Cockburn, A. and Greenberg, S. (1995). TurboTurtle: A Collaborative Microworld for Exploring Newtonian Physics, Proc1st CSCL Conference on Computer Supported Collaborative Learning, pp. 62-66, Bloomington, Indiana, October 17-20, LEA Press.

TurboTurtle: A Collaborative Microworld for Exploring Newtonian Physics

Andy Cockburn[1] and Saul Greenberg[2].

[1] *Department of Computer Science, University of Canterbury, Christchurch, New Zealand.*
[2] *Department of Computer Science, University of Calgary, Calgary, Canada*

Abstract

This paper describes TurboTurtle, a dynamic *multi-user* microworld for the exploration of Newtonian physics. With TurboTurtle, students can alter the attributes of the simulation environment, such as gravity, friction, and presence or absence of walls. They can also manipulate the "turtle" (a movable ball) directly. Students can adjust its position, velocity and mass; change its kinetic and potential energy; and apply a force to it by strapping a rocket to its back. Through TurboTurtle's "group-awareness," several students, each on their own computer, can simultaneous control the microworld and gesture around the shared display.

This paper focuses on the rationale behind the major "group-awareness" design decisions made during our development of TurboTurtle.

Keywords — microworlds, groupware design, user interface design.

1. Introduction

Microworlds, or computer simulations of restricted environments, are an intuitively appealing way to promote discovery and exploratory learning [4]. One type of microworld, and the subject of this paper, simulates an adjustable Newtonian universe. In it, students can experiment with concepts such as gravity, friction, force, velocity, and so on, and see how changes in their value affect the objects moving within the simulation.

Microworlds—Newtonian or otherwise—are not new. They were first conceptualized by Papert in his 1980 book "Mindstorms," but in that era they were implemented as crude systems that required cryptic and error-prone command line interfaces e.g., Logo. In the 80's and early 90's microworld simulations became dynamic environments that students could alter on the fly, allowing direct manipulation of microworld objects. Smith's Alternative Reality Kit is one such example [5]. In this paper, we claim that another evolutionary step is about to take place: microworlds will become *group-aware*, actively allowing several students to view and manipulate the simulation.

We are investigating the application of collaboration-aware groupware technology and methods to build microworlds that re-enforce discussion by students around the learning tool. In TurboTurtle, each student has their own computer screen and input devices. They share the view of the simulation, have telepointers to promote deictic references and gesturing, and can simultaneously manipulate the microworld. Since students do not have to be co-located, we assume that they can talk to each other over an audio channel such as a speaker-telephone.

This paper presents the user interface of TurboTurtle, and discusses the design rationale that governed the development of TurboTurtle's group-aware features.

2. TurboTurtle's User Interface

To provide the context for our later discussion of TurboTurtle's collaboration-aware facilities, this section presents the single-user interface to TurboTurtle.

We wanted to provide a seamless interface that allowed all the student's cognitive effort to be directed at the contents of the microworld. Beyond the "see and point" premise of modern graphical user interfaces [3], we wanted TurboTurtle to make extensive use of sound, colour, and animation to capture the interest of young users.

What do students using TurboTurtle see and do? Figure 1 is a snapshot of a student's session. The lower half displays the simulation with the turtle being the ball at its centre. The turtle's location can be changed directly by dragging it with the mouse, and its direction and velocity altered by "throwing" it. The top of the figure shows a control panel, where tangible properties are set through constantly visible graphical sliders. These include the controls to change the turtle's size, mass, speed, the degree of friction and gravity, and so on. Students use the pull-down menus to access advanced features of TurboTurtle.

Within the simulation, the turtle's trail, a line of ink that follows the turtle's movement, can be switched on or off. The walls in the microworld can be changed as well. The turtle bounces off "hard" walls and passes through "transparent" ones (which causes it to wrap-around the display). When only the ground is hard, the relative location of the ground to the turtle is remembered as it wraps through successive screens. Students can also display a mountain scenery backdrop, which provides additional visual cues to the altitude of the turtle. As the turtle gains altitude the backdrop changes to show smaller mountains, a row of aeroplanes, and then satellites. Of course, the trails and the mountain backdrop can be cleared at any point.

Figure 1 shows the turtle's trail after a series of user-driven changes to the microworld[1]. Starting in the middle of the screen, the turtle moved down and to the right with no mass or gravity. After seeing and hearing it bounce off the walls four times, the student added mass and positive gravity, causing the turtle to bounce under gravity (the sin curve). She then changed gravity to a small negative value, causing the turtle to bounce off the roof of the microworld. Finally, she added friction, causing the turtle to eventually slow to a stop.

[1]Naturally, the figure fails to show the turtle's movement, the dynamically changing slider values, the colour, and the audio output that are fundamental to the student's sense of fun.

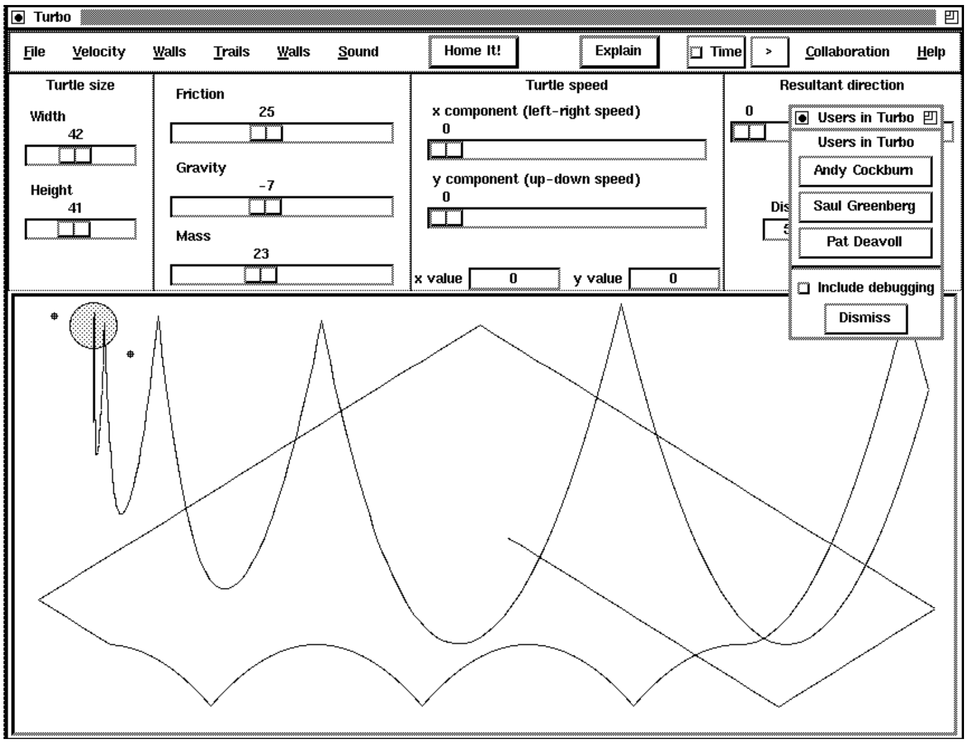


Figure 1: The main window to TurboTurtle.

The microworld clock (set by the *time* button) lets the student freeze the microworld. This allows specific values to be set prior to running a new experiment. Time can run smoothly, giving a continuous real-time simulation, or discretely which allows students to scrutinise the change in variable values at critical instants. For example, the student could investigate changes in potential and kinetic energy by discretely stepping through the turtle's motion as it hits the floor and as it reaches the apex of its motion under gravity.

TurboTurtle is intended for students ranging from 7–17 years in age, and for peer groups where individuals have different knowledge and talents. TurboTurtle supports this wide range of abilities through two sets of controls: concrete and abstract. Concrete controls, which are continuously visible, present concepts that are familiar and frequently accessed by the youngest students (as shown in Figure 1). Abstract controls for more sophisticated manipulations are revealed on demand by mature users. For example, TurboTurtle lets advanced users view and manipulate values in Kinematic equations, which are selected as menu options in the "Laws" pulldown menu (figure 2). Choosing the first "Energy" equation creates a window into the microworld that dynamically displays the turtle's potential and kinetic energy. The second "Rocket" equation creates a control panel that allows students to attach rockets of varying force and fuel-time to the turtle, which lets them examine the inter-relation between force, acceleration, mass, gravity and friction. Other kinematic equation options provide dynamic simulations of the behaviour of a user-specified set of formula values: essentially they provide an animated calculator.

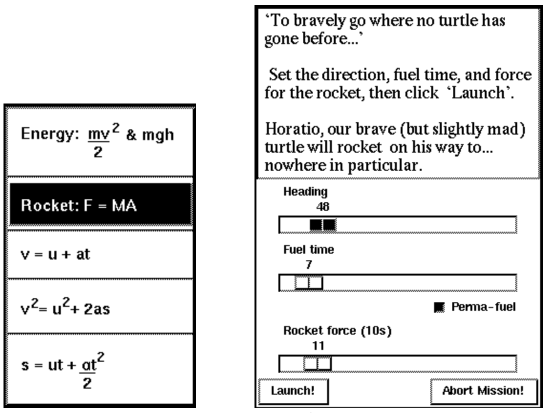
Recoverability in TurboTurtle allows users to experiment with features, safe in the knowledge that they can get back to their starting state. Exploring a dynamic microworld is risky because it can change rapidly. In TurboTurtle, for instance, a student may arrange a group of slider values to simulate a rocket working against a certain friction, gravity, and mass. When the rocket is launched, the simulation runs and slider values will change to reflect the dynamically changing environment. In early trials of the system, we noted that students frequently forgot or mistook one or more slider values. When they ran the simulations, they were often immediately aware of their error, and found it annoying to have to reset the values that the system had changed. Similarly, students may be reluctant to change system parameters away from an interesting state for fear of corrupting them. TurboTurtle lets students recover from their ventures by allowing them to save and reload named states of the microworld. Of course, this is an explicit action that students must take, and they will likely do this for only highly interesting states. Allowing time to run backwards is also a type of undo, and is a high priority in our further work.

3. The Communal Microworld

TurboTurtle's group-awareness allows small groups of students (diads or triads) to simultaneously manipulate and talk about the simulation. This section is primarily concerned with the design decisions that governed the development of TurboTurtle's multi-user features. It begins with an overview of the system's group-awareness, and continues with the design decisions made[2].

In static images, such as the screen snapshot in this paper, collaborative use of TurboTurtle appears to be almost identical to single user usage. Group awareness, however, makes its style of use significantly different. In the description below we focus on these differences by assuming that two or three distance-separated students, each with their own computer, are looking at the screen and are talking to each other by a speaker-phone.

[2]TurboTurtle's collaborative features are supported by GroupKit, a toolkit for groupware.


'To bravely go where no turtle has gone before...'

Set the direction, fuel time, and force for the rocket, then click 'Launch'.

Horatio, our brave (but slightly mad) turtle will rocket on his way to... nowhere in particular.

Figure 2: Selecting and using "formal" experiments in TurboTurtle. The 'Laws' menu, and 'Rocket' configuration options.

Each student sees exactly the same running simulation on their display. The turtles are in the same position and move at the same speed, the trails are in the same place, and the background scenery is identical. Similarly, the controls are *mostly* identical. They are in the same window location and have the same setting. However, students can decide to change their view of some of the controls. For instance, one could be examining turtle speed by its x-y components, and the other by speed and direction (Turtle meets Pythagoras!) Similarly, one could display independently some of the advanced control panels, such as the Energy panel.

All students can work simultaneously doing anything they want at any time. For example, one student might move the turtle, while another adjusts its speed, and another alters the world's gravity. As in real life, they could even try to adjust the same control, which would cause it to "bounce" back and forth as they fight over its position! As any control is being adjusted, the new position is immediately reflected on all displays.

Students can see the other person's location on the screen by a *telepointer*, shown as the multiple cursors in Figure 1 (the dots by the ball). Not only is a student's own cursor continuously drawn and updated on the display, but so are the cursor's of their partners. A special menu option called "collaborators," presents a dynamic list of all the students in the learning session (Figure 1, top right). Pressing a student's name will raise an information window describing that student.

3.1 Design decisions

In spite of the conceptual simplicity of collaborative interface, many groupware design decisions had to be made. These included how students viewed the simulation, how they would control it, and how they could share their deictic references.

Viewing the simulation. What does it mean to have several students view the simulation? We considered four alternative approaches to view sharing.

1. *Strict WYSIWIS views*[3]. Every student would view exactly the same thing on their display: the ball as it was bouncing, the changes in background scenery; the ball's location in the scene; the tracing of ball movements; and so on.
2. *Relaxed WYSIWIS views*. While the state of the simulation would be the same, every student could have different viewports on it. That is, one student could be looking at (say) a zoomed out view, while the other could be zoomed in on a particular scene.
3. *Unconnected views, same simulation parameters.* The parameters of the simulation would be the same across all systems, but the effects of the parameters on the ball would be local. This could simply be a matter of each student's computer moving the ball at its own speed, but since performance of the computers would differ slightly, so would the position of the ball. Alternatively, a student could create a smaller simulation room by shrinking the window, which means that the ball would be bouncing off the walls at different places and frequencies. In either case, the ball position in the simulation would differ across the views.
4. *Unconnected views, different simulation parameters.* The parameters of each student's simulation would differ, thus affecting not only the position of the ball on a local display, but its overall behaviour as well.

We wanted the view to act as a conversational prop providing a focus for the students discussion [2]. We thought the strict WYSIWIS view would be the best choice to encourage this. The display becomes a shared cognitive artifact, and speech references would remain within the context of the shared image. Strict-WYSIWIS would allow students to pose questions and comments to each other such as "why did the ball bounce that way?" or "the ball just moved into outer space" or "look at the shape of the trace."

In contrast, views 2 through 4 would cause progressively greater breakdown in the discussion, probably resulting in greater confusion and ultimately less interaction between students (a similar observation was made by Tatar *et al.*, [8]). Relaxed WYSIWIS causes people to ask "can you see this" or respond "which one?" Students using the unconnected view with the same parameters would have to explain what their ball is doing on their display. With different parameters, they would also have to explain the settings.

Although the relaxed and unconnected approach does give the student the ability to customise their view, the strict WYSIWIS view seems preferable as it reinforces the microworld's role as a conversational prop.

Controlling the simulation. The simulation is directed by manipulating the controls on the control panel: sliders, buttons, menu selections; and by directly moving the ball position in the view. Given a strict WYSIWIS view and identical simulation parameters across the system, there remain several options for presenting the controls and for having students interact with them.

First, how do students view the controls? Controls could be identical on all displays (strict WYSIWIS), or different students may see different controls in their view (relaxed WYSIWIS). The choice is not as clear here. In a complex simulation system such as TurboTurtle, the number of controls, including the pull-down menus and the pop-up panels, are huge and can clutter the display quickly. It seems reasonable to have a strict WYSIWIS view of the primary controls,

[3]What-you-see-is-what-I-see, or WYSIWIS was coined by Stefik *et al.*, (1987) in a discussion about a shared whiteboard system.

while having a relaxed WYSIWIS view of advanced controls.

Second, how do students see the setting of a changed control? In a "parcel post" model [8], the changed value of the control would be delayed until the student had completed their action. For example, if one student adjusted the gravity slider from 0 to 20, the other student would only see the slider jump instantly to 20. In contrast, the "interactive" model causes the control's state to be transmitted as it is being manipulated. Sliders move, buttons get pressed, pulldowns selected. Clearly, the interactive model is preferable, as students will be able to see the changes as they are made, and are less likely to miss the actions of the others.

Finally, who has permission to use what controls? Several choices are possible. Students could be assigned to a mutually exclusive subset of controls. A turn-taking model could be enforced, where only one student at a time can manipulate the controls. Or students may be allowed simultaneous access to all controls, constrained perhaps by some mechanism to minimise confusion if two people try to manipulate the exact same control. We have opted for simultaneous access because we believe it will encourage each student to explore and control the simulation. Anyone is allowed to do anything at any time. The key to making this work is to provide rich dynamic feedback between students that leaves them constantly aware of each other's actions, and encourages them to talk.

In summary, students have mostly the same image of the core controls, with advanced controls being optional to avoid screen clutter. Anyone can manipulate any control at any time, and all user's manipulations are constantly visible.

Deictic references allow people to point to things and refer to them using words such as "there," "this one," and "that" [8]. A strict WYSIWIS view by itself does not provide enough information to let students understand each other's deictic references, for they cannot tell what part of the screen they are attending. Breakdown of deixis has been a common failing of groupware [8].

The easiest way to support deictic reference is through *telepointers* [1,7], which are cursors, one for each student, that are continuously visible on all displays (as in Figure 1). Telepointers are useful in microworlds for deictic and other types of references. First, they act as a locus of attention; one student can assume that the other is directing their gaze at their cursor. Second, they become an artifact that they can talk around e.g., the phrase "look at this" is tied to the spot on the screen that the person is pointing to. Third, their animation becomes a gesture. For example, a student circling an area of the screen tells others to attend to all of the items in that area. Finally, they provide a cue of someone's intent. If the telepointer is moving towards a slider, then one expects that the next action could be to change the setting of the slider. This helps mediate who is doing what on the display.

Telepointers were included in all parts of TurboTurtle. People can gesture around the shared view, focus attention to settings on the control panels, and implicitly indicate both their intent and their action when manipulating a control.

4. Summary

There are many directions for further work in TurboTurtle. With respect to refinements of the microworld, the world's our oyster: there is no obvious end to the types of domain that can be covered by a group-aware simulation. There is, of course, much to be done investigating the nuances of adding collaboration to CSCL. To date, our design of TurboTurtle has been primarily motivated by technical interests. Although we have run ad-hoc usability studies that detect the "large grain" usability flaws, we have yet to take TurboTurtle to the battlefield. We are very aware of the disparity between a designer's expectations of use and the end-users' behaviour. Extensive observation of TurboTurtle in use is required.

Acknowledgments

The collaboration between the two authors was made possible by the University of Canterbury's Erskine Fellowship.

Availability

TurboTurtle is available directly from the first author of this paper.

References

1. Greenberg, S, Roseman, M, Webster, D, & Bohnet, R. 1992. Issues and Experiences Designing and Implementing Two Group Drawing Tools. *Pages 139--150 of: Proceedings of the 25th Annual Hawaii International Conference on the System Sciences, Hawaii, January,* vol. 4.
2. Hill, RD, Brinck, T, Patterson, JF, Rohall, SL, & Wilner, WT. 1993. The Rendezvous Language and Architecture. *Communications of the ACM*, **36**(1), 62–67.
3. Shneiderman, B. 1987. Direct Manipulation : A Step Beyond Programming Languages (excerpt). *Pages 461–467 of*: Baecker, RM, & Buxton, WAS (eds), *Readings in Human-Computer Interaction: A Multidisciplinary Approach.* Morgan Kaufmann.
4. Smith, DC, Cypher, A, & Spohrer, J. 1994. KIDSIM: Programming Agents Without a Programming Language. *Communications of the ACM*, **37**(7), 55–67.
5. Smith, RB. 1987. Experiences with the Alternate Reality Kit: An Example of the Tension between Literalism and Magic. *Pages 61--67 of: Proceedings of ACM CHI+GI'87 Conference on Human Factors in Computing Systems and Graphics Interface.*
6. Stefik, M, Bobrow, DG, Foster, G, Lanning, S, & Tatar, D. 1987. WYSIWIS Revised: Early Experiences with Multiuser Interfaces. *ACM Transactions on Office Information Systems*, **5**(2), 147–167.
7. Tang, JC. 1991. Findings from Observational Studies of Collaborative Work. *International Journal of Man-Machine Studies*, **34**, 143–160.
8. Tatar, DG, Foster, G, & Bobrow, DG. 1991. Designing for Conversation: Lessons from Cognoter. *International Journal of Man-Machine Studies*, **34**(2), 185–209.