By: Modester Mwangi
Bello Mousa

Project Proposal

The objective of the project

This project will work on a simple game called the Chess game, which includes two players gaming against each other. The reality of this chess is a pastime because playing chess results in better brain function, improved memory, and cognitive abilities, strategic thinking, and attention improvement. Scientists also claim that playing chess can improve mental age by up to 14 years.

The other reason for us to work on chess games as our main project is to help as experience, practice, and improve our programming skills under inheritance.
We will also show a unit test to determine whether our creating functions work as required and know our functions' limitations.

What is a chess game?
This is a board game for two players. It is played on a square board, made of 64 smaller squares, with eight squares on each side. Each player starts with 16 pieces, 8 pawns, 2 knights, 2 bishops, 2 rooks, a king, and a queen. The goal of the game is for each player to try and checkmate the king of the opponent. Checkmates is a threat('check') to the opposing king, which no move can stop. It ends the game.

Making of the game:
Here we decided to create classes to represent and show how inheritance occurs in this game by creating our first parent_class(Piece) then following the children classes(king, queen, pawn, knight, bishop, rook)

Classes

To make our game algorithm easier and precise, we will use different classes to hold the variables and methods.

First-class: Piece_class

This is the class parent, whereby the class's children will be inheriting some of the traits from it. This is the basic building of the system's block, as the class parent some of the class children to include will be (Queen, Pawn, Bishop, Knight, Rook).

Attributes: colour (str): The colour of the piece, either 'white' or 'black.'
can_jump (bool): Whether or not the piece can "jump" over other pieces on a chessboard.

Can_make_a_take_over (bool): This checks whether the piece can “eat”(make a takeover) an enemy piece.

Arguments:
colour (str): The colour with which to create the piece.
can_jump (bool): The value with which the attribute can_jump must be initialized.

Test Case number	Test Case description	Test data	Expected results	Actual results	Pass/Fail
1.	Check if the program is running when the test is done	By printing out a statement such as **“----Test cases running___”**	This statement should be printed out	“----Test cases running___”	Pass
2.	Check if the applet is executed	Running the programme	The applet should pop up and display in a window	Applet appeared In a window	pass
3.	Check if all the pieces of both sides are available or not.	Running the programme	All pieces are available on both sides	Applet contained all pieces on the board	pass
4.	Check whether all pieces are properly organised or not	Running the programme	All pieces are fairly organised on both sides	All pieces available and properly arranged on the chess board.	pass

5.	Check move piece	genericpiece OnBoard.can_move_to(1,1) assertEqual(sampiece, generic piece onboard	The piece should move from one grid of the board to another	Piece move from current position to +1 grid next position	pass

Second class: Queen_class
This is a child class of the parent class Piece. This is the most powerful piece in the game.

Attributes: can_move_to (bool): Whether or not the piece can "move" over other chessboard pieces as attacking the opponent.
A move for a queen is valid if she moves in a row or column.
Colour (str): The Queen piece's colour, either 'white' or 'black.'
Here we use methods directly from the class piece.

Test Case number	Test Case description	Test data	Expected results	Actual results	Pass/Fail
1.	Check if the programming is running when the test is done	By printing out a statement such as **"----Test cases running___"**	This statement should be printed out	"----Test cases running___"	pass
2.	Text Queen movements	straightMovementCheck(testQueen)	Queen moves from a position to another in straight line	Dialog box appears showing Valid move of the Queen in straight line	pass
3.	Test whether queen can	assertTrue(testPiece.canMo	Queen should move	Queen makes valid move	pass

	move diagonally	veTo(a2, b2)	diagonally(a2,b2)	given (a2,b2)	
4.	Test whether each piece have colour "black or "white"				pass
5.	Test if a queen can kill	assertTrue(testPiece,can_kill)	The queen piece should be able to kill the opponent	Queen kill opponent from current position	pass

Third class: King_class
This is a child, and it is the most important piece in the game. It is also the weakest piece in the game until it ends. The king represents the leader in the game.

Attributes: can_move (bool): Whether or not the piece can "move" over other chessboard pieces as attacking the opponent. A king can move one square in a line, row, or column
colour (str): The King piece's colour, either 'white' or 'black.'

Test Case number	Test Case description	Test data	Expected results	Actual results	Pass/Fail
1.	Check if the programming is running when the test is done	By printing out a statement such as **"----Test cases running___"**	This statement should be printed out	"----Test cases running___"	pass
2.	Test-King movements	assertTrue(testKing.can_move_to(f2,h2)	Queen moves from a position to another	Dialog box appears showing Valid move of the King	pass
3.	Check for checkmate	assertraises("King checked')	King should be "eaten" that is checked by	King checked by opponent	pass

			the opponent		
4.	Test if a King can kill	assertTrue(testPiece,can_kill)	The King piece should be able to kill the opponent	King kill opponent from current position	pass

Fourth class: Pawn_class
This is a child piece and is the most basic and fundamental in the game. This piece represents armed peasants or pikemen in the game.

Attributes: can_move (bool): Whether or not the piece can "move" over other chessboard pieces as attacking the opponent.
Colour (str): The colour of the Pawn piece, either 'white' or 'black.’
Can_make_a_take_over (bool): This checks whether the piece can “eat”(make a takeover) an enemy piece.

Test Case number	Test Case description	Test data	Expected results	Actual results	Pass/Fail
1.	Check if the programming is running when the test is done	By printing out a statement such as **“----Test cases running___”**	This statement should be printed out	“----Test cases running___”	pass
2.	Checking the move of the pawn	assertTrue(testPawn.can_move_to(2,3)	Pawn moves from a position to another (2,3)	Dialog box appears showing Valid move (2,3) of the Pawn	pass
3.	Test if a Pawn can kill	assertTrue(testPiece,can_kill)	The queen piece should be able to kill	Queen kill opponent from current	pass

			the opponent	position	

Fifth class: Bishop_class

This is a child class. This is the third most crucial piece in the game. Bishop represents the religion, and he is placed on either side of the king and queen.
The bishop moves diagonally while distant between a row and a column must be the same.

Attributes: can_move (bool): Whether or not the piece can "move" over other chessboard pieces as attacking the opponent.

Test Case number	Test Case description	Test data	Expected results	Actual results	Pass/Fail
1.	Check if the programming is running when the test is done	By printing out a statement such as **"----Test cases running___"**	This statement should be printed out	"----Test cases running___"	pass
2.	Test Bishop movements	diagonalMovementCheck(testBishop) assertTrue(TestBishop.can_move-to(3,3)	bishop piece should move diagonally	The bishop piece moves diagonally	pass
3.	Test if a Bishop can kill	assertTrue(testPiece,can_kill)	The Bishop piece should be able to kill the opponent	Bishop kill opponent from current position	pass

Sixth class: Knight_class

This is a child class. Knights are placed on either side of the bishop. The night moves unconventionally compared to other pieces. Here they move in an "L-shape."

Attributes: can_move (bool): Whether or not the piece can "move" over other chessboard pieces as attacking the opponent.

Test Case number	Test Case description	Test data	Expected results	Actual results	Pass/Fail
1.	Check if the programming is running when the test is done	By printing out a statement such as **“----Test cases running___”**	This statement should be printed out	“----Test cases running___”	pass
2.	Check the move of the Knight	assertTrue(testKnight.can_move_to(2,3))	Knight moves from a position to another	Dialog box appears showing Valid move of the Knight	pass
3.	Test if a Knight can kill	assertTrue(testPiece,can_kill)	The Knight piece should be able to kill the opponent	Knight kill opponent from current position	pass

Seventh class: Rook_class
This is a child class. This piece represents the castle in the game, where it protects the king and the queen.
Attributes: can_move (bool): Whether or not the piece can "move" over other chessboard pieces as attacking the opponent.

Test Case number	Test Case description	Test data	Expected results	Actual results	Pass/Fail
1.	Check if the programming is running when the test is done	By printing out a statement such as **“----Test cases running___”**	This statement should be printed out	“----Test cases running___”	pass

2.	Check the move the rook	straightMovementCheck(testRook)	Rook moves from a position to another	Dialog box appears showing Valid move of the Rook	pass
3.	Test if a Rook can kill	assertRook(testPiece,can_kill)	The Rook piece should be able to kill the opponent	Rook kill opponent from current position	pass