Rapid Prototyping for Multiplayer Serious Games

Christian Reuter, Thomas Tregel, Florian Mehm, Stefan Göbel, Ralf Steinmetz
TU Darmstadt, Darmstadt, Germany
christian.reuter@kom.tu-darmstadt.de
thomas.tregel@kom.tu-darmstadt.de
florian.mehm@kom.tu-darmstadt.de
stefan.goebel@kom.tu-darmstadt.de
ralf.steinmetz@kom.tu-darmstadt.de

Abstract: Multiplayer games allow players to play together with friends, enriching gameplay with social experiences. While being more motivating for players, these social interactions can also help them develop their social skills (like teamwork or leadership). Especially in serious games there is also the potential for collaborative learning through multiplayer. Since players are able to complement each other in pre-existing knowledge and learning style, this can more effective than learning alone.

Especially the motivational aspect has made multiplayer mechanics ubiquitous in leisure games, where multiplayer modes are also used to extend play time by giving the players the ability to challenge their friends after finishing the (singleplayer) story mode.

However this is not the case for serious games, where a great majority of games is still developed for single players only, leaving the potential of multiplayer serious games unused. One of the reasons for this is the fact that multiplayer games contain an additional layer of complexity evolving around the interactions between the players, the simultaneous actions of multiple players at once and technical challenges like network connections for online games. Having lower budgets and being developed by smaller teams, serious game developers are often unable to handle this additional complexity.

One particular problem regarding complexity is that testing multiplayer games requires multiple players to be present at the time, discouraging rapid prototyping by small teams or single authors. While it might be possible in some cases to start multiple instances of the game and manually switch between them, this solution is quite complicated to use and in most cases not feasible at all.

We addressed this challenge by creating a rapid prototyping environment for multiplayer games, allowing a single author or game tester to control several players simultaneously. To do so we developed a concept to present visual and audio information in such a way, that it is possible to observe multiple player characters at one while still being able to discern at which character the information was directed. We also enabled the author to simulate parallel player actions by adding a record and replay function to queue several actions which are then executed simultaneously. Although implemented for scene-based games created with our authoring tool *StoryTec*, an API has been designed that allows other games to be connected to the prototyping environment as well.

Keywords/Key Phrases: serious games, multiplayer, rapid prototyping, testing

1. Introduction

It has been known for quite some time now that collaborative learning, where learners are able help each other's understanding, is more effective than learning alone (Johnson & Johnson 1988). When supported by digital media, this approach to learning is also called computer-supported collaborative learning (CSCL) (Dillenbourg 1999).

Since games in particular have been shown to support collaborative learning (Voulgari & Komis 2010), it feels natural to combine the principles of collaborative learning with game-based learning by creating learning games that are tailored for collaborative settings. The most natural way to do so is to support multiplayer gaming, which is a common feature of videogames since their early days. In these modes players are able to play together or against each other, either by sharing the controls on a single device or by connecting different devices over a network. Aside from enabling collaborative learning, multiplayer modes can train the players' social skills, too (Voulgari & Komis 2010).

Although there is a multitude of leisure games support multiplayer, both in collaborative and competitive settings, there are only few serious games supporting this kind of gameplay (Lönnroth & Cronqvist 2009; Leong & Liu 2007; Warmerdam et al. 2006) and even fewer learning games designed for collaborative learning explicitly (Zea et al. 2009; Wendel et al. 2013).

One of the main reasons for this fact is that the development of multiplayer games is much more complex compared to singleplayer games, with design, technology and evaluation being all affected. Since serious games are usually developed by small teams with comparatively low budgets (in relation to high-end leisure games) this additional complexity cannot be handled in most cases. This is even more evident in use cases were one author, for example a teacher, wants to prepare a small game for his or her lecture.

One of the challenges we identified is that the testing and evaluation process of multiplayer games requires multiple players to be present even during the first tests. Therefore testing ideas quickly for a rapid prototyping approach requires additional testers to be available at short notice. In this paper we aim to address this issue by describing our rapid prototyping environment for multiplayer games, allowing single authors to "play" a multiplayer game as they were four different players at once.

Another reason for the small number of multiplayer serious games might be authors worrying that the interactions between the players might negatively impact the expected outcome, for example the understanding of learning content. However, this challenge can be addressed by observation and guidance from a game master (Wendel et al. 2014) and is therefore not discussed in this paper.

The paper is structured as follows: First we describe existing approaches and fundamentals related to rapid prototyping for multiplayer games. This is followed by a description of the challenges we identified when testing multiplayer games and our approach for addressing them. Afterwards our tool is described in detail, including the solutions for conveying visual and audio information whose receiving players have to be distinguished as well as for simulating simultaneous input by multiple players. The last sections discuss limitations of the tool and give our conclusion and future work on the topic.

2. Related Work

It has been noted that rapid prototyping, i.e. testing the game with players early and often is a crucial part of the development process for serious games (Kelly et al. 2007). This is also important for the authors themselves to try out their ideas without having to fully implement them (Mehm et al. 2010), potentially wasting lots of time and money on an idea that does not work as planned.

However there is an important limitation to this approach concerning multiplayer games. Since these games are designed to be played by multiple players simultaneously it is very difficult for single authors to test them without getting help by additional players, which is highly impractical for some development settings. The level editor of the leisure game *Portal 2* (Valve Corporation 2011) provides the functionality for an author to play as two separate players by splitting the devices screen into separate views for each player. But since the author has to press a key to switch his controls between the two players, there is no way to attribute sound sources to one of the screens and this functionality does not allow testing concurrent actions.

A possible solution to allow multiple user input is to record user input, as it is often done when studying human-computer interaction (Kukreja et al. 2006). This input could be recorded for each player and then feed into the game at the same to so simulate multiple players at the same time.

To the best of our knowledge there are no further approaches that specifically address the challenge of allowing single authors to prototype multiplayer games on their own, with other approaches for multiplayer rapid prototyping focussing more on the technical development process and still requiring multiple test users (Suomela et al. 2005).

3. Challenges & Approach

During development of several multiplayer serious games (Reuter et al. 2012; Wendel et al. 2013) it was noticed that a rapid prototyping approach is much more difficult when using multiplayer mechanics, especially if the game is meant to be played on different devices over a network. While testing these kinds of games as they are meant to be played (i.e. using multiple testers and PCs) is the standard approach in leisure game development, there are settings where there are not enough testers or devices available. This is typically the case in serious game development where development teams are much smaller and especially when single domain experts are using an authoring tool to create a learning game (Mehm et al. 2009).

Currently running a networked multiplayer game as a single user on one pc can usually be done by starting multiple instances of the same game, connecting them to each other and switching between these instances to represent the corresponding player. However this approach yields several important limitations:

- Visualising all players at once requires switching all game instances into windowed mode and placing these windows next to each other on the users screen. Aside from requiring additional effort by the user, technical reasons sometimes prevent games from being played in windowed mode or from being updated when their window loses focus (so there is only one window showing up-to-date information).
- When sounds are played there is either no way to recognize which instance it came from or in some cases the user is only able to hear the focussed instance. Some operating systems allow adjusting the sound volume for individual instances, but again this requires additional effort controlling another window (sound controls) and it is not available for everyone.
- Since only focussed windows receive user input, the user must switch between them to control different players. This additional effort introduces an artificial delay to the natural reaction time of the user, preventing him or her to simulate concurrent input from different players. This is especially problematic for games including time-critical events.
- And last but not least there is the additional time and effort required to setup multiple instances each time something is meant to be tested, setting one instance to host a game and the other ones to connect to it. Especially when small changes are meant to be tested in rapid iterations, this overhead slows prototyping immensely.

Our approach to address these challenges is to enhance an existing rapid prototyping environment in such a way that it can visualize multiple players at the same time, making the visual and auditory information they receive discriminable. Aside from conveying information, the environment also offers means to interact with the game as multiple players at once and reduces testing overhead by automating game setup.

4. Rapid Prototyping Environment for Multiplayer Games

The rapid prototyping environment for multiplayer games is based on *StoryPlay* (formerly *Bat Cave*) which offers rapid prototyping support for singleplayer games (Mehm et al. 2010; Reuter et al. 2013). Aside from it running the game from a player's perspective, the tool is used to display additional information for the author. This includes the state of hidden variables, the path taken by the player through the game and the state of the internal models used for adaptation purposes like the player or learner model (figure 1).

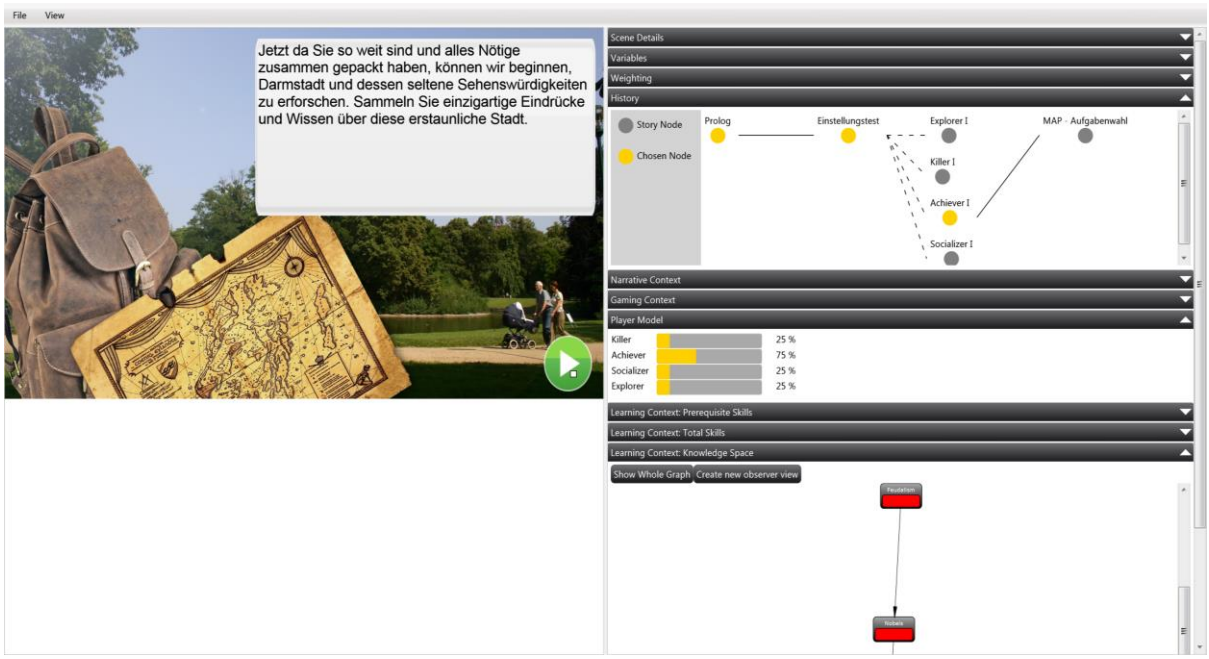

Figure 1: The singleplayer version of StoryPlay with the game running on the left side and additional information about the state of internal adaptation models on the right side.

4.1 General

When extending the prototyping environment to emulate multiple players it was important that the user can easily differentiate between the players he or she is impersonating and is able to quickly recognize which player an information has to be attributed to. To provide this a colour coding scheme for the players was established. Each element of the prototyping environment then uses these colours to mark elements that are specific for a single player.

To automate testing setup the authoring environment is able to start multiple instances of the same game and connect them to each other without requiring additional user input. For networked multiplayer games the environment still establishes network connections pointing to the local machine. An alternative would be to use local function calls instead, but this would require additional development effort by the game's developer and was therefore dropped in favour of compatibility.

4.2 Visualisation

In order to display visual information for multiple players at once, several visualization approaches are available. One prominent method is the shared screen approach. In this approach the game world is displayed on one big screen, trying to fit all existing characters into the view. This is commonly done by zooming out until all characters are visible. The drawback of this method is however that it only works when the players are in close proximity to each other. Therefore it was not suitable for a framework which is intended to work with as many games as possible.

Another approach would be to use multiple screens, but that does not match the requirement of requiring only one device. Instead a split-screen approach was chosen where the screen is separated into parts, showing one player each (figure 2). To save space, this was implemented as a flexible arrangement, adding and removing partitions according on the number of players needed to play the game. Saving space this way is important because of split-screen forcing the individual partitions to decrease in size in order to fit all of them into one viewport. Decreasing the size too much may result in the user not being able to see certain details or, even worse, being unable to interact with them. This flexible arrangement however always uses up the available screen space as good as possible while maintaining the largest player screen size possible. To indicate which view belongs to which player the screens have a coloured border matching the global colour scheme.

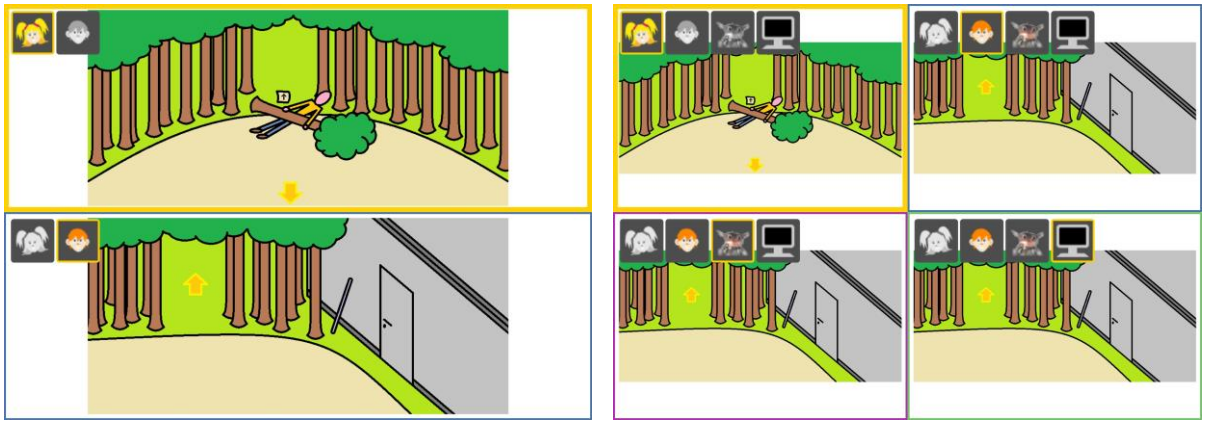

Figure 2: Split-screen setups for two (left) and four players (right).

4.3 Sound

While it is no problem to display multiple viewports on one screen, it is not possible to do the same when it comes to sounds. When playing sound files for multiple players simultaneously over the same speakers there is no way for the user to recognize at which player(s) the sound was directed.

Allowing only one sound at a time would result in lots of interruptions and therefore loss of information when a new sound is started while another one is not finished yet. Queuing up the sounds instead would remove this problem, but can also not be a proper solution because of the queue eventually getting longer and longer. Furthermore this results in sound files being played later than the game events that caused them, which makes it impossible to understand this connection.

It was therefore decided to introduce a transcript which shows every sound played during runtime along with its timestamp and a textual representation if provided by the game (figure 3). There is also

an option to replay sounds in order to prevent loss of information, for example when the user was unable to hear them properly because of multiple sounds overlapping. Every player has its own transcript showing the sounds which have been audible for him or her. These different transcripts are shown in tabs with each one displaying a notification for new entries when not selected. This notification again incorporates the global colour-coding scheme.

In addition to the transcript a mute button for every player was introduced. With this approach the user is able to choose what he or she wants to hear and is able to react when there are too many sounds playing at the same time. While it would have been possible to couple the muting functionality to the currently selected player in split-screen view by muting all deselected players automatically, this option was not implemented in order to give more control to the user. Although one could argue that giving the tester so much options will put additional mental load on him or her, these options are seldom used since rapid prototyping is more focussed on the players' progress through the game and less on individual sound effects.

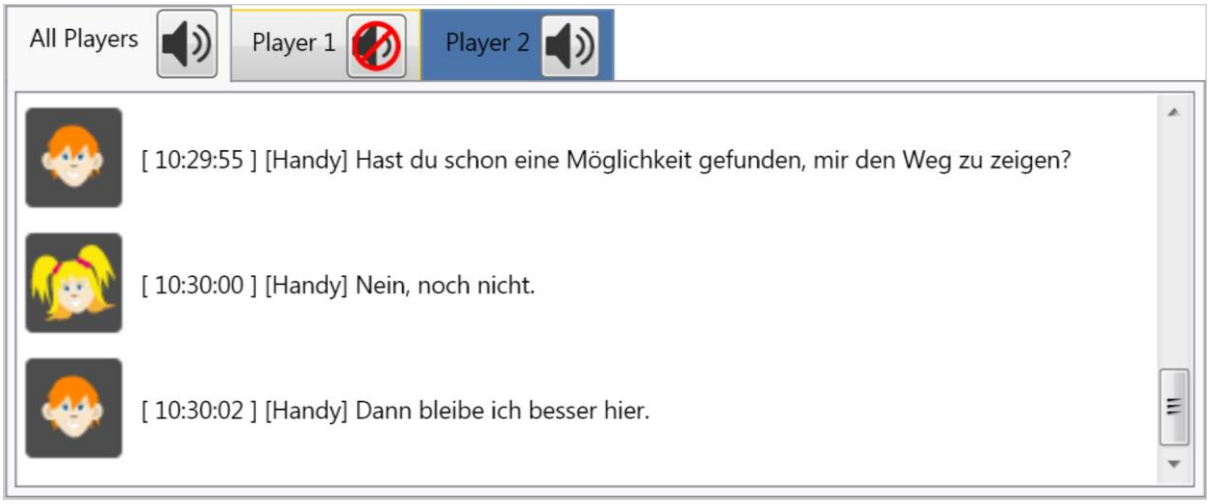

Figure 3: Sound transcript containing speech (with a textual representation). The first player is muted; the second is signalling new entries.

4.4 Input

For input recognition it is important to note that the game as it will be viewed by the players later consists solely of the split-screen views that were assigned to them (see section 4.2). Therefore every user interaction that could also happen in the final game must be connected to these views while interactions on other interface elements (such as the sound transcript, see section 4.3) are additional help functions for the testing user, As with multiple screens, using multiple input devices (aside from technical restrictions) was no option in this scenario.

In the current version of the prototyping environment mouse and keyboard input is supported. Mouse input attribution is trivial because the mouse's position in the overall screen can be directly mapped to one of the partitions and can be easily converted into valid (translated and scaled) coordinates in relation to that partition. This way the individual instance is getting the same mouse coordinates as it would when running as a standalone application.

Keyboard events however consist solely of button presses, so there is no direct way to recognize which player they should be attributed to. It was therefore decided to allow the user to select one screen partition by clicking on its border. The corresponding game instance then receives keyboard events until it is deselected again. As an indicator the currently selected partition gets highlighted with an emphasized border.

4.5 Record & Replay

Some games contain time-critical puzzles where different players need to do certain actions within a small window of time or even simultaneously. This becomes nearly impossible to test for a single person, because of the concurrent inputs and player switching that would need to be performed.

To give a single user the ability to control multiple characters in time-critical events we decided to manipulate the execution process in terms of time. There are multiple possible approaches to this problem. First slowing down the execution process would be an option that gives the user more time to fit multiple actions into small time windows, although this only mitigates the problem. Another possible approach is to introduce a rewind feature and a timeline. This enables the user to go back in time and modify the course of the game from that point on, making up for input events he or she might have missed. This approach however is quite complicated to understand.

It was therefore decided to implement a pause function for the game while simultaneously allowing the user to record player actions to give him or her as much time as needed. When the game is paused all input-events are recorded and inserted into an ordered queue (figure 4) while the tester is able to switch between the players at will. The queue then stores the action performed as well as their order. Actions are again colour-coded and can be deleted and re-sorted using drag & drop. Using this functionality the user can create a list of actions that then are performed instantly in the given order as soon as he or she un-pauses the game and thereby resumes the execution process. Since the actions are executed one after another the testing environment does not introduce race conditions (although naturally we cannot guarantee that the game itself is race condition free). Executing them instantly though makes the events feel simultaneously without requiring the game under test to support parallel processing.

The concept of pre-recording player actions however yields a potential pitfall: For example, a user might record a number of actions with the last one being a click on a button. Should one of the earlier actions now disable this button, the action would not work as intended. To counter this problem a callback was integrated, which can be used by the game to signal that the possibility space for player actions has changed. Upon receiving it, the environment pauses itself automatically during execution of the queue and displays a hint, allowing the user to modify the queue accordingly.

Figure 4: Record & replay queue consisting of mouse events. A timeline was also considered as a visualisation technique, but since all actions happen without any delays and do not have a execution time there is no need to display them in relation to time.

4.6 Internal Adaptation Models

Since it is one of the main features of the prototyping environment to display the internal adaptation models of games created with the *StoryTec* authoring tool during runtime, it was important to extend them for multiplayer settings, too. As they are displayed as visual information similar to the games output, a universal solution would be to also multiply the corresponding UI elements. However, since they require a multitude of screen space in this case it was decided to merge multiple models for each player into a single visualisation. Please note that while the models themselves are specific for the adaptation approach described in (Mehm et al. 2010), the visualisation techniques that were used for multiplayer environment can easily applied to similar models as well.

Figure 5: Player model of four players (left) in comparison to the singleplayer version (right).

Merging bar charts like the player model is relatively straightforward. For singleplayer there are four bars which represent separate dimensions along which the player is assessed. For multiplayer

settings each of the bars is split into a number of sub-bars, each representing the value for one player and following the global colour-coding scheme (figure 5). This also allows direct comparison between the players and easy outlier detection compared to special separated charts.

A similar approach was chosen for the knowledge space graph detailing the learning content of the game, the relationship between individual topics and the player's estimated knowledge level. Since the topics and their relations are identical for every player, only the acquired knowledge may vary between them. Originally indicated by a single colour (the more green the higher the probability the player learned this topic), it was decided to replace the coloured bar with a bar chart (figure 6). The bars display the probability for each player using the familiar colour coding – again enabling easy comparison.

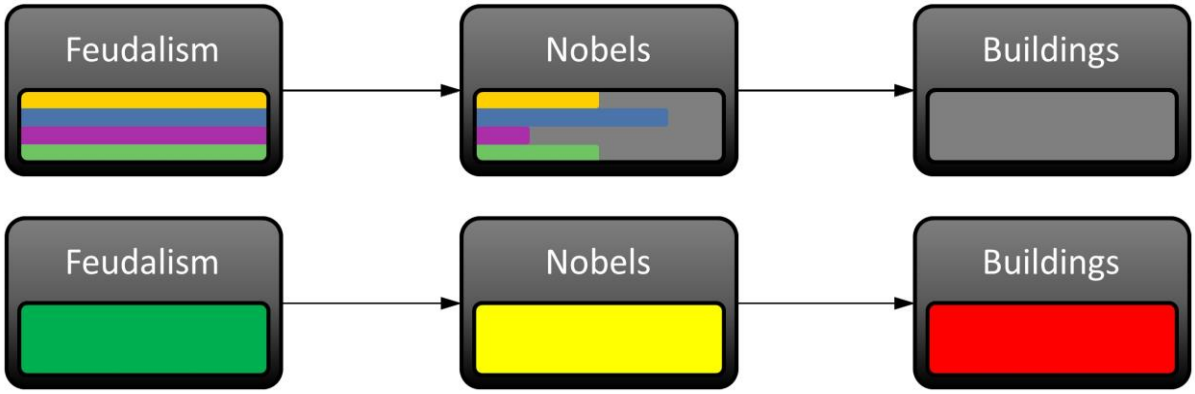

Figure 6: Knowledge space of four players (top) in comparison to the singleplayer version (below).

The model where the most changes had to be made was the history of visited scenes. Aside from displaying the players' progress through the game in a linear fashion, the environment also displays alternative scenes as well as their calculated appropriateness score when the game made adaptive choices. When used for multiple players, this can get very complex however. Players can be in completely different scenes if the game allows them to split up. They can also move through the game at a different pace, meaning that one player might be in his / her fifth scene while another one is in the second. However, it should also be recognisable when multiple players visit the same scene at the same time.

It was therefore decided to move the alternatives and their score to an overlay and use the additional space to display one row for each player (figure 7). Entering and leaving scenes is displayed on a timeline to preserve the global order of events between the players. Boxes represent the time a player was visiting a scene with small gaps for when a transitions into another scene occurred. The colour of the box shows which other players were in the same scene at the same time, making it easy to not only see where each player was at a given time, but to also see who was there with them. If a player is alone in a scene his or her colour is used for the whole box, when multiple players are present the box is divided into equal parts which are then coloured in the individual player colors.

Figure 7: Scene history for multiple players. The intro is visited by all players simultaneously, for example.

4.7 Extensibility

Although the prototyping environment was originally developed to work with games created by a specific authoring tool, it was an important goal that it can be used by other games as well. Therefore an API (written in C#) was defined, containing functions that a game must implement in order to be connected to the environment. Depending on the prototyping features that the game's developer

wants to use, only a subset of the functions must be implemented (table 1). The functions related to the "basic" feature are used for game setup and are required regardless of which specific features are meant to be used. While some functions are always required, others are not strictly needed for a specific feature to work but can provide additional functionality for the user.

Table 1: API functions and the features they are related to.

Function	**Feature**
Starting multiple instances of the game	Basic (required)
Setting a canvas to draw the graphical output on	Basic (required)
Triggering the main loop	Basic (required)
Hosting and joining a game	Basic (required)
Triggering keyboard input events	Basic (if the game uses keyboard controls)
Triggering mouse input events	Basic (if the game uses mouse controls)
Muting the sound	Sound (required)
Providing a transcript for sound output	Sound (optional)
Pausing the game	Record & Replay (required)
Signaling when a players options have changed	Record & Replay (optional)

There are further functions for displaying the internal adaptation models, but since their visualisations are tailored specifically towards scene-based games created with the corresponding authoring tool they are not part of the general API.

5. Limitations

While in theory our prototyping environment works for almost every game, there are practical limitations to controlling multiple players at once. Especially in fast paced games requiring continuous input from the players (like action or racing games) the record & replay feature had to be used all the time, which would increase testing time immensely and is therefore not a viable option.

Another limitation is that the approach only works for a comparatively small number of players. It was decided to allow a maximum of four players, since a greater number of players requires the individual split-screen views to be smaller. This makes it very hard to notice smaller details in the game and to interact with them. Controlling more players also increased the mental load on the user who has to keep track them. Multiplayer games on a much bigger scale however also require much more development effort and are therefore typically developed by larger teams. In this case the application scenario of single authors testing their game is not applicable anymore.

Furthermore it must also be kept in mind that the environment is intended for rapid prototyping only. In order to get valid evaluation results for example, the final product still has to be tested with multiple players.

6. Conclusion

In this paper we described a rapid prototyping environment allowing single authors to easily test multiplayer games without requiring additional testers or devices. To do so we developed concepts to convey visual and auditory information simultaneously, using split-screen and audio transcript techniques, while still allowing users to identify at which of the players the information was directed. We also enabled authors to emulate simultaneous input by multiple players using a record & replay approach. While being originally developed for a specific group of games, an API was defined allowing other games to use the environment as well.

The environment as described in this paper has enabled us to use rapid prototyping approaches for multiplayer serious games, has accelerated testing and is therefore used regularly during game development. It also works great for demonstration purposes when showing the games without having to setup multiple devices and recruiting players to control them, allowing quick demo sessions with potential investors.

As a next step we want to share these positive outcomes with other serious game researchers and developers, therefore our future work will include the prototypical connection to a wider range of games. We will also investigate alternative rendering methods for the visuals since the ability to draw the games graphical output onto a C# canvas is the main difficulty when implementing the API. Aside from that we will also refine the user interface and visualisations of the tool.

References

Dillenbourg, P., 1999. *Collaborative Learning: Cognitive and Computational Approaches. Advances in Learning and Instruction Series.* P. Dillenbourg, ed., Oxford: Elsevier Science, Inc., PO Box 945, Madison Square Station, New York, NY 10160-0757 (72). Web site: http://www. elsevier. com.

Johnson, R.T. & Johnson, D.W., 1988. Cooperative Learning: Two Heads Learn Better Than One. *Transforming Education: In Context*, 18, p.34. Available at: http://www.context.org/iclib/ic18/johnson/.

Kelly, H. et al., 2007. How to Build Serious Games. *Communications of the ACM*, 50(7), pp.44–49.

Kukreja, U., Stevenson, W. & Ritter, F., 2006. RUI: Recording user input from interfaces under Windows and Mac OS X. *Behavior Research Methods*, 38(4), pp.656–659. Available at: http://dx.doi.org/10.3758/BF03193898.

Leong, C. & Liu, C., 2007. A framework for Collaborate SCORM-Compliant Web-Based Authoring System. , (250), pp.292–294.

Lönnroth, L. & Cronqvist, B., 2009. Jirafa Learning World - Massively Multiplayer Online Role- Playing Game for Primary School Math Education. In *Proceedings of the 3rd European Conference on Games Based Learning*.

Mehm, F. et al., 2009. Authoring Environment for Story-based Digital Educational Games. In Y. Cao et al., eds. *Proceedings of the 1st International Open Workshop on Intelligent Personalization and Adaptation in Digital Educational Games*. CEUR Workshop Proceedings, pp. 113–124.

Mehm, F. et al., 2010. Bat Cave: A Testing and Evaluation Platform for Digital Educational Games. In *Proceedings of the 4th European Conference on Games Based Learning*. pp. 251–260.

Reuter, C. et al., 2013. Evaluation of Adaptive Serious Games using Playtraces and Aggregated Play Data. In C. V. de Carvalho & P. Escudeiro, eds. *Proceedings of the 7th European Conference on Games Based Learning*. Porto, Portugal: Academic Conferences Limited, pp. 504–511. Available at: ftp://ftp.kom.tu-darmstadt.de/papers/RMGS13-1.pdf.

Reuter, C. et al., 2012. Multiplayer Adventures for Collaborative Learning With Serious Games. In P. Felicia, ed. *6th European Conference on Games Based Learning*. Reading, UK: Academic Conferences Limited, pp. 416–423.

Suomela, R., Koskinen, K. & Heikkinen, K., 2005. Rapid prototyping of location-based games with the multi-user publishing environment application platform. *IET Conference Proceedings*, pp.v2:143–v2:143(1). Available at: http://digital-library.theiet.org/content/conferences/10.1049/ic_20050228.

Valve Corporation, 2011. Portal 2.

Voulgari, I. & Komis, V., 2010. Antecedents of Collaborative Learning: Insights from Massively Multiplayer Online Games. In *2010 International Conference on Intelligent Networking and Collaborative Systems*. Ieee, pp. 32–39. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5702074 [Accessed March 27, 2012].

Warmerdam, J. et al., 2006. *SimPort: a Multiplayer Management Game Framework*,

Wendel, V. et al., 2013. Designing Collaborative Multiplayer Serious Games. *Education and Information Technologies*, 2(18), pp.287–308. Available at: http://link.springer.com/article/10.1007/s10639-012-9244-6.

Wendel, V. et al., 2014. Instructor Support in Collaborative Multiplayer Serious Games for Learning - Game Mastering in the Serious Game “Woodment.” In *Proceedings of the 6th International Conference on Computer Supported Education*. SciTePress, pp. 49–56.

Zea, N.P. et al., 2009. Design of Educational Multiplayer Videogames: A Vision From Collaborative Learning. *Advances in Engineering Software*, 40(12), pp.1251–1260.