Odd Semester (2021)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Algorithm & Programming
Final Project
Minesweeper

Student Information:

Name: Wilbert Wirawan Ichwan **NIM:** 2501963186

Course Code : COMP6047001 **Course Name :** Algorithm and Programming

Class: L1AC **Lecturer:** Jude Joseph Lamug Martinez, MC

Project Specification

- ## Objective

  The sole objective for my project is to create a game that is fun for everyone to play. The game is very simple and suitable for all ages to play.

- ## Description

  I recreated the legendary “Minesweeper” game. Minesweeper is a single-player puzzle video game. It may be a puzzle game but it is very simple and only requires you to use your mouse to play. The objective of the game is to clear a rectangular board containing hidden bombs without detonating any of them, with help from the numbers you get by clicking on blocks. The numbers represents the number of bombs that are next to it. if it shows a 1, that means that there is only 1 bomb around it (the 8 blocks sorrounding it). You click using your left mouse click and you can flag a place where you know where a bomb is by using right click. By flagging a block, you will not be able to open (left click) said block so you can't accidentally open it and die.

- ## The Modules used

  For this project I only used pygame. Pygame is a cross-platform set of Python modules designed for writing video games.

Project Design

I have 4 python files for this project:

1. MineClass.py
2. BoardClass.py
3. PiecesClass.py
4. Minesweeper.py

3 of the 4 files are class files, the logic behind the game and the last file is the main file to run the game (Minesweeper.py).

1. MineClass.py

The purpose of this class file is that it's the main game file. Every other class file will go through here.

```
import pygame
import os
import time
from PiecesClass import *
from BoardClass import *
```

First, I imported the main module, pygame.Then, I also imported other modules like time and os some time while coding, the use of the time module is to use it for the **time.sleep** function so I can add a delay to the win and lose states. The use of the os module is to use the function **os.listdir** to go to the game directory to retrieve the image files.

- I initialize the file with **def __init__():**

```
class Game:
    def __init__(self, SSize, board): #Initializer
        self.board = board
        self.SSize = SSize
        self.gridsize = self.SSize[0] // self.board.getSize()[1], self.SSize[1] // self.board.getSize()[0]
        self.Limg()
```

- Then I made the main game function to run the game itself.

```
def game(self):
    pygame.init()
    self.screen = pygame.display.set_mode(self.SSize)
    pygame.display.set_caption("Classic Minesweeper") #just a caption (Top left window name)
    run = True
    while run: #basically while game is running
        for event in pygame.event.get():
            if (event.type == pygame.QUIT): #Quitting the gmae
                run = False
            if (event.type == pygame.MOUSEBUTTONDOWN): #Click
                position = pygame.mouse.get_pos()
                rightClick = pygame.mouse.get_pressed()[2] #Index 2 is for right click
                self.Click(position, rightClick)
        self.grid()
        pygame.display.flip()
        if self.board.getWon(): #Win state
            print("You won! =D")
            vic = pygame.mixer.Sound("Minesweeper/victory.wav") #Plays a sound of 8bit victory =D
            vic.set_volume(0.01)
            vic.play()
            time.sleep(3)
            run = False #Closes game
        elif self.board.getLose(): #Lose state
            boom = pygame.mixer.Sound("Minesweeper/explosion.wav") #Plays a sound of 8bit explosion =D
            boom.set_volume(0.01) #I set it so low cause it jumpscared me when i tested it T_T
            boom.play()
            print("You Lost :(")
            time.sleep(4)
            run = False #Stops loop
    pygame.quit()
```

As you can see from the picture above, I set a **while True:** loop to run the game. Basically, while run = True, pygame will take in events such as mouse clicks and if the game has been closed or not. **if (event.type == pygame.QUIT):** means that if the game is closed, it will stop the while loop and quit the game. The 2nd if will detect if the user clicks their mouse. It will first get the position with the **get_pos()** function and secondly, it will detect if the mouse click is a right click or not. If it is, it will go to the **Click** function which I will explain further later. I also made the win and lose state in the main game function, it will play a sound if you win or lose.

- Then I drew the board with this function

```
def grid(self): #Draw the board
    topLeft = (0, 0)
    for row in range(self.board.getSize()[0]):
        for col in range(self.board.getSize()[1]):
            piece = self.board.getPiece((row, col)) #Getting the peice
            image = self.getImg(piece)#Get image based on piece
            self.screen.blit(image, topLeft) #Placing images in the correct places
            topLeft = topLeft[0] + self.gridsize[0], topLeft[1]
        topLeft = 0, topLeft[1] + self.gridsize[1]
```

This function is to draw the board. Usually the 0,0 coordinate is in the middle of the screen, but for pygame it's on the topLeft. I made the for loop for each row and column then it gets the pieces and places it starting from the top left, and it will gradually move to the right and keeps filling it until it fills the entire board.

- This function is to load the images from the images folder

```
def Limg(self): #Load Images
    self.imgs = {}
    for fileName in os.listdir("Minesweeper/images"):
        img = pygame.image.load(f"Minesweeper/images/{fileName}")
        img = pygame.transform.scale(img, self.gridsize) #Scales the image
        self.imgs[fileName.split(".")[0]] = img #removes the png
```

This function is to load the images from the images directory. I used a for loop to get every file inside of the folder and for every image, I scale it based on the block size of the grid and then I split the name to remove the .png part of the filename.

- This function will get the image based on the actions that were done

```
def getImg(self, piece): #Change the images based on actions
    string = ""
    if piece.getOpened():
        if piece.getBomb():
            string = "bombclicked" #If you click on a bomb
        else:
            string = str(piece.getNumAround()) #If you click on a number, it will get the images based on the number
    else:
        if piece.getFlagged():
            string = "flag" #This is for the flagged status
        else:
            string = "empty" #If its not opened, and not flagged, then yea its empty =D
    if self.board.getLose():
        if piece.getBomb():#If user lose, it will reveal all the bombs
            if not piece.getOpened():
                string = "bomb"
            else:
                string ="bombclicked"
    if self.board.getWon():
        if piece.getBomb(): #If user wins, it will auto flag the blocks that has not been flagged yet.
            if not piece.getOpened():
                string = "flag"
            else:
                string = "empty"
    return self.imgs[string] #Returns the image so it can be placed to each block
```

This is the get image function, it is mainly used to get the images based on the actions that the user does. I first set the string to an empty string so I can replace it with other strings later. Secondly there are a bunch of if statements to thoroughly judge a user's input. If the block is opened and it has a bomb, then the image would be the bombclicked image and the user dies. If its opened and doesnt have a bomb then it will place an image based on the number of bombs around it using the **getNumAround()** function. If the block is not opened, then it will decide if it's flagged or not, if it is, then it will change the image to the flag image, if it isn't, then it's just an empty block cause nothing has happened to it.

The 2nd main if statement is to judge whether a player has died or not. If the player has died and the piece has a bomb and its not opened, then it will change the image to a non-clicked-bomb. Else, it will be a clicked bomb.

And lastly, the 3rd main if statement is to judge whether the player has won or not. If the player has won and some of the remaining blocks still has a bomb in it, then it will automatically place a flag on the blocks. Else, its an empty block.

- And the last function in this class will be the handle clicking function.

```
def Click(self, position, rightClick):
    coords = position[1] // self.gridsize[1], position[0] // self.gridsize[0] #Getting the coordinate for the blocks
    piece = self.board.getPiece(coords) #Gets the piece based on the position (coordinates, in pygame top left is (0, 0))
    self.board.Click(piece, rightClick)
```

It will get the "coordinates" or index of the mouse position. Starting top left is 0,0 then it will go 0,1 if it goes one to the right and so on and so forth. Secondly, it will get the piece of the block based on the index of the block by using the getPiece function.

2. BoardClass.py

The purpose of this class file is to make the board. It will set up the board, like setting the numbers.

```
import random
from PiecesClass import *
```

First, I imported a module called random, the purpose of this module is to make the spawn rate of the bomb. Second, I imported all of the functions from PiecesClass.py

- The Initializer

```
class Board: #This is the Board class, mostly for stuff around the board
    def __init__(self, size, prob): #THis is the Initializer of the code
        self.size = size #The size of the game (blocks)
        self.prob = prob #The probability of the block being a bomb
        self.lose = False #Lose obviously false in the beginning or else already GG we lose =D
        self.numOpened = 0 #The number of opened Blocks
        self.setBoard()
```

This function is pretty self explanatory, it is just the initializer of this class.

- The setBoard function

```
def setBoard(self): #Setting up the board
    self.numNotBombs = 0 #The number of blocks that does not have a bomb in them
    self.board = []
    for row in range(self.size[0]): #Row is size index 0
        row = []
        for col in range(self.size[1]): #column is size index 1
            Bomb = random() < self.prob #Bombs
            if not Bomb:
                self.numNotBombs += 1 #Every block that isnt a bomb, numNotBombs + 1
            piece = Pieces(Bomb) #pushes the bomb to piece
            row.append(piece)
        self.board.append(row)
    self.setNumbers()
```

This is the function to properly set up the board. First, I made a variable to count how many blocks that aren't a block in the board. Then I made a for loop to loop each row and column block per block. Inside the loop, theres a bomb variable which will generate a random number of bombs based on the probability of there being a bomb. Then I made an if statement to see if a block is a bomb or not. If it is not a bomb, it will increase the **numNotBombs** variable. And then it the Bomb variable will get pushed to the Pieces class for further process.

- The setNumbers function

```
def setNumbers(self): #setting the numbers
    for row in range(self.size[0]):
        for col in range(self.size[1]):
            piece = self.getPiece((row, col)) #getting the pieces for each coods
            numbers = self.getNumList((row, col)) #getting the numbers for each coods
            piece.setNumbers(numbers) #Setting each piece's numbers based on the list of numbers that it got from getNumList
```

This function is used to set the numbers in the board based on how many bombs are around it. I did the usual for loop to go to every block by column and row. Then it will get the piece 1 by 1 using the **getPiece** function and it will get the number of the said piece by using the **getNumList** function, then it will set the number to the piece.

- The getNumList function

```
def getNumList(self, coords): #Getting the numbers for the adjacent blocks
    numbers = []
    for row in range(coords[0] - 1, coords[0] + 2):
        for col in range(coords[1] - 1, coords[1] + 2):
            if row < 0 or col < 0 or row >= self.size[0]  or col >= self.size[1]:
                continue
            numbers.append(self.getPiece((row, col)))
    return numbers
```

This is the function to get the neighbors of the piece of interest. It will go to each block with the for loop. The if statement is basically just to make sure the row and column doesn't go out of bounds. Then it will append the the piece to the numbers and return it.

- The get functions

```
def getSize(self): #Self explanatory get funcs
    return self.size

def getPiece(self, index):
    return self.board[index[0]][index[1]]

def getLose(self):
    return self.lose

def getWon(self):
    return  self.numNotBombs == self.numOpened #If the numbe
```

These functions should be self explanatory.

- The Click handling function

```
def Click(self, piece, flag):
    if piece.getOpened() or (not flag and piece.getFlagged()): #If
        return
    elif flag:
        piece.toggleFlag() #if we right clicked we flag the block
        return
    piece.click() #opens the block
    if piece.getBomb():
        self.lose = True #if block has bomb we lose
        return
    self.numOpened += 1
    if piece.getNumAround() != 0: #Checks if a neighboring block i
        return
    for i in piece.getNumbers(): #If the block is 0 , itll keep op
        if not i.getBomb() and not i.getOpened():
            self.Click(i, False)
```

This is the handle clicking function. It is basically the function to make us able to interact with the game. It will check the state of the block everytime we click.

3. PiecesClass.py

The PiecesClass file is just a class with the get and set function for the program to work.

I didn't need to import any module into this Class because it doesn't need anything from other classes nor other modules.

- The Initializer

```
class Pieces:
    def __init__(self, Bomb): #Initializer
        self.Bomb = Bomb
        self.opened = False
        self.flagged = False
```

Self.opened is false on default because the block isn't opened yet, same thing goes for the self.flagged.

- The get functions

```
def getBomb(self): #Self explanatory get funcs
    return self.Bomb

def getOpened(self):
    return self.opened

def getFlagged(self):
    return self.flagged

def getNumbers(self):
    return self.numbers

def getNumAround(self):
    return self.numAround
```

Self explanatory get functions

- The click and flag function

```
def toggleFlag(self): #toggling the rightClick flag
    self.flagged = not self.flagged

def click(self): #Handle click
    self.opened = True
```

If the person clicks, the block is opened. And as for the flag, it's a toggle function that can switch between flagged and unflagged.

- The setNumbers function

```
def setNumbers(self, numbers):
    self.numbers = numbers
    self.setNumAround()
```

Setting the numbers on the board, called on the BoardClass.py, calls the setNumAround function to set the numbers around the block of interest.

- The setNumAround function

```
def setNumAround(self): #setting the number according the amount of bombs it has around it
    self.numAround = 0
    for i in self.numbers:
        if i.getBomb():
            self.numAround += 1 #If it has a bomb, the number will rise to the amount of bombs it has around it
```

What this function does is it basically getting the correct number of bombs around it. It first has the numAround value set to 0 and then it does a for loop to detect if the neighboring blocks has a bomb or not. If it does it will increase the value by 1 for every bomb.

4. Minesweeper.py

This file is the main file where the game runs.

```
import time
import os
from MineClass import *
from BoardClass import *
```

I imported the time and os modules. I used the time module for the **time.sleep** function and I used the os module to do the **os.system("cls")** function. I will explain further below.

- The tutorial

```
print("Welcome to the classic minesweeper game!")
time.sleep(2)
print("Are you new to minesweeper? (y/n)")
tutor = input("")
if tutor.lower() == "yes" or tutor.lower() == "y":
    print("The objective of the game is to clear a rectangular board that contains hidden bomb without detonating them!")
    time.sleep(2)
    print("As long as you dont hit the bombs, numbers or empty blocks will appear!")
    time.sleep(2)
    print("Use those numbers to help you through this puzzle game!")
    time.sleep(2)
    print("The numbers represents the number of mines or bombs that are on the neighboring blocks!")
    time.sleep(2)
    print("Good Luck and Have Fun!")
    time.sleep(2)
elif tutor.lower() == "no" or tutor.lower() =="n":
    pass
#This is just a tutorial of the game =D
#To make sure everyone can play the game :)
```

I made a simple and easy to understand tutorial for minesweeper. It first asks if the user is new to minesweeper or not, if they are it will show a tutorial. If not, it will continue the program. I used a bunch of **time.sleep** functions here just so the tutorial does not just all pop out.

- Customizing the board

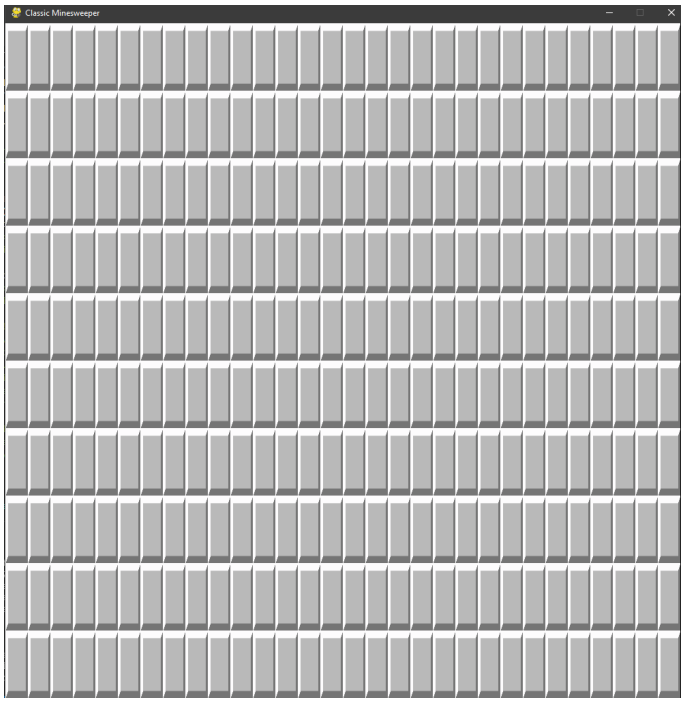
```
print()
print("Make sure that you make a square, width and height has to be the same!")
time.sleep(2)
h = int(input("How many blocks (height) do you want to have in the board? (min. 5, max. 30): ")) #height of board
while h < 5 or h > 30:
    h = int(input("How many blocks (height) do you want to have in the board? (min. 5, max. 30): "))

w = int(input("How many blocks (width) do you want to have in the board? (min. 5, max. 30): ")) #width of baord
while w < 5 or h > 30:
    w = int(input("How many blocks (width) do you want to have in the board? (min. 5, max. 30): "))

while w != h:
    os.system("cls")
    print("Make sure that you make a square, width and height has to be the same!")
    time.sleep(2)
    h = int(input("How many blocks (height) do you want to have in the board? (min. 5, max. 30): ")) #height of board
    while h < 5 or h > 30:
        h = int(input("How many blocks (height) do you want to have in the board? (min. 5, max. 30): "))

    w = int(input("How many blocks (width) do you want to have in the board? (min. 5, max. 30): ")) #width of baord
    while w < 5 or h > 30:
        w = int(input("How many blocks (width) do you want to have in the board? (min. 5, max. 30): "))
```

This is just a basic while loop to make sure that the height and width is at minimum a 5 and a maximum a 30. There is also a while loop to make sure the board is a square so the block size does mess up.



This is what the board will look like if it's a rectangle. This is because of how I made the block size.

```
self.gridsize = self.SSize[0] // self.board.getSize()[1], self.SSize[1] // self.board.getSize()[0]
```

I got the block size by floor dividing the screen size (1000,1000) with the size of the board (the width x the height)

```
size = (h, w) #The size of the board, i usually play 20x20
```

Setting the size of the board.

```
prob = float(input("Whats the bomb probability u want to have? (in decimal, (i.e. 0.1 is 10%)) (min 0.1, max 0.9): ")) #The
while prob < 0.1 or prob > 0.9:
    prob = float(input("Whats the bomb probability u want to have? (in decimal, (i.e. 0.1 is 10%)) (min 0.1, max 0.9): "))
```

The probability of a bomb. It has a max value of 0.9 (90%) and a minimum value of 0.1 (10%)

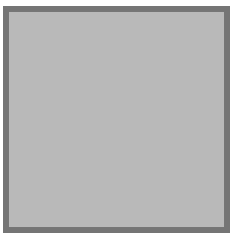
- The main run function

```
board = Board(size, prob) #To set up the board
SSize = (1000, 1000) #The screen size
mine = Game(SSize , board)
mine.game() #The main run function
```

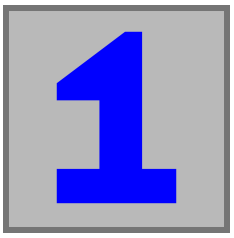
I first called the Board class to set up the board. Then I set the size of the screen (resolution) to 1000x1000. Then, I called the main game function with 2 parameters of the screen size and the board. Lastly, I run the game with the **mine.game()** function.

5. The Resources (Images and Sounds)

- The number 0 block (0.png)

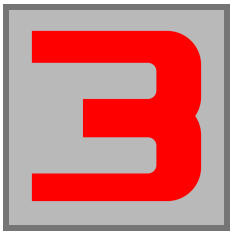


- The number 1 block (1.png)



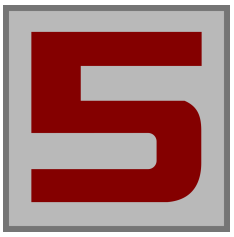
- The number 2 block (2.png)

- The number 3 block (3.png)



- The number 4 block (4.png)

- The number 5 block (5.png)

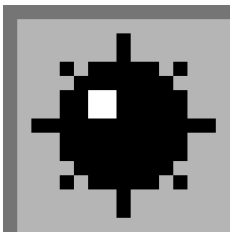


- The number 6 block (6.png)

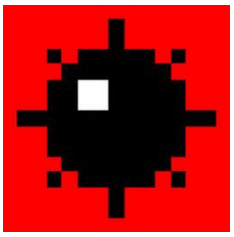
- The number 7 block (7.png)

- The number 8 block (8.png)

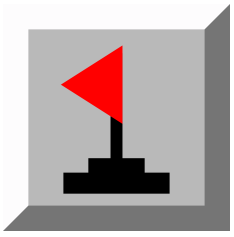
- The uncovered mine block (bomb.png)



- The clicked bomb image (bombclicked.png)

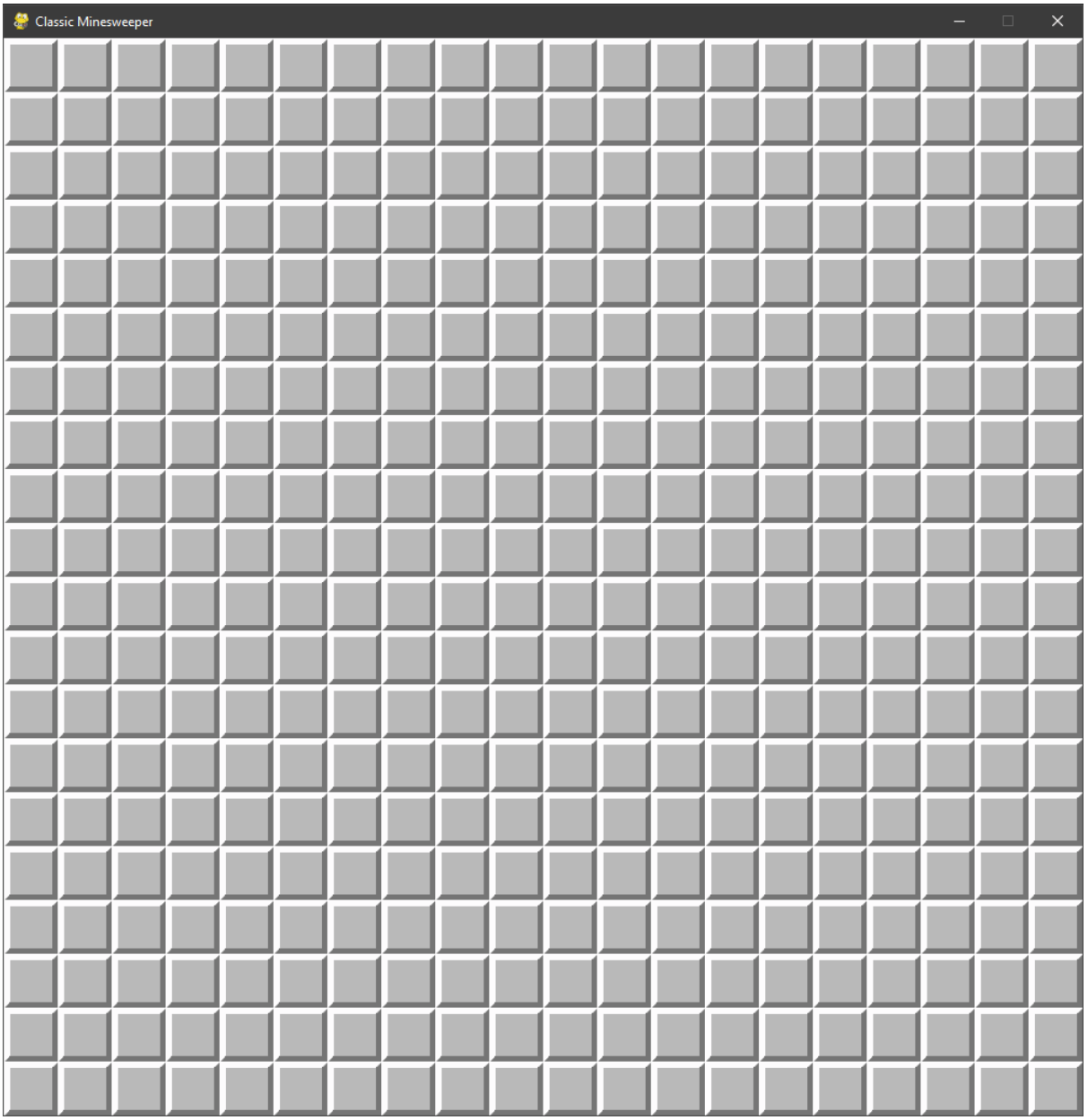


- The flag image (flag.png)



- The covered block image (covered.png)

- The victory sound (victory.wav)
- The lose sound (explosion.wav)



This is what the game will look like if you do a 20x20 board