Proc. of 1st IFIP Entertainment Computing Symposium "New Frontiers for Entertainment Computing" (ECS-2008), Sep. 2008, Milan, Italy, published in IFIP International Federation for Information Processing, vol. 279/2008, pp.105-115.

Comic Layout for Automatic Comic Generation from Game Log

Ruck Thawonmas and Tomonori Shuda

Abstract The paper presents our system for generating comics from game log. In particular, comic layout is focused. In order to achieve more comic-like expressivity, we extend an existing comic layout process proposed by Shamir et al. as follows. First, tiny frames are introduced for being placed vertically in the same row. Second, splash frames taking up space of several rows are introduced for emphasizing the corresponding frames. Third, slant frames are introduced for shooting events. Comic sequences generated with the proposed layout process and with the existing one are compared and discussed.

1 Introduction

Summary of user experiences in an entertaining fashion can help not only augmenting their personal memory but also promoting communication among user communities. Two representative applications are Nokia's Life Blog [1] and Microsoft's MyLifeBits [2]. Both enable individuals to record daily experiences in a form of multimedia contents such as photos and videos.

Our system targets user experiences in games, especially online games. The goal is to summarize player experiences in a game in a form of comics based on their game log. Comic-style representation has been used for summary of conference [3] and diary [4] experiences, as well as video contents [5]. Our work was inspired by a work in [6] that also aims at creating a sequence of comic-like images summarizing

Ruck Thawonmas
Intelligent Computer Entertainment Laboratory, Graduate School of Science and Engineering, Ritsumeikan University, Kusatsu, Shiga, 525-8577, Japan, e-mail: ruck@ci.ritsumei.ac.jp

Tomonori Shuda
Intelligent Computer Entertainment Laboratory, Graduate School of Science and Engineering, Ritsumeikan University, Kusatsu, Shiga, 525-8577, Japan

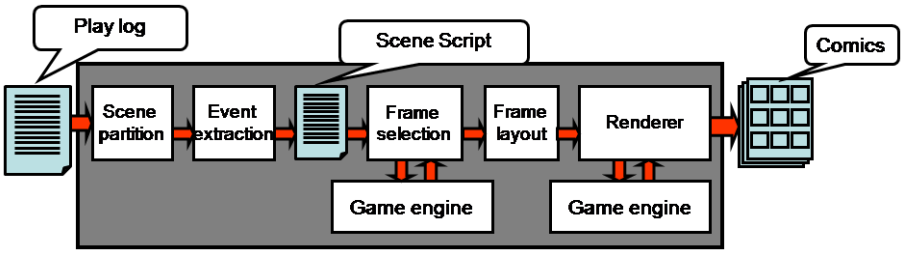


Fig. 1 Overview of the comic generation system.

main game events. The expressivity of comics generated in [6] is, however, limited mainly due to all rows being constrained to the same height.

This paper attempts to achieve more comic-like expressivity by extending the comic layout process in [6]. Our extensions are described in Section 3, after an outline of the comic generation system in Section 2. In Section 4, a comic sequence generated with the proposed layout process from an online game's log is compared to that generated with the layout process in [6] from the same log.

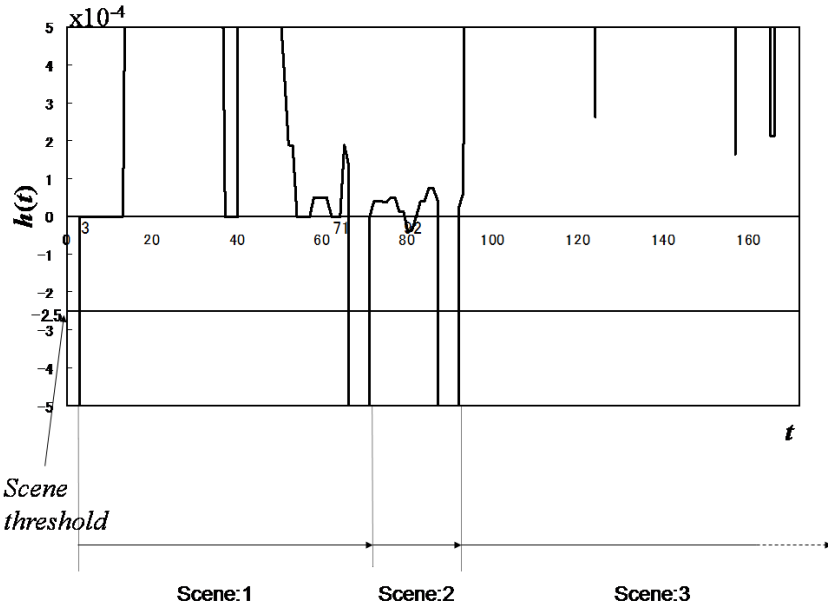
2 Comic Generation System

Figure 1 gives an overview of our system where the input is game log and the output is a resulting comic sequence. Game log consists of a sequence of actions, such as *shoot* and *talk*, recorded when an action is invoked by a player character (PC) or a non-player character (NPC) seen at the user's game client. It is stored at the client side. In order to obtain game events, the game log is processed by the scene partition module and the event extraction module. A game event is a part of play that has interaction level beyond a given threshold. Interaction level at time t, $l(t)$, is defined based on the importance of related actions and that of involving entities (PCs, NPCs, and objects) as

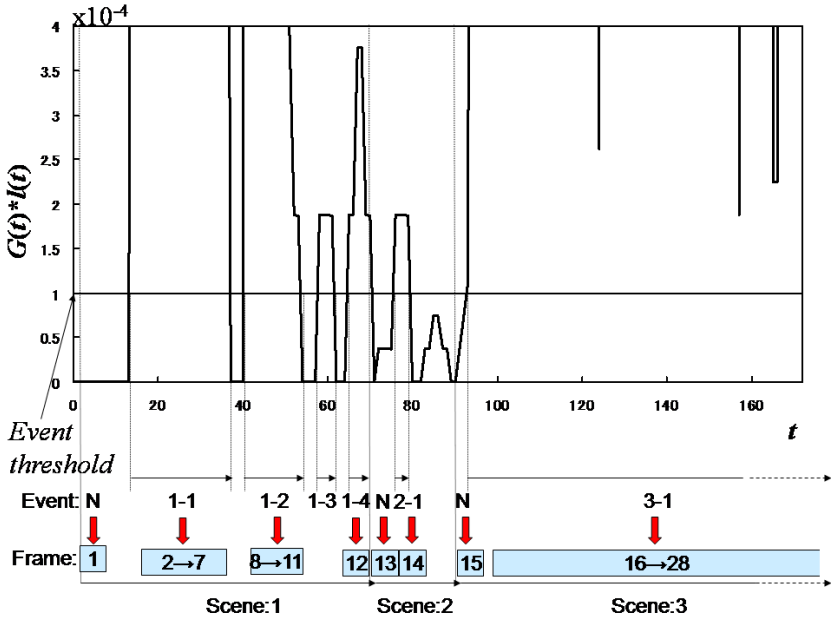
$$l(t) = \sum_{a} \sum_{e} \sum_{e'} w_a \max(w_e, w_{e'}) i(t, a, e, e'),$$

where a denote an action; e and e' denote entities; w_a, w_e, and w'_e denote the weights of a, e, and e', respectively, with the range from 0 to 1; and $i(t, a, e, e')$ is 1 if a is performed at t whose subject and object (if any) are e and e', respectively, and 0 otherwise.

The whole play is partitioned into multiple intervals called scenes. This is done at the scene partition module based on $h(t)$ (c.f., Fig. 2.a), a smoothed version of $l(t) - s(t)$, i.e., $h(t) = G(t) * (l(t) - s(t))$,where $G(t)$ is a Gaussian function and $s(t)$ is the number of moves conducted at time t. A scene starts at a point with rising $h(t)$ up from a scene threshold T_s and ends at the starting point of its subsequent scene. Next, for each scene, all intervals with Gaussianly smoothed value of $l(t)$,$G(t) *$



(a)



(b)

Fig. 2 (a) Scene partition based on $h(t)$, (b) event extraction based on $G(t) * l(t)$.

$l(t)$, beyond an event threshold T_e are considered events constituting the scene (c.f., Fig. 2.b). This is done at the event extraction module whose output is a scene script.

A scene script consists of a sequence of tuples of $(t, l(t, a, e, e'), a, e, e')$, henceforth called interaction tuples, and special tags for indicating the beginning and ending of each scene and each event. Each interaction tuple is considered a frame candidate for being selected at the frame selection module described in the next section. An event tag includes the information on an idiom used for defining rendering parameters such as the initial frame size, the camera target and camera position. Idiom assignment of an event is based on a majority voting of the action types in its tuples. In our system, five idioms are used, i.e., *New Scene*, *Talk*, *Approach*, *Shoot*, and *Mixed* whose initial frame sizes are big (B), fixed (F), neutral (N), N, and small (S), respectively, with $\mathrm{B} > \mathrm{N} = \mathrm{F} > \mathrm{S}$. In order to emphasize frames with relatively high interaction level, the initial size of the frame with the largest interaction level in each event is expanded one step, except for B and F frames. B frames are expandable, S frames are condensable, and N frames are both expandable and condensable.

3 Comic Layout

Some rendering information, such as a list of all game objects, PCs, NPCs in a frame candidate of interest, however, is not directly available in the scene script. This kind of information is obtained at the frame selection module by the game engine, simulating the corresponding play, and is augmented into frame candidates. Another technical issue is that adjacent frame candidates are usually similar, which decreases comic readers' interest. We solve this issue using Habituated Self-Organizing Map (HSOM) [7]. HSOM removes frame candidates that are similar to preceding ones within the same scene. The remaining candidates are selected for the layout and renderer modules. The effectiveness of using HSOM for this task will be reported elsewhere.

At the frame layout module, the width of each selected frames is repeatedly adjusted from its initial size, except for F frames, in order to achieve, say, *k* desired columns in a row, based on the layout process in [6]. The layout process in [6] constrains all rows to the same height. In our layout process, we set the height of each row in proportion to the width of the biggest frame in the row. In addition, we apply three layout techniques commonly used in real comics, namely, tiny frames (e.g. those two frames placed vertically at the top left corner of Fig. 4.b), splash frames (e.g. the frame at the bottom left corner of Fig. 5.a), and slant frames (e.g. the four frames placed at the middle and bottom rows of Fig. 4.b). Our layout process is described below as follows:

- **Step 1** Generate a new row, scan the selected frame sequence, and place *k* frames into the row. For this row, determine tiny frames, a splash frame, and all frame widths and heights as follows:

  - If this row has pairs of S frames, of N and S frames, or of F and S frames, the frames of the first pair will become tiny frames. Place them vertically in the row and consider them as one frame in determination of frame widths described below.
  - If this row has a B frame at the beginning, this frame will become a splash frame with the probability P_b. And for the newly defined splash frame, expand its height to cover several rows and associate it to the top row on its right side in determination of frame widths described below.
  - Apply the layout process in [6] for determining the widths of all frames in the row. If a splash frame exists, this task will also be done for the remaining rows on its right side. As in [6], if frame widths cannot be determined due to violation of their constraints on allowable widths, set their widths to the same default value, i.e., the width of the N frame.
  - Set the row height in proportion to the width of the biggest frame in the row. If a splash frame exists, this task will also be done for the remaining rows on its right side.

- **Step 2** Repeat Step 1 until the sizes of all selected frames are determined.
- **Step 3** Generate slant frames by alternatively upward slanting and downward slanting a row-partition line that satisfies all of the following conditions.

  - it is not the top or bottom page border,
  - there exists at least one *Shoot* frame adjacently above or under it, and
  - there are no *New Scene* frames adjacently under it.

Finally, the resulting comic layout is given to the renderer module for rendering comics. The game engine is used again here to simulate the play at the time of a frame of interest. Such a play is then rendered into an image. In order to achieve comic-like images, the grey-scale filter, median filter and Laplacian filter are applied to the background, characters, and objects in each frame accordingly.

4 Resulting Comic Sequences

We tested our comic generation system with an online game called the ICE [1], under development at the authors' laboratory, where typical online-game missions, such as monster fighting and item trading, are available. A screenshot of the ICE is shown in Fig. 3. The main parameters are $k = 3$, $T_s = -2.5 \times 10^{-4}$, $T_e = 1 \times 10^{-4}$, and $P_b = 0.5$.

Figures 4 and 5 show the first half and second half of a comic sequence generated with the proposed layout process, where frame numbers are superimposed. For comparison, Figs. 6 and 7 show the first half and second half of a comic sequence generated with the existing layout process in [6]. Both comic sequences were from the same game log whose scenes and events were derived in Fig. 2.a and Fig. 2.b,

[1] http://www.ice.ci.ritsumei.ac.jp/mmog.html

Fig. 3 The ICE screenshot.

respectively. The corresponding idioms and initial sizes of these 28 frames are as follows: 1: New Scene (B), 2-7: Talk (F), 8-11: Mixed (S), 12: Approach (N), 13: New Scene (B), 14: Approach (N), 15: New Scene (B), 16-19: Shoot (N), 20: Shoot (B), and 21-28: Shoot (N); note that the initial size of frame 20 is B expanded from N because it is the frame with the largest $G(t) * l(t)$ in event 3-1.

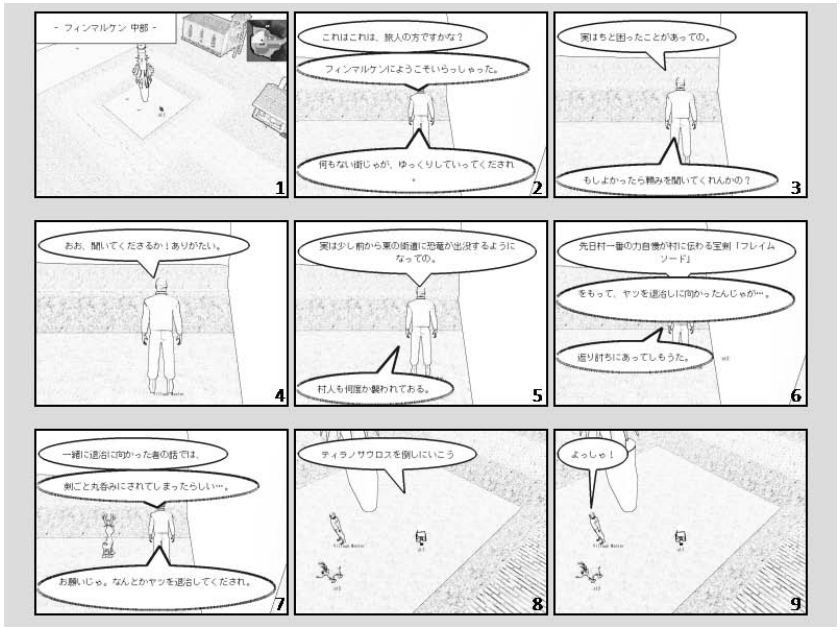
The tiny frames in Fig. 4.b, frames 10 and 11, result from the fact that they are two consecutive S frames. The splash frame in Fig. 5.a, frame 20, exists because it is the B frame located at the beginning of the middle row. Slant frames are seen in Figs. 4.b, 5.a, and 5.b because they are related to shooting events.

5 Conclusions and Future Work

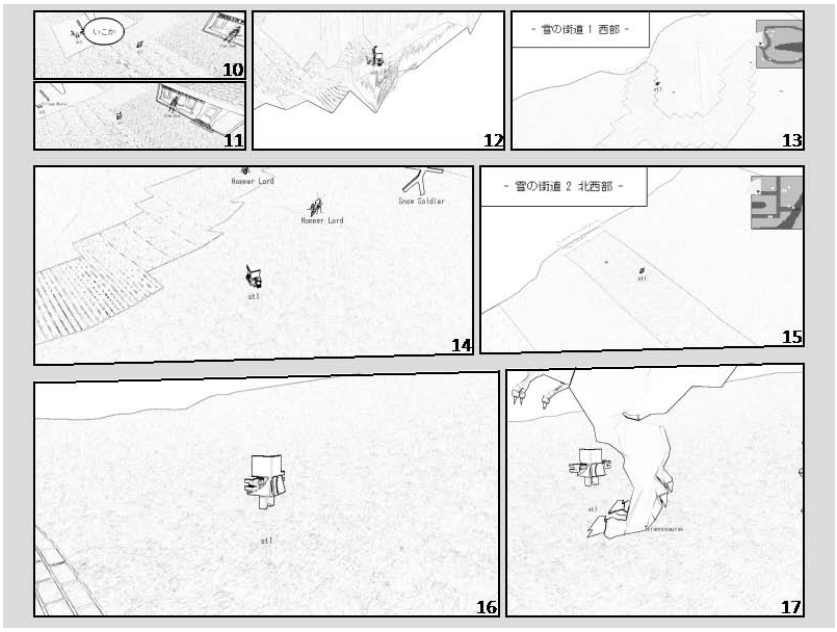
This paper described our system for generating comics from game log. To achieve more comic-like expressivity, three extensions to [6] on comic layout were presented. The resulting comic sequence is visually better than a fixed row-height style done in [6]. Our future work includes improvement on camera control, special effects, and story development.

References

1. http://r2.nokia.com/nokia/0,,71739,00.html
2. http://research.microsoft.com/barc/mediapresence/MyLifeBits.aspx
3. Y. Sumi, R. Sakamoto, K. Nakao, and K. Mase, ComicDiary: Representing individual experiences in a comic style, *Ubicomp 2002*, pp. 16–32, 2002.
4. S.B. Cho, K.J. Kim, and K.S. Hwang, Generating Cartoon-Style Summary of Daily Life with Multimedia Mobile Devices, *IEA/AIE 2007*, pp. 135–144, 2007.
5. J. Calic, D.P. Gibson, and N.W. Campbell, Efficient Layout of Comic-like Video Summaries, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17(7), pp. 931–936, 2007.
6. A. Shamir, M. Rubinstein, and T. Levinboim, Generating Comics from 3D Interactive Computer Graphics, *IEEE Computer Graphics and Applications*, vol. 26(3), pp. 53–61, 2006.
7. S. Marsland, U. Nehmzow, and J. Shapiro, A real-time novelty detector for a mobile robot, *EUREL European Advanced Robotics Systems Masterclass and Conference*, 2000.

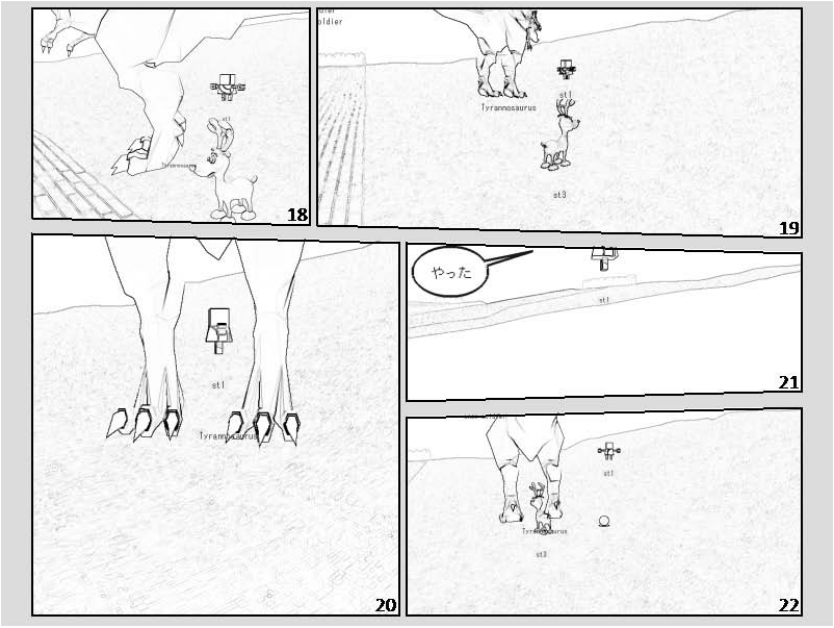


(a: page one)

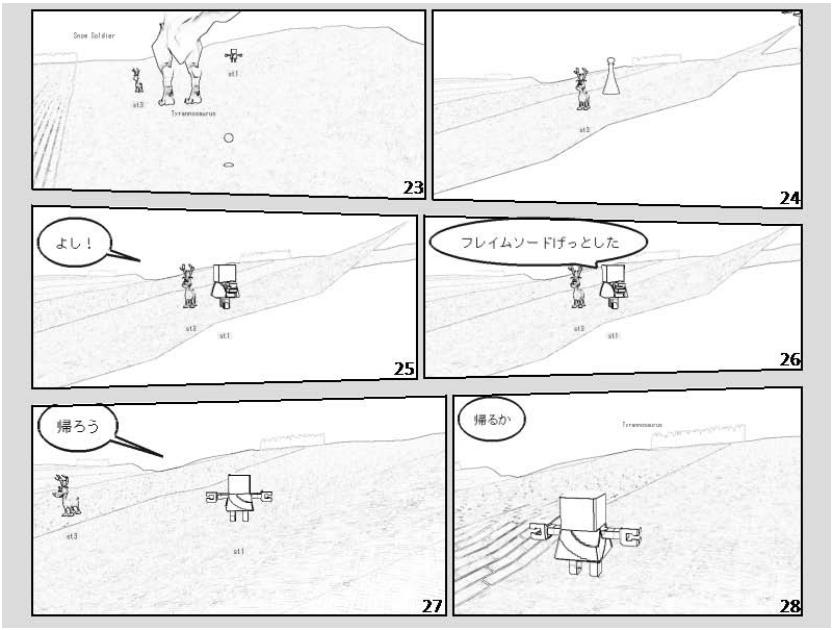


(b: page two)

Fig. 4 The first half of a comic sequence generated with the proposed layout process: (a) page one and (b) page two.

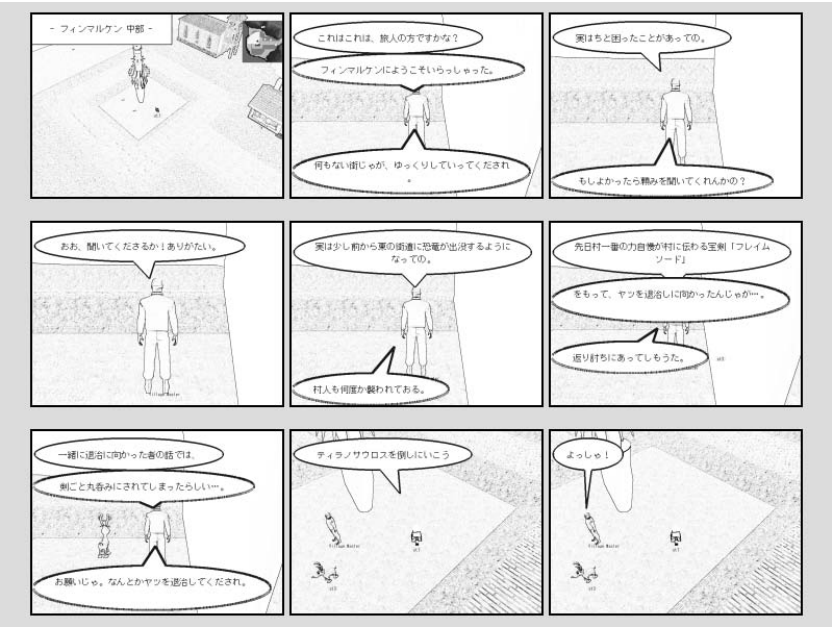


(a: page three)

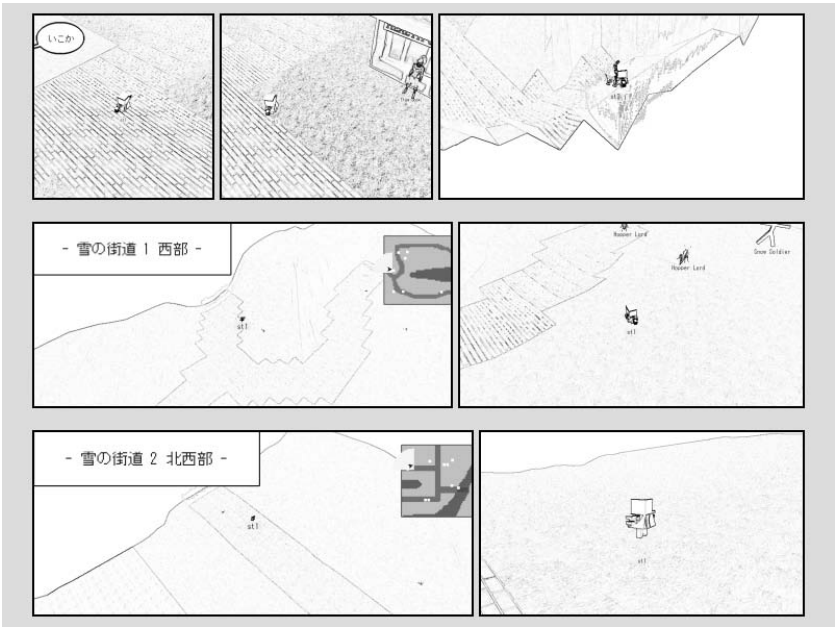


(b: page four)

Fig. 5 The second half of a comic sequence generated with the proposed layout process: (a) page three and (b) page four.

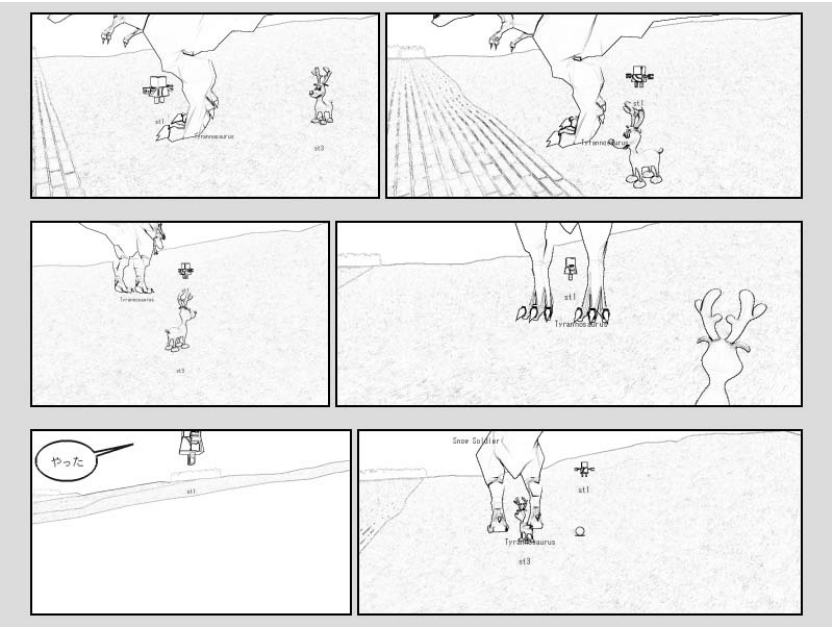


(a: page one)

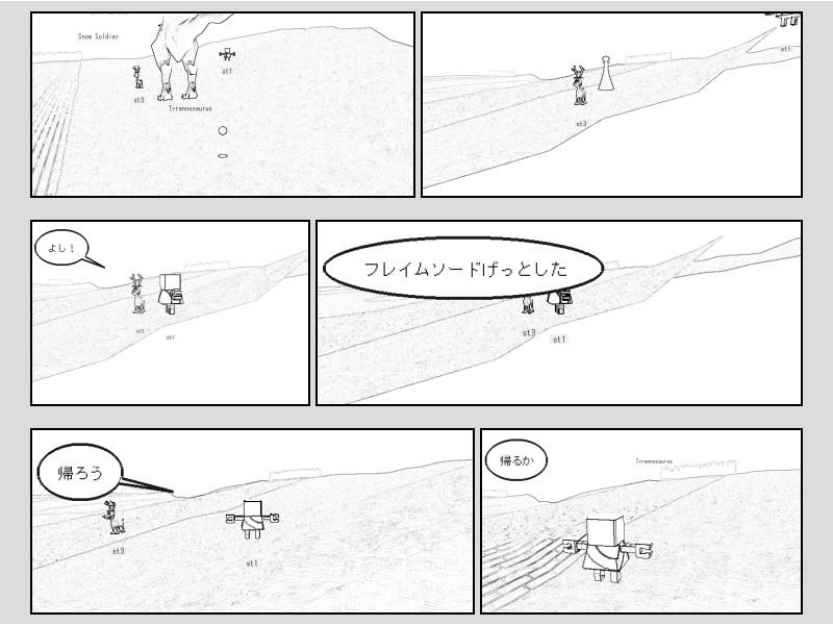


(b: page two)

Fig. 6 The first half of a comic sequence generated with the existing layout process: (a) page one and (b) page two.



(a: page three)



(b: page four)

Fig. 7 The second half of a comic sequence generated with the existing layout process: (a) page two and (b) page four.