Escape From Hellspawn - Textual Data Files

Toronto Metropolitan University

Course: CCPS406 - Intro to Software Engineering - P2022

Student Names: Jay Sharma, Dryden Zarate, Clement Zhang

Date: May 15, 2022

Object Interaction

Instead of specializing every single different type of entity as being its own unique class, our group tried to generalize it as much as possible. The idea is to view everything as entities placed on the map and when a player-entity enters a zone, they interact with entities in that zone according to their interaction-priority property which is configured via JSON. How an entity interacts with or reacts to a player's commands is specified via JSON as well and this is one dynamic aspect of our game engine. Below is the format of this set of reactions to the player-entity:

```
"reactions": [
    {
        "preconditions": [
            {
                "operation": "",
                "target": "",
                "targetState": "",
                "value": ""
            }
        ],
        "print": "",
        "mandatoryAction": "",
        "optionalTarget": "",
        "postconditions": [
            {
                "operation": "",
                "target": "",
                "targetState": "",
                "value": "",
                "print": ""
            }
        ],
        "help": ""
    }
]
```

We can see that there could be "preconditions", if specified, must be satisfied to trigger the entity's "print" message. After which, the player is prompted for a "mandatoryAction" input with an optional "optionalTarget" input. The receiving entity waits and verifies it receives this input combination and if it does the game engine will trigger it's "postconditions". Preconditions and postconditions have the format above, which essentially allows the game designer to target one or more entities and modify their state. The specifics of each variable can be seen below:

operation can be a Create, Read, Update, Delete (CRUD) type of value. For example, in the case of Create it could append to an entity, for Read it may check if a list or dictionary contains a value and Delete would be a simple removal.

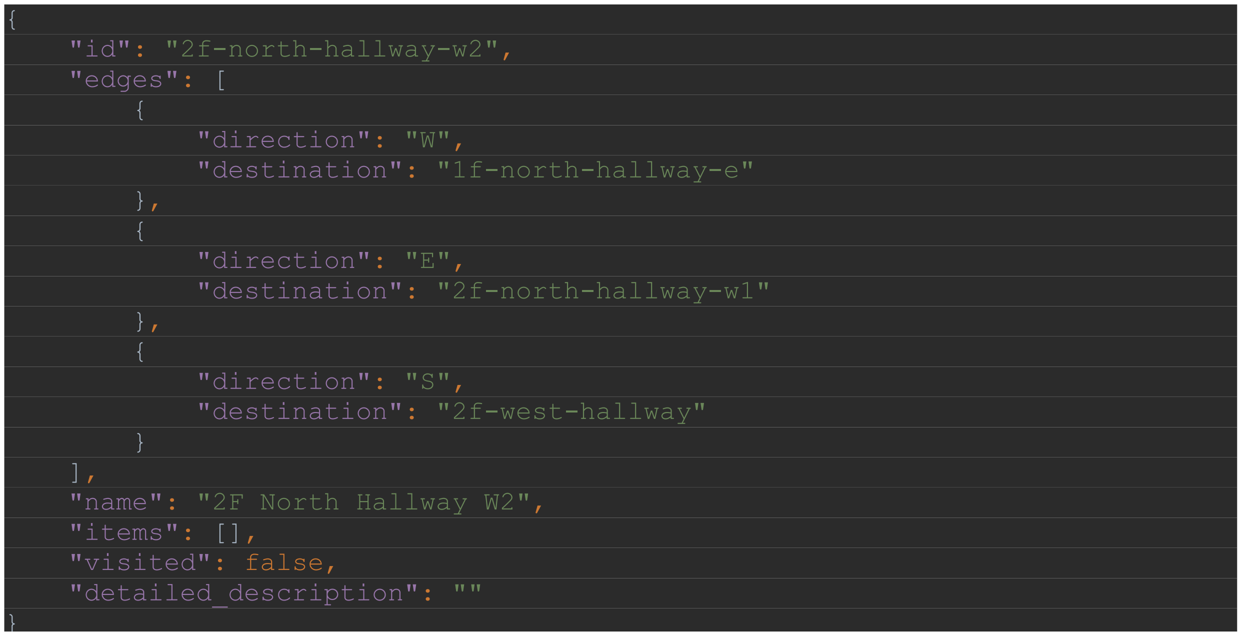
target is the target entities, which could be the "caller" or the "callee". It is feasible to also target all entities of a certain type, in which case its value would be "type=zombies" or we could target a specific entity by id, "id=123".

targetState is the state we want to apply the CRUD operation on.

value is CRUD operation's value.

Game Map

The game map is set up as a graph, where each vertex is an object and contains a list of edges. A vertex will represent a room on the map, with the edges in each of the four cardinal directions being connections to another vertex in the graph. It will know what item objects are stored in the room, while also returning a description based on whether the player is visiting for the first time or not. The character objects do not need to be specified because the characters objects contain what room they are in, which will be added to the room's data when that character is loaded in.



```json
{
    "id": "2f-north-hallway-w2",
    "edges": [
        {
            "direction": "W",
            "destination": "1f-north-hallway-e"
        },
        {
            "direction": "E",
            "destination": "2f-north-hallway-w1"
        },
        {
            "direction": "S",
            "destination": "2f-west-hallway"
        }
    ],
    "name": "2F North Hallway W2",
    "items": [],
    "visited": false,
    "detailed_description": ""
}
```

Character Entities

Characters such as the player, NPCs, and enemies are all extended from the main entities class, but they all have the same structural implementation. This makes differentiating them quite easy, as all that needs to be done is to set the “isPlayer” or “isEnemy” flags depending on what kind of character is being loaded in, with the default being friendly/neutral NPCs. An example of this can be seen below:

```json
{
   "id": "nurse",
   "name": "placeholder",
   "active": false,
   "inventory": [],
   "vertexId": "1f-east-hallway-s",
   "stats": {
      "Atk": 3,
      "Def": 0,
      "Current HP": 50,
      "Max HP": 50,
      "Current Mana": 0,
      "Max Mana": 0
   },
   "interactionPriority": 2,
   "greeting": "Nurse: Hi",
   "reactions": [
      {
         "preconditions": [
            {
               "operation": "not in",
               "target": "caller",
               "targetState": "adventureLog",
               "value": "healed-by-nurse"
            }
         ],
         "print": "Thank the heavens you are okay! Let me heal your wounds.",
         "mandatoryAction": "accept",
         "optionalTarget": "heal",
         "postconditions": [
            {
               "operation": "append",
               "target": "caller",
               "targetState": "adventureLog",
               "value": "healed-by-nurse",
               "print": "The nurse heals you."
            }
         ],
         "help": "accept heal"
      }
   ]
}
```

Game Engine

The game engine itself only has a few things that it really needs to hard-code, as most of the implementation comes from what objects are created through the JSON files that are read upon startup. This starts with reading all the required files in order to build up the game world through initializing every single entity object. Once that is done a string parser will take user input and determine what action the player would like to do, see if it is a valid action, and return an action based on that. The types of commands are also hard-coded, such as movement only being in the four cardinal directions, thus rooms cannot have more than one edge in each direction. Finally, an entity mediator handles how the reactions of entities based on their related preconditions and postconditions, allowing those to also be implemented by the game-maker and not the game engine.

Since the creator can use JSON files in order to make a game map, characters, and interactions, the game engine only needs to handle the general implementation of these objects. This means that as long as the objects are formatted as shown above, many different games can be built on our engine.