Programming Course and Project

Summer Term 2024/25

Tutorial 3 - Git, algorithms, let's play!

Felix Lundt - May 5, 2025

Tentative outline for the first phase

	Content Software Carpentry	Algorithm/ Game Play	General
Week 1 April 14	Project Setup		Intro, Python & Numpy primer
Week 2 April 28	TDD	Code skeleton	
Week 3 May 5	Git `game_utils.py` example	Code to play Random Agent Algorithms	
Week 4 May 12	Debugging & Documentation		Exam Registration (May 12th)
Week 5 May 19	Profiling		Submission Prototype (end of week)

Plan for today

- Recap
- 'Good code' example
- Version control & git
- Algorithms
- Let's play
  - Code to play a game
  - Random agent example

Bad example

```python
def pretty_print_board(board: np.ndarray) -> str:
    """…
    delimiter_line = '|=============|\n'
    numbered_line = '|0 1 2 3 4 5 6|'

    lines = [_pretty_print_board_line(board_row) for board_row in board[::-1, :]]
    ret_str = delimiter_line + ''.join(lines) + delimiter_line + numbered_line

    return ret_str

def _pretty_print_board_line(board_row: np.ndarray) -> str:
    def board_pos_str_converter(elem: np.int8) -> str:
        if elem == PLAYER1:
            return PLAYER1_PRINT
        elif elem == PLAYER2:
            return PLAYER2_PRINT
        else:
            return NO_PLAYER_PRINT
    line = '|'
    for piece in board_row:
        line += board_pos_str_converter(piece) + ' '
    line = line[:-1] + '|\n'  # remove last space and add newline
    return line
```

- Bad naming (x, y)
- Nested loops/conditionals
- No use of provided types (board shape, BoardPiece)
- Even better: Loop over arrays directly
- Top/bottom delimiter (|==..==|) should be a variable
- Lines 32/33 inconsistent
- As a result:
  Tests are cumbersome!

Good example

```python
def pretty_print_board(board: np.ndarray) -> str:
    """…
    delimiter_line = '|=============|\n'
    numbered_line = '|0 1 2 3 4 5 6|'

    lines = [_pretty_print_board_line(board_row) for board_row in board[::-1, :]]
    ret_str = delimiter_line + ''.join(lines) + delimiter_line + numbered_line

    return ret_str

def _pretty_print_board_line(board_row: np.ndarray) -> str:
    def board_pos_str_converter(elem: np.int8) -> str:
        if elem == PLAYER1:
            return PLAYER1_PRINT
        elif elem == PLAYER2:
            return PLAYER2_PRINT
        else:
            return NO_PLAYER_PRINT
    line = '|'
    for piece in board_row:
        line += board_pos_str_converter(piece) + ' '
    line = line[:-1] + '|\n'  # remove last space and add newline
    return line
```

```python
def test_should_convert_empty_row_to_spaces():
    from game_utils import _pretty_print_board_line
    empty_row = np.zeros(BOARD_SHAPE[1], BoardPiece)
    n_characters = BOARD_SHAPE[1] * 2 - 1
    expected_output = '|' + ' ' * n_characters + '|\n'
    output = _pretty_print_board_line(empty_row)
    assert output == expected_output

def test_should_convert_player1_to_x():
    from game_utils import _pretty_print_board_line
    p1_row = np.array([1, 0, 0, 1, 0, 0, 1], BoardPiece)
    expected_output = '|' + 'X     X     X' + '|\n'
    output = _pretty_print_board_line(p1_row)
    assert output == expected_output

def test_should_convert_player2_to_o():
    from game_utils import _pretty_print_board_line
    p2_row = np.array([0, 0, 0, 0, 2, 0, 0], BoardPiece)
    expected_output = '|' + '        O     ' + '|\n'
    output = _pretty_print_board_line(p2_row)
    assert output == expected_output

def test_should_convert_full_row():
    from game_utils import _pretty_print_board_line
    full_row = np.array([1, 2, 1, 2, 1, 2, 1], BoardPiece)
    expected_output = '|' + 'X O X O X O X' + '|\n'
    output = _pretty_print_board_line(full_row)
    assert output == expected_output
```

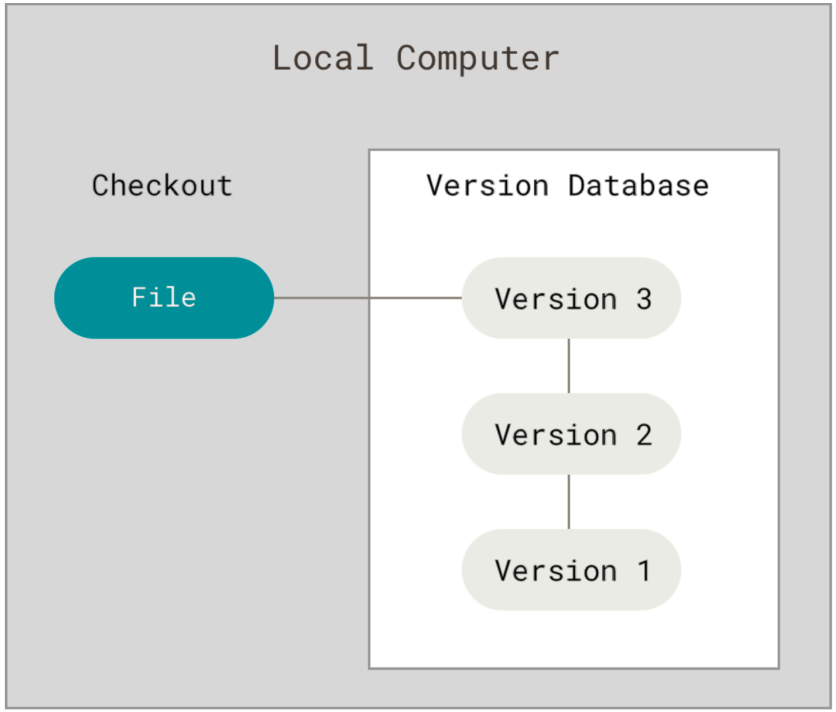
Good example

- Limited complexity for tests of helper function
- Clear setup
- One test for the full `pretty_print_board` function to verify output format would be enough
- Clear names
- One assertion per test

```python
def test_should_convert_empty_row_to_spaces():
    from game_utils import _pretty_print_board_line
    empty_row = np.zeros(BOARD_SHAPE[1], BoardPiece)
    n_characters = BOARD_SHAPE[1] * 2 - 1
    expected_output = '|' + ' ' * n_characters + '|\n'
    output = _pretty_print_board_line(empty_row)
    assert output == expected_output

def test_should_convert_player1_to_x():
    from game_utils import _pretty_print_board_line
    p1_row = np.array([1, 0, 0, 1, 0, 0, 1], BoardPiece)
    expected_output = '|' + 'X     X     X' + '|\n'
    output = _pretty_print_board_line(p1_row)
    assert output == expected_output

def test_should_convert_player2_to_o():
    from game_utils import _pretty_print_board_line
    p2_row = np.array([0, 0, 0, 0, 2, 0, 0], BoardPiece)
    expected_output = '|' + '        O    ' + '|\n'
    output = _pretty_print_board_line(p2_row)
    assert output == expected_output

def test_should_convert_full_row():
    from game_utils import _pretty_print_board_line
    full_row = np.array([1, 2, 1, 2, 1, 2, 1], BoardPiece)
    expected_output = '|' + 'X O X O X O X' + '|\n'
    output = _pretty_print_board_line(full_row)
    assert output == expected_output
```

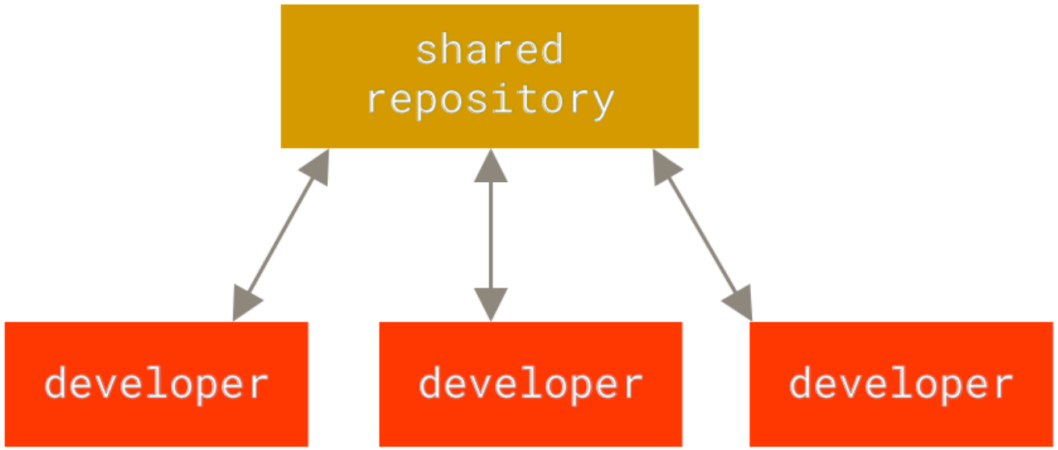
Git Crashcourse

Version Control with Git



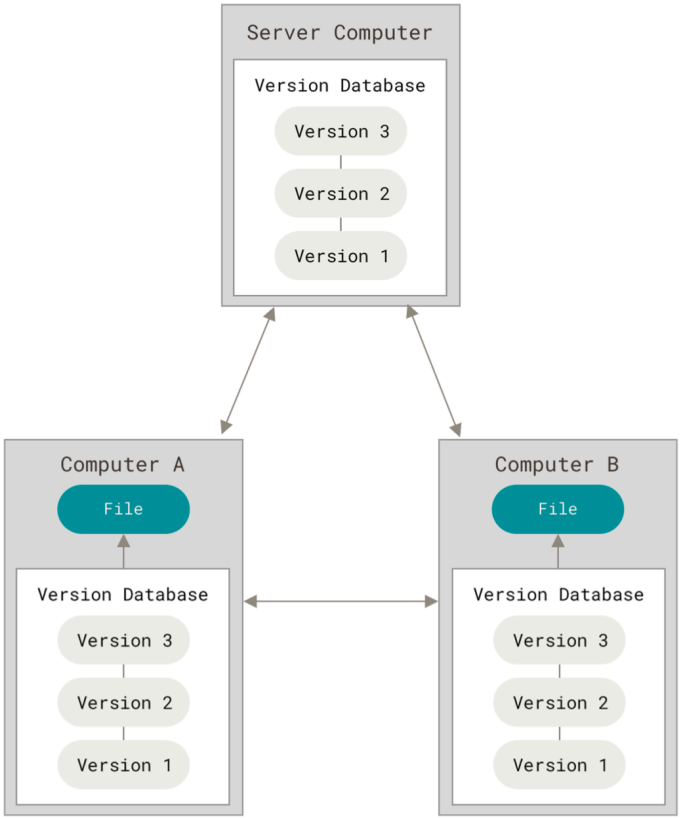
Source: Chacon & Straub: Pro Git
(linked on Moodle page)

Version Control with Git



Source: Chacon & Straub: Pro Git
(linked on Moodle page)

Version Control with Git



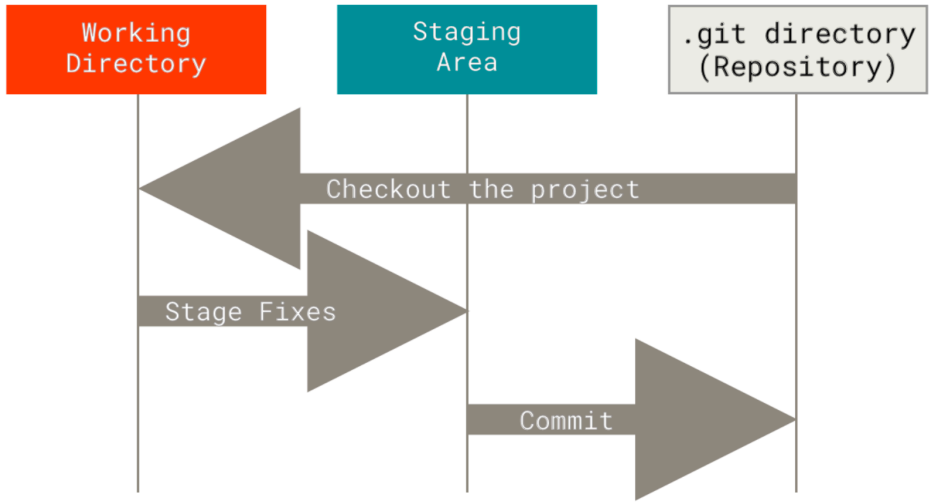
Example: Git!

Source: Chacon & Straub: Pro Git
(linked on Moodle page)

Version Control with Git

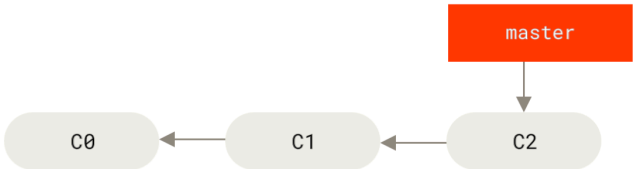
Git characteristics

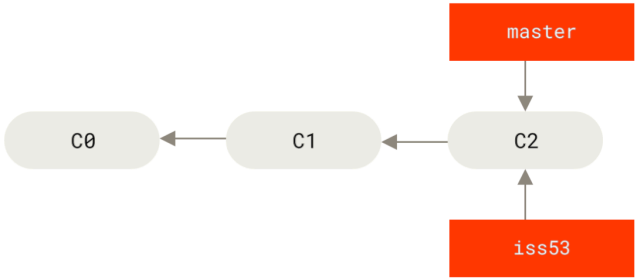
- Snapshots, not differences
- (Almost) all operations are local
- Integrity
- Git only adds data (most of the time)
- Three stages

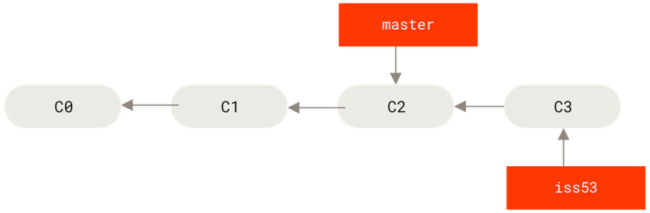


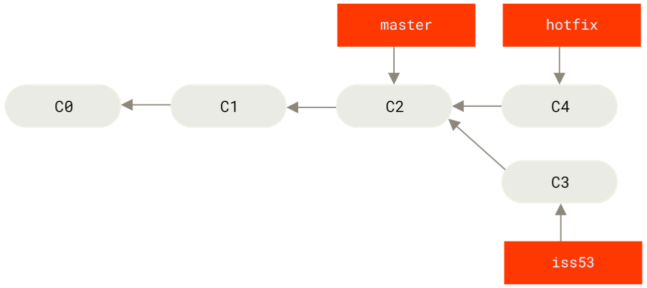
Source: Chacon & Straub: Pro Git
(linked on Moodle page)

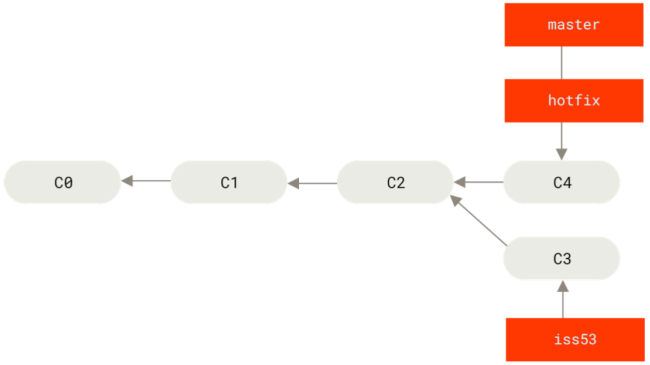
Branching in Git



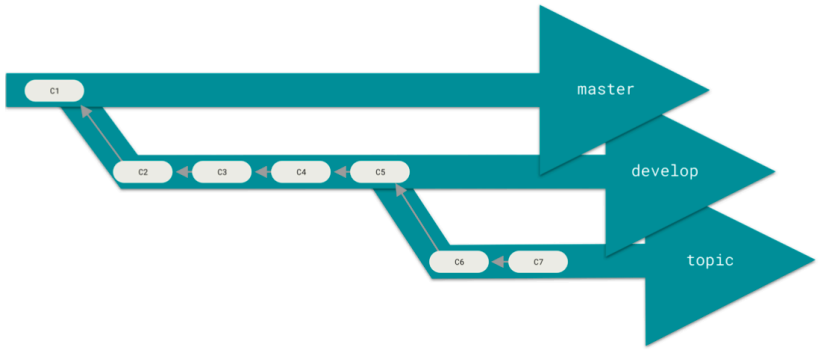




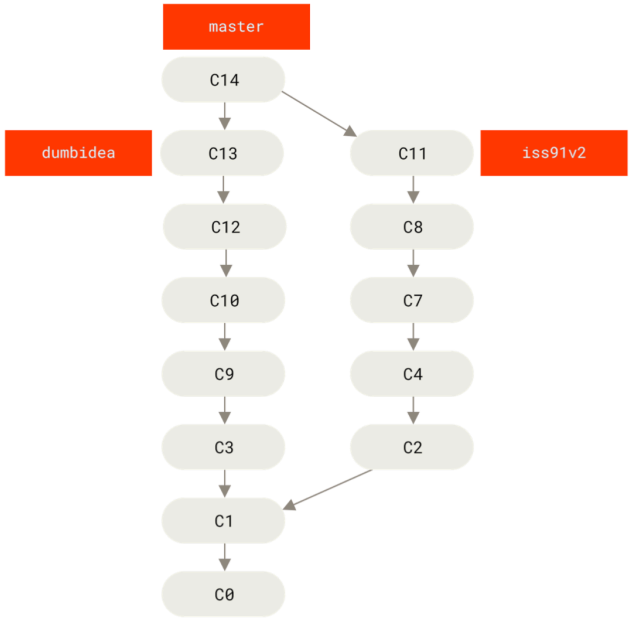




Workflows



Progressive-stability branching



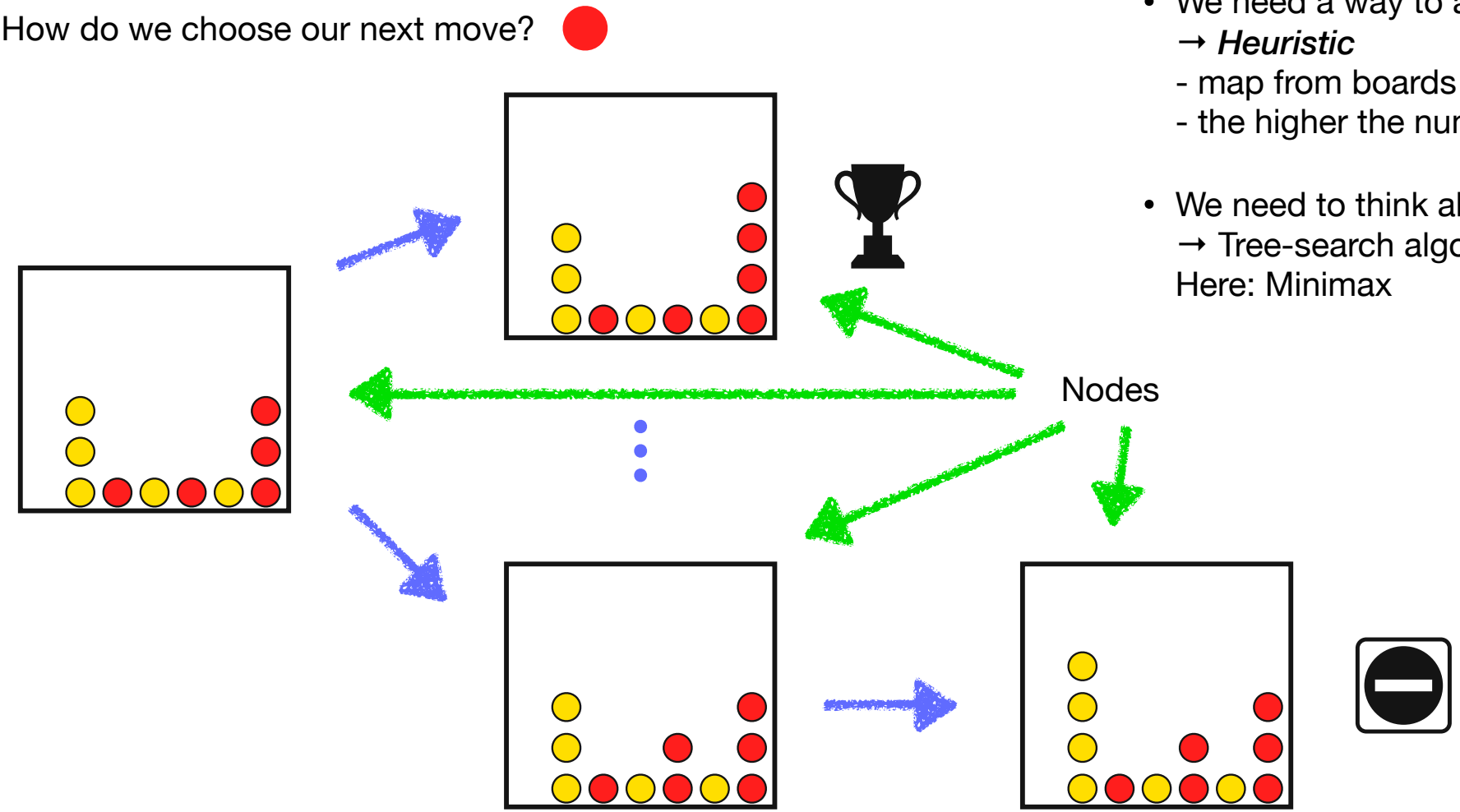
Topic branches

For more examples, see book 'Pro Git', chapter 'Distributed Git'!

Algorithms Crashcourse

The Minimax algorithm

How do we choose our next move?

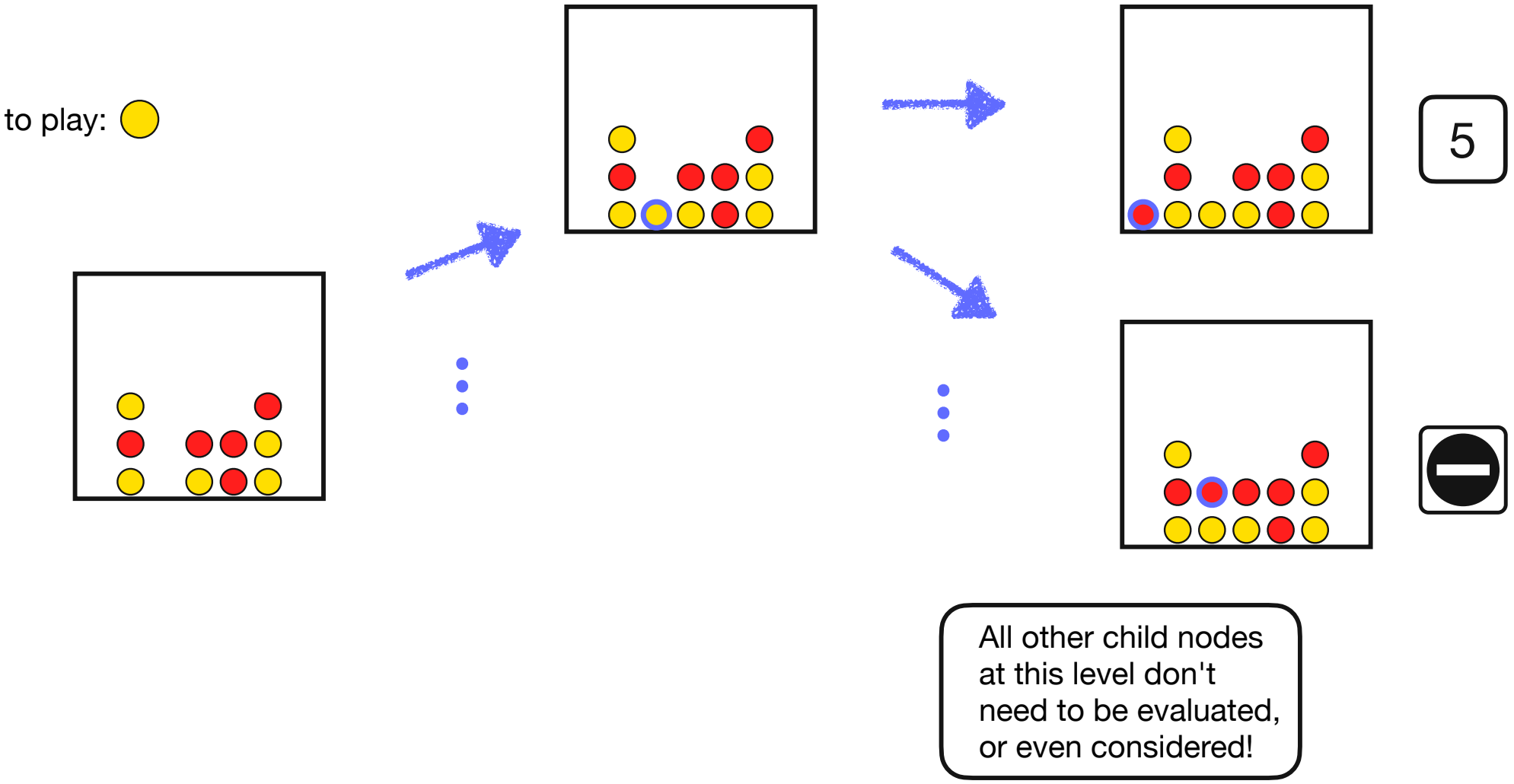


Two ingredients:

- We need a way to assess boards
  → *Heuristic*
  - map from boards to real numbers
  - the higher the number, the better

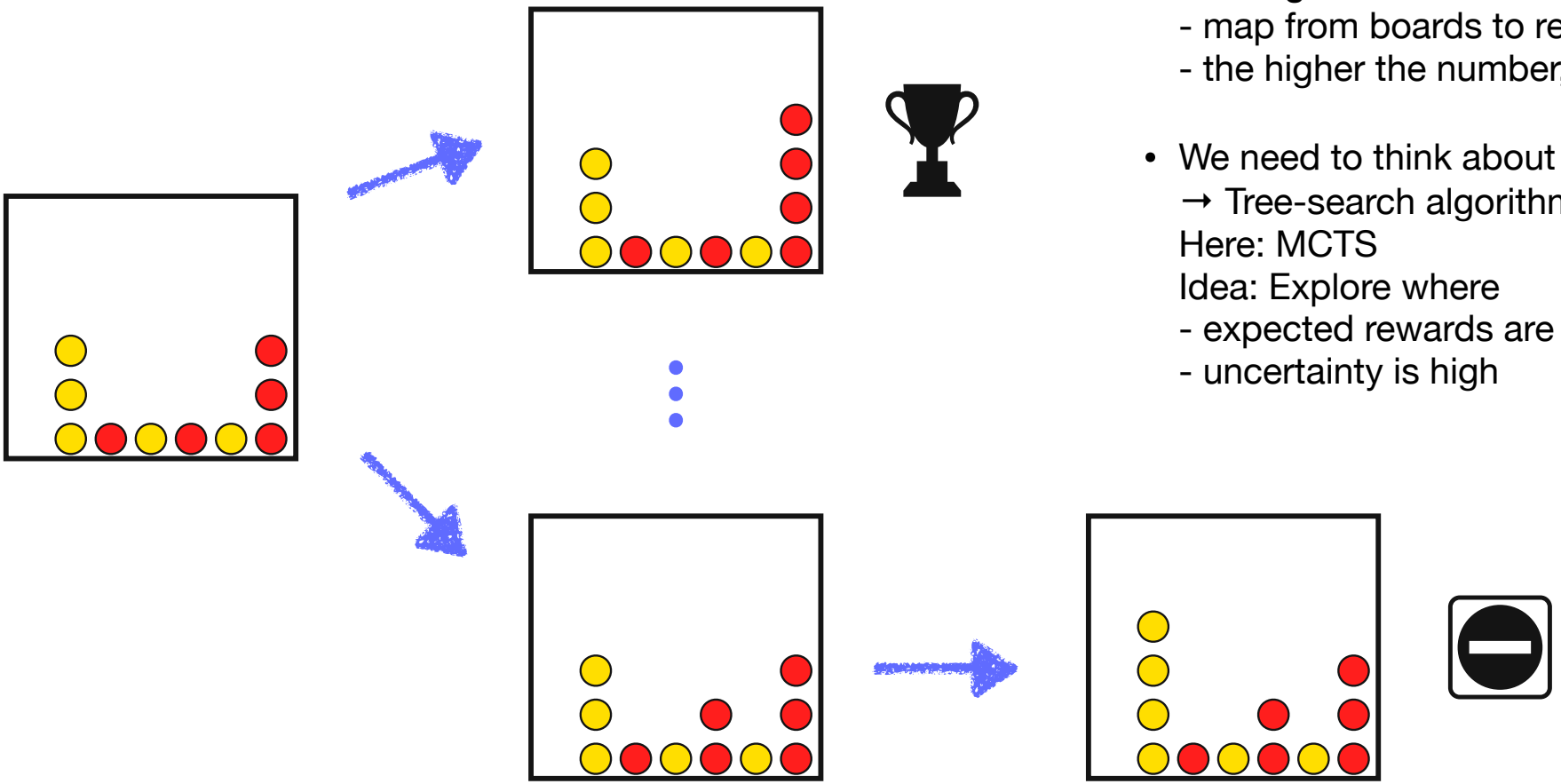
- We need to think about *future* moves
  → Tree-search algorithm
  Here: Minimax

Alpha-beta pruning



The MCTS algorithm

How do we choose our next move?

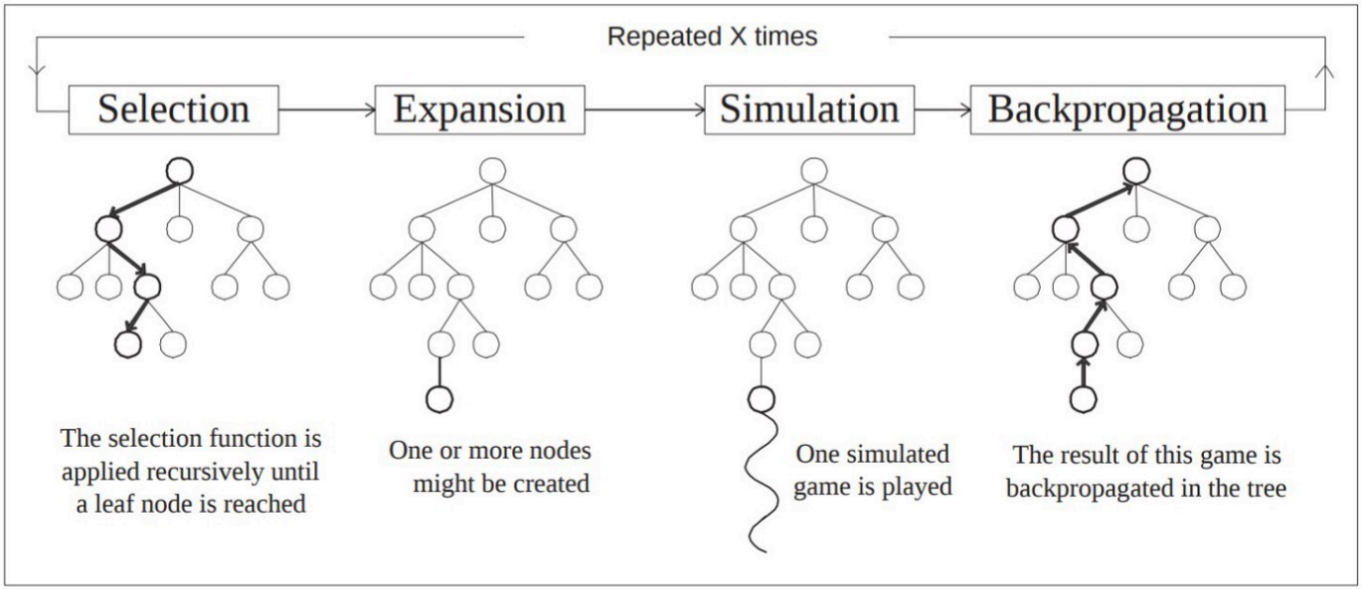


Two ingredients:

- We need a way to assess boards
  → ***Simulate hypothetical games and use winning rate***
  - map from boards to real numbers
  - the higher the number, the better

- We need to think about *future* moves
  → Tree-search algorithm
  Here: MCTS
  Idea: Explore where
  - expected rewards are high
  - uncertainty is high

The MCTS algorithm



MCTS algorithm, diagram from Chaslot (2006)

Two ingredients:

- We need a way to assess boards
→ ***Simulate hypothetical games and use winning rate***
- map from boards to real numbers
- the higher the number, the better

We need to think about *future* moves
→ Tree-search algorithm
Here: MCTS
Idea: Explore where
- expected rewards are high
- uncertainty is high

Needed:

- Selection policy (UCT: $\frac{w_i}{s_i} + c\sqrt{\frac{\ln s_p}{s_i}}$)
- Simulation policy (random, …)
- Policy to choose a move at the end (highest UCT value, most visits, highest winning rate, ..)

Assignments this week

1. Put your project under version control using Git (see links in material) and GitHub
2. Check out the git tutorials (links in material)
3. Add code snippets (see material) and make sure you understand them.
4. Implement an agent playing randomly (more info in material).
5. Start to implement your agent using Minimax/Negamax (with alpha-beta pruning) or MCTS