修士論文

アイトラッキング指標に基づく
プレイヤの退屈検出とイベント生成

池田 龍治

主指導教員 池田 心

北陸先端科学技術大学院大学
融合科学共同専攻
（融合科学）

令和8年3月

アイトラッキング指標に基づくプレイヤの退屈検出とイベント生成 (Eye-Tracking– Based Player Boredom Detection and Event Generation)

北陸先端科学技術大学院大学 学生番号 2450001
氏名 池田 龍治
主任研究指導教員氏名 池田 心

1. はじめに

ゲームにおいて，プレイヤが楽しさを維持し続けるためには，スキルレベルと挑戦レベルのバランスが重要であることが，フロー理論において示されている [1]．簡単すぎる課題は退屈をもたらし，難しすぎる課題は不安をもたらすため，プレイヤの状態に合わせた動的難易度調整（Dynamic Difficulty Adjustment; DDA）が古くから研究されてきた [2]．従来の DDA 手法は，クリア時間やスコアなどのゲーム内成績に基づいて難易度を調整するものが一般的であるが，これらはプレイヤの内面的な心理状態を直接捉えているわけではなく，本来必要な調整とは異なる調整を行ってしまう恐れがある．プレイヤの心理状態を推定するために生体情報を用いる試みも存在するが，脳波や心拍数などの計測には装着型センサが必要であり，自然なプレイ体験を阻害する可能性がある．一方，瞳孔径は認知負荷や覚醒水準と関連することが報告されており，Strauch らは単純なゲーム Pong において瞳孔径とゲーム難易度の対応関係を確認し，瞳孔径に基づく DDA の可能性を示した [3]．そこで本研究では，非接触で取得可能な視線情報に着目し，Tobii 社製の非装着型アイトラッカー [4] を用いてプレイヤの瞳孔径および視線データを取得し，退屈状態をリアルタイムに推定するシステムを構築した．推定結果に基づいてゲーム内イベントの発生タイミングを動的に制御することで，プレイヤの退屈を抑制することを目指す．対象ゲームには，MOD による拡張性が高く研究プラットフォームとしても広く利用されている Minecraft[5] を採用した．

2. 研究方法

本システムは，視線計測および退屈推定を担う Python 計測・推定器と，ゲーム内制御を担う Minecraft Mod（Fabric Mod）の 2 つのサブシステムから構成される．両者は TCP ソケット通信（JSONL 形式）により接続され，独立した開発・変更が可能である．Python 側では，Tobii SDK を通じて視線座標および瞳孔径データを取得し，リングバッファに蓄積する．一定間隔でウィンドウベースの特徴量（視線速度平均，視線分散，瞳孔径平均，瞳孔径標準偏差）を算出し，練習フェーズで計測したベースラインに対する z 値として標準化した上で，sigmoid 関数により 0 – 1 の退屈スコアへ変換する．退屈スコアが閾値を超えると退屈状態（ON）と判定し，ヒステリシス制御により状態遷移の安定化を図っている．Minecraft 側では，Python 側から受信した退屈推定結果に基づき，タスク種別に応じたイベント発火を行う．実験環境として，石ブロックで構成された $25 \times 25 \times 10$ の閉鎖型闘技場を構築し，視覚的要因による瞳孔径への影響を低減した．プレイヤはゾンビ等の敵 Mob と戦闘を行い，刺激イベントとして敵 3 体の追加出現（ADD_ENEMIES）および装備付きゾンビ 1 体の出現（SPAWN_ELITE）を定義した．実験では 2 種類の条件を設定した．条件 A（退屈検出ベース）では，退屈状態への遷移をトリガーとして刺激イベントを発生させる．条件 B（固定頻度）では，退屈推定とは無関係に 1 分に 1 件の頻度でイベントを発生させるベースライン条件とした．参加者 5 名に対し，各参加者が ABAB または BABA の順序で計 4 試行（各 10 分間）を行い，全 20 試行のデータを収集した．各試行後には 6 項目の主観評価アンケートを実施した．

3. 結果と考察

退屈推定について，全 20 試行の退屈スコア平均値は 0.471 ± 0.098，退屈状態が ON であった時間の割合（boredom_on_ratio）は 0.456 ± 0.297 であった．退屈状態の検出頻度には参加者間で大きなばらつきがあり，参加者 04 は boredom_on_ratio が 0.630 – 0.822 と一貫して高い値を示した一方，参加者 03 では退屈状態がほとんど検出されない試行も観測された．このような参加者間の差異は，瞳孔径の個人差やゲームに対する取り組み方の違いを反映している可能性がある．刺激イベントの発生状況について，条件 B では設計通り全試行でイベント件数が 10 件であった一方，条件 A ではイベント件数が 1 件から 31 件まで幅広く分布し，退屈検出頻度に応じた適応的な制御が実現されていた一方で，想定を超える幅となっており，個人ごとの閾値調整が今後必要であることが示唆された．主観評価では，条件 A は条件 B に比べて「退屈を感じた」が低く（A: 2.10 ± 1.10, B: 3.20 ± 1.62），「集中できていた」が高かった（A: 4.60 ± 0.52, B: 3.80 ± 1.03）．また，「ストレスを感じた」も条件 A の方が低く（A: 1.50 ± 0.53, B: 2.60 ± 1.58），退屈検出に基づくイベント制御が退屈感の抑制と集中の維持に寄与した可能性が示唆された．一方，「敵増加タイミング適切」は条件 A（2.70 ± 0.48）が条件 B（1.70 ± 0.48）よりも「適切」側に近く，退屈検出に基づくタイミング制御が参加者の主観的な期待により合致していた．「難易度適切」は両条件とも「やや簡単すぎる」寄りであり，イベント内容の段階化による改善の余地がある．ただし，主観評価には参加者間で大きな個人差が認められた．参加者 04 は「退屈を感じた」が 4.00 ± 0.82 と高い値を示したのに対し，参加者 03 は全試行で 1 と回答した．しかし参加者 03 は条件 A でイベントが 1 回しか発火しなかった試行もあり，提案手法によって退屈を解消することに失敗していた．これは，退屈推定の精度に個人差が大きく影響しており，閾値設定やベースラインの個人適合が十分でない可能性を示唆している．

4. まとめ

本研究では，非装着型アイトラッカーを用いて瞳孔径および視線データからプレイヤの退屈状態をリアルタイムに推定し，推定結果に基づいて Minecraft 内の刺激イベントを動的に制御するシステムを提案した．5 名の参加者による実験の結果，退屈検出に基づくイベント制御（条件 A）は固定頻度のイベント制御（条件 B）と比較して，主観的な退屈感の低減および集中感の維持に寄与する傾向が確認された．一方で，退屈推定の精度には参加者間の個人差が大きく影響しており，今後は参加者ごとに推定感度を調整するキャリブレーション手続きの導入や，イベント内容の多様化・段階化が課題である．

参考文献（最大 5 件）

[1] Mihaly Csikszentmihalyi. Flow: The psychology of optimal experience. Harper & Row, 1990.

[2] Robin Hunicke and Vernell Chapman. AI for dynamic difficulty adjustment in games. In AAAI-04 Workshop on Challenges in Game Artificial Intelligence, pp. 91 – 96, 2004.

[3] Christoph Strauch, Michael Barthelmaes, Elisa Altgassen, and Anke Huckauf. Pupil dilation fulfills the requirements for dynamic difficulty adjustment in gaming on the example of Pong. In ACM Symposium on Eye Tracking Research and Applications, ETRA ’20, pp. 1 – 9, June 2020.

[4] Tobii. Tobii Pro Spark. https://connect.tobii.com/s/products/eye-trackers/tobii-pro-spark?language=ja. Accessed: January 29, 2026.

[5] Daniel Lee, Gopi Krishnan Rajbahadur, Dayi Lin, Mohammed Sayagh, Cor-Paul Bezemer, and Ahmed E. Hassan. An empirical study of the characteristics of popular Minecraft mods. Empirical Software Engineering, Vol. 25, No. 5, pp. 3396 – 3429, August 2020.

目 次

図 目 次

表 目 次

第1章　はじめに

近年，深層学習をはじめとする人工知能 (AI: Artificial Intelligence) 技術の発展は目覚ましく，画像認識，自然言語処理，ロボティクスなど多岐にわたる分野で実用化が進んでいる．ゲーム分野においても，AI の活用は重要な研究テーマの一つであり，プレイヤに新たな体験を提供する手段として注目されている．ゲームは，定義が明確であること，実装や追試が比較的容易であること，実問題に比べて単純なものであっても実問題と同様の思考の難しさを持つことなどから，古くから AI のテストベッドとして使われてきた．人間中心 AI の研究が重要になってきていることは，楽しませることが主目的であるゲームという対象領域をより価値あるものにしている．

ゲーム AI と聞くと，敵キャラクタの行動制御や対戦相手の戦略決定を思い浮かべることが多い．実際，囲碁や将棋，チェスといったボードゲームにおいては，AI が人間の トッププレイヤを超える性能を示すまでに至っている [1]．しかしまず，人間の相手をする相手プレイヤという役割に限定しても，それはただ強ければ良いものではなく，楽しませるための手加減や，人間らしさが要求される [2]．あるいは，良い仲間プレイヤ AI を作るためには，プレイヤの好みを読み取って “ 活躍させる ” ような仕組みも必要だろう [3].

さらに，ゲーム AI の役割は対戦相手の実現だけにとどまらない．近年では，ゲーム環境そのものの制御，例えばステージ生成 [4]，パズル生成 [5]，難易度調整 [6] といった要素にも AI 技術が導入されつつある．これらは，プレイヤの体験の質を向上させ，長時間のプレイにおいても興味を持続させるために重要な役割を果たす.

特に，プレイヤの状態や行動に応じてゲームの難易度を動的に調整する仕組みは，Dynamic Difficulty Adjustment(DDA) と呼ばれ，古くから研究されてきた．DDA は，プレイヤがゲームに対して感じる困難度を適切に保つことで，退屈や挫折を防ぎ，没入感を維持することを目的とする．従来の DDA 手法では，プレイヤの成績 (クリア時間，死亡回数，スコアなど) をもとに難易度パラメータを調整する方法 [7] が一般的であった．これらの手法は一定の効果を上げているものの，プレイヤの内面的な状態，すなわち「飽きている」「集中力が低下している」といった心理状態を直接捉えて調整を行っているわけではないため，本来行うべき調整とは反する調整を行ってしまっている恐れがある．

プレイヤの心理状態を推定するために，生体情報を用いる試みも存在する．例えば，脳波 (EEG)，心拍数 (HR)，皮膚電位 (GSR) などの生理指標は，プレイヤの覚醒度や感情状態を反映することが知られており [8]，これらを用いたゲーム体験の評価や適応的制御に関する研究が報告されている [9]．しかし，これらの生体情報を取得するためには，多くの場合，頭部や身体に装着型のセンサを必要とするという制約がある．頭部に電極を取り付けたり，手首や指にセンサを装着したりすることは，精度が高い一方でプレイヤにとって煩わしく，自然なプレイ体験を阻害する可能性がある．また，装着の手間やセンサの調整が必要となるため，日常的なゲームプレイ環境への導入は現実的ではない．

このような背景から，本研究では非接触で取得可能な生体情報として視線情報に着目する．視線情報には，プレイヤが注視している画面上の位置だけでなく，瞳孔径の変化も含まれる．瞳孔径は，認知負荷や注意状態，覚醒水準といった内的状態と関連することが報告されており [10]，プレイヤの飽きや注意低下の検出に有用な指標となる可能性がある．視線計測装置 (アイトラッカー) にはゴーグル状のような装着型のものと，ディスプレイ下部に接地して使用する非装着型の物が存在するが，本研究では，Tobii 社製の非装着型アイトラッカー [11] を採用する．アイトラッカーを用いて視線情報を取得し，プレイヤの飽きや注意力低下を推定し，その推定結果に基づいてゲーム内イベントの発生タイミングを動的に制御するシステムを構築する．

対象とするゲームには，オープンワールド型サンドボックスゲームである Minecraft を採用した．Minecraft は世界中で広くプレイされており，プレイヤが自由に探索，建築，採掘などの幅広い活動を行うことができることが特徴である．Minecraft は MOD やプラグインといった，外部プログラムによる挙動の変更が容易に行うことができる [12] ため拡張性が高く，研究用のシステムを組み込みやすいという利点がある．

本論文では，視線計測を用いた適応的なイベント制御システムを提案し，評価結果を報告する．第 2 章では，フロー理論と DDA，飽きや感情状態の検知に関する研究，瞳孔径や視線から得られる情報やその応用について，既存研究を整理する．第 3 章では，対象ゲームである Minecraft の特徴と，本研究での利用方法について述べる．第 4 章では，提案システムの設計について具体的に説明する．第 5 章では，提案システムを用いた実験の結果を述べる．第 6 章では，本研究の成果をまとめ，今後の課題について議論する．

第2章　関連研究

本研究は，Minecraft を対象ゲームとし，瞳孔径や視線をモニタリングすることで飽きを検知し，適切なイベントを行うことでその飽きを抑制しようとするものである．Minecraft については 3 章で概説することとし，本章では，ゲームにおける飽きを扱う研究でしばしば用いられるフロー理論，飽きを防ぐための動的難易度調整 (DDA)，飽きや感情を検知するための「ゲーム内」および「ゲーム外」の情報を用いた各種アプローチ，ゲーム外情報の中でも非接触で得られる瞳孔径や視線に関する研究を簡単に紹介する．

2.1　フロー理論と動的難易度調整 DDA

多くの場合，ゲームとは楽しむために行うものである．楽しみをもたらす要因は美しい音楽や映像，金銭的報酬，長期的なゲーム内アイテムの収集や上達の実感など多岐にわたる．最も重要な要因の一つは，与えられたタスク（挑戦）を達成する快感や，達成するために思考したり試行錯誤する過程そのものであると考える．この際に，与えられるタスクや環境（ステージや対戦相手）は，プレイヤーの技量に対して適切な難易度であることが好ましい場合が多い．簡単すぎるタスクには思考や試行錯誤が必要ないためにやりがいがなく，難しすぎるタスクは達成そのものが難しいため達成の快感を得られにくいためである．フロー理論は，ゲームに限らず，技量と難易度が心理状態に与える影響を論じたものである．動的難易度調整（DDA）は，プレイヤーの技量，心理状態，好みなどを踏まえてゲーム中の難易度を動的に調整しようとするものである．以下それぞれを少し詳しく説明する．

心理学用語としての「フロー」は，人間がある活動に集中し，のめりこみ，かつそのことが苦になっていないような精神的な状態のことを表すもので，心理学者の Mihaly Csikszentmihalyi が段階的に提唱した概念である [13]．この概念が提唱される以前にも例えば日本語では「無我の境地」などの似た概念は存在しており，スポーツ，旅行，アトラクションなどさまざまな分野でそれぞれの言葉で表現されていた精神的状態がフローに含まれる，あるいは関連しているとされる．フ

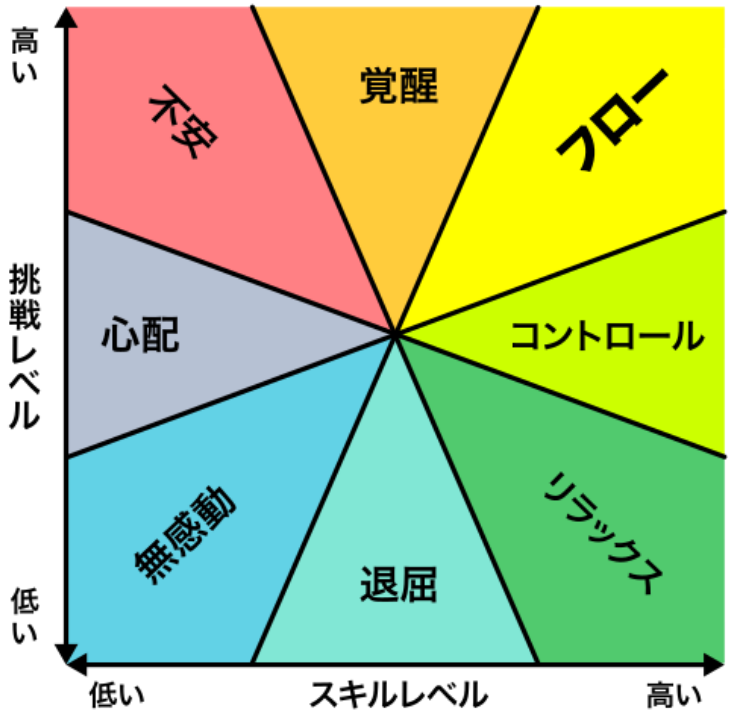

図 2.1: フローモデルによるメンタルステート図．wikipedia フロー（心理学）より抜粋.

ローに達するとどうなるのか（目の前の活動に完全に注意が向き，時間間隔が変容し，自己意識が薄れ，活動自体を楽しめるようになるなど），フローに達するためにはどんな条件が必要なのかなどは重要で，さまざまな応用が行われている．

フローに達するための条件としては，目標が明確であり，行動の結果（成功・失敗）がすぐに分かり，スキルレベルと挑戦レベルのバランスが取れていることが重要であるとされる [13]．図 2.1 は，横軸をスキルレベル，縦軸を挑戦レベルとし，どのような精神的状態になりやすいかを示した例である[1]．ゲームにおいては，多くの場合，目標は明確で，行動の結果はすぐにわかるため，フローに達するための条件はスキルレベルと挑戦レベルが釣り合っていることである．簡単すぎる課題は退屈であり，難しすぎる課題は不安をもたらす．そこで，プレイヤのスキルレベルに合わせた動的な難易度調整が効果的になることが多いわけである．

プレイヤのスキルレベルがさまざまで，それによって適切な難易度が変わることはコンピュータゲーム黎明期から意識されていた．そのため，いわゆる easy/normal/hard など，難易度をプレイヤが選択できるタイプのゲームは古くからある．一方で，プレイヤのスキルレベルというのはゲーム中に向上するものであり，また多面的であって「ある動作はまだ下手」「ある状況では適切な判断ができる」といったことはよくある．そこで，プレイヤの行動の結果（成功率，死亡回数，クリア時間やゲーム内スコアなど）をもとに，現状・現環境が簡単すぎるのか適正なのか難しすぎるのかを判断し，調整するという動的難易度調整の概念が導入された [7]．この論文で提案された Hamlet システムでは，ゲーム内イベント（強い敵の登場，アイテムドロップ，敵のミス）の確率を調整することで，プレイヤに気づかれにくいように難易度を連続的に調整するという思想が導入されている．

Hamlet の登場から 20 年以上が経過し，DDA は，フロー理論との統合，教師あり学習・強化学習・ベイズ推論との組み合わせなど，さまざまな方向性に発展を遂げてきた．これらの研究はいくつかの着眼点で分類することができるが，ひとつは，Hamlet のようにゲーム内情報だけを用いているか，それともゲーム外の情報（プレイヤの発言，表情，心拍数，皮膚電位など）を用いているかである．実装コストやプレイヤの負担・不安を考えれば前者のほうが優れている一方で，ゲーム内情報の中でも特に結果に関するものだけを見ていると，プレイヤの好みに関する個人差に対応できないという限界が存在する．そこで，次節で述べるような，飽きや感情をゲーム内情報やゲーム外情報から推測する手法も研究されるようになっている．

[1] この図にはさまざまなバリエーションがあり，スキルレベルや挑戦レベル両方が低くてもフローに達するとしている図もしばしば用いられる

2.2 飽きや感情の検知

人間の感情を推定しようとした研究では Picard のもの [14] が古典的に有名である．感情は人間の意思決定や行動に不可欠で，計算機システムもそれを認識したり，表現・利用すべきであると工学的に初めて打ち出した．Affective computing は枠組みとしてはかなり広く，認識も生成もしたいし，そのために生理信号・行動・表情や姿勢などどれを用いても良いとしている．

飽きや集中が重要な応用領域としては，ゲームのほかに教育分野が挙げられる．例えば，D'Mello らは，飽きが学習成果に強く影響を与えることを示した [15]．学習中の飽きを検出する方法として，問題回答ログ，操作ログなど，外部センサを用いずに飽きを推定する枠組みも提案されている [16]．外部センサを用いる方法としては，古くは姿勢や動作を用いたもの [17] から，表情認識を用いたもの [18] までさまざまにある．

ゲームにおいても，さまざまな取り組みがある．例えば Yannakakis らは，身体的にプレイするゲームにおいて，プレイヤがどれだけ楽しんでいるかを，BVP（血流量パルス），SC（皮膚電動）などの生理信号から推測する仕組みを提案している．この論文で用いた環境では，センサの装着は（被験者であった子どもたちにとって）それほど不愉快ではなかったようだが，一般にはセンサ装着はゲーム体験を阻害する要因になりうる．一方で，生体情報を用いず，ゲーム内情報だけで推測をしようという試みもある [19]．ここでは，ゲーム画面そのものと，操作入力だけをたよりに，engagement（没入度合い）を推測しようとしている．一般には，これらの試みは一定程度成功していて用途によっては十分なものの，生理信号を用いるものに比べれば精度が低いという傾向がある．

2.3 瞳孔径や視線から得られる情報とその活用

生理信号を用いた感情推定は，精度が高くできる場合が多い一方で，特に接触式の場合，使用者に負担をかけるというデメリットがある．表情やポーズをカメラで撮って推定に用いる手法は，各個人の癖や意識的な亢進と抑制によって精度が高くなりにくい．そこで，非接触かつ自律的制御が行いにくいものとして，目の挙動を用いて感情推定を行う研究が進んでいる．

Barker らは，VR 環境下で瞳孔径の変化を用いて幸福・悲哀・怒り・恐怖の 4 感情を分類する試みを行った [20]．瞳孔径は覚醒度や認知負荷，感情的刺激によって変化することが知られており，その時系列を特徴量化して教師あり学習を行うことにより，98 %の精度が得られたと報告している．

このようなアプローチをゲームに適用した取り組みもある．例えばChristophらは，単純なゲームPongを対象に，アイトラッカーを用いて瞳孔径を連続計測し，ゲームの難易度と瞳孔径がきれいに対応しており，ちょうどよい難易度で最も瞳孔径が大きくなることを確認した[10]．すなわち，ゲーム側で「現在のゲームの難易度はどのくらいか？」をモデル化せずとも，プレイヤの瞳孔径変化を観測していれば難しすぎたから簡単にしたり，簡単すぎたから難しくするといったDDAを行うことができるということである．

使うことができる情報は瞳孔径に限らない．例えばAntunesらは，視線情報を用いてゲームを適応的に制御する枠組みを提案した[21]．FPSゲームを対象に，プレイヤの視線行動をもとに，敵の出現位置や頻度，攻撃タイミング，アイテム配置などを自動で生成・調整することで，プレイヤの理解度や負荷状態に応じた適応が可能であることを示している．

第3章　Minecraft

Minecraft は，ブロックで構成された仮想世界において，探索，採掘，建築，戦闘などを行うサンドボックス型のゲームである．プレイヤは広大なワールドを自由に移動し，地形や資源を利用しながら自らの目標を設定してプレイすることができる．Minecraft は発売以降，幅広い年齢層に支持されており，世界的にも極めて高い人気と普及率を持つゲームである．

ゲーム内の世界は立方体状の ブロックによって構成されており，地形，建造物，資源などはすべてブロックの集合として表現される．プレイヤはこれらのブロックを破壊・配置することで環境に介入できる．また，動物や敵対的なモンスターなどの キャラクタ（エンティティ） が存在し，プレイヤはそれらと戦闘したり，回避したり，利用したりするなど，多様なインタラクションを行う．

Minecraft では，採掘した資源を組み合わせて アイテムをクラフトすることができる．初期段階では簡単な道具しか作れないが，鉄やダイヤモンドといった資源を入手することで，より高度な装備や装置を作成できるようになる．このような段階的な発展により，プレイヤはあたかも文明が発展していくかのような体験を得ることができる．

Minecraft の大きな特徴の一つは，自由度の高さと拡張性である．公式に MOD（Modification）が広く許容・利用されており，ゲームのルール，環境，エンティティ，イベントなどを柔軟に変更できる．そのため，Minecraft は AI 研究のプラットフォームとしても多く利用されてきた [12]．例えば，大規模言語モデル (LLM) を用いて自律的に探索・クラフトを行わせる Voyager[22] や，村の自動生成の品質を競うコンペティション [23] などが有名である．

さらに，MOD を用いることで，環境条件やイベントを細かく制御できるため，被験者実験のプラットフォームとしても適している．特定のタスク（例：鉱石を採掘しながら敵を倒す）を被験者に与え，ゲーム内行動や進捗を記録したり，瞳孔径を監視する別プログラムと通信することも可能である．プレイヤが退屈していると推定された場合に，雷を発生させる，敵を追加する，環境を変化させるといった介入も実装可能であり，動的な難易度調整や体験制御の実験を行うことができる．

このように Minecraft は，複雑すぎない基本ルールと高い拡張性を併せ持ち，プ

レイヤ体験を制御・観測する実験環境として有用である.

第4章　システムの設計

本章では，退屈推定に基づくゲーム内イベント制御システムの設計について述べる．本システムは，視線計測および退屈推定を担う Python による計測・推定器と，ゲーム内制御およびユーザインタフェースを担う Minecraft Mod の 2 つの機能から構成される．以下では，これら 2 機能の相互通信の仕組み，状態遷移，および実験シーケンスについて説明する．

4.1　システム全体構成

4.1.1　構成要素

本システムは，機能的に独立した 2 つのサブシステムから構成される．

Python クライアントは，Tobii SDK を用いた視線データの収集，収集データに基づく退屈推定，ベースライン計測，実験フロー全体の制御，および Minecraft Mod とのソケット通信を担当する．退屈推定には，特徴量計算，しきい値判定，ヒステリシス制御が含まれる．実験フロー制御では，二種類のタスク順序の管理，練習・タスクの開始/終了管理を行う．視線データ，特徴量，通信ログ，メタ情報といった各種ログの保存も Python 側の機能である．

Minecraft Mod は，Fabric Mod として実装される．Python からの制御コマンドの受信と実行，プレイヤの配置・リセット・テレポート，ゲーム内イベントのトリガ，UI/HUD による状態表示，および参加者のボタン操作に応じた応答送信を担当する．ゲーム内で発生したイベントやステータスのログ保存も Minecraft 側で行う．

このような機能の分離により，視線計測・推定のアルゴリズム変更は Python 側のみで完結し，ゲーム内演出や UI 変更は Minecraft 側のみで完結する．両機能は TCP ソケット通信によってのみ結合されるため，独立した開発が可能である．

4.1.2 データフロー

視線データは Tobii 視線計測装置から Python 側へ取得される．Python 側では，取得した視線データをリングバッファに蓄積し，一定間隔で特徴量を計算する．練習フェーズにおいてベースラインを計測・確定した後，タスクフェーズでは特徴量をベースラインで正規化し，退屈推定を行う．推定結果はソケット通信により Minecraft 側へ送信される．

Minecraft 側では，受信した推定結果を保持し，タスク種別に応じてイベント発火の判定を行う．発火したイベントはゲーム内で実行され，その記録はローカルログへ保存されるとともに，Python 側へも送信される．Python 側では受信したイベント情報をログファイルに記録する．

このように，データは「視線取得→特徴量計算→退屈推定→推定結果通知→イベント制御→ログ記録」という流れで処理される．図 4.1 にシステムの大まかな構成を示す．

4.2 Python 計測・推定器の設計

Python 計測・推定器は，視線計測，退屈推定，実験フロー制御，通信，およびログ記録を担う複数のモジュールから構成される．

4.2.1 モジュール構成

表 4.1 に主要モジュールの一覧を示す．main.py はアプリケーション本体であり，計測スレッドと推定スレッドを起動し，Minecraft との通信およびフロー制御を統括する．comm_mc.py は Minecraft Mod との TCP クライアントであり，送受信は JSONL 形式で改行区切りによりフレーミングを行う．acquisition.py は Tobii SDK を利用した視線データの取得を担当し，boredom.py はベースライン管理と退屈度推定を担当する．features.py はウィンドウベースの特徴量計算を行う．

4.2.2 実験フロー制御

実験フロー全体の制御は，main.py 内の FlowController が担当する．FlowController は状態機械として実装され，Minecraft 側からの応答メッセージに応じて状態遷移を行い，適切なコマンドを送信する．

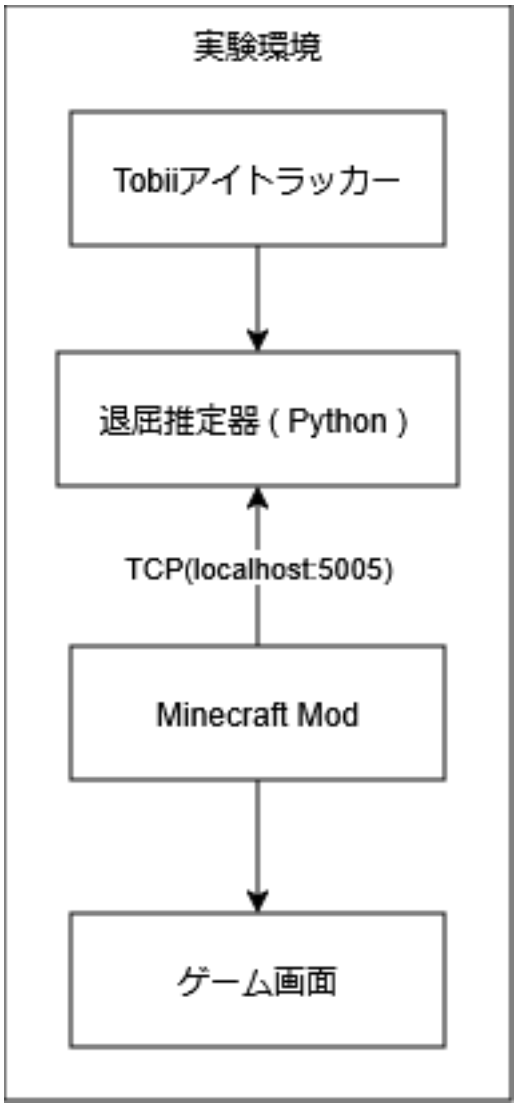

図 4.1: システム全体アーキテクチャ

表 4.1: Python 側主要モジュール一覧

モジュール	役割
main.py	アプリケーション本体，フロー制御
comm_mc.py	Minecraft Mod との TCP 通信
acquisition.py	Tobii SDK による視線データ取得
boredom.py	ベースライン管理と退屈度推定
features.py	ウィンドウベースの特徴量計算
events_logger.py	受信イベント・ステータスの保存
comm_logger.py	通信ログの保存
constants.py	実験パラメータの集中管理

初期状態では練習フェーズの準備完了を待つ．Minecraft 側から READY を受信すると START_PRACTICE を送信し，STARTED を受信すると練習時間の計測を開始する．練習時間終了後，END_PRACTICE を送信し，FINISHED を受信するとタスクフェーズの準備待ちへ遷移する．以降はタスクフェーズについて同様の遷移を繰り返す．すべてのタスクブロックが完了すると終了状態へ遷移する．

コマンド送信後に応答が得られない場合の再送制御は，タイムアウト時間と最大リトライ回数により制御される．

4.2.3 計測・推定処理

視線データは Tobii SDK を通じて取得され，リングバッファへ投入される．分析スレッドが定期的にリングバッファからデータを読み出し，ウィンドウベースの特徴量を計算する．

ベースラインは練習フェーズ中に計測される．練習フェーズにおいて収集された視線データから統計値を算出し，計測が成功した場合，その統計値を以後のタスクフェーズにおける正規化の基準として使用する．ベースラインの計測状態は Minecraft 側へ通知される．

タスクフェーズでは，特徴量をベースラインで正規化し，しきい値判定およびヒステリシス制御により退屈状態を推定する．推定結果はスコア更新および状態遷移として Minecraft 側へ送信される．

4.2.4 ログ設計

Python 側では，実験の再現性確保と事後分析のために，表 4.2 に示す複数種類のログを出力する．meta.json には実験の識別子である run_id，タスク順序，各

フェーズの所要時間等が記録される．この run_id は各ログファイルに共通して記録されるため，異なるログファイル間でのデータ統合が可能である．

表 4.2: Python 側出力ログ一覧

ファイル名	形式	内容
raw_gaze_pupil.csv	CSV	生の視線座標および瞳孔径データ
window_features.csv	CSV	特徴量および退屈推定結果
comm_log.jsonl	JSONL	Minecraft との送受信ログ
messages_rx.jsonl	JSONL	受信メッセージ
events.csv	CSV	受信イベント情報
status.csv	CSV	受信ステータス情報
meta.json	JSON	実験メタ情報

ログ形式として CSV を採用している理由は，表形式データの可読性と汎用的な分析ツールとの互換性である．JSONL を採用している理由は，構造化されたメッセージデータの柔軟な表現と，行単位での追記が容易であることによる．

4.3 Minecraft Mod 設計

Minecraft Mod は Fabric Mod として実装される．Python からの制御コマンドに応じたゲーム内制御，イベント発火，UI 表示，およびログ記録を担う．

4.3.1 クラス構成

表 4.3 に主要クラスの一覧を示す．Research.java は Mod の中心クラスであり，ソケット受信，コマンド処理，フェーズ管理，テレポート，UI 送信を実施する．PythonBridgeClient.java は Minecraft 側の TCP サーバであり，Python 側からの接続を待ち受ける．EventDirector.java はタスク中のイベント発火ロジックを担当する．ExperimentLogger.java はローカルファイルへのログ記録および Python へのイベント送信を担当する．

4.3.2 状態管理

Minecraft 側では，2 種類の状態管理機構が用いられている．

McState は Python との通信に基づく細粒度の状態を表現する．状態値として，`DISCONNECTED`（Python 未接続），LOBBY_WAIT_PRACTICE（練習開始待ちロ

表 4.3: Minecraft 側主要クラス一覧

クラス	役割
Research.java	Mod 中心クラス，コマンド処理
PythonBridgeClient.java	TCP サーバ，JSON 送受信
EventDirector.java	イベント発火ロジック
ExperimentLogger.java	ログ記録，イベント送信
BoredomStateStore.java	退屈推定結果の保持
PlayerResetService.java	プレイヤ装備・位置の初期化
ResearchClient.java	クライアント側処理
ClientState.java	HUD 表示用状態管理

ビー），WAIT_START_PRACTICE（練習開始指示待ち），PRACTICE_RUNNING（練習実行中），LOBBY_WAIT_TASK（タスク開始待ちロビー），WAIT_START_TASK（タスク開始指示待ち），TASK_RUNNING（タスク実行中），DONE（実験完了）がある．

ExperimentPhase は実験全体の進行段階を表現する．状態値として，IDLE（初期状態），CONNECTED_WAIT（接続待ち），PRACTICE_RUNNING（練習実行中），BASELINE_RUNNING（ベースライン計測中），TASK_RUNNING（タスク実行中），QUESTIONNAIRE_WAIT（アンケート待ち），FINISHED（終了）がある．

これら 2 種類の状態は，それぞれ異なる粒度での制御に用いられる．McState は Python とのコマンド送受信に基づく即時的な状態遷移を，ExperimentPhase は実験全体の進行フェーズを表現する．

4.3.3 イベント発火設計

タスク中のイベント発火は EventDirector が担当する．タスク実行中のみイベント発火の判定が行われる．

A タスクと B タスクでは，イベント発火の条件が異なる．A タスクでは，Python から受信した退屈推定において「退屈 ON」への状態遷移が発生した時点でイベントが発火する．B タスクでは，退屈推定とは無関係に，一定時間が経過した時点でイベントが発火する．発火するイベントの種類は防具及び武器を装備した「エリート敵の出現」または複数体の「敵の追加」である．

4.3.4 ログ設計

Minecraft 側では，ログが 2 つの経路で記録される．第一に，ゲーム内で発生したイベントおよびステータスがローカルファイルへ直接記録される．第二に，

Minecraft 側で発生したイベントはソケット通信により Python 側へも送信され，Python 側でも記録される．このような二重記録により，Minecraft 側と Python 側の双方でイベント情報を保持できる．また，両者のログには共通の run_id が含まれるため，事後的なデータ統合が可能である．

4.4 通信プロトコル設計

Python 計測・推定器と Minecraft Mod は，TCP ソケット通信により双方向に情報を交換する．

4.4.1 接続方式

通信には TCP ソケットを使用する．接続先は localhost（同一マシン上での通信を想定）であり，デフォルトのポート番号は 5005 である．Minecraft 側が TCP サーバとして接続を待ち受け，Python 側が TCP クライアントとして接続を行う．Python 側には再接続機能が実装されており，接続が切断された場合には自動的に再接続を試みる．

4.4.2 メッセージ形式

メッセージ形式として JSONL（JSON Lines）を採用している．各メッセージは 1 つの JSON オブジェクトとして表現され，改行文字によりフレーミングを行う．JSONL を採用した理由は以下のとおりである．JSON は構造化データの表現に適しており，メッセージ種別やパラメータを柔軟に記述できる．改行区切りによるフレーミングは実装が単純であり，ストリーム上でのメッセージ境界識別が容易である．人間可読な形式であるため，デバッグや通信ログの確認も容易である．

4.4.3 信頼性確保

通信の信頼性を確保するため，ACK/NACK による応答確認と再送制御を実装している．コマンド送信側は，送信後に相手側からの ACK または NACK を待つ．所定時間内に応答が得られない場合，同一コマンドを再送する．再送は最大リトライ回数まで行われる．この仕組みにより，一時的な通信遅延や処理遅延による応答の遅れを吸収し，コマンドの確実な伝達を実現する．

4.4.4 メッセージ種別

表 4.4 にメッセージ種別の一覧を示す．Minecraft 側から Python 側へは，参加者の準備完了を示す READY，フェーズ開始を示す STARTED，フェーズ終了を示す FINISHED が送信される．Python 側から Minecraft 側へは，練習やタスクの開始・終了を指示するコマンド，退屈推定結果の通知，ベースライン計測状態の通知が送信される．

表 4.4: メッセージ種別一覧

方向	種別	説明
MC → PY	READY	準備完了通知
MC → PY	STARTED	フェーズ開始通知
MC → PY	FINISHED	フェーズ終了通知
PY → MC	START_PRACTICE	練習開始指示
PY → MC	END_PRACTICE	練習終了指示
PY → MC	START_TASK	タスク開始指示
PY → MC	END_TASK	タスク終了指示
PY → MC	WAIT_READY	準備待ち移行指示
PY → MC	BOREDOM_UPDATE	退屈スコア更新通知
PY → MC	BOREDOM_TRANSITION	退屈状態遷移通知
PY → MC	BASELINE_STATUS	ベースライン状態通知

以下に代表的なメッセージの例を示す．

```
1 {"type": "MC_READY", "timestamp": 1234567890}
2 {"type": "PY_START_TASK", "task_type": "A", "block": 1}
3 {"type": "BOREDOM_TRANSITION", "state": "ON", "score": 0.75}
```

4.5 実験の流れ

本節では，システム全体の動作を理解するため，実験の流れを示す．図 4.2 に実験のシーケンス図を示す．

Python 側を起動する際，コマンドライン引数によりタスク順序（ABAB または BABA）を指定する．Python 側は Minecraft Mod への接続を試み，接続に成功すると実験が開始可能となる．

練習フェーズでは，参加者がゲーム内のボタンを押下すると，Minecraft 側から READY が Python 側へ送信される．Python 側は START_PRACTICE を送信し，

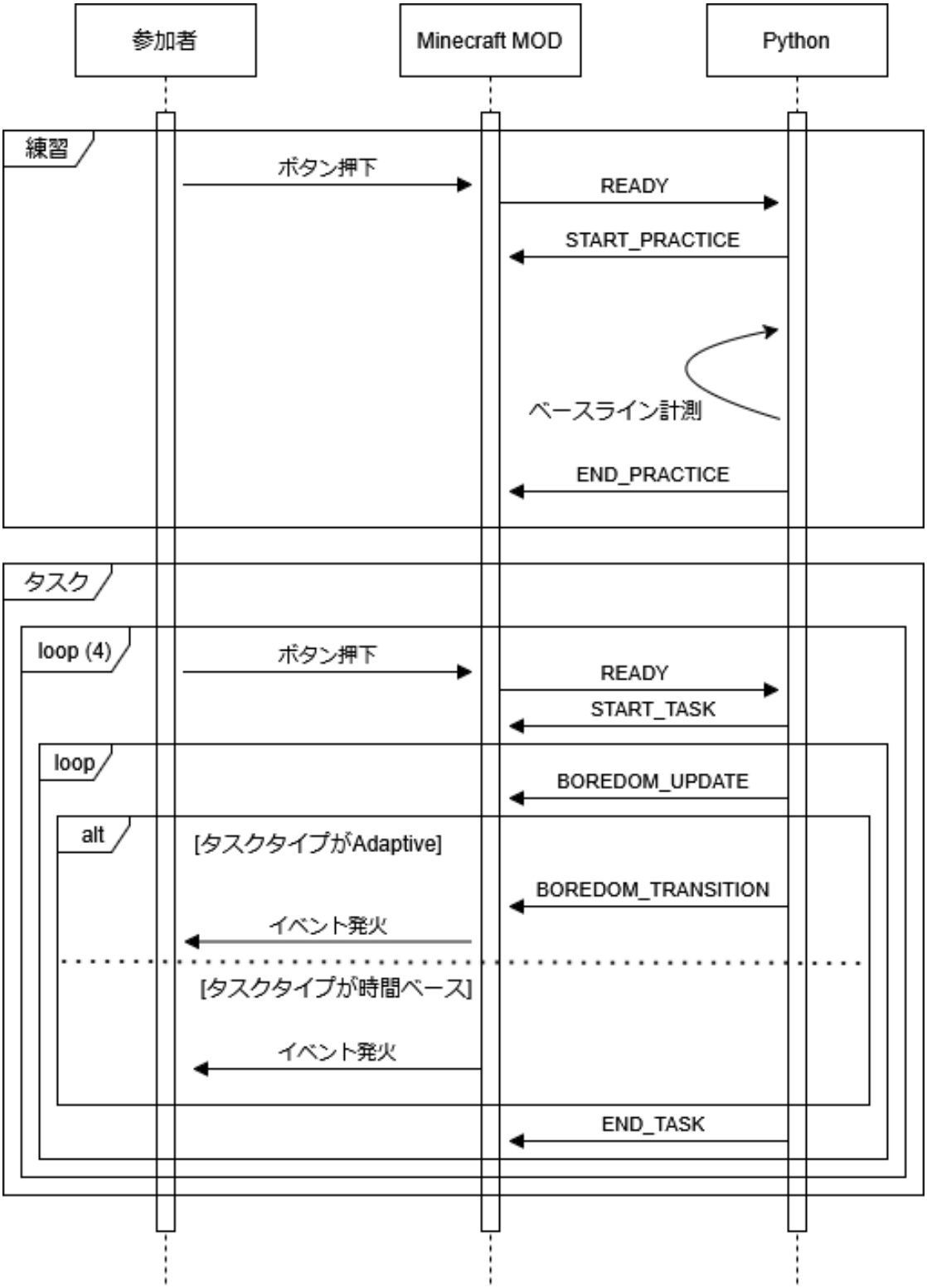

図 4.2: 実験シーケンス図

Minecraft 側は STARTED で応答する．練習フェーズ中，Python 側ではベースライン計測が行われる．所定時間が経過すると，Python 側は END_PRACTICE を送信し，Minecraft 側は FINISHED で応答する．

タスクフェーズは 4 ブロックで構成される．ABAB 順序の場合は A → B → A → B，BABA 順序の場合は B → A → B → A の順でタスクが実行される．各ブロックについて以下の手順を繰り返す．参加者がボタンを押下し READY が送信される．Python 側は START_TASK を送信しタスク種別を指定する．タスク実行中，Python 側は退屈推定を行い結果を Minecraft 側へ送信する．A タスクでは退屈 ON 遷移時に，B タスクでは一定時間経過時にイベントが発火する．所定時間が経過すると Python 側は END_TASK を送信する．Minecraft 側は FINISHED で応答し，参加者をアンケート室へテレポートする．

すべてのタスクブロックが完了すると，Minecraft 側の状態は DONE へ遷移し，実験が終了する．

4.6 本章のまとめ

本章では，退屈推定に基づくゲーム内イベント制御システムの設計について述べた．システムは，視線計測と退屈推定を担う Python 計測・推定器と，ゲーム内制御を担う Minecraft Mod の 2 つの機能から構成される．両プログラムは TCP ソケット通信により結合されて動作する．Python 側では FlowController による状態機械ベースの実験フロー制御，リングバッファとウィンドウベースの特徴量計算による退屈推定が実装されている．Minecraft 側では 2 層の状態管理，タスク種別に応じたイベント発火ロジックが実装されている．

第5章　実験

5.1　実験シナリオと刺激設計

本節では，本実験で使用した Minecraft 上の実験環境と，瞳孔径に基づく退屈推定を利用した刺激イベントの設計について述べる．

5.1.1　実験環境

実験環境として，Minecraft 内に閉鎖型の闘技場マップを構築した．闘技場は石ブロックのみで構成された $25 \times 25 \times 10$ の平坦な空間であり，プレイヤは闘技場の外に出ることができない．これにより，プレイヤの行動範囲を制限し，ゲーム内で発生するイベントを体験する条件を均一化した．また，闘技場内の床や壁などの素材を石ブロックで統一することで，画面全体の明るさや色調の変化を抑制し，視覚的要因による瞳孔径への影響を可能な限り低減した[1]．

実験に用いた闘技場の外観を図 5.1 に示す．プレイヤは 1 人称視点でこの闘技場内に配置され，出現する敵モブ（Mob）と戦闘を行う．本実験で出現する敵 Mob は，ゾンビを基本とし，後述する刺激イベントに応じてスケルトン，装備付きゾンビが出現する．通常のゾンビは，プレイヤが通常の近接攻撃で 4 回殴ることで討伐できる耐久度を持ち，ゾンビから受ける被ダメージはプレイヤの体力をおよそ 1/20 削る程度である．スケルトンは，ゾンビよりも移動速度が速い点が異なる．装備付きゾンビは通常のゾンビのおよそ 2 倍の耐久力を持つ．また，攻撃力は通常のゾンビのおよそ 6 倍である．

1 回の実験では，2 種類のタスクをそれぞれ 2 回ずつ，計 4 回実施する．各回の所要時間は 10 分間である．タスク実施中は，エリア内のゾンビが討伐された場合，新たにゾンビを出現させ，常に 3 体以上のゾンビが存在するように制御している．さらに，後述の条件に従って，敵の追加出現を発生させる．プレイヤは敵 Mob を倒すことでスコアを獲得する．

[1]瞳孔径は，心理的要因の他，環境・画面の明るさやその変化によっても変化する．本実験ではできるだけ心理的要因に限定するためにこのような設定としたが，実用上はこのようなコントロールは難しいため，なんらかの補正操作が必要となるだろう．

図 5.1: 実験で使用した Minecraft 内タスク用エリアのスクリーンショット

5.1.2 イベント種別と発生条件

本システムでは，タスク中に発生する刺激イベントとして以下の 2 種類を定義した．

`ADD_ENEMIES` ゾンビおよびスケルトンを合計 3 体，プレイヤ付近に追加出現させるイベント．

`SPAWN_ELITE` 装備付きのゾンビを 1 体，プレイヤ付近に出現させるイベント．

これらのイベントの発生条件は，タスクの種別によって異なる．各タスクには A または B のラベルが割り当てられ，被験者ごとに ABAB または BABA の順序で交互に配置される．以下に各タスクの仕様を述べる．

条件 A（退屈検出ベース） 瞳孔径データからリアルタイムに算出される退屈スコア（boredom_score，0–1 の連続値）を監視する．退屈スコアが閾値を超えて退屈状態（boredom_state = 1）へ遷移した場合，その遷移（`BOREDOM_TRANSITION`）をトリガーとして刺激イベント（`ADD_ENEMIES` または `SPAWN_ELITE`）を発生させる．したがって，条件 A におけるイベント発生頻度はプレイヤの技量や没入度，およびそれに伴う瞳孔径変動に依存し，試行ごとに異なる値をとる．

退屈状態の遷移には，退屈状態への突入（$0 \to 1$）と退屈状態からの復帰（$1 \to 0$）の双方が含まれるが，刺激イベントの発生は退屈状態への突入時にのみ行われる．そのため，1 試行内の遷移回数（n_transitions）とイベント発生件数（n_events_total）はおおむね 2 : 1 の関係となる．

条件 B（固定頻度）　条件 B では，退屈推定の結果にかかわらず，固定の頻度（1 件/分，1 試行あたり計 10 件）で刺激イベントを発生させる．条件 B は，退屈検出に基づくイベント制御（条件 A）と比較するためのベースライン条件として位置づけられる．なお，条件 B においても退屈推定自体は継続して動作しており，分析のために退屈状態の遷移回数は記録されるが，イベント発生のトリガーとしては使用しない．

5.2　実験設定と結果

本節では，実験の参加者および実験計画を示した上で，退屈推定の挙動，刺激イベントの発生状況，主観評価の順に結果を報告する．

5.2.1　参加者と実験計画

本実験の参加者は 5 名（参加者 01–05）であった．各参加者に対し実験を 1 回ずつ実施し，計 5 回（S011, S021, S031, S041, S051）のデータを収集した．各セッションは 2 種類の条件それぞれ 2 回ずつの，計 4 ゲーム（ABAB または BABA の順序）で構成され，試行総数は 20 である．表 5.1 にセッション一覧を示す．

表 5.1: セッション一覧

セッション ID	参加者 ID	順序計画
S011	01	BABA
S021	02	ABAB
S031	03	BABA
S041	04	ABAB
S051	05	BABA

表 5.2: 全 20 試行の記述統計

指標	平均 ± SD
mean_boredom	0.471 ± 0.098
boredom_on_ratio	0.456 ± 0.297
events_per_min（件/分）	1.425 ± 0.777
score	$11{,}773.8 \pm 3{,}088.7$

events_per_min は，総イベント数を実験時間（分）で正規化した値である．

表 5.3: 試行別の実験結果

参加者	条件	mean_boredom	boredom_on_ratio	n_transitions	n_events_total	score
01	B	0.368	0.168	40	10	8 450
01	A	0.343	0.112	30	15	8 750
01	B	0.410	0.295	38	10	10 400
01	A	0.389	0.155	34	17	5 750
02	A	0.510	0.628	43	22	11 600
02	B	0.477	0.485	30	10	13 500
02	A	0.564	0.765	36	18	14 800
02	B	0.555	0.743	32	10	11 950
03	B	0.404	0.157	22	10	14 550
03	A	0.326	0.005	2	1	13 750
03	B	0.295	0.000	0	10	8 850
03	A	0.376	0.078	12	6	15 150
04	A	0.530	0.630	45	23	13 400
04	B	0.599	0.822	36	10	9 250
04	A	0.579	0.741	56	28	6 825
04	B	0.619	0.821	33	10	12 950
05	B	0.545	0.667	53	10	14 750
05	A	0.520	0.644	62	31	13 525
05	B	0.501	0.558	63	10	10 075
05	A	0.516	0.644	48	24	17 200

条件 A は退屈検出に基づく刺激制御，条件 B は固定頻度（1 件/分）の刺激制御を表す．

5.2.2　退屈推定の挙動

全 20 試行における退屈指標の記述統計を表 5.2 に，試行別の詳細を表 5.3 に示す．

退屈スコア（boredom_score）は，複数の生理・行動特徴量について「ベースラインからの低下度」を算出し，それらを統合した 0–1 の連続値である．具体的には，ベースライン区間において各特徴量の中央値と分散（MAD×1.4826）を求め，各ウィンドウの観測値をベースラインに対する z 値として標準化した上で，$\mathrm{sigmoid}((-z)/K)$（K はスケール係数）により 0–1 へ正規化し，「低下度」指標とする．対象とした特徴量は `gaze_velocity_mean`，`gaze_dispersion`，`pupil_mean`，`pupil_std` の 4 つであり，それぞれ `low_gaze_velocity`，`low_gaze_dispersion`，`pupil_drop`，`low_pupil_variability` として算出される．退屈スコアは，これら 4 指標の平均

$$\mathrm{boredom\_score} = \frac{\mathrm{lgv} + \mathrm{lgd} + \mathrm{pd} + \mathrm{lpv}}{4} \tag{5.1}$$

で定義した．ここで，lgv は `low_gaze_velocity`，lgd は `low_gaze_dispersion`，pd は `pupil_drop`，lpv は `low_pupil_variability` を表す．

なお，ベースラインが未確立である場合，またはウィンドウの妥当性が不足する場合には，当該ウィンドウのスコアを 0.0 として扱う．

退屈状態（boredom_state）は，boredom_score が閾値を上回った場合に 1，下回った場合に 0 となる 2 値の状態である．

全試行での退屈スコア平均値（mean_boredom）は 0.471 ± 0.098（平均 ± 標準偏差）であった．退屈状態がオンであった時間の割合（boredom_on_ratio）は 0.456 ± 0.297 であり，標準偏差が平均に対して大きいことから，退屈状態の検出頻度にはプレイヤ間で大きなばらつきがあることがわかる．

参加者ごとの傾向を表 5.3 から確認する．参加者 04 は mean_boredom が 0.530–0.619 の範囲にあり，boredom_on_ratio も 0.630–0.822 と，全参加者の中で一貫して高い値を示した．一方，参加者 03 は mean_boredom が 0.295–0.404 と低く，特に 3 試行目（条件 B）では boredom_on_ratio = 0.000，2 試行目（条件 A）では boredom_on_ratio = 0.005 と，退屈状態がほとんど検出されない試行が観測された．このような参加者間の差異は，瞳孔径の個人差やゲームに対する取り組み方の違いを反映している可能性がある．

退屈状態の検出とそれに基づくイベント制御が顕著に観察された試行の例として，図 5.2（参加者 04，3 試行目，条件 A）を示す．この試行では events_per_min = 2.80 件/分，mean_boredom = 0.579，boredom_on_ratio = 0.741 であり，退屈状態の遷移が頻繁に生じ，それに応じてイベントが高頻度に発生していることが確認できる．一方，退屈状態がほとんど検出されず，イベント発生が抑制され

た試行の例として，図 5.3（参加者 03，2 試行目，条件 A）を示す．この試行では events_per_min = 0.10 件/分，boredom_on_ratio = 0.005 と，退屈状態がほぼ検出されず，発生したイベントは SPAWN_ELITE の 1 件のみであった．これら 2 つの例は，退屈推定の検出頻度がイベント発生頻度に直接的に影響する本システムの基本的な挙動を端的に示している．

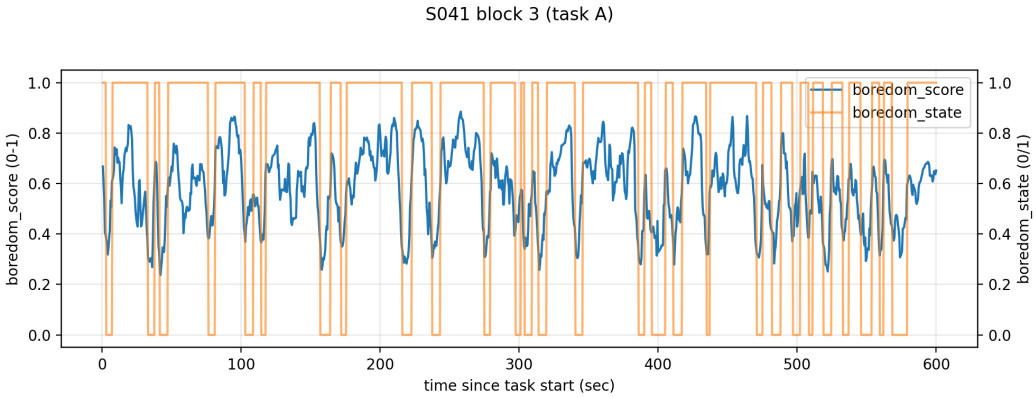

図 5.2: 退屈状態の検出頻度およびイベント発生頻度が高い試行の例

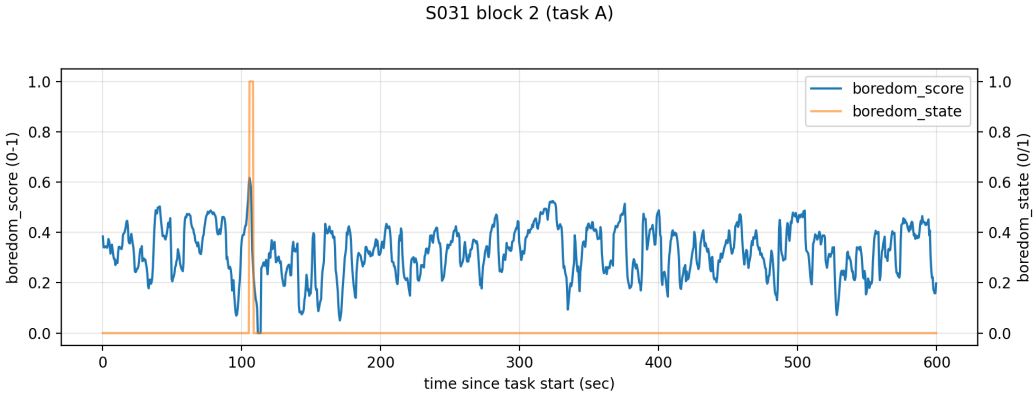

図 5.3: 退屈状態がほとんど検出されずイベント発生が抑制された試行の例

5.2.3　刺激イベントの発生状況

全 20 試行における刺激イベントの発生頻度（events_per_min）は 1.425±0.777 件/分であった（表 5.2）．

タスク条件ごとの傾向を表5.3から確認する．条件B（固定頻度条件）では全10試行においてn_events_total = 10件（events_per_min = 1.00件/分）であり，設計どおりの固定頻度が実現されていた．一方，条件A（退屈検出ベース条件）ではn_events_totalが1件（S031, 2試行目）から31件（S051, 2試行目）まで幅広く分布した．

条件Aにおける遷移回数（n_transitions）とイベント件数（n_events_total）の関係に着目すると，たとえばS041の3試行目ではn_transitions = 56，n_events_total = 28，S051の2試行目ではn_transitions = 62，n_events_total = 31であった．いずれもおおむね2 : 1の比率を示しており，退屈状態への突入と復帰の各遷移のうち突入時のみがイベントを発生させるという設計（5.1.2節）と整合する．より重要な点は，条件Aにおけるイベント件数が試行ごとに大きく変動することである．表5.3に示すように，条件Aのn_events_totalは1件から31件まで幅広く分布した．この変動は，退屈状態がオンであった割合（boredom_on_ratio）や退屈スコア（mean_boredom）の水準と一定の対応を示す．例えば，boredom_on_ratioが低い参加者（01および03）の条件Aではイベント件数が相対的に少なく，参加者01では平均16.0件（15件，17件），参加者03では平均3.5件（1件，6件）であった．一方，boredom_on_ratioが高い参加者（02，04，05）ではイベント件数も多い傾向がみられ，参加者05では24件および31件と高い値が観測された．

ただし，退屈指標とイベント件数の関係は単純な一次対応ではなく，参加者02のように退屈指標が高いにもかかわらずイベント件数が中程度にとどまる例もみられる．このことは，瞳孔径の個人差やプレイスタイルの違いに加え，閾値設定や正規化（ベースライン）の個人適合が十分でない可能性を示唆する．今後は，参加者ごとに推定の感度を調整するキャリブレーション手続きの導入が課題である．

5.2.4　ゲーム内成績

ゲーム内成績として，スコアを記録した．全20試行の平均スコアは11,773.750±3,088.737であった（表5.2）．

試行別の結果を表5.3に示す．スコアは5,750（参加者01, 4試行目）から17,200（参加者05, 4試行目）まで幅広く分布した．

5.2.5　主観評価

各試行終了後に，6項目の主観評価アンケートを実施した．回答は5段階尺度とし，数値化して集計した（$N = 20$試行）．表5.4に全体の結果，表5.5に参加者ごとの結果を示す．

表 5.4: 主観評価の結果（$N = 20$ 試行）

項目	平均 ± SD	尺度
退屈を感じた	2.65 ± 1.46	1: 全くそう思わない–5: とてもそう思う
集中できていた	4.20 ± 0.89	1: 全くそう思わない–5: とてもそう思う
忙しすぎた	2.05 ± 1.23	1: 全くそう思わない–5: とてもそう思う
ストレスを感じた	2.05 ± 1.28	1: 全くそう思わない–5: とてもそう思う
敵増加タイミング適切	2.20 ± 0.70	1: 遅すぎる–5: 早すぎる
難易度適切	2.30 ± 0.73	1: 簡単すぎる–5: 難しすぎる

表 5.5: 主観評価の結果（参加者別，$n = 4$ 試行/参加者）

参加者	退屈を感じた	集中できていた	忙しすぎた	ストレスを感じた	敵増加タイミング適切	難易度適切
1	3.75 ± 0.96	4.00 ± 1.41	2.75 ± 1.26	3.25 ± 1.50	1.75 ± 0.50	2.00 ± 0.82
2	1.75 ± 1.50	4.75 ± 0.50	2.25 ± 1.89	1.00 ± 0.00	2.25 ± 0.50	2.25 ± 0.96
3	1.00 ± 0.00	5.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	2.50 ± 0.58	2.75 ± 0.50
4	4.00 ± 0.82	3.50 ± 0.58	1.50 ± 0.58	2.50 ± 1.29	2.25 ± 0.96	2.25 ± 0.50
5	2.75 ± 0.96	3.75 ± 0.50	2.75 ± 0.96	2.50 ± 1.00	2.25 ± 0.96	2.25 ± 0.96

全体傾向

退屈に関する項目について,「退屈を感じた」は 2.65 ± 1.46 であり，中間値（3: どちらともいえない）よりやや低い方向にあった．一方,「集中できていた」は 4.20 ± 0.89 と高い値を示した．このことから，本実験の戦闘課題が全体として参加者の注意をある程度惹きつけており，主観的には集中が維持されやすい状況であったと考えられる．しかし,「退屈を感じた」の標準偏差が大きいことから，退屈感は参加者や試行によって大きく変動しており，平均値だけでは実験中の退屈感の実態を十分に捉えきれていない可能性がある．

負荷に関する項目について,「忙しすぎた」（2.05 ± 1.23）および「ストレスを感じた」（2.05 ± 1.28）はいずれも低い値であり，過度な負荷は参加者に感じられていなかったことが示唆される．このことから，刺激イベントによって負荷をより強く操作できる余地があると考えられる．

イベント制御の適切さに関する項目について,「敵増加タイミング適切」は 2.20 ± 0.70 であった．本項目は「遅すぎる」を 1,「早すぎる」を 5 とする尺度であり，得られた値は「やや遅すぎる」（2）に近い．この結果は，退屈検出に基づくイベント発生のタイミングが参加者の主観的な期待よりもやや遅れていた可能性を示す．「難易度適切」は 2.30 ± 0.73 と「やや簡単すぎる」（2）寄りであったため，敵 Mob の強さや数，イベント内容（出現する敵 Mob の種類など）を段階化することで,「ちょうどよい」（3）へ近づけられる可能性がある．

条件 A と条件 B の比較

条件 A（退屈検出ベース）と条件 B（固定頻度）の比較のため，主観評価を設定別に集計した（各設定 $n = 10$）．本実験は同一参加者が両条件を行うものであるため，以下では条件間の平均値の差を中心に傾向を述べる．

表 5.6 に示すように,「退屈を感じた」は条件 A が 2.10±1.10，条件 B が 3.20±1.62 であり，条件 A の方が低かった（差 −1.10）．一方,「集中できていた」は条件 A が 4.60 ± 0.52，条件 B が 3.80 ± 1.03 であり，条件 A の方が高かった（差 +0.80）．この結果は，退屈状態への突入をトリガーとして刺激イベントを提示する条件 A が，参加者の主観的な退屈感を抑えつつ，集中の維持に寄与した可能性を示唆している．

負荷に関しては,「忙しすぎた」は条件 A の方が高い一方（差 +0.70）,「ストレスを感じた」は条件 A の方が低かった（差 −1.10）．また,「敵増加タイミング適切」は条件 A が 2.70 ± 0.48，条件 B が 1.70 ± 0.48 であり，条件 A の方が大きかった（差 +1.00）．条件 A では刺激イベントが相対的に多く提示されることにより，先頭の負荷や難易度が上昇した可能性がある．ただし,「ストレスを感じた」は条件 A の方が低くなっているため，負荷増加が必ずしも不快感の増大には直結していない可能性が示された.「難易度適切」も条件 A の方が大きく（差 +0.80），条件 A では相対的に難しめ側に評価が寄った．本実験での条件 B は固定頻度（1 件/分，1 試行あたり 10 件）として設計したが，この頻度自体が参加者の期待に対して十分でなかったと考えられる．もし条件 B の提示頻度を高めた場合，退屈感が低下する可能性はあるが，その場合に負荷やストレスがどのように変化するかは本実験からは判断できない．

参加者間の個人差

表 5.5 に示すように，主観評価には参加者間で大きなばらつきが認められた.「退屈を感じた」について，参加者 3 は全試行で 1（まったくそう思わない）と回答した（1.00 ± 0.00）のに対し，参加者 4 は 4.00 ± 0.82 と高い値を示した．同様に「集中できていた」についても，参加者 3 が 5.00 ± 0.00 であったのに対し参加者 4 は 3.50 ± 0.58 であり，退屈の自覚と集中の自己評価が参加者間で対照的な傾向を示した．

負荷に関しても個人差がみられた．参加者 2 および参加者 3 は「ストレスを感じた」に対して全試行で 1 と回答した（1.00 ± 0.00）一方，参加者 1 は 3.25 ± 1.50 であり，同一のイベント制御に対する主観的負荷の受け止め方が参加者によって異なることが確認された．

このような個人差は，ゲーム経験，プレイ方針，戦闘の得意不得意など，参加者の行動特性や経験などの影響を受けている可能性がある．したがって，今後は参加者の経験水準やプレイ特性を考慮した分析や設計の検討が重要となる．

なお，参加者 03 ではアンケート結果の一部項目において，最小値付近の回答が一貫して選択され，標準偏差が 0 となった．このことから，主観評価の個人差や回答傾向の影響を含む可能性も考慮する必要がある．

表 5.6: 主観評価の設定別比較（各設定 $n = 10$）

項目	条件 A（平均 ±SD）	条件 B（平均 ±SD）	差（A−B）
退屈を感じた	2.10 ± 1.10	3.20 ± 1.62	-1.10
集中できていた	4.60 ± 0.52	3.80 ± 1.03	$+0.80$
忙しすぎた	2.40 ± 1.43	1.70 ± 0.95	$+0.70$
ストレスを感じた	1.50 ± 0.53	2.60 ± 1.58	-1.10
敵増加タイミング適切	2.70 ± 0.48	1.70 ± 0.48	$+1.00$
難易度適切	2.70 ± 0.48	1.90 ± 0.74	$+0.80$

参加者 03 の条件差（イベント提示が少ない参加者の例）

参加者 03 では，条件 A におけるイベント件数が少なく（表 5.3），刺激による介入が最小限にとどまった．この参加者の主観評価（参加者別の条件集計）を見ると，表 5.7 に示すように,「退屈を感じた」「集中できていた」「忙しすぎた」「ストレスを感じた」は条件 A・条件 B で同一の値を示し（それぞれ 1.00，5.00，1.00，1.00），主観的には両条件の差がほとんど表れなかった．

一方,「敵増加タイミング適切」は条件 A が 3.00，条件 B が 2.00 であり，条件 A の方が相対的に「適切（中立）」側に近かった．これは，退屈状態がほとんど検出されない参加者に対しては条件 A が刺激イベントの提示を抑制し，プレイを過度に妨げなかった可能性を示唆する．ただし本参加者では，多くの項目で最小値（1）または最大値（5）の回答が一貫しており，選択肢の上限・下限付近に回答が集中したため，条件差が評価値として表れにくかった可能性にも留意する必要がある．

試行間の推移

試行ごとの集計では,「退屈を感じた」の平均値が 1 試行目で 2.60，2 試行目で 2.20，3 試行目で 2.80，4 試行目で 3.00 と推移した．単調な増加ではないものの，

表 5.7: 参加者 03 の主観評価（条件 A と条件 B の比較，各条件 $n = 2$ 試行）

項目	条件 A（平均 ±SD）	条件 B（平均 ±SD）	差（A−B）
退屈を感じた	1.00 ± 0.00	1.00 ± 0.00	$+0.00$
集中できていた	5.00 ± 0.00	5.00 ± 0.00	$+0.00$
忙しすぎた	1.00 ± 0.00	1.00 ± 0.00	$+0.00$
ストレスを感じた	1.00 ± 0.00	1.00 ± 0.00	$+0.00$
敵増加タイミング適切	3.00 ± 0.00	2.00 ± 0.00	$+1.00$
難易度適切	3.00 ± 0.00	2.50 ± 0.71	$+0.50$

後半の試行でやや高い値を示しており，反復に伴う飽きの蓄積が生じていた可能性がある．

第6章　おわりに

本研究では，プレイヤの退屈を非接触で検出し，それに応じてゲーム内イベントを動的に制御するシステムを提案した．従来の動的難易度調整（DDA）では，クリア時間やスコアなどのゲーム内の成績をもとに難易度を調整するのが一般的であったが，本研究では非装着型のアイトラッカーから得られる瞳孔径や視線の情報を用い，プレイヤが「退屈しているかどうか」をリアルタイムに推定してイベント制御に活用することを試みた．対象ゲームには Minecraft を採用し，視線計測と退屈推定を行う Python プログラムと，ゲーム内の制御を行う Minecraft Mod を，TCP ソケット通信で連携させるシステムを設計・実装した．退屈推定では，視線速度平均，視線分散，瞳孔径平均，瞳孔径変動の 4 つの特徴量を用いた．練習フェーズで計測したベースラインに対する低下度をシグモイド関数で 0–1 に正規化し，その平均を退屈スコアとした．この退屈スコアに閾値判定とヒステリシス制御を適用して退屈状態を二値化し，退屈状態に突入したタイミングで敵の追加出現やエリート敵の出現といったイベントを発生させる仕組みとした．5 名の参加者による計 20 試行の実験を行い，退屈検出に基づいてイベントを発生させる条件 A と，固定の頻度でイベントを発生させる条件 B を比較した．

実験の結果，条件 A では，退屈指標が高い参加者にはイベントが多く発生し，退屈がほとんど検出されない参加者にはイベントが抑制されるという，プレイヤの状態に応じた適応的な制御が実現された．主観評価では，条件 A は条件 B に比べて「退屈を感じた」が低く（条件 A: 2.10，条件 B: 3.20）,「集中できていた」が高い（条件 A: 4.60，条件 B: 3.80）という傾向がみられた．加えて，条件 A では「ストレスを感じた」も条件 B より低く，退屈検出に基づくイベント制御が退屈感を抑えつつ，過度な不快感を与えていない可能性が示された．一方で，いくつかの課題も明らかになった．最も大きいのは，退屈推定における個人差への対応である．参加者 03 のように退屈状態がほとんど検出されずイベントが極端に少なくなる場合もあれば，参加者 04 のように一貫して高い退屈指標が観測される場合もあり，参加者間のばらつきが大きかった．これは，現在のベースライン計測や閾値の設定では個人差を十分に吸収しきれていないことを意味しており，参加者ごとのキャリブレーション手続きの導入が今後必要になると考える．また，推定された退屈状態が実際にプレイヤの感じている退屈とどの程度一致しているかについ

ても，プレイ中のリアルタイムな自己報告と照合するなど，より詳しい検証が求められる．さらに，本実験は 5 名・20 試行のデータに基づいており，条件間の比較で得られた傾向は興味深いものの，十分な人数での検証とは言えないため，より多くの参加者を対象とした実験を行い，この傾向が安定して再現されるかを確認する必要がある．

今後の展望としては，まず退屈推定に用いる特徴量の拡張が考えられる．本研究では 4 つの特徴量を等しい重みで統合したが，瞬き頻度やサッケード特性といった追加の視線指標を取り入れたり，機械学習によって特徴量の重みや統合方法を最適化したりすることで，推定精度の向上が期待できる．次に，イベントの多様化と制御の工夫が挙げられる．本実験では 2 種類の刺激イベントのみを用いたが，主観評価で難易度が「やや簡単すぎる」と評価されたことを踏まえると，イベントの種類を増やしたり，退屈の程度に応じてイベントの強さを段階的に変えたりする制御が有効だろう．また，本実験では闘技場を石ブロックで統一して画面の明るさの変動を抑えたが，実際のゲーム環境ではこのような制約は現実的でないため，画面輝度の変化に応じた瞳孔径の補正も重要な課題となる．本研究が，非接触な生体情報を用いた適応的なゲーム体験制御の発展に少しでも貢献できれば幸いである．

参考文献

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, Vol. 352, No. 6419, pp. 1140–1144, 2018.

[2] Chu-Hsuan Hsueh and Kokolo Ikeda. Improvement of move naturalness for playing good-quality games with middle-level players. *Applied Intelligence*, Vol. 54, No. 2, pp. 1637–1655, January 2024.

[3] 板東宏和, 池田心, Chu-Hsuan Hsueh. 人間プレイヤを活躍させる協力型ゲームの味方 AI. Technical report, 北陸先端科学技術大学院大学, Mar 2023. 第 49 回 GI 研究発表会.

[4] SangGyu Nam, Chu-Hsuan Hsueh, Pavinee Rerkjirattikal, and Kokolo Ikeda. Using reinforcement learning to generate levels of Super Mario Bros. with quality and diversity. *IEEE Transactions on Games*, Vol. 16, No. 4, pp. 807–820, December 2024.

[5] Barbara De Kegel and Mads Haahr. Procedural puzzle generation: A survey. *IEEE Transactions on Games*, Vol. 12, No. 1, pp. 21–40, March 2020.

[6] Mohammad Zohaib. Dynamic difficulty adjustment (DDA) in computer games: A review. *Advances in Human-Computer Interaction*, Vol. 2018, pp. 1–12, 2018.

[7] Robin Hunicke and Vernell Chapman. AI for dynamic difficulty adjustment in games. In *AAAI-04 Workshop on Challenges in Game Artificial Intelligence*, pp. 91–96, 2004.

[8] S. Koelstra, C. Muhl, M. Soleymani, Jong-Seok Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, and I. Patras. DEAP: A database for emotion analysis ;us-

ing physiological signals. *IEEE Transactions on Affective Computing*, Vol. 3, No. 1, pp. 18–31, January 2012.

[9] Adi Stein, Yair Yotam, Rami Puzis, Guy Shani, and Meirav Taieb-Maimon. EEG-triggered dynamic difficulty adjustment for multiplayer games. *Entertainment Computing*, Vol. 25, pp. 14–25, March 2018.

[10] Christoph Strauch, Michael Barthelmaes, Elisa Altgassen, and Anke Huckauf. Pupil dilation fulfills the requirements for dynamic difficulty adjustment in gaming on the example of Pong. In *ACM Symposium on Eye Tracking Research and Applications*, ETRA ' 20, pp. 1–9, June 2020.

[11] Tobii. Tobii Pro Spark. https://connect.tobii.com/s/products/eye-trackers/tobii-pro-spark?language=ja. Accessed: January 29, 2026.

[12] Daniel Lee, Gopi Krishnan Rajbahadur, Dayi Lin, Mohammed Sayagh, Cor-Paul Bezemer, and Ahmed E. Hassan. An empirical study of the characteristics of popular Minecraft mods. *Empirical Software Engineering*, Vol. 25, No. 5, pp. 3396–3429, August 2020.

[13] Mihaly Csikszentmihalyi. *Flow: The psychology of optimal experience*. Harper & Row, 1990.

[14] Rosalind W. Picard. *Affective Computing*. The MIT Press, September 1997.

[15] D ' Mello, Sidney, Art Graesser. Dynamics of affective states during complex learning. *Learning and Instruction*, Vol. 22, No. 2, pp. 145–157, April 2012.

[16] Ryan SJd Baker, Sujith M Gowda, Michael Wixon, Jessica Kalka, Angela Z Wagner, Aatish Salvi, Vincent Aleven, Gail W Kusbit, Jaclyn Ocumpaugh, and Lisa Rossi. Towards sensor-free affect detection in cognitive tutor algebra. In *Proceedings of the 5th International Conference on Educational Data Mining*, 2012.

[17] D ' Mello, Sidney, Art Graesser. Automatic detection of learner's affect from gross body language. *Applied Artificial Intelligence*, Vol. 23, No. 2, pp. 123–150, February 2009.

[18] Jacob Whitehill, Zewelanji Serpell, Yi-Ching Lin, Aysha Foster, and Javier R. Movellan. The faces of engagement: Automatic recognition of student engagementfrom facial expressions. *IEEE Transactions on Affective Computing*, Vol. 5, No. 1, pp. 86–98, January 2014.

[19] Kosmas Pinitas, David Renaudie, Mike Thomsen, Matthew Barthet, Konstantinos Makantasis, Antonios Liapis, and Georgios N. Yannakakis. Predicting player engagement in Tom Clancy ' s The Division 2: A multimodal approach via pixels and gamepad actions. In *International Conference on Multimodal Interaction*, ICMI ' 23, pp. 488–497, October 2023.

[20] Darlene Barker and Haim Levkowitz. Thelxinoë: Recognizing human emotions using pupillometry and machine learning. *Machine Learning and Applications: An International Journal*, Vol. 11, No. 1, pp. 01–14, March 2024.

[21] João Antunes and Pedro Santana. A study on the use of eye tracking to adapt gameplay and procedural content generation in first-person shooter games. *Multimodal Technologies and Interaction*, Vol. 2, No. 2, p. 23, May 2018.

[22] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

[23] Christoph Salge, Michael Cerny Green, Rodrigo Canaan, Filip Skwarski, Rafael Fritsch, Adrian Brightmoore, Shaofang Ye, Changxing Cao, and Julian Togelius. The AI settlement generation challenge in Minecraft: First year report. *KI - Künstliche Intelligenz*, Vol. 34, No. 1, pp. 19–31, January 2020.