情報処理学会論文誌 Vol.61 No.11 1680–1696 (Nov. 2020)

GameControllerizer：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

栗原 一貴[1,a)] 土井 伸洋[2]

受付日 2020年1月30日, 採録日 2020年9月10日

概要：本論文では，多様な機器および情報源からの信号を統合して，最終的にゲームコントローラ操作へと変換することで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．これにより，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，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抽象的なゲーム制御情報を生成する手順を記述できるビジュアルプログラミング部と，専用ハードウェアにより抽象的なゲーム制御情報を実機ゲーム機向けの入力信号へと変換するゲーム入力エミュレーション部からなる．複数のプログラミング環境を準備し，多様なユースケースを提示するとともに，提案システムの公開と販売，ハッカソンやハンズオンイベントの開催，開発者コミュニティを通じた普及活動により，多様なユーザに「作ってみたくなる」心の動きをもたらしたことを分析し，報告する．

キーワード：ゲーミフィケーション, Toolification of Games, 入力エミュレーション, IoT, Maker Movement

GameControllerizer: Middleware for Programming Input to Existing Digital Game

Kazutaka Kurihara[1,a)] Nobuhiro Doi[2]

Received: January 30, 2020, Accepted: September 10, 2020

Abstract: This study proposes middleware, GameControllerizer, that enables the use of diverse devices and sources of information and converts 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. GameControllerizer consists of a visual programming element to allow users to program easily to convert diverse formats of information into inputs to games and a game input emulation element whereby hardware-based emulation generates inputs for gaming devices. We provided users with optional multiple programming environments and demonstrated a variety of use cases. We also conducted a series of community outreach activities such as selling our proposed hardware devices, managing hackathon/hands-on events and an online developer's community. Finally, we analyzed how these strategies resulted in motivating diverse users' minds toward being creative.

Keywords: gamification, toolification of games, input emulation, IoT, maker movement

1. はじめに

Maker 文化の広まりとともに，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あるいは自身がそれぞれ抱えている課題の解決手段として，独自の情報システムを構築することへの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，動画共有サイトなどで膨大な事例を観覧することができる．

このような市民主体の情報システムの構築において，そのエンタテインメント性から，ディジタルゲームがよく題

[1] 津田塾大学
Tsuda University, Kodaira, Tokyo 187–8577, Japan
[2] 有限会社来栖川電算
Kurusugawa Computer Inc., Nagoya, Aichi 460–0007, Japan
[a)] kurihara@tsuda.ac.jp

材として取り上げられている．その際，自身でディジタルゲームをゼロから完成させて用いることは開発の労力，および質の担保の面で難しいため，既存ディジタルゲーム，すなわちすでに公開されているディジタルゲームを用いることもよく行われている．知的財産権に留意しつつ適切に既存ディジタルゲームを活用することで，自身が注力したい要素へと開発労力を集中させることができ，全体として魅力的な情報システムとすることができるからである．これにより，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などが可能になる [1]．

我々はそのような既存ディジタルゲームの活用・拡張 [2], [3], [4] 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．本研究では，そのなかでも既存ディジタルゲームへの入力を改変する技術を扱う．既存の同種技術は，利用にあたって電気的知識や技能が必要とされたり，実現できる拡張も「物理ボタンなどの入力インタフェースと，ゲームパッド上のボタンをマッピングする」などに限られていた [5], [6], [7], [8], [9]．そのため，多様で複雑な事象をプログラミングによって加工し，これをゲームへの入力として扱うことは，その開発負荷の高さから容易ではなかった．

本論文では，多様な機器および情報源から既存ゲームへの入力を生成しディジタルゲームを操作するプログラミングを可能に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．これにより，ゲーム拡張に興味を持つ一般市民や，ゲームを活用し非ゲーム的目的に活用しようとしている開発者や研究者の活動を支援することを狙いとする．

このような開発支援技術は想定されるユーザのスキルや興味が多用である．ユーザの試行錯誤を誘発するために「作ってみたくなる」と思わせるための各種工夫が必要となる．我々はこれについて 2 つのアプローチで推進した．1 つは，利活用の参入障壁を下げることである．これは，電気回路的知識を必ずしも必要としない，Node-RED を用いたビジュアルプログラミング環境を提供できるミドルウェアを開発することにより実現を図った．その有用性については Entertainment Computing 2018 において Qualification [10] を申請し，認定を受けている [11]．付録 A.1 にその際に用いられた，心をどう動かしたいかの宣言およびそのデザインの記述である EDA を掲載する．本論文ではさらに MakeCode を用いたビジュアルプログラミング環境を提供することで，開発環境の選択肢を増やし，より多様なユーザのニーズに応えることを目指した．

もう 1 つのアプローチは，普及活動である．これは，開発物の公開と販売，および既存ディジタルゲーム利活用の楽しさや可能性を示し，人々に作ってみよう，と思わせる交流活動であり，開発物の商品化と販売，およびハッカソン・ハンズオンイベント開催，開発者コミュニティづくりなどにより実現を図った．その結果，ハッカソンにおいて多用な作例が発表されたり，開発者コミュニティ内で興味深い作例が共有されるなど，「作ってみたくなる」とユーザを思わせ，具体的行動にまで誘導できた事例が集積された．

本論文は以下の構成である．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．4 章では多様な活用シナリオを提案する．5 章では提案手法を用いたハッカソンイベントおよび開発者コミュニティ内で共有された活用事例を紹介し分析する．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．

2. 関連研究

2.1 外付けによる既存ディジタルゲーム拡張

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，ソースコードの入手と活用が一般的には難しいため，既存ディジタルゲームをそのまま内包し，拡張用情報システムをゲーム外部に「外付け」するアーキテクチャをとることが多い．これは定式化すれば，**図 1** のような構造をしている．拡張用情報システムの機能として，以下があげられる．既存ゲーム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機能①，既存ゲームの内部状態を拡張用情報システムの意図する状態へと遷移させるためのコントローラ操作信号を生成し送出する機能②，任意の入出力インタフェースによってユーザと通信する機能③④である．ただし，これらの構成要素は必ずしもすべて含まれなくてもよい．

これまで，①については，電子音のマッチングを扱う Picognizer [12]，および想定された使用法ではないものの画像マッチングにより拡張情報システムを構築できる Sikuli [13] などで開発者支援が進んでいる状態である．一方で本研究が扱うのは②の開発，および③をより多様化する探求である．次項以降にこれに関する関連研究について述べる．

2.1.1 入力エミュレーション技術

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 RobotGo [14]，Mac OS に標準搭載されている Automator [15] などは，テキストプログラミングあるいは例示プログラミング，ビ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 [13] 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

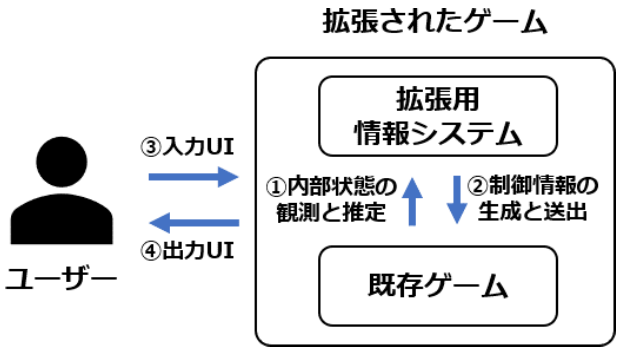

図 1 外付けによる既存ディジタルゲーム拡張
Fig. 1 External augmentation for digital games.

貼り付けることで「この画像を処理対象にする」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 [16] は，Sikuli を発展させ人間の身体動作やロボットの姿勢など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．本研究では，多様なゲームプラットフォームに対応するため，標準的な USB 入出力デバイスとして振る舞うハードウェア層での入力エミュレーションを扱う．

マクロ定義という形で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は PHANTOM-S [17] 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．

2.1.2 入力デバイスの拡張

ゲームへの入力デバイスを拡張する関連研究として，ゲーム愛好家が自分にあったコントローラを制作できる電子工作キット [5], [6]，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へと変換するハードウェアエミュレータである Makey Makey [8]，障害者用マンマシンインタフェースデバイス群を Xbox への入力にマッピングしゲーム入力方法として活用できる Xbox Adaptive Controller [9] などがあげられる．本研究はこれらの機器も 1 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．

特殊な関連技術として，Nintendo Labo [18] について言及する．Nintendo Labo は，インストラクションどおりにダンボールを組み立てて入出力機器を制作し，Nintendo Switch 上の専用ゲームをプレイすること，およびビジュアルプログラミング環境により，実世界とのインタラクションを含むようなコンテンツをユーザが創造することができる．ゲームパッドなどの従来の手法に限定せず，ゲームへの入力機器をユーザがプロトタイピングできるという点で提案手法と類似性がある．一方で提案手法は，操作対象となるゲームについて Nintendo 製のものに限定されず，ユーザが自由に選ぶことができる点が異なる．

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ている．

たとえば，GDC（Game Developers Conference）付随の展示会 [19] はゲームコントローラに焦点を当てた展示会であり，芸術性の高い独自のゲームコントローラなどが発表されている [7]．またゲーム「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 Life-Size Katamari [3] もある．これらの制作にはゲームパッドの電子回路を直接改変して入力信号を出力する手段や先述の電子工作キット [5], [6] などがよく用いられている．

TwitchPlaysPokemon [4] 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パッド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，「1 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 [20], [21]．

また，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，伝統的な子供遊びであるケンケンパの身体的動作を，ゲーム「テトリス」の入力に応用したプロジェクトが最優秀賞を受賞している [2]．

さらに，近年では AI 研究の対象としてゲームの自動操作がよく取り上げられる．藤井ら [22] 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 [23]，Pommerman [24] 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，このような広まりつつある既存ゲームの再利用のニーズに応えるべく，アドホックに簡便な手順で様々な情報源を既存ディジタルゲームへの入力とするためのミドルウェアを構築する．

3. GameControllerizer

図 2 は，本論文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．本章では最初に GameControllerizer の設計指針について述べ，そののち各部の構成と，その使い方について説明する．

3.1 設計指針

図 2 にあるとおり，GameControllerizer 本体は「ビジュアルプログラミング部」「ゲーム入力エミュレーション部」の 2 つからなり，さらにゲーム制御のための素材を提供する「外部入力部」がその外側にある．

入力側から見ると，GameControllerizer は「外部入力信号をビジュアルプログラミング部でもって加工できる環境」と説明できる．一方，ゲームプラットフォーム側から見ると，GameControllerizer はプラットフォーム規定のゲーム

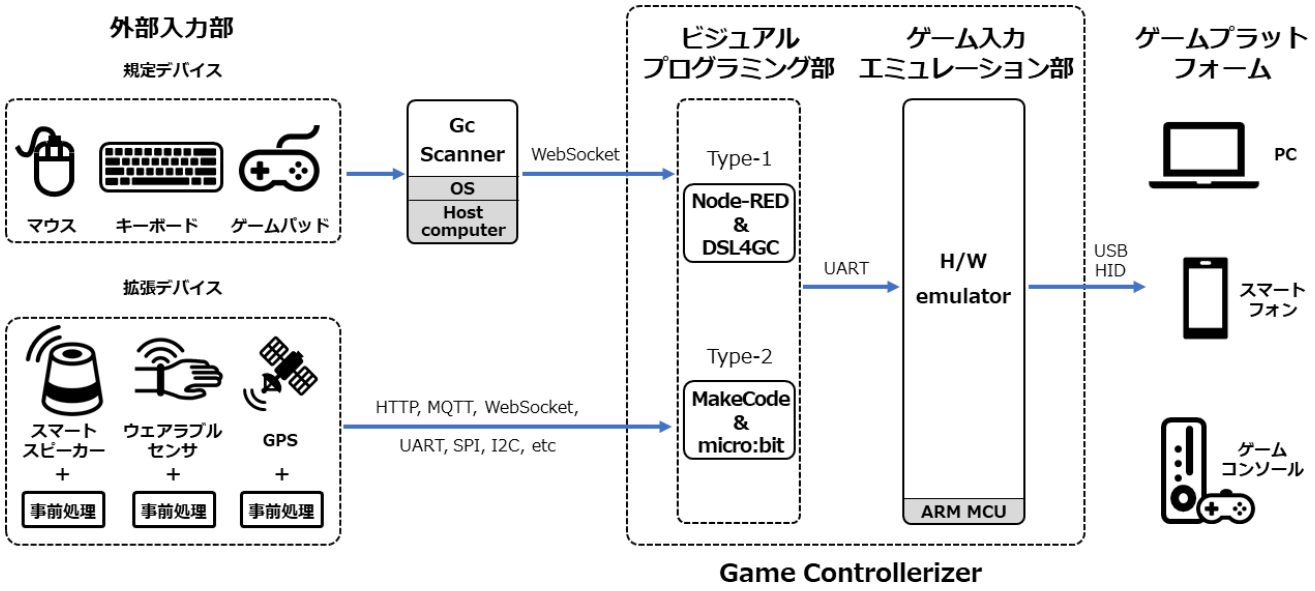

図 2 GameControllerizer の構成図
Fig. 2 Structure of GameControllerizer.

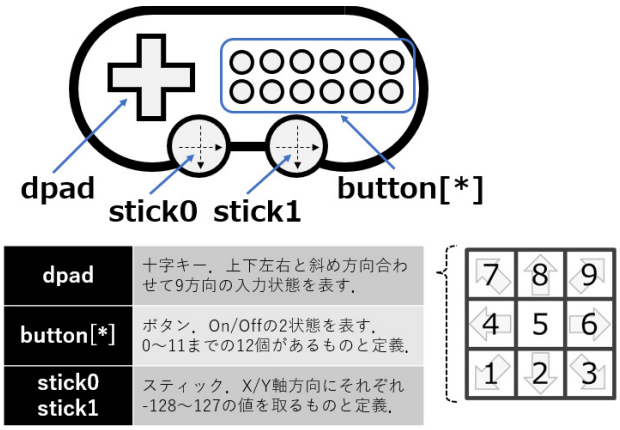

図 3 GameControllerizer が模擬するゲームパッドの構成要素
Fig. 3 Elements of the gamepad to be emulated by GameControllerizer.

コントローラハードウェア（ゲームパッド，マウス，キーボード）を模擬するように構成されている．この構成により，GameControllerizer は「ゲームプラットフォームから見ると規定のゲームコントローラに見える」ものの，「その実ビジュアルプログラミング部を通じて電子的に操作できる」ところが，提案ミドルウェアの設計上のポイントである．これにより，ターゲットとなるゲームプラットフォームを改造することなく入力を拡張できる．

GameControllerizer の模擬するゲームコントローラとして，最も利用ニーズの想定されるゲームパッドを例にとると，模擬対象は十字キー（1 個），ボタン（12 個），スティック（2 個）を持つゲームパッド 1 台である（**図 3**）．この構成は，一般的に使用されているゲームパッドのスーパーセットとなる構成をしており，一部の特殊なゲームを除き制御するに十分な要素を兼ね備えていると考える．

GameControllerizer は，その他のゲームコントローラであるマウスやキーボードについてもサポートしているが，煩雑になるためここでは記載を省く．

以降，様々な制御を行うための素材を提供する「外部入力部」，素材からゲームの制御情報を生成する「ビジュアルプログラミング部」，ゲームの制御情報からゲームプラットフォーム向けの電気信号を生成する「ゲーム入力エミュレーション部」について順に説明する．

3.2 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，ゲームコントローラとして使用が想定されている「規定デバイス」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．

「規定デバイス」からの制御入力は，GameControllerizer の補助ツール「GcScanner」を介してビジュアルプログラミング部へ接続される．GcScanner はクロスプラットフォーム（Win，Mac，Linux）で動作するソフトウェアであり，規定デバイスからのキーやボタンの入力を読み取り，これをゲーム制御の抽象表現である DSL4GC へと変換し，ビジュアルプログラミング部へ送出する役割を果たす．入力読み取りサイクルは 1/60 秒（指定可能）であり，送出時のプロトコルは通信パフォーマンスを考慮し WebSocket とした．また GcScanner はビジュアルプログラミング部が動作している環境と物理的に同じ環境で動作してもよいし，別の環境で動作してもよい．

一方「拡張デバイス」は，通常ゲームコントローラとして想定されていない入力デバイスや情報源である．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．ただし，これらの情報源をゲームへの入力として活用する場合には，個々の情報源に合わせた「事前処理」を行い，拡張デバイスの出力を数値/文字列/音声/画像などビジュアルプログラミング部で取り扱える形式に変換しておく必要がある．たとえば，スマートスピーカーからの入力でゲーム制御を行う場合，「上」「下」という発話を「ウエ」「シタ」という文字列に置き換える，または「0」「1」という数値ラベルに置き換えておくことがこれにあたる．またウェアラブルセンサであれば振動や加速度，GPS であれば一定期間内で移動した距離や緯度経度といった値を取得することもこれと同様である．事前処理は各情報源に付属した SDK やライブラリを使い，GameControllerizer の外側で行われる前提である．

事前処理がなされ，数値/文字列/音声/画像などへと変換された拡張デバイスの出力は，HTTP，MQTT，WebSocket など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，ゲームへの入力を生成するための素材として活用される．なお，送信時，数値/文字列/音声/画像などは，プロトコルに応じて適切な形で送信する．たとえば HTTP であれば，これらデータを URL 自体に埋め込む，QueryString として埋め込む，POST メソッドのボディ部に格納するなどの手段がある．さらに文字列として DSL4GC そのものを送出するような事前処理を行えば，直接これをゲーム制御情報として利用することもできる．

3.3 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てきた各種情報源を素材として，ゲーム制御情報を生成する処理を記述するプログラミング環境である．この部分は，データフロー型の処理記述に適する Node-RED 版と，手続き型の処理記述に適する MakeCode 版の 2 環境を構築した．以降，本章では両環境の構成について述べ，その利用事例分析や考察については 5 章で行う．

3.3.1 Node-RED 版のビジュアルプログラミング環境

データフロー型のビジュアルプログラミング環境は，Node-RED と DSL4GC からなる．構成を**図 4** に示す．

まず Node-RED [25] であるが，これは 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．一般的なブラウザ上で動作するグラフィカルなフレームワークであり，様々な処理に相当するブロック（＝“ノード”と呼ばれる）をドラッグアンドドロップで配置・接続するだけで，入力データに対して様々な処理を適用できる．そのため，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．

さらに，開発者が自身の目的にあわせたカスタムノードを開発・配布することが可能になっており，本研究においてはゲーム制御のための専用ノードを開発した（**表 1**）．各ブロックは外部入力部からの入力信号をトリガとして，ゲーム制御のための DSL4GC を生成したり，DSL4GC をこれと等価なバイトコード列に変換しゲーム入力エミュレーション部へと送り込む働きをする．

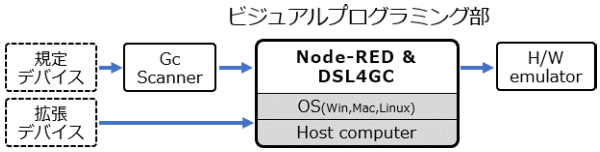

図 4 Node-RED 版のビジュアルプログラミング環境
Fig. 4 Structure of Node-RED based visual programming environment.

表 1 ゲーム制御のためのカスタムノード
Table 1 Node-RED custom nodes for game control.

ブロック名	ブロックの処理内容
dpad	十字キーを制御するためのDSLを生成する．
button	ボタンを制御するためのDSLを生成する． 12個いずれかのボタンのOn/Offを指定する．
stick	スティックを制御するためのDSLを生成する． 2個いずれかのスティックのX/Y両軸の値を指定する．
gamepad	dpad, すべてのbutton, すべてのstickを一度に制御するDSLを生成する．
duration	指定された期間，dpad, button, stick0, stick1の状態を維持するだけのDSLを生成する．指定する期間の単位は1 frame(=1/60 sec)である．
binary-serialize	DSLをこれと等価なH/Wエミュレータ向けバイトコード列へとエンコードする．

次に DSL4GC について説明する．DSL4GC（Domain Specific Language for Game Control）はゲームの制御情報を抽象的に記述するためのドメイン特化言語であり，本環境向けに我々が独自に定義した．

DSL4GC は JSON 形式で表現されており，一般的な通信プロトコルでの送受信はもちろん，特殊なエディタを必要とすることなく記述，複製，改変が可能である．また DSL4GC は，JSON 形式という “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” を利用していることから，今後応用上の必要に応じて言語を拡張することも可能である．

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 USB HID 規格にのっとった Joystick/Mouse/Keyboard のフォーマット [26] がある．だが，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．

DLS4GC の文法概要を**図 5** に示す．いずれの構文も，図 3 中で示した各制御要素（dpad，button，stk0，stk1）をキーとし，目的の遷移状態を値とする．各要素に対する制御を表現するものが gc_gamepad_word であり，これを時

DSL4GC定義全体

```
gc_sentence =     Array[gc_gamepad_word]          |
                  Array[gc_mouse_word]            |
                  Array[gc_keyboard_word]
gc_gamaped_word = {c | c ∈ {“dpad”, “btn”, “stk0”, “stk1”, “dur”}}
gc_mouse_word = {c | c ∈ {“btn”, “mov”, “dur”}}
gc_keyboard_word = {c | c ∈ {“key”, “mod”, “dur”}}
```

gc_gamepad_word

ゲームパッドの制御情報を抽象的に表現する記述.

“dpad”：dpadの制御を定義するキー

```
{“dpad”: 6} // 十字キーを右方向へ入力.
```

“btn”：buttonの制御を定義するキー

```
{“btn”: [0, 1, 2]}      // 0/1/2番のボタンを押し，そのほかは離す.
{“btn”: {“3”: true}}    // 3番のボタンを押し，そのほかは現状維持.
{“btn“: {“4”: false}}   // 4番のボタンを離し，そのほかは現状維持.
```

“stk0”, “stk1”：stick0, stick1の制御を定義するキー

```
{“stk0”: [127, -63]}    // 左スティックの状態を(x,y)=(127,-63)にする.
{“stk1”: {“x”: 31}}     // 右スティックの状態を(x)=(31)にし(y)は現状維持.
```

“dur”：入力状態を維持する期間を定義するキー．単位はframe

```
{“dur”: 2}  // 入力状態を2 frame (=2/60 sec)維持する.
{“dur”: -1} // 次の入力がなされるまで現状態を維持する
```

※gc_mouse_word, gc_keyboard_wordの説明は省略する.

図 5 DSL4GC の文法

Fig. 5 Grammar of DSL4GC.

系列順に並べた配列が gc_sentence である．たとえば，「十字キーの左方向を押す」，「1 番のボタンを押す」，「2 番のボタンを離す」という動作を 2frame（2/60 sec）にわたって行う，という制御情報は，

```
[{“dpad”: 4, “btn”: {“1”: true, “2”: false}, “dur”: 2}]
```

と表現できる．別の例として，ストリートファイター II に代表される格闘ゲームで登場する波動拳コマンド（一定時間内に，十字キーを下・右下・右と動かしボタン押下）のように，時間的連続性を持つ制御情報についても，

```
[{“dpad”: 2, “dur”: 2},
 {“dpad”: 3, “dur”: 2},
 {“dpad”: 6, “dur”: 2},
 {“btn”: {“0”: true}, “dur”: 2}] // 押下するボタンは0番とした
```

と表現できる．DSL4GC の詳しい言語仕様は [27] に記載してあるので参照されたい．

Node-RED 版ビジュアルプログラミング環境を用いてゲーム入力を拡張する様子については，活用事例集映像 [28] を参照されたい．

3.3.2 MakeCode 版のビジュアルプログラミング環境

Entertainment Computing 2018 における Qualification 認定時のビジュアルプログラミング部は 3.3.1 項の Node-RED 版のみであった．Node-RED 版ビジュアルプログラミング環境の特徴として，外部入力部との結合を，DSL4GC を通じた通信へと抽象化し，多様な入力を扱えるようにしたことがあげられる．しかし筆者らがインフォーマルな試用を重ねるなかで，多くのライトユーザはネット経由の通信の手続きなしで，センサ類をより簡便に接続し，カジュアルなビジュアルプログラミングを行える環境を望んでいることが分かった．この観点からは Node-RED 版ビジュアルプログラミング環境には以下のような課題があると考えられる．

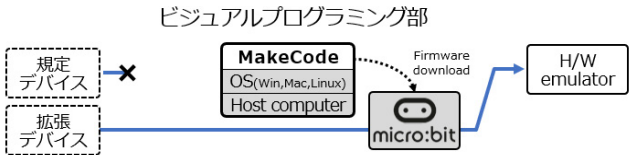

図 6 MakeCode 版のビジュアルプログラミング環境

Fig. 6 Structure of MakeCode based visual programming environment.

(1) 入力拡張において利用の多い，押しボタン型スイッチ，加速度センサなどを Node-RED へ接続する場合，事前処理部が必要になることから，構成が複雑化し，開発者の負荷が増える．

(2) Node-RED の利用には，URL，HTTP リクエストなど通信にかかわる基礎知識が必要とされ，これらに通じていない開発者（たとえば，電気や機械を専門としており，通信の知見が少ないエンジニア）にとっては利用の敷居が高い．

(3) Node-RED はデータフロー型のビジュアルプログラミング環境である．一般的なプログラミングでよく活用される条件付きループなどを記述しようとする場合，記述が煩雑になる．

我々はこれらを Node-RED 版ビジュアルプログラミング環境上で解決することは困難と考えた．そこで，これらを解決する別の選択肢を開発することで，より多様なユーザのスキルおよびニーズに応え，「作ってみたくなる」と心を動かすことに寄与すると考え，新たに手続き型のビジュアルプログラミング環境である MakeCode 版を開発した．

このビジュアルプログラミング環境は，プログラミング環境である MakeCode [29] と，プログラム実行環境である micro:bit [30] からなる．構成を図 6 に示す．

まず，MakeCode であるが，これはブロックプログラミング環境であり，一般的なブラウザ上で動作する．処理ブロックをドラッグアンドドロップで配置するだけでプログラムの構築が可能である．ただし記述しただけでは動作せず，プログラムをコンパイルすることで生成されるバイナリファイルを対応 H/W に流し込むことで動作する．いい換えると，MakeCode は対応 H/W の振舞いを記述する環境ともいえる．Node-RED と同様，カスタムブロックの定義・配布が可能であり，表 1 に示す dpad/button/stick/gamepad/duration に相当するブロックを定義した（図 7）．この環境においては制御情報を DSL4GC で取り扱うことは不要である．各ブロックは H/W エミュレータむけのバイトコード列を直接生成するため，制御情報をどういうデータ形式で表現し送出するか，という点はすべてカスタムブロック内に隠蔽されるか

図 7 MakeCode むけのゲーム制御用処理ブロック（抜粋）
Fig. 7 Custom blocks for game control on MakeCode.

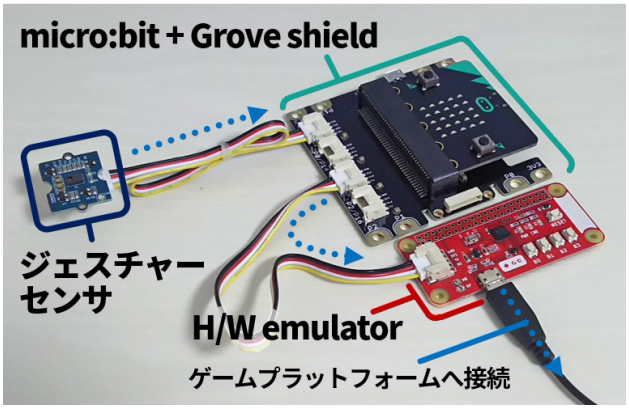

図 8 micro:bit と H/W エミュレータの接続外観
Fig. 8 Connection between micro:bit and H/W emulator.

らである．なお，MakeCode 環境で生成されるバイトコードは，Node-RED 環境で生成されるバイトコードと同一の仕様であり，H/W エミュレータは共通に利用可能である．

次に，プログラムの実行環境であるが，今回は micro:bit を利用した．micro:bit は 6 cm 角の小型マイコンボードであり，STEM 教材として世界的に利用されている．本体にスイッチ，LED，加速度センサなどが備えられているほか，数百種類が販売されている拡張部品についても Grove と呼ばれる共通規格に沿ったコネクタ付きケーブルで接続するだけで利用可能である．

入力拡張部品としてジェスチャーセンサ，出力拡張部品として H/W エミュレータを接続した様子を図 8 に示す．4 色のコネクタ付きケーブルが汎用ケーブルであり，H/W エミュレータが市販のジェスチャーセンサと同様に接続できていることが分かる．

我々が Node-RED 版に加えてこのように MakeCode 版のビジュアルプログラミング環境を構築した根拠を示す．

まず，先述の Node-RED 版の問題点 1 を解決するために，センサなどの接続を簡便に行える micro:bit を用いることが最適であると考えた．すでに大規模に普及が進んでおり，安価な micro:bit を活用することは，提案システムの普及にも貢献すると考えた．micro:bit の標準的なビジュアルプログラミング環境は MakeCode であり，micro:bit の普及とともに MakeCode のプログラミング知識を持つユーザの数も増加している．また，MakeCode においては，独自の機能を持つカスタムブロックをユーザが自由に定義・配布することができる．さらに，micro:bit に接続されるセンサなどは MakeCode 内から簡単にアクセスするためのカスタムブロックライブラリとセットで販売されていることが多く，それを活用することは自然な選択である．したがって micro:bit のプログラミング環境として MakeCode を活用することとし，提案システムのための新しいビジュアルプログラミング環境を開発する必要はないと考えた．

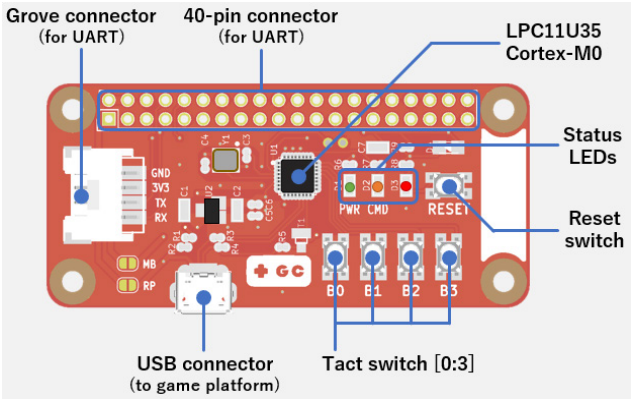

図 9 H/W エミュレータ
Fig. 9 H/W emulator.

micro:bit と MakeCode を組み合わせることにより，センサなどとの情報のやりとりはローカル環境内で完結する．よってネットワーク経由の通信に関する知識が不要になり，先述の問題点 2 は解消すると考えた．さらに，MakeCode は手続き型のビジュアルプログラミング環境であるため，先述の問題点 3 も解消すると考えた．なお，この構成では，外部システムとの通信が難しいという制約が生じるが，これは問題点 2 の解決とトレードオフであるため，許容すべきであると考えた．

MakeCode 版ビジュアルプログラミング環境および micro:bit を用いてゲーム入力を拡張する様子については，同じく活用事例集映像 [28] を参照されたい．

3.4 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．ビジュアルプログラミング部から DSL4GC と等価なバイトコード列を受け取り，各プラットフォームが規定する電気信号に変換し送信する．GameControllerizer では専用基板として実装したため，H/W エミュレータと呼称するものとした．外観を図 9 に示す．

基板上部，左部にはそれぞれ Pin connector，Grove con-

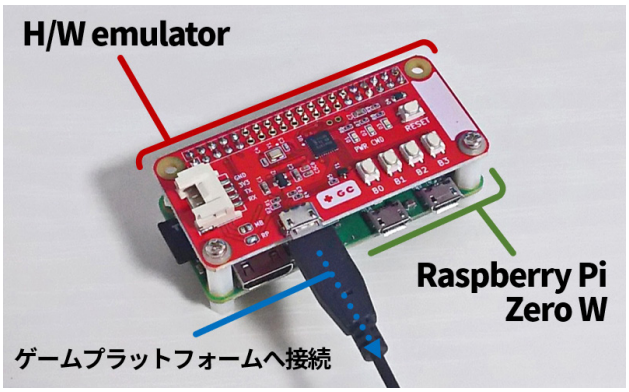

図 10 Raspberry Pi Zero W へ接続された H/W エミュレータ
Fig. 10 H/W emulator mounted on Raspberry Pi Zero W.

nector が存在し，ここが H/W エミュレータの入力部である．これらのコネクタいずれかを通じ，UART 方式で Array[gc_gamepad_word] と等価なバイトコード列を受け取る．

入力されたバイトコード列は基板中央の ARMCortex-M0 マイコンで処理され，USB HID 規格の制御信号へと変換される．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 規格の制御信号を受け付けることが可能であるため，この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．実装は，Mbed 開発環境 [31] に対応した ARM マイコン，および USB HID デバイスエミュレーションのためのライブラリ [32], [33] を用いて行った．

基板下部にある USB ポートは，ゲームプラットフォームへの出力部である．H/W エミュレータとゲームプラットフォームは一般的な micro USB ケーブルで接続できる．

これ以外の特徴として，H/W エミュレータの Pin connector は，Raspberry Pi 40-pin GPIO の仕様に合わせて設計されている [34]．この仕様は，同 Raspberry Pi シリーズにとどまらず，他社の複数の小型 Linux ボードでも採用されており，H/W エミュレータはこれら小型 Linux ボードのいずれにも接続可能である．

また，接続元ホストの小型 Linux ボード上にて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，キット単独で GameControllerizer の全機能を実現できる．接続例を**図 10** に示す．図 10 では接続元ホストとして Raspberry Pi Zero W を用いた例であり，Raspberry Pi Zero W にインストールされた Linux 上で Node-RED 版ビジュアルプログラミング部が動作している．そのため，拡張ゲーム開発者は図 10 のキットを対象のゲームプラットフォームへ接続するだけで既存ゲームを拡張することができる．

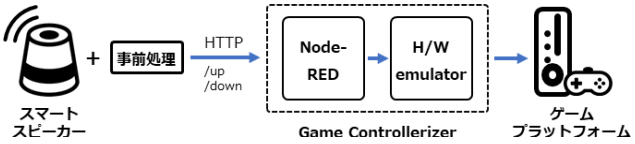

図 11 スマートスピーカーをゲーム入力装置として活用する例
Fig. 11 Use smart speaker as a game controller.

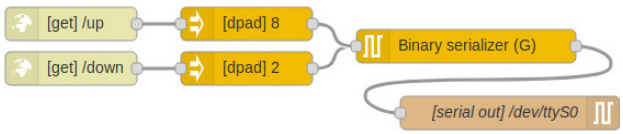

図 12 HTTP GET リクエストを受けて十字キーを制御するビジュアルプログラミングの例
Fig. 12 Visual programming example for pressing a gamepad's dpad after receiving an HTTP GET request.

3.5 GameControllerizer の使用例

3.5.1 Node-RED 版

ビジュアルプログラミング環境として Node-RED を用いた場合の使用例として，ここでは 3.2 節で取り上げた「スマートスピーカーをゲーム入力装置として活用する」例の構成手順を示す（**図 11**）．

具体的には「上」もしくは「下」の発話がなされた際に，それぞれ「十字キーの上（もしくは下）方向を 1 回入力する」ものを目標とする．

最初のステップは外部入力部にあたる部分の構築である．スマートスピーカー付属の開発環境を利用し「上」，「下」の発話がなされた場合に，

- 「上」：http://x.x.x.x/up
- 「下」：http://x.x.x.x/down

のアドレスへの HTTP GET アクセスを発するものを作成する．なお x.x.x.x の部分は GameControllerizer のビジュアルプログラミング部が動いているサーバのアドレスを指し示すものとする．

次に Node-RED 環境上で，2 種の URL に応じて対応する DSL4GC を生成する部分を構築する．処理フローは**図 12** のようになる．まず URL を受け取る部分は Node-RED に初期状態で準備されている “http in” ノードで定義し，それぞれの URL に対して十字キー制御のための DSL4GC を生成する “dpad” カスタムノードを配置・接続する．“dpad” カスタムノードの設定画面は**図 13** のようになっている．項目 [DPAD] では「入力方向」，[input mode] では「入力方法」を指定する．後者には瞬間的な押下である Push ({“dur”: 3}) と永続的な押下である Hold ({“dur”: -1}) の 2 選択肢を用意してある．“http in” と “dpad” の 2 つのノードの連続処理により HTTP GET アクセスをトリガとして，{“dpad”: 8, “dur”: 3} といった DSL4GC が生成される．

最後に生成された DSL4GC を H/W エミュレータへと送

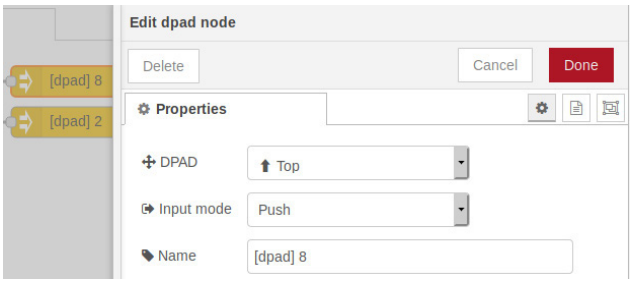

図 13 dpad ノードの設定画面
Fig. 13 GUI settings screen for “dpad” node.

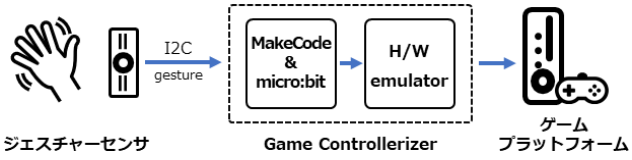

図 14 ジェスチャーセンサをゲーム入力装置として活用する例
Fig. 14 Usage example of using a gesture sensor as a game controller.

出する．これには “binary-serializer” カスタムノード，および Node-RED 上で UART 通信を取り扱うための “serial out” ノードを配置・接続する（図 12）．“dpad” カスタムノードの出力である DSL4GC はこれら 2 つのブロックを通ることで，H/W エミュレータむけのバイトコード列へと変換・送出される．そしてこのバイトコード列を受け取った H/W エミュレータは，指定された制御を行うための電気信号を生成し，ゲームプラットへと送出する．

なお，1 つの HTTP GET リクエストをトリガとして複数のキー・ボタン入力をともなう先述の波動拳のような複雑なコマンドを生成したい場合は，“dpad” カスタムノード 3 個と “button” カスタムノード 1 個を直列に連結することで表現できるほか，Node-RED の “function” ノード内で，3.3.1 項内で示した JSON 表現を直接的に記述し，このノードの出力を “binary-serializer” カスタムノードへと渡すことでも実現できる．

3.6 MakeCode 版

次に，ビジュアルプログラミング環境として MakeCode を用いた場合の使用例として「ハンドジェスチャーセンサをゲーム入力装置として活用する」例の構成手順を示す（**図 14**）．具体的にはジェスチャーセンサの前で手を「右」もしくは「左」へ移動させた際に右（もしくは左）へ十字キーを入力しつづけ，手を「振る」とボタン 0 番を 3 回入力するようなものとした．

まず機器構成であるがこれは図 8 で示したものに等しい．次に MakeCode 環境上で，ジェスチャーセンサのからの入力をトリガとして，所望のゲームパッド操作を実現する部分を構築する．処理フローは**図 15** のようになる．図中にはジェスチャーセンサからの入力を受け取る処理ブロック（紫）の子要素として，図 7 で示したゲーム制御用処理ブロック（黄）が配置され，外部入力に応じて適切な H/W エミュレータ向けバイトコード列を直接生成し送出する．

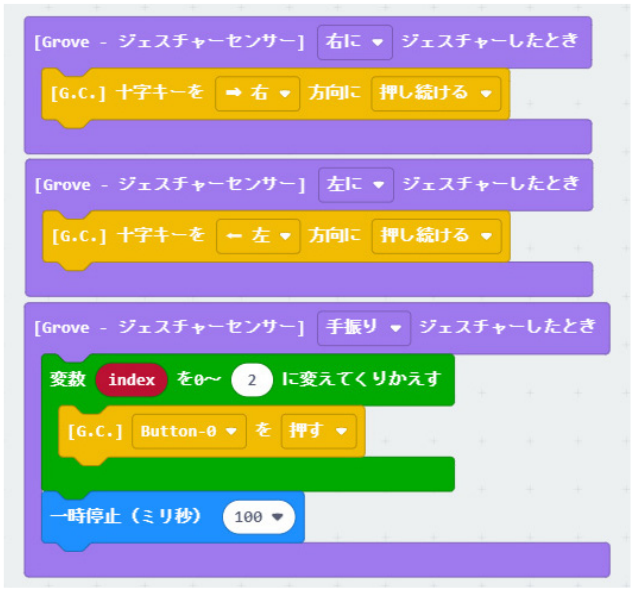

図 15 ジェスチャーセンサからの入力を受けて十字キーとボタンを制御するビジュアルプログラミングの例
Fig. 15 Visual programming example for pressing gamepad’s buttons after receiving a trigger signal from a gesture sensor.

すべての制御用処理ブロックは平易な自然文で記述されており，また文法的に正しい構造でしか配置できないようになっているため，初学者でも間違いを起こしにくいといった利点がある．

なお，この例おいては，すべての入出力は micro:bit と MakeCode 環境内で閉じているため，DSL4GC を用いずともフローの実現が可能である点に留意されたい．また，もし 3.5.1 項で示すようなネットワークを通じての外部入力や DSL4GC を取り扱いたい場合は，ネットワーク通信を行う Grove ユニットや「DSL4GC を解析」するカスタムブロックを，入手もしくは独自開発することで実現可能となるだろう．

4. 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．活用事例集映像 [28] 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．

4.1 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．まず，スマートフォン搭載の加速度・ジャイロセンサに基づく筐体の傾き情報およびシェイクなどのジェス

チャ検出，Bluetooth 接続のスマートフォン用シャッターボタンデバイス，Sony MESH のボタンタグあるいは動きタグなど，単純なセンサ情報を直接的にゲームへの入力として扱うことができる．さらに，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，あるいは画像処理に基づく動植物の運動など，センサ信号列を機械学習などの AI 技術でパターン検出したものもゲームへの入力として扱うことができる．さらに，為替情報 API による為替の騰落など，非生物的な情報をゲームへの入力として扱うことも可能である．

このように入力機器や方式を拡張できることは，ゲームの操作主体をより柔軟にデザインできることを意味する．ゲーム操作負荷を複数のプレイヤに分配する TwitchPlaysPokemon [4] のような応用や，操作の一部を AI・動植物・非生物に肩代わりさせたりといったデザインも可能になる．

4.2 Toolification of Games を構成する

前節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．たと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など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．また一定のリズムが重要とされる行為，たとえば心肺蘇生などのタイミングを習得するのに利用可能かもしれない．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．

また提案システムを用いた，特定のゲームに依存しない Toolification of Games の事例として，初等プログラミング教育への応用が考えられる．プログラミングを学習する動機づけとして，「ゲームを作る」という課題は魅力的であるが，実際に初学者が制作するゲームは商用のゲームに対して低品質になりがちであり，高い達成感を感じられていないと我々は予想している．GameControllerizer を用いれば，初学者に対し「あなたの好きな既存ゲームに対して自分だけのコントローラを作り，その挙動をプログラミングしてみよう」あるいは「あなたの好きな既存ゲームを自動操作する AI をプログラミングしてみて，友達と戦おう」というような課題を課すことができる．ゲームの入力をプログラミングすることは，ゲーム全体をプログラミングすることに比べて容易である一方で，変数，繰り返しや条件分岐，状態遷移など，基礎的なプログラミングの概念を内包できる複雑性を維持しているため，入門教育として優れていると期待している．

4.3 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，たとえば格闘ゲームにおいて入力の難しい必殺技コマンドをボタン型デバイスあるいは音声入力などにより代行するような，補助的な入力の介入を拡張することが可能である．

これは 4.1 節で述べたような必ずしも入力自由度の高くない多様な入力機器を用いてユーザが既存ゲームへの入力方法を拡張する際，実用上有効な既存ゲーム入力拡張方法である．なぜならばそのゲームの操作に必要な入力自由度を完全に網羅する方法をユーザが新たに準備しなくてよいからである．

4.4 複数のゲームを扱う

提案手法により，1 人のプレイヤ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 1 対 1 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，などが実現可能である．これは一見荒唐無稽なように思えるが，既存ゲームに対し自らプレイに制約を課して高難易度のゲームプレイを披露するパフォーマンスコンテンツの 1 つの形として，活用可能であると考える．たとえば，ダンスゲームの Dance Dance Revolution [35] 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 [1] がさかんに行われていることから，このような行為の意義は正当化されるであろう．

4.5 外付けゲーム拡張において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 [12] などが，そして映像の解析には Sikuli [13] などが活用可能であろう．応用例として，ロールプレイングゲームの単純操作を GameControllerizer と Picognizer を併用することで自動化する例が考えられる．より高度な解析技術を用いれば，4.2 節で述べた Toolification of Games の構成や 4.3 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．

5. 普及活動および活用事例の分析

一般市民が趣味の工作あるいは自身がそれぞれ抱えている課題の解決手段として既存ディジタルゲームを拡張した情報システムの開発を行うことを支援する本研究においては，想定されるユーザのスキルや興味が多用であり，ユーザの試行錯誤を誘発するために「作ってみたくなる」と思わせるための多角的なアプローチが求められる．我々はここまで記述してきた，利活用の参加障壁を下げるようなミ

ドルウェアの開発だけではなく，開発物の普及活動，および既存ディジタルゲーム利活用の楽しさや可能性を示し，人々に作ってみよう，と思わせる交流活動についても推進した．具体的には，本研究の提案ミドルウェアはすでにソフトウェア部についてはオープンソースとして公開しており，またH/Wエミュレータについては販売を行っている．さらにハッカソン・ハンズオンイベントを開催し，開発者コミュニティづくりを行っている．本章では，ハッカソンイベントおよび開発者コミュニティ内で共有された作例について報告し，分析を行う．

5.1 ハッカソン開催と事例共有

既存ディジタルゲームの入力を拡張することで新たなエンタテイメントを模索することをテーマとして2019年3月にハッカソンイベントを開催した．募集はIT関連イベント告知サイトで行い，当日は20名が参加した[*1]．参加者の年代は10～50代であり，ボリュームゾーンは20～40代であった．また性別は男性17名，女性3名であった．全員が仕事もしくは趣味レベルでのプログラミング経験を持ち，うち4名はHCI/EC関連の研究を行う大学・研究機関に所属していた．

参加者は1～3名の計14チームに分かれ，ゲーム拡張に取り組んだ．参加者に対しては，ゲームプラットフォーム（Playstation 2，Playstation 3，Nintendo Switch，ファミリーコンピュータ互換機，PCのうちいずれかを選択），ゲームソフト，各種センサ，加工可能な素材と工具（段ボール，模造紙，ハサミ，テープ，カラーペンなど）を提供し，また自主的な持ち込みも可とした．開発支援技術としてGameControllerizerのレクチャー時間を1時間とり，実際の開発時間は8時間というスケジュールで執り行った．参加者へ提示した開発目標は「自分が面白いと思うゲーム拡張を実現する」だけとし，これ以外の制約は設けなかった．また，参加者に対し，GameControllerizerの使用は強制しなかった．

5.1.1 アンケートの実施

イベント終了後，参加者にアンケートを行い，有効回答として16人分を得た．質問したのは，利用環境は何か（Node-RED版使用・MakeCode版使用・GameControllerizer不使用），ミドルウェアの使用感（使いやすい・普通・使いにくい），楽しかった・面白かったこと（自由記述），苦労した・実現できなかった項目（自由記述）の4項目である．

5.1.2 結果

ハッカソンイベントの結果としては，14チームすべてが何らかの形で既存ゲームを拡張する作品を完成させることができた．ほとんどのチームは1人であったが，複数人のチームおいては，「センサを主とした入力部と，GameControllerizerを主とした制御部」（エンジニア2人のチーム），「入力部，制御部，段ボールによる外装」（エンジニア2人とデザイナ1人のチーム）といった分業体制が見られた．

GameControllerizerの使用状況については，14チーム中13チームが使用したのに対し，不使用のチームが1チームあった．後者については次のような状況であった．当該チームの参加者のプログラミングスキルが高く，キーエミュレーションにかかわる知識をすでに持っていた．そのため，ターゲットプラットフォームとしたPC上のゲームを，自作のプログラムとキーエミュレーションライブラリで制御していた．当日初めて利用するGameControllerizerより，扱い慣れておりプログラミング次第でトラブルにも柔軟な対応が可能な手段をとり，作品の完成を優先した背景があった．ビジュアルプログラミングは初学者にとっては敷居の低い選択肢となりうるが，熟練者にとっては柔軟性が損なわれる側面がありうる，ということが示唆された．

イベント終了後のアンケート項目のうち「利用環境」「ミドルウェアの使用感」の質問項目に対する答えは**図16**となった．

まず「利用環境」については，前記の利用しなかった1名を除くと，5名がNode-RED版，10名がMakeCode版であった．MakeCode版の利用が多かったのは，参加者の多くが多様なセンサデバイスを活用したゲーム拡張を志向したからであると考えられる．3.3.2項であげたように，MakeCode版のビジュアルプログラミング環境から利用するmicro:bitは，micro:bit本体に搭載されたセンサもしくは拡張用センサを接続し若干のブロックプログラミングを行うだけで，センシングを実施し，これをトリガ情報として取り込むことが可能である．この一連の作業の負荷の低さがユーザに歓迎されたと考える．

次に「ミドルウェアの使用感」については，Node-RED版，MakeCode版ともに，6割の利用者が使いやすいと答えており，使いにくいと回答したユーザは0人であった．全利用者が当日初めて提案ミドルウェアを使う状況であるにもかかわらず，全員が何らかの形で利用可能なものだと感じたと受け取ることができる．

次に，自由記述欄について「楽しかった」項目として最も多かったのが「既存ゲーム機にセンサ類を簡便につなげられたこと（4名）」であり，提案ミドルウェアの有効性を感じていた参加者がいたことが裏付けられた．

一方，「苦労した」項目としては，「キーマップの合わせこみが大変（2名）」「会場LANが貧弱で遅延が大きい（2名）」「センサデバイスの扱いに苦労した（2名）」「ビジュアルプログラミング環境の操作で不明な部分があった（2名）」があげられた．このうち「キーマップの合わせこみ」についてはミドルウェア自体の問題であり，改良が必要であることが示唆された．

[*1] https://mashupawards.connpass.com/event/120523/

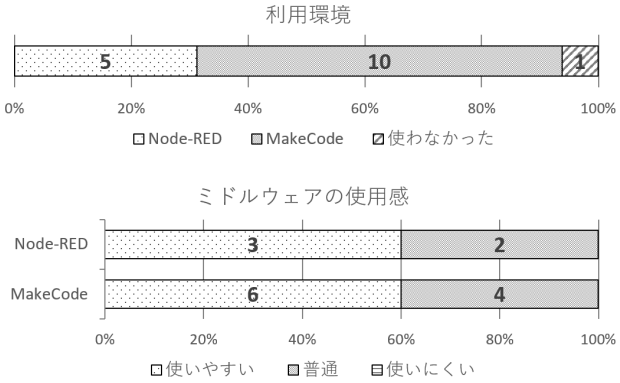

図 16 アンケート結果
Fig. 16 Summary of questionnaire.

図 17 ゲームキャラクターの腕を模したコントローラの作例
Fig. 17 Participant's use case: an arm-like controller for a boxing game.

5.1.3 ハッカソンでの開発事例

ここではハッカソン参加者により開発された事例をいくつか示す．

作品のうち最も多かったのが 4.1 節で述べた「入力機器や方法を拡張する」コンセプトに相当するものである．たとえば，加速度センサを用いてパンチやジャンプのモーションをセンシングしゲームの入力としたり，ゲーム入力とは無縁な「飲料を飲む」「お菓子を食べる」という動作を Sony MESH で検出，入力トリガとして活用した作品もあった．

Life-Size Katamari [3] で見られたように，ゲーム内のオブジェクトを現実世界の物体として表現する，といった拡張を行った作品もあった．開発者は，3 人称視点で展開されるボクシングゲーム「ARMS」の入力として “ゲーム中に登場するキャラクタの腕を模したコントローラ” を段ボールで制作していた（**図 17**）．コントローラの動きとゲームキャラクタのアクションとが一体化しているように感じられ，体験者からも評価が高かった．

また，Twitter への発言をゲーム入力とする拡張例が見られた．これは特定のハッシュタグを付与して「A（を押せ）」と発言すると，GameControllerizer を通じてこの操作がなされ，さらに操作結果画面を撮影して，Twitter の

図 18 腹筋動作をリズムゲームの入力とする作例
Fig. 18 Participant's use case: a headband controller for a music game.

返信として送り返す，というものである．これは TwitchPlaysPokemon [4] のように不特定多数のプレイヤに 1 つのゲームへの操作を分散させる試みの 1 つと見なせる．他には，4.2 節「Toolification of Games を構成する」コンセプトに相当する作品として，“腹筋運動中の頭の動作を，リズムゲームの入力に利用する” という拡張が見られた．開発者は micro:bit を用いてヘッドバンド型のコントローラを制作し，上半身を起こす/寝かすの 2 アクションを検出して入力としていた（**図 18**）．トレーニングのモチベーションアップのため既存ゲームを活用する，という Toolification of Games の事例であると考える．

さらに，手動操作と自動操作の 2 種の入力モードを持ったコントローラを制作した例もあった．開発者によれば「プレイに疲れてきて，少し休みたいとき」に，手動モードから自動モードへと切り替えると，コントローラ自体が簡易な制御ルーチンにより自キャラクタを自動で動かしゲームを継続してくれるという仕組みである．これは 4.3 節で述べた，「既存手法と共存する形で入力拡張する」コンセプトに相当するものである．また自動モードは MakeCode 上で条件付きループを記述することで実現されていた．これは MakeCode 版ビジュアルプログラミング環境が活用され，3.3.2 項で述べた Node-RED 版の問題点が解決できたことを示す作例と考えられる．

また，4.3 節「既存手法と共存する形で入力拡張する」と 4.1 節で述べた「ゲーム操作負荷を複数のプレイヤに分配する」コンセプトを同時に実現した作品として「2 人で操作するボンバーマン」を構築した作品もあった．ただし開発者は，筆者らが 4.3 節で述べたような “入力補助” ではなく，逆に “邪魔する” という着眼点で拡張を行っていた．すなわち，メインのプレイヤが（通常の操作手法である）キーボードで操作する傍らで，サブプレイヤがスマートフォン上のインタフェースを操作することで，メインプレイヤが意図しないタイミングで爆弾を置き自滅に追い込もうとする，という拡張デザインである（**図 19**）．

開発者は「兄弟で 1 つのゲーム機の占有権を争ったよう

図 19 “邪魔する” 拡張入力を実現した作例
Fig. 19 Participant’s use case: uncooperative bomber man.

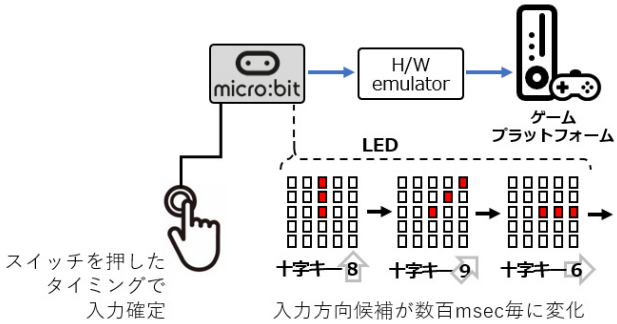

図 20 オートスキャンタイプの方向入力装置
Fig. 20 A directional input device with micro:bit.

な経験を，ゲームの要素として盛り込めないか？」というところからアイデアを膨らませたということであった．

5.2 開発者コミュニティでの事例共有

本ミドルウェアに関する Facebook 上の開発者コミュニティ [36] に共有された興味深い活用事例について，許可を得て紹介する．

開発者は，身体障害当事者でありながら，ゲームに応じたコントローラを独自に開発し，プレイを楽しんでいる方である．試作例では，H/W エミュレータを購入した後，MakeCode 環境を用いて，オートスキャンタイプの方向入力装置を構成した事例があげられている [37]．すなわち，micro:bit の LED 上で入力方向候補を順に示し，所望の方向が示されたときにボタンを押すことで，対応する dpad 押下信号がゲームプラットフォームへと入力される（図 **20**）．この構成であれば，身体機能が限られている場合でも，1 つのボタンで意図する方向を入力できる．

彼にヒアリングを行ったところ，提案ミドルウェアについては，①ビジュアルプログラミングで拡張が可能なこと，② 1 つのトリガに対して複雑な操作をマッピングできること，③少ない機器で拡張を構成でき，再現や公開が簡単なこと，のフィードバックを得た．①②については我々の開発意図どおりの感想であるため割愛し，ここでは③について述べる．

彼のこれまでの開発では，センサとゲームプラットフォーム間に PC や補助ソフト介在させることで入力の拡張やカスタマイズを行っていた．しかしセンシングからゲームプラットフォームへの入力経路にて複数の機器を設定しなければならないため，不安定であったり設定が煩雑となることもあった．そのため，障害をかかえる人むけへの入力方式を開発し公開したとしても「ユーザ側で再現するための負荷が高く使ってもらいにくい」という状況があった．しかし提案ミドルウェアを利用することで，拡張に関する記述がブロックプログラミングに集約され，またこれを規定の公開方法にのっており配布することもできる．そして上記公開コードと既存部品を組み合わせるだけで提案する入力拡張を再現できることで，配布から再利用のための負荷を大きく下げられる，とのコメントをいただいた．

5.3 考察

我々は電気回路的知識を必ずしも必要としない抽象度で，Node-RED と MakeCode を用いた複数のビジュアルプログラミング環境の選択肢を提供し，既存ディジタルゲームの拡張における参加障壁を下げることを図った．ハッカソンイベントにおいてどちらの選択肢もともに用いられたことから，このような複数の選択肢を用意することでより多くのユーザのニーズに対応でき，「作ってみたくなる」と思わせ，実際に行動へと誘導できたと考えられる．

一方で，1 名ではあるが提案手法を用いなかった参加者がいた．より多様な選択肢を開発し提供すること，あるいは使用を強制しないことの必要性も現時点では否定すべきではないと示唆される．

また，開発者コミュニティから共有された事例は，提案ミドルウェアの販売と開発者コミュニティの運営といった普及活動が潜在的ユーザに「作ってみたくなる」と思わせ，実際に行動へと誘導できたことを示す，明確な成果の 1 つであるといえる．本事例では個人がかかえるバックグラウンドと課題領域に対し，当事者として開発と試遊を試行錯誤する活動において，提案手法が効果的に活用されていることが観察された．さらに，第三者に成果を共有する際に，配布から再利用までの負荷が低いという提案手法の性質が重要であったという指摘は示唆的である．すなわち，より多くの人々に「作ってみたくなる」と思わせる要因の 1 つとして，制作物の配布の容易さも検討すべきであるという教訓が得られた．

短期間のハッカソンイベント，そして開発者コミュニティから共有された事例から，開発者である著者らの想定しない活用事例も多く得られており，既存ディジタルゲームを拡張した情報システムの開発において提案手法の切り拓いたデザインスペースの豊かさが示された．このような普及活動をより進めることは，既存ディジタルゲームの活用・拡張の可能性を探求するうえで有効な手法であると考えられるため，今後も継続していきたい．

6. 議論

6.1 遅延の評価

GameControllerizer では，通常のコントローラ操作と比較してミドルウェア内部の処理遅延が発生する．この処理遅延が，ゲームシステムそのもの起因する遅延に対して極端に大きければ，本来あるべきゲーム体験を損なってしまう．そこで，両者の遅延を比較し，考察を行った．

まず，ゲームシステムそのものに起因する遅延の状況については Ivkovic らの報告がある [38]．Ivkovic らはネットワーク遅延のないスタンドアロンなゲームシステムについて，コントローラ入力がゲーム画面上で反映されるまでの時間を計測している．報告では，ゲームシステム全体の遅延は 23〜243 ms と幅があり，特に専用ゲーム機とテレビを組み合わせた環境では遅延が大きい傾向にあった．文献内ではその原因についても述べており，主要な原因は平均で 30〜60 ms 程度の入力遅延をもつモニタだとしている．

次に提案ミドルウェアの遅延計測状況について説明する．我々は，**図 21** および**表 2** に示す評価構成を構築し，トリガ情報の生成から H/W エミュレータからの入力を受け取るまでの遅延を計測した．その結果，図 12 に示すフローを構成した場合の遅延は 17.6 ms であり，また図 3 で示すゲームパッドの全要素（dpad，button，stick0，stick1）のすべてを一度に制御する場合もその遅延は 18.1 ms であった．これは前記で述べたゲームシステムに起因する遅延と比べても大きな値ではないと考える．

ゲームを生業とする一部のプレイヤを除き，多くのゲームプレイヤがこれら一般的な機材を用いてゲームを楽しんでいる状況に鑑みれば，ミドルウェア遅延はユーザの体験を著しくは損なうものではないと考えられる．

ミドルウェアの遅延そのものをさらに減少させる工夫として，USB のポーリング間隔を短くすることも検討できる．基本的な USB HID 機器の仕様では，ホストマシンからのポーリングに合わせて “ボタンを押している” などの状態を返答する．Joystick/Mouse/Keyboard で一般的に用いられているポーリング周期は 8 ms であり，ポーリングのタイミング次第で最大 8 ms の遅れが発生する．この周期を 1 ms などに設定すれば，負荷は増大するが遅延は減らすことはできるだろう．これはそのような設定をホスト側で行える PC 上のゲームなどに限定した工夫である．

また，入力拡張の際に外部機器を用いる場合，これらの処理遅延や通信遅延が支配的になる可能性がある．実際前記のハッカソンにおいても会場内のネットワーク環境の貧弱さゆえに通信遅延が発生し，意図していた拡張ができなかったと報告した参加者も 2 名いた．応用時にはこれらの遅延を含めての拡張システム構築が必要になる．また，これらの遅延があることを前提として，どのようなゲーム拡張デザインにしたらよい体験が得られるかについては，そのヒントをユーザに提案することが必要かもしれない．

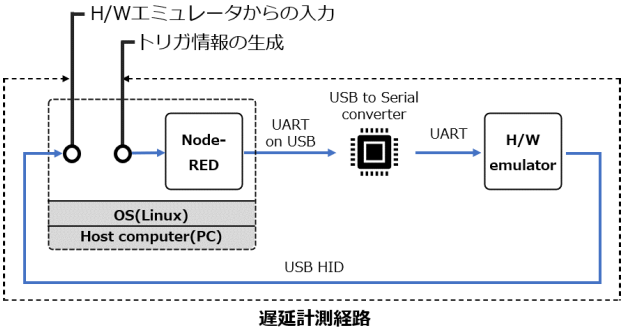

図 21 評価構成
Fig. 21 Evaluation architecture.

表 2 評価用ハードウェア仕様
Table 2 Specification of evaluation hardware.

PC	
CPU	AMD Ryzen 5 3600 (3.6 GHz, 6 core)
Memory	32G Bytes (DDR4 3200MHz)
GPU	NVIDIA GeForce GTX 1660 SUPER
HDD	SSD 512 GB
OS	Ubuntu Linux 18.04 (64bit)
USB to Serial converter	
Chip	FTDI FT234x

なお，ゲームの内容に踏み込む必要があるため一般的な議論は難しいが，ゲームデザインや通信アルゴリズムにより遅延の影響を軽減することも検討できる．たとえば FPS において遅延時にプレイヤが不利にならないような補助的な自動操縦を適用するもの [38]，事前予測された複数のユーザ操作候補を投機的に実行することより遅延を感じさせないようにするもの [39]，ゲームを対象とした試みではないものの音楽ゲーム（音楽のリズムに合わせてタイミングよくキー入力を行うもの）への適用が可能な事例として，鍵盤楽器演奏において鍵盤が指に触れてから押下し音が発生するまでの 50 ms 程度の遅延を先取りし補正するもの [40] などが参考になるだろう．

6.2 キーマッピング作業の効率化

5.1.2 項で述べたが，キーマッピング作業の効率化が求められる．GameControllerizer は多種のゲームプラットフォームに接続可能であることが特徴であるが，同じボタンでも接続する機器によってその効果が変化することがある．たとえば，多くのディジタルゲームにおいては，開始時に「スタート」に相当するボタンを押す必要があるが，これが図 3 中の何番のボタンに相当するかはゲームプラットフォームに依存する．ハッカソンにおいては，開発者に対して事前に調査した対応関係をドキュメントとして配布したが，設定間違いなどで時間を要したチームもあった．

これを解決するため，あらかじめ機器ごとに用意したプリセットを適用できる環境をビジュアルプログラミング部

に用意するなどの工夫が必要であると考える．

6.3 より多様な入力エミュレーション

本論文ではゲーム入力エミュレーション部で扱うゲームコントローラとして主にゲームパッドについて記述した．マウスやキーボードで操作するゲームについては，ゲームパッドの入力を特定のキー/マウス入力へと置き換える外部補助ソフトウェア [41], [42] を併用することでプレイ可能になるものもある．しかし，全キーを自由に押せるわけではないため，たとえばテキストチャットを行いながらゲームを進める，といった場合に対応できない．マウスやキーボードデバイスを電気的にエミュレートする H/W エミュレータは未公開であるが開発済であり，図 9 中の ARM Cortex-M0 マイコンのファームウェアを書き換えることで使用可能である．

一方で，マウスやキーボードデバイスをソフトウェアでエミュレートする S/W エミュレータも開発済みである．この S/W エミュレータは，入力エミュレーション技術である RobotGo [14] を用いて開発されており，図 5 に示す DSL4GC のうち Array[gc_mouse_word] や Array[gc_keyboard_word] を受け取り，動作する．ただし，これは PC 上でしか動作しないことに加え，H/W エミュレータと上記 [41], [42] を組み合わせることでカバーできる場合がほとんどであるため，現在は公開していない．

また，現在実現している電気信号レベルの H/W エミュレーション，そして OS レベルの S/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 [43] の検討も興味深い．これは今後の課題である．

6.4 「チート」の倫理的問題への対応

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「チート」技術となりかねない．特にあるプレイヤの行動が他のプレイヤの行動に影響を与えるような近年のオンラインゲームの多くは，外部ツールによる操作入力を規約により禁じていることが多い．提案手法を社会に普及させるフェーズでは，このような規約の存在を周知し遵守することを啓蒙することが必要であろう．一方で接続先のゲーム名を自動検出し，特定のゲームには提案手法を適用できなくする，などの技術的な対応も可能かもしれない．それは今後の課題である．

7. おわりに

本論文では多様な機器および情報源からの信号を統合して，最終的にゲームコントローラ操作へと変換することで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 を提案した．Node-RED もしくは MakeCode を用いた複数のビジュアルプログラミング環境を開発し，また提案システムの公開と販売，ハッカソンやハンズオンイベントの開催，開発者コミュニティを通じた普及活動により，多様なユーザに「作ってみたくなる」心の動きをもたらしたことを示した．今後はミドルウェアの性能と機能のさらなる向上に取り組むとともに，普及に向けたハッカソン・ワークショップなども引き続き実施する予定である．すでにテレビ番組での紹介や，スマートアパレルを開発するベンチャー企業が企業展示会において GameControllerizer を採用したデモ展示を行うなど，活用の裾野は広がりつつある．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，科学技術融合振興財団助成，JSPS 科研費 JP15H02735，JP16H02867 の研究助成を受けた．また，久池井淳様，長尾恭子様，丸山泰史様，ユカイ工学株式会社様，mizuhara 様，△×△×番長様から研究助成を受けた．ここに謝意を示す．さらに，活用事例紹介を快諾いただいた岡元雅様，ハッカソンイベント運営に協力いただいた一般社団法人 MA，およびハッカソン参加者の皆様に謝意を示す．

参考文献

[1] Kurihara, K.: Toolification of Games: Achieving Non-game Purposes in the Redundant Spaces of Existing Games, *Proc. ACE'15*, pp.31:1–31:5 (2015).
[2] 遊びの Re ハック（オンライン），入手先 ⟨https://we-are-ma.jp/blog/hackathon-oosaka2017/⟩（参照 2020-01-29）.
[3] Life-Size Katamari (online), available from ⟨https://iamarenaissancegeek.wordpress.com/2013/06/22/life-size-katamari-ball/⟩ (accessed 2020-01-29).
[4] TwitchPlaysPokemon (online), available from ⟨https://www.twitch.tv/twitchplayspokemon⟩ (accessed 2018-05-31).
[5] Revive USB（オンライン），入手先 ⟨https://bit-trade-one.co.jp/adrvmic/⟩（参照 2020-01-28）.
[6] Picade X HAT USB-C (online), available from ⟨https://shop.pimoroni.com/products/picade-x-hat-usb-c⟩ (accessed 2020-01-28).
[7] Octopad (online), available from ⟨https://www.gamasutra.com/view/news/337322/AltCtrlGDC_Showcase_Octopad.php⟩ (accessed 2020-01-28).
[8] Beginner's Mind Collective, Shaw, D.: Makey Makey: improvising tangible and nature-based user interfaces, *Proc. TEI'12*, pp.367–370 (2012).
[9] Xbox Adaptive Controller (online), available from ⟨https://www.xbox.com/xbox-one/accessories/controllers/xbox-adaptive-controller⟩ (accessed 2020-01-29).
[10] Entertainment Computing 2018 Qualification（オンライン），available from ⟨http://ec2018.entcomp.org/

qualification/⟩ (accessed 2019-01-10).
[11] Entertainment Computing 2018 Qualified EDAs（オンライン），available from ⟨http://ec2018.entcomp.org/qualified-edas/⟩ (accessed 2019-01-10).
[12] Kurihara, K., Itaya, A., Uemura, A., Kitahara, T. and Nagao, K.: Picognizer: A JavaScript library for detecting and recognizing synthesized sounds, *Proc. ACE'17*, pp.339–359 (2017).
[13] Yeh, T., Chang, T.H. and Miller, R.C.: Sikuli: using GUI screenshots for search and automation, *Proc. UIST'09*, pp.183–192 (2009).
[14] RobotGo (online), available from ⟨https://github.com/go-vgo/robotgo⟩ (accessed 2018-05-31).
[15] Automator (online), available from ⟨https://support.apple.com/ja-jp/HT2488⟩ (accessed 2018-05-31).
[16] Kato, J., Sakamoto, D. and Igarashi, T.: Picode: Inline photos representing posture data in source code, *Proc. CHI'13*, pp.3097–3100 (2013).
[17] PHANTOM-S (online), available from ⟨http://www.aten.com/global/en/product-landing-page/phantoms/⟩ (accessed 2018-05-31).
[18] Nintendo Labo (online), available from ⟨https://www.nintendo.co.jp/labo/⟩ (accessed 2019-01-30).
[19] alt.ctrl.GDC (online), available from ⟨https://gdconf.com/alt.ctrl.gdc/⟩ (accessed 2020-01-28).
[20] Matsuura, Y. and Kodama, S.: Cheer Me!: A video game system using live streaming text messages, *Proc. ACE'17*, pp.311–317 (2017).
[21] Sykownik, P., Emmerich, K. and Masuch, M.: Exploring patterns of shared control in digital multiplayer games, *Proc. ACE'17*, pp.847–867 (2017).
[22] Fujii, N., Sato, Y., Wakama, H., Kazai, K. and Katayose, H.: Evaluating human-like behaviors of video-game agents autonomously acquired with biological constraints, *Proc. ACE'13*, LNCS 8253, pp.61–76 (2013).
[23] Gym Retro (online), available from ⟨https://github.com/openai/retro⟩ (accessed 2018-05-31).
[24] Pommerman (online), available from ⟨https://www.pommerman.com/⟩ (accessed 2019-01-11).
[25] Node-RED (online), available from ⟨https://nodered.org/⟩ (accessed 2018-05-31).
[26] USB HID Usage Tables (online), available from ⟨https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf⟩ (accessed 2020-01-29).
[27] GameControllerizer (online), available from ⟨https://github.com/GameControllerizer/DSL4GC⟩ (accessed 2020-01-29).
[28] GameControllerizer 活用事例集映像（オンライン），入手先 ⟨https://youtu.be/P-z1Aa4zKzk⟩（参照 2020-08-13）.
[29] Microsoft MakeCode (online), available from ⟨https://www.microsoft.com/en-us/makecode⟩ (accessed 2019-01-10).
[30] micro:bit (online), available from ⟨https://microbit.org/⟩ (accessed 2020-01-28).
[31] Mbed (online), available from ⟨https://www.mbed.com⟩ (accessed 2018-05-31).
[32] Mbed USBHID (online), available from ⟨https://os.mbed.com/handbook/USBHID⟩ (accessed 2018-05-31).
[33] Mbed USB Joystick Device (online), available from ⟨https://os.mbed.com/users/wim/notebook/usb-joystick-device/⟩ (accessed 2018-05-31).
[34] Raspberry Pi GPIO (online), available from ⟨https://www.raspberrypi.org/documentation/usage/gpio/⟩ (accessed 2020-01-28).
[35] Dance Dance Revolution (online), available from ⟨https://en.wikipedia.org/wiki/Dance_Dance_Revolution⟩ (accessed 2018-05-31).
[36] GameControllerizer-UG（オンライン），available from ⟨https://www.facebook.com/groups/gamecontrollerizer/⟩ (accessed 2020-01-28).
[37] タッチ一つで 8 方向移動のテスト（オンライン），入手先 ⟨https://youtu.be/u2HXBbM8FSs⟩（参照 2020-01-28）.
[38] Ivkovic, Z. et al.: Quantifying and mitigating the negative effects of local latencies on aiming in 3D shooter games, *Proc. CHI'15*, pp.135–144 (2015).
[39] Lee, K. et al.: Outatime: Using speculation to enable low-latency continuous interaction for cloud gaming, MSR-TR-2014-115 (online), available from ⟨https://www.microsoft.com/en-us/research/publication/outatime-using-speculation-to-enable-low-latency-continuous-interaction-for-cloud-gaming/⟩ (accessed 2020-01-28).
[40] 宮下芳明：予知する鍵盤～リズム感のなさをリアルタイムで補正する～（オンライン），入手先 ⟨https://youtu.be/IvXN6yrNatk⟩（参照 2019-01-10）.
[41] JoyToKey (online), available from ⟨https://joytokey.net/en/⟩ (accessed 2020-01-28).
[42] Enjoyable (online), available from ⟨https://yukkurigames.com/enjoyable/⟩ (accessed 2020-01-28).
[43] Davidoff, S., Villar, N., Taylor, A.S. and Izadi, S.: Mechanical hijacking: How robots can accelerate UbiComp deployments, *Proc. UbiComp'11*, pp.267–270 (2011).

付　　録

A.1 EDA（Entertainment Design Asset）

我々は Entertainment Computing 2018 の Qualification に際し，GameControllerizer によって，心を「作ってみたくなる」状態へと動かしたいと宣言した．そのためのアプローチは以下のとおりである．

ある程度の IT スキルと創造意欲を持つような，主にハッカソンイベントなどに参加するようなエンジニア層を想定ユーザとし，アドホックに簡便なプログラミングを行うことで既存ゲームへの入力方法改変システム開発を支援するミドルウェアが用意される．これは可読性の高いゲーム入力制御情報の表現形式である DSL4GC，Node-RED を用いたヴィジュアルプログラミングによる入力方法変換ロジックの記述，およびハードウェアとソフトウェア両面による入力エミュレーションからなる．

提案手法により，既存のゲームに対し入力方法が様々に変更されており，新たなエンタテインメント価値の発掘，あるいは非ゲーム的目的を達成が体験できるコンテンツが容易に構成可能になる．これには (1) 多様なデバイス・情報源をゲームへの入力として用いたもの，(2) それらを既存の入力手法と共存させたもの，(3) 複数のゲームを同時に扱ったもの，(4) ゲームの外の世界に便益があるようにしたもの（ゲーミフィケーション），などが考えられる．

具体的な作例として，(1) についてはスマートフォン，micro:bit，m5stack，Sony MESH タグなどに搭載されている加速度センサ，タッチセンサなどに連動して，より多様

な身体動作や他者との協調作業に基づきスーパーマリオブラザーズをプレイできるコンテンツ，(2) については通常のコントローラ入力に加えて Amazon Echo などのスマートスピーカデバイスを用いることで音声による抽象的な指示入力を可能にした格闘ゲームなどが提供される．(3) については，1 つのコントローラの入力によりテトリスとぷよぷよを同時にプレイするコンテンツなどが提供される．(4) については，肩たたきの動作をウェアラブルセンサで検知し，それを音楽ゲームの入力にすることで家族などと楽しくコミュニケーションを図れるコンテンツなどが提供される．

栗原 一貴 （正会員）

津田塾大学学芸学部情報科学科教授．博士（情報理工学）．HCI および EC 分野において物議を醸すシステム開発研究を得意とする．著書に『消極性デザイン宣言』がある．2012 年イグノーベル賞等受賞．

土井 伸洋

早稲田大学大学院情報生産システム研究科博士後期課程修了．博士（工学）．現在は（有）来栖川電算で深層学習による物体認識の業務に携わる傍ら，EC 分野にかかわる作品創作や発表も行う．