Flip: a visual and textual programming environment for teaching computation through game creation

Flip is a programming environment designed to help 11-14 year olds develop computational skills whilst creating their own 3D role-playing games. The combination of a visual language (based on an interlocking blocks design) and a natural language description of the script under creation is intended to support young people in developing an understanding of computational concepts as well as the skills to use these concepts effectively. Situating the activity in the context of creating a game with impressive 3D graphics makes this a motivating way to introduce young people to computation. Here we outline the background to the project, describe the participatory design activities we have undertaken in order to create and refine our design, and give a detailed description of the environment.

Background

Learning to think 'computationally' has long been recognised as important (Papert, 1980), and the recent computational thinking drive has refocused attention on this as a significant issue in modern computing (Wing, 2006). There is broad agreement that it is important to teach computational thinking skills from a young age, and to people who may never learn to program (Guzdial, 2008; Fletcher and Lu, 2009), but deciding which specific skills should be taught is still an emerging endeavour. Perhaps the most fundamental of the evolving set of computational thinking skills is the ability to define clear, specific and unambiguous instructions for carrying out a process: it is the very basis of computer programming, and its applicability to almost every domain of endeavour is easy to appreciate, as we have argued previously (Howland, Good et al., 2009). Flip aims to help young people develop this ability by scaffolding them as they script events as part of the process of creating their own computer game.

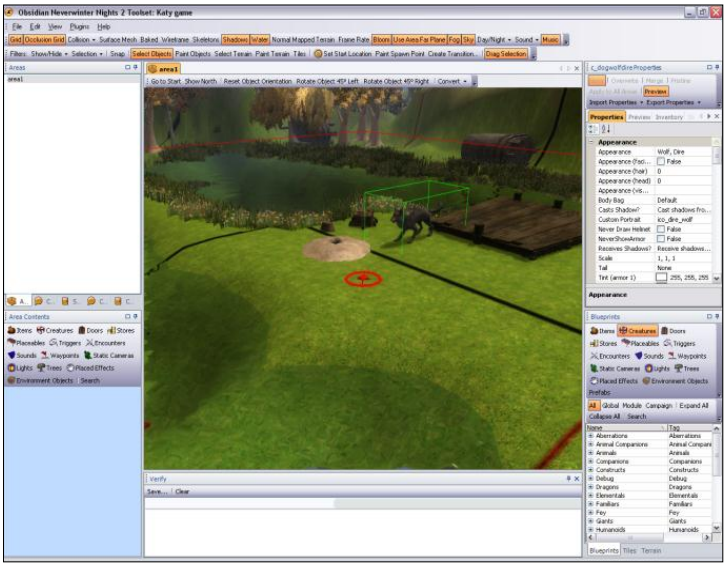
Over the past 10 years, our research has focused on empowering young people (aged 10-15) to work with game-creation software to develop their own commercial quality video games, as a creative and technical exercise (Good and Robertson, 2006; Howland, Good et al., 2008). We have found game-creation to be highly motivating for young users, who are drawn to the activity for a variety of reasons including personal interests in computer games, art and design activities or telling stories. As such it provides an ideal context for introducing them to the often difficult topic of programming, and the related computational thinking skill of rule specification.

We work with commercial computer games software called Neverwinter Nights 2 (NWN2), published by Atari in 2006. NWN2 is a computer role-playing game (RPG), in which players explore a large fantasy world and take part in a dramatic interactive story, with their own choices having an effect on how the plot progresses. The game is part of the Dungeons & Dragons franchise, a series of 'medieval fantasy' RPGs. Included in the NWN2 software package is the Electron Toolset: a professional game-development application which was used by developers, Obsidian Entertainment, to build the game itself. This means that, in addition to playing through the included game, users can go on to build their own video games using the developers' tools and art resources to design new 3D worlds and adventures. These user-created games feature new plots, dialogues, locations and challenges, but retain the same high-quality graphics and complex game mechanics one would

expect from a commercial product. Although there are many complexities to working with this toolset, around five minutes of instruction is enough to empower users to begin independently working on their own 3D landscapes, complete with rivers, trees, buildings, characters, and other points of interest.

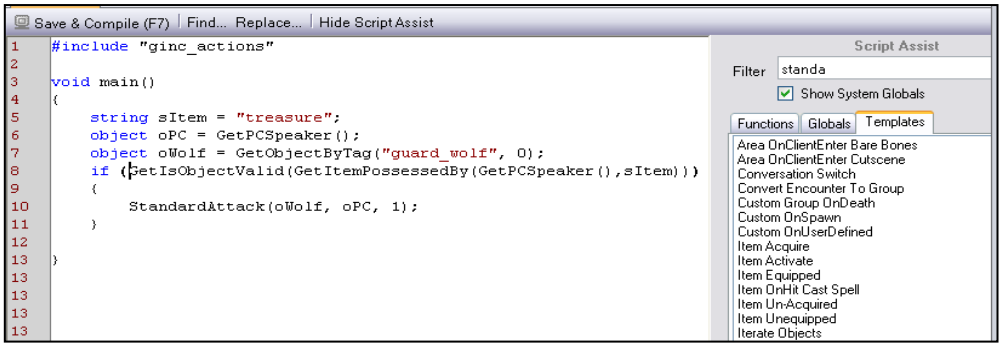
As such, unlike most game-based programming environments such as AgentSheets (Repenning, Ioannidou et al., 2000), Scratch (Maloney, Kafai et al., 2008) and Alice (Kelleher, Cosgrove et al., 2002), including those designed more specifically to support storytelling (Kelleher and Pausch, 2007), it is not immediately necessary to engage with computer programming in order to create something meaningful and impressive. Instead, users are able to explore the possibilities of the toolset, and use it to build something which appeals to their own sensibilities and which they feel ownership over, approaching game building first and foremost as a creative task.

This initially shallow learning curve is invaluable for motivation and gives users the confidence to persevere with the activity. When they ultimately do need to engage with programming in order to specify custom behaviour for the objects in their game, their investment in their game world and the narrative they are constructing often gives them a drive to try to achieve programming outcomes. A quest to steal the dragon's treasure is not much fun if the dragon has not been told to defend it! The behaviour of the dragon, or any other object in the game, forms a key part of the narrative the user is constructing, and so the game will be incomplete until the user can achieve at least some level of success with programming.



Electron Toolset

Due to the interactive nature of the games under creation, a number of computational concepts are introduced quite naturally to the users. Additionally, because the activity is learner-led and the game ideas are learner-generated it is likely that these concepts will be easier to grasp than when introduced in a more abstract manner. However, engaging with and mastering these concepts at present requires use of the toolset's inbuilt scripting language. The language used in conjunction with the Electron Toolset is called NWScript, and is similar to C in its syntax and level of complexity. Young users are almost invariably intimidated and frustrated by this element of the game creation process, and are unable to proceed without substantial help from experts.



NWScript Example

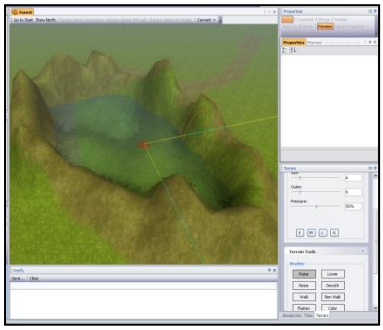
It is in this context that we seek to encourage and scaffold young people's attempts to add custom behaviours and events to their games by replacing the inbuilt scripting language with Flip, a visual language which aims to help users develop computational skills and understanding. Through the numerous game creation workshops we have been involved with, we have anecdotal evidence that young people can specify computational rules whilst describing narrative gameplay elements in the context of an informal conversation. However, users often require prompting from workshop staff before they fully state the conditions and actions involved, meaning that they face difficulties even before they hit the barrier of the intimidating syntax of NWScript. Flip supports young people in moving from their intuitive understanding of narrative events in gameplay towards a computational understanding. In this way we introduce computational concepts through a motivating activity, and offer a route to computation through the narrative understanding which users have already developed about their game.

Flip's role in the game creation process

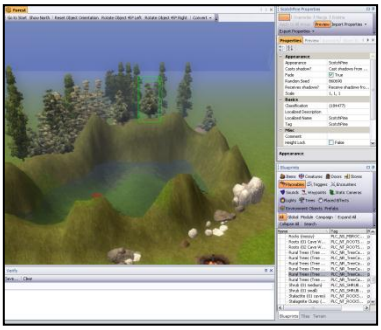
The Flip environment is a plug-in to the existing Electron Toolset which replaces the inbuilt NWScript environment. There follows an overview which explains how some key game creation tasks are carried out, where scripting is required, and hence where Flip fits in.

Creating environments

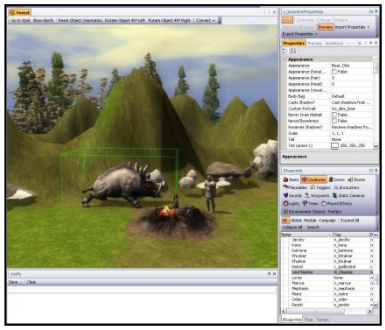
The toolset GUI allows users to create rich 3D environments and populate them with scenery and characters – this aspect of the game creation process does not require any scripting.



Landscaping areas with terrain tools



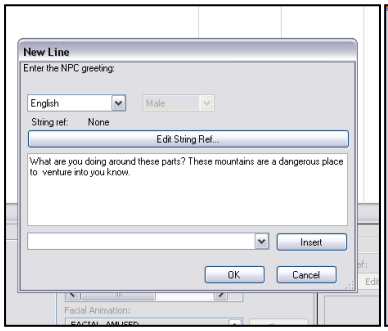
Painting scenery by dragging and dropping



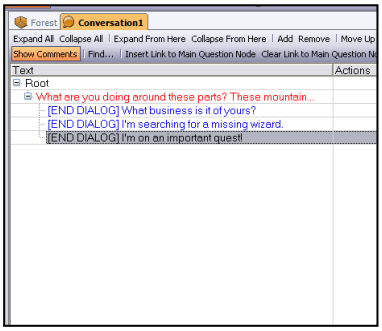
Adding characters by dragging and dropping

Writing conversations

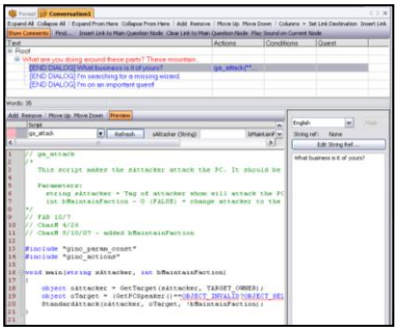
The inbuilt conversation writing tool allows users to create interactive conversations in which the player can choose from multiple options when conversing with other characters. Writing conversations does not require scripting, but if a user wants to place conditions on certain conversation lines becoming available (e.g. only when a quest has been completed) or for an event to happen when a certain line is said, then scripting is required.



Entering conversation lines



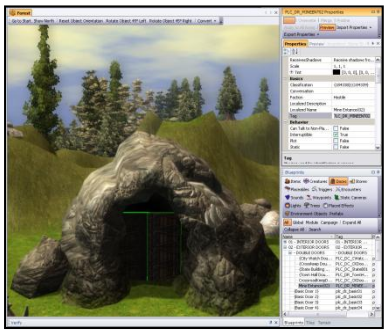
Building up a branching conversation



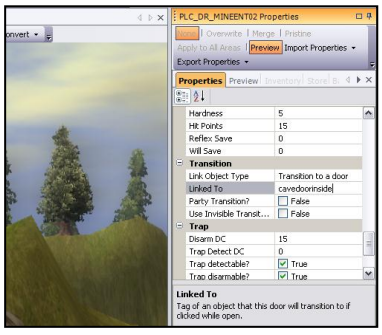
Scripting an action to take place when a line is said

Creating transitions between areas

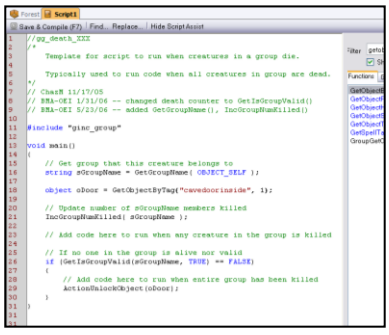
Users can create multiple areas for the player to explore (including indoor areas) and connect these us by adding doors or trigger points which connect the areas together. Adding doors and triggers does not require scripting, nor does the connection process (although this does require users to create and use a unique identifier string). Users can require a specific key to be carried in order for a door to unlock (using a further unique identifier) without scripting, but for any other unlock conditions (e.g. giving the 'right' answer in a conversation or a certain character being dead) scripting is required.



Adding doors by dragging and dropping



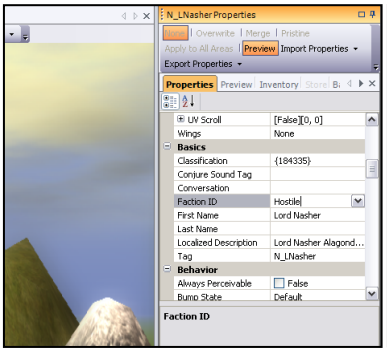
Entering unique identifier to create transition

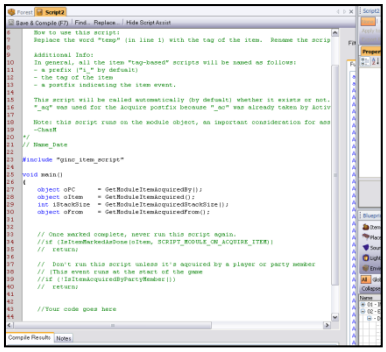


Script to make door unlock when a creature dies

Creating custom behaviours for characters

When characters are added to a game they come with a set of inbuilt behaviours depending on which 'faction' they are. For example, if they are hostile to the player then when they see her they will run towards her and attack using basic AI to engage in a fight, whereas if they are of a friendly faction they will stay still and will converse with the player if a conversation has been written. If a young game designer wants a character to behave in a way which differs from their inbuilt behaviours they must script this behaviour.





Default behaviours are defined by faction

Script to make a character attack when player picks up a certain item

Where Flip comes in

Flip replaces the inbuilt scripting language only, with the existing functionality and interfaces remaining the same for other game creation activities. As the examples above illustrate, users can create a simple game without scripting, but as soon as they want to create more complex or custom behaviour, scripting is required. When using the toolset off-the-shelf this is a problem, because once young people start getting inspiration for their games they have very specific ideas about how the game should pan out, and are naturally disappointed if they have to change their story because they cannot achieve the effect they want to. However, using the Flip plugin this problem becomes an opportunity for young people to explore their ideas and engage with underlying computational concepts such as conditionals.

Design

A Learner Centred Design (LCD) approach was adopted for the Flip language. We consider it important to design educational tools with extensive input from the target users, and followed the CARSS (Context, Activities, Role, Stakeholders, Skills) framework to guide our LCD activities (Good and Robertson, 2006). We have carried out a number of participatory design sessions in order to generate and refine our design. The activities we have undertaken include observing non-programmers working with the natural-language based programming environment Inform 7 (Nelson, 2006), gathering data on the way in which young people naturally describe game-based events which involve computational concepts, the design and user testing of a low-fidelity prototype version of the Flip language, and asking young people to design their own visual representations for different rule elements.

A key requirement for Flip was that young people without programming experience should be able to use it to create the events they wanted to in their games. However, empowering them to achieve their game creation goals would not, by itself, be sufficient. A central aim of the environment was that it should also improve users' understanding of the computational processes and concepts which underlie the events in their games.

References

Fletcher, G. and J. Lu (2009). "Human computing skills: rethinking the K-12 experience." *Communications of the ACM-Association for Computing Machinery-CACM* 52(2): 23-25.

Good, J. and J. Robertson (2006). "CARSS: A framework for learner-centred design with children." *International Journal of Artificial Intelligence in Education* 16(4): 381-413.

Good, J. and J. Robertson (2006). Learning and motivational affordances in narrative-based game authoring. *Narrative and Interactive Learning Environments (NILE 06)*, Edinburgh, UK.

Good, J., K. Howland, and K. Nicholson, " Young People's Descriptions of Computational Rules in Role-Playing Games: an Empirical Study," Submitted IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2010).

Guzdial, M. (2008). "Paving the way for computational thinking." *Communications of the ACM-Association for Computing Machinery-CACM* 51(8): 25-27.

Howland, K., J. Good, et al. (2008). A Game Creation Tool which Supports the Development of Writing Skills: Interface Design Considerations. *Narrative and Interactive Learning Environments (NILE 08)*, Edinburgh, UK.

Howland, K., J. Good, et al. (2009). Language-based support for computational thinking. *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Corvallis, OR, USA.

Kelleher, C., D. Cosgrove, et al. (2002). ALICE2: programming without syntax errors. *User Interface Software and Technology*, Paris, France.

Kelleher, C. and R. Pausch (2007). "Using storytelling to motivate programming." *Communications of the ACM* 50(7): 64.

Maloney, J., Y. Kafai, et al. (2008). *Programming by choice: urban youth learning programming with scratch*. *39th SIGCSE Technical Symposium on Computer Science Education*. Portland, Oregon**:** 367-371.

Nelson, G. (2006). "*Natural Language, Semantics Analysis and Interactive Fiction.*", from http://www.informfiction.org/I7Downloads/Documents/WhitePaper.pdf. Accessed March 12, 2010.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*, New York: Basic Books.

Repenning, A., A. Ioannidou, et al. (2000). "AgentSheets: End-User Programmable Simulations." *Journal of Artificial Societies and Social Simulation* 3(3).

Wing, J. (2006). "Viewpoint-Computational Thinking." *Communications of the ACM-Association for Computing Machinery-CACM* 49(3): 33-35.