A Learner-Centred Design Approach to Developing a Visual Language for Interactive Storytelling

Katherine Howland
Department of Informatics
University of Sussex
Brighton, BN1 9QH, UK
+44(0)1273 877218
K.L.Howland@sussex.ac.uk

Judith Good
Department of Informatics
University of Sussex
Brighton, BN1 9QH, UK
+44 (0)1273 873228
J.Good@sussex.ac.uk

Judy Robertson
MACS
Heriot-Watt University
Edinburgh, EH14 4AS, UK
+44 (0)131 451 8223
Judy.Robertson@hw.ac.uk

ABSTRACT
Creating interactive stories in the form of narrative-based games can have motivational and educational benefits for children, but scripting languages can be a barrier to the activity. This paper describes a learner-centred design (LCD) approach to creating a visual programming language (VPL) for scripting plot events in game-based interactive stories. The LCD approach, unusual in the field of VPLs, was essential to ensure that the software effectively supported our child target users in the game creation process.

Author Keywords
Learner-centred design with children, interactive storytelling, computer game design, visual programming languages.

ACM Classification Keywords
D.2.6 [**Software Engineering**]: Programming Environments - *graphical environments*; D.2.10 [**Software Engineering**]: Design Tools and Techniques; H.5.2 [**Information Interfaces and Presentation (e.g., HCI)**]: User Interfaces - *user-centered design*; K.3.1 [**Computers and Education**]: Computer Uses in Education.

INTRODUCTION
The educational potential of computer game play has long been recognised, but in recent years attention has widened to investigate game creation as a beneficial activity (e.g. [1,15]). Work in this area tends to be united in recognising the motivational benefits of game creation, but the learning affordances cited are wide ranging and differ between works. For some game creation is seen as a method of introducing children to computer science concepts [10, 9, 12], for others the focus has been investigating how creating games can encourage more sophisticated thinking about learning [6, 5, 4]. However, we believe that game creation can support storytelling particularly well. Creating an engaging interactive story within a game environment involves creating realistic characters, developing interesting plotlines and writing compelling dialogue. These skills are transferable to a number of creative writing activities. Game creation can also be a creative outlet for young people who have difficulty expressing themselves in more traditional creative writing tasks. Strong motivation for children to create an interesting and enjoyable interactive story comes with the knowledge that their game may be played by their peers. Receiving and taking into account feedback from peers also helps children to develop audience awareness skills.

It is only recently that software tools which make game creation an achievable activity for children have been available. Previously, creating a computer game required advanced technical skills which most children would be unlikely to have. Now a number of commercial games ship with toolsets which allow game players to become game creators, and some even allow users to create simple interactive worlds without any programming. However, whilst users without programming skills can create simple games, writing interactive stories often requires scripting due to the need to add more complex events to the game. This can cause frustration for children, and works against the usability of toolsets designed for novices.

This paper outlines a specific instance of this problem and describes a LCD approach to developing a solution. In the following section we outline the project background and discuss related work, we then give a detailed account of the design process, before describing the language developed. In the final section, some conclusions are given.

BACKGROUND AND RELATED WORK
We have been investigating the potential for game creation to improve children's storytelling skills at Gamemaker workshops over the past three years (see e.g. [18, 16, 17, 1]). The workshops, which have been attended by around 300 participants in total, give secondary school children (aged 12-15) the chance to create their own computer game. The software used at the workshops is the Aurora toolset which is distributed with BioWare's 3D role-playing game Neverwinter Nights (NWN). The toolset allows children to develop game modules which can be

played in NWN. Children can use the toolset's graphical user interface (GUI) to quickly create areas, objects and characters by dragging and dropping GUI elements. Wizards and menus allow the user to set up events to take place within the game, including interactive conversations between the player and a non-player character (NPC). Using these tools, children are able to create simple stories quickly and easily. Gamemaker workshop attendees have created games with elaborate plots, interesting characters, and sophisticated themes, with some participants writing extensive dialogues for their characters [17, 1].

However, there is presently an artificial limiter on the plot events which children can create using the toolset GUI. To create many plot events in a game (such as making an NPC react to the player trying to steal an object) it is necessary to use NWScript, an inbuilt scripting language based on the C programming language. For example, to create an event where a guard turns hostile and attacks the player when the player opens a forbidden chest, the following script would need to be written:

```
#include"nw_io_generic"
 void main()
        {
          object oPC = GetLastUsedBy();
          if (!GetIsPC(oPC)) return;
          object oTarget;
          oTarget = GetObjectByTag("NW_LUSKANITE");
          AdjustReputation(oPC, oTarget, -100);
          AssignCommand(oTarget,
          DetermineCombatRound(oPC));
          AssignCommand(oTarget, ActionAttack(oPC));
        }
```

Very few children of the target age range have the necessary programming skills to write a script such as this. Being unable to add an event which is important to the story they want to tell can be very frustrating for young people, something which was observed from the first Gamemaker workshops [16].

At present, when a child wants to create a plot event which requires scripting, workshop helpers have to write the script for them. This can cause children to lose a sense of ownership of their story, and can also prevent them from creating the story they would like, as helper time is inevitably limited. If our aim was to use game creation as an introduction to computer science it would be natural to teach scripting to the children as a solution to this problem. However, we are primarily interested in developing storytelling skills, and scripting would draw children away from the storytelling process and cause them to focus on low level issues of programming and debugging rather than the overall plot and storylines. So, an alternative was sought; a method of scripting plot events which was supportive of the creative process. Instead of a language which detracted from the story creation activity, we wanted to create a scripting method which would be in keeping with the way that target users naturally thought about the plot events they wanted to include in their games.

VPLs, which allow users to specify commands primarily graphically, and "natural programming languages" (NPLs), which allow users to specify commands in a way closer to their natural mode of expression, often aim to make programming more accessible and intuitive to children and other novice programmers. Show and Tell [8], an early example is a general purpose VPL designed for use by school children. More recently there has been growing interest in languages and environments which allow children and novice programmers to create simulations and animations. The Alice 3D Authoring system aims to make programming easier by allowing users to program objects in a 3D environment by dragging-and-dropping program elements rather than typing language syntax [7]. Other recent languages aimed at children include StarLogoTNG [9] and Stagecast Creator [19]. Languages of this type were sensible models to look to in developing our solution.

In order to ensure that the system under development effectively supported children in their story creation, it was essential that we involved the target users in the design process. A LCD approach, based on the CARSS framework described in [2], was adopted to ensure that the software provided the required functionality and was accessible for the target users. Although VPLs and NPLs have often been designed with children in mind, very few have adopted a user-centred design approach in the past. The main consideration in the design of these languages is usually programming theory, with little or no end-user input to the design process. A notable exception to this is Pane's HANDS programming system, the design of which involved detailed investigation of the way that non-programmers conceptualise algorithms [13, 14].

DESIGN PROCESS

Requirements Gathering

Before design of the system began, it was essential to identify the needs of the target users. First, existing data was analysed in order to determine the functional and usability requirements of the system. This was followed by the gathering and analysis of new data to give additional information on the usability requirements.

Analysis of Existing Data

In order to determine the appropriate coverage for the system, it was necessary to ascertain which story events requiring NWScript were most popular with children who have used the Aurora toolset to design their own interactive stories. Analysis was carried out on twenty-six NWN modules created at Gamemaker workshops. The scripts attached to each of these modules were noted and those that were responsible for plot events were recorded. All event components which appeared regularly in NWN modules created by target users were considered for inclusion in the system. The final decision over inclusion required consideration of the features currently available in the toolset. There are a number of wizards provided, which allow users to create plot events and simple quests without entering the NWScript by hand. The wizards have been cited by target users as tedious and time consuming to use, but the users are able to make use of them to add the required scripts to their games when necessary. Therefore, it was decided that the events dealt with by these wizards (such as quests and conversation-based events) were not a priority for coverage and would not be included in the version of the system under development.

In addition to establishing functional requirements it was important to establish suitable usability requirements for the software. The language under creation needed to be intuitive for target users, and to allow them to focus on the creative task at hand. Transcripts of interviews conducted at the end of a week-long Gamemaker workshop provided useful information about where the inbuilt toolset provision causes problems for our target users [16].

After attempting to create complex stories using the toolset, these participants were well placed to indicate the ways in which the current method of event creation failed to support them in the creative process. Scripting was unsurprisingly singled out by most participants as a particularly difficult aspect of module creation. Participants had not been able to create scripts for themselves, but many had watched and listened while workshop staff wrote scripts for them and explained the process. One participant cited a case where scripting was required and stated that a drag and drop system for setting up the event would be much easier to use than the current system. Calls for a less textual and more pictorial method of using the toolset also came from other sources. A different participant stated that certain aspects of the toolset could be improved by adding more pictures or icons to give the user a better idea of what the player would experience in the game. Even tasks which require a small amount of text to be inputted seem to cause problems for many target users. Cases where a specific textual identifier needed to be entered exactly were highlighted as difficult by participants as they found it hard to remember the identifier accurately. This problem was compounded by the fact that identifiers tend not to be natural language words, for example a textual identifier for a door might be “NWNvaultdoor2”.

Scripting and other text based interactions both introduce the possibility of syntax errors, something which can be particularly frustrating for children. The information gathered suggested that children had difficulty with the more complex aspects of the toolset because the advanced features involve a move away from the pictorial interface which makes the basic features of the toolset so intuitive for children to use. It was therefore decided that the method of story event construction should be primarily graphical, with the system eliminating the need for users to type in textual identifiers.

Gathering New Data

At this stage we had gathered detailed information about the ways in which the current tools conflict with target users underlying thought processes about game creation. We now needed to find out more about these underlying thought processes so we could better support them. In order to ensure that the general purpose natural programming language, HANDS, was usable by the target users Pane examined the way in which children without programming experience constructed solutions to algorithmic problems in order to better understand the way they naturally thought about programming concepts [13, 14]. We adopted a similar approach to determining usability requirements for our system. In our case the language under development had a very specific application, and as such we focussed our attention on the way in which target users conceptualised the creation of plot events in their games.

We organised requirements gathering workshops to collect information on the way target users verbalise their ideas about the story events. Eleven children aged 11-12 took part in a series of after school workshops which consisted of four ninety-minute sessions. Requirements gathering occurred during the first three sessions. This took place concurrently with teaching participants how to use the NWN Aurora toolset to create their own interactive stories. These training activities were key to the process, as for the target users to contribute effectively to the design process, they needed specialised knowledge and skills. As participants began to learn the toolset basics, we were able to observe their interactions with the toolset and record the way which they described plot events in conversation.

Requests for help provided the most useful information about the way target users conceptualise story events. One common request was for a non-player character to join the player’s team as a ‘henchman’, and the way participants asked for help with this is indicative of the general trends observed. The participants typically asked questions such as “How can I make this Dragon help me?” and “How can I make this guy fight on my side?” They described in simple natural language what they wanted to happen from the player’s point of view whilst identifying the objects involved by pointing at them on screen.

This type of description was common to most requests for help, as was the tendency for participants not to initially state when and under what conditions they wanted the event to occur. It often emerged that the participant saw these as secondary details about the event. There were exceptions to this behaviour, but in general the variation was observed between different types of event rather than between different participants (for example, the condition was recognised as important where the story event was the player getting a reward after completing a task).

Established Requirements

The data collected provided important information about the way our target users thought about game creation tasks and allowed us to establish the key requirements for a supportive scripting tool. It was decided that users should be able to identify game objects by recognition and selection of a primarily graphical interface object with no entering of textual identifiers required. Participant requests for help indicated that target users first think of the event that they want to take place, and often need to be prompted to explain the conditions for the event occurring. It was decided that this 'prompting' should be built into the system, with the user first being asked to state which event should take place, and then being prompted to state when the event should occur along with any conditions on its occurrence. By asking users to select and insert event components into a number of predetermined structures, we hoped to scaffold the story creation process and eliminate errors due to incorrect structure.

The information gathered about the needs of target users also had implications for the method of interaction between the system under development and the NWN module. A system which allowed the creation of NWScript using a graphical interface, but required the composed NWScript to be attached to the correct object in the module would not be appropriate. This is a fairly complex process which requires some knowledge of NWScript and the game file formats. The interview data made it clear that encounters with non-natural language text was difficult for target users, so it was decided that our system should allow users to create story events without coming into contact with NWScript, and that the entire process should be carried out in a primarily graphical interface.

Interface Design

We designed a semi-formal visual language (a language which supplements a visual representation of commands and objects with a natural language element). The metaphor of a set of cards was chosen as the basis for the interface design with some of the key usability requirements in mind. This metaphor was seen to be particularly suitable for satisfying the requirement that story event construction should be primarily graphical, and the requirement that users should construct story events by adding components to an existing structure. Children of the target age range are likely to be comfortable with dragging cards to different positions on the screen from playing popular computer-based card games. The concept of cards only 'fitting' in certain slots depending on the category they belong to is also one that target users are liable to be familiar with, and was used to provide scaffolding in the event creation process.

Low-Fidelity Prototype

To aid the design of the interface a low-fidelity prototype was constructed according to an initial design. The prototype consisted of laminated cards which could be moved around and attached to a base by Velcro as shown in Figure 1.

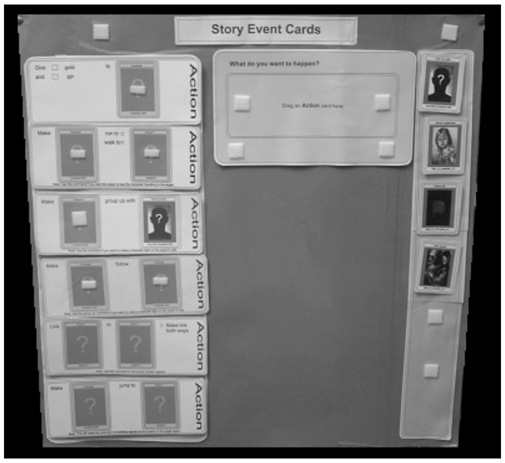


Figure 1: Low-fidelity prototype

The low-fidelity prototype was tested at the final after-school workshop in order to assess how well target users could use the proposed interface to create story events. Nine participants aged 11 and 12 took part in the testing of the prototype and were set two tasks to complete using the card based interface. The tasks were designed to test whether children of the target age range could construct a representation of a story event described in natural language using the proposed interface. The first task required the child to construct a sequence of two actions, and the second task required construction of a conditional event. Participants were observed completing the tasks and then interviewed to further determine the ease of use of the system and highlight any difficulties experienced.

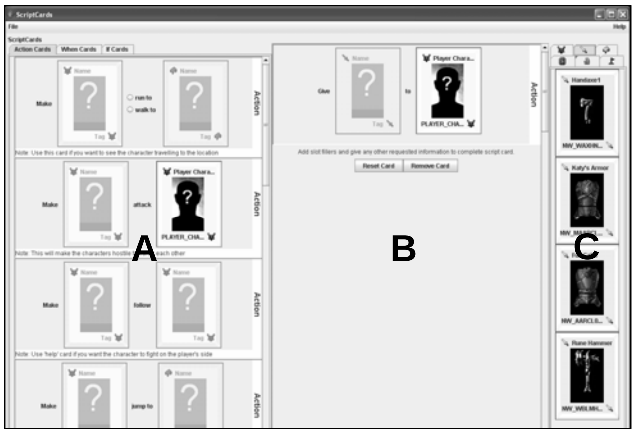
Most participants were able to use the prototype successfully, with a number making unprompted comparisons with the current toolset facilities and stating that the prototype was much easier to use. However, it emerged that the scaffolding incorporated into the design was too restrictive in some cases, as the preferred order for creating commands changed depending on the participant and task. For example, contrary to the conclusion drawn in requirements gathering, target users sometimes preferred to state the event conditions before stating what actions should occur. The system scaffolding was adapted to be more flexible in allowing for different preferences in the final design.

THE SCRIPT CARDS ENVIRONMENT

A high-fidelity system prototype was implemented in Java to meet the established functional requirements and the modified usability requirements.

The Script Cards Interface

In the Script Cards environment, sets of cards are manipulated to produce scripting commands. Events are constructed using building block cards referred to as *script cards*. There are three different types of script cards: *action cards*, *when cards* and *if cards*, representing action statements, when statements and if statements respectively. An array of action cards can be seen on the left-hand side of the interface in Figure 2, note the tabs for the other varieties of script cards.



A: **Script Cards Panel**: Currently displaying *action cards* ready to be used in a plot event.

B: **Composer Pane**: One *action card* has been added to the pane. The user now needs to fill in the ‘object’ slot by dragging over a card from the *slot filler card* panel. They can then drag over further *script cards* to complete the plot event.

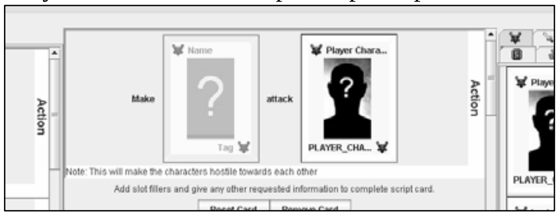
C: **Slot-filler Cards Panel**: Currently displaying *item cards* to be used filling the slots in *script cards*.

Figure 2: The Script Cards Interface

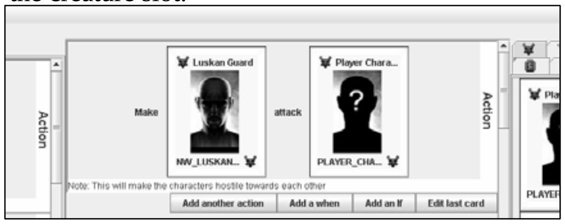
Script cards represent an underlying command which is communicated as a natural language expression with slots missing where a specific object needs to be referred to. To compose a plot event, the user drags a script card over from the panel on the left to the composer pane and fills in the blank slots by dragging and dropping appropriate *slot-filler cards* over from the panel on the right. The six different categories of slot-filler cards, which represent in-game objects, are *creature cards*, *item cards*, *placeable-object cards*, *door cards*, *merchant cards* and *waypoint cards*. A selection of item cards can be seen on the right-hand side of the interface in Figure 2, note the tabs for other varieties of slot-filler cards. Slot-filler cards portray an object primarily graphically by displaying the object’s associated in-game ‘portrait’. The object’s name and textual identifier are also displayed on the card to allow distinction between objects with the same appearance. The system scaffolds the user by only allowing cards of the correct type to be placed in a slot on the script composer pane, or within a script card. The cards and slots are colour coded and use icons to give the user additional visual clues as to where a card will fit. This ensures that only well-formed story events can be created, and eliminates the possibility of syntax errors. Some script cards have slots already filled in with a ‘Player Character’ creature card, this is the case wherever the player character is the only possible creature who can perform a certain action (e.g. use an item).

Earlier we saw an example script for a plot event where a guard turns hostile when the player opens a chest. Figure 3 shows the steps needed to create the same event using the Script Cards environment.

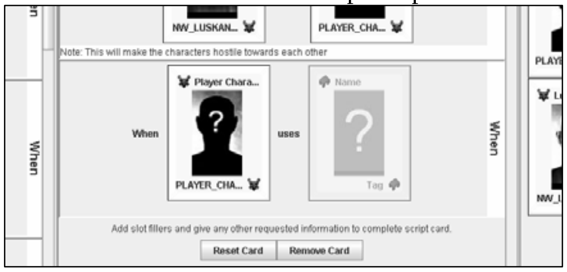
1. User drags *action card* ‘Make <creature> attack Player Character’ to script composer pane.


Action
Make
Name
attack
Player Chara...
Tag
PLAYER_CHA...
Note: This will make the characters hostile towards each other
Add slot fillers and give any other requested information to complete script card.
Reset Card
Remove Card

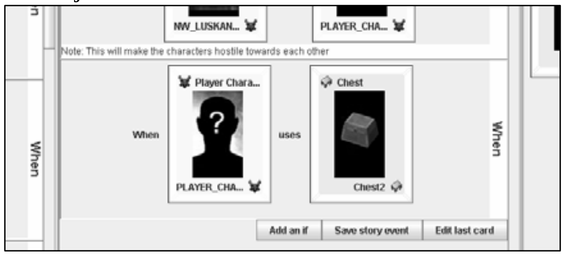
2. User drags *creature card* ‘Luskan Guard’ over to the creature slot.



3. User clicks ‘Add a when’ and drags when card ‘When Player Character opens <placeable object>’ over to the new slot on the composer pane.


NW_LUSKAN...
PLAYER_CHA...
Note: This will make the characters hostile towards each other
When
Player Chara...
uses
Name
Tag
Add slot fillers and give any other requested information to complete script card.
Reset Card
Remove Card

4. User drags placeable object card ‘Chest’ over to the object slot.



5. User clicks ‘Save story event’.

Figure 3: Adding an Event

System Level

In order to achieve direct interaction between the software and the NWN module, as specified in the established requirements, the system made use of a Java package which facilitates writing data to NWN modules. The package, 'Nwnintf', was created by the University of Alberta Artificial Intelligence Scripting Group during the development of their ScriptEase software (a menu-based scripting tool created with professional game designers in mind). [11].

The high fidelity prototype was designed to allow the construction of sequences in any sensible form. This behaviour was achieved by configuring the script composer pane to adapt its behaviour according to the script cards a user adds to it. Using Java drag and drop support, the script composer pane was set up so that only certain script cards could be 'dropped' onto it at a given time. Initially the user can lay any type of script card, but after the first card has been placed the system begins to make deductions about the structure of the story event under construction. Once a script card has been placed options appear beneath it on an intermediary panel which asks the user to select the type of card they would like to add next. Only options which will lead to a well-formed story event are displayed. For example if a when card has already been placed there will be no option to add another when card, because a story event can only have one trigger for its occurrence. Once an option is chosen, the script composer pane is configured and displays information about the type of script card which can currently be laid. The user can then drag over a script card of the appropriate category to continue construction of the story event. This configuration allows users to construct story events in the order which makes most sense to them, whilst still scaffolding to ensure that nonsensical story events cannot be created. All user actions are easily reversible through buttons which allow removal of the current script card from the panel, and resetting of the script composer pane to allow a different type of script card to be laid or to start a new story event. This allows the user to go back as many steps as necessary to correct mistakes.

The Java Advanced Imaging package was used to enable the display of a recognisable portrait-style picture on slot-filler cards for every possible object a user could create using the toolset. This was key to meeting the requirement of allowing users to refer to a game object by recognising and selecting it, thus eliminating the need for entering textual identifiers. Scaffolding of the slot-filling process is again provided by the Java drag and drop facility; it allows 'dragging' of any slot-filler card but at the 'dropping' point only allows the addition of cards of the correct category.

Evaluation

Target users also had a key part to play in evaluating the high-fidelity prototype system. A testing session took place at the end of a week-long Gamemaker workshop. Ten participants aged 13-15 took part in the session, all of whom had learnt how to use the toolset to create basic games during the workshop. Users tested the Script Cards environment by undertaking three exercises of graduating difficulty using the system. The tasks set were designed to test whether the system would allow users to create correct scripts given a natural language description of a story event. The exercises were documented by video footage and notes on performance, all requests for help and instances of exercises being completed incorrectly were recorded. A direct comparison of Script Cards with NWScript was not possible because only one of the participants had been able to make any attempt to use the textual language.

Exercise set	Notes
1. Storyline: A black dragon guards the key to a secret room. The dragon is friendly until you pick up the key from beside him, then he turns hostile and attacks you! Task: Using ScriptCards make the black dragon attack the player when the player gets the secret room key.	Requires creation of an event which occurs when the player obtains a specific plot item.
2. Storyline: In the secret room there is a portal which can transport you to a new land. Task: Using ScriptCards make it so that when the player uses portal1 the player jumps to portal2.	Requires creation of an event which occurs when the player uses a specific plot object.
3. Storyline: The portal can only be used by someone wearing the magic travelling necklace. Task: Using ScriptCards make it so that when the player uses portal1 the player jumps to portal2, but only if the player has the travelling necklace equipped.	Requires creation of an event which occurs when the player uses a specific plot object but is conditional on the player wearing a plot item.

Table 1: Testing Exercises

The exercises used for testing were analogous to the exercises used in the testing of the low-fidelity prototype. The participants in the high-fidelity prototype testing were older than the participants in low-fidelity prototype testing, and had more experience with the toolset. In addition most of the participants had witnessed scripts being written for their modules by workshop staff and had developed a very basic understanding of the process. As a result, the three tasks used here were of a higher level of difficulty than the two used in low-fidelity prototype testing. Details of the exercises set can be seen in Table 1.

A whole group demonstration of Script Cards was given on the morning of the day of testing. The demonstration showed how to create a story event of the form ‘sequence <when> event’ by means of an example different from those to be used in the tasks set. The module being edited was then opened up in the NWN Aurora toolset so the participants could see that the creation of an event in Script Cards had caused a script to be added to one of the script slots in the module. Finally the module was opened in the game so participants could see the effect of the script that had been added on the experience of the player.

Task completion rates indicated that comprehension of the system functionality, and ease of use, were quite high. Out of thirty tasks attempted twenty-seven were completed successfully, a particularly positive success rate considering that the participants had not used the system before undertaking the tasks. Of the three tasks completed incorrectly one was due to a user selecting the wrong radio button on a card. The other two incorrect task completions seemed to be due to a weakness in the language interface, a possible case of what Green terms ‘premature commitment’[3]. When creating a story event the user must first select a script card and drag it to the composer pane without having the opportunity to test whether the slot-filler cards they have in mind will actually fit in the script card. Although the icons and colour-coding guide the user as to where cards will fit, some participants preferred testing a fit via the drag and drop mechanism. These participants found that they did not have this resource open to them early enough to make appropriate decisions.

The changes to the interface design brought about after the low-fidelity prototype testing proved to be successful. Participants made use of being able to construct story events and add slot-filler cards in their preferred order, something which the scaffolding in the original design would not have allowed.

Participants were interviewed after completing the prototype tasks to allow us to gain further insight into their experiences with the system. Participants generally stated that they found the language very easy to use, and in particular thought it was much easier than what they had seen of the inbuilt NWScript editor. One participant, who is dyslexic, stated “I can’t be bothered [with scripting using a text editor]” and added that he thought Script Cards was “a lot better than going ‘blah, blah blah’ ‘oHenchman’ ‘oMaster’” making reference to the variable names required in the scripts which workshop staff had created for him. Two other participants also made reference to the specialist language and syntax required to create scripts with a text editor. One complained about the “funny words, like ‘object’, and computer talk” which are required to create scripts by hand, and the other mentioned that there is no need for “special words” with Script Cards as “you just use simple sentences”. Two participants stated that Script Cards seemed to make scripting faster as well as easier, and one of these went on to say that it seems like “you make less mistakes, and if there is a mistake you can fix it a lot easier”. When asked to elaborate on this point he explained that when scripting in a text editor “you have to put in exact words, all the grammar, you make one mistake and it’s all a waste of time. With [Script Cards] it just does all that for you, you just need to choose what it is and where it goes”. A further participant noted that you need specialist programming skills in order to use a text editor, whereas with “[Script Cards] it’s really easy, you can use it first time”.

CONCLUSIONS

Script Cards is a semi-formal VPL for scripting plot events when using the NWN Aurora toolset. It allows young people to use a combination of graphics and natural language descriptions to express plot events in a way which is as close as possible to the way they describe these events to others. An evaluation of Script Cards suggests that the prototype system achieved its aim, and was both easy to understand and use by young people of the target age range.

The LCD approach adopted in this project was crucial to its success. The system allows users to create plot events without entering any code or textual identifiers. This requirement was established through analysis of target users’ comments about the difficulties they had had with the existing tool provision. During the system evaluation the most commonly cited benefit of Script Cards was the fact that there was no need to enter any technical language, indicating that this was a highly valued feature of the system.

One of the main strengths of Script Cards lies in the fact that young people can express their story events in a way that comes naturally to them. This was achieved by working with target users to gather information about the way they conceptualise the story creation process. Script Cards allows users to create events by arranging components in the order they feel most comfortable with (an option not available with most conventional programming languages). This facility was only built into the system design after low-fidelity prototype testing indicated that target users differ in the way they like to order sequences. It proved to be a successful feature which was made use of by many participants in the system evaluation. The system strikes a balance between allowing young people this freedom in expressing their plot events, and providing sufficient scaffolding so that young people do not create uninterpretable scripts.

The most successful features of Script Cards can be clearly traced back to input from target users throughout the design process. It is surprising that LCD is so infrequently used in the design of programming environments aimed at children. It is an approach we which think should be very beneficial to anyone designing a language or programming environment which aims to support children in educational activities.

Development of Script Cards is ongoing. We are currently considering extending the scope of the system to cover a wider range of possible plot events. In addition, we plan to assess the usability of the Script Cards interface using Green's Cognitive Dimensions of Notations [3]. This would allow us to investigate the issue of premature commitment and diagnose any other problems in the interface. All further development will continue to use an LCD approach.

ACKNOWLEDGMENTS
We would like to thank the pupils and staff at Thomas A Becket Middle School and all Gamemaker workshop participants.

REFERENCES

1. Good, J. and Robertson, J. (2006). Learning and motivational affordances in narrative-based game authoring. In the *Proceedings of the 4th International Conference for Narrative and Interactive Learning Environments (NILE)*, pp. 37-50.
2. Good, J., and Robertson, J. (2006). CARSS: A Framework for Learner Centred Design with Children, *International Journal of Artificial Intelligence in Education* vol.16, no. 4, pp. 381-413.
3. Green, T.R.G & M. Ptre (1996) Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework, *Journal of Visual Languages and Computing, 7*, pp. 131-174.
4. Habgood, M.P.J., Ainsworth S.E.& Benford, S. (2005) The educational content of digital games made by children. Presented at *CAL '05: Virtual Learning?*, Bristol, UK.
5. Harel, I. & Papert, S. (1991) Constructionism. Norwood, New Jersey: Ablex Publishing Corporation.
6. Kafai,Y. B. (1995). *Minds in play: Computer game design as a context for children's learning*. Mahwah, NJ: Lawrence Erlbaum.
7. Kelleher, C., Cosgrove, D., Culyba, D., Forlines, C., Pratt, J.and Pausch, R. (2002). Alice2: Programming without Syntax Errors. *User Interface Software and Technology*. Paris, France.
8. Kimura, T.D., Choi, J.W. and Mack, J.M. (1990).Show and tell: A visual programming language. In *E.P. Glinert, (Ed.) Visual Programming Environments: Paradigms and Systems*, pp. 397-404.
9. Klopfer, E. and A. Begel (In Press). StarLogo TNG. An Introduction to Game Development. In Press for *The Journal of E-Learning*.
10. Korte, L. (2006). Constructive Game-Based Learning: Encorporating Narrative into the Theoretical Computer Science Curriculum. In the *Proceedings of the 4th International Conference for Narrative and Interactive Learning Environments (NILE)*, p. 113.
11. M. McNaughton, M. Cutumisu, D. Szafron, J. Schaeffer, J. Redford, D. Parker (2004). ScriptEase: Generating Scripting Code for Computer Role-Playing Games, *IEEE International Conference on Automated Software Engineering (ASE'04)*, pp. 386-387.
12. Overmars, M. (2004) Teaching computer science through game design, *Computer, vol. 37, issue 4, April 2004*, pp.81-83.
13. J.F. Pane, C.A. Ratanamahatana, and B.A. Myers (2001). Studying the Language and Structure in Non-Programmers' Solutions to Programming Problems, *International Journal of Human-Computer Studies*, vol. 54, no. 2, February 2001, pp. 237-264.
14. J.F. Pane, B.A. Myers, and L.B. Miller (2002). Using HCI Techniques to Design a More Usable Programming System, In proceedings of *IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC 2002)*, Arlington, VA, September 3-6, 2002, pp. 198-206.
15. C. Pelletier and A. Burn. (2005) Making games. developing games authoring software for educational and creative use. Presented at *CAL '05: Virtual Learning?*, Bristol, UK.
16. Robertson, J. and Good, J. (2005). Children's narrative development through game authoring. *TechTrends*, *49*: 43-59.
17. Robertson, J. and Good, J. (2006). Supporting the development of interactive storytelling skills in teenagers. In Z. Pan et al. (Eds.): *Edutainment 2006, Lecture Notes in Computer Science 3942*, pp. 348-357.
18. Robertson, J. and Good, J. (2004) Children's narrative development through computer game authoring. *Interaction Design and Children 2004*, Washington DC, USA. 57-64 .
19. Smith, D.C., Cypher, A., and Tesler, L. (2000). Programming by example: Novice programming comes of age. *Communications of the ACM*, 43(3), 75-81.