バグトラッキングシステムを組み込んだゲーム開発環境のアーキテクチャの設計

2015SE074 高木裕也
指導教員：沢田篤史

1　はじめに

ゲーム機本体の高度化が進んでいる現代において, ゲーム開発も複雑化が進んでいる．ゲーム開発を支援する目的でゲームエンジンと呼ばれるソフトウェア開発環境が提供されている．ゲームエンジンによって開発は効率化されたが, ゲーム上で発生する故障 (failure), 一般的にバグと呼ばれるものの大半はゲーム開発の現場でデバッグを行う作業者であるデバッガによって探されている．

デバッガはプログラミングなどの知識を持たないアルバイトなど, 十分な専門知識を持つエンジニアではない場合が多い. それ故に報告されるバグの情報が漠然としておりバグ修正を行うエンジニアにかかる負担が大きくなる. エンジニアにかかるバグ修正の負担が大きいほど, デバッグ作業が可能な時間内に修正可能なバグの数が減ってしまう．エンジニアではないデバッガでも具体的なバグ情報を共有出来るようにして, 効率的なデバッグのできる機能が必要である．

本研究ではデバッグ中に発生したバグの情報を自動でファイルにまとめて, バグ情報として共有を支援する機能の実現を目的とする. その実現に向けて, ゲームエンジンのデバッグツールにデバッグ中に関連するデータを一つのファイルにまとめて共有できるようにする機能を持たせるようにソフトウェアアーキテクチャの設計と実現手段の検討を行う. この機能によりバグ情報が詳細化され, 一つのバグ修正にかかる原因解明の手間が軽減される．

バグ情報の編集をする機能と, その情報を共有できるように, バグトラッキングシステムを組み込んだゲームエンジンのアーキテクチャを設計する. 設計するアーキテクチャは Blackboard アーキテクチャパターンに基づく. このメッセージ送信機能ではデータを一時的に一括で蓄積する必要があるので, このアーキテクチャパターンを用いる．

2　背景技術

2.1　デバッグツール

一般的なゲームエンジンに搭載されたデバッグツールには図 1 のようなコンポーネントが実装されている．

記録と再生はゲーム実行中の映像の記録と再生を行い, バグ発生時の再現ができる．メモリとパフォーマンス統計情報はメモリリソースの解析・ゲーム実行中の処理時間の計測を行うことで, メモリリソース不足や冗長性の改善につながる．ゲーム内メニューまたはコンソールはゲーム実行中の画面にメニューやコンソールを表示し, ゲーム内の設定を変更することができ, ゲームをプレイ中に様々な状態を簡単に作ることができる．

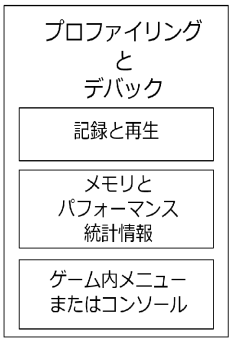

図 1　ゲームエンジンに含まれるデバッグツール

2.2　Blackboard アーキテクチャパターン

このパターンは図 2 のように Blackboard・Knowledge Source・Control の 3 つで構成されている．Blackboard にはデータが箇条書きのように羅列してある． Control は Blackboard の状態を最初の loop 開始から nextSource を用いて繰り返し監視しており, Blackboard の状態に応じて利用する Knowledge Source を決める．Knowledge Source は単一から複数の独立したコンポーネントからなり, Blackboard のデータを理解し読み書きすることができる．

Knowledge Source が Control から受信した activates をもとに, Blackboard に operates on 処理をすることで課題を解決するようにアクションを起こすシステムである．

Blackboard を用いてバグに関連したデータを一括で蓄積できるように, このアーキテクチャパターンを適用する．

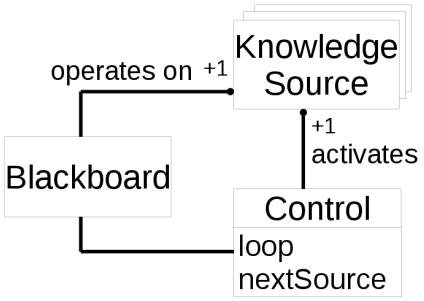

図 2　Blackboard アーキテクチャパターン

2.3　バグトラッキングシステム

バグトラッキングシステムは発見したバグに関する情報の管理・共有を支援するシステムである．

デバッガは発見したバグの内容・発生条件などの情報をこのシステムに送信する. システムはバグ情報を受信すると, その情報をデータベースに保存し権限のあるものに閲覧可能な状態にする. エンジニアはその情報をもとにバグ修正を行う.

3　デバッグ支援ツールの設計

3.1　アーキテクチャ

Blackboard を用いて設計したアーキテクチャを図 3 に表す.

ゲームをプレイ中のデータを持つメモリとパフォーマンス統計と映像と記録や実行中のソースコードのあるソースファイルをデータの蓄積をさせる Knowledge Source にする. 抽出コンポーネントをデータからバグ情報を抽出する Knowledge Source にする. バグトラッキングシステムをバグ情報を管理する Knowledg Source にする. loop の開始はデバッガがバグを見つけ, バグ情報の送信命令を出すさいに命令がでるので, デバッガが操作することのできるゲーム内メニューを Control にする.

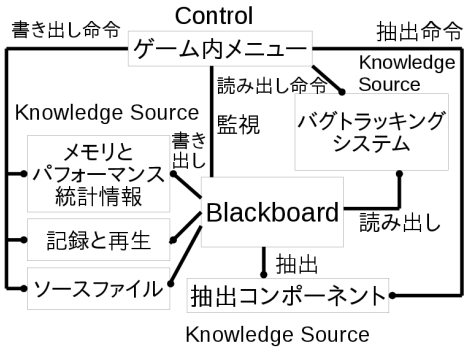

図 3　設計したアーキテクチャ

3.2　コンポーネントの役割

ゲーム内メニューを Control として Blackboard の監視と各 Knowledge Source にデータの処理を行わせる. デバッガがゲーム内メニューを呼び出してデータを Blackboard に蓄積させる Knowledge Source に書き出し命令を送る.Blackboard にデータが蓄積されたらバグ情報を抽出する Knowledge Source に抽出命令を送る. バグ情報が抽出されたらバグ情報を共有する Knowledge Source に読み出し命令を送る.

メモリとパフォーマンス統計情報・記録と再生・ソースファイルが Knowledge Source として Blackboard にデータを書き込むことで発生したバグに関連するデータが蓄積される. ゲーム内メニューから送信命令が送信されたらその時実行中のプレイデータを Blackboard に書き込む.

抽出コンポーネントが Knowledge Source として Blackboard に蓄積されたデータから必要なものを集め, バグの種類などのデータを加えてバグ情報としてまとめる. ゲーム内メニューから抽出命令を受信したらデータを Blackboard から読みだしてバグ情報の抽出をして, まとめたバグ情報を Blackboard に書き込む.

バグトラッキングシステムが Knowledge Source として Blackboard のバグ情報を読み込み, 管理･共有をする. ゲーム内メニューから読み出し命令を受信したら Blackboard からバグ情報を読み出して管理し, 開発チーム全体で共有出来るようにする.

4　考察

4.1　ツールの有用性とアーキテクチャの妥当性

本研究の設計を実装することで, 専門知識のないデバッガであってもバグに関連した詳細な情報を送信することが出来るので, 正確なバグ情報が容易に共有出来るようになると考える.

アーキテクチャ設計の代替案として Pipes and Filters アーキテクチャパターンを適用する方法が挙げられる. 抽出処理を Filter にしてデータストリームを構成することで実現することが可能であるうえに, 本研究では抽出処理が追加・変更がされるが, このパターンを用いることで対応出来る. ただし本研究では扱うデータの種類によって処理を動的に再構成する必要があるので Pipes and Fiters では難しい. したがって処理を動的に再構成することが可能で処理の追加・変更が容易な Blackboard アーキテクチャパターンを用いるのが妥当である.

4.2　関連研究との比較

関連研究として丹野の提案するデバッグツールと比較すると, 本研究は情報共有の点で優れている.

丹野のデバッグツールではリアルタイムにバグ情報の取得するという点で優れているが, デバッガが専門知識を持っていることが前提であり, バグ情報を送信する機能がない. これに対してバグ情報の共有の面で優れていると考える.

5　おわりに

本研究はゲームのバグ情報をメッセージとして送信し, 管理・共有をする機能をゲームエンジンに組み込むためのアーキテクチャ設計をし, その機能によってもたらされる結果と課題を考察した. 今後の課題として抽出前に送信されるデータをより正確なものにし, 抽出の手間を減らすことで, リアルタイム性の高い機能にする必要があると考える.

参考文献

[1] 丹野　治門：“ゲームプログラムに適したリアルタイム性の高いデバッガの提案と実装”, 情報処理学会論文誌 プログラミング, vol. 1 no. 2, pp.42-56 (2008.9).

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-OrientedSoftware Architecture, John Wiley&Sons, Ltd, 1996

[3] Michael Huttermann, Agile ALM, Manning Publications, 2012

[4] ジェイソン・グレゴリー（訳　大貫宏美, 田中幸）, ゲームエンジン・アーキテクチャ第 2 版, SB クリエイティブ株式会社, 2015