2805ICT, Minesweeper Project

Natnicha Titiphanpong, s2940970

September 22, 2017

Contents

1 Minesweeper

1.1 Implementation of Task 1

The current version of minesweeper successfully meets the requirements of the traditional minesweeper game.

The programming language chosen - Python allows the program to be executable on Windows and Linux. The Graphic User Interface (GUI) modules are available to both platforms. Therefore, when creating a program that requires a GUI, Python was considered first when accounting for cross-platform applications.

2 Software Development Practices

2.1 Version Control History / Log

Date	User	Activity
Sep 22, 2017	Natnicha	Modified button bindings to deal with left and right handler properly.
Sep 22, 2017	Natnicha	Ajusted model to refer to boolean list for toggled and flagged for left and right click.
Sep 21, 2017	Natnicha	Finalised MainMenu class. Destroys Frame after selecting difficulty.
Sep 20, 2017	Natnicha	MainMenu class: created set difficulty function to be passed to the model.
Sep 19, 2017	Natnicha	Finalised Button Controller class to be a Facade for model.
Sep 15, 2017	Natnicha	Imported images for Main Menu.
Sep 5, 2017	Natnicha	Made all views extend tkinter.Frame.
Sep 5, 2017	Natnicha	Added MainMenu class, extends tkinter.Frame.
Sep 4, 2017	Natnicha	Resolved Merging conflict.
Sep 4, 2017	Natnicha	Added diagrams to Report
Sep 3, 2017	Natnicha	Separated create board function in three separate functions for low coupling, removed nested functions.
Sep 2, 2017	Natnicha	Added UML section to report. Put MV files into classes.
Sep 1, 2017	Natnicha	Added timer and reset button functions to Program.
Sep 1, 2017	Natnicha	Added test section to Report.
Aug 31, 2017	Natnicha	Modified test file and fixed formatting in report. Restructured program into MVC architecture.
Aug 31, 2017	Natnicha	Merging conflict resolved.
Aug 31, 2017	Natnicha	Merging files
Aug 30, 2017	Natnicha	Modified Software Architecture section in Report.

Aug 30, 2017	Natnicha	Completed use case progression and added software architecture section to Report.
Aug 30, 2017	Natnicha	Added more use cases.
Aug 30, 2017	Natnicha	Modified activation button function to handle functionality depending on the backing grid
Aug 29, 2017	Natnicha	Added game end function to handle the program when the game is over.
Aug 29, 2017	Natnicha	Fixed cascading reveal to display empty cells and cells adjacent to.
Aug 29, 2017	Natnicha	Added colour scheme to GUI minesweeper.py
Aug 26, 2017	Natnicha	Added images to buttons for GUI.

Note: Repository is private. | This log was created by using the command

```
git log --pretty=format:'%h;%an;%s' > ./log.csv
```

3 Design Principles

Explain design principles including of least privilege and fail-safe defaults, separation of concerns, information hiding, coupling and cohesion, and encapsulation.

4 Design Process

Describe the design process for a software development project for each of the main software design methods

The minesweeper project used an agile software development process,

5 System Models

Create appropriate system models for the structure and behaviour of software products from their requirements specifications Note: discussion of code and UML may suffice

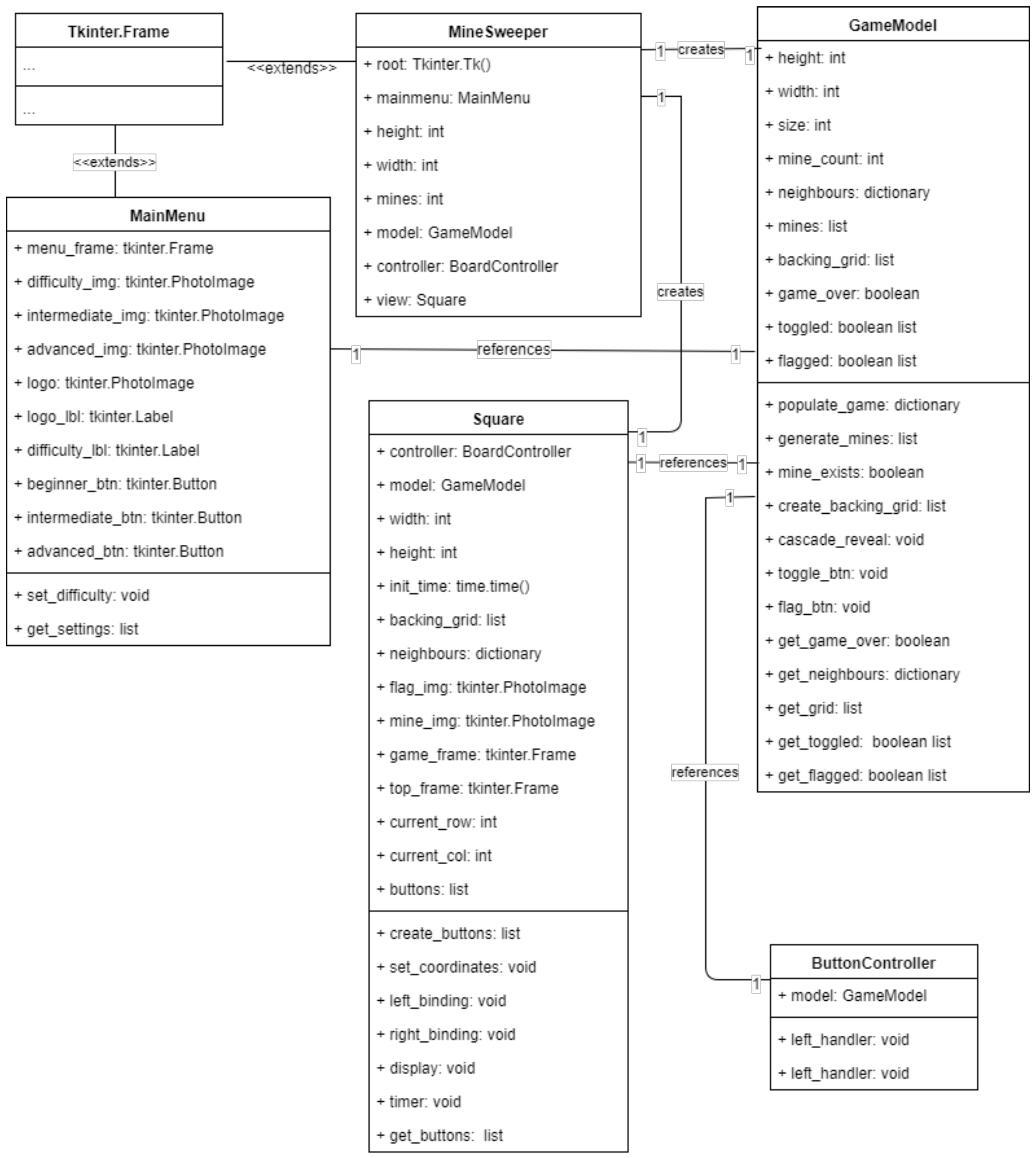

Figure 1: Class diagram for Python implementation of Minesweeper using MVC architecture

```python
import sys
from GameModel import GameModel
from SquareView import Square
from BoardController import BoardController
from MainMenu import MainMenu
from tkinter import *

#Main handles creating GUI, initialises data and calls mainloop
class MineSweeper:
    def __init__(self):

        #create main window
        self.root = Tk()
        self.root.title("Minesweeper")
        self.root.resizable(False, False)
        self.mainmenu = MainMenu(self.root)
        self.root.mainloop()
        self.height = self.mainmenu.height
        self.width = self.mainmenu.width
        self.mines = self.mainmenu.mines
        self.model = GameModel(self.height, self.width, self.mines)
        self.controller = BoardController(self.model)
        self.view = Square(self.root, self.controller, self.model, self.height, self.width)
        self.root.mainloop()

MineSweeper()
```

Figure 2: Main class Minesweeper

The program creates object of each class in order of the model-view controller

6 Design Paradigm

Use a design paradigm to design a simple software system, and explain how system design principles have been applied in this design.

7 Software Architecture

Select an appropriate software architecture as the design basis for a given software requirements specification, justify the selection based on its advantages over alternative architectures.

8 Design Patterns

The model-view controller heavily uses the Observer pattern especially in the Model and View. When the data in the model class is modified, the view class needs to be notified and updated as it is run.

The Board Controller class uses a Facade design pattern. The facade design pattern simplifies the interface to the model, which can be said to be a complex system. It abstracts the implementation and acts like a proxy, shielding the user from the details stored in the model. Instead leaving the controller to manage between the view and controller. This decouples the program that uses the system from the details of the subsystem, increasing efficiency and understanding for modification later on.

9 User Interface

The user interface uses an event driven based design. There is an event and a listener, the listener is notified when an event occurs by button click. The event which is bind to the button using an event-handler. The program binds each click to a button. As the player interacts with the game, the game is played with the event of a button press. The event handler function reacts depending on the hidden backing board and the minesweeper board is modified in one of the four different ways. The first way the board can be modified is by clicking a button with a number, which only reveals its own underlying square. The second way is for the square to be empty which checks adjacent squares if they are also empty, this check is done in the cascade reveal function. Thirdly, the underlying cell could be a mine ultimately disabling the state of the board and ending the game. Lastly, the user can flag the square and not activate the button at all but uses the right mouse click event to bind the flag as an overlaying image.

10 Model-View Controller

Model view controller (MVC) is an architecture that allows different parts of a system to be loosely coupled (or separated). MVC keeps the code well-organised into three distinctly different sections of code. Views display data from the model and take user actions. Models represent the data and domain logic in the system. And controllers liaise between the View and the Model, often controlling flow. MVC have separated functions will solve the most common software problems of today: readability, modularity, and coupling. This helps programmers reuse and change code as they are able to work in smaller subset that may be more or less isolated from the larger piece of code or functionality. This structure also helps when testing the program, especially unit testing when sections of code need to be tested. As similar code is sectioned into groups, there is better coverage of test cases.

11 Good Software Design

A good software design should correctly implement all the requirements and functionalities defined in Software Requirement Specification document. With no formal methods of design from the start, the software can very quickly become very difficult to manage
After the deployment of the software, it should be able to be maintained and easily amenable to change.

12 Different Designs

For software developers to create a variety of different designs for their program, comments illustrate how other developers may extend or enhance the already completed program by explaining the purpose of each function. Comments create the opportunity for developers to reuse code and it is imperative to fully understand a block of code with a single, simple and concise comment. It is the only way to capture the intention of the code at the time of writing, when going back to the program later on, it is more effective to continue and progress if there is are basic explanations documented on the same page. Figure 3 below shows the functions in the GameModel class and the associated comments.

```
# function that populates the board and returns a dictionary of each cell's
    adjacent cells
def populate_game(self):

# function that populates the grid with mines, returns a list of mines
def generate_mines(self):

# function that returns a boolean: True if mine exists in current position, False
    if otherwise
def mine_exists(self, row, col):

# function that creates the backing board for the game, assigns a number or letter
    depending on its value
def create_backing_grid(self):

# called when an empty cell of value 0 is clicked, will reveal all surrounding
    cells if the condition is met
def cascade_reveal(self, i, j):

# mutator function for right click
def toggle_btn(self, i, j):

# mutator function for right click
def flag_btn(self, i, j):

# gets the game_over boolean variable
def get_game_over(self):

#gets the dictionary of neighbours
def get_neighbours(self):

# gets the toggled boolean list of lists, function is used for other classes that
    need to access the list
def get_toggled(self):

# gets the toggled boolean list of lists, function is used for other classes that
    need to access the list
def get_flagged(self):
```

Figure 3: Functions in GameModel class