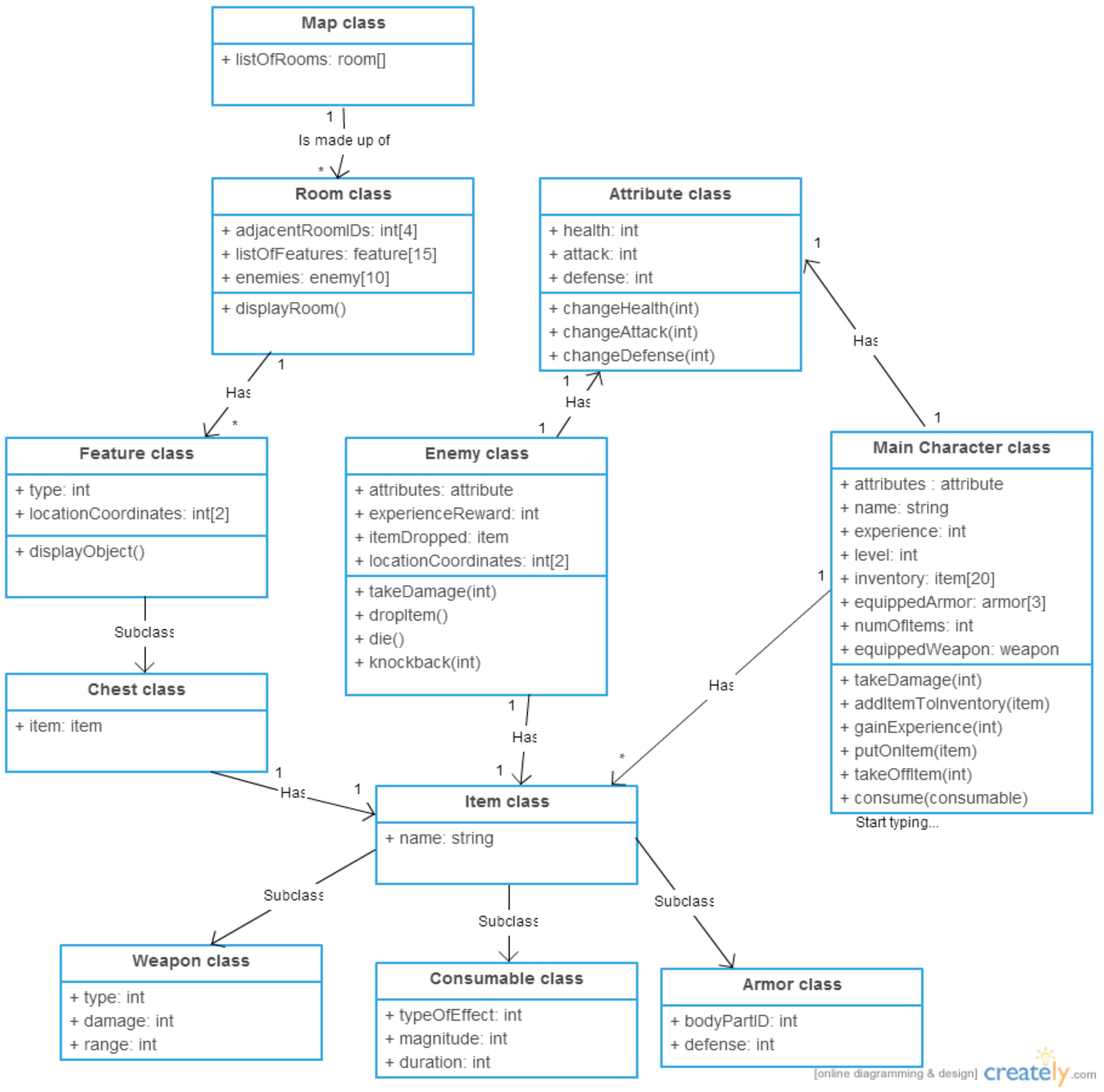
   Solution:
   We choose option 2, because it offers the most flexibility and makes the game more fun. Option 3 would slow down the entire game performance, as we have to re-load each floor constantly.

5. Progression
   When the user plays the game for a long time, the game will get bored eventually.
   Option 1: We can have a leveling-up system, the more floor/enemy the main hero has explored, the more powerful it become.
   Option 2: We will allow the user the save their progress each time they log off the game.

   Solution:
   We combine both option1 and option2, so the user will be able to have a continuous adventure with increase amount of content each time.

Design Details

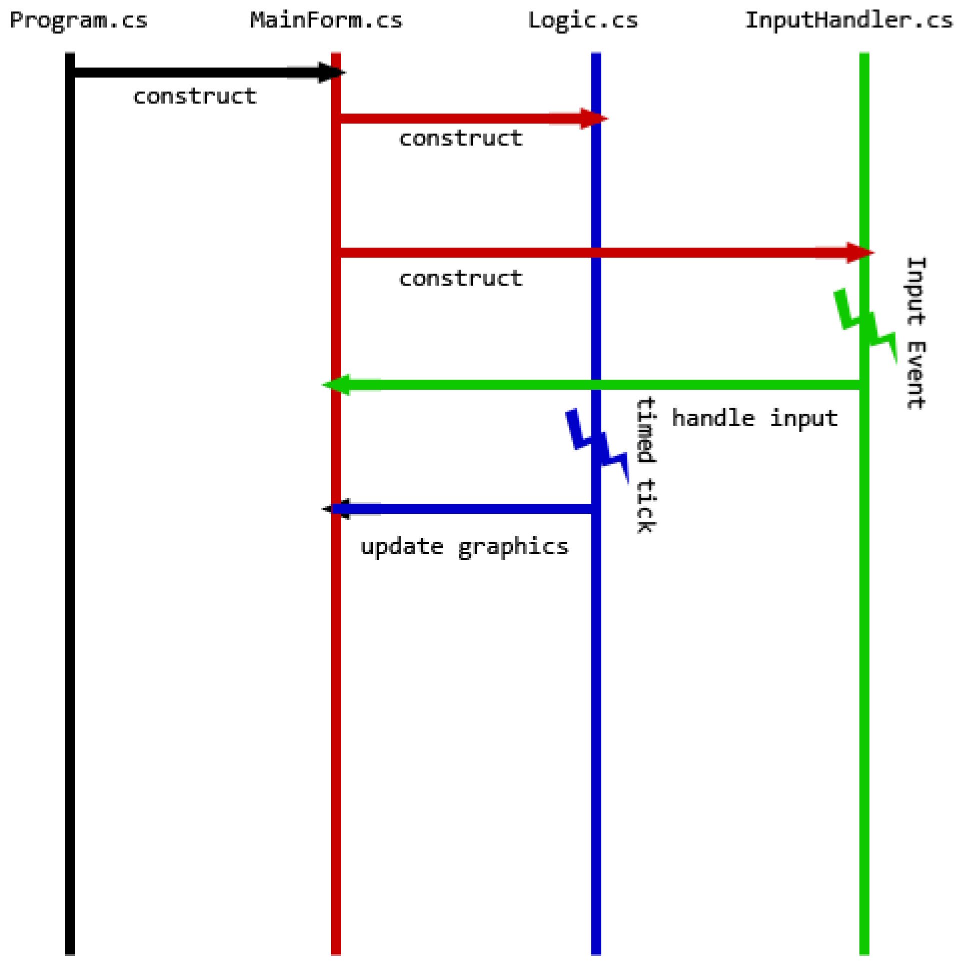
(a) Include class level design of the system (i.e. class diagrams) and be as detailed as you can.



(b) Describe the classes and interactions between the classes.

Program.cs: This is where main() is and only created MainForm.cs
MainForm.cs: This is where the game is being drawn and creates the timer to run Logic.cs's tick function as well as connects InputHandler.cs' handler functions to itself.
Logic.cs: This holds the function that is run every tick then updates MainForm's graphics.
InputHandler.cs: This holds the functions that deal with user input and updates the game's variables accordingly.

(c) Add sequence diagrams for different activities in the system, which will be helpful at the later stages of your project.


Program.cs
MainForm.cs
Logic.cs
InputHandler.cs
construct
construct
construct
Input Event
handle input
timed tick
update graphics