Roleplaying Coder (Combat) - System Design

Nicole Hurley
Seth Lunn
Vermont Technical College

Introduction

The goal of RPC is to introduce children of ages ten to fourteen to programming concepts. The game will focus on a male or female character (player's choice) who goes on a quest. During the combat stages and the gear creation phases of the game the player will have to use creative, logical, and problem solving skills in order to advance in the game.

Some key concepts of a combat stage that a player would experience in the game are:

- Turn based combat, player versus computer
- The ability to logically plan out attacks in a procedural manner.
- Keep a history of what the player did during the current combat phase.
- The player may be in combat against multiple computer agents.
- The player will face different challenge difficulties of AI.

All of these capabilities must be reflected in the combat system design, as well as the game's functionality.

Design Goals

RPC will be released as a fully completed game. The design for combat must keep in mind this is one aspect of a full RPG game and it must be compatible with all of the technologies used for the game as a whole.

Another important goal for the design of the combat phase is the art, music, UI and overall gameplay must be consistent with the rest of RPC.

The main purpose of the combat phase is to give the player the ability to customize their combat sequences in any way they see fit. The design should allow almost complete control over the character's combat movements and actions, within reasonable constraints. The player should be able to learn from past experiences, the design should allow for the player to replay sequences to see outcomes.

Key Components

These are the key components of the combat phase of RPC.

- Rendering
- Turn Controller
- Player Controller
- Artificial Intelligence
- A Plug-in that manages the players input
    - File manager
    - User Interface to select the combat logic
    - Interpreter for the combat logic

The **Rendering** is the graphical component to display all aspects of the combat onto a computer screen. This includes both the characters' movement as well as the user interface. It handles all of the calculations necessary to properly display the game and it's elements. This rendering should be consistent for not only the combat but also the rest of RPC.

The **Turn Controller** will be the "brain" of the entire combat phase. The turn controller will be responsible for keeping track of the turn-based feature of the game, making sure each agent's turn is accounted for. It will be responsible for multiple AI agents as well as the player's agent. The Turn Controller knows each agent's move after they have selected them, and must make sure they all happen at their appropriate times.

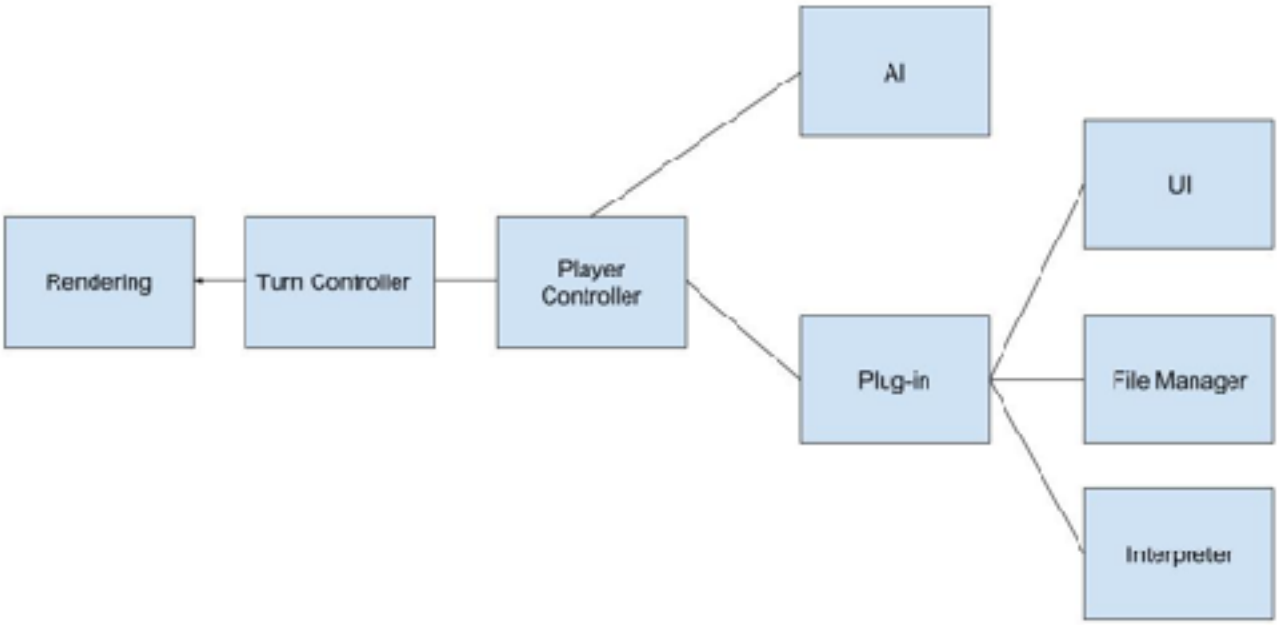
The **Player Controller's** purpose is to validate the input of either the AI or the player and make sure it is within game constraints. The player controller is only responsible for one agent's input at a time. The player controller also packages the input to be sent to the turn controller. This includes things like their health and what they are using for equipment.

The **AI component** could be one or many agents. There will be an algorithm to determine the best move(s) for the agents. The AI component will be responsible for determining how difficult the AI agent will be for the player to defeat based on the current level of the player as well as how far they have progressed in the game's story.

The **Plugin** is the input of the player's logic for their combat turn. This will have three different sub-components. The main goal of the plugin is to determine what the player has chosen for their sequence. The player will use the **user-interface component** for selecting their moves and what order they would like them to be done in. Once the player has selected their actions via the UI, the choices will then be sent to the **interpreter.** The interpreter will package the choices the player has made so be sent back to the main plugin component. The code will also be sent to the **file-manager** in order to keep a history of what the player has done. The player will be able to review their previous move and the outcomes it had.

Component Interactions

The diagram below shows the major components as well as how they interact with each other. The straight non-arrow lines indicate that there will be back-and-forth messaging between components. The arrow indicates there is only one-way communication.



Rendering & Turn Controller

The rendering component does not send messages in this portion of the design. However, it does receive messages from the turn controller, the messages would include the moves to be graphically rendered as well as the state of the players that the rendering component would then reflect graphically.

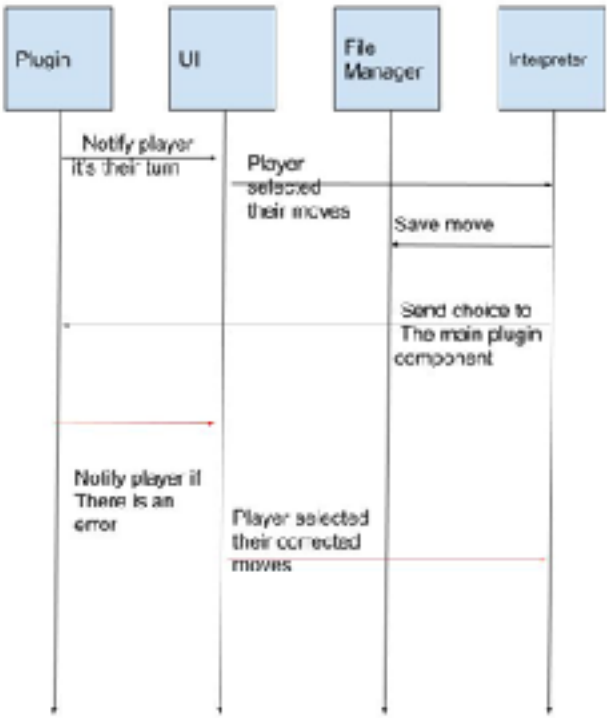
The turn controller sends these messages after receiving them from the player controller. First, the turn controller will send a request to the player controller for a specific agent's turn. The player controller will then request the choices of either the AI agent or the player agent. The player controller then either gets a message from the AI component about the AI agent's choices, or the player's choices, respectively.

Error Handling

The player controller will validate the choice before sending it back to the turn controller. If the selected choices are invalid for any reason, there is an error message sent back to the appropriate component. If the error is due to the player's choices, the plugin will send a message to the UI component to be displayed to the player, with a text description of why the error occurred and the possible solutions.

The Player's Turn

The diagram below shows what would be a typical sequence during a player's turn between the plugin and the sub-components. The arrows in red only happen in the event of an error. All of the sub components in the plugin can communicate with each other.



The player will be told that it is their turn to select their moves. They will select their moves with the UI. Once the moves are confirmed by the player, they will be sent to the interpreter. The interpreter will then save the move choice by sending the interpreted results to the file manager. Then the interpreter will also send them to the main plugin component in order to have the choice sent off to the player controller. If there is an error because the player attempted to violate game constraints, the player must reselect their moves accordingly.

One option that is not reflected in the sequence diagram is the player wishes to review their last move. The player will select this in the UI. and then the UI will request the previous move from the file manager. Since the file manager contains the already interpreted result, the file manager will send this choice directly to the main component.