H. Reiterer & O. Deussen (Hrsg.): Mensch & Computer 2012
München: Oldenbourg Verlag, 2012, S. 399-408

2DGree: Rapid Prototyping for Games

Joerg Niesenhaus[1], Burak Kahraman[1], Johannes Klatt[2]

Interactive Systems Group, University of Duisburg-Essen[1]
Novacore Studios, Muelheim a.d. Ruhr[2]

Abstract

This paper introduces 2DGree, an experimental game prototyping framework, which enables users to implement and playtest their game ideas within minutes. The framework's main purpose is the evaluation of different interaction techniques as well as methods of programming by demonstration and visual programming for the application within the context of rapid game prototyping. The core of the 2DGree framework consists of a game world editor tool and a script editor, which can be connected to further components like game asset sharing platforms or evaluation tools. The paper describes the current stage of the framework development, presents a user test and provides an outlook on the future plans for the framework development and application.

1 Introduction

User participation in the context of digital games exists as long as the games itself but the possible degree of participation changed a lot over the years (Edery & Mollick 2009, Niesenhaus 2009). Back in 1962 students added new content and features to the game Spacewar! and brought it back to the community afterwards, making it one of the first game modifications. For years the modification of a game was only possible for skilled programmers using tools like hex editors to manipulate the binary game files. In the mid-eighties the first graphical level toolkits appeared[1], giving users the opportunity to design and exchange new level designs. The major breakthrough in the history of end-user development in the area of digital games was the success of the modification CounterStrike[2] for the game Half-Life, which changed the attitude of developers and publishers towards user-generated content and game modifications. Nowadays, several developers and publishers offer games with a focus on user-generated content. Two examples of games building upon the success of user-generated

[1] The game Lode Runner offered one of the first level toolkits in 1983 (Amiga 800, Broderbund Software).

[2] The modification CounterStrike was made part of official Half-Life franchise and sold 10 million products under its label.

content in games are Spore (PC, Electronic Arts) and Little Big Planet (PS3, Sony Entertainment). With Spore's creature editor users generated over 171 million different creatures[3], which are distributed by the game's servers to all users. The Little Big Planet games offer an intuitive toolset enabling gamers to create compelling level designs without the knowledge of technical background information[4]. The toolset is directly connected to the game, which gives players the opportunity to test their design whenever they want to. Although a lot of effort goes into the toolsets for creating user-generated content, most games offer user participation only in the areas of graphical or level design content. Modifications, complete makeovers (called "total conversions") or the creation of a new game idea with frameworks like XNA still have higher requirements at the users skills and knowledge, often including higher-level programming skills (Niesenhaus 2009). Consequently, only a few gamers are able to generate a game prototype based on their own ideas and game mechanics. Furthermore, game designers with little time for coding and the need for rapid game prototype tools to test game mechanics and balancing issues expand the target user group for more powerful but still rapid and intuitive prototyping tools.

The tools evaluated within the 2DGree game prototyping framework aim at this group of users by offering interaction techniques to generate game mechanics via programming by demonstration and visual programming. Applying these methods shall provide an intuitive introduction into the framework tools for beginners and save time for professional game designers. Although coding is an option to set up game mechanics within the 2DGree framework, the vast majority of the game world and its rules can be generated without any coding.

2 Related Work

There are several definitions and taxonomies, which offer different dimensions and criteria to classify visual programming languages and environments. Burnett (2000) defines visual programming as programming in which more than one dimension is used to convey semantics. According to Myers (1986) the term visual programming refers to any system that allows the user to specify a program in a two (or more) dimensional fashion. Both authors emphasize the additional information, which is added through the use of visual elements.

Programming by example is defined by Halbert (1984) as a process in which the user builds an algorithm by working through a concrete example. The term is related to programming by demonstration, which describes system being able to infer the program structure based on the user's inputs, recognizing patterns and apply them to an algorithm (Halbert 1984). Shu (1986) distinguishes visual programming languages by the three categories "levels of language", "scope of language" and "extend of visual expression". In contrast to this definition

[3] Sporepedia: http://www.spore.com/sporepedia (Last visit: 2012-03-31)

[4] There are more than 6 milion level designs available for Little Big Planet 1 & 2 (Playstation 3, Media Molecule)

Myers proposes three binary categories, which are labeled “binary vs. batch”, “visual programming (or not)” and “programming by example (or not)” (Myers 1986). Burnett adds common strategies in visual programming languages and how they are applied in different environments (Burnett 2000). Kelleher and Pausch (2005) classify visual programming languages through their goals and distinguish between empowering systems and teaching systems. Although visual programming languages offer a lot of advantages for programming tasks of lower complexity, more complex tasks are solved less efficiently compared to traditional text-based programming languages (Schiffer 1996).

There are several popular visual programming environments in research and on the commercial market. The visual programming environment Alice focuses on digital storytelling and the creation of games with 3D graphics to teach students basics of programming (Pausch 1995). The software-authoring environment AgentSheets allows its users to build domain-oriented design environments including games, applications and simulations (Repenning 2004). Scratch is a visual programming environment using a puzzle metaphor to enforce the formulation of syntactically correct expressions (Maloney et al. 2004). StarLogo TNG consists of the visual programming language StarLogoBlocks and an integrated programming environment (Resnick 1996). It uses a similar metaphor like Scratch by offering code blocks of different shapes and colors which can be combined via drag and drop. Microsoft’s Kodu Games Lab is available both on the Xbox360 console and the PC with the intention to offer kids a tool to design, build and play user-generated games (MacLaurin 2009). Although these tools provide a lot of good ideas how to set up game mechanics without coding, most of them focus on teaching kids and students programming rather than supporting gamer communities and game designers. In addition, all tools have a focus on specific interaction techniques to generate programming logic rather than being made for exploring and comparing different interaction techniques.

Next to the visual programming tools there are several toolkits and frameworks, which are often used by professional game developers for prototyping game ideas or by students and hobbyists to breathe life into their own game ideas. There are several toolkits and frameworks, which support the development of (prototypical) games like e.g. XNA[5], GameMaker[6] or the Unity 3D[7] game engine. The tools have different requirements in terms of skills and knowledge, but most of them allow users to develop basic games without prior programming skills and offer more complex development options for advanced users as well. Although these tools and frameworks show potential on attracting beginners to the area of game development, we are convinced that the application of methods of visual programming and pro-

5 Microsoft XNA is a set of tools with a managed runtime environment that facilitates computer game development. More information: http://create.msdn.com/en-US/ (Last visit: 2012-04-01)

6 GameMaker is a Windows and Mac integrated development environment published by YoYo Games based on the Delphi programming language: http://www.yoyogames.com/gamemaker/ (Last visit: 2012-04-01)

7 Unity is an integrated authoring tool for creating 3D games and supports all major gaming platforms as well as iOS and Android. More information: http://www.unity3d.com (Last visit: 2012-04-01)

gramming by demonstration can be beneficial for the intuitive access and efficient usage of the toolkits and frameworks.

For this reason, we started with the development of our own game prototyping framework with interchangeable components in order to implement a flexible and controlled environment for the evaluation of different interaction techniques as well as to test the application of different methods of visual programming and programming by demonstration. The acronym *2DGree* relates to the term „to develop games – rapidly, efficiently, easily“ and replaces the generic game prototyping framework description (Niesenhaus et al. 2009).

3 The experimental game prototyping framework

As already pointed out, some of the existing game prototyping tools need a lot of effort to learn how to use the tools and often require at least basic programming skills. In order to lower the entrance barrier for the user and to save time for professional developers, 2DGree enables the users to create their game prototypes with no coding at all. 2DGree's goal is to combine the accessibility of game level editing tools with the depth and flexibility of visual programming languages. In comparison to other available game prototyping frameworks, 2DGree uses methods of direct manipulation and programming by example to set up the game world, place software sensors and events and change properties of game entities (Niesenhaus et al. 2009). The basic idea of this approach is to keep designers as long as possible in the world editor before setting up more complex rules through visual programming in the script editor or by optional script coding. The 2DGree framework architecture is build around the world editor, which enables the users to set up the game world. The component structure of the framework allows exchanging single components for evaluation purposes. Within the past development cycles of the framework three different script editor tools were developed and tested within the framework in order to gather knowledge, which interaction techniques, logical structures and operators work best within the context of game prototyping (Niesenhaus et al. 2009).

2DGree is the fourth iteration of an experimental game prototyping framework, which started out as a basic game engine for 2D Flash games. All currently available tools are implemented with Adobe Flex4 and run in all common browsers. World and script editor components communicate via XML datasheets, which describe the game world's tile set grid, all game entities, software sensors and scripts.

Before presenting the results of a user study for the fourth framework iteration, the current state of two 2DGree main components will be explained.

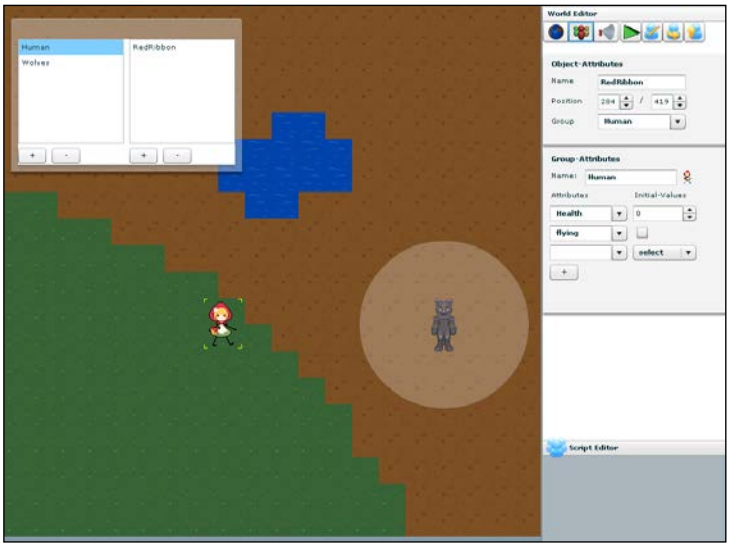

Figure 1: Graphical User Interface of the World Editor component

3.1 World editor

Within the world editor the user creates the visual representation of the game world by painting the tile set and placing the player object as well as all further game entities. In the current version only a two-dimensional representation of the game world is available, which forces the users to focus on the basic game mechanics and concepts rather than exploring the vast possibilities of 3D world design. After creating the game's entities, the user can place different sensors to initiate or track game events. These software sensors are either world or entity-bound and are represented through basic geometries (e.g. circle, square) or can be drawn by the users (with a polygon tool) and are available as generic sensors or special presets (e.g. vision, sense of hearing). The user can place a sensor via drag-and-drop on a game entity or a grid tile in the game world. All sensors can be changed in size and alignment, some of them, like the vision sensor offer additional parameters like an offset value, the angle of sight and the length of the sensor. The parameters can be changed by direct manipulation with the mouse cursor or by changing values in the right menu bar.

The direct manipulation commands are inspired by typical controls of well-known graphics editing programs like Adobe Photoshop or Microsoft Paint. The acoustic sensors can be adjusted to a certain tracking range to get connected to acoustic transmitters of the frequencies within this range. Collisions are detected as soon as an entity moves into the radius of a software sensor. In addition, collisions are subdivided into touch, enter and leave collision events, offering the user possibilities to differentiate the respective phases of a collision event, which are each visualized by their own symbol. When a collision is detected a small symbol (e.g. an eye for the vision sensor) appears at the crossing border of both entities highlighting the collision. A context window opens with a direct input option to select an event,

which shall be triggered in the case of the corresponding collision during the game. Typical collision outcomes are the manipulation of entity properties (e.g. health points of characters), deletion of entities, or the connection to higher level game mechanics created in the script editor.

Table 1 provides some examples of typical goals within the 2DGree framework and how and where the goals can be achieved. Most of these goals can only be achieved within one component, but there are several goals, which can be solved in multiple ways within different components. An example for a multiple-way solution is the already mentioned generation of a collision event. To set up this event, a user can use the world editor component to maneuver a character into an entity via drag-an-drop or – analogue to the gameplay experience - with the control keys to generate a collision event. Another option would be the generation of an IF-THEN-clause within the script editor (see description within chapter 3.2). These three approaches generate the same outcome but offer users a choice of which interaction technique fits their preference and their level of experience best.

Goal	Action	Component
Generate environment	Use brushes to paint the tiles on the grid	World Editor
Place entities	Drag and drop entities from taskbar to grid OR generate through object manager	World Editor
Place software sensors	Drag and drop sensors on existing entities or tiles	World Editor
Set up collision events	Move entity A via drag & drop onto entity B OR use game controls to move entities into a collision OR use script editor collision event	World Editor OR Script Editor
Set up collision outcome	Choose presets via drop-down menu OR use visual script editor	World Editor OR Script Editor
Set up global game mechanics	Use IF-THEN script boxes to compose game mechanics	Script Editor
Organize script hierarchy	Use drag & drop to setup relationships and hierarchies between scripts	Script Editor

Table 1: Examples for goals and actions using the different components

3.2 Script editor

The current iteration of the script editor uses graphical representations of IF-THEN-clauses (Condition & Action) to define game events, which can be attached to existing software sensors already placed in the world editor. In addition, global variables and goals can be defined in the script editor like listeners for global functions (e.g. a player's health bar or a hierarchical quest structure). The script editor GUI subdivides into three distinctive sub windows: the script manager, the main workspace and the script property window (see figure 2, from left to right). The script manager allows the user to organize the collection of previous generated scripts. On the main workspace the user generates new scripts, sets up relation-

ships and hierarchies. Figure 3 shows the generation of a condition. Below the active condition facet the action facet is deactivated and downsized. The properties of the selected script can be changed on the right hand side in the script property window.

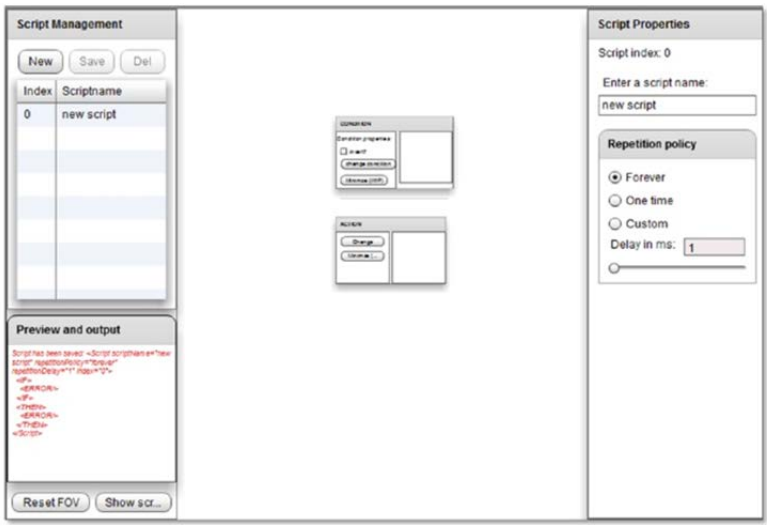

Figure 2: Script editor graphical user interface with workspace

Although the script editor differentiates a lot from the world editor GUI in terms of interaction techniques and visualization of states, pretests showed that most users understand the interrelation between both components.

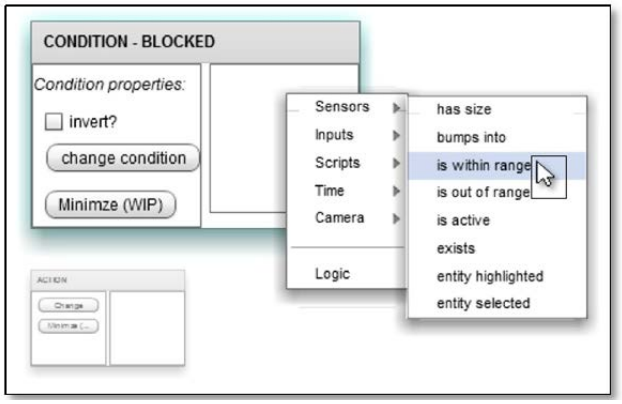

Figure 3: Choosing a sensor condition

The first version of the script editor used building blocks to generate scripts, which worked similar to the logical building blocks Kodu or other visual programming languages offer. Although this process was fairly easy to understand for the users it introduced a lot of constraints regarding the generation of more complex queries and dependencies (Niesenhaus et al. 2009). The second implementation introduced a hierarchical generation of scripts with a context-sensitive selection method for appropriate choices of logical elements. Building upon our experiences and several tests, the current version of the script editor offers a higher flexibility in terms of hierarchical dependencies and a better overview of the scripts with the option to zoom in via fish-eye view.

4 Evaluation

During the past two years of the framework development several user tests and expert reviews were executed in order to evaluate the different components of the framework from different perspectives (usability, user experience, performance) (Niesenhaus et al. 2009). The latest study we present in this paper focuses on the world editor with the sensor setup and modification. Twenty subjects (12 male, 8 female) participated in this study. All subjects are computer science students and have basic knowledge of digital games. After a short tutorial the subjects were asked to complete a scenario with four typical tasks within the world editor. The tasks included the generation of a landscape with different textures and assets (1), the creation of two characters with different visual representations and properties (2), setting up sound sensors and emitters to enable the 'communication' between both characters (3) and creating non-personal characters with movement and collision behavior (4). After the subjects finished the four tasks they filled out two questionnaires. The first questionnaire offered the subjects the possibility to rate the quality of the tools and their functions and asked them for their experience within the area of digital games and their knowledge of authoring tools and general computer software. The second part was based on the German ISONORM usability questionnaire, which is closely related to the ISO 9241-110 usability standard and was used to judge the quality of the tools. Afterwards the subjects were animated to comment on the 2DGree tools and to provide additional feedback.

	Current study			Previous study			Score
	N	Mean	SD	N	Mean	SD	Diff.
Suitability for the task	20	4,40	1,08	21	4,10	1,04	0,30
Self-descriptiveness	20	3,26	1,08	21	2,86	0,96	0,40
Controllability	20	4,37	1,26	21	3,05	1,24	1,32
Conformity with user expectations	20	4,10	1,02	21	3,86	1,20	0,24
Error tolerance	20	3,71	0,92	21	3,29	1,06	0,42
Suitability for individualization	20	3,47	1,09	21	3,52	1,17	-0,05
Suitability for learning	20	4,20	1,38	21	2,76	0,94	1,44

Table 2: Average item scores of the ISONORM questionnaire within both studies

Although most of the subjects were able to successfully complete the tasks, the time for the completion of the tasks varied strongly. Subjects with previous knowledge of visual programming tools were significantly faster ($p \leq 0.001$) than subjects without previous knowledge. The analysis of the ISONORM usability questionnaire indicated average values (between 3.26 and 4.40 on a scale between 1 and 7, with 7 being the best score). The self-descriptiveness (M=3.26), suitability for individualization (M=3.47) and error tolerance (M=3.71) achieved the lowest scores. The feedback of the subjects indicated that most of the visual metaphors work very well for both beginners and experts, but also hinted on differences in the understanding of basic programming paradigms like parent-child relationships or recursion. Further comments addressed the lack of an undo function, which was not available at the time of the user test.

We compared the results of the latest study with our previous findings, which revealed slightly higher average scores for the items error tolerance (Previous version: M=3.29; Current version M=3.71), self-descriptiveness (M=2.86; M=3.26), suitability for the task (M=4.10; M=4,40) and suitability for learning (M=2.76; M=4.20) in the current version compared to the previous framework iteration and significant higher scores for the items controllability (M=3,05; M=4,37) and suitability for learning (M=2,76; M=4,20). These results reflect the improvements within the graphical user interface and the general interaction process optimization of the current framework prototype, however there is still potential for further improvements.

5 Future work

In response to the user test feedback, we are currently adding screencasts, tutorials and mouse-over tool tips to all functions in order to give users a better impression of what each GUI element represents. After the world and the script editor reach the beta status both components will be connected to a community platform, which is currently under development. The community platform will feature a shop system where users can select presets of game entities and scripts for setting up their game. Next to predefined sets of characters, environmental assets or scripts the users will be able to upload their creations to the community shop system to give all other users access to their self-generated content. The main purpose of the community platform is the evaluation of the different components through online user tests, which will be extended through questionnaires and user feedback boards to get as much feedback from users as possible. In addition, we are preparing further lab studies using a larger variety of components and a large-scale online test in order to evaluate the implemented methods of direct manipulation and programming by demonstration with a larger number of subjects including both professional game designers as well as community users. Further research will also compare existing prototyping tools with the 2DGree tools.

References

Burnett, M. (2000). Visual Programming In: *Encyclopedia of Electrical and Electronics Engineering*. Chichester, NY: Wiley.

Edery, D. & Mollick, E. (2009). *Changing the game. How video games are transforming the future of business.* Upper Saddle River, New Jersey: Pearson Education.

Halbert, D. C. (1984). *Programming by Example*. University of California at Berkeley: Doctoral Thesis.

Kelleher, C. & Pausch, R. (1995). Lowering the Barriers of Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Computing Surveys*. 37 (2), 83-137.

MacLaurin, M. (2009). Kodu: End-User Programming and Design for Games. In: *Proceedings of the 4th International Conference on Foundations of Digital Games (FDG'09)*. New York, NY: ACM.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B. & Resnick, M. (2004). Scratch: A Sneak Preview. In: *Second International Conference on Creating, Connecting, and Collaborating through Computing*. Kyoto, Japan.

Myers, B. A. (1986). Visual programming, programming by example, and program visualization: a taxanomy. In: *Proceedings of the International Conference on Human Factors in Computing Systems (CHI'86)*. New York, NY: ACM.

Niesenhaus, J. (2009). Challenges and Potentials of User Involvement in the Process of Creating Games. *International Reports on Socio-Informatics: Open Design Spaces Supporting User Innovation.* 6 (2), 56-68.

Niesenhaus, J., Löschner, J & Kahraman, B. (2009). Förderung der Nutzerinnovation im Rahmen digitaler Spiele durch intuitive Werkzeuge am Beispiel des Game Prototyping Frameworks. In: *Grenzenlos frei!? Workshop Proceedings der Tagung Mensch & Computer 2009*. Berlin: Logos Verlag.

Pausch, R. (1995). Alice: Rapid Prototyping for Virtual Reality. *IEEE Computer Graphics and Applications*. 15 (3), 8-11.

Repenning, A. (2004). Agent-based end-user development. In: *Special Issue: End-User Development*. New York, NY: ACM.

Resnick, M. (1996). StarLogo: an environment for decentralized modeling and decentralized thinking. In: *Proceedings of the International Conference on Human Factors in Computing Systems (CHI'96)*. New York, NY: ACM.

Schiffer, S. (1996). Visuelle Programmierung – Potenzial und Grenzen. In: Meyer, H.C. (Editor): *Beherrschung von Informationssystemen*. Munich: Oldenbourg.

Shu, N. C. (1986). Visual Programming Languages: A Perspective and a Dimensional Analysis. In: Chang, S.-K. (Editor): *Visual Languages*. New York, NY: Plenum Press.

Contact Information

Joerg Niesenhaus, joerg.niesenhaus@uni-due.de, Forsthausweg 2, 47057 Duisburg, +49 (0)203- 379 1420