Title	オブジェクト指向方法論を用いたテレビゲームソフトウェアの開発実験
Author(s)	三浦，陽平
Citation	
Issue Date	2004-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1808
Rights	
Description	Supervisor:片山 卓也，情報科学研究科，修士

修 士 論 文

オブジェクト指向方法論を用いた
テレビゲームソフトウェアの開発実験

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

三浦 陽平

2004 年 3 月

修 士 論 文

オブジェクト指向方法論を用いた テレビゲームソフトウェアの開発実験

指導教員　片山卓也 教授

審査委員主査　片山卓也 教授
審査委員　二木厚吉 教授
審査委員　落水浩一郎 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

210090 三浦 陽平

提出年月: 2004 年 2 月

概 要

本稿では，オブジェクト指向方法論を用いてテレビゲームソフトウェア開発に関する設計と実装を行うことにより，テレビゲームソフトウェア開発へのオブジェクト指向方法論の適用の有用性を調査，問題点や改善案を提案した．

目 次

第 1 章　はじめに **1**
1.1　研究の背景 1
1.2　研究の目的 1
1.3　開発実験の流れ 1
1.4　論文の構成 3

第 2 章　計算機支援環境 F-Developer **4**
2.1　F-Developer 4
2.1.1　F-Developer の操作法 5
2.2　モデルエディタ 6
2.2.1　基本クラスモデル 6
2.2.2　基本クラスモデルの作成法 7
2.2.3　基本ステートチャートモデル 8
2.2.4　基本ステートチャートモデルの作成法 8
2.3　F-Prototyper 10
2.3.1　スクリプトエディタ 10
2.3.2　スクリプトエディタの使用法 10
2.3.3　アニメータ 14
2.4　F-Developer の使用例 15

第 3 章　実装環境と開発対象ソフトウェアの仕様 **22**
3.1　実装環境 22
3.2　シリアル通信 24
3.3　セグメント方式によるメモリ管理 24
3.4　仕様 25
3.5　コントローラと操作方法 26
3.6　アルゴリズム 27

第 4 章　オブジェクト指向方法論を用いないソフトウェア開発 **29**
4.1　オブジェクト指向方法論を用いない開発の流れ 29
4.2　従来の手法での設計 30

4.3 従来の手法での実装 31

第 5 章 オブジェクト指向方法論によるソフトウェア開発 33

5.1 オブジェクト指向方法論を用いた開発の流れ 33
5.2 オブジェクト指向分析 34
5.3 オブジェクト指向設計 38
5.3.1 クラス図 38
5.3.2 状態遷移図 39
5.4 プロトタイプ実行 43
5.5 テスト 49
5.6 オブジェクト指向方法論を適用した実装 52
5.6.1 実装手法 52
5.6.2 実装内容 53
5.6.3 実装結果 55
5.6.4 状態の確認 56

第 6 章 おわりに 59

6.1 まとめ 59
6.2 評価 59
6.3 結論 60
6.4 今後の展望 60

図 目 次

1.1 開発実験の流れ 2

2.1 F-Developer の概要 4
2.2 F-Developer の動作モデル 5
2.3 F-Developer 6
2.4 クラスモデルの例 7
2.5 ステートチャートモデルの例 9
2.6 スクリプトエディタ 11
2.7 アニメータ 14
2.8 人と犬のクラス図 17
2.9 人の状態遷移図 17
2.10 犬の状態遷移図 18
2.11 人と犬のスクリプトエディタ 20
2.12 人のアニメータ 21
2.13 犬のアニメータ 21

3.1 実装の流れ 23
3.2 画面構成 24
3.3 WonderSwan のキー配置 27

4.1 オブジェクト指向方法論を用いない開発の流れ 29
4.2 フローチャート 30
4.3 オブジェクト指向方法論未使用時のゲーム画面 32

5.1 オブジェクト指向方法論を用いた開発の流れ 33
5.2 クラス図 38
5.3 My の状態遷移図 39
5.4 Block の状態遷移図 40
5.5 Pow の状態遷移図 40
5.6 Touch の状態遷移図 41
5.7 Disp の状態遷移図 41
5.8 Cont の状態遷移図 42

5.9 オブジェクトとアウトプットイベントの流れ 44
5.10 プロトタイプ実行 46
5.11 オブジェクト m のアニメーション 46
5.12 オブジェクト c のアニメーション 47
5.13 接触判定のプロトタイプ実行 47
5.14 オブジェクト m のアニメーション 48
5.15 オブジェクト t のアニメーション 48
5.16 My のテスト 51
5.17 オブジェクト指向方法論使用時のゲーム画面 55
5.18 プロトタイプ実行からの流れ 56
5.19 PC 側の状態の受信 57
5.20 設計段階の状態遷移図 58

6.1 拡張例 61

第1章　はじめに

1.1　研究の背景

近年，コンピュータのハードウェアの進歩に伴い，テレビゲームハード用に供給されるソフトウェアは巨大化，複雑化している．大規模で複雑なソフトウェア開発に有効な手法として，オブジェクト指向方法論が挙げられる．しかしテレビゲームソフトウェア開発において，オブジェクト指向方法論を用いてシステムを分析，設計することは一般的ではない．

テレビゲームには多数のキャラクタやアイテムが登場する．それぞれのキャラクタやアイテムはパラメータを持ち，他のオブジェクトに対してアクションを起こす．これらはメンバ変数，メンバ関数として扱うことができ，オブジェクトとしてモデル化することが自然であると考える．また，テレビゲームソフトウェアの開発には多くの時間が必要であり，続編の作成や似た内容の作品の作成ではコードの再利用性も重要である．このようなテレビゲームの性格から，オブジェクト指向方法論がテレビゲームソフトウェア開発に有用であると考える．

1.2　研究の目的

本研究は，オブジェクト指向方法論を用いてテレビゲームソフトウェア開発を行うための具体的な手法を示すために，テレビゲームソフトウェアをオブジェクト指向方法論により開発し，テレビゲームソフトウェア開発におけるオブジェクト指向方法論の有効性の調査を行うことを目的とする．

本研究では，テレビゲームソフトウェア開発においてオブジェクト指向方法論による開発実験を行う．実験後，この開発手法の導入により開発の負荷の緩和，時間の節約に役立つかなどの評価を行う．これによりゲームソフトウェア開発におけるオブジェクト指向方法論の有効性を示すことができると期待できる．

1.3　開発実験の流れ

本開発実験の流れを説明する．

本実験では最初に使用する機器とそのシステムの調査を行う．具体的には，オブジェ

クト指向方法論を用いた開発の設計とプロトタイプ実行の際に使用する，計算機支援環境 F-Developer についてと，実装時のターゲットマシンである，家庭用携帯ゲームハード WonderSwan(©BANDAI CO.,LTD.) についてである．本論文では第 2 章と第 3 章の前半で触れる．

次に開発するソフトウェアの対象を決定する．作成するゲームソフトウェアの内容を決定し，おおまかなシステムの分析を行う．本論文では第 3 章の後半に示す．

次に，オブジェクト指向方法論を用いずにソフトウェアの開発実験を行う．図 1.1 の左側の線である．オブジェクト指向方法論を適用せず従来の方法で，対象を WonderSwan への実装まで行う．これによりオブジェクト指向方法論適用時との比較が容易になる．本論文では第 4 章に記す．

次に，オブジェクト指向方法論を用いた開発実験を行う．図 1.1 では右側の線である．分析をした後，計算機支援環境 F-Developer にて設計を行う．この設計において記述，定義された情報はそのままプロトタイプ実行時に使用される．プロトタイプ実行とテストを行い，その後 WonderSwan へ実装する．実装後，設計したモデルと実装したものが対応した動作をするのかを確かめる．本論文では第 5 章にて扱う．

最後にこれらの開発実験から考察を行った．本論文では第 6 章にまとめる．

本実験の流れを図 1.1 に示す．

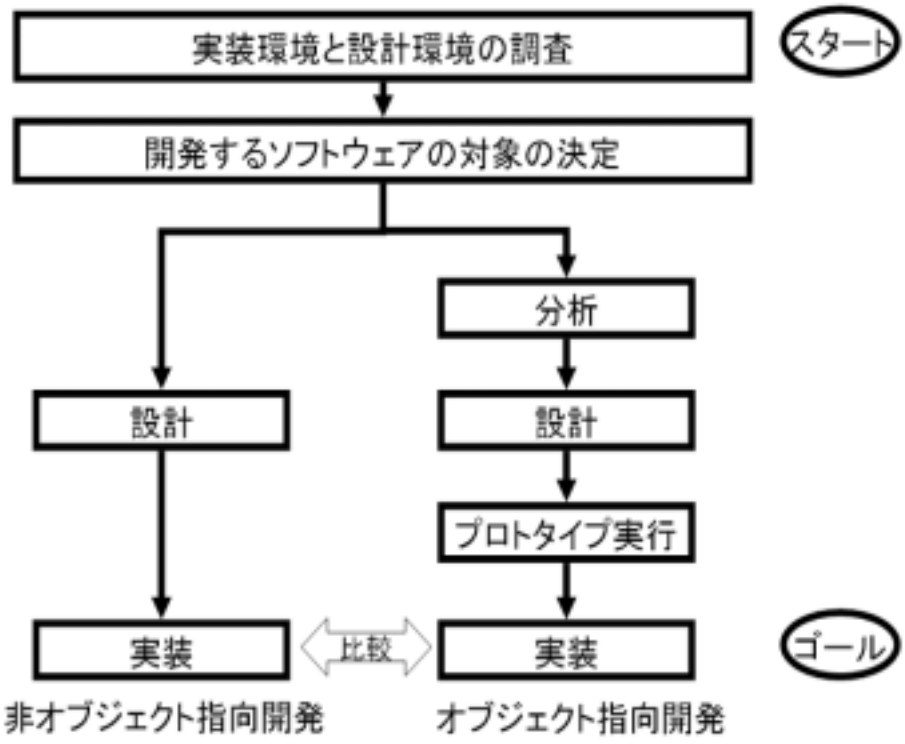

図 1.1: 開発実験の流れ

1.4 論文の構成

第 2 章ではオブジェクト指向方法論を用いた開発の設計とプロトタイプ実行に使用する CASE ツール，計算機支援環境 F-Developer についての説明を行う．

第 3 章では実装に使用する家庭用携帯ゲームハード WonderSwan についての説明と，実際に作成するソフトウェアの仕様や操作方法などを定める．

第 4 章ではオブジェクト指向方法論を用いずに，従来の方法でソフトウェアを作成する過程である，設計と実装について示す．

第 5 章ではオブジェクト指向方法論に基づき，テレビゲームソフトウェアを作成する過程である，分析，設計，プロトタイプ実行，実装について示す．

第 6 章は本論分のまとめである．

第2章　計算機支援環境 F-Developer

オブジェクト指向方法論によるソフトウェア開発の，設計に用いる CASE ツール，計算機支援環境 F-Developer について紹介する．

2.1　F-Developer

計算機支援環境 F-Developer はオブジェクト指向分析環境である．プログラミング言語 Java，ML，定理証明器 HOL により実装されており，3 つのサブシステムにより構成されている．1 つは UML のクラス図と状態遷移図を記述するモデルエディタ，2 つめに，モデルエディタに記述，定義したクラス図と状態遷移図から ML プログラムを自動生成し，ML インタプリタ上でオブジェクトを動作させてプロトタイプ実行を行う F-Prototyper，3 つめに，定理証明器 HOL を用い検証を行うシステム F-Verifier からなる．図 2.1 に F-Developer のモデルエディタと F-Prototyper の概要について示す．

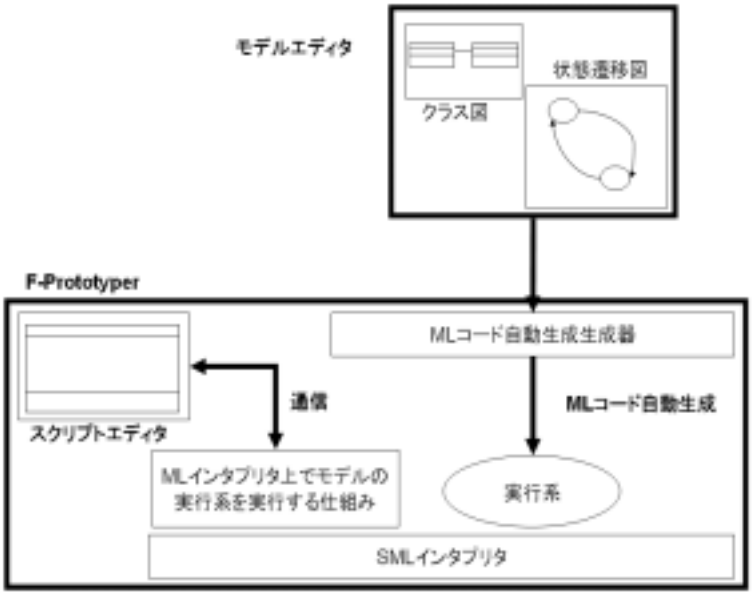

図 2.1: F-Developer の概要

2004 年 2 月現在，F-Developer には OS:Solaris，CPU:Sparc 用のものと OS:Windows の

2 種類がある．本開発実験では前者の Solaris 版を使用する．

F-Developer のモデルエディタと F-Prototyper の動作モデルついて説明する．図 2.2 にクラス図と状態遷移図とオブジェクト間のイベント送信について示す．

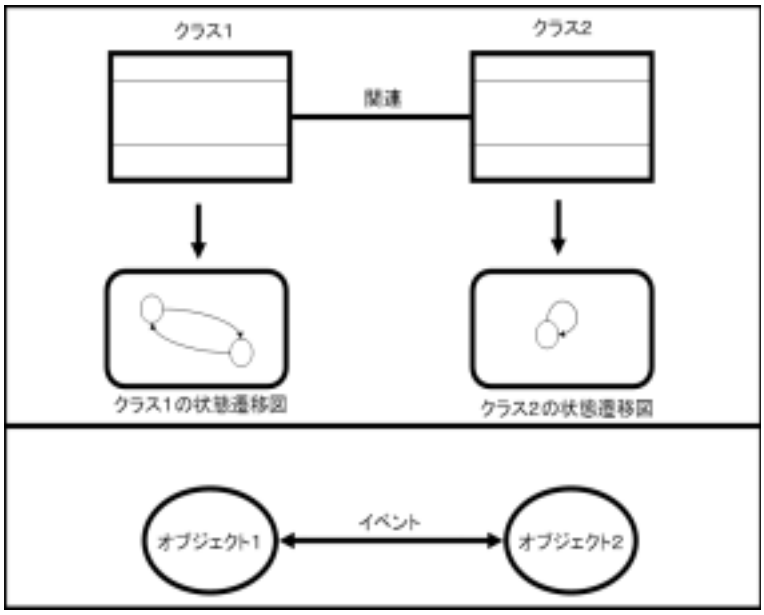

図 2.2: F-Developer の動作モデル

この図 2.2 の上の部分では，2 つのクラスとこれらのクラスが関連で結ばれていることを示す．これらのクラスからはそれぞれの状態遷移図が作成される．図の下部では，2 つのオブジェクトがイベントを送ったり，受け取ったりする様子が示されている．イベントは関連を通り，オブジェクトの状態が遷移を起こし動作する．この図のような仕組みを計算機支援環境 F-Developer では実現しており，モデルエディタでクラス図と状態遷移図を記述し，F-Prototyper でプロトタイプ実行ができる．

本開発実験では，モデルエディタと F-Prototyper の 2 つの機能を使用している．ここからモデルエディタと F-Prototyper についての説明と使用例を紹介する．

2.1.1 F-Developer の操作法

F-Developer 起動直後の操作法について解説する．上方に並んでいる機能のアイコンについて説明を行う．番号は左から何番目にあるかを示す．図 2.3 に起動直後の画面を示す．

1. model ファイルの保存
   現在編集している model ファイルを保存する．アイコンを選び保存場所を選び名前を入力すると保存できる．拡張子は”model”となる．

図 2.3: F-Developer

2. model ファイルの読み込み
   model ファイルを読み込む.
3. 新規作成
   新しくモデルを作成する.
4. モデルを閉じる
   現在編集しているモデルを閉じる.
5. モデルエディタ
   モデルエディタを起動する．基本クラスモデルと基本ステートチャートモデルを作成，編集する.
6. F-Prototyper
   プロトタイプ実行のための機能 F-Prototyper を起動する.
7. F-Verifier
   検証のための機能 F-Verifier を起動する.
8. F-Developer の終了
   F-Developer を終了する.

2.2 モデルエディタ

モデルエディタは UML のクラス図とステートチャート図を描画し，各クラスの持つ変数や状態，イベント，クラス間の関連，イベントの流れなどの定義を行うものである．以下にこれらのクラス図とステートチャート図のモデリングについて紹介する.

2.2.1 基本クラスモデル

モデルエディタにて構築するのが基本クラスモデルである．クラスモデルには GUI にて UML のクラス図と状態遷移図を記述し，それぞれのクラスや関連に名前，メンバ変数，メンバ関数，状態，イベントなどの定義をプログラミング言語 ML の型や，属性評価式にて

行う．ここで定義された情報は全て，後に紹介するプロトタイプ実行機能，F-Prototyper にて使用される．クラスモデルの例を図 2.4 に示す．

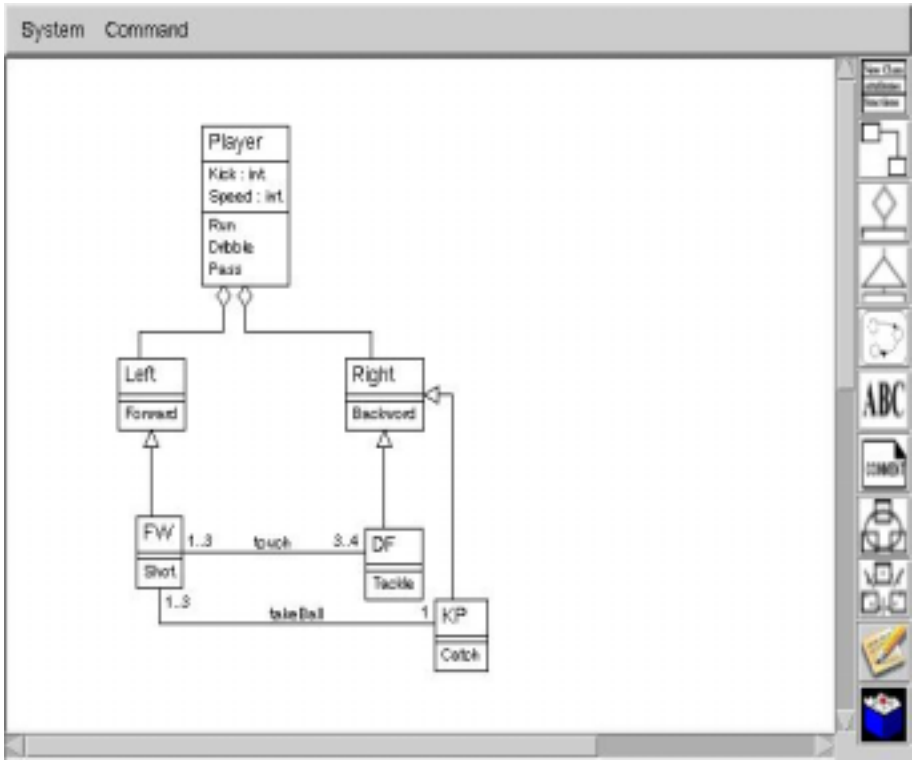

図 2.4: クラスモデルの例

2.2.2 基本クラスモデルの作成法

基本クラスモデルの作成について解説する．利用者は，モデルを作成するために図 2.4 の右側に縦に並んでいる，機能のアイコンをマウスで選択してクラスや関連などの記述を行っていく．アイコンの具体的な説明を行う．番号は上から何番目にあるかを示す．

1. クラス
   クラスを作成する．アイコンを選びクラスを作成する場所を選ぶと，クラス名を入力するウィンドウが開くのでここにクラス名を書き込む．詳細は 10 番目のアイコンで定義できる．
2. 関連
   クラス間の関連を作成する．選択したら，2 つのクラスを選ぶ．すると関連が作成される．詳細は 10 番目のアイコンで定義できる．
3. 集約

クラス間の集約を作成する．選択したら，2 つのクラスを選ぶ．すると集約が作成される．

4. 継承
クラス間の継承を作成する．選択したら，2 つのクラスを選ぶ．すると継承が作成される．
5. 状態
クラスの取りうる状態，遷移を定義する．クラスをひとつ選び，このアイコンを選ぶと 2.2.2 で紹介する基本ステートチャートモデルの定義をするウィンドウが開く．
6. 文字入力
任意の文字列をクラス図中に記述する．
7. コメント
任意の文字列をクラス図中の四角の枠の中に記述する．この枠にはこのコメントがどの部分を指すのかの線があり，この線は自由に動かすことができる．
8. グループ化
複数のクラスを，シフトキーを押しながら選択し，このアイコンを選ぶとこれらのクラスがグループ化される．
9. グループ化の解除
グループ化されているクラスを解除する．
10. 詳細
ひとつのクラスや関連を選び，このアイコンを選択するとそれぞれの詳細を定義できる．クラスの場合はそのクラスの持つ変数や関数，コメント，画面への表示の仕方などを選ぶことができる．関連の場合は関連の名前，各クラスの数などを選ぶことができる．
11. ごみ箱
クラス図中のクラスや関連を選択しこのアイコンを選択すると，それらのクラスや関連は削除される．

2.2.3 基本ステートチャートモデル

次に，クラスモデルで定義したクラスがどのような状態を取り得るのか，イベントやアクション，ガード条件や遷移などの記述を行う．ここで定義された情報もやはり F-Prototyper にて使用される．ステートチャートモデルの例を図 2.5 に示す．

2.2.4 基本ステートチャートモデルの作成法

基本ステートチャートモデルの作成について解説する．利用者は，クラスの状態遷移図を作成するために図 2.5 の右側に縦に並んでいる，機能のアイコンをマウスで選択して状態や遷移などの記述を行っていく．アイコンの具体的な説明を行う．番号は上から何番目

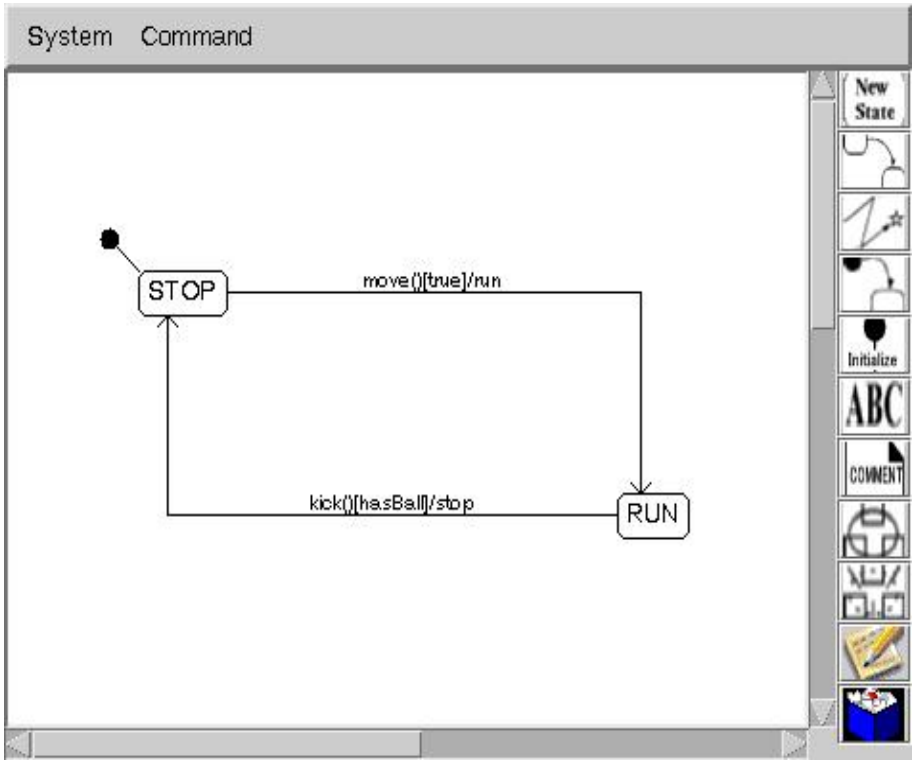

図 2.5: ステートチャートモデルの例

にあるかを示す.

1. 状態
   状態を作成する．選択したら作成する場所を決め，名前を入力する．詳細な定義は10番目のアイコンで決定する.
2. 遷移
   選択した後，2つの状態を選び，遷移を作成する．この詳細な定義は10番目のアイコンで行う.
3. イベント
   このクラスに起こるインプットイベントと，他のクラスへ渡すアウトプットイベントを定義する．他のクラスで定義されている同じイベントがあるならば，これを読み出して定義することができる．イベントが起こりその際に授受する仮引数も定義する.
4. 初期状態
   はじめにどの状態から遷移が始まるのかを決める．ひとつの状態を選んでからこのアイコンを選択する.
5. 初期化
   はじめの状態で，オブジェクトの持つ変数がどのような値を取っているのかを定義する.

6. 文字入力
   任意の文字列をステートチャート図中に記述する．
7. コメント
   任意の文字列をステートチャート図中の四角の枠の中に記述できる．この枠にはこのコメントがどの部分を指すのかの線があり，この線は自由に動かすことができる．
8. グループ化
   複数の状態をシフトキーを押しながら選択し，このアイコンを選ぶとこれらの状態がグループ化される．
9. グループ化の解除
   グループ化されているクラスを解除する．
10. 詳細
    ひとつの状態や遷移を選び，このアイコンを選択するとそれぞれの詳細を定義できる．遷移の場合はインプットイベント，アウトプットイベント，ガード条件，アクションなどが定義できる．
11. ごみ箱
    ステートチャート図中の状態や遷移を選択しこのアイコンを選択すると，それらの状態や遷移は削除される．

2.3 F-Prototyper

F-Prototyper は，モデルエディタにて構築したモデルのプロトタイプ実行を行うものである．モデルエディタで構築されたモデルは自動的に ML プログラムに変換される．F-Prototyper は，これから紹介する 2 つの機能により，構築された分析モデルの振る舞いを確認するツールである．

2.3.1 スクリプトエディタ

生成された ML のソースコードにはオブジェクトの実体化やオブジェクトの振る舞いを定義した関数が含まれている．構築されたモデルは，SML インタプリタ上でシミュレーションされ，F-Prototyper のスクリプトエディタ上で対話的にプロトタイプ実行を行うことができる．スクリプトエディタを図 2.6 に示す．

2.3.2 スクリプトエディタの使用法

スクリプトエディタの使用法について記述する．スクリプトエディタの画面は図 2.6 のようになっており，中央下部にはアイコンが並んでいる．基本的にはこのアイコンを使いモデルエディタで定義された情報を引き継いで使うことになる．アイコンには押下するこ

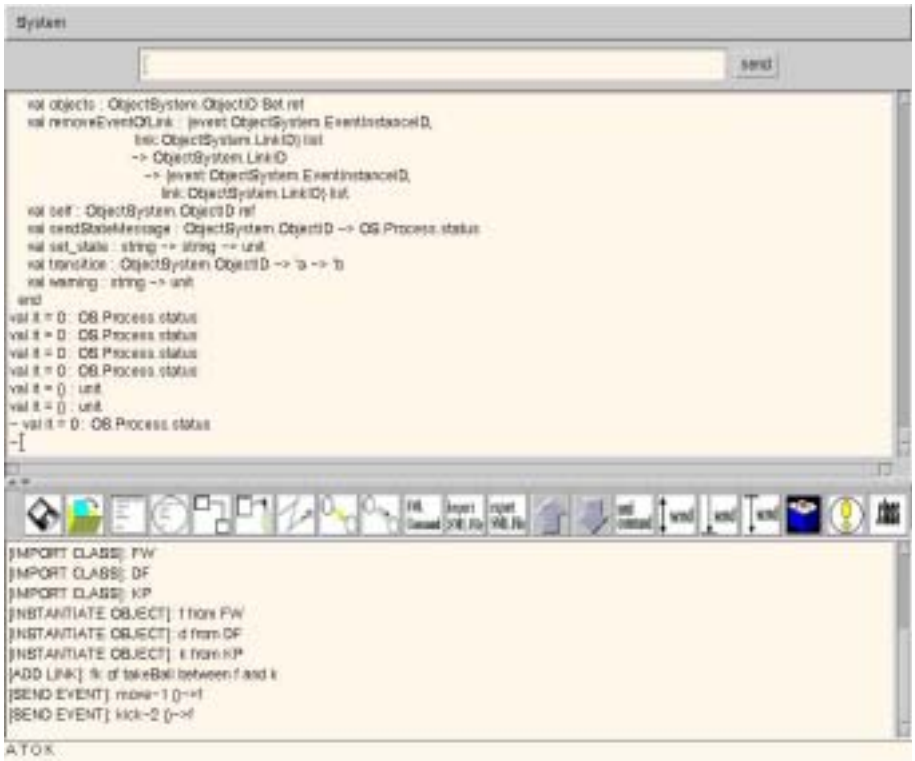

図 2.6: スクリプトエディタ

とですぐに効果が出るものと，下段の実行系列に送られるものがある．下段にあるコマンドは実行する種類のアイコンによって実行される．ここで，アイコンについて解説する．番号は左から何番目にあるかを示す．

1. coms ファイル保存
   現在の実行系列を coms ファイルに保存する．このアイコンの下段には実行できるコマンドが羅列されるが，この列を保存することができる．その際，ファイルの拡張子が”coms”となる．このファイルを作成することで，プロトタイプ実行の効率が大幅に良くなる．
2. coms ファイル読み込み
   保存されている coms ファイルを読み込む．読み込んだ内容は下段に表示される．
3. クラス読み込み
   モデルエディタで作成したクラスを読み込む．これを読み込むと，このクラスから作られるオブジェクトを生成できる．
4. イベント読み込み
   モデルエディタで作成したイベントを読み込む．これを読み込むと，イベントを実行する準備ができる．
5. 関連読み込み

モデルエディタで作成したクラス間の関連を読み込む．関連を読み込むと複数クラス間でのイベントのやり取りができるようになる．

6. オブジェクト作成
   読み込んでおいたクラスから，実体であるオブジェクトを生成する．オブジェクトには名前をつける．
7. イベント作成
   読み込んでおいたイベントから，起こすイベントの詳細，どのオブジェクトに送るかを決める．
8. 同期作成
   読み込んでおいた関連から，関連の実体を生成する．これには名前をつける．
9. 同期実行
   同期を作成した後に実行できる．実際に同期を行う．
10. SML コマンド実行
    SML のコマンドを実行する．
11. SML ファイル読み込み
    保存されている SML ファイルを読み込む．
12. SML ファイル保存
    SML インタプリタに表示されているものをファイルに保存する．拡張子は”sml”である．
13. コマンドを上へ移動
    選択されているコマンドをひとつ上のコマンドと入れ換える．
14. コマンドを下へ移動
    選択されているコマンドをひとつ下のコマンドと入れ換える．
15. コマンドを実行
    選択されているコマンドを実行する．
16. 全てのコマンドを実行
    実行系列の全てのコマンドを上から順に実行する．
17. コマンドの最初から選択された行までを実行
    実行系列にあるコマンドのうち，最初から選択されているコマンドまでを実行する．イベントや，同期を起こす前までをこのアイコンで実行してアニメータを起動，プロトタイプ実行を開始というパターンが多い．
18. 選択されたコマンドの行のひとつ下から最後まで実行
    コマンド部分にあるコマンドのうち，現在選択されている行のひとつ下から最後まで実行する．
19. ごみ箱
    現在選択されているコマンドを削除する．
20. アニメータ
    プロトタイプ実行の要であるアニメータを起動する．表示するオブジェクトを選択

する．

21. 外部と連携
    F-Prototyper には外部のプログラムと連携してプロトタイプ実行を行う機能がある．このために使用する．

ここでは直接，人が SML のコードを実行することにより，オブジェクトの変数の値を変更したり，クラスの状態を変更したりできる．また，SML のプログラムによって容易にテストスクリプトを作成することもできる．これによりモデルの定義を変えずに，複雑なプロトタイプ実行ができる．以下に SML のコードを直接実行するときに使う書式をいくつか挙げる．

- 変数の変更
  オブジェクトの変数を変更するときに使用

  ```
  Class_クラス名.set_変数名 "オブジェクト名" 数値;
  ```

  ここでクラス名，変数名，オブジェクト名にそれぞれの名前を入力する．数値のところは値を書き込むが，マイナスの数値の場合は，”-”ではなく，”~ ”と書く．これは以下の場合も同じである．

- 変数の表示
  ML インタプリタ上でオブジェクトの変数の値を表示するときに使用

  ```
  Class_クラス名.lookup_変数名 "オブジェクト名";
  ```

- イベント送信
  オブジェクトにイベントを送信するときに使用

  ```
  Class_クラス名.transition "オブジェクト名"
   [{event = "イベント名", id = idナンバー}];
  ```

- ループ
  何度も同じ処理を行うときに使用すると自動的にプロトタイプ実行が進行

  ```
  int i = ref 0;
  while !i < 回数 do (Class_クラス名.transition "オブジェクト名"
   [{event = "イベント名", id = idナンバー}]; i := !i + 1);
  ```

  これは"回数"分，処理を繰り返す例である．

- クラス生成
  クラスを作成する

```
newObject_クラス名()
```

ここに挙げた他にもさまざまなコマンドを実行できる．ML コマンドは F-Prototyper が自動生成するが，人が直接 ML コマンドを実行することで柔軟なプロトタイプ実行ができる．

2.3.3 アニメータ

アニメータは，SML インタプリタ上の作業の結果を蓄積することにより，現在の各オクジェクトの状態や属性変数などを GUI で表示する機能である．SML インタプリタで構築された結果は，JNI(Java Native Interface) を使用してアニメータに渡され，アニメータが GUI に表示する．アニメータを図 2.7 に示す．

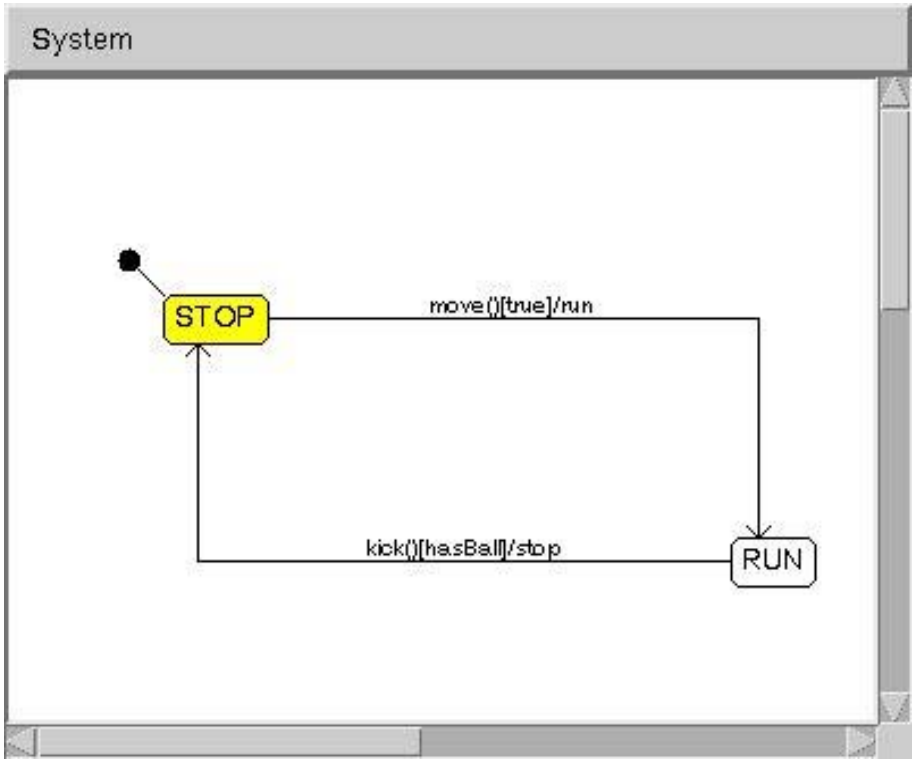

図 2.7: アニメータ

ひとつのオブジェクトにつきアニメータをひとつ表示することができる．スクリプトエディタのコマンドを実行することでオブジェクトにイベントが送られ，アニメータが動作する．ガード条件が満ちていれば定義された遷移をし，また，変数を変える．あるオブジェクトにイベントを直接送らなくても，他のオブジェクトのアウトプットイベントが起こりオブジェクト間の関連を通り，同期してイベントが起こることもあり，このパターンのほうが多い．

2.4 F-Developer の使用例

F-Developer の使用法を簡単な例を挙げて説明する．この節で示す F-Developer の例は飼い主である人間と飼われている犬である．犬は勝手に走るが，飼い主が声をかけると止まる．ここで人間と犬をそれぞれクラスにする．

- Man
  属性: 命令する

- Dog
  属性: 速度，走る

この手順を以下に示す．説明文中の F-Developer と F-Prototyper のアイコンについては”2.1 F-Developer”，”2.2 モデルエディタ”と”2.3 F-Prototyper”を参照されたい．説明文中のアイコンを示す数は画面右側のアイコンを上から数えたときの順番である．この詳細は ”2.2.2 基本クラスモデルの作成法” と”2.2.4 基本ステートチャートモデルの作成法”を参照されたい．

1. F-Developer
   F-Developer を起動する．
   左から 5 番目のアイコンでモデルエディタを起動すると基本クラスモデル作成画面になるので，クラス図の記述を行う．
2. クラス図

   (a) クラスを作成
   F-Developer の 1 番目のアイコンを選択し，クラスの位置と名前，ここではクラス”Dog”を作成する．もうひとつのクラス”Man”も作成する．
   (b) クラスの詳細を決定
   ひとつクラスを選び，10 番目のアイコンを選択してクラスの変数と関数を作成する．Dog の属性として”speed”と”Run”を作成する．ここでは関数の中身も定義できるが，ここでは簡単のため省略した．Man は属性として”Order”を定義する．また，クラス図の属性を表示する型も選べる．ここでは HOL のタイプは全く使用しないので，ML のみ表示することにした．
   (c) 関連の作成
   Man と Dog の間に関連を作成する．
   2 番目のアイコンを選択し，Man と Dog を結ぶ．
   (d) 関連の詳細を決定
   関連を示す線を選択し，10 番目のアイコンでこの関連の名前を決定する．名前は MandD とした．ここでは各クラスの個数も決定できるが，ここでは 1 対 1 のままにする．

3. 状態遷移図

状態遷移図を作成する

ひとつクラスを選び，5 番目のアイコンを選択すると基本ステートチャートモデル作成画面になる．最初に作成するのは Man の状態遷移図であるが，Dog についても同様である．

(a) 状態の作成

状態を作成する．

1 番目のアイコンを選択し，状態の位置と名前，ここでは状態名を”LOOK”とした．

(b) 初期状態の作成

作成した状態をひとつ選び，4 つ目のアイコンを選択すると，この状態が初期状態として設定される．

(c) 初期値の設定

5 番目のアイコンを選択し，edit ボタンを押下すると，初期化するときの名前と初期化する式を書くことが出来る．ここでは何もしない．Dog クラスの状態遷移図の場合は，名前を”Init”とし初期化する式は”speed := 0”とする．これで Dog の属性 speed は 0 に初期化される．

(d) イベントの作成

3 番目のアイコンを選択し new ボタンを押下すると，新しくインプットイベントが作成できる．Man では”order”というイベントを作成した．ここではイベントの授受の際に渡す引数を決めることも出来る．この例では使用しない．また，アウトプットイベントの作成ではどの関連を通ってイベントを流すかを選択する．”inst”というイベントを作成し，関連 MandD を選択した．Dog では Man で左記に作成されているイベントを import ボタンによってコピーできる．

(e) 遷移の作成

ひとつの状態を選択し，2 番目のアイコンを押下すると，遷移を作成できる．遷移元を選択してから遷移先を選ぶ．

(f) 遷移の詳細の決定

ひとつの遷移を選び，10 番目のアイコンを選択する．インプットイベントを先ほど作成したイベントの中から選ぶ．ここでは order を選んだ．アクションの edit ボタンを選ぶとアクション式が定義できる．Dog では”speed := 10”として属性の値を変更するようにした．コンディションのボタンを選ぶとガード条件を定めることが出来る．アウトプットイベントの add ボタンを押すと，作成したイベントをアウトプットできる．Man には”inst”イベントをアウトプットイベントとした．

クラス図を図 2.8 に示す．

人の状態遷移図を図 2.9 と犬の状態遷移図を図 2.10 に示す．

プロトタイプ実行の様子を以下に示す．

この手順を以下に示す．説明文中のアイコンを示す数は画面右側のアイコンを上から数えたときの順番である．この詳細は 2.3 節，F-Prototyper の説明を参照のこと．

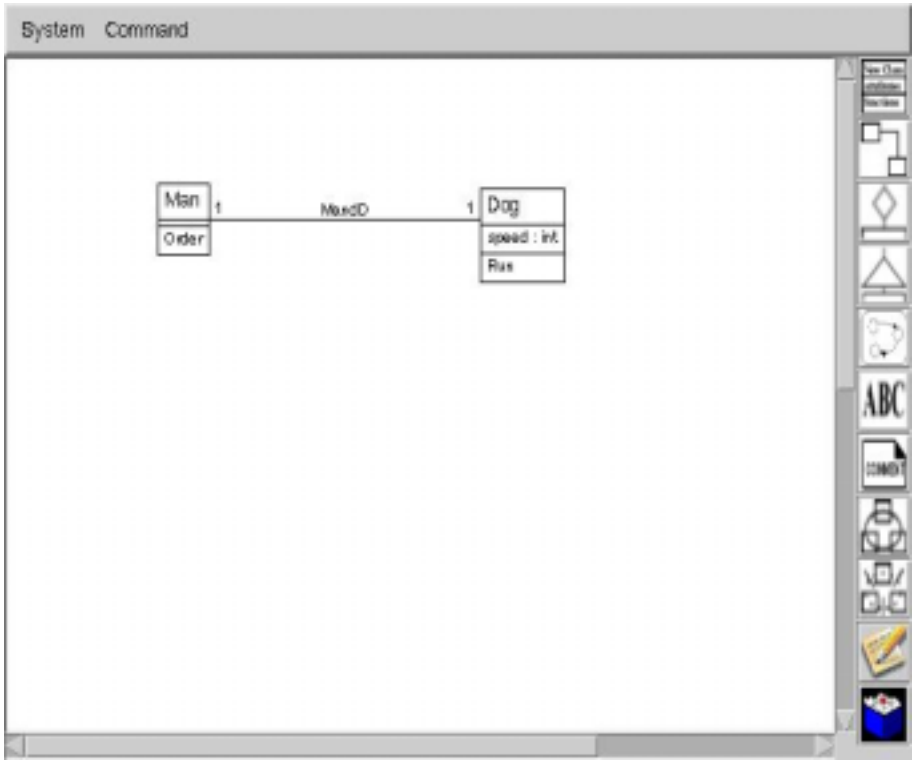

図 2.8: 人と犬のクラス図

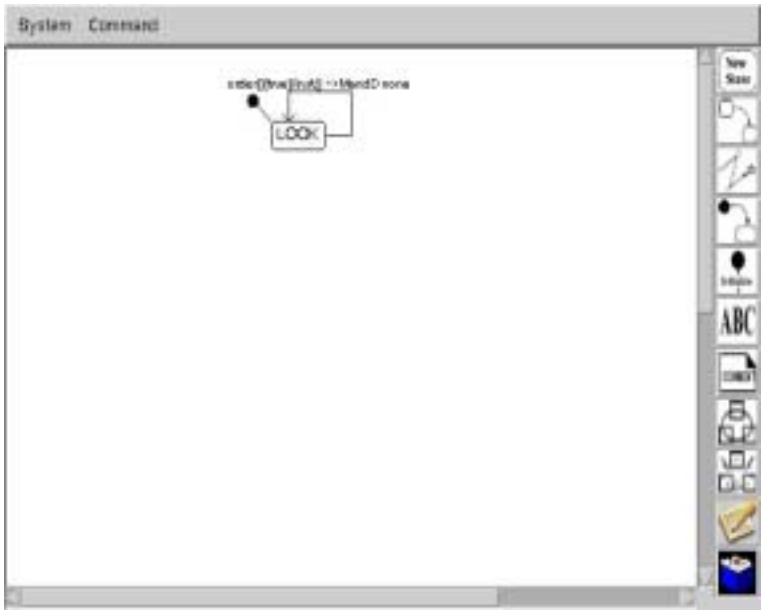

図 2.9: 人の状態遷移図

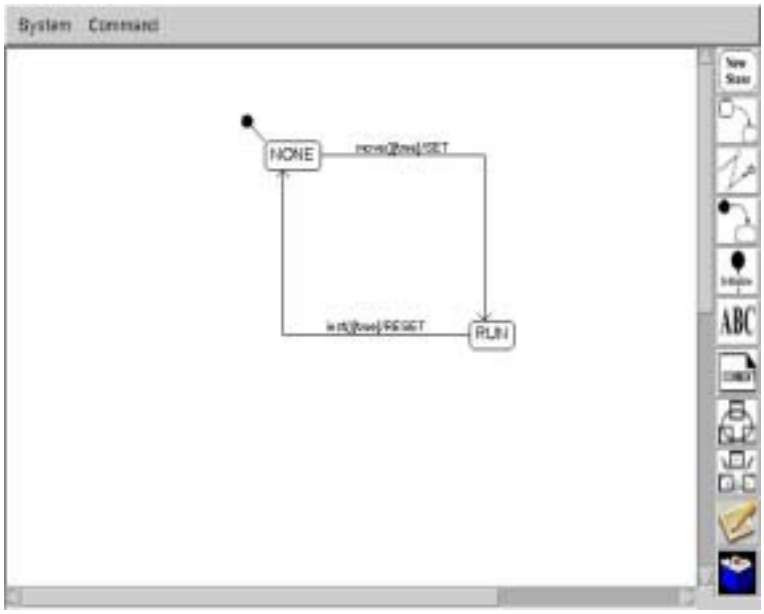

図 2.10: 犬の状態遷移図

1. F-Prototyper
   F-Prototyper を起動する．
   スクリプトエディタ画面になる．
   (a) イベント読み込み
   4 番目のアイコンを選び，読み込むイベントを全て読み込む．ここでは order，move，inst の 3 つである．読み込んだイベントは下段に表示される．
   (b) 関連読み込み
   5 番目のアイコンで関連を読み込む．ここでは MandD を選ぶ．
   (c) クラス読み込み
   3 番目のアイコンでクラスを読み込む．ここでは Man と Dog である．ここまでで一度 16 番目のアイコンで全てのコマンドを読み込み，ML インタプリタに渡しておく．読み込まないと，以下のことが出来ない．
   (d) オブジェクト作成
   6 番目のアイコンを選択し，読み込んだクラスのうちオブジェクトを生成するものを選び，名前をつける．Man から”m”，Dog から”d” というオブジェクトを生成した．ここでコマンドを直接選んで 15 番目のアイコンを選択していくか，17 番目または 18 番目などのコマンドを使い，今作成したオブジェクトを ML インタプリタに渡す．
   (e) リンク作成
   8 番目のアイコンで読み込んだ関連の名前を決定する．MandD から md という名前をつけた．

(f) イベント作成

7番目のアイコンでイベントを作成する．読み込んだイベントからひとつ選び，送るオブジェクトを選ぶ．newボタンを押すと，id番号とともにイベントが出来るので，選択してOKボタンを押す．ここでは

```
order-1()->m
move-2()->d
```

の２つを作成した．

(g) 同期の作成

関連のリンクを通りアウトプットイベントを送るために同期を作成する．9番目のアイコンを選び，送信元と送信先を選択する．ここではリンクmdから，"from m to d using md"を作成した．ここまで読み込んでいないコマンドは全て読み込む．画面上方の"System"から"Reboot SML"を選び，16番目のコマンドを実行して一気に読み込んでも良い．

2. プロトタイプ実行

(a) アニメータ

20番目のアイコンを選択し，オブジェクトを選んで，アニメータを起動する．mとdの両方を表示する．アニメータには属性の変数も表示される．Dogのspeedには基本ステートチャートモデルの作成の初期化で定義した0が入っている．

(b) イベント送信

スクリプトエディタの下段にあるコマンドの中から，

```
[SEND EVENT]: move-2()->d
```

を実行する．dの状態がNONEからRUNに遷移し，speedが基本ステートチャートモデルのアクション式で定義した通り，speedに10が設定される．

(c) イベント送信

スクリプトエディタの下段にあるコマンドの中から，

```
[SEND EVENT]: order-1()->m
```

を実行する．mのアニメータが一瞬動く．

(d) 同期送信

スクリプトエディタの下段にあるコマンドの中から，

```
[SYNCHRONIZATION]: from m to d using md
```

を実行する．mのアウトプットイベント"inst"によりdの状態がNONEに戻り，speedが基本ステートチャートモデルのアクション式で定義した通り，0に戻される．

(e) イベント送信

スクリプトエディタの下段にあるコマンドの中から，

```
[SEND EVENT]: move-2()->d
```

を実行する．dの状態がNONEからRUNに遷移し，speedに10が設定される．

(f) イベント送信

スクリプトエディタの下段にあるコマンドの中から，

```
[SEND EVENT]: move-2()->d
```

を実行する．d の状態が RUN のときにこのイベントを送っても変化しない．

スクリプトエディタを図 2.11 に示す．

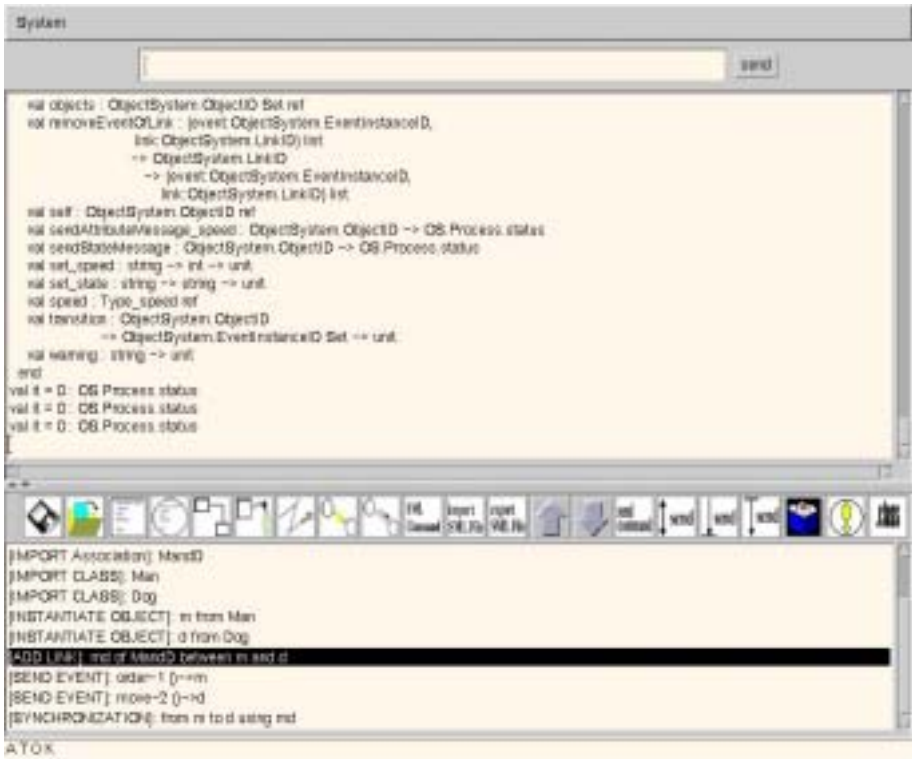

図 2.11: 人と犬のスクリプトエディタ

人のアニメータの様子を図 2.12 に，犬のアニメータの様子を図 2.13 に示す．

このようにして期待した状態遷移をするか，変数の値は正しいかということを確かめていく．簡単な例であったが，F-Developer の使用例を示した．

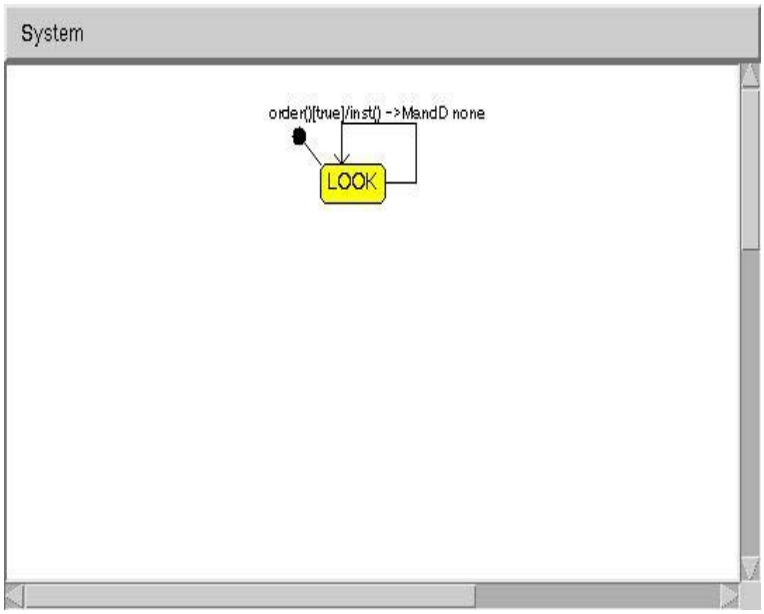

図 2.12: 人のアニメータ

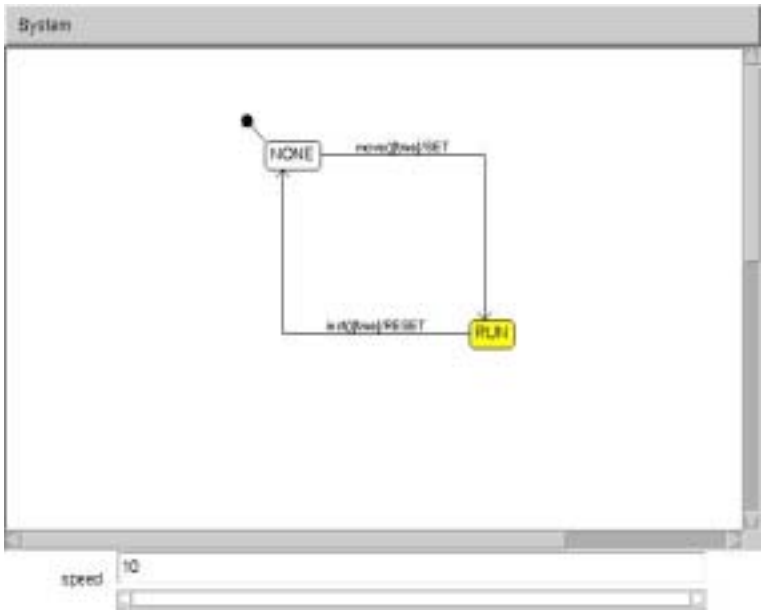

図 2.13: 犬のアニメータ

第3章　実装環境と開発対象ソフトウェアの仕様

本実験で使用する機器と，本実験で開発を行うソフトウェアの仕様を紹介する．

3.1　実装環境

オブジェクト指向方法論を用いた開発において，対象となるソフトウェアの仕様の設計を行うのに，CASE ツール，計算機支援環境 F-Developer を使用する．

設計後，ソフトウェアを実装するハードウェアとして，携帯ゲーム機 WonderSwan(株式会社バンダイ) をターゲットマシンとする．市販されている家庭用ゲームハードウェアに実装することにより，より現実的な評価ができる．

以下に実装環境について示す．

- WonderSwan ©BANDAI CO.,LTD.
  - CPU: 16 ビット, 3.072MHz
  - LCD パネル: 横 224 × 縦 144 ドット
- WonderWitch ©Qute Corporation ©BANDAI CO.,LTD.
  - Personal Software Development Kit
  - C 言語

携帯ゲームハード WonderSwan への実装の流れについて示す．開発者は PC 上のエディタでプログラミングを行い，このコードをコンパイルする．生成されたファイルを WonderSwan 側のカートリッジに転送する．送られたデータを WonderSwan で実行する．実装における開発の流れを図 3.1 に示す．

図 3.1 について解説する．

1. プログラミング
   PC 上のエディタにてプログラミングを行う．
2. コンパイル
   作成したプログラムコードをコマンドライン上でコンパイルする．

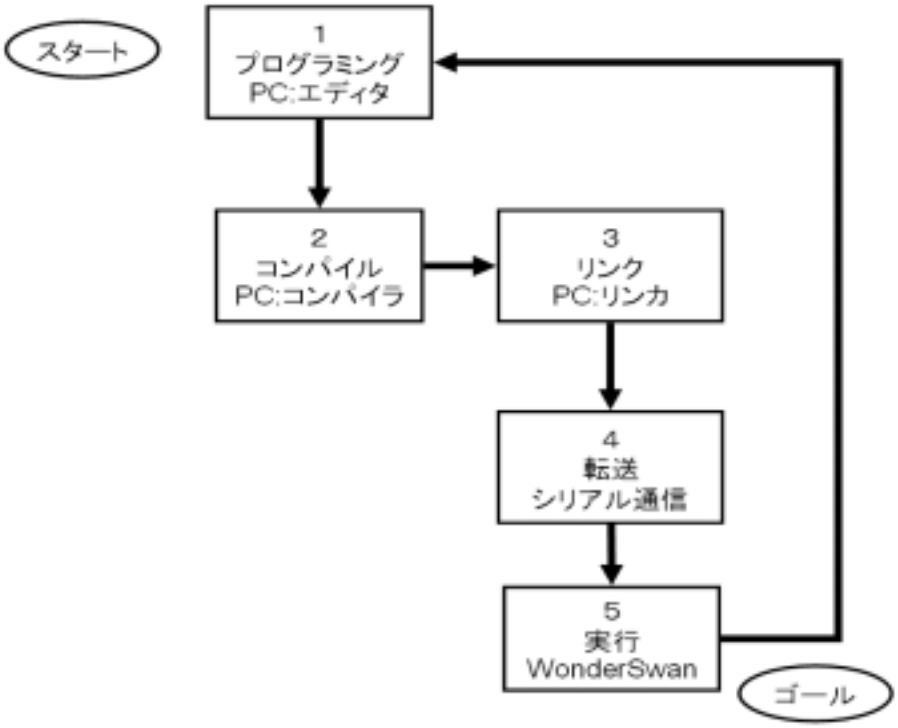

図 3.1: 実装の流れ

3. リンク
   オブジェクトファイルとライブラリをリンクして実行ファイルを作成する．
4. 転送
   転送用のソフトウェアとシリアル通信を使い，実行ファイルをPC上からWonderSwanへと転送する．転送された実行ファイルはWonderSwanのカートリッジへと書き込まれる．
5. 実行
   WonderSwan上で実行コードを実行する．この実行結果がゲームソフトである．

実装の流れについてはこの方法に限定されてはいない．他の方法で実装することもできる．

WonderSwanの画面の構成を図3.2に示す．

この画面について説明する．LCDパネルは横長の長方形であり，横と縦のドット数は横が224ドット，縦が144ドットであり，8ドット × 8ドットのキャラクタが横に28個，縦に18個入る大きさである．横と縦の比率は14対9である．左上の座標が(0,0)で，右下が(223,143)である．

図 3.2: 画面構成

3.2 シリアル通信

シリアル通信はPC上のデータをWonderSwanに転送する際に使用する．この他にも実装環境に用意されているシリアル通信BIOSを利用することにより，WonderSwan上で動作しているプログラムに書き込まれた任意の文字列をPCに転送することができる．この機能を利用し実装の際にプログラムコード中にプログラムの状態を書き込んでおき，逐次PCに転送することにより，プログラムの状態をPC上で確認することができる．こうすることにより，あるオブジェクトが現在どのような状態にあるのかをリアルタイムに知ることができる．

3.3 セグメント方式によるメモリ管理

家庭用携帯ゲーム機WonderSwanをターゲットマシンとするゆえに現在の一般的なPC上のCUIプログラムやGUIプログラムとは異なるところがある．この点について記述する．

WonderSwanはCPUにIntel80186とバイナリ互換性のあるものを搭載している．

このCPUについて少し解説する．このCPUがメモリを指定する方式は，セグメント方式である．セグメント方式とはセグメントアドレスとオフセットアドレスの2つのアドレスを使い，メモリのアドレスを指定する方式である．メモリのアドレスを指定するには

セグメントアドレスとオフセットアドレスの両方を使用する仕組みである．このうちセグメントアドレスの値はCPUのセグメントレジスタに格納される．そしてこのセグメントレジスタにオフセットアドレスを足すことでメモリアドレスを指定する．セグメントレジスタにはコードセグメント，データセグメント(以下，DSと略す)，スタックセグメント(以下，SSと略す)などがある．

WonderSwanに話を戻す．WonderSwanはこのCPUを搭載している．WonderSwanではDSとSSが別の場所にある．WonderSwan上でプログラミングする場合，このことを理解してプログラミングをしなければならない．プログラミングをする上での簡単な対策を示す．

- 対策1
  ポインタには"far"をつけて宣言をし，セグメントを越えてのアクセスを可能にする．
- 対策2
  配列は関数外で宣言してグローバルにするか，もしくは関数内での宣言には"static"をつける．

これら2つの対策1と対策2を両方とも行う．

3.4 仕様

実験で作成するソフトウェアのジャンルと全体の仕様について紹介する．
対象はテレビゲームソフトである．

- ジャンル
  - アクション
  - 横スクロール
  - サイドビュー
- 全体の仕様
  - プレイヤーはキャラクタMyをボタンで，左方向の速度，右方向の速度，上方向の速度を変更
  - Myには慣性が働く
  - Myには重力が働く
  - 壁Blockが画面上に存在する

- My が Block に触ると，My の速度が 0 になる
  (My と Block は重ならない)

- 各キャラクタの仕様
  - キャラクタ My の仕様
    - 左移動，右移動，ジャンプをする
    - 慣性と重力が働く
    - 空中にいるときはジャンプできない
    - プレイヤーに操作される
    - 左ボタンを押下すると横方向の速度 vx が減る
    - 右ボタンを押下すると横方向の速度 vx が増える
    - A ボタンを押下すると縦方向の速度 vy が減る
  - 壁 Block の仕様
    - 画面を構成する
    - Block があるところには My は重なることができない
  - 慣性
    - My は速度を持ち，My の速度が My の座標に加算される
  - 重力
    - My には常に重力がかかり，My を画面下方向に移動させる

3.5 コントローラと操作方法

プレイヤーは WonderSwan 本体に付いているコントローラで，主人公のキャラクタを操作する．
WonderSwan のコントローラは
”X1”，”X2”，”X3”，”X4”，
”Y1”，”Y2”，”Y3”，”Y4”，
”A”，”B”，
”START”，”SOUND”，”POWER”
の 13 個のボタンから成っており，基本的には音量を変える”SOUND”ボタンと電源スイッチの”POWER”ボタンを除いた 11 種類のボタンを押すことでゲームを進める．このうち X と Y ボタンは左側に上下左右，または前後左右に配置されており，視覚的に分かりやすくなっている．A と B ボタンは右側に並んで配置されている．図 3.3 のような配置である．

作成するゲームの操作方法を示す．プレイヤーは主人公のみ操作できる．以下は主人公の動作に影響を与える操作である．

図 3.3: WonderSwan のキー配置

- ”X4”ボタン押下: 左へ移動
- ”X2”ボタン押下: 右へ移動
- ”A” ボタン押下: ジャンプ

3.6 アルゴリズム

プレイヤーは主人公の操作をするが，その際の仕組みについて示す.

- 左右移動
  ”X2”ボタンを押すと画面右方へ移動する．しかしこれをそのまま右を押した分そのまま右へ移動するのではアクションゲームとしては物足りない．そこで以下のようにした.
  - ”X2”，”X4”ボタンを押している時間だけ加速が付く
  - ”X2”，”X4”ボタンを離すと慣性の働きにより少しずつ止まっていく
- ジャンプ
  ”A”ボタンを押すと画面上方へ移動する.
  - ”A”ボタンを押すと上方へ移動するが，重力が働いており，徐々に速度が下がり，頂点に達すると下方向へ戻る

- 高いところから降りる際には速度が上がり勢いがつく
- ブロックの上に来ると速度が 0 になり停止する．仕組みとしてループを使い，頂点から同じだけ下に移動するという方式が簡単であるが，この場合途中にブロックがあるときや，高いところから降りたときに自然に動いているように見えなくなる．よってこの仕組みにはしなかった．

第4章　オブジェクト指向方法論を用いないソフトウェア開発

オブジェクト指向方法論を適用したソフトウェアを作成する前に，オブジェクト指向方法論を適用せず，従来の手法で開発したソフトウェアについて紹介する．

4.1　オブジェクト指向方法論を用いない開発の流れ

オブジェクト指向方法論を用いないソフトウェア開発を行う手順を説明する．図 4.1 に全体の開発の流れを示す．

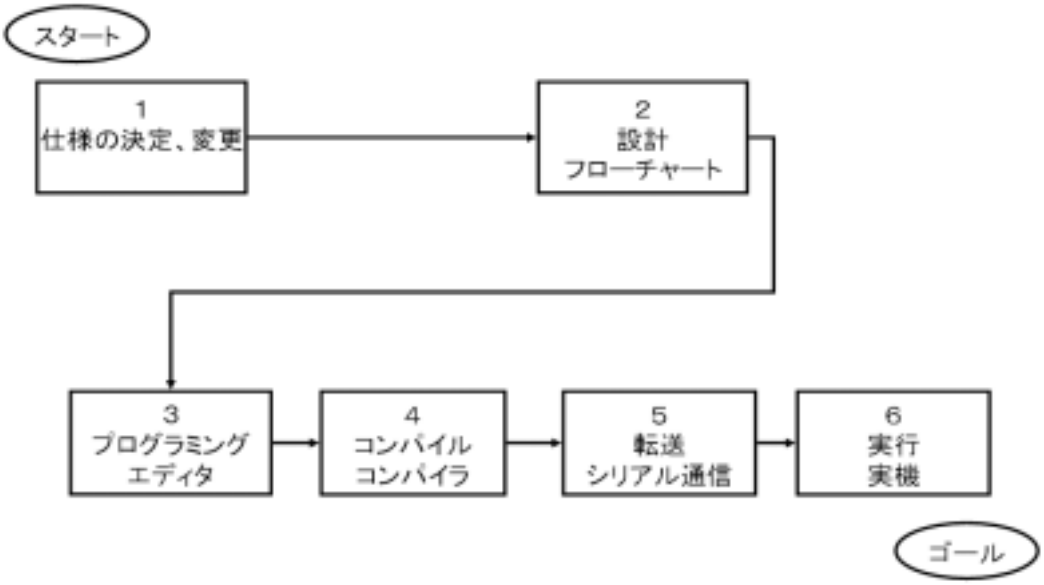

図 4.1: オブジェクト指向方法論を用いない開発の流れ

図 4.1 の説明をする．

1. 仕様の決定と変更
   作成するソフトウェアの使用や詳細を決定する．
2. 設計
   設計段階としてフローチャートを作成する．
3. プログラミング
   PC 上のエディタでプログラミングを行う．携帯型ゲーム機 WonderSwan へ実装する．
4. コンパイル
   エディタにて作成したプログラムをコンパイルし，実行コードを生成する．
5. 転送
   シリアル通信にて，出来上がった実行コードを携帯型ゲーム機 WonderSwan に転送する．
6. 実行
   WonderSwan 上で実行コードを動作させる．

4.2 従来の手法での設計

設計の一部として，フローチャートを作成した．図 4.2 にフローチャートを示す．

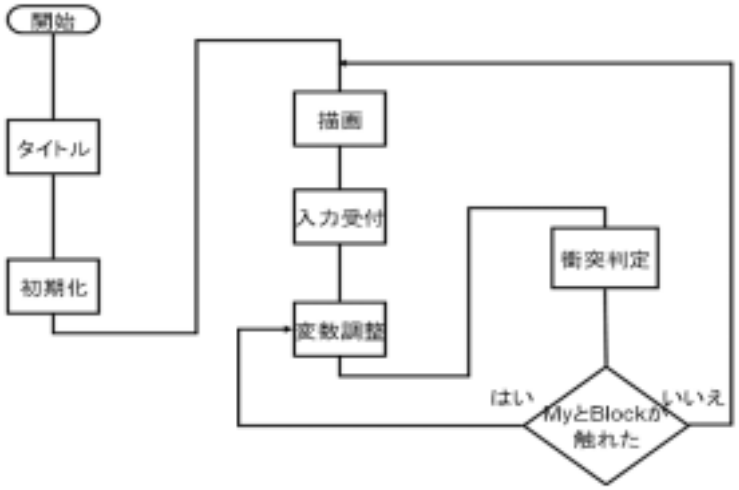

図 4.2: フローチャート

フローチャートの解説をする．

- 開始
  家庭用のテレビゲーム機は，電源を入れると同時に開始となる場合が多い．本実験

で使用した環境は特殊で，電源を入れた後に起動するソフトウェアを選ぶことができる．本実験では”開始”は起動するソフトウェアを選んだ時点である．

- タイトル
  ”press start”という表示のまま，プレイヤーが”START”ボタンを押すのを待つ．これは家庭用ゲームソフトが一般的に電源を入れてから 1 ステップ置いてゲームの本体が動くので本実験でもこのようにした．

- 初期化
  WonderSwan ソフトウェアのシステム部分や，各キャラクタの変数などの初期化，定義を行う．

- 描画
  各キャラクタや背景を画面に表示している．

- 入力受付
  プレイヤーがコントローラを通して WonderSwan へのキー入力を受け付けるところである．

- 変数調整
  プレイヤーからの入力や，プレイヤーの入力に依存しない変数を動かすところである．

- 当たり判定
  主人公 My と他のキャラクタ，ここでは壁 Block が衝突したかどうかを判定している．衝突しているかどうかにより分岐が起こり，変数を調整するか，”描画”へ戻りループを続けるかが決まる．

  実装においてもこのフローチャートをほぼ踏襲した．

4.3 従来の手法での実装

WonderSwan の開発環境にて用意されているプログラミング言語，C 言語を用いて実装を行った．本開発実験での実装の際の手順を示す．

1. プログラミング
   ソースプログラムを PC 上のエディタで作成．
2. コンパイル
   作成したソースプログラムをコマンドライン上でコンパイルし，オブジェクトファイルを生成．
3. リンク
   必要なオブジェクトファイルとライブラリをリンクし実行ファイルを生成．

4. 転送
   PC 上で作成した実行ファイルを WonderSwan に転送し，カートリッジに書き込む．転送方法はシリアル通信で行う．
5. 実行
   WonderSwan に転送されたファイルを実行する．拡張や変更があればPC でのプログラミングへ戻る．

ソースコードはおよそ 800 行で，単純な for 文カウンタを 10000 回まわしてちょうど良いくらいのゲーム処理速度となった．以下に単純な for 文のコードの例を挙げる．実際のコードと同じではないが，このような仕組みである．

```
for (counter = 0; counter < 10000; counter++){
    ;
}
```

実装した結果のゲーム画面を図 4.3 に示す．

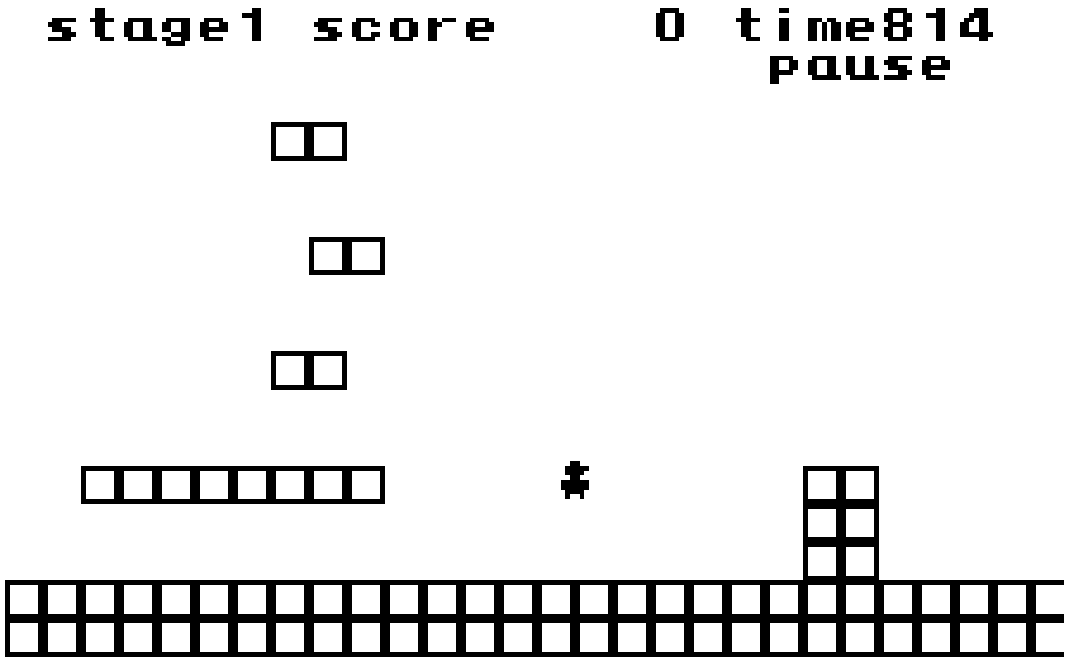

図 4.3: オブジェクト指向方法論未使用時のゲーム画面

中央下部にいるのが主人公であるキャラクタ My であり，この画面では，地面からジャンプしているところである．画面全体にある四角いものが壁である Block である．画面上部に表示されている time の右側の数字は残り時間を表す．

第5章　オブジェクト指向方法論によるソフトウェア開発

オブジェクト指向方法論を適用し開発を行ったソフトウェアについて紹介する.

5.1　オブジェクト指向方法論を用いた開発の流れ

オブジェクト指向方法論によるソフトウェア開発を行う手順を説明する．図 5.1 に全体の開発の流れを示す.

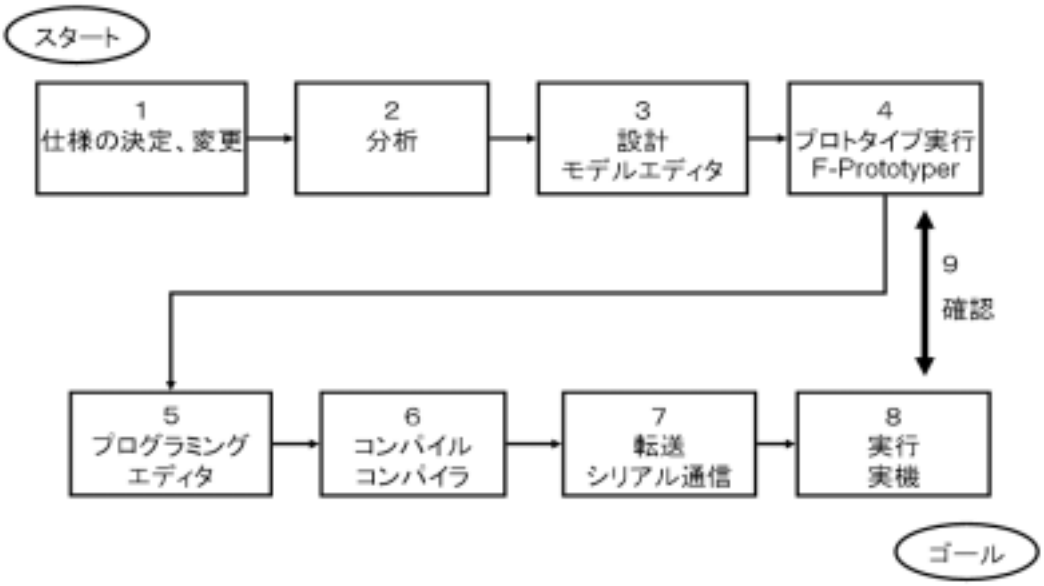

図 5.1: オブジェクト指向方法論を用いた開発の流れ

図 5.1 の説明をする.

1. 仕様の決定と変更
   作成するソフトウェアの使用や詳細を決定する．
2. 分析
   仕様に基づきシステムの分析を行う．オブジェクト指向方法論を適用するので，クラスに分けやすいように分析を行う．
3. 設計
   計算機支援環境 F-Developer の機能，モデルエディタを使用し，クラス図と状態遷移図を描画する．各クラスが持つ各要素や，クラス間の関連などを定義する．
4. プロトタイプ実行
   計算機支援環境 F-Developer の機能，F-Prototyper を使用し，プロトタイプ実行を行う．実装の前にバグを発見するためのテストも行う．
5. プログラミング
   PC 上のエディタでプログラミングを行う．携帯型ゲーム機 WonderSwan へ実装する．
6. コンパイル
   エディタにて作成したプログラムをコンパイルし，実行コードを生成する．
7. 転送
   シリアル通信にて，出来上がった実行コードを携帯型ゲーム機 WonderSwan に転送する．
8. 実行
   WonderSwan 上で実行コードを動作させる．
9. 確認
   設計したものと実装したものを比較し，仕様通り，予期した通りに動作しているのかを比較する．

5.2 オブジェクト指向分析

ここでは仕様に基づき分析を行った結果を示す．仕様から名詞の抽出をして，クラス候補を挙げた．表 5.1 に示す．

この表 5.1 クラス候補で，”プレイヤー”はゲームをする人間のことである．ゲームの操作を行う人間は開発実験中には登場しないこととし，”アプリケーション外”とした．”画面”と”ボタン”は”コンピュータ領域”とした．”コンピュータ領域”とはテレビゲームソフトウェアの中身ではなくコンピュータそのものを使用するものである．”主人公”，”壁”，”重力”，”慣性”は”問題領域”とした．”問題領域”とはテレビゲームソフトウェア本体の部分である．このように分けることで，実装環境の変化に対応しやすくなる．

各クラス候補の詳細を示す．

- クラス候補の詳細
  - My

表 5.1: クラス候補

名詞	説明
プレイヤー	(アプリケーション外)
画面	画面 (コンピュータ領域)
ボタン	ボタン (コンピュータ領域)
My	主人公 (問題領域)
Block	壁 (問題領域)
gravitation	重力 (問題領域)
inertia	慣性 (問題領域)

    * 左移動，右移動，ジャンプをする
    * 左移動時，右移動時には慣性がつく
    * 空中にいるときにジャンプはできない
    * プレイヤーが
      左ボタンを押下すると横方向の速度 vx が減る
      右ボタンを押下すると横方向の速度 vx が増える
      Ａボタンを押下すると縦方向の速度 vy が減る

  - Block
    * 画面を構成するもの
    * Block があるところには My は重なることができない
  - 慣性
    * My は速度を持っていて，My の速度が My の座標に加算される
  - 重力
    * My には常に重力がかかっていて，My を画面下方向に移動させる

名詞の候補を，実際に使用するクラスとして絞り込んだ．アプリケーション外のプレイヤーは除外する．コンピュータ領域の画面とボタンをそれぞれ Display から”Disp”，Controller から”Cont”とする．問題領域のキャラクタである主人公 My，壁 Block の 2 つはそのまま”My”と”Block”として使用する．重力 gravitation と慣性 inertia はひとつにまとめ，Power から”Pow”とする．My と他のキャラクタの衝突判定をするクラスを”Touch”とする．

各クラスの状態について表 5.2 に示す．

この表 5.2 クラスが遷移する状態の説明をする．My は，停止して止まっている STOP，左移動中の LEFT，右移動中の RIGHT，ジャンプ中または高いところから降りたときの

表 5.2: クラスが遷移する状態

クラス	要素	状態
My	停止中	STOP
	左移動中	LEFT
	右移動中	RIGHT
	空中中	AIR
Block	特になし	NONE
Pow	特になし	NONE
Touch	特になし	WHICH
Disp	特になし	NONE
Cont	ニュートラル	NEUTRAL
	左ボタン	LEFT
	右ボタン	RIGHT
	A ボタン	A

AIR という 4 つの状態を取る．ここで空中中となっているのは空中にいる間という意味である．Block，Pow，Disp の 4 つは動作としては一瞬のものしかなく，NONE という状態以外は取らない．Touch の WHICH は判定をしている状態だが，上記 Block などの NONE とほぼ同じである．Cont は，どのボタンも押されていない NEUTRAL，左ボタンである X4 ボタンが押されている LEFT，右ボタンである X2 ボタンが押されている RIGHT，A ボタンが押されている A という状態を取りうる．WonderSwan には他にも多くのボタンがあるが，この 3 つのボタン以外は特に定めない．

各クラスの属性を表 5.3 にコンピュータ領域，問題領域ともに示す．

表 5.3 各クラスの属性について説明する．

My と Block について記述する．”現在位置”はそのキャラクタが画面上のどこにいるのかの座標を表す．”幅”はそのキャラクタが持つ横幅と縦幅を表す．”速度”は My の速度で，クラス Pow によってこの数値が My の座標に加算される．この値が正の数のときは画面右方へ負のときは画面左方へと移動することになる．”加速度”は My の速度の数値を変更する数値で，速度の数値変化は基本的に加速度が速度に加算されて決定する．”空中判定”は My が現在空中にいるかどうかを現す．これを設定することによってジャンプ中にジャンプしてしまわない仕組みができる．

Pow について記述する．”慣性”は My の横方向に作用する力，”重力”は My の縦方向に作用する力であり，My の座標 mx，my を”My を動かす”で変更する．

Touch について記述する．”Block の座標”，”Block の幅”はクラス Touch が Block の座標を取得して”接触したか判定”を行う際にこの数値を利用するために使われる．

Disp については特に属性を持たなくても役目を果たす．

Cont について記述する．”入力を受け取り My の速度を変更”ではプレイヤーの入力を

表 5.3: 各クラスの属性

クラス	要素	属性
My (主人公)	現在位置	mx, my
	幅	mw, mh
	速度	vx, vy
	加速度	ax, ay
	空中判定	air
	関連	Pow, Cont, Touch, Disp
	位置を取得	My_mxGet, My_myGet
	幅を取得	My_mwGet, My_mhGet
	速度を取得	My_vxGet, My_vyGet
	加速度を取得	My_axGet, My_ayGet
	空中状態を取得	My_airGet
	移動	My_mxChange, My_myChange
	速度変更	My_vxChange, My_vyChange
	空中状態変更	My_airChange
Block (壁)	現在位置	bx, by
	幅	bw, bh
	関連	Touch, Disp
	位置を取得	Block_bxGet, BlockbyGet
	幅を取得	Block_bwGet, Block_bhGet
Pow (慣性，重力)	慣性	iner
	重力	grav
	関連	My
	慣性，重力を取得	Pow_inerGet, Pow_gravGet
	My を動かす	Pow_move
Touch (接触判定)	Block の座標	bx, by
	Block の幅	bw, bh
	関連	My, Block
	接触したか判定	Touch_xHantei, Touch_yHantei
Disp (画面)	関連	My, Block
	描画	Disp_Chara
Cont (コントローラ)	関連	My
	入力を受け取り My の速度を変更	Cont_My

判断し，My の速度 vx，vy を変更する役目を果たす．

5.3　オブジェクト指向設計

分析の後，設計を行った．設計の際には計算機支援環境 F-Developer の機能，モデルエディタを使用し，次に挙げるクラス図と状態遷移図を記述，各クラスの変数や関連などを定義する．

5.3.1　クラス図

モデルエディタで記述したクラス図を図 5.2 に示す．作成するテレビゲームソフトウェアで最も中心となるのが主人公であるキャラクタ My である．このことからクラス My を中心としたクラス図とした．キャラクタ My と壁 Block は，接触判定を行う Touch を介して接触判定を行う．キャラクタ My へはコントローラ Cont からどのように動くか指示があり，My が持つ速度が変化する．慣性と重力の Pow クラスは My の速度に従い，My を直接動作させる．

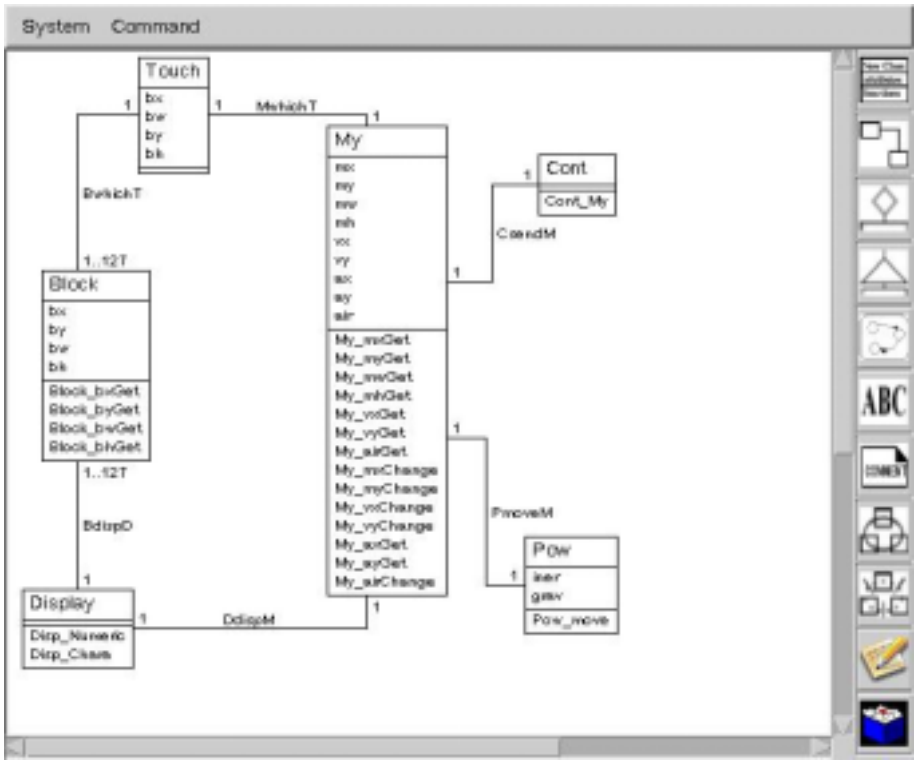

図 5.2: クラス図

本開発実験のクラス図について説明する．本章の”5.2 オブジェクト指向分析” でオブ

ジェクト指向方法論を適用したシステムの分析を行い，表 5.3 に各クラスの属性を示したが，ここで示した属性がクラス図に反映されている．例えば主人公 My は属性として現在位置”mx”と”my”や速度”vx”と”vy” などを持つが，クラス図の図 5.2 の My クラスの属性にそれが記述されている．現在位置の属性を例に挙げると，mx，my はプログラミング言語 ML の変数の int 型で定義されている．具体的な式を挙げる．mx は

```
Name:      mx
ML Type:   int
```

と定義され，my は

```
Name:      my
ML Type:   int
```

と定義されている．本開発実験では各クラスの持つ変数は全て int 型で定義した．よって詳しく表を作ることは省略する．これらの属性の変数は全て F-Prototyper でプロトタイプ実行をする際に ML のコードに変換され，利用される．利用されるということはプロトタイプ実行の信頼性にかかる問題である．このような理由で全ての変数には明確な型をつける必要がある．

5.3.2 状態遷移図

モデルエディタで記述した状態遷移図を図 5.3 から図 5.8 まで示す．主人公 My と，My に指示を出すコントローラ Cont の状態が多い．イベントが起こりガード条件にかなっていれば遷移が起こり，アウトプットイベントが発生する．このアウトプットイベントが関連のある他のクラスのイベントを起こす．この繰り返しでモデルは動作し，進行する．

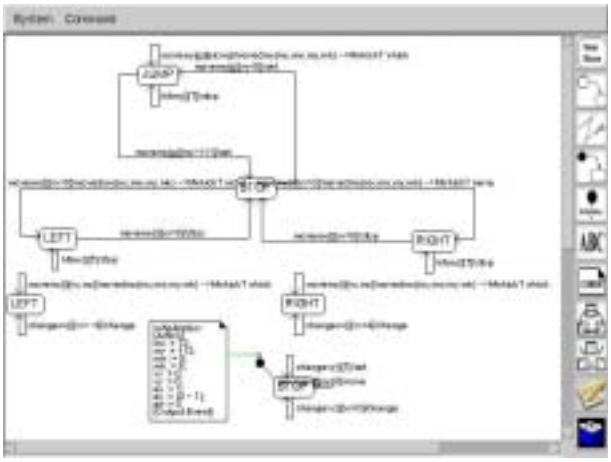

図 5.3: My の状態遷移図

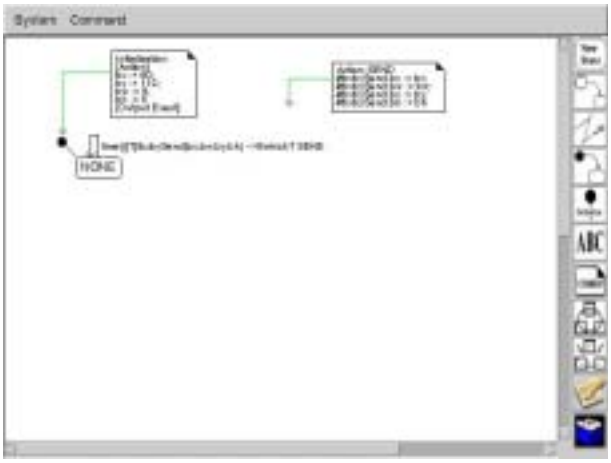

図 5.4: Block の状態遷移図

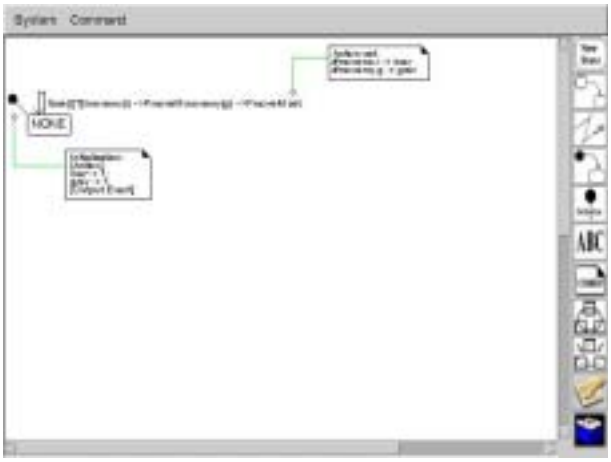

図 5.5: Pow の状態遷移図

本開発実験の状態遷移図について説明する．オブジェクト指向方法論を適用したシステムの分析を行い，表 5.2 にクラスが遷移する状態を示したが，ここで定義した状態が，状態遷移図に反映されている．例えば主人公 My は状態として停止状態の”STOP”，右移動中の”RIGHT”，左移動中の”LEFT”，空中にいる状態の”AIR”を持つが，それが My の状態遷移図の図 5.3 に記述されている．記述されているのは状態だけではない．表 5.3 には My の関連として”Pow”，”Cont”，”Touch”，”Disp”が挙げられているが，関連があるゆえにイベントの送信，または受信がある．イベントの送信にはアウトプットイベントを，イベントの受信にはインプットイベントで行っている．またイベント自体にもさまざまな要素がある．イベント自体には名前と受信側が受け取る引数の型も定義される．例えばイベント”movemx”は

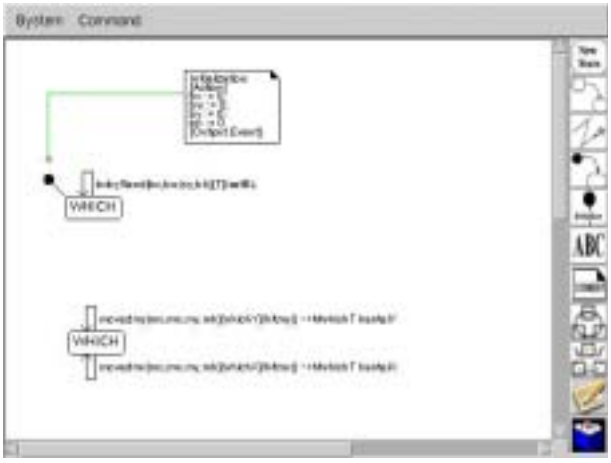

図 5.6: Touch の状態遷移図

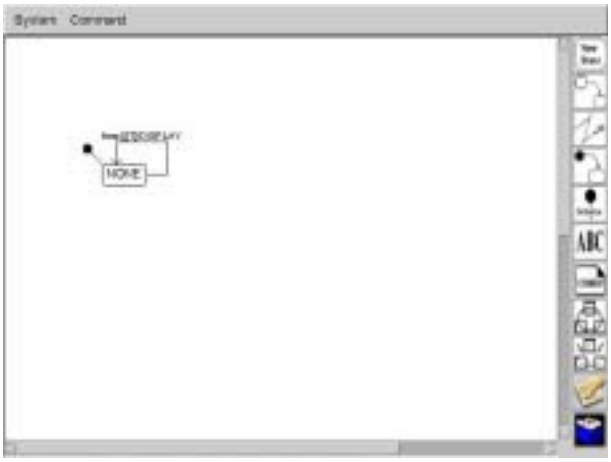

図 5.7: Disp の状態遷移図

```
Name:               movemx
Attribute:
           Name:    i
           ML Type: int
```

と定義されている．遷移はさらに詳しく定義されている．どのインプットイベントにより遷移が起こるのか，遷移が起こった際のアクション式，遷移が起こる条件であるガード条件，アウトプットイベントなどを定義しなければならない．もちろん何もないところは記入，変更する必要はない．具体例を挙げる．My の状態 STOP から状態 RIGHT に遷移するには次のような定義がなされている．

```
Action:       move
```

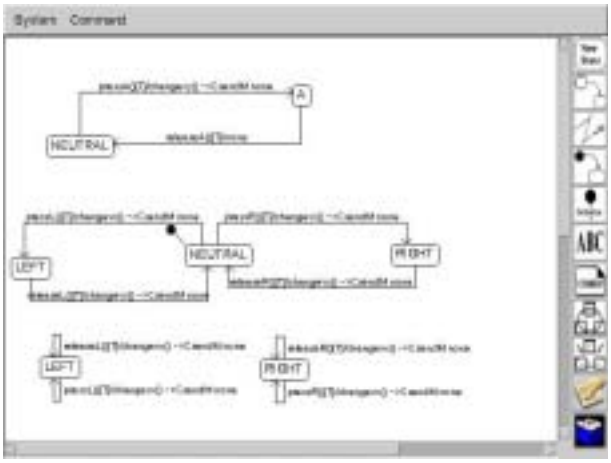

図 5.8: Cont の状態遷移図

```
Condition:    vx>0
Entry Action: none
Input Event:  movemx
Output Event: movedmx
```

と定義されている．それぞれについて詳しく示す．
”Action”は

```
Name: move
Action Expression: vx := (vx + #movemx.i);
                   mx := (mx + vx);
                   #movedmx.mx := mx;
                   #movedmx.my := my;
                   #movedmx.mw := mw;
                   #movedmx.mh := mh
```

となっている．move が起こると，最初に My の速度”vx”には vx 自身に Pow のアウトプットイベント movemx の引数”i”が加算される．2 番目に My の座標が変更される．3 番目にこの遷移のアウトプットイベントである，”movedmx”が Touch に渡す引数”mx”が設定される．my，mw，mh についても同様である．
”Condition”は

```
Name: vx>0
Condition Expression: vx > 0
```

となっている．My の属性”vx”が正の場合にのみ遷移が起こる．
”Entry Action”は

```
Name: none
```

となっている．つまり特に定めない．
”Input Event”はイベント作成時に定義されているものの中から選ぶ．ここでは”movedmx”が起きる．このイベントはイベント作成時に

```
Name:                movedmx
Attribute:
            Name:    i
            ML Type: int
```

となっている．”Output Event”はイベント作成時に定義されているものの中から選ぶ．
”Output Event”は

```
Name:  movedmx
Where Association: TwhichM
Attribute:
            Name:    mx
            ML Type: int
            Name:    my
            ML Type: int
            Name:    mw
            ML Type: int
            Name:    mh
            ML Type: int
```

となっている．ここで”Where Association”は，どの関連を通りアウトプットイベントが送信されるかを表す．引数の型は全てint型である．

このようにそれぞれが明確に定義されている．これらの状態や，遷移などの要素はクラス図の作成時と同様全てF-Prototyperでプロトタイプ実行をする際にMLのコードに変換され，利用される．明確に定義するのはこのような理由であるが，これはUMLが実際に動き，プロトタイプ実行ができるというF-Prototyperの最大の特徴に結びついている．

5.4 プロトタイプ実行

設計段階で使用したモデルエディタにてクラス図と状態遷移図の記述と各クラスの変数や関連などを定義した後，F-Developerの機能，プロトタイプ実行ツールF-Prototyperを使用しプロトタイプ実行を行った．

このプロトタイプ実行にはモデルエディタで記述した情報がそのまま利用される．情報はMLインタプリタに読み込まれる．アニメーションの機能により，状態の変化と変数の値をGUIで見て，期待した通りに動作しているか判断，確認ができる．

例を挙げると，常にキャラクタ My と壁 Block は重なっていないか，My が空中にいるときは My はジャンプできないという制約が守られているかなどを調べることができる．

このジャンプ中にジャンプできないという制約については，プロトタイプ実行をしたことにより，ガード条件が甘く，ジャンプの頂点でもう一度ジャンプできてしまったことに気付いた．具体的には，ジャンプをするということは My の縦方向の速度 vy に My の縦方向の加速度 ay を加算することで実現するが，修正する前の条件は vy が 0 のとき，ジャンプができるというものであった．確かにこれではジャンプの頂点で vy が 0 になったときにここからジャンプができてしまう．このことに気付いたので，続くプロトタイプ実行や実装の前に，バグとなりうるものを検出することができたといえる．

プロトタイプ実行では，クラス図で記述した各クラスからオブジェクトが生成され，このオブジェクトが実際に動作する．本実験では記述した各クラスからどのようなオブジェクトが作られたのか，アウトプットイベントの流れも同時に示した図を 5.9 に示す．

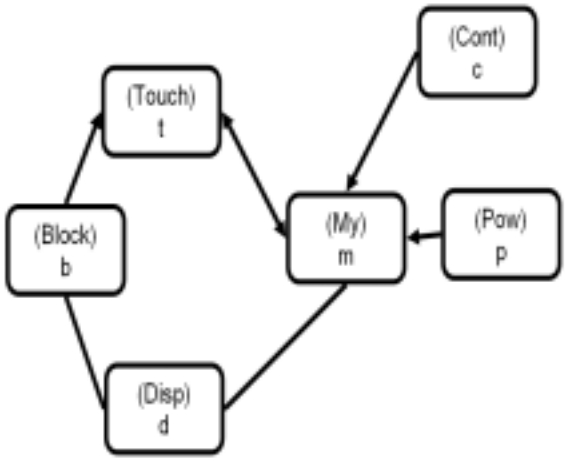

図 5.9: オブジェクトとアウトプットイベントの流れ

この図の矢印は，オブジェクトのアウトプットイベントの流れを示している．括弧の中の文字はクラス名であり，小英字はオブジェクトの名前である．

プロトタイプ実行において行ったテスト項目を示す．ジャンプ時のプロトタイプ実行の様子を図 5.10 から図 5.12 に示す．

このプロトタイプ実行時に得た値を表 5.4 に示す．My，Cont，Pow からそれぞれ m，c，p というオブジェクトを作り出した．m の初期値は my = 112，vy = 0 である．この結果からジャンプは正常に作動するといえる．

表 5.4 ジャンプのプロトタイプ実行の説明をする．

表 5.4: ジャンプのプロトタイプ実行

順序	イベント・同期	オブジェクト	状態・変数変化
初期値		m, c	my = 112, vy = 0, STOP, NEUTRAL
1	pressA	c	NEUTRAL → A
2	c → m	m	mx = 112, vy = -7
3	time	p	なし
4	releaseA	c	A → NEUTRAL
5	p → m	m	STOP → JUMP
6	time	p	なし
7	p → m	m	my = 106, vy = -6
8	time	p	なし
9	p → m	m	my = 101, vy = -5
10	time	p	なし
11	p → m	m	my = 97, vy = -4
12	time	p	なし
13	p → m	m	my = 94, vy = -3
14	time	p	なし
15	p → m	m	my = 92, vy = -2
16	time	p	なし
17	p → m	m	my = 91, vy = -1
18	time	p	なし
19	p → m	m	my = 91, vy = 0
20	time	p	なし
21	p → m	m	my = 92, vy = 1
22	time	p	なし
23	p → m	m	my = 94, vy = 2
24	time	p	なし
25	p → m	m	my = 97, vy = 3
26	time	p	なし
27	p → m	m	my = 101, vy = 4
28	time	p	なし
29	p → m	m	my = 106, vy = 5
30	time	p	なし
31	p → m	m	my = 112, vy = 6
32	time	p	なし
33	p → m	m	my = 112, vy = 0, JUMP → STOP

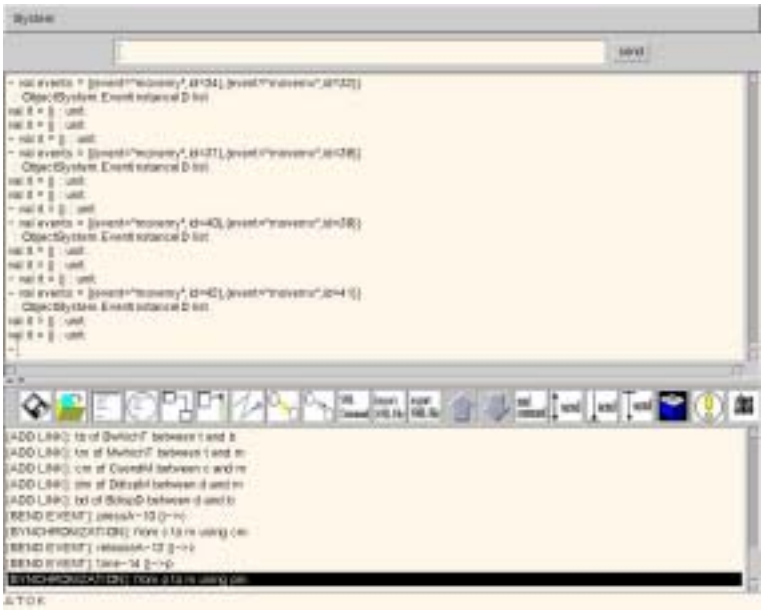

図 5.10: プロトタイプ実行

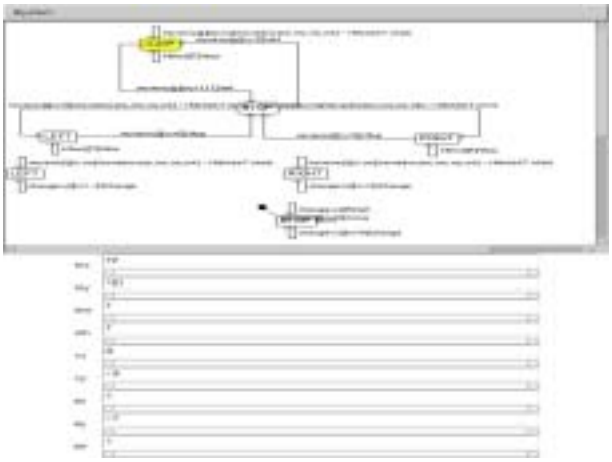

図 5.11: オブジェクト m のアニメーション

”初期値”はプロトタイプ実行前の各オブジェクトの変数と状態を示す．”イベント・同期”は特定の”オブジェクト”に対して作用する．”状態・変数変化”はイベントを受けたオブジェクトの状態と変数がどのように変化したかを，変数は”変数 = 数値”という形で，状態は”イベント・同期前 → イベント・同期後”という形で示している．

”順序”の 1 番目では”pressA”というイベントがコントローラのクラス Cont のオブジェクト”c”に働き，”c”の状態が”NEUTRAL”から”A”に変化したことを示す．”順序”の 2 番目では同期が”c”から主人公クラス My のオブジェクト”m”に伝わり，”m”の変数である”mx”

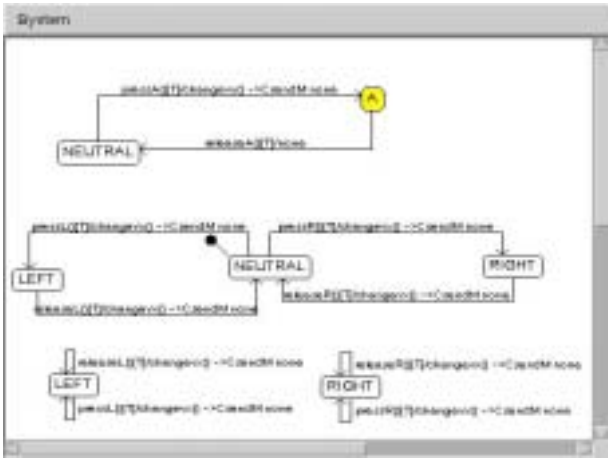

図 5.12: オブジェクト c のアニメーション

は変化せず，”vy”が-7 になったことを示す．

この表は”A”ボタンがプレイヤーに押されたときに”m”がジャンプをして，元の場所に戻るまでを示している．”順序”の 19 番目でジャンプの頂点に達し，33 番目で元の場所に戻ったことが分かる．

接触判定時のプロトタイプ実行の様子を図 5.13 から図 5.15 に示す．

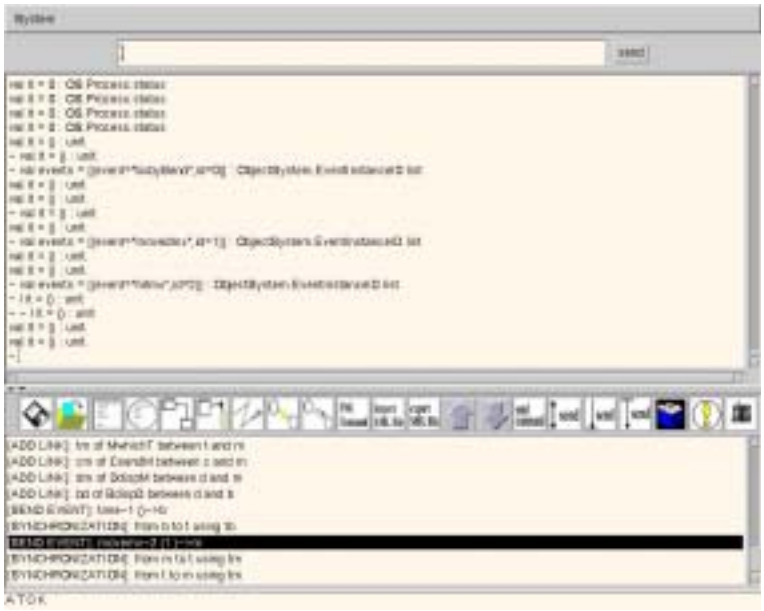

図 5.13: 接触判定のプロトタイプ実行

My，Touch，Block からそれぞれ m，t，b というオブジェクトを作り出した．前もって，スクリプトエディタで m.vy を 0 から 1 に変更しておく．スクリプトエディタで

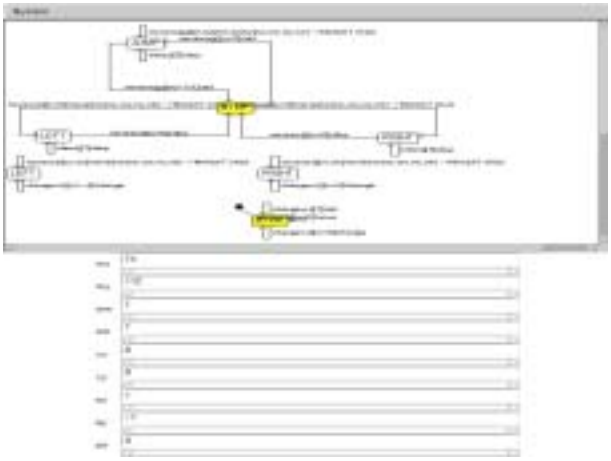

図 5.14: オブジェクト m のアニメーション

図 5.15: オブジェクト t のアニメーション

```
Class_My.set_vx "m" 1;
```

というコマンドを ML インタプリタに読み込ませると m.vx が 1 に設定される．m の初期値は mx = 72，vx = 1 である．また，イベント movemx はクラス Pow のオブジェクト p のアウトプットイベントであるが，スクリプトエディタで直接，イベント movemx を起こす．

このプロトタイプ実行時に得た値を表 5.5 に示す．

表 5.5 接触判定のプロトタイプ実行の説明をする．表の見方は表 5.4 時の説明部分と同じである．

”順序”の 1 番目では”time”というイベントが壁のクラス Block のオブジェクト”b”に働いたことを示す．”順序”の 2 番目では同期が”b”から接触判定クラス Touch のオブジェク

表 5.5: 接触判定のプロトタイプ実行

順序	イベント・同期	オブジェクト	状態・変数変化
初期値		m	mx = 72, vx = 1, STOP
		b	bx = 78, bw = 8, by = 112, bh = 8
		t	bx = 0, bw = 0, by = 0, bh = 8
1	time	b	なし
2	b → t	t	bx = 78, bw = 8, by = 112, bh = 8
3	movemx	m	mx = 74, vx = 2, STOP → RIGHT
4	m → t	t	なし
5	t → m	m	mx = 74, vx = 0
6	movemx	m	mx = 74, vx = 0, RIGHT → STOP
7	m → t	t	なし
8	t → m	m	なし

ト"t"に伝わり，"t"の変数である"bx"，"bw"，"by"，"bh"に値が設定されたことを示す．"順序"の 3 番目では"time"というイベントが主人公クラス My のオブジェクト"m"に働き，My の変数"mx"，"vx"が変化したことを示す．

この表は主人公 My のオブジェクト"m"が壁 Block のオブジェクト"b"に接触し，座標"mx"と速度"vx"が修正される様子を示している．"順序"の 3 番目で"m"と"b"は接触する位置になり，"順序"の 5 番目で"m"の速度の補正と状態の変化が起きた．

5.5 テスト

ソフトウェア開発の重要な過程としてソフトウェアのテストが挙げられる．本実験におけるソフトウェアのテストの項目として次の 2 つが考えられる．

- ドメインテスト
  ドメインテストは，境界条件などに誤りがないかをテストするものである．
- 状態テスト
  有り得ない状態へも遷移できるかを調べ，遷移に誤りがないかをテストするものである．

ここで挙げた状態テストであるが，テストの項目として表 5.6 が挙げられる．しかし F-Developer のプロトタイプ実行機能 F-Prototyper では，遷移の線とイベントがないと遷移のしようがない．この状態テストは F-Prototyper ではなく実装の際に役立つ．

上記の事柄から本実験でのソフトウェアのテストでは，ドメインテストを行うのが良いと判断した．本実験でのテスト法は，プロトタイプ実行機能 F-Prototyper を使い実現

表 5.6: 状態テスト

クラス	遷移前	遷移後
MY	RIGHT LEFT	LEFT RIGHT
CONT	RIGHT LEFT	LEFT RIGHT

する．F-Prototyper を使用することにより，実装前の設計段階でテストを行うことができる．F-Prototyper の機能としてスクリプトエディタがある．このスクリプトエディタは ML によって動作していて，ML のコードを人が記述し ML インタプリタに読み込ませることで，モデルの定義を変えずに各オブジェクトの変数や状態を自由に変えることができる．この機能を使いこなせばテストは容易になる．ドメインテストの値の例を次に示す．正常な動作をするか確かめることによりプロトタイプ実行から実装した結果がより正確になる．ここで，”m”は My のオブジェクトであり，”mx”，”my”はそれぞれ m の x 座標，y 座標である．

- 最小値-1:
  - m.mx := 7, m.my := -1
- 最小値:
  - m.mx := 8, m.my := 0
- 最小値+1:
  - m.mx := 9, m.my := 1
- 最大値-1:
  - m.mx := 214, m.my := 111
- 最大値:
  - m.mx := 215, m.my := 112
- 最大値+1:
  - m.mx := 216, m.my := 113
- 典型的な値:
  - m.mx := 72, m.my := 112

- 範囲外の値:
  - m.mx := 1, 300 m.my := -10,150

これらはF-Prototyperのスクリプトエディタを使用しテストを行うものである．この際，直接プログラミング言語ML のコードをスクリプトエディタに入力すると元のモデルの定義を変更せずにテストが行える．具体的なテストの様子の一部を図5.16に示す.

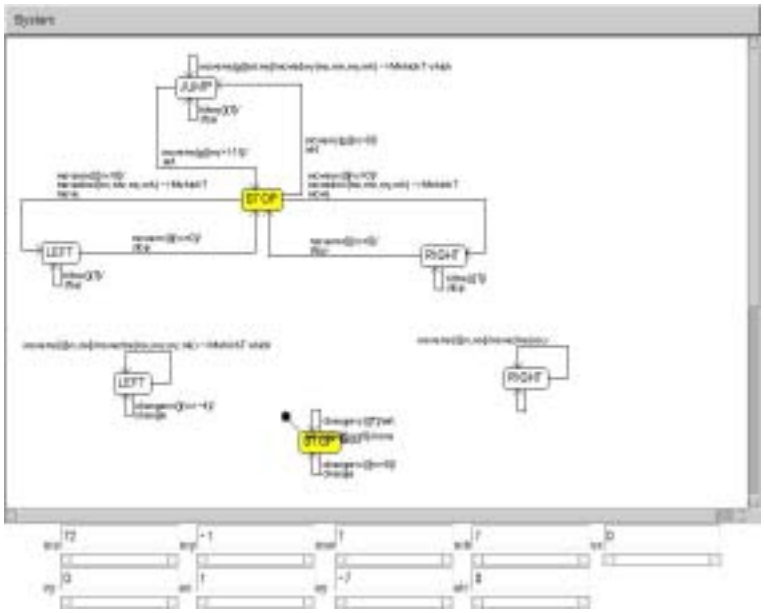

図 5.16: My のテスト

ジャンプのテストを行った．開発者が直接MLコードを書きスクリプトエディタに読み込ませることにより，My のy座標myを”最小値-1”に設定した．この設計ではmyの標準的な値は上限の”0”から下限の”112”である．このテストは主人公Myクラスのオブジェクト”m”の変数”my” に何らかの原因で不正な値が設定された場合に，そのための対策がなされているかをテストしている．値を表5.7に示す.

表5.7ジャンプのドメインテストの説明をする．表の見方は表5.4の説明部分と同じである.

”順序”の2番目では主人公Myクラスのオブジェクト”m”の変数”my”に，画面外を表す-1 が設定されていることが分かる．もちろん設定したのはスクリプトエディタの使用者がテストのために設定した．”順序”の5番目では”my”の数値が-7となり画面上部へはみ出してしまっている．しかし”順序”の7番目でしっかり元に戻ってきて，状態も正常に戻っている.

”最小値-1”という範囲になってもそのための対策がなされていることが分かる．また，もしその対策がなされていない，または不十分であるのが発見できたのであれば，このテ

表 5.7: ジャンプのドメインテスト

順序	イベント・同期	オブジェクト	状態・変数変化
初期値		m	my = -1, vy = 0, STOP
		c	NEUTRAL
1	pressA	c	NEUTRAL → A
2	c → m	m	my = -1, vy = -7
3	time	p	なし
4	releaseA	c	A → NEUTRAL
5	p → m	m	my = -7, vy = -6
			STOP → JUMP
6	time	p	なし
7	p → m	m	my = 112, vy = 0
			JUMP → STOP

ストにより実装前にバグを発見できたのであるから，開発全体において有効であることが確認できた．

5.6 オブジェクト指向方法論を適用した実装

F-Developer にて分析，設計を行った後，携帯ゲーム機 WonderSwan 用にプログラミングを行った．この実装段階について紹介する．

5.6.1 実装手法

開発に使用したプログラミング言語は C 言語である．C 言語でオブジェクト指向を実現する．オブジェクト指向言語として設計されていない C 言語であるが，今回はヘッダ部分とソース部分の 2 つのファイルでひとつのクラスを表すこととし，クラス間で通信を行っているようにした．

設計されたものを実装する際，プロトタイプ実行の結果がどのように実装結果に現れるようにしたのかを示す．

クラス中の変数と関数の実装法を示す．

- 変数
  - 構造体の中に変数を定義
- 関数

- ヘッダファイル中に，他のイベントから呼ばれる関数のプロトタイプ実行を記述
- ソースファイル中に，他のイベントから呼ばれる関数の中身を記述

My の右移動がどのように実装され，シリアル通信を使った状態の表示をしているのかを示す．

プレイヤーは WonderSwan の”X2”ボタンを押すと，Cont は右ボタンが押されたと判断し，Cont は My の速度 vx を変化させる．次に Pow クラスが My の座標 mx を変化させ，My は右に移動するのであるが，このとき My の速度 vx は正の数になっている．この My の vx が正の数か負の数かで，My がどのような状態にいるか判断している．ここでは My の vx は正の数，WonderSwan では右側が正なので右に移動していると判断する．判断したらシリアル通信で，PC へと現在の状態を，プログラム中で定義した文字列，ここでは”RIGHT”を転送する．PC 側ではこの文字列を受け取り，表示をしている．このようにして，設計，プロトタイプ実行が生かされているかを判断している．この判断については本章の最後，”状態の確認” に記述する．

5.6.2 実装内容

どのように設計を活かし実装したのか，具体的な例を挙げる．ここに記載しているコードは実際のコードをそのものを載せたのではなく，見やすさ，理解しやすさのため，アルゴリズムをそのままに変更している．ソースコードはおよそ 1000 行になった．

クラス図の構造を C 言語で実装した方法を示す．各クラスはヘッダとソースの 2 つのファイルで構成される．以下に作成したファイルのうち，クラスとして実装したもののファイル名を列挙する．

- クラス (.h ファイルと.c ファイル)
  - My
  - Block
  - Pow
  - Touch
  - Disp
  - Cont

これは設計の段階で作成したクラス図と同じである．

状態遷移図の My の構造をどのように実装したかを示す．状態を管理するには，状態を表す変数を My に持たせ，which 文で判断する関数を作成するのが良いであろう．
以下はファイル My.c の中のコードである．ここで”m”は My のオブジェクトを，”vx”は My の速度を表す．速度の正負により状態が”RIGHT”なのか”LEFT”なのか判断する．”status”は状態を保持する変数である．

```
if (m.vx > 0){            // 右へ移動
    m.status = RMOVE;
}
else if (m.vx < 0){       // 左へ移動
    m.status = LMOVE:
}
else if (m.air == FALSE){ // 空中にいない
    m.status = STOP;
}
```

ここで変数"status"に現在の状態を代入している．これを Disp クラスのオブジェクトである"d"がシリアル通信により，PC に転送している．以下はファイル Disp.c 中のコードのうち左右移動に関するものだけを示したものである．

```
switch (m.status){
case RMOVE:    // 右へ移動
    sprintf(send_buf, "RIGHT\r\n", SEND_BUF_SIZE);
    break;
case LMOVE:    // 左へ移動
    sprintf(send_buf, "LEFT \r\n", SEND_BUF_SIZE);
    break;
case STOP:    // 停止中
    sprintf(send_buf, "STOP \r\n", SEND_BUF_SIZE);
    break;
}
comm_send_string(send_buf);
```

"send_buf"は文字列を格納するために用意しておいた配列である．switch 文で send_buf に格納する文字列を選び，switch 文後の関数"comm_send_string" で，WonderSwan から PC へと send_buf の内容を送っている．このようにして状態の遷移を実装した．

接触判定の仕組みを示す．

m.mx，m.mw，m.my，m.mh はそれぞれクラス My のオブジェクト m の x 座標，横の幅，y 座標，縦の幅を表す．t.bx，t.bw，t.by，t.bh はそれぞれクラス Touch のオブジェクト t の x 座標，横の幅，y 座標，縦の幅を表す．

```
(
    (m.mx        <= t.bx + t.bw) &&
    (m.mx + m.mw >= t.bx       ) &&
    (m.my        <= t.by + t.bh) &&
    (m.my + m.mh >= t.by       )
)
```

オブジェクト指向方法論を適用しない開発と同じく，単純な for 文カウンタを 10000 回まわすウエイト処理を組み込んだ．以下に単純な for 文のコードを挙げる．

```
for (counter = 0; counter < 10000; counter++){
    ;
}
```

5.6.3 実装結果

実装を終えて実際にこのゲームを起動，動作させた．図 5.17 にゲーム画面を示す．

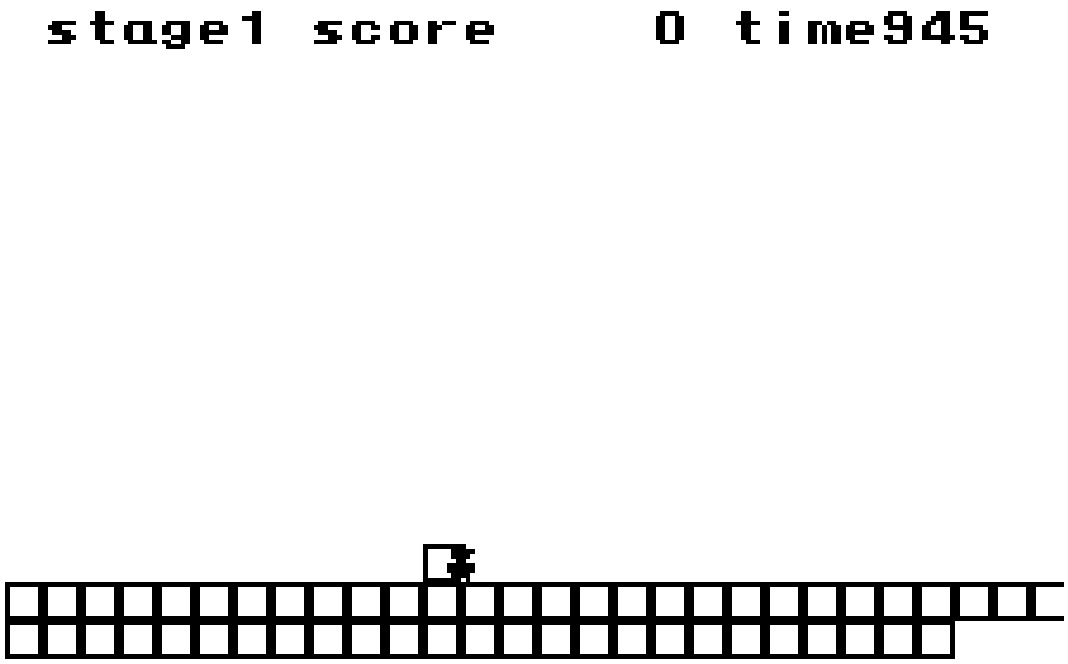

図 5.17: オブジェクト指向方法論使用時のゲーム画面

このゲーム画面は起動して間もない時間で取り込んだものである．画面下部のブロックの部分がまだ描画されきっていない．また主人公がブロックと重なっているが，接触判定が追いつかずに重なっている．またウエイト処理として組み込んだ，for 文を外すとプレイヤーが目で追えず，操作しきれないほど処理速度が速くなった．オブジェクト指向方法論を用いない開発ではウエイト用の for 文はオブジェクト指向方法論を用いて実装されたウエイト用の for 文と同じ 10000 回であるので，回数を 10000 回以外には設定しない．

5.6.4 状態の確認

WonderSwan と PC 間をシリアル通信で結び，WonderSwan から任意の文字列を PC に送ることができる．本実験ではプログラム中に My の状態を文字列で記述しておき，My の状態が常に PC に表示されるようにした．図 5.18 にプロトタイプ実行からの流れを示す．

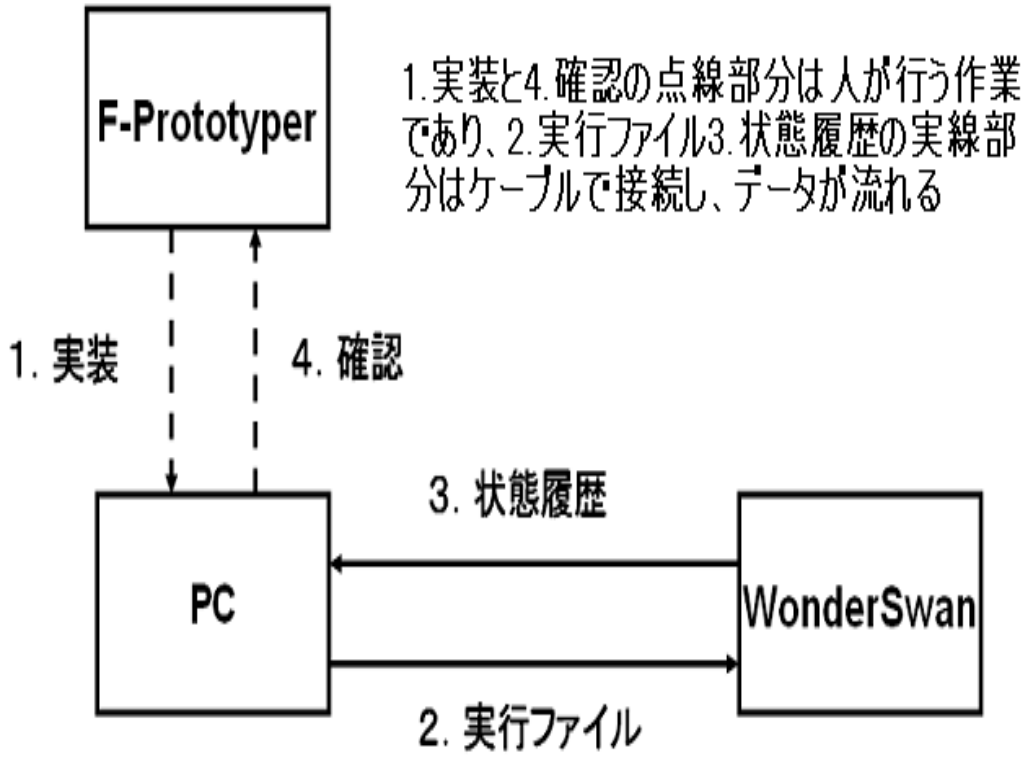

図 5.18: プロトタイプ実行からの流れ

PC 上で転送された内容を設計の内容と比較した．ここでの比較は目視による確認であるので，厳密に結果を示すことはしない．

ここで設計とは異なる結果を発見した．設計したモデルの My は停止，ジャンプ，右移動，左移動という 4 つの状態を有すると設計した．しかしこの実装だと斜めにジャンプすることができる．確かにこれは仕様に想定した，サイドビューの横スクロールアクションゲームということではきわめて自然である．しかし，仕様に厳密に記載されていない．とはいえ設計ではこのことを包含するモデルにはなっていない．F-Developer の使用により，実装段階で仕様の抜けを発見できたことになる．

またこの後に出来ることとして設計に戻り My の状態遷移図を変更することが考えられ

る．また設計の段階でこのことが分かれば，斜めジャンプだけに限らず，他に状態自体や遷移を拡張することによりゲームとしてのアイデアが発見される可能性がある．図 5.19 に PC 側の状態の受信の様子を，図 5.20 に設計段階の状態遷移図を示す．

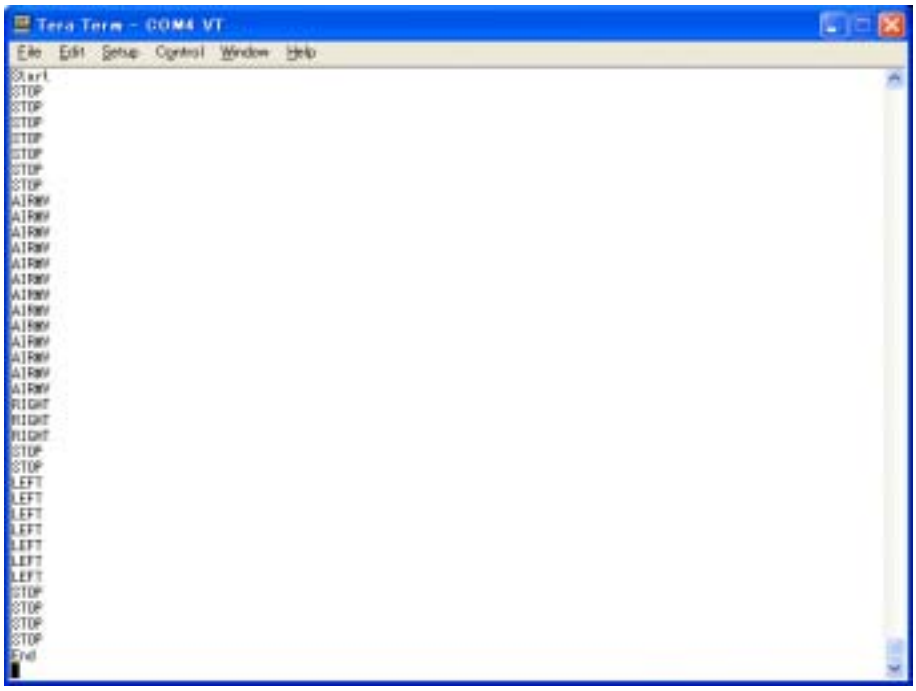

図 5.19: PC 側の状態の受信

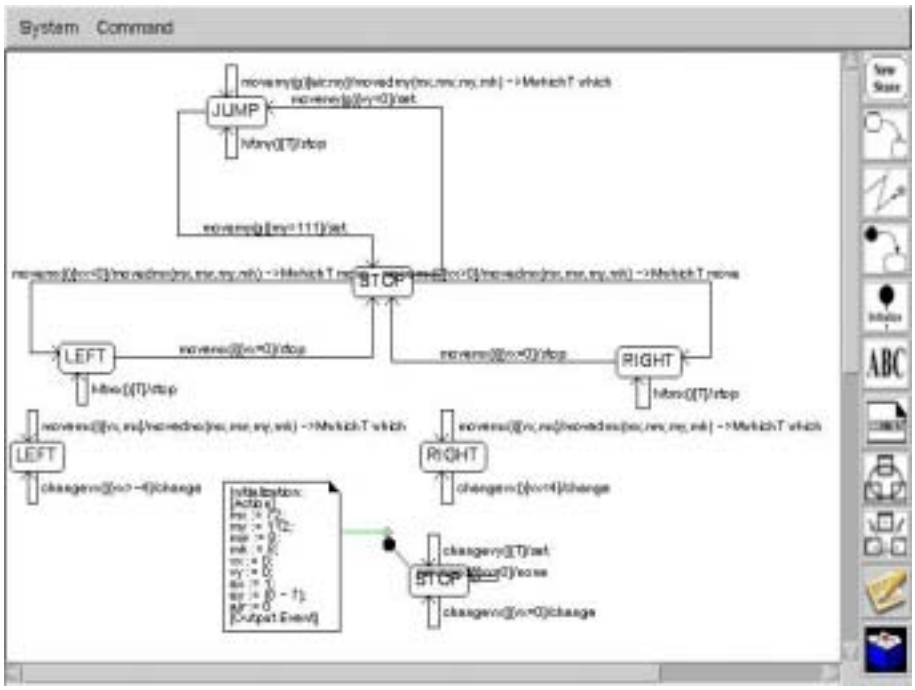

図 5.20: 設計段階の状態遷移図

第6章　おわりに

6.1　まとめ

本研究ではテレビゲームソフトウェア開発にオブジェクト指向方法論を適用することが有効であると考え，有効性の調査のためにテレビゲームソフトウェアの開発実験を行った．具体的には計算機支援環境 F-Developer，ターゲットマシンである携帯型ゲーム機 WonderSwan の使用法の調査，開発を行うテレビゲームソフトウェアの仕様の決定，オブジェクト指向方法論を用いない従来の方法での設計，実装，オブジェクト指向方法論を用いた分析，設計，実装などである．オブジェクト指向方法論を用いた開発での設計には計算機支援環境 F-Developer を使用し，プロトタイプ実行，テストを実施し，この結果を実装結果と比較した．これらの事柄を行い，オブジェクト指向方法論はテレビゲームソフトウェア開発に有用なのかを考察した．

6.2　評価

本研究は，テレビゲームの性格から，オブジェクト指向方法論がテレビゲームソフトウェア開発に有用であるとするところから始まった．

本論文第 1 章で記述した評価法として，テレビゲームソフトウェア開発に，オブジェクト指向方法論による開発手法の導入し，これにより開発の負荷の緩和，時間の節約に役立つかなどの評価をするとした．このことについて記述する．

本開発実験ではオブジェクト指向方法論を用いない従来の手法，そして計算機支援環境での支援とオブジェクト指向方法論を用いた手法の 2 種類の方法で開発を行った．開発の負荷の緩和ということだが，今回，本実験は 1 人で行っているのでソフトウェア開発自体は作業が並行して進んでいくことはなく，直線的であった．よって自然にコードを 1 人で拡張していくことになった．このことについて負荷の緩和になったのか評価をすると，オブジェクト指向方法論という考えた方の導入という問題はあるが，機能を中心とした従来の開発手法よりも，各クラスの役目が明確で拡張はしやすかったと感じた．時間の節約についても同様で，コストの削減になった．

ここでひとつの疑問が生じるかもしれない．開発実験の手順として，先に従来の方法で作成してから同じ仕様のものをオブジェクト指向方法論を用いて作成したら，開発しやすいのではないかというものである．これは確かにいえることである．しかし今回の開発実験ではオブジェクト指向方法論の適用以外にも計算機支援環境 F-Developer を使用して

設計，プロトタイプ実行，テストと行っている．これはバグの減少や設計段階の厳密な定義による実装の容易さなど実装時に有利なだけではなく，複数人でのチームでの開発の場合，実際にプログラムコードを書かない同じチームの人間にも分かりやすくなる．これらのことや先程の感想も統合して考え，本開発実験ではオブジェクト指向方法論を用いた手法は有効であったと評価する．

6.3 結論

本開発実験ではオブジェクト指向方法論を用いない手法，計算機支援環境での支援とオブジェクト指向方法論を用いた手法の両方で開発を行った．開発手法としてはオブジェクト指向方法論を適用したほうが設計の段階，プログラムの変更ともに容易であった．出来上がったソフトウェアの性能については，オブジェクト指向方法論未使用時のほうがオブジェクト指向方法論使用時に比べ速い速度で作動した．今回の実験では主に，描画速度と接触判定の速度が目に見えて変わった．

オブジェクト指向方法論がテレビゲームソフトウェア開発に有用か．保守的な意見として，従来の方法で開発するのはどうかという提案には，開発コストの削減の面からオブジェクト指向方法論を使用したほうが良い．この場合，オブジェクト指向方法論未使用時に比べて実行速度があまりに遅い．原因として本実験ではイベント通信の量がとても多い点があげられる．これに対処する方法として，イベントが集中している場所を探し，この部分を改善する．その際，クラスの設計を見直す必要も出るであろう．この対処法ならばオブジェクト指向方法論の枠組みの中で解決可能である．

また，オブジェクト指向方法論から少し離れ，クラスとして設計したものを崩す方法がある．提案として，キャラクタなどの主だったキャラクタはオブジェクト指向方法論を適用し，他の背景などにはクラスとして設計したものを崩して速度などを改善する方法がある．

本実験ではオブジェクト指向方法論を用いた開発には計算機支援環境 F-Developer を使用したが，このような形式化されたモデリングやプロトタイプ実行により実装が容易になる面もあった．しかしこのようにして開発を行うことはまだ浸透していないので，このようなツールの使用について一般的になるには時間が必要であろう．

6.4 今後の展望

今後の展望として，いくつか列挙する．計算機支援環境 F-Prototyper にはネットワークを介して他のコンピュータとリアルタイムにデータをやり取りできる機能がある．また，プログラミング言語 Java を用いての GUI と連携をし，アニメーションと連動してプロトタイプ実行ができる．これを利用し，F-Prototyper と WonderSwan をつなげて，WonderSwan でのキャラクタの状態を F-Prototyper のアニメーションと連動して動作さ

せたり，逆に F-Prototyper のアニメーションから WonderSwan を動かしたりできる可能性があると考えられる．F-Prototyper と実機，ここでは WonderSwan がつながれば，より確かなプロトタイプ実行ができる．

今回使用したゲーム機 WonderSwan の開発環境は C 言語のみのサポートであった．より設計の内容を実装に反映するには，開発環境がオブジェクト指向言語である C++言語や Java をサポートしていれば容易になる．しかし実機の開発環境を提供しているゲーム機はごく少数であり，これを実現するには自らコンパイラを工夫するか，何かしらの工夫が必要であろう．

今回の開発実験で作成したモデルは，比較的小さなものであった．そしてテレビゲームソフトウェア開発において必ず必要になるものとして絵，画像などソフトウェア開発とは異質なことがらが絡んでくる．今回のグラフィック面のリソースは簡単なものであった．これからの拡張として，他にも多くのキャラクタを追加した場合はこの開発手法がうまく機能するか．より大きなモデルであった場合にどうなるか．音声や効果音をどう扱うか．これらを扱えるようになるとよりオブジェクト指向方法論の有効性が際立つ．

図 6.1 に簡単な拡張例を示す．

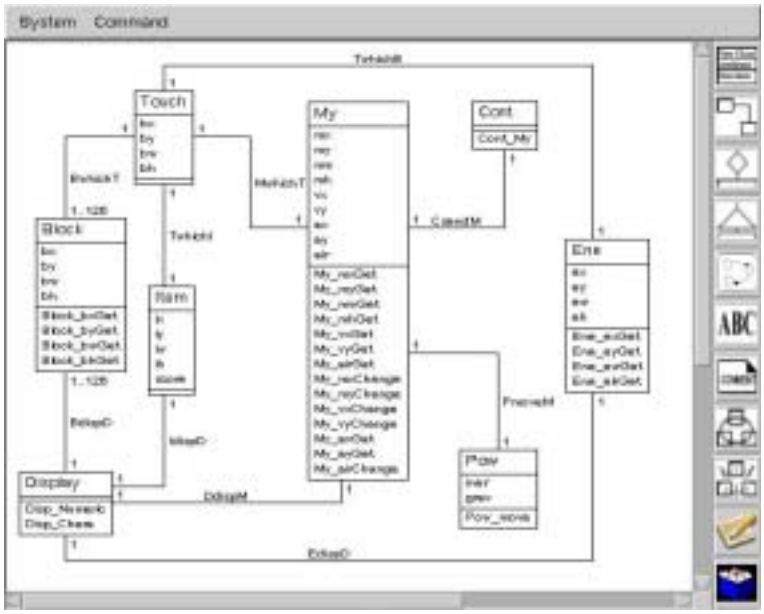

図 6.1: 拡張例

拡張後はこの手法による開発により，処理速度やメモリ容量といった性能が追いつかない場合があるであろう．この場合にどのようにコンピュータ資源を有効利用するか，また最新の家庭用ゲームハードへ実装した際，この手法が有用なのか，課題である．

謝辞

本研究を行うにあたり，御指導賜りました片山卓也教授に深く感謝いたします．本研究を進めるにあたり助言，アドバイス等をしてくださった，青木利晃助手，権藤克彦助教授，岡崎光隆氏，そしてソフトウェア基礎講座の皆様に厚くお礼を申し上げます．

参考文献

[1] Toshiaki Aoki and Takuya Katayama, Prototype Execution of Independently Constructed Object-Oriented Analysis Model, Automating the Object-OrientedSoftware Development Methods, ECOOP2001 Workshop, 2001.

[2] Toshiaki Aoki, オブジェクト指向方法論の形式化とその計算機支援環境, 博士論文, 1999.

[3] Toshiaki Aoki, Takaaki Tateishi, Takuya Katayama, 定理証明技術のオブジェクト指向分析への適用, pp.18-47, 日本ソフトウェア科学会学会誌コンピュータソフトウェア Vol.18 No.4, July 2001.

[4] Tucker!, 憂鬱なプログラマのためのオブジェクト指向開発講座, 株式会社翔泳社, 1998.

[5] 有限会社 キュート, Wonder Swan 取扱説明書 Magical Book, 株式会社バンダイ, 2000.

[6] 長久 勝, ワンダースワンゲームプログラミング ワンダーウィッチでゲームを作ろう!, ソフトバンク パブリッシング株式会社, 2001.

[7] Boris Beizer 著, 小野間彰, 山浦恒央 訳, ソフトウェアテスト技法, 日経BP 出版センター, 1994.

[8] 蒲地輝尚 著, 村瀬康治 監修, はじめて読む 8086, アスキー出版局, 1987.

[9] 有馬元嗣 著, 改定第 2 版 ゲームプログラミング 遊びのレシピ アルゴリズムとデータ構造, ソフトバンクパブリッシング株式会社, 2001.