Pitsupai - Collaborative Scripting in a Distributed, Persistent 3D World

Philipp Engelhard Robert Hirschfeld Jens Lincke

Hasso-Plattner-Institut, University of Potsdam
philippengelhard@gmail.com
{*jens.lincke, hirschfeld*}*@hpi.uni-potsdam.de*

Abstract

In this paper we present an authoring tool and an accompanying workflow to create interactive scenarios in a distributed 3D environment by multiple users. With Pitsupai we designed an environment that allows to create collaboratively simple games or game prototypes with a minimal effort, without the need to be a professional game developer. To facilitate collaborative work, our authoring tool uses a scripting language for easy programming and animation in the virtual world and provides awareness aspects - information regarding the whereabouts and current activities of the other participants. Scripts can be edited from within the running virtual world without the need for stopping or restarting it.

1 Introduction

The immense gain in computing power over the last decade led to the development of online, 3D worlds that many people can visit and explore. Virtual worlds offer user experiences that are not found in real life and they also offer the chance for people from all over the world to meet. But the creation of an appealing 3D environment is usually reserved for professionals. This is because of the complexity of the task as well as the huge amount of work that needs to be accomplished. There are some approaches to give none-professionals access to such a task by developing simpler user interfaces and restricting the power of what can be done. This still leaves users with the problem of the huge amount of work at hand. A collaborative workflow would allow this amount of work be shared among multiple users, and collaborative teamwork can generally be more fun and motivating.

Creating or enhancing a virtual 3D world can be an entertaining experience. People like to express themselves, create games or tell interactive stories. Looking at Second Life [11] it becomes obvious that many people build virtual homes just because they enjoy doing it and not because they have a direct benefit. Some people have ideas for games but no tools to create such games, and yet others love to tell stories, but they need a “virtual dollhouse” in which to tell their story. People like sharing their creations with others and even sharing the experience to create itself.

Besides having fun in a 3D world, it can also be an excellent place for learning. Collaborative environments offer many opportunities for distant learning. Groups of students can meet online to do group work, and a tutor might join in to help [12]. An Example for an successful 3D learning environment to teach programming is Alice [18], but Alice is not an collaborative environment. In a 3D world problems can be communicated that would be hard to explain on a sheet of paper. To have a bigger impact on the learner, such a scenario can be made interactive. An interactive scenario could also be used to pitch ideas to other people or companies and might leave a stronger impression through higher immersion.

Building a virtual world incorporates not only creating and arranging 3D objects, but also bringing those objects to life. With scripting languages more people can program, who could not do so with traditional programming languages. This makes scripting languages an interesting choice for animation in 3D worlds. There are many ways to animate objects and scripting - as one of them - allows great flexibility and requires only little programming experience. Scripts tend to have shorter code length compared to traditional languages, which is an advantage not only for collaboration but for novice programmers as well. At the example of *Pair Programming* the benefit of collaborative software development can be seen. Much more errors are spotted during coding, the resulting design is better, code length is shorter, projects end with multiple people understanding each piece of the system and people learn to work together [2, 3, 4].

Many existing systems neglect or at least do not actively support the opportunities of working in a team. One of the biggest benefits for working in a team is that there are other

978-0-7695-3620-0/09 $25.00 © 2009 IEEE
DOI 10.1109/C5.2009.11

people that can help when someone is stuck with a problem. As long as the team works at the same place it is no problem to come over and help, but with a locally dispersed team this is no option. That is why we attached great importance to the subject of collaboration and helping each other.

In [1] Bartle describes a persistent world as a virtual world that -like the real world- exists continuously. “World” in this context does not necessarily mean an entire planet, it is an environment that its inhabitants regard as being self-contained. The worlds are implemented by a computer (or network of computers) that simulates an environment. Persistent worlds are always available and world events continue to happen even though no user is logged into it. When a user changes the world, the changes persist and it is impossible to return to a prior state. Persistent is here the opposite to temporary. In a persistent world it is crucial to ensure that changes to the program can be made without having to restart the world. This becomes obvious by the example of Second Life where up to 60,000 people are online, at the same time in the same world. Even if only a fraction of those people were working and changing programs, it would result in such an amount of restarts that inhabiting in this world would be pointless.

Working together either on the same objects or code raises problems where the users need support from their software. People want to know where other participants are and where their attention is. When people are working together, relevant questions are always: *Who am I collaborating with? Where are they? What are they looking at? What are they up to?* These questions can get hard to answer if the participants are locally dispersed. Raising the arm to point at a problem as you could do it if the other person would stand right next to you is no option in this situation. Besides text or voice communication - which is crucial - awareness aspects help in computer supported collaborative work to answer those questions [9, 13].

In this paper we present an environment where people with little programming background can collaboratively shape a virtual persistent world via scripting. To facilitate this process, a workflow was designed and tools were created in its support. We target entertainment computing as an example field of application. For out project Pitsupai we integrated a code editor in to Qwaq Forums and use Python as scripting language for the virtual world.

The remainder of the paper is organized as follows: Section 2 gives a short overview of Pitsupai, the concept and the designed workflow. Section 3 shows the implementation of Pitsupai. Section 4 discusses related work. The last Section draws a summary and gives an outlook on future work.

2 Pitsupai

The aim of this work is to create an end-user scripting environment in Qwaq Forums. In a collaborative framework, multiple users should be able to create interactive scenarios for a distributed 3D environment. For convenient collaboration, awareness and presence aspects need to be introduced to the authoring process.

Pitsupai[1] is based on Qwaq Forums, that is a collaboration platform for distributed work in a 3D space. It provides a distributed and replicated 3D world in which people can collaborate [15, 14].

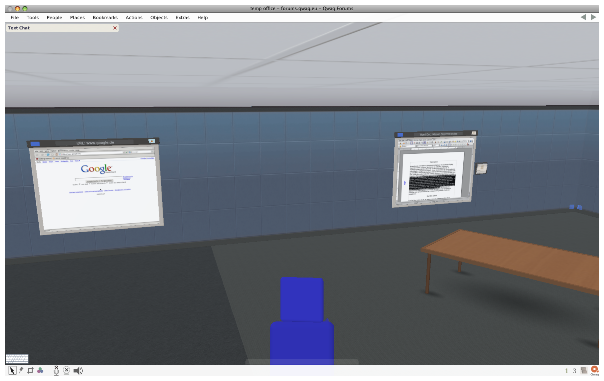


Figure 1. A Qwaq Forums office with two shared applications at the room’s wall.

There are a couple of possible fields of application for Pitsupai as demonstration, personalization, simple games, and education. This leads to the conclusion that the authoring environment of this work should be simple to use, but yet at least powerful enough to do the task at hand, otherwise not enough ideas can be implemented using it. To not being frustrating for beginners, the learning curve must be flat enough. This is especially interesting for education, since many students will learn using such an authoring tool. Also of interest for teaching is the collaboration aspect of the environment, that teams of students can work together and a tutor can join them for help. When creating a prototype an important factor is, that the creation of it can be done in a short period of time, so the idea can be tested in a short feedback loop.

2.1 Pointing: A Major Collaboration Aspect

It is a convenient way for us to raise the arm to the screen and point with the finger at positions that seem relevant to

[1]The project name ”Pitsupai” was chosen, because it means “Hello”. A salutation seemed suitable for a collaborative environment. This word is taken from the language of the Mehinaku Indian culture from the South American Amazon basin.

the problem. Especially for helping and demonstrating, it is important to have the possibility that other people can see where someone is pointing at. Not only from our own experience, but also in conversations with other people, we saw the ability to be able to point at all working location as critical to communicate a problem. This is a heavily used action when two or more people are standing next to each other looking at the same screen.

This is not possible when people are not at the same location and are just communicating via voice or text chat. As an result it is often a long and complicated task to first ensure that everybody is looking at the same content, and second to describe exactly what location on the screen a persons wants to point out. Using only verbal communication for this task can be very tiring and often leads to miscommunication.

Realizing this leads to the conclusion, that the implemented system needs cursors that are able to point at all locations in the 3D world or the 2D user interface that might be relevant to help each other while working together. Since pointing at a computer is commonly done with a 2D pointing device like a mouse, it is the mouse location that needs to be propagated through the system to all other users. This may sound basic, but still is an extremly important functionality.

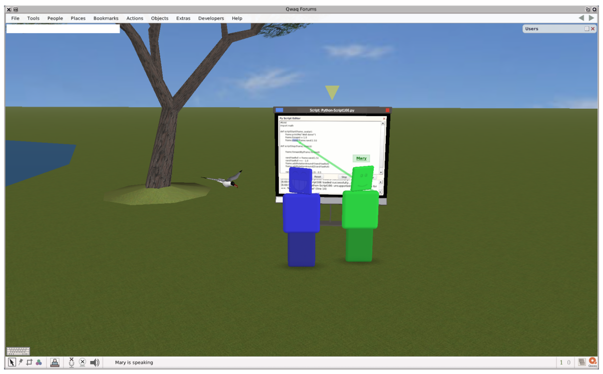


Figure 2. Two users collaborating with Pitsupai

2.2 Collaborative Editor / Error output

When designing a scripting workflow for a distributed environment, it would have been a missed opportunity not to enable collaboration in this workflow. Besides being able to work together at the same project, a collaborative environment is a good place for learning and helping each other. Like in Second Life collaboration can be a social event.

Therefore in Pitsupai the authoring tool has a collaborative editor and the source code of the script is a replicated document. The editor has a shared output, so that errors or printed output from the script can be examined together with other participants. All information is visible to everyone, this includes a shared view on scripts, but also on the scripted objects. Furthermore, awareness aspects as well as voice and text chat facilitate collaboration. As explained before it is important to know where participants are and they need a possibility to point out matters. Some information can be said more easily to others and some - like source code - can easier be written, what makes both text and voice chat important.

2.3 Why Using a Scripting Language

In Pitsupai we use Python as scripting language, because using a scripting language has a couple of advantages. The target audience of the system are people with at least little programming experience, who are able to use textual scripting. Using a textual scripting language empowers a range of people to use such a system, even though it surely rules out people who have not a minimal affinity for programming or people who cannot read and write - like preschool children.

Croquet is written in *Squeak* a *Smalltalk* dialect. To add functionality to Croquet or Qwaq Forums Smalltalk code can be used. It has most of the features a modern scripting language has and still there are benefits of using an external scripting language. Python has a large user base and good documentation. Choosing Python gives more people the possibility to do programming in Qwaq Forums than Smalltalk does, without the need for learning a new language. Python offers a solid documentation for beginners and advanced programmers. For a non-programmer scripting languages are often much easier to learn. High level features and conceptual simplicity of a scripting language will make writing code significantly less burdensome to a game designer. There is no need for explicit memory management, and common data types and functions such as lists, queues, stacks, sorting and searching are usually a few lines of code [10, 7].

The second major advantage of using an external scripting language is safety. A scripting language can be generally speaking integrated in two ways: As a symbiosis with the host system or in a *sandbox*. The first way gives the scripting language the same power as the host system in terms of access to resources. The second way lets the scripting language only work within the boundaries given to it by the host system. This restricts which resources can be accessed by the scripting language and can prevent it from using critical areas like the users' personal data or system resources like network connections. Another safety advantage of the sandbox integration is the possibility to make it run independently from the host system, which means, if the user writes code that locks the scripting environment, it need not lock the host system and the operation system as well.

Multi-language development adds an extra layer of complexity. It is difficult to debug in both languages simultaneously, and time must be spent on maintaining the glue code that ties the languages together.

Dynamic languages like Python have no "compile time" type checking. This is something programmers which are accustomed to a typed language as Java or C/C++ need to get used to first, but it merely results in a different set of run-time problems to worry about, and they are generally much easier to deal with.

In some scripting languages performance gets a problem, if the written code heavily relies on floating point math. Comparing such a task to performance of well written C++ code is often disappointing. Every variable reference is a hash table lookup, and so is every function call. This will not give performance that can compete against C++, where the locations of variables and functions are decided at compile time. But this is not what a scripting language should be used for. A good example what a fitting use for a game is, is shown in [5]. For the game SimCity 4 e.g., the behavior of the people or vehicles moving around in a city is done in a scripting language, while the core functionality of simulating a city is not [10, 6].

2.4 Programming in Persistent Worlds

Scripting in a persistent world has a couple of entailments. Since a persistent world should have as little downtime as possible, it is not much of an option to restart it every time something does not work as expected. It is a common approach to test a program and if it goes out of hand to simply restart everything. Scripting in a persistent world should not deny a programmer such an approach of *trial and error* programming and therefore offer a restart option that affects only the written program and not the whole world. This includes the script itself as well as the scripted objects. Also has a multi-user virtual world, that allows to be programmed from within itself, many uses: From exploratory collaborative development, to simple extensions in real-time response to user requests [16].

Besides a convenient way of programming in a persistent world it is also important that a program does not affect the underlaying system in a way that it compromises its functionality especially for other users. That is why it is a good choice to run the scripting extension in a sandboxed environment thus restricted in its power. Persistent world scripting entails also specials in debugging. Since the persistent world should not be restarted for changes in a script, reediting and debugging needs to be done inside the running world. And when a script is no longer wanted, it must be removable without requiring to restart the whole system.

2.5 Integrated Editor

One of Pitsupai's goals is to have an obvious and direct relation between the source code and its outcome. Thereby making it easier for its users to see the relation between their input and the resulting output in the virtual world. To support this goal, the source code editor is integrated into the development environment as part of the virtual world. Also an integrated editor can highlight the code line where an error occurs. For the approach of *trail and error* programming these factors are important. Such an immediate feedback is less likely to achieve, if the users must switch between two or more applications. Furthermore it is desirable to have an editor adapted to the workflow. Pitsupai's workflow adapts Qwaq Forums' workflow where saving and replicating files is handled internally and the user only deals with File Cards. These File Cards are also used to assign a script to an object by dropping a File Card onto the target object. A drag and drop workflow should work with external files that are dropped in to the virtual world from the user's operation system as well, but not in such a convenient way.

3 Implementation

Croquet is an novel approach to developing collaborative interactive media applications. Every part is designed around enabling real-time collaboration. TeaTime and Islands are the basis for Croquet's replicated computation and synchronization. Islands are containers for arbitrary objects. Croquet guarantees that the progression of state of a particular Island replica is identical to any other Island replica of the same Island. An Island encapsulates its content and enforces a rigorous content-hiding and a message passing model. This is necessary to guarantee identical state of the Island and responses to external events.

3.1 Language Bindings

The version of Qwaq Forums we used included Python bindings that do the basic communication between Squeak and Python. This makes it possible to call external Python functions from Squeak. It does not translate Python code to Squeak code and executes it, instead it runs a Python interpreter and can use external modules. The Scripts are executed on every participant's machine. This means, they are replicated as the rest in Croquet: No data or output gets distributed, but every machine replicates the computation.

With Pitsupai we concentrate on scripting of objects of the 3D world. A Python script is not hard-wired to an object, but can be executed for multiple objects. Every scripted object in the 3D space[2] gets a *proxy* object that

[2] All graphical objects in Croquet are based on the frame class

```
def pointerOver(frame, event):
        frame.addRotationAroundY(5)
```

Figure 3. A Python call on an event.

knows which scripts must be executed for its object. I.e., if one script *s1* should be run for object *o1* and *o2*, there is just one script *s1* that gets executed twice, once for *o1* and once for *o2*. Therefore it is important to hand in to every function a reference to the scripted object. In figure 3 an example script is shown, that rotates an object around its y-axis by 5 degrees, when the mouse pointer goes over it. This is done by calling the `addRotationAroundY` function of the scripted object *frame* and *frame* is passed in the function call.

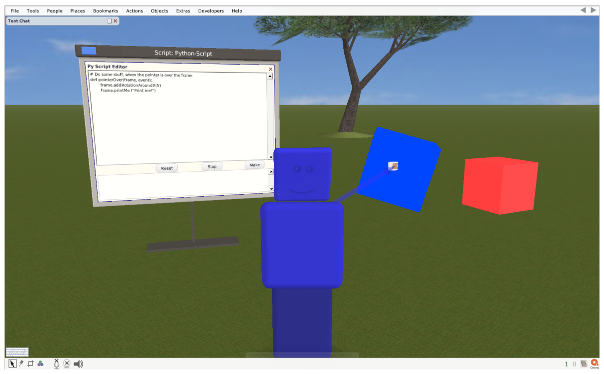
This example also demonstrates two other features: Scripts can be executed on events and Forums offers a function library of a couple of basic, predefined functions for frame objects. In the example the function acts on the event that a pointer is over the object. The `pointerDown` event (when an object is clicked) is another example for such events. When `pointerOver` is called the `addRotationAroundY` gets executed. Other useful examples are `translateBy` or `colorize`, where the first one translates the object by a given vector and the second sets the object's color.

3.2 Replicated Code Editor

A key aspect of Pitsupai is collaboration. It is a major aspect in which this work differs from Second Life scripting. Creating and modifying a script is supposed to be done in a small group of people. Therefore all actions done in the script editor need to be distributed. A Python script can be edited in an editor that is placed on a stand inside the 3D world. Every user in a forum can see and work with such an editor.

We aim to enable small groups of 2-5 people in a forum rather than a large group of 10 or more people to work together. A larger group can still work together with the implemented system but should break up in groups that work in different forums. Their parts can be combined in one forum afterwards, but the creation can be done more effectively in smaller groups.

Heading for this target, user avatars gather around a stand inside the 3D world to edit a script. As for other applications in Qwaq Forums they have a WYSIWIS (what you see is what I see) view on an editor. Multiple users can interact with the editor, but have only one mouse and keyboard focus. This is a drawback, but it weighs not so much, since groups should be small and scripts are supposed to be short, at maximum rather a few ten lines than a few hundred. The



```
# Do some stuff, when the pointer is over the frame
def pointerOver(frame, event):
        frame.addRotationAroundX(5)
        frame.printMe ("Print me!")
```

Figure 4. A script is dragged onto a cube from the code editor on the left.

local position of an avatar indicates at what a user is working and a *Laser* shows where a user is pointing at with his mouse, as seen in figure 4. If one user selects a passage of code, other users can see this selection.

At the bottom of the editor is an output for status information, like successful compilation or error output, as well as output from print statements. This output is replicated and shown to all users in a forum. When working together, especially helping each other, it is important to have the same information available. To reduce confusion among users, statements have a time stamp and the name of the corresponding script printed with the message.

When a script is attached to multiple objects and this script is changed by an user, the changed version is executed for all objects it is attached to. Another functionality of the editor is to bring all objects back to the point where and when this script got attached to them. This way objects that run a "out of control" script, can be brought back and do not get lost.

Technically the Python editor inherits from the *embedded application* class. Subclasses of the embedded application class are represented inside the 3D world and share a functionality base for collaboration. This includes the shared and tightly coupled view on the document and can show visualizations of other participants' pointers. Those embedded applications also bring the functionality to be placed inside the 3D world, they are not intended to be part of the not replicated user interface. Embedded applications work in a similar way as VNC for one application. They all have a single input and if one users make an input it is propagated to all other machines.

3.3 2D and 3D Pointing

Qwaq Forums offers a cursor visualization for the cursor of all other users in a forum. This visualization is shown when a user points at an application on a stand inside the

3D world. The cursor has the same color as the avatar of its owner. Figure 5 shows two avatars and their cursors.

As stated earlier in section 2.1, pointing is a integral method for collaboration. Thus it was important to have a representation of cursors in every setup: When a person points at a 3D stand, having a cursor over the 3D stand as well as in all open 2D windows and when a person points at a 2D window having a representation in all 2D windows as well as at the 3D stand. Our implementation of Pitsupai adds this to Qwaq Forums, for the 2D window avatar-colored cursors are used that look like a standard mouse cursor from a modern graphical operating system. In figure 5 both pointers are shown.

3.4 Scripts as File Cards: Drag and Drop

In Qwaq Forums a widely used interaction method is drag and drop. It is used in many cases. A few examples are dropping an office document from the user's desktop onto a wall in Forums. This opens a display at the wall containing the corresponding office application with this document opened. It works the same way for images. Another example is the user's inventory. The inventory opens inside the 3D space next to the avatar. If a user wants to get a copy of a document, he grabs a File Card from the top right corner of a display containing the document and drops it inside his inventory. Now he can move to another place or forum and drag the document out of his inventory and drop it into the world again.

It was desired to achieve a scripting workflow using the same principles. Therefore the first step was to create a file card type for scripts. This allows - combined with further extensions to the given system - to store scripts in the user's inventory, have a menu entry for creating a new script by creating a File Card with an empty script and opening an script editor when a script File Card is dropped into the world.

Also following the drag and drop principal, attaching scripts to an object was implemented. When the user drags a script File Card onto an object inside the world, the script gets attached to that object and is executed for that object. To get a script running an user can simply take the File Card from an editor displaying the scripts source code and drop it onto a nearby object or take the File Card to his inventory and pull it out there later at a different place to attach it to an object there. Using drag and drop for the scripting workflow integrates very well with the existing system and gives a consistent usage.

4 Related Work

Hintze [8] and Takada [17] designed environments that use tile scripting for animation of 3D objects, because their focus lies on children and pupils, respectively. Tile scripting results in a smoother learning curve but also in a less powerful system as it would be with textual scripting. Furthermore Hintze's system supports no collaboration and Takada's system has only a shared view on the scripted objects. This makes both approaches unsuitable for this work, since here collaboration in the creation process is of bigger significance.

The Kansas project [16] has an interesting approach to the problem of forming groups for collaboration, here people can work in privacy by moving away from the other users, just as in the real world. Kansas has a lot in common with Pitsupai, as it also allows to change its world from within. But unlike Pitsupai it supports no 3D graphics and programs are written in the Self language. 3D graphics are important for many games and the Self language suffers from the same problem as Squeak/Smalltalk of having a small user base.

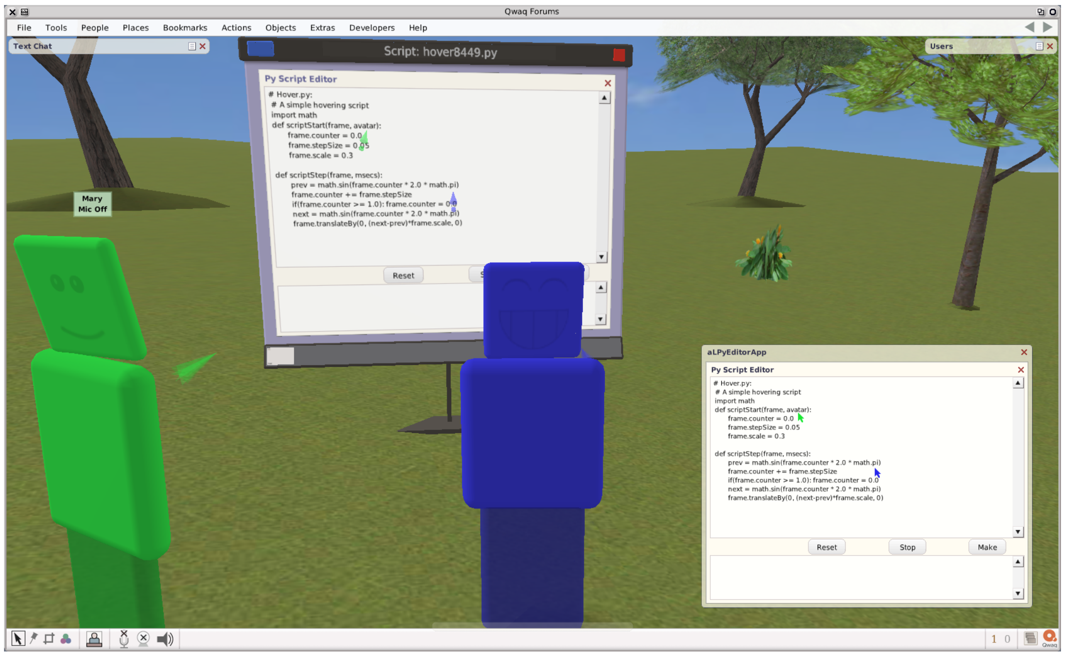
SubEthaEdit [19] is a very mature collaborative source code editor that offers multiple foci for editing the same document. This is a strength the editor in Pitsupai is missing but instead it has a tightly coupled view with only one focus. SubEthaEdit also can be switched to a tightly coupled view like Pitsupai what makes it easier to work together at the same passage of a document, but SubEthaEdit still has multiple input foci, what is a big advantage over the Pitsupai editor. However SubEthaEdit can not be integrated well into the Pitsupai workflow, because it is an external tool.

Second Life [11] has a much more powerful framework for manipulating its scripted objects. Pitsupai offers only a fraction of those functions, but the basic manipulations like translation and rotation are possible. Also Second Life offers receiving information from the world, e.g., that information can be used to make a bouncing ball, that actually bounces of the ground and not just back from an arbitrary distance. Second Life shows its deficiencies against Pitsupai when it comes to collaboration. In Second Life it is technically not possible to work at the same script or object with other people at the same time and also the Second Life scripting workflow does not facilitate collaboration.

5 Summary & Future Work

We designed an easy to use authoring tool as well as a workflow that facilitates collaboration. The designed workflow consists roughly of these steps:

- Create a new script via the document menu
- Drop the created File Card nearby into the world to open a source code editor
- Write the script together with other participants. Everybody can contribute by writing code or pointing at problems



```
# Hover.py:
 # A simple hovering script
 import math
 def scriptStart(frame, avatar):
       frame.counter = 0.0
       frame.stepSize = 0.05
       frame.scale = 0.3

 def scriptStep(frame, msecs):
       prev = math.sin(frame.counter * 2.0 * math.pi)
       frame.counter += frame.stepSize
       if(frame.counter >= 1.0): frame.counter = 0.0
       next = math.sin(frame.counter * 2.0 * math.pi)
       frame.translateBy(0, (next-prev)*frame.scale, 0)
```

```
# Hover.py:
 # A simple hovering script
 import math
 def scriptStart(frame, avatar):
       frame.counter = 0.0
       frame.stepSize = 0.05
       frame.scale = 0.3

 def scriptStep(frame, msecs):
       prev = math.sin(frame.counter * 2.0 * math.pi)
       frame.counter += frame.stepSize
       if(frame.counter >= 1.0): frame.counter = 0.0
       next = math.sin(frame.counter * 2.0 * math.pi)
       frame.translateBy(0, (next-prev)*frame.scale, 0)
```

Figure 5. The pointers of both participants are visible inside the world as well as in the 2D interface.

- Attach the script to an object by dragging the File Card from the editor onto the object. All participants can see what happens and can correct potential errors

To design and validate this workflow, a software prototype was implemented. Therefore an existing system (Qwaq Forums) was enhanced. The principal of File Cards was used and a type for Python scripts introduced. File Cards are a graphical representation of files and manage the replication of the file for all users. This concept is also used to assign scripts to objects by dragging a File Card onto an object.

To edit File Cards with Python scripts as content an editor was designed, which exists inside the virtual world of a forum. An editor inside a forum gives the opportunity not only to let all participants see the editor and the script, but also to let everybody edit the script. Thus, in Pitsupai editing does not work exclusively by locking the document for other users while one user is editing it. Everybody can edit at every time, but participants need to arrange their editing of one script, because the editor has only one keyboard focus. Multiple editors for different scripts can be placed inside a forum at the same time. The implemented editor is based on the embedded application class, which offers the collaboration functionality. This class was extended so that the users' pointers (mouse cursors) are replicated and thereby visible to all participants. This is important for collaboration, so the users can point out matters more easily. Furthermore, the implemented editor not only has a shared view on the script source, but also on error, compiler or print statements.

The current implementation of Pitsupai leaves some issues untouched. For higher convenience there are some aspects that could be added to Pitsupai. E.g., currently one script can be attached to multiple objects and one object can have multiple objects attached. Further in the script menu of an object the user can see all scripts which are attached to this object, but he currently can not see all objects that a script is attached to. It would also be helpful to have more information about what other people are currently doing. Second Life shows a note over the head of an avatar for example "Philipp is editing appearance", when a person is editing the appearance of the avatar. It would be helpful to see information like that on which script or object someone is working.

One point of importance for Pitsupai was the aspect of helping each other. This was only considered when people are online at the same time (synchronous collaboration). Two further ideas for asynchronous collaboration with scripts are that the creation process of a script can be reviewed later like a movie. But unlike the passive roll of a user watching a movie, it could be stopped at any time and the viewer could try the script out as it currently is. The second idea concerns the lack of comments in source code. For example, when a user asks another participant to explain a specific line or section of code, the conversation (voice or text chat) could be saved with a reference to that section of code. So explanations on code sections would not get lost and others could benefit from previous explanations without the extra effort of explicitly writing code comments.

An issue that was not touched at all in current implementation of Pitsupai is safety. In Second Life it is a problem that other Residents infiltrate meetings or annoy their fellow Residents with scripted objects placed in a way that the people cannot carry on with their task or are at least distracted from it. However, different from Second Life in Qwaq Forums usually only people meet who know each other well. Still it should made sure that user-written scripts do not compromise other participants' personal data on their computers. Safety measures were neither implemented nor was it tested what the existing system does to secure the participants' machines.

In this paper we presented an easy to use authoring tool as well as a workflow that facilitates collaboration. The target audience is non-professional users and the field of application is entertainment computing. To achieve a yet powerful but easy to learn system, the authoring tool uses a scripting language (Python) for animating and programming. This can be done right inside the virtual world for a collaborative workflow and an immediate feedback.

6 Acknowledgements

First and foremost we would like to recognize Andreas Raab for his insightful discussions, valuable contributions, and extensive support. We would like to thank Maic Masuch and Christine Strothotte for their fruitful ideas, and Evelyn Eastmond for her comments on an early draft of this paper.

References

[1] R. Bartle. *Designing Virtual Worlds*. New Riders Games, July 2003.

[2] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, October 1999.

[3] A. Cockburn and L. Williams, editors. *The Costs and Benefits of Pair Programming*, Cagliari, Sardinia, Italy, June 2000. XP 2000.

[4] C. Cook. *Towards Computer-Supported Collaborative Software Engineering*. PhD thesis, University of Canterbury, Christchurch, New Zealand., 2007.

[5] C. Crowell. I speak for the little people! And...their cars! Scripting Automata, 2003. as of September 20, 2008, `http://simcity.ea.com/about/inside_scoop/scripting1.php`.

[6] B. Dawson. GDC 2002: Game Scripting in Python, 08 2002. as of September 20, 2008, `http://www.gamasutra.com/features/20020821/dawson_01.htm`.

[7] T. Gutschmidt. *Game Programming With Python, Lua, and Ruby*. The Course PTR Game Development Series. Premier Press, 2003.

[8] J. Hintze. 3D-Animations-Skripting für nicht-professionelle Benutzer. Diplomathesis, 2003. Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik.

[9] T. Holmer, J. Haake, and N. Streitz. Kollaborationsorientierte synchrone Werkzeuge. In G. Schwabe, N. Streitz, and R. Unland, editors, *CSCW-Kompendium : Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten*, pages 180–193. Springer, August 2001.

[10] B. Hook. The Secret Life of Game Scripting, 2002. as of September 20, 2008, `http://web.archive.org/web/20051127004125/http://www.bookofhook.com/Article/GameDevelopment/TheSecretLifeofGameScript.html`.

[11] LindenResearch. Second Life: Official site of the 3D online virtual world, 2008. as of September 20, 2008, `http://secondlife.com/`.

[12] M. Masuch and M. Rueger. Challenges in Collaborative Game Design Developing Learning Environments for Creating Games. *C5 '05: Proceedings of the Third International Conference on Creating, Connecting and Collaborating through Computing*, pages 67–74, 2005.

[13] W. Prinz. Awareness. In G. Schwabe, N. Streitz, and R. Unland, editors, *CSCW-Kompendium : Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten*, pages 335–350. Springer, August 2001.

[14] Qwaq. Qwaq Forums, 2007. as of September 20, 2008, `http://www.qwaq.com/qwaq_forums.html`.

[15] Qwaq. Qwaq Forums FAQ, 2007. as of September 20, 2008, `http://www.qwaq.com/qwaq_faq.html`.

[16] R. B. Smith, M. Wolczko, and D. Ungar. Thrown from Kansas to Oz: Collaborative Debugging when a Shared World Breaks, 1997. as of September 20, 2008, `http://web.media.mit.edu/~lieber/Lieberary/Softviz/CACM-Debugging/Kansas/Kansas.html`.

[17] H. Takada. A 3D Collaborative Creation Environment with Tile Programming on Croquet. In *C5 '07: Proceedings of the Fifth International Conference on Creating, Connecting and Collaborating through Computing*, pages 125–130, Washington, DC, USA, 2007. IEEE Computer Society.

[18] TheAliceTeam. Alice Webpage, 2008. as of September 20, 2008, `http://www.alice.org/`.

[19] TheCodingMonkeys. SubEthaEdit, 2008. as of September 20, 2008, `http://www.codingmonkeys.de/subethaedit/index.html`.