## horizontal line



Our Very First Minesweeper

01.12.2022

**─**

Elaine, Jing Fang, Melvin, Sean, Yan Qi, Ye Chuan

Sum Uni Kids

(Please think of better name)

Your City, ST 12345

# Overview

We will be making a simple minesweeper game.

This Game Design Document (GDD) will be used as a guide to make sure we are all on the same page throughout the project.

Since this is a collaborative project, we will need to discuss how to split the project up into smaller tasks, and assign the person-in-charge for each task.

More than anything, this is supposed to be a learning experience for all of us, so let’s try not to be disheartened when things get difficult :>

# Goals

1. Learn more about how to build an actual project, and get familiar with using objects in Python
2. Get into the habit of following good coding practices and standards :D
3. Make a fully functioning minesweeper game of course!

# Coding Standards

I know coding standards can be a little turn off because “everyone has their own coding style” but c’mon let’s all agree that good code also means consistent code.

Nobody really cares how you test or write your code in private, but I think it’s fair to expect that eventually, all code that gets pushed up into our repository must be consistent with the same style.

Bigger teams usually have everyone install linters to check their coding styles, but it’s just a few of us here so let’s not go through that trouble yeah, anyways it’s good practice to practice these anyways.

Some of these are subjective so we can always discuss if anyone wants a change.

However, most of these standards here are according to Python’s Official Style Guide, [PEP8](https://peps.python.org/pep-0008/), and is generally adhered to industry-wide so we should really stick to it.

## 

## Strings

Double Quotation “ for all strings (like in C :>)

## Identifiers

**Variables / Objects** - snake\_case (as per PEP8 + Pygame Standard)

**Constants** - CAPITALISED (as per PEP8 + Pygame Standard)

**Functions** - snake\_case (as per PEP8 + Pygame Standard)

**Classes** - PascalCase (as per PEP8 + Pygame Standard)

**Private Variables** - \_leading\_underscore (as per PEP8)

## Layout / Spaces

**Indentation**

4 spaces (most code editor allows for Tab to mean 4 spaces, IDLE doesn’t tho)

**Comments**

Always start comments with 1 space after the #

*# This is a comment*

**Argument/Parameter List**

1 space after the comma, none before

Keyword arguments & Default Arguments: no spaces around their =

def my\_func(arg1, arg2, kwarg1="somestr")

*(Type-Hinting not shown here, we will decide if we want to Type Hint)*

**Assignments / Comparators**

1 space before and after the =

var1 = “value”

var1 >= var2

**Operator Expressions**

Generally have 1 space on both sides of the operator.

However, for more complex expressions, it’s good to break the standard for readability

x = 2\*5 + 3\*5 - 10\*\*2

is preferred over the general rule:

x = 2 \* 5 + 3 \* 5 - 10 \*\* 2

**Line Length**

There’s also a general rule to keep lines **up to 79 characters**, this is so that we don’t need to side scroll.

To achieve this we can break longer expressions into multiple lines using parenthesis continuation, or a \

However, I wouldn’t set this as a hard limit, going over a little every now and then is fine.

## Type Hinting

We will decide if we want to type-hint.

## Docstring

Docstrings are crucial when collaborating. Since Python didn’t really standardize docstring format, we will use the [Google Python Style Guide](https://google.github.io/styleguide/pyguide.html) which Google Devs follow.

All functions must have a docstring unless it fulfills ALL of the following:

* not externally visible (say a nested function, or a private method)
* very short
* obvious

The point is that anyone else using your function should be able to use it without having to read the actual code.

If the function is simple, a one-liner docstring will do:

**def** adder(x, y):

"""Adds 2 numbers"""

**return** x + y

Docstrings should be **descriptive**, not **imperative**. Meaning it should tell us what the function does, *not tell the function what to do*.

"""Adds 2 numbers"""

NOT

"""Add 2 numbers"""

For a more detailed docstring:

"""Brief description of the function here

Elaboration if required.

Blah blah.

Args:

arg1 (type): What is arg1's role in the function

n (int): What n is

Returns:

(Optional if your brief description is something like "Returns a ...")

A 2D int matrix where `1` = Mine, `0` = No Mine

Example:

[[0,0,0,0],

[0,1,0,1],

[0,1,1,0],

[1,0,0,0]]

Raises:

TypeError: Explain the various useful errors our function might raise if any

CustomError: Blah Blah

""" *# Notice this """ is on a newline by itself*

Classes and modules should also have their own docstring, you can find the format through the link above.

# Progress

## Kickoff Meeting

Be on the same page with the coding standard to follow.

Setting up our GitHub repository.

Familiarization with Object-Oriented Programming.

## Ideation

Preferably during the Kickoff meeting.

Decide on the general feel and look of our minesweeper (see Specifications)

First assignment of tasks

## Implement Minefield Backend

* Sean implements Cell class’s expose method
* Melvin generates minefield
* Ye Chuan work on creating a GUI button

## Integrate Frontend & Backend

* Sean works on creating a timer
* Melvin works allowing user to expose around an already exposed cell
* Ye Chuan works on generating minefield after first press using the backend Melvin method’s built

See [Current Work](#_4p0725mqa1f0) work specifics

## ???

## ???

## Profit!

We get a nice working minesweeper!

# Specifications

We will collate the different things we’ve discussed related to our Minesweeper here.

## General UI

Color Palette: ?

### Buttons

* Pause
* Resume

## Game Interface

Minefield Size: 16x16

Timer: Yes

## Mechanism

Using a 2D array of Cell instances to store the minefield.

Cell class with attributes

* val - No. of mines around
* is\_mine - If cell is a mine
* coord - (row, col) coordinates of cell in minefield
* exposed - if cell is exposed by user
* minefield - The Minefield object that the cell is part of

Methods

* expose(self)- Invoked when user clicks the cell
* expose\_around(self)- Exposes the cells around this

# 

# Current Work

## Melvin

Cell’s .expose\_around should **only expose unflagged cells**

* This is to integrate with the pressing on revealed cell feature, where only unflagged cells are exposed
* This will not negatively affect the expose\_around() when the cell is 0
* The idea is that if the player thinks there's a mine at a cell, we will not expose it for them even if they are wrong
* Alternatively, you could also decide to just expose the flagged cells either ways, which is the classic minesweepers behavior, but I think is less fun :>
* If you choose the alternative then nothing really needs to be done to .expose\_around for now

Implement a .attempt\_expose\_around() in Cell that checks for the flags around and only call .expose\_around() only if the number of flags is correct

* This will be called when the player clicks on an opened cell in an attempt to expose the cells around it
* Can implement a .flags\_around() for Cell as a helper function

## Sean

Game Timer

* As of now just get a nice working timer that starts when the program starts
* We will implement starting only after clicking 1st cell later on

## Ye Chuan

Generate the minefield only after user clicks on the first cell

* In our current implementation, having a clickable board implies already having generated the minefield (else the board wouldn't exist)
* So one way is to create temporary minefield first, then after the player clicks we quickly generate another to replace this temporary minefield

# Work 2

## Melvin

* Make it such that the first cell opened is **never a mine**
* Can go the extra mile to give **an option** for the first opened cell to always be 0
  + Give an option because this is non-standard minesweeper

## Sean

* Create a Timer (at top left) using Pygame
* Display number of bombs not flagged yet (Bombs - Flagged)
  + Don’t care if it’s correctly flagged or not

## Ye Chuan

Implement a button that will eventually work as cells in the minefield

# 

# Work 1

## Melvin

Create a 16x16 Minefield of Cell objects

* Each cell.coord = Coordinate (y, x) starting top-left of this cell
* Each cell.val = No. of mines around this cell (None if is\_mine)
* Each cell.is\_mine = Boolean

Minefield would look like:

>>> minefield

[[Cell, Cell, Cell, ...],

[Cell, Cell, Cell, ...],

[Cell, Cell, Cell, ...],

... ]

Implementation:

E.g.

>>> minefield[0][0]

<Cell Object>

>>> minefield[0][0].val

3 # (means 3 mines around the top-left cell)

>>> minefield[2][1].is\_mine

True # (means the 3rd row, 2nd cell is a mine)

## 

## Sean

* Work on expose()

class Cell():

def \_\_init\_\_(self, coord, val, is\_mine=False, is\_flagged=False, exposed=False):

self.coord = coord # coord = (y, x) where top-left is 0,0

self.val = val # Number of bombs around this cell (None if is\_mine)

self.is\_mine = is\_mine # Boolean

self.is\_flagged = is\_flagged # Don't care for now

self.exposed = exposed # Sean will take care of this

def expose(self, minefield):

# Sean will work on this

# Expose this cell and other cells around it if 0

pass

## Ye Chuan

Implement a button that will eventually work as cells in the minefield