NOTE TO USERS

Implementation of 3D Snooker Simulator

-- Foundation classes development

Bing Zhu

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

April 2005

Your file Votre référence
ISBN: 0-494-04462-4
Our file Notre référence
ISBN: 0-494-04462-4

Abstract

The implementation of 3D Snooker Simulator – Foundation classes development project is one part of the 3D Snooker Simulator project to support the 3D implementation of snooker simulator. It mainly focuses on developing a set of foundation classes to simplify the user interface development and 3D graphic development for porting the existing 2D Snooker Simulator from 2D to 3D by using OpenGL and ANSI C++.

In the report, a brief introduction is given to the current situation of game programming, game programming using OpenGL and the features of the 2D Snooker Simulator. In the last decade, 3D graphic and user interface has become the main stream of computer-based game development and OpenGL is the main graphics-programming library, software interface to graphic hardware, and a "standard" for 3D graphics.

The target of the project is to implement the Snooker Simulator as a platform-independent 3D game with user-friendly, game-like User interface while preserving all the functionalities of the 2D version. The development of foundation classes makes it easy to implement and maintain. The author describes how to reach these goals from designing, 3D object modeling and implementation. Meanwhile, lots of implementation problems and solutions are discussed in the report.

Finally, the author points out that the current 3D Snooker Simulator has achieved most of the design goals, but still has space for improvement.

Table of Contents

List of Figures

1. Introduction

The 3D Snooker Simulator (abbreviated as the 3D version) is the successor version of the 2D Snooker Simulator (abbreviated as the 2D version) that was developed by Professor Peter Grogono. The 3D version reserves all the functionalities of the 2D version, including four options of playing mode – manual, track, advise and auto, time testing, demo, parameter setting for game environment and players.

The 2D version is platform-dependent. It has a 2D user interface which is based on Fastgraph library©™ and win32 GDI. One main task of this project is to implement a user-friendly, game-like 3D user interface. To make it easy to be ported to other platforms in the future, the whole project is implemented with OpenGL and ANSI C++ only.

The 2D version has already embedded a good mechanism to calculate the ball's movements according to the environment properties, such as ball-ball collision, ball-cushion collision and hitting position, speed and direction. It also supplies an optimized algorithm to select the best action for an auto player. In the 3D version, all of these logic components are preserved.

In the 2D version, the UI components are not separated very clearly from logic components. How to integrate the new 3D user interface with the existing logic part, which is complicated and has been tested, is the most difficult part of the project. By using software reverse-engineering approach, we isolated the classes and functions only or mainly for user interface from those classes/functions for

game logic. We also defined the interfaces between the UI components and logic components. With this approach, we can focus our attention only on UI components without affecting the logic part. This method saved us lots of time on UI development and system testing.

To support UI implementation with OpenGL, a set of classes based on an extendible hierarchical structure is designed and implemented. With the help of these classes, it's very easy to create autohidable/ non-autohidable windows with customized position, size, background, contained controls, and controls of different size, position, shape, text, image, tool tip, border-style, font, etc.

The followings are the new features implemented in the 3D version:

- 3D snooker table with wooden cover
- Light reflection on balls
- Multiple views – Top view, Side view and Fine view
- Continuous moving – zooming, turning, moving
- Foldable Scoreboard
- Auto-hidable subwindow
- Tool tip for immediate help
- Exact cue speed shown as tooltip
- Full screen game
- Change cursor to show the safe hitting area of the cue ball in the Adjust Window
- Separate main menu page to make UI clearer
- Animated 3D cue

- External resources, supplies a possibility to use change-skin technology to make the UI more fun and pleasant to use (not implemented yet)

The following features of the 2D version are preserved and/or re-implemented in the 3D version.

- Four play modes – manual, trace, advise and auto
- AI for auto/advise play mode
- Player parameter setting
- Environment parameter setting
- Time testing for best setting
- Demo
- Sound effect control
- Cue speed control
- Cue ball hitting position control
- 3D view for adjusting colliding pot between cue ball and target ball more precisely
- Ball-like cursor in Snooker table for positioning a ball precisely
- Keyboard support

2. Background

2.1 Game Programming

In the last decade, computer-based game has grown into a multi-billion-dollar market.

A computer-based game mainly consists of the following elements. [3]

- Graphics
- Input
- Music and sound
- Game logic and artificial intelligence
- Networking
- User interface and menu system
- A good game interface makes playing the game as easy as possible

As to an individual game, not all the aspects are necessarily present, such as the networking element may not present in a non-network game. But the above elements have already covered all the essential aspects of a game.

First of all, a game must be amusing and interesting. A beautiful user interface and exciting 2D/3D graphics are vital to a game, they must be able to seize the players' attentions at the first viewing.

In addition, the game must be powerful but easy to use. Nobody is willing to spend several hours just learning how to play the game. So, a clear, friendly user interface with convenient input approach is very important to give players a relaxed playing environment.

Anyway, to achieve a fascinating game, high artificial intelligence is the key factor. Beautiful UI, exciting music and graphics can only pull the players to the play station, the game itself is the key to keep the players staying there and playing.

From all the aspects we discussed above, we know PC Game is a complicated engineering system. It involves many fields of computer science, hardware, software, multi-media, computer graphic, etc.

2.2 OpenGL Game Programming

As the PC Game industry evolves, the players are not satisfied with 2D images any more. They want the game images to be more fun to watch, as close to real life as possible, even more interesting and exciting.

3D user interface is becoming the main stream of PC game development. Many powerful 3D development environments are available for developers to create 3D graphics without any low-level programming knowledge, such as SoftImage, 3D Studio, Maya, Photo Styler, and Inferno. All these applications are based on low-level 3D API, such as OpenGL, DirectX, and Heidi.

OpenGL is not only a graphics-programming library, but also a software interface to graphic hardware, and a "standard" for 3D graphics. More and more hardware manufacturers and operating system developers promise to support OpenGL by including OpenGL library as one part of their products, such Microsoft's window system. OpenGL has become the main 3D graphic standard, main 3D graphic developing tool in last ten years.

OpenGL is also an excellent tool and powerful library for the development of PC games. It has the following advantages:

- Device independence. OpenGL is an interface between hardware and software. With OpenGL's support, developers do not need to care about the technical details of display devices, but can put all their attentions on the 3D graphics itself.
- Simple, easy to learn. OpenGL is just a set of commands. All the complicated algorithms are encapsulated very well. This feature can shorten the learning curve and accelerate the development speed.
- Powerful, flexible feature set. OpenGL is powerful and flexible. Developers can implement any kind of feature they want by using OpenGL.
- Cross-platform portability. OpenGL is platform-independent; it can be ported to any system that supports OpenGL. If the game is developed with pure OpenGL, it can be compiled and run on many systems, so as to save development cost (money and time) and extend the potential market of the game.
- Established: More and more third-party libraries are available in the market. All these tools are tested very well. This makes OpenGL more powerful, stable and easier.
- Performance. OpenGL are implemented at assembly level and all the algorithms and codes are optimized. With hardware's support, OpenGL

can gain very good performance enough to develop any complicated 3D graphics.

- Interactive: OpenGL is good at developing interactive applications. Good interactivity is the vital feature of Games.

2.3 2D Snooker Simulator

The 2D Snooker Simulator was developed based on an earlier snooker simulation that runs on PC under DOS, which was also developed by Professor Grogono. 2D Snooker Simulator did a good job on simulating the motions of the balls much accurately.

As an interesting feature of 2D Snooker Simulator distinctive from other snooker games, environment properties are considered for calculating the movements of balls. The player can control these properties by modifying the parameter settings. So, it's not only a game, but also a real simulation of snooker game.

In the simulator, the effects of ball/ball and ball/cushion collisions are simulated by solving differential equations numerically, rather then by relying on analytical solutions of simplified equations. The behavior of balls in the simulation is very similar to the behavior of balls on a real table. There are some limitations, however the cue is always horizontal, so massé shots are not possible, and balls remain on the table. [2]

Meanwhile, 2D Snooker Simulator has a lot of built-in intelligence. It supports four playing modes – manual, track, advise and auto. The player can play snooker by himself with/without tracking, or watch the computer simulating the game. He can also play against the simulated player. It is worth mentioning that the game supports personalized auto-play/advise mode. Following an auto-play algorithm, the computer can calculate and select the best action according to current situation and the selected auto-player's ability is not always the ideal action. This mechanism increases the amusement of the game.

The graphics and user interface of 2D Snooker Simulator was developed using Fastgraph™ and window GDI. The applications tried to simulate 3D effect in 2D context, such as light reflection on the balls, pseudo 3D view for fine adjustment .The user interface is clear and concise with enough help information. But it's based on standard windows with standard drop-down menus. As a game, it's not fun or exciting enough.

3. Design

3.1 Project Goal

The 3D Snooker Simulator is based on the 2D version. We kept all the useful functions in the new version. Meanwhile, we're trying to improve the game in following aspects:

- 3D graphic and User Interface with multiple views, allows players to switch between different views to get more details as real game
- Fun and interesting UI, like a real game
- More friendly UI, allows the player to get help more convenient and quickly
- Platform independent

OpenGL libraries supply the strong ability to create any 3D objects the developer wants, but do not implement objects for generating and managing UI components quickly and easily – such as window and controls. Now, many third-party libraries based on OpenGL are available for creating user interface components, but most of them are not suitable for game development, such as Glui. Meanwhile, most of the libraries are platform-dependent, mainly for windows system. So, we decide to create our own foundation classes to support the UI components implementation. These classes are designed as an extensible architecture. This architecture makes it possible to extend the classes to a library for common UI components development in the future.

In addition, in 2D Snooker Simulator, a lot of global variables are used to pass parameters across the classes. This structure doesn't conform to the principle of object-oriented development and is not easy to manage. So, we will try to encapsulate most of these global variables in some parameter management classes. These classes are responsible for managing the parameter passing across the classes. In this way, the application will be more robust and flexible.

3.2 UI Components Design

Window and control are main kinds of components of user interface. Because the implementation of 3D UI of Snooker Simulator (3D) is based on pure OpenGL and ANSI C++, it's difficult and time-consuming to generate and manage windows and controls without UI library's support.

So, a set of classes (which may be extended to a UI library in the future) are designed and implemented to support the implementation of 3D user interface in following aspects:

- Simplify the UI components management
- Supply enough functions for the implementation
- Supply an extensible structure, make it easy to add new functions to the architecture
- Wrap OpenGL functions as much as possible

3.2.1 Window

The window system of the application is a multi-level hierarchical architecture. The base class of the window hierarchy is the class VirtualWindow. This class includes all the common functions of windows, such as creating, displaying, moving, resizing and event handling. Most of the functions are overridable, VirtualWindow only supplies a simple implementation. In all the children window classes, these functions can be re-implemented according to their special requirements.

Meanwhile, each window maintains a collection of subwindow and a collection of controls to deal with all the subwindows and controls contained in this window.

To meet the requirement of this project, three second-level window classes are derived from VirtualWindow:

- MainWindow – OpenGL main window
- Glui Window – deal with Glui windows
- SubWindow – handle all subwindows (for various reasons, VirtualWindow is not implemented as an abstract class. It is mixed with SubWindow during implementation.)

Based on these three window classes, developers can derive as many children window classes as they want to meet their different needs. Since almost all the main functions have been implemented in the parent windows, the developer just needs to add or re-implement some functions to meet their special requirements in children windows, such as adding its own control, images and override the display function.

The following window classes are created to support the 3D user interface implementation.

- MainMenuWin: derived from VirtualWindow, acts as a welcome page to players, list the main functions;
- Scoreboard: derived from VirtualWindow, shows all statistic data for players;
- TaskBar: derived from VirtualWindow, lists all available functions;
- AdjustWin: derived from VirtualWindow, acts as a container of three subwindows for adjusting cue speed, direction, hit point on cue ball.
- FineAdjustWin: derived from VirtualWindow. Adjust the hitting point between target ball and cue ball more precisely;
- PanelCueSpeed: derived from VirtualWindow. Adjust cue speed;
- PanelCueBall: derived from VirtualWindow. Adjust the hitting point of cue on cue ball;
- PanelHelper: derived from VirtualWindow. Shows help topics;

- NominateWin: derived from VirtualWindow for selecting nominated color ball;
- SettingWin: derived from VirtualWindow. Shows buttons for settings;
- ToolTipWin: derived from VirtualWindow. Shows tooltips for main components;
- PlayerModeSetWin: derived from GluiWindow. To set players' parameters;
- ParameterSetWin: derived from GluiWindow. To set environment parameters;
- GluiHelpWin: derived from GluiWindow. Shows help for Glui windows;

The following figure depicts the static structure of the classes.

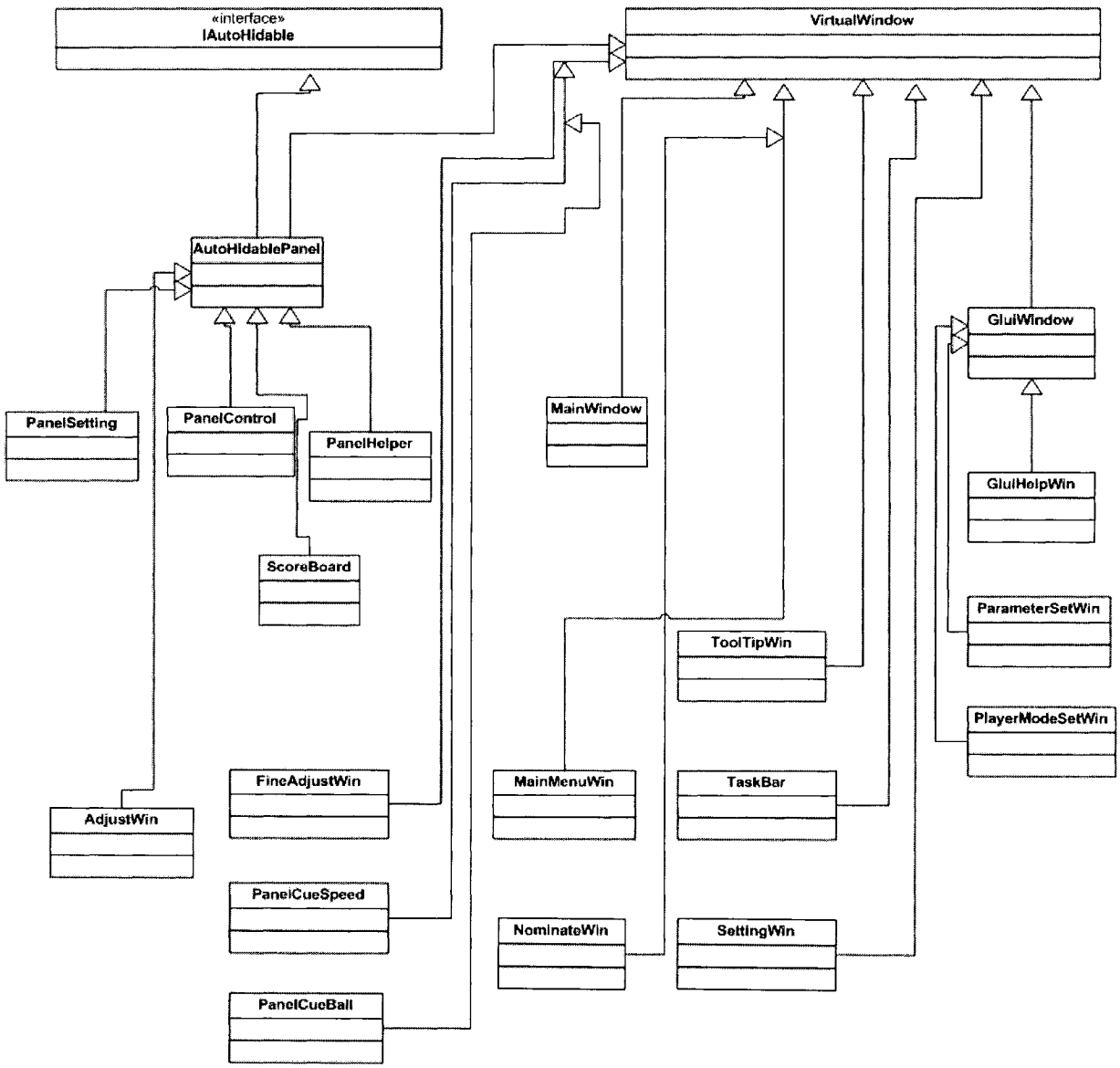

Figure 3.1 Class diagram of all foundation classes

In addition, a mechanism is supplied to extend the functions of window – interface implementing. The developer may be able to define interface to supply more functions, such as auto-hidable, foldable, draggable. To achieve these new features, the window only needs to inherit the interface and implement the methods defined in the interface.

3.2.2 Control

Controls are also implemented in a hierarchical architecture since the controls may share many common attributes and functions.

VirtualControl is the base class of all controls. In this class, all the common attributes are declared, such as position, size, color, image, border, font, text, tooltip text, etc. The VirtaulControl is responsible for managing these attributes. All the common functions are also declared. Some of them are implemented — if the mouse is on the control, which control is pressed, highlight the control, hide/show control, etc.

In this project, only two types of controls are implemented — ButtonCtl and LabelCtl. Since the architecture of the project is extensible, any new kind of control can be created by deriving from VirtualControl according to developer's needs, such as TextBox, RadioBox, CheckBox, ComboBox, etc.

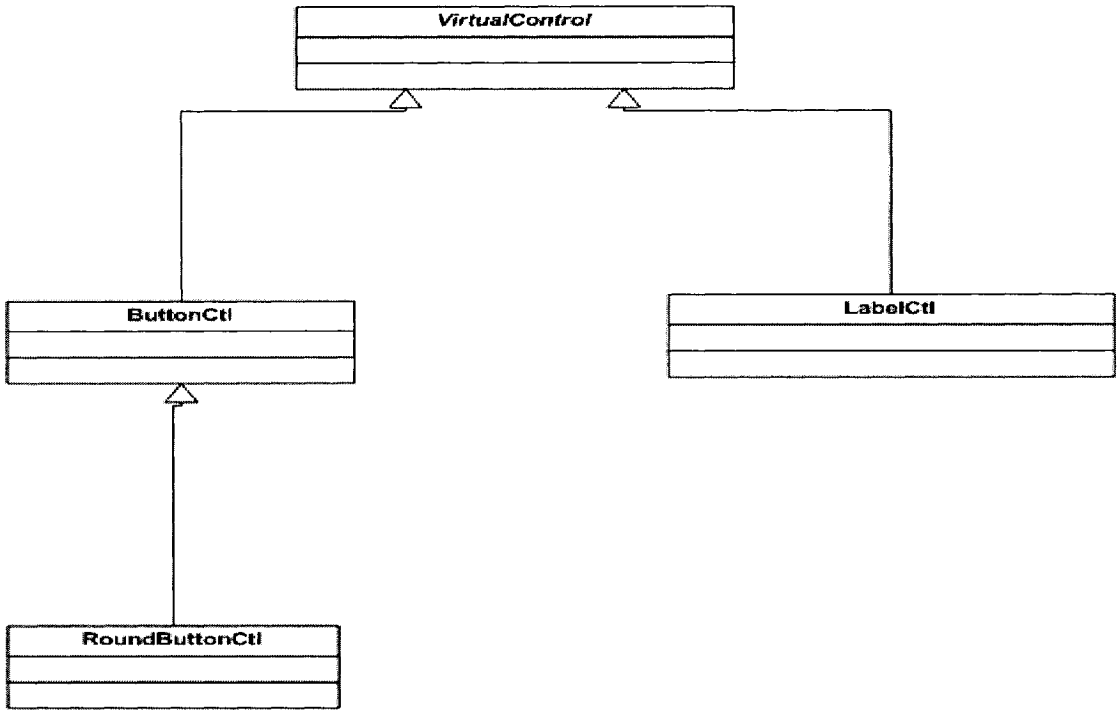

Figure 3.2 Class diagram for controls

3.2.3 Control Collection

To simplify the management of controls, a ControlCollection class is created as a container to manage a collection of controls in a window. The developer needn't touch the controls directly. Normally, each window class maintains one or more ControlCollection objects. The developer manages the controls with the help of this collection, such as adding new control, getting control by ID or index, removing specified control from collection, querying which control is pressed. This mechanism allows the developers to communicate with the controls without caring about the details of different controls.

Figure 3.3 Relationships between Window, ControlCollection and Control

3.3 Parameter management class Design

In 2D Snooker Simulator, a lot of parameters were implemented as global variables, such as environment parameters -- ball-ball friction, ball-cushion friction, delta_t, time_increment. These variables are declared in globals.cpp, initialized by reading from external data file. The parameters may be modified in parameter setting window, and be referenced in many classes. Here is the simplified diagram to show this global variable pattern.

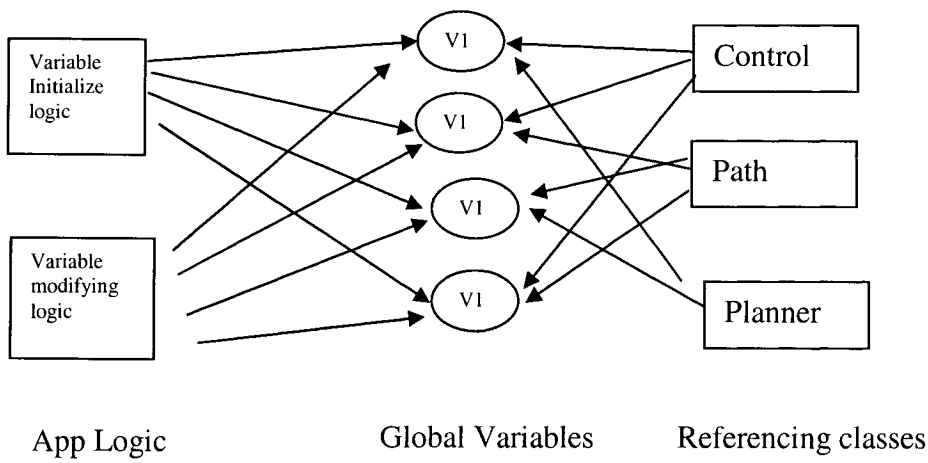

Figure 3.4 Parameter passing in 2D version

In the picture above, it's shown clearly that the relationships between global variables and the related classes are very complicated. It makes the management of variables very difficult. Meanwhile, when reading the code, it's very difficult to figure out how these global variables are created, initialized, modified and referenced.

If these variables can be encapsulated in a parameter management class, it could make the parameters easy to manage as well as the application easy to understand. The parameter management class is the only thing exposed to other classes, the pointer of parameterManager object is passed to the related classes as a parameter. All the management on the variables is wrapped in this class, declaring, and initialization. To modify or add new features to these parameters, we just need to work on this class. This class has been encapsulated very well, it exposes a predefined interface to outside; any modification inside the class will not affect other user classes. The following diagram depicts this implementation pattern. It's much clearer and simpler than the previous global variable pattern.

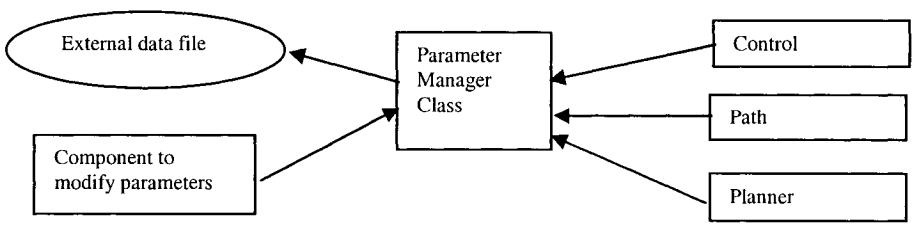

Figure 3.5 Parameter passing in 3D version with ParameterManagerClass

The parameters wrapped in the parameter management class are listed below. Most of them are implemented as global variables in the 2D version. The parameters are divided into three categories.

- Environment Parameters:

  defaultBallBallFriction, the default friction factor between balls;

  defaultBallCushionFriction, the default friction factor between ball and cushion;

  defaultSlidingFriction, the default friction factor for ball's sliding;

  defaultRollingFriction, the default friction factor for ball's rolling;

  defaultDisplayFactor, the default factor for displaying;

  defaultTimeIncrement, the default time increment for redrawing;

  defaultZSpinFriction, the default friction factor for Z spin;

  defaultDeltaT, the default Delta time;

  ballBallFriction, the real friction factor between balls;

  ballCushionFriction, the real friction factor between ball and cushion;

  slidingFriction , the real friction factor for ball's sliding;

  rollingFriction, the real friction factor for ball's rolling;

  displayFactor, the real factor for displaying;

  m_timeIncrement, the real time increment for redrawing;

  zSpinFriction, the real friction factor for Z spin;

  and sfg, rfg, zfg, zSpinMin, sqColRad, kappa.

- Player's information

  Play mode, the play mode of this player;

  Player name, the name of this player;

  Advisor name, the advisor's name selected for this player;

  Auto-player index, the selected index for auto-playing;

  Max adjust times, the maximum times of allowed adjustment;

  Used adjust times, the used times of adjustment;

  Score, the current score of this player;

  Frames, the number of frames this player won;

  Breaks, the current score of this break of this player;

  Cumulative, the cumulative score of this player;

  High breaks, the high break of this player;

  Current player index, index of this player

- Game parameters

  Cue speed, the speed of the cue;

  Side spin, the speed of side spinning;

  Top spin, the speed of top spinning;

  Sound effect, if the sound effect is on or off

4. Implementation

4.1 Window system

4.1.1 VirtualWindow

In VirtualWindow, the common attributes of windows are declared, such as position, size, border, title, ID, pointer to parent window and contained control collection.

VirtualWindow m_parentWindow;*

ControlCollection m_Controls;*

double m_leftPos;

double m_topPos;

double m_winWidth;

double m_winHeight;

double m_parentWidth;

double m_parentHeight;

int m_thisWin;

int m_parentWin;

bool m_isVisible;

char m_winTitle;*

int m_borderStyle;

*char** texFileNames;*

bool m_needBlend;

WindowType m_winType;

The public/protected methods in the VirtualWindow are of four categories.

- overridable methods. These methods are the most important functions of the window. For different child window, the implementation of each method may be different; it will be re-implemented if necessary. The VirtualWindow only supplies a simple implementation.

```
virtual int CreateWin(double leftPos, double topPos, double width, double height);
virtual int CreateWin();
virtual void Display();
virtual void Init();
virtual void AddControls();
```

- Accessors and Mutators. These functions are used to get or set the attributes of VirtualWindow. Most of them are implemented in VirtualWindow. But some of them are overridable, such as SetWinPos() and SetWinSize(). Different actions might be taken in different child window.

```
int GetThisWindowID();
int GetParentWindowID();
double GetLeftPos();
void SetLeftPos(double left);
double GetTopPos();
```

void SetTopPos(double top);

virtual void SetWinPosition(double left, double top);

double GetWinWidth();

void SetWinWidth(double width);

double GetWinHeight();

void SetWinHeight(double height);

virtual void SetWinSize(double width, double height);

virtual void SetParentSize(double width, double height);

void SetWinTitle(char title);*

char GetWinTitle();*

void SetBorderStyle(int style);

int GetBorderStyle();

bool IsVisible();

void SetVisible(bool isVisible);

int GetWinType();

VirtualControl GetControlByID(int controlID);*

virtual int GetPressedButtonID(int x, int y);

VirtualWindow GetParentWindow();*

- Event handlers. To deal with mouse click event, mouse moving event and timer event.

virtual void DoMouse(int button, int state, int x, int y);

virtual void DoMouseMoving(int x, int y);

virtual void DoTimer(int i);

- Operations on Window. Support basic operations of windows, moving, refreshing, showing, hiding, etc. These methods encapsulate the OpenGL commands. The developer can manage the windows without caring about the implementation details using OpenGL.

void RefreshWin();

void SetAsCurrentWin();

void MoveWin(double x, double y);

void BringToFront();

void SendToBack();

virtual void ShowWin();

virtual void HideWin();

4.1.2 MainWindow

MainWindow is a little different from other windows. As the top-level window, it should be able to be iconified so as to allow players switch between the games and other applications, so a new function IconifyWindow() is added to implement this feature. In addition, a new attribute m_fullScreen is added to indicate if the game is running at full-screen mode.

In OpenGL, creating the main window is more complicated than creating a subwindow. So, in the MainWindow, CreateWin function is re-implemented as following.

```
int MainWindow::CreateWin(double leftPos, double topPos, double width, double
 height)
{
 m_leftPos = leftPos;
 m_topPos = topPos;
 m_winWidth = width;
 m_winHeight = height;
 glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
 glutInitWindowSize(m_winWidth, m_winHeight);  //Create Main Window
 m_thisWin = glutCreateWindow("Snooker Simulation");
 GLUI_Master.set_glutSpecialFunc(OnSpecial);
 GLUI_Master.set_glutMouseFunc(OnMouseClicked);
 glutIdleFunc(idle_function);
 GLUI_Master.set_glutKeyboardFunc(OnKeyboard);
 GLUI_Master.set_glutDisplayFunc(OnDisplay);
 glutTimerFunc(100, OnTimer, m_thisWin);
 glutPassiveMotionFunc(OnMouseMoving);
 if(m_fullScreen)
     glutFullScreen();
 return m_thisWin;
}
```

4.1.3 GluiWindow

GLUI is a system-independent, GLUT-based C++ user interface library. It provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications.

GluiWindow is a wrapper class for Glui windows that contain Glui controls. In GluiWindow, the wrapped Glui windows are responsible for handling their events, so the GluiWindow doesn't need to care about the event handling. Meanwhile, Glui window is responsible for managing all the variables that are bound with the Glui controls, so a new function Sync() is added to synchronize the bound variables and the controls.

```
void GluiWindow::Sync()
{
    m_gluiWin->sync_live();
}
```

4.1.4 Other windows

All the other windows are derived from three base classes discussed above. Most of the attributes and methods have been declared and/or implemented in the base classes. To meet the special requirement of each derived window class, the specific overridable methods need to be re-implemented, such as Init(), Display(),

AddControls(). New attributes and methods might be added to the derived window for new features.

4.2 Controls

4.2.1 VirtualControl

VirtualControl is the top-level base class of all controls; it is implemented as an abstract class. In this class, the common attributes and methods shared by all controls are declared.

The declared attributes are listed below.

int m_ctlID;

int m_parentWin;

int m_imageIndex;

int m_disableImageIndex;

GLfloat m_topPos;

GLfloat m_leftPos;

GLfloat m_Width;

GLfloat m_Height;

int m_borderStyle;

int m_borderSize;

int m_foreColor;

int m_backColor;

int m_fontIndex;

int m_borderColor;

int m_highLightColor;

GLuint m_texturePtr;*

char m_ptrText;*

bool m_swapText;

double m_finalBottomPos;

bool m_ctlVisible;

bool m_ctlEnabled;

bool m_ctlHighLighted;

char m_toolTipText;*

int m_textJustification;

int m_textWidth;

int m_textMargin;

double m_textX;

double m_textY;

int m_finalBottom;

bool m_mouseOnCtl;

Common methods declared in VirtualControl are divided into three categories.

- Pure virtual functions: These functions are declared in VirtualControl class but not implemented. As to these functions, each derived control will have its own implementation to meet the specific needs.

virtual void DrawControl() = 0;

virtual const int WhatType() = 0;

- Common methods. These methods have already been implemented in the VirtualControl to supply common functionalities. But some of them are declared as virtual, the derived classes can re-implement these methods for individual requirements.

  virtual int GetBottom() ;

  virtual const bool IsPressed(GLfloat mouseX, GLfloat mouseY);

  *void bitmap_output(int x, int y, char *string, void *font);*

- Accessor and Mutators: They're used to retrieve or modify the attributes of controls, such as position, size, font, text, border, background, tooltip, images and visibility.

  const int GetControlID();

  const int GetParentWin();

  void SetParentWin(const int parentWin);

  const int GetImageIndex();

  void SetImageIndex(const int imageIndex);

  void SetDisableImageIndex(const int disableImageIndex);

  GLuint GetImageArray();*

  void SetImageArray(GLuint newImages);*

  const GLfloat GetLeft();

  void SetLeft(GLfloat left);

  const GLfloat GetTop();

void SetTop(GLfloat top);

const GLfloat GetWidth();

void SetWidth(GLfloat width);

const GLfloat GetHeight();

void SetHeight(GLfloat height);

const int GetBorderStyle();

void SetBorderStyle(int borderStyle);

const int GetBorderSize();

void SetBorderSize(int borderSize);

const int GetForeColor();

void SetForeColor(int foreColor);

const int GetHighLightColor();

void SetHighLightColor(int highLightColor);

const int GetBorderColor();

void SetBorderColor(int borderColor);

const int GetBackColor();

void SetBackColor(int backColor);

const int GetFontIndex();

void SetFont(int fontIndex);

char GetText();*

void SetText(char newText);*

const bool NeedSwap();

void RequestSwap(bool needSwap);

const double GetFinalBottomPos();

const bool IsVisible();

void SetVisible(bool isVisible);

const bool IsEnabled();

void SetEnable(bool isEnabled);

const bool IsHighLighted();

void SetHighLight(bool isHighLighted);

void SetToolTip(char tipText);*

char GetToolTip();*

double GetTextX();

double GetTextY();

void SetTextJustification(int just);

void GetFont();*

void SetMouseOnCtl(bool isOn);

bool IsMouseOnCtl();

4.2.2 Button Control

Button is the most common control used to execute a command. ButtonCtl class inherits all the attributes and methods of VirtualControl. It implements the pure virtual function DrawControl() to draw a button. To improve the user-friend feature, in the DrawControl function, it allows the user to set different appearance of the button for different state – enabled and disabled. Meanwhile, the button will be enlarged when the mouse is located above the button. It's very easy for the users to get the state of buttons.

RoundButtonCtl is the child class of ButtonCtl to supply a special button of round shape. In this class, the DrawControl method and IsPressed methods are re-implemented.

4.2.3 Label Control

The label control is used to show text in a specified area. Compared with Button control, label control has more requirements on showing text. In LabelCtl, WriteText method is added to write the text in the scope of the control. The developers can set the text to be wrapped or not. When text-wrap is not set, if the text is too long to be displayed as one line in the control, the text will be truncated to meet the width of the label control. If text-wrap is requested, the text will be wrapped automatically when it reached the horizontal margin of label control. The vertical size of label control will be increased to display all the text.

4.3 Control Collection

ControlCollection class is used to manage a set of controls. This class maintains a fix-size array of VirtualControl pointers. When adding new control to the collection, the ControlCollection will check the array from the beginning, get the first null pointer and set the pointer to the new control there. If collection wants to work on a specific

control, it will check the array and compare the ID of target control with each item's ID (each control is assigned a unique ID when it's created). The ControlCollection supplied following functions on controls.

- Add new control
- Remove an existing control
- Retrieve an existing control by ID or index
- Draw all controls by calling each control's DrawControl function
- Check which control is pressed

4.4 ParameterManager Class

ParameterManager class is implemented as a data center; it stores most of the shared parameter, which will be referenced by data-consumer classes, such as Control, Path, and Planner. Retrieving or modification on these parameters will be acted on the ParameterManager class directly; this ensures all the data consumers can get the up-to-date data at any time.

The data stored in ParameterManager class belong to two different categories, shared data and data for individual player.

ParameterManager class manages the shared data directly, such as all the environment parameters, cue speed, top spin and side spin degrees. The data consumer can access these data directly.

In Snooker game, there are exactly two players. The structure, PlayerMode, is created to store the data for individual player. The ParameterManagerClass maintains an array of PlayerMode with size of 2 since only two players may be involved in a Snooker game. The data consumer can only access the specified player's data through ParameterManager class by passing in the player's index (as ID) or request the current player's data directly — the current player index is maintained by the class.

The PlayerMode stores all the properties of an individual player, such as play mode, name, advisor name, max adjust times, current scores, breaks, won frames. It is declared as:

```
struct PlayerMode
{
    Mode mode;
    char player_name[30];
    char advisor_name[30];
    int advisor_index;
    int auto_player_index;
    int max_adjust_times;
    int used_adjust_times;
    int profileIndex;
```

```
    int score;
    int breaks;
    int frames;
    int cumulatives;
    int highBreaks;
};
```

Meanwhile, ParameterManager class wraps some functions related to parameter management, such as:

- Initialize ParameterManager class by reading default settings from file;
- Read profile data from file for initialize the Profile classes. Though there exists a Profile class, it is mainly responsible for managing one profile. In the game, there are 10 profiles for auto/advise play mode; we still need a class to manage all the profiles. ParameterManager class is used to read profile from file, and supplies the formatted data as request. In the player.dat file, the raw data for a player is a mixture of name, scores in ten aspects, and the total score. If there is no class to parse the raw data, it's difficult to get the required data quickly.

ParameterManager class is created to encapsulate the global variables. To allow the consumer class be able to access the ParameterManager, a pointer to ParameterManger is declared as a member data in each class which wants to access ParameterManager. The ParameterManager object pointer is passed into these classes through the constructor. During the lifetime of consumer objects, they can access the ParameterManager object freely to get or set the target parameters. Since all ParameterManager pointers point to the same ParameterManager object, any change with the ParameterManager object will take affect to all the related objects immediately.

The following code describes how to initialize a ParameterManager object and how to pass the object to the parameter consumer classes.

```
parameterManager = new ParameterManager();
parameterManager->ReadProfilesFromFile();
for(int i = 0; i < parameterManager->GetProfileNum(); i++)
{
  profiles[i] = new Profile(parameterManager->GetProfileByIndex(i));
}
planner = new Planner(parameterManager);
path = new Path(waves_ok,parameterManager);
ref = new Referee(parameterManager);
```

4.5 Implementation problems and solutions

4.5.1 Implement Auto-Hidable subwindow

In some cases, the user may want the window to be able to hide/show automatically on their request so as to save the space and make the user interface simple and clean. To gain such effects by using our UI component classes, the developers only need to create a subwindow class that derived from VirtualWindow and IAutoHidable and implement the IAutoHidable interface in this class.

The most important parameters for hiding/showing window automatically are original position and hiding direction. The original position, which decides the original position of the window when it's displaying, is required. The hiding direction, which decides the direction (top, bottom, left or right) that the window is hiding to or appearing from, is optional. The developer can select his preferred direction and the window will act as his request. He can also just set the original position and the application will calculate the hiding direction automatically according to the distance between original position and the parent window's margin.

To implement auto-hidable feature, two main steps are needed:

1. Calculate the direction of auto-hiding

if the developer doesn't designate the hiding direction. Three options are supplied for calculating the auto-hiding direction:

- FULL_AUTO: The window will slide to/from the nearest margin of parent window in both vertical and horizontal directions;
- VER_AUTO: The window will slide to/from the nearest margin of parent window only in vertical direction, top or bottom;
- HOR_AUTO: The window will slide to/from the nearest margin of parent window only in horizontal direction, left or right;

Once the size of the parent window is changed, the auto hiding direction will be recalculated to ensure the best hiding effect.

This feature is implemented in CalculateAutoHideDir() function.

```
void AutoHidablePanel::CalculateAutoHideDir()
{
    if(m_autoSetHideDir)
    {
        double leftSpace = m_leftPos;
        double topSpace = m_topPos;
        double rightSpace = m_parentWidth - m_winWidth - m_leftPos;
```

```
double bottomSpace = m_parentHeight - m_winHeight - m_topPos;

switch(m_autoSetHideDir)
{
case FULL_AUTO:
        if(leftSpace <= topSpace && leftSpace <= rightSpace &&
        leftSpace <= bottomSpace)
                m_hideDirection = HIDE_DIR_LEFT;
          else if (topSpace < leftSpace && topSpace <= rightSpace &&
          topSpace <= bottomSpace)
                m_hideDirection = HIDE_DIR_TOP;
          else if( rightSpace < leftSpace && rightSpace < topSpace &&
          rightSpace <= bottomSpace)
                m_hideDirection = HIDE_DIR_RIGHT;
        else
                m_hideDirection = HIDE_DIR_BOTTOM;
        break;

case VER_AUTO:
        if(topSpace > bottomSpace)
                m_hideDirection = HIDE_DIR_TOP;
        else
                m_hideDirection = HIDE_DIR_BOTTOM;
        break;

case HOR_AUTO:
```

```
                if(leftSpace >= rightSpace)
                        m_hideDirection = HIDE_DIR_LEFT;
                else
                        m_hideDirection = HIDE_DIR_RIGHT;
                break;
        }
        SetStartPos();
    }
}
```

2. Action of hiding/showing

The hiding/showing action is implemented by using SetTimer function supplied by OpenGL. The OpenGL runtime keeps executing the Hiding_Showing function periodically. If no request is raised to change the current status of auto-hidable window, no action will be taken. When a user requests a status change, to hide if showing or to show if hiding, the m_statusChanging will be set to true, the window will start to slide in the predefined direction at the predefined hiding speed every period (the window will be redisplayed after each moving so as to generate the animation) till it reaches the predefined showing position or moves out of the parent window completely. Then, it will change the m_statusChanging to false and wait for next request to change the window status.

4.5.2 Simplify Texture Mapping

Texturing is a powerful technique supplied by OpenGL. By using Textures, the developer can import existing pictures and make the 3D object more beautiful and similar to the real object. But using texture mapping requires a lot of repeated jobs, we have to load the bitmap from file and initialize it as a global image array every time. It's time-consuming because we have to repeat the same steps for each component (window or control) if we want to use texture mapping in this component.

To simplify the process of using texture mapping, we wrap the basic common steps, load bitmap and init text, in the base classes – VirtualWindow and VirtualControl. Then, it's easy to handle the texture mapping in any class derived from these two base classes.

Here are the functions implemented in VirtualWindow to load bitmap and initialize the texture.

```
void VirtualWindow::InitTexture(const int textureNumber, GLuint* images)
{
        AUX_RGBImageRec **ControlImage = new
        AUX_RGBImageRec*[textureNumber];
        memset(ControlImage, 0, sizeof(void *)*textureNumber);
        glGenTextures(textureNumber, images);
```

```
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
        GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
        GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
        GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
        GL_NEAREST);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
        GL_DECAL);

        for(int i=0 ;i<textureNumber; i++)
        {
                ControlImage[i]=LoadBMP(texFileNames[i]);
                glBindTexture(GL_TEXTURE_2D, images[i]);
                gluBuild2Dmipmaps (GL_TEXTURE_2D, 3,
                ControlImage[i]->sizeX, ControlImage[i]->sizeY, GL_RGB,
                GL_UNSIGNED_BYTE, ControlImage[i]->data);
        }
}

//load bmp file for texture mapping
AUX_RGBImageRec * VirtualWindow::LoadBMP(char *Filename)
{
        FILE *File=NULL;
        if (!Filename)
```

```
{
        return NULL;
}

File=fopen(Filename,"r");
if(File)
{
        fclose(File);
        return auxDIBImageLoad(Filename);
}
return NULL;
}
```

To use the texture mapping, the developer only needs to follow the steps below:

- Declare a static global array to store the textures. (OpenGL requires it implemented as a static global array.)

```
static GLuint newCtlName[22];
```

- Initialize a bmp file name array and call InitTexture to load bmp file and initialize the texture.

```
const int textureNumber = 22;
texFileNames = new char*[textureNumber];
texFileNames[0] = "resource/exit.bmp";
texFileNames[1] = "resource/left_arrow1.bmp";
...
```

texFileNames[20] = "resource/end.bmp";

texFileNames[21] = "resource/minimize.bmp";

InitTexture(textureNumber,newCtlName);

- Use the texture freely.

glBindTexture(GL_TEXTURE_2D, newCtlName[17]);

This mechanism makes the texture mapping easy to implement. Because of OpenGL's requirement, we cannot wrap everything into classes; global arrays are still being used. If this array can be implemented as an instance data of a class without missing OpenGL's requirement, the texture mapping will be much easier to manage.

4.5.3 Event handling

In the game, each window needs to handle its own events, such as MouseClicked, MouseMoving, Timer, Keyboard, Display, Reshaping. OpenGL requires global callback function for each event. If we define a global callback function for each event of each window, the code will become very big and confusing.

To simplify the event handling, we define a global callback function for a kind of event to meet the requirement of OpenGL (so far, we have not found a better way to wrap these callback functions to a class). The same event from

different windows will call the same callback function. In this callback function, it will check the event publisher and call the corresponding class's event handler. Each window will deal with their own event in the class.

To implement this event handling mechanism, we need to perform the following actions.

- Define global callback functions in Snooker.cpp file: OnKeyboard, OnMouseClicked, OnMouseMoving, OnTimer, OnSpecial, OnReshape, OnDisplay,OnIdle
- Register event handlers for each window.

  All windows are derived from VirtualWindow class, so we register the handler in VirtualWindow for all sub windows.

```
if(m_winType == Win_Type_SubWindow)
{
        glutDisplayFunc(OnDisplay);
        glutMouseFunc(OnMouseClicked);
        glutPassiveMotionFunc(OnMouseMoving);
        glutTimerFunc(100,OnTimer,m_thisWin);
        glutKeyboardFunc(OnKeyboard);
}
```

Registering event handler for main window is a little different from that for subwindows – special events are needed in the main window to support Glui windows. So, we register event handlers in MainWindow class with different implementation.

```
GLUI_Master.set_glutSpecialFunc(OnSpecial);
GLUI_Master.set_glutMouseFunc(OnMouseClicked);
glutIdleFunc(OnIdle);
GLUI_Master.set_glutKeyboardFunc(OnKeyboard);
GLUI_Master.set_glutDisplayFunc(OnDisplay);
glutTimerFunc(100, OnTimer, m_thisWin);
glutPassiveMotionFunc(OnMouseMoving);
glutReshapeFunc(OnReshape);
```

Glui window will handle the events by itself, so we don't register event handler for Glui windows.

- Declare overridable event handler in VirtualWindow

```
virtual void DoMouse(int button, int state, int x, int y);
virtual void DoMouseMoving(int x, int y);
virtual void DoTimer(int i);
```

- Re-implement the overridable event handler in each derived window class

4.5.4 Text size calculation in ToolTipWin

ToolTipWin will show the tooltip text for the control/window to give players useful help information when the mouse is hovering on the control/window. The ToolTipWin will be resized automatically to display all the tooltip text.

To write text in a ToolTipWin, we need to calculate the maximum width and total height of the text. The ToolTipWin will be resized according to the calculated maximum width and total height. As to the ToolTipWin, the displayed tooltip text will not be wrapped automatically which means a new line will be created only when "\r\n" or "\n" is met.

The algorithm for calculate the maximum width and total height of tooltip text is as follows:

1. Init mostX to 0, lineNo to 0
2. Check each character of tooltip text till the end of text

   If current character is "\r" and the next one is "\n", skip these two characters, if curX larger than mostX, assign curX to mostX increment lineNo, set curX to 0.;

   Else if current character is "\n", skip this character, if curX larger than mostX, assign curX to mostX increment lineNo, set curX to 0;

   Else increment curX by

glutBitmapWidth(m_currFont, m_tipText[i])

3. Get maximum width from mostX;

   Get total height by sum up all the line's heights and spaces:

   *lineNo * m_fontHeight + lineSpace * (lineNo - 1)*

The following function implements the algorithm discussed above for calculating the tooltip text size.

```
void ToolTipWin::CalculateWinSize()
{
    int lineSpace = 2;
    int len = (int) strlen(m_tipText);
    int curX = 0;
    int mostX = 0;
    int curY = 0;
    int lineNo = 0;
    if(!m_needSwap)
    {
        int i = 0;
        while(i<len)
        {
            lineNo++;
            for (; i < len; i++)
            {
                if(m_tipText[i]=='\r' && m_tipText[i+1] == '\n')
```

```
                {
                    i++;
                    i++;
                    if(curX > mostX)
                        mostX = curX;
                    curX = 0;
                    curY = 0;
                    break;
                }
                else if(m_tipText[i] == '\n')
                {
                    i++;
                    if(curX > mostX)
                        mostX = curX;
                    curX = 0;
                    curY = 0;
                    break;
                }
                else
                {
                    curX += glutBitmapWidth(m_currFont,
m_tipText[i]);
                }
            }
        }
    }
```

```
    if(curX > mostX)
        mostX = curX;
    m_winWidth = mostX + 4;
    m_winHeight = lineNo * m_fontHeight + lineSpace * (lineNo - 1) + 4;
}
```

4.5.5 Auto arrange control in Taskbar

The control arrangement in Taskbar is a little different from other windows, all the controls will be arranged in one line, one by one. For this case, to simplify adding new controls to taskbar, it's better to arrange the control automatically according to the adding sequence of the controls. The developer can add controls without caring where to put it. Meanwhile, it ensures the controls arranged in a good layout.

Since all the controls maintain their position and size, it's very easy to query the previous neighbor's position and size when adding a new control. Then, we can calculate the new control's position easily.

```
void TaskBar::AddControlAtProperPosition(int ctlID,int imageID,double width,
char* help, int type)
{
    double newLeft = 0;
```

```
    VirtualControl* lastControl = 0;
    int index = m_Controls->GetLastIndex();
    if(index < 0)
            newLeft = -m_winWidth/2+m_btnSpace;
    else
    {
            lastControl = m_Controls->GetControlByIndex(index);
            newLeft = lastControl->GetLeft() + lastControl->GetWidth() +
            m_btnSpace;
    }
    if(type)
    {
            m_Controls->AddButton(ctlID,imageID,newLeft,m_winHeight/2-
m_btnTop-m_statusBarWidth,width,m_btnHeight,help);
    }
    else
    {
            m_Controls->AddLabel(ctlID,imageID,newLeft,m_winHeight/2 -
            m_btnTop-m_statusBarWidth,width,m_btnHeight,help);
    }
}
```

4.5.6 Switch Scoreboard between detail report and simple report

To leave more space for playing area, we supply two types of Scoreboard report – detail report and simple report. The detail report shows all the available data while the simple report just shows the data for the current frame. Switching between two types of Scoreboard is not so easy as expected. In window system, the original point is the top-left point. It's very easy to shrink or enlarge the window by hiding some controls and setting the new window size.

But in OpenGL window, it's a bit difficult. OpenGL's coordination system is different from that of windows; its original point (0,0) is at the center of the window rather than the left top. When changing the window size, if we just to decrease the window size, the coordination of all visible controls will be changed, the controls will be distorted as well. To solve this problem, we have to recalculate the new coordination and size of controls according to their relative positions and reset the controls' position and size when the window size is changed.

```
void ScoreBoard::PositionBoxes(double top, double ctlHeight, int columns)
{
    int height = (ctlHeight-2*m_topSpace)/columns;
    m_Controls->GetControlByID(ID_PLAYER_COLUMN)->SetTop(top-
m_topSpace);
    m_Controls->GetControlByID(ID_PLAYER_BOX_1)->SetTop(top-m_topSpace);
```

```
    m_Controls->GetControlByID(ID_PLAYER_BOX_2)->SetTop(top-m_topSpace);
    m_Controls->GetControlByID(ID_BREAK_COLUMN)->SetTop(top -
m_topSpace- height);
    m_Controls->GetControlByID(ID_BREAK_BOX_1)->SetTop(top -m_topSpace-
height);
    m_Controls->GetControlByID(ID_BREAK_BOX_2)->SetTop(top -m_topSpace-
height);
    m_Controls->GetControlByID(ID_SCORE_COLUMN)->SetTop(top -
m_topSpace- 2*height);
    m_Controls->GetControlByID(ID_SCORE_BOX_1)->SetTop(top -m_topSpace-
2*height);
    m_Controls->GetControlByID(ID_SCORE_BOX_2)->SetTop(top -m_topSpace-
2*height);

    m_Controls->GetControlByID(ID_PLAYER_COLUMN)->SetHeight(height);
    m_Controls->GetControlByID(ID_PLAYER_BOX_1)->SetHeight(height);
    m_Controls->GetControlByID(ID_PLAYER_BOX_2)->SetHeight(height);
    m_Controls->GetControlByID(ID_BREAK_COLUMN)->SetHeight(height);
    m_Controls->GetControlByID(ID_BREAK_BOX_1)->SetHeight(height);
    m_Controls->GetControlByID(ID_BREAK_BOX_2)->SetHeight(height);
    m_Controls->GetControlByID(ID_SCORE_COLUMN)->SetHeight(height);
    m_Controls->GetControlByID(ID_SCORE_BOX_1)->SetHeight(height);
    m_Controls->GetControlByID(ID_SCORE_BOX_2)->SetHeight(height);
}
```

4.5.7 Modal window

In some cases, the current window should always hold the input focus and stay on top until it is closed by user — modal dialog window, such as windows for setting environment variables and player modes. If the setting is not finished, switching to other window may cause unexpected results. To avoid these cases, the user is not allowed to work on another window until he has finished the work on the current window.

To implement the modal window, we declare a VirtualWindow pointer to store the current modal window. When users try to click on other window, the event handler will check if the modal window pointer is null or pointing to a modal window. If it's null, it will continue dealing with the current event, otherwise, ignore the event and bring the current modal window to the topmost so as to remind the user a modal windows is still shown.

The steps are as following:

First, declare a VirtualWindow pointer to store the pointer to current modal window.

VirtualWindow modalWin = 0;*

Then, set the modalWin point to the window, which is displayed as a modal window.

```
if(!modalWin)
{
        modeSetter->ShowWin();
        modalWin = modeSetter;
}
```

When handling events from a window, check if the modalWin is null in the event handler function. If modalWin is null, continue handling the event, otherwise, bring the modal window to top and ignore the event.

```
if(modalWin)
{
        modalWin->BringToFront();
        return;
}
```

4.5.8 Transform window coordinate to world coordinate

Window coordinate and world coordinate are two different coordination system used in the OpenGL applications. The original point (0,0) of window coordination system is at the top-left of window, and world coordination system's original point is at the center of the window.

World coordinate is used for most of the OpenGL operations, drawing, locating, moving, resizing. When handling a mouse event, we can only get the window coordinates of the mouse's location. If we want to know if the mouse position is above a specific control, we cannot compare the coordinates directly since the mouse position is in window coordinates and the control's position is in world coordinates. We have to transform the coordinate by mapping the window coordinates to world coordinates.

The following codes implements such a transformation by using OpenGL's glUnProject function.

```
WorldCoord ControlCollection::GetWorldCoord(double xm, double ym)
{
 WorldCoord wCoor;

 GLint viewport[4];
 GLdouble mvmatrix[16];
 GLdouble projmatrix[16];
 GLint realy;    //OpenGL y coordinate position
 double xw, yw, zw;
 glGetIntegerv(GL_VIEWPORT, viewport);
 glGetDoublev(GL_MODELVIEW_MATRIX, mvmatrix);
 glGetDoublev(GL_PROJECTION_MATRIX, projmatrix);
```

```
  realy=viewport[3] -(GLint)ym-1;        //viewport[3] is height of window in
pixels

  gluUnProject((GLdouble) xm, (GLdouble) realy, 1.0, mvmatrix, projmatrix,
viewport, &xw, &yw, &zw);

  wCoor.xw = xw;
  wCoor.yw = yw;
  return wCoor;
}
```

This function can only get the proper world coordinates if the window is not in perspective view, which means the world coordinates may not change when the distance between camera and target window is changed.

In the main window, which contains the snooker table, the case is more complicated. The main window is set in a perspective view by calling the following function.

```
gluPerspective(alpha,w/h, 0.5, 500.0);
```

When we transfer from window coordinates to world coordinates, we need to consider the view port. After getting *xw* and *yw* by calling *gluUnProject*, we need to do the following mapping for the real world coordinates at the TopView.

```
wCoor.xw  =  xw * top_positionZ/500;
wCoor.yw =  yw * top_positionZ/500;
```

In the Side View, the graphic is distorted; it's difficult to get the corresponding world coordinates from mouse's window coordinates. To avoid any discrepancy, we don't allow any exact mouse location operations in the Side View so as to simplify the calculation.

5. Enhancement

To make the game user interface meet the players' requirement better, besides Top View and Side View, a new view is added to the project – Player View.

5.1 Introduction to Player View

This view is almost the same as the Fine Adjust Window. It allows the players to observe the balls clearer, mainly the relationship between target ball and cue ball, on the snooker table along with the cue from the player's viewpoint by zooming in and zooming out. To avoid degrading the performance, the Player View is not linked tightly with the Fine Adjust Window. This means that any viewpoint change in the Player Viewer will not affect the Fine Adjust Window. Meanwhile, any viewpoint change in the Fine Adjust Window will not affect the Player View either, besides the hit point of cue on cue ball in the Fine Adjust Window.
In the Player Viewer, when the cue ball is shot, the camera will not follow the movement of the cue ball or target ball. The camera will still aimed at the original point of the center of the target ball. When the cue ball stopped, it will switch to Top View automatically since it's allowed to select a target ball only in the Top View.

5.2 Implementation of Player View

In the Player Viewer, the camera is always located behind the cue ball with the fixed distance and aim at the center of the target ball, as the figure below:

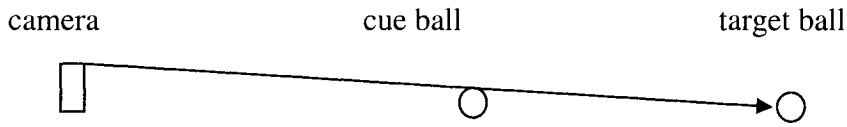

Figure 5.1 Camera and balls positions

To display at the proper viewpoint, we need to get the camera's position the point it aims at. What we know now is the position of the cue ball (cueX, cueY, cueZ), the position of the target ball (targetX, targetY, targetZ), and the distance between the camera and the cue ball. We need to calculate the X, Y, Z coordinate of the camera. Since the Z-coordinate of the cue ball, target ball and the camera are the same (2.7), we only need to consider the X, and Y coordinates of the camera.

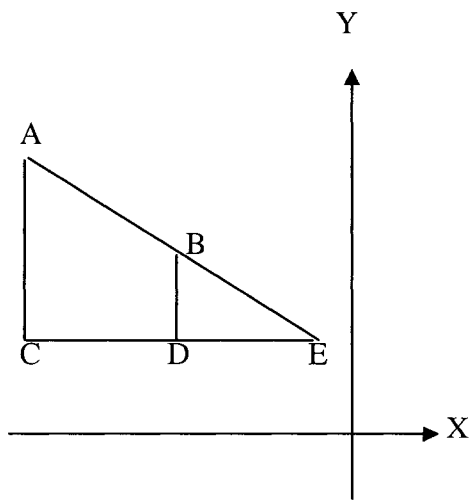

Figure 5.2 Calculate Camera Position

See figure at the left, suppose B is cue ball, E is target ball and A is the camera.

The distance between the cue ball and the target ball is:

$$|AE| = \sqrt{(targetX - cueX)^2 - (targetY - cueY)^2}$$

Assume the angle between AE and CE is A, so we can get:

$$\cos A = (cueX - targetX) / |AE|$$
$$\sin A = (cueY - targetY) / |AE|$$

Assume the distance between camera and cue ball is always Dcc, we can get:

cameraX = Dcc* cosA + cueX

cameraY = Dcc * sinA + cueY

Then, we can use *glutLookAt* function to get the view we want.

The algorithm is implemented in the display function as below:

```
if (VIEW_MODE == PLAYER_VIEW)
{
        if(isAiming)
        {
                player_aimX = cueBall->get_targetX();
                player_aimY = cueBall->get_targetY();
                player_aimZ = 2.7;
                float a=sqrt((( player_cueball_origX -
                player_aimX)*( player_cueball_origX -
                player_aimX))+(( player_cueball_origY -
                player_aimY)*( player_cueball_origY - player_aimY)));
                 if( a == 0)
                 {
                   pointX = player_cueball_origX + 1;
                   pointY = player_cueball_origY + 1;
                 }
                 else
                 {
                    cosA = (player_cueball_origX - player_aimX)/a;
                    sinA = (player_cueball_origY - player_aimY)/a;
                    pointX = (player_cueball_distance * cosA) +
                    player_cueball_origX;
                    pointY = player_cueball_origY + player_cueball_distance *
                    sinA;
                 }
                 player_DistanceX = pointX;
                 player_DistanceY=pointY;
        }

         gluLookAt(player_DistanceX,player_DistanceY,player_DistanceZ,
                        player_aimX, player_aimY, player_aimZ,  0.0,0.0,1.0);
}
```

The following picture shows the Snooker table at the Player View.

Figure 5.3 Snooker table at Player View

When implementing the player view, we met a problem with the performance of drawing. We allow the user to view the balls' movement at the player view. But, if the user zooms too close to the target ball, especially when there are a lot of balls in the view, the balls' movement becomes very slowly since it takes much more time to draw the objects. To solve this problem, we reset the alpha of camera to the original value and redraw the table immediately when the shot button is clicked, so that the user can view the balls moving at the player view smoothly.

6. Discussion

This OpenGL-based UI component architecture is designed to support the 3D UI implementation. It implements an extensible hierarchical structure for managing user interface components, mainly for windows and controls. It takes advantage of the object-oriented development. During implementing 3D user interface of Snooker Simulator using this architecture, we feel it's helpful in following aspects.

- Create UI components easily and quickly. Just need to declare a new component and implement little overridable functions. Most of the technical details are transparent to the final developer;
- Easy to maintain. Any modification on the base class will affect all the children objects. It's very easy to change the performance of UI components.
- Extendible. Easy to add new features to the UI components by adding new interface.

This architecture increases our development speed to a great extend. Meanwhile, its feature of easy maintenance also saves us lots of time in defect-fixing and feature enhancement.

But, since the current classes were originally designed for the UI implementation of Snooker Simulator (3D) only, not designed as a generic UI components library

which can be used anywhere, the design and implementation is not perfect, still need to be improved.

- The hierarchical structure needs improvement. VirtualControl and VirtualWindow have some common features, so a base class may be created to handle the common attributes.
- Need to wrap all global callback functions and variable required by OpenGL to make the library as pure object-oriented;
- Attributes and methods of each class need to be re-designed.
- Implementations of each method, especially in the base class need to be improved.

7. Conclusion

Porting Snooker Simulator from 2D to 3D is a complicated project. Though our project mainly focuses on graphic and UI implementation, we experienced almost every step of software development -- understanding the existing application, redesign, 3D object modeling, 3D graphic and UI implementation, testing, system integration, system testing and software maintenance.

This project involves a lot of concepts and technology across many fields of computer science, such as software reverse engineering, object-oriented analysis, design and programming, computer graphic programming, User Interface design, software and system testing. Through working on this project, we got a big improvement with all these knowledge and a deeper understanding on software development, esp. the game development.

After long-time hard working, we preserve all the functionalities of 2D version in the new 3D Snooker Simulator, implement a real-like, smooth 3D graphic and friendly game-like user interface. We achieved almost all our project goals.

Of course, there're still some aspects need to be improved, such as 3D movement performance. Anyhow, we still can feel some flashing when ball is moving, especially when the machine is not fast enough. In addition, though we're trying to make the application to be platform-independent, there still exist two functions

depending on Win32 API – MessageBox and PlaySound. To make 3D Snooker Simulate to be able to run other operating system, some modifications are still needed.

In conclusion, after working on this project, we really feel 3D game development is an interesting, exciting and challenging field. We're fascinated by it.

8. References

[1]. OpenGL Architecture Review Board, OpenGL Programming Guide, second edition, 1997, Addison-Wesley Developers Press. ISBN: 0-201-46138-2

[2]. Peter Grogono, The Snooker Simulation
http://www.cs.concordia.ca/~faculty/grogono/snooker.html

[3]. Dave Astle, et al , OpenGL Game Programming, 2002, Premier Press. ISBN: 0761533303. http://www.codeguru.com/columns/gamedev/OpenGL-01.html

[4]. Xiang, Shiming OpenGL Programming and Examples, 1999, Publishing House of Electronic Industry, China. ISBN:7-5053-5625-9/TP • 2880

Appendix A. Main Source code for 3D implementation

1. main function

```
int main (int argc, char *argv[])
{
        srand(time(NULL));
        glutInit(&argc,argv);
        windowWidth = SIZE;
        windowHeight = SIZE;

        mainWindow = new MainWindow(true);
        mainWindow->CreateWin(0,0,windowWidth,windowHeight);
        InitParameterManager();
        InitObjects();
        mainWindow->Init();

        mainMenu = new MainMenuWin(mainWindow->GetThisWindowID(),
        parameterManager);
        mainMenu->CreateWin(0,0,w,h-40);
        toolTip = new ToolTipWin(mainWindow->GetThisWindowID());
        toolTip->CreateWin();
        ctlHelper = new PanelHelper(mainWindow->GetThisWindowID());
        ctlHelper->CreateWin(0,420,140,120);
```

```
ctlHelper->SetHideDirection(HIDE_DIR_BOTTOM);
ctlHelper->SetOriginalVisible(false);
scoreBoard = new ScoreBoard(mainWindow->GetThisWindowID(),
parameterManager, 0);
scoreBoard->CreateWin(0,0,400,140);
taskBar = new TaskBar(mainWindow->GetThisWindowID());
taskBar->CreateWin(0,h-36,w,36);
adjustWin = new AdjustWin(mainWindow->GetThisWindowID(),
parameterManager);
adjustWin->CreateWin(adjustLeft,adjustTop,adjustWidth,adjustHeight);
controlPanel = new PanelControl(mainWindow->GetThisWindowID(),
parameterManager);
controlPanel ->CreateWin(ctlLeft, ctlTop, ctlWidth, ctlHight);
controlPanel->SetParentSize(w,h);
speedSetter = new PanelCueSpeed(adjustWin->GetThisWindowID(),
parameterManager);
speedSetter->CreateWin(speedLeft,speedTop,speedWinWidth,
speedWinHeight);
ctlCueBall = new PanelCueBall(adjustWin->GetThisWindowID(),
parameterManager);
ctlCueBall->CreateWin(5,speedTop+speedWinHeight,cueWinWidth,
cueWinHight);
```

```
        fineAdjuster = new FineAdjustWin(adjustWin->GetThisWindowID(),
        parameterManager);
        fineAdjuster->CreateWin(2*SPIN_RAD + 2 + 7,
        speedTop+speedWinHeight, 184, 2*SPIN_RAD + 2);
        table2->InitTable();
        fineAdjuster->Init();
        adjustWin->SetCtlPos(2*SPIN_RAD + 2 + 7+184+4, speedTop +
        speedWinHeight + 20);
        control->AddControls(taskBar,controlPanel, ctlCueBall,speedSetter,
        scoreBoard);
        helpWin = new GluiHelpWin(15);
        helpWin->CreateWin(250,150);
        glutMainLoop();
        return 0;
}
```

2. *Display event handler*

```
void OnDisplay(void)
{
        int currentWin = glutGetWindow();
        glutSetWindow(currentWin);
        if(currentWin == mainWindow->GetThisWindowID())
                display();
```

```
else if(currentWin == mainMenu->GetThisWindowID() && mainMenu->
IsVisible())
        mainMenu->Display();
else if(currentWin == toolTip->GetThisWindowID() && toolTip->
IsVisible())
        toolTip->Display();
else if(ctlHelper && currentWin == ctlHelper->GetThisWindowID() &&
ctlHelper ->IsVisible())
        ctlHelper->Display();
else if(setWin && currentWin == setWin->GetThisWindowID() &&
setWin->IsVisible())
        setWin->Display();
else if(currentWin == scoreBoard->GetThisWindowID() && scoreBoard-
> IsVisible())
        scoreBoard->Display();
else if(currentWin == taskBar->GetThisWindowID() && taskBar->
IsVisible())
        taskBar->Display();
else if(currentWin == adjustWin->GetThisWindowID() && adjustWin->
IsVisible())
        adjustWin->Display();
else if(currentWin == controlPanel->GetThisWindowID() &&
controlPanel->IsVisible())
```

```
            controlPanel->Display();
        else if(currentWin == speedSetter->GetThisWindowID() && speedSetter-
        > IsVisible())
            speedSetter->Display();
        else if(currentWin == ctlCueBall->GetThisWindowID() && ctlCueBall->
        IsVisible())
            ctlCueBall->Display();
        else if(currentWin == fineAdjuster->GetThisWindowID() &&
        fineAdjuster->IsVisible())
            fineAdjuster->Display(); //display_fine();
        else if(nominator && currentWin == nominator->GetThisWindowID()
        && nominator->IsVisible())
            nominator->Display();
}
```

3. Draw a control

```
void ButtonCtl::DrawControl()
{
    if(!m_ctlVisible)
        return;
    if(m_borderStyle == BORDER_3D)
    {
```

```
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, red);
        glPushMatrix();
        glBegin(GL_LINES);
        glVertex3f(m_leftPos+1,m_topPos,1);
        glVertex3f(m_leftPos+m_Width,m_topPos,1);
        glVertex3f(m_leftPos,m_topPos-m_Height,1);
        glVertex3f(m_leftPos,m_topPos,1);
        glEnd();

        glColor3f(0.0,0.0,0.0);
        glBegin(GL_LINES);
        glVertex3f(m_leftPos+m_Width ,m_topPos,1);
        glVertex3f(m_leftPos+m_Width,m_topPos-m_Height,1);
        glVertex3f(m_leftPos+m_Width,m_topPos-m_Height,1);
        glVertex3f(m_leftPos+1,m_topPos-m_Height,1);
        glEnd();
        glPopMatrix();
}
if(m_imageIndex >= 0 || (!m_ctlEnabled && m_disableImageIndex >= 0))
{
        int imageIndex = m_imageIndex;
        if(!m_ctlEnabled && m_disableImageIndex >= 0)
                imageIndex = m_disableImageIndex;
```

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, m_texturePtr[imageIndex]);
glBegin(GL_QUADS);
if(m_ctlEnabled && m_mouseOnCtl)
{
        glTexCoord2f(0.0, 0.0);   glVertex3f(m_leftPos-2,
        m_topPos - m_Height-2, 0.0);
        glTexCoord2f(0.0, 1.0);   glVertex3f(m_leftPos-2,
        m_topPos+2, 0.0);
        glTexCoord2f(1.0, 1.0);   glVertex3f(m_leftPos + m_Width
        +2 , m_topPos+2 , 0.0);
        glTexCoord2f(1.0, 0.0);   glVertex3f(m_leftPos + m_Width
        +2, m_topPos - m_Height-2 , 0.0);
}
else
{
        glTexCoord2f(0.0, 0.0);   glVertex3f(m_leftPos, m_topPos
        - m_Height, 0.0);
        glTexCoord2f(0.0, 1.0);   glVertex3f(m_leftPos, m_topPos,
        0.0);
        glTexCoord2f(1.0, 1.0);   glVertex3f(m_leftPos +
        m_Width , m_topPos , 0.0);
```

```
                glTexCoord2f(1.0, 0.0);   glVertex3f(m_leftPos +
                m_Width , m_topPos - m_Height , 0.0);
        }
        glEnd();
        glDisable(GL_TEXTURE_2D);
}
else
{
        glDisable( GL_LIGHTING );
        glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
        glPushMatrix();
        glColor4fv(RGBColors[m_backColor]);
        glBegin(GL_POLYGON);
        glVertex3f(m_leftPos+1,m_topPos,1);
        glVertex3f(m_leftPos+m_Width ,m_topPos,1);

        glVertex3f(m_leftPos+m_Width,m_topPos-m_Height,1);
        glVertex3f(m_leftPos+1,m_topPos-m_Height,1);
        glEnd();
        glPopMatrix();
}
glEnable(GL_LIGHTING);
```

```
        if(m_ptrText != 0)
        {
                glDisable( GL_DEPTH_TEST );
                glDisable( GL_LIGHTING );
                glColor3fv(RGBColors[m_foreColor]);
                bitmap_output(m_leftPos + 2,m_topPos - m_Height + 2,
                m_ptrText,  GLUT_BITMAP_TIMES_ROMAN_10);
                glEnable( GL_LIGHTING );
                glEnable( GL_DEPTH_TEST );
        }
}
```

Appendix B. User Manual

Introduction

Introduction to the 3D Snooker Simulator

The 3D Snooker Simulator is a real 3D snooker game. Current version is a window-based standalone version.

It supplies four play modes – manual, track, advise, and auto.

♦ **Manual.** In manual mode, you have to do all the work of playing a shot yourself. To set a player to manual mode, first click on the `Manual` button for either Player 1 or Player 2, and then enter the player's name in the `Name` box at the top of the dialog box.

♦ **Tracking.** Tracking mode is similar to manual mode except that Snooker displays the "tracks" that the balls will follow. The tracks are not displayed until you have set the ball's speed. After you have set the ball's speed, Snooker responds to any change in the controls by re-displaying the tracks. To set a player to manual mode, first click on the `Tracking` button for either Player 1 or Player 2, and then enter the player's name in the `Name` box at the top of the dialog box.

The track of a ball usually consists of a white part and a black part. The white part corresponds to sliding motion and the black part to rolling motion. The cue ball will slide after impact unless you hit it on the "natural roll" line, indicated by a horizontal red line on the ball image at the center of the control window.

♦ **Advise.** In advise mode, Snooker suggests a shot for you to play. You can accept the suggestion and simply play the shot, or you can use the controls to change the shot.

To set a player to advise mode, first click on the `Advise` button for the player. Next, choose an advisor from the `Auto-Players` window near the bottom of the dialog box. Finally, enter the player's name in the `Name` box at the top of the dialog box

- **Automatic.** In automatic mode, Snooker plays the shot itself. To set a player to automatic mode, first click on the `Automatic` button for the player and then choose a player from the `Auto-Players` drop-down list.

The players are allowed to change the environment properties, such as ball-ball collision, ball-cushion collision, etc.

To gain the best 3D performance, the player can change display factor and time increment. The Snooker Simulator supplies a Time-testing function to calculate the best time increment automatically.

The game is playing at full-screen mode. The player can go to desktop by clicking the Minimizing the game button at any time and go back to the game by clicking the icon of the game on the system task bar.

Introduction to Snooker Game

How to Play Snooker

The objective of snooker is to pot balls. If the position prevents you from potting a ball, you try to prevent your opponent from potting balls.

The game begins with 15 red balls on the table. A player starts by trying to pot a red. If the shot is successful, the player nominates a colored ball, and tries to pot that. If the color goes in, the player goes for another red, and so on.

When the last red has been potted, the player nominates and tries to pot a color. After this color has been potted, it is re-spotted, and all of the colors are potted in order of increasing value: yellow, green, brown, blue, pink, and black.

If a player fails to pot a ball, the "innings" ends, and the other player comes to the table. The following shots are "fouls".

- The cue ball hits the wrong ball first. (The "wrong ball" is a color if the player is aiming for reds, and a red or a color other than the nominated color if the player has nominated a color.)
- A ball other than a red or the nominated color goes into a pocket. Potting more than one red in a single shot is allowed.
- The cue ball does not hit any balls at all.
- The cue ball goes into a pocket, either directly or after hitting another ball.
- The cue ball goes off the table. (This should not happen in The Snooker Simulation.)

After a foul, the player's innings ends and the other player comes to the table with an option: to play a shot or to "pass". In The Snooker Simulation, the `Pass` button (on the right of the `Shoot` button) is highlighted after a foul. The incoming player passes by clicking on it.

In The Snooker Simulation, it is not usually necessary to "nominate" a color, because Snooker can determine which ball the cue ball will strike first. However, if you set the shot up in such a way that the cue-ball will not strike a color, Snooker will display a dialog box asking which color you are trying to hit.

Reds that are potted remain off the table. After a color has been potted, it is returned to the table and "spotted". The rules for spotting a ball are quite complicated, because there may

be another ball on or close to the color's normal spot. You do not need to worry about these rules, because Snooker always spots balls correctly.

Snookers

The word "snooker" is used in two senses in the game of Snooker. If your opponent plays a shot that leaves the balls positioned so that you cannot hit any legal ball directly, you are "snookered". There is no score or penalty associated with a snooker, but your opponent is hoping to gain points either because you miss or because, in escaping from the snooker, you will leave an easy shot.

Suppose that your opponent plays a foul shot that leaves you without a clear shot on a legal ball. You have a "clear shot" if you can hit the object ball at any desired angle without your shot being blocked by another ball. If you do not have a clear shot on at least one legal ball, you are "snookered by a foul" and you have a "free ball". This means that you can nominate any ball on the table and try to pot it. The ball you have nominated is treated, for scoring purposes, as if it was a legal ball.

For example, suppose there is one red on the table. Your opponent tries to pot it, but misses altogether and leaves you without a clear shot. You claim a "free ball", nominate the green as your free ball, and pot it. You score one point for the green (as if it was a red), and the green is repotted. You then nominate a color and try to pot it, just as if you had potted a red.

Since Snooker implements the "snookered by a foul" and "free ball" rules, you do not have to worry about the details.

Scoring

Since Snooker keeps the score in the window that appears below the table image on the screen, it is not necessary to describe scoring in detail here. Here is a summary of the scoring rules.

- If you pot a red, you get one point. If more than one red goes into a pocket, you get one point for each red potted.
- If you pot a color, you get the value of the color: yellow, 2 points; green, 3 points; brown, 4 points; blue, 5 points; pink, 6 points; and black, 7 points.
- If you play a foul shot, your opponent receives penalty points. The minimum penalty is 4 points. If the foul involves a ball with a value higher than 4 points, the penalty is the value of the ball. A foul on the blue incurs 5 penalty points; on the pink, 6 penalty points; and on the black, 7 penalty points.

The program displays a quantity called "Reserve" in the Information box. The reserve is calculated as follows:

1. Calculate the "points on the table" assuming that the current player will pot a black after each red. Thus the number of points on the table is 8 times the number of reds plus the value of all the colors, namely, 27. For example, at the beginning of a frame, there are 15 reds on the table and each player has a reserve of $8 \times 15 + 27 = 147$.
2. Add the current player's score.
3. Subtract the other player's score.

The reserve is therefore the amount by which the current player would win by playing a "perfect" game. A player whose reserve is negative must force the other player to foul in order to win the game. An automatic player with a small, negative reserve may attempt to snooker the other player. When the shortfall gets large enough, the player resigns.

Installation

System Requirement

- Hardware (suggested):

  P4 1.5G CPU, 256M memory, at least 10M available hard disk space. 32M Video memory. 3D graphic accelerate card. Sound card and speaker.

- Software:

  Windows 9x, 2000 (not tested on XP and NT). Glut32.dll (included with game)

Install/Uninstall:

- Install: Double click Snooker_Setup.exe, follow the installation wizard, select the target folder. The installation wizard with create an icon on desktop and a shortcut in the Start->Programs menu system.

- Uninstall: click Start on the taskbar, click Programs, go to Snooker Simulator (3D), and click Uninstall Snooker Simulator. All installed files will be removed from the machine.

Play game

Start the Snooker Simulator

- Double click the Snooker Simulator (3D) icon on the desktop; or
- Click Start on the taskbar, click Programs, go to Snooker Simulator (3D), click 3D Snooker Simulator

Introduction to Scenes:

1. Main menu page

After start the Snooker Simulator, the Main Menu Page is the first scene your will meet.

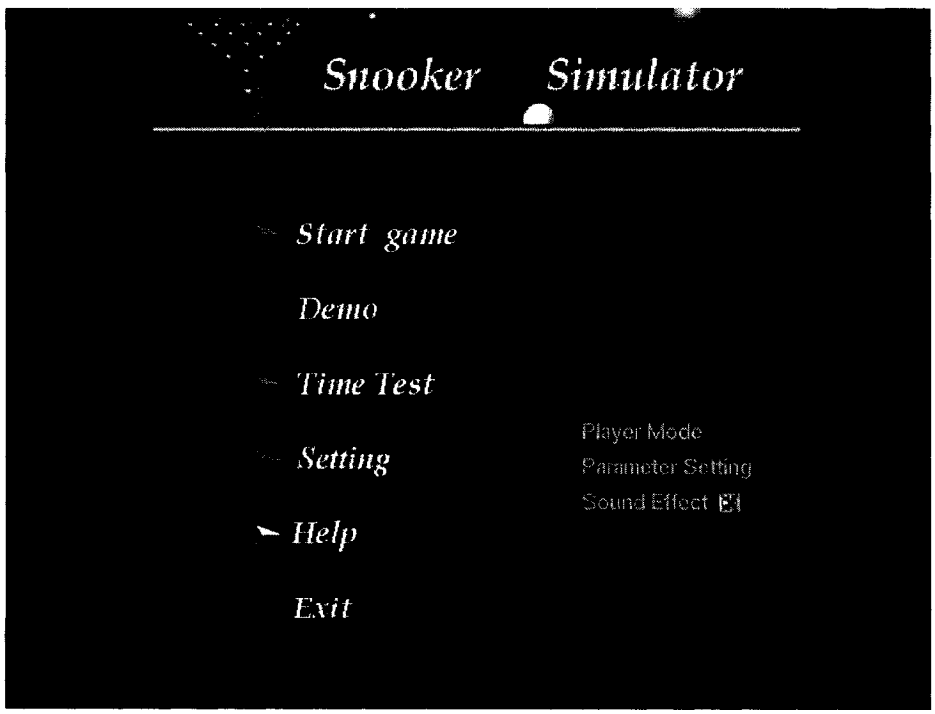

Figure B.1 Main menu page

In this page, the player can:

- Start a game. Click Start game. The default play mode is manual for all players
- Demo. Click Demo. The compute will keep playing game automatically till the player stop it
- Time testing. Click Time test. The computer will simulate a full-speed hit and calculate the best Time increment according to the computer's performance
- Setting. Click Setting. Three sub-functions supplied.
    - Player Mode. Change players' properties
    - Parameter Setting. Change environment parameters
    - Sound Effect. Toggle to turn on/off sound effect
- Leave the game, click Exit
- Read helps, click Help, then click on the help topic

2. GAME PAGE

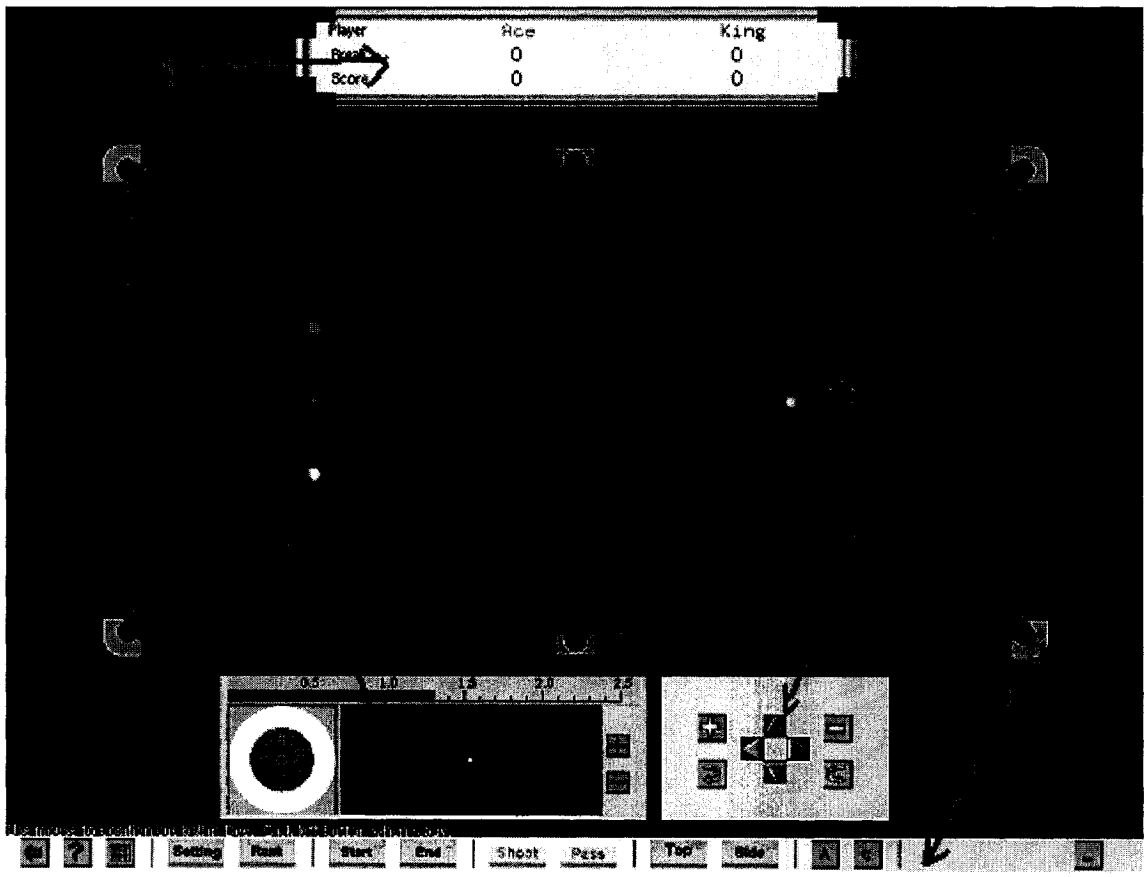

Figure B.2 Game page

The Game page consists of 5 parts:

- **Scoreboard**: show players current score and statistics on all the frames;
- **Snooker table**: the area where the players actually play the game
- **Adjust Window**: set cue speed, where the cue hits on the cue ball and adjust the hitting position between cue ball and target ball more precisely
- **Control panel**: switch between different views, adjust the camera's position and focal distance
- **Taskbar**: list all the functions available for the game

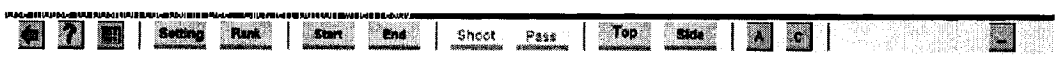

Figure B.3 Taskbar

From left to right, the buttons are for:

- Finish game and go back to main menu
- Show help
- Toggle the Scoreboard between detail report and simple report
- Show setting panel
- Show rank of auto players
- Start a game
- Stop the current game
- Shoot
- Pass
- Switch to Top view of the snooker table

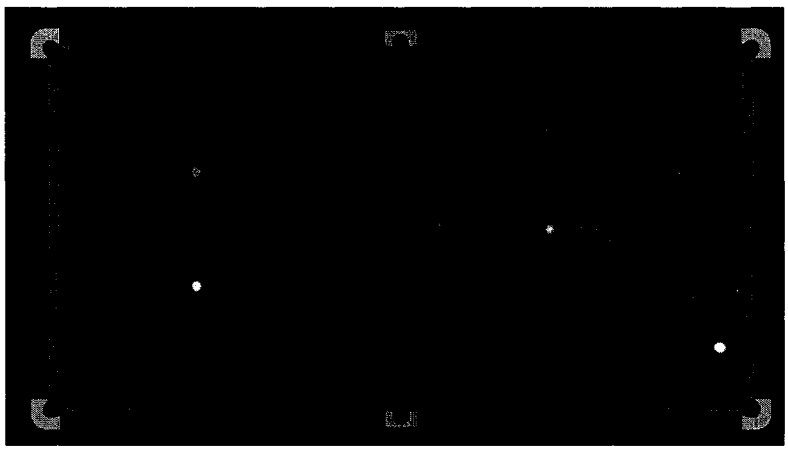

Figure B.4 Snooker table at Top View

- Switch to Side view of the snooker table

Figure B.5 Snooker table at Side View

- Hide/Show the adjustment window
- Hide/Show the control panel
- Minimize the game

Playing a Shot

To play a shot, you first have to set the "controls". There are four controls to set: they determine the coarse direction, fine direction, speed, and spin of the cue ball.

- **Coarse Direction.** To set the direction, move the mouse cursor over the playing-area of the table. (The cursor is a small circle while it is within the playing-area.) Choose a point through which you want the cue ball to pass, and click the left mouse button.

- **Fine Direction.** Move the mouse cursor into the 3D view (fine adjustment) at the bottom right of the Adjustment Window. Clicking the left mouse button in this area will make small changes to the direction of the cue ball. The amount by which the direction changes depends on how far the mouse cursor is from the center of the 3D view: if it is close to the center, the change will be very small. If the mouse cursor is left of center, the direction changes in the counterclockwise direction; if the mouse cursor is right of center, it changes in the clockwise direction.

- **Speed.** Move the mouse cursor into the calibrated scale at the top of Adjustment Window. The real speed for your current mouse position will display. Click on the position with your ideal speed to set the cue speed. The red, thermometer-like bar indicates the speed you have chosen. The maximum speed is 2.5 meters per second.

- **Spin.** Move the mouse cursor into the white and green circular area at the bottom left of the Adjustment Window. The small spot in the circle will move to the cursor position. This

spot corresponds to the point where the cue will strike the cue ball. If it is near the top of the green area, you are applying top spin ("follow"); a point near the bottom corresponds to back spin ("draw"); and points to the left and right sides correspond to left and right side-spin ("english") respectively. You cannot move the impact point outside the green circle because impacts near the edge of the ball are likely to cause a miscue.

If the player is in manual mode, there will be no visible response to changing a control except that the 3D view will change in response to a change of direction. In tracking and advise modes, Snooker will display the paths that the balls will take each time you change a control.

When you have adjusted the controls to your satisfaction, move the mouse cursor to `Shoot` button on the taskbar and click it. The `Shoot` button is highlighted whenever the current player is allowed to play a shot.

If you reach the adjust limitation, no action but shooting will take effect.

Question and Answer

1. What is DEMO

Snooker will play until you stop it. Each frame will be played by a randomly chosen pair of automatic players. To stop the demonstration, click the END on the taskbar or press the ESC key, the demo will stop until the balls come to rest at the end of a shot.

2. What is Time Testing

The program will perform a full-speed hitting to find the best value for the parameter `Time Increment'. The players can take this suggested value by click "Yes" when a message is prompted.

3. How to Set Players' mode

Click Setting->Player Mode on Main Menu Page, or click Setting on the taskbar.

A Player mode setting window will be shown as:

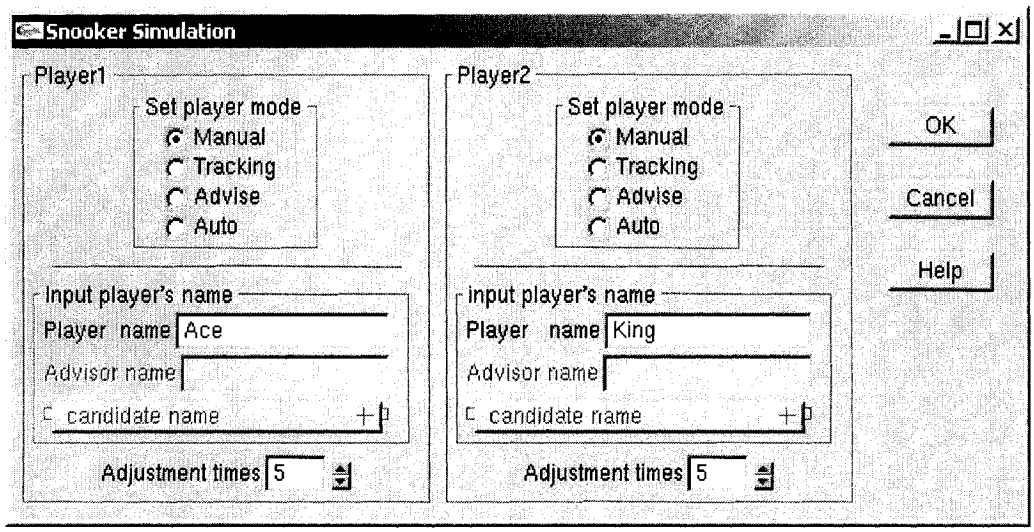

Figure B.6 Player mode set window

In this window, you can set properties for each player. You can:

- Change player mode:

  For Manual and tracking mode, you have to input the player's name

  For Advise and auto mode, candidates and their performance will be shown, select one person, the name will input in the proper field automatically.

- Change adjustment times: maximum times allowed to adjust before shooting

Click OK to save the modifications, click cancel to cancel the modifications. For more information click help.

4. How to set the Environment Parameters

Click Setting->Parameter Setting on Main Menu Page, or click Setting on the taskbar. A Player mode setting window will be shown as:

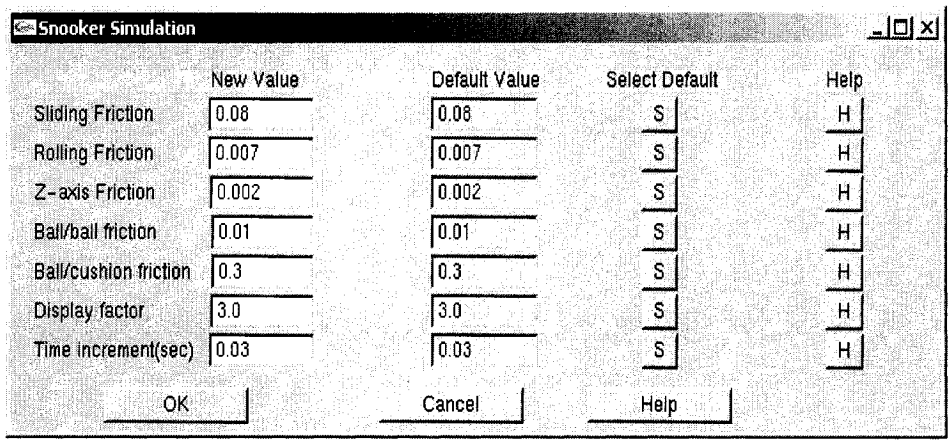

Figure B.7 Environment parameters set window

In this window, you're allowed to change values for Sliding Friction, Rolling Friction, Z-axis Friction, Ball/ball friction, Ball/cushion friction, Display factor and Time increment.

Change value: click the proper white edit box, input new value

Reset the value to default: click the S button for the target parameter

For the help on this parameter, click the H button for the target parameter

Click OK to save the settings, click Cancel to reserve the old setting.

For more information, click Help

See details on Q&A14 – How about Environment Parameters

5. What's the difference between Detail report and Simple report?

Detail report contains:

Players' name, current player, break, Score, Frames, Cumulative and High breaks

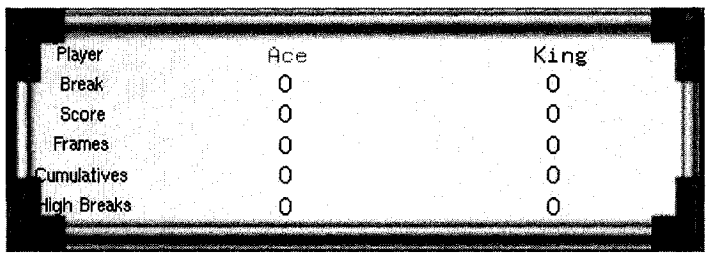

Figure B.8 Detail Report

Simple report contains:

Players' name, current player, break, Score

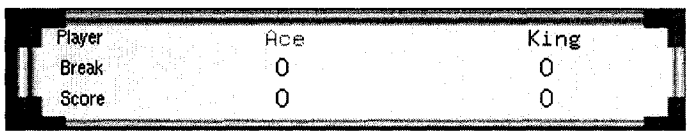

Figure B.9 Simple Report

How to switch:

- Click on the Scoreboard, or
- Click the  button on the taskbar.

6. What is the Tracking Mode

Tracking mode will show the moving track of balls when you select a target ball. This will help the player preview the result of his shoot.

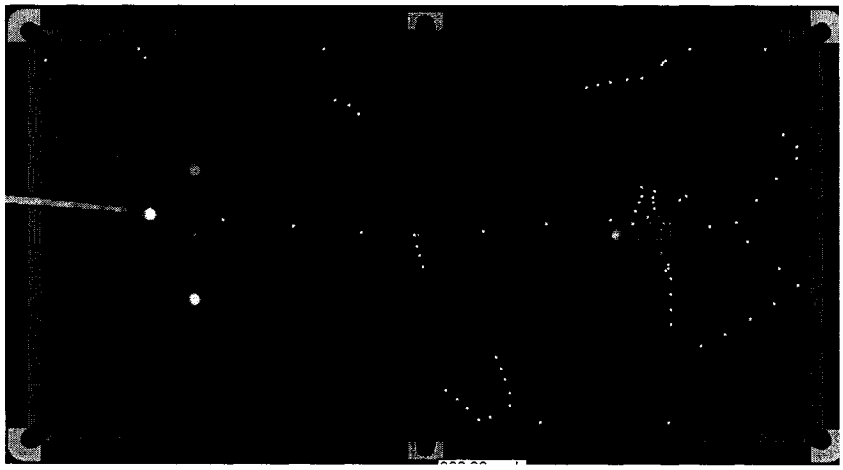

Figure B.10 Tracking mode

7. What is the Advisor Mode

Advisor mode is still played by you, but the your selected advisor will give you suggestion before you shoot (show movement tracking as Tracking mode), you can adjust speed or direction on the suggestion shoot or ignore the suggestions.

8. Can I leave the game temporarily without closing it?

Yes. In the Game page, click the rightmost Minimize button on the taskbar will make you go back to desktop. Click the icon on the system taskbar will bring you back to the game.

9. How to close the game

If you are in Main Menu Page, click Exit or press Escape

If you are in Game Page, click leftmost button on the taskbar or press Escape will bring you back to Main Menu Page, then click Exit or press Escape again.

10. How to adjust in Adjustment Window

This window allows you to adjust cue speed, hitting position of cue on cue ball, hitting position between cue ball and target.

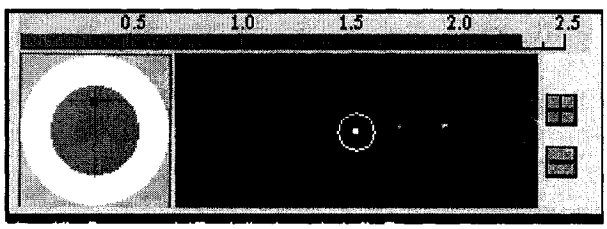

Figure B.11 Fine-Adjust window

To set cue speed: move the mouse about the scaled bar on the top, the real speed will be shown, when get the ideal speed, click the mouse on the spot.

To set the hitting position of cue on cue ball (direction): move mouse to the green area of the white ball on the left, when cursor changed to cross, you're in the valid area, click on the ideal position.

To adjust the hit position between cue ball and target ball: The white circle is the mapping of cue ball, click on the right window will move the circle a bit to left or right, depend on the distance between mouse and circle (the larger the distance is, the more the circle will move). If the ball is too small to too big, click the + and – button on the right.

Attention: During adjusting, if new ball move between the cue ball and target ball, it will not show in the adjustment window. It's suggested to observe the snooker table at the same time to ensure no other ball will block the cue ball.

11. What is Adjustment Limit

In manual modes (that is, **Manual**, **Tracking**, and **Advise**), you can set the number of adjustments that can be made for each shot in the `Player Mode setting window`. If the Adjustments field is set to a small number, such as 5 or 10, the player can make only a limited number of changes before playing the shot. Two players with different skill levels can used the Adjustments field as the basis for handicapping.

12. What is Automatic Players

You can adjust the level of skill and style of play of each automatic player by editing the text file `players.dat` (see the section "**Editing the Data Files**" below). The program maintains rankings for the players. You can see the current rankings by clicking on `Rank` Button on the taskbar.

13. How to edit the Data Files

The folder in which you have stored The Snooker Simulation contains two data files: "players.dat" containing information about the automatic players and "settings.dat" containing information about the table parameters. By editing these files you can customize Snooker and avoid wasting time changing settings when the program is running.

Both files are in "plain ASCII" format. This means that you should edit them with an ASCII text editor, such as Microsoft Notepad. If you edit them with a word processor, such as Microsoft Word or Corel WordPerfect, you should save them in "ASCII" or "text" mode after making the changes.

The file "players.dat" should look something like this:

```
#comment          ACC LON STR PRE POS DEF HI  CLU GTL PER
Hardy Potts       10  10  5   10  10  10  10  10  10  10
Careful Kate      7   7   10  10  10  10  6   3   9   10
Sharky Pete       10  10  2   8   10  10  2   2   5   10
Gofer Black       7   6   9   6   9   6   10  5   8   8
Artful Ann        8   4   7   8   5   2   5   10  4   7
Willy Wacker      4   9   9   2   2   2   5   10  0   4
Duffer Dave       2   2   8   1   1   1   8   3   5   1
#end
```

Snooker ignores lines beginning with "#". Each of the other lines consists of the name of a player followed by exactly ten numbers. The numbers must be integers (no

fractions or decimals), not less than zero, and not greater than 10. The numbers determine the style of the automatic player. They are answers to the following questions, with 0 indicating "definitely not" and 10 indicating "definitely yes".

1. I am an accurate potter.
2. I am good at long pots.
3. I prefer straight pots to cuts.
4. I prepare each shot very carefully.
5. I am good at positional play.
6. I am good at defensive play.
7. I go for balls with high scoring value.
8. I break up clusters of balls whenever possible.
9. I prefer gentle shots to strong shots.
10. I give up only when I have no chance of winning.

You can change the values of the existing players or you can create new players of your own.

The file "settings.dat" contains information that describes properties of the snooker table. Each line consists of a number followed by an explanation in English. Only the number is important; Snooker ignores the explanation. The numbers are all fractions less than 1, except the display factor, which is an integer greater than 1. Snooker does not check to see if the numbers in the file are reasonable. If you enter unlikely values, Snooker may behave — or misbehave — in curious ways. For example, if rolling friction is zero, the balls will roll for a very long time. If ball/cushion friction is also

zero, they will roll forever! The practical problem with zero friction is that the program will try to construct infinitely long trajectories for the balls and will crash.

Here is the file `settings.dat` with default value.

```
0.08    Sliding friction
0.007   Rolling friction
0.002   Spin friction
0.01    Ball/ball friction
0.3     Ball/cushion friction
3       Display factor
0.03    Time increment (seconds)
```

The data in this file is used to set default value, during the game, you can change all the settings in Parameter Setting window. But the new setting will not saved to this file.

14. How about Environment Parameters

The Snooker Simulation allows you to change the properties of the table you are playing on and to adjust the speed of the game to your computer. To change the properties, select `Settings` from the menu bar and then `Parameters`. Snooker will open a dialog box that you can use to read and change various values.

♦ **Sliding Friction.** This is the coefficient of friction between a sliding ball and the cloth on the table. A ball slides for a short time after being struck by the cue or another ball. The effect of friction is particularly important for the cue ball: a high coefficient of friction makes back spin ("draw") and top spin ("follow") more effective.

♦ **Rolling Friction.** After sliding for a while, a ball begins to roll. Its speed is affect by friction, but much less than when it is sliding. Rolling friction is typically about one-tenth the value of sliding friction.

♦ **Spin Friction.** Spin friction is the friction experienced by a ball spinning about a vertical axis. (Spin friction is also called "Z-axis friction" because the Z-axis is the vertical axis.) The vertical spin affects the behavior of a ball when it hits a cushion and, to a lesser extent, when it hits another ball. If spin friction is high, the ball will lose its vertical spin as it travels and will not be deflected by spin when it hits a cushion. If spin friction is low, the ball will retain its vertical spin and be deflected even at a distant cushion.

♦ **Ball-Ball Friction.** The ball-ball friction is the friction experienced by balls that are in contact during a collision. It is small and, ideally, would be zero. After a collision between frictionless balls, one ball travels along the line of centers at the time of collision and the other travels at right angles to this direction. If there is ball-ball friction, these directions are altered slightly. The automatic players in Snooker occasionally miss long pots: this is because they do not allow for ball-ball friction. If you set ball-ball friction to zero, you will find that the automatic players do not miss pots as often as they do when the default value is used.

♦ **Ball-Cushion Friction.** This is the friction experienced by a ball when it is sliding against a cushion. If the ball rolls directly into a cushion, there is very little sliding, and ball-cushion friction has little effect. If, however, the ball hits the

cushion obliquely, or is spinning about a vertical axis, the ball-cushion friction will affects its motion after the impact. If ball-cushion friction is small (less than 0.1), the cushions will be "fast" (balls will not lose much speed) but side-spin will have little effect. If ball-cushion friction is large (0.4 or more), the cushions are slower but side-spin is effective. When ball-cushion friction is set to 0.3 (the default value), the effects are quite close to what you would see on a real snooker table.

- **Display Factor.** The display factor determines how often the screen is refreshed while the balls are in motion.

- **Time Increment.** The time increment determines how often Snooker computes the positions of the balls.

Suppose that the display factor is set to 3 and the time increment is set to 0.01. This means that:

- The positions of the balls will be computed every 0.01 seconds or, in other words, 100 times each second.
- The screen will be refreshed at every third computation: in this case, 100/3 = 33 times per second.

The accuracy of the simulation depends on the time increment: the smaller the value, the more accurate the simulation. But, if the value is too small, the simulation will be unacceptably slow.

The speed of the balls on the screen, and the jerkiness of their motion, depends on the display factor. If the display factor is 1, the motion will be smooth but slow. If the value is large, say 10, the balls will move ten times faster, but in jerks.

You should try to choose settings of the display factor and time increment that give smooth motion that is fast enough. Snooker provides a "timing test" to help you. Use it as follows:

- Start Snooker or, if it is running, end the current frame.
- Select `Settings` from the menu bar and then `Timing Test`. When the message window appears, click on `OK`.
- Snooker will play a "lag shot" which sends the cue ball up and down the table. It will compute both the simulated time and the actual time required for this shot, and will report the value of the time increment that would make them equal.
- If you click on `OK` in the final message box, Snooker will set the value of the timing increment to the value that it has computed.

Although the value of the timing increment is now correct, you may still want to change the display factor. Here is a rule of thumb that will help you: find the smallest value of the display factor for which the timing test gives a value of 0.01 or less to the timing increment. When you have found the best value for the display factor, run the timing test again to get the appropriate value of the time increment.

If you want to save the timer settings permanently, so you do not have to change them every time you play, edit the file "`settings.dat`", as described in Q&A13.