***Maze Pseudocode***

Where to find modules

*General Attributes-pg1*

*Colour library-pg1*

*Global variables-pg1*

*Display class-pg1*

*Main menu code-pg1-4, including: -Satisfies Main Menu Requirements Specifications*

* *Button class*
* *Text class*
* *First screen function*

*Maze class (2D array example) - pg4 - Satisfies Maze generation requirements specifications*

*Maze subcclass(es) -pg4-5*

*mainSprite class - pg5-7. Class methods that move the sprite include recursion*

*Key assignment function - pg7-8*

*In game menu class - pg8-9 - Satisfies in-game menu requirements specifications*

*Main game function - pg9*

*Load/Save features -pg9-10 - Satisfies Save/Load requirements Specifications*

*Main function and calling of main function - pg10*

*Save data file structure - pg11*

*Notes:*

* If there is no lineWidth value for a draw function, assume the shape has been drawn filled in

*General Attribute Code*

*IMPORTS*

IMPORT Pygame

IMPORT OS

*Colour library*

CLASS colours:

DECLARE white AS list of integers [255, 255, 255]

DECLARE black AS list of integers [0, 0, 0]

DECLARE buttonBlue AS list of integers [124, 176, 255]

DECLARE buttonDarkC AS list of integers [89, 103, 181]

DECLARE space AS list of integers [29, 41, 81]

DECLARE crimson AS list of integers [220,20,60]

*Global variables*

DECLARE SpriteClock AS pygame.time.clock()

*Display class*

CLASS mainDisplay():

INITIALISATION:

DECLARE screenDisplay AS Surface display(1200px,720px)

DECLARE Display.caption AS ‘Maze’

DECLARE edges AS Rect FROM screenDisplay

END INITIALISATION

*Main Menu Code*

*Button class*

CLASS button (float x, float y, float width,float height, string text, integer fontSize,list\_of\_integers buttonColour,list\_of\_integers borderColour, list\_of\_integers fontColour, list\_of\_integers textX, float textY):

INITIALISATION:

DECLARE x AS parameter x

DECLARE y AS parameter y

DECLARE width AS parameter width

DECLARE height as parameter height

DECLARE buttonBorder AS Rect FROM (x,y,width,height)

DECLARE colour AS parameter buttonColour

DECLARE borderColour AS parameter borderColour

DECLARE textx AS parameter textX

DECLARE texty AS parameter textY

DECLARE textFont AS pygame.font.SysFont FROM (font = ‘Candara’, fontSize = parameter fontSize)

DECLARE textSurface AS Surface FROM textFont.render(text = parameter text, antialiasing = False, colour = parameter fontColour)

END INITIALISATION

METHOD drawButton(display\_class Display):

DRAW Rect buttonBorder ON (parameter Display).screenDisplay WITH (colour = colour, lineWidth = 0)

DRAW Rect buttonBorder ON (parameter Display).screenDisplay WITH (colour = borderColour, lineWidth = 10)

DRAW Surface textSurface ON (parameter Display).screenDisplay WITH (x = x, y = y)

UPDATE pygame.display

END CLASS

*Text class*

CLASS text(float x, float y, integer fontSize, list\_of\_integer fontColour,string text, Display):

INITIALISATION:

DECLARE textFont AS pygame.font.sysFont FROM (font = ‘Candara’, fontSize = parameter fontSize)

DECLARE textSurface AS Surface FROM textFont.render(text = parameter text, antialiasing = False

fontColour = parameter fontColour) DRAW Surface textSurface ON (parameter Display).screenDisplay WITH ( x=parameter x

y=parameter y)

UPDATE pygame.display

END INITIALISATION

*First screen*

FUNCTION firstScreen(display\_class Display):

FILL parameter Display.screenDisplay WITH colour = colours.black

DECLARE title AS text(x=(1200/24)\*10,y=(720/20)\*2,fontSize=96, colour=colours.buttonDarkC, text=’Maze’, Display=parameter Display)

DECLARE startButton AS button(x=(1200/24)\*4, y=(720/20)\*5, width=(1200/24)\*16,height=(720/20)\*2, text=’START’, fontSize=48, buttonColour=colours.buttonBlue, borderColour=colours.buttonBlue, fontColour=colours.buttonDarkC, textX=(1200/48)\*21,textY=(720/40)\*11)

CALL startButton.drawButton(Display=parameter Display)

DECLARE continueButton AS button(x=(1200/24)\*6,y=(720/20)\*8, width=(1200/24)\*12, height=(720/20)\*2,text=’Continue’, fontSize=36, buttonColour=colours.buttonBlue, borderColour=colours.buttonBlue, fontColour=colours.buttonDarkC, textX=(1200/48)\*21,textY=(720/40)\*17)

CALL continueButton.drawButton(Display=parameter Display)

DECLARE quitButton AS button(x=(1200/24)\*6,y=(720/20)\*11, width=(1200/24)\*12,height=(720/20)\*2, text=’Quit’, fontSize=36, buttonColour=colours.buttonBlue,borderColour=colours.buttonBlue, fontColour=colours.buttonDarkC, textX=(1200/48)\*22,textY=(720/40)\*23)

CALL quitButton.drawButton(Display=parameter Display)

DECLARE mainMenu AS Boolean True

WHILE mainMenu IS True:

FOR event IN pygame.event.get():

IF event.type IS pygame.MOUSEBUTTONDOWN:

IF pygame.mouse.get\_position() IS IN Rect startButton.buttonBorder:

SET mainMenu to False

DECLARE player AS mainSprite(mazeID=1)

CALL mainGame(Display=parameter Display, mazeID=1, player=player, loading=False, clock=SpriteClock)

END IF

IF pygame.mouse.get\_position() IS IN Rect continueButton.buttonBorder:

IF os(./sav.txt) EXISTS:

SET mainMenu TO False

CALL loadGame(parameter Display)

*/\*See LoadGame function\*/*

ELSE:

DECLARE errorMessage AS button(

x=(1200/20)\*4,y=(720/20)\*8,  
 width=(1200/20)\*12, height=(720/20\*2), text=’No Save file available’,fontSize=36,

buttonColour=colours.buttonDarkC,

borderColour=colours.space,

textX=(1200/20)\*7,textY=(720/40)\*17)

errorMessage.drawButton(Display= parameter display)

END IF

IF pygame.mouse.get\_position() IS IN Rect quitButton.buttonBorder:

UNINITIALISE pygame

UNINITIALISE python

END IF

END IF

END FOR

END WHILE

END FUNCTION

END CLASS

*Maze class -Includes 2D ARRAYS*

CLASS maze(integer row, integer column, integer mazeID):

INITIALISATION:

DECLARE mazeNo AS parameter mazeID

DECLARE rows AS parameter row

DECLARE columns AS parameter column

DECLARE boxWidth AS float 1200/columns

DECLARE boxHeight AS float 720/rows

DECLARE unrendered AS array [[]]

END INITIALISATION

METHOD drawMaze(surface Display):

DECLARE bx AS float 0

DECLARE by AS float 0

DECLARE boxList AS list of Rect

FOR index IN range(0,length(unrendered)):

FOR i IN range(0,length(unrendered[index])):

IF unrendered[index][i] IS 1:

DECLARE currentBox AS Rect FROM (x=bx,y=by, width=boxWidth height=boxHeight)

APPEND currentBox TO boxList

END IF

SET bx TO bx+boxWidth

END FOR

SET bx TO 0

SET by TO by+boxHeight

END FOR

UPDATE pygame.display

END METHOD

END CLASS

*Maze Subclass - Mazeone*

SUBCLASS(maze) mazeone:

INITIALISATION:

INITIALIZE SUPERCLASS(row=10,column=10,mazeID=1)

unrendered = [[1,1,1,1,1,1,1,1,1,1],

[1,0,0,0,0,0,0,0,1,1],

[1,0,1,1,1,0,1,0,1,1],

[1,0,1,0,0,0,1,0,0,1],

[1,0,0,1,0,1,1,1,1,1],

[1,1,1,1,0,0,0,0,0,1],

[1,0,0,0,1,1,1,0,1,1],

[1,0,1,0,1,1,1,0,1,1],

[1,0,1,0,0,0,0,0,0,1],

[1,0,1,1,1,1,1,1,1,1]]

END INITIALISATION

END SUBCLASS

*Main Sprite class*

CLASS mainSprite(integer mazeno):

INITIALISATION:

DECLARE x AS float 0

DECLARE y AS float 0

DECLARE mazeused AS maze class

DECLARE mazeno AS parameter mazeno

DECLARE speedX AS integer 10

DECLARE speedY as integer 9

DECLARE spawnX as float 0

DECLARE spawnY as float 0

DECLARE currentX as float 0

DECLARE currentY as float 0

END INITIALISATION

METHOD spawn(display\_class Display):

DECLARE width AS float mazeused.boxWidth-20

DECLARE height AS float mazeused.boxHeight-20

SET currentX TO x

SET currentY TO y

DECLARE playerRect AS Rect FROM (x,y,width,height)

DRAW playerRect ON Surface Display.screenDisplay WITH colour = colours.crimson

UPDATE pygame.display

END METHOD

METHOD rectAnimate(pygame.clock clock, display\_class Display):

DRAW playerRect ON Surface Display.screenDisplay WITH colour = colours.black

SET playerRect TO Rect FROM (x,y,width,height)

DRAW playerRect ON Surface Display.screenDisplay WITH colour = colours.crimson

DECLARE currentTime AS pygame.time.get\_ticks()

TICK clock BY 10

END METHOD

METHOD collisionDetection(float expectedX,float expectedY,

maze\_class mazeused):

DECLARE temporaryRect AS Rect FROM (parameter expectedX, parameter expectedY,width,height)

IF temporaryRect COLLIDES WITH ANY rect in mazeused.boxList:

RETURN True

ELSE:

RETURN False

END IF

END METHOD

/\*Movement, including RECURSION\*/

METHOD moveRight(float expected,maze\_class mazeused,display\_class Display):

IF CALLING collisionDetection(parameter expected,y,parameter mazeused) RETURNS False:

SET x TO x+speedX

rectAnimate(SpriteClock,parameter Display)

IF x IS NOT parameter expected:

CALL moveLeft(expected=currentX+(parameter mazeused.boxWidth) mazeused=parameter mazeused, Display=parameter display)

ELSE:

SET currentX TO x

END IF

END IF

END METHOD

METHOD moveLeft(float expected,maze\_class mazeused,display\_class Display):

IF CALLING collisionDetection(parameter expected,y,parameter mazeused) RETURNS False:

SET x TO x-speedX

rectAnimate(SpriteClock,parameter Display)

IF x IS NOT parameter expected:

CALL moveLeft(expected=currentX-(parameter mazeused.boxWidth) mazeused=parameter mazeused, Display=parameter display)

ELSE:

SET currentX TO x

END IF

END IF

END METHOD

METHOD moveUp(float expected,maze\_class mazeused,display\_class Display):

IF CALLING collisionDetection(x,parameter expected,parameter mazeused) RETURNS False:

SET y TO y-speedY

rectAnimate(SpriteClock,parameter Display)

IF y IS NOT parameter expected:

CALL moveLeft(expected=currentY-(parameter mazeused.boxHeight) mazeused=parameter mazeused, Display=parameter display)

ELSE:

SET currentY TO y

END IF

END IF

END METHOD

METHOD moveDown(float expected,maze\_class mazeused,display\_class Display):

IF CALLING collisionDetection(x,parameter expected,parameter mazeused) RETURNS False:

SET y TO y+speedY

rectAnimate(SpriteClock,parameter Display)

IF y IS NOT parameter expected:

CALL moveLeft(expected=currentY+(parameter mazeused.boxHeight) mazeused=parameter mazeused, Display=parameter display)

ELSE:

SET currentY TO y

END IF

END IF

END METHOD

METHOD WIN(Rect screenRectangle):

IF screenRectangle DOES NOT contain playerRect:

RETURN True

END IF

END METHOD

END CLASS

*Key Assignment function*

FUNCTION keys(display\_class Display,maze\_class mazeused,mainSprite\_class playerused, menu\_class menuused):

FOR event2 IN pygame.event.get():

IF event2.type IS pygame.KEYDOWN:

IF event2.key IS pygame.K\_LEFT:

CALL playerused.moveRight( expected=playerused.currentX+mazeused.boxWidth, mazeused=parameter mazeused, Display=parameter Display)

ELIF event2.key IS pygame.K\_LEFT:

CALL playerused.moveLeft( expected=playerused.currentX-mazeused.boxWidth, mazeused=parameter mazeused, Display=parameter Display)

ELIF event2.key IS pygame.K\_UP:

CALL playerused.moveUp( expected=playerused.currentY-mazeused.boxHeight, mazeused=parameter mazeused, Display=parameter Display)

ELIF event2.key IS pygame.K\_DOWN:

CALL playerused.moveDown( expected=playerused.currentY+mazeused.boxHeight, mazeused=parameter mazeused, Display=parameter Display)

ELIF event2.key IS pygame.K\_ESCAPE:

CALL menuused.open(Display = parameter Display)

END IF

END IF

END FOR

WHILE menuused.escMenu:

FOR event3 IN pygame.event.get():

IF event3.type IS pygame.KEYDOWN:

IF event3.key IS pygame.K\_ESCAPE:

CALL menuused.close(Display = parameter Display)

END IF

ELIF event3.type IS pygame.MOUSEBUTTONDOWN:

IF pygame.mouse.get\_position() IS IN Rect menuused.quitButton2:

CALL firstScreen(Display = Display)

ELIF pygame.mouse.get\_position() IS IN Rect menuused.saveButton2:

CALL saveGame(mazeused=parameter mazeused, playerused=parameter playerused)

END IF

END IF

END FOR

END WHILE

END FUNCTION

*In-game menu class*

CLASS inGameMenu():

INITIALISATION:

DECLARE MenuBorder AS Rect FROM (x=(1200/25)\*9,y=(720/20)\*7, width=(1200/25)\*7, height=(720/20)\*7

DECLARE escMenu AS Boolean False

END INITIALISATION:

METHOD open(display\_object Display):

SET escMenu TO True

DRAW Rect MenuBorder ON surface parameter Display.screenDisplay WITH colour=colours.buttonDarkC

DRAW Rect MenuBorder ON surface parameter Display.screenDisplay WITH (colour=colours.space,lineWidth=5)

DECLARE saveButton AS button(x=(1200/25)\*10, y=(720/20)\*8, width=(1200/25)\*5, height=(720/20)\*2,text="SAVE" , fontSize=38, buttonColour=colours.buttonBlue, borderColour=colours.space, fontColour=colours.buttonDarkC,textX=(1200/48)\*22,textY=(720/40)\*17

CALL saveButton.drawButton(Display=parameter Display)

DECLARE quitButton2 AS button(x=(1200/25)\*10, y=(720/20)\*11, width=(1200/25)\*5, height=(720/20)\*2, text="QUIT", fontSize=38, buttonColour=colours.buttonBlue, borderColour=colours.space, fontColour=colours.buttonDarkC, textX=(1200/48)\*22, textY=(720/40)\*23)

CALL quitButton2.drawButton(Display=parameter Display)

UPDATE pygame.display

END METHOD

METHOD close(display\_object Display,maze\_object mazeused, player\_object playerused):

SET escMenu TO False

DRAW Rect MenuBorder ON surface parameter Display.screenDisplay WITH colours=colours.black

DRAW Rect MenuBorder ON surface parameter Display.screenDisplay WITH colour=colours.black,lineWidth=5

CALL mazeused.drawMaze(Display=parameter Display)

DRAW Rect parameter playerused.playerRect ON parameter Display.screenDisplay WITH colours=colours.crimson

UPDATE pygame.display

END METHOD

END CLASS

*Main Game Function*

FUNCTION mainGAME(display\_object display,maze\_object currentMaze,mainSprite\_object currentPlayer,Boolean loading,pygame.clock(),clock):

DECLARE menu AS inGameMenu object

DECLARE Maze1 AS mazeone() object

DECLARE maze1bool AS Boolean False

IF loading IS True:

IF currentMaze IS 1:

SET maze1bool TO True

SET parameter currentPlayer.mazeused TO Maze1

CALL Maze1.drawMaze(Display = parameter display)

CALL parameter currentPlayer.spawn(Display = parameter display)

END IF

ELIF loading IS False:

IF currentMaze IS 1:

SET maze1bool TO True

SET parameter currentPlayer.x TO (1200/10)+10

SET parameter currentPlayer.y TO (720/10)+10

SET parameter currentPlayer.mazeused TO Maze1

CALL Maze1.drawMaze(Display = parameter display

END IF

END IF

WHILE maze1bool: CALL keys(Display=display,mazeused=Maze1,playerused=parameter currentPlayer,menuused=menu)

IF parameter currentPlayer.win(screenRectangle = parameter display.edges) RETURNS True:

FILL parameter display.screenDisplay WITH colour = colours.black

DECLARE winningText AS text(x=(1200/25)\*10, y=(720/20)\*9, fontSize=76, fontColour=colours.white,text=”You win!”, Display=parameter display)

END WHILE

END FUNCTION

*Load/Save game code*

FUNCTION saveGame(maze\_class mazeused,mainSprite\_class playerused):

DECLARE saveFile AS file(type=write,overwrite=True)

WRITE LINE TO saveFile STRING(mazeused.mazeno)+’\n’

WRITE LINE TO saveFile STRING(playerused.x)+’\n’

WRITE LINE TO saveFile STRING(playerused.y)+’\n’

CLOSE saveFile

END FUNCTION

FUNCTION loadGame(display\_class Display)

DECLARE saveFile AS file (read) FROM open(sav.txt)

DECLARE data AS list

FOR line IN saveFile.readlines():

APPEND line TO data

END FOR

FOR number IN range(0, length(data):

REMOVE ‘\n’ FROM data[number]

END FOR

DECLARE loadedPlayer AS mainSprite(mazeID = INTEGER(data[0])

SET loadedPlayer.x TO FLOAT(data[1])

SET loadedPlayer.y TO FLOAT(data[2])

CLOSE saveFile

CALL mainGame(Display=parameter Display, mazeID=INTEGER(data[0]), loading=True,clock=SpriteClock)

*/\*See Main Game function\*/*

END FUNCTION

*Main Function (Starting off whole game), upon python opening,*

FUNCTION main():

INITIALIZE pygame

DECLARE display AS mainDisplay object

CALL firstScreen(display=display)

WHILE True:

FOR event IN pygame.event.get():

IF event.type IS pygame.QUIT():

UNINITIALISE pygame

UNINITIALISE python

END IF

END FOR

END WHILE

END FUNCTION

*Calling main function*

CALL main()

*Save file structure*

Save file is a .txt file. The first line in the txt file is the mazeID, the second line is the player’s x position and the third line is the player’s y position.

Example of a save file structure:

sav.txt

---------------------------------------------------------------------

1

200

400