Design Document

Abstracts

Entity

Is the base for any character, door, treasure chest or death location. It merely contains the id of the object and is purely abstract. The entities are displayed on the map.

Attributes:

- id: Integer

Character

Inherits from the Entity abstract class
This class represents either a player or an enemy. This is also just a base class and therefore completely abstract. The big difference is that `inventory` works as an inventory for players, but as the loot for enemies. Meaning that Enemy type class can't access it's inventory

Attributes:

- type: Char - A single char representing the type: B, R, W, Z, G or L
- type_name: String - Complete name
- maximum_health: Int - A number, affected by vitality
- health: Int - Starts out at max_health
- armor: Armor
- weapon: Weapon
- strength: Int
- vitality: Int
- inventory: Vector<Item> - List of Items

Methods:

- move(row: Integer, column: Integer): Int
- attack(character: Character, damage: Int): Int

Item

Can be a consumable, armor or weapon and it's exact data is stored in an external JSON from which we pull the data in the constructor to build the object. Not displayed in a room and only available in inventory

Attributes:

- abbreviation: String - Abbreviation of an item
- name: String - Name of the item
- dice: Dice - Contains an unrolled dice with the stats of the item, once the item is used, the dice is rolled and afterwards destroyed with the item.

Classes

> **Getter and Setters**
>
> Getter and Setters have been omitted from the class description and will be implemented as needed

Player

Inherits from the Character abstract class
This class represents a single player. It gets its position from the Entity class and most of its attributes from the Character class. Furthermore it now also has an ID, a name and can equip an item from its inventory. The Player will be created at the beginning of the game and controlled from there.

Attributes:

- name: String - Name of a player (max 10 chars)
- resistant_to: Enum (Fire, cold, ...)

Methods:

- printPlayer() - Prints player related data
- rollDice(amount_of_dice: Integer, dice_type: Integer): Integer

Enemy

Inherits from the Character abstract class
This class is used to create an enemy or the final boss of the dungeon. Its inventory is used as loot and therefore it can't access it. It has a weapon which is always equipped and an unlimited amount of ammunition. The enemy generates on each room and can either attack a player or move.

Attributes:

- is_boss: Bool - If the enemy is a boss enemy, aka. if its Lich

Methods:

- printEnemy() - Prints the enemy signature

Dungeon

Equivalent to the Map() class in Assignment 1. It basically contains most of the game logic, saves the active room, has a list of all the rooms and can print the map. It does NOT contain the game cycle, but most of the functions to interact with it.

Attributes

- active_room: Room - Currently active room
- room_list: Vector<Room> - List of rooms
- occurred_enemy_types: vector<Character>
- actionCount: Integer

Methods:

- printMap()
- printStory()
- getCompletedRooms()
- Static:
  - checkStoryFile(file_path: String)
  - checkDungeonFile(file_path: String)

Room

Each room has an ID, which is assigned from the start in Ascending order. The first room has an ID of 1, and proceeding in each room, the ID increases. Note that a player does not need to go from room to room in the order of its IDs, its just an internal order.
Each room has a grid of fields with doors or enemies on it and a story. The room is completed once all enemies in it are defeated. When the players enter a room, the doors get locked until the room is completed.

Attributes:

- id: Int - Number of the room (may be static)
- fields: Vector<Vector<Field>> - 2D vector of fields in a grid
- is_completed: Bool

Methods:

- print_room()

Field

Represents a single tile in a Room. It knows all the adjacent tiles, also diagonal ones, meaning at any time it can have a vector of 8 fields as adjacent fields.
Each field can contain an entity being an enemy, player, treasure chest, death location or door.
If there is an entity on the field, it also contains a status symbol, showing if its locked or anything.

Attributes:

- id: Int - is either `<ENEMY_ID>` (if the entity is an enemy) or `<ROOM_ID>` (if the entity is a door, this is the id of the room it leads to, or `0` if it is the dungeon entrance/exit)
- symbol: Char - gives additional information about the current state of the entity
- entity: Entity - shows the type of entity that is on the field
- adjacent_fields: Vector<Field> - List of Fields directly next to this field (also diagonal). Should be 8 unless on the edge

Methods:

- print_field()

Treasure chest

Inherits from the Entity abstract class
A treasure chest can contain different types of Items as loot and is either locked or unlocked. The default is locked, until a player unlocks it.

Attributes

- loot: Vector<Item> - List of items
- is_locked: Bool - If the chest is locked
- min_value: Int

Door

Inherits from the Entity abstract class
Represents the door to a different room. Is always locked by default and unlocks when the players completed a room. It contains a reference to the next room, which the players will get into when traversing the door.

Attributes:

- is_locked: Bool - If the door is locked
- leads_to: Int - Room ID the door leads to

Death location

Inherits from the Entity abstract class
Is a point where a character has died on the field and generally contains it's inventory or loot for others to collect.

Attributes:

- items: Vector<Item>

Potions

Inherits from the Item abstract class
Potions can only be used once and are then deleted from the inventory. The potions are created by passing a given Key to the JSON file. Potions of Resistance are active until the room is changed.

Attributes

- effect: Enum(Health, Fire, ...)

Ammunition

Inherits from the Item abstract class
Can be either a bolt or an arrow. Is used up when player performs an attack with a ranged weapon, depending on which ranged weapon is being used. Has no strength or effect attributes.
An enemy does never use up its ammunition, but only drops it as loot. Is parsed from a JSON file.

Attributes:

- type: Enum (Bolt, Arrow)

Armor

Inherits from the Item abstract class
Armor can be equipped with the `use` command and has a specific defensive value. Each Armor type is parsed from a JSON file.

Attributes:

- armor_value: Int - The amount of defense added by the armor

Weapon

Inherits from the Item abstract class
Weapons can be equipped via the `use` command. and have an attack type, pattern and damage. The weapons and attack patterns will be stored in an Enum. Enemies have a fixed weapon, while players can switch it using their inventory.

Attributes:

- attack_type: Enum - Either Melee or Ranged
- damage_type: Enum - Either Physical or an Effect
- damage_pattern: Enum - Pattern of the damage
- damage_amount: Int - Amount of damage dealt
- damage_addition: Enum

Command

Similar to the Command class from Assignment 1.

Attributes:

- type: Enum { ... }
- parameters: vector<String>

StorySegment

Contains the a segment of the story. Overloads the output operator to implement the ** and !! for the error messages.

Attributes:

- story_key: String
- text: String
- type: Enum { Narrator, Error }

Methods:

- operator<<(lhs: ostream, rhs: StorySegment): ostream
- getKey(): String

Game

Contains the entire game cycle and manages players, enemies, phases and more. It also parses the config files and starts the game accordingly.

Attributes:

- story: vector<StorySegments>
- players: array<Player*, 3>
- dungeon - Dungeon with the list of rooms
- current_phase - Either player or enemy phase
- print_map: Boolean
- print_story: Boolean

- current_phase: Enum { Player, Enemy }
- is_running: Boolean

Mathods:

- initializePlayers() - Initializes the players
- startGame() - Starts the game and prints the first message
- changePhase() - Checks and changes the current phase
- isGameRunning()
- execute(command: Command)

Dice

Helper class that simply represents an "unrolled dice". It simply contains the dice type and the amount to roll. It is used to save stats for the Items and is destroyed with the item upon usage.

Attributes:

- dice_type: Int
- roll_amount: Int

Methods:

- roll(): Int - Returns the rolled value and should be desctructed shortly after

---

Utilities

> **Mostly static**
>
> The utility classes are normal classes, but the functions they contain are almost all static for easier access. The idea behind the utility classes is to have a central place for repeated code

Utility class

A class that contains abstract functions to perform utility tasks like parsing User input, parsing files and calculations.

Attributes:
None

Methods:

- cleanInput(input) - Cleans the user input of whitespaces and converts it to lowercase
- readFile(file, line) - Reads a single line of a given file path

Exceptions/Errors

A class containing exceptions(error messages and returns), all referenced by a key for easier access. Just like a small database. The idea behind it is that we can call exceptions or error messages using a simple function anywhere in the code without duplication or boilerplate.

Attributes:

- Exception File - Contains the path to a JSON file containing the exceptions in an ordere matter

Methods:

- getException(key): [return_value: Int, error_message: String] - Returns the exception text
- getError(key): [return_value: Int, error_message: String] - May be replaced by the getException function, but would else return an error text (for example when user input is wrong)

Props

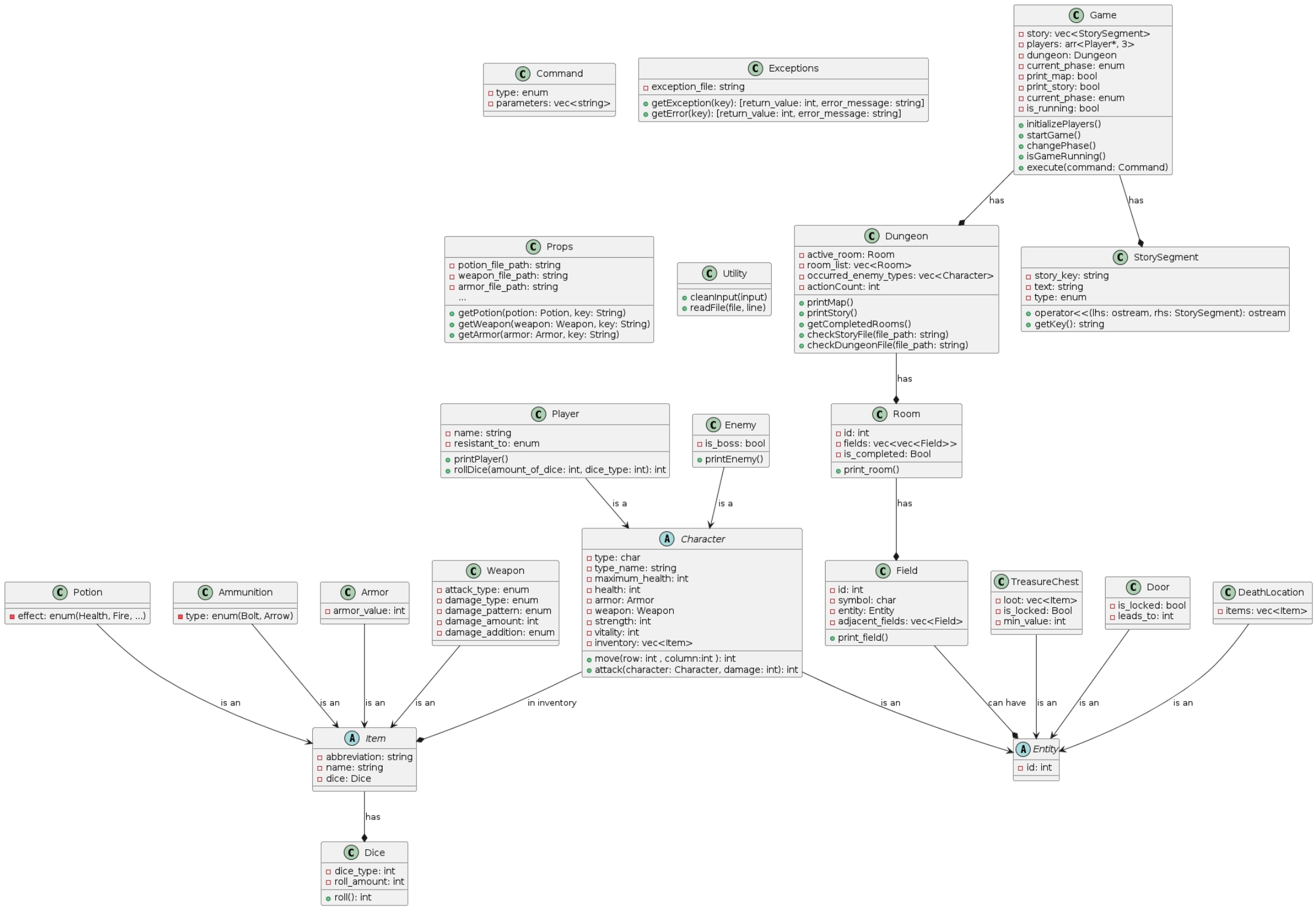
A class managing the JSON files for the props (potions, weapons, armor, entities, ...). We store the data in JSON for easier access and a cleaner code base. Then we can simply retrieve them using this class

Attributes:

- potion_file_path: String
- weapon_file_path: String
- armor_file_path: String
- ...

Methods:

- getPotion(potion: Potion, key: String) - Sets the retrieved data of a potion to the given potion reference. Needs a key to search for the correct potion
- getWeapon(weapon: Weapon, key: String)
- getArmor(armor: Armor, key: String)
- ...


Command
type: enum
parameters: vec<string>
Exceptions
exception_file: string
getException(key): [return_value: int, error_message: string]
getError(key): [return_value: int, error_message: string]
Game
story: vec<StorySegment>
players: arr<Player*, 3>
dungeon: Dungeon
current_phase: enum
print_map: bool
print_story: bool
current_phase: enum
is_running: bool
initializePlayers()
startGame()
changePhase()
isGameRunning()
execute(command: Command)
has
has
Props
potion_file_path: string
weapon_file_path: string
armor_file_path: string
...
getPotion(potion: Potion, key: String)
getWeapon(weapon: Weapon, key: String)
getArmor(armor: Armor, key: String)
Utility
cleanInput(input)
readFile(file, line)
Dungeon
active_room: Room
room_list: vec<Room>
occurred_enemy_types: vec<Character>
actionCount: int
printMap()
printStory()
getCompletedRooms()
checkStoryFile(file_path: string)
checkDungeonFile(file_path: string)
StorySegment
story_key: string
text: string
type: enum
operator<<(lhs: ostream, rhs: StorySegment): ostream
getKey(): string
has
Player
name: string
resistant_to: enum
printPlayer()
rollDice(amount_of_dice: int, dice_type: int): int
Enemy
is_boss: bool
printEnemy()
Room
id: int
fields: vec<vec<Field>>
is_completed: Bool
print_room()
is a
is a
has
Character
type: char
type_name: string
maximum_health: int
health: int
armor: Armor
weapon: Weapon
strength: int
vitality: int
inventory: vec<Item>
move(row: int , column:int): int
attack(character: Character, damage: int): int
Potion
effect: enum(Health, Fire, ...)
Ammunition
type: enum(Bolt, Arrow)
Armor
armor_value: int
Weapon
attack_type: enum
damage_type: enum
damage_pattern: enum
damage_amount: int
damage_addition: enum
Field
id: int
symbol: char
entity: Entity
adjacent_fields: vec<Field>
print_field()
TreasureChest
loot: vec<Item>
is_locked: Bool
min_value: int
Door
is_locked: bool
leads_to: int
DeathLocation
items: vec<Item>
is an
is an
is an
is an
in inventory
is an
can have
is an
is an
is an
Item
abbreviation: string
name: string
dice: Dice
Entity
id: int
has
Dice
dice_type: int
roll_amount: int
roll(): int