Augmented roleplaying tabletop game

by Mathieu LOUVET & Paul MEUNIER

Description:

This augmented roleplaying tabletop game system has an extension aim. A roleplaying game such as Dungeons & Dragons or RuneQuest is a game that puts emphasis on two parts: roleplay & combat.

Roleplay is often players interacting with their Dungeon Master or DM, acting their own characters to *roleplay* situations. This doesn't need more than a bit of imagination and maybe a few characters sheets to remember data, such as the investment allotted to each of their character's skills.

Combat is explicit. Players use their characters to fight the absolute evil represented by hideous creatures. However, the absolute evil is also the lazy evil, aka the DM. During a fight, depicting by voice every enemy move can be tiring, furthermore with sometimes inattentive players. The common tool to solve this situation is the plain old squared sheet of paper, which is efficient and reusable. Players simply use figurines as their tokens to fight enemies represented by various objects, such as dice.

Yet, the tool is still not enough. Spatial coordinates are indeed helpful, but the immersion in the fight is not absolute: obstacles and environment layout are still invisible. Printing maps can improve game experience, but ink is not unlimited, and printing has loads of defaults. Therefore, we propose this project.

Motivation:

Our motivation comes mainly from the fact that we ourselves are roleplaying game players. And as such, we had several times problems during games, mainly during combat, where players wouldn't know exactly where they were, what distance they could run, or which enemy they could hit relatively to their current position. That's the main problem when playing with real tokens.

So, our goal was to augment real tokens, so players could instantly see their range of movement and actions and display the actions possible, spells/actions which were already used, etc… This would really make group fight easier to understand and to manage, for the dungeon master and for players. The motivation of using real tokens was to recreate the fight with tokens and potentially effects to make it livelier than just a screen with a mouse.

Using a Kinect was one of our goal, as it would have been easier for the users: they would just have to move the tokens, and the Kinect would have followed it, as well as their hand, and the software would have acted in consequence.

Architecture:

For starters, and as stated precedingly, and goal was to use the KinectV2 to interact with our software. The Kinect recognized the tokens, and our hands. The problem was that the Kinect couldn't recognize the back of our hands. So, our plan to have it on the ceiling, or above the table was busted. Even top-notch recent research by Microsoft could follow the fingers from the front with precision, but not from the back of the hand. We had to abandon this idea at two third of the project.

One way of doing it was to follow an object by brutal image-processing, which was done. This object would have been a token used by players. However, it only worked with rectangular shapes and would have failed with tokens seen from another angle than the working one. Moreover, it made our Kinect as useful as a regular HD camera. So the approach was dismissed to try to improve interactivity on the software part.

We then had to get it back on rail and decided to do it as we first planned it, on a touch based Windows10 tablet.

Our project is based on Unity, so the architecture is a Unity one. It's composed of components which are all independent one from another, and we attached scripts to each one of these components to manage their behavior. The central object is the game board which manage the game in itself (turn, position, etc). All the scripts are in C#.

Each token(player) is an instance of the same object. The only thing which is variable is the range on which the player can move, and the variety of actions they can execute. The unity project is made of several components, the most important ones being the board and the players.

Code explanation:

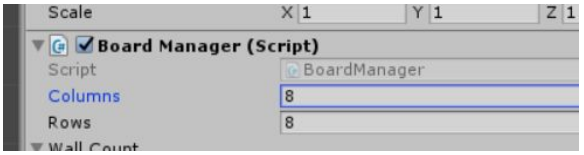
The code in itself is simple, what is complicated is the fact that we use unity. In that sense, all the complicated functions are native to the game engine. Unity uses a system which associates a script with an object. This system permits to rapidly add several objects of the same type, called prefabs, and instantiate them.

Board:

The board is generated procedurally, and can take any size. The board is made of tiles, which helps to generate it easily. Furthermore, thanks to Unity, it's possible to change easily the texture of the tiles of the floor. There are several types of tiles: ground, walls, and holes (the last one is not yet implemented). Tiles are aligned as if they were on a grid, and are saved in a Unity *GameObject* array. There are three arrays: one for the floor tiles, one for the walls inside the board and one for the outer walls of the board. The number of

columns and rows of the board are modifiable are they are set as public field of the *BoardManager* class:

```
public int columns = 10;
public int rows = 10;
```



This permit to have a modifiable board which is generated randomly by the following method.

Player:

The player token is movable with the finger, as the software is running on a touch tablet, but also with the keyboard, if one is connected to the tablet. The player is managed by a script attached to the player object, which manage the movement of the token across the board.

To move the player, we chose to use the tablet touch, which is also a click. Unity understands these interactions as the same kind of events, which facilitates programming.

The same script also manages the display of the multiples feedback given to the player, as the range of movement possible when it's a player turn. When a token is clicked, it displays the possible moves of the token according to its token speed. However, this is only an indicator, as the physical token is not bound by the overlay. Moreover, it might have a speed boost under special circumstances (such as a speed spell).

Files are available at this url : https://github.com/Tlospock/KinectUnity