PY RAT

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

PyRat c'est quoi
Bastien Pasdeloup
06/09/2023

OUTLINE

- Introduction
- Présentation du jeu
- Le code
- Les outils de travail
- Programme de l’Episode 1

OUTLINE

- Introduction
- Présentation du jeu
- Le code
- Les outils de travail
- Programme de l’Episode 1

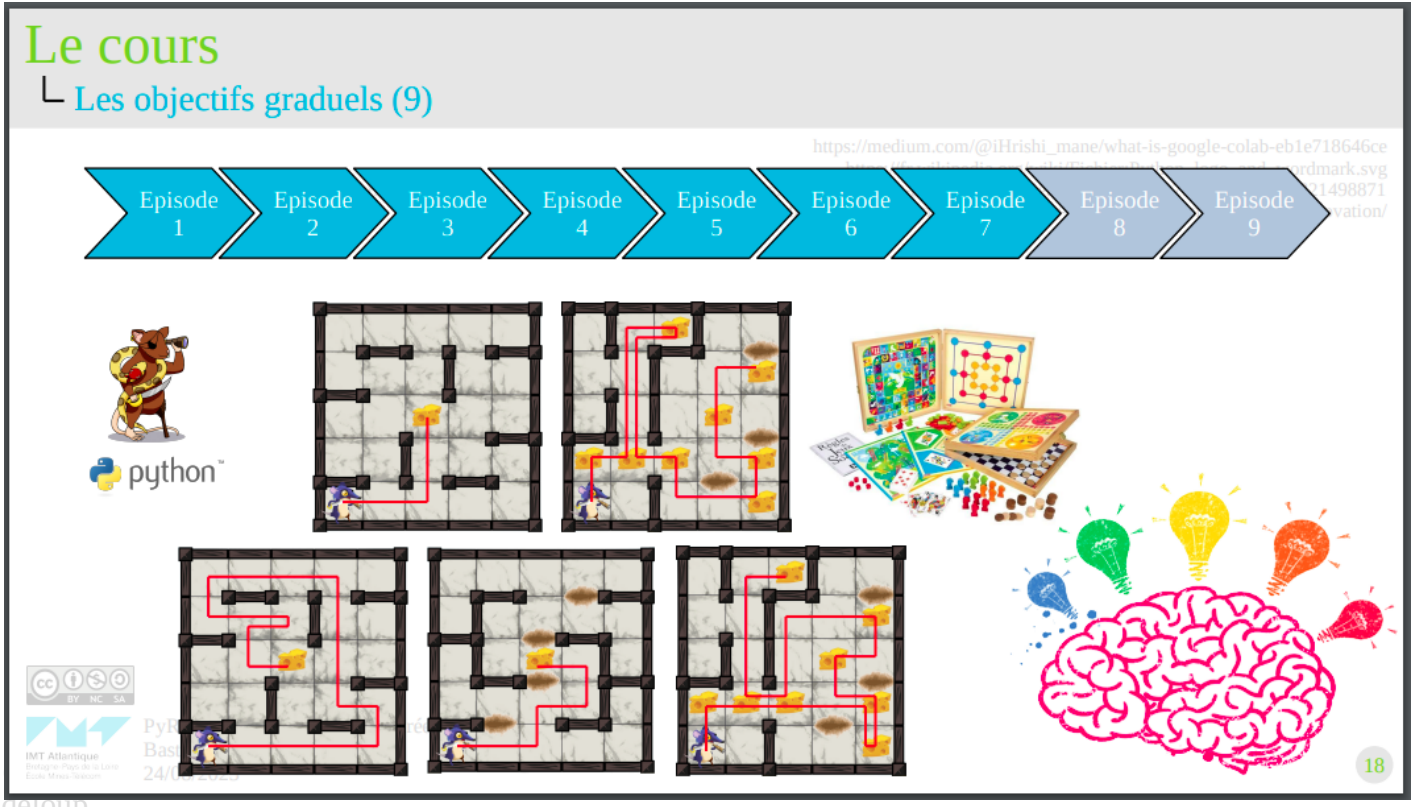
IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

PyRat c'est quoi
Bastien Pasdeloup
06/09/2023

Introduction

Rappel des éléments importants (1)

https://formations.imt-atlantique.fr/pyrat/



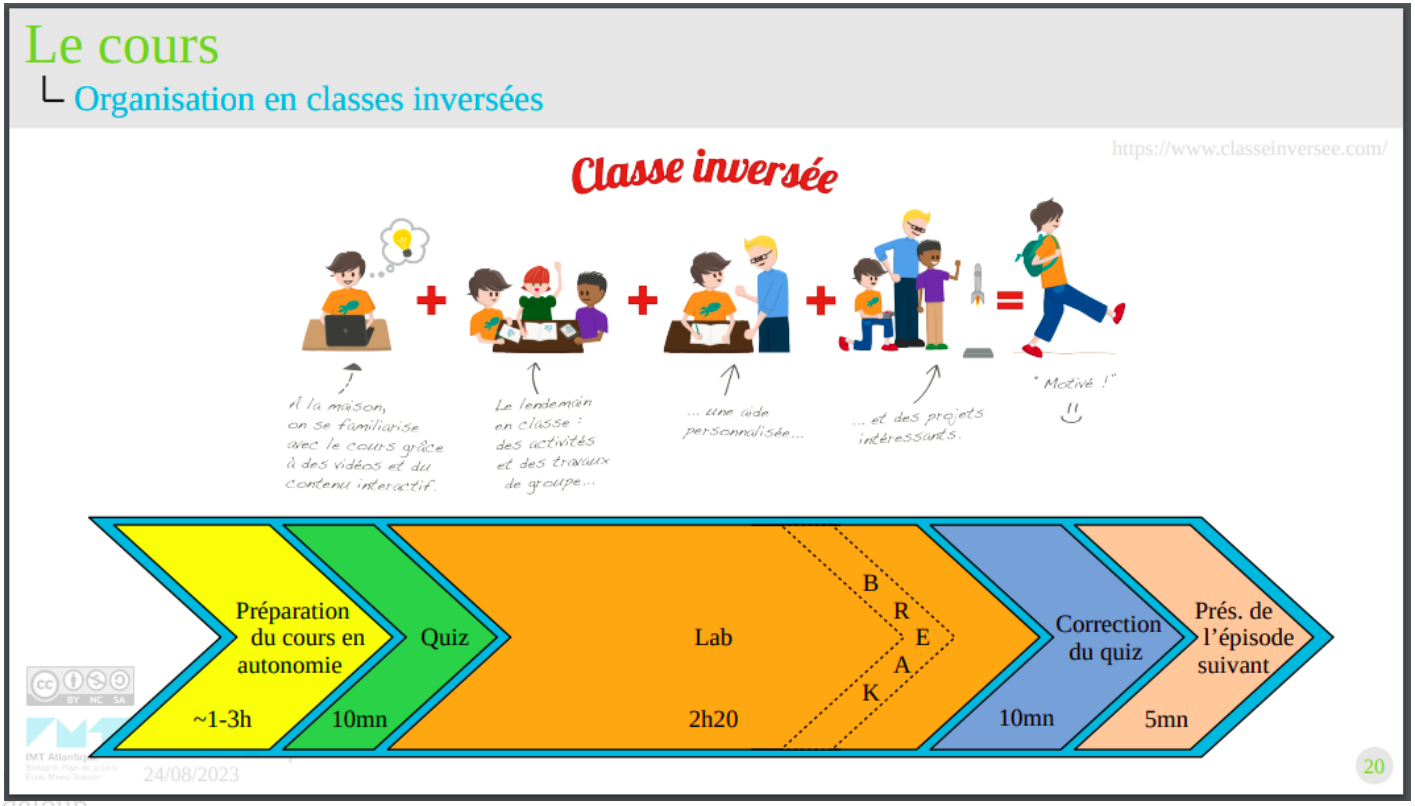
Introduction

Rappel des éléments importants (2)

https://formations.imt-atlantique.fr/pyrat/

Le cours

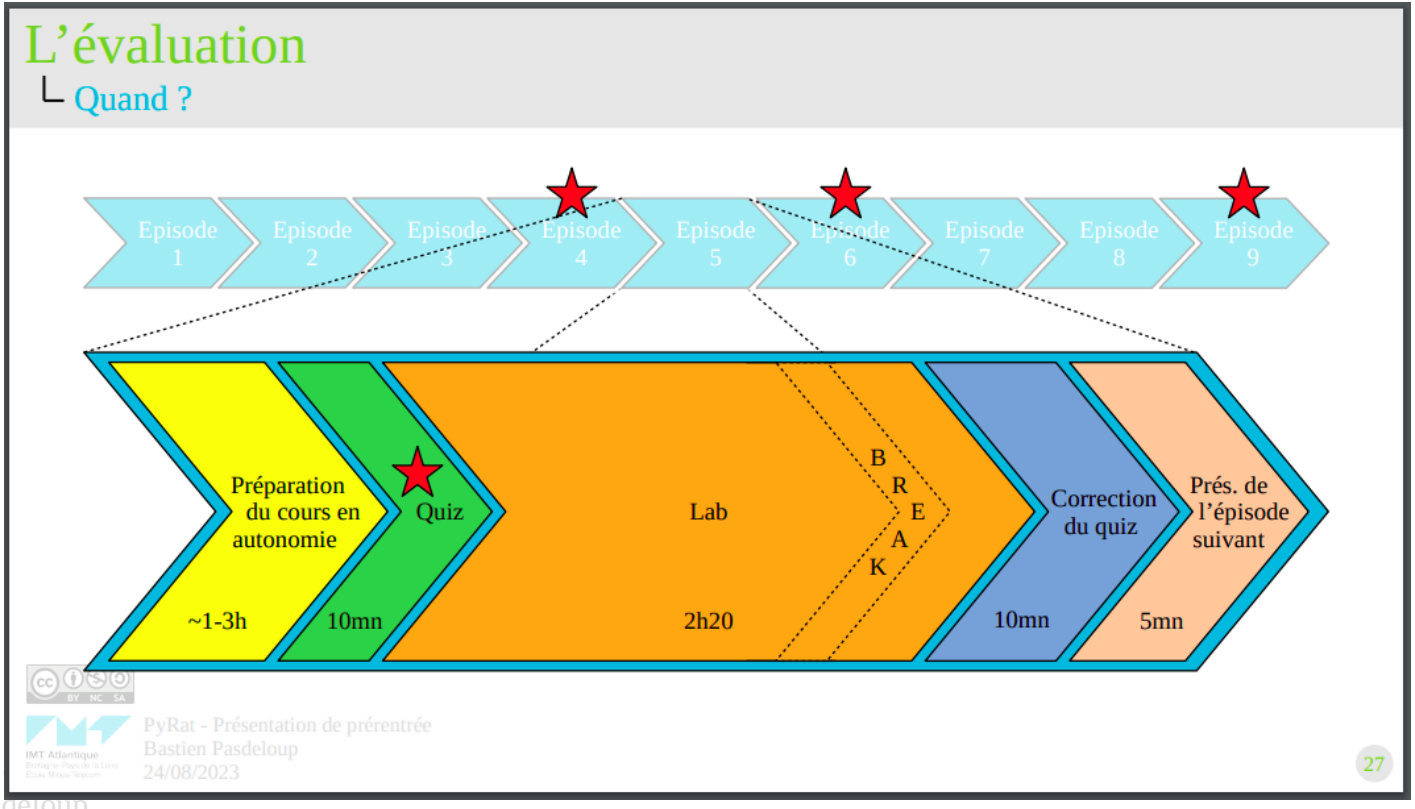
Organisation en classes inversées



Introduction

Rappel des éléments importants (3)

https://formations.imt-atlantique.fr/pyrat/



Introduction

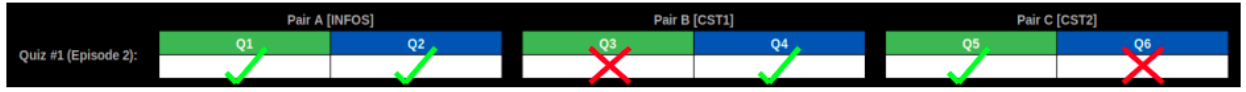
Rappel des éléments importants (4)

https://formations.imt-atlantique.fr/pyrat/

L'évaluation

Le système de rattrapage (4)

Rendu :



Compte comme :

2e chance : toutes ressources autorisées (internet, collègues, etc.)

Au final :

OUTLINE

- Introduction
- Présentation du jeu
- Le code
- Les outils de travail
- Programme de l’Episode 1

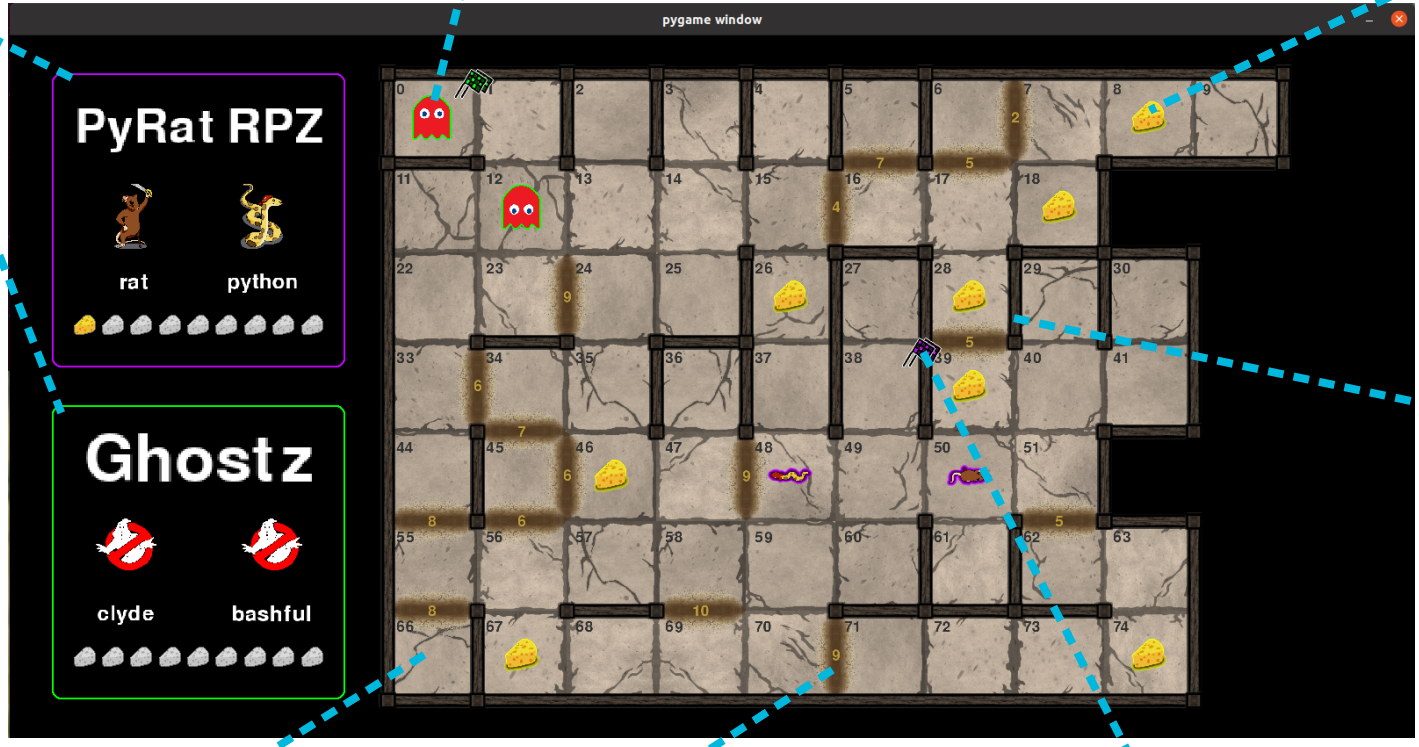
Présentation du jeu

Les éléments du jeu

Infos sur équipes & scores

Joueurs (rat/python/default/custom)

Fromages à ramasser



Murs

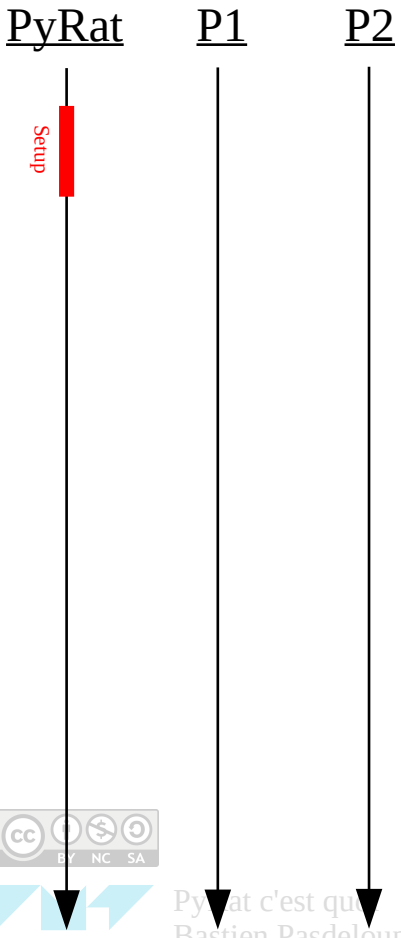
Cases (de 0 à L*H-1)

Boue (>1 mouvement)

Start (random/n°/same/center)

Présentation du jeu

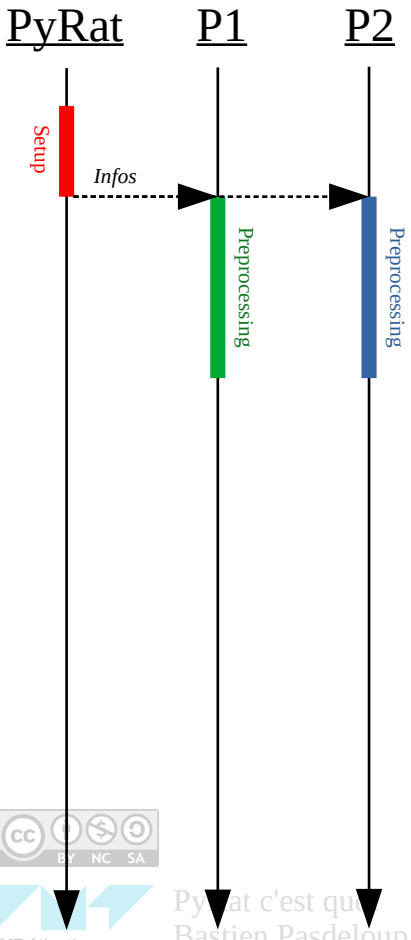
Déroulement temporel d'une partie (1)



(Cas nominal)

Présentation du jeu

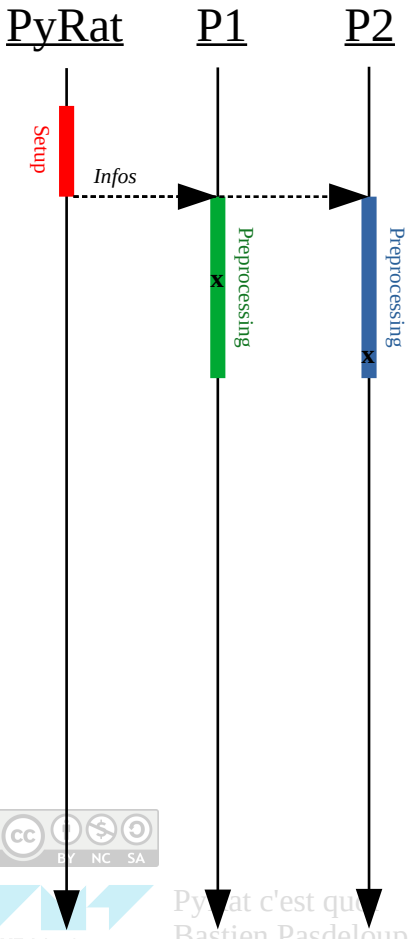
Déroulement temporel d'une partie (2)



(Cas nominal)

Présentation du jeu

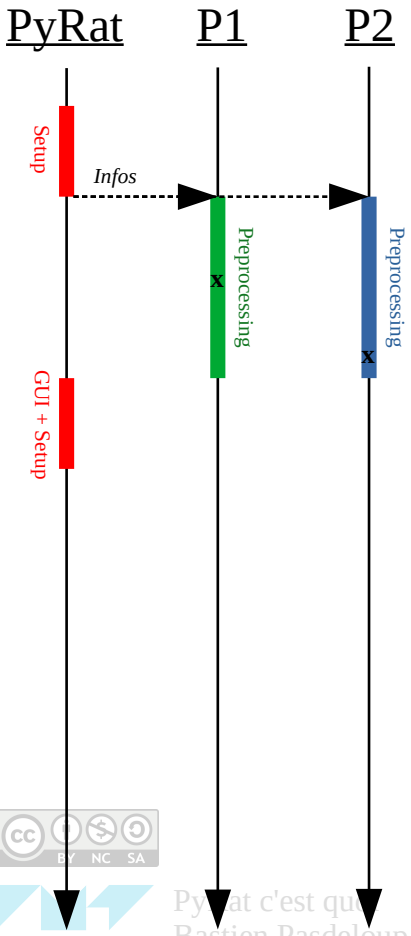
Déroulement temporel d'une partie (3)



(Cas nominal)

Présentation du jeu

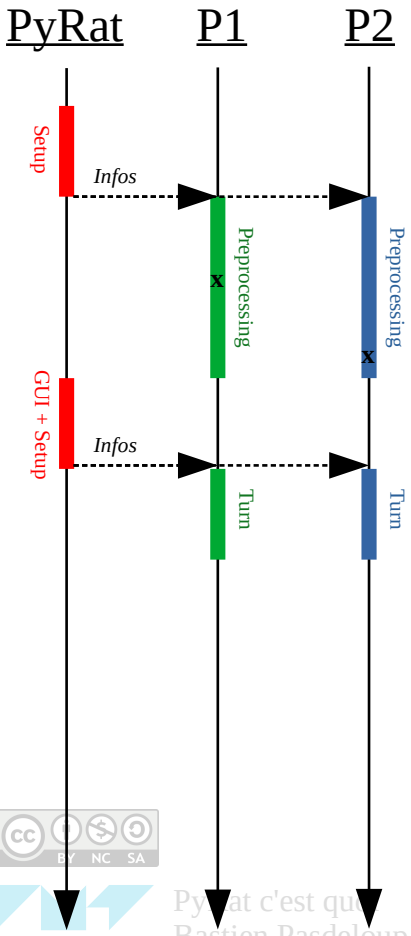
Déroulement temporel d'une partie (4)



(Cas nominal)

Présentation du jeu

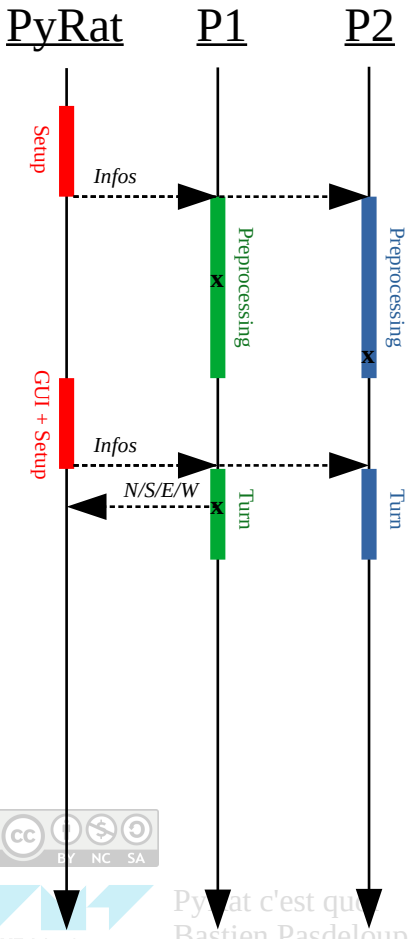
Déroulement temporel d'une partie (5)



(Cas nominal)

Présentation du jeu

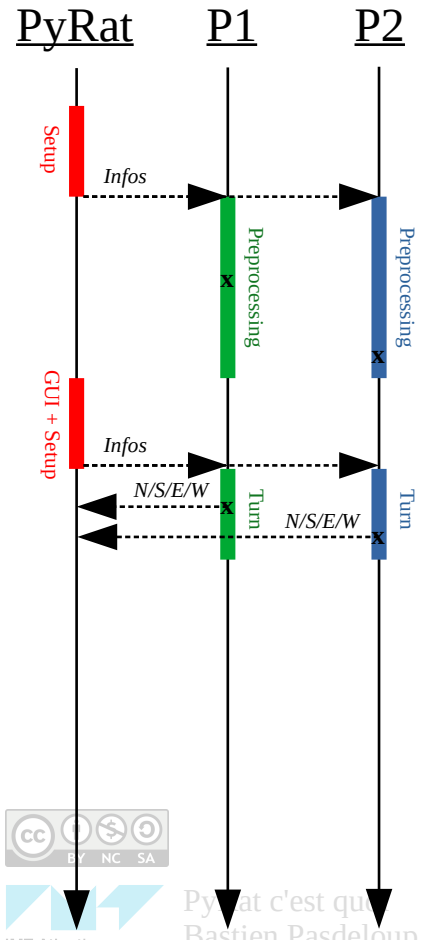
Déroulement temporel d'une partie (6)



(Cas nominal)

Présentation du jeu

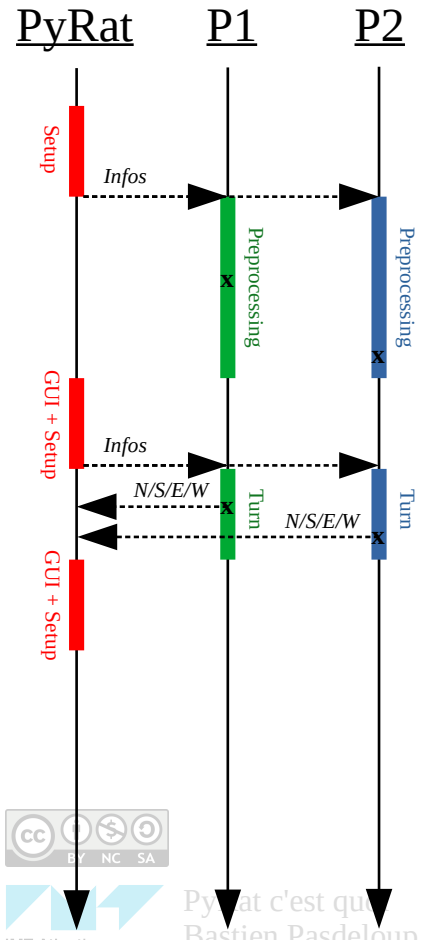
Déroulement temporel d'une partie (7)



(Cas nominal)

Présentation du jeu

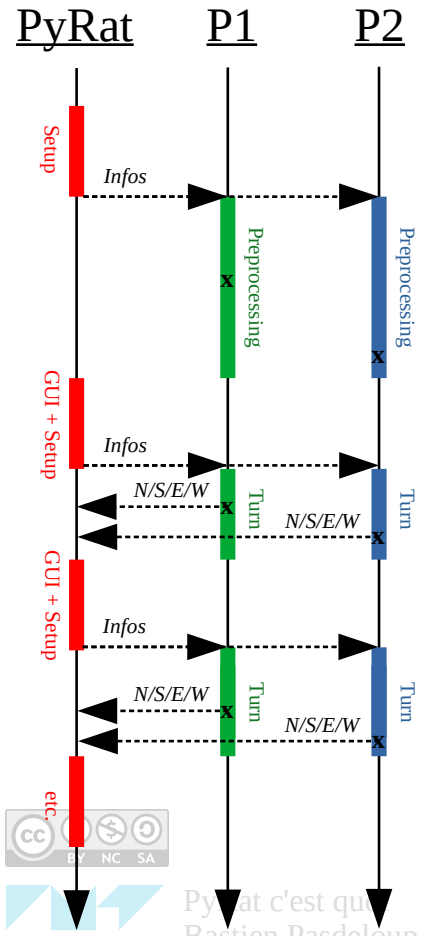
Déroulement temporel d'une partie (8)



(Cas nominal)

Présentation du jeu

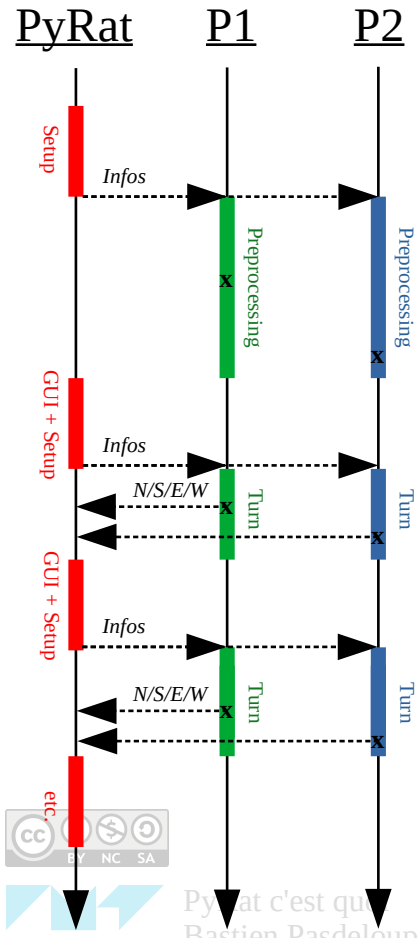
Déroulement temporel d'une partie (9)



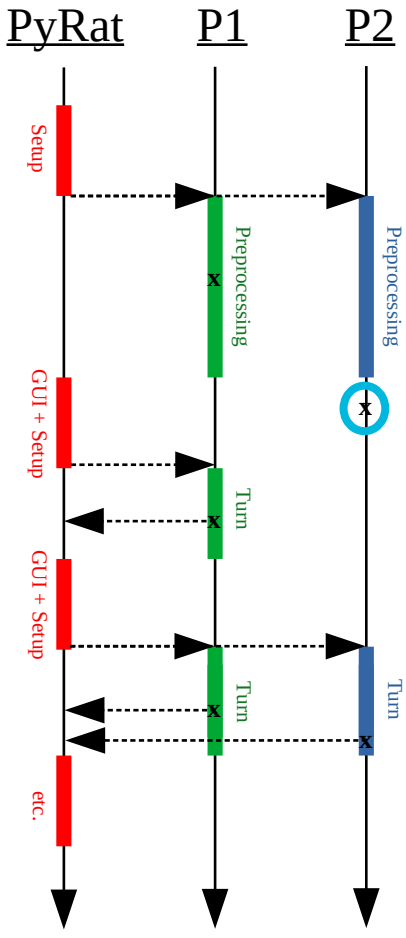
(Cas nominal)

Présentation du jeu

Déroulement temporel d'une partie (10)



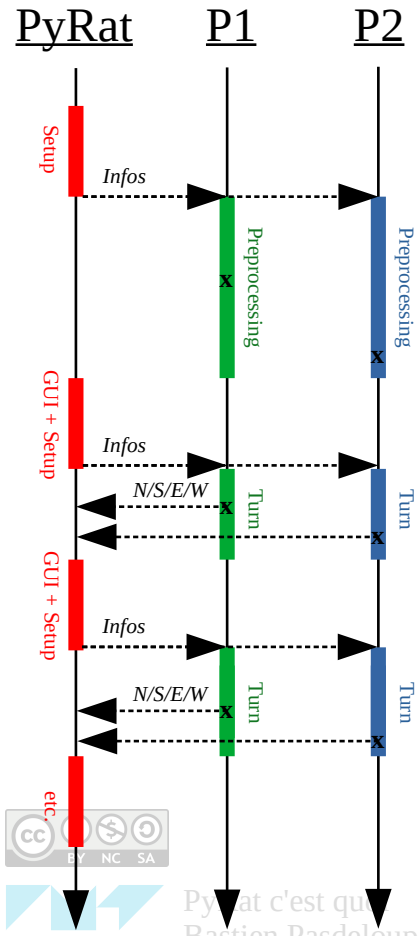
(Cas nominal)



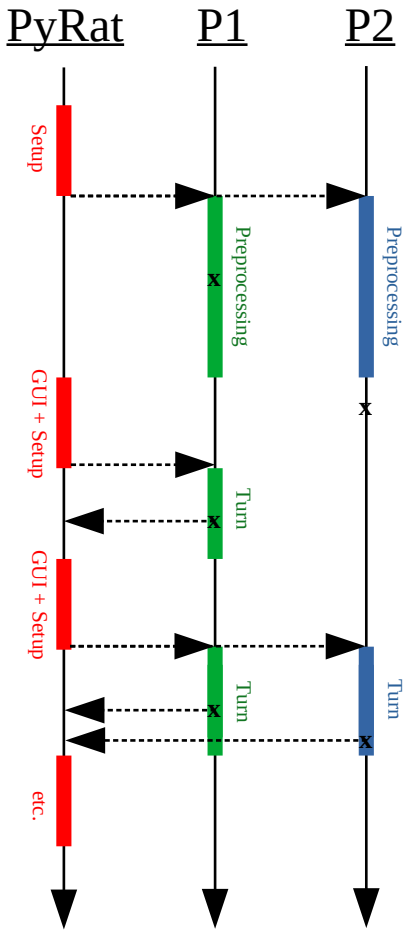
(Cas nominal + Preprocessing trop long)

Présentation du jeu

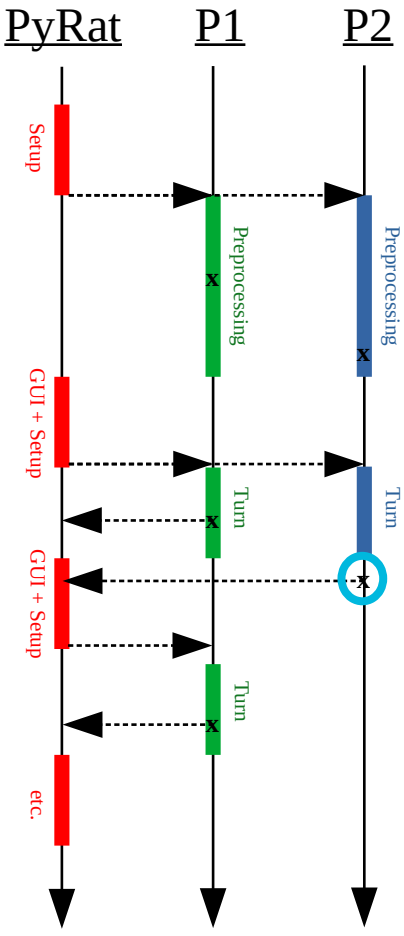
Déroulement temporel d'une partie (11)



(Cas nominal)



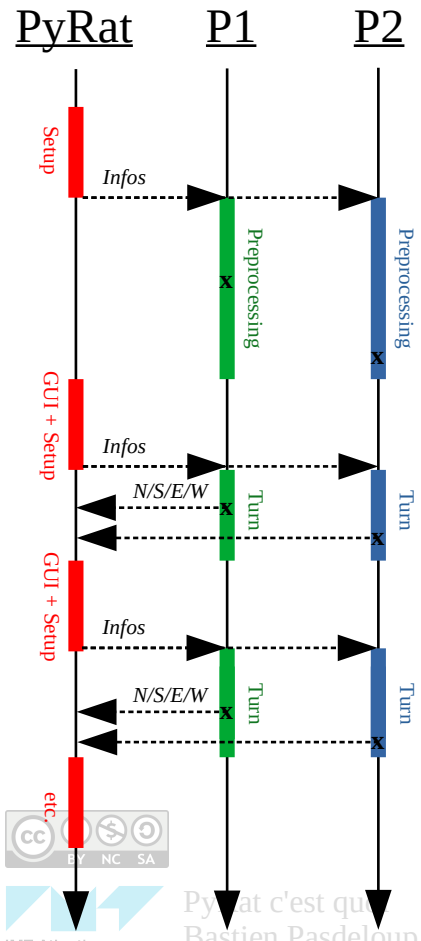
(Cas nominal + Preprocessing trop long)



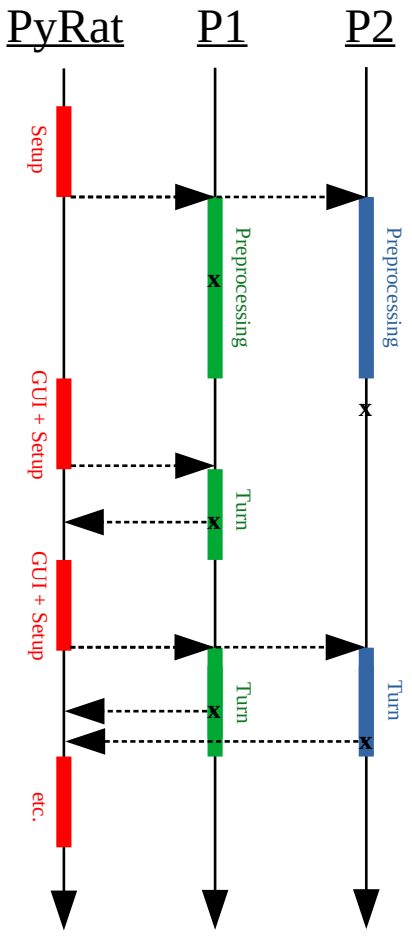
(Cas nominal + Turn trop long)

Présentation du jeu

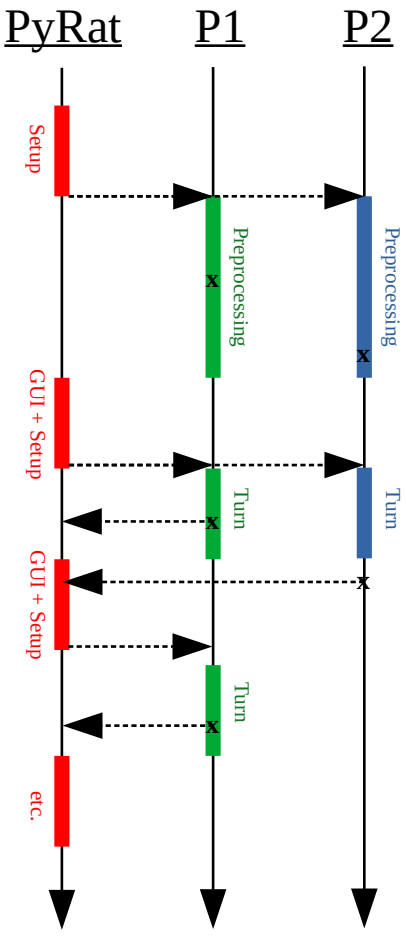
Déroulement temporel d'une partie (12)



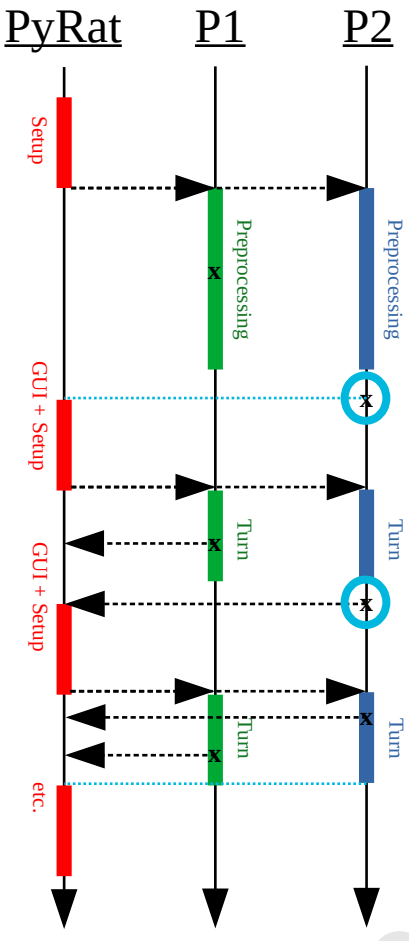
(Cas nominal)



(Cas nominal + Preprocessing trop long)



(Cas nominal + Turn trop long)



(Jeu synchrone)

Présentation du jeu

Les trucs customisables

```
t$ python3 BFS.py --wall_percentage 10
```

```
t$ python3 BFS.py -h
```

```python
if __name__ == "__main__":

    # Map the functions to the character
    players = [{"name": "BFS", "preprocessing_function": preprocessing, "turn_function": turn}]

    # Customize the game elements
    config = {"maze_width": 15,
              "maze_height": 11,
              "mud_percentage": 0.0,
              "nb_cheese": 1,
              "trace_length": 1000}

    # Start the game
    game = PyRat(players, **config)
    stats = game.start()

    # Show statistics
    print(stats)
```

BFS.py

```
"""
    This function is the constructor of the class.
    In:
        * self:                Reference to the current object.
        * players:             List of players to register to the game, given as dictionaries with keys as defined in _register_player.
        * random_seed:         Global random seed for all elements.
        * random_seed_maze:    Random seed for the maze generation.
        * random_seed_cheese:  Random seed for the pieces of cheese distribution.
        * random_seed_players: Random seed for the initial location of players.
        * maze_width:          Width of the maze in number of cells.
        * maze_height:         Height of the maze in number of cells.
        * cell_percentage:     Percentage of cells that can be accessed in the maze, 0%% being a useless maze, and 100%% being a full rectangular maze.
        * wall_percentage:     Percentage of walls in the maze, 0%% being an empty maze, and 100%% being the maximum number of walls that keep the maze connected.
        * mud_percentage:      Percentage of pairs of adjacent cells that are separated by mud in the maze.
        * mud_range:           Interval of turns needed to cross mud.
        * maze_representation: Representation of the maze in memory as given to players.
        * fixed_maze:          Fixed maze in any PyRat accepted representation (takes priority over any maze description and will automatically set maze_height and maze_width).
        * nb_cheese:           Number of pieces of cheese in the maze.
        * fixed_cheese:        Fixed list of cheese (takes priority over random number of cheese).
        * render_mode:         Method to display the game, or no_rendering to play without rendering.
        * render_simplified:   If the maze is rendered, hides some elements that are not essential.
        * trace_length:        Maximum length of the trace to display when players are moving (GUI rendering only).
        * fullscreen:          Renders the game in fullscreen mode (GUI rendering only).
        * save_path:           Path where games are saved.
        * save_game:           Indicates if the game should be saved.
        * preprocessing_time:  Time given to the players before the game starts.
        * turn_time:           Time after which players will miss a turn.
        * synchronous:         If set, waits for all players to return an action before moving, even if turn_time is exceeded.
        * continue_on_error:   If a player crashes, continues the game anyway.

    Out:
        * self: Reference to the current object.
"""
```

__init__.py

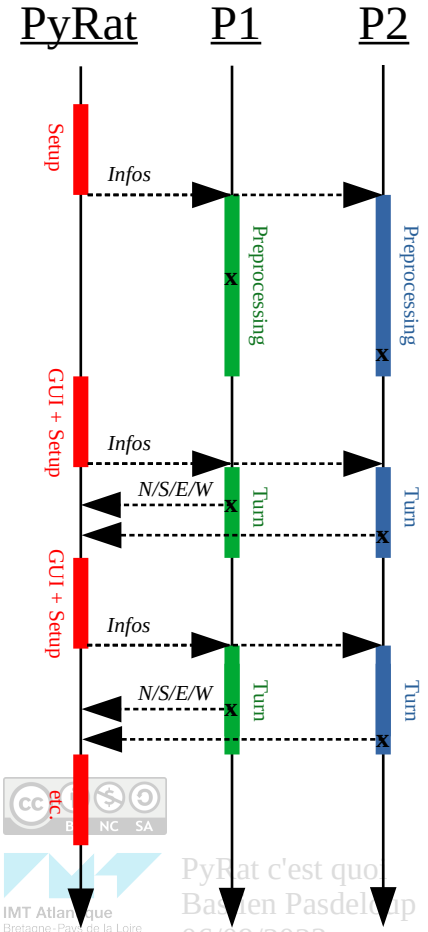
OUTLINE

- Introduction
- Présentation du jeu
- Le code
- Les outils de travail
- Programme de l’Episode 1

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Le code

Preprocessing

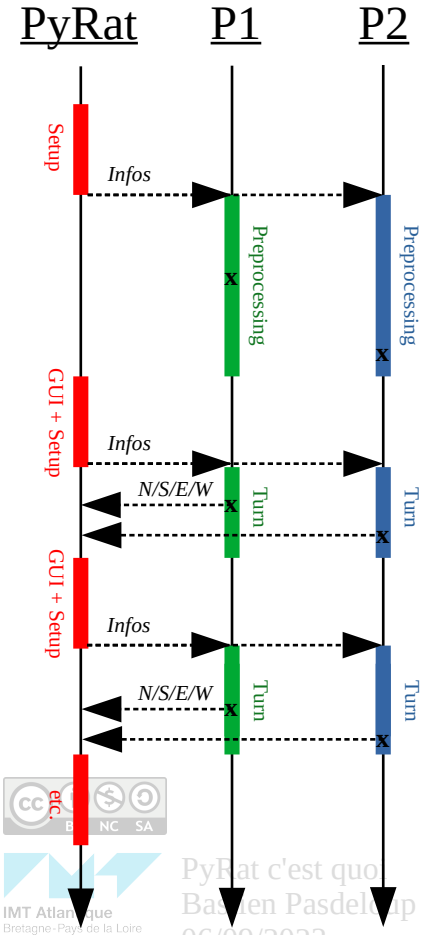



```python
def preprocessing ( maze:             Union[numpy.ndarray, Dict[int, Dict[int, int]]],
                    maze_width:       int,
                    maze_height:      int,
                    name:             str,
                    teams:            Dict[str, List[str]],
                    player_locations: Dict[str, int],
                    cheese:           List[int],
                    possible_actions: List[str],
                    memory:           threading.local
                  ) ->                None:

    """
        This function is called once at the beginning of the game.
        It is typically given more time than the turn function, to perform complex computations.
        Store the results of these computations in the provided memory to reuse them later during turns.
        To do so, you can crete entries in the memory dictionary as memory.my_key = my_value.
        In:
            * maze:             Map of the maze, as data type described by PyRat's "maze_representation" option.
            * maze_width:       Width of the maze in number of cells.
            * maze_height:      Height of the maze in number of cells.
            * name:             Name of the player controlled by this function.
            * teams:            Recap of the teams of players.
            * player_locations: Locations for all players in the game.
            * cheese:           List of available pieces of cheese in the maze.
            * possible_actions: List of possible actions.
            * memory:           Local memory to share information between preprocessing, turn and postprocessing.
        Out:
            * None.
    """
```

Le code

Turn




```python
def turn ( maze:             Union[numpy.ndarray, Dict[int, Dict[int, int]]],
           maze_width:       int,
           maze_height:      int,
           name:             str,
           teams:            Dict[str, List[str]],
           player_locations: Dict[str, int],
           player_scores:    Dict[str, float],
           player_muds:      Dict[str, Dict[str, Union[None, int]]],
           cheese:           List[int],
           possible_actions: List[str],
           memory:           threading.local
         ) ->                str:

    """
        This function is called at every turn of the game and should return an action within the set of possible actions.
        You can access the memory you stored during the preprocessing function by doing memory.my_key.
        You can also update the existing memory with new information, or create new entries as memory.my_key = my_value.
        In:
            * maze:             Map of the maze, as data type described by PyRat's "maze_representation" option.
            * maze_width:       Width of the maze in number of cells.
            * maze_height:      Height of the maze in number of cells.
            * name:             Name of the player controlled by this function.
            * teams:            Recap of the teams of players.
            * player_locations: Locations for all players in the game.
            * player_scores:    Scores for all players in the game.
            * player_muds:      Indicates which player is currently crossing mud.
            * cheese:           List of available pieces of cheese in the maze.
            * possible_actions: List of possible actions.
            * memory:           Local memory to share information between preprocessing, turn and postprocessing.
        Out:
            * action: One of the possible actions, as given in possible_actions.
    """
```

Le code

Main

```python
if __name__ == "__main__":

    # Map the functions to the character
    players = [{"name": "BFS", "preprocessing_function": preprocessing, "turn_function": turn}]

    # Customize the game elements
    config = {"maze_width": 15,
              "maze_height": 11,
              "mud_percentage": 0.0,
              "nb_cheese": 1,
              "trace_length": 1000}

    # Start the game
    game = PyRat(players, **config)
    stats = game.start()

    # Show statistics
    print(stats)
```

Le code

Les statistiques de fin de partie

```python
if __name__ == "__main__":

    # Map the functions to the character
    players = [{"name": "BFS", "preprocessing_function": preprocessing, "turn_function": turn}]

    # Customize the game elements
    config = {"maze_width": 15,
              "maze_height": 11,
              "mud_percentage": 0.0,
              "nb_cheese": 1,
              "trace_length": 1000}

    # Start the game
    game = PyRat(players, **config)
    stats = game.start()

    # Show statistics
    print(stats)
```


{'players':
  {'BFS':
    {'moves': {'mud': 4, 'error': 0, 'miss': 0, 'nothing': 0, 'north': 3, 'east': 0, 'south': 2, 'west': 6, 'wall': 0},
     'score': 1,
     'turn_durations': [2.935300000000019e-05, 1.4319999999999958e-05, 1.7828999999999588e-05, 1.4726000000001363e-05,
                        1.4471000000000275e-05, 1.3036000000000991e-05, 1.4217000000000396e-05, 1.3265999999999625e-05,
                        1.4694000000002594e-05, 1.4458000000001914e-05, 1.4860000000001955e-05],
     'preprocessing_duration': 0.0010728270000000002}},
 'turns': 15}

OUTLINE

- Introduction
- Présentation du jeu
- Le code
- Les outils de travail
- Programme de l’Episode 1

Les outils de travail

Quizzes sur Moodle

Question **3**

Pas encore répondu

Noté sur 1,00

Marquer la question

Modifier la question

- **Category:** Representing graphs in memory
- **Skill:** CST1 – Analyze, reformulate, structure
- **Question difficulty:** Green
- **Paired with:** Q4

What structures have been introduced to allow storing graphs in computer memory?

- [ ] Lists.
- [ ] Dictionaries.
- [ ] Trees.
- [ ] Arrays.

Question **4**

Pas encore répondu

Noté sur 1,00

Marquer la question

Modifier la question

- **Category:** Representing graphs in memory
- **Skill:** CST1 – Analyze, reformulate, structure
- **Question difficulty:** Blue
- **Paired with:** Q3

When considering a graph represented as a list of list, what solution allows extending to weighted graphs?

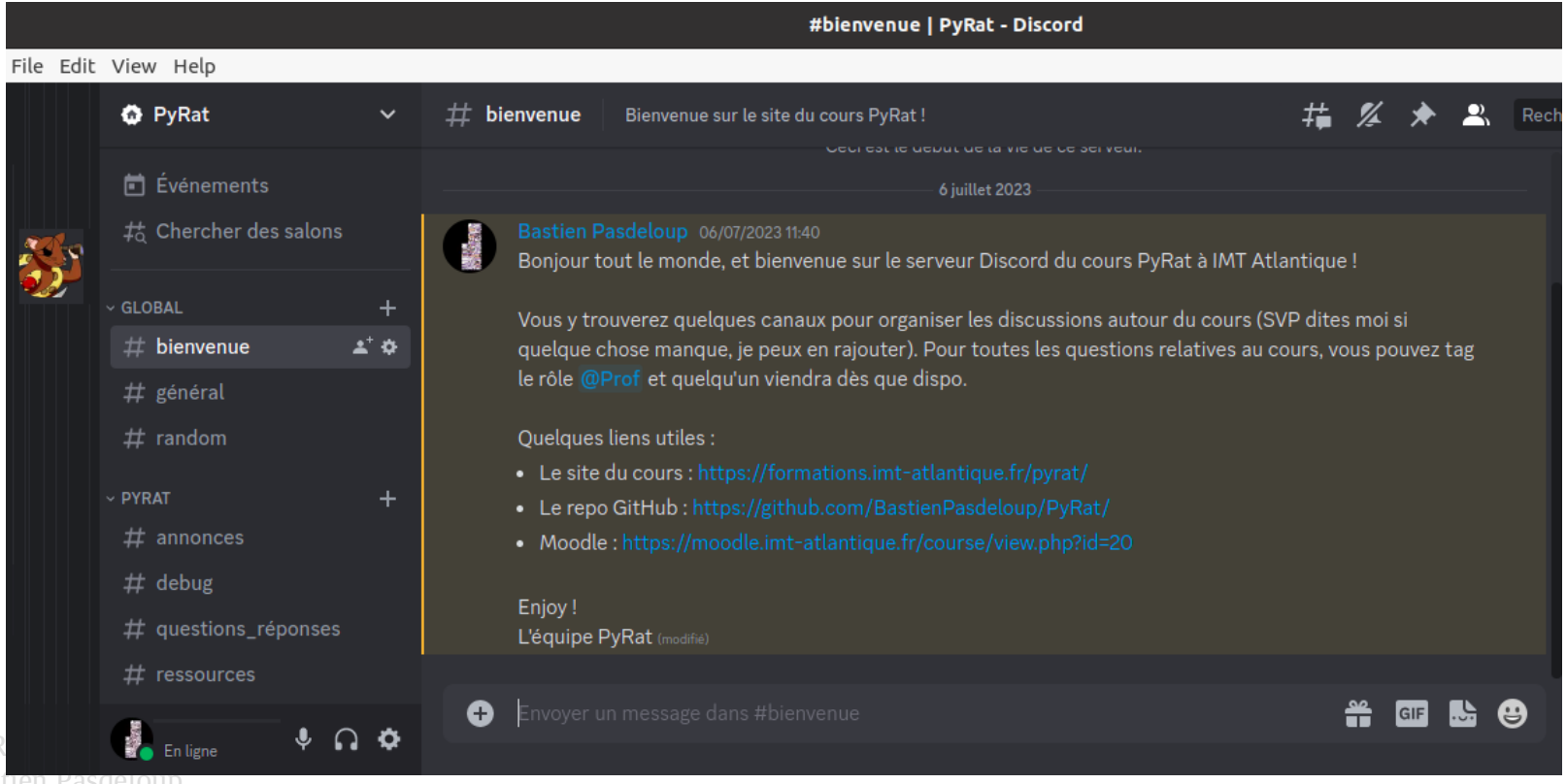
- [ ] Replacing
- [ ] Replacing
- [ ] Replacing
- [ ] This is not

Quiz #1 (Episode 2):	Pair A [INFOS]		Pair B [CST1]		Pair C [CST2]	
	Q1	Q2	Q3	Q4	Q5	Q6

Les outils de travail

Serveur Discord

Facultatif Permet d'avoir du suivi entre les cours https://discord.gg/Q8DzbVfZ



Les outils de travail

Pour coder

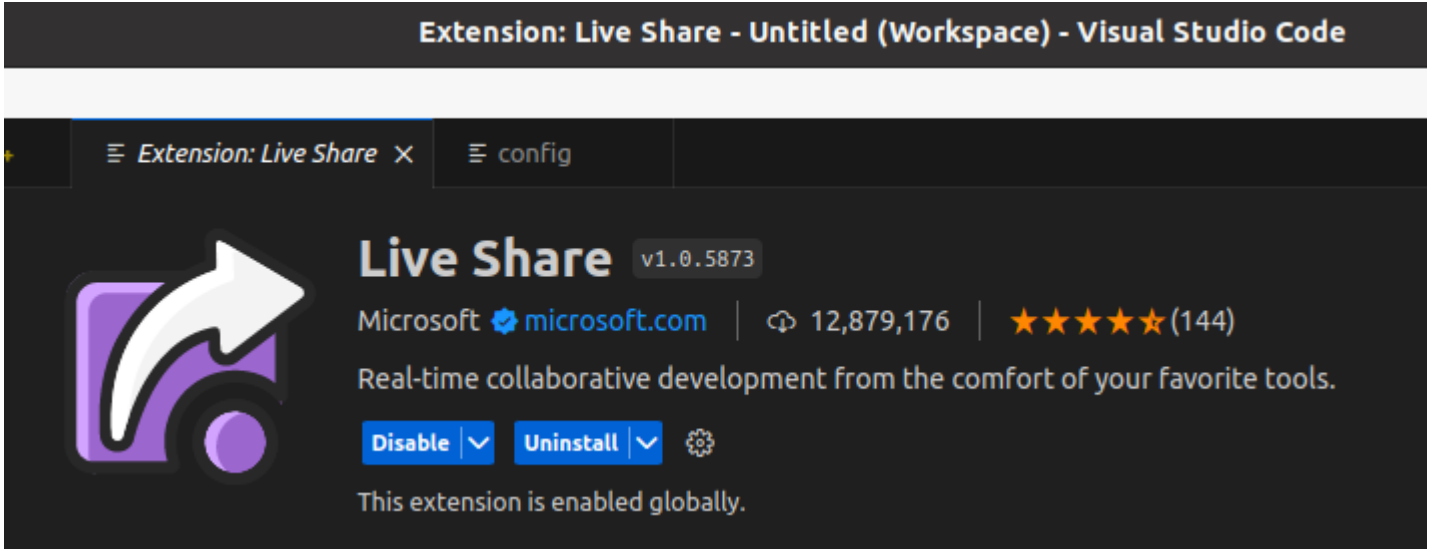
Comme vous voulez !

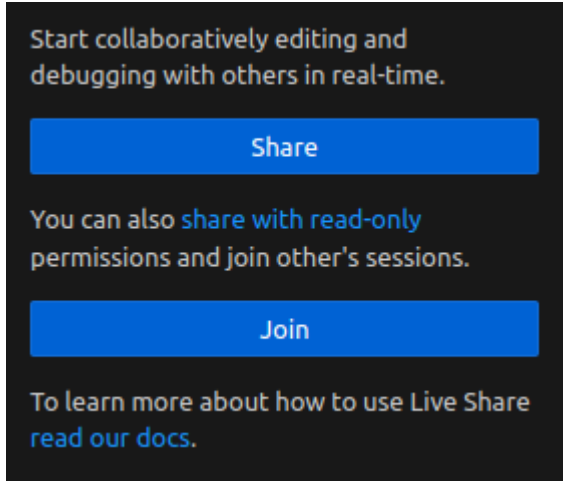
Regardez les fonctionnalités proposées

Pas d'idée ? VSCode est pas mal
(notamment intégration LiveShare & Copilot)

Les outils de travail

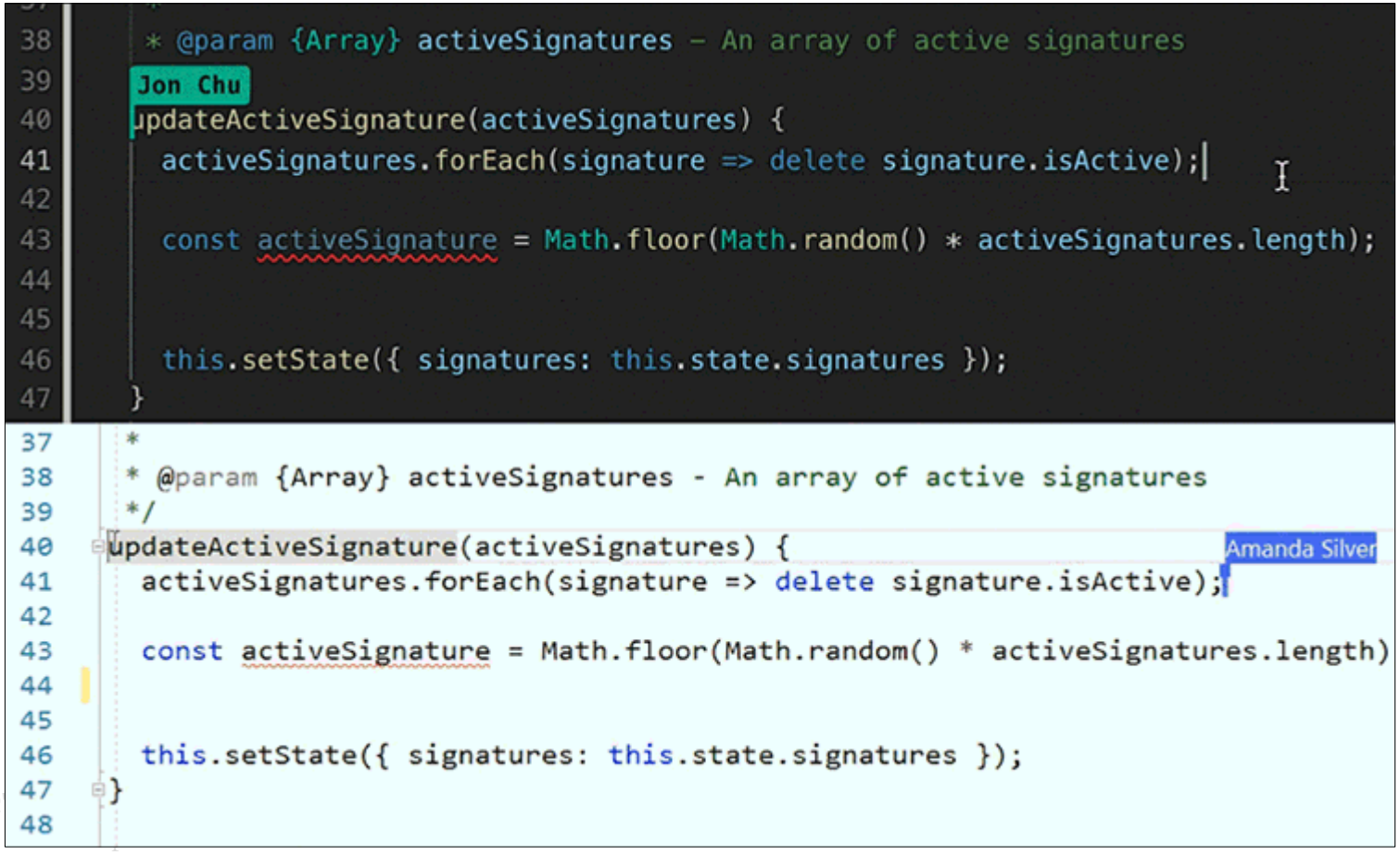
Développement collaboratif transparent (1)





Les outils de travail

Développement collaboratif transparent (2)



https://formations.imt-atlantique.fr/pyrat/

OUTLINE

- Introduction
- Présentation du jeu
- Le code
- Les outils de travail
- Programme de l’Episode 1

Programme de l'Episode 1

└ 2 créneaux de cours

Jour 1 (maintenant) :

- Rappels de Python
- Pause
- Installation du logiciel PyRat
- Constitution des binômes

Jour 2 (dans ~1 semaine) :

- Découverte de l'environnement PyRat
- Regarder les fichiers fournis
- Premières manipulations du labyrinthe
- Tests unitaires et statistiques

PY  RAT

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

PyRat c'est quoi
Bastien Pasdeloup
06/09/2023