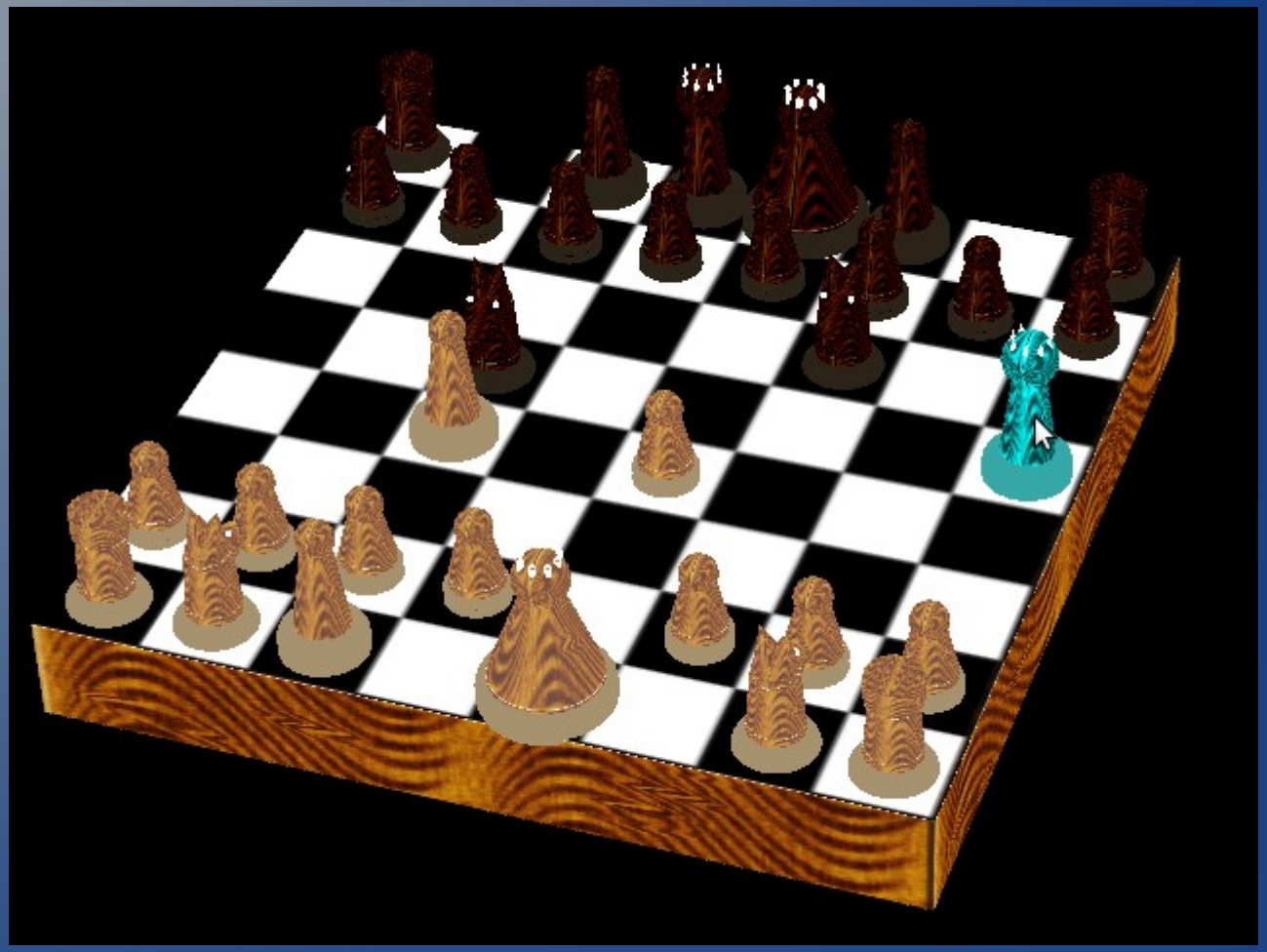
Cross-Platform 3D Chess

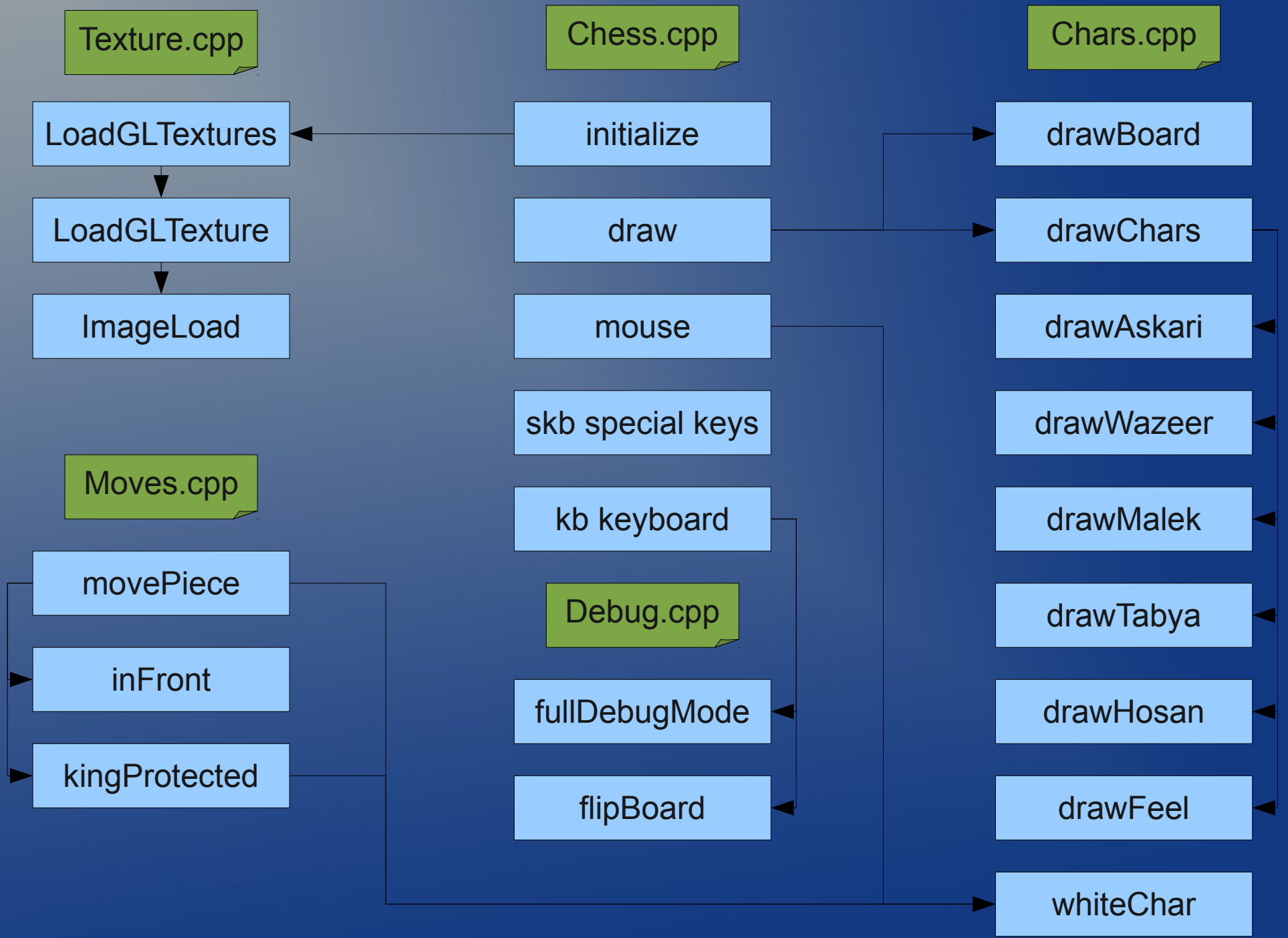


by:
ComputeristGeek

Project Outline

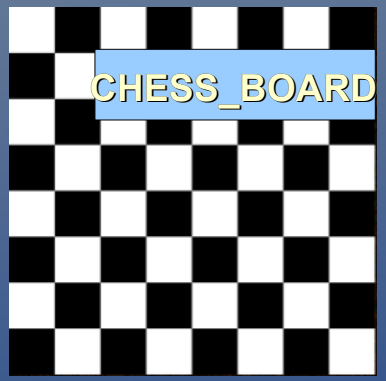
- Using Textures
- Graphics: Chess Pieces & Board
- Object Picking
- Game Rules
- Debugging
- Testing
- Future Enhancements

Functions FlowChart



Using Textures

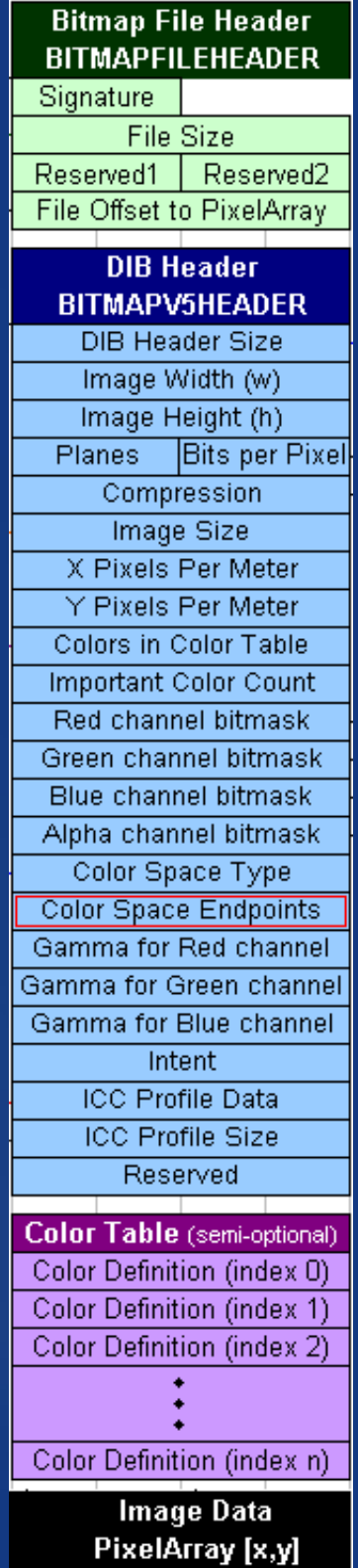
- Loading A Bitmap Image
- Generating & Binding Textures
- Using Textures On Different Surfaces



Loading A Bitmap Image

```
struct Image {
unsigned long sizeX;
unsigned long sizeY;
char *data;
};

int ImageLoad(const char *filename, Image *image)
{
FILE *file;
unsigned long size;                 // size of the image in bytes.
unsigned long i;                    // standard counter.
unsigned short int planes;          // number of planes in image (must be 1)
unsigned short int bpp;             // number of bits per pixel (must be 24)
// Make sure the file exists
file = fopen(filename, "rb")
// Skip to bmp header
fseek(file,18, SEEK_CUR);
// read width
i = fread(&image->sizeX, 4, 1, file)
//read the height
i = fread(&image->sizeY,4,1,file)
//printf("Height of %s: %lu\n", filename, image->sizeY);
// calculate the size (assuming 24 bpp)
size = image->sizeX * image->sizeY * 3;
// read the planes
fread(&planes, 2, 1, file)
// read the bpp
i = fread(&bpp, 2, 1, file)
// seek past the rest of the bitmap header
fseek(file, 24, SEEK_CUR);
// Read the data
image->data = (char *) malloc(size);
i = fread(image->data,size,1,file)
// reverse all of the colours bgr => rgb)
for (i=0;i<size;i+=3)  swap(image->data[i] = image->data[i+2]);
```



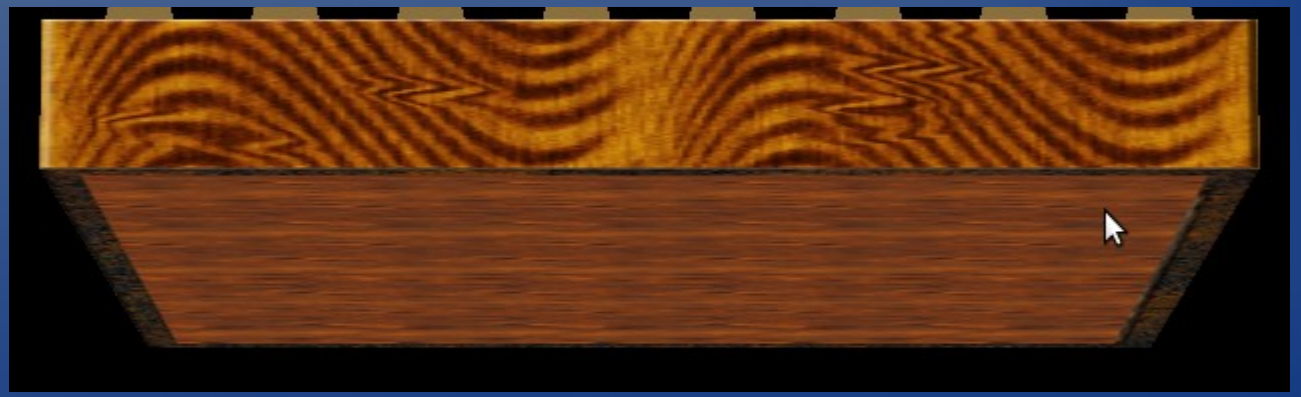
Generating & Binding Textures

// create Texture
- glGenTextures(1, &texture[textNum]);

//linear scaling
- glBindTexture(GL_TEXTURE_2D, texture[textNum]);
// scale linearly when image is of a different size than texture
- glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER, GL_LINEAR);
- glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
- glTexImage2D(GL_TEXTURE_2D, 0, 3, image1->sizeX, image1->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE, image1->data);

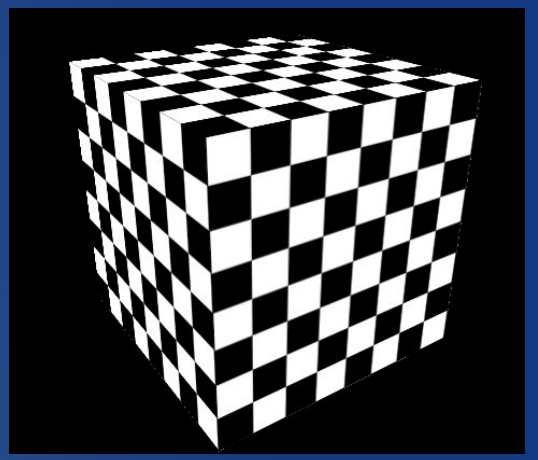
Using Textures On Different Surfaces

- The most important thing to notice is this function:
  - glBindTexture(GL_TEXTURE_2D, texture[textNum]);
- This function binds the texture referred to, to any usage of it



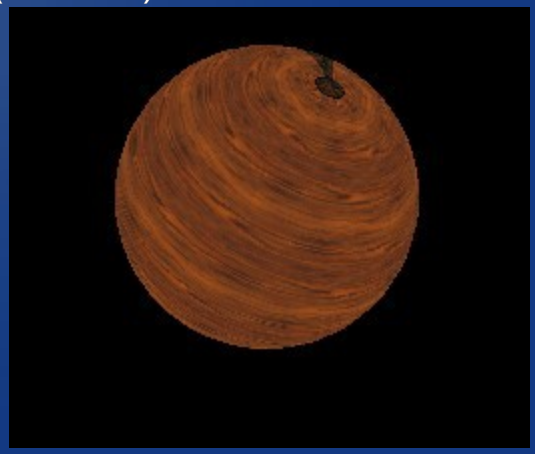
Examples on Using Textures

- If used on a flat 2D polygon/quad:
  - glBindTexture(GL_TEXTURE_2D, texture[SIDE_WOOD]);
  - glBegin(GL_QUADS);
  - glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
  - glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, -1.0f, 1.0f);
  - glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0, 1.0f);
  - glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);
  - glEnd();



- If used on a Quadric Object:
  - glBindTexture(GL_TEXTURE_2D, texture[((blackWhite)?((chosen)?
    WHITE_WOOD_C:WHITE_WOOD):((chosen)?
    BLACK_WOOD_C:BLACK_WOOD))]);
  - GLUquadricObj *AskariSphere=gluNewQuadric();

    //activate Texture
  - gluQuadricTexture(AskariSphere,true);
  - gluSphere(AskariSphere,DIV16-0.02,64,64);

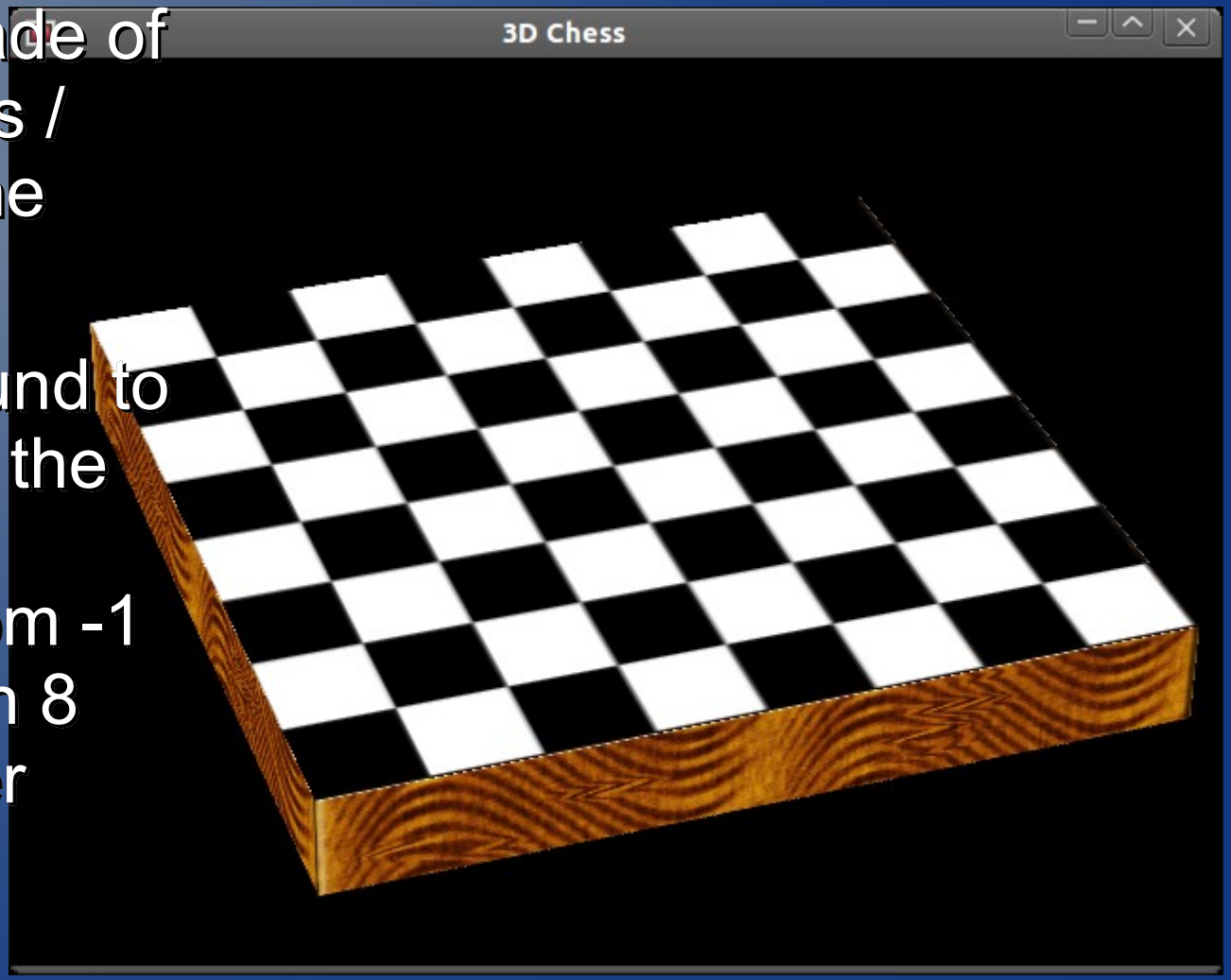


Graphics: Chess Pieces & Board

- Cuboid For Board
- Types Of gluQuadric Objects Available
- Types Of glut 3D Drawings
- Transformation & Inverse Transformation
- Linking Chess Pieces With BitBoard

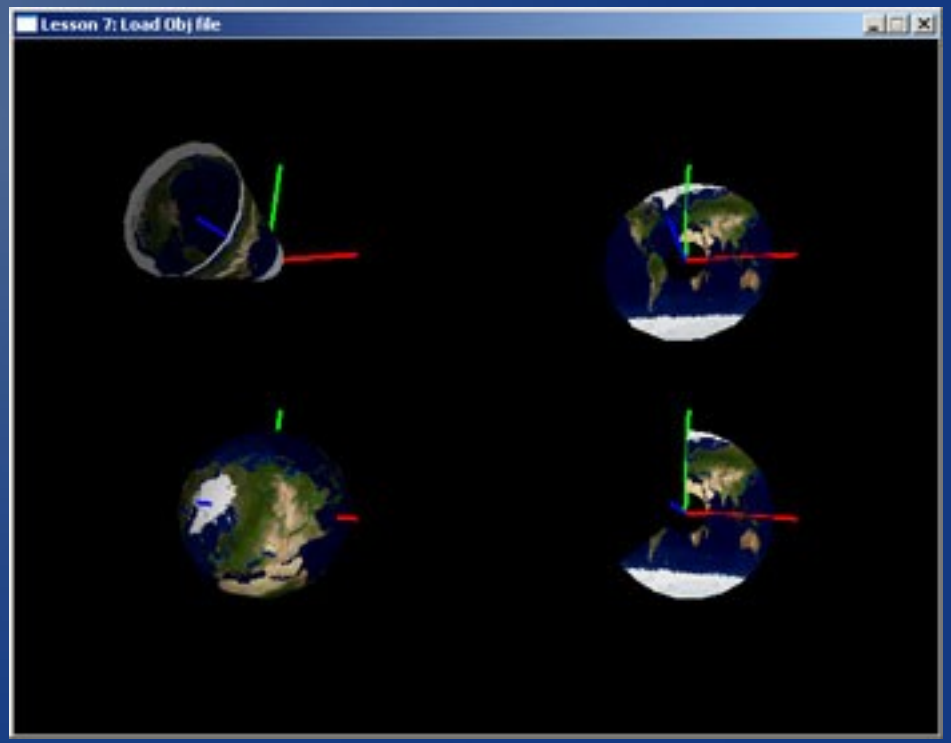
Cuboid For Board

- The board is made of 6 quadrilaterals / polygons for the cuboid shape
- Textures are bound to each side with the top side's dimensions from -1 to +1 divided in 8 parts for proper positioning of pieces



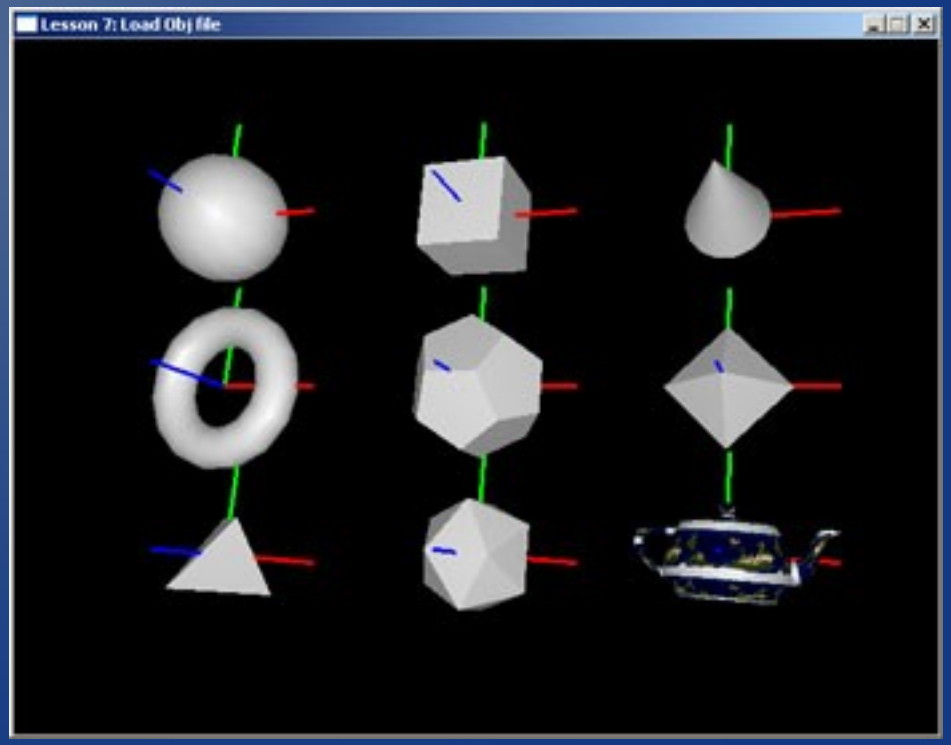
Types Of gluQuadric Objects Available

- gluCylinder //We Can Draw Cones With It Too
- gluDisk
- gluPartialDisk
- gluSphere



Types Of glut 3D Drawings

- glutSolidSphere
- glutSolidCube
- glutSolidCone
- glutSolidTorus
- glutSolidDodecahedron
- glutSolidOctahedron
- glutSolidTetrahedron
- glutSolidIcosahedron
- glutSolidTeapot



Transformation & Inverse Transformation

- glRotatef, glScalef and glTranslatef all work on world coordinates, meaning they will transform all objects which is inconvenient most of the time...
- Transforming before drawing an object then inverse transforming will allow us to transform only the object!
  - e.g. glTranslatef(-1,-1,-1);//then draw object
  - glTranslatef(1,1,1);//this transforms everything back leaving only the object transformed

Linking Chess Pieces With BitBoard

- Two main arrays were made to link pieces:
  - Board[8][8] //stores pieces' names as int
  - charsPosition[32] //stores pieces' positions
- These arrays are the basis of the game, the Board array simulates the chessboard and returns/stores the name of the piece relative to the position
- The charsPosition array facilitates fast searching of any character's position

Linking Chess Pieces With BitBoard

- An important equation

```
Board[i][j]=whiteAskari0
charsPosition[whiteAskari0]=i*8+j
```

- How to get 'i' and 'j' from position when we have 54 for example?
  - int tempi=54/8; //this removes the j as fraction
    //it also returns i (i*8/8=i)
  - int tempj=54-tempi*8; //returns j

Object Picking

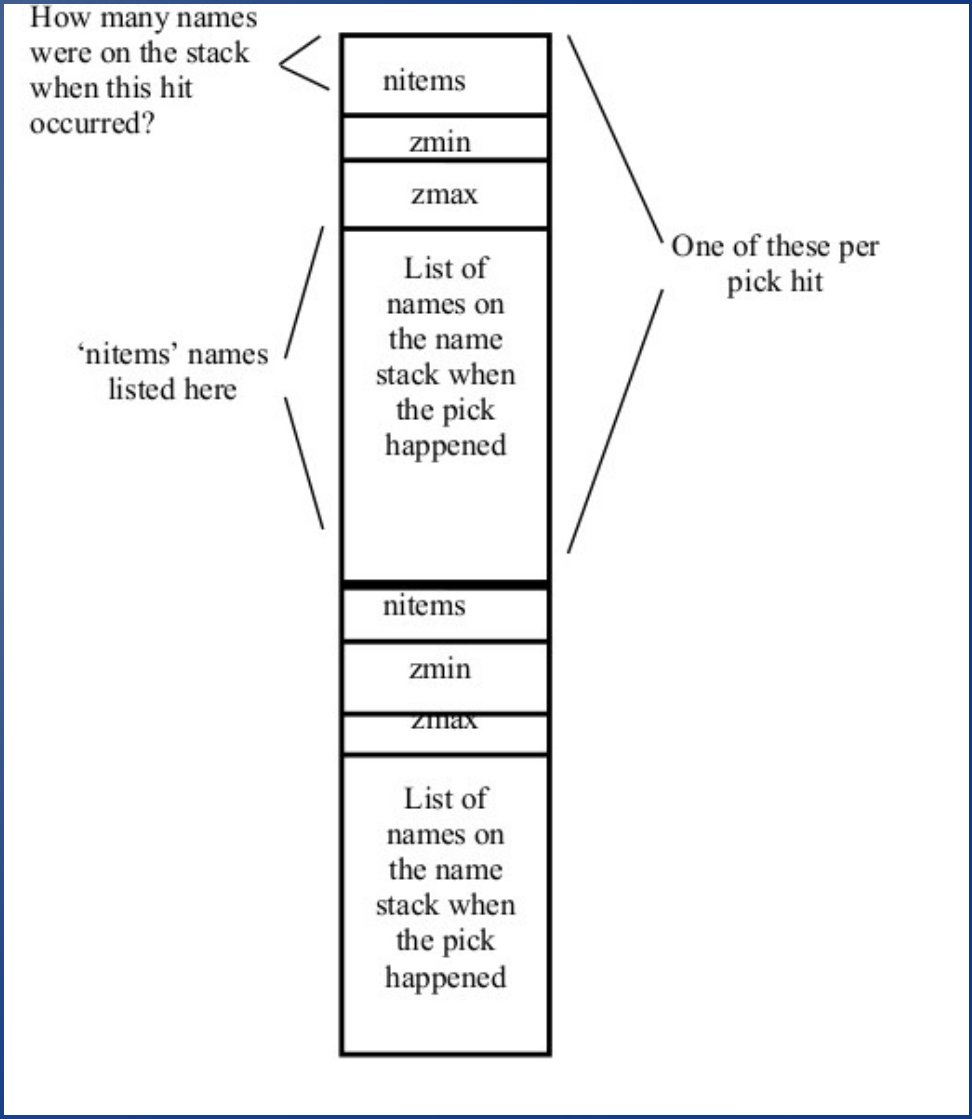
- Select and Render Modes
- Pick Buffer
- Ray Intersection With Simulated Graphics
- Naming Objects

Select and Render Modes

- There are three render modes
  - GL_RENDER
  - GL_SELECT
  - GL_FEEDBACK
- We choose GL_RENDER mode for drawing (default) and GL_SELECT mode for selecting
- Changing the mode is done through glRenderMode(/*mode*/)

Pick Buffer & gluPickMatrix

- A picking buffer is needed to store the information returned from picking, PickBuffer
- Linking to the buffer is done through glSelectBuffer(BUFFERSIZE ,PickBuffer)
- gluPickMatrix(Xmouse,Ymouse,toleranceX,toleranceY,viewport); Defines the region that picking will be done in



Ray Intersection With Simulated Graphics

- We go into select mode and call the draw function
- *(nothing is drawn, but selection is done)*

Ray Intersection With Simulated Graphics

- A ray from the (x,y) of the mouse position is simulated to intersect the objects
  - Minimum and maximum z-value of the last hit object
  - z-value [0,1] is scaled by 2^{32}-1 (0xFFFFFFFF)
- Then the contents of the PickBuffer are picked from

Naming Objects

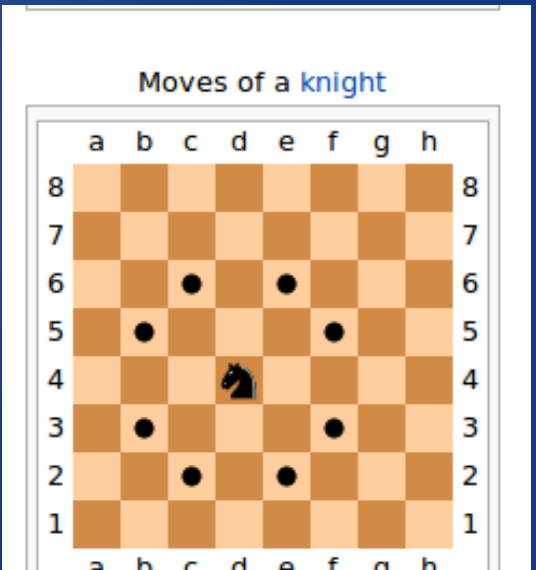
- Prior to picking, every set of graphics drawn after the function glLoadName(int) will be named after that integer
- So when any shape/pixel is selected, the name returned will be that integer
- These names are initialized by glInitNames()

Game Rules

- Each Piece & Its Own Motion
- Objects In-between Position & Destination
- Prohibition of Any Move That Leaves King Exposed
- Detecting Win/Loss/Draw

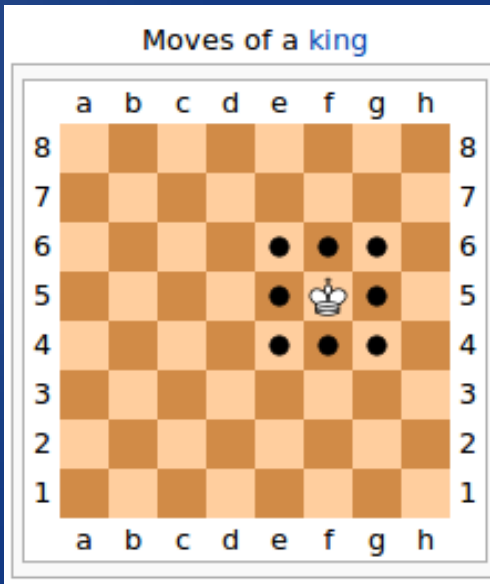
Each Piece & Its Own Motion

- Hosan/Knight
- The knight moves 2 steps then 1 or 1 step then 2, either vertically or horizontally
- The condition for its movement is
  - (abs(ito-ifrom)==2 && abs(jto-jfrom)==1)
  - or
  - (abs(ito-ifrom)==1 && abs(jto-jfrom)==2)



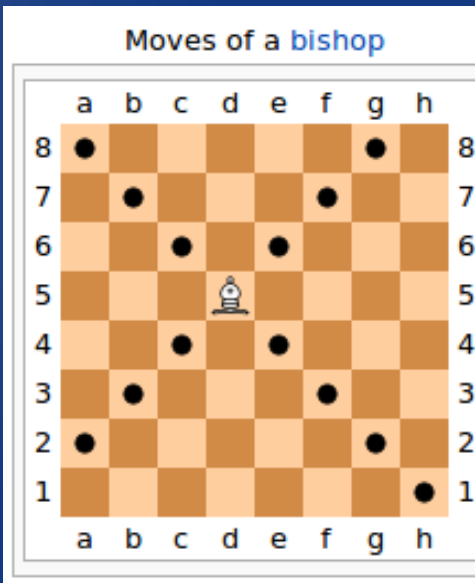
Each Piece & Its Own Motion

- Malek/King
- The king can only move one step in any direction
  - abs(ito-ifrom)<=1)
  - abs(jto-jfrom)<=1)



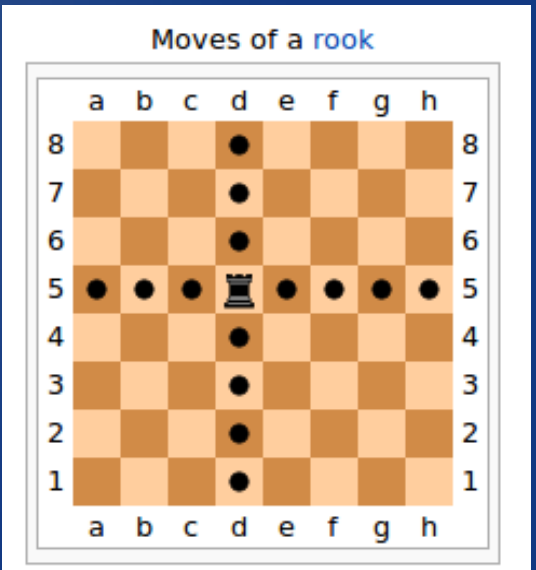
Each Piece & Its Own Motion

- Feel/Bishop
- The bishop can only move in a diagonal i.e. any move in x=any move in y
  - abs(ito-ifrom)==abs(jto-jfrom)
- For it to move also:
  - Nothing should be in its way



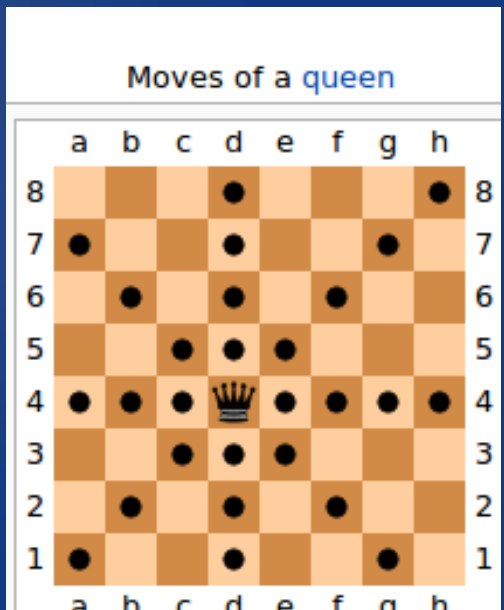
Each Piece & Its Own Motion

- Tabya/Rook
- The rook moves in either a vertical motion or a horizontal motion
  - ito==ifrom
  - or
  - jto==jfrom
- For it to move also:
  - Nothing should be in its way



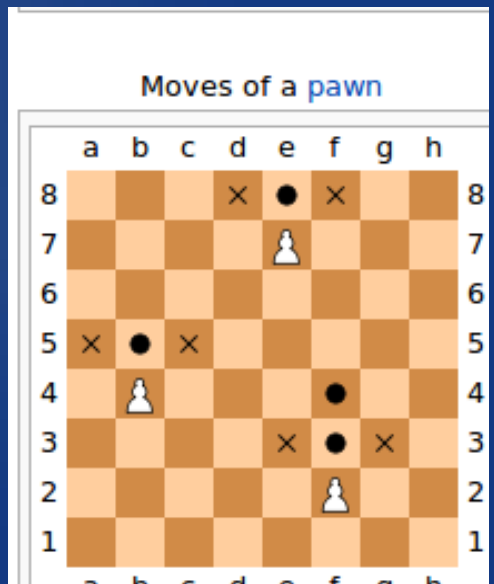
Each Piece & Its Own Motion

- Wazeer/Queen
- The queen moves like the rook or the bishop
  - ito==ifrom
  - or
  - jto==jfrom
  - or
  - abs(ito-ifrom)==abs(jto-jfrom)
- For it to move also:
  - Nothing should be in its way



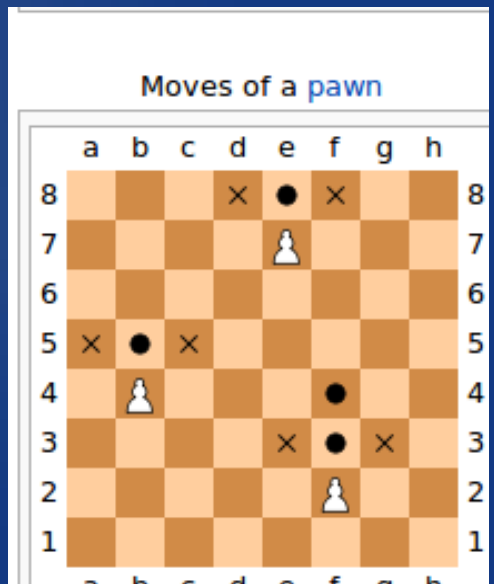
Each Piece & Its Own Motion

- Askari/Pawn
- The pawn can move two steps forward if it's in its starting position, otherwise it moves only one step forward (it can not kill by this move)
  - jto==jfrom
  - && (
    - onlyOneMoveForward
    - || (ifrom==begin && twoMovesForward)
  -)
  - && !inFront(ifrom, jfrom, ito+((begin==1)?1:-1), jto)



Each Piece & Its Own Motion

- Askari/Pawn
- The pawn can kill sideways if the object present is the enemy and it's one step ahead in x and one step ahead in y
  - || (
    - Board[ito][jto]!=-1
    - && whiteChar(Board[ito][jto])^turn
    - && abs(jto-jfrom)==1
    - && onlyOneMoveForward
  -)

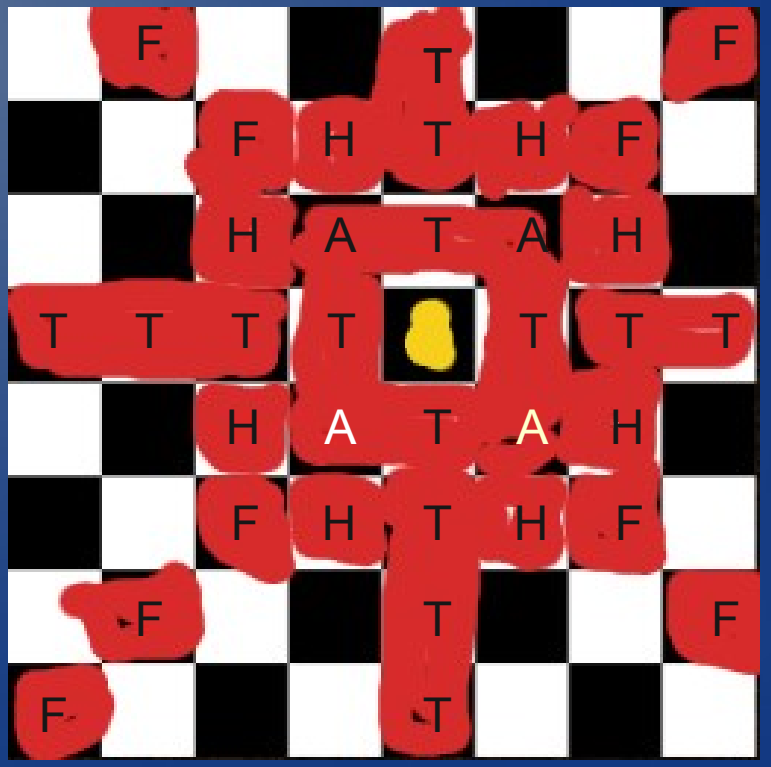


Objects In-between Position & Destination

- if(j1==j2)
  - for(int i=Min(i1,i2)+1;i<Max(i1,i2);i++)
    - if(Board[i][j1]!=-1) return true;
- else if(i1==i2)
  - for(int j=Min(i1,i2);j<Max(i1,i2);j++)
    - if(Board[i1][j]!=-1) return true;
- else if(abs(i1-i2)==abs(j1-j2))
  - int inc=(i1<i2)?j1:j2;
  - inc=(inc==Min(j1,j2))?1:-1;
  - for(int i=Min(i1,i2)+1,j=((i1<i2)?j1:j2)+inc;i<Max(i1,i2);i++)
    - if(Board[i][j]!=-1) return true;
    - j+=inc

Prohibition of Any Move That Leaves King Exposed

- Very important: Malek/King can NEVER DIE
- Therefore, a set of rules should be placed to simulate the move and check if it's going to expose the king

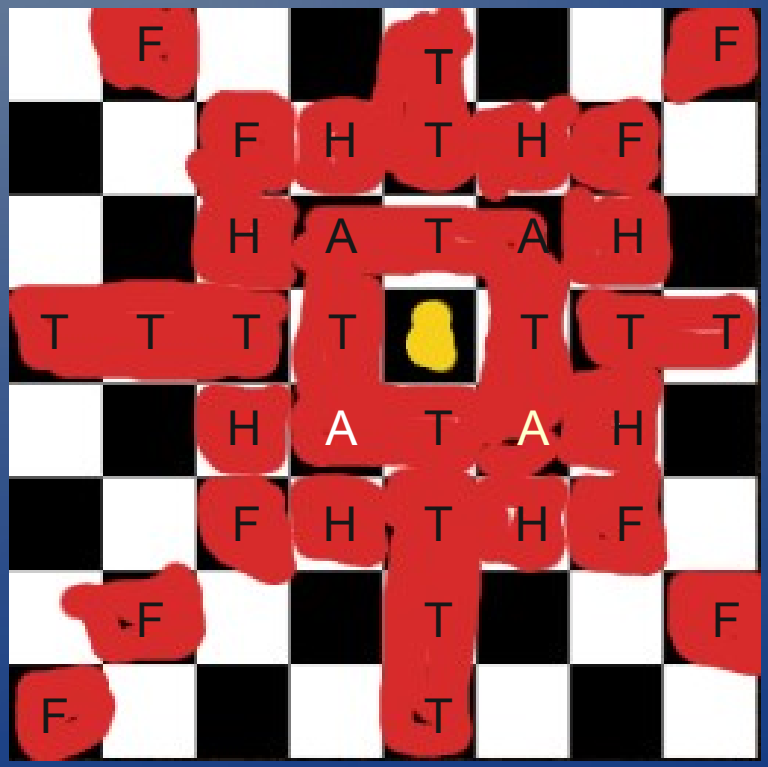


Prohibition of Any Move That Leaves King Exposed

- TWO choices are available
  - Check in all endangering directions until a piece is found
    - Might consume more CPU but it is much easier in programming
  - Check only the places where the respectable objects can attack
    - Consumes less CPU but it is a programming HEADACHE!

Prohibition of Any Move That Leaves King Exposed

- I chose the harder-to-implement but more efficient solution...



Prohibition of Any Move That Leaves King Exposed

- Checking for the Hosan/Knight and anything next to the Malek/King is top priority
- Because if these exist in endangering places, they can only be stopped by
  - conquering the pieces
  - by moving the king away

Prohibition of Any Move That Leaves King Exposed

- Checking for a Hosan/Knight threat
- I made an algorithm to follow the sequence
- (-2,-1),(-1,-2),(+1,-2) ,(-2,+1),(+2,+1),(+1,+2),(-1,+2), (+2,-1)
- Which can easily be implemented by
  - for(int i=0,inci=-2,incj=-1;i<8;i++)
    - int temp1=Mali+inci,temp2=Malj+incj;
    - if(temp1<=7 && temp2<=7 && temp1+temp2==abs(temp1)+abs(temp2) && simulateBoard[temp1][temp2]==hosan)
      - return false; //king not protected
    - if(i%2==0)swap(&inci,&incj); else inci=-inci;

Attack

Prohibition of Any Move That Leaves King Exposed

- Checking for any threat one step beside the king
  - Check for the other Malek (illegal move)
  - Check for the Askari

Attack

Prohibition of Any Move That Leaves King Exposed

- Check for the other Malek/King
- if(
- abs(otherMali-Mali)<=1
- &&
- abs(otherMalj-Malj)<=1
-)
  - return false; //king not protected

Attack

Prohibition of Any Move That Leaves King Exposed

- int begin=(white?1:6);
- if(flipped) begin=(white?6:1);
- //check for any part around the malek (+1,+1) (-1,-1) (+1,-1) (-1,+1) that can have a askari
- if((begin==6 && Mali+1<8) || (begin==1 && Mali-1>-1))
  - if(Malj+1<8)
    - test=simulateBoard[Mali+(begin==6?1:-1)][Malj+1];
    - if(test<=Askari7 && test>=Askari0)) return false;//king not protected
  - if(Malj-1>-1)
    - test=simulateBoard[Mali+(begin==6?1:-1)][Malj-1];
    - if(test<=Askari7 && test>=Askari0)) return false;//king not protected

Attack

Prohibition of Any Move That Leaves King Exposed

- The last part is the simplest
- We check vertically, horizontally and diagonal-wise
  - if we first come across a piece of our own, we stop checking that side
  - if we come across a piece that attacks in that direction
    - e.g. Tabya/Rook attacks in horizontal and vertical directions, Feel/Bishop attacks diagonal-wise, Wazeer/Queen attacks in both ways like Tabya/Rook and Feel/Bishop

  Malek/King is in danger!

Attack

Detecting Win/Loss/Draw

- Detecting Win is an alternative to detect Loss
  - To detect loss is to prove that there is no move possible and the Malek/King is endangered
  - Brute force method:
    - Try all possible moves for each remaining piece
    - Complexity averagely = 64 (positions) * 32 (pieces) (constant)
  - Check source of Danger and detect any possible move to counter it
    - According to the piece and closeness

Detecting Win/Loss/Draw

- Detecting Draw
  - To draw, the only possible move is for the Malek/King but that move would kill it
    - Like Win/Loss but not endangered
  - OR
  - To draw, the only remaining pieces would be the black Malek/King and the white one
    - Very easy, checking the characters array

Detecting Win/Loss/Draw

Debugging

- Current Problems
- Methods Used to Debug

Current Problems

- Till now, no problems...
- Earlier, there were many problems with the logic, most notable were:
  - Askari/Pawn needs to check an extra block infront of it because it doesn't conquer that way
  - Order of what is to be done first matters a lot
  - All pieces and their moves need to halt (except those that can protect the Malek/King)

Methods Used to Debug

- A couple of methods were used to debug
  - cout<<Error<<” “<<value<<endl;
  - Game cheats were made to flip the board (function flipBoard gets called)

    These game cheats were made using keyboard commands being checked as strings
    - e.g. 'c3cfreee' allows free manipulation of pieces
    - e.g. 'c3cflipp' flips the board

Testing

- Normal Testing
- Abnormal Testing
- *There is no extreme testing...*

Normal Testing

- Many game characters were tested to see if they moved properly and conquered properly
- However, due to the huge range of moves and possible gameplays, it is impossible to test the game fully.

Abnormal Testing

- Illegal moves were tried to see if the game would accept it
  - e.g. Illegal piece moves, or moves that exposed the Malek/King etc.
- Again, due to the huge possibilities of gameplays and scenarios, it wasn't possible to test all situations

Future Enhancements

- Finish the win/loss/draw states
- Enable storing of chess files and reading them
  - Generic chess file
- Enable variations of chess games and arrangements
- Usage of menus and a user-friendly interface
- Artificial Intelligence for Human vs. Computer
- Enabling Network Gaming

DEMO

Thank You For Your Attention