

# 卒 業 論 文

## 「研究室に来ること」へのゲーミフィケーションの 適用

指導教員 上野 雄大 准教授

新潟大学  
工学部知能情報システムプログラム

T22B920D 村井 一彬

提出年月日: 2025 年 2 月 28 日

卒業年月日: 2025 年 3 月 24 日

# 概要

大学の研究室においては、学生の自主性を重視し、特定の時間に研究室へ来ることを強制しない方針を取る場合がある。このような研究室では、学生が自ら研究に取り組む姿勢を持つことが期待されている。しかし、コアタイムが設定されていないことで、研究に対するモチベーションが維持できず、研究室にほとんど来ない学生がいる場合がある。研究室に来ないことが、研究の進捗に必ずしも影響を及ぼす可能性があるわけではないが、研究室に来ることで、他の学生や教員との雑談や議論を通じて新たな発見が得られるなど、研究にとって有意義なメリットがある。そこで、本研究では、学生が研究室に来ることのメリットを実感し、自然に足を運ぶようになる仕組みを構築することを目的とする。その手法として、ゲーム以外の行動にゲームの要素を取り入れ、モチベーションを高める「ゲーミフィケーション」の考え方を活用する。研究室に来て「人と話す」メリットを実感させることにゲーミフィケーションを導入することで、学生の自主的な研究室への来訪を促進する。

# Abstract

Some university research laboratories do not impose working hours on students in order to respect their autonomy. Such laboratories expect students to have motivation independently. However, there are cases in which some students cannot maintain their motivation for research, and they do not come to their laboratories. It is not necessary to visit the laboratory for research, but visiting laboratories should have some advantages. For example, if we visit our laboratories, we can have the opportunity to talk with other students or our professors. This should give us new discoveries and enhance our research. For these reasons, this study aims to build a system that helps students recognize the benefits. The system should encourage students to visit their laboratories more frequently. To achieve this, I employ the concept of "gamification", which integrates game-like elements into non-game activities to enhance motivation. This approach seeks to help students recognize the advantages of "talking with people" and visit their laboratories voluntarily.

# 目次

<b>第1章 序論</b>	<b>1</b>
1.1 背景と目的	1
1.2 構成	2
<b>第2章 研究室に來ない学生への対応とその課題</b>	<b>3</b>
2.1 研究室に行くことのメリット	3
2.2 研究室に行くことの煩わしさ	4
<b>第3章 ゲームフィケーションによる現実問題の解決</b>	<b>5</b>
3.1 ゲームフィケーションによる動機づけ	5
3.2 ゲームフィケーションの適用に関する研究事例	6
<b>第4章 研究室に來ることのゲームフィケーション</b>	<b>8</b>
4.1 ゲームフィケーションの狙い	8
4.2 ゲームとしてのルール設計	8
4.3 狙いを達成するためのゲームフィケーションの要素	10
4.4 ゲーム不参加者による不利益の回避	11
<b>第5章 ゲーム進行を補助する bot の実装</b>	<b>12</b>
5.1 Discord bot の利用	12
5.1.1 Discord アカウントの作成	12
5.1.2 Discord サーバーの作成	13
5.1.3 Discord bot の導入	15
5.2 Python による Discord bot の開発	16
5.3 ゲーム参加者の管理	17
5.4 チーム分けのアルゴリズム	18
5.5 ChatGPT を活用した出題	23
5.6 各チームに問題を割り振るアルゴリズム	27
5.7 問題分割のアルゴリズム	29
5.8 答え合わせと解答回数制限	31

<b>第 6 章</b>	<b>結論と今後の課題</b>	<b>34</b>
6.1	結論 . . . . .	34
6.2	今後の課題 . . . . .	34

# 第1章 序論

## 1.1 背景と目的

私の所属する上野研究室は、ゼミの時間以外で必ず研究室に来なければいけない時間は設定されておらず、自分を含めたほとんどの学生がゼミの日以外で研究室に来ていない状況があった。頭の中では研究室に行かなければならないという思いがあったものの中々研究室に足が運ばないという経験をした。しかし、研究室に来ることで得られるメリットというのは確かに存在する。研究室に来ると周囲の環境によって研究が捗るといったものの他に、周りの先輩や同学年の学生と何気ない雑談をすることや研究に関する本格的な議論を通して自身の研究を振り返る機会を得ることができる。私自身研究室に行き、先輩と研究の議論をすることで、自分の研究の目標や現段階での進み具合を確認することができた経験がある。本研究では、このような他の研究室に所属している学生に研究室に来て「人と話す」メリットを知ってもらうことを目的とする。

研究室に来ることは、煩わしい。これは自分の体験を根拠にしているが、研究室に来ていない学生のうち、同じ様に思っている学生がいるはずである。その煩わしさを超える動機があれば、そのような学生でも研究室に来るはずである。井上 [1] によると動機には内発的動機と外発的動機の2種類がある。内発的動機は卒業研究で例えると、研究という行動自体に動機づけがされていることを指す。研究が元から好きで、研究をしている学生がこれにあたる。ただ、研究が好きな学生は研究室にも顔を出している場合が多いので、今回の研究からはその様な対象は除外して進めていく。外発的動機を同じ卒業研究で例えると、単位や教授によって研究室に来る時間が指定されている場合(コアタイム制)がある。教授に「明日から毎日研究室に来ないと、単位を上げない」と言われると、学生は嫌でも研究室に来るはずである。しかし、その様な外発的動機づけでは、先述した「人と話す」メリットを実感してもらえない。そこで、外発的動機づけを無理なく与える手法として、ゲーミフィケーションと呼ばれる手法を研究室に来ることに適用する。

ゲーミフィケーションとは、ゲームが持つ、「人を熱中させる」要素をゲーム以外に適用することである。本研究では、「研究室に来ること」にゲーミフィケーションを適用し、それに沿ったゲームを作ることで、研究室に来ない学生に対して、外発的動機づけを与える。そのゲームを通して学生に「人と話す」メリットを自覚してもらうことを目的とする。具体的には、研究室に所属する学生が1つの問題に取り組む過程で議論を促すクイズゲームを設計する。そのクイズゲームを学生にプレイさせることで、背景で述べた煩わしさを超えるような動機づけをすることを狙いとしている。さらに、Discord bot を用いて

学生がゲームに参加しやすく、ゲーム管理者が学生を管理しやすい様なクイズゲームの実装を試する。

## 1.2 構成

本論文の構成としては、第2章に本研究の背景である研究室に来ることの煩わしさとそれでも研究室に来るによって起こりうるメリットについて記述する。第3章ではゲーミフィケーションとは何かについて動機づけを元に説明する。また、研究の事例についてもここで述べる。第4章では、第3章で述べたゲーミフィケーションをどの様にして「研究室に来ること」に適用するのか述べる。また、その適用に沿ったゲームの設計についても記述する。第5章では設計したゲームを Discord bot にて実装する過程について説明する。最後に第6章では結論と今後の課題について述べる。

## 第2章 研究室に來ない学生への対応とその課題

### 2.1 研究室に行くことのメリット

本研究の背景として、上野研究室では、学生が研究室に來なければいけない時間(コアタイム)が設定されていない。その狙いとしては学生の自主的な研究活動を尊重していると考えられる。しかし、自分を含めた多くの学生が研究室に來ていない時期があった。そのような学生の中で、どうしても研究室に行くことができない特別な事情がある学生を除く、全ての学生の根底にある考えとして研究室に行くことが煩わしいと感じているという事実がある。この事実の根拠として、研究室に行くことを煩わしいと思っていない学生は、研究に対して意欲的でありコアタイムの有無に関係なく、自ら進んで研究室に來ようとするためである。

研究室に行かないことと研究ができないこととは直接的な関係はない。つまり、必ずしも研究のために研究室に行く必要はないと考えられる。ただ、研究室は同じ教授の元、同じ分野を研究する同志が集まる場所である。特に、研究室に配属されたばかりの学生は「研究」というアクティビティを知らないため、教授や先輩から自身の研究について指摘及び指導してもらいやすい環境はその様な学生が研究を進めていく上でとても役立つはずである。また、共同で研究を行わない場合、研究はとても孤独を感じやすい。研究の話のみならず雑談をすることで、他人との関わりを深め、「研究」に対するストレスの低減にもつながる。研究室にはそれだけの利点があることを知らない学生は、学術的なアクティビティをする上でそれだけ損をしていると感じ、学生にそのメリットを自覚しないまま研究室に配属され続けることは、解決すべき課題だと考えられる。

先ほど記述した「研究室に來る」メリットは「人と話す」と繋がっている。しかし、それらのメリットはただ研究室に行けば無条件で獲得できるものではない。初対面で出会う人が感じる壁によって中々「人と話す」ことができなかった結果、メリットを感じる前に研究室に來なくなってしまう場合がある。よって、学生が研究室に行くことを煩わしいと感じる前に、研究室に來ることのメリットを感じてもらうことが研究室に人を來させる場合に考えなければならない事であると言える。

## 2.2 研究室に行くことの煩わしさ

私自身も研究室に配属された当初は研究室に行くことを煩わしいと感じていた。自分が研究室に行くことを煩わしいと思うのは以下の理由が挙げられる。

1. 身だしなみを整えるのがめんどくさい
2. 家にいる限り、人目を気にせずに研究できる
3. 研究室が遠い

1の「身だしなみを整えるのがめんどくさい」については、大学に行くために髪を整えたり、服を着替えたりする際の煩わしさを指している。大学の授業を受講していた時は、単位の取得のためいやいやしていたことに対して、家でも可能な研究をするためにすることはめんどくさいと感じる理由になり得る。2の「家にいる限り、人目を気にせずに研究できる」に関しては、研究室に来るということは、人と会う可能性がある。特に、研究室に配属されて日が浅い場合に、知らない先輩や同期に対して気まずさを覚える人もいるはずである。家では、それを気にせずに、のびのびと研究できると考える人も一定数いる。3の「研究室が遠い」については、実家に住んでおり、研究室に来るまでに時間がかかるため、研究室に来ることが面倒に感じる場合がある。

いずれの理由も研究室に行くことに関して煩わしいと感じる理由として納得ができるものである。しかし、これらの煩わしさを凌駕する2.1節で述べたメリットを学生に感じてもらうことで、学生の「研究室に来る」モチベーションを高めることは可能なはずである。ただし、学生に対し無理強いをして研究室に来させたとしても、学生自身に議論をするモチベーションがないため、「人と話す」メリットは自覚しづらいと考えられる。そこで、学生が負担を感じにくく、なおかつ研究室に来ることのメリットを自覚してもらうための手法として「ゲーミフィケーション」がある。次章ではゲーミフィケーションとは何かを動機づけの観点から説明する。

## 第3章 ゲームフィクションによる現実問題の解決

本章では、ゲームフィクションとは何かを動機づけの観点から説明し、その事例について議論する。

### 3.1 ゲームフィクションによる動機づけ

この段落と次段落の文章は井上 明人氏著「ゲームフィクションゲームがビジネスを変える。」[1]の要約である。近年、ビデオゲームは趣味として大いに普及しており、誰もが一度は遊んだことがあると言える。それほどまでにビデオゲームは数多くの人を夢中にしている。井上によるゲームフィクションの簡単な定義は「ゲームの要素をゲーム以外のものに使う」[1]である。これはゲームの「面白い」という要素をゲーム以外の行動に結びつけることによって、本来なら煩わしさを感じる行動に対して自主的に取り組めるように促すこととも言える。

本書ではゲームフィクションとして様々な事例が挙げられていた。例えば「ウォーキング」の場合、健康に良いとされながらも「歩くこと」自体に対して楽しいと感じていない人が多い、そもそも外に出て歩くのが面倒に感じている人もいる。「健康の為」に煩わしい「外に出て歩く」ことをしなければならない。そのような思惑の中、任天堂が1998年にリリースした「ポケットピカチュウ」[2]はそれらの煩わしさをゲームによってかき消している。「ポケットピカチュウ」は万歩計に似た外見をしており、歩数をカウントすることによって端末内のキャラクターとの親密度が増していくというゲーム内容になっている。歩けば歩くほど、ゲーム内のキャラクターと仲良くなり、歩くことをやめれば愛想をつかれる。これにより、「歩くこと」自体にモチベーションを見出していなくても「ゲーム内のキャラクターと仲良くなりたいから」という動機づけがゲームによって与えられている。ゲームに沿って歩くことを繰り返していれば「健康の為」という元来の目的も達成できる。「ゲームのキャラクターと仲良くなりたいから歩く」と「健康の為に歩く」とはそれぞれ歩くことに対する動機である。井上によると、前者は「外発的動機づけ」と呼ばれており、報酬や罰を理由に動機づけられることを指す。後者は「内発的動機づけ」と呼ばれており、自らの活動それ自体に動機づけられることを指す。「ポケットピカチュウ」の場合、「キャラクターと仲良くなる」ことは報酬に過ぎず、本質的な目標は「健康」である。そのため、ゲームフィクションの主な働きとしては、外発的動機によって行動

の煩わしさ見えにくくし、報酬を追っていくうちに本来の目的である内発的動機へと辿り着くメカニズムである。

前述したメカニズムを研究室に来ることに適用することを本研究では行う。その場合における外発的動機と内発的動機についてそれぞれ議論する。外発的動機に関しては、「人と話す」ゲームを設計し、実際にプレイしてもらうことが当てはまる。しかし、ただゲームをプレイしてもらうだけでは、外発的動機は駆動できたとしても、そこに内発的動機に繋がる要素がなければゲーミフィケーションとは言えない。そこで、ゲームをプレイするうちに、「人と話す」ことに対して自然と内発的動機を持つ様なゲームを第4章で議論する。

## 3.2 ゲーミフィケーションの適用に関する研究事例

ゲーミフィケーションを研究に適応した事例として、鳴海らによる卒業論文・修士論文を執筆する学生と学生に助言を行うメンターに向けた補助システム「卒論ウォッチ」[3]がある。鳴海らは論文の説明中で「卒論ウォッチ」は Google spread sheet を用いた Web ページのシステムであり、執筆した卒論のページ数を Google spread sheet で管理するシステム、卒論を執筆する過程で「20 ページ執筆した」等あらかじめ決められた出来事を達成した際に称号とその象徴であるバッジを付与するバッジ管理システム、他の参加者がバッジを獲得した際にそれを twitter で通知する twitter 通知機能があることを述べている。

鳴海らはこの研究事例において、動機づけの議論をしていなかったが、このゲーミフィケーションにおいても外発的動機づけ及び内発的動機づけの関係が成り立っている。外発的動機づけについては、締切までに論文を書かなければならないという強制された環境が前提にあり、その上でバッジの取得や他の学生との進捗の競争といった達成感や競争心を煽るものとなっている。内発的動機づけについては、卒論ウォッチを通して効率的に執筆が進んだことによる「卒論提出後の卒業」である。学生はゲーミフィケーションが形作った外発的動機づけである「バッジ取得による達成感」を追い続けていくうちに卒論の執筆が進んでおり、内発的動機づけである「卒業」ができるといった体験をする。

「卒業論文の執筆」は「卒論ウォッチ」がなくとも、卒業するために学生は必ず執筆する。つまり、ゲーミフィケーションを用いなくてもすでに外発的動機づけがされている状態である。「卒論ウォッチ」はその外発的動機づけをゲーミフィケーションで補強している。しかし、「研究室に来る」ことは強制されておらず、外発的動機づけをゲーミフィケーションのみで行う必要がある。「卒業」というわかりやすい報酬が存在しないため、本研究で用いるゲーミフィケーションで内発的動機づけを駆動できない場合、学生としては、ゲーミフィケーションを用いたとしても退屈なゲームをしているように感じてしまう。外発的動機づけと内発的動機づけの関連をしっかりと考える必要がある。

他の研究事例では、市村らが提案する「掃除を楽しくするゲーミフィケーションデバイス」[4]がある。これは掃除機にセンサを取り付けることで掃除機の運動をデータとして受け取り、それを twitter にて共有するものである。市村らはこの研究事例の中で「掃除

が面倒だと思う人や、継続しようとしてできない人、掃除をする時のモチベーションが上がらない人などが利用することを想定している。」[4]と述べていた。この事例においても内発的動機づけと外発的動機づけの関係が成り立っていると考えられる。市村らが想定している掃除に対して煩わしさを感じている人が外発的動機であるこのゲームに夢中になり、進めていくうちに「部屋が綺麗になる」と言う内発的動機づけが駆動しているといった状態になり得る。「部屋が綺麗になる」経験をした人は、今後ゲーミフィケーションがなくても、自発的に掃除をする場合があると考えられる。

## 第4章 研究室に来ることのゲーミフィケーション

本章では、第3章で述べたゲーミフィケーションによる動機づけを「研究室に来ること」に適用するために必要な要素を議論する。

### 4.1 ゲーミフィケーションの狙い

第1章で述べた通り、本研究の目的は学生を研究室に来させるだけではなく、研究室に来ることで「人と話す」メリットを自覚してもらうことにある。そのためにはゲーミフィケーションを通して、学生にメリットを理解してもらう必要がある。

上記の目標を達成するために、ゲームを通して学生同士が会話をする要素が必要不可欠であると考えた。会話を必要とするゲームの例として、「人狼ゲーム」の様な他人を騙し合うゲームも存在するが、本研究が目指すべき「人と話す」メリットはその様な人同士を対戦させる様なものではなく、人と協力し共に課題に立ち向かっていく様な協力がクリアに必要なゲームだと考えられる。理由としては、研究室は互いの研究を比較し合うものではなく、共に認め合い「研究」という困難に立ち向かっていくためにあると私が考えているからである。そこで、本研究では学生同士で会話が必要となる協力型のクイズゲームを設計した。ゲームを通して議論させることで「話し合う」ことの楽しさや有用性をわかってもらう。また、単純に研究室に所属する他人と話し合うことで親睦も深められると考えられる。

### 4.2 ゲームとしてのルール設計

4.1節で述べた協力型クイズゲームの詳細を述べる。以下の基本ルールの元でゲームが進行する。

1. 参加するプレイヤーからなるチームを決定する
2. クイズの問題が作成される
3. 問題文を分割する

4. 分割された問題がプレイヤー毎に分配される
5. 分配された問題を受け取ったプレイヤーが研究室に集まる
6. 問題文を完成させる
7. 問題を集まったプレイヤーで解く

1 から 7 の流れを繰り返すことによってゲームは進行する。繰り返す周期については、毎日を想定している。チームはプレイヤーの参加数とチームを構成する最小人数から決定する。具体的なアルゴリズムは 5.4 節で述べる。

以下にプレイヤーが 3 人でチームを 1 つ構成する場合のゲーム進行例を記述する。クイズの問題文が「世界一高い山は？」とする場合、以下の図の様に問題文を分割、配布される。図にある通り、プレイヤーが 3 人に指導教員を加え 4 分割しそれぞれ配布する。プレ

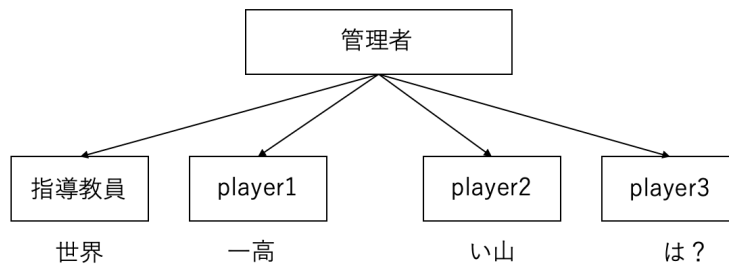


図 4.1: 問題文の分割

プレイヤー 3 人に対し、3 分割せずに指導教員を加える理由は、プレイヤーのみに問題を配布すると研究室に来ずとも、SNS を使うことで、問題文が完成してしまうためである。そのようなズルを防止するために、指導教員をチームに入れ監視する。また、ゲーム管理者と指導教員は違うため、指導教員も交えて問題に取り組むことができる。プレイヤーは研究室に集まり、「世界」「一高」「い山」「は？」のパーツを持ち寄って話し合い、議論する。解答はそれぞれが行うが、同じチームの解答は同じになるはずである。

問題に正解した場合、ゲームポイントが 1pt 配布される。ゲームポイントを一定まで貯めることで、プレイヤーのレベルがレベルアップする。表 4.1 にプレイヤーレベルの概要を示す。レベルアップにはそれぞれ必要な経験値 (ポイント) が必要である。また、レベルアップすることでクイズの問題に答えやすくなる特殊能力を獲得できる。参加する全プレイヤーのレベルを参照して、問題解答を行うステージのレベルを変更する。ここでいうステージとは、プレイヤーがクイズを解くための条件を変更するものである。表 4.2 にステージレベルの概要を示す。ステージ 1 ではクイズ 1 問にかけられる人数が 2 人になるようにチームが決定される。また、1 日に解答できる回数も 3 回となっている。最終ステージであるステージ 5 ではプレイヤー全員で問題に取り組める代わりに、1 日で解答できる回数も 1 回のみとなっている。プレイヤーレベルとステージの導入はクイズゲームをより面白

表 4.1: プレイヤーレベルの概要

	レベルアップに必要な経験値	能力内容
レベル 1	0pt	なし
レベル 2	10pt	1 週間に一度だけ問題に回答できる回数を上げることができる
レベル 3	15pt	問題回答時に発動できる能力。正解時に得られる経験値が 2 倍になる。 1 週間に 1 回のみ使える。
レベル 4	20pt	問題が出題される前日に使用できる能力。問題のパーツをランダムに 2 つ得ることができる。 1 週間に 1 度だけ使える。
レベル 5	30pt	ChatGPT が使える

表 4.2: ステージレベルの概要

	1 問毎に貰える経験値	1 問にかける人数	ステージレベルアップに必要な合計レベル	1 日に解答できる回数
ステージ 1	1pt	2 人	レベル 10	3 回
ステージ 2		3 人	レベル 20	2 回
ステージ 3		4 人	レベル 30	1 回
ステージ 4		5 人	レベル 40	1 回
ステージ 5		全員	レベル 50	1 回

くするために行った。これらはゲーミフィケーションにおける要素であり、4.3 節にて述べる。

### 4.3 狙いを達成するためのゲーミフィケーションの要素

4.2 節で述べたゲームのゲーミフィケーションの要素を一言で言うと、「ゲームにおける協力要素」が挙げられる。今日では、数多くあるゲームのジャンルのうち、オンラインゲームと呼ばれるものが普及している。例えば、League of Legends[5] では、5 人对 5 人のチームゲームである。このゲームにて勝利するためには、チーム同士の協力が必須である。このような「ゲームにおける協力要素」を今回のゲームに取り入れた。ただし、今回のゲームでは対人戦ではなく、あくまで 1 つの課題に取り組む際に協力することに主眼を置いている。

プレイヤーのレベルはゲーミフィケーションにおける「レベリング」と呼ばれる要素を元になっている。有名なゲームであれば、ドラゴンクエスト [6] における敵を倒すと経験値がもらえ、一定数に達するとレベルアップし、レベルに応じた新しい技やスキルを習得できる。この要素は単調なゲーム自体の内容が単調な時に、飽きをこさせない様になっている。ドラゴンクエストは、毎ターンコマンドを選択するゲームであるが、レベルアップの要素とそれに応じた新しい技を習得できる要素によって技の組み合わせなどの戦略性がゲームをより深くしている。

ステージのレベルはゲーミフィケーションにおける「ステージング」と呼ばれる要素を元になっている。広く知られているゲームで言えば、スーパーマリオ [7] である。このゲームでは、ステージをクリアするたびに、段々と難易度が上がっていきより難しいアクションを求められる。ステージングによって、ゲームのプレイヤーは「次のステージはどれくらい難しくなっているんだろう」と言うゲームの進行に関する期待や「自分はこのステー

ジまでクリアしてきたんだ」といったゲームの達成感を煽ることができる。

本研究では、クイズゲームを彩り、より学生が自主的にプレイしたくなるようなゲーミフィケーションの要素を取り入れた。これらの要素なしでは、3.2節で述べた様にただ退屈なゲームを淡々とこなすだけになってしまう。これでは、ゲームという外発的動機づけから「研究室に来て議論をするメリット」という内発的動機づけを駆動できなくなってしまうため、ゲームを設計する上で必須の要素であると言える。

## 4.4 ゲーム不参加者による不利益の回避

4.2節で述べたゲームの進行の中で、チームの作成の場面があるが、ステージ毎にかかる人数を分ける欠点として、チームメイトが1人でも不参加の場合そのチームは問題にチャレンジできないことが挙げられる。また、人数を分けた際にあぶれた人も出てくる。この問題に対する解決策として、チームが完成している人からあぶれている人に対して、チームにかかる人数の上限を無視して、チームに勧誘できる機能を追加する。

チームに勧誘した学生は自ら進んであぶれている人に対してコンタクトをとったという功績があるため、問題に正解した場合、追加で1pt獲得できる。また、あぶれていてチームに勧誘された学生は追加ptを獲得できないが、問題に正解した場合に通常通り1pt獲得できる。このゲームが学生にとって過度なプレッシャーになることを防ぐため、ゲーム不参加者に対するペナルティーは特に考えていない。

## 第5章 ゲーム進行を補助する bot の実装

本章では、第4章で述べたゲームの進行を円滑にするために利用した、Discord Bot の実装についての報告である。

### 5.1 Discord bot の利用

作成したゲームの点数及び問題の提出及びプレイヤーからの回答の管理を目的とした bot を開発した。bot の機能の説明の前に Discord[8] について述べる。Discord とは windows, Mac, ブラウザ上で動作するグループチャットアプリケーションである。管理者が新しいサーバーを建て、参加者を招待することによってグループ内でチャットや通話を楽しむ。また、1on1 でチャットができるダイレクトメッセージ機能もある。似たようなチャットアプリケーションとして、LINE や Slack が挙げられる。Discord bot とは、Discord のサーバー毎に新しい機能を追加できるものである。例えば、音楽をサーバー内で再生できる bot やサーバー内のメッセージの送受信の権限を細かく設定できる bot などがある。Discord 自身が作成した bot を導入できるのはもちろん、第三者が開発した bot もサーバーに導入できる。

本研究では、自身が開発した bot を自身が作成した Discord サーバーに導入した。その Discord サーバー内にゲーム参加者の Discord アカウントを招待し、Discord bot がこれらのアカウントを管理できるようにした。Discord 公式が Python 用に配布している「discord.py」ライブラリ [9] を使用することで Discord アプリ内で使用できる機能が Python プログラム内で開発できるため、Discord bot の開発言語は Python で行った。Discord bot の主な機能としては、ChatGPT を用いた問題の生成、問題の出題、ゲーム参加者からの回答受付、点数管理がある。ゲーム参加者の Discord アカウントは出題された問題の回答、現時点での点数確認ができる。図 5.1 は Discord サーバーの構成図である。

#### 5.1.1 Discord アカウントの作成

Discord を利用するためには、アカウントを作成する必要がある。以下に Discord アカウント作成手順を述べる。

1. Discord アカウント作成のためのサイトにアクセスし、メールアドレス、表示名、ユーザー名、パスワード、生年月日を入力する。(図 5.2)

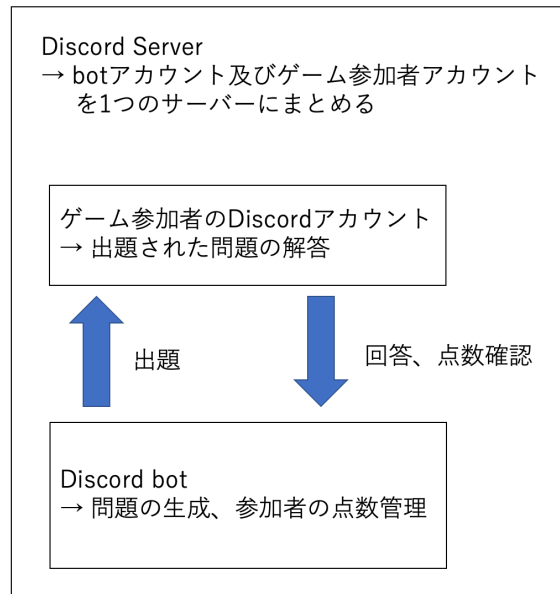


図 5.1: Discord 全体の構成図

2. セキュリティチェックを受ける。(図 5.3)
3. Discord アカウントが作成され、Discord にログインできるようになる。
4. Discord にログインすると、Discord から認証メールの確認が求められるので、メールから認証を行う。

図 5.4 が Discord ログイン画面である。

### 5.1.2 Discord サーバーの作成

Discord bot を利用するには、Discord 内にて自身で作成したサーバーが必要である。以下に Discord サーバー作成手順を述べる。

1. 図 5.4 の左側にある「+」をクリックする。
2. 図 5.5 の画面が表示されるので、「オリジナルの作成」を選択する。
3. サーバーのアイコンやチャンネルをそれぞれ設定する。

図 5.6 が実際に運用している Discord サーバーである。

## アカウント作成

メールアドレス \*

表示名

ユーザー名 \*

パスワード \*

生年月日 \*

年

▼

月

▼

日

▼

☐

Discord からアップデート、互換通知、セール情報のメールを受け取る。オプトアウトはいつでもできます。

はい

登録を行うことで、Discord のサービス利用規約 及び プライバシーポリシー に同意したものとみなされます。

既にアカウントをお持ちですか？

図 5.2: アカウント作成フォーム

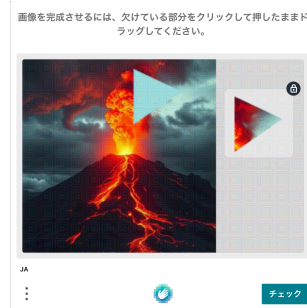


図 5.3: セキュリティチェック



図 5.4: Discord ログイン画面



図 5.5: Discord サーバー作成画面



図 5.6: 上野研 Discord サーバー

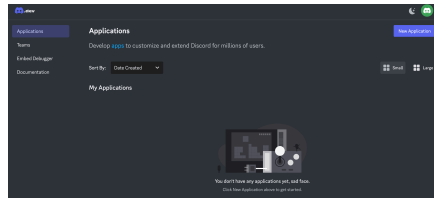


図 5.7: 新しい Bot の作成

図 5.8: Bot 名の決定

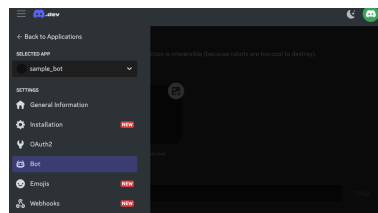


図 5.9: Bot タブの選択

### 5.1.3 Discord bot の導入

Discord アカウントを作成すると Discord bot 作成のためのポータルサイト [?] にログインできる。bot 作成のためには、Discord が発行した TOKEN が必要である。TOKEN とは Discord bot を識別するための文字列であり、これを Python プログラム内で定義することで Discord bot を利用できる。以下に Discord bot 作成のための TOKEN 取得方法について述べる。方法はショウによる解説記事 [10] と同じ方法である。

1. Discord bot 作成のためのポータルサイトにアクセスする。
2. 「New Application」をクリックする。(図 5.7)
3. bot の名前を決める。(図 5.8)
4. サイト内のタブを開き、「Bot」を選択。(図 5.9)
5. 「TOKEN」に移動し、「Reset Token」をクリックし新しいトークンを取得する。(図 5.10)

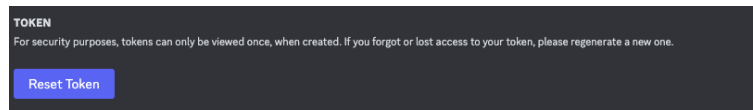


図 5.10: トークンの取得

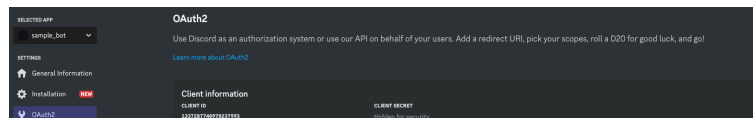


図 5.11: Discord 機能の選択 1

6. 再びサイト内のタブを開き、「OAuth2」を選択。(図 5.11)
7. 「URLGenerator」に移動し、「SCOPES」の「bot」にチェックを入れる。(図 5.12)
8. 「BOT PERMISSION」に移動し、「Read Message/View Channels」と「Send Messages」を選択する。(図 5.13)
9. Discord サーバーの招待 URL が生成されるので、URL を別タブで開き、5.1.2 節で作成したサーバーに招待する。

## 5.2 Python による Discord bot の開発

5.1.3 節で述べた TOKEN の取得後に Python のプログラムから Discord bot の実装が可能になる。ただし、Discord の機能を実装するための Python ライブラリは外部ライブラリであるため、ライブラリ管理システムである pip を用いて、以下のコマンドによりインストールする必要がある。

```
pip install discord.py
```

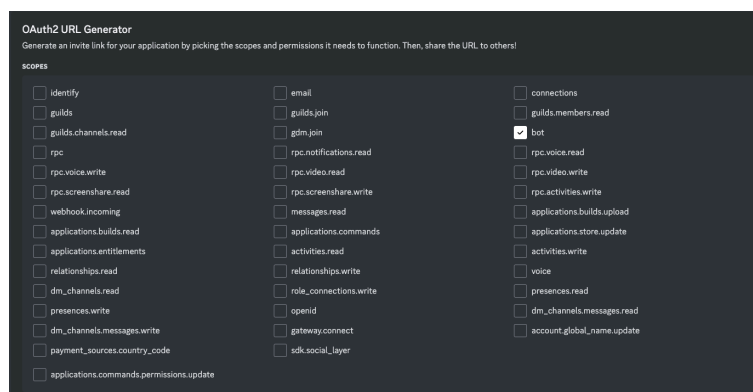


図 5.12: Discord 機能の選択 2

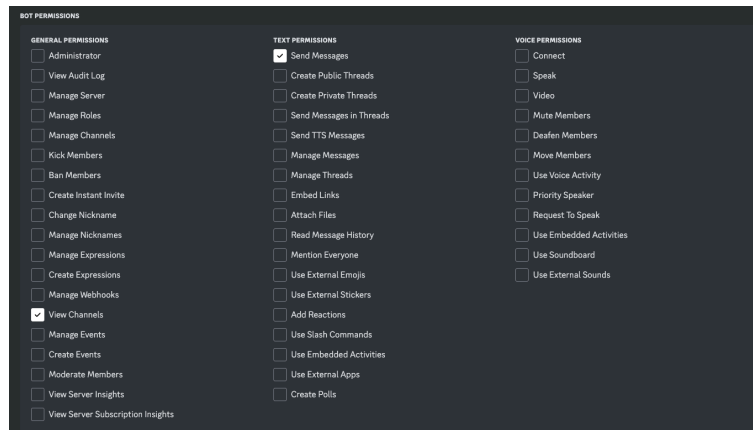


図 5.13: Bot 権限の選択

その後、Python プログラム本文内にて import 構文を用いて discord.py を実装した。

```
import discord
from discord.ext import commands, tasks
```

## 5.3 ゲーム参加者の管理

ゲーム参加者は必ずそれぞれが Discord のアカウントを作成し、5.1.2 節で述べた Discord サーバーに参加しなければならない。Discord Bot がゲーム参加者に対して DM(ダイレクトメッセージ)を送信するため、ゲーム参加者は Discord のユーザー ID を確認し Discord bot を作成するための Python プログラムにて定数として定義する必要がある。ユーザー ID とは各 Discord ユーザーがアカウント作成時に割り当てられる 17 または 18 文字の識別子である。

Python プログラム内において、ユーザー名をキーとして、ユーザー ID を値とした辞書型でゲーム参加者を定義した。以下に実際の python プログラムを示す。

```
player_id = {"maron" : 386135056323706883,
             "げんが" : 361419327037243392,
             "UC_fly18" : 826376106919067678,
             "としき" : 1329367045476847616,
             "アイリス" : 959108811333992478,
             "shohei_tanabe" : 1120305598421028865,
             "kairi_sasaki_t20" : 1331111102074851360,
             "からと" : 1329723939722756259}
```

## 5.4 チーム分けのアルゴリズム

クイズゲームを進めていく上でチーム分けのプログラムを実装した。具体的には、「チーム分けの人数を決定する」関数 `team` と「決定した人数から実際にチーム分けを行う」関数 `team_make` の2つを定義した。

「チーム分けの人数を決定する」関数 `team` はユーザー数 (`int` 型) とチームを構成する最小人数 (`int`) 型を受けとる。出力は、チームを構成する最小人数を満足するようにチーム分けがされ、チームの数と同じになるようにリストの要素の数を決定し、そのリストに各チームの人数を格納する。さらにそのリストのサイズを見ることでチームの数を出力させる。関数の定義の形は以下のようになる。

```
def team(player_number, few_number): #(プレイヤーの数, チームの最小人数)
    global how_many_team
    a = []
    if player_number < few_number:
        print("チームが作れません")
    else:
        div = player_number // few_number # チームの数
        a = [0] * div
        for i in range(player_number):
            if i < player_number: # 不要な条件を削除
                a[i % div] += 1 # i を div で割った余りをインデックスとして使用
        how_many_team = len(a)
    return a, how_many_team
```

関数の細かい内容について述べる。

```
    global how_many_team
```

の部分において、“`how_many_team`”変数は他の関数でも使用されるグローバル変数であるため、この関数での変更をコード全体に反映させるためである。次に、

```
    a = []
```

の部分について、このリストは、チーム数だけ要素数が決まり、中の数値は各チームが所属する人数が入る。このリストの細かい処理はのちに行う。

```
    if player_number < few_number:
        print("チームが作れません")
    else
        ...
```

の部分では、ゲームに参加するユーザー数がチームを構成する最小人数を下回っていた場合、ゲームができない(チームが組めない)ため print 文を出して関数を終了する。それ以外の場合は...にある処理(以下に説明する処理)を実行する。

```
div = player_number // few_number
a = [0] * div
```

上記の部分では、ユーザー数からチームを構成する最小人数を割り商を求める。この商はチーム数を表しており、そのチーム数だけ中の数値が0のリストを作成する。例えば、「8人のユーザーを最低3人からなるチームを作る」場合、行う処理は

```
8 // 3 = 2
```

となる。これによりチーム数が2であることがわかる。この数だけ中の数値が0のリストを作成すればいいので、以下の処理を行う。

```
a = [0] * 2
```

つまり、以下のようにリストが作成される。

```
a = [0, 0] # チームが2つなため、要素数は2つ
```

その後ユーザーの数(つまり8回)だけ、以下の繰り返し処理を行う

```
for i in range(player_number):
    if i < player_number:
        a[i % div] += 1
```

この繰り返しの処理の部分を先程の例えで表すと以下のようなになる。

```
for i in range(8):
    a[i % 2] += 1
```

これにより、ユーザー8人の数だけ元々0であったリストの数値を増やす。最初のループでは、

```
a[0 % 2] += 1
```

が評価され、

```
a[0] = 1 # チーム1の人数を+1する
```

となる。このループが終わった後に出力される2つ(=チーム数)のリストの数値はそれぞれのチームに所属するプレイヤーの数となっている。具体的にはリストの要素の詳細は以下のようなになる。

```
a = [4, 4] # [チーム 1 の構成人数, チーム 2 の構成人数]
```

ループ後の処理は

```
how_many_team = len(a)
```

とすることで、チーム数を `how_many_team` 変数に格納する。この関数は以下のように、それぞれのチームにおける人数を格納したリスト `a` とそのリストの長さ (=チームの数) を出力する。

```
return a, how_many_team
```

「決定した人数から実際にチーム分けを行う」関数 `team_make` は 5.3 節で定義したユーザー名 (str 型) : ユーザー ID (int 型) の辞書型と、チームを構成する最小人数 (int 型) を受け取る。出力は、各チームの人数が、それぞれの要素数の中に格納されたリスト及び「チーム分けの人数を決定する」関数から受け取ったチームの数をそのまま返す。関数の定義は以下のようなになる。

```
def team_make(player_id, team_number): #(参加者 → チーム)
    if len(player_id) < team_number:
        print("チームが組めません")
    else:
        player_list = list(player_id.values())
        # リストをランダムに入れ替え
        random.shuffle(player_list)
        # プレイヤーごとのチーム割り当て辞書
        team_assignments = {}
        count = 0
        # プレイヤーを割り振る
        team_div, how_many_team = team(len(player_id), team_number)
        for i, a in enumerate(team_div):
            for j in range(a):
                team_assignments[player_list[count]] = i + 1
                count += 1
        return team_assignments, how_many_team
```

関数の細かな説明としては、まず、

```
if len(player_id) < team_number:
    print("チームが組めません")
else:
    ...
```

の部分において、`len(player_id)` はユーザー数を表しており、それ以下のチーム数ではゲームができないため、`if` 文の条件を満たす場合、`print` 文を表示し関数を終了させる。満たさない場合、ゲームが進められるため `else` 以降の処理を実行する。

```
player_list = list(player_id.values())
random.shuffle(player_list)
```

の部分について、この関数の第一引数である `player_id` は、5.3 節に記述されているように、ユーザー名 (`str` 型) : ユーザー ID (`int` 型) の辞書型で定義されている。関数内の処理においてはユーザー名の部分は必要ないため、

```
list(player_id.values())
```

を用いて、ユーザー ID のみのリスト型として抽出している。その後、このリスト型に格納された値をランダムにシャッフルすることで、以下の `for` 文が呼び出されるごとに行われるチーム分けをランダムに行われるようにしている。

```
team_assignments = {}
```

では、以下の処理において作成されるユーザー ID (`int` 型) : チームナンバー (`int` 型) を格納するための空の辞書型を先に定義している。チームナンバーは 1 から始まるチームを分けるための数である。(2 つチームがある場合チームナンバーは 1, 2 となる。)

```
count = 0
```

では、以下の `for` 文内にて使用されるカウンタを定義している。

```
team_div, how_many_team = team(len(player_id), team_number)
```

においては、先に説明した、「チーム分けの人数を決定する」関数から、各チームに振り分ける人数のリストを `team_div` に、チームの数を `how_many_team` にそれぞれ受け取っている。これまで説明した変数 `player_list`、`team_assignments`、`count`、`team_div` を用いて以下の `for` 文の処理を実行する。

```
for i, a in enumerate(team_div):
    for j in range(a):
        team_assignments[player_list[count]] = i + 1
        count += 1
```

この `for` 文では、振り分けるべき人数を格納したリスト `team_div` に沿って、実際のプレイヤー ID と所属するチームナンバーを辞書 `team_assignments` に格納している。「チーム分けの人数を決定する」関数の説明でも使用した「8 人のユーザーを最低 3 人からなるチームを作る」ことをこの関数でも例えるとすると、`team_div` の内容は以下ようになる。

```
team_div = [4, 4] # (チーム1の構成人数, チーム2の構成人数)
```

よってネスト構造の外側における最初の for 文は以下ようになる。

```
for i, a in enumerate([4, 4]):  
    ...
```

ここでネスト構造となっている for 文の外側において、最初のループを考える。enumerate は i に 0 からの回数を格納し、a には enumerate 内のリストから最初の要素が格納されるため

```
for 0, 4 in enumerate([4, 4]):  
    ...
```

となる。この状態におけるネスト構造の中の for 文は

```
for j in range(4):  
    team_assignments[player_list[count]] = i + 1  
    count += 1
```

となる。この for 文は空の辞書 team\_assignments に、あらかじめプレイヤー ID(int 型) がシャッフルされ、格納されたリスト player\_list をキーとし、そのプレイヤー ID が所属するチームナンバーが値となるように追加させるためのものである。この for 文の最初のループを見ると、

```
for j in range(4):  
    team_assignments[player_list[0]] = 0 + 1  
    count += 1
```

このループにおいて、以下の処理が実行される。

```
team_assignments = {player_list[0] : 1}  
count += 1
```

range(4) が示すように、ループを 4 回行い、リスト player\_list[0] から player\_list[3] をキーとしてチームナンバー 1 が値となる以下のような辞書ができる。

```
team_assignments = {player_list[0] : 1, player_list[1] : 1  
    , player_list[2] : 1, player_list[3] : 1}
```

ループ終了時、ネスト構造における外側の for 文に処理が戻り、以下の処理が実行される。

```
for i = 1, a = 4 in enumerate([4, 4])  
    for j in range(4)  
        team_assignments[player_list[count]] = 1 + 1  
        count += 1
```

最初に紹介した for 文との違いは i の値が 1 となっているため、team\_assignments に格納されるチームナンバーの値が 1 ではなく、2 となる点である。また count は初期化されないため、先程の for 文からカウンタは引き継がれる。つまり、以下の処理が辞書 team\_assignments になされる。

```
team_assignments = {player_list[0] : 1, player_list[1] : 1
, player_list[2] : 1, player_list[3] : 1, player_list[4] : 2,
player_list[5] : 2, player_list[6] : 2, player_list[7] : 2}
```

その後、for ループのネストを抜け、作られた辞書 team\_assignments 及びチーム数を格納した how\_many\_team が出力される。

```
return team_assignments, how_many_team
```

これにより、「8 人のユーザーを最低 3 人からなるチームを作る」プログラムを処理した結果、「チーム分けの人数を決定する」関数が出力した 4 人 2 チームを「決定した人数から実際にチーム分けを行う」関数により実際にチーム分けがなされる。もちろん関数に渡す引数の値を変えることで上記の例以外のチーム分けも行うことができる。

## 5.5 ChatGPT を活用した出題

ChatGPT[11] とは「OpenAI 社が開発し、公開している AI を用いたチャットボット」[11] である。普段の疑問に答えてくれる他、アイデア出しなどの要望などもプロンプトとして入力すると、それに沿った回答を AI を用いて答えてくれる。また、OpenAI 社はチャットサービスの他に API を用いることで自分が作成したプログラム内に ChatGPT に向けた、プロンプトとその応答を実装できる。本研究では、その ChatGPT の API を用いることで、Discord bot 内に毎日ランダムに問題を生成する機能を実装した。

Python プログラム内に ChatGPT の API を実装する手順として以下が挙げられる。

1. API キーの取得
2. OpenAI ライブラリのインストール

1 の API キーとは Python プログラム上に API を実装するために必要となる文字列のことである。これは OpenAI の公式プラットフォームにログインし、サイトが表示する手順に沿って取得することができる。また、API キーの取得自体は無料でできるのだが、Python プログラム内で使用するためには、あらかじめお金を OpenAI のプラットフォームに支払わないといけない。入金 は 5 ドルから 95 ドルの間でユーザーが任意の値で行うことができる。あらかじめ入金された金額から、API を使用した時に入出力される文字列を参照してプログラム実行時に自動で引き落とされる仕組みとなっている。2 の OpenAI ライブラリのインストールについては、Python の標準ライブラリに OpenAI のライブラリが存在しないため、外部ライブラリとして導入する必要がある。導入は pip を用いて以下のコマンドを実行し、行なった。

```
pip install openai
```

その後、以下の様に、Python プログラム内にてクラスのオブジェクトを作成する際に 1 で取得した API キーを用いることで、ChatGPT の API が使用できる様になる。

```
# OpenAI API キーを設定
client = OpenAI(api_key = config.OPENAI_KEY)
```

ChatGPT の API を用いることの目的は、Ueno\_Quest のための問題の生成をさせることである。Python プログラムを実行するたびに、あらかじめ定義されているプロンプトから API を通して、ChatGPT の出力を文字列として受け取る。以下に定義したプロンプトを示す。

```
messages = [
    {"role": "system", "content": "あなたは優れた謎なぞ作成者です。"},
    {"role": "user", "content": """"一次方程式を一文だけ出力し、
それをキー (str 型) として、解答を値 (int 型) とする辞書型 ({問題: 答え})
を Python の辞書の形で出力してください。
解答は **正の整数 (1 以上の自然数)** であること。

余計な説明やコメントは不要です。

出力例:
{"6x - 60 = 240": 50}
"""}
]
```

上記のプロンプトの狙いは、ChatGPT に一次方程式を出力させ、キーが問題文、値がその問題の答えとなるような辞書型を出力してもらうことである。

```
{"role": "system", "content": "あなたは優れた謎なぞ作成者です。"}
```

の部分は、ChatGPT のシステムにおける役割を与え、望むような答えを出力させる。今回はより、ユニークな問題を生成してもらうために「あなたは優れた謎なぞ作成者です。」と定義した。

```
{"role": "user", "content": """"一次方程式を一文だけ出力し、
それをキー (str 型) として、解答を値 (int 型) とする辞書型 ({問題: 答え})
を Python の辞書の形で出力してください。
解答は **正の整数 (1 以上の自然数)** であること。

余計な説明やコメントは不要です。
```

出力例:

```
{"6x - 60 = 240": 50}
"""}
```

上記の部分は実際に送信するプロンプトのうち、ChatGPT が回答して欲しい具体的な内容が記述してある。Python プログラムは返り値として辞書型のみを期待しているため、プロンプトにも辞書型で出力する旨を記述している。また、出力例を示すことで望んだ回答により近いものを得られるようにしている。

上記の様にプロンプトを文字列として定義して、ChatGPT からの回答を得られる関数 `make_question` を Python プログラムにて定義した。以下に関数を示す。

```
def make_question(messages):
    # OpenAI の API を呼び出し、与えられたメッセージに対する応答を取得
    completion = client.chat.completions.create(
        model="gpt-4",
        messages=messages
    )
    try:
        response = eval(completion.choices[0].message.content)
        if is_single_key_value_dict(response) ==
        True and check_none_in_dict(response) == True:
            return response
        else:
            make_question(messages)

    except Exception as e:
        make_question(messages)
```

関数の細かい説明としては、先程記述したプロンプトを関数の引数 `message` に格納し、関数を呼び出している。

```
completion = client.chat.completions.create(
    model="gpt-4",
    messages=messages
)
```

上記の意味としては、先程プロンプトにて定義した `message` の内容を ChatGPT に入力した結果が変数 `completion` に辞書型として格納される。`client` は `openai` のクラスオブジェクトを作成する際に定義したものである。`chat.completion` はクラス名、`create` はメソッドの名前を示している。メソッドの引数の

```
model="gpt-4"
```

は ChatGPT のバージョンの一つである”GPT-4o”を使用することを指定している。

```
message=message
```

は左側の message にプロンプトとして定義した message を格納している。

```
try:
    response = eval(completion.choices[0].message.content)
    if is_single_key_value_dict(response) ==
    True and check_none_in_dict(response) == True:
        return response
    else:
        make_question(messages)

except Exception as e:
    make_question(messages)
```

の部分は、変数 completion に格納された辞書型から、メッセージの内容だけを取り出して response に格納している。is\_single\_key\_value\_dict() 及び check\_none\_in\_dict() は予め定義した自作関数であり、response が辞書型かつ 1 組のキーと値のペアを持つかの確認と辞書型のキーまたは値が None になっていないかを判定している。これらの条件を満たす場合、プロンプトにて定義した「キーが問題文、値がその問題の答えとなるような辞書型」を満足するためそのまま response を返して関数を終了する。仮に条件を満たさない場合、ChatGPT からの回答に何かしらのエラーが含まれていると判断し、もう一度この関数自体を実行し直す。Python の例外処理を用いることで、通信エラー等プログラムにおける何かしらの外的な問題が発生しても、もう一度この関数自体を実行し直す様にしている。

関数 make\_question から出力される辞書型から、(問題, 答え) のタプル型を取り出すための関数 today\_quest を定義した。

```
def today_quest():
    # 問題を生成する
    q = list(make_question(messages).items())
    return q[0] #(問題, 答え) のタプル
```

この関数の役割はとても簡単で、list() 構文を用いることで関数 make\_question の出力を辞書型→リスト→タプルの順で変換し取り出している。タプルを出力として利用する理由は後に 5.6 節で述べるアルゴリズムの処理が簡単になることである。

## 5.6 各チームに問題を割り振るアルゴリズム

5.4 節で定義した関数 `team_make` 及び 5.5 節で定義した関数 `today_quest` を用いて、「日替わりで確定したチーム毎に問題を割り当てる」関数 `team_make` を定義した。定義した関数を以下に示す。関数の出力は、チームナンバーがキー、チーム毎に割り当てられる問題と答えをそれぞれ値とした辞書 `team_quest` と `team_ans` である。

```
def team_quest(team_number):
    team_quest = {}
    team_ans = {}
    team, how_many_team = team_make(player_id, team_number)
    team_keys = list(set(team.values()))

    for i in range(how_many_team):
        team_quest[team_keys[i]], team_ans[team_keys[i]] = today_quest()
    return team_quest, team_ans
```

関数の引数となる `team_number` は 5.4 節で定義した関数 `team_make` の引数と同じ、チームを構成する最小人数 (`int` 型) である。

```
team_quest = {}
team_ans = {}
```

これらの空の辞書は、関数が最終的に出力するためのものであり、関数の冒頭部分で先に定義している。

```
team, how_many_team = team_make(player_id, team_number)
```

上記の部分は 5.4 節で定義した関数 `team_make` の返り値であるプレイヤーの ID と所属するチームナンバーの辞書を変数 `team` に、全体のチーム数を変数 `how_many_team` に格納している。

```
team_keys = list(set(team.values()))
```

変数 `team_keys` に全体のチーム数がそれぞれ格納されるリストを作成している。仮に変数 `team` の値が

```
team = {player_list[0] : 1, player_list[1] : 1,
        player_list[2] : 2, player_list[3] : 2}
```

であった場合、`list(set(team.values()))` は辞書 `team` から値であるチームナンバーを取り出し、重複しているチームナンバーを 1 つにまとめたリストを作成する。上記の例えの場合以下のリストが `team_keys` に格納される。

```
team_keys = [1, 2]
```

上記の例えをそのまま用いて、以下の for 文の説明を行う。

```
for i in range(how_many_team):  
    team_quest[team_keys[i]], team_ans[team_keys[i]] = today_quest()
```

range(how\_many\_team) のうち、how\_many\_team はチーム数を表しているため、今回の例えの場合 2 が格納されている。よって for 文は

```
for i in range(2)  
    ...
```

の様に表すことができる。これにより繰り返す回数が 2 回であることがわかる。そのうち最初のループでは、 $i = 0$  となるため以下の処理がされる。

```
team_quest[team_keys[0]], team_ans[team_keys[0]] = today_quest()
```

これは、関数冒頭で定義した辞書 team\_quest 及び team\_ans に対して、team\_keys[0] をキー、5.5 節で定義した関数 today\_quest からの返り値を値とする辞書を追加している。today\_quest は ChatGPT が作成した (問題, 答え) のタプルを返し、team\_keys[0] の値は 1 であるため、以下の様なキーと値のペアが追加される。

```
team_quest = {1 : (ChatGPT が生成した問題文 1)}  
team_ans = {1 : (ChatGPT が生成した答え 1)}
```

次のループでは、 $i = 1$  となるため、

```
team_quest[team_keys[1]], team_ans[team_keys[1]] = today_quest()
```

の処理が実行される。また、ループ毎に関数 today\_quest は実行されるため、最初のループで生成される問題文と答えと 2 回目で生成される問題文と答えは違うものとなる。上記の式によって辞書 team\_quest 及び team\_ans は以下の様になる。

```
team_quest =  
    {1 : (ChatGPT が生成した問題文 1), 2 : (ChatGPT が生成した問題文 2)}  
team_ans =  
    {1 : (ChatGPT が生成した答え 1), 2 : (ChatGPT が生成した答え 2)}
```

これらの辞書が意味しているのは、冒頭で述べた通り、チームナンバーがキー、チーム毎に割り当てられる問題と答えをそれぞれ値とした辞書である。この関数の返り値としてこれらの辞書が出力される。

```
return team_quest, team_ans
```

## 5.7 問題分割のアルゴリズム

Ueno\_Quest はチーム毎に問題を生成し、その問題文を分割してユーザーに割り当てる。以下に Python プログラム内に定義した問題を分割する関数 `divquestion` を示す。

```
def divquestion (question, count):
    if count <= 0:
        raise ValueError("分割後の要素数は正の整数である必要があります。")

    length = len(question)
    base_size = length // count    # 各要素の基本の長さ
    remainder = length % count    # 余り

    result = []
    start = 0

    for i in range(count):
        # 余りがある間は1文字多く割り当てる
        end = start + base_size + (1 if i < remainder else 0)
        result.append(question[start:end])
        start = end

    return result
```

引数としての入力には第一引数 (`question`) に問題文 (`str` 型)、第二引数 (`count`) に何分割するかを指定する整数 (`int` 型) とする。関数の出力は、分割された文字列 (`str` 型) である。今回の説明では、例として第一引数に問題文「世界一広い湖は？」を渡し、第二引数に2分割させるために2を渡すと仮定する。以下に関数の細かい説明を行う。

```
if count <= 0:
    raise ValueError("分割後の要素数は正の整数である必要があります。")
```

最初の `if` 文では第二引数に指定した分割する整数が負の場合、問題文が分割できないため、例外として排除している。

```
length = len(question)
base_size = length // count
remainder = length % count
```

上記の部分では、文字列 `question` = 「世界一広い湖は」 の文字数をカウントし、その長さを `int` 型として `length` に格納している。次に格納した長さを分割したい整数で割、商と剰余をそれぞれ `base_size` と `remainder` に格納する。今回の例では、「世界一広い湖は」の文字列は7文字であり、2分割したいため以下の処理がなされる。

```
length = 7
base_size = 7 // 2 = 3
remainder = 7 % 2 = 1
```

次に空のリストである `result` と文字列分割のために使用する変数 `start` を定義以下の様に定義している。

```
result = []
start = 0
```

次に以下の `for` 文の説明を行う。

```
for i in range(count):
    end = start + base_size + (1 if i < remainder else 0)
    result.append(question[start:end])
    start = end
```

`count` は2分割を行うため2となっているので

```
for i in range(2):
    ...
```

となる。つまりこの `for` 文は2回繰り返すことを意味する。

```
end = start + base_size + (1 if i < remainder else 0)
result.append(question[start:end])
start = end
```

最初のループについて、`start` の初期値は0であり、`base_size` の値は3である。`remainder` は1であり `i = 0` の時には `()` 内の `if` 文を満足するため

```
end = 0 + 3 + 1 = 4
```

となり、`end` には4が格納される。この4は `question` に格納された文字列のうち左から4番目の文字列を抜き取ることを表している。次の

```
result.append(question[start:end])
```

は `start` と `end` の値がそれぞれ0と4であるため、リスト `result` に、文字列 `question` のうち0から4番目の文字を追加する。よって `result` の値は以下のようになる。

```
result = ["世界一広い"]
```

ループの最後に、`start` に `end` の値が格納される。2回目のループでは、`start` の値が4であり、`base_size` の値は3である。`remainder` は1であり `i = 1` の時には `()` 内の `if` 文を満足しないため

```
end = 4 + 3 = 7
```

となり、end には7が格納される。

```
result.append(question[start:end])
```

は start と end の値がそれぞれ4と7であるため、リスト result に、文字列 question のうち4から7番目の文字を追加する。よって result の値は以下のようになる。

```
result = ["世界一広い", "い湖は"]
```

関数の最後にて、このリスト result を出力する。

```
return result
```

## 5.8 答え合わせと解答回数制限

ゲームに参加するプレイヤーはクイズに対する解答として Discord bot のコマンド機能を使用する。例えば、「世界一広い湖は」という答えに対して「琵琶湖」と解答する場合、以下の様になる。

```
!answer 琵琶湖
```

「琵琶湖」と記述する前の「!answer」がコマンドであり、「!」はコマンドの接頭辞を表している。以下にこのコマンドを実装するにあたり作成した関数 answer を示す。

```
@bot.command()
async def answer(ctx, *, user_answer: int):
    global player_answer
    global player_right
    global player_point
    user_id = ctx.author.id
    player_right = load_variables_from_json(player_right_log)
    player_right = {int(key) : value for key, value in player_right.items()}
    if player_right[user_id] == False:
        await ctx.send("解答は1日1度までです")
        return

    if player_answer.get(user_id) == user_answer:
        await ctx.send("正解です")
        point_add(player_point, user_id)
```

```

player_right[user_id] = False
save_variables_to_json(player_right, player_right_log)
else:
    await ctx.send("違います")
    player_right[user_id] = False
    save_variables_to_json(player_right, player_right_log)

```

この関数は非同期処理によって行われる関数であり、関数を定義する前に記述されている

```
@bot.command()
```

はこの関数が discord.py によって提供されていることを明示すると同時に、関数名である answer を使用してコマンドを実行することを示している。関数定義の部分は

```
async def answer(ctx, *, user_answer: int):
```

である。引数である ctx にはこのコマンドを実行したユーザーの ID 情報とこのコマンドが実行された場所(どのサーバーのどのチャンネルで実行されたか)が格納されている。同じく引数である user\_answer は、ユーザーがこのコマンドを呼び出す時に記述した文字列(上記の例で言えば「琵琶湖」)が格納されている。

```

global player_answer
global player_right
global player_point

```

の部分は、プログラム全体で使用するグローバル変数を定義している。player\_answer はプレイヤー毎に割り当てられた正解の答えが格納されている。player\_right にはプレイヤーがその日解答できるか否かを bool 型で格納している。player\_point にはプレイヤーが現在保持している経験値を格納している。

```
user_id = ctx.author.id
```

上記の部分は user\_id 変数にこのコマンドを入力したユーザーの情報を格納している。

```
player_right = load_variables_from_json(player_right_log)
```

の部分は json ファイルからデータを取り出している。この Python プログラムでは、bot がバグり、プログラムが停止してしまった際に変数に格納されているデータが消えてしまうことを防ぐためにプレイヤーの解答権とプレイヤーが現在保持している経験値 (point) を外部 json ファイルに保存している。player\_right\_log.json はそのうち、ユーザー ID をキー、プレイヤーの解答権を bool として値ので保存している json ファイルである。

```

player_right =
    {int(key) : value for key, value in player_right.items()}

```

の部分は、json ファイルに格納されている辞書から、プレイヤーの ID を int 型に変換している。

```
if player_right[user_id] == False:
    await ctx.send("解答は1日1度までです")
    return
```

の部分は、プレイヤーが1日の解答権を有していない場合に、コマンドを入力したユーザーに対し、解答権がないことを出力し関数を終了する。

```
if player_answer.get(user_id) == user_answer:
    await ctx.send("正解です")
    point_add(player_point, user_id)
    player_right[user_id] = False
    save_variables_to_json(player_right, player_right_log)
else:
    ...
```

の部分は、プレイヤーが解答権を有していた時と、プレイヤーの解答が正解であった場合に、コマンドを入力したユーザーに対し正解であることを出力している。その後、プレイヤーの解答権を False として失わせている。point.add 関数は指定したプレイヤーのポイントを増加する関数であり、それによりプレイヤーのポイントを増やしている。最後に変動したプレイヤーの解答権とポイントを json ファイルに保存し、関数を終了している。

```
else:
    await ctx.send("違います")
    player_right[user_id] = False
    save_variables_to_json(player_right, player_right_log)
```

は、プレイヤーが解答権を有していた時と、プレイヤーの解答が不正解であった場合に、コマンドを入力したユーザーに対し不正解であることを出力している。その後、プレイヤーの解答権を False として失わせている。最後に変動したプレイヤーの解答権を json ファイルに保存し、関数を終了している。

## 第6章 結論と今後の課題

### 6.1 結論

「研究室に来ること」に対してゲーミフィケーションを適用することにあたり、研究室に来ることのメリットを学生にどのように知ってもらうかを主に考えた。自分の経験からその「研究室に来る」メリットとは人と話すことである。そのメリットは多くの学生が研究室に来ることによってより多くの効果を生むため、研究室に所属する学生がなるべく来てくれるようなゲームを設計した。ゲーミフィケーションについて勉強、調査する過程で、内発的動機づけや外発的動機づけの動機づけの種類、レベリングやステージングといったゲーミフィケーションの要素を知ることができた。また、先行研究にて実際にゲーミフィケーションが使用されている事例を参考にして、その研究と自分がしたいゲーミフィケーションの違いを比較、検討した。さらに、設計したゲームをチャットアプリである Discord の bot を Python を用いて開発することで、ゲーム実装を試みた。実装に関しては、ゲーム参加者の管理や ChatGPT を用いた問題の生成、問題分割や解答のアルゴリズムについて考えた。ゲームの運用に関してはまだできていないため、この過程で生まれる問題点や評価については今後の課題としたい。

### 6.2 今後の課題

前節で述べた通り、ゲームが実際に運用できていないため、その評価ができていないことが一番の課題である。ステージングやレベリングといったゲーミフィケーションの要素が果たして、今回の研究で設計したゲームに適切であったかを実際の研究室の来訪数などのデータを元に考えてみたい。5.5 節で述べた、ChatGPT の出題に関して、出題する問題を一次方程式のみにしているためクイズの問題としては物足りないと感じている。学生同士の議論をより活発とする様な、興味深い問題の生成についても今後の課題としたい。

# 謝辞

本研究を進めるにあたり、どの様にしたら研究室に人が来るのかといった自分の興味があつた課題に対し、「ゲーミフィケーション」という自分が知らない手法を提案して下さった上野准教授に感謝申し上げます。さらに、内発的動機づけと外発的動機づけの位置付けについて混乱することがあつた時に、上野准教授に助言をいただけなかった場合自分の研究は成り立たなかつたと思っています。研究を進めていく上で「人と話す」メリットについて、実際に自分の研究の話を聞いて議論に応じて下さった上野研究室修士課程学生の武田氏と佐藤氏に感謝申し上げます。

## 参考文献

- [1] 井上 明人. ゲーミフィケーション<ゲーム>がビジネスを変える. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.
- [2] ポケットピカチュウ. <https://www.nintendo.co.jp/n09/pokepika/index.html>
- [3] 鳴海 拓志, 谷川 智洋, 廣瀬 通孝. ゲーミフィケーションを利用した研究活動の可視化と活性化. The 29th Annual Conference of the Japanese Society for Artificial Intelligence, 2015
- [4] 市村 哲, 矢澤 崇史, 戸丸 慎也, 渡邊 宏優. 家事をゲーミフィケーションする試み〜掃除への適用〜. 「マルチメディア, 分散, 協調とモバイル (DICOM2014) シンポジウム」平成 26 年 7 月
- [5] RIOTGAMES LeagueofLegends <https://www.leagueoflegends.com/ja-jp/>
- [6] SQUARE ENIX ドラゴンクエスト <https://www.jp.square-enix.com/game/?s=1>
- [7] Nintendo スーパーマリオブラザーズ <https://www.nintendo.com/jp/famicom/software/smb1/index.html>
- [8] Discord <https://discord.com/>
- [9] discord.py <https://discordpy.readthedocs.io/ja/latest/index.html>
- [10] Qiita 「[Python]Discord Bot のつくりかた」 [https://qiita.com/shown\\_it/items/6e7fb7777f45008e0496](https://qiita.com/shown_it/items/6e7fb7777f45008e0496)
- [11] OpenAI ChatGPT <https://openai.com/ja-JP/chatgpt/overview/>