進捗報告

1　今週やったこと

- 実験に用いるサンプルゲームの作成
- 実験環境および実験方法の検討

2　サンプルゲーム

森先生から頂いたテーマに即して, 実験で用いる自作のカードゲームを Unity を用いて作成した. Windows10 を搭載した PC で作成し, 使用した Unity のバージョンは 2020.3.29f1 であった.

図 1 にゲームのプレイ画面を示す.

図 1: 作成したカードゲームのプレイ画面

図 1 において, 画面上下部に存在する Player と Enemy がゲームの参加者である. Player , Enemy の右側に存在する赤丸は HP を表しておりこれを 0 にすることが本ゲームの勝利条件となる. 左側に存在する緑丸はマナコストを表しカードを場に出すとこの数値はカードのコストに応じて減っていく. 画面右側の青色のボタンを押すとターンが代わり, ボタンの上の数字は Player , Enemy それぞれの持ち時間を示している. ターン終了後にはマナコストは以前のターンより 1 つ多く回復する.

図 2 に示すようにカードは 3 種類の数字を持つ. 図 2 において左下の赤い数字はカードの攻撃力を示す. また, 右下の青い数字はカードの HP を示す. 左上の緑色の数字はカードのコストを表す. 今回作成したゲームではその場に出た時にすぐ行動できるカード, 自身が場に出た時は敵カードは自身以外に攻撃できないといった

図 2: カードの UI

特殊効果を持つカードも実装している. また一般的なカードゲームにおけるスペルカードも実装している. スペルカードは自身が場に出ない代わりに, 使用すると何らかの効果を与えるカードである.

今回実装したスペルの種類を以下に示す.

- 場に出ている敵カード 1 枚に一定数ダメージを与える.
- 場に出ている敵カード全てに一定数のダメージを与える.
- 敵に一定数のダメージを与える.
- 場に出ている味方カード 1 枚を一定数回復する.
- 場に出ている味方カード全てを一定数回復する.
- 自身を一定数回復する.

3　実験環境および実験方法の検討

本研究テーマ「自作カードゲームの難易度調整 AI の作成」に類似した研究を参考に, 実験環境および実験方法の検討を行った.

3.1　MCS-AI 動的連携モデル

ゲーム AI のモデルアーキテクチャとして, ゲーム全体を俯瞰的に認識しゲーム内のあらゆる要素をコントロールするメタ AI , キャラクターの頭脳と身体を含めたキャラクター AI , 自律的に環境を解析し必要な情報を提供することでキャラクター AI とメタ AI をサポートするスパーシャル AI と大きく 3 つに分けそれらを相互に左右させる MCS-AI 動的連携モデルが考案されている [1].

3.2　MCS-AI 動的連携モデルにおける類似研究

宋, 三宅らは MCS-AI 動的モデルを用いて, ゲームの難易度に関わるパラメータを unity 側が一定間隔で Python 側に送信し, Python 側で k 近傍法を用いてチューニング処理を行ったパラメータを Unity 側が反映するといった形で Unity で作成した FPS ゲームの難易度自動調整を試みた [2]. [2] の結果としてはチューニング

が十分ではなく, 改善が必要という結論に至ったが, 使用された図 3 の Unity と Python の連携方法は本研究に活かせると考えた.

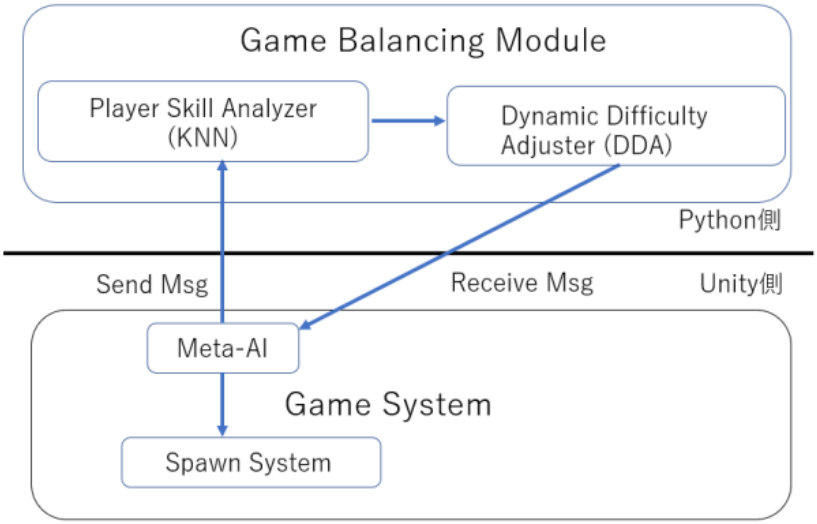

図 3: FPS 自動難易度調整の研究で用いられた Unity と Python の連携

3.3 検討手法

以上のことを踏まえ, 本研究では図 4 のような実験環境を想定した. PlayerAI と EnemyAI が対戦を行いそのログを Python 側 へ送る. Python 側でそのログからパラメータのチューニングを行い, Unity 側に送り反映する. これを繰り返すといった形で実験できると考えた.

3.4 試したこと

- Ubuntu への Unity のインストール
  インストールと実行確認はしたが, Unity で強く推奨されている VisualStudio が使えなかったためコードを書く際に補完機能などが無く開発が難しい.
- Socket 通信
  ネット上のサンプルコード [3] を試した結果, 研究室の PC の環境で Unity と Python 間の相互の socket 通信は正常に動作した. サンプルコードは Unity3D 用だったが作成した 2D のカードゲームに適用する作業が必要である.

4 今後やること

- サンプルゲームの Ubuntu への移植
- サンプルゲームの改良 (ログの生成, リファクタリング,Socket 通信)
- Python 側の内部処理の方法の検討
- 調整するパラメータの選定, playerAI,enemyAI の動作といった実験条件の決定

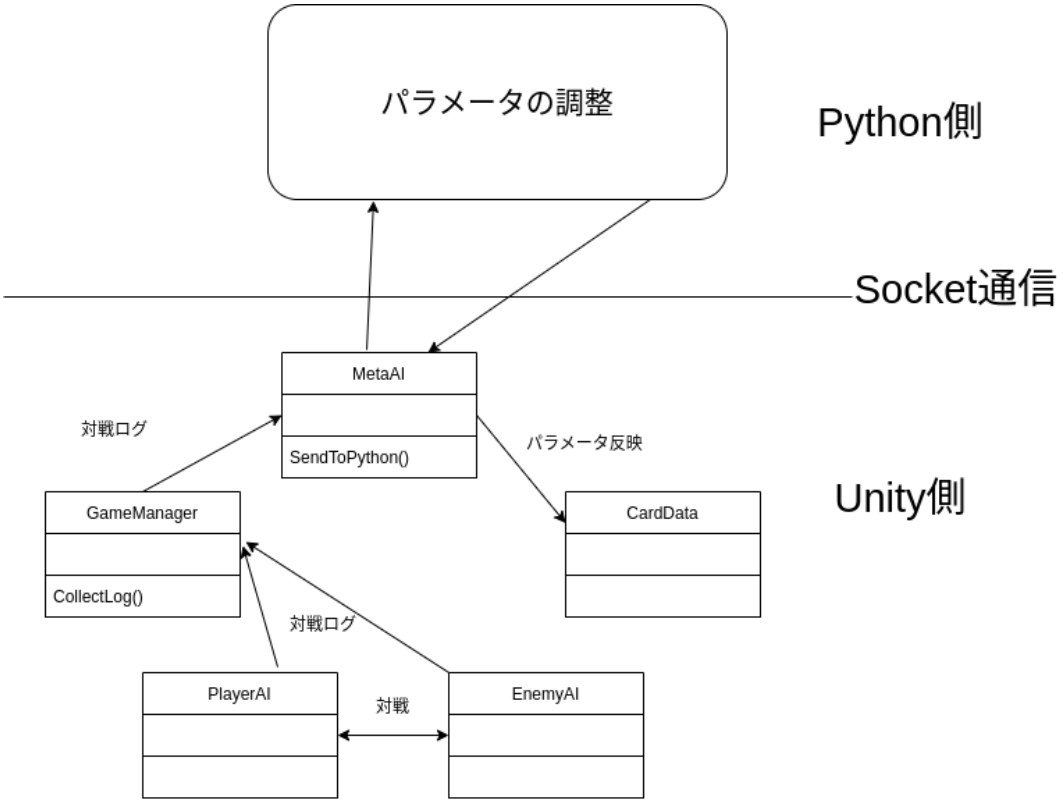

図 4: 本研究での実験環境案

参考文献

[1] 三宅陽一郎. ディジタルゲームにおけるメタ ai-キャラクター ai-スペーシャル ai 動的連携モデル. 人工知能学会全国大会論文集, Vol. JSAI2020, pp. 1P4GS701–1P4GS701, 2020.

[2] 亜成宋, 陽一郎三宅. メタ ai を用いた fps ゲーム難易度の自動チューニング. ゲームプログラミングワークショップ 2021 論文集, Vol. 2021, pp. 49–56, 11 2021.

[3] Siliconifier. Python-unity-socket-communication, 2019.06. `https://github.com/Siliconifier/Python-Unity-Socket-Communication`.