Gameplay

- Board
  - Board of 3
  - Ask grid size (9x9 max.), use coordinates (e.g. a2)
  - Ask size to win
  - Smart winning_row size
    - No row of 3 if board > 3 (too easy)
    - No row larger than board
- Save
  - Save games
  - Load message: WELCOME_BACK
  - Delete saved file on load
  - Refactor start into load() or start_new()
- Control / Flow
  - Alternate between starting players ← Dani 🏳
    Currently, is_wrong_player() makes rematches start with the loser.
  - New game: 'n'
  - Restart round: 'r'
  - Make load optional
  - Save more games
  - Rematch
  - Request alternating player input (x, o)
  - Check gameplay flow
- Detect xxx or ooo
- History awareness
  - Mark multiple winning rows if present
    `winning_row` should be a set, not a list: add every winning pass to it.
  - Undo ← Dani 🏳
  - Won in N steps
  - Mark newest edit
    Store last edited coordinates, compare printing of item with that
  - Mark winning row
    Store winning coordinates from `did_player_win()`, pass it to `print_board()`, compare like before

AI

- Program winning strategy
  - https://en.wikipedia.org/wiki/Tic-tac-toe
  - https://en.wikipedia.org/wiki/Tic-tac-toe_variants
  - https://en.wikipedia.org/wiki/M,n,k-game
  - https://xkcd.com/832/
- Program levels of difficulty
  Think novice strategies, use personality (always puts next to yours / always puts somewhere else, defensive / offensive)
- Ask to type 'AI' for AI mode
- Don't print instructions when AI is playing

Error handling

- Break out inline error handling
  - A: Sacrifice some custom text for cleaner code `@handle_input_errors`
  - B: Don't sacrifice custom text, have a dedicated file with different error handling decorators for different functions…
- Make all inputs check for exit commands ('q', 's', etc.)
  - A: `@accept_exit_commands`
  - B: If above not feasible due to function not starting w/ input, but there could be a function to which all inputs are passed…

```
incoming = input()
if not is_exit_command(input):
  proceed with functionality
```

- board_size
- to_win
- coordinates
  - if input < 1 raise IndexError (don't go into negative)
  - IndexError if out of range
  - ValueError if I input number for letter or letter for number
- Error handling should not catch KeyboardInterrupt
  - KeyboardInterrupt exception should be in main

Bugs

- did_player_win() doesn't work with urll shape near edges of board
- Announces tie if player wins with last step
- Handle ties
- Limit row length to board size
- On load, Player 1 starts no matter who was last
- Empty strings allowed as names

- Typing enabled during sleep()

Formatting

- Write nicer print_board() using Unicode characters ← Dani ⚐
- Make cursor stop at end of line on wait
- Frame it: say hi, say goodbye
- XO bold
- Proper prints
- Wait between prints
- Stabilize board position
  Every screen should print same number of rows above board

Testing

- Have it tested by ppl continuously ← Dani ⚐
- Implement automated testing
- List test cases (situations & desired outcomes)

Refactor

- Create README.md
- Clean code
  - Add """comments""" to functions
  - Make state of 'winner' more meaningful
    According to Réka, None is weird, and True/False or similar would better.
  - Re-evaluate function & variable names
    - `did_player_win()` should be `find_winning_row()` - returns `winning_row` if found, returns `None` if haven't.
  - Disallow side effects unless explicit via return
    Functions modify game instance w/o returning (e.g. place_mark())
- Segmentation
  - Group bottom monster into main()
  - Break up main() further into functions
  - Break file into modules
    - Pass dependencies around
    - Clean up duplicate dependencies
  - Organize functions into hierarchical layers
  - Regroup functions by concern (not by hierarchy)
    Modules f1 f2 f3 is better than nothing, but it doesn't help w/ teamwork or separation of concerns
    - game
    - ui - (should separate UI & logic anyway)
    - ai
    - player
    - board
- Design patterns
  - Eliminate horizontal calls
  - Eliminate double recursion: control is with upper function
- OOP
  - Group arguments commonly passed together into collections/classes
    - create `class Game` and pass instance instead
    - Pass used-to-be-global variables to functions
    - Rule: if more than 2 attributes are passed, pass entire class instance
  - Import all constants into all namespaces
  - Make rogue variables game attributes too
  - Transform `f.game_*` functions into methods
    - Functions following this pattern `game = f.game_function(player, game)` should be `game.method(player)`
    - Tried, ran into problems with circular dependencies and insufficient OOP know-how
  - Make `Game.reset_round()`
- Clean up hacks
  - There has to be a cleaner way to tell if game is from load or not
  - Destroy duplication in find_winning_row
- Rearrange prompts into while loops (eliminating recursion)
  Recursion carries the danger of stack overflow
- Improve `find_winning_row()`
  - Don't check full row, stop as soon as 1 place is different than MARK[player]
  - Word "row" means two different things, change 1 of them
  - Wouldn't have to define range for shapes if we used `except IndexError: continue`
  - Comprehension is not comprehensible
- Rename `is_it_a_tie()` to `spaces_left()` and do that… Cleaner.
- Simplify if/else in `get_winning_size`
  Use dictionary to determine how board size corresponds to row size
- Make `game_load` look for file instead of try/excepting (it's misleading)
- Consider storing 0, 1, 2 in board instead of 'X', 'Y', ' '