Raptor: Sketching Games with a Tabletop Computer

J. David Smith and T. C. Nicholas Graham
School of Computing
Queen's University
Kingston, Ontario, Canada
{smith, graham} @cs.queensu.ca

ABSTRACT

Game sketching is used to identify enjoyable designs for digital games without the expense of fully implementing them. We present *Raptor*, a novel tool for sketching games. *Raptor* shows how tabletop interaction can effectively support the ideation phase of game design by enabling collaboration in the design and testing process. *Raptor* heavily relies on simple gesture-based interaction, mixed-reality interaction involving physical props and digital artifacts, Wizard-of-Oz demonstration gameplay sketching, and fluid change of roles between designer and tester. An evaluation of *Raptor* using seven groups of three people showed that a sketching tool based on a tabletop computer indeed supports collaborative game sketching better than a more traditional PC-based tool.

Keywords

End-User Development, Video Games, Sketching, Tabletop, Computer-Supported Collaborative Work, Game Sketching

Categories and Subject Descriptors

H.5.3 [**Group and Organization Interfaces**]: Computer-supported Cooperative Work.

1. INTRODUCTION

Sketching is an emerging concept in user interface development that supports the early ideation phase of interaction design [4]. Collaboration and iterative design has been shown to be beneficial when building interactive systems [6], and sketching is an important component of the design process [12]. Sketches are not intended to provide a reusable piece of software. Instead, they help explore user experience without dwelling on the details of implementation. Sketching is an inherently collaborative process, where designers, end-users, and developers can all work together to develop and communicate design concepts.

We present *Raptor*, a novel sketching tool for video games. *Raptor* addresses the problem of trying to identify entertaining gameplay without having to build a working prototype. Games now cost tens of millions of dollars to build and involve teams often in excess of 100 people [9]. Given these costs, it has become critically important to assess early in the development process whether the game will actually be fun to play. Sketching allows players to experience game ideas without requiring fully functioning prototypes. Rather than providing a full game development tool, the scope of *Raptor* is limited to the early design process where sketching is most beneficial. The tool is an improvement over existing game sketching tools [1] because it supports modern, tabletop-based interaction techniques that enable rapid development by people without extensive technical backgrounds.

Figure 1: *Raptor* is a game sketching tool based on a tabletop surface. The tool helps groups of users to brainstorm early in the game design process.

The primary design goal of *Raptor* (figure 1) was to create a tool that encourages rapid ideation. We demonstrate that a tabletop interface improves over a desktop interface [1] by providing:

- *Tool transparency*, where sketches can be created using gestures and physical props rather than pointing and clicking;
- *Engaging collocated collaboration*, where the physical layout of a table allows intimate communication, and where the table's multi-touch input allows more than one person to interact with the sketch at a time;
- *Egalitarian design process*, where non-programmers are not disadvantaged, and where roles can be fluidly set and changed.

ACM FuturePlay 2010 May 6-10, 2010, Vancouver, Canada

The paper is organized as follows. We first introduce the concept of game sketching and motivate why it is a helpful part of the game development process. We then present the design of *Raptor*, and illustrate its features via two usage scenarios. We then report on an experiment comparing the effectiveness of *Raptor* to a similar tool based on a desktop PC. The paper concludes with a discussion of what we have learned from our experiences with Raptor.

2. SKETCHING USER INTERFACES

The benefits of sketching in design activities have been widely investigated [4, 20]. As opposed to reusable prototype implementations, user interface sketches allow users to experience how the system will work. They focus more on content and structural aspects than on look-and-feel and visual details of the final product. Sketches often rely on Wizard-of-Oz techniques [7] where the program's behaviour is simulated by acting it out at runtime rather than completely programmed. This allows much faster iteration than is possible with programmed prototypes.

Tool support for interactive sketching has received increasing attention in the HCI community [12]. Interaction with digital sketches has been shown to provide benefits to the user interface design process over paper prototypes [17]: designers are able to go through more iterations with digital sketches than with paper, because digital sketch components can be more easily reused.

2.1 Sketching Video Games

Existing industry processes rely on playable standalone game prototypes, often based on previous games [19]. Producing these prototypes requires a significant programming effort, which limits the degree to which non-technical people such as end-users can participate in the early design process. Additionally, working at the level of code causes the designer to focus on implementation details early in the process, which can inhibit creativity. And creating coded prototypes costs time and developer resources, limiting the number of tests that can be performed.

Sketching is an ideal way of helping with this problem. Sketches permit very rapid, low-fidelity testing of game ideas. Because sketching is fast and inexpensive, there is little resistance to throwing ideas away that don't appear to be working. The most promising ideas generated from game sketching can then be carried forward for higher-fidelity prototyping.

Recent work has tried to apply sketching concepts to video game design. Agustin et al. introduced the term "game sketching" and describe a tool for developing sketches of linear narrative games [1]. This tool provides a simple scripting language for describing interactive 3D scenes, and behaviors are implemented through "Wizard-of-Oz" control of non-playable characters. Wizards connect to the game sketch via the local area network and control behaviors through an interface resembling common control mechanisms of a first-person shooter game. The tool was designed to include non-programmers and testers throughout the design process. The tool reduces the focus of design sessions on implementation details and operation of complex tools such as game engines and programming environments. However, collaboration on design issues beyond playtests is done on a single PC with the design team sharing a single desk.

Our own *Raptor* tool carries this work further, examining how tabletop interaction can support game sketching better than traditional point-and-click desktop interfaces.

3. TABLETOP COMPUTERS

Raptor's tabletop interface enables natural interaction based on familiar gestures, making it accessible to non-technical users. The physical layout of a tabletop surface also permits more fluid and direct same-place collaboration than available when using the mouse and keyboard of a traditional PC.

Tabletop computers consist primarily of a large touch-screen display that covers the surface of a table, around which multiple users can work at the same time (figure 1). They differ from desktop computers by providing a gesture-based user interface that more closely mimics the way people interact with everyday physical objects. The technology used to implement tabletop computer systems has recently become inexpensive enough to create commercial tabletop computers [13, 22], and a need exists for identification and understanding of potential application domains.

An advantage of tabletop computers is support for collaboration among groups of co-located users [21]. A group can sit around a table and work together face-to-face on a task rather than taking turns using a desktop computer. Tabletops typically offer a touch-based interface that allows users to interact with digital media using simple gestures. Tabletop applications benefit from these interface components because they provide tool transparency and afford use by non-technical people.

4. RAPTOR

When designing *Raptor*, our principle goal was to demonstrate how a tabletop computer could support collaborative game sketching. Specifically, *Raptor* demonstrates that people find sketching using gestures and physical props to be easier and more fun than traditional desktop-based tools, that collaboration is better supported by the tabletop's horizontal layout, and that *Raptor*'s natural interface is accessible by non-programmers.

With *Raptor*, designers use simple gestures to create and manipulate interactive virtual worlds. Gestures are also used to connect game controllers to virtual objects in the game world with a mixed-reality "interaction graph" [10]. *Raptor* also provides support for "Wizard of Oz" [7] style prototyping, allowing designers to rapidly prototype and test new game ideas.

4.1 Creating virtual worlds

To begin creating a game prototype, users create a virtual world. Designers are first presented with a flat, bare terrain on the table surface. The terrain can be navigated using panning and zooming gestures commonly found in multi-touch map applications, such as found on the iPhone [2]. To contour the terrain, designers use gestures that mimic manipulating sand in a sandbox. To create a hill, designers use a "scooping" gesture (figure 2). Similarly, to create a valley, a "spreading" gesture is used.

To add objects to the scene, designers perform a "stamping" gesture with a physical object, somewhat similarly to *Build-It* [18]. For example, figure 3 shows the addition of a car into a racing game by "stamping" a physical car onto the tabletop. The virtual car is added to the scene at the location where the physical car is stamped. To remove objects, a rubber eraser is used. Players can also drag items around in the scene with their fingers.

This use of common gestures to manipulate the virtual world simplifies interaction over the pointing, clicking and dragging required by a traditional mouse and keyboard-based interface.

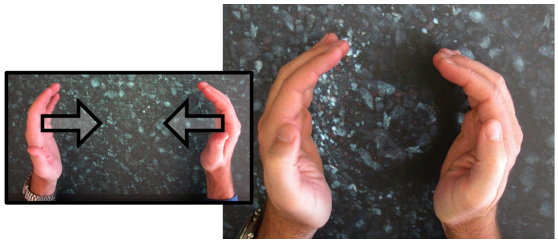

Figure 2: Designers can manipulate the virtual terrain with physical gestures, e.g., using a scooping motion to create a hill.

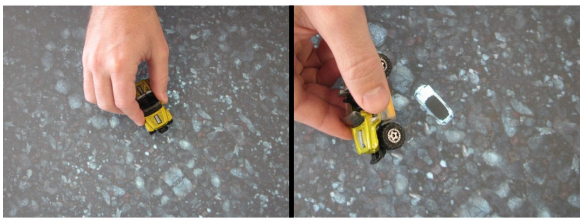

Figure 3: Stamping a physical car on the table adds a virtual car into the game.

4.2 Adding Interactivity

Console games are normally played using a game controller, a special-purpose hand-held input device providing buttons and joysticks for manipulating play in a virtual world. The responsiveness and intuitiveness of input mechanisms is crucial to the success of games [16]. *Raptor* allows designers to easily prototype game input via mixed-reality interaction with the tabletop. Figure 4 shows how a designer places an Xbox controller input device onto the table. A ring of "pins" surrounds the controller, representing the different input and output channels the controller provides (e.g., the different buttons and joysticks.) As the controller is moved around the tabletop surface, the ring of pins moves with it, creating a truly mixed physical-virtual entity. The pins on the controller can be connected to pins on other objects; the designer attaches the "A" and "B" buttons to the car's gas pedal and brake pins, allowing these buttons to be used to accelerate and decelerate. To connect two pins, the designer touches the source pin, and then touches the target pin. A line is drawn between them to show the connection. The designer can pick up the controller to try out the interaction immediately, and then return it to the tabletop to refine input/output connections.

The primitive behaviour of interactive objects, like the car, must be programmed in advance. Pins are created by annotating attributes of the classes implementing these objects.

4.3 Controlling Network Displays

To view the game world from an angle other than the table's top-down view, designers can connect a display via the local area network. The viewing position and angle is controlled by placing a small plastic camera on the table surface. To move the camera position, the designer simply positions the camera on the the table surface and rotates it to the desired angle. Designers can also tap an interactive object with the plastic camera to create a "chase" view. When a camera is in "chase" mode, it will follow closely behind the interactive object as it moves about the game world.

4.4 "Wizard-of-Oz" Prototyping

One of *Raptor*'s most powerful aspects in supporting game sketching is its support for a "Wizard of Oz" style of play testing. Rather than having to program new game ideas, designers can act them out, using the tabletop display to rapidly manipulate the game while testers play the game while sitting at a traditional display. This supports fluid and easy collaboration, allowing designers to rapidly move between between design and playing roles, and supporting a very rapid iterative design cycle.

Figure 5 shows a tester sitting at a PC playing a game as the designers modify it. Here, the designers are creating a racing game. Testers and designers have different viewpoints on the game. Designers have a top-down, two-dimensional view of the game world, while testers see the game in a more traditional 3D form (figure 5).

The tester immediately sees changes such as adding a new car to the game, repositioning an obstacle or modifying the terrain. Similarly, changes in *gameplay* are reflected immediately in the tester's view of the game: for example, the controls used to manipulate a car can be changed on the fly. Designers simulate game play by manipulating objects on the table, either via gestures or manipulation of physical props. This approach allows testers to experience a game idea without the expense of fully implementing it.

4.5 Benefits and Limitations

As will be discussed in our evaluation section, *Raptor* provides strong support for collaborative game sketching. Designers – including programmers and non-programmers – are indeed able to rapidly demonstrate gameplay; it is easy to fluidly transition between design and testing roles, and same-place collaboration is effectively supported. *Raptor* nevertheless has limitations.

One notable shortcoming is that the tabletop interface is useful only when a set of interactive objects exists that is suitable for desired type of game. For example, a car racing game can be prototyped on the table only after someone has prepared a "Car" interactive object. To create an interactive object, a programmer must create a C# library with an accompanying directory of 3D models and textures and a configuration file tying them together. Our experience to date leads us to believe that a relatively small number of such interactive objects are necessary to sketch a wide range of games. An interesting approach would be to create an online database of object libraries that designers could use to share interactive objects in a manner similar to the Google 3D Warehouse [8].

Additionally, the tabletop provides designers with only a top-down view. While we have successfully applied this approach to genres as diverse as racing games, role playing games and first-person shooter games, this limitation precludes games with large amounts of vertical motion or occlusion from above. For example, it would be difficult to create a sketch of Nintendo's Super Mario Galaxy [15] because the game world consists primarily of movement around a series of three-dimensional spheres.

4.6 Implementation

Raptor is implemented in C# with Microsoft XNA Game Studio [14]. For the tabletop computer, we used Microsoft Surface [13], a commercially available tabletop computer that supports multi-touch sensitivity, object recognition through spatial tags called "dominos", and high-quality 3D graphics processing.

The system is built around a central scene graph data structure.

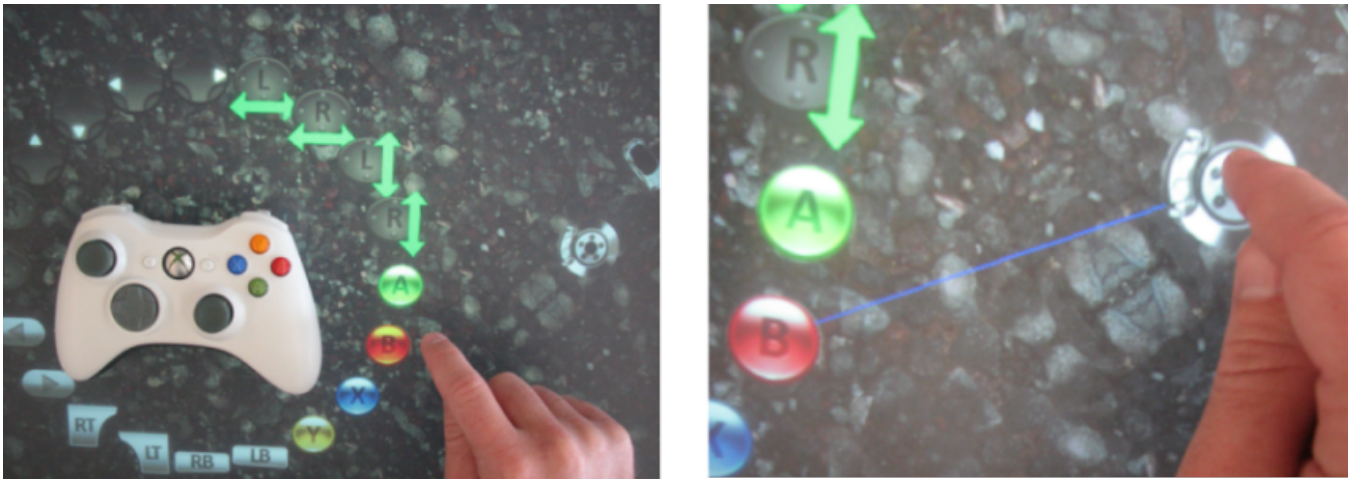

Figure 4: When an XBox 360 controller is placed on the table, a ring of "pins" surrounds the controller showing its various outputs. These pins can be connected to inputs on interactive objects.

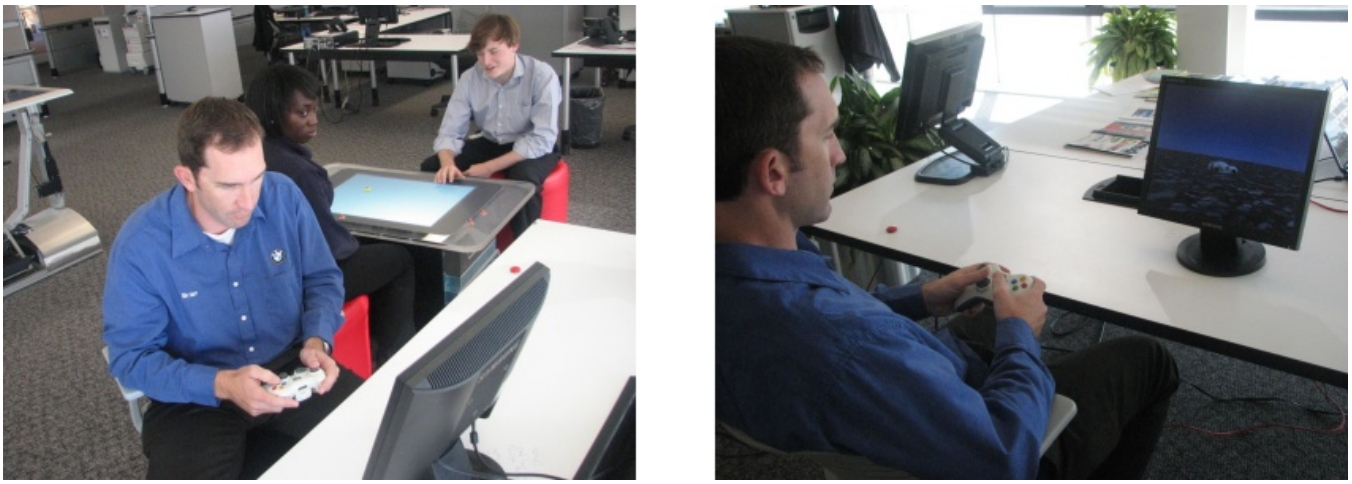

Figure 5: Designers and testers collaborate using *Raptor*. A tester can play a game as it is being created, creating a fluid testing process and supporting Wizard of Oz prototyping.

Stamping a new object into the game adds it to the scene graph; moving an object changes its properties in the scene graph. Adding a camera object determines the viewport seen by testers. For rendering speed, the scene graph is replicated to all tester computers, and updates are propagated at runtime.

Interactive objects are built into XNA Game Libraries which are registered with *Raptor* via a configuration file. At runtime, all registered assemblies are examined for objects extending a known base type and marked with an "InteractiveObject" attribute. To mark a particular object member as a "pin", the programmer annotates the member with a "Pin" attribute.

5. USAGE SCENARIOS

To illustrate *Raptor*, we have created two scenarios. These scenarios synthesize our observations of how people use *Raptor* to sketch games, particularly illustrating collaborative development and "Wizard of Oz" prototyping. The scenarios also show that *Raptor* can be used to sketch a diverse range of game styles, including fantasy role playing games and multiplayer shooter games, as well as the racing game described in section 6. All screenshots in this section show the results of enacting these scenarios with *Raptor*.

5.1 Building a Role-Playing Game

A fundamental element of many role-playing games is the game narrative. In this scenario, a team of designers wishes to experiment with simple story ideas. They are creating a fantasy game including knights, dragons, monsters, and other common medieval story elements.

The team decides to work on a scene in a forest. In this particular scene, the player must collect several hidden items. After all the items have been collected, a large impediment will be removed from the player's path so (s)he can progress to the next scene.

The team begins by bringing out a box of plastic objects resembling entities in the game. They use a series of "scooping" and "spreading" gestures to contour the terrain and to create a smooth path through the middle. To create the forest, the team uses a small plastic tree in a "stamping" gesture. The team then hides the vari-

Figure 6: A role-playing game sketched with *Raptor*.

ous items throughout the forest by stamping the objects throughout the scene. Similarly, they place a large boulder at the end of the path to inhibit progression to the next scene.

To add the playable character, a designer stamps a small plastic knight. To make the character playable, she place(s) an Xbox 360 controller on the table. A ring of "pins" appears around the controller. The designer then taps the character and a ring of "pins" appears around the character, including "walk", "run", and "attack" pins. The designer then connects character pins to the desired controller pins by tapping on the icons. The team wishes to view the game with a chase camera on a nearby television, so they tap the character with a small plastic camera. Immediately, a view of the world from the perspective of the character is shown on the television.

5.1.1 "Wizard-of-Oz" Behaviors

One of the designers then carries the XBox controller to the television and assumes the role of "player". The remainder of the team stays by the table to act as wizards. As the "player" walks through the environment, (s)he finds the various items hidden in the forest. When the player bumps into each hidden item, the wizards use the eraser to remove the items from the scene, indicating that the player has collected them. Once all the items have been collected, a designer drags the large boulder from the path so the player can continue to the next scene.

This scenario shows the ease of collaboratively creating a new scene and animating it through Wizard-of-Oz techniques. The scenario also shows the limitations of the sketching technique. The sketch does not involve non-player characters with detailed dialogue trees, or combat with spectacularly animated special effects. When the design has advanced to the point where such details need to be pinned down, designers should consider moving to more traditional prototyping techniques.

5.2 Creating a First-Person Shooter

Our next scenario involves sketching elements of a first-person shooter (FPS). FPS games, such as the Halo series [3], often place a heavy emphasis on environment design and realistic physical simulation. They commonly are comprised of a series of "levels" that consist of an interactive 3D environment and convey a part of the game's narrative. Additionally, some first-person shooters include a multi-player component, where several players occupy the same specialized level. Multi-player FPS games include "Deathmatch", where players score points by shooting other players, and "Capture the Flag", where players attempt to penetrate the opposing team's base, collect a flag, and return it to their own base. This scenario examines how *Raptor* can be used to design the environment of a multi-player first-person shooter and explore its gameplay rules.

5.2.1 Creating the Environment

A team of designers building a new World-War-II style first-person shooter gather around the table. They decide to create a realistic urban battlefield experience for multiple players. To begin, they load a rocky concrete terrain onto the table surface and contour it to include several craters, giving the terrain a battle-worn look. To add interactive content, the designers open a box of plastic toy soldiers, buildings, and various weapons. They begin by stamping several buildings into the scene to create an urban environment. They then stamp weapons and power-ups into the environment. The team then adds in two avatars for the playable characters. Each character has several input pins, including "walk", "side-step", "fire", and "special weapon". To make the characters playable, the designers place the controllers on the table and connect them to the avatars. To create a playable view on the network display, the designers touch one avatar with a small plastic camera. Immediately, a designer can assume the role of that "player" by carrying the controller to the network display, and can begin to explore the environment.

5.2.2 "Wizard-of-Oz" Multi-player Game Rules

To create a multi-player experience, a second designer connects a controller and display to the other avatar and walks to a network display. The designers wish to prototype a simple "capture the flag" game in the new environment. The game "wizards" at the table place "flags" at opposite ends of the environment and move the avatars into their starting positions. When the game begins, the wizards keep a careful watch on the table to keep track of how many times each player is shot by the other. They decide that a total of three shots constitutes a "kill". To prototype picking up the flags, when a wizard sees a character pass over a flag, (s)he erases it from the environment to create the appearance of the flag being picked up. The designers determine that when players are killed, they immediately drop any flags they are carrying and return to their starting position. To enforce this rule, when the wizards count that a player has been shot three times, they use their hands to place the dropped flag at the location of the kill and then drag the avatar back

Figure 7: A first-person shooter game sketched with *Raptor*.

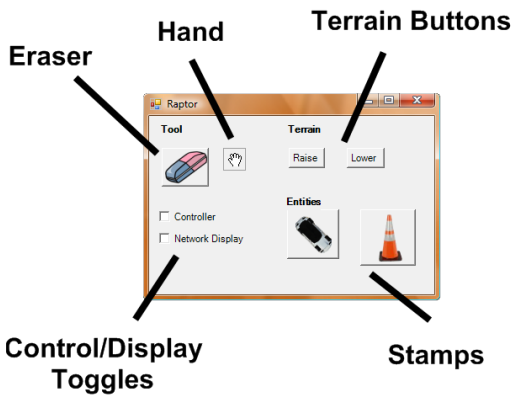

Figure 8: The desktop condition provided a simple point-and-click interface to the same functional components used in the tabletop version of *Raptor*.

to the starting position. When a player brings the opposing teams' flag back to its starting position, the player is deemed to have won the game.

This scenario shows that complex game rules can be "acted out" using Wizard of Oz techniques, and that *Raptor* can be used to simulate interesting multi-player gameplay. The tabletop surface is crucial to being able to do such scenarios, as the use of gestures and physical props allows designers to rapidly manipulate the scene.

6. EVALUATION

We performed an empirical study to determine whether the physical, gesture-based interaction afforded by *Raptor* could indeed be used by non-programmers to sketch a video game prototype, and whether using a tabletop computer indeed provides a superior collaborative experience to a desktop PC. The study does not examine the space of games that can be prototyped with *Raptor*, or compare *Raptor* to existing game sketching toolsets.

6.1 Overview

The evaluation consisted of groups of end-users creating a game sketch with *Raptor* in both a tabletop and desktop format. We did not intend to compare *Raptor* to desktop-based sketching tools in general, but rather to specifically examine the effect the tabletop computer had on the sketching experience.

Participants were asked to to create a sketch of an off-road racing game. We chose this style of game because its rules are simple and commonly known, and because it makes effective use of all of *Raptor*'s features. This allowed participants to create enough of a sketch to form an opinion on both the tabletop and desktop versions. Participants answered a series of questions designed to determine which system they preferred and were were asked to provide general feedback in a discussion format.

6.2 Conditions and Apparatus

Participants interacted with the tabletop version of *Raptor* as well as with a comparable desktop interface (figure 8). The desktop interface was functionally equivalent to the tabletop version of *Raptor*, but used a mouse instead of multi-touch to interact with the game world. In the desktop version, users maniplate the game by choosing a tool from a palette and clicking on the scene. Additionally, the physical objects used in the tabletop interface (stamps, controllers, etc.) were replaced with pictures of those objects. For example, a toggle was used to show/hide a picture of a controller on the scene.

6.3 Provided Content

The tabletop and desktop conditions used the same collection of interactive objects. The objects were sufficient to create a simple sketch of a 3D off-road racing game:

- *A Car* provided an interactive vehicle that could be driven on the terrain. The car included three pins: gas, brake, and steering. The car's behavior was modeled with a physics simulator to resemble an off-road buggy.
- *A Parking Cone* provided a simple marker used to lay out a desired track on the terrain. The cone had no input pins.

The objects were placed on a rocky terrain to give the feeling of an off-road environment. In the remote display, a textured sky was also provided to provide a more realistic look.

6.4 Procedure

Participants were placed in groups of three. All subject groups performed both the tabletop and desktop conditions, rotated on a Latin square to control for learning effects. For each condition, the groups were given a tutorial on how to operate the interface. The training session for each apparatus began with the experimenter demonstrating usage of the system and ended when each participant independently indicated that (s)he understood the system's operation and was able to demonstrate use of each feature.

Trials were untimed. Users were asked to create a sketch of an off-road racing game with the following properties:

- Players should traverse an off-road track as quickly as possible.
- Each lap must contain an element not present in the previous lap. For example, the game designers might introduce new obstacles into a specific area while the player was driving on a different portion of the track. This was done to encourage subjects to use the "Wizard of Oz" technique to sketch an in-game behavior.
- The terrain must contain multiple hills and valleys.

A trial ended when every participant had both performed the role of designer on the table/desktop and player with the remote display. Trials lasted between 10 and 20 minutes. At the end of each trial, each participant individually completed a questionnaire (table 1). When the participants had completed both trials, they filled out another questionnaire that asked them to directly compare the two conditions (figure 2). Finally, they were given a series of discussion questions and asked to reply to those to which they wished to make a comment.

6.5 Participants

21 people participated in the study, and were randomly divided into seven groups of three. The participant group consisted of 14 males and 7 females ranging in age from 21 to 61 (average age of 36). 33% of participants had used a tabletop computer before the study.

6.6 Collected Data

Data was collected using a series of Likert items and discussion questions. The Likert items and their responses are shown in table 1. Also shown are the results of a Kruskal-Wallis [11] test comparing the two conditions and Cohen's D value [5] indicating the effect size. Participants were also asked to directly compare the conditions on a variety of criteria. For each question, participants were asked to circle either "tabletop", "desktop", or "same". The questions and their response totals are shown in table 2.

6.7 Summary of Results

For the Likert items, the results show significance for 5 questions. The strongest results were seen in the two Likert items dealing with collaboration with partners, indicating that the tabletop interface does in fact support local, synchronous collaboration better than the comparable desktop interface. Additionally, the results show that participants found it easier to prototype game rules and produce a more fun prototype with the table.

When asked to choose which condition they preferred, participants showed a strong preference for the tabletop condition. The results show that participants felt the table was easier to use, more fun, supported collaboration better, and generally preferred to use it. Interestingly, participants felt neither the tabletop or desktop condition produced a better prototype, which conflicts with the result from the Likert item comparison indicating the tabletop produced a more fun prototype.

6.8 Analysis

The results of the empirical study support our hypothesis that the novel physical, gesture-based interface built into *Raptor* is useful for game sketching. In both cases, users were able to build a playable game prototype. Given that participants were not at all familiar with the interface before participating in the study and had a variety of experience programming computers, we feel that the design of the system proved simple enough to be operated effectively by end-users. Given this result, we feel a particularly interesting area for future work would be in educational environments where *Raptor* could be used as a creative educational tool.

Additionally, the results clearly indicate a strong preference for the tabletop version of *Raptor*. While this supports our hypothesis that the tabletop computer would better support collaboration among co-located groups, the results do not provide any indication about other forms of collaboration or, in particular, single-user operation.

Table 2: Participants were asked to circle which condition they felt best answered each question.

Question	Table	Desktop	Same
Was Easier to Use?	15	3	3
Was More Fun to Use?	15	3	3
Produced a Better Prototype?	6	1	14
Supported Collaboration Better?	14	1	6
Did You Prefer to Use?	16	1	4

However, our finding that the tabletop interface was more fun to use supports the hypothesis that designers would prefer to use *Raptor*'s physical stamping gestures and touch-based interface even when working alone.

6.8.1 Tabletop vs. Desktop Computers

While our results suggest the tabletop version was an improvement over the desktop version of the *Raptor* interface, the study is not sufficient to make any claim about tabletop versus desktop computers in general. Still, our study highlights some key differences. The tabletop computer provided a more collaborative experience where multiple users could interact with the tool simultaneously rather than following an explicit turn-taking mechanism as with the desktop computer. One participant commented that with the tabletop, *"instead of having observers and one manipulator interacting with the environment, multiple users could stay active and interested at once."* Additionally, participants appeared to enjoy sitting in a circle around the table rather than crowding together at a single desk and facing the same direction towards a monitor. This made for a more social experience with more conversation.

6.8.2 Noted Shortcomings

Participants regularly noted a frustrating aspect of the tabletop tool, where one designer would interfere with another by manipulating the tabletop's shared view of the world. For example, a designer would stamp an interactive object in an unintended place because, just before the stamp touched the table, another designer panned/zoomed the table's view of the world. Additionally, when designers worked close together on the table surface, the gesture recognition system could malfunction. For example, two designers dragging interactive objects towards each other could be interpreted as a "zoom out". These observations illustrate a common problem with tabletop computers, where application designers must carefully manage public and private spaces on the table surface.

7. DISCUSSION

In out experience with *Raptor*, we have been surprised at how much functionality can be sketched with a relatively limited amount of in-game objects. Still, we have encountered situations in almost all of our sketches where programming would be required to complete a game idea. For example, when building a first-person shooter, some designers wished to add playable functionality to their in-game characters, such as a "reload" action. However our FPS character object did not support this, so adding it would require the FPS character's behavior to be modified. In practice, this leads us to iteratively move between programming and sketching phases. A sketching phase lasts until it can no longer proceed without more complex behaviors programmed into the in-game objects. A programming phase then follows that accepts the findings of the sketching phase as a set of requirements. When the necessary functionality has been added, the design group again meets around the table and continues sketching. We believe this iterative approach that includes both design and development phases to be beneficial, and a particularly interesting area of future work will examine how this approach could be used in a professional game development context.

Also, this iterative process suggests a need for end-user programming features. For example, it could be helpful to create simple character behaviors by expressing them through programming-by-example on the table surface. A user could express a path-following behavior by dragging a character across the table with his/her hands. This approach could eliminate the need to switch

Table 1: Comparison of means of responses to Likert items

Question	Mean		Kruskal-Wallis (p)	Cohen's D
	Tabletop	Desktop		
I was able to build a playable game prototype.	4.52	3.95	0.003	0.96
The game I designed was fun.	4.33	3.52	0.001	1.09
I consider my design experience a success.	4.62	4.10	0.034	0.79
The environment was easy to use.	4.38	3.90	0.067	0.69
Creating my 3D scene was easy.	3.95	3.52	0.113	0.49
Adding input to my game was easy.	4.62	4.19	0.083	0.65
Prototyping game rules was easy.	4.00	2.71	0.001	1.53
I enjoyed building the game.	4.57	3.95	0.016	0.90
Having a partner to work with was useful.	3.62	2.00	0.001	2.15
I collaborated with my partners on the game.	4.43	2.00	0.001	3.61

to a programming phase when the sketch requires simple behaviors that are not programmed into the interactive objects and where Wizard-of-Oz prototyping may be difficult or tediously repetitive.

Additionally, we have observed that the fidelity of sketches created with *Raptor* is in most cases sufficient to evaluate game ideas. For example, during the study many participants informally commented that they were pleased to have created game sketches that actually felt like a real racing game. However some participants were also frustrated that Raptor does not include support for fine details they were accustomed to having in racing games. We've noticed that the most common request for new features is support for rapid sketching of heads-up display elements on the remote display. For example, in a first-person shooter sketch, it is currently difficult to show how much ammunition the player has left. We think by adding a simple heads-up display system, many of these problems can be addressed.

8. SUMMARY

We have presented *Raptor*: a novel end-user tool for game sketching based on a tabletop interaction. The system provides tool transparency through physical, gesture-based interactions suitable for end-users. *Raptor*'s "Wizard-of-Oz" style prototyping of game rules and behaviors allows non-programmers to engage in the design process on an equal footing with programmers. The physical layout of sketches on a tabletop surface helped provide engaging collaboration with fluid movement between the roles of game designer and play tester.

We performed a study to verify that end users could actually create a game sketch with the tool and to examine the role played by the tabletop computer in the user experience. Our results indicate that Raptor is indeed usable by end-users, and that participants found that the tabletop better supported local collaboration, and was generally preferred to a desktop interface.

9. REFERENCES

[1] M. Agustin, G. Chuang, A. Delgado, A. Ortega, J. Seaver, and J.W. Buchanan. Game sketching. In *DIMEA '07*, pages 36–43. ACM, 2007.

[2] Apple, Inc. iPhone, 2007.

[3] Bungie Studios, Inc. Halo, 2001.

[4] B. Buxton. *Sketching User Experiences*. Morgan Kaufmann, 2007.

[5] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 1969.

[6] M.A. Cusumano. Extreme programming compared with Microsoft-style iterative development. *CACM*, 50(10):15–18, 2007.

[7] N. Dahlbäck, A. Jönsson, and L. Ahrenberg. Wizard of Oz studies: why and how. In *IUI '93*, pages 193–200. ACM, 1993.

[8] Google, Inc. 3D Warehouse, 2009.

[9] T.C.N. Graham and W. Roberts. Toward quality-driven development of 3D computer games. In *Proc. DSV-IS*, pages 248–261. Springer LNCS, 2006.

[10] S. Huot, C. Dumas, P. Dragicevic, J. Fekete, and G. Hégron. The MaggLite post-WIMP toolkit: draw it, connect it and run it. In *Proc. UIST '04*, pages 257–266. ACM, 2004.

[11] W.H. Kruskal and W.A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

[12] J.A. Landay and B.A. Myers. Sketching interfaces: Toward more human interface design. *Computer*, 34(3):56–64, 2001.

[13] Microsoft, Inc. Microsoft Surface. *www.microsoft.com/surface*, 2008.

[14] Microsoft, Inc. XNA Game Studio, 2009.

[15] Nintendo of America, Inc. Super Mario Galaxy, 2007.

[16] D. Pinelle, N. Wong, and T. Stach. Heuristic evaluation for games: usability principles for video game design. In *Proc. CHI '08*, pages 1453–1462. ACM, 2008.

[17] B. Plimmer and M. Apperley. Making paperless work. In *Proc. CHINZ '07*, pages 1–8. ACM, 2007.

[18] M. Rauterberg, M. Fjeld, H. Krueger, M. Bichsel, U. Leonhardt, and M. Meier. BUILD-IT: a computer vision-based interaction technique for a planning tool. In *HCI'97*, pages 303–314, 1997.

[19] C. Remo. GDC: Molyneux demonstrates internal Lionhead experiments, March 2009.

[20] D.A. Schon. *The reflective practitioner: How professionals think in action*. Basic Books, 1983.

[21] S.D. Scott, K.D. Grant, and R.L. Mandryk. System guidelines for co-located, collaborative work on a tabletop display. In *Proc. ECSCW 2003*, pages 159–178. Kluwer Academic Publishers, 2003.

[22] C. Wolfe, J.D. Smith, and T.C.N. Graham. A low-cost infrastructure for tabletop games. In *Proc. Future Play*, pages 145–151. ACM, 2008.