既存ディジタルゲームへの入力をプログラミングするためのミドルウェアの研究

土井 伸洋[†1]　　栗原 一貴[†2]

本論文では，IoT 機器，Web サービス，AI などの処理を統合して，最終的にゲームコントローラ操作へと変換し既存ディジタルゲームを操作するためのミドルウェア，GameControllerizer を提案する．多様な機器および情報源を既存ゲームへの入力として扱えるようになることにより，新たなエンタテインメントの創出やゲーミフィケーションの構成のための試行錯誤を容易に行うことが可能となる．GameControllerizer は Node-RED により各種入力情報を既存ゲーム入力に変換するプログラミングを簡便に行うビジュアルプログラミング部，およびハード・ソフト両面のエミュレーションにより実際のゲーム機への入力信号を発生させるゲーム入力エミュレーション部からなる．基礎的な性能評価を行うとともに，多様なユースケース事例を提示することでその有用性を示す．

Study on Middleware for Programming Input to Existing Digital Game

NOBUHIRO DOI[†1]　KAZUTAKA KURIHARA[†2]

This study proposes middleware, GameControllerizer, that allows users to combine the processes of Internet of Things (IoT) devices, Web services, and applications of Artificial Intelligence (AI), and to convert them into game control operations to augment existing digital games. The system facilitates easy trial-and-error development of new forms of entertainment and the configuration of gamification by enabling the use of diverse devices and sources of information as inputs to games. GameControllerizer consists of a visual programming element that uses the Node-RED tool to allow users to program easily to convert diverse formats of information into inputs to games, and contains a game input emulation element whereby hardware- and software-based emulation generates inputs for gaming devices. Evidence of the usefulness of the system was provided by a performance assessment and the proposal of a variety of use cases.

1. はじめに

身の回りの様々なモノがインターネットに接続しインテリジェントに振る舞う IoT，そして公開された API によりエンドユーザプログラマが活用できる様々な Web サービス，さらには近年急速に性能が向上し，活用の敷居が下がっている機械学習等の AI 技術によって，我々の生活をより豊かにする情報システムの構築の可能性はこれまでになく広がっている．一般市民が趣味の工作としてそのような情報システムの構築を行うことも Maker 文化の広まりとともに認知度が高まっており，今やハッカソンイベント，成果展示イベント，動画共有サイト等で膨大な事例を観覧することができる．

我々はそのような複合的な情報システムを構築する上で，人々の認知度が高いことなどからよく用いられる題材の一つである既存ディジタルゲームの活用・拡張[1][2][3]をより簡便に行うことができる開発支援技術を研究している．既存ディジタルゲームを拡張することにより，新たなエンタテインメント価値を付与したり，非ゲーム的目的，たとえば社会善の達成を実現することが可能になる．後者は栗原[4]がこれまでに提唱した，ゲーミフィケーションの周辺概念である Toolification of Games を指す．これまで既存ディジタルゲームを再利用した拡張的システム開発は，知的財産権の問題，およびソースコードの入手経路の問題から実現が難しかった．

そこで我々は，「外付けによる既存ディジタルゲーム拡張アーキテクチャ」を提案する．これは既存ディジタルゲームをそのまま内包し，その音声や映像の出力を各種パターン認識技術で検出・認識することでゲームの内部状態を推定するとともに，任意の情報処理を行った上でゲームコントローラ入力へと変換しフィードバックすることにより既存ディジタルゲームを拡張する情報システムアーキテクチャである（図 1）．このアーキテクチャにおいては既存ディジタルゲームは改変されずそのまま活用されるため，先述の問題を解決できる点が特徴である．このようなアーキテクチャはこれまでも個々の開発事例において独自に採用されて来た経緯があるが，我々の貢献はこのようなシステム構成を定式化し要素分解・コンポーネント化し，それぞれ支援技術を開発・提供する点にある．

†1 有限会社来栖川電算
Kurusugawa Computer Inc.
†2 津田塾大学
Tsuda University

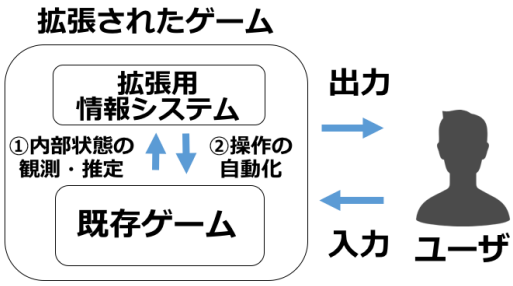

図 1　外付けによる既存ディジタルゲーム拡張アーキテクチャ

Figure 1　External augmentation architecture for digital games.

これまで，図 1 において①ゲームの内部状態の観測・推定については，画像を扱う Sikuli[5]，および電子音を扱う Picognizer[6]などで開発者支援が進んでいる状態である．一方で②操作の自動化，すなわち拡張用情報システムの意図する状態に既存ゲームの内部状態を制御するために，ゲームコントローラ入力へと変換し接続する部分についての開発者支援は少ない．そのような開発を行う場合，毎回アドホックにキーボード・マウスなどのエミュレーションを行うための UI オートメーションライブラリを用いてコーディングを行ったり，ゲームコントローラの電子回路を直接改変して入力信号を出力するなどの手段が取られていたのが現状である．

本論文ではその解決のため，多様な機器および情報源を既存ゲームへの入力へと変換しディジタルゲームを操作するためのミドルウェア，GameControllerizer を提案する．提案手法により，多様な機器および情報源を既存ゲームへの入力として扱えるようになることにより，新たなエンタテインメントの創出やゲーミフィケーションの構成のための試行錯誤を容易に行うことが可能となる．

GameControllerizer は，様々な既存ディジタルゲームプラットフォームへの入力をソフトェア（S/W）またはハードウェア（H/W）によりエミュレートするゲーム入力エミュレーション部，およびエンドユーザプログラマが多様な機器および情報源と通信しながら最終的にゲーム入力エミュレーション部へと操作入力を送信する手順を記述するビジュアルプログラミング部からなる．後者は視覚的に手軽にプログラミングが行える Node-RED[7]を用いて実装されている．ディジタルゲームへの入力信号の時系列情報は，一次元の文字列として記述する可読性の高い簡易言語である DSL4GC（Domain Specific Language for Game Control）で記述される．GameControllerizer のソフトウェアはオープンソースとして公開しており[8]，ハードウェアも配布予定である．

本論文ではまず 2 章で関連研究を俯瞰し，3 章で GameControllerizer の実装について述べる．その後 4 章で入力の遅延等に関する基礎的な性能評価を行い，5 章で多様な活用事例を示すことにより提案手法の有効性を示す．その後 6 章で現状の問題点および今後の展望を議論し 7 章でまとめとする．

なお，提案システムついて我々は主にハッカソンイベント等に参加するようなエンジニアを想定ユーザ層とし，そのようなユーザの心に対し，２つの動きをもたらすことを目指す．一つには，既存ゲームへの入力をプログラミングできることによってゲームのエンタテインメント価値は多様に変容・拡張させることができるという気づきをもたらすことである．これは 5 章およびそれに付随するデモ映像で提示される事例の列挙によって試みる．これにより，既存ゲームの入力の変容・拡張を場当たり的にではなく系統立てて探求することの妥当性と意義が共有されるだろう．

続いてもう一つには，それら事例の基盤技術となっている GameCotrollerizer を用いることで自分も既存ゲームへの入力をプログラミングしたコンテンツを作ることが可能なように感じさせることである．これは 3 章で記述されている，GameControllerizer の実装手段，および公開されている利用者向けドキュメント，そしてコンテンツ制作の様子を記録したデモ映像により達成を試みる．これにより，提案システムの実装手段の妥当性が共有されるだろう．

2. 関連研究

2.1 入力デバイスのエミュレーション

本研究では，各種入力デバイスのエミュレーションにより，あたかもそのような入力が人力によってなされたように振る舞うプログラミング技術を扱う．これは GUI オートメーションの領域において研究が進んでいる．Go 言語のライブラリである Robogo [11]，Mac OS に標準搭載されている Automator[9]等は，テキストプログラミングあるいは例示プログラミング，ヴィジュアルプログラミングの形式で GUI 操作を自動化するプログラミングを可能にしている．Sikuli[5]はアイコン画像などをスクリーンショット機能で撮影し，それを Python のプログラミングコードに直接貼り付けることで「この画像を対象にする」という指示が行える GUI オートメーションプログラミング環境である．Kato らの Picode[10]は，Sikuli を発展させ人間の身体動作やロボットの姿勢等のスナップショットデータを検出対象とする拡張を行っている．

また，ゲームパッドの入力をエミュレーションできるハードウェアとして[12]などが流通しているが，入力に関する柔軟なプログラミングに対応できないという問題がある．

一方，AI 研究の対象としてゲームの自動操作はよく取り上げられる．藤井ら[13]はスーパーマリオブラザーズの自動操作における人間らしさの評価を行っており，Gym Retro[14]は AI 研究のためにゲームのエミュレータを公開し，自動操作のプログラミングを可能にしている．

本研究では，プログラミング環境として Node-RED を採用し，またハードウェア，ソフトウェアを双方活用したゲームコントローラのエミュレーションを採用することで，ネットワーク上で活用可能な様々な情報源をゲームへの入力へと「変換」するハードウェア構築およびプログラミングの支援に特化している点が従来研究と異なる点である．このようにゲームへの入力を拡張する関連研究として，身の回りの物体へのタッチジェスチャーを検出し，キーボード入力とするハードウェアエミュレータである Makey Makey[22]が挙げられる．本研究は Makey Makey も一つの入力デバイスとして扱うことができ，その挙動を単一のキー入力へのマッピングだけでなく，より複雑な挙動を既存ゲームへと入力するためのプログラミングが可能である点に特徴がある．

2.2 既存ゲームの再利用事例

Maker ムーブメントやハッカソンイベントなどにおいて，既存ディジタルゲームの拡張は身近な題材としてよく取り上げられる．例えば既存の遊びを再活用したハッカソンイベントである「遊びの Re ハック」では，テトリスをモチーフにしたプロジェクトが最優秀賞を受賞している[1]．また Life-Size Katamari[2]は「塊魂」用のコントローラを巨大トラックボールとして実装した個人プロジェクトである．TwitchPlaysPokemon[3]は，ゲーム実況中継において不特定多数の視聴者がゲームコントローラへの入力をチャットテキスト入力により行うことでゲームが進行するコンテンツであり，このような「一つのゲームを不特定多数でプレイする」というゲームデザインについての有益な示唆を与え後続研究を刺激した[15][16].

また，ダンスゲームの Dance Dance Revolution[23]などにおいて，システムが検出できない手の振り付けを自主的に行うことで新たなエンタテインメント価値を付与するような試み（栗原の定義による Customized Game[4]）もこれまで盛んに行われてきており，動画共有サイトで閲覧することは容易である．本研究では，このような広まりつつある既存ゲーム再利用のニーズに応えるべく，アドホックに簡便なプログラミングを行うことで様々な情報源を既存ゲームへの入力とするためのミドルウェアを構築する．

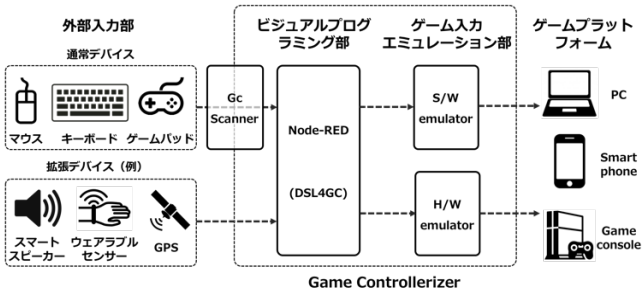

図 2　GameControllerizer の構成図

Figure 2　Structure of GameControllerizer.

3. GameControllerizer

図 2 は，本稿で提案する拡張ゲーム開発支援ミドルウェア， GameControllerizer の構成図である．以降，各部の構成，その使い方について説明する．

3.1 外部入力部

外部入力部は GameControllerizer の外側にあり，ユーザからの入力を担う．外部入力を構成するデバイスはゲーム入力機器として通常想定されている「通常デバイス」と，それ以外の「拡張デバイス」の 2 種類に分けられる．ゲームパッド, キーボード, マウスが前者である．これらは Web ブラウザ上で動作し，ユーザ入力を後述の DSL4GC に変換の後 Websocket で送信するマルチプラットフォームなヘルパーツールである GcScanner によりビジュアルプログラミング部への接続を可能としている．

「拡張デバイス」は，通常ゲーム入力機器として想定していない入力デバイス・情報源である．スマートフォン，スマートスピーカー，ウェアラブルセンサ，GPS，光学カメラ，天気予報 WebAPI など，直接ゲームに関係ないものであってもよい．これらの情報源は HTTP, MQTT 等のプロトコルによりビジュアルプログラミング部に送信され，ゲームへの入力デバイスとして利活用可能となる．

3.2 ビジュアルプログラミング部

ビジュアルプログラミング部は，外部入力部から伝送された各種情報源を，ゲームへの入力として変換するロジックを規定するプログラミング環境であり，Node-RED で実装されている．

Node-RED は, Node-RED サーバ上に送信されてくるあるメッセージを受信・改変・送信するためのフレームワークである．そして各処理に相当するものが GUI 上での処理ブロック（ノード）として定義されており，これらを接続するだけで入力データに対して各種処理を適用できるため，学習のコストが低く，本研究が想定しているようなアドホックなシステム開発に適している．

3.2.1 DSL4GCDSL4GC

GameControllerizer においては，ゲーム制御情報を DSL4GC（Domain Specific Language for Game Control）という抽象形式で記載し，Node-RED でこの制御情報を適切に改変・再送出することで，既存ゲームへの入力のマッピングを実現する．DSL4GC の文法を図 3 に示す．たとえば方向キーおよびアナログスティックをニュートラル位置とし，1 番のボタンを押す動作を 2 フレーム（2/60 秒）にわたって行う，という情報は：

{"dpad":5, "btn":[1], "dur":2, "ang":[0,0,0,0]}

のように JSON で表現されており, 可読性が高い．DSL4GC の詳しい言語仕様は[8]に記載してあるので参照されたい．コンピュータに接続するデバイスの共通な制御メッセージとしてはすでに HID 規格にのっとった

Joystick/Mouse/Keyboard のフォーマット[17]があるが，これらは拡張性に富むものの非常に複雑であり，一般的なゲームを簡便に制御・拡張するためにはオーバースペックであると考えた．そこで我々は，必要十分であり，制御情報を簡便に扱うためのドメイン特化言語を定義した． JSON 形式は，既存の定義との互換性をとりつつ拡張が可能である表現形式であることから，今後応用上の必要に応じて言語を拡張することができる．

```
gc_sentence = Array[gc_word]
gc_word = gc_gamepad_word | gc_mouse_word | gc_keyboard_word
gc_gamepad_word = {“dpad”:Int, “btn”:Array[Int], ”ang“:Array[Int], “dur”:Int}
gc_mouse_word = {“btn”:Array[Int], “mov”:Array[Int], “dur”:Int}
gc_keyboard_word = {“key”:Array[String], “mod”:Array[Int], “dur”:Int}
```

図 1　DSL4GC の文法

Figure 1　Grammar of DSL4GC.

3.2.2 ビジュアルプログラミング部における拡張

GameControllerizer 環境において，ゲームの制御情報を表す DSL4GC は，GcScanner 経由で通常デバイスから生成されるか，もしくは拡張デバイスからの入力をトリガとし，ビジュアルプログラミング部で生成され，Node-RED 上で伝搬されていく．これを簡易に改変するために必要な専用補助ノードを開発した．ボタンや方向キー，アナログスティックにそれぞれ対応した remap-button ノード, remap-dpad ノード，remap-ang ノードである．また，Node-RED 上で任意の DSL4GC を生成する Virtual device ノード（Virtual gamepad，Virtual keyboard，Virtual mouse の 3 種），そして最終的に DSL4GC をゲーム入力エミュレーション部に送出する HwEmulator ノードといった専用補助ノードが実装されている．

3.3 ゲーム入力エミュレーション部

ゲーム入力エミュレーション部は，GameControllerizer の出力であり，既存ゲームを動作させるプラットフォームへの入力となる部分である．DSL4GC を各プラットフォームが規定する制御信号に変換・送信する．GameControllerizer では 2 形態の実装をおこなった．

3.3.1 H/W エミュレータ

前節で述べた通常デバイスの基本的な動作を電気的に模擬する H/W モジュールである（図 4 下）．H/W エミュレータは USB あるいは独自コネクタによるゲームプラットフォームへの入力，および UART による制御入力を備えている（図 5）．すなわち，ゲームパッドやマウス，キーボードを DSL4GC により電子的に制御することを可能にする．実装は， mbed 開発環境[18]に対応した ARM マイコン，および HID デバイスエミュレーションのためのライブラリ[19][20]を用いて行った．

図 4　（左）H/W エミュレータおよび（右）ビジュアルプログラミング部が駆動する RaspberryPi

Figure 4　(Left) H/W emulator and (right) Raspberry Pi Zero W, which runs the visual programming element.

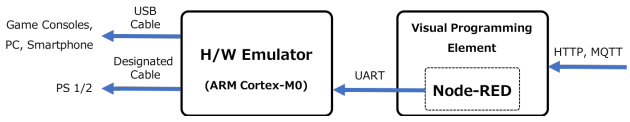

図 5　H/W エミュレータの構成と通信手法

Figure 5　Structure and method of communication of the H/W emulator.

H/W モジュールの主処理を担当している ARM マイコンの Firmware を変更することで，C 上のゲーム，スマートフォンのゲーム，Playstation1/2 などのゲーム入力デバイスとしてふるまう． PC， Playstation3/4, XBOX/XBOX-360, Nintendo switch といった主要な既存ゲームプラットフォームは, 直接もしくは市販の変換器を経由して USB HID 規格のデバイスを接続することが可能であることから，上記 H/W 構成をとることで多くのゲームプラットフォームが制御可能となる．

さらに H/W モジュールについては RaspberryPi と簡易に接続可能な形状として設計されている（図 4）．RaspberryPi 上で上記ビジュアルプログラミング環境を動作させることで， このセット単独で H/W エミュレーション型の GameControllerizer の全機能を実現でき，拡張ゲーム開発者は本セットをネットワークに接続するだけで既存ゲームを拡張することができる．

3.3.2 S/W エミュレータ

S/W エミュレータは既存入力デバイスの基本的な動作を模擬するプログラムである. S/W エミュレータは MQTT を通じて DSL4GC を受け取りこれに応じたデバイスの操作をエミュレートする（図 6）．Go 言語によるデバイスエミュレーションライブラリ Robotgo[11]で実装されており，マルチプラットフォームに対応する． なお現状では，Keyboard/Mouse のみエミュレーションが可能であり，Bluetooth ゲームパッドとして振る舞うデバイスエミュレーション手法は開発中である．

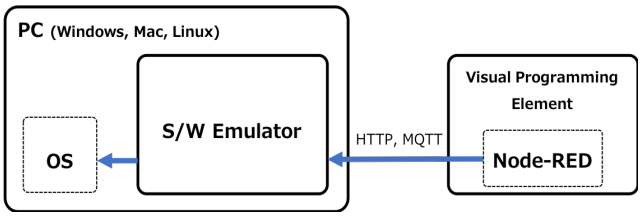

図 6　S/W エミュレータの構成と通信手法

Figure 6　Structure and method of communication of the S/W emulator.

3.4 GameControllerizer の使用例

図 7 は，ある HTTP の GET リクエストを受け，Gamepad のボタンを押す，という制御をおこなうフローである．専用補助ノード（Virtual gamepad ノード）の設定画面は図 8 のようになっており，これらを GUI で設定するだけで，拡張デバイスをゲーム入力に利用することができる．GET リクエストは，PC ブラウザアクセス，Sony MESH 等からのイベント送信などどのような方法で行ってもよい．

あるいは，Virtual gamepad ノードを Node-RED の標準機能である function ノードに置き換え，以下のように記述すると，たとえばストリートファイターII における波動拳コマンド（方向キーを下・右下・右と動かし 1 番ボタン押下）を送出できる．

```
msg.payload = [
    {"dpad":2, "btn":[], "dur":2, "ang":[0,0,0,0]},
    {"dpad":3, "btn":[], "dur":2, "ang":[0,0,0,0]},
    {"dpad":6, "btn":[1], "dur":2, "ang":[0,0,0,0]}
];
return msg;
```

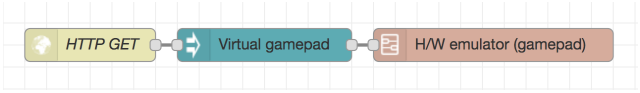

図 7　HTTP GET リクエストを受けて Gamepad のボタンを押すビジュアルプログラミングの例

Figure 7　Visual programming example for pressing a gamepad button after receiving an HTTP GET request.

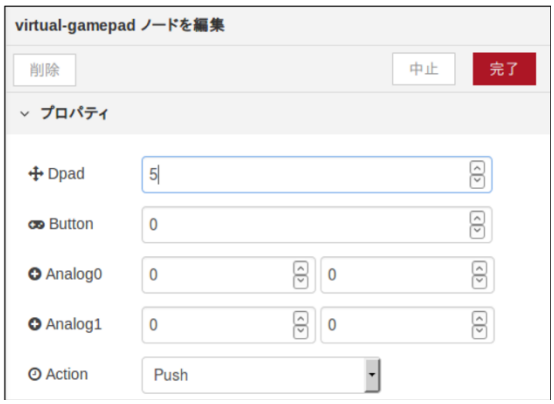

図 8　Virtual gamepad ノードの GUI による設定画面

Figure 8　GUI settings screen for virtual gamepad nodes.

表 1　文献[21]におけるゲームの種類と許容される遅延．

Table 1　Game types and delay thresholds (from [21]).

Group	Model	Perspective	Example Genres	Sensitivity	Delay Thresholds
1	Avator	First Person	FPS, Racing	High	100 ms
2	Avator	Third Person	Sports, RPG	Medium	500 ms
3	Omnipresent	Varies	RTS, Sim	Low	1000 ms

4. 基礎性能評価

本章では，GameControllerizer の基本性能評価の方法・結果について述べる．もし，ミドルウエアにおける通信遅延，処理遅延が大きければ，本来あるべきゲーム体験を損なってしまう．許容される遅延（Delay Thresholds）については，近年広く遊ばれているオンラインゲームに関する研究で詳しく論じられており[21][24][25]，文献[21]では表 1 のように結論付けている．具体的には，遅延に比較的敏感な First Person Shooting (FPS) ゲーム等では 100ms，比較的鈍感な Real-time Stroategy (RTS)ゲーム等では 1000ms である．本論文では，提案したミドルウエアが，“ゲームにおける許容可能な遅延時間” を満たせているかどうか，という観点で計測・評価を行った．なお，本評価において許容される遅延についてはオンラインゲームの値を参照したが，提案手法は基本的にはインターネット，すなわち LAN の外への常時アクセスのないゲームおよび通信経路での活用を想定しているため，インターネットに由来する通信遅延は考えないものとする．そして，遅延時間については，同一経路で 100 回計測したものの平均値を採用した．

4.1 評価方法

評価は単体評価と総合評価の 2 種類を行った．単体評価は提案システム内部の遅延を測定するものであり，総合評価は想定される一般的な提案システムの使用環境である家庭用 wifi 接続使用条件下での総合的な遅延を測定するものである．

4.1.1 単体評価

通単体評価では，入力されてきた単一のボタン入力を表す DSL4GC の MQTT 経由での受領・処理・送出までの時間を計測した（図 9）．外部入力部として 3 種のデバイスを用い，ビジュアルプログラミング部において作成した専用補助ノードで利用可能なものは活用し，ゲーム入力エミュレーション部で H/W および S/W のエミュレーションを組み合わせた合計 6 種類の条件下で行った．結果を表 2 に示す．

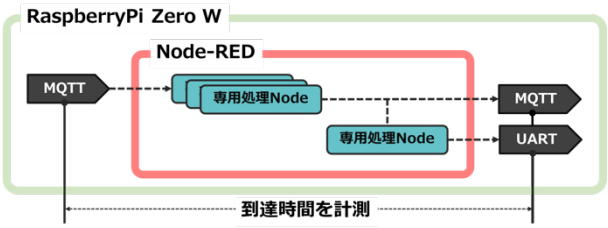

図 9　単体評価の計測環境

Figure 9　Measurement environment for stand-alone assessment.

表 2　単体評価結果

Table 2　Results of stand-alone assessment.

Input	Processed Nodes	Emulation	Average Delay
Gamepad	remap-button remap-ang remap-dpad	S/W	15 ms
Gamepad	remap-button remap-ang remap-dpad H/W Emulator	H/W	16 ms
Mouse	(none)	S/W	9 ms
Mouse	H/W Emulator	H/W	14 ms
Keyboard	(none)	S/W	9 ms
Keyboard	H/W Emulator	H/W	9 ms

4.1.2 総合評価

総合評価では, 単一のマウス入力を GcScanner が 60fps で読み取り, wifi を経由し, H/W エミュレーション部の UART 出力までの遅延計測を行った（図 10）. ビジュアルプログラミング部において作成した専用補助ノードで利用可能なものは全て活用する意図で，function，remap-button，remap-ang，remap-dpad，HwEmulator Gamepad ノードを接続した. 結果は平均遅延 44ms であった.

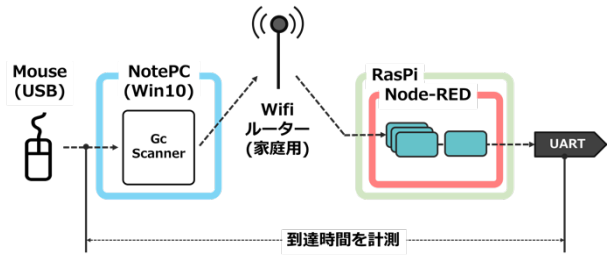

図 10　総合評価の計測環境.

Figure 10　Measurement environment for overall assessment.

4.2 考察

単体評価の結果を観察するに，ミドルウェアとしての Node-RED および専用処理モジュールを利用する際の遅延増加は最大 16ms であった. この値は，表 1 からするに，タイミングクリティカルなゲームにおいてもユーザの体験を著しくは損なわないものと判断できる.

続いて，結合評価の値を見ると遅延は 50ms 程度までに抑えられ, 計測中に遅延が 100ms を超えることもなかった. この環境であれば，表 1 で示される 3 タイプのゲームをいずれもプレイした場合にも許容範囲内と考えらえる.

注意が必要な点として, 結合評価には Wifi 遅延が含まれていることである. 計測してみるとマシン間の ping 値は平均 7ms であった. 計測環境は，障害物および接続台数の面で極めて良好な環境であるためこの程度の遅延で済むが，ユーザが専有できないネットワークを利用した場合や外部の MQTT サーバあるいは API を経由してゲームを制御する場合には，このネットワーク遅延が支配的問題となる. この問題を解決する方法としては，すべての環境（NodeRED, GcScanner）を単独のコンピュータでホスティングすることである.

5. 応用シナリオ例

本章では提案手法の応用シナリオとして実現可能なコンセプト例について述べる. デモ映像[8]には実現している実際の応用事例が収録されているので参照されたい.

5.1 入力機器や方式を拡張する

提案手法により，本来の入力機器として想定されていない機器や情報源を既存ゲームへの入力として扱う拡張が可能である. スマートフォンのジェスチャ，あるいは Bluetooth 接続のスマートフォンシャッターボタンデバイス，Sony MESH，スマートウォッチを装着した状態での身体動作, スマートスピーカーによる音声入力, 為替情報 API による為替の騰落や画像処理に基づく動植物の運動等をゲームへの入力として扱うことができる. また，TwitchPlaysPokemon のように少数あるいは不特定多数のプレイヤーに一つのゲームへの操作を分散させることも可能である.

5.2 Toolification of Games を構成する

前項で述べたように様々な機器や情報源をゲームへの入力として扱うことにより，既存ゲームを用いたゲーミフィケーションの実現方法である Toolification of Games を構成することが可能である. 例えばリズムよく肩たたきをすることをスマートウォッチ等のセンサで検出し音楽ゲームへの入力とすることで，マッサージ効果および対人コミュニケーションの促進を図れるかもしれない. あるいは靴磨きや掃き掃除・拭き掃除の動作を加速度センサで検出し，単純なボタン連打で派手な演出を楽しめるアクションゲームに接続することで作業モチベーション維持に寄与できるかもしれない.

5.3 既存手法と共存する形で入力拡張する

提案手法により，外部入力部にゲームパッドを接続するなどして標準的なプレイ方式を担保しつつ，状況に応じて補助的な入力の介入を行うことが可能である. たとえば入力の難しいコマンドをホットキーあるいは音声入力等により代行することが考えられる. また, 5.1 節で述べたように多様な入力機器を扱う場合でも，それをゲーム内の特定の

シチュエーションに限定しそれ以外の場合は標準的な入力手段を用いることで活用の機会を増大することができる．

5.4 複数のゲームを扱う

提案手法により，一人のプレイヤーの操作入力について，独立動作している複数のゲームへと同時にあるいは選択的に入力することにより，本来一対一の格闘ゲームにおいて一度に多数の敵と戦う，あるいはテトリスとぷよぷよを同時にプレイする，などの新たなエンタテインメントを模索可能である．

5.5 外付けゲーム拡張アーキテクチャによる入出力ループを形成する

ゲームが提示する音声や映像を解析することでゲームの内部状態を推定し，任意の情報処理を行い，提案手法によりゲームへの入力を返すループが形成可能である（図 1）．音声の解析には Picognizer[6]などが，そして映像の解析には Sikuli[5]などが活用可能であろう．これにより，ゲームの自動操作が実現できるだけでなく，5.2 節で述べた Toolification of Games の構成や 5.3 節で述べた補助的な入力の介入について，さらに高い精度で行うことが可能となる．

6. 議論と今後の展望

6.1 遅延対策

評価実験により，遅延についてはある程度の基準をクリアしていることが示されたが，改善の余地がある． 通信プロトコルの改善，あるいはインタフェースデザイン的な改善を検討することなどが考えられる．

前者については，現在 DSL4GC を JSON で記述しているが，データサイズが大きいため，軽量プロトコルである MQTT の活用に問題がある可能性がある．DSL4GC をバイナリエンコードすることによるデータ量圧縮は有意義だが，可読性が下がる点とトレードオフである．

後者については，たとえば身体動作のジェスチャを検出しゲームへの入力とする場合，ユーザに教示するジェスチャを冗長な長さのものにし，実際はそのジェスチャの途中で検出を終了することで体感的な遅延を減少させる工夫などである．これらについては今後の課題である．

6.2 対応ゲーム機およびデバイスの拡充

現状の開発システムでは，ゲーム入力エミュレーション部としてマウス，キーボード，ゲームパッドを扱っているが，対象ゲーム機の違いを完全には吸収できていない．その増強が今後の課題である．また，現在実現している OS レベルの S/W エミュレーション，そして電気信号レベルの H/W エミュレーションに加えて，ロボットデバイスを用いて機械的に既存入力デバイスへの入力を生成するエミュレーション手法[26]の検討も興味深い．

6.3 「チート」の倫理的問題

本研究では既存ゲームへの入力方法を拡張することで，今までになかったゲーム操作体験を試行・実現したり，ゲーミフィケーションへの応用の可能性を探求するための敷居を下げることを目的としている．しかし配慮なく提案システムを用いれば，単に既存ゲームを自分に有利なように自動操作する行為を助長する技術となりかねない．特に近年のオンラインゲームにおいて，あるプレイヤーの行動が他のプレイヤーの行動に影響を与えるような場合には，ゲーム内の倫理・規約に基づいて慎重な活用が求められるだろう．使用時にユーザに適切な使用を促すよう同意を求める，あるいは特定のゲームには適用できなくするなどのシステム的な対応については今後の課題である．

7. おわりに

本論文では IoT 機器，Web サービス，AI などの処理を統合して，最終的にゲームコントローラ操作へと変換しディジタルゲームを操作するためのミドルウェア，GameControllerizer を提案し，基礎的な性能評価をするとともに，様々なユースケース事例を示すことでその有用性を示した．今後は遅延対策，対応ゲーム機およびデバイスの拡充，「チート」対策に取り組むとともに，システムの普及に向けたハッカソン・ワークショップ等を開催する予定である．そのような取り組みを通じて，GameControllerizer を前提としたゲームデザインのあり方，Toolification of Games の構成に向けたノウハウなどを蓄積していく予定である．

謝辞 本研究は中山隼雄財団研究助成，JSPS 科研費 JP15H02735，JP16H02867 の助成を受けた．

参考文献

1) 遊びの Re ハック．
http://mashupaward.jp/2017/10/hackathon-oosaka2017/ (2018/5/31 確認)
2) Life-Size Katamari: http://www.kellbot.com/life-size-katamari-lives/ (2018/5/31 確認)
3) TwitchPlaysPokemon: https://www.twitch.tv/twitchplayspokemon (2018/5/31 確認)
4) K. Kurihara. Toolification of Games. Achieving Non-game Purposes in the Redundant Spaces of Existing Games. *Proc of ACE'15*, pp.31:1-31-5. 2015.
5) Yeh, T., Chang, T. H., Miller, R. C.: Sikuli: using GUI screenshots for search and automation. Proc. of UIST'09, 183–192 (2009).
6) Kurihara, K., Itaya, A., Uemura, A., Kitahara, T., Nagao, K.: Picognizer: A JavaScript library for detecting and recognizing synthesized sounds. Proc. of ACE'17, 339–359 (2017).
7) Node-RED. https://nodered.org/ (2018/5/31 確認)
8) GameControllerizer.
https://github.com/nobu-e753/GameControllerizer (2018/8/7 確認)
9) Automator.https://support.apple.com/ja-jp/HT2488 (2018/5/31 確認)
10) Kato, J., Sakamoto, D., Igarashi, T.: Picode: Inline photos representing posture data in source code. Proc. of CHI'13, 3097–3100 (2013).
11) RobotGo:
https://github.com/go-vgo/robotgo (2018/5/31 確認)

12) PHANTOM-S. http://www.aten.com/global/en/product-landing-page/phantoms/ (2018/5/31 確認)
13) Fujii, N., Sato, Y., Wakama, H., Kazai, K., Katayose, H.: Evaluating human-like behaviors of video-game agents autonomously acquired with biological constraints. Proc. of ACE'13, LNCS 8253, 61–76 (2013).
14) Gym Retro.
https://github.com/openai/retro (2018/5/31 確認)
15) Matsuura, Y., Kodama, S.: Cheer Me!: A video game system using live streaming text messages. Proc. of ACE'17, 311–317, 2017.
16) Sykownik, P., Emmerich, K., Masuch, M.: Exploring patterns of shared control in digital multiplayer games. Proc. of ACE'17, 847–867, 2017.
17) USB HID Usage Tables.
http://www.usb.org/developers/hidpage/Hut1_12v2.pdf (2018/5/31 確認)
18) Mbed: https://www.mbed.com (2018/5/31 確認)
19) Mbed USBHID.
https://os.mbed.com/handbook/USBHID (2018/5/31 確認)
20) Mbed USB Joystick Device.
https://os.mbed.com/users/wim/notebook/usb-joystick-device/ (2018/5/31 確認)
21) Claypool, M., Claypool, K.: Latency and player actions in online games. Communications of the ACM, 49(11), 40–45 (2006).
22) Beginner's Mind Collective and David Shaw. Makey Makey: improvising tangible and nature-based user interfaces. *Proc of TEI'12*, pp.367-370, 2012.
23) Dance Dance Rovolution.
https://en.wikipedia.org/wiki/Dance_Dance_Revolution (2018/6/15 確認)
24) Claypool, Mark, and Kajal Claypool. "Latency can kill: precision and deadline in online games." *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*. ACM, 2010.
25) Jarschel, M., Schlosser, D., Scheuring, S., Hoßfeld, T.: An evaluation of QoE in cloud gaming based on subjective tests. Innovative mobile and internet services in ubiquitous computing (IMIS 2011), 2011 fifth international conference on. IEEE, (2011).
26) Davidoff, S., Villar, N., Taylor, A. S., Izadi, S.: Mechanical hijacking: how robots can accelerate UbiComp deployments. Proc. of ACM UbiComp'11, 267-270 (2011).