ゲーム開発支援ツール

6W−02

坪田 睦生　藤本 真由美　三浦 孝夫　塩谷 勇

法政大学工学部電気電子工学科
産能大学経営情報学部

概 要

ゲーム開発時, アイディアとシナリオ, 背景知識, 処理, アルゴリズム, プログラムなどの知識が必要となる. しかし細部に渡る深い知識がないと実際に稼動させるためには多大な時間を要する. 本論文ではアイディアからテスト稼動に到るプロセスをできるだけ簡略化させるツールの設計, 実装について述べる.

1　前書き

ゲーム開発時において, 第1にゲームのアイディアの提示, およびストーリの作成. 第2に作りたいゲームのアルゴリズムを考える（デザインを含む）. 第3にアイディアとアルゴリズムを元にプログラムを組む. しかしこれら一つ一つの工程には多大な時間を要し, ゲームのタイプによって開発手順は異なる. そこで各工程においての手間の削減, 自動化するために「ゲーム開発支援ツール」を提案する. ここで提案する手法は, アイディア生成の支援, 手順化の支援, 開発工程の大幅な短縮を目標とする.

2　動機と解法

ゲーム開発時にはまずストーリ例やアイディアを作成する.　そのアイディアから自動的にストーリを作成し, 必要であればゲーム画面として表示させたい.　そこでHTML,PostgreSQL,GKT+ を用いて「ゲーム開発支援ツール」を構築する. これにより, 利用者はストーリの元となる話とアイディアを HTML 形式で作成し個々に Postgres へ格納することで, データをランダムに組み合わせストーリを作成し, プロトタイプ化して表示させ, 動作確認を行う. これによりワンパターンなストーリになるのを防ぐことができ, PostgreSQL を用いているのでデータの拡張も可能である. さらに, 作成したストーリを GTK+ をベースに作られた"Game"クラスを利用することで, ゲームとして実行することができる. またツールの動作環境は PostgreSQL サーバが稼動していること, X-Windows がインストールされているという二つの条件だけでよい. 次に対象となるゲーム内容を示す.

2.1　ゲーム内容

ゲーム自体は画面をクリックすることで進行していく. 実際には次のように処理を行っている.

(1) ウィンドウがクリックを感知
(2) フラグ操作のイベントを実行
(3) キャラクタのパラメータ変化のイベントの実行
(4) 画面に画像, テキストの順で表示
(5) 場面の移動
(6) 画面がクリックされるのを待つ

次に (2) 〜 (5) について説明する. また作成されるゲームのサンプルを図 1 に示す.

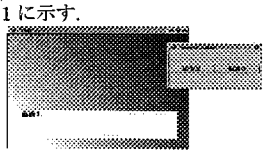

図 1: サンプルゲームウィンドウ

2.1.1　フラグ操作

画面クリックを感知するとまずこの処理を行う. ゲーム自身は各場面において操作, 参照するフラグ番号の情報を持っており現在いる場所にしたがってフラグ番号を操作する. また参照したフラグが"ON"の時に移動すべき場面のアドレスを持っており, 参照結果が"ON"の時のみそこへ移動. この場合以下の処理は移動先の場面にしたがって行う.

2.1.2　パラメータ処理

フラグ操作が終わったら次はキャラクタのパラメータ処理を行う. ゲーム自身は各場面において, キャラクタの変化させるパレメータとその値の情報を持っており, 現在いる場面に従ってキャラクタのパラメータを変化させる.

2.1.3　画面表示

次に画面表示を行う. ゲーム自身は各場面において表示すべき画像, テキストの情報を持っており, 現在いる場面に従って画面にまず画像, 次にテキストを表示する.

2.1.4　場面の移動

最後に場面の移動を行う. ゲーム自身は現在の場面から移動が可能な場面の情報を持っている. 移動先が一つの場合はそのまま移動し, 二つ以上ある場合は別ウィンドウで移動先を表示し, 選ばれた方へ移動する. また選ぶ間はゲームのメインとなっているウィンドウはクリックを受け付けない. 選び終わると再度ウィンドウはクリックを感知できるようになる. そしてウィンドウは再びクリックされるのを待ち, クリックされたらまたフラグ操作から順に処理を行っていく.

3　システム設計

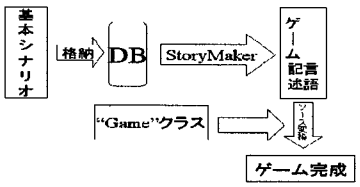

図 2: ゲーム開発支援システムの全体

Toolkit for Game Programming
Mutsuo Tsubota, Mayumi Fuzimoto, Takao Miura
Hosei University, Dept.of Elec. and Elec. Eng.
Kajino-cho 3-7-2, Koganei, Tokyo, JAPAN
Isamu Sioya
Sannou University, Dept.of Management. and Information.
Kamikasuya 1573, Iseharasi, Kanagawa, JAPAN

システムの流れは図 2 のようになる. まず初めに基本シナリオを DB へ格納する. その後 Story Maker でストーリ全体を構築し, 構築されたストーリでよければゲーム記述言語に沿った形で出力. 最後に"Game"クラスのメソッドに置換えプログラムのソースを出力する.

3.1 Story Maker

ゲームストーリの流れは「1. 誰が」「2. 何をした」「3. 結果どうなった」の 3 つから成り立ち, これの繰り返しである. そこで 1 〜 3 を「story」2 を「card」として PostgreSQL に格納する.Story Maker では「card」からランダムにデータを引き出し「story」と組み合わせてストーリを作成, ファイルへ出力する. その後, ゲームストーリの流れを HTML でプロトタイプ化して表示. さらに作成されたストーリで実際にゲームを作る場合, 後に示すゲーム記述言語の形に変換する.

3.1.1 "story"の登録

storyname,storyNO,naiyou の 3 つの属性を作成.storyname には枠組みとなるストーリの名前,storyNO には storyname ごとに naiyou の流れに沿った番号を入力する.naiyou には基本シナリオを HTML 形式で記述する. この際,「1. 誰が」「2. 何をした」「3. 結果どうなった」の順で記述し,「card」からランダムに引き出すため「2. 何をした」の部分には「><」と入力, そして「card」の種類によって起こる「3. 結果どうなった」の内容を'A' 〜'Z' の 26 分岐で用意し, <A NAME="AA-分岐"></A> の形で入力する.

3.1.2 "card"の実施

cardNO,type,action,link の 4 つの属性を作成し,cardNO にはタプルに対して番号を入力する. type には action,link の配列数を入力する.action には「2. 何をした」にあてはまる動詞を, ストーリ内に選択を用いたいときは配列として入力する. link には action と naiyou に記述された結果が対応するようなリンクを'A' 〜'Z' で記述する.

3.1.3 ストーリ作成

C インターフェースである libpq ライブラリを使用し, SQL によって DB へアクセスする. まず,story の中から用意されているストーリ名を$SELECT DISTINCT storyname FROM story で表示されるので, その中からゲームストーリとして利用したいストーリ名を入力. 次に, カーソルを使って選択したストーリの内容を$DECLARE mycursor CURSOR FOR SELECT naiyou FROM story WHERE storyname=story_choice ORDER BY storyNO で問い合わせ,$FETCH IN mycursor で 1 行ずつデータを取得していく. そして pg_exec() でタプルを返しファイルに出力していくのだが, この時の取得結果が'><' ならば乱数を発生させ, その結果と card の cardNO が一致するタプルの action と link のデータを$SELECT action[],link[] FROM card WHERE cardNO=card_random で取得し, pg_exec() でタプルを返し, <A HREF="#AA-link"> action</A> の形に当てはめてファイルに出力する. これにより,HTML 形式でストーリの流れを作成することができる. 作成されたストーリで実際にゲームを作成するときは, HTML からゲーム記述言語の形に変換をしなおし, 再びファイルへ出力しなおす.

3.2 ゲーム記述言語

対象となるゲームに必要な情報を決まった形で記述する. 1 つの場面に必要となる情報は今いる場面の名前, 表示するテキスト, 表示する画像, その場で起きるイベント内容, 最後に次に移動する場面の名前. この 5 つである. ここではそれぞれを「POSITION」,「TEXT」,「GRAPHIC」,「ACTION」,「LINK」と決める. 次に表記例を示す.

POSITION<Name> TEXT<Sentence>
GRAPHIC<Address> ACTINO<Type,Point>
LINK<Label,Address>

3.3 "Game" クラス

対象のゲーム全体で必要となる情報は表示画像, 表示テキスト, イベント内容, 場面の名前, 移動可能な場面のアドレス, キャラクタ情報, フラグ情報である. これらの情報をクラスとして管理する. そこで"Character"クラスはキャラクタ情報を,"Flag"クラスはフラグ情報を管理させ, メソッドにはそれぞれの持つ値を変化させるものを用意. また"AdvStage"クラスでは一つの場面で表示する画像やテキスト, イベントに必要な情報, 移動可能な場面のアドレスを管理する. そして"AdvStage"クラス = 場面の名前, とすることで, "AdvStage"クラスを参照していくことでゲームは進行する. メソッドには画面に画像とテキストを表示するもの, イベントの実行に必要な情報を"Character","Flag"クラスへ渡すもの, 次の場面へ移動するもの, 各情報をセットするものを用意. 各情報をセットするものにおいては前述のゲーム記述言語に対応させる. 対応表を表 1,2 に示す.

3.4 ゲームソースの作成

ゲーム用のウィンドウのサイズとゲームタイトルを入力する. (図 3 参照) 入力されたウィンドウサイズを元にウィンドウ作成の命令をファイルに出力. またゲーム記述言語で書かれたファイルを読み込み, 読み込まれた内容を元に表 1,2 に沿って変換しファイルに出力していく. 最終的にゲームソースが出力される.

表 1: ゲーム記述言語とメソッドの対応

ゲーム記述言語	メソッド	意味
POSITION	AdvStage	"AdvStage"の宣言
GRAPHIC	set_fore	表示画像の設定
TEXT	set_text	表示テキストの設定
LINK	next_stage_set	移動可能な場面の設定

表 2: "ACTION"の"Type"部とメソッドの対応

"Type"部	メソッド	意味
パラメータ名	p_change_パラメータ名	パラメータの加減算
flag_on	flag_on_point_set	指定フラグ"ON"
flag_off	flag_off_point_set	指定フラグ"OFF"
flag_check	flag_check_set	指定フラグのチェック
flag_go	sp_next_stage_set	フラグ"ON"時移動

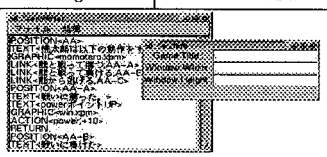

図 3: ゲーム記述言語の変換

4 評価

今回構築したゲーム開発支援ツールを使用することで, 一つの基本シナリオから複数のストーリを作成することを可能とし, さらにストーリ作成からゲーム完成までの流れを簡略化することで, ゲーム開発に必要な知識は少なくなり, また開発時間の大幅な削減を可能とした.

5 結び

今回の実験により, 単純なゲームにおいてはこのような手法が有効であることが分かった. だが複雑なゲームを作る場合, どこまでこの手法が有効であるかは不明である. そこで今後, イベントの数を増やしより複雑なゲームに対応できるかを検討していく.

参考文献

[1] 竹田 英二: GTK+ ではじめる X プログラミング, 技術評論社 (1999)
[2] 石井 達夫: PostgreSQL 完全攻略ガイド, 技術評論社 (2001)