Project: DXF 0662

Video Game Learning and Tutorials

A Major Qualifying Project Report:

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Andrew Bangs **Christopher St. Pierre**

Approved:

Professor David X. Finkel, Advisor

Professor Robert W. Lindeman, Advisor

Abstract

Since video games have become more complicated, requiring more comprehensive in-game help, our project's goal was to provide a tool to fulfill the need to study in-game help. Our tool is built on a custom game and utilizes different types of help in different instances of the game, records time spent using help and is fully extensible. By providing a video game tool that collects empirical data of learning habits, researchers can analyze the effectiveness of in-game teaching methods.

Table of Contents

1. Introduction

As the video game industry has grown over the past years, a lot of attention has been paid to the marketing of games for their aesthetics and complexity. Consequently, games have become more complex and more difficult to learn. Such an explosion in the sales of games and the overall success of the video game industry in the past few years has warranted research into more ways to attract players and keep players' attention throughout game play.

Our project was to create a video-game tool that could be used to research the learning habits of video game players with regards to video games themselves. Using such a tool, research and statisticians could easily pinpoint the most effective learning methods incorporated in games today. The development of the game and tool was divided into three broad but important steps to insure the creation of an effective final product.

First, we created a custom game to build the data gathering tool on. The biggest challenge here was quickly designing a short, custom and robust game with the ability to utilize multiple learning tools. Although the first part of the project was only to incorporate the design process, and no actual game development, it still involved story development, game design, art collection and other design elements.

After the design of the game was finalized and the development process was started, we would then move on to deciding on the types of learning tools that would be implemented in the tool itself. Using a list of games we compiled and the learning tools they incorporate, we decided on the best tools to implement. We would then use our development platform to build these tools into the game.

Finally, throughout the entire development process we would need to develop a way to record the number of times specific types of help were utilized, as well as the length of time they were used by the player. With this final piece of the tool implemented, we would have a final product that is ready for use in learning research.

This MQP report was designed to help describe the need for this video game tool, as well as explain the development process we followed throughout the project.

2. Purpose of the Tool

The Double Vision video game tool was designed specifically for use in a research environment. Before the gameplay begins, testing players should have access to the user's manual as they would if the game were a normal off-the-shelf purchase. They should not be given any information about the data collection processes in the game other than the knowledge that what they are doing will be recorded and analyzed. However, how much information is revealed to the test subjects is left up to the research group.

To gather data on the learning habits of players, the person administering a test (the proctor) using our game tool can choose from many different help functions available. Double Vision includes the following learning tools as part of the executable:

- Audio Help – The information gathered on audio help is the number of times that the player requested the audio help.
- Controls Maps – The information gathered on the controls map is the number of times that the player requested the controls map to be displayed, as well as how long the player left the controls map on the screen. An example of a controls map is seen in Figure 1.

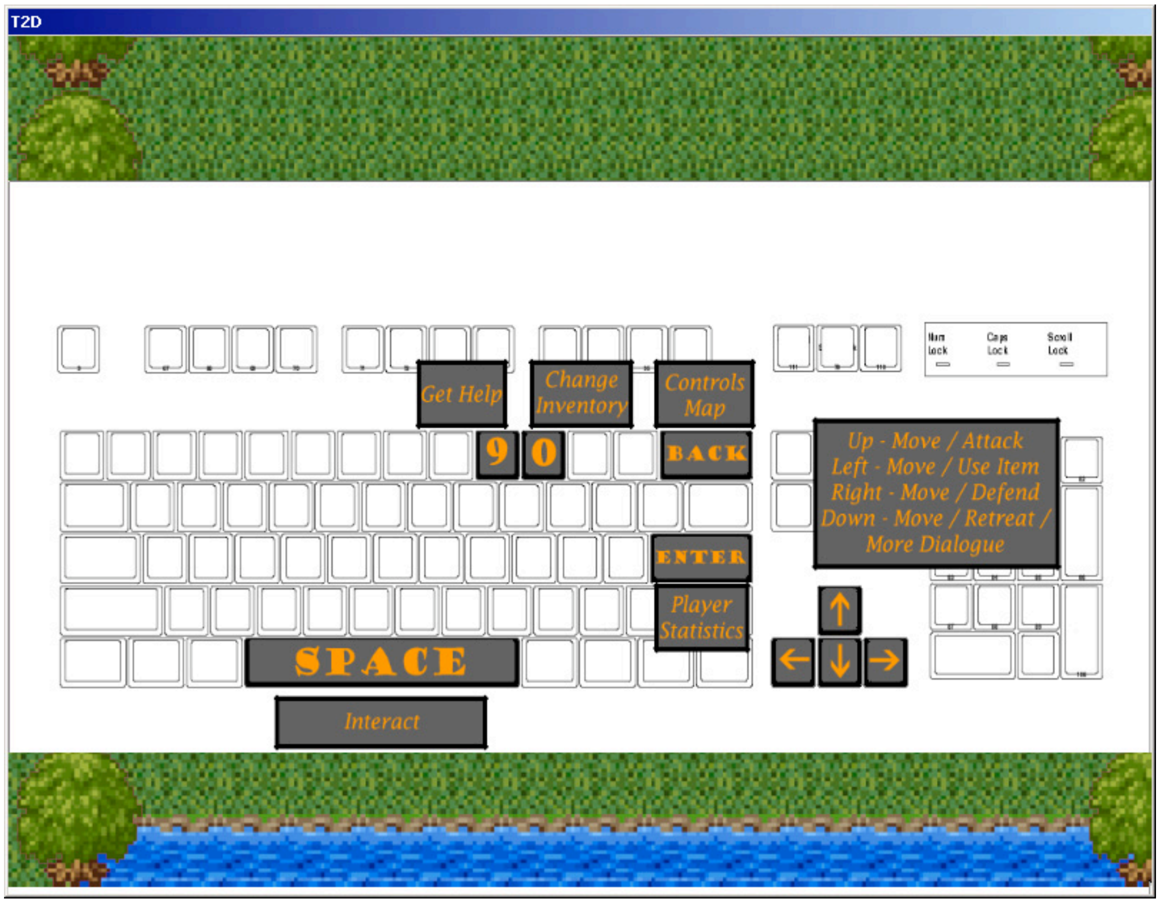
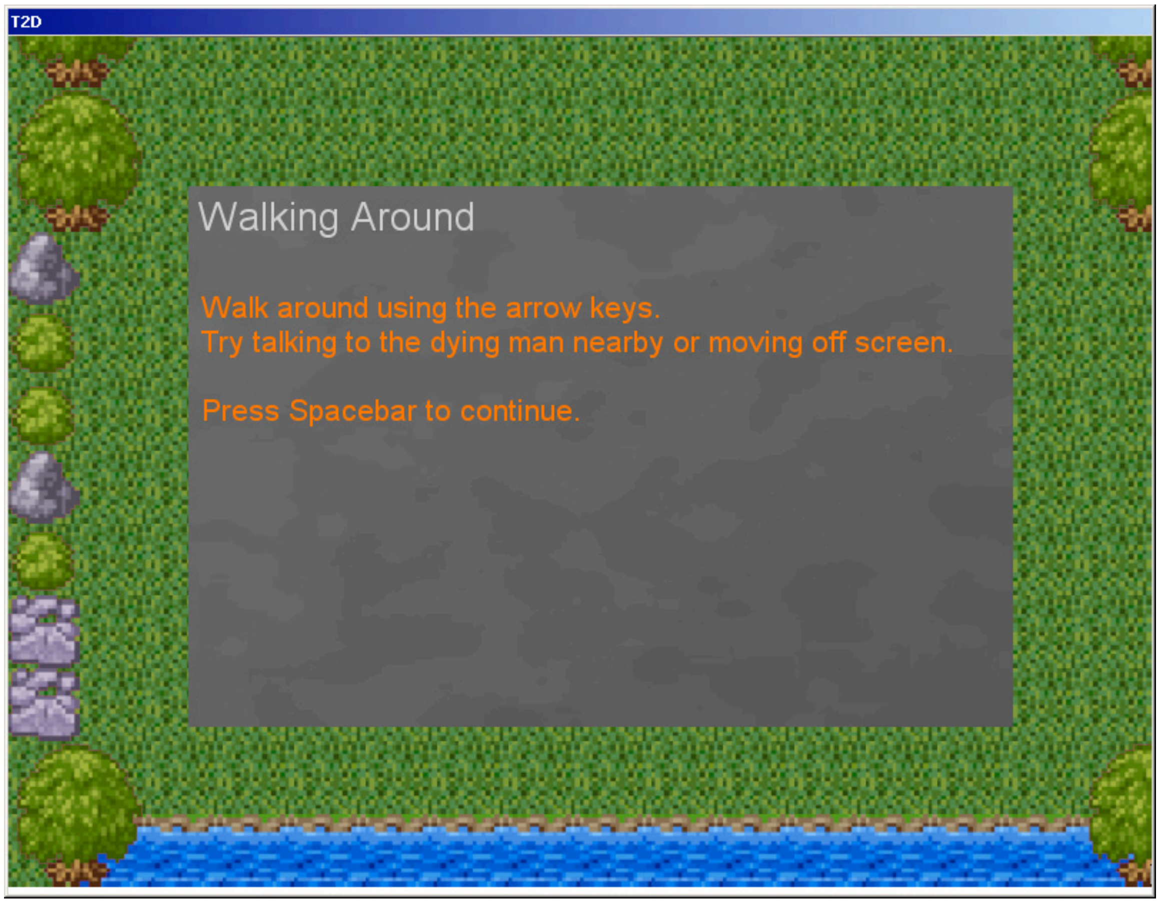


Figure 1 - An Example of a Controls Map

- Popup Help - The information gathered on the popup help is the number of times that the player requested the controls map to be displayed, as well as how long the player left the popup help on the screen. An example of popup help is seen in Figure 2.



Walking Around

Walk around using the arrow keys.
Try talking to the dying man nearby or moving off screen.

Press Spacebar to continue.

Figure 2 - An Example of Popup Help

The proctor can begin the experiment by running any one of the batch files provided. By executing the proper batch file, they are able to utilize each of the different learning tools, all learning tools or no learning tools for the player to use in the game. Each learning method has different logging events associated with it that will be recorded into a log file in a format suitable for the data being saved. Therefore, such data as times in menus, number of key presses, which help was used most often during gameplay and any other data that the proctor deems relevant to their study can be collected. By providing the ability to separate the use of the various learning tools into different batch files, and therefore different instances of playing the game, we created the ability to record many different combinations of statistical data.

3. Background and Research

Before we began, we first examined the possibility of other organizations utilizing video games as a teaching tool. The first recognized organization we found to use games to teach was the United States Army. The United States Army Research Institute for the Behavioral Sciences performed many different experiments using PC based computer games. These experiments studied behavioral patterns in players in order to find out whether or not video game playing had any impact on social behavior, as well as reaction time, focus, and other mental and motor skills [GAME INSTRUCTIONS].

Using these findings, the United States Army found that they could design and build their own video games, or simulations, in order to train soldiers. The idea was that using the tools they found to be most effective in the learning processes of players, they could create a simulation game effective in properly training soldiers how to think and react in situations the military can control [GAME INSTRUCTIONS]. Although all of this research did prove to be effective for the United States Army in aiding their simulation designs, it did not effectively describe how it could be practically used to create more effective video game learning tools.

Another document we examined before beginning our tool was a record of the results of a study done by London Knowledge Lab, Institute of Education at the University of London. Their study looked into how to make video game players better learners, but only in the context of video games themselves. They focused on such things as how much people remembered after specific tutorial types and learning methods. They also recorded how much better at the game the people became based on the learning tools that they implemented into the gameplay [LIBERTY]. Although this information gave us some insight into the types of learning methods we could possibly insert into our tool, it was not exactly the type of study we would be designing our tool for. It did help to learn about what types of skills people acquire as they play games, but just as the United States Army research, it did not give insight into what types of learning worked best.

These were our primary sources before beginning our own coding and analysis. Further research did not yield any other resources that indicated the type of tool we wanted to create had been attempted or created in the past. We therefore decided the endeavor was worthwhile and began organizing our efforts to begin planning the development of our game and tool simultaneously.

4. Tools and Design

4.1 Game Genre

Using a game as the base for a learning tool imposed a difficult decision: what is the best genre for the task? Creating an effective tool meant designing a game that would be challenging to learn to play without the use of the built-in learning methods. Therefore, a genre had to be selected that as few people as possible would be able to fully understand and master from the start. There were also time constraints involved in creating the game. Since both the game and the learning tools needed to be created, little development time could be allocated to, for example, creating a large amount of artwork or developing a complicated strategy game. The choices were narrowed down to two: an action shooter game, or a role-playing game.

The shooter game was initially a strong choice, because many action games have simple controls, low artistic requirements, would be easy to create, and can be picked up and quickly enjoyed by many people. The problem with using an action game for this project would come as a result of the ease of playability. Creating an action game with purposely difficult controls would not be a useful endeavor, because then the player's greatest challenge would be figuring out which buttons correspond to which actions. Alternatively, the game could be more strategic; for example, each enemy could only be defeated in one way. However, these challenges would have felt contrived, rather than arising as a natural part of playing the game. This artificiality could confuse potential test subjects and disrupt any conclusions that might be drawn from the study.

Role-Playing games (RPGs) are commonly accepted to contain elements of character building, a strong narrative, and a combat system. Contrary to action games, RPGs have a lot of information available to the player, have huge worlds requiring a great amount of art development, and have a steep learning curve. Players need to understand what each statistic does, gather information and solve puzzles from non-player characters (NPCs), and be able to understand and master the combat system. It was this learning curve that gave the strongest case for RPGs, since the inherent difficulty of learning one of these games would be worthy of analysis.

4.2 Tortoise SVN

To facilitate content control we used freeware known as Tortoise SVN. Tortoise SVN is a concurrent-versioning system (CVS) we had installed on the WPI Game Development Club's (GDC's) main server in Fuller Laboratories at WPI. It utilizes a Windows context menu interface and offers all of the CVS functions necessary for our project [TORTOISE].

Although we researched many different tools to use for our project, "Tortoise SVN" was the first CVS software we looked at and used. "Tortoise SVN" is readily available on the GDC main server and integrates well with the GDC project server, which runs "Subversion Control" [SVN]. Since it offered a shell based context menu system, there was no reason to choose any other version control software.

4.3 Torque 2D

For our project, we chose Torque 2D (T2D) as the development engine. Before we began our own development process, we looked at examples of games produced with T2D that were similar to the game we were expecting to produce. We examined the scripts used to make and run these games and decided that the codebase was complex enough to provide a good starting point and work well with our time constraints. Since we had previously determined that our tool and game were going to be 2D, T2D became one of our primary development tools to assess [GG1]. The key aspect of our game was that it be challenging to learn. Gameplay elements such as aesthetics and story were secondary goals. Therefore, a game engine that was simple to learn and easy to use was favored over others. Before deciding to use T2D, we first examined the feasibility of using pre-built game making tools, including RPG Maker and Game Maker.

RPG Maker is a closed-source RPG development tool that uses a large system of menus to produce games. Although the tool would have been adequate for producing just a game without our learning tools implemented, it was not feasible for use in our project. Since the software was closed source, there was no way of manipulating the engine in such a way to keep track of the statistics we would need. This lack of a tracking ability would have defeated the purpose of the project, and we therefore decided not to use RPG Maker as an option [RPG]

Game Maker is a closed-source game development tool, similar to RPG Maker. The difference between RPG Maker and Game Maker is that the types of games that can be produced with Game Maker are in a much broader category than RPG Maker. Game

Maker also had a higher level of programmer control. Unfortunately, since Game Maker's engine is also closed source, we ran into the same problem we ran into with RPG Maker: we were unable to track the statistics we needed for use with the tools we were planning on implementing in the game. We then decided that Game Maker was not going to work as our development tool either [GAMEMAKER].

Although it seemed feasible to cut down on development time by using a pre-built game making tool such as Game Maker or RPGMaker, these tools could not provide the level of control we required. We felt that using a 3rd party pre-built engine (meaning an engine that is ready to use and available for purchase), despite requiring extra time to produce art and design levels, would make the tool itself easier to make. We looked at several game engines, but quickly realized that most options exceeded our budget. The complexity of these engines was also much more than we needed, and the learning curve was far steeper. We chose T2D because we already had some experience programming and developing with it, the source was available as a development tool through the WPI Game Development Club and, although all of our code was done on the scripting level, the source code was available for editing.

T2D is a game engine developed by GarageGames that provides rigid-body physics, a particle effects editor, sprite based animations, TorqueScript, static and animated tiles and a collision detection system [GG1]. T2D was developed as a subset of GarageGames' flagship product, the Torque Game Engine (TGE). T2D provides a platform to make sprite-based games using only a scripting language. It has a built-in GUI editor and a tilemap editor, which allows developers to immediately see the changes they make. The scripting language itself is relatively easy to learn and script with. The script syntax and semantics were reminiscent of C#, which made editing and using naming conventions straightforward.

4.4 Learning Tools

An empirical analysis was done on a range of games to create a list of common methods for game learning, which can be seen in Appendix E. The most common methods, on-screen help and controls map, were implemented, as well as one we had not seen: audio cues. By providing this method, researchers may be able to compare the prevalent methods with one that is seldom used. One method that was not included was a tutorial, which was found in 16 of the 57 games. This decision was made because the creation of a tutorial requires significant additional development of the game world, as well as new user interfaces. Additionally, tutorials provide a sample of all the game’s mechanics, some of which may not be implemented by the end of this project.

5. Evaluation and Testing

The Double Vision tool provides a foundation for future projects to expand on. The tool was created with this in mind, so some of the functionality of the game is primitive. For instance, the artwork and level design were not thoroughly expanded, since gathering or creating those materials is very time consuming. Instead, since this project may expand in many directions, the tool was created to be strong programmatically, in terms of the game the user plays, the learning tools, and the data gathering. Because there is a strong technical base, the tool can grow in many ways: furthering the development of the game, to make it more enjoyable or challenging; expanding on the tool, providing more opportunities to teach the user how to play; or artistically, by creating new graphics, levels, and story.

It was important to create a game that a user would be able to play without encountering disruptive elements such as gameplay bugs, flaws in the levels, or program instability. Eliminating bugs in the gameplay was done during playtesting. That is, for each new feature added, that feature would be repeatedly tested. At the same time, testing was done on existing capabilities, to ensure this new element did not adversely affect any other features. So the entire tool was tested each time a feature was added. Having done this, a user should not encounter anything that was not planned. Level design flaws were eliminated by repeated testing, and by good level making strategies, such as placing all the same objects at once, and putting collidable objects on their own layer. By using an established, pre-built game engine, no program or system crashes were encountered while testing the tool.

Data management is a critical element of the learning tools themselves, so it was important to have a reliable logging system in place, so that future groups can easily deal with the information gathered by the tool. This was done by developing the current logging system, which outputs statistical information in a human-readable format. This provides future groups with an explicit definition of what each variable in the log represents. If a different format, such as the Comma Separated Value file format, would be easier when processing the data, the group only needs to modify the strings that are sent to the log output.

Testing of the tool was done each time a new feature was added, to ensure the usability of each component. Creating a strong feature set was important to the tool, more so than the overall cohesiveness of the game. This is the reason why testing was done modularly during development, rather than on the entire tool upon completion. The result is a tool with strong individual elements (such as session logging, combat, and help screens), but has shortcomings with regards to the game contained therein (little story, easy gameplay, no replayability). These problematic aspects, however, were not part of the initial project goals, and can be corrected by a future project group.

6. Future Work

Double Vision was designed to be a tool that could easily be extended by future groups. Therefore, we tried to build the tool as a core piece of software. We believe that most of the critical elements have been implemented. Standards for things like player controls, help logging, and level design have all been established. The format and examples will all be available to a follow-up group, which should make their tasks much easier.

There are several opportunities to expand Double Vision to make it a stronger tool. The first thing would be to make it more visually appealing. Currently, about half of the art and sounds were made by us, and half of them were taken from freely available sources. Unfortunately, this practice resulted in stark style differences between the pieces. While the focus of the tool is tracking the learning process and the aesthetics of the game, updating the visual and auditory qualities would make for a more familiar gaming environment for the user.

Another option, which may tie in with the first, would be to extend the length of the game. Since art tiles can be substituted freely, a half dozen more levels could be added to the game using temporary art. When art becomes available, give the new files the same name as the old ones and place them in the appropriate directory. T2D will automatically resize the files if they do not have the same dimensions. Additionally, the game itself could be made more challenging by providing more enemies, item pickups, more challenging fights and so on.

Finally, an extension of the tool itself is a possibility. Expansion of the current types of learning tools is possible, such as providing various pop-up boxes with different visual appeal. Also, new learning tools entirely could be added. Our hope was to include a tutorial, or a non-interactive video that would be used to demonstrate gameplay. While we were unable to include either of them, we feel they are important to consider.

7. Conclusion

We developed this project because there is a lack of consensus in the game development industry regarding the best method for teaching players how to play games. There are many ways to teach game concepts to a player, but development time can limit which methods are implemented. Therefore, it is necessary to understand the quickest, most enjoyable, and most effective methods of learning how to play, while ensuring the knowledge is retained during gameplay. One potential problem is that different game genres might require different methods of learning the game. For example, pop-up help could be too distracting for racing games, and video tutorials might give away too much information in puzzle games. Our tool could be used to determine which learning methods are best for each genre by creating a different game in a different genre, and testing which learning method works best.

Our plan was to create the framework for an application that could be used to empirically determine the best way that people learn to play games. We first decided to make the tool as self-contained as possible, so that the tool could be used immediately, without requiring any additions by the research group. Our second objective was to create a tool that could be changed and adapted to fit the needs of the research group that would use it. To accomplish the first goal, we presented the tool as a complete game, one that contains thorough logging of player-requested help. Our game does not have a pre-determined end state, but otherwise fulfills our first goal. The second goal was achieved by writing the scripting code in a modular manner, such that additions or modifications to the script files could be done logically. For example, we encapsulated as much of the code that deals with the combat system within the script file “combat.cs.” Additionally, we have provided several guides to expanding the tool, which can be found in the appendices. With these factors in mind, we feel that we succeeded in our initial plan and have provided the groundwork for future groups to extend.

8. References

[GAME INSTRUCTIONS] Chen, Jessie Y. C., "Utility of Game Instructions". United States Army Research Institute for the Behavioral and Social Sciences: April 2003.

[GG1] Garage Games. "Torque 2D Game Builder". <Available> http://www.garagegames.com/products/62, March 8, 2006.

[LIBERTY] Oliver, Martin; Pelletier, Caroline, "The things we learned on Liberty Island: designing games to help people become competent game players". London Knowledge Lab, Institute of Education, University of London: 2005

[GAMEMAKER] Overmars, Mark. "GameMaker". <Available> http://www.gamemaker.nl/, March 3, 2006.

[TORTOISE] "Tortoise SVN". <Available> http://tortoisesvn.sourceforge.net/, January 15, 2006.

[SVN] "Subversion Control". <Available> http://subversion.tigris.org/, March 8, 2006.

[RPG] "The RPG Maker Resource Kit". <Available> http://www.crankeye.com/index.php, February 2, 2006.

9. Art Resources

Artwork was borrowed from the following websites. Either permission was given by the original author, or the artwork was provided for free use:

http://olls.impressur.com/battle_charset.html

http://www.rmxp.net/resources/rmxp/Graphics/

http://www.gamingw.net/

Appendix A - Manual for Use of Double Vision

Expanding upon the Double Vision tool is very straightforward, once a few things have been established. Without greatly modifying any prewritten script, the two things that can be edited are the learning methods implemented and the game itself.

A-1. Using the Tool

Double Vision has been packaged such that it is ready to use to test how people learn to play video games. As presented, the tool needs only be extracted to a directory on the target machine. Development was done exclusively on Windows, but it should run on Linux and Mac. From here, any of the batch files can be run. Each of these starts the game with a different set of learning tools activated. The following is a list of those batch files:

Filename	**What it does**
DV-all.bat	Starts the game with all learning tools enabled.
DV-audio.bat	Starts the game with only audio tools enabled.
DV-controls.bat	Starts the game with only the controls map enabled.
DV-controls_audio.bat	Starts the game with both the controls map and audio tools.
DV-none.bat	Starts the game with no learning tools enabled.
DV-popups.bat	Starts the game with only on-screen help enabled.

The user can be given knowledge of which learning tools will be available, or it may be kept a secret. You should, however, let them be aware that pressing ‘9’ (or whichever button you may change it to) will bring up any and all help that is available to them. The exception to this is if the Controls Map is available to the user. We have chosen to have this learning tool bound to a separate key (Backspace), since the Controls Map overlays everything, potentially skewing the time spent looking at Pop-Up Help. Therefore, if the Controls Map is available, let the user know the key to which it is bound (default is 'backspace'). Pressing 'Esc' will end the session, however premature termination by the user will likely invalidate any data gathering methods you have set up. This functionality may be removed or the control key may be changed if the research group wishes to do so.

You might also consider giving the user the Player Manual, included in Appendix C. This contains background story, as well as a list of items in the game and some enemies. This is meant to be a supplement to the user, to give them a feel for the game before they play it. We have intentionally left the list of controls out of the document, because we felt that providing the user with the controls ahead of time may skew the use of the learning tools in-game. Additionally, it is impossible to record how often or how long a user refers to the Player Manual with our tool.

Once the user understands how the game operates, and where they can get help, the player will be on their own. The game has no defined end state, but once the player confronts the final enemy (located at the water fountain at the end of the second level), the game will provide no new experiences, and at that point, the game can be assumed to be over. If the player defeats the enemy, he'll be returned to the game, at which time you should take control of the system again. Press Esc, and the game will automatically end. The game then displays the game over screen and writes any gathered information to the log file. If the player's character dies at any point, the game will immediately end. Once at the game over screen, you should then manually close the program out.

The logged information can be examined at any time. The log is located in /mqp/output.txt located in the in the directory that the tool was installed. Assuming the tool is terminated by pressing 'Esc' (rather than by a program or system crash), each running of the game will produce a new section of data that is appended to the end of the log. As delivered, the log outputs which help tools were called for (and for how long, if appropriate), followed by a summary of how much each type of help was used, then which tools were enabled and how much health the player had at the end of the game. This information may not be what is most relevant to your project, so we encourage you to look at the logHelp() function in player.cs. Refer to Appendix D for a sample log file.

A-2. Changing and Adding Tools

To create a version of the game with a combination of learning tools available, create a batch file containing "T2D.exe -mod mqp" and add as many of the following flags to the end of the string:

Flag	Help Option(s) Enabled
-DVpopup	Popup tooltips
-DVaudio	Voiceover help
-DVControls	Controls map
-DVall	All

To add additional types of learning tools, such as a tutorial or video of gameplay, would likely require intensive additions to the code structure. If you wish to pursue this course, please read the Developer's document. However, adding more occurrences of currently existing tools is much easier. To do so, look in the getHelp() function in player.cs, and add or change the popup or audio calls.

A-3. Expanding the Game

All the artwork used in the game is located in the file /mqp/client/datablocks.cs. To change any of the art in the game, find the corresponding .png file located in the mqp/client/images directory and replace it with the new art. Torque 2D is good at scaling images, but you will get the best result if you use images of the same dimensions as the originals. If you change any character images that consist of character maps, you must either maintain the same image ratio, or change the cell width/height in datablocks.cs.

Adding additional maps requires a little more work. Since a licensed copy of Torque 2D is necessary for the creation and implementation of maps, further information can be found in the Developer's Document (Appendix B).

Appendix B - Manual for Expansion of Double Vision

This document provides an overview of the technical aspect of the Double Vision tool. It explains the purpose of all script files for the game, as well as a stronger explanation of how to create additional tilemaps for the game. The later is provided because at the time of writing, there exists no thorough tutorial for the built-in Tile Editor. This document assumes the reader has a valid developer's license to Torque 2D, and a working knowledge of TorqueScript, including having gone through the tutorials and looked at the reference documentation. Once you have your account for GarageGames and have purchased Torque 2D, the download link can be found at http://www.garagegames.com/myAccount/. These files are included in the SDK bundled with Torque 2D. The GarageGames developer forums are also very useful, and can be found at http://www.garagegames.com/mg/forums/result.area.php.

The first thing that should be done is to look at all the .cs files contained within the mqp/ directory. The two .cs files in this directory should not need to be edited, as one sets preferences and the other calls the startup scripts. In the mqp/client/ directory are all of the script files that should need to be edited. Mqp/client/effects contain particle effects held over from the previous game this was built on, and are not used in Double Vision. These files remain to provide future groups with examples of graphical effects that could be added to the game. Mqp/client/help contain help files that can be accessed through the Torque2D engine. Future groups may wish to take advantage of this built-in help, but it was not helpful to us. Mqp/client/images contain many of the game images, as well as some that are not used in the game. Most of these images were taken from freely available libraries, which are listed in section 8, while some of the images were made by us. However, all of the files in mqp/client/images/custom were created by us. Mqp/client/maps contain all the tilemap files for the game, as well as some others that were created during the development process. Mqp/client/sounds are all sound files for the game, with appropriate sources listed in Audio Sources.txt.

The following is a rundown of the included files in mqp/client/:

audioDatablocks.cs
Creates the datablocks for all the audio files used in the game. More can be freely added by following the examples given. Voiceover help can then be called using doVoice(), passing a datablock as an argument.

client.cs
Initializes the layer and group constants. Also executes all the script files for the game. The order that they are initialized is important, because datablocks must be created before they can be used. Also contains code that handles what happens when the game ends, specifically, the writing to the log.

datablocks.cs
Like the audio datablocks, these create the references to all images that are used in the game. Datablocks in mode = full; use the entire image, but those in mode = cell; are divided up into frames to be used as animations.

DoubleVisionScreen.gui
The main GUI for the game. It can be edited here, or through the built-in GUI editor in T2D.exe.

npc.cs
Creates the NPC and places him on the screen. Each individual NPC is created in sprites.cs, where the image, size, position, and chat are specified for each.

player.cs
The largest body of script, this is where the majority of commands are called. Player.cs contains all actions associated with the player, such as all the controls scheme, interaction with NPCs, moving the player when he reaches a boundary, and collisions with other objects. Also, here's where much of the learning tool related code is written, such as creating all the help on screen (controls map, popup help, audio) and logging the help to a file.

prefs.cs
Torque 2D preferences. These are all set to default settings.

scene.cs
Sets the camera size and position, as well as which map to draw. Objects on tile-layer 0 are assumed to be walked on, while those on tile-layer 1 are assumed by be collidable (Collisions MUST be set to ACTIVE on EACH TILE). Also contains the fading to and from black code.

splashScreen.gui
Displays GarageGame's logo before our game's logo. By agreeing to the EULA for Torque2D, this must remain in any game made with the T2D engine.

sprites.cs
Creates all the sprites for each map. A sprite is either a combat, an NPC, or some transition (called a Town). Combats must first be checked if they've already been encountered, so that the same battle does not appear twice. Then they must be given a position, which enemy is to be fought, what the tilemap will be, the name of the fight, and the area that the fight will be located in. NPCs are given a position, a name, an image map, what they'll say (delimited by '//'), and what they will say after their chat lines have been exhausted. A town is given a position, a position where the player will be transported to, a name, the camera coordinates where the player will move to, and the name of the map the player will be in. This is done so a town will be able to move a player within the same map, or to a new map (like between the woods and the city). Reset the size of $town immediately after making the town, to make a larger area the player can collide with.

t2d_exitScreen.gui
Displays the game over screen. Any click will then close the program.

t2d_launchMenu_DoubleVision.gui
Displays the title screen. Any click will start the game.

tools.cs
Initializes the learning tools. Also creates and removes the popup help boxes, plays the audio help, and has a timer function.

town.cs
Creates the town sprite.

An interesting fact about scripting with Torque2D is that variables and functions can be called from any of the scripts that are executed at runtime in *client.cs*. This means that functions can be placed in whichever script file most logically would use them.

B-1. Creating New Maps

To create a new map for Double Vision, first load up the game and gain control of the character. This loads T2D.exe with all the appropriate paths and datablocks for the game. Now enter the tile editor by pressing F12. Create a new map, with two tile layers, each 32 tiles wide by 24 tiles high, and 40 by 40 pixels. Larger maps are possible, as long as the programmer keeps track of where the camera position should be when the player moves to a different area.

Go to layer 0, and select a tile from the Image Library pull down menu. Tiles on layer 0 are assumed to be the background, so the player will be able to walk on them. Then select layer 1. It helps to select View-> Up to Current Layer when you're editing, so you can see what the final image will be. When you're placing objects in layer 1, they will appear above layer 0, but will only be collided with if you've checked the box next to the pull down menu before you added the tiles. To bring up more information about a tile, CTRL + click on the tile, and you can set collisions active, or change the collision polygon. Right clicking will remove the tile.

When you're completed, save the tilemap with a name and a .map extension. Then move the map from tileeditor/client/maps/ to mqp/client/maps/. Now you can add a town from an existing map, and link it to your new map. Make sure to move the camera to the correct area: (0 , 0) if the new area is the top left corner, (0 , 640) if it is one screen to the right of the top left, (480 , 0) if it is one screen down from the top left, etc.

Appendix C - Player's Manual for Double Vision

Welcome to Double Vision, a game whose story takes you to the very heart of human corruption, whose puzzles will strain the mind, and whose fierce battles will rattle the very core of your being. You will enter a world of the near future, in a land not far from here, where science is advancing faster than humanity is ready to accept it. Cloning has become cheap and efficient, despite government regulations preventing it. Nanomachines are able to restore the human body to optimum health. Scientists are able to extract a person's thoughts and memories through minimally invasive brain surgery. Pharmaceutical companies are fighting to become the sole provider to the masses of these amazing technological breakthroughs. Corporations vying for government grants are sabotaging one another. The world is either on the brink of transcendence, or on the brink of destruction.

In Double Vision, you play as Dr. Wysooki, a renowned neurological surgeon who has been working with the GeoComm Corporation for 30 years. He's been working in a classified section of the Brain Research wing for over a decade, and is on the verge of a breakthrough. One night, as his wife is once again complaining of how rarely the children see him, he decides to bring the whole family in to work. They wouldn't be allowed in the restricted area, but he's been with the organization long enough, that they let him do just about whatever he pleases. That very next day as the Wysooki family relaxes in the waiting area, GeoComm comes under attack, and the entire facility is set ablaze! Dr. Wysooki barely saves himself as an explosion rocks the building and levels it to the ground! As he crawls from the wreckage, Dr. Wysooki vows to find out who destroyed his work and his family.

Items:

Apples – Sweet fruit that can help restore one's health, allowing them to continue fighting. Taste particularly good in pies. Can be eaten whether you're hungry or not.

Nanomachines – These helpful robots are found in small, single use quantities. When swallowed, they search out imperfections in the body, such as viruses, broken bones and even wounds. Fully restores a person to their maximum potential.

Nitroglycerin (Nitro) – Found in small, portable vials, this dangerous substance can cause a horrifying explosion or can be used to decrease blood pressure.

Cell Phone – Dr. Wysooki always used it to keep in touch with his family, so he doesn't have much use for it anymore. Still, it may come in handy.

Enemies:

Bears - The mighty grizzly, *Ursus arctos horribilis*, should be avoided at all costs. Should one be encountered, play dead! If the creature still persists, only then should it be attacked.

Wolves – A frightening creature, wolves often hunt in packs. They are a formidable opponent, until you acquire a weapon.

Renegades – Sometimes found in cities, these opponents lie in wait until a victim walks by. Renegades are brutal and cunning, and dealing with one is often life threatening.

Getting Help:

Press 9 to bring up any available context sensitive help. This can be done at any time.

Appendix D - Sample Log Output

The following is what the log might look like after the game is run once:

```
---------------User Help Statistics---------------
Controls Map seen for (ms): 200
General Popup seen for (ms): 2400
General Audio heard
Controls Map seen for (ms): 1500
General Popup seen for (ms): 900
General Audio heard
Controls Map seen for (ms): 500
General Popup seen for (ms): 1100
General Audio heard
Controls Map seen for (ms): 2000
General Popup seen for (ms): 1400
General Audio heard
Controls Map seen for (ms): 2900
NPC Popup seen for (ms): 2900
Controls Map seen for (ms): 800
General Popup seen for (ms): 200
General Audio heard
Controls Map seen for (ms): 1600
General Popup seen for (ms): 1000
General Audio heard
Controls Map seen for (ms): 2700
NPC Popup seen for (ms): 2700
Controls Map seen for (ms): 1800
NPC Popup seen for (ms): 1800
Controls Map seen for (ms): 200
General Popup seen for (ms): 600
General Audio heard
Controls Map seen for (ms): 2400
General Popup seen for (ms): 1800
General Audio heard

Total combat popups seen: 0 for a total of (ms): 0
Total NPC popups seen: 3 for a total of (ms): 7400
Total general popups seen: 8 for a total of (ms): 9400
Total popups: 11, seen for (ms): 16800

Controls Map seen: 11 times for a total of (ms): 16600

Audio clips heard: 8 times.

All tools on? 1  Popup tools on? 0  Audio tools on? 0  Controls map on?
0
Player health ended at: 100
```

Each time the game is run a sequence of information such as the sample log above will be appended to the end of output.txt located in the /mqp/ directory. The first set of information is a list of the types of help the user requested listed in the order that the user called them. Where applicable, the length of time that the user looked at each piece of help is also indicated. The next section is a summary of the popup help tools. Currently, there are three categories of popup help, determined by when they appear to the player: during combat, while speaking with a non-player character (NPC), and any other time. The total number of requests for each type of popup is listed, as well as the total time they appeared on screen.

The next two sections of the log summarize the controls map and audio tips in the same manner. Finally, the help types that were enabled are listed, as well as the final player health. This last item is the player's ending health: an example of what a research group could use to test how well the player understood how to play the game. Other information that might be gathered here is: total time played, character level, or enemies defeated.

Appendix E - Video Game List

Title	Platform	Genre	Learning Method
Abuse	PC	Action	On-screen Help
Amplitude	PS2	Music	Tutorial
Anarchy Online	PC	MMORPG	Tutorial
Avernum	PC	RPG	On-screen Help
Baiten Kaitos	GC	RPG	On-screen Help
Betrayal at Krondor	PC	RPG	None
Beyond Good and Evil	Xbox	Action	On-screen Help
BreakQuest	PC	Arcade	None
Bridge Builder	PC	Puzzle	None
Burnout 3	Xbox	Sports	Video
Castle of the Winds 1/2	PC	RPG	None
DDR	Xbox	Music	Tutorial
Dead or Alive 3	Xbox	Fighting	Tutorial
Devil May Cry 1 thru 3	PS2	Action	On-screen Help
Diablo 1/2	PC	Action	Controls
Doom 3	PC	FPS	On-screen Help
Dungeon Siege 1/2	PC	Action	Tutorial
Fallout 1/2	PC	RPG	Tutorial
Final Fantasy I thru IX	PS1	RPG	On-screen Help
Final Fantasy X and X-2	PS2	RPG	On-screen Help
Final Fantasy: Crystal Chronicles	GC	RPG	Tutorial
Frequency	PS2	Music	Tutorial
Fuzion Frenzy	Xbox	Party	Controls
God of War	PS2	Action	On-screen Help
GTA 1	PC	Arcade	Controls
Gunbound	PC	Strategy	None
Half-Life 1/2	PC	FPS	Tutorial

Hunter: The Reckoning	Xbox	Action	On-screen Help
Jazz Jackrabbit 1/2	PC	Platform	On-screen Help
Jet Set Radio Future	Xbox	Sports	Tutorial
KoToR 1/2	Xbox	RPG	On-screen Help
Legend of Zelda	GC	Action	On-screen Help
LotR: The Third Age	Xbox	RPG	On-screen Help
Mechassault	Xbox	Action	Controls
Meteos	DS	Puzzle	Tutorial
Metroid Prime	GC	FPS	Unlocked Abilities
Metroid Prime 2: Echoes	GC	FPS	Unlocked Abilities
Myst 1-4	PC	Adventure	None
Neverwinter Nights	PC	RPG	Tutorial
Ninja Gaiden	Xbox	Action	On-screen Help
Oddworld: Munch's Oddysee	Xbox	Puzzle	Controls
Phantom Dust	Xbox	Action	Unlocked Abilities
Prince of Persia 1/2	Xbox	Puzzle	On-screen Help
Project Eden	PS2	Puzzle	On-screen Help
Psychonauts	Xbox	Puzzle	Tutorial
Sega GT	Xbox	Racing	Controls
Soul Calibur 2	GC	Fighting	On-screen Help
Spider-man 2	Xbox	Action	On-screen Help
Splinter Cell: Chaos Theory	PC	Stealth	Tutorial
Star Wars: Jedi Starfighter	Xbox	Arcade	Tutorial
Star Wars: The Clone Wars	Xbox	Action	Controls
Syberia 1/2	PC	Adventure	None
Tales of Symphonia	GC	RPG	On-screen Help
Teenage Mutant Ninja Turtles	Xbox	Action	Controls
Tetris Worlds	Xbox	Puzzle	None
Wreckless	Xbox	Racing	Controls
X-Men Legends	GC	Action	Tutorial