# **Design Document for a Chess Game**

1. **Brief description of Our project**

Our project implements a classic version of Chess with OpenGL . The Chess game follows the basic rules of chess, and all the chess pieces only move according to valid moves for that piece. It is played on an 8x8 checkerboard, with a dark square in each player's lower-left corner. The game was developed using OpenGL, Pygame and Blender.

1. **What is a chess game?**

Chess is a game for 2 players each of whom moves 16 pieces according to ﬁxed rules across a checkerboard and tries to checkmate the opponent's king. Our project implements the chess game with a graphical user interface. The chess game

follows the basic rules of chess and all the chess pieces only move according to valid moves for that piece.

Chess is a game where the battle of minds takes place between two people. It is a game of strategy where two people play with each other’s minds. It’s a board game that requires patience, concentration, intuition, perseverance, etc. Chess is a mind game that involves a lot of thinking and time. It requires prediction and problem-solving skills. We are living in a world that is connected despite being in different locations. So we have the ability to play and challenge other people with games.

1. **What Are The Chess Pieces?**

The chess pieces are what you move on a chessboard when playing a game of chess. There are six different types of chess pieces. Each side starts with 16 pieces: eight pawns, two bishops, two knights, two rooks, one queen, and one king.

**3.1 The Pawn**

When a game begins, each side starts with eight pawns. White's pawns are located in the second rank, while Black's pawns are located in the seventh rank. The pawn is the least powerful piece and is worth one point. If it is a pawn's ﬁrst move, it can move forward one or two squares. If a pawn has already moved, then it can move forward just one square at a time. It attacks (or captures) each square 7

diagonally to the left or right. In the following diagram, the pawn has just moved from the e2-square to the e4-square and attacks the squares d5 and f5.

**3.2. The Bishop**

Each side starts with two bishops, one on a light square and one on a dark square. When a game begins, White's bishops are located on c1 and f1, while Black's bishops are located on c8 and f8. The bishop is considered a minor piece

(like a knight) and is worth three points. A bishop can move diagonally as many squares as it likes, as long as it is not blocked by its own pieces or an occupied square. An easy way to remember how a bishop can move is that it moves like an

"X" shape. It can capture an enemy piece by moving to the occupied square where the piece is located.

**3.3. The Knight**

Each side starts with two knights—a king's knight and a queen's knight. When a game starts, White's knights are located on b1 and g1, while Black's knights are located on b8 and g8. The knight is considered a minor piece (like a bishop) and is worth three points. The knight is the only piece in chess that can jump over another piece! It moves one square left or right horizontally and then two squares up or down vertically, OR it moves two squares left or right horizontally and

then one square up or down vertically—in other words, the knight moves in an "L-shape." The knight can capture only what it lands on, not what it jumps over!

**3.4. The Rook**

Each side starts with two rooks, one on the queenside and one on the kingside. All four rooks are located in the corners of the board. White's rooks start the game on a1 and h1, while Black's rooks are located on a8 and h8. The rook is

considered a major piece (like the queen) and is worth ﬁve points. It can move as many squares as it likes left or right horizontally, or as many squares as it likes up or down vertically (as long as it isn't blocked by other pieces).

**3.4. The Queen**

The queen is the most powerful chess piece! When a game begins, each side starts with one queen. The white queen is located on d1, while the black queen is located on d8. The queen is considered a major piece (like a rook) and is worth nine points. It can move as many squares as it likes left or right horizontally, or as many squares as it likes up or down vertically (like a rook). The queen can also move as

many squares as it likes diagonally (like a bishop).

**3.5. The King**

The king is the most important chess piece. The goal of a game of chess is to checkmate the king. When a game starts, each side has one king. White's king is located on e1, while Black's king starts on e8. The king is not a very powerful piece, as it can only move (or capture) one square in any direction. When a king is attacked, it is called a "check”.

## **The Problem**

Our goal is to implement a chess program. We will not implement an AI for the program. Rather, this game is meant for two human opponents to play against each other. A game of chess involves chess pieces, and a chess board.

### **4.1. The Board**

The chess board is an 8 by 8 grid. The initial configuration of the pieces is as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| R | N | B | Q | K | B | N | R |
| P | P | P | P | P | P | P | P |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| P | P | P | P | P | P | P | P |
| R | N | B | Q | K | B | N | R |

Where R=rook, N=knight, B=bishop, Q=queen, K=king, P=pawn

### **4.2. The Pieces**

1. Pawns
   1. Normally move forward one space.
   2. Move diagonal one space to kill.
   3. May move two spaces forward on the first move.
      1. But may be killed En Passant after this move by attacking the space behind the pawn.
   4. May be exchanged for any piece except a King by reaching the opposite side of the board.
2. Bishops
   1. Move diagonally any distance.
   2. Kill by landing on a space occupied by an opponent's piece.
3. Knights
   1. Move in "L" shape (one space one direction, two in a perpendicular direction).
   2. May go "over" other pieces.
   3. Kill by landing on a space occupied by an opponent's piece.
4. Rooks
   1. Move horizontally or vertically any distance.
   2. Kill by landing on a space occupied by an opponent's piece.
   3. May "castle" with the king under certain circumstances .
5. Queens
   1. Move any distance horizontally, vertically, or diagonally.
   2. Kill by landing on a space occupied by an opponent's piece.
6. Kings
   1. Move one space horizontally, vertically, or diagonally.
   2. Kill by landing on a space occupied by an opponent's piece.
   3. May "castle" with a rook under certain circumstances (see below).
   4. The game is over:
      1. when the opponent's king cannot escape being taken (i.e. Checkmate).
      2. or one player can make no move without putting his/her king in danger (i.e. Stalemate)

**4.3. Implementation**

The followings are some of the classes we used to implement the chess game.

|  |  |  |
| --- | --- | --- |
| # This class is used to bind texture to a particular object  class Material:  def \_\_init\_\_(self, filepath):  self.texture = glGenTextures(1)  glBindTexture(GL\_TEXTURE\_2D, self.texture)  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT)  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT)  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST)  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR)  image = pg.image.load(filepath).convert()  image\_width, image\_height = image.get\_rect().size  img\_data = pg.image.tostring(image, 'RGBA')  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, image\_width, image\_height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, img\_data)  glGenerateMipmap(GL\_TEXTURE\_2D)  def use(self):  glActiveTexture(GL\_TEXTURE0)  glBindTexture(GL\_TEXTURE\_2D, self.texture)  def destroy(self):  glDeleteTextures(1, (self.texture,))  # This class is used to place the camera to the object to show it’s position while the object is moving  class Camera:  def \_\_init\_\_(self):  self.camera\_pos = Vector3([0.0, 4.0, 3.0])  self.camera\_front = Vector3([0.0, 0.0, -1.0])  self.camera\_up = Vector3([0.0, 1.0, 0.0])  self.camera\_right = Vector3([1.0, 0.0, 0.0])  self.mouse\_sensitivity = 0.25  self.jaw = -90  self.pitch = 0  def get\_view\_matrix(self):  return matrix44.create\_look\_at(self.camera\_pos, self.camera\_pos + self.camera\_front, self.camera\_up)  def process\_mouse\_movement(self, xoffset, yoffset, constrain\_pitch=True):  xoffset \*= self.mouse\_sensitivity  yoffset \*= self.mouse\_sensitivity  self.jaw += xoffset  self.pitch += yoffset  if constrain\_pitch:  if self.pitch > 45:  self.pitch = 45  if self.pitch < -45:  self.pitch = -45  self.update\_camera\_vectors()  def update\_camera\_vectors(self):  front = Vector3([0.0, 0.0, 0.0])  front.x = cos(radians(self.jaw)) \* cos(radians(self.pitch))  front.y = sin(radians(self.pitch))  front.z = sin(radians(self.jaw)) \* cos(radians(self.pitch))  self.camera\_front = vector.normalise(front)  self.camera\_right = vector.normalise(vector3.cross(self.camera\_front, Vector3([0.0, 1.0, 0.0])))  self.camera\_up = vector.normalise(vector3.cross(self.camera\_right, self.camera\_front))  def process\_keyboard(self, direction, velocity):  if direction == "FORWARD":  self.camera\_pos += self.camera\_front \* velocity  if direction == "BACKWARD":  self.camera\_pos -= self.camera\_front \* velocity  if direction == "LEFT":  self.camera\_pos -= self.camera\_right \* velocity  if direction == "RIGHT":  self.camera\_pos += self.camera\_right \* velocity  class App:  “””  This Class is the main class that runs the game. It setups the chess board , the pieces and drives all the other classes.  ”””  def \_\_init\_\_(self):  pg.init()  pg.display.gl\_set\_attribute(pg.GL\_CONTEXT\_MAJOR\_VERSION, 3)  pg.display.gl\_set\_attribute(pg.GL\_CONTEXT\_MINOR\_VERSION, 3)  pg.display.gl\_set\_attribute(pg.GL\_CONTEXT\_PROFILE\_MASK,  pg.GL\_CONTEXT\_PROFILE\_CORE)  pg.display.set\_mode((WIDTH, HEIGHT), pg.OPENGL | pg.DOUBLEBUF | pg.RESIZABLE)  pg.mouse.set\_visible(False)  pg.event.set\_grab(True)  self.clock = pg.time.Clock()  glClearColor(0.1, 0.2, 0.2, 1)  self.shader = self.createShader("vertex.txt", "fragment.txt")  .....  self.modelMatrixLocation = glGetUniformLocation(self.shader,"model")  self.mainLoop() | # This class accepts the path for the obj file and loads it  class Mesh:  def \_\_init\_\_(self, filename):  # x, y, z, s, t, nx, ny, nz  self.vertices = self.loadMesh(filename)  self.vertex\_count = len(self.vertices) // 8  self.vertices = np.array(self.vertices, dtype=np.float32)  self.vao = glGenVertexArrays(1)  glBindVertexArray(self.vao)  self.vbo = glGenBuffers(1)  glBindBuffer(GL\_ARRAY\_BUFFER, self.vbo)  glBufferData(GL\_ARRAY\_BUFFER, self.vertices.nbytes, self.vertices, GL\_STATIC\_DRAW)  # position  glEnableVertexAttribArray(0)  glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 32, ctypes.c\_void\_p(0))  # texture  glEnableVertexAttribArray(1)  glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 32, ctypes.c\_void\_p(12))  def loadMesh(self, filename):  # raw, unassembled data  v = []  vt = []  vn = []  # final, assembled and packed result  vertices = []  # open the obj file and read the data  with open(filename, 'r') as f:  line = f.readline()  while line:  firstSpace = line.find(" ")  flag = line[0:firstSpace]  if flag == "v":  # vertex  line = line.replace("v ", "")  line = line.split(" ")  l = [float(x) for x in line]  v.append(l)  elif flag == "vt":  # texture coordinate  line = line.replace("vt ", "")  line = line.split(" ")  l = [float(x) for x in line]  vt.append(l)  elif flag == "vn":  # normal  line = line.replace("vn ", "")  line = line.split(" ")  l = [float(x) for x in line]  vn.append(l)  elif flag == "f":  # face, three or more vertices in v/vt/vn form  line = line.replace("f ", "")  line = line.replace("\n", "")  # get the individual vertices for each line  line = line.split(" ")  faceVertices = []  faceTextures = []  faceNormals = []  for vertex in line:  # break out into [v,vt,vn],  # correct for 0 based indexing.  l = vertex.split("/")  position = int(l[0]) - 1  faceVertices.append(v[position])  texture = int(l[1]) - 1  faceTextures.append(vt[texture])  normal = int(l[2]) - 1  faceNormals.append(vn[normal])  # obj file uses triangle fan format for each face individually.  # unpack each face  triangles\_in\_face = len(line) - 2  vertex\_order = []  """  eg. 0,1,2,3 unpacks to vertices: [0,1,2,0,2,3]  """  for i in range(triangles\_in\_face):  vertex\_order.append(0)  vertex\_order.append(i + 1)  vertex\_order.append(i + 2)  for i in vertex\_order:  for x in faceVertices[i]:  vertices.append(x)  for x in faceTextures[i]:  vertices.append(x)  for x in faceNormals[i]:  vertices.append(x)  line = f.readline()  return vertices  def destroy(self):  glDeleteVertexArrays(1, (self.vao,))  glDeleteBuffers(1, (self.vbo,)) |  |

**4.4. Referances:**

1. <https://stackabuse.com/advanced-opengl-in-python-with-pygame-and-pyopengl/>
2. <https://stackabuse.com/brief-introduction-to-opengl-in-python-with-pyopengl/>
3. <https://pypi.org/project/PyOpenGL/>
4. <https://learnopengl.com/>
5. <https://www.youtube.com/watch?v=LqPPvPKUfV4&list=PL1P11yPQAo7opIg8r-4BMfh1Z_dCOfI0y>
6. Introduction to Computer Graphic , December 2021 David J. Eck Hobart and William Smith Colleges
7. INTERACTIVE COMPUTER GRAPHICS ,A TOP-DOWN APPROACH WITH SHADER-BASED OPENGL, B EDWARD ANGEL and DAVE SHREINER