Air Hockey User Manual

Luis Almanzar, Gunnar Wambaugh, Daniel Lopez

November 2015

Introduction

This is an Air Hockey Table, product of a group project. This program is the 9th Assignment from UNR course CS 480, which had the purpose of teaching students how to work with Bullet, a physics engine. The program will be written with a C++ implementation of OpenGL alongside other third-party open source software, and compiled on Ubuntu 15.04, a Debian distribution.

This User Guide will include the functionality of the game, which can be found under the 'Quick Start' section of 'README.md', which will also be included in this master Manual. This Guide will also include some extra commentary on the code documentation and the creation of a new element that can aid multiple programmers in efficiently creating elements with Assimp, Magick++, and Bullet; a GLD.

Dependent Libraries

To give credit where credit is due, you should know that this project was done with the help of third party, open-source software. To install these, if they haven't already been installed, is to include the following:

1. Assimp - The model loader. The instruction to install is 'sudo apt-get install libassimp-dev'
2. Bullet - The Physics engine. The instruction to install is 'sudo apt-get install libbullet-dev'
3. Magick++ - The Image Loader for C++. The instruction to install is 'sudo apt-get install libmagick++-dev'
4. OpenGL - The graphics library upon which the graphics library will talk to.

Gameplay

1. In figure 1, we can see a figure of one of the initial views of the screen when the program is running. From this screen, the user has the option to play the game, view the top ten scoreboard, or end the game.
2. In figure 2, we can one of the screens from when the game is in play, and the camera has been tilted a bit to the left. From this screen we can see that there's a hockey table in a room with two paddles. The game will default to start with two players, but will be able to switch to AI upon right clicking and toggling the AI for Player 2. Note that there is indications on the top left and the top right corners that indicate the "Health" (score) of the players and specify the controls that the player needs to manage the game.
3. In figure 3, we can see another game play screenshot, except this one includes an example of the powerup once activated. When the puck hits the Mystery Box in the center, it will activate a powerup to appear and float about the table. The powerups change attributes of the game such as change the friction, restitution, gravity, and the number of the pucks during the game.
4. In figure 4, we can see an example of one of the powerup's features. There are multiple pucks on the board at the same time.
5. In figure 5, we can see the screen that appears when one of the two players win. Once one of the two players wins, we have a screen that says who won and prompts the user to quit the game.

Known Issues

1. **Collision Detection**

   Currently, the game is not using callbacks for collisions. Instead, the game is checking the x and z positions of the goal and the pucks, to see if a puck has scored. If we had more time we would have liked to add have spent more time learning bullet. Originally, we were implementing collision detection,

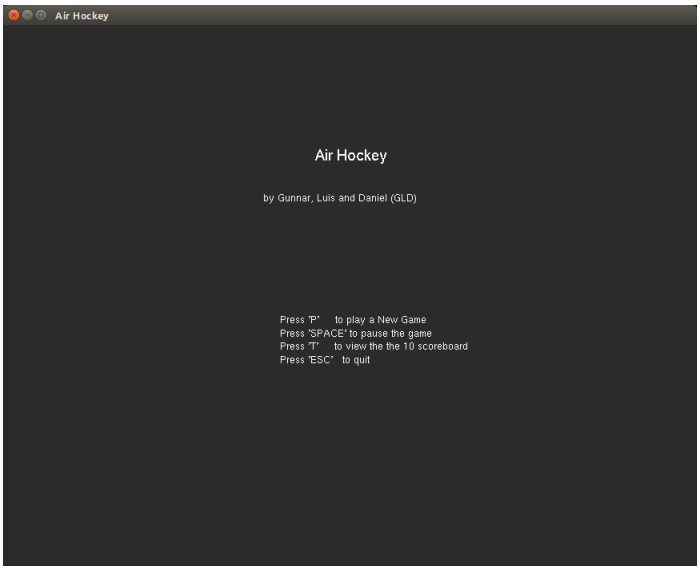

Figure 1: Initial Screen

but we were not able to find a proper solution in time. In order to make our game function in time, we had to handle the issue in a not so friendly way, and apply this method of checking "collision".

2. **Memory Leaks**

   The game is full of them. We spent most of our time designing and building our project, while trying to add new features. Thus, we did not have enough time within our time constraints to deallocate our data types. If we could go back, we should have updated our destructor whenever we created a new dynamic object. Then, we would not have a large memory leak problem. Thankfully, we have not had any blue screen problems as of yet.

3. **Mouse Tracing**

   The game was originally designed to use mouse tracing for the mouse controls. However, due to time constraints, we were not able to allocate the time to perfect them. Instead, the mouse controls are determined by the mouse's current x, y positions on the screen. We map the screen into sixteen regions that determine which direction the paddle will move in. For instance, if the map is in the upper right hand corner of the screen, the paddle will move toward that point relative to the camera angle. If we had known that the ray tracing was so difficult we could have saved time doing our plan B first. This would have allowed us to add more features that would have added to our gameplay and overall quality of a game.

4. **Powerup Debugging**

   We wanted to take our game to the next level by adding dynamic power ups that can change your puck, score, table, and paddle properties. However, again, since having strict time constraints we could not finish perfecting these powerups. Currently, we have problems with the animation of when a new powerup is added. Also, we have a known issue of spawning multiple power effects off of the same powerup spawner.

5. **Artificial Intelligence**

   We have implemented a very basic AI, almost brain dead. If we had the time we would have liked to make it a bit more competent, with the possibility of strategy. Currently our AI only moves toward the puck.

Figure 2: Normal Game Play

Figure 3: Powerup Game Play

Figure 4: MultiPuck Example

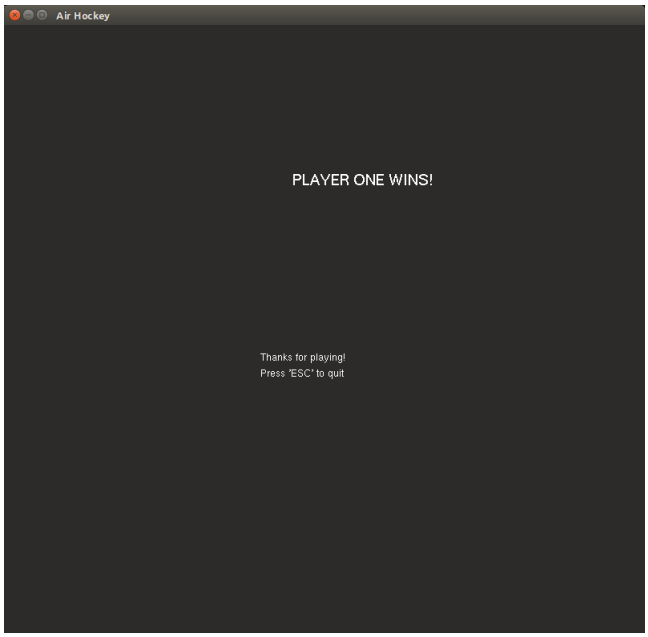

Figure 5: MultiPuck Example