Advanced Software Engineering (LAB)

Stefano Forti

name.surname@di.unipi.it

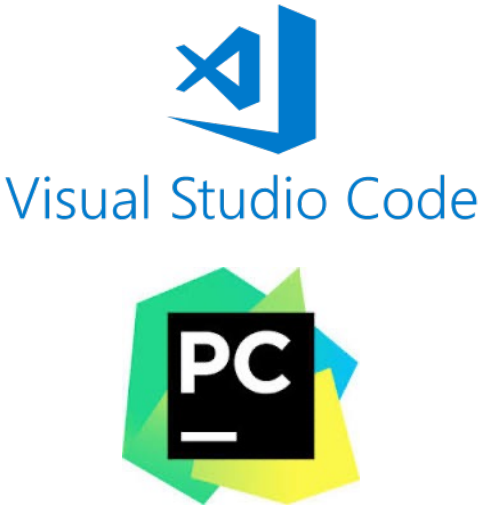
Department of Computer Science, University of Pisa

What will I do?

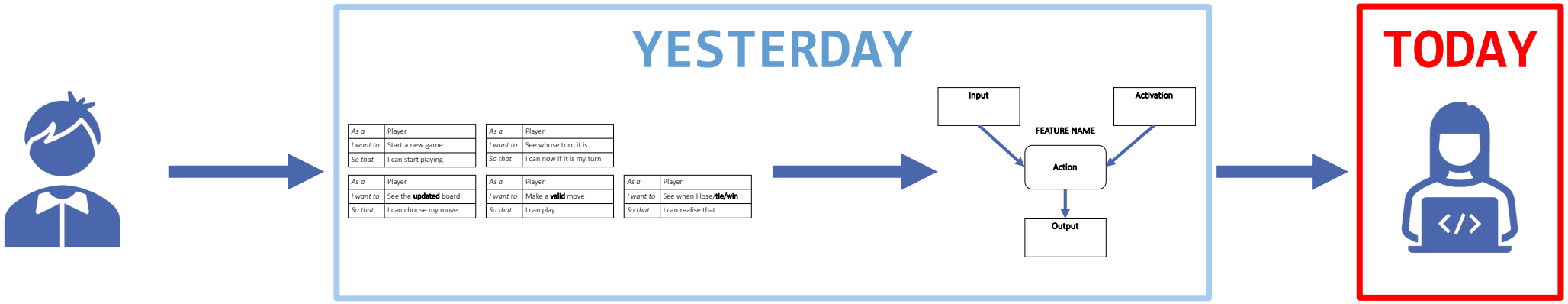
- User story prototyping with Python
- GitHub flow (commits, branches, pull requests, merge)

Checklist

- Install **Python 3.9** [https://www.python.org/]
- Install an **IDE/editor** of your choice
  (e.g., Visual Studio Code [https://code.visualstudio.com/], PyCharm [https://www.jetbrains.com/pycharm/download/])
- Install **Git** [https://git-scm.com/downloads]
- Create your **GitHub** account [http://github.com]



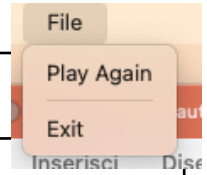
Roadmap



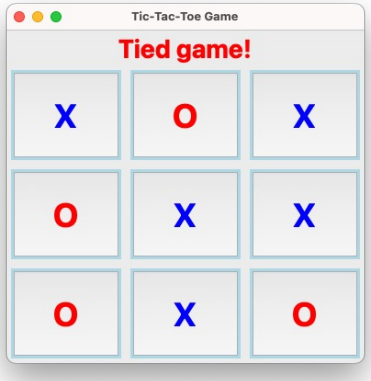
- Download skeleton from Moodle, you will find:
  - a `game.py` file where to implement four missing features
  - a `board.py` file containing the GUI business logic (not to be changed)
  - a `main.py` file to launch the application `python3 main.py`

User stories

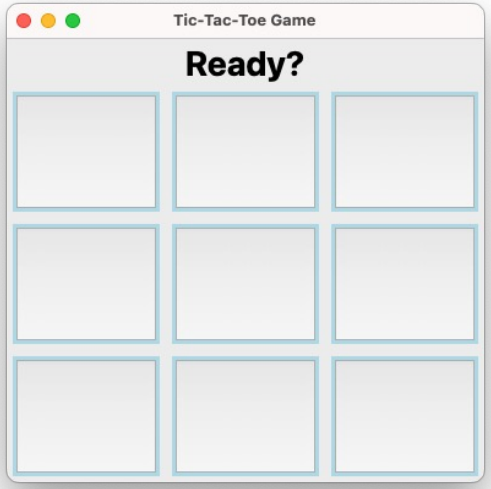
As a	Player
I want to	Start a new game
So that	I can start playing



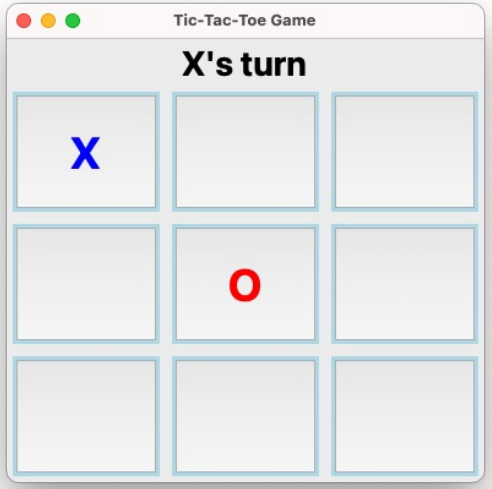
As a	Player
I want to	See whose turn it is
So that	I can now if it is my turn



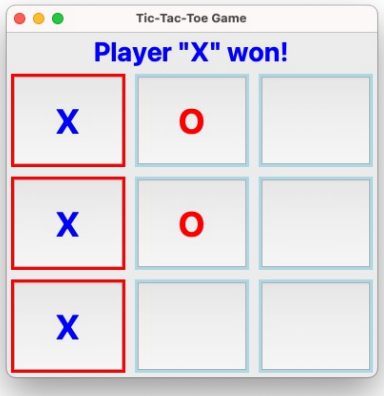
As a	Player
I want to	See the **updated** board
So that	I can choose my move

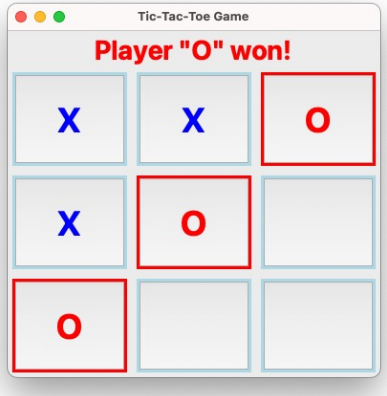


As a	Player
I want to	Make a **valid** move
So that	I can play



As a	Player
I want to	See when I lose/**tie/win**
So that	I can realise that



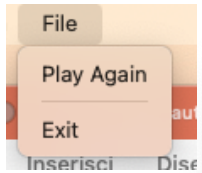


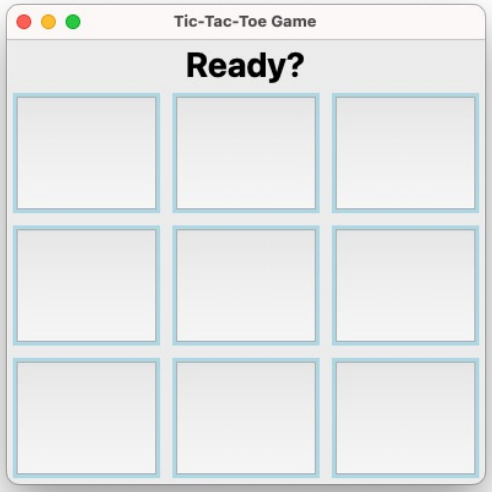
Skeleton walkthrough

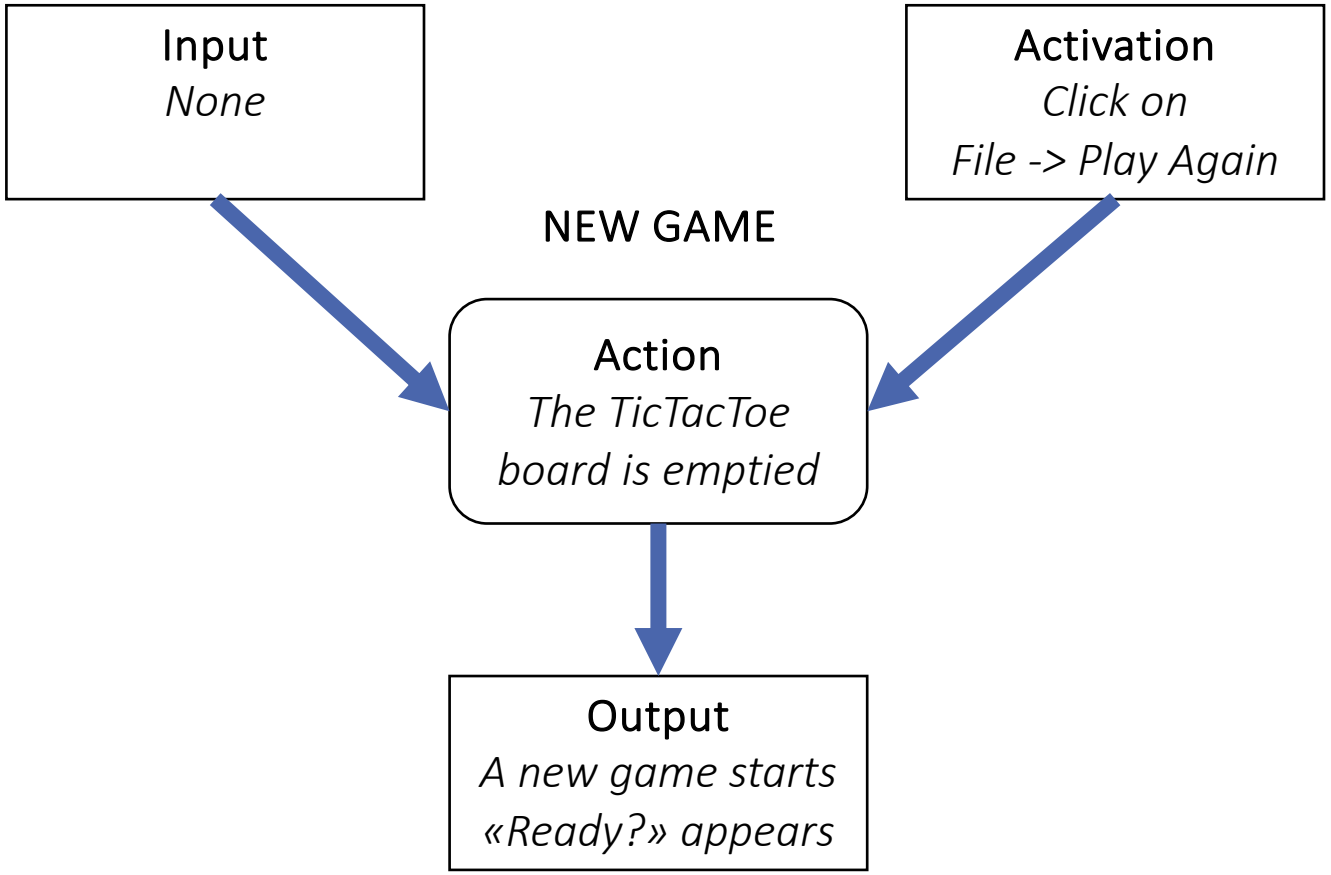
```
4   class Player(NamedTuple):
5       label: str
6       color: str
7
8
9   class Move(NamedTuple):
10      row: int
11      col: int
12      label: str = ""
13
14
15  BOARD_SIZE = 3
16  DEFAULT_PLAYERS = (
17      Player(label="X", color="blue"),
18      Player(label="O", color="red"),
19  )
```

```
22    class Game:
23        def __init__(self, players=DEFAULT_PLAYERS, board_size=BOARD_SIZE):
24            self._players = cycle(players)
25            self.board_size = board_size
26            self.current_player = next(self._players)
27            self.winner_combo = []
28            self._current_moves = []
29            self._has_winner = False
30            self._winning_combos = []
31            self._setup_board()
```

As a	Player
I want to	Start a new game
So that	I can start playing

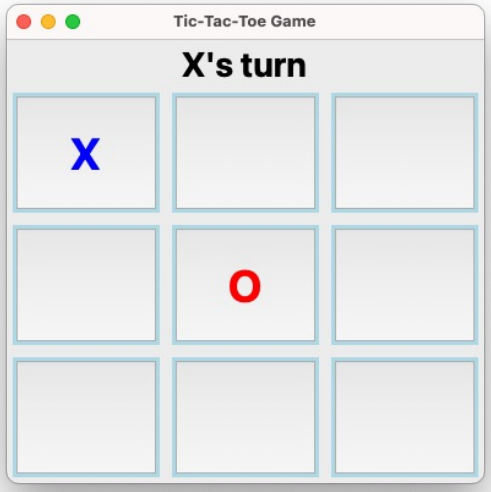


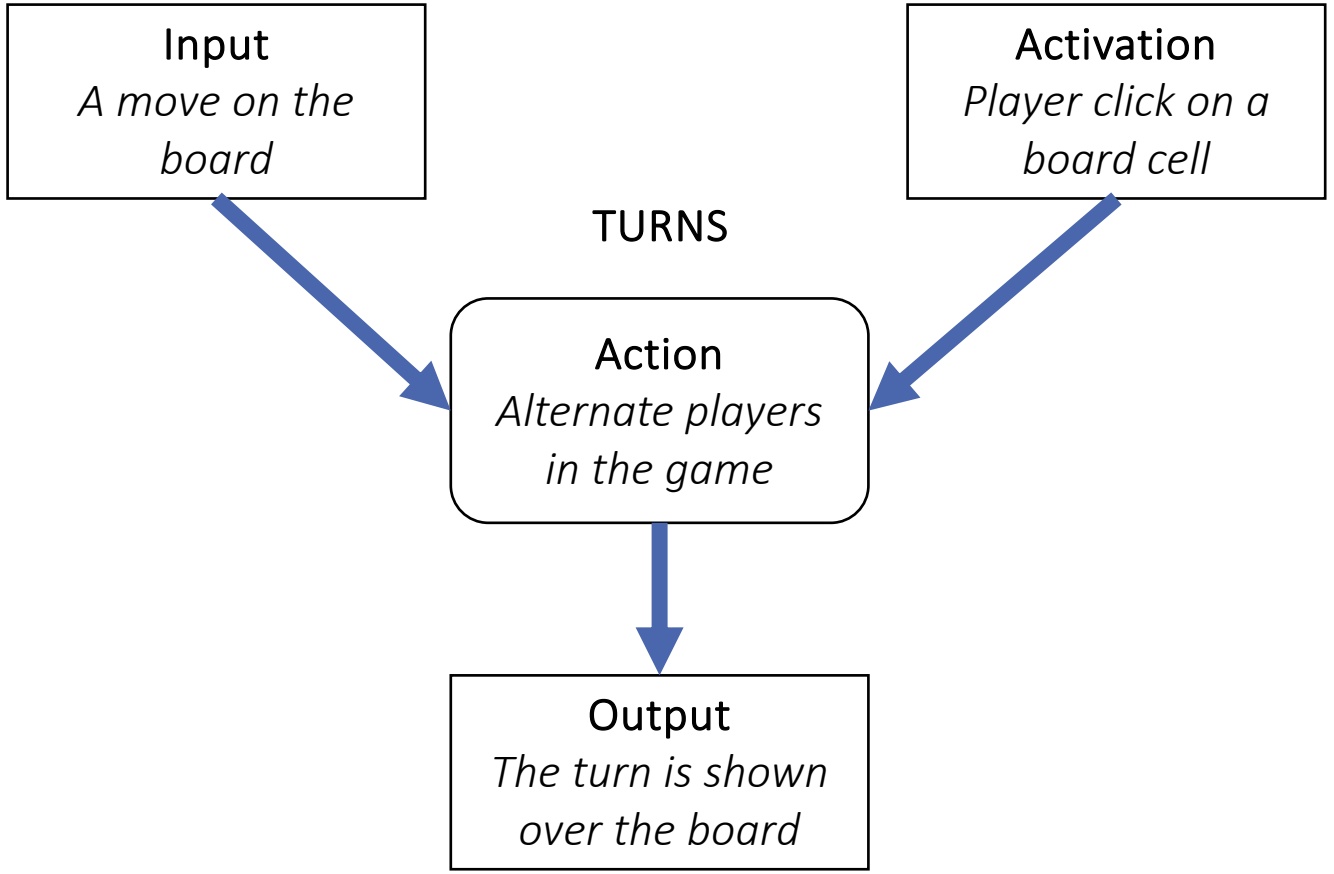





```
85      def reset_game(self):
86          """Reset the game state to play again."""
87          for row, row_content in enumerate(self._current_moves):
88              for col, _ in enumerate(row_content):
89                  row_content[col] = Move(row, col)
90          self._has_winner = False
91          self.winner_combo = []
```

As a	Player
I want to	See whose turn it is
So that	I can now if it is my turn

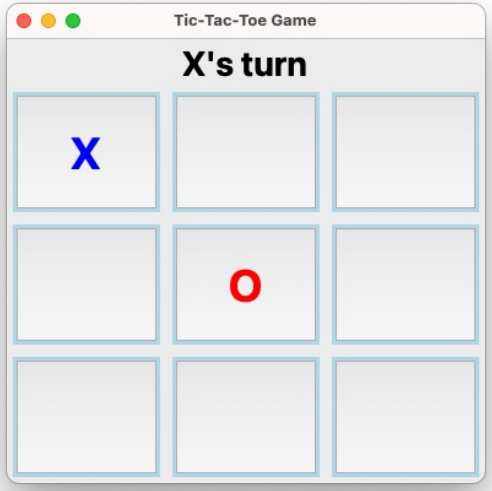


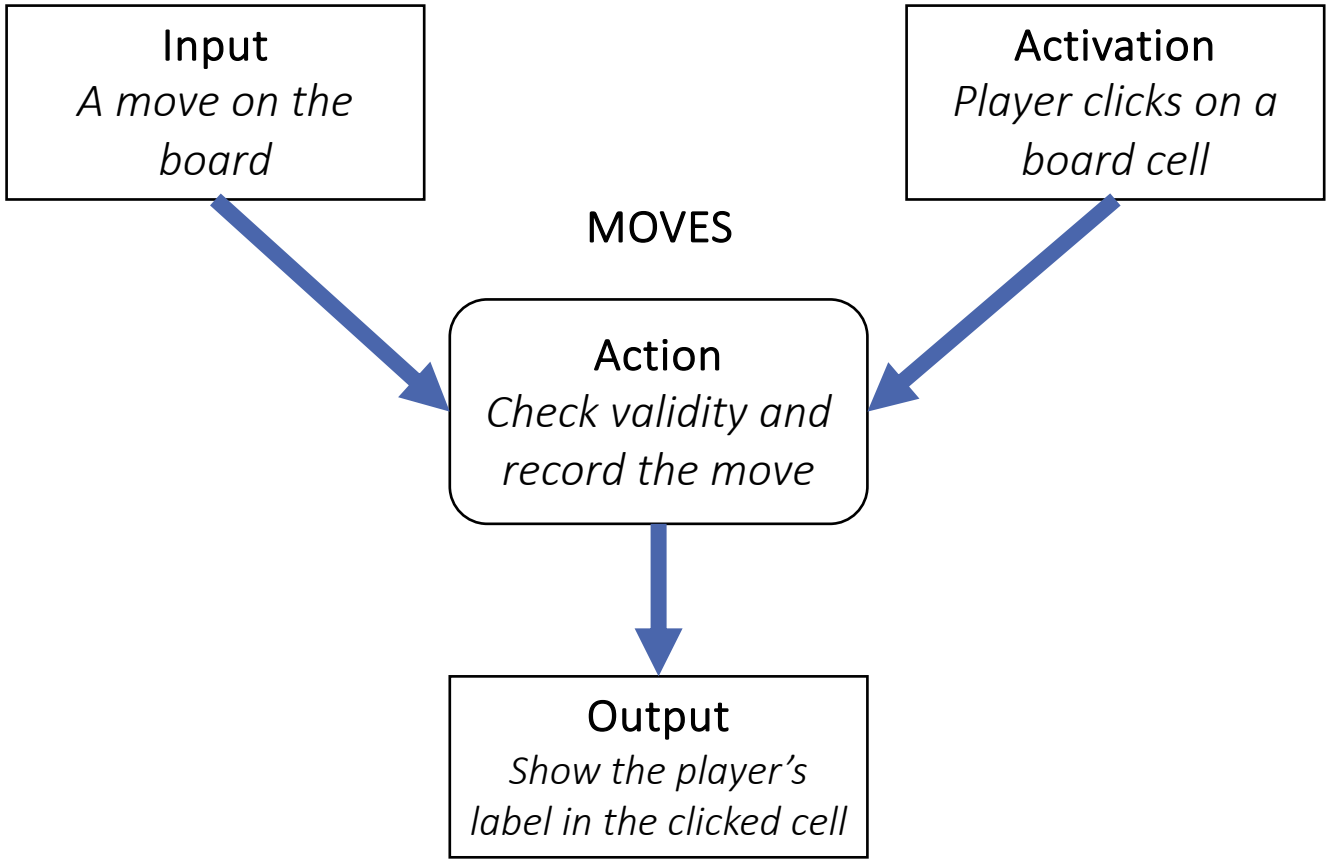



```
80      def toggle_player(self):
81          """Return a toggled player."""
82          # TODO: switches self.current_player to the other player.
83          # Hint: https://docs.python.org/3/library/functions.html#next
```

As a	Player
I want to	See the **updated** board
So that	I can choose my move

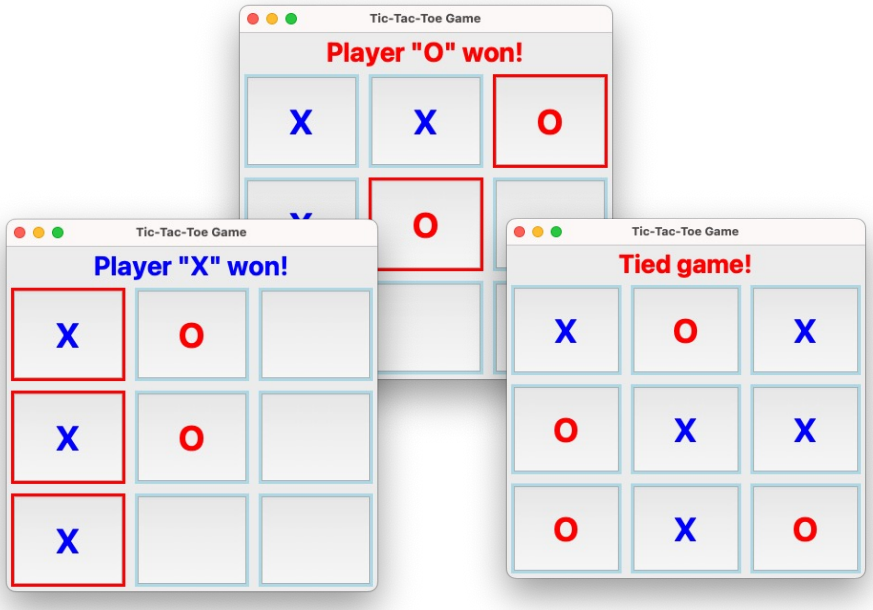
As a	Player
I want to	Make a **valid** move
So that	I can play

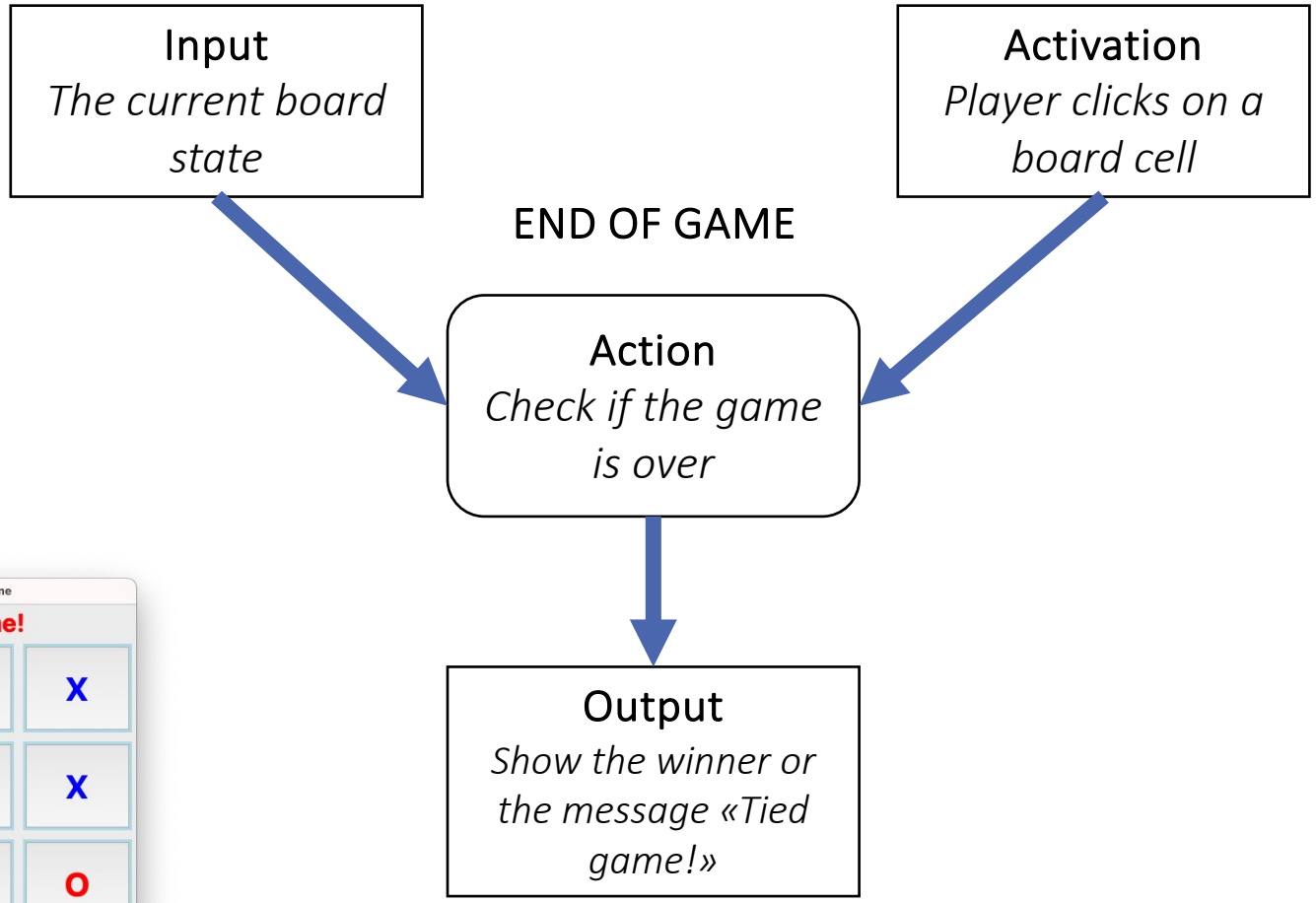





```
51      def is_valid_move(self, move):
52          """Return True if move is valid, and False otherwise."""
53          row, col = move.row, move.col
54          # TODO: check that the current move has not been played already
55          # and that there is no winner yet. Note that non-played cells
56          # contain an empty string (i.e. "").
57          # Use variables no_winner and move_not_played.
58
59          return no_winner and move_not_played
```

As a	Player
I want to	See when I lose/**tie/win**
So that	I can realise that






```
61      def process_move(self, move):
62          """Process the current move and check if it's a win."""
63          row, col = move.row, move.col
64          self._current_moves[row][col] = move
65          # TODO: check whether the current move leads to a winning combo.
66          # Do not return any values but set variables  self._has_winner
67          # and self.winner_combo in case of winning combo.
68          # Hint: you can scan pre-computed winning combos in self._winning_combos
69
70
71      def has_winner(self):
72          """Return True if the game has a winner, and False otherwise."""
73          return self._has_winner
74
75      def is_tied(self):
76          """Return True if the game is tied, and False otherwise."""
77          # TODO: check whether a tie was reached.
78          # There is no winner and all moves have been tried.
```

Create a Repo

- Go to github.com and enter with your credentials.
- Repositories are the place where your projects live.
- In the upper-right corner of any page, click **+** and then **New Repository**.
- Type a short, memorable name, e.g. `ase-fall-23`.
- This repo will be **Public**.
- Initialise it with a **README**.
- Click **Create a repository**.

Clone your Repo

- Open GitBash (or bash).
- Move to the directory where you want to store your work (e.g., **ASE**).
- Use the command **`git clone [your repo url]`**.
- Create a new folder named **`Lab_1`** and move there the **`TicTacToe`** project folder. Then:

```
git add *
git commit -m "first commit"
git push
```

- Add your team mates as repository collaborators

Manage branches

- Create a new branch

  ```
  git branch [name_of_your_new_branch]
  ```

- Move to the branch on your local machine

  ```
  git checkout [name_of_your_new_branch]
  ```

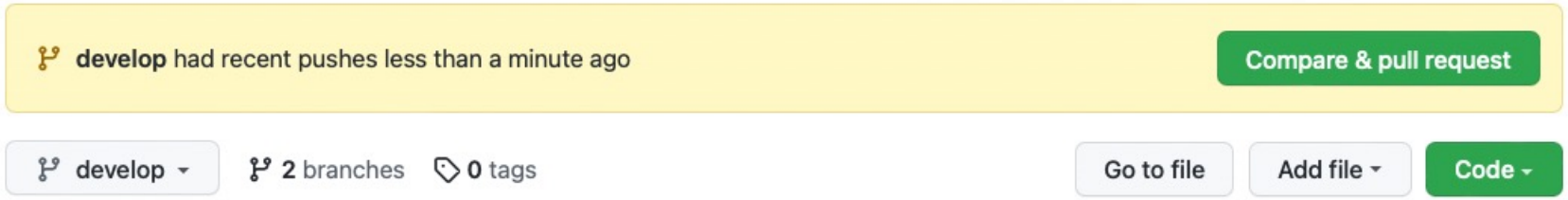
- Push it on GitHub

  ```
  git push origin [name_of_your_new_branch]
  ```

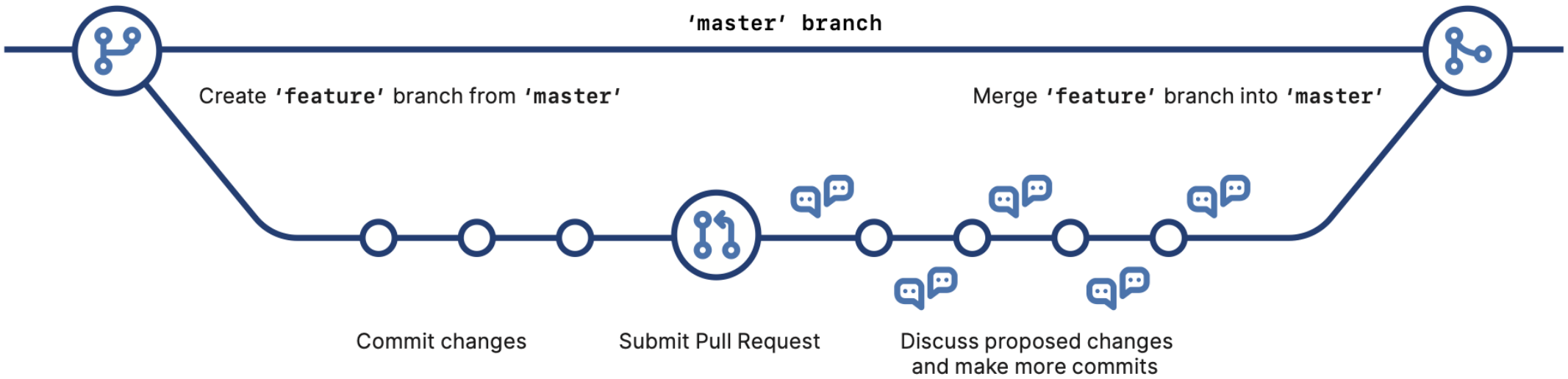
- List all branches with

  ```
  git branch –a
  ```

- Submit pull requests and merge branches from the web interface



GitHub Flow



https://guides.github.com/introduction/flow/

Lab activity

- Split into groups of four.
- Complete all TODOs in the `game.py` file without changing the interface
- You have to implement 4 functions. Split them across team members and implement one each in 4 different branches of the shared repo
- Merge all branches into a single main branch.

- Solution will be posted on Moodle later.