mulTetris - a Test of Graspable User Interfaces in Collaborative Games

Samuel Audet, Matios Bedrosian,
Cyril Clement, Monica Dinculescu
McGill University
3480 University Street
Montreal, Quebec
saudet@cim.mcgill.ca, mbedro@ece.mcgill.ca,
ccleme6@cs.mcgill.ca, mdincu@cs.mcgill.ca

ABSTRACT
Most PC games are limited to a keyboard and a mouse, where the physical capabilities of the human hand are not fully taken advantage of. Based on Tetris, a traditionally single player keyboard-game, *mulTetris* brings a new easy-to-use interface involving a brick-like device as part of a collaborative game. In *mulTetris*, the controlling brick is tracked by a camera and is used to directly manipulate the tetrominoes. The interface is easy to learn and natural, using either the user's existing skills, or easily transferable ones. This paper discusses each of development stages of *mulTetris*, the iterative design, as well as the HCI principles used in the development of the final system.

ACM Classification H5.2 User Interfaces - Graphical user interfaces.

General Terms Design, Human Factors

KEYWORDS: Tetromino, brick, two-handed interaction, magnet metaphor, collaborative interface

INTRODUCTION

mulTetris is a Graspable User Interface[3] concept based on the traditional Tetris game. The user can move and rotate falling tetrominoes (blocks of predefined shape) in such a way that they align and form complete lines at the bottom of a board. The falling rate of the tetrominoes increases with time, and the game is over when a tower of tetrominoes has reached the top of the board.

The *mulTetris* interface is designed as a collaborative, two-handed interaction system, making use of the magnet metaphor. It is aimed to use natural mappings, existing or easily transferable skills in order to improve the interaction between the user and the system. Two physical artifacts, henceforth referred to as "bricks", are used to interact with the game tetrominoes displayed on a horizontal screen. The user can manually interact with the falling tetrominoes by moving or rotating the bricks, thus taking advantage of kinesthetic feedback, and lessening the cognitive load.

Figure 1. User interacting with the *mulTetris* alpha system.

In addition, *mulTetris* is a one-input one-output system. The output system is represented by a horizontal display screen, while the input system consists of a colour video camera coupled with computer vision software to track the position and the orientation of the controlling bricks. The computer vision software, dubbed CV4HCI, is based on OpenCV[4], an open source computer vision library written in C by Intel. CV4HCI adds functionality on top of OpenCV, and is used to track the position of the bricks.

RELATED WORK

Tangible user interfaces[1], as introduced by Ullmer, use physical objects as representations and controls of digital models. For example, the BuildIT system uses brick-shaped physical handles in creating furniture layouts[2], while the Zowie system allows the user to manipulate physical models of game characters to interact with a virtual play set[1].

ActiveDesk[3] developed by Fitzmaurice, Ishii and Buxton is a similar Graspable User Interface system. The interface allows the user to control virtual objects through physical artifacts that become the input devices, or control handles.

Krueger's VideoDesk [5]is an early desk-based system that allows the user to interact with projections. An overhead camera detects the user's hand and body position, while a horizontal light provides high-contrast gesture input for interactions.

SYSTEM OVERVIEW

The Magnet Metaphor

From the user's perspective, the brick can be thought of as a magnet, and the tetrominoes as metal pieces, placed underneath the display screen. Placing the brick on top of the screen causes the tetromino to become selected, and to be "attached" to the brick. Moving or rotating the brick will also move or rotate the tetromino. This is a very natural action, as moving a magnet will also move any metal objects physically attached to it. Finally, raising a brick away from the screen will de-select the tetromino, and remove the attachment to the tetromino.

Perceived affordances (such as highlighting) were added to the tetrominoes to suggest whether a tetromino is attached to a brick or not.

Graspable User Interface

Mackenzie introduced the idea of prehensile behaviour, which he defined as "... the application of functionally effective forces by the hand to an object for a task, given numerous constraints."[6] Implementing *mulTetris* as a Graspable User Interface employs existing motor skills, as well as kinesthetic feedback, allowing the user to directly interact with the brick and thus with the virtual tetrominoes.

In addition, Graspable User Interfaces diminish what Raskin considers to be the "Operating System" problem of modern Graphic User Interfaces: introducing an intermediate layer between a user and an application. In the case of the traditional Tetris interface, the user is forced to use an unnatural device, such as a keyboard, in order to control physical blocks. This does not aid the interface, as the user needs to create an additional mapping between the keyboard keys and the direction of movement/rotation of a tetromino. However, in *mulTetris*, this mapping is not needed: the user can physically move or rotate the brick as he/she would want the tetromino to move or rotate.

By removing the additional mapping, as well as taking advantage of prehensile behaviour, the user's cognitive load is lessened.

Two-handed Interaction

Another important advantage of Graspable User Interfaces is that they encourage two-handed interactions by using spatial reasoning and kinesthetic memory.

Studies performed have shown that tactile/kinesthetic feedback allows a user to engage in other tasks, while manipulating a physical object[3]. In addition, users can, and prefer to, perform at least two different operations in parallel: moving and rotation of a physical object.

It is important to note that allowing a tetromino to be moved and rotated simultaneously is the main advantage of the *mulTetris* interface. Combined with the use of muscle memory, the *mulTetris* interface allows a user to find the most appropriate location for a given tetromino, and simultaneously move the tetromino to that location, without needing to change the visual focus.

Input System

The input system consists of a colour video camera, coupled with the CV4HCI computer vision software. CV4HCI is used to track the bricks and pass on their position, size and orientation to *mulTetris* running in a Java Virtual Machine.

The main components of the system are CAMSHIFT[7] (see section 6 for details), a fast colour tracker used with a maneuvering Kalman filter to smooth out noise from the tracking information, and a control panel used to adjust parameters of the CAMSHIFT tracker and receive proper feedback on its performance.

DESIGN STRATEGY

HCI Principles

A successful user-centered design requires attention to essential HCI principles. Specific guidelines, such as the following three principles of system design as suggested by Gould [8]were followed, to ensure the consistent development of *mulTetris*:

1. *Early Focus on Users and Tasks*: Paper prototypes and a very simple functional prototype in the early design stages helped us discover some limitations of the brick interface, such as the rotation mechanism employed; the early design change allowed us to user-test the new rotation concept before introducing it in the final system.

2. *Empirical Measurement*: At each stage, from paper prototyping to the alpha system, the feedback given by the evaluators was thoroughly analyzed; any problems that appeared were discussed and resolved in the following releases of the system.

3. *Iterative Design*: Closely related to the previous step, user comments and complaints regarding a certain stage were taken under consideration leading to several iterations of design, implementation, testing, and re-design.

Usability Heuristics

Also, the group's strategy was heavily centered on the following HCI usability heuristics[9]:

- *Aesthetic and Minimalist Design*: The Graphic User Interface does not contain any irrelevant or rarely needed information.
- *Visibility of System Status*: All the available options are made clearly visible to the user. In addition, enough feedback is given to inform of the result of a specific action. The user's score and other information, as well as the preview boards showing the next tetrominoes are easily found without interfering with the gameplay.
- *Recognition Rather than Recall*: The options available are displayed in such a way to be easily recognizable. Furthermore, should the user need additional help, game instructions can be readily accessed through a help button.

- *Consistency and Standards*: Careful attention was paid to the use of consistent terms, as to avoid user confusion. For example, the term "brick" always refers to the physical device used to interact with the game, while "tetromino" refers to the virtual objects in the game.
- *Recovery from Errors / User Control and Freedom*: The user may restart the game at any time, for example if he/she made a strategic mistake. Moreover, like most games, the game may be paused at the user's discretion.

SYSTEM EVOLUTION

The development of *mulTetris* consisted of four main stages. Each new release included interface improvements from the previous and was evaluated by various groups who may or may not have been familiar with HCI principles depending on the type of results required (for example, it was important that during usability tests, a user with no HCI knowledge was asked to participate, in order to determine how usable and easy to learn the interface was).

Initial Paper Prototype

A small group of users were asked to participate in a guided paper prototyping session. The users had experience with the traditional keyboard & mouse Tetris games and were generally unfamiliar with the principles of HCI. Comments and suggestions on the concept of controlling tetrominoes with a 'brick' in an environment absent of a keyboard and mouse were sought. The techniques presented by Snyder [10] were used during the sessions.

The paper prototype used a Lego block to represent the controlling brick. Selection was performed by placing the brick directly above the desired tetromino. Moving the brick away from the same tetromino would deselect it. Once the tetromino is selected, the user could then translate it anywhere but upwards, as in the actual game. The rotation mechanism was designed such that once the brick has been rotated, the tetromino would keep rotating at a speed determined by the angle of rotation of the brick (i.e. the greater the angle made by the brick and its initial axis, the faster the tetromino would rotate, where 0 and 180 degrees represented the normal stopped state).

Some of the comments received after the tests were:

- *Magnet Metaphor*: The brick tool was compared to a magnet by some of the users. This showed us that the selection of tetrominoes was very well understood.
- *Rotation*: This aspect proved to be problematic. Users were not able to grasp quickly that the rotation angle of the brick controlled the tetromino rotation speed.
- *Collision Detection*: Since multiple tetrominoes will fall during the game, it is possible that two or more may collide before landing during gameplay. This issue was not addressed before.
- *Use of Two Bricks by a Single User*: Although not intended for this purpose, some users liked to use both hands to control two bricks simultaneously.

Preliminary Software Prototype

Many of the limitations of the paper prototype - such as animation and response time - were overcome by the software prototype. The latter was developed in Java 1.5 and the brick interface was implemented using a keyboard and a mouse. The functions provided to the user included tetromino selection, translation, and rotation in a "practice" mode setting. At this stage, specific feedback about the interface and the manipulation of tetrominoes, and not the actual gameplay was desired. Hence, the prototype asked the user to go through four tasks each exploring the different types of tetromino manipulation (see Figure 2). Later on, the user was given the chance to compare the original keyboard-only interface with the new design. Throughout the tasks, the prototype stored the completion times and number of restarts for quantitative analysis.

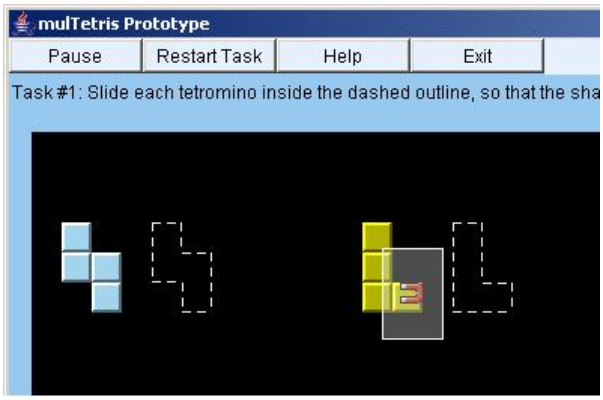

Figure 2. Translation of a tetromino

The rotation method was changed in this prototype. We decided that instead of using the increasing rotation speed concept, a transmission-like brick would be used: a rotation of 22.5 degrees ($\pi/8$ rads) will rotate a tetromino by 90 degrees in the same direction. Hence, only a 90 degree brick rotation is required to produce a full 360 degree tetromino rotation. Such a device would not exceed the human hand's articulatory limits, like a direct mapping of the brick's rotation to a tetromino's rotation would.

Implementation

- Input data from the mouse and keyboard was handled by the Java events manager. As mentioned earlier, the brick was simulated by a mouse as follows:
- Moving the mouse laterally while holding the SHIFT key simulated the brick rotation.
- Pressing the left mouse button was equivalent to bringing the brick down on top of the screen and selecting the tetromino below it.
- Releasing the left mouse button was equivalent to raising the brick away from the screen, and deselecting the tetromino previously selected; this caused the simulated brick to return to its default orientation, i.e. perfectly aligned on the vertical axis.

Evolution of the Software Prototype

The evaluating team appreciated the "easy-to-use" interface and its consistency with a typical Tetris game. Specifically, the selection process involving a mouse click on the block

was very well understood. Furthermore, the quantitative analysis showed improved performance regarding the rotation of a tetromino.

Alpha System

The previous prototype of *mulTetris* involved the keyboard and mouse. However, the alpha system was implemented using a visual tracking system, while coloured brick-like objects were used to interact with game. The setup consists of a laptop, with its screen laid completely horizontally, and a camera facing down perpendicularly to it (see Figure 3). As opposed to the previous prototypes which were entirely practice oriented, the user has the chance to play a complete tetris game, keeping score.

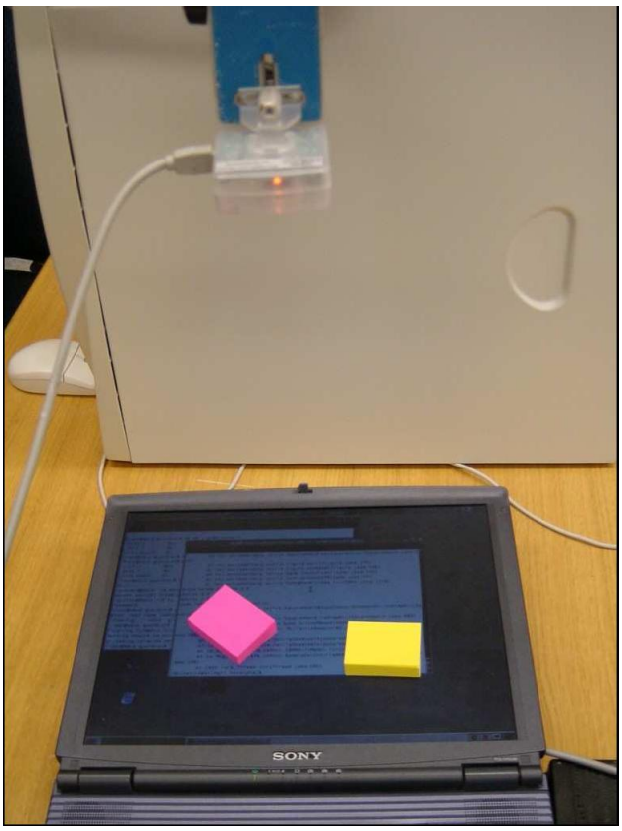

Figure 3. The hardware setup of the alpha system consisting of a laptop, camera and post-its.

Although still subject to improvements, the alpha system represented the concept of the final design. The limitations of the system were:

- The alpha system used solid coloured blocks of post-its to represent the controlling bricks. Post-its were chosen because of their colour intensity, thus making their tracking and detection easier. Also, before proceeding into using custom made translucent blocks, it was imperative to first ensure the functionality of the tracking system with readily available objects. The next release uses such blocks, so that the user can see through the block and receive visual feedback from the tetromino (i.e. selection, rotation...).
- The bricks could not be used to navigate around the system. Thus, to start/pause a game, to get help and to switch between game and practice mode, a conventional mouse was still needed. The beta system involves solely translucent blocks as described above.
- The installation of the system is rather clunky and inconvenient, due to the calibration process. The final system provides for a simpler installation.

Implementation

In the alpha system, with the real brick device, it was obviously impossible to make the brick go back to a default orientation, nor force the user to hold the brick orientated along the vertical axis when selecting a tetromino.

The implemented solution was to consider the vertical default axis to be the initial axis of the brick oriented by the user when selecting a tetromino. The tetromino would only be rotated if the angle of rotation is more than $\pi/8$ relative to the new axis. The user can select tetrominoes with the brick orientated in any direction then translate the selected piece without rotating it as long as the original orientation is kept.

Since the brick does not have any buttons, it was decided that selection and deselection would be detected by the height of the brick from the screen (i.e. the user has to raise the brick to deselect and bring it back on screen to select again). Two different approaches were considered:

- A second camera was placed beside the screen to record all the vertical variations of the brick: if the brick was raised above a specific height it was considered as an deselection; there was a lot of trouble if the user positioned his fingers on the side of the brick;
- The first camera could also be used to notice variations in the area of the brick: if the area is bigger it means that the brick is raised, closer to the camera, which is considered as an deselection; this was the solution adopted.

Input is handled with CV4HCI. All the data pertaining to the state of the bricks are constantly fed to the system through a handler method. The latter simply gives as parameters, arrays indicating the position, angle and change in the selection state for all the bricks.

Evaluation of the Alpha System

A second evaluation team tested the new system, keeping HCI principles in mind, and provided the following feedback:

- *Calibration*: As expected, it was suggested that providing an automated process would make the setup and running of the application easier.
- *Selection*: It was found that the user is prone to make mistakes when selecting and deselecting a tetromino with the brick. Having a LED attached to the brick that flashes, when a button is pressed, and is recognized by the camera, was suggested.
- *Bricks*: The team confirmed the need of translucent bricks to allow a clear view of the tetromino. It was also commented that they were too “square-shaped” and a more

elongated device would prevent tracking disruption caused by occlusion with the hand.
- *Interface*: Initially, the three preview boards in the alpha system were aligned vertically on the right. It was suggested to place them horizontally on top instead to provide a better visual mapping.
- *Rotation*: It was suggested to use $\pi/6$ increments instead of $\pi/8$ to allow for smoother rotation since it would require the user to rotate the brick slightly more to get the same effect.

Beta System

The next version addresses the issues discussed earlier. Installation and setup have been greatly improved, as calibration is now an interactive process which can be triggered from a menu item. The user is asked to move a brick to the four corners of the screen then center it when done. Built upon one of the suggestions given by the alpha system evaluators, the user will be given the option to change the default angle required to detect rotation.

Implementation

During the calibration, the software will take note of the following information:

- Minimal and maximal values for the x and y axes (will be used to map the coordinates from the camera's frame of reference to that of the computer screen);
- Dimension of the brick (will be used to select a tetromino with any part of the brick, see below for more details);
- Area of the brick located just on the top of the screen (variations in the area will be used to detect selection/deselection, see the implementation of the alpha system for more details).

In the alpha system, in order to select a tetromino, the user was required to bring the brick on screen such that the center of the brick was exactly over one of the tetromino's squares. This was inconvenient and unnatural, as a tetromino could not be selected by a smaller part of the brick (e.g. a corner).

In the beta system, due to the new calibration process, the system is aware of the dimension of the brick. As such, it is now possible to know whether a part of the brick is overlapping a part of a tetromino.

However, if the user unintentionally brings a small part of the brick over a small part of a tetromino, it would probably be not expected for the tetromino to be selected. To solve this issue, the selection will only be effective if a pre-defined minimal area of the tetromino is covered by the brick. Similarly, if the brick overlaps two different tetrominoes, the tetromino covered by the largest area of the brick will be the one selected (provided regular selection requirements are satisfied). To further prevent accidental selections, the brick will be required to remain over the desired tetromino for a preset time (e.g. 200 ms). This is preferred over the earlier suggestion of adding a LED and a button to the brick, as it minimizes the actions needed to be made by the user, thus lessening the cognitive load.

Furthermore, it was decided to keep the preview boards in their original position since due to the laptop's screen size, it was desired to have maximal height for the game board. Width was an acceptable trade off, despite that having the preview boards vertically aligned on the side does not present a very direct mapping.

Finally, we have agreed with the evaluators of the alpha system that a more elongated brick is desirable. The bricks of the beta system are translucent and approximately 2 by 1 inches and an inch thick.

CAMSHIFT Algorithm

Algorithm Description

The CAMSHIFT (Continuously Adaptive Mean Shift) algorithm was designed to track coloured objects (in our case, the bricks). One of the main advantages of this technique is that it can efficiently track an object in real time. More complex methods give better results, but have a hard time running in real time on today's personal computers. This method was also designed to run adequately on cheap video cameras (i.e.: Webcams). In this section, we will described in some details the way it works.

First, the tracker needs to be "trained". The model of the object that is kept in memory is a simple histogram representing the distribution of the colours of the object. As colour information, typically and as used by CV4HCI, the H channel (hue) of the HSV colour space is used. As "training set", an area of the object can be selected by the user as described in the next section. The histogram is scaled so that the maximum fixed-point value is 255 (i.e.: "probability" = 1.0). This does not make for a real probability mass function, but is used as such. Typically, 16 bins are used. Models with larger numbers of bins have a greater hue discrimination power, but might not work well enough under varying lighting conditions, since the hue changes even when just the angle of the incident light on the object changes.

Then, after training, in order to track the object, a "colour probability distribution image" needs to be computed. The process starts by converting the whole camera image into the HSV colour space. For each pixel, the hue value is used as an index into the histogram bins. The corresponding histogram bin value is used as the "probability" of the pixel being of the colour of the tracked object. This bin value is stored into the "colour probability distribution image".

This colour probability distribution image is then used with the CAMSHIFT algorithm in order to track the coloured object. An initial window size (either the whole image or from the user selection as described in the next section) is used as a starting point for the CAMSHIFT algorithm. This iterative algorithm alternates between two sub-algorithms until convergence (until the window movement and adaptation is negligible): the mean shift algorithm, and the window size adaptation. The mean shift algorithm is also iterative. For each iteration until convergence (when the window shift is minimal), the mean of the pixels (centroid or center of gravity) in the window is located, and the window is shifted (moved) to this location. For the window size adaptation, the window size is set to a function of the zeroth moment of the pixels in the window (which is related to their area).

The rest of the image moments of the pixels in the window are then computed to find the orientation and the axis lengths. The recovered tracking information (window position, axis lengths, and orientation) can be quite noisy, so it is then smoothed using a standard maneuvering time-correlated velocity Kalman filter (a simpler version of the Singer acceleration model [11]). The filtered data is then sent to *mulTetris* as input. Also we want our program to select a tetromino when the user places the brick on it. With the information available, the best option is to use the two axis lengths as an indication of the "height" of the brick with regards to the camera. However, if the user does not hold the brick flat and unobstructed under the camera, this procedure breaks down.

CAMSHIFT Setup with CV4HCI

In order to train and setup the parameters for CAMSHIFT, CV4HCI comes with a view panel written in Java as seen in Figure 4. Two separate trackers are used to track two differently coloured bricks. More tracker can be used depending on available processing power. One can switch the view from Camera, BackProject (the colour distribution image) or Histogram. To train a tracker, one can simply select a region from the Camera View. An ellipse then shows the currently tracked object and its information (position, axis lengths and orientation). As for the parameters, Vmin indicates the minimum value of V from the HSV colour space a pixel has to have to be considered. Ignoring those pixels can help since the hue value becomes a lot noisier for low V values. The same concept applies to Vmax (maximum V value), and Smin (minimum S value), which helps CAMSHIFT ignore washed out or pastel colours respectively. Min Area, the minimum zeroth moment (i.e.: summation of the back-project pixels in the window), is used to detect when a track is "lost". If this value is not reached, the window size is reset to the image size.

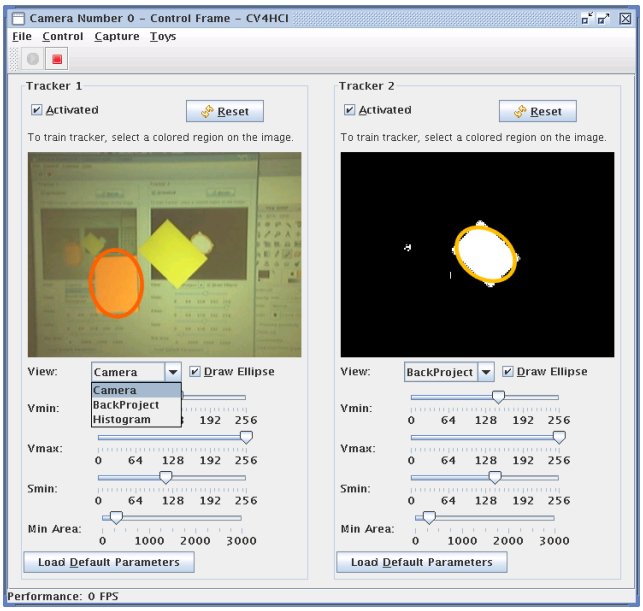

Figure 4: CAMSHIFT setup with CV4HCI

As can be understood from this brief discussion of the required parameter setup, it can be quite involved. For this reason, parameters and training information (the histogram) can be saved to file and reloaded. If however the environment happens to change (lighting conditions, non-constant camera settings), the setup has to be redone. This is the current main limitation to the wide acceptance by the public of such an interface.

CONCLUDING REMARKS AND FUTURE WORK

By using an iterative design process as well as frequent usability testing, *mulTetris* has been developed as an easy-to-use game, making use of the user's already existing skills. Taking advantage of prehensile behaviour and kinesthetic feedback, the interface reduces the cognitive load, as the user does not need to concentrate on how to move the physical bricks, and can use muscle memory to infer their location.

The concept of the controlling brick could not be fully developed due to technical limitations of the system. Due to the small setup, small bricks had to be used and users often gripped them completely, thus covering a large portion with their hands. As a result, the bricks were not visible to the camera, and their position could not be tracked. Future work could involve a pressure sensitive screen, where the tracking of the brick would be done by a set of touch sensors. This would be less intrusive, as the user could handle or grip the brick in any way he/she feels more comfortable, without affecting the usability of the system.

Furthermore, the camera is currently used over a flat screen, but to avoid tracking problems that appear when the user hides the brick from the camera, as well as to improve the selection mechanism, a hardware setup using video back-projection and a camera looking from behind, such as in Ullmer's metaDESK [12], should be evaluated.

With a better tracking mechanism, *mulTetris* can be used to evaluate the importance of Graspable User Interfaces in the context of collaborative games.

REFERENCES

1. Ullmer, B., Ishii, H., Emerging frameworks for tangible user interfaces, IBM Systems Jour-nal, 2001 http://www.research.ibm.com/journal/sj/393/part3/ullmer.html

2. M. Fjeld, M. Bichsel, and M. Rauterberg, BUILD-IT: An Intuitive Design Tool Based on Direct Object Manipulation, Gesture and Sign Language in Human-Computer Interaction, Lecture Notes in Artificial Intelligence, Vol. 1371, Wachsmut and Frhlich, Editors, Springer-Verlag, Berlin (1998), pp. 297308 http://www.idemployee.id.tue.nl/g.w.m.rauterberg/publications/GW98paper.pdf

3. Fitzmaurice, G., Ishii, H., Buxton, W. ,Bricks: Laying the Foundations for Graspable User Interfaces, Proceedings of ACM CHI 1995 Conference on Human Factors in Computing Systems, p. 442-449.

4. OpenCV. Intel Corporation. 2005. http://www.intel.com/technology/computing/opencv/

5. Krueger, M., Artificial Reality II. Addison-Wesley, New York, 1991.

6. MacKenzie C, Iberall T: The Grasping Hand. Elsevier, 1994.

7. Bradski, G.R., Computer vision face tracking as a component of a perceptual user interface. In Workshop on Applications of Computer Vision, p. 214-219, Princeton, NJ, Oct. 1998.

8. Gould, J.D., Boies, S.J., Levy S., Richards, J.T. and Schoonard, J. The 1984 Olympic Message System: A Test of Behavioral Principles of System Design. Comm. of the ACM. 30, 9 (Sept. 1987), 758-769.

9. Nielsen, J. Heuristics for User Interface Design. http://www.useit.com/papers/heuristic/heuristic_list.html, 1994.

10. Snyder, C. Paper Prototyping, December 2003. http://www-128.ibm.com/developerworks/library/us-paper/?dwzone=usability

11. Li, X.R, Jilkov, V.P. Survey of Maneuvering Target Tracking. Part I: Dynamic Models. IEEE Transactions on Aerospace and Electronic Systems, 39(4):1333–1364, Oct. 2003.

12. Ullmer, B., Ishii, H., The metaDESK: Models and Prototypes for Tangible User Interfaces, Proceedings of the ACM UIST'97 Symposium on User Interface Software and Technology, 1997, p. 223-232.