**SOFTWARE DOCUMENTATION**

**FOR**

**THE DEVELOPMENT OF THE PING-PONG (SINGLE PLAYER) GAME**

***BEING DEVELOPED AS PART OF THE REQUIREMENTS FOR CSC 514***

***(INTRODUCTION TO COMPUTER GRAPHICS)***

**INTRODUCTION**

Ping-pong is a game similar to the table tennis game. It involves hitting a light plastic ball across a table (fashioned like a mini tennis court) by racquets or paddles or bats as the case maybe. It could be played as a one-player game or a two- player game. However, this software is a one-player ping pong game, by one player it denotes that it is played against the computer.

This software is developed using C programming language with OpenGL, it can run on a LINUX operating system but not Windows operating system, this is due to the fact that C programming language is not portable across platform. The OpenGL Utilities (GLU) library and the OpenGL Utilities Toolkit (GLUT) are utilized.

**DESCRIPTION OF THE FIELD OF PLAY**

The field of play simply refers to the environment in which the ping pong game is played. For the purpose of this documentation, it is referred to as the draw board and it is of size 400x400 pixels. The draw board consists of the table, the ball, racquets, paddles or bats and the score panel.

**The Table**

It consists of two parts; the computer’s side which is coloured dark gray and the player’s side which is coloured orange.

**The Ball**

This is a light solid or hollow object that is round in shape. There is just one ball in the field of play and the player aims at controlling her bat to hit the ball since a miss indicates the loss or end of the game.

**The Bats**

There exist just two bats for this game; one is used by the player and the other by the computer, the colour of the computer's bat is red while that of the human player is gray. For every hit made, the colour of the player's bat changes.

**The Score Panel**

The score panel is the board that pops up when the game is lost by the player. It displays the total number of points made by the player. It consists of a base rectangle for displaying the points made, and two other containing rectangles labeled “play again” and “exit” respectively.

**DESCRIPTION AND OPERATION OF THE GAME**

The game is developed in such a way that it is played against the computer by just one player. The player is to ensure that the ball is hit against his own bat at every pass made towards him. For every hit made, he gets a point and after every hit made, the point increases. After every ten (10) points attained, the game moves to the next stage automatically and this is indicated by the increase in the speed of the game. This game is developed in a way such that at a point the user is unable to keep up with the increase in speed, then at that point the ball misses the bat and he looses the game. The time for ball movement is measured in milliseconds and the ball moves in the vertical and horizontal directions simultaneously. To pause the game, the player needs to press the space bar key on the keyboard as well as to continue playing after pausing the game. If the game has ended, to exit the user either clicks on the ‘exit’ button on the score panel or presses the letter ‘E’ of the alphabet on the keyboard. Also to move the bat, the player uses the right and left arrow keys.

**CODE DESCRIPTION**

System call back and user call back mechanisms are used in the program.

Methods in the program are:

* *adjustComp(),*
* *animate(),*
* *ballCollidedWithRect(),*
* *continueAnimation(),*
* *displayFcn(),*
* *drawBoard(),*
* *faster(),*
* *getStringRepresentation(),*
* *getTimeInMillis(),*
* *init(),*
* *keyFcn(),*
* *killBackgroundMusic(),*
* *main(),*
* *onMouseClicked(),*
* *paintball(),*
* *paintComp(),*
* *paintRect(),*
* *resetGame(),*
* *showScore(),*
* *specialKeyFcn(),*
* *startBackgroundMusic(),*
* *stopAnimation(),*
* *winReshapeFcn(),*
* *writeText(),*

**Init()**

This is the method in which all the codes to initialise the program's environment is bundled up within. *verticalStartTimeForAnimation* takes the current time when the game starts and this is done by the g*etTimeInMillis()* method. The time taken to move the ball towards vertical and horizontal positions, all in milliseconds, is also handled here.

Additionally, this routine sets up:

1. The time taken to move the bat vertically and horizontally;
2. The position of the player’s bat and the computer’s bat;
3. The initial positions of the ball and the bats (for both the player and computer);
4. The sizes of the bats.

**rect.HitColumn()**

This is responsible for the change in colour when the human bat hits the ball.

**startBackgroundMusic()**

This is responsible for calling the background music to start playing in the course of the game.

**stopAnimation()**

This enables the player to pause the game using appropriate command.

**animate()**

This is the most frequently called method and it uses *glutIdleFunc()* to carry out its task.

It is has the following responsibilities:

1. Trigger OpenGL to update, refresh or repaint the screen.
2. Housekeeping the current state of the game.
3. Ensure that sound is produced when the ball hits either the player or computer's bat.
4. Movement of the ball and the bats at their current location during or after repainting the screen
5. Increasing the player's points and changing the colour of the player's bat at every hit.
6. Manages the collision of the ball with the bats.

**paintComp()**

It is called during animation to repaint or refresh the display to account for the new position of the computer's bat.

**paintBall()**

This is called during animation to repaint the ball in its new position.

**faster()**

This is the method responsible for the increase in the movement of the ball after the hit is a multiple of ten(10.). The time taken for the vertical ball to get to the opposite end reduces by one-tenth of a second.

**ballCollidedWithRect()**

This is responsible for checking the conditions for the hit of the ball. It checks for the necessary and sufficient conditions required to ensure that a collision has actually occurred. For the bat to make a hit with the ball, the ball and the bat has got to be collinear( necessary) and the top of the player's racquet must be in contact with the bottom of the ball (sufficient).

**showScore()**

This triggers animation to stop and then displays the player's score when the player misses a hit.

**glutPostRedisplay()**

This is responsible for requesting that a repaint of the display be made in order to reflect its current state. This happens when the state of the game changes.

**displayFcn()**

This clears the display and paints its current state. It calls *drawboard()* and also checks for state of the game (i.e. paused or playing). If the result of the test revealed that the game ended, then the score panel is drawn.

**drawBoard()**

This is responsible for setting the colour for the human and computer fields i.e it draws the field of the game.

**gameEnded()**

This is responsible for indicating the end of the game when the human player looses.

**continueAnimation()**

This is responsible for the housekeeping routine work as well as keeping the current track of the some state variables. It also sets the animate() method to animate.