

Launch

前編 Hot Soup Processor 3.6 を用いたモックアップの実装

2024 年 3 月 31 日

Virtual Causality

1 背景・目的

筆者が大学院を中途退学してから、1 年が経過しようとしている。本プロジェクトは、大学院における研究計画が自身の技術力不足によって破綻し、学術の世界に留まれなくなったことがきっかけで始動した。

前提として、筆者にはシステム開発の実務経験が無く、実務に関係しそうなその他の経験としては、学部における演習、制作、研究と、学部から大学院にかけての 3 年間で受け持ったいくつかの演習のティーチングアシスタント程度であった。未経験者が実力を証明する手段のひとつとして、自主的に何らかのシステムを開発し、ポートフォリオを先方にアピールすることが挙げられる。

本プロジェクトでは、昨今の急速な情勢変化の影響を受けにくく、業務を遂行するうえで便利と考えられる、低いレイヤーの制御を含むシステムとして、パズルゲームを開発する。

歴史が長く、人気の高いパズルゲームとして、4 つのブロックからなるブロック群を移動、回転し、盤面の下方に着地させ、1 行がブロックによって満たされたとき当該行が削除され、それより上のブロックが削除された行数だけ降下するものがある（以下、「ゲーム T」とする）。ゲーム T では、回転操作の結果を含めると、操作するブロック群の幅は最大 4 ブロック、高さも最大 4 ブロックである。幅も高さも最大の 4 ブロックとなるブロック群は 16 個のブロックからなるが、ゲーム T において登場しない 4 個以外の個数からなるブロック群を含めて同様のゲームルールを実現したとき、果たしてそれは評価できるゲームとなるのだろうか。ならない場合、どのようなブロック群を登場させない必要があり、どのようなゲームルールを追加するとよいか。本プロジェクトでは、これらの疑問を解決するため、実際に新機能を実装したゲーム「Shaft」を開発する。

本資料は前編である。前編は、実際に動作、プレイが可能なモックアップを実装するまでの過程を収録する。

2 知識

ゲーム T には、数多くの派生作品が存在する．ここでは著作権等の問題により特定の作品に関する記述は避けるが，ブロックの数を増やすことでゲーム性を変化させる方法はあまり見られなかった．

3 進行

3.1 用語

本プロジェクトにおいて用いる特定の意味を持つ単語とその意味を挙げる．

3.1.1 盤面

後述のブロックが存在できる空間である．予告と保持のための領域，操作中のブロック群を保持するための領域はそれぞれ 4 行 4 列の大きさである．また，ブロック群を操作でき，着地したブロックが存在できる領域が存在する．操作中のブロック群の振る舞いについて言及する場合，後者の領域を指す．

3.1.2 ブロック

空白でないとき，ゲームシステムが盤面の 1 行 1 列，いわゆる 1 マスを満たしていると判定する対象である．いくつかの種類を定義することができる．

3.1.3 ブロック群

1 個以上 16 個以下のブロックからなる，4 行 4 列のグループであって，まだ着地していないものをいう．

3.1.4 着地

盤面における当該領域に，プレイヤーの操作の結果到着し，ゲームシステムの判定によってプレイヤーの操作対象でなくなった状態をいう．

3.1.5 予告

プレイヤーが次に操作するブロック群を表す．

3.1.6 保持

プレイヤーが操作していたブロック群を着地させずに予告されたブロック群へ操作対象を移す操作，また，そのとき操作されていたブロック群が移動する先の領域を表す．

3.2 機能要件

3.2.1 画面

システムは，以下の 16 画面からなる．

3.2.1.1 盤面（インゲーム，表示名なし）

実際にゲームをプレイする画面であり，盤面の一部，スコア等ゲームシステムの統計情

報を表示する.

3.2.1.2 ポーズ (PAUSE)

プレイヤーによって一時停止の操作が行われた際に表示される画面であり、ゲームのリセット、終了、コンフィグ等の画面へ移動することができる.

3.2.1.3 クリア (CLEAR)

ゲームシステムが一定の条件を満たしたとき自動的に移行する画面であり、スタッフロールの閲覧やゲームのリセット、終了ができる.

3.2.1.4 ゲームオーバー (GAME OVER)

ゲームシステムが一定の条件を満たしたとき自動的に移行する画面であり、ゲームのリセット、終了ができる.

3.2.1.5 タイトル (SHAFT)

ゲームを起動したとき最初に表示される画面である. ゲームプレイのためのレベル選択、コンフィグ等の画面に移動できる.

3.2.1.6 レベル選択 (LEVEL)

ゲームの難易度、モードを選び、ゲームを開始することができる画面である.

3.2.1.7 設定 (CONFIG)

ゲーム内の設定を変更することができる.

3.2.1.8 表示 (SEGMENT)

統計情報の表示、非表示等の切り替えを行う.

3.2.1.9 画質 (GRAPHICS)

画面解像度の変更を行う.

3.2.1.10 音声 (SOUND)

音量の変更を行う.

3.2.1.11 操作 (CONTROL)

操作方法の変更を行う.

3.2.1.12 情報 (INFO)

本ゲームおよびその実行環境に関する情報を閲覧することができる.

3.2.1.13 概要 (ABOUT)

本ゲームの概要を、文章や画像を用いて表示する.

3.2.1.14 システム (SYSTEM)

本ゲームの実行環境を簡易的に表示する.

3.2.1.15 権利 (LICENSE)

本ゲームの実装に用いたライブラリを紹介し、関連ドキュメントの場所を表示する.

3.2.1.16 スタッフ (STAFF)

本ゲームの開発を行ったスタッフを列挙する。

3.2.2 画面遷移

画面遷移は、下図のとおりである。

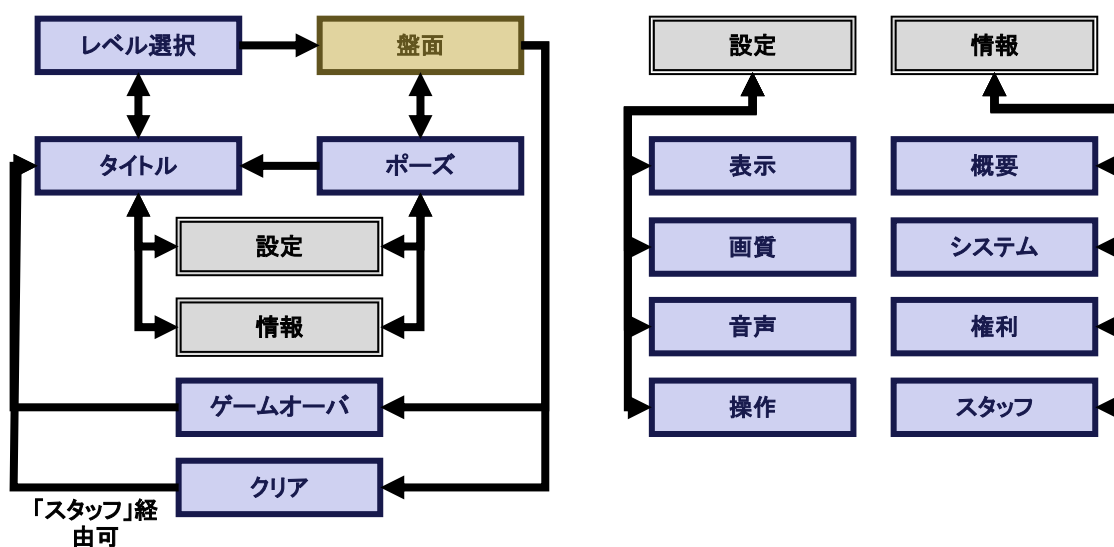


図 1 画面遷移図

3.2.3 ゲームシステム

3.2.3.1 基本

ゲームシステムは一定時間ごとに操作対象のブロック群を一行下に移動させる。盤面の一番下または他のブロックと重ならない最後の位置に到達したとき、一定時間の猶予ののち着地させる。このとき、1 行の中に 1 つも空白が存在しないとき、その列を消去し、それより上のすべての列を、同時に消去した列の数だけ下に移動させる。また、消去されたブロックの種類と同時に削除された行数に応じてスコアが加算され、条件を満たしたときゲームのレベルを上げる、またはクリアとする。着地したブロック群が高さ 28 行を超えたとき、ペナルティを科し、条件を満たしたときゲームオーバーとする。

3.2.3.2 移動

ブロック群を盤面の中で左右に動かすことができる。

3.2.3.3 回転

ブロック群を盤面の中で左右に 90 度回転させることができる。

3.2.3.4 落下

早送りボタン押下によって待ち時間を一定以下にするものと落下ボタン押下によって 1 フレームごとに 1 列分落下するものの 2 種類用意する。

3.2.3.5 保持と使用 (PUSH, POP)

プレイヤーは、保持の領域にブロック群を保持することができ、操作によって最後に保持したものを利用することができる。

3.2.3.6 メニュー画面関連

ゲームシステムがインゲーム以外の状態であるとき、メニュー画面に関する操作体系に移行する。プレイヤーはカーソル移動、決定、キャンセル等の操作を行い、画面遷移図のように移動することができる。

3.2.3.7 ユーティリティ

画面をそのまま画像として保存するスクリーンショット機能、盤面をテキストファイルに書き出すログ出力機能を備える。

3.3 非機能要件

3.3.1 サポート

付属のドキュメントにおける FAQ の設置、ファイル共有サービスのコメント欄を用いた直接サポートを用意する。本ゲームにおいて体験を損なう予期せぬ挙動が報告された際、随時対応のうえ、修正版の公開に努める。

3.3.2 データの保証

本ゲームに関するスコア等の統計情報を公式に収集、比較、公開することは無い。

3.3.3 インゲーム

プレイヤーが可能な操作によって、ベストな状態を必ずしも実現できる必要は無いが、ベターな状態を実現できる可能性が十分に高く、所定のクリア条件を十分な確率で達成できること。

3.3.4 メニュー画面等

ゲーム外の演出を挿入せず、速やかなレスポンスを実装すること。操作方法の案内を必要に応じて表示すること。

3.3.5 想定されるプレイ環境

携帯端末を除くパーソナルコンピュータにおいて、オペレーティングシステムが快適に動作する程度の性能をもつハードウェア、Windows 10 以降の Windows、Java (OpenJDK 17)、LWJGL 3.3.0、Vulkan 1.0 以降が動作する環境、出力先として、フル HD (1920x1080) 以上の解像度、60Hz 以上のリフレッシュレートを実現でき、24bit フルカラーに対応するディスプレイ、入力デバイスとして 60%以上のキーボードが接続されていることを想定する。全ての設定においてプレイする場合、80%以上のキーボードを要求する。

3.3.6 技術要件

開発言語はモックアップの実装には Hot Soup Processor 3、完成版には Java と GLSL を

使用する．モックアップには外部ライブラリを使用しない．完成版の実装には LWJGL (特に, Vulkan, GLFW), 必要に応じて JOML を用いる．本プロジェクトの目的は, システム開発における成果物の妥当性評価と修正に欠かせない, 企画から実装に至るまでの能力の確認であるから, 全工程において生成 AI を利用しないこととする．

3.3.7 開発

本ゲームの開発は前期と後期に分割する．

3.3.7.1 前期：企画からモックアップ実装まで

はじめに, ゲーム T のシステムを基に, 操作対象を複雑化したときプレイヤーの体験を維持するために必要な要素を検討しつつ, 画面の構成, デザインやゲームシステムの挙動について, アイデアをまとめる．

次に, 開発人員の制約から素材点数の最小化および制作の省力化を図るため, 効率的な手法と共通化できる部分を見つける．また, 並行してモックアップの実装を進め, 実際に開発者がプレイすることで, 不満点等の洗い出しを行い, モックアップに改善策を随時反映する．

最後に, Hot Soup Processor 3 から Java に移植する際のソースコード読み替えについて検討する．

3.3.7.2 後期：完成版の実装からリリースまで

まず, 前期の最終段階において検討を行った読み替えに基づいて速やかに実装を進める．

続いて, モックアップ実装時と同様にデバッグを行う．並行して配布のためのファイル共有サービスの選定, 配布ページの作成を行う．後期の要件については, 追って検討する．

4 経過

4.1 ゲームルールの変更と追加

表示できる盤面の大きさには限りがあるため, 盤面の外の活用について考える．上下左右のうち, 上以外の領域にはみ出す場合, 視認性を必要以上に悪化させたり, スクロールによって視界を確保する場合でも操作が煩雑になったりするなど, 不都合が多い．一方, 上の領域はブロック群がプレイヤーの制御対象になったあとのわずかな時間を除いて視認性に影響を与えることは無い．そこで, 盤面の上 4 行は不可視でありながら操作可能な領域とし, 操作対象のブロックが最初に出現する位置は最も上の行に固定とする．







ブロック群に含まれるブロックが最大 16 個となることで, 着地させるにはあまりに都合の悪いパターンが多く出現することが考えられる．これらを解決するためには, 出現パターンの調整, 保持領域の拡大という方法が挙げられる．テストプレイによる調整の結果, 従来の 4 ブロックからなるブロック群, 2 行 3 列または 4 列が満たされたブロック群の出現率を上げる, 3 行 3 列を残して空白に置き換える, 四隅を空白に置き換える, 中央の 4

ブロックを充填する等の処理を適用することで、隙間の発生を多少軽減できることが分かった。また、予告を 5 つ表示することで、より計画的な操作を可能とした。保持領域の拡大については、保持と使用の操作だけで高い自由度を確保する必要があったことから、いくらかの検討を要した。まず、拡大する領域は予告の末尾に見えないものを 1 つ、保持に見えるものを 2 つとする。つまり、プレイヤーにとって、保持として 3 つの領域が見えており、予告に追加されたものは見えないことになる。予告に追加された領域は保持していたブロック群を使用する際の、操作対象としていたブロック群の移動先であり、空でない場合、操作対象のブロック群が着地したのち、予告の 5 つ目が生成される代わりに再利用され、再び空となる。さらに、使用の操作において使用されるブロック群は最後に保持したものというルールを変更せずに検討したため、保持の領域はスタック構造となった。ただし、スタック構造では最後以外に保持したブロック群が使用しにくいことから、予告の追加領域が全て満たされている場合、操作対象のブロック群と直接交換する仕様とすることで、着地直後のように予告の追加領域が空の場合であれば、2 回続けて使用して保持領域の 2 つめのブロック群を使用することができるようにした。

従来のゲームルールでは、次のブロック群の出現を妨げるような位置にブロックが積まれているとき、即座にゲームオーバーとなるが、ゲームルールを変更したことでそうした状況が発生しやすい本ゲームでは、プレイヤーから見える範囲のうち上から 8 行のブロックと空白およびはみ出したブロックによって決定されたダメージ、すなわちペナルティを受け、体力が 0 以下になった場合ゲームオーバー、そうでない場合は当該範囲を全て空白に書き換え、スコアを加算せずにゲームを続行する仕様に変更した。また、一連の変更によって体力を回復する、またはより激しく消耗するイベントを挿入する余地が生じたため、ブロックの種類を増やすことでゲーム性の向上を試みた。

回復や消耗、スコアの計算に関わる追加要素として、変種のブロックを以下の通り 6 種類用意する。

表 1 変種のブロック一覧と解説

種類	画像	解説
宝石		普通のブロックより回復、ペナルティともに大きい
発見されていない宝		そのまま着地すれば普通のブロックより回復効果が小さい
発見された宝		発見されていない宝をスタックに投入することで発生する回復、ペナルティともにきわめて大きい
有効な爆弾		発見されていない宝と排他的に発生する 着地すると無効化され、スタックに投入されると爆発する
無効な爆弾		有効な爆弾が着地したときのみ発生 回復、ペナルティともに大きい
残骸		有効な爆弾がスタックに投入されたとき、ブロック群が変化 回復は小さく、ペナルティが極端に大きい

最後に、レベルの設計について考える。ゲームルールを複雑化したことで、考える時間が長くなることが予想されるため、ブロック群の落下時間は特に低レベル帯において長く設定する。また、1 行を揃えること自体が難しくなっているため、レベルアップに必要な行数は少なめに設定する。

4.2 実装

4.2.1 概要

Hot Soup Processor (以下 HSP と表記) は、Basic に似た構文を持つ言語である。オブジェクト指向プログラミング等には向かない言語仕様のため、手続き型プログラミングによる実装を行う。

はじめに、主要な変数について考える。盤面データは、完成版においてシェーダに直接与えるため、符号付き整数型 32bit (以下整数型変数はすべて左記の仕様) の 1 次元配列に格納する。上位 24bit には演出用の色情報を格納し、下位 8bit にはブロックの種類の情報を格納する。インゲームやポーズ画面等の、ゲームシステムの状態は整数型変数に格納する。操作対象のブロック群の左上の座標には 2 つの整数型変数を用いる。状態変数の値の意味や真偽値をはじめとする各種定数は予め定義しておき、必要に応じて使用する。

次に、大きなデータを格納する領域について考える。テクスチャ用、サウンド用のバッファは HSP に用意された命令を用いて確保し、使用する。

4.2.2 不具合と対処

HSP を用いたモックアップは原始的な方法を用いて実装されている。そのため、不具合が発生した際には、エラーや例外を読む方法があまり通用せず、挙動の異常から関係する変数や処理を予想する必要がある。本節では、実装中に発生したいくつかの不具合について述べる。なお、HSP には符号無し整数型が存在しないため、符号無し整数型を用いて不具合を解消することはできない。完成版の実装に用いる Java も同様である。

4.2.2.1 ブロック群の座標が取りうる値

ブロック群の操作領域は 4 行 4 列であり、左上のブロックを基準に座標を管理していることは前述のとおりだが、当初この座標をもとに盤面側面の当たり判定を実装していたことで、一部のブロック群が左右端に到達できなくなる症状が発生していた。また、ブロック群の座標は 2 次元、盤面の座標は 1 次元であったことから、修正作業においてループカウンタ等の更新ミスによってブロック群がせん断されたように見える現象も発生した。

4.2.2.2 ビットフラグの使い方

本ゲームでは、状態管理等にビットフラグを用いているが、その一部において、誤って加算によってビットフラグを立てる実装が存在した。結果、最上位ビット、すなわち符号ビットが既に立っている変数に別のフラグを立てたとき、期待した出力が得られないことがあった。加算を論理和に置き換えることで、不具合を解消した。

4.2.2.3 空白の条件

一部のブロックには色情報が存在するが、赤色の最上位ビットは符号ビットである。当初、これを失念し、ブロックの存在条件を「0 より大きい」としたことで、限られた状況下、すなわち干渉するブロックすべてが色情報を持ち、そのうち赤が 128 以上の値をとるとき、それらのブロックが上書きされてしまうという、極めて分かりにくいバグが発生していた。

4.2.3 出力：画面の描画

HSP には、何もコードを書かない状態で実行した場合でも、クライアント領域に何も表示されない幅 640、高さ 480 ピクセルのウィンドウが出現する仕様が存在する。これは命令によって変更可能であり、表示されない仮想画面の作成も簡単である。使用例として、描画中の画面が表示されることで発生する画面のちらつきを無くすために利用されるダブルバッファリングの実現、テクスチャを保存するバッファの確保等が挙げられる。ただし、色空間は 24bit であり、ピクセルアルファブレンド等の処理には、HSP において用意された別の方法を用いる必要がある。以下に、本ゲームの画面描画に用いる命令とその解説を示す。

表 2 描画に用いる命令と用途

命令	用途
screen	画面または画像バッファの初期化
gsel	画面または画像バッファの選択
gmode	画像コピーの方法指定（ピクセルアルファブレンド等）
color	背景色の変更
boxf	指定範囲の塗りつぶし
pos	描画先左上の座標の設定
gcopy	指定範囲からのコピー
gzoom	指定範囲からの拡大・縮小コピー

4.2.4 ソースコードの解説

本節では、本ゲームの実装について、概ねラベルによって区切られた範囲ごとに大まかな役割と解説を記す。

4.2.4.1 配列変数と定数の宣言：ラベル名無し

HSP では、整数型配列の初期化に dim 命令を、ラベル型には ldim 命令、文字列型には sdim 命令をそれぞれ用いる。モックアップにおいて、大半の制御は整数型変数を通して行うので、大半の変数は整数型配列である。また、配列への代入は Java 等とは異なり、添え字を用いずに値をカンマ区切りで並べることで可能である。

定数（const）の宣言は、#const プリプロセッサ命令を用いる。モックアップでは、主に真偽値、特定の意味を持つ配列のオフセット、インデックス、状態を表す 32bit のビット

フィールドを定義する。

4.2.4.2 各種リソースの読み込み：ラベル名無し

画面に表示されない仮想画面に画像ファイル，バッファにテキストファイルを読み込み，事前処理を済ませる．HSP では特定のファイルの読み込みについて専用の命令が用意されている．

4.2.4.3 ゲームループ：main

アプリケーションの起動から終了まで何度も実行される，いわゆる無限ループを持つ．全ての状態において共通する入力取得，状態によって異なる入力取得，描画の順に行い，画面を更新して 1 フレーム分の時間プログラムを停止し，ラベルの最初にジャンプする．ゲームの各種変数の更新は，入力を処理する部分が担う．

4.2.4.4 各種変数の初期化：init

ゲームを開始する際に毎回初期化される変数群である．一部の変数はプレイヤーの操作によって，このサブルーチンが呼び出されるより前に設定された値を用いるため，ここでは初期化されず，必要に応じて別個に初期化を行う．

4.2.4.5 降下時間の増加要求：add_wait

プレイヤーがブロック群の移動，回転等の操作をした際，操作が完了した状態を認識しやすいように，わずかに降下までの猶予を延長する処理である．ただし，連打によって際限なく延長する不正を防ぐため，レベルごとに設定された猶予時間を超えることはできないようになっている．

4.2.4.6 キー押下判定とリセット：key_pshdn

ゲームループにおいて，対象のキーと押下判定が真になった場合に呼び出されるサブルーチンを与えられた状態で毎フレーム実行されるサブルーチンである．キーが押された最初のフレームのみ与えられたサブルーチンを呼び出し，以降キーを離すまでフラグによってロックする．このフラグによって長く押しすぎることで発生する多重処理を避けられる．

4.2.4.7 目的別の操作：主に k_*

押下判定のサブルーチン k_pshdn に続く多数のサブルーチンは，押下判定があったとき呼び出されるサブルーチンである．押されたキーと呼び出されるサブルーチンの対応は，事前に宣言されたラベル型配列によって管理されているが，主にインゲームとメニューで区別されているのみであり，詳細な状態はサブルーチン内部でさらに条件分岐を用いて判別される．インゲーム用のものは，対応する後述のサブルーチンを呼び出すだけである．対してメニュー画面用のものは，状態ごとに異なるカーソルの位置の制御や状態遷移，設定変更の機能を呼び出す等の処理を行う．

4.2.4.8 ブロック群の回転：rot_l, rot_r

ブロック群を左右に 90 度回転させる処理である．本ゲームにおいて，ブロック群の回転軸は固定されているため，単純なスワップ処理だけで記述することができる．また，回転

後のブロック群が他のブロックと干渉する場合は操作を無効にしなければならないので、判定後に逆回転の操作を呼び出し、操作の影響を打ち消している。打ち消しの逆回転において、回転後の結果は入力による操作の前と同じであり、ブロック群が存在できる条件を満たしているから、判定は不要である。

4.2.4.9 ブロック群の移動：mov_l, mov_r

ブロック群を左右に 1 ブロック移動させる処理である。ブロック群を構成する 4 行 4 列の中に、1 つもブロックを含まない列が存在する場合、当該列が盤面をはみ出す状態を許容しなければならないので、ここでは盤面の範囲を判定しない。回転の操作と同様に、移動後にブロック群が存在できない場合、打ち消すために逆方向への移動操作を呼び出す。

4.2.4.10 着地前の確認：check

ブロック群を 1 ブロック下へ移動し、降下できるか確認する。降下できない場合、着地前の状態に状態変数を更新する。逆に、ブロックを移動または回転し、降下が可能になった場合、着地前の状態から通常の状態に更新する。確認後は 1 ブロック上へ移動する。

4.2.4.11 降下：mov_dn

ブロック群を 1 ブロック下へ移動する。制限時間内に一度も操作が行われなかった場合等、ブロック群の降下先の判定は行われていない可能性があるため、ここでも判定を行う。着地前の状態になったとき、状態変数を更新するとともに降下操作を打ち消し、高速落下のために猶予時間が書き換えられていた場合は元に戻す。

4.2.4.12 移動または回転の確認：req_mr

指定された位置にブロック群が存在できるか検証する。操作中のブロック群は一部が盤面をはみ出している可能性があるため、ここで盤面外の判定を行う。盤面外またはブロックが既に存在する盤面内に操作中のブロックが 1 つ以上存在していた場合、返り値に反映する。返り値が 0 のとき、存在条件を満たしている。





4.2.4.13 判定：judge

着地が確定したときに呼び出されるサブルーチンである。盤面の行内すべてにブロックが存在する場合、消去のためのフラグを立てる。発見されていない宝石が存在した場合、背景を黒に書き換え、有効な爆弾が存在した場合、無効な爆弾に書き換える。

4.2.4.14 ブロック群の生成：gen_c1

乱数を用いてブロック群の生成を行う。はじめに、変数 `i_v1` に 16 個のビットをランダムに立て、変数 `i_v2` に 0 から 15 の値をランダムに代入する。変数 `i_cnt` に、変数 `i_v1` に立っているビットの数を代入する。次に、変数 `i_cnt` の値が 8 未満であれば、10 進数で 1632 と表現される中央の 4 ブロックを充填する。続いて、変数 `i_v2` の値に応じて下表のようにブロックを書き換える。

表 3 ブロックの書き換え条件と結果

条件	画像	備考
$0 \leq i_v2 < 7$		再抽選後、全部使用の 1 種と既存の 7 種のいずれかに
$7 \leq i_v2 < 10$		再抽選後、4 種のいずれかに
$10 \leq i_v2 < 15$		再抽選後、上下左右の 9 個を切り出す
$i_v2 = 15$		角の 4 つを削除

返り値は操作領域の 16 ブロックについて、ブロックが存在する場合は 1、しない場合は 0 として、整数の下位 16bit を用いて表現したものである。

4.2.4.15 ブロックの生成：gen_c2

乱数を用いてブロックの種類を決定する。はじめに、変数 i_v2 を用いて排他的なブロックのうち 1 種類を決定する。次に、操作領域の 16 ブロックについて、ブロックが存在する場合、普通のブロック、宝石ブロック、排他的なブロックの中から乱数を用いて決定する。この割合はレベルによって異なり、高いレベルでは排他的なブロックが増え、スタックの扱いに制約が生じることで難易度がより上昇し、そのうえ見た目が派手になる仕組みである。

4.2.4.16 ブロック背景色の生成：selcolor

一部のブロックに用いられる背景色を決定する。2 つの乱数を用いて、色相環上にある色からランダムに選択される。返り値は 32bit のカラーコードであり、ブロックの種別を表す下位 8bit と組み合わせて使用される。

4.2.4.17 体力バー背景色の生成：barcolor

体力を表すバーの背景色を決定する。残り体力によって青、シアン、緑、黄、赤、黒と滑らかに変化する。

4.2.4.18 予告キューの初期化：gen_i

ゲーム開始時にキューを充填するために呼び出される。

4.2.4.19 ブロック群の出発：dept

ブロック群をキューの先頭から制御中の領域へ移動する。隠された最後尾のキューが空の場合、新たに生成する。そうでない場合、そのまま取得する。

4.2.4.20 スコアの加算と回復：calc_score

消去されるブロックの種類に応じて得点を加算し、体力を回復する。ブロックの種類はビットマスクをかけてカラーコード部分を読まないようにする。回復後の体力が上限を超過するとき、上限の値にクランプする。

4.2.4.21 ペナルティ：calc_dmg

着地したブロックが不可視の領域、すなわち上 4 行に存在するとき、不可視の領域にあるブロックと可視の領域にある空白を含むすべてのブロックの種類に応じてダメージを受ける。体力が 0 以下になったとき、ゲームオーバーとして状態遷移を行う。

4.2.4.22 保持操作：push

操作中のブロック群をスタックに保存する。専用のカウンタを用いて格納する場所を決める。投入されたブロックの中に爆弾があった場合、ブロック群の中のブロックを全て残骸に置き換え、爆弾の数に応じたダメージを受ける。発見されていない宝があった場合、発見された宝石に変化する。投入後は、次のブロック群をキューから取得するため、出発状態に遷移する。スタックが埋まっている場合、操作中のブロック群と直接交換される。ただし、交換後にブロックが存在できる条件を満たさない場合、交換は無効となる。

4.2.4.23 使用操作：pop

保持しているブロック群のうち、最後に格納したものを取り出す。スタックが空の場合、または取り出すブロック群が存在できる条件を満たさない場合、操作は無効となる。操作中のブロックは、隠されたキューが未使用のとき、そこに移動する。そうでなければ、取り出すブロック群と交換する。スタックに爆弾および発見されていない宝が移動した場合、保持操作と同様の処理を行う。

4.2.4.24 行の削除と移動：del_drop

空白の存在しない行を削除し、下に詰める。下の行から順に移動距離を計算し、必要に応じて行ごと移動する。また、移動を完了したあとで上 4 行にブロックが残っている場合、ダメージを受ける処理を行うフラグを立てる。ブロックが着地した時点で上 4 ブロックにブロックが存在した場合でも、移動後にすべてのブロックが見えていればダメージを受けないが、これは瀬戸際を演出するための、意図的なものである。

4.2.4.25 操作対象のブロック群の盤面からの削除：clr_ctrl

判定または描画等のため盤面へ一時的に描画された、着地していない操作対象のブロック群を消去する。

4.2.4.26 頂点座標の計算：ipt

盤面を描画するための座標を計算するためのサブルーチンである。

4.2.4.27 テキストの描画：rnt

テキストファイルを読み出し、仮想画面に文字列を描画する。テクスチャとして用いられる。

4.2.4.28 操作対象のブロック群の盤面への転写：dr_ctrl

判定または描画のため一時的に、または着地のために、盤面に操作対象のブロック群を描画する。

4.2.4.29 盤面の描画：dr_board

頂点座標，ブロック情報をもとに，盤面を描画する．

4.2.4.30 削除対象の行へのエフェクト描画：dr_del

削除対象の行を暗く強調する．後から追加された機能であり，モックアップにおいてはテクスチャの余り部分を適当に加工して使用している．

4.2.4.31 統計情報の表示：dr_seg

統計情報の各桁をテクスチャの座標に変換し，セグメントを 0 から 9 および消灯の状態にする．

4.2.4.32 統計情報の非表示：dr_seg2

何も表示しないとき，dr_seg の代わりに呼び出され，消灯したセグメントを表示する．

4.2.4.33 メニュー画面の描画：dr_menu

状態に応じて文字列や画像を表示する．一部の座標はここで求められる．

5 評価（モックアップ実装まで）

本プロジェクトでは，実装工程における注意力，視野の変化が与える影響を分かりやすくするため，グローバル変数，整数型変数に固執した状態管理等，不具合のもとになりやすい要素を意図的に多く採用した．その結果として，制限時間がシビアでなく，未知の技術を用いないモックアップの実装を行う工程までの，数千行程度のプログラムであれば，かろうじてプロジェクトを破綻させないことが確認できた．ただし，各機能を十分に分割し，コーディングからテストまでの区間を短くする必要がある．複合的な要因による不具合は多くの場合，解決までに非常に長い時間を要する．筆者の主観にはなるが，同時に許容される原因の数は 3 個程度までであり，その数に対して指数関数的に増加するように体感できた．

また，コーディングの実力を確認する目的のプロジェクトでありながら，実際に要した時間の大半は機能の検討や修正，素材の調達やライセンスの調査等，コーディング以外の部分であった．これらのタスクはソフトウェア工学の教科書にも記載されているものであり，実際の業務に活用することが期待できる．

ゲームに新要素を追加したことで生じたゲームバランスの調整は，ブロックの種類を増やす，ライフ制の導入，キューおよびスタックの拡大によって，単調な操作によってクリアできてしまう，あらゆる操作をもってもゲームオーバーになってしまう等，極端な状況が発生することを防ぐことに成功した．

6 まとめ

前編では，本来の目的である低いレイヤーの制御を含むシステムを制作するための準備

段階として、企画からモックアップの実装までの工程を完了した。新たな要素を追加したことで発生したゲームバランスに関する疑問は現段階において概ね解決することができた。

付録 A：スクリーンショット

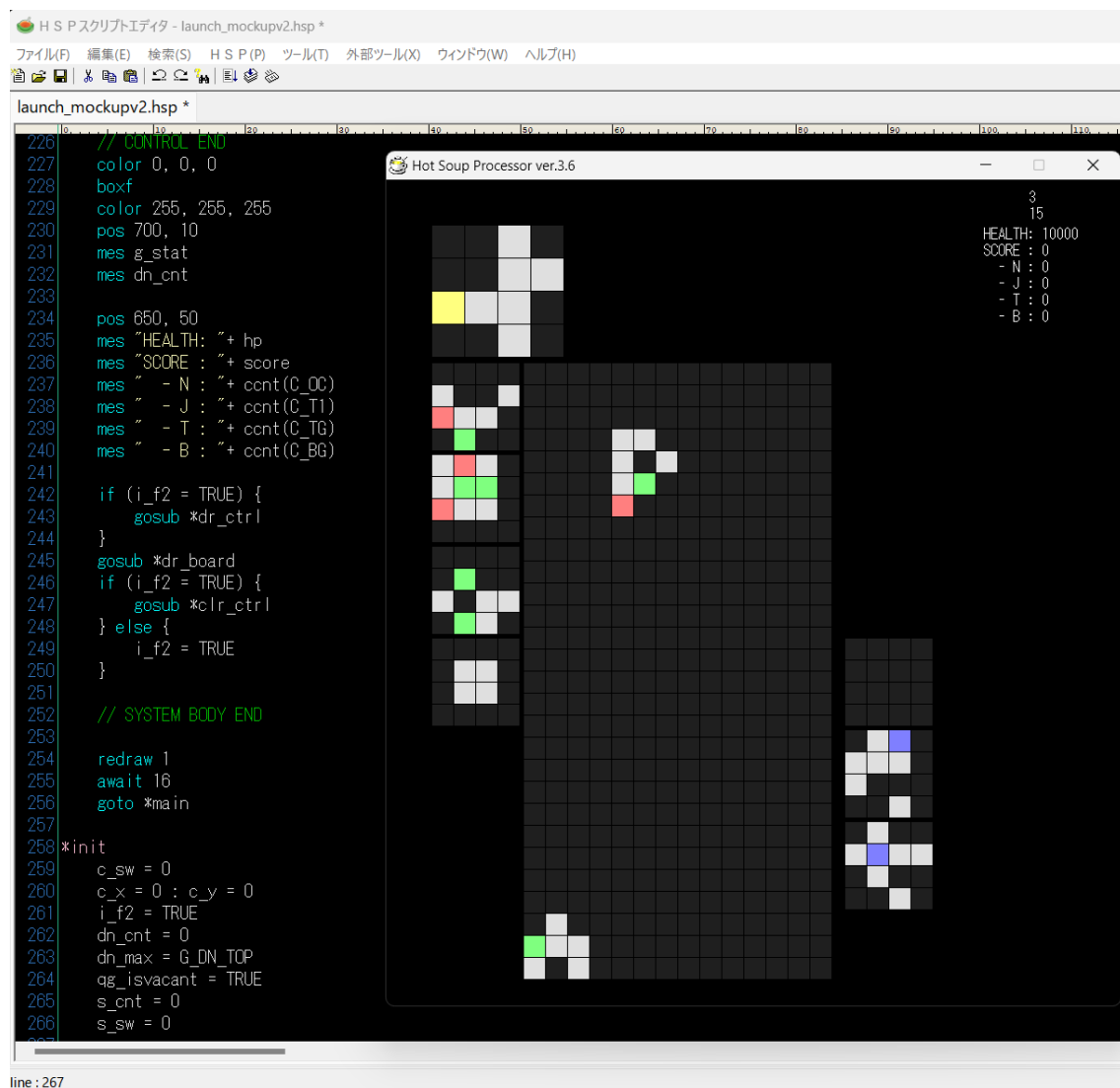


図 2 制作初期の Shaft

