

# Ludo

---

## Introduction

---

In this section we're going to create a game which makes use of some of the topics we've covered in the last few pages such as record structure within memory blocks and slider components.

### Rules of the Game

The game, Ludo, is a simple board game with a very long history. The board contains four home areas (coloured red, yellow, blue and green), a playing area around which pieces are moved, and a goal area. The aim of the game is for a player to move all of his pieces from the home area to the goal area.

Each player has four pieces which match the colour of their home area. Pieces are moved around the playing area of the board which is marked off in squares. The number of squares covered on a move is determined by a dice throw. A six must be thrown before a piece can move off the home area.

Options available to a player depend on the value thrown:

- 1-5 If the player has one or more pieces in the playing area, one of these pieces can be moved.  
If the player has no pieces in the playing area, then the player cannot move.
- 6 If the player has a piece in the home area it may be moved to the playing area OR  
a piece in the playing area may be moved

AND

the player is entitled to another move every time a six is thrown.

#### *Restrictions*

A piece may not finish on a square containing another piece belonging to the same player. If no other move is possible, the player misses a turn.

A piece may finish on a square containing another player's piece. In this case, the other player's piece must be returned to its home area to start again.

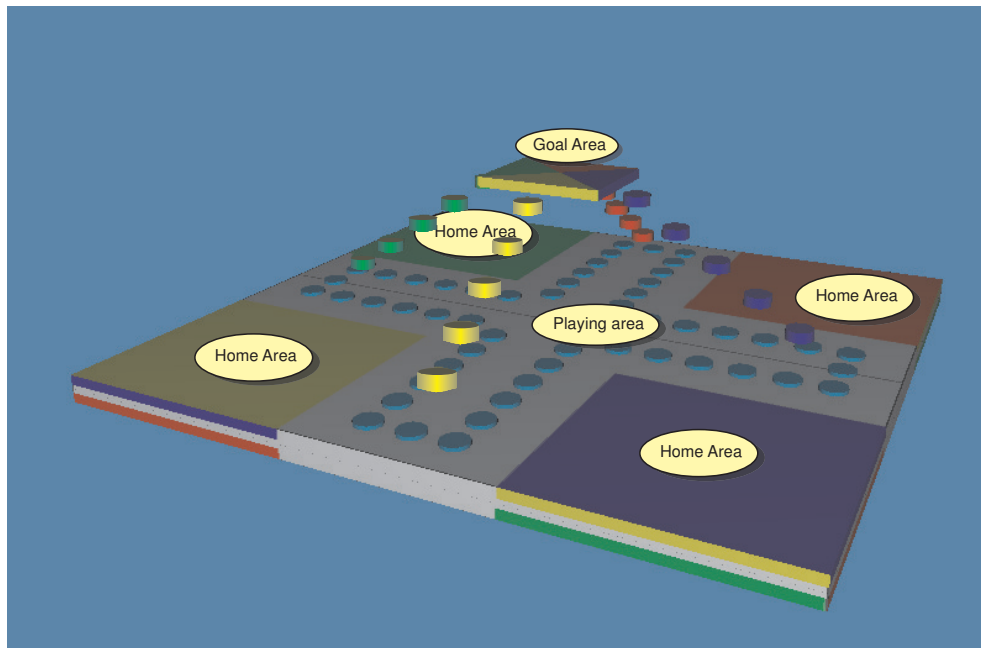
Movement to the goal area can only occur if exactly the required value is thrown on the dice.

### Computerising the Game

Traditional Ludo is played on a flat board, but since we are constructing a computerised version of the game there is no reason for us to restrict ourselves in this way and so the goal area of the board has been raised to give a 3D feel to the game.

The 3D components required by the game are shown in FIG-Sup024.

**FIG-Sup024** The Ludo Game Board



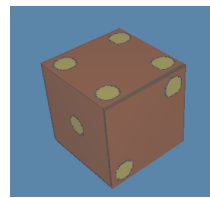
The other 3D components used in the game area are the players' pieces and a dice. These are shown in FIG-Sup025.

**FIG-Sup025**

Other 3D Models Used in the Game



Piece



Dice

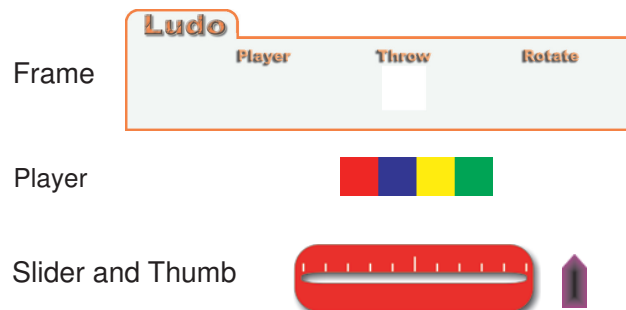
Look out for HANDS ON MILKSHAPE in 2009.

All 3D models were created using Milkshape.

Additionally, the game makes use of several sprites. These are shown in FIG-Sup026.

**FIG-Sup026**

The Sprites Used in the Game

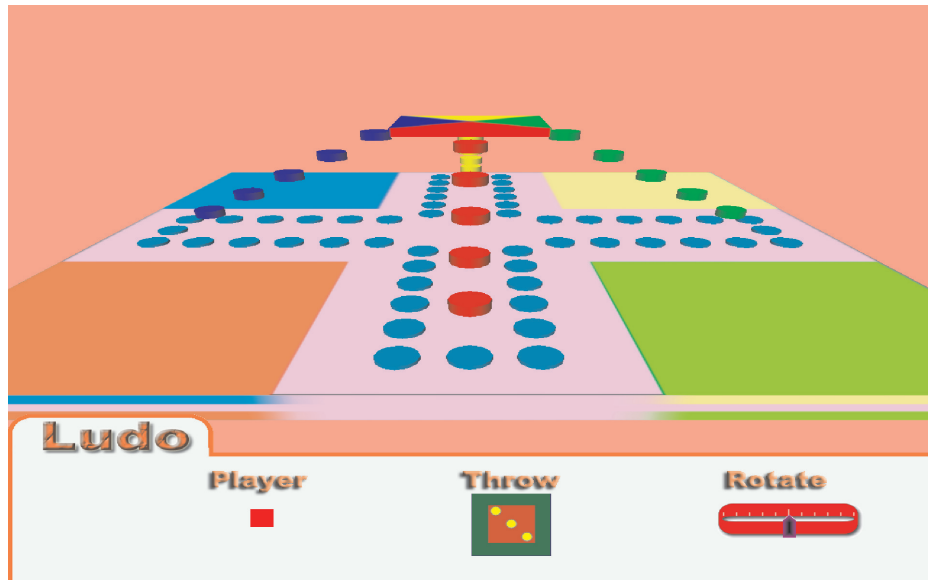


The layout of the main game screen is shown in FIG-Sup027. Notice that the dice is displayed by positioning the output of a second camera at a transparent section of the frame sprite.

**FIG-Sup027**

The Final Game Layout

NOTE: In part 1 the player's pieces do not appear.



## Creating the Game Set Up

Since the code for the game is quite long, we'll develop the project in easy sections. In this first section we'll set up the screen layout and allow the slider to rotate the player's viewpoint. In the next section, we'll create the data structures necessary to implement the game. The third section will implement the game and the fourth section add embellishments.

We'll start the coding by including the code necessary for a slider

```
#INCLUDE "PointType.dba"
#include "SliderType.dba"
```

and the required global variable:

```
GLOBAL memblockID = 0
```

Next, we need to define any constants we will be using. This is normally image, sprite and 3D object IDs:

```
REM *****
REM ***   Named Constants   ***
REM *****
REM 3D Objects ****
#CONSTANT boardObj      1
#CONSTANT diceObj      2
#CONSTANT firstRedPiece 3
#CONSTANT firstBluePiece 7
#CONSTANT firstYellowPiece 11
```

```

#CONSTANT firstGreenPiece      15

REM *** Images ***
#CONSTANT sliderImg           1
#CONSTANT thumbImg           2
#CONSTANT pointerImg         3
#CONSTANT frameImg           4
#CONSTANT playerImg          5

REM *** Sprites ***
#CONSTANT pointerSpr         3
#CONSTANT frameSpr           4
#CONSTANT playerSpr          5

```

Although we'll want other global variables to represent game components, at the moment the only one we require is for the slider:

```
GLOBAL slider AS SliderType
```

Now we need the main logic of the game. At the moment we'll allow only rotation of the view using the slider, so the logic required is:

```

REM *****
REM ***   Main Game Logic   ***
REM *****
SetUpScreen()
SetUpGame()
DO
    HandleSlider()
LOOP
END

```

---

## First Level Routines

---

### *SetUpScreen()*

This routine sets the screen resolution, background colour and camera positions as well as defining areas of the screen to which camera output is to be directed:

```

REM *****
REM ***   Level 1 Routines   ***
REM *****

FUNCTION SetUpScreen()
    REM *** Set screen details
    SET DISPLAY MODE 1280, 1024,32
    AUTOCAM OFF
    COLOR BACKDROP RGB(255,150,155)
    REM *** Set white as the transparent colour ***
    SET IMAGE COLORKEY 255,255,255
    REM *** Set up camera zero ***
    POSITION CAMERA 0,10,-20
    SET CAMERA VIEW 0,0,0,1280,860
    POINT CAMERA 0,0,0
    REM *** Set up camera 2 ***
    MAKE CAMERA 2
    POSITION CAMERA 2, 0,-53.5,0
    SET CAMERA VIEW 2,641,860,749,972
    POINT CAMERA 2,0,-55,0
    SET CAMERA ASPECT 2,1
ENDFUNCTION

```

## *SetUpGame()*

This routine positions all of the components of the game such as sprites and 3D objects. In fact, it off-loads the actual work to subordinate routines.

```
FUNCTION SetUpGame ()
  LoadImagesUsed ()
  LoadModels ()
  LoadControlFrame ()
ENDFUNCTION
```

## *HandleSlider()*

This routine rotates the player's viewpoint in response to dragging the slider control:

```
FUNCTION HandleSlider ()
  SPRITE pointerSpr, MOUSEX(), MOUSEY(), pointerImg
  xoffset = MouseOverThumb(slider,pointerSpr)
  WHILE MouseOverThumb(slider,pointerSpr)
    DragThumb(slider,xoffset)
    SPRITE pointerSpr, MOUSEX(), MOUSEY(), pointerImg
    angle = GetSliderValue(slider)-180
    POSITION CAMERA 20*SIN(angle),10,20*COS(angle)
    POINT CAMERA 0,0,0
  ENDWHILE
ENDFUNCTION
```

---

## Second Level Routines

---

The routines at this level are those called by the *SetUpGame()* function.

### *LoadImagesUsed()*

This function loads in all the images used in the game.

```
REM *****
REM ***   Level 2 Routines   ***
REM *****

FUNCTION LoadImagesUsed ()
  LOAD IMAGE "slider.bmp", sliderImg, 1
  LOAD IMAGE "thumb.bmp", thumbImg, 1
  LOAD IMAGE "white.bmp", pointerImg, 1
  LOAD IMAGE "frame.bmp", frameImg, 1
ENDFUNCTION
```

### *LoadModels()*

This function loads the 3D models used. At the moment the *piece* model is not included since positioning all 16 playing pieces requires sets of 3D coordinates - something which we will add in the next section.

```
FUNCTION LoadModels ()
  LOAD OBJECT "ludoboard.x", boardObj
  LOAD OBJECT "dice.x", diceobj
  SCALE OBJECT diceobj, 20, 20, 20
  POSITION OBJECT diceObj, 0, -55, 0
ENDFUNCTION
```

### *LoadControlFrame()*

This function loads the main control area at the bottom of the screen. This area contains a Ludo-title frame, a four-element player sprite used to display the colour of the current player and finally, the slider control. The dice which appears in the centre of the Ludo frame is actually "seen" through a transparent hole in the sprite.

```
FUNCTION LoadControlFrame ()
  SPRITE frameSpr, 0,723,frameImg
  CREATE ANIMATED SPRITE playerSpr,"player.bmp",4,1,playerImg
  SPRITE playerSpr,335,890,playerImg
  SET SPRITE FRAME playerSpr,1
  SetUpSlider ()
ENDFUNCTION
```

---

## Third Level Routines

---

### *SetUpSlider()*

The only third level routine at the moment is the *SetUpSlider()* routine which initialises the slider component.

```
FUNCTION SetUpSlider ()
  slider = CreateSliderType ()
  SetBackImageID (slider,sliderImg)
  SetThumbImageID (slider,thumbImg)
  SetBackSpriteID (slider,1)
  SetThumbSpriteID (slider,2)
  SetBackCoords (slider,980,880)
  SetThumbCoords (slider,1070,900)
  SetThumbMinX (slider,980)
  SetThumbMaxX (slider,1160)
  SetMinValue (slider,0)
  SetMaxValue (slider,720)
  ShowSlider (slider)
ENDFUNCTION
```

All the models and images required are in the *LudoResources.ZIP* file on our website.

#### **Activity Sup058**

Create a new project (*Ludo.dbpro*) and add all the code given so far.

Make sure you've copied the 3D models, images and *.dba* files used by the program into this folder.

Type in the code given so far and check that the player can rotate around the playing board.