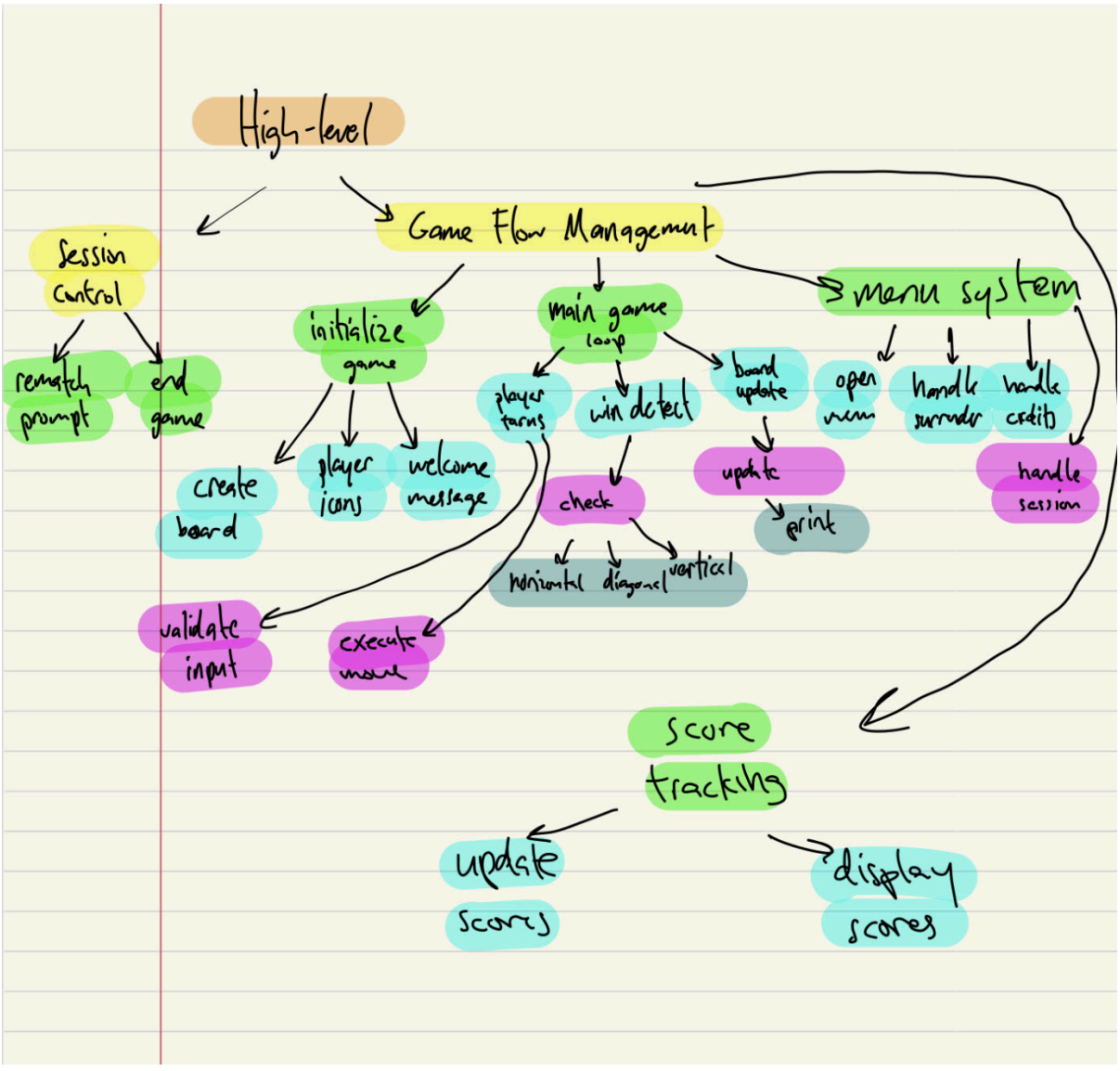
Connect 4 Python Game Plan

Team 3: Andrew Feng, Jacob Jones, Mason Kielinski, Daniel Wisa

Hierarchy Chart:



Choice of Game:

We chose Connect 4 because we felt it was a simple game that could be implemented fairly easily while also allowing us to tack on a bunch of different features (having a .txt log and a main menu) and mess around with it visually (such as the design of the board). We initially worked on Bingo before deciding that it was both too boring as a game (100% RNG) and too messy visually to implement into a terminal.

Connect 4 Python User Manual

Team 3: Andrew Feng, Jacob Jones, Mason Kielinski, Daniel Wisa

Files:

These files will be included in the actual submission as *separate files*, but in order for the program to work properly you need to put them together in the same folder ***OR*** download the .zip file/access this project from the GitHub repository (github.com/abzf227/engr102_connect4), where it should include

- `game_plan.pdf (this file)`
- `fun_game.py`
- `fun_game_log.txt`
- `connect4_sound.mp3`
- `README.md`

Note: `README.md` does not actually contain anything of note and will not be included in submission. This user manual serves as the main reference in place of that.

Prerequisites:

- **Python 3** must be installed. This program is incompatible with Python 2 and versions older.
- Python module **playsound 1.2.2** must be installed. Normal playsound caused issues with the program. This can be done in the console of your IDE by inputting the following:
  `pip uninstall playsound` (if you already have playsound installed)
  `pip install playsound==1.2.2`

Running the Game:

Run the game through `fun_game.py`. If all prerequisites were met, this should open a splash screen that displays:

```
 _____                          _       ___
|     |___ ___ ___ ___ ___| |_    | | |
|   --| . |   |   | -_|  _|  _|   |_  |
|_____|___|_|_|_|_|___|___|_|       |_|
>>>> Connect 4 <<<<
By Team 3 - Andrew Feng, Jacob Jones, Mason Kielinski, Daniel
Wisa
(Title ASCII Art credit to patorjk.com)
```

Inputting anything into the console will prompt the user if they'd like to read the rules. After reading the rules (or not), the game will start by displaying a board and an input prompt for the current player. Also displayed are each player's icon, the current player's icon, the current turn, and the current game.

```
Player 1: ●    ||    Player 2: ◎
Current game: 1
Current turn: 1

 _1__2__3__4__5__6__7_
| .  .  .  .  .  .  . |
| .  .  .  .  .  .  . |
| .  .  .  .  .  .  . |
| .  .  .  .  .  .  . |
| .  .  .  .  .  .  . |
| .  .  .  .  .  .  . |
-----------------------
//                     \\
It's Player 1's ● turn...
Please enter a column from 1-7 (or press 0 for more options):
```

Entering any number from 1-7 will cause a chip to be placed in that column. If 0 is inputted, the main menu will pop up where you can choose to see the rules again, surrender to your opponent, see the credits for the game, end the game session (stop the program from running), or return to the game. Invalid inputs will re-prompt the user.

```
>>>> MAIN MENU <<<<

*Rules [H]
*Surrender [R]
*Credits [C]
*End Game Session [X]
*Close Game Menu [Z]

What would you like to do?
```

Choosing to see the rules or credits will reprompt the main menu after the player is done viewing them. The others will skip directly to the game end splash screen, the game over splash screen, or back to where the player left off, respectively.

If the game isn't surrendered or ended, it continues until a player has won (+1 for player) or a draw is obtained (full board with no win; +0.5 for both players). At this point, the console will bring up the game end splash screen, which displays the board, the player who won, and the current record between the two.

```
  _1__2__3__4__5__6__7_
| .  .  .  .  .  .  . |
| .  .  .  .  .  .  . |
| ●  .  .  .  .  .  . |
| ●  ◎  .  .  .  .  . |
| ●  ◎  .  .  .  .  . |
| ●  ◎  .  .  .  .  . |
-----------------------
//                   \\
Connect 4! Player 1 ● has won on turn 7!
Current Record: ● 1 - 0 ◎
--------------------------------
Rematch? (y/n):
```

If the player chooses to rematch, a new game is started and the game count is updated. Elsewise, the game over splash screen is displayed, which includes the final record between the two, and the program ends function.

```
--------------------------------
Game Over!
Final Record: ● 1 - 0 ◎
Thanks for playing!!
--------------------------------
```

Each move and ending outcome is also updated in `fun_game_log.txt`, which is created at the start and can be accessed after the entire session is ended.
Example output:

```
====GAME 1====
Player 1 placed chip in column 1
Player 2 placed chip in column 5
Player 1 placed chip in column 4
Player 2 placed chip in column 6
Player 1 placed chip in column 2
Player 2 placed chip in column 4
```

```
Player 1 placed chip in column 3
Connect 4! Player 1 has won on turn 7!
Current Record: Player 1 (1) - (0) Player 2
====GAME 2====
Player 1 placed chip in column 3
Player 2 placed chip in column 2
Player 1 placed chip in column 3
Player 2 placed chip in column 2
Player 1 placed chip in column 4
Player 2 has surrendered to Player 1 on turn 6!
Current Record: Player 1 (2) - (0) Player 2
====GAME 3====
Player 1 placed chip in column 3
Player 2 ended the session.
Final Record: Player 1 (2) - (0) Player 2
====END LOG====
```

Checklist:

The program…

- Displays rules
    - At the beginning of the game and can be viewed again at any time
- Displays instructions and other features
    - Through prompts in the game loop and the main menu
- Uses 3 if-elif-else statements (and a lot more if-else) (req. 1)
- Uses 20 loops (req. 1)
- Uses 1 dictionary (req. 1)
- Uses 9 functions with Docstrings (req. 3)
- Uses 1 try-except block (req. 1)
- Uses file I/O for logging
- Incorporates audio into the game (not learned in class)

How the Code Works:

The game was built bottom-up, starting with all the functions and then building the main code from there. The program consists of 9 functions (+1 helper) and 8 variables; all functions have their own Docstring which explains their purpose, parameters, and returns (if any). This section will mostly explain the variables and how the program loops.

The board is a 6-by-7 2D list that is created by calling `create_board()`. The log is opened as `log_text` to write to. Player icons are stored in the dictionary `icons` with keys 1 and 2 for players 1 and 2, respectively. `p1_wins`, `p2_wins`, and `current_game` are counters that store player 1 wins, player 2 wins, and the game count, respectively. The two overarching booleans, `game_running` and `early_end` determine if the game continues or not: `game_running` only matters at the end of each game, whereas `early_end` will cause the game to end midway (in case of ending through the main menu).

The game itself is structured by a while loop nested within a while loop. The outer while loop determines if the game will keep running after each game and is dependent on `game_running`, while the inner (game) loop actually lets the players play and is dependent on the boolean `game_end`, which is initialized inside the outer loop. `current_turn` is also initialized here and is incremented in the game loop.

The game loop begins by calling `create_board()` as `board`. The current player's turn is determined using `(current_turn + 1) % 2 + 1` which will display "Player 1" if `current_turn` is odd (since it starts at 1) and "Player 2" otherwise. After prompting the player for an input for `col`, the input is sent into another while loop that checks if the input is valid using `valid_input(board, col)`; if the invalid input is "0", the main menu is brought up, else it re-prompts another input.

When the main menu is brought up, `menu_action` is created as an instance of `open_menu()`. If "surrender" is returned, `game_end` is set to `True` and the game loop `break`s. If "end session" is returned, `early_end` is set to `True`, `game_running` is set to False, `log_text` is closed, and the game loop `break`s. Otherwise (meaning the player closed the menu), the program re-prompts `col`.

After a valid input is received, the board is updated by assigning `update_board(board, col, icons[(current_turn + 1) % 2 + 1])` and the log is updated. The program then checks the new board for a win with `check_for_win(board)`; if found, the current player is immediately declared the winner and the record is updated while `game_end` is set to True, ending the current game loop. If a winner isn't found, the game then checks for a tie by calling `check_for_tie(board)`. Otherwise, `current_turn` is increased by 1 and the game continues. The log is updated after each move and ending.

Once the game loop has ended, the program runs a check for `early_end`. If it's `False`, the program will prompt the user if they'd like to rematch; if yes, `current_game` increases by 1 and if not, the program ends by closing `log_text` and setting `game_running` to `False`. Otherwise, the program skips over the prompt entirely and ends (due to `game_running` being set to `False` earlier).