# Unity の研究

## オンラインアクションゲームの作成

情報工学科 3 年 15 番

御福 宏佑

# 目次

## １．目的

ネットワークに接続され、クライアントサーバシステムを形成しているオンラインゲームは、日常生活で最も身近にあるシステムのひとつである。システムとして非常に理解しやすく、開発をスムーズに行いやすいと考えたためである。世界のゲームコンテンツ市場の中でも、デジタル配信ゲーム市場の市場規模は約７兆円となっており、オンラインゲームコンテンツに対する需要が高くなっている。また将来性があると考えられるためである。情報工学科で今まで学習したアルゴリズム、オブジェクト指向、関係データベースなどのプログラミングや大規模システム開発に関する、様々な情報技術をすべて利用することができ、集大成とするのにふさわしいためである。そして何よりも、私がゲームを趣味としているため、意欲的に研究できるテーマであると考えたためである。

3D のアクションゲームを選択した目的は、臨場感や迫力があり、開発者側もプレイヤー側も楽しいと思えるコンテンツを開発できると考えたためである。また近年はコンピュータの性能の向上に伴い、3DCG の描画が容易に行えるようになり、これを駆使したゲームが普及している背景がある。VR 技術の進化により、今後も需要が高まっていくと考えられるためである。

ゲームの開発環境として Unity を選択した目的は、主に開発の効率化や工数の削減である。ゲームを実行するためのプラットフォームには Windows、Android、iOS などさまざまなものがあり、その各々で実行プログラムの形式は異なっている。そのため効率化のために、一つのソースコードなどの資源から複数のプラットフォームに対応した実行プログラムを生成する、クロスプラットフォームの概念が重要になる。Unity には標準でクロスプラットフォームのビルド機能が搭載されており、この問題の解決に最も適していると考えたためである。また 3D ゲームの開発の際には物理演算や光源処理を代表とした、3D レンダリングの処理が必須となる。Unity はこのようなクラスやコンポーネントを包括おり、ゲーム開発の作業に集中することが可能になる。今回の研究を行う際にもこのことは効率化の面で非常に適していると考えたためである。

RDBMS として MySQL を選択する目的は、主にコスト面の問題を解決するためである。授業では SQL を学習する際の RDBMS として、Oracle を用いた。Oracle は商用 RDBMS として圧倒的なシェアを獲得しており、多くの大規模基幹システムで利用されている。しかし利用するためには高い料金が必要となる。このようなコストは小規模開発のうえでは大きな負担となる。それに比べオープンソースソフトウェアである MySQL は、料金が無料となるため、コスト面の問題を解決することができる。この背景もあり MySQL は、Web 開発業界ではかなり多く利用されている RDBMS である。さらに大規模開発に利用している事例も存在する。実際に Web 開発を行っている中で、非常に興味関心を持ち学習したいと考えた。

ゲームにリアルタイム双方向通信の機能を搭載する目的は、授業でこれを扱ったことがなく、強く学びたいと思ったためである。VR 技術で実現できること

は数多くあるが、バーチャルオフィスを例に挙げるとリアルタイム双方向通信の技術は必須となる。このように将来リアルタイム通信に対する需要は、ますます増えていくと考えられる。そして私自身がリアルタイム通信を駆使したゲームが好きであるうえ、他プレイヤーと対戦や協力などのコミュニケーションを図れる、ゲームはコンテンツとして盛り上がりやすいと考えたためである。

リアルタイム双方向通信を実現するためには、Node.js およびそのライブラリの Socket.IO を使用する必要がある。近年 HTML5 の策定とともに WebSocket が注目されている。またそれに伴って Socket.IO を利用したリアルタイム双方向通信が普及しており、非常に興味関心を持ったためである。Socket.IO では、リアルタイム通信を実現するために用いられている、ポーリング、Comet、WebSocket などの技術を抽象化して幅広い環境でリアルタイム双方向通信を行うことができるように設計されている。そのため、クロスプラットフォームや開発の効率化を目指す今回の課題研究には最も適したライブラリだと考えたためである。

以上により、私は Unity、MySQL、Socket.IO を用いて「リアルタイム 3D オンラインアクションゲーム」を開発することを課題研究のテーマとした。

## ２．理論

### ２．１　オブジェクト指向プログラミング

OOP(オブジェクト指向プログラミング、Object Oriented Programming)は、システム全体をオブジェクトの集合体として考え、オブジェクト同士のメッセージのやり取りに視点をおくプログラミング技法である。継承などの機能を用いることで構造化プログラミングなどのプログラミング技法に比べて、開発効率とメンテナンス性を大幅に向上させることができる。以下にクラスベースの OOP における主要な用語を列挙する。

（１）プロパティ

プロパティ(Property)は、オブジェクトが保持しているデータ、変数のことである。

（２）メソッド

メソッド(Method)は、オブジェクトが保持している動作、処理、関数のことである。

（３）オブジェクト

オブジェクト指向プログラミングにおいてシステムはオブジェクトの集合体として考えられる。オブジェクトはプロパティ(データ、変数)とメソッド(動作、処理、関数)の集合体である。

（４）クラス

クラス(Class)は、同じ性質を持つオブジェクトの抽象概念である。設計図として複数のオブジェクトが共通に保有しているプロパティやメソッドを定義する。

（５）インスタンス

インスタンス(Instance)は、クラスに基づいて作成されたオブジェクトのことを指す。また、このようにオブジェクトを作成することをインスタンス化(Instantiation)という。

（６）カプセル化

カプセル化(Encapsulation)は、オブジェクトが保有しているプロパティやメソッドといった要素の内部仕様や内部構造を外部から隠蔽することである。OOP においては、非常に重要な概念である。公開されたメソッドを用いてのみ、内部のプロパティを操作できるようにすることで、クラス外からの干渉を受けないようになり内部のプロパティの完全性を保護することができる。また、システム全体のメンテナンス性を向上させることができる。

（７）コンストラクタ

コンストラクタ(Constructor)は、インスタンス化を行う際に呼び出され、オブジェクトの初期化に用いられるメソッドである。

（８）継承

継承(Inheritance)は、既存のクラスをもとにそれの特性や構造を継承した新たなクラスを定義することをいう。既存のコードを再利用できるため、大きくプログラミングの生産性を向上させることができる。継承元のクラスをスーパークラスなどと呼び、継承先のクラスは、サブクラスなどと呼ばれる。

（９）オーバーライド

オーバーライド(Override)は、スーパークラスで定義されているメソッドをサブクラスで再定義(上書き)し、動作を上書きすることである。

（１０）ポリモーフィズム

ポリモーフィズム(多態性、Polymorphism)は、インスタンスによって同一のメソッド名で異なる処理が行われる性質のことをいう。スーパークラスのメソッドをサブクラスでオーバーライドすることで、外部からメソッドを参照する際の動作を振り分けることが可能になる。

（１１）is-a 関係、has-a 関係

オブジェクト指向設計における物事の関係性である。is-a 関係は継承の関係性である。has-a 関係は、含有の関係性である。

## ２．２　C#

C#はマイクロソフト社が開発したプログラミング言語である。同じくマイクロソフト社が開発したアプリケーション開発・実行環境である Microsoft .NET Framework とともに開発された。クラスベースのオブジェクト指向を主とした様々なプログラミングパラダイムに対応している。Unity においてもスクリプト言語のひとつとして採用されている。

（１）型

C#における主要なデータ型を以下に列挙する。

１）int

符号付き整数を格納する。サイズは 4 バイトで構成される。

２）float

浮動小数点数を格納する。サイズは 4 バイトで構成される。

３）string

文字列を格納する。

４）bool

真(true)と偽(false)からなる論理値を格納する。サイズは 1 バイトで構成される。

５）enum

名前付き定数の集合で構成される列挙型である。

（２）修飾子

修飾子は宣言の際に用いられる。

１）abstract

インスタンス化を禁止する。継承してのみ利用される仮想クラスを作成する。

２）virtual

サブクラスにおいてオーバーライドすることを許可する。

３）override

スーパークラスのメソッドをオーバーライドする。

４）sealed

クラスの継承を禁止する。

５）アクセス修飾子

OOP におけるカプセル化を実現するために使用する修飾子である。プロパティやメソッドに対してアクセス権を設定することができる。

① public

どこからでもアクセス可能になる。

② private

そのクラス内でのみアクセス可能になる。

③ protected

そのクラス、もしくはサブクラスでのみアクセス可能になる。

## ２．３ JavaScript

JavaScript は動的ウェブサイトの開発に用いられることがプログラミング言語である。Node.js などのサーバサイドの実行環境も提供されている。

（１）プロトタイプベース

オブジェクト指向プログラミング言語のうち､プロトタイプ(Prototype)を基礎(Base)としてオブジェクトを扱うもの。クラスベースとは異なり、静的なクラスを用意せず、既存のオブジェクトを元に継承などを行う。JavaScript はプロトタイプベースのオブジェクト指向プログラミング言語である。

（２）型

JavaScript は動的型付け言語(Dynamically Typed Language)であるた

め、型の宣言を明示的に行う必要はない。

１）number
整数や浮動小数点数を扱う。

２）string
文字列を扱う。

３）boolean
真(true)と偽(false)の論理を扱う。

４）null
存在しないことを表す。

５）undefined
未定義を表す。

６）object
配列や連想配列、関数などはすべて object となる。

（３）JSON

JSON(JavaScript Object Notation、ジェイソン)はデータ交換フォーマットである。Web ページの開発で主に使用されているプログラミング言語である JavaScript のオブジェクト表記法をベースにしていて、人間からも構造を理解しやすい記法となっている。JavaScript 以外でも様々なプログラミング言語で JSON に対応しているため、広く普及しているデータ交換フォーマットとなっている。

１）JSON.stringify(value)
value に指定されたオブジェクトを JSON 文字列に変換して返す関数である。

２）JSON.parse(text)
text に指定された文字列を JSON として解析し、オブジェクトとして返す関数である。

## ２．４　Unity

Unity(Unity3D)は、IDE(統合開発環境、Integrated Development Environment)を内蔵したゲームエンジンである。開発はユニティ・テクノロジーズ社である。スクリプト言語には C#、UnityScript(JavaScript)、Boo の 3 種類を用いることができる。ゲームエンジンであるため、ゲームにおいて必ず使用されるような物理演算やレンダリングに関する機能が包括されている。また、Windows、Android、Web ブラウザなど様々な動作環境で動作するようにプロジェクトをビルドすることができる。これをクロスプラットフォーム、またはマルチプラットフォームという。

（１）プロジェクト

プロジェクト(Project)は、Unity におけるゲーム開発の単位の一つである。基本的に、一つのプロジェクトに一つのゲームが入る。

（２）シーン

シーン(Scene)は、ゲーム内における画面である。一つのゲームは複数のシーンによって構成される。

（３）ゲームオブジェクト

ゲームを構成するすべてのオブジェクトである。シーンに複数のゲームオブジェクトを設置する。

（４）ヒエラルキー

ヒエラルキー(Hierarchy)は、ゲームオブジェクト間の親子関係である。

（５）タグ

ゲームオブジェクトに関連付ける属性である。ゲームオブジェクトをタグによって分類することで、ゲームオブジェクトの走査を高速かつ容易に行えるようになる。

（６）コンポーネント

コンポーネント(Component)は、ゲームオブジェクトの動作の基礎となる部品である。Unity には標準で様々なコンポーネントが用意されている。自分で作成したスクリプトは基本的にコンポーネントとして利用することができるようになる。コンポーネント(Component)は、ゲームオブジェクトに取り付けて使用する。取り付ける操作をアタッチ(Attach)、取り外す操作をデタッチ(Detach)という。ゲームオブジェクト自体は明確な機能を保持していないため、コンポーネントをアタッチすることで、様々な機能を取り付けていく。ユーザが作成した MonoBehaviour のサブクラスもコンポーネントとして利用できるようになる。

（７）クラス

Unity において用意されているクラスを以下に列挙する。

１）MonoBehaviour

すべてのスクリプトのスーパークラスであり、アタッチ可能なすべてのコンポーネントはこの MonoBehaviour を継承して作成される。

① Start()

最初のフレームで呼び出され、実行される。

② Update()

毎フレーム呼び出され、実行される。

③ LastUpdate()

毎フレーム呼び出され、Update()の後に実行される。

２）GameObject

シーンに存在するすべてのゲームオブジェクトのベースクラスである。

① string name

ゲームオブジェクトの名前を表すプロパティである。

② GetComponent<Type type>()

ゲームオブジェクトにアタッチされている他のコンポーネントを取得する。

③ SetActive(bool value)

ゲームのアクティブ・非アクティブを設定するメソッドである。

④ GameObject Find(string name)

ゲームオブジェクトを名前で検索し、該当するものを返すメソッドである。

⑤ GameObject[] FindGameObjectsWithTag(string tag)

ゲームオブジェクトをタグマネージャーに登録したタグ名で検索し、該当するものを配列で複数個、返すメソッドである。

３）Transform

ゲームオブジェクトの位置、回転、スケールを扱うクラスである。また、ヒエラルキーも管理している。GameObject は Transform オブジェクトを保有している。

① Vector3 forward

オブジェクトの正面方向の単位ベクトルを表すプロパティである。

② Vector3 position

ワールド空間におけるオブジェクトの位置を表すプロパティである。

③ Quaternion rotation

ワールド空間におけるオブジェクトの回転を表すプロパティである。

④ Transform Find(string name)

子オブジェクトを検索し、取得することができるメソッドである。

⑤ LookAt(Transform target)

引数に指定したオブジェクトの方向を向くように回転することができるメソッドである。

⑥ Rotate(Vector3 eulerAngles, Space relativeTo = Space.Self)

第一引数に指定したベクトルの軸方向に回転をかけることができるメソッドである。第二引数が Space.Self の場合はローカル軸、Space.World の場合はワールド軸が回転軸として使用される。

⑦ SetParent(Transform parent, bool WorldPositionStays)
親オブジェクトを設定するメソッドである。WorldPositionStays が true の場合は親との相対的な位置・回転・スケールを維持して変更する。
⑧ RotateAround(Vector3 point, Vector3 axis, float angle)
座標 point を中心とした軸 axis で angle 度回転させるメソッドである。
⑨ Transform parent
親オブジェクトを表すプロパティである。

４）Vector3
3 次元ベクトルを表すクラスである。3 つのプロパティによって三次元空間における位置や方向を表現することができる。同様に 2 次元ベクトルを表す Vector２クラスも存在する。
① float x, y, z
ベクトルのそれぞれの成分を表すプロパティである。
② float magnitude
ベクトルの長さを読み取るプロパティである。
③ Vector3 normalized
単位ベクトル(ベクトルの長さ、magnitude が 1)を返すプロパティである。
④ Lerp(Vector3 a, Vector3 b, float t)
ベクトル a, b を補間係数 t で線形補間する。t の範囲は 0≦t≦1 となっている。

５）Quaternion
3 次元空間における回転情報を表すクラスである。3 次元空間による回転をオイラー角(Vector3 の 3 つのプロパティ)で表現しようとすると、ジンバルロックが発生し回転軸が制限されてしまうという問題がある。Quaternion では、四元数を利用してこの問題を解決している。しかし、これは複素数の概念を拡張(実部が 1 つ虚数部が 3 つ)した数体系であり、値の内容を直感的に理解することは難しい。そのため Unity ではすべての回転を表現するために、内部的に Quaternion を用いている。また、プログラミングを行ううえで、Quaternion のプロパティを直接編集するようなことは行わないような設計となっている。
① float w, x, y, z
クォータニオンのそれぞれの成分を表すプロパティである。
② Vector3 eulerAngles
回転をオイラー角(Vector3)で扱うことができるプロパティである。
③ Slerp(Quaternion a, Quaternion b, float t)

回転 a, b を補間係数 t で補間する。t の範囲は 0≦t≦1 となっている。

④ LookRotation(Vector3 forward)

forward に指定したベクトル方向への回転を返す。

⑤ Quaternion Euler(float x, float y, float z)

オイラー角で指定した回転をクォータニオンで返す。

６）CharacterController

物理特性を使用しないオブジェクト(主にキャラクター)の制御に利用するコンポーネントである。(物理特性を使用する場合は Rigidbody コンポーネントを利用する。)

① bool isGrounded

地面に接地している場合 true、そうでない場合 false を返すプロパティである。

② Move(Vector3 motion)

引数に指定したベクトル分座標を移動する。衝突判定があり、障害物がある場合は移動できない。

７）Mathf

数学定数、数学関数を扱うためのクラスである。

① float Deg2Rad

掛けることで度数法から弧度法（ラジアン）に変換できる定数である。

② float Rad2Deg

掛けることで弧度法（ラジアン）から度数法に変換できる定数である。

③ float PI

円周率(3.141592…)である。

④ float Sin(float f)

f のサインを返すメソッドである。f はラジアンで指定する。

⑤ float Cos(float f)

f のコサインを返すメソッドである。f はラジアンで指定する。

⑥ float Tan(float f)

f のタンジェントを返すメソッドである。f はラジアンで指定する。

８）WWW

WWW(World Wide Web)にアクセスすることができるクラスである。

① WWW(string url, WWWForm form) ※コンストラクタ

url にアクセスする URL を指定する。また、form に POST するフィールドを定義した WWWForm オブジェクトを指定する。

② string url

アクセスし、ダウンロードするURLが格納されているプロパティである。

③ bool isDone

ダウンロードが完了したかどうかを確認するために読み込むプロパティ。trueになっている場合、ダウンロードが完了していることを確認することができる。

④ string text

取得したページの文字列を読み取ることができるプロパティである。

⑤ string error

アクセス・取得エラーが発生した場合のエラー内容が格納されているプロパティである。

9）WWWForm

WWWクラスでWebサーバにアクセスする際に、データをPOSTメソッドで送信するためのクラスである。

① AddField(string fieldName, string value)

POSTのデータに値を追加する。fieldNameをキー、valueを値とする。

## 2．5 ネットワークとセキュリティ

### （1）HTTP

HTTP(ハイパーテキスト転送プロトコル、HyperText Transfer Protocol)とは、Web サーバとクライアントがデータを送受信する際に用いられるプロトコルである。ウェルノウンポート番号は80番である。

### （2）POST

POSTメソッドは、HTTPで定義されているメソッドの一つ。クライアントがWebサーバにデータを送信することができる。

### （3）HTTPステータスコード

HTTPにおいてWebサーバからのレスポンスを表現する3桁のコードである。

1）200 OK

リクエストの成功を表す。

2）404 Not Found

クライアントが要求するリソースを検出することができなかったことを表す。

（４）WebSocket

WebSocketは、Webサーバとクライアントの双方向通信の規格である。従来用いられていたAjaxやCometなどの手法よりも低コストでリアルタイム双方向通信を実現することができる。

（５）パスワードの暗号化

一般的に、漏えいのリスクを考慮し、ユーザのパスワードをデータベースに平文で直接保存してはならないとされている。そのため、パスワードには、不可逆変換アルゴリズムを適用することで暗号化する必要がある。また、暗号化の際に付与するデータをSALTという。暗号化アルゴリズムにはAES256などがある。

（６）SQLインジェクション

SQLインジェクション(SQL Injection)は、アプリケーションが想定していないSQL文を実行させることによって、不正な操作を行わせる脆弱性のことである。文字列を適切にエスケープすることで防止することができる。そのためには、プログラミング言語やライブラリ毎に用意されたバインド機構を利用する。

２．６　RDB

RDB(関係データベース: Relational DataBase)は、データモデルのひとつである関係モデルに基づいたデータベースである。現在使用されているデータベースのデータモデルとして最も主流なものである。また、RDBを扱うシステムを関係データベース管理システム(RDBMS: Relational DataBase Management System)という。システムのデータ全体をテーブル(表)の集合体として関連付けて表現する。様々な制約を付与することが可能で、高いメンテナンス性と整合性を持っている。代表的なRDBMSには、有償のものでOracle DatabaseやSQL Server、無償のものでMySQLやPostgreSQLがある。

（１）SQL

SQL(Structured Query Language)は、RDBMSを操作するためのデータ問い合わせ言語である。ISO(国際標準化機構: International Organization for Standardization)やANSI(米国国家規格協会: American National Standards Institute)によってある程度標準化が行われており、これに準拠した文法を標準SQLと呼ぶ。

（２）テーブル

テーブル(Table)は、RDBにおける基本単位である。カラムの定義に基づいた複数のレコードが格納された表のようなものである。RDBでは、複数のテーブルを関連(Relation)させる。

（３）レコード

レコード(Record)は、テーブルを構成する単位である。データ１件分のことであり、表の行に相当する。

（４）カラム

カラム(Column)は、テーブルを構成する単位である。レコードに存在する属性をカラムとして定義する。表の列に相当する。

（５）正規化

正規化(Normalization)とは、余分なフィールドを除去し、テーブルを適切な手法で関連付けることで、データベースを正しく関数従属した構造になるように設計することである。RDB の設計においては、データ間の一貫性を確保するために必須の作業となっている。

１）非正規形

正規化が行われていない状態。

２）第１正規形

1 レコードのフィールドの繰り返し項目を削除する。また、導出フィールド(他のフィールドから算出可能なフィールド)を削除する。

３）第２正規形

部分関数従属(キーによって値が定まる関係)しているフィールドを別のテーブルに分離する。

４）第３正規形

推移関数従属(間接的に関数従属している関係)しているフィールドを別のテーブルに分離する。

（６）ISO5218

国際標準化機構(ISO)が作成したヒトの性別の表記に関する国際規格である。

0 … 不明(not known)
1 … 男性(male)
2 … 女性(female)
9 … 適用不能(not applicable)

（７）DDL

DDL(データ定義言語、Data Definition Language)は、SQL においてデー

タベース構造を定義するための構文である。

１）CREATE

テーブルを作成する。

２）DROP

テーブルを削除する。

３）ALTER

テーブルの構造を変更する。

（８）DML

DML(データ操作言語、Data Manipulation Language)は、SQL においてデータの検索・登録・更新・削除を行うための構文である。

１）SELECT

データを検索する。

２）INSERT

データを挿入する。

３）UPDATE

データを更新する。

４）DELETE

データを削除する。

（９）DCL

DCL(データ制御言語、Data Control Language)は、SQL においてデータに対するアクセス権限を制御するための構文である。

１）GRANT

データベースユーザに権限を与える。

２）REVOKE

データベースユーザから権限を剥奪する。

（１０）制約

制約とは、フィールドに指定する値等に制限をかけることである。テーブルを定義する際、カラムごとに指定する。

１）NOT NULL 制約

NULL 値を禁止する制約である。SQL において、NULL 値は 3 値論理を考慮する必要があるうえ､NULL を含む演算はすべて NULL となってしまう。このような NULL の伝播を防ぐため、なるべく用いらないことが多い。

２）チェック制約

フィールド値に条件を設ける。

３）一意性制約

カラムに重複したデータの挿入を禁止する。

４）主キー制約

テーブル内のレコードを一意に識別するカラムに設定される。RDB においてテーブル間の関連付けに用いられるキーとなる。

５）参照整合性制約

外部テーブルを参照し、存在しないレコードのデータの挿入を禁止する。

## ２．７　MariaDB(MySQL)

MySQL は、Oracle 社が開発・公開するオープンソースソフトウェアの RDBMS である。MariaDB は MySQL から派生したプロジェクトで、MySQL の高い後方互換性を持つ RDBMS である。Linux サーバの CentOS7 では MariaDB がプリインストールされている。また、データベースエンジンとして InnoDB が標準搭載されている。

### （１）データ型

１）INT

整数値を扱う。

２）DOUBLE

浮動小数点数を扱う。

３）CHAR(文字数)

固定長の文字列を扱う。

４）VARCHAR(文字数)

可変長の文字列を扱う。

５）DATETIME

日付と時刻を扱う。

６）NULL

不定や不明を表す特殊な値である。

### （２）構文

１）SHOW

①　SHOW TABLES

テーブル一覧を表示する。

②　SHOW DATABASES

データベース一覧を表示する。

③　SHOW COLUMNS

テーブルのカラム一覧を表示する。

２）CREATE

①　CREATE TABLE

テーブルを作成する。

②　CREATE DATABASE

データベースを作成する。

３）DROP

① DROP TABLE

テーブルを削除する。

４）ALTER

① ALTER TABLE

テーブルのカラム構成を変更する。

４）USE

使用するデータベースを選択する。

（３）制約

１）NOT NULL

NOT NULL 制約を設定する。

２）PRIMARY KEY

主キー制約を設定する。

３）UNIQUE

一意性制約を設定する。

４）REFERENCES

参照性制約を設定する。

５）AUTO_INCREMENT

自動的に連番の値を格納する。

## ２．８ Node.js

Node.js は Linux サーバサイドの JavaScript 実行環境である。Apache などと同じように Web サーバシステムを構築することができる。また、処理が非常に効率的であり、サーバに対する大量のアクセスを処理することを得意としている。さらに、様々な拡張モジュールが提供されている。これらは、npm コマンドでインストールすることができる。利用するためにはスクリプト内から、require()関数を用いて参照する必要がある。

（１）C10K 問題

C10K(クライアント 1 万台問題)とは、サーバに対するアクセス要求が、1 秒間当たり約 1 万を超えることで、スレッドの増大によりメモリリソースが不足し、処理速度が急速に低下してしまう問題である。Apache などの Web サーバソフトウェアではマルチスレッドによるメモリ不足によってこの問題が発生してしまう。

（２）ノンブロッキング I/O

Node.js では、一度に一つの処理を行うシングルスレッドの形式を採用している。ノンブロッキング I/O では、シングルスレッドで複数の処理を非同期的に切り替えながら実行することができる。そのため、マルチスレ

ッド形式の Apache と違い、メモリリソースの消費が少なく、C10K 問題を解決することができる。

（３）events モジュール

Node.js におけるイベントリスナを扱うためのモジュールである。イベントリスナとは、イベントと処理を結びつける仕組みのことである。http モジュールや Socket.IO モジュールにおいても使用されている。

① EventEmitter

イベントを表現するクラスである。

② EventEmitter.on(event, listener)

イベントを登録する。event にイベント名、listener に処理を指定する。

③ EventEmitter.once(event, listener)

一度限りのイベントを登録する。

④ EventEmitter.emit(event, 引数)

②や③で登録したイベントを実行する。

（４）http モジュール

HTTP サーバを作成する場合に使用するモジュールである。

① Server createServer()

新規の Web サーバオブジェクトを作成する。また、作成した Web サーバを Server オブジェクトとして返す。

② Server.listen(port)

port に指定したポート番号でアクセスの受け入れを開始する。

③ request(IncomingMessage, ServerResponce)イベント

リクエストが発生する度に実行されるイベントである。IncomingMessage オブジェクトと ServerResponce オブジェクトを引数に渡す必要がある。

④ Server.on(event, listener)

サーバの処理を設定する。event に’request’を設定することで、③のイベントを記述することができる。

⑤ IncomingMessage

HTTP リクエストのステータス、ヘッダ、およびデータにアクセスするオブジェクトである。

a　method

リクエストメソッドを表すプロパティである。

b　url

HTTP のリクエスト URL を表すプロパティである。

⑥　ServerResponse

HTTP リクエストに対するクライアントへのレスポンスを表すオブジェクトである。

a　writeHead(statusCode, [statusMessage])

レスポンスヘッダを送信する。statusCode は HTTP ステータスコード、オプションを[statusMessage]に指定する。

b　end([data])

レスポンスの終了を伝える。[data]には、送信するデータを指定する。

（５）querystring モジュール

クエリ文字列(HTTP リクエストパラメータ)を解析し、オブジェクトに変換するためのモジュールである。

①　perse(str)

クエリ文字列 str をオブジェクトに変換して返す。

（６）mysql モジュール

MySQL や MariaDB に接続するためのモジュールである。

①　createConnection({options})

データベースとの接続を作成し、Connection オブジェクトを返す。オプションは連想配列で指定する。主要なオプションを以下に列挙する。

a　user

データベースのユーザ名を指定する。

b　password

a で指定したユーザのパスワードを指定する。

c　database

アクセスするデータベースを指定する。

②　Connection.connect()

データベースとの接続を開始する。

③　Connection.query(str)
SQL 文を実行する。

（７）crypto モジュール
Cryptographic API を利用するためのモジュールである。Cryptographic API とは、Windows で提供されている暗号化などを目的とした標準フレームワークである。

①　createCipher(algorithm, password)
アルゴリズムとパスワードを指定して Cipher オブジェクトを返す。アルゴリズムには、aes256 などがある。

②　Cipher.update(data, [input_encoding], [output_encoding])
data で暗号を更新する。[input_encoding]や[output_encoding]には、utf8、ascii、binary(2 進法)、hex(16 進法)などを指定する。

③　Cipher.final([output_encoding])
暗号化されたコンテンツを返す。

（８）Socket.IO
リアルタイム双方向通信を実現することに特化した Node.js のモジュールである。モジュール名は socket.io となっている。ポーリングや Comet、WebSocket といった複数の通信技術を包括して一つの統一された API を提供しているため、簡単にリアルタイム双方向通信を実現できるようになる。

１）sockets.on(‘connection’, function(socket){} )
第二引数にメインの処理を設定する。

２）socket.on(event, function(data){} )
イベントを設定し、待ち受ける。

３）socket.emit(event, [data] )
イベントを発火したクライアントのみにイベントとデータを送信する。

４）socket.broadcast.emit(event, [data] )
イベントを発火したクライアント以外全部にイベントとデータを送信する。

５）io.sockets.emit(event, [data] )
接続している全てのクライアントにイベントとデータを送信する。

６） socket.on( ‘disconnect’ , function(data){} )
切断時のイベントを設定する。

## ３．方法

### ３．１　システム

（１）クライアント

クライアントは Unity からビルドされたものを使用する。ビルドは WindowsPC を対象に行う。

（２）サーバ

サーバの OS は Linux の一つである CentOS7.0 を使用する。

（３）MariaDB

CentOS7.0 にはプリインストールされている。ユーザのゲームプレイ情報を格納するデータベースを作成する。

（４）Node.js

Web サーバの作成に用いるため、CentOS7.0 にインストールする。npm コマンドにより、MariaDB にアクセスするための mysql モジュールや、リアルタイム通信を実現するための socket.io モジュールも同時に導入を行う。

（５）app.js

自作の Node.js ファイルである。/var/www/html/に保管し、node コマンドで指定し、実行することでゲームのマスタサーバを起動することができる。

### ３．２　Unity

作成したクラスおよび開発手法を以下に列挙する。

（１）Character

ゲームのキャラクターを表現する抽象クラスである。プレイヤーやモンスターといったゲームキャラクターおよびそのゲームキャラクターが生成する攻撃弾等はすべて Character のサブクラスとなる。また、プレイヤーの操作は PlayablePlayer クラスにより行われる。

１）int hp

メンバ変数 hpValue にアクセスするためのプロパティである。ゲームキャラクターのヒットポイントを表している。0 未満もしくは maxHp を超えた値は設定できないようになっている。

２）int maxHp

キャラクターの最大 HP を表すプロパティである。

３）Damage(int dmg)
被弾時に実行されるメソッドである。dmg だけ hp が減少する。

４）CreateAttack(GameObject prafab, float mgn, int attackCount)
prefab に指定したプレハブをもとに攻撃弾を生成する。mgn には攻撃力倍率、attackCount にはキャラクター自身の累計攻撃回数を代入する。生成した攻撃弾子オブジェクトには、キャラクター自身の様々な情報を渡す。

５）SearchTarget(float d)
距離 d の範囲内でターゲット(攻撃対象)を走査する。検出したターゲットは、target プロパティに格納する。

６）bool RemoveTarget()
ターゲットを解除する。また、ターゲットの解除処理が行われた場合は true を返す。

７）string role
キャラクターの勢力、派閥を表している。現在の仕様では、“Player”もしくは”Monster”が入力される。？？？？？？？？？？？？

（２）Player
プレイヤーキャラクターの動作を記述しているクラスである。Character クラスを継承している。

１）Update()
Update()メソッドをオーバーライドし、アクション時の処理を記述した。
２）Damage(int dmg)
Damage メソッドをオーバーライドし、既存の機能に加え、効果音やパーティクル生成、カメラの回転など様々なエフェクトを追加した。
３）Avoid()
回避行動が開始した場合に呼び出されるメソッドである。
４）Attack()
攻撃行動を開始した場合に呼び出されるメソッドである。
５）ChangeWeapon(int n)
武器変更行動を開始した場合に呼び出されるメソッドである。n には武器番号を入力する。

（３）PlayablePlayer

Player のオブジェクトを操作するためのクラスである。Player がアタッチされたゲームオブジェクトにアタッチして使用する。このクラスは通常通り MonoBehaviour を継承している。

１）Update()

キーボード入力の受付や、入力に対する適切な Player クラスメソッドの呼び出しを行っている。

２）decelerateMoving()

移動速度を 0.9 倍に減速するメソッドである。Update()メソッドの中から呼び出される。

３）decelerateAngle()

向き回転速度を 0.9 倍に減速するメソッドである。Update()メソッドの中から呼び出される。

（４）Monster

通常の敵キャラクターの動作を記述する抽象クラスである。個別の全てのモンスターのクラスのスーパークラスとなる。Character クラスを継承している。

１）Damage(int dmg)

Damage()メソッドは、オーバーライドし、死亡時のパーティクル生成や、ゲームオブジェクト破棄機能が追加している。

（５）CharacterControllerGravity

CharacterController の動作に重力を実装するためのクラス。CharacterController からは sealed 修飾子によりサブクラスを作成することができない。そのため、CharacterControllerGravity は MonoBehaviour を継承し、GetComponent<CharacterController>() を用いて CharcterController を制御する構造になっている。

１）Update()

落下速度の加速やゲームオブジェクトの移動、着地判定などを行っている。

２）Jump(float power)

ジャンプ処理を実装するためのメソッドである。

（６）Attack

プレイヤーやモンスターなどの様々なキャラクターが生成する攻撃弾を表現する抽象メソッドである。全ての攻撃弾はこのクラスを継承している。

１）OnTriggerEnter(Collider col)

敵キャラクター(role が自分自身と異なる)との当たり判定を行うメソッドである。また、接触した場合の Character.Damage()メソッドの呼び出しや、パーティクル生成、ダメージテキストの生成も行っている。

## ３．３　DB

以下に MariaDB で作成したテーブルを列挙する。

（１）sexes

すべての性別を格納したテーブルである。なお、レコードは ISO5218 に準拠した内容のものを予め挿入しておく。

１）id

性別番号を格納する主キーである。

２）name

性別名を格納する。

（２）users

ゲームユーザの情報を格納したテーブルである。

１）id

プレイヤー番号を格納する主キーである。なお、自動的に連番が行われるため、挿入時に番号を指定する必要性はない。

２）uid

プレイヤーのユーザ ID を格納する。ユーザが自由に設定することができるが、重複は許可されていない。

３）name

プレイヤーのニックネームを格納する。ユーザが自由に設定することができる。

４）password

プレイヤーがログインの際に用いるパスワードを暗号化して格納する。暗号化は個人情報の漏洩を防止のために行う。暗号化アルゴリズムには、AES256 を用いる。

５）sex

プレイヤーの性別番号を格納する。番号は sexes テーブルを参照している。

## ４．結果


図 1 ゲームスタート時

ログインに成功するとこのような画面に遷移する。画面中央下に表示されている市松模様の球体がプレイヤーである。←→キーを押下すると視点および移動方向を回転、↑↓キーを押下すると前後移動をする。また、HP(プレイヤーの体力、残機)が左上の緑色のバーで示されている。青いバーは下記に挙げる各々のアクション実行時の残り所要時間を表している。

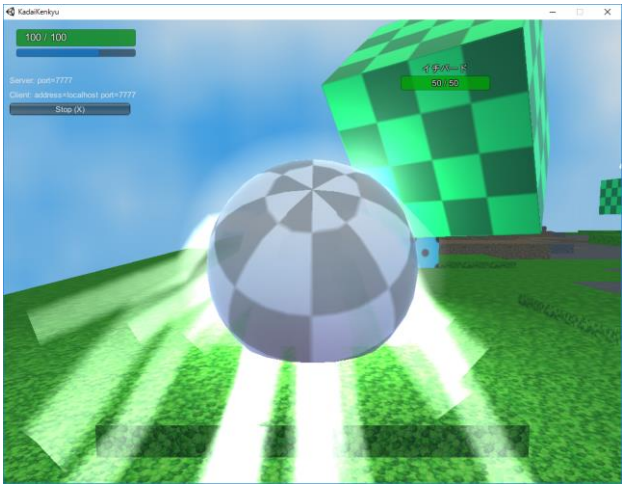


図 2 回避アクション

S キーを押下すると、向いている方向に高速移動しながら回避行動をとる。白い光と残像エフェクトが表示されている間は、敵キャラクターの攻撃弾に接触した場合も、ダメージアクションが発生しない無敵時間となる。

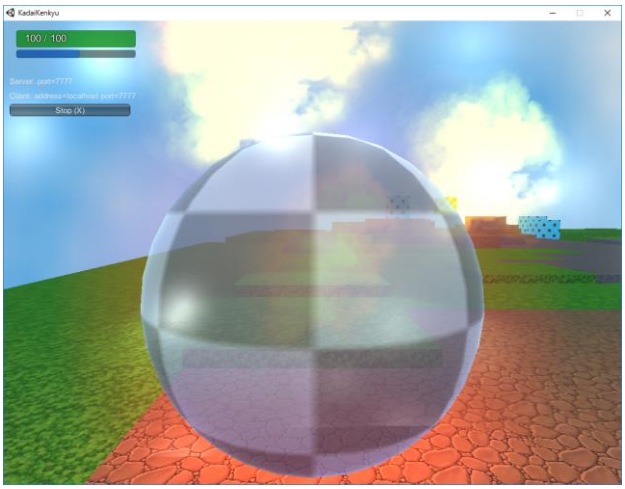


図 3 武器変更アクション

プレイヤーが所持している武器を Q, W, E キーでそれぞれ対応した武器に変更することができる。武器変更中は赤いパーティクルエフェクトが現れる。なお、攻撃は A キーを押下することで行う。

Q … ソード
W … ダガー
E … アロー


図 4 ダメージアクション

敵キャラクターが生成する攻撃弾オブジェクトにプレイヤーが接触するとダメージアクションが実行される。画面中央に減少した HP の量が赤字で表示され、左上の HP ゲージに受けたダメージ量が反映される。

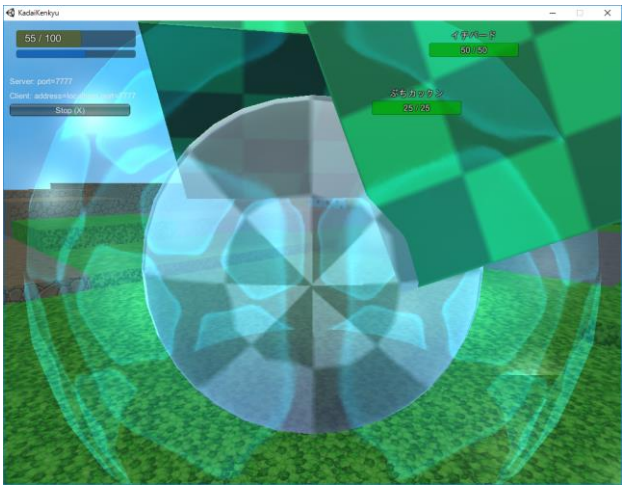


図 5 ダメージ後の無敵時間

図 4 のようにダメージを受けたあとは、一定時間プレイヤーを囲む青いエフェクトが表示される。このエフェクトが表示されている間は無敵となり、ダメージアクションが発生しない。この無敵時間もアクションの一つであるため、この間は他のアクションを実行することができない。


図 6 ソード

武器「ソード」を使用した場合の図。青い斬撃エフェクトがプレイヤー前方向を球面上に上から下へ回転する。攻撃力倍率は 1.0 倍。


図 7 ダガー

武器「ダガー」を使用した際の図。緑の斬撃エフェクトがプレイヤーの前方向を直線的に落下する。攻撃力倍率は 0.4 倍。


図 8 アロー

武器「アロー」を使用した際の図。赤紫の斬撃エフェクトとパーティクルエフェクトがプレイヤーの前方向の遠距離に落下していく。攻撃力倍率は 0.8 倍。

## ５．考察

今日アプリケーションの入力装置および出力装置には、マウスやキーボードや通常の液晶ディスプレイの他にも、タッチパネルや VR に関連したヘッドマウントディスプレイおよびハプティックデバイスなど様々なものがある。Unity ではクロスプラットフォームなビルド機能を実装しているため、これらの複数のプラットフォームに対応した、実行プログラムを生成することができる。しかしクロスプラットフォームなアプリケーションを実現するためには、これに加えて、それぞれのデバイスに最適なユーザインターフェースを一つ一つ開発していく必要がある。現時点のユーザインターフェースは、キーボードとディスプレイを搭載した PC しか考慮できていない。Unity のクロスプラットフォームなビルド機能の利便性を享受することも、本研究の目的の一つとして掲げているため、ユーザインターフェースの追加が今後の課題と考えられる。

また本アプリケーションは PC 上で作成したため、PC よりスペックが低いスマートフォンなどで動作させた場合、描画処理などの負荷が大きくなってしまい軽快に動作しない可能性が懸念される。実際に描画処理の最適化を適切に行えていないために、スマートフォン端末が激しく発熱したり、処理落ちが発生したりしてしまうゲームアプリは、商用のものでも数多く見受けられる。Unity では、解像度の変更機能や、描画処理負荷の指標として DrawCall という値が設けられているため、これらを活用してよりアプリケーションの軽量化を図っていきたい。

また今回開発した作品において、Unity クライアントとマスタサーバ間の通信方式の一つとして、HTTP の POST メソッドが用いられている。そのためゲームクライアント外から発行されたような、不正な HTTP リクエストでも値が正常であれば、マスタサーバ側は処理を実行してしまうことになる。またプレイヤーの能力値などのゲーム進行において、重要な値はメモリ上に直接保存している。そのためエンドユーザがメモリ操作ツールを用いた場合、不正に変更されチートが発生してしまう可能性が懸念される。さらにオンラインゲームでランキングの改竄や、課金通貨の偽装などのチートが発生してしまった場合は、ユーザの減少や損害賠償問題、利益の減少につながってしまう。そのため今回の課題研究のようにエンドユーザを対象とした、大規模システムを構築する場合は、セキュリティ管理が特に重要となる。POST メソッドを利用している場合は、ハッシュなどを用いてゲームアプリケーション間の正しい通信であると認識する仕組みを、実装することが必要になると考えられる。またメモリ改竄の問題を解決するためには、重要な値のビット列を排他的論理和演算した後に、格納するといった方法が考えられているようである。そのためこのような機能を実装したクラスを実装し、利用していくことが必要になると考えられる。そして SQL インジェクションや、クロスサイトリクエストフォージェリなどの主要な Web アプリケーション脆弱性を対象に、ペネトレーションテストの実行を試みることも重要であると考えている。

## ６．感想

今回の課題研究では、情報工学科で習得したさまざまな知識を活かし、理解を深めることができたと思う。特に、オブジェクト指向やそのポリモーフィズムに関してはかなり理解を深めることができた。キャラクターの動作を記述するためにオブジェクト指向プログラミングのポリモーフィズムの概念が頻繁に利用された。また、ゲームオブジェクトの挙動を実装するために三角関数をはじめとしたさまざまな数学関数を利用した。そのため、三角関数の性質および実践的な利用方法について理解することができた。さらに、正しく正規化された関係データベースを構築し利用したため、RDB の設計も柔軟に行えるようになった。ゲームは、科展や総科祭、学校説明会などのさまざまな学校行事で展示することができ、来校者やプレイユーザから良いレスポンスを多数いただくことができたので、非常にうれしく思った。

また、情報工学科の授業では MySQL や node.js、Socket.IO について取り扱わなかった。近年 Web 開発業界で流行しているリアルタイム通信や大量アクセスに対応した技術や、MySQL について理解を深めることができたのは、とても今後のためになると思った。

しかし、ライブラリだけでも node.js、Socket.IO、そして、プログラミング言語では C#、JavaScript、さらに Unix 関連のサーバサイドの知識を総合的に活用するような課題研究の内容だったため、概念や知識の習得に非常に時間を費やしてしまった。個々の技術の仕組みもそれぞれ難解な部分があり、バグや予知しない挙動の発生により大幅にスケジュールが遅れた場面も数多くあり非常に大変であった。その結果、ゲームクライアントのグラフィックやインターフェース、セキュリティ関連の対策など個々の作りこみが少し浅くなってしまった。この部分に関しては反省したいと思っている。特に、セキュリティ関連の対策は他のすべての大規模システムで必須となっているので、ここは特に努力して改善したいと思っている。

今回は長い年月をかけて大規模な作品を開発することができたため、セキュリティ問題を解決した後、Web 上に公開することで、エンドユーザからさまざまなフィードバックを得ることで、新たな問題点を見つけて改善することができたら良いと思っている。さらに、課金通貨の導入やゲーム内広告の導入を行い、ビッグデータを収集し、データマイニングを行うことで収益化やビジネス化に向けた研究に発展できると良いと思っている。

今回の課題研究で習得した知識を活かし、大学でも情報科学や画像工学を専攻していきたいと思っている。また将来、システム開発プロジェクトのメンバとなった場合も、これらの知識を活かして開発に貢献していきたいと思っている。

## ７．参考文献


・C#
（ URL: https://msdn.microsoft.com/ja-jp/library/kx37x362.aspx ）
・JavaScript | MDN
（ URL: https://developer.mozilla.org/ja/docs/Web/JavaScript ）
・Unity5 入門 最新開発環境による簡単 3D&2D ゲーム制作
荒川 巧也(著), 浅野 祐一(著)
・Unity - マニュアル
（ URL: https://docs.unity3d.com/ja/current/Manual/ ）
・Unity - スクリプトリファレンス
（ URL: https://docs.unity3d.com/ja/current/ScriptReference/ ）
・ゼロからはじめるデータベース操作 SQL
ミック(著)
・ドキュメント | Node.js
（ URL: https://nodejs.org/ja/docs/ ）
・Socket.IO - Docs
（ URL: http://socket.io/docs/ ）

８．添付資料

８．１　Unity

１）Attack.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class Attack : Character {

	public int attackCount;
	public float mgn;

	// オブジェクト
	private GameObject damageEffect, canvas, damageText;

	public override void Start () {
		base.Start ();

		// オブジェクトの取得
		canvas = GameObject.Find ("Canvas");
		damageText = Resources.Load
("Prefabs/Effects/DamageText") as GameObject;
		damageEffect = Resources.Load
("Prefabs/Effects/DamageEffect") as GameObject;
	}

	public override void Update () {
		base.Update ();
	}

	// 敵対キャラクターとの当たり判定
	void OnTriggerEnter(Collider col) {
		if (col.gameObject.tag == "Character") { // Character スクリプトを保持しているかどうかを判定
			Character colC = col.GetComponent<Character> (); // Character スクリプトを取得

			// role が異なり、無敵状態でないとき攻撃が成立する
```

```
                    if ( role != colC.role && !colC.invincible ){

                        int dmg = Mathf.RoundToInt ( atk * mgn *
Random.Range(0.9f, 1.1f) ); // ダメージ量の計算
                        colC.Damage (dmg); // ダメージを与える

                        //「カメラの視界内に入っている」場合にダメ
ージテキストを生成
                        if
(Vector3.Angle(                    Camera.main.transform.forward,
Camera.main.transform.position - transform.position ) > 120) {
                            GameObject          go          =
(GameObject)Instantiate  (damageText,  Camera.main.WorldToScreenPoint
(transform.position + new Vector3(0, -0.3f, 0)), Quaternion.identity);
                            go.transform.SetParent
(canvas.transform); // 親を Canvas に設定
                            go.GetComponent<Text>   ().text   =
dmg.ToString (); // ダメージテキストにダメージ量を反映
                        }

                        Instantiate(damageEffect,
transform.position, Quaternion.identity); // ダメージエフェクト生成

                    }

            }
        }
}
```

２）Attack_Arrow.cs

```
using UnityEngine;
using System.Collections;

public class Attack_Arrow : Attack {

	private Quaternion angle;
	private Vector3 pos, dir;

	public override void Start () {
		base.Start ();

		pos = transform.position; // 位置を取得
		angle = transform.rotation; // 攻撃者の向きを取得
		transform.position = pos; // 位置設定
		dir = angle * new Vector3 (0, 2.3f, 12.5f); // 移動単位
の設定
		transform.SetParent(null); // 親解除
	}

	public override void Update () {
		base.Update ();

		transform.position += dir * Time.deltaTime; // 移動
		dir.y += Physics.gravity.y * Time.deltaTime; // 落下速度
を加速
	}

}
```

３）Attack_Axe.cs

```
using UnityEngine;
using System.Collections;

public class Attack_Axe : Attack {

	private Quaternion angle;
	private Vector3 pos, axis;
	private float rot;

	public override void Start () {
		base.Start ();

		pos = transform.position; // 位置を取得
		angle = transform.rotation; // 向きを取得
		transform.position += new Vector3 (0, 0.7f, 0); // 位置
設定

		// 回転軸を設定
		axis = angle * new Vector3(1, 0, 0);
	}

	public override void Update () {
		base.Update ();

		if ( rot < 180 ) {
			transform.RotateAround(pos,	axis,	speed	*
Time.deltaTime); // 回転
			rot += speed * Time.deltaTime;
			speed += 15;
		}
	}

}
```

４）Attack_Dagger.cs

```
using UnityEngine;
using System.Collections;

public class Attack_Dagger : Attack {

	private Quaternion angle;
	private Vector3 pos, dir;

	public override void Start () {
		base.Start ();

		angle = transform.rotation; // 攻撃者の向きを取得
		attackCount = GetComponent<Attack>().attackCount; // 攻撃者の攻撃回数を取得
		transform.position += transform.up * 0.5f + transform.forward * 0.3f; // 位置設定

		// attackCount の値を-10 or 10 に変更
		attackCount %= 2;
		if ( attackCount == 0 ) attackCount = -1;
		attackCount *= 10;

		angle *= Quaternion.Euler(0, attackCount, 0); // 角度調整
		dir = angle * new Vector3 (0, -1.0f, 0.6f); // 移動単位の設定
	}

	public override void Update () {
		base.Update ();

		transform.position += dir * 0.1f; // 移動
	}

}
```

５）Attack_IchiBird.cs

```
using UnityEngine;
using System.Collections;

public class Attack_IchiBird : MonoBehaviour {

	public float speed;

	void Start () {
		transform.position += transform.up * 0.6f;
		transform.SetParent (null);
	}

	void Update () {
		transform.position += transform.forward * speed *
Time.deltaTime;
	}
}
```

６）Attack_Sword.cs

```
using UnityEngine;
using System.Collections;

public class Attack_Sword : Attack {

	private Quaternion angle;
	private Vector3 pos, axis;

	public override void Start () {
		base.Start ();

		pos = transform.position; // 位置を取得
		angle = transform.rotation; // 向きを取得
		attackCount = GetComponent<Attack>().attackCount; // 攻撃回数を取得
		transform.position += new Vector3 (0, 0.7f, 0); // 位置設定

	 // attackCount の値を-10 or 10 に変更
	 attackCount %= 2;
	 if ( attackCount == 0 ) attackCount = -1;
	 attackCount *= 10;

	 // 回転軸を設定
		axis = Quaternion.Euler(0, attackCount, 0) * angle * new Vector3(1, 0, 0);
    }

	public override void Update () {
		base.Update ();

		transform.RotateAround(pos, axis, 120.0f / 20); // 回転
    }

}
```

７）AvoidEffect.cs

```
using UnityEngine;
using System.Collections;

public class AvoidEffect : MonoBehaviour {

	private Light lightComponent;
	private int frame;
	private GameObject go;
	private int i;

	public GameObject avoidTrail;

	void Start () {
		lightComponent = gameObject.GetComponent<Light> (); // Light コンポーネントを取得
		frame = -1; // frame が-1 の場合は Update の内容が動作しない
		// avoidTrail を複製する。
		for (i = 0; i < 20; i++) {
			go = Instantiate (avoidTrail);
			go.transform.SetParent (transform);
		}
	}

	void Update () {
		if (frame != -1) {
			frame++;
			if (frame <= 10) {
				lightComponent.intensity += 0.3f;
			} else if (frame >= 17 && frame < 27) {
				lightComponent.intensity -= 0.3f;
			} else if (frame >= 27) {
				lightComponent.enabled = false;
				lightComponent.intensity = 0;
				frame = -1;
				foreach (Transform child in transform) {
					child.GetComponent<TrailRenderer>().enabled = false;
```

```
                        }
                    }
                }
        }

        public void EffectStart () {
                lightComponent.enabled = true;
                frame = 0;
                foreach (Transform child in transform) {
                        child.GetComponent<TrailRenderer>  ().enabled  =
true; // トレイルレンダラーを有効にする
                        child.GetComponent<TrailRenderer> ().Clear(); //
トレイルを消す
                        // 位置をランダムに配置
                        child.position    =    transform.position    +
Quaternion.Euler (
                                Random.Range (0, 361),
                                Random.Range (0, 361),
                                Random.Range (0, 361)
                        ) * new Vector3 (Random.Range (0.0f, 0.4f), 0, 0);
                }

        }
}
```

８）Buchikakkun.cs

```
using UnityEngine;
using System.Collections;

public class Buchikakkun : Monster {

	public float rotateSpeed;

	private int frame;
	private Vector3 vec, dir;

	// コンポーネント
	private CharacterController cc;
	private CharacterControllerGravity ccg;
	private Animator animator;

	// オブジェクト
	private GameObject attack;

	public override void Start () {
		base.Start ();

		// コンポーネントの取得
		cc = GetComponent<CharacterController> ();
		ccg = GetComponent<CharacterControllerGravity> ();
		animator = GetComponent<Animator> ();

		// オブジェクトの取得
		attack                                                    =
Resources.Load("Prefabs/Attacks/Attack_Buchikakkun") as GameObject;
	}

	public override void Update () {
		base.Update ();

		// ターゲットが存在しないときは、ターゲットを探す
		if (target == null) {
			SearchTarget (3.0f);
		}
```

```
            // ターゲットが存在するとき
            else {

                vec = target.transform.position - transform.position; // ターゲットの方向を算出

                // 非攻撃時(着地時)の挙動
                if (action == "" && cc.isGrounded) {

                    dir = Vector3.zero; // 移動ベクトルの初期化

                    // 何もしない
                    if (frame < 45) {
                    }
                    // ゆっくりとターゲットの方向を向く
                    else if (frame < 90) {
                            transform.rotation = Quaternion.Slerp (transform.rotation, Quaternion.LookRotation (new Vector3 (vec.x, 0, vec.z)), rotateSpeed * Time.deltaTime);
                        }
                    // 何もしない
                    else if (frame < 135) {
                        }
                    // 移動
                    else if (frame < 180) {
                            dir = vec.normalized * speed;
                            dir.y = 0;
                        }
                    // 何もしない
                    else if (frame < 225) {
                        }
                    // ジャンプ
                    else if (frame == 225) {
                            dir = vec.normalized * speed;
                            dir.y = 0;
                            ccg.Jump (3.5f);
                        }
                    // 接地したらフレームを戻す
```

```
                        else {
                                frame = 0;
                        }

                        // ターゲットと距離が近いときに攻撃モードに移行させる
                        if (vec.magnitude <= 1.5f) {
                                dir = Vector3.zero;
                                action = "Attack";
                                frame = 0;
                        }

                        // ターゲット解除
                        if (RemoveTarget(4.5f)) {
                                frame = 0;
                                dir = Vector3.zero;
                                action = "";
                        }

                }

                // 攻撃時の挙動
                else if (action == "Attack") {

                        // 何もしない
                        if (frame < 60) {
                                transform.rotation = Quaternion.Slerp (transform.rotation, Quaternion.LookRotation (new Vector3 (vec.x, 0, vec.z)), rotateSpeed * Time.deltaTime);
                        }
                        // 攻撃準備
                        else if (frame == 60) {
                                animator.SetTrigger ("AttackReady");
                        }
                        // 何もしない
                        else if (frame < 180) {
                        }
                        // 攻撃
                        else if (frame == 180) {
```

```
                        animator.SetTrigger ("Attack");
                        CreateAttack (attack, 1); // 攻撃の
生成
                    }
                    // 何もしない
                    else if (frame < 210) {
                    }
                    // 攻撃が終了したので通常モードに移行する。
                    else if (frame == 210) {
                        animator.SetTrigger ("Normal");
                    }
                    // 何もしない
                    else if (frame < 330) {
                    }
                    // フレームを戻し、通常モードに移行する。
                    else {
                        frame = 0;
                        action = "";
                    }

                }

            }

            cc.Move(dir * Time.deltaTime); // 移動処理
            frame++; // フレームカウント
        }
}
```

９）CanvasController.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class CanvasController : MonoBehaviour {

	// オブジェクト
	public GameController gc;
	private Transform canvas;
	private Slider hpBar, actionBar;
	private SliderValueLerp hpBarLerp;
	private Text hpBarText;
	private Player player;

	void Start () {

		// オブジェクトの取得
		gc = GameObject.FindGameObjectWithTag
("GameController").GetComponent<GameController> ();
		canvas = GameObject.Find ("Canvas").transform;
		hpBar = canvas.FindChild ("HpBar").GetComponent<Slider>
();
		actionBar = canvas.FindChild
("ActionBar").GetComponent<Slider> ();
		hpBarLerp = canvas.FindChild
("HpBar").GetComponent<SliderValueLerp> ();
		hpBarText =
hpBar.transform.FindChild("Text").GetComponent<Text>();

	}

	void Update () {

		// プレイヤーオブジェクトがある場合の処理
		if (player) {

			hpBarLerp.ChangeValue ((float)player.hp /
player.maxHp); // HP バーの変更
```

```
                    hpBarText.text = (Mathf.RoundToInt (player.maxHp *
hpBar.value)) + " / " + player.maxHp; // HP バーの Value から表示する数
値を計算

                    // アクションバーの変更
                    if (player.action == "") {
                        actionBar.value = 1;
                    } else {
                        actionBar.value  =  (float)player.frame  /
player.maxFrame;
                    }

                }

                // プレイヤーオブジェクトがない場合の処理
                else {
                    // プレイヤオブジェクトの取得
                    if (gc.player) {
                        player = gc.player.GetComponent<Player> ();
                    }
                }

        }
}
```

１０）Character.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.Networking;

public class Character : MonoBehaviour {

    // 役割一覧
    public enum Roles {
        PLAYER,
        MONSTER
    }

       private int hpValue;
       public int maxHp, atk;
       public int hp {
              get {
                     return hpValue;
              }
              set {
                     if (value < 0) {
                            hpValue = 0;
                     } else if (value > maxHp) {
                            hpValue = maxHp;
                     } else {
                            hpValue = value;
                     }
              }
       }
       public float speed;
       public bool invincible = false, useStatusWindow = true;
       public GameObject target = null;
    public string charactorName, action;
    public Roles role;
       private GameObject foundTargetEffect;

       public virtual void Start () {
              foundTargetEffect            =            Resources.Load
("Prefabs/Effects/FoundTargetEffect") as GameObject;
```

```
            // useStatusWindow にチェックが入っている場合はStatusWindowを生成
            if ( useStatusWindow ) {
                  GameObject statusWindow = Resources.Load("Prefabs/StatusWindow") as GameObject;
                  GameObject go = Instantiate ( statusWindow );
                  go.GetComponent<StatusWindow>().target = this.gameObject;
            }

            hp = maxHp; // hpの調整
      }

      public virtual void Update () {
      }

      // ダメージを受けるときの関数
      public virtual void Damage (int dmg) {
            hp -= dmg;
      }

      // 自分自身を破棄する関数
      public void CharacterDestroy () {
            Destroy (this.gameObject); // 自分自身を破棄
      }

      // 攻撃をする(攻撃オブジェクトを生成する)関数
      public void CreateAttack ( GameObject prefab, int mgn, int attackCount = 0 ) {
            GameObject go = (GameObject)Instantiate(prefab, transform.position, transform.rotation); // 攻撃の生成
            go.transform.SetParent (transform); // 親の設定
            Attack attack = go.GetComponent<Attack>(); // Attackスクリプトの取得

            // さまざまな値を渡す
            attack.attackCount = attackCount;
            attack.atk = atk;
            attack.mgn = mgn;
```

```
            attack.role = role;
        }

        // ターゲットを探す処理
        public void SearchTarget ( float d ) {
            // 変数宣言
            float targetDistance, minDistance;
            bool flag = false;

            // 初期値のセット
            if (target == null) {
                minDistance = 9999;
            } else {
                minDistance   =   (target.transform.position   -
transform.position).magnitude;
            }

            // オブジェクトの走査
            foreach(GameObject                  go                  in
GameObject.FindGameObjectsWithTag("Character")) { // キャラクタータグ
があるゲームオブジェクト
                Character goC = go.GetComponent<Character> (); //
キャラクタータグの取得
                if ( role != goC.role ) { // 自分と相手の role が
異なるとき（たまにこの行でエラーが発生する…再現性は低め？）
                    targetDistance = (go.transform.position -
transform.position).magnitude; // 距離計算
                    if (targetDistance < d) { // 条件より距離
が近いかどうか
                        if (targetDistance < minDistance )
{ // 距離が今までで最小の場合
                            target = go;
                            minDistance = targetDistance;
                            flag = true;
                        }
                    }
                }
            }

            // ターゲットの取得が行われたなら
```

```
            if (flag) {
                  Instantiate                    (foundTargetEffect,
transform.position, Quaternion.identity); // エフェクト生成
            }
      }

      // ターゲットを解除する処理(解除した場合 true を返す)
      public bool RemoveTarget ( float d ) {
            bool flag = false;
            if             ((target.transform.position             -
transform.position).magnitude > d) {
                  target = null;
                  flag = true;
            }
            return flag;
      }

}
```

１１）CharacterControllerGravity.cs

```
using UnityEngine;
using System.Collections;

public class CharacterControllerGravity : MonoBehaviour {

	private CharacterController cc;
	private Vector3 dir;

	void Start () {
		cc = GetComponent<CharacterController> ();
		dir = Vector3.zero;
	}

	void Update () {

		dir.y += Physics.gravity.y * Time.fixedDeltaTime; // 落下速度を加速

		cc.Move(dir * Time.deltaTime); // 移動

		// 接地時には落下速度を 0 にする
		if ( cc.isGrounded ) {
			dir.y = 0;
		}

	}

	// ジャンプ処理
	public void Jump (float power) {
		dir.y = power;
		cc.Move(dir * Time.deltaTime); // isGrounded 判定されないようにするため、移動する
	}
}
```

１２）CubeTiling.cs

```
using UnityEngine;
using System.Collections;

public class CubeTiling : MonoBehaviour {
    /*
    private float h;
       private Vector3 vec;
       private GameObject up, down, forward, back, left, right;

       void Start () {
              // オブジェクトの取得
              up = transform.FindChild("Up").gameObject;
              down = transform.FindChild("Down").gameObject;
              forward = transform.FindChild("Forward").gameObject;
              back = transform.FindChild("Back").gameObject;
              left = transform.FindChild("Left").gameObject;
              right = transform.FindChild("Right").gameObject;
       }

       void Update () {

       }

       public void Tiling () {
              // 縦幅を調整
              vec = transform.localScale;
              vec.y = h;
              transform.localScale = vec;

              // Y座標を調整
              vec = transform.position;
              vec.y = h / 2;
              transform.position = vec;
       }*/
}
```

１３）DamageText.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class DamageText : MonoBehaviour {

	private float speed;
	private Vector3 dir;
	private Color color;

	void Start () {
		speed = 10.0f;
		dir = Quaternion.Euler (0, 0, Random.Range(-15,16)) * new
Vector3 (0, 1, 0);
	}

	void Update () {
		transform.position += dir * speed;
		speed *= 0.9f;
		color = GetComponent<Text> ().color;
		color.a *= 0.9f;
		GetComponent<Text> ().color = color;
	}
}
```

１４）GameController.cs

```
using UnityEngine;
using System.Collections;
using SocketIO;

public class GameController : MonoBehaviour {

    private GameObject playerPrefabs;
    private SocketIOComponent socket;
       public GameObject player;

    void Start () {
        playerPrefabs = Resources.Load("Prefabs/Player") as GameObject;
        socket                                                          =
GameObject.Find("SocketIO").GetComponent<SocketIOComponent>();
        StartCoroutine("Join");

        // 受信
        socket.On("succeeded in joining", SucceededInJoining);
        socket.On("joined new user", JoinedNewUser);
        socket.On("left user", LeftUser);
    }

    // 参加要求
    private IEnumerator Join() {
        yield return new WaitForSeconds(1);
        socket.Emit("join");
        yield return null;
    }

    // 参加成功
    public void SucceededInJoining (SocketIOEvent e) {
        // 自分の生成
        GameObject go = Instantiate(playerPrefabs);
        go.name = e.data.GetField("id").ToString();
        go.AddComponent<PlayablePlayer>();
        player = go;
        // 他人の生成
        var users = e.data.GetField("users").ToDictionary();
```

```
            foreach ( var key in users.Keys ) {
                go = Instantiate(playerPrefabs);
                go.name = key;
            }
        }

        // 別ユーザ入室
        public void JoinedNewUser(SocketIOEvent e) {
            Debug.Log("a");
            GameObject go = Instantiate(playerPrefabs);
            go.name = e.data.ToString();
        }

        // 別ユーザ退室
        public void LeftUser(SocketIOEvent e) {
            Debug.Log(e.data + "が退室");
        }

        // 同期
        public void OthersTransform (SocketIOEvent e) {
            Transform                              t                              =
GameObject.Find(e.data.GetField("id").ToString()).transform;
            Debug.Log(t.name);
            t.position = new Vector3(
                e.data.GetField("posX").n,
                e.data.GetField("posY").n,
                e.data.GetField("posZ").n
            );
            t.rotation = new Quaternion(
                e.data.GetField("rotW").n,
                e.data.GetField("rotX").n,
                e.data.GetField("rotY").n,
                e.data.GetField("rotZ").n
            );
        }
    }
```

１５）GroundBlock.cs

```
using UnityEngine;
using System.Collections;

public class GroundBlock : MonoBehaviour {

	public float h;
	public Vector3 vec;
	public Material material;

	private GameObject up, down, forward, back, left, right;

	void Start () {
		// オブジェクトの取得
		up = transform.FindChild("Up").gameObject;
		down = transform.FindChild("Down").gameObject;
		forward = transform.FindChild("Forward").gameObject;
		back = transform.FindChild("Back").gameObject;
		left = transform.FindChild("Left").gameObject;
		right = transform.FindChild("Right").gameObject;

		// 縦幅を調整
		vec = transform.localScale;
		vec.y = h;
		transform.localScale = vec;

		// Y座標を調整
		vec = transform.position;
		vec.y = h / 2;
		transform.position = vec;

		// マテリアルの変更
		ChangeMaterial(new GameObject[6] {up, down, forward, back,
left, right}, material);

		// Tilingの調整
		ChangeTilingY(new GameObject[4] {forward, back, left,
right}, h);
	}
```

```
	public void ChangeMaterial (GameObject[] gos, Material m) {
		foreach (GameObject go in gos) {
			go.GetComponent<Renderer> ().material = m;
		}
	}

	public void ChangeTilingY (GameObject[] gos, float f) {
		foreach (GameObject go in gos) {
			go.GetComponent<Renderer>
().material.mainTextureScale = new Vector2(3, 3 * f);
		}
	}
}
```

１６）IchiBird.cs

```
using UnityEngine;
using System.Collections;

public class IchiBird : Monster {

	public float rotateSpeed;

	private int frame;
	private Vector3 vec, dir;
	private Vector2 v2_1, v2_2;

	// コンポーネント
	private CharacterController cc;
	private Animator animator;

	// オブジェクト
	private GameObject attack;
	private ParticleSystem optionCubeParticle;

	public override void Start () {
		base.Start ();

		// コンポーネントの取得
		cc = GetComponent<CharacterController> ();
		animator = GetComponent<Animator> ();

		// オブジェクトの取得
		attack                                                    =
Resources.Load("Prefabs/Attacks/Attack_IchiBird") as GameObject;
		optionCubeParticle         =         transform.FindChild
("OptionCubeParticle").GetComponent<ParticleSystem> ();
	}

	public override void Update () {
		base.Update ();

		// ターゲットが存在しないときは、ターゲットを探す
		if (target == null) {
```

```
            SearchTarget (5.5f);
        }

        // ターゲットが存在するとき
        else {

            // 非攻撃時(着地時)の挙動
            if (action == "") {

                vec = target.transform.position + new Vector3 (0, 0.3f, 0) - transform.position; // ターゲットの方向を算出
                dir = Vector3.zero; // 移動ベクトルの初期化

                // ゆっくりとターゲットの方向を向く
                if (frame < 90) {
                    transform.rotation = Quaternion.Slerp (transform.rotation, Quaternion.LookRotation (vec), rotateSpeed * Time.deltaTime);
                    dir = transform.forward * speed;
                }
                // 何もしない
                else if (frame < 150) {
                }
                // ターゲットと距離が近い場合に攻撃モードに移行させ、そうでない場合はフレームを戻す
                else {
                    if (vec.magnitude <= 4.0f) {
                        dir = Vector3.zero;
                        action = "Attack";
                    }
                    frame = 0;
                }

                // ターゲット解除
                if (RemoveTarget(7.0f)) {
                    frame = 0;
                    dir = Vector3.zero;
                    action = "";
                }
```

```
                }

                // 攻撃時の挙動
                else if (action == "Attack") {

                        vec    =    target.transform.position    -
(transform.position + new Vector3 (0, 0.6f, 0)); // ターゲットの方向を
算出

                        // ゆっくりとターゲットの方向を向く
                        if (frame < 60) {
                                transform.rotation                =
Quaternion.Slerp (transform.rotation, Quaternion.LookRotation (vec),
rotateSpeed * Time.deltaTime);
                        }
                        // 攻撃準備
                        else if (frame == 60) {
                                animator.SetTrigger
("AttackReady");
                                optionCubeParticle.Play ();
                        }
                        // 何もしない
                        else if (frame < 180) {
                        }
                        // 攻撃
                        else if (frame == 180) {
                                CreateAttack (attack, 1); // 攻撃の
生成
                        }
                        // 何もしない
                        else if (frame < 240) {
                        }
                        // フレームを戻し、通常モードに移行する。
                        else {
                                frame = 0;
                                action = "";
                        }

                }
```

```
            }

            cc.Move(dir * Time.deltaTime); // 移動処理
            frame++; // フレームカウント
        }
    }
```

１７）KillMyself.cs

```
using UnityEngine;
using System.Collections;

public class KillMyself : MonoBehaviour {

	// 一定フレーム後に自殺するスクリプト

	public int frame;
	private int i;

	void Start () {
		i = 0;
	}

	void Update () {
		if (i >= frame) {
			Destroy (this.gameObject);
		} else {
			i++;
		}
	}
}
```

１８）MapEditorController.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class MapEditorController : MonoBehaviour {

	public GameObject p, stage, tile, startPosition;
	public Slider zoomSlider, absoluteHeightSlider, relativeHeightSlider;
	public Text absoluteHeightText, relativeHeightText, sizeXText, sizeZText;
	public Dropdown modeDropdown, tileDropdown;
	public GameObject[] modeObjects;

	private int i, j;
	private float h, v;
	private Vector3 vec;
	private GameObject go;
	private MapJson mj;

	public int mapSizeX, mapSizeZ;
	public float speed;
	public Material[] tileMaterials;

	void Start () {
		// コンポーネント読み込み
		mj = GetComponent<MapJson>();

		// タイル配置
		for (i = 0; i < mapSizeX; i++) {
			for (j = 0; j < mapSizeZ; j++) {
				CreateTile (i, j);
			}
		}
	}

	void Update () {
		PlayerMove (); // プレイヤーの移動等
```

```
	}

	void PlayerMove () {

		// プレイヤーの移動
		h = Input.GetAxis("Horizontal"); // 横方向入力の取得
		v = Input.GetAxis("Vertical"); // 縦方向の取得
		p.transform.position += new Vector3(h, 0, v) * speed *
Time.deltaTime; // 移動

		// 移動制限
		if (p.transform.position.x < 0) {
			vec = p.transform.position;
			vec.x = 0;
			p.transform.position = vec;
		}
		if (p.transform.position.x > mapSizeX) {
			vec = p.transform.position;
			vec.x = mapSizeX;
			p.transform.position = vec;
		}
		if (p.transform.position.z < 0) {
			vec = p.transform.position;
			vec.z = 0;
			p.transform.position = vec;
		}
		if (p.transform.position.z > mapSizeZ) {
			vec = p.transform.position;
			vec.z = mapSizeZ;
			p.transform.position = vec;
		}

		// ズーム
		p.transform.position = Vector3.Lerp(
			p.transform.position,
			new Vector3(
				p.transform.position.x,
				(1 - zoomSlider.value) * 30.0f +
zoomSlider.value * 2.0f,
				p.transform.position.z
```

```
                ), 0.5f
            );
        }

        // モード変更
        public void ChangeMode () {

            // 表示するUIの切り替え
            for (i = 0; i < modeObjects.Length; i++) {
                if (i == modeDropdown.value) {
                    modeObjects [i].SetActive (true); // 対象のUIを表示
                } else {
                    modeObjects [i].SetActive (false); // それ以外は非表示
                }
            }

            // 初期位置の表示の切り替え
            if (modeDropdown.value == 3) {
                startPosition.GetComponent<Animator>().SetTrigger ("Show");
            } else {
                startPosition.GetComponent<Animator>().SetTrigger ("Hide");
            }

            // タイル全体の変更
            foreach (GameObject go in GameObject.FindGameObjectsWithTag("Stage")) {
                MapEditorTile goT = go.GetComponent<MapEditorTile>();

                // 0番タイル以外のみ処理
                if (goT.tileNumber != 0) {
                    Renderer goR = go.GetComponent<Renderer>();

                    // タイル変更モード・初期位置変更モード
                    if (modeDropdown.value == 0 || modeDropdown.value == 3) {
```

```
                                goT.HideText(); // 文字を非表示
                                goR.material.color = new Color (1, 1,
1); // 白い(通常)
                        }
                        // 高さ変更モード
                        else if (modeDropdown.value == 1 ||
modeDropdown.value == 2) {
                                goT.ShowText(); // 文字を表示
                                goR.material.color = new Color (0.3f,
0.3f, 0.3f); // 黒い
                        }
                }
        }

    }

    // 高さ(絶対)変更
    public void ChangeAbsoluteHeight () {
        absoluteHeightText.text                                    =
absoluteHeightSlider.value.ToString ("#0.0");
    }

    // 高さ(相対)変更
    public void ChangeRelativeHeight () {
        relativeHeightText.text                                    =
relativeHeightSlider.value.ToString ("#0.0");
    }

    // 横サイズを 1 拡張
    public void AddSizeX () {
        if (mapSizeX < 50) {
            mapSizeX++;
            for (i = 0; i < mapSizeZ; i++) {
                CreateTile (mapSizeX - 1, i);
            }
            sizeXText.text = "x" + mapSizeX;
        }
    }

    // 縦サイズを 1 拡張
```

```
	public void AddSizeZ () {
		if (mapSizeZ < 50) {
			mapSizeZ++;
			for (i = 0; i < mapSizeX; i++) {
				CreateTile (i, mapSizeZ - 1);
			}
			sizeZText.text = "x" + mapSizeZ;
		}
	}

	// タイルを生成する関数
	void CreateTile ( int x, int z ) {
		go = (GameObject)Instantiate (tile);
		go.transform.position = new Vector3 (x, 0, z);
		go.name = x + "," + z;
		go.transform.SetParent (stage.transform);
	}

	// JSONの作成
	public void CreateJson () {
		mj.Reset ();
		foreach (GameObject go in
GameObject.FindGameObjectsWithTag("Stage")) {
			MapEditorTile goT = go.GetComponent<MapEditorTile>
();
			if (goT.tileNumber != 0) {
				mj.AddBlock (go.transform.position.x,
go.transform.position.z, goT.height, goT.tileNumber);
				// 初期位置の設定
				if (startPosition.transform.position.x ==
go.transform.position.x && startPosition.transform.position.z ==
go.transform.position.z) {
					mj.SetStartPosition
(go.transform.position.x, goT.height + 0.4f, go.transform.position.z);
				}
			}
		}
		GUIUtility.systemCopyBuffer = mj.ToJson ();
	}
```

```
}
```

１９）MapEditorTile.cs

```
using UnityEngine;
using System.Collections;

public class MapEditorTile : MonoBehaviour {

	[SerializeField] private GameObject text;

	private MapEditorController mec;
	public int tileNumber;
	public float height;

	void Start () {
		tileNumber = 0;
		SetHeight (10);
		mec = GameObject.FindGameObjectWithTag
("GameController").GetComponent<MapEditorController> ();
	}

	// オブジェクト上でクリックしたときと、クリックしたままオブジェ
クトに入ったときにイベントを実行
	void OnMouseDown () {
		ClickEvent ();
	}
	void OnMouseEnter () {
		if (Input.GetMouseButton(0)) {
			ClickEvent ();
		}
	}

	void ClickEvent () {
		// タイル変更モード
		if (mec.modeDropdown.value == 0) {
			ChangeTileNumber (mec.tileDropdown.value); // タ
イル番号
		}
		// 高さ変更(絶対)モード
		else if (mec.modeDropdown.value == 1) {
			SetHeight (mec.absoluteHeightSlider.value);
```

```
                }
                // 高さ変更(相対)モード
                else if (mec.modeDropdown.value == 2) {
                        AddHeight (mec.relativeHeightSlider.value);
                }
                // 初期位置変更モード
                else if (mec.modeDropdown.value == 3) {
                        mec.startPosition.transform.position = new Vector3
(transform.position.x, 0.4f, transform.position.z);
                }
        }

        public void ChangeTileNumber (int n) {
                tileNumber = n;
                GetComponent<Renderer> ().material = mec.tileMaterials
[n]; // マテリアルの変更
                // 0番に変更された場合はheightも初期化
                if (n == 0) {
                        height = 10;
                }
        }

        public void HideText () {
                text.SetActive (false);
        }

        public void ShowText () {
                text.SetActive (true);
        }

        public void SetHeight (float n) {
                height = n;
                text.GetComponent<TextMesh> ().text = height.ToString
("#0.0");
        }

        public void AddHeight (float n) {
                SetHeight (height + n);
                if (height < 1) {
                        SetHeight (1);
```

```
                }
                if (height > 20) {
                        SetHeight (20);
                }
        }
}
```

２０）MapJson.cs

```
using UnityEngine;
using System.Collections;

public class MapJson : MonoBehaviour {

	// Jsonデータの定義
	[System.Serializable] public class Root {
		public StartPosition start_position;
		public Block[] blocks;
	}

	[System.Serializable] public class StartPosition {
		public float x, y, z;
	}

	[System.Serializable] public class Block {
		public float x, z, height;
		public int material;
	}

	// 変数・オブジェクトの宣言
	private Root root = new Root ();
	private StartPosition start_position = new StartPosition();
	private int i;

	public void Reset () {
		i = 0;
		root.blocks = new Block[3000];
	}

	public void SetStartPosition ( float x, float y, float z ) {
		start_position.x = x;
		start_position.y = y;
		start_position.z = z;
		root.start_position = start_position;
	}

	public void AddBlock ( float x, float z, float height, int
```

```
material ) {
                root.blocks [i] = new Block ();
                root.blocks [i].x = x;
                root.blocks [i].z = z;
                root.blocks [i].height = height;
                root.blocks [i].material = material;
                i++;
        }

        public string ToJson () {
                System.Array.Resize(ref root.blocks, i);
                return JsonUtility.ToJson(root);
        }

        public Root FromJson (string json) {
                return JsonUtility.FromJson<Root>(json);
        }

}
```

２１）Monster.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class Monster : Character {

	private GameObject dieEffect;

	public override void Start () {
		base.Start ();
		dieEffect = Resources.Load ("Prefabs/Effects/DieEffect")
as GameObject;
	}

	public override void Update () {
		base.Update ();
	}

	// ダメージを受ける処理
	public override void Damage (int dmg) {
		base.Damage (dmg);

		if (hp <= 0) {
			Instantiate (dieEffect, transform.position,
Quaternion.Euler (270, 0, 0)); // 死亡時にパーティクル生成
			CharacterDestroy(); // キャラクター破棄
		}
	}
}
```

２２）PlayablePlayer.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.Networking;

public class PlayablePlayer : MonoBehaviour {

	public float cameraSpeed = 720, h, v, speedMgn = 1;

	// コンポーネント
	private Player p;
	private CharacterController cc;

	// オブジェクト
	public GameObject sphere;

	void Start () {
		// コンポーネントの取得
		p = GetComponent<Player>();
		cc = GetComponent<CharacterController>();

		// オブジェクトの取得
		sphere = transform.FindChild("Sphere").gameObject;
	}

	void Update () {

		// 方向入力の取得(通常時、ダメージ時後半)
		if ( p.action == "" || (p.action == "Damage" && p.frame
> p.maxFrame / 3) ) {
			h = Input.GetAxis("Horizontal"); // 横方向
			v = Input.GetAxis("Vertical"); // 縦方向
		}

		// 入力の取得
		if ( p.action == "" ) {
			// 回避入力の取得
			if (Input.GetButtonDown ("Avoid")) {
				// 方向転換
```

```
                transform.rotation *= Quaternion.Euler (
                        0,
                        Mathf.Atan2 (h, v) * Mathf.Rad2Deg,
                        0
                );
                // 入力方向を前方にし、速度 UP
                speedMgn = 4;
                v = 1;
                p.Avoid ();
        }
        // 攻撃入力の取得
        else if (Input.GetButtonDown("Attack")) {
                p.Attack ();
        }
        // 武器変更入力の取得
        else if (Input.GetButtonDown("Weapon1")){
                p.ChangeWeapon (0);
        }
        else if (Input.GetButtonDown("Weapon2")){
                p.ChangeWeapon (1);
        }
        else if (Input.GetButtonDown("Weapon3")){
                p.ChangeWeapon (2);
        }
}

// 回避中の処理
if (p.action == "Avoid") {
        decelerateAngle ();
        // 減速
        if (p.frame <= 20) {
                speedMgn *= 0.95f;
        }
        // 停止
        if (p.frame == 21) {
                speedMgn = 1;
                v = 0;
        }
}
```

```
            // 攻撃中の処理
            else if (p.action == "Attack") {
                  // 動作の減速
                  decelerateMoving ();
                  decelerateAngle ();
            }

            // ダメージ中の処理
            else if (p.action == "ChangeWeapon") {
                  // 動作の減速
                  decelerateMoving ();
                  decelerateAngle ();
            }

            // ダメージ中の処理
            else if (p.action == "Damage") {
                  // 動作の減速
                  decelerateMoving();
                  decelerateAngle();
                  if (p.frame < p.maxFrame / 3) {
                        // 何もしない
                  } else if (p.frame == p.maxFrame / 3) {
                        speedMgn = 3;
                  } else if (p.frame+1 < p.maxFrame) {
                        // 何もしない
                  } else {
                        speedMgn = 1;
                  }
            }

            cc.Move( (transform.rotation * Vector3.forward) * v *
p.speed * speedMgn * Time.deltaTime); // 移動
            transform.Rotate (0, h * cameraSpeed * Time.deltaTime,
0); // 横回転
            sphere.transform.Rotate (v * p.speed * speedMgn, 0, 0);
// 縦回転

      }

      // 移動速度を落としていく関数
```

```
	void decelerateMoving () {
		if (Mathf.Abs (v) < 0.1) {
			v = 0;
		} else {
			v *= 0.9f;
		}
	}

	// 方向回転速度を落としていく関数
	void decelerateAngle () {
		if (Mathf.Abs (h) < 0.1) {
			h = 0;
		} else {
			h *= 0.9f;
		}
	}

}
```

２３）Player.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using SocketIO;

public class Player : Character {

	public int frame, maxFrame, attackCount, weapon;
    private Weapon[] weapons;

	// コンポーネント
	private AudioSource audioSource;
	private Animator animator;

	// オブジェクト
	private GameObject changeWeaponEffect, damageEffect, myDamageText;
	private AvoidEffect avoidEffect;
	private AudioClip audioAvoid, audioChangeWeapon, audioBeforeAttack, audioDamage;
	private GameController gc;
	private Animator displayFilter;
	private PlayerCamera playerCamera;
    private SocketIOComponent socket;

    // 武器クラス
    class Weapon {
        public string weaponName;
        public GameObject prefab;
        public int frame, magnification;

        public Weapon(string weaponName, GameObject prefab, int frame, int magnification) {
            this.weaponName = weaponName;
            this.prefab = prefab;
            this.frame = frame;
            this.magnification = magnification;
        }
```

```
    }

        public override void Start () {
                base.Start ();

         // 名前の変更
         charactorName = name;

                // コンポーネントの取得
                audioSource = GetComponent<AudioSource>();
                animator = GetComponent<Animator> ();

                // オブジェクトの取得
                changeWeaponEffect                                    =
transform.FindChild("ChangeWeaponEffect").gameObject;
                damageEffect                                          =
transform.FindChild("DamageEffect").gameObject;
                avoidEffect             =             transform.FindChild
("AvoidEffect").GetComponent<AvoidEffect> ();
                audioAvoid = Resources.Load ("SE/Avoid") as AudioClip;
                audioChangeWeapon = Resources.Load ("SE/ChangeWeapon") as
AudioClip;
                audioBeforeAttack = Resources.Load ("SE/BeforeAttack") as
AudioClip;
                audioDamage   =   Resources.Load   ("SE/Explosion")   as
AudioClip;
                gc          =          GameObject.FindGameObjectWithTag
("GameController").GetComponent<GameController> ();
                myDamageText = GameObject.Find ("MyDamageText");
                displayFilter             =             GameObject.Find
("DisplayFilter").GetComponent<Animator>();
                playerCamera = Camera.main.GetComponent<PlayerCamera>();
         socket                                                       =
GameObject.Find("SocketIO").GetComponent<SocketIOComponent>();

         // 武器の設定
         weapons = new Weapon[3];
         weapons[0]     =     new     Weapon("    ソ    ー    ド    ",
Resources.Load("Prefabs/Attacks/Sword") as GameObject, 30, 100);
         weapons[1]     =     new     Weapon("    ダ    ガ    ー    ",
```

```
Resources.Load("Prefabs/Attacks/Dagger") as GameObject, 15, 40);
        weapons[2]     =     new     Weapon("    ア    ロ    ー    ",
Resources.Load("Prefabs/Attacks/Arrow") as GameObject, 80, 60);
    }

        public override void Update () {
                base.Update ();

         // socket.io のテスト
         /*JSONObject data = new JSONObject(JSONObject.Type.OBJECT);
         data.AddField("posX", transform.position.x);
         data.AddField("posY", transform.position.y);
         data.AddField("posZ", transform.position.z);
         data.AddField("rotW", transform.rotation.w);
         data.AddField("rotX", transform.rotation.x);
         data.AddField("rotY", transform.rotation.y);
         data.AddField("rotZ", transform.rotation.z);
         socket.Emit("my transform", data);*/

                // 回避中の処理
                if ( action == "Avoid" ) {
                        frame++;
                        // 回避処理終了
                        if (frame == maxFrame) {
                                FinishAction ();
                                invincible = false;
                        }
                }

                // 攻撃中の処理
                else if ( action == "Attack" ) {
                        frame++;
                        if (frame < maxFrame / 2) {
                                // 何もしない
                        } else if (frame == maxFrame / 2) {
                                attackCount++;
                                CreateAttack      (weapons[weapon].prefab,
weapons[weapon].magnification, attackCount); // 攻撃の生成
                        } else if (frame < maxFrame) {
                                // 何もしない
```

```
                } else {
                        FinishAction ();
                }
        }

        // 武器変更中の処理
        else if ( action == "ChangeWeapon" ) {
                frame++;
                if (frame >= maxFrame) {
                        FinishAction ();
                }
        }

        // ダメージ中の処理
        else if (action == "Damage") {
                frame++;
                if (frame < maxFrame / 3) {
                        // 何もしない
                } else if (frame == maxFrame / 3) {
                        animator.SetTrigger
("DamageInvincibleEffect");
                } else if (frame < maxFrame) {
                        // 何もしない
                } else {
                        FinishAction ();
                        invincible = false;
                }
        }

    }

    // アクション変更時の関数
    void ChangeAction(string actionName, int maxFrame){
            action = actionName;
            this.frame = 0;
            this.maxFrame = maxFrame;
    }

    // アクション終了時の関数
    void FinishAction() {
```

```
            action = "";
        }

        // ダメージを受ける処理
        public override void Damage (int dmg) {
            base.Damage (dmg);

            ChangeAction ("Damage", 180);
            invincible = true;
            audioSource.PlayOneShot (audioDamage); // 効果音
            damageEffect.GetComponent<ParticleSystem> ().Play (); //
ダメージパーティクル

            // 操作キャラクターだった場合のエフェクト
            if (gc.player == this.gameObject) {
                displayFilter.SetTrigger ("Damage"); // 画面を赤
色に点滅
                playerCamera.Damage (); // カメラを回転
                myDamageText.GetComponent<Text>().text          =
dmg.ToString(); // ダメージテキストの文字変更
                myDamageText.GetComponent<Animator> ().SetTrigger
("Damage"); // ダメージテキストのアニメーション再生
            }
        }

        // 回避開始時の処理
        public virtual void Avoid () {
            ChangeAction ("Avoid", 30);
            invincible = true;
            audioSource.PlayOneShot(audioAvoid); // 効果音
            animator.SetTrigger ("AvoidEffect"); // アニメーション再
生
            avoidEffect.EffectStart (); // 回避エフェクト
        }

        // 攻撃開始時の処理
        public virtual void Attack () {
            ChangeAction ("Attack", weapons[weapon].frame);
            // 拘束時間が長い攻撃の場合、攻撃前に効果音を鳴らす
            if ( maxFrame >= 60 ) {
```

```
                audioSource.PlayOneShot(audioBeforeAttack);
            }
        }

        // 武器変更開始時の関数
        public virtual void ChangeWeapon (int n) {
            ChangeAction ("ChangeWeapon", 45);
            weapon = n;
            animator.SetTrigger ("ChangeWeaponEffect"); // アニメーション再生
            changeWeaponEffect.GetComponent<ParticleSystem>  ().Play(); // パーティクル再生
            audioSource.PlayOneShot(audioChangeWeapon);
            attackCount = 0;
        }
}
```

２４）PlayerCamera.cs

```
using UnityEngine;
using System.Collections;

public class PlayerCamera : MonoBehaviour {

	private Vector3 pos, vec;
	private Quaternion angle, targetAngle, vAxis;
	private float vRotation;
	private int frame;

	// オブジェクト
	public GameObject target;
	private GameController gc;

    void Start () {
		frame = -1;

		// オブジェクトの取得
		gc = GameObject.FindGameObjectWithTag
("GameController").GetComponent<GameController>();
	}

	void LateUpdate () {

		// ターゲットがあるときの指定
		if (target) {

			// ダメージを受けていたなら、回転させる。
			if (frame != -1) {
				frame++;
				if (frame < 40) {
					vRotation -= 15;
				} else if (frame == 40) {
					vRotation = 10;
				} else if (frame >= 170) {
					frame = -1;
				}
			} else {
```

```
                        vRotation = 5;
                }

                // 通常時の処理
                targetAngle = target.transform.rotation; // ターゲットの向きを取得
                pos = target.transform.position + targetAngle * new Vector3 (0, 0.2f, -1.0f); // 移動先の座標を計算

        }

        // ターゲットがいないときは上のほうへ
        else {
                targetAngle = Quaternion.Euler (90, 0, 0);
                pos = new Vector3 (10, 30, 10);
                vRotation = 0;
                // GameControllerからプレイヤーオブジェクトを取得
                if (gc.player) {
                        target = gc.player;
                }
        }

        // 移動先の角度を計算
        vAxis = Quaternion.AngleAxis(vRotation, Quaternion.Euler(0, targetAngle.eulerAngles.y, 0) * Vector3.right); // 上下の回転を作成
        angle = vAxis * targetAngle; // 作成した上下の回転をターゲットの向きと合成する

        transform.position = Vector3.Lerp (transform.position, pos, 0.1f); // 線形補間
        transform.rotation = Quaternion.Slerp (transform.rotation, angle, 0.1f); // 球面補間

    }

    public void Damage () {
        if (frame == -1) {
                frame = 0;
        }
```

```
        }
    }
```

２５）SaveData.cs

```
using UnityEngine;
using System.Collections;

public class SaveData : MonoBehaviour {

	public static Root saveData; // シーン間で保持されるセーブデータ
	public static string userId, password; // シーン間で保持されるID、パスワード情報

	// Jsonデータの定義
	[System.Serializable] public class Root {
		public string name;
		public string email;
		public int sex;
		public string moodMessage;
		public string created;
		public string logined;
		public int hp;
		public int atk;
		public int level;
		public int skin;
		public int money;
		public int weapon1;
		public int weapon2;
		public int weapon3;
	}

	// セーブデータをセットする関数
	public void SetSaveData ( string json ) {
		saveData = JsonUtility.FromJson<Root>(json);
	}
}
```

２６）SliderColorLerp.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class SliderColorLerp : MonoBehaviour {

	// このスクリプトは Slider > Fill Area > Fill に追加すること。

	public Color minColor, maxColor;
	private Image image;
	private Slider slider;

	void Start () {
		image = GetComponent<Image> ();
		slider = transform.parent.transform.parent.GetComponent<Slider> ();
	}

	void Update () {
		image.color = Color.Lerp (minColor, maxColor, slider.value);
	}
}
```

２７）SliderValueLerp.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class SliderValueLerp : MonoBehaviour {

	// このスクリプトは Slider に追加すること。

	private float newValue;
	private Slider slider;

	void Start () {
		slider = GetComponent<Slider> ();
	}

	void Update () {
		slider.value = 0.9f * slider.value + 0.1f * newValue;
		// 誤差補正
		if (Mathf.Abs(newValue - slider.value) <= 0.001f) {
			slider.value = newValue;
		}
	}

	public void ChangeValue( float n ) {
		newValue = n;
	}
}
```

２８）Stage.cs

```
using UnityEngine;
using System.Collections;

public class Stage: MonoBehaviour {

	private GameObject go;
	private MapJson.Root root;

	[SerializeField] private GameObject groundBlock;
	[SerializeField] private Material[] materials;
	[SerializeField] private TextAsset map;
	[SerializeField] private Transform spawnPosition;

	void Start () {

		root = GetComponent<MapJson> ().FromJson (map.text); //
Jsonの読み込み

		// 初期位置の設定
		spawnPosition.position = new Vector3 (
			root.start_position.x,
			root.start_position.y,
			root.start_position.z
		);

		// フィールドの生成
		foreach (MapJson.Block block in root.blocks) {
			go = (GameObject)Instantiate(groundBlock, new
Vector3(block.x, 0, block.z), Quaternion.identity);
			go.transform.parent = transform;
			go.name = "Block(" + block.x + "," + block.z + ")";
			GroundBlock gb = go.GetComponent<GroundBlock> ();
			gb.material = materials [block.material];
			gb.h = block.height;
		}

	}
}
```

２９）StatusWindow.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class StatusWindow : MonoBehaviour {

	public GameObject target;

	private GameObject canvas;
	private Character targetC;
	private Animator animator;
	private Text swName, swHpText;
	private Slider swHp;
	private SliderValueLerp swHpLerp;

	void Start () {
		// 取得
		canvas = GameObject.Find ("Canvas");
		animator = GetComponent<Animator>();
		swName = transform.FindChild ("Name").GetComponent<Text>
(); // 名前欄の取得
		swHp = transform.FindChild
("HpBar").GetComponent<Slider> (); // HP バーの取得
		swHpLerp = swHp.GetComponent<SliderValueLerp>(); // HP バ
ーの SliderLerp を取得
		swHpText = transform.FindChild
("HpText").GetComponent<Text> (); // HP テキストの取得
		targetC = target.GetComponent<Character>();

		// 設定
		transform.SetParent(canvas.transform);
		swName.text = targetC.charactorName; // 名前欄に名前を反
映
	}

	void Update () {
		// target が存在しなければこのステータスウィンドウを削除
		if (!target) {
```

```
                Destroy (this.gameObject);
                return;
            }

            // ステータスウィンドウの座標を移動
            transform.position                                          =
Camera.main.WorldToScreenPoint(target.transform.position)     +     new
Vector3(0, 65, 0);

            // カメラの位置が近く、カメラがこちらを向いていればステータスウィンドウを表示にする
            if          (          (target.transform.position          -
Camera.main.transform.position ).magnitude < 5.0f
                &&  Vector3.Angle(  Camera.main.transform.forward,
Camera.main.transform.position - target.transform.position ) > 90 ) {
                if ( !animator.GetBool("Enabled")) {
                    animator.SetTrigger("Show");
                    animator.SetBool("Enabled", true);
                }
            // カメラの位置が遠かったり、カメラがこちらを向いていない場合はステータスウィンドウを非表示にする
            } else if ( animator.GetBool("Enabled") ) {
                animator.SetTrigger("Hide");
                animator.SetBool("Enabled", false);
            }

            // HP バーの変更
            swHpLerp.ChangeValue          ((float)targetC.hp          /
targetC.maxHp); // Value の変更
            swHpText.text   =   (Mathf.RoundToInt   (targetC.maxHp   *
swHp.value)) + " / " + targetC.maxHp; // HP テキストの変更
        }
}
```

３０）TitleController.cs

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class TitleController : MonoBehaviour {

    private string url = "http://172.16.134.224:1337/";
      public GameObject main, newGame, loadGame, successedToCreateUser,
successedToLoadUser;
       public      InputField      newGameUserId,      newGamePassword,
newGamePasswordCheck, newGameName, loadGameUserId, loadGamePassword;
       public Dropdown newGameSex;
       public Text newGameError, loadGameError;
       private SaveData saveData;

       void Start () {
              saveData = GetComponent<SaveData> ();
       }

       public void ButtonNewGame () {
              main.SetActive (false);
              newGame.SetActive (true);
       }

       public void ButtonLoadGame () {
              main.SetActive (false);
              loadGame.SetActive (true);

       }

       public void ButtonSuccessedToCreateUser () {
              successedToCreateUser.SetActive (false);
              loadGame.SetActive (true);
              loadGameUserId.text = SaveData.userId;
              loadGamePassword.text = SaveData.password;
       }
```

```
	public void ButtonSuccessedToLoadUser () {
		SceneManager.LoadScene ("Entrance");
	}

	public void ButtonRegister () {
		// ユーザ ID が 4 文字以上あるかどうか
		if (newGameUserId.text.Length < 4) {
			newGameError.text = "【!】ユーザ ID は 4～16 文字で
入力してください";
		}
		// パスワードが 8 文字以上あるかどうか
		else if (newGamePassword.text.Length < 8) {
			newGameError.text = "【!】パスワードは 8～16 文字で
入力してください";
		}
		// パスワードが確認と同じかどうか
		else if (newGamePassword.text !=
newGamePasswordCheck.text) {
			newGameError.text = "【!】パスワードが確認欄と異な
っています";
		}
		// 名前が 2 文字以上あるかどうか
		else if (newGameName.text.Length < 2) {
			newGameError.text = "【!】名前は 2～12 文字で入力し
てください";
		}
		// 登録
		else {
			newGameError.text = "通信中...";
			StartCoroutine ("RegisterUpload");
		}

	}

	IEnumerator RegisterUpload () {
		// フォームの作成
		WWWForm form = new WWWForm ();
		form.AddField ("uid", newGameUserId.text);
		form.AddField ("password", newGamePassword.text);
		form.AddField ("name", newGameName.text);
```

```
            form.AddField ("sex", newGameSex.value);
            // アップロード
            WWW www = new WWW(url + "newGame", form);
            yield return www;
            // エラーチェック
            if (!string.IsNullOrEmpty(www.error)) {
                  newGameError.text = www.error;
            } else {
                  Debug.Log (www.text);
                  if (www.text == "Success.") {
                        SaveData.userId = newGameUserId.text;
                        SaveData.password = newGamePassword.text;
                        // 画面遷移
                        newGame.SetActive (false);
                        successedToCreateUser.SetActive (true);
                  } else if (www.text == "UserID existed.") {
                        newGameError.text = "【!】ユーザ ID『" +
newGameUserId.text + "』は既に存在しています";
                  } else {
                        newGameError.text = "【!】エラーが発生しま
した";
                  }
            }
      }

      public void ButtonLogin () {
            loadGameError.text = "通信中...";
            StartCoroutine ("LoginUpload");
      }

      IEnumerator LoginUpload () {
            // フォームの作成
            WWWForm form = new WWWForm ();
            form.AddField ("uid", loadGameUserId.text);
            form.AddField ("password", loadGamePassword.text);
            // アップロード
            WWW www = new WWW(url + "loadGame", form);
            yield return www;
            // エラーチェック
            if (!string.IsNullOrEmpty (www.error)) {
```

```
                    loadGameError.text = www.error;
                } else {
                    if (www.text == "Error." || www.text == "") {
                        loadGameError.text = "不明なエラーが発生しました。";
                    } else if (www.text == "Failed.") {
                        loadGameError.text = "ユーザ ID かパスワードを間違えています。";
                    } else {
                        SaveData.userId = loadGameUserId.text;
                        SaveData.password = loadGamePassword.text;
                        SaveData.saveData = JsonUtility.FromJson<SaveData.Root>(www.text);
                        successedToLoadUser.SetActive (true);
                        loadGame.SetActive (false);
                GotoStageScene();
            }
              }
        }

    public void GotoStageScene () {
        SceneManager.LoadScene(1);
    }
}
```

３１）Wall.cs

```
using UnityEngine;
using System.Collections;

public class Wall : MonoBehaviour {

	public int n, i;
	public int[] ns;

	void Start () {
		// 2進数に分解
		ns = new int[] {0, 0, 0, 0};
		for ( i = 0; i < 4; i++ ) {
			ns[i] = n % 2;
			n /= 2;
		}

		// 不要な壁を削除
		if (ns [0] == 0) {
			Destroy (transform.FindChild
("Front").gameObject);
		}
		if (ns [1] == 0) {
			Destroy (transform.FindChild
("Right").gameObject);
		}
		if (ns [2] == 0) {
			Destroy (transform.FindChild
("Back").gameObject);
		}
		if (ns [3] == 0) {
			Destroy (transform.FindChild
("Left").gameObject);
		}
	}
}
```

８．２　create_table.sql

```
# 性別テーブル
CREATE TABLE sexes (
       id INT NOT NULL PRIMARY KEY,
       name VARCHAR(255)
) ENGINE = InnoDB;
INSERT INTO sexes (id, name) VALUES (0, '不明');
INSERT INTO sexes (id, name) VALUES (1, '男');
INSERT INTO sexes (id, name) VALUES (2, '女');
INSERT INTO sexes (id, name) VALUES (9, '適用不能');

# ユーザテーブル
CREATE TABLE users (
       id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
       uid VARCHAR(255) UNIQUE,
       name VARCHAR(255),
       password VARCHAR(255),
       sex INT, FOREIGN KEY (sex) REFERENCES sexes(id)
) ENGINE = InnoDB;
```

## ８．３　app.js

```
// モジュールの読み込み
var http = require('http');
var qs = require('querystring');
var io = require('socket.io')({
	transports: ['websocket']
});
var mysql = require('mysql');
var crypto = require('crypto');

// MySQL の接続
var connection = mysql.createConnection({
	database : 'gk', // いけるかな？
	// host : 'localhost', 消しても OK?
	user : 'root',
	password : 'books' // 文字化けしてたら charset
});
connection.connect();
// connection.query('USE gk');

// HTTP サーバの設定
var server = http.createServer();
server.on('request', function(req, res) {

	res.writeHead(200, {'Content-Type': 'text/plain'});

	if ( req.method === 'POST' ) {
		req.setEncoding('utf8');

		// POST データを取得
		var body = '';
		req.on('data', function(data){
			body += data;
		});
		req.on('end',function(){
			POST = qs.parse(body); // POSTデータが格納される。

			// New Game
			if ( req.url === '/newGame' ) {
```

```
                        connection.query('INSERT INTO users set ?',
{
                                uid : POST.uid,
                                name : POST.name,
                                password                        :
encryption(POST.password),
                                sex : POST.sex
                        }, function(err){
                                if(err){
                                        res.end('UserID existed.');
                                } else {
                                        res.end('Success.');
                                }
                        });
                }

                // LoadGame
                else if ( req.url === '/loadGame' ) {
                        connection.query(
                                'SELECT * FROM users WHERE uid = ?
AND password = ?',
                                [POST.uid,
encryption(POST.password)],
                                function(err, results){
                                        if ( !err && results.length
=== 1 ) {

       res.end(JSON.stringify(results[0]));
                                        } else {
                                                res.end('Failed.');
                                        }
                                }
                        );
                }

                // 例外処理
                else {
                        res.end('Error.');
                }
```

```
		});

	// POST 通信でなかった場合はエラー処理
	} else {
		res.end('Error.');
	}

});
server.listen(1337);

// Socket.IO の処理
io.attach(4567);
var users = {}; // ユーザ配列
io.sockets.on('connection', function(socket) {

	// 接続しているソケットのみ
	// socket.emit('emit_from_server', socket.id + ' : ' + data);
	// 接続しているソケット以外全部
	// socket.broadcast.emit('emit_from_server', socket.id + ' : '
+ data);
	// 接続しているソケット全部
	// io.sockets.emit('emit_from_server', socket.id + ' : ' + data);

	// 参加
	socket.on('join', function(data){
		users[socket.id] = true; // ユーザ配列に追加

		// 本人へ参加成功の通知(ユーザ ID、参加者一覧)
		var usersWithoutMe = clone(users);
		delete usersWithoutMe[socket.id];
		var send = {
			id: socket.id,
			users: usersWithoutMe
		};
		socket.emit('succeeded in joining', send);

		// 本人以外の参加者全員へ新規参加者の通知
		socket.broadcast.emit('joined new user', socket.id);

		console.log(users);
```

```
	});

	// 退出
	/*socket.on('disconnect', function(data){
		delete users[socket.id]; // ユーザ配列から削除
		socket.broadcast.emit('left user', socket.id); // ユーザの退出を参加者全員へ通知
	});*/

	// Transformの同期
	socket.on('my transform', function(data){
		data.id = socket.id; // IDデータを付与して、
		socket.broadcast.emit('others transform', data); // 参加者全員へ送信
	});
});


// 暗号化関数
function encryption ( planeText ) {
	var cipher = crypto.createCipher('aes256', 'X!(Ea2BD/Z&8');
	cipher.update(planeText, 'utf8', 'hex');
	var cipheredText = cipher.final('hex');
	return cipheredText;
}

// オブジェクト複製関数
function clone ( src ) {
	var dst = {};
	for ( var k in src ) {
		dst[k] = src[k];
	}
	return dst;
}
```