Modifying the Quake II Engine for 'Civilian' Use

Timothy Weale, Brad Mellen and Donna K. Byron
email:{weale,mellen,dbyron}@cse.ohio-state.edu

The Ohio State University
Department of Computer Science and Engineering
Columbus, Ohio 43210

Technical Report: OSU-CISRC-11/05-TR74

April 11, 2006

Abstract

This report describes our modifications to the Quake II source code in order to 'sanitize' the content for a general audience, so that it can be used for spoken dialog corpus collection in an academic setting. Our changes include eliminating all weapons, disabling player damage, hiding specific elements of the GUI and removing the player stats from the screen. These modifications create a more immersive experience for the end user, which is expected to lead to a more natural dialog corpus.

This material is based on work supported by the OSU Computer Science and Engineering department.

Contents

1 Quake II Engine for Situated Collaboration

Researchers in Artificial Intelligence, Human-Computer Interaction, Human-Robot Interaction, and Discourse Analysis are increasingly interested in studying collaboration strategies for situated tasks. Situated tasks are those in which the activity and language of the task partners are influenced by their physical environment. Whether the end goal of their research is to develop intelligent agents that will be deployed as partners in such tasks, or whether the goal is to study human-to-human communication and collaboration behaviors, the starting point for many researchers is to record human partners working together on a task in a controlled, experimental setting that can be instrumented to provide precise recordings of experimentally-relevant aspects of their interaction. For spoken dialog systems development, the ideal experimental setting is one that can be inhabited by either teams of human partners or heterogeneous teams of human-intelligent agent partners to solve the identical task. The experimental setting described in this report, which fits this requirement, is a computer game engine called Quake II. Quake II can be used as a virtual world setting for collaborative tasks. The game engine has certain benefits and limitations that may make it useful as an experimental tool by researchers with a variety of experimental goals. The purpose of this report is to document changes that we found to be necessary in the game engine code in order to make the look-and-feel of the interface more appropriate for academic research on collaborative interaction.

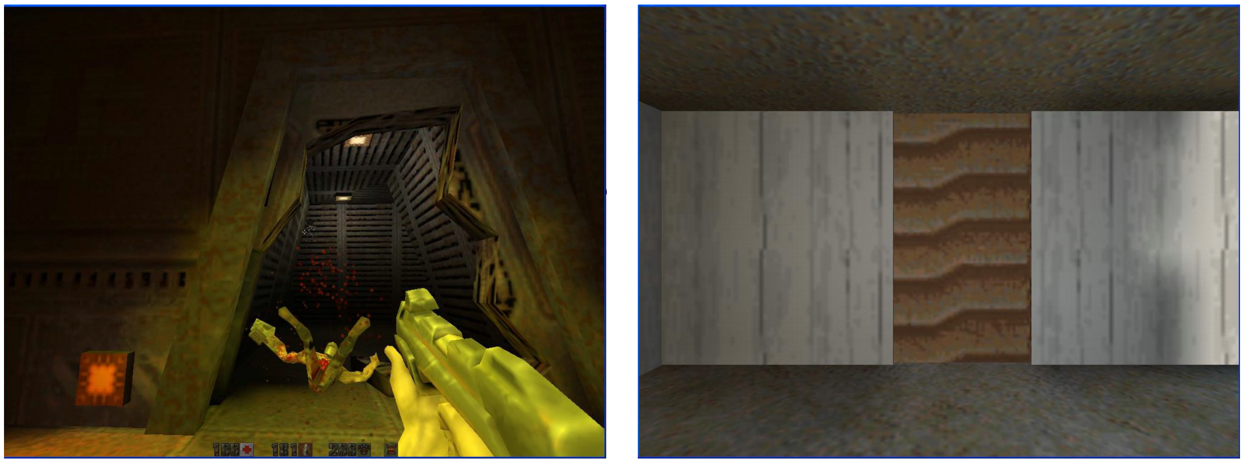
Our initial use for the Quake II software has been to record the discourse produced by pairs of human partners solving situated tasks [Byr05]. These collected dialogs will be used as attested language examples to motivate the development of spoken dialog agents for situated tasks. A variety of other task-based dialog corpora exist, for example TRAINS [HA95], Maptask [HGD$^+$93], ATIS [ATI93], and COMMUNICATOR [WRP$^+$02]. Although these corpora have provided the spoken dialog systems research community with a valuable resource to investigate dialog phenomena such as grounding, speech act sequencing, disfluencies and spoken language parsing, etc, the corpora were collected in experimental conditions that prevented the partners from sharing extra-linguistic context. In situated language, a speaker is located at a specific position in a three-dimensional environment; therefore, the influence of the extra-linguistic context on discourse is crucial. Although spatial properties such as 'on the left', 'to the right', 'there' or 'here' are widely studied in fields such as cognitive science, linguistics and psychology, the systematic investigation of non-linguistic context for intelligent conversational agents is in its infancy. In order to develop agents which can successfully function within a physical space, we must first understand how the environmental context changes the interpretation and production of linguistic acts. Quake II provides a low-cost environment that can be utilized by researchers for a range of situated collaboration experiments.

Other simulation environments have been used in dialog research before. For example, the game *Neverwinter Nights* has been used to create an immersive Intelligent Tutoring System that understands and responds to natural language [LDHS03] and for a conversational agent that collaborates with the user to complete a simple task [GR05a, GR05b]. Although the Neverwinter Nights engine places each player at a particular position in the world, their view of the task world is a top-down view rather than a true first-person view. So, while solving a task within the game, the user experiences a god's-eye-view of his avatar moving about in the world, which does not produce a strong sense of immersion. Another game engine, *Unreal Tournament*, has been used in several immersive training simulations created at USC. The virtual world rendered by the game engine is populated with animated characters that speak to each other and the trainee [SHG$^+$01, JT05, QJ05]. In some implementations, [TR02] the user is not allow to move about within the scene. Also, the source code for this simulation engine is not open source and is quite expensive to acquire.

The virtual world rendering engine described in this report, Quake II, has certain properties that we believe to be advantages over these other computer game engines. In terms of cost, the game is open source software that is available on the web and freely distributed under the GNU General Public License. Unlike true VR systems or high-end graphical world software engines such as Maya, Quake II runs on commodity hardware and under several commonly-used operating systems – Windows, Linux or OS X. Additionally, Quake II was developed in the early 1990's for use on consumer hardware of the time (Pentium 90 with 16 MB of RAM) and therefore requires little in terms of computing resources. These factors together mean that a researcher can use Quake II on desktop computers he already owns, with minimal expense to acquire the software. Because the software is open source, it is simple to make customizations such as changing the behavior of items in the virtual world, adding interaction with other software such as dialog engines, planning, or reasoning software, or adding instrumentation and logging functions necessary for experiments. Also, Quake II provides a built-in mechanism for recording sessions for playback. This mechanism includes logging of the user's spatial position, orientation and gaze direction at each instant of the game. Using this information, each subject's situational context variables can be precisely calculated and time-aligned with data from other modalities, such as recorded speech. Finally, map creation for Quake II is simple. There are many free and commercial software packages for map creation, which will allow researchers to easily create new environments for their studies.

In terms of the user's experience, Quake II renders the user's view of the world from the more immersive first-person perspective (see Figure 1 for an example). Navigation is also intuitive. For example, "Turning to the right" is accomplished by pressing the right-arrow key, which results in the scene depicted on the computer monitor moving as it would if the user turned his head to the right while standing in the scene. Additionally, Quake II does not allow for fine-grained movements that could be used as communicative gestures. This eliminates arm gestures and body language from the communication pathway. However, large-scale pointing gestures such as moving towards an object of interest are available. Depending the research objectives of a particular study, this may or may not be a desirable trait.

Despite these advantages, the default configuration for Quake II is inappropriate for most academic studies involving human subjects. As a first-person shooter video game, the behavior of the world and default graphical content delivered with the game is oriented around violence (see Figure 1a). The default interaction in the game world assumes a model in which a player attempts to avoid injury to himself in order to stay alive. This introduces stress, time-pressure, and fear of harm into the subject's experience, which a researcher may wish to remove unless they are important to his study. In addition, the computer display includes game-relevant statistics that detract from the user's feeling of immersion. The underlying source code can be customized to remove these elements. Figure 1b shows the resulting user view after all the changes in this report are applied to the Quake II engine.



a. Default Engine

b. Revised Engine

Figure 1: Changes to the Quake II environment for embodied dialog collection.

This Technical Report documents the modifications made by our research team to the Quake II engine to prepare it for use as a data collection environment. Other research teams who wish to use Quake II as a data collection tool may wish to make these same modifications to the distributed source code. Section 2 describes how to disable the world's ability to inflict harm on players. Section 3 shows how to eliminate any reference to weapons from the user's interface. Section 5 describes changes to remove unwanted heads-up-display output, which increases the user's immersion in the virtual world. Finally, Sections 4 and 6 describe additional modifications we made to the behavior of particular items in the world to make them more compatible with the collaborative task that we designed for our first dialog collection experiment [Byr05]. These item modifications are described merely as examples of additional changes that can be made to the game.

1.1 Quake II Installation

After downloading and unpacking the Quake II source, available from www.idsoftware.com, there will be several directories in the source tree:

```
client/
ctf/
game/
irix/
linux/
null/
q_common/
ref_gl/
ref_soft/
rhapsody/
server/
solaris/
win32/
```

In the following section, we will refer to files based on this directory structure. For example, the file *game/g_combat.c* is the file *g_combat.c* located in the *game* directory.

Installation instructions can be found in the source tree and online. This source tree has been compiled by the authors under both Windows XP and Red Hat Linux. A separate source tree can be found for OS X[1]. Please note that you will need a copy of the Quake II retail disk for textures and models before you can run the game.

[1]http://www.fruitz-of-dojo.de/php/download.php4?dlnr=6

2 Disabling Damage

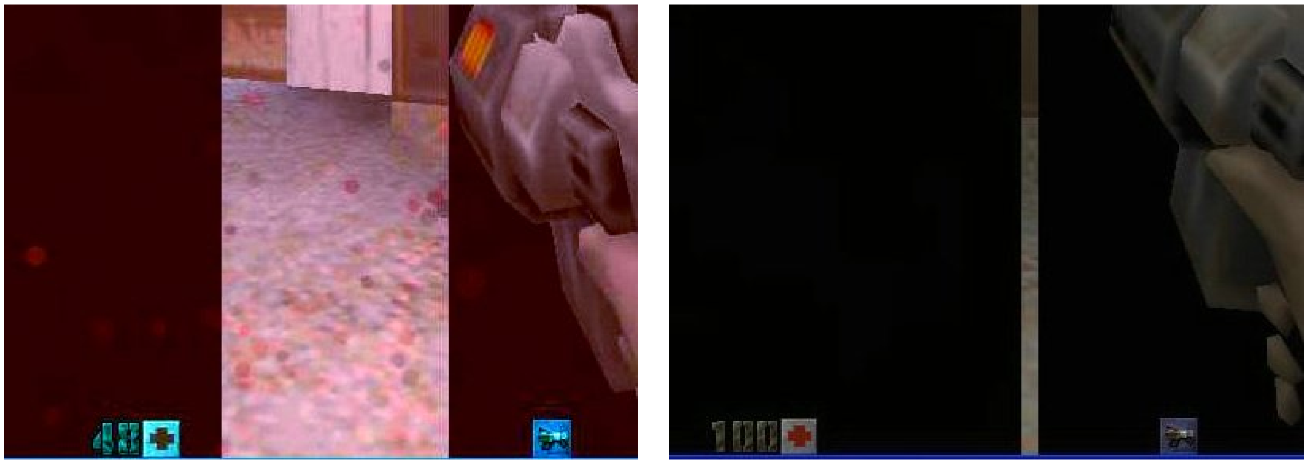
An essential part of every first-person shooter is the notion of ‘health’. The threat of death drives the drama of the game events forward. However, for human-subject studies in an academic environment, health should not even be a factor in the dialog. People should just be able to move and talk normally.

Removing monsters and weapons from our maps won’t solve the problem. Even in what seems like a completely harmless map, players can still get hurt by, e.g., falling from a height, getting caught in a door, etc.). It’s desirable to avoid ‘hurting’ or ‘killing’ subjects participating in a study. Therefore, we must disable the game’s ability to deal damage. The simplest way to do this is to eliminate damage calculations.

`Steps to Remove Damage Calculation` `(See A-1)`

1. `Open game/g_combat.c`
2. `Find the function T_Damage`
3. `Place a return statement on the first line of the function`[2]
4. `Comment out or eliminate the original code for T_Damage`

By doing this, the damage calculations are “short circuited”. When the engine tries to apply damage calculations the function has no effects – the method simply returns and gameplay continues as expected.



a. Default b. Modified

Figure 2: By removing damage, the user is no longer dealt damage by getting caught in a closing door.

[2] *Note that this will disable damage calculations and effects for all entities. This includes not only the players, but also any monsters, boxes, doors, buttons or walls which have health attributes. For the purposes of this research, we do not believe that this is a problem.*

3 Removing Weapons

In the game Quake II, there is considerable emphasis placed on weaponry. This is to be expected, as it is built upon a kill-or-be-killed model of gameplay. However, for our data collection purposes, we don't want the subjects to feel like they are in a war situation. To do this, we remove the engine's ability to both display and also shoot weaponry. By doing this, the subjects can avoid accidentally attacking each other and focus completely upon the given task.

`Steps to Remove Weapons` `(See A-2)`

1. `Open game/p_weapon.c`
2. `Find the function Think_Weapon`
3. `Place a return statement on the first line of the function`
4. `Comment out or eliminate the original code for Think_Weapon`

Through these modifications, we eliminate the engine's ability to utilize the Think_Weapon function, which controls both the firing and displaying of weapons. Weapons are never displayed and cannot be fired, effectively removing them from the environment.

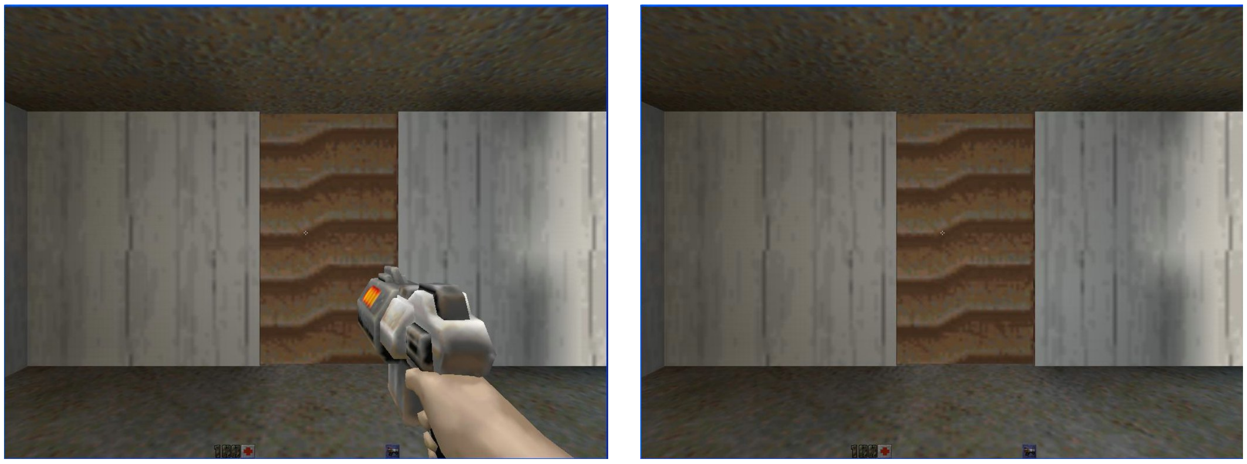

a. Default b. Modified

Figure 3: Removing the prominent visual positioning of weapons aids in environment immersion.

4 Changing Weapon and Item Names

Section 3 describes how to disable weapons. However, the Blaster will still show up in the user's inventory (see Figure 4 for an example inventory display). The engine seems designed to provide each player with a Blaster by default, and we can't seem to remove the item from the user's inventory without crashing the game. This is potentially confusing. Our solution is to rename the item – the Blaster is still in the inventory, but it now looks like something else.

4.1 Changing the Blaster Name

`Renaming the Blaster` (See A-6)

1. Open game/g_items.c
2. Look for the following comment:

   ```
   /* weapon_blaster (.3 .3 1) (-16 -16 -16) (16 16 16)
   always owned, never in the world
   */
   ```

   This is the area of the code which declares the properties of the blaster, including what Quake 2 names the item.
3. Going down a few lines, you should see the comment /* pickup*/. Right after it, there should be a string (default is ‘‘Blaster’’).
4. Change this to whatever you like, although we recommend ‘‘player’’

By doing this, the name of the item is changed in the Quake II environment. Thus, the inventory will display "player", removing all traces of the Blaster.

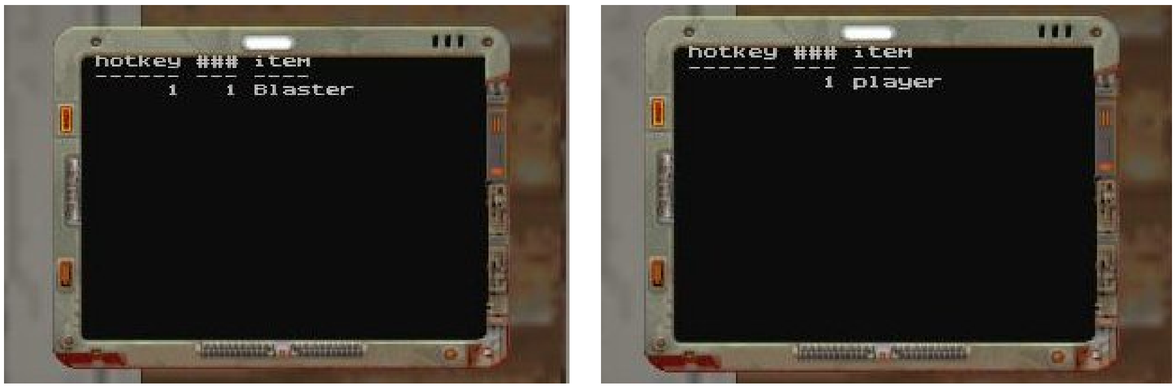
4.2 Changing the Blaster Initialization

Quake II gives the Blaster to the player at the beginning of the game and seems to always assume that the user is in possession of it. Removing this code causes the environment to crash. Therefore, we must give the item to the player.

However, we renamed it in the previous section and Quake II will not find a “Blaster” item. So, we change the name of the default item to match our changes.

`Initializing the New ‘‘Blaster’’` (See A-7)

1. Open game/p_client.c
2. Find the function InitClientPersistant
3. Near the beginning, there should be a call to FindItem with the parameter being the string ‘‘Blaster’’.
4. Change this to whatever you renamed the blaster to be (in the example case, ‘‘player’’).
5. *** NOTE: New names must match exactly. ***



```
hotkey ### item
------ --- ----
     1   1 Blaster
```

```
hotkey ### item
------ --- ----
         1 player
```

a. Default b. Modified

Figure 4: The user’s inventory display with ‘blaster’ renamed to ‘player’.

4.3 Changing Additional Item Names

The Blaster isn't the only item which can be renamed. Items such as the default names used by the Quake II community aren't lexical items known to the general public. Such items (Quad Damage, Rebreather, Silencer) can have their names changed in order to facilitate more natural dialog.

`Changing Item Names` (See A-6)

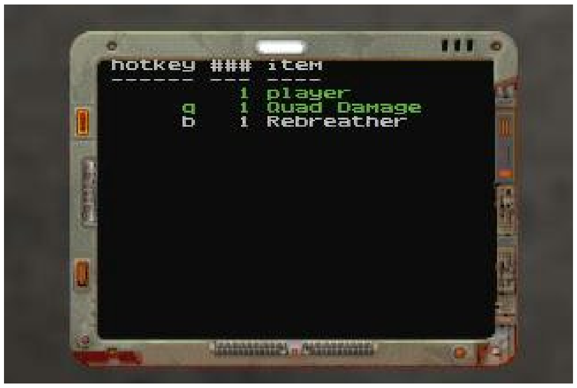
1. Open game/g_items.c
2. Find the following comments to change the given items

   Quad Damage `/*QUAKED item_quad (.3 .3 1) (-16 -16 -16) (16 16 16)`

   Rebreather `/*QUAKED item_breather (.3 .3 1) (-16 -16 -16) (16 16 16)`

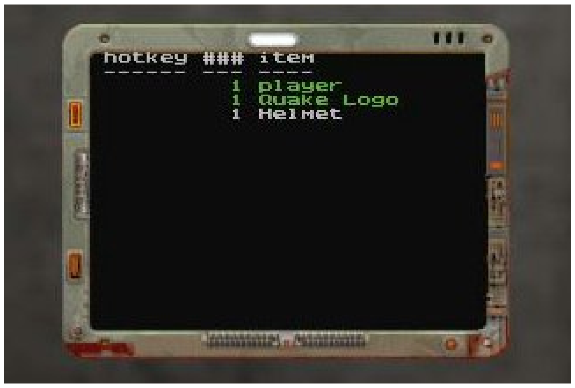
   Silencer `/*QUAKED item_silencer (.3 .3 1) (-16 -16 -16) (16 16 16)`

3. Change the name on the line which contains the comment ''/* pickup */''.



```
hotkey ### item
------ --- ----
         1 player
  q      1 Quad Damage
  b      1 Rebreather
```

a. Default



```
hotkey ### item
------ --- ----
         1 player
         1 Quake Logo
         1 Helmet
```

b. Modified

Figure 5: Using common terminology to label inventory items.

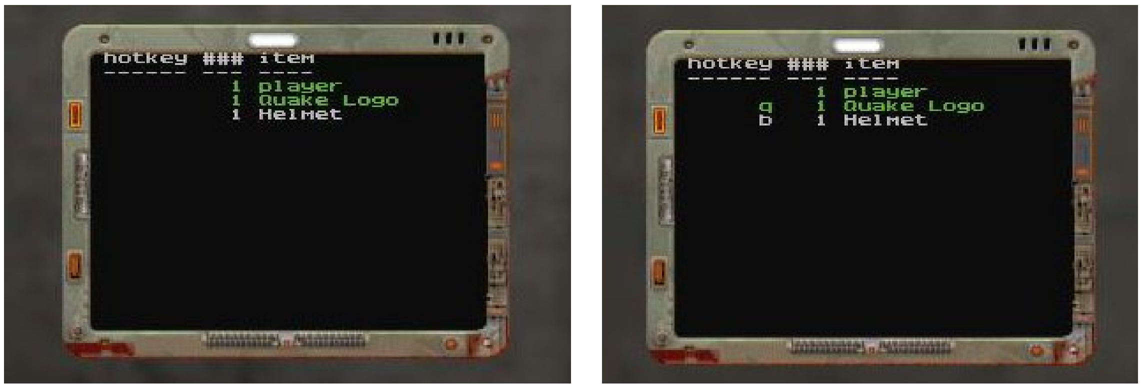
4.4 Configuration File

Quake II includes a default config file for keyboard layout(baseq2/config.cfg). However, the configuration is explicitly coded for the default names of items. The changes we made to the item names breaks these hotkey bindings. If we want to be able to use these items, we have to make changes to the configuration file for each renamed item.

```
Previous Bindings
bind b "use rebreather"
bind q "use quad damage"
bind s "use silencer"
```

```
Example Configuration Modifications
bind b "use helmet"
bind q "use quake logo"
bind s "use rectangle box"
```

These changes re-map the items to their hotkey bindings. They do not seem to be case sensitive.



```
hotkey ### item
------ --- ----
         1 player
         1 Quake Logo
         1 Helmet
```

a. Default

```
hotkey ### item
------ --- ----
         1 player
  q      1 Quake Logo
  b      1 Helmet
```

b. Modified

Figure 6: Remapping gives us hotkey options.

5 Changing the Heads-Up Display

This section describes modifications to the portion of the Quake II engine responsible for displaying information about the current state of the user to the screen. This information is also known as the Heads-Up Display (HUD). By removing the HUD, we will get a more immersive experience, without losing any information.

5.1 Removing the User Stats

User stats are the final visual reminder to the user that they are not actually embodied within the world. Without the following modifications, the environment will still show the user's health and weapon selection. Since we have disabled both elements, neither really needs to be shown. We can therefore remove these from the screen. By removing the stats, we get a surprising increase in the immersive effect of the engine.

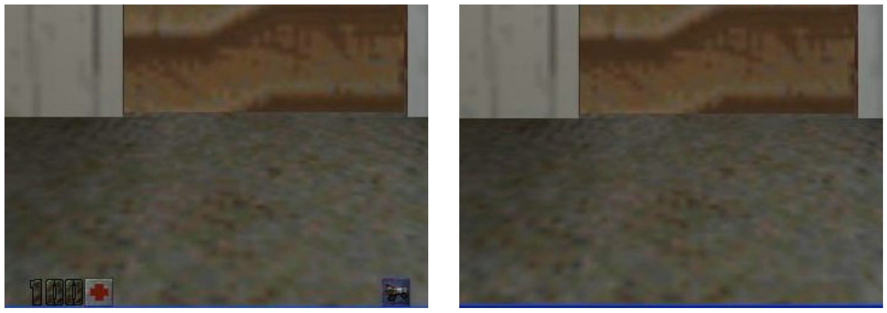
5.1.1 details how to remove all HUD output, regardless of source. 5.1.2 removes the health, ammo and weapon displays, but allows additional information to be displayed (such as when an item is picked up). Both are essentially equal in the traditional immersion, but one allows for a bit more information to be given to the user.

5.1.1 Removing all HUD Output

`Steps to Completely Remove HUD Output` `(See A-4)`

1. `Open client/cl_scrn.c`
2. `Find the function SCR_DrawStats`
3. `Remove the line containing SCR_ExecuteLayoutString`
   `(the only line in the function)`

This completely eliminates all functionality of the HUD, because the method call is never executed. Therefore, nothing will be displayed on the screen.



a. Default b. Modified

Figure 7: Removing all HUD output gives us the most immersive view possible.

5.1.2 Retaining Special HUD Output

`Steps to Retain Item HUD Output` `(See A-5)`

1. `Open client/cl_scrn.c`
2. `Find the function SCR_ExecuteLayoutString`
3. `Remove or comment out the following 'if' blocks:`
   - `(!strcmp(token, "pic"))`
   - `(!strcmp(token, "client"))`
   - `(!strcmp(token, "ctf"))`
   - `(!strcmp(token, "picn"))`
   - `(!strcmp(token, "num"))`
   - `(!strcmp(token, "hnum"))`
   - `(!strcmp(token, "anum"))`
   - `(!strcmp(token, "rnum"))`

   `*** NOTE: These 'if' blocks should be consecutive within the code. ***`

Most traditional HUD output is suppressed in this modification in that all armor, weapon and health-related displays will not be shown. However, if the user picks up an item, this will be displayed to the screen. This provides immediate feedback for the user, and might be useful for subjects inexperienced with the Quake II environment.

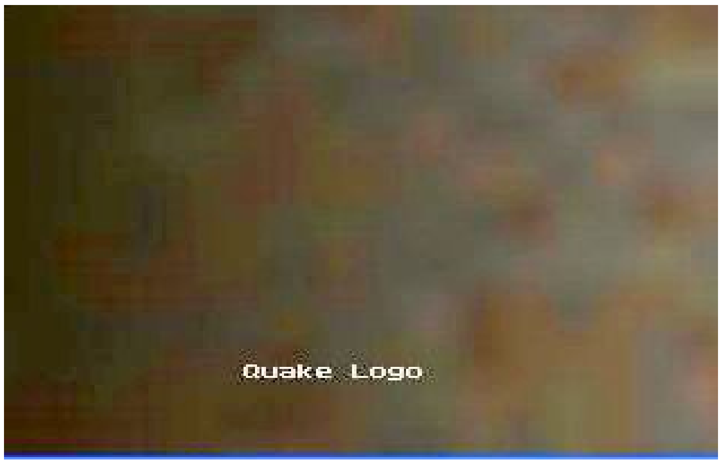


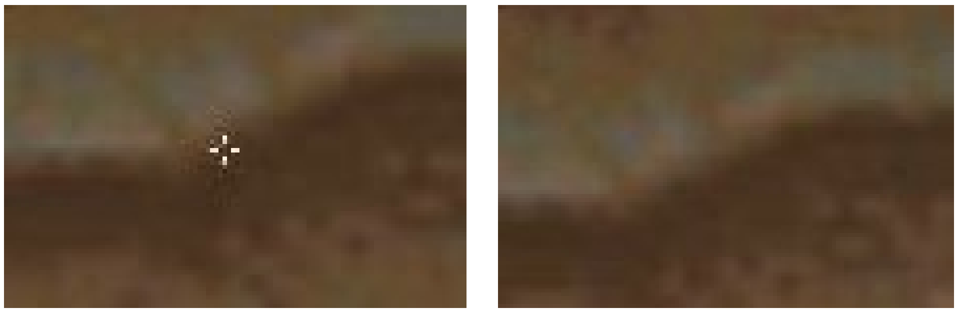
Figure 8: HUD notification of item pickup.

5.2 Removing the Crosshair

A minor annoyance, the crosshair is a constant reminder to the user that they are not truly in the environment. By removing it, the subject is given a slightly better feeling of immersion. While it can be removed as a command-line option, removing the crosshair in the code allows us to fully ensure that it will not be displayed.

`Steps to Remove the Crosshair` `(See A-3)`

1. `Open client/cl_view.c`
2. `Find the function V_RenderView`
3. `Remove the line containing SCR_DrawCrosshair`



a. Default b. Modified

Figure 9: Removing the crosshair.

6 Changing Item Interaction

Since we're using Quake II as an environment only and not playing the game, we can do non-traditional things with the given items. For example, the task assigned to subjects of the Quake2004 corpus [Byr05], was a scavenger hunt. This required the subjects to pick up, carry and drop off the items in different locations of the map. In order to accomplish this, several modifications to the source code were required.

6.1 Disabling Automatic Item Pickup

Quake II automatically picks up an item once you position yourself over it. By explicitly forcing the user to take some kind of action (push a button) before the item is added to their inventory, they could feel comfortable using the term "pick up" to describe their actions, rather than automatically just acquiring the item whenever it was touched.

```
Disabling Auto Pickup                                              (See A-6)

  1. Open game/g_items.c

  2. Find the function Touch_Item

  3. Find the lines:
     if(!other->client)
     return;

  4. Add the following after the previous lines:
     if(!(other->client->buttons & BUTTON_ATTACK))
     return;
```

Touch_Item is the function that is called when another entity is positioned over an item (weaponry, ammo, armor, quad damage, etc.). Usually, the item is automatically acquired when touched. This modification requires the user to not only be positioned on top of the item, but also push the key assigned to the 'Attack' button (in our case, Ctrl). Only then will the item be added to the inventory.

6.2 Disabling Item Activation

When an item is acquired, it is typically used automatically. For example, picking up a Quad Damage will give the player increased strength for a period of time, after which the item goes away. In a scavenger hunt environment, we don't want our items to be used up and disappear! So, we made modifications to the code which allow the user to carry items without activating the item functions.

`Disabling Item Activation` (See A-6)

1. Open `game/g_items.c`
2. Find the following functions:
   - `Use_Quad`
   - `Use_Envirosuit`
   - `Use_Invulnerability`
   - `Use_Silencer`
   - `Use_Breather`
   - `Use_PowerArmor`
3. Comment everything out except for the following lines:
   ```
   ent->client->pers.inventory[ITEM_INDEX(item)]-- --;
   ValidateSelectedItem (ent);
   ```
4. If both lines aren't in the function already, add them (as in Use_PowerArmor)

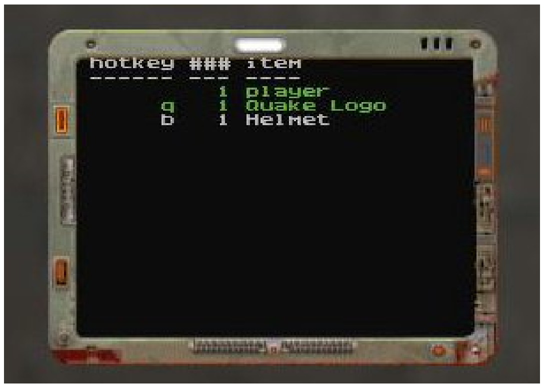
6.3 Dropping Items

The Quake2004 scavenger hunt task required picking up and dropping items, but the default engine does not have a ‘drop’ command. Therefore, the Quake II engine had to be modified so that an item could be ‘dropped’.

`Adding Item` (See A-6)

1. Open game/g_items.c
2. Find the following functions:
    - Use_Quad
    - Use_Envirosuit
    - Use_Invulnerability
    - Use_Silencer
    - Use_Breather
    - Use_PowerArmor
3. Find the following line: ‘‘ValidateSelectedItem (ent);’’
4. Add the following line after the ValidateSelectedItem call: ‘‘Drop_Item(ent, item);’’

Now, instead of using an item automatically, you throw the item if you hit its hotkey (shown in the inventory display popup).



```
hotkey ### item
------ --- ----
         1 player
   q     1 Quake Logo
   b     1 Helmet
```

Figure 10: Key bindings to ‘throw’ the items.

7 Summary

In this technical report, we have detailed our changes to the Quake II engine. These modifications sanitize the content of the game, taking it away from its roots as a violent first-person shooter game. Ultimately, we believe these changes allow us to use the Quake II engine as an immersive environment for embodied dialog research.

First, we disabled damage to the player, removing the threat of accidental ‘injury’ and ‘death’ for the subjects. Then, all weapons are removed from the field of view and the firing mechanism cannot be executed. Then, the HUD is removed, allowing for an optimal immersion within the environment. Additional modifications such as item/weapon name changes and item pickup/drop-off are discussed. Finally, changes to the engine configuration file are discussed.

The Quake 2004 Corpus [Byr05] contains an initial discussion of our human/human dialog recording environment, including a description of the equipment involved and the client/server setup for the Quake II environment. Future work in this domain include detailing our final recording environment. This will allow for a complete system for corpus collection, including not only the words spoken by the subjects, but also a listing of the subject’s positions and actions during their time in the environment. Additionally, we plan to extend the Quake II domain to include an automated agent capable of understanding and generating embodied dialog.

References

[ATI93] ATIS. The ATIS corpus http://www.ldc.upenn.edu/catalog/catalogentry.jsp?catalogid=ldc93s4a, 1993.

[Byr05] Donna K. Byron. The OSU Quake 2004 corpus of two-party situated problem-solving dialogs. Technical report, The Ohio State University, 2005.

[GR05a] Peter Gorniak and Deb Roy. Probabilistic grounding of situated speech using plan recognition and reference resolution. In *Proceedings of the 7th international conference on Multimodal Interfaces*, pages 138–143, Torento, Italy, 2005. ACM Press.

[GR05b] Peter Gorniak and Deb Roy. Speaking with your sidekick: Understanding situated speech in computer role playing games. In *Proceedings of Artificial Intelligence and Digital Entertainment*, 2005.

[HA95] P. Heeman and J. Allen. The Trains spoken dialog corpus. CD-ROM, Linguistics Data Consortium, 1995.

[HGD+93] Charles T. Hemphill, John J. Godfrey, George R. Doddington, John Garofolo, Jonathan Fiscus, Nancy Dahlgren William Fisher, Brett Tjaden, and David Pallett. Hcrc map task corpus: Linguistics data consortium catalog no. ldc93s12, 1993.

[JT05] Dusan Jan and David Traum. Dialog simulation for background characters. *5th International Working Conference on Intelligent Virtual Agents*, 2005.

[LDHS03] Susann Luperfoy, Eric Domeshek, Elias Holman, and David Struck. An architecture for incorporating spoken dialog interaction with complex simulations. In *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC 2003)*, 2003.

[QJ05] Lei Qu and Lewis Johnson. Detecting the learner's motivational states in an interactive learning environment. In *Proceedings of the 12th International Conference on Artificial Intelligence in Education (AIED 2005)*, 2005.

[SHG+01] W. Swartout, R. Hill, J. Gratch, W.L. Johnson, C. Kyriakakis, C. LaBore, R. Lindheim, S. Marsella, D. Miraglia, B. Moore, J. Morie, J. Rickel, M. Thibaux, L. Tuch, R. Whitney, and J. Douglas. Toward the holodeck: Integrating graphics, sound, character and story. *Proceedings of the fifth international conference on Autonomous agents*, 2001.

[TR02] David Traum and Jeff Rickel. Embodied agents for multi-party dialogue in immersive virtual worlds. In *Proceedings of first International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS '02)*, pages 766–773, Bologna, Italy, 2002.

[WRP+02] M.A. Walker, A. Rudnicky, R. Prasad, J. Aberdeen, E. Owen Bratt, J. Garofolo, H. Hastie, A. Le, B. Pellom, A. Potamianos, R. Passonneau, S. Roukos, G. Sanders, S. Seneff, and D. Stallard. DARPA communicator: Cross-system results for the 2001 evaluation. In *ICSLP-2002:Inter. Conf. on Spoken Language Processing*, volume 1, pages 273–276, Denver, CO USA, Sept. 2002.

8 Appendices

The following pages include sections of our modified code. This is intended to show more context for the source code changes, and may be used for reference. All modifications are done in the *game* and *client* directories of the original source tree.

See 1.1 for more information about the Quake II source tree.

A-1 game/g_combat.c

```
void T_Damage (edict_t *targ, edict_t *inflictor, edict_t *attacker, vec3_t dir, vec3_t point,
               vec3_t normal, int damage, int knockback, int dflags, int mod)
{
  return;
}
```

A-2 game/p_weapon.c

```
void Think_Weapon (edict_t *ent)
{
  return;
}
```

A-3 client/cl_view.c

```
void V_RenderView( float stereo_separation )
{
  extern int entitycmpfnc( const entity_t *, const entity_t * );

  if (cls.state != ca_active)
    return;

  ...

    SCR_AddDirtyPoint (scr_vrect.x, scr_vrect.y);
    SCR_AddDirtyPoint (scr_vrect.x+scr_vrect.width-1,
      scr_vrect.y+scr_vrect.height-1);

  //SCR_DrawCrosshair ();
}
```

A-4 client/cl_scrn.c (Version 1)

```
void SCR_DrawStats (void)
{
  //SCR_ExecuteLayoutString (cl.configstrings[CS_STATUSBAR]);
}
```

A-5 client/cl_scrn.c (Version 2)

```
void SCR_ExecuteLayoutString (char *s)
{
  ...

  while (s)
  {
    token = COM_Parse (&s);
    if (!strcmp(token, "xl"))
    {
      token = COM_Parse (&s);
      x = atoi(token);
      continue;
    }

    ...

    if (!strcmp(token, "yv"))
    {
      token = COM_Parse (&s);
      y = viddef.height/2 - 120 + atoi(token);
      continue;
    }

    /*if (!strcmp(token, "pic"))
    { // draw a pic from a stat number
      token = COM_Parse (&s);
      value = cl.frame.playerstate.stats[atoi(token)];
      if (value >= MAX_IMAGES)
        Com_Error (ERR_DROP, "Pic >= MAX_IMAGES");
      if (cl.configstrings[CS_IMAGES+value])
      {
        SCR_AddDirtyPoint (x, y);
        SCR_AddDirtyPoint (x+23, y+23);
        re.DrawPic (x, y, cl.configstrings[CS_IMAGES+value]);
      }
      continue;
    }

    ...

    if (!strcmp(token, "rnum"))
    { // armor number
      int color;

      width = 3;
      value = cl.frame.playerstate.stats[STAT_ARMOR];
      if (value < 1)
        continue;

      color = 0; // green
      if (cl.frame.playerstate.stats[STAT_FLASHES] & 2)
        re.DrawPic (x, y, "field_3");

      SCR_DrawField (x, y, color, width, value);
      continue;
    }*/

    if (!strcmp(token, "stat_string"))
    {
      token = COM_Parse (&s);
      index = atoi(token);
      if (index < 0 || index >= MAX_CONFIGSTRINGS)
        Com_Error (ERR_DROP, "Bad stat_string index");
      index = cl.frame.playerstate.stats[index];
      if (index < 0 || index >= MAX_CONFIGSTRINGS)
        Com_Error (ERR_DROP, "Bad stat_string index");
      DrawString (x, y, cl.configstrings[index]);
      continue;
    }

    ...

    if (!strcmp(token, "if"))
    { // draw a number
      token = COM_Parse (&s);
      value = cl.frame.playerstate.stats[atoi(token)];
      if (!value)
      { // skip to endif
        while (s && strcmp(token, "endif") )
        {
          token = COM_Parse (&s);
        }
      }

      continue;
    }

  }
}
```

A-6 game/g_items.c

```
void Use_Quad (edict_t *ent, gitem_t *item)
{
  ent->client->pers.inventory[ITEM_INDEX(item)]--;
  ValidateSelectedItem (ent);
  Drop_Item(ent, item);
}

void Use_Breather (edict_t *ent, gitem_t *item)
{
  ent->client->pers.inventory[ITEM_INDEX(item)]--;
  ValidateSelectedItem (ent);
  Drop_Item(ent, item);
}

void Use_Envirosuit (edict_t *ent, gitem_t *item)
{
  ent->client->pers.inventory[ITEM_INDEX(item)]--;
  ValidateSelectedItem (ent);
  Drop_Item(ent, item);
}

void Use_Invulnerability (edict_t *ent, gitem_t *item)
{
  ent->client->pers.inventory[ITEM_INDEX(item)]--;
  ValidateSelectedItem (ent);
  Drop_Item(ent, item);
}

void Use_Silencer (edict_t *ent, gitem_t *item)
{
  ent->client->pers.inventory[ITEM_INDEX(item)]--;
  ValidateSelectedItem (ent);
  Drop_Item(ent, item);
}

void Use_PowerArmor (edict_t *ent, gitem_t *item)
{
  ent->client->pers.inventory[ITEM_INDEX(item)]--;
  ValidateSelectedItem (ent);
  Drop_Item(ent, item);
}

void Touch_Item (edict_t *ent, edict_t *other, cplane_t *plane, csurface_t *surf)
{
  qboolean taken;

  if (!other->client)
    return;
  if(!(other->client->buttons & BUTTON_ATTACK))
    return;
  //REST OF CODE...
```

```
}

/* weapon_blaster (.3 .3 1) (-16 -16 -16) (16 16 16)
always owned, never in the world
*/
  {
    \"weapon$_$blaster\",
    NULL,
    Use_Weapon,
    NULL,
    Weapon_Blaster,
    \"misc/w_pkup.wav\",
    NULL, 0,
    \"models/weapons/v$_$blast/tris.md2\",
/* icon */    \"w_blaster\",
/* pickup */  \"player\",
    0,
    0,
    NULL,
    IT_WEAPON|IT_STAY_COOP,
    WEAP_BLASTER,
    NULL,
    0,
/* precache */ \"weapons/blastf1a.wav misc/lasfly.wav"
  },

/*QUAKED item_quad (.3 .3 1) (-16 -16 -16) (16 16 16)
*/
  {
    "item_quad",
    Pickup_Powerup,
    Use_Quad,
    Drop_General,
    NULL,
    "items/pkup.wav",
    "models/items/quaddama/tris.md2", EF_ROTATE,
    NULL,
/* icon */    "p_quad",
/* pickup */  "Quake Logo",
/* width */   2,
    60,
    NULL,
    IT_POWERUP,
    0,
    NULL,
    0,
/* precache */ "items/damage.wav items/damage2.wav items/damage3.wav"
  },

/*QUAKED item_silencer (.3 .3 1) (-16 -16 -16) (16 16 16)
*/
  {
    "item_silencer",
    Pickup_Powerup,
    Use_Silencer,
    Drop_General,
    NULL,
    "items/pkup.wav",
    "models/items/silencer/tris.md2", EF_ROTATE,
    NULL,
/* icon */    "p_silencer",
/* pickup */  "Rectangle Box",
/* width */   2,
    60,
    NULL,
    IT_POWERUP,
    0,
    NULL,
    0,
/* precache */ ""
  },

/*QUAKED item_breather (.3 .3 1) (-16 -16 -16) (16 16 16)
*/
  {
    "item_breather",
    Pickup_Powerup,
    Use_Breather,
    Drop_General,
    NULL,
    "items/pkup.wav",
    "models/items/breather/tris.md2", EF_ROTATE,
    NULL,
/* icon */    "p_rebreather",
/* pickup */  "Helmet",
/* width */   2,
    60,
    NULL,
    IT_STAY_COOP|IT_POWERUP,
    0,
    NULL,
    0,
/* precache */ "items/airout.wav"
  },

/*QUAKED item_enviro (.3 .3 1) (-16 -16 -16) (16 16 16)
*/
  {
    "item_enviro",
    Pickup_Powerup,
    Use_Envirosuit,
    Drop_General,
    NULL,
    "items/pkup.wav",
    "models/items/enviro/tris.md2", EF_ROTATE,
    NULL,
/* icon */    "p_envirosuit",
/* pickup */  "Suit",
/* width */   2,
    60,
    NULL,
    IT_STAY_COOP|IT_POWERUP,
    0,
    NULL,
    0,
/* precache */ "items/airout.wav"
  },
```

A-7 game/p_client.c

```
void InitClientPersistant (gclient_t *client)
{
  gitem_t *item;

  memset (&client->pers, 0, sizeof(client->pers));

  item = FindItem("player");
  client->pers.selected_item = ITEM_INDEX(item);
  client->pers.inventory[client->pers.selected_item] = 1;

  client->pers.weapon = item;

  client->pers.health = 100;
  client->pers.max_health = 100;

  client->pers.max_bullets = 200;
  client->pers.max_shells = 100;
  client->pers.max_rockets = 50;
  client->pers.max_grenades = 50;
  client->pers.max_cells = 200;
  client->pers.max_slugs = 50;

  client->pers.connected = true;
}
```