Research Project

By Guillem Turmo

What will I be talking about?

- Quest Variation
- Event System
- Data driven
- How do we implement a Quest Manager?

Why do we need a Quest Manager?

- Key component
- Underestimated
- Quest manager elements
  - Simplicity
  - Linearity
  - Capacity
  - Generation
  - Modularity
    - Grouped
    - Unique

Implementing a Quest System

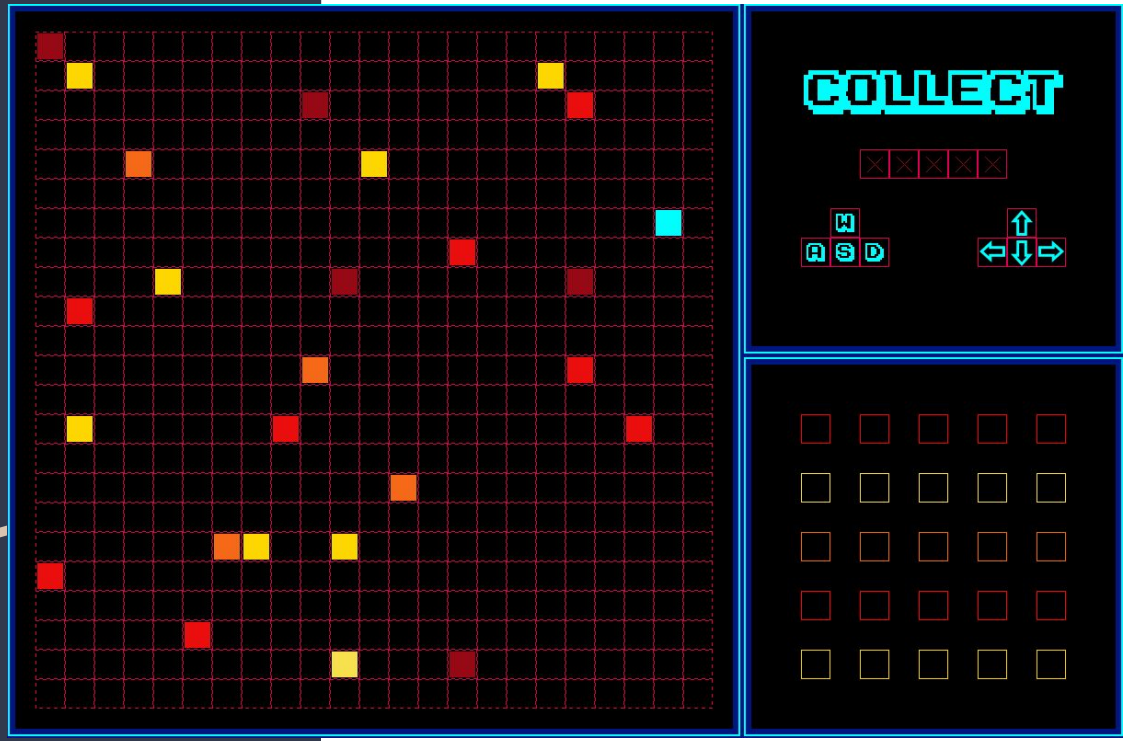
Basic structure:

```cpp
class j1QuestManager : public j1Module
{
public:

    j1QuestManager();
    ~j1QuestManager();

    bool Awake(pugi::xml_node& file);
    bool Start();

    pugi::xml_document quest_data;

    std::list<Quest*> loaded_quests;
    std::list<Quest*> active_quests;
    std::list<Quest*> finished_quests;
};
```

```cpp
class Quest
{
public:

    Quest() {};
    ~Quest() ;

    int id;
    int trigger;
    int reward;
    int requisites;
    int type;

    std::string title;
    std::string description;

    bool completed = false;
};
```

TODO’s

Environment:



TODO 0 & 1

- Start by taking a look at the basis I showed you above and make sure you understand the elements of the Quest System Structure and the Quests.
- This should only take a moment but it's crucial to keep up with the rest

- Once you've understood this, take a bit more time to read and observe the syntax of the xml that we will be using to load the data called quest_data

TODO 2

- All of our data will be parsed from that XML, therefore we need to create a new function in the App that will read and load that XML for us

```
//Load quest file
pugi::xml_node LoadQuests(pugi::xml_document& quest_file) const;
```

TODO 3

- Now that we have the XML loaded, we'll begin loading all of the info into our Quest Manager.
- Remember to use the LoadQuest function we just created and to use the proper syntax.
- We will code a loop that creates a new_quest and loads all of its info for every quest on the XML

```
Quest* new_quest = new Quest();
```

TODO 4

- Right now, we are creating the Quests but we are not storing them properly, therefore after every loop we will push them into the loaded_quests list, however if a quest's trigger is equal to 1 (meaning that is always active) we will directly put those into the active_quests list

TODO 5

- Now that we have our base working, it's time to implement it with the context, therefore we are gonna create a simple function that checks the events that we are interested in.
- The skeleton is already implemented, therefore you will only need to create a loop that will iterate the active_quests list and checks those conditions

TODO 6

- We are almost done, we are currently correctly checking the active_quests list but we aren't doing anything with a quest once it's completed, therefore we need to transfer the complete quests to the finished_quests list

TODO 7

- As the final TODO, just take a look at how we are only drawing an achievement (quest completed) once they are in the finished_quests list. I'd also like you to take a look at how we make the 4th achievement a different quest by forcing the player to complete the side-quests in a particular order

Thanks!