# **GAME PLAN FOR TEEKO**

### Why we chose Teeko:

We chose Teeko because it seemed easier to implement. The board is smaller, and there seems to be less involvement. Teeko is also extremely simple. Basically, Teeko seemed easier. The coding looked hard, so we wanted the easier option.

IMPORTANT FOR USE/GRADING:

We used PyCharm and had to install the following additional modules. You have to add them to your IDE config if not already there.

ffmpeg, pydub, numpy/turtle, math (you probably already have the last two)

There is also an issue with the pydub module accessing temp files that it creates. It gives an RunTime error as well as a permission error. We couldn’t get the code to work on 2/4 of our machines due to this module issue. We spent like 2-3 hours troubleshooting, but we couldn’t figure it out. We did get it to run though, as shown in the attached video.

# **Hierarchy:**

### Functions:

whoseSquare:

Check to see who owns a circle (we are calling them squares because it makes more sense in our coding logic)

Return array value board[square[0]][square[1]]

Include try except statement in case the value doesnt exist

*Win Functions*

winVertical(square, board)

Check to see if all four of the user’s markers are in vertical stack by taking in tuple of current move

Create two while loops to see if squares above and below belong to the current player

Create a count variable = 1 and add 1 for each square above or below that belongs to the player

If count >= 4 return true which will then be referenced later

If count < 4 return false

winHorizontal(move, user, board)

Check to see if all four of the user’s markers are in horizontal row, exact same as vertical but with horizontal

While loops to see if square left or right instead of above or below

Rest is same as winVertical

winDiagonal((move, user, board)

Check to see if all four of the user’s markers are in diagonal row by taking in tuple of current move

Create two while loops, one for left diagonal and one for right diagonal

For both, check if upper corner is valid, if valid, keep going, if not valid go opposite way

Add count same way as winVertical and return same way

winSquare(move, user, board)

Check to see if user’s markers make a square

Check to see if square above is valid, check to see if square to the right is valid, check to see if square to the left is valid, check to see if the square below is valid

If any not valid, dont check the corners that are around the square

Check valid to see if square

If square return true

If not square return false

didMoveWin(move, user, board)

Check to see if user won

If winVertical or winHorizontal or winDiagonal or winSquare return true

Else return false

*Sound Functions*

from pydub import AudioSegment

from pydub.playback import play

import threading

storm\_sound()

Play fortnite storm sound

win\_sound()

Play clip of Dr. O saying ‘you win!’

cam\_beat()

Play cams beat

explosion()

Play explosion sound

jazz()

Play jazz because who doesnt like jazz

place()

Play thud noise like youre putting a piece down

*Turtle Functions*

turt\_time()

radius = 16

import turtle

import math

Create a 5\*5 board via turtle

Create a for loop to create the rows and columns within that for loop

Create another for loop to create the last row

Create x’s

Set the pen position for the next function

turt\_plop(x ,y,color ,speed=0,hide=True)

Plop a color onto a circle

Import turtle

Pen down and fill the circle

write\_win(center\_x, center\_y, text)

Write text on the screen

from turtle import Screen, Turtle, shape, clear

Define variables: board and font

Go to the center coordinates and write

Use screen to display explosion

write\_text\_keep(ypos, text, size)

Write text on the Screen

from turtle import Screen, Turtle

Set board and font

Write in black or red depending on who won

Go to the y position and write

### Program:

*Set up*

from scripts import \* (scripts contains our win functions)

from turt import turt\_time, turt\_plop, write\_text, write\_text\_keep

from sounds import jazz

jazz() (we want to listen to jazz while we play)

Turtle = false

Board = 5x5 array of “0” in each array value

*Begin interacting with user*

# Force user to use turtle or file replay

While input ! = no:

Input for if user wants turtle to draw or file replay

If input = yes:

Turt = true

Display ‘Teeko! By your favorite group!’ using write\_text\_keep

Elif input = fr:

Input for file name or e for exit

If input != e:

Try:

With open file to read as fr:

Call turt\_time() function

Split line up and gather color and coordinates

If length of line == 2:

turt\_plop(x,y,color,3,False)

Elif length of line == 3:

Calculate move using if/elif statement for all 8 directions

And update x,y coordinates correspondingly

turt\_plop(x,y,'White',3,False)

turt\_plop(a,b,color,3,False)

Except:

Print something about a file error

exit()

*More set up*

Create empty list called move\_arch

Dictionaries of user and letter: {‘Black’ : ‘b’, ‘Red’ : ‘r’} (evil and lawful are for weird edge cases)

Turn = ‘Black’ (black starts)

# set up necessary variables

i = 0, GO = True, bt = 0, rt = 0

*Begin drop phase*

While i < 8:

If not using turtle, print all x in board

Get input for turn as a tuple, accept f for forfeit, or accept r for rules

If user enters f print that they lost, set GO = False, and break

If user enters r, print:

The Teeko board consists of twenty-five spaces arranged in a five-by-five grid. There are eight markers in a Teeko game, four black and four red. One player, Black plays the black markers, and the other, Red, plays the red. Black moves first and places one marker on any space on the board. Red then places a marker on any unoccupied space; black does the same; and so on until all eight markers are on the board. The object of the game is for either player to win by having all four of their markers in a straight line (vertical, horizontal, or diagonal) or on a square of four adjacent spaces. (Adjacency is horizontal, vertical, or diagonal, but does not wrap around the edges of the board.) If neither player has won after the drop (when all eight pieces are on the board), then they move their pieces one at a time, with Black playing first. A piece may be moved only to an adjacent or diagonal space.

Continue

Try to split user tuple into x and y variables

Except print invalid input if user doesnt enter valid tuple or letter and continue

*Check if space is valid/open*

if ((not 1 <= x <= 5) or (not 1 <= y <= 5)) print invalid input and continue

if board[y - 1][x - 1] != 'O' print Occupied space and continue

Convert input from list input to human input and append to move\_arch

*Place and switch turn*

place()

If black’s turn, bt += 1 and i+= 1, set turn = red, and plop that space if using turtle

Vice versa for if red’s turn

if (didMoveWin((x-1, y-1), evil\_st[turn], board)):

print(f"{lawful\_st[evil\_st[turn]]} won")

GO = False

break

*Change to move mode*

If GO is true, print DROP COMPLETE

While GO is true:

If not using turtle, print all x in board

Get input for turn as a tuple, accept f for forfeit, or accept r for rules

If user enters f print that they lost, set GO = False, and break

If user enters r, print:

The Teeko board consists of twenty-five spaces arranged in a five-by-five grid. There are eight markers in a Teeko game, four black and four red. One player, Black plays the black markers, and the other, Red, plays the red. Black moves first and places one marker on any space on the board. Red then places a marker on any unoccupied space; black does the same; and so on until all eight markers are on the board. The object of the game is for either player to win by having all four of their markers in a straight line (vertical, horizontal, or diagonal) or on a square of four adjacent spaces. (Adjacency is horizontal, vertical, or diagonal, but does not wrap around the edges of the board.) If neither player has won after the drop (when all eight pieces are on the board), then they move their pieces one at a time, with Black playing first. A piece may be moved only to an adjacent or diagonal space.

Continue

Try to split user tuple into x and y variables

Except print invalid input if user doesnt enter valid tuple or letter and continue

*Check if current user owns the spot*

if ((not 1 <= x <= 5) or (not 1 <= x <= 5)):

print('invalid input')

continue

if board[y - 1][x - 1] == 'O':

print('Empty space')

continue

if board[y - 1][x - 1] != st[turn]:

print('Not your piece')

continue

*Check for block rule/boxed corners*

if/elif statements checking each corner and if your turn is there

print('Boxing is cheap')

continue

*Check for input direction or go back if user wants a different piece*

Ask user for input of arrows (<,>,^,v,v>,etc,) or b (for back)

If user inputs b, print going back and continue

Create if/elif statements for all directions that update x and y coordinates

*Check if new spot empty*

if ((not 1 <= a <= 5) or (not 1 <= b <= 5)) print Out of bounds, going back

continue

if board[b - 1][a - 1] != 'O' print Occupied space, going back

continue

*Turn repeat rule*

Create moveid and set it equal to x coordinate and y coordinate converted to game form

if moveid in move\_arch print('Do something new')

continue

move\_arch += [moveid]

*Place and switch turn*

place()

If blacks turn, switch to reds turn and update board, plop if using turt

Vice versa for reds turn

if (didMoveWin((a-1, b-1), evil\_st[turn], board)):

print(f"{lawful\_st[evil\_st[turn]]} won")

GO = False

break

If using turtle, write ‘\_\_\_ won, congrats!’ to the screen using write\_win function

with open('Game\_Archive.txt', 'w') as GA:

for line in move\_arch:

GA.write(line+'\n')

# 

# **User Manual:**

How to run:

Install the necessary modules and files used in our game. Make sure all of the files are in the same directory as the main file, and run the main file. If everything is set up properly, turtle should begin printing, and you should hear jazz music.

Modules to download: pydub, ffmpeg

It might give a file permission error, there is no way we could find to fix this. The module has a bug at the moment where in some peoples’ computers it tries to open the same file twice, but it doesn’t do it for me (Jordan) and all the code works perfectly fine. If it does run the error, all of the sound functions are next to a comment that says “@sound” so you can comment them out.

Valid inputs with expected outputs:

Upon booting up you must first choose your mode by answering the first statement with yes for turtle mode, fr for replay, or no for console mode. In replay mode, you will then name the data file or type e to exit the program. The program will try to read your file and show the game in Turtle before giving you the ability to exit. The only difference between console and turtle mode is the display, the user guide for both is the rest of this paragraph. First, you may get rules or forfeit at any time by inputting r or f respectively. The first 8 turns are for dropping, where you simply type x,y to place a marker of the current user's color at that spot. 1,1 is bottom left, 5,5 is top right, and 2,4 is 2 columns across 4 rows up. After 8 markers are down, the current user will specify one of their pieces to move via its x,y coordinate and then be asked to input a direction to move via arrow keys (^,v,^>,<v,>) or b to go back and pick a new marker. This will continue until a player has made a line or a square with 4 markers and won.

Brief description:

Our code is built almost entirely upon if/elif/else statements nested inside each other. It works by storing user input in an array that is referenced using coordinates. This is used throughout the entire project, with most implementations being related to Turtle, our file replay, and checking nearby spaces. We have two different sections that are very similar, the drop section and the move section, with most of the differences coming from checking nearby spaces. We also implemented sounds throughout, such as jazz in the background, thuds when placing, the Fortnite zone noise, and even a voice line from Dr. O! These sounds come from functions using threading, defined in the sounds.py file.