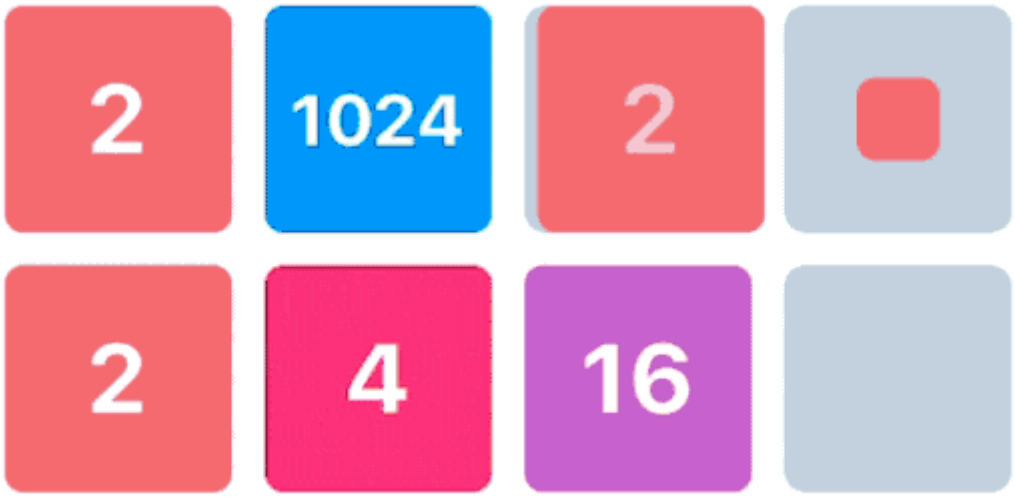
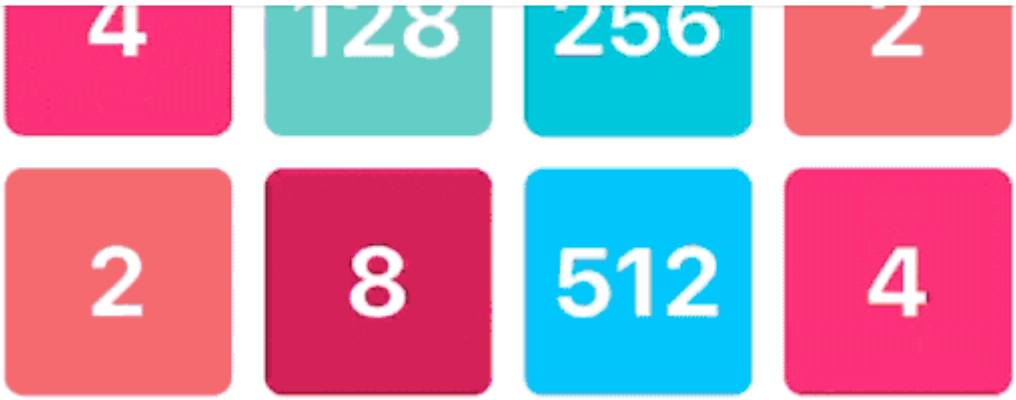
2048 - Solving 2048 with AI 🤖

Leveraging Monte-Carlo (MC) Move Evaluation

Greg Surma
Nov 5, 2018 · 6 min read

In today's article, I am going to show you how to solve the famous **2048** game with **Artificial Intelligence**. You will learn the essentials behind the **Monte-Carlo** algorithm and at the end of this article, you will be able to create an agent that without any domain-specific knowledge beats average human scores in 2048.





AI gameplay

Table of Contents

If you are an **iOS** user, feel free to check my **2048 AI** app that allows to both play 2048 and watch how AI solves it! It would definitely help you follow this article.

2048 - AI Solver

Classic 2048 game enhanced with AI Solver! * Challenge yourself with amazing and addictive gameplay! * Check hints...

itunes.apple.com

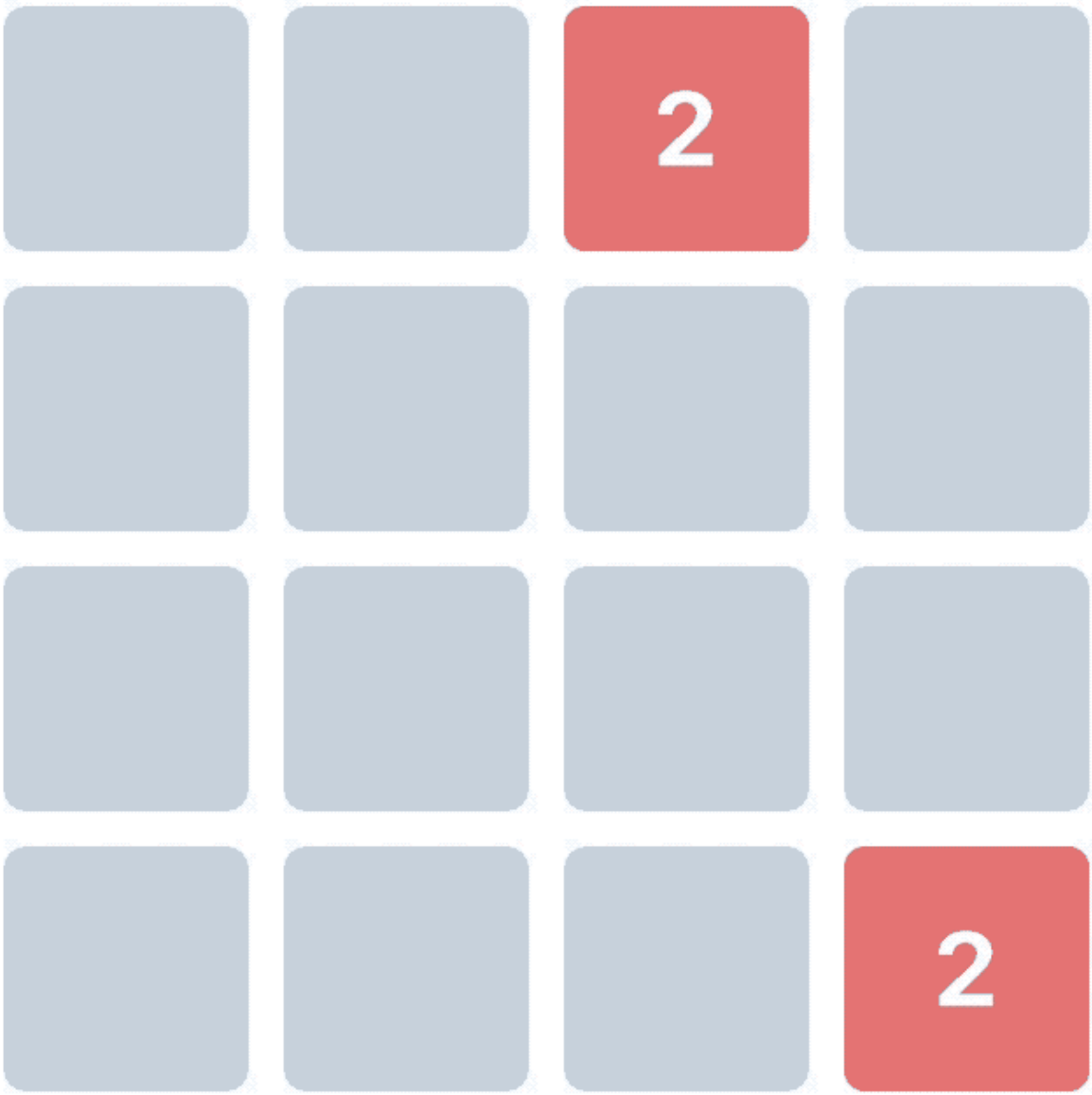
About 2048

> **2048** is a single-player sliding block puzzle game designed by Italian web developer Gabriele Cirulli. The game's objective is to slide numbered tiles on a grid to combine

> “Candy Crush for math geeks”, Wall Street Journal

How to play?

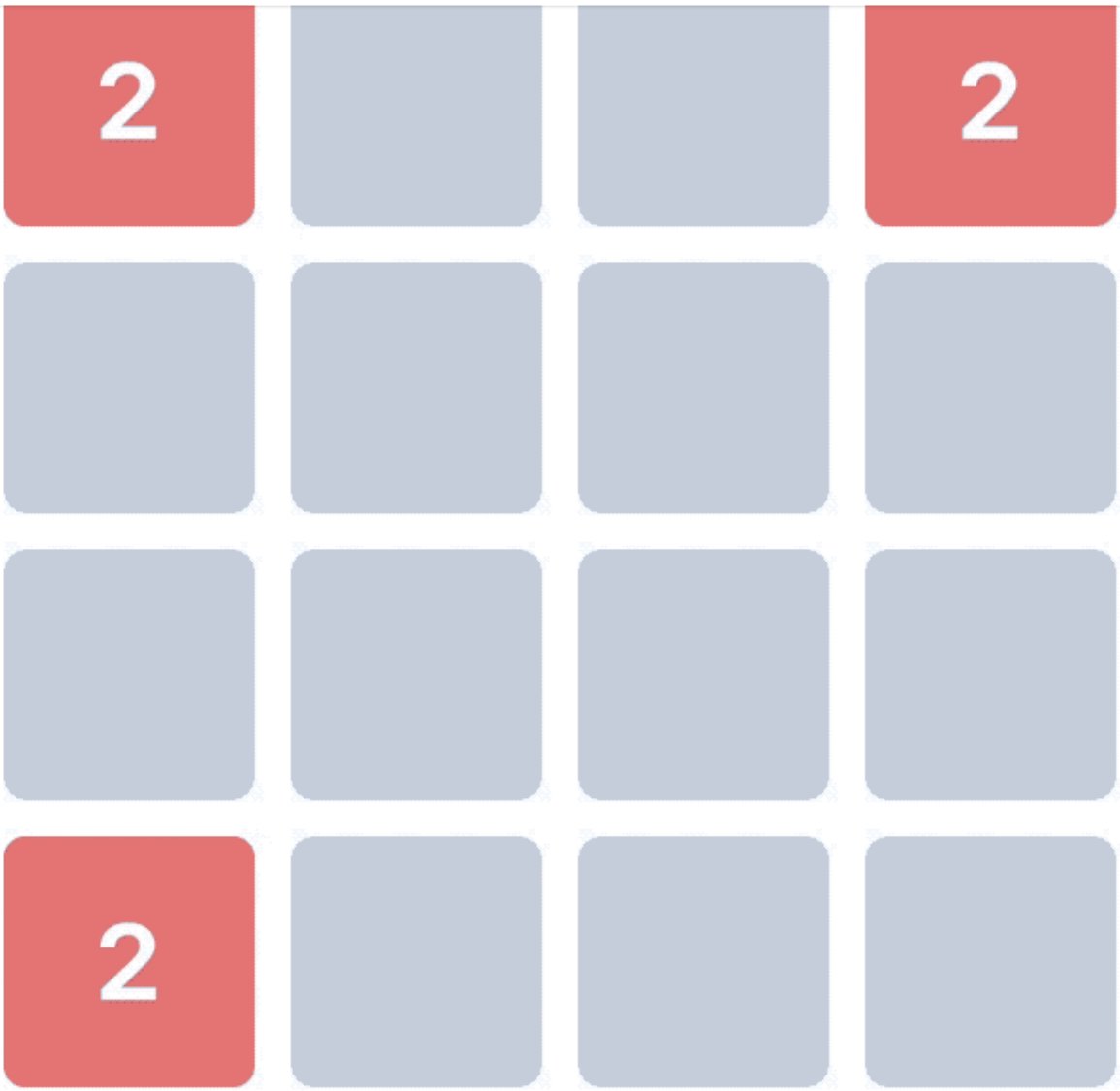
Swipe ↑, →, ↓, or ← to move the tiles. Every move generates a new tile at a random unoccupied position.



Moving tiles with ← swipe

How to score?

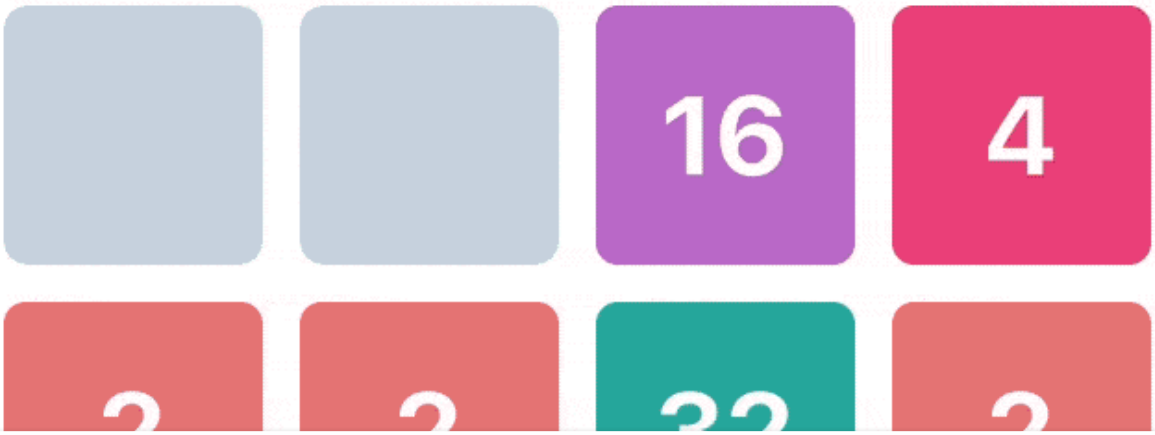
Merge tiles with the same values. Every merged tile is added to your score.

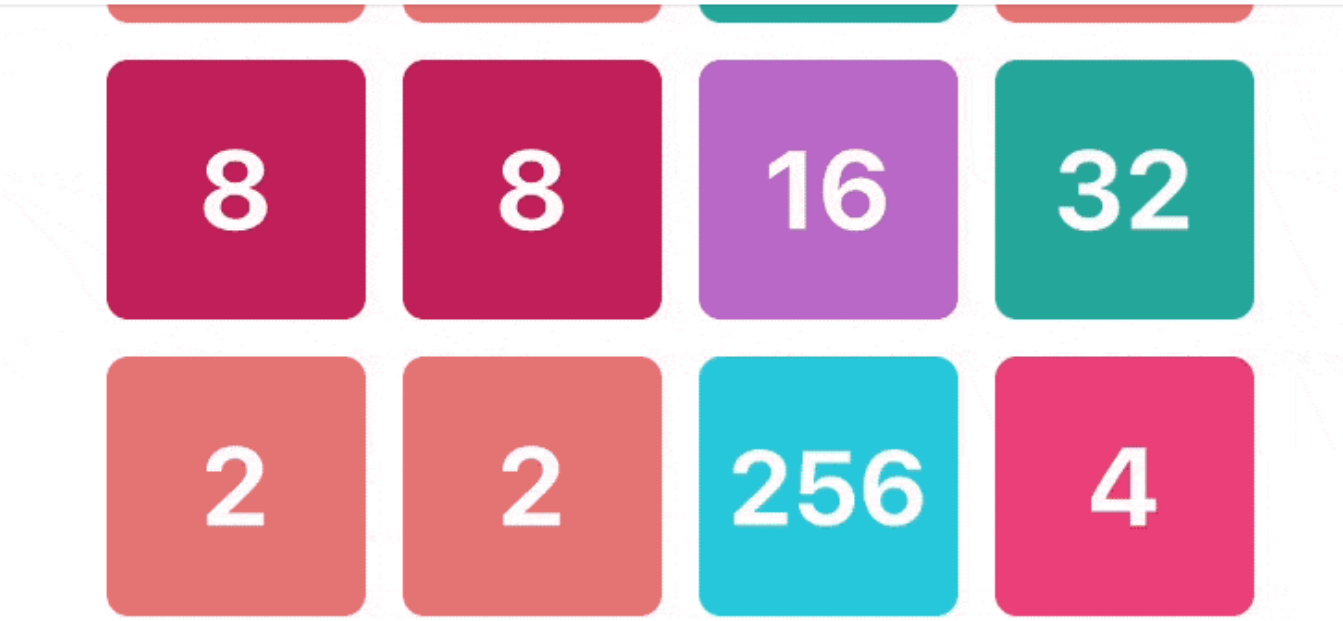


Merging tiles with ↓ action

What's the goal of the game?

To score as many points as possible.





Gameplay example

Before proceeding to the AI solver part, I would recommend you to play the game yourself. You will get more familiar with its rules and possibilites.

Monte-Carlo (MC)

One of the possible ways to solve the game of 2048 is to exploit the MC algorithm. Its biggest advantage is that it is a general-purpose solver, which means that it can yield output without any game specific input.

So how does it work?

Monte-Carlo is as a heuristic search algorithm that is based on applying the most promising moves.

But how can we determine which one of the available moves ($\uparrow$, $\rightarrow$, $\downarrow$, $\leftarrow$) is the most promising?

In order to do so, we can for every move:

1. Execute a series of background runs.
2. Group them by the initial move.
3. Count an average final score for each initial move.

. . .

For the more detailed explanation of the magic behind the MC algorithm, please check the following article that was deeply focused on the general purpose solvers.

Slitherin - Solving the classic game of Snake🐍 with AI🤖 (Part 2: General Purpose - Random, Monte...

Welcome to Part 2 of the Slitherin - Solving the classic game of Snake🐍 with AI🤖 project! If you missed Part 1, don't...

towardsdatascience.com

Now that we know the underlying concept of the MC algorithm, let's apply it to the game of 2048!

2048 AI Solver

Let's get straight to the point. Our goal is to predict the best move for a given board state.

```
func getBestMove(tiles: [Tile]) -> ShiftDirection? {
    var grids = [[Tile]]()
    for _ in 1...numberOfRuns {
        grids.append(copyTiles(originalTiles: tiles))
    }
    var runs = [Run]()
    for grid in grids {
        if let run = generateRun(grid: grid) as Run? {
            runs.append(run)
```

```
12      return getBestMoveForRuns(runs: runs)
13  }
```

2048_mcts.swift hosted with ❤ by GitHub view raw

Line 1

To receive the best move (ShiftDirection), we are supplying our getBestMove function with the current board state i.e an array of tiles, where each tile has a value and a position.

. . .

Lines 2–5

We need to copy our board *n* times, where *n* is the number of runs that we are going to perform.

. . .

Lines 6–11

For each copied board, we are going to perform a random run. Generating random run is very simple, we just need to execute random actions until we have no available moves left. Ultimately we are going to persist its initial move and the final score.

```
1  struct Run {
2      var initialMove: Direction
3      var finalScore: Int
4  }
```

2048_run.swift hosted with ❤ by GitHub view raw

. . .

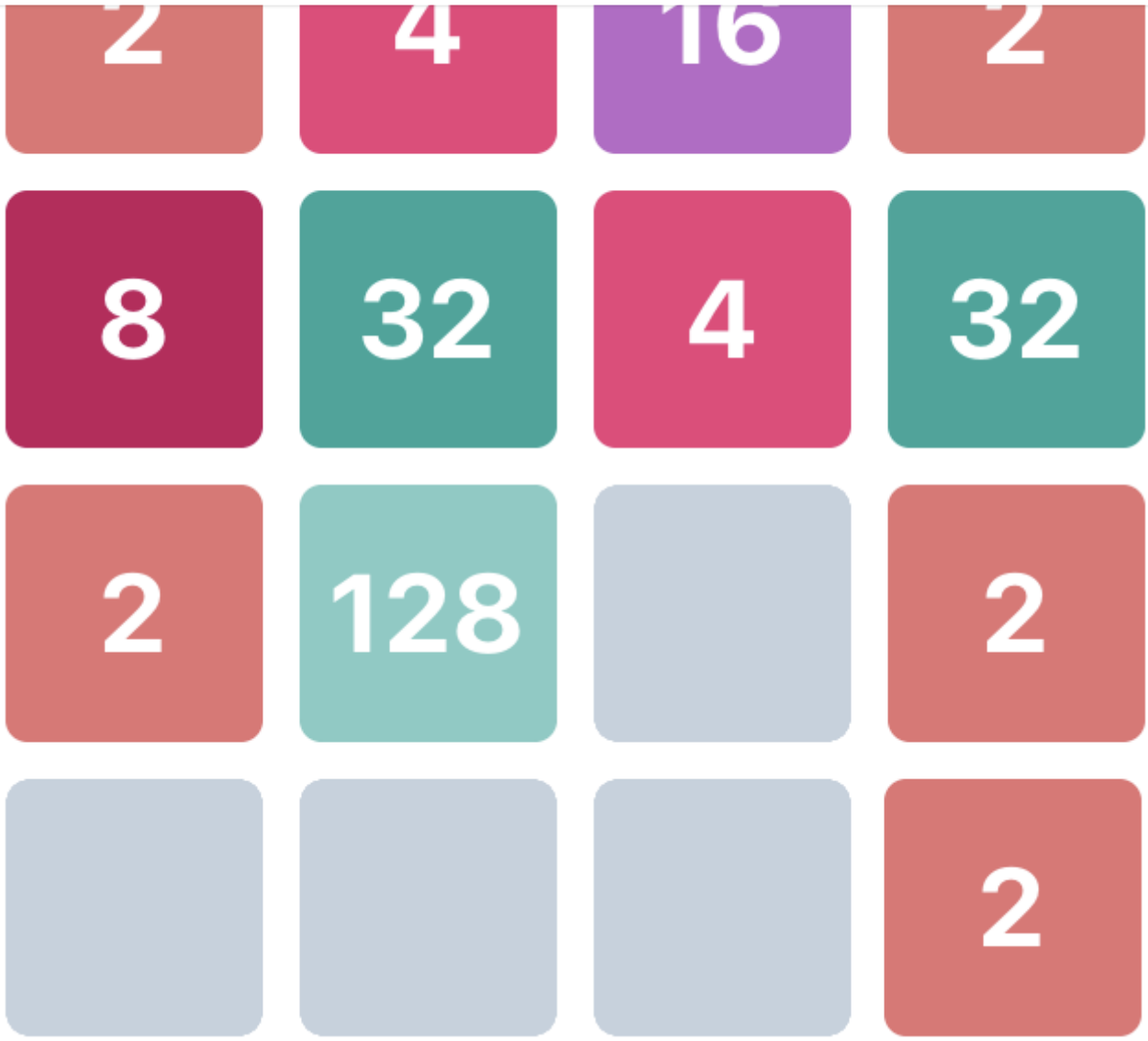
Line 12

```swift
private func getBestMoveForRuns(runs: [Run]) -> ShiftDirection? {
    guard !runs.isEmpty else {
        print("Received no runs - end of game")
        return nil
    }

    let runsForUp = runs.filter { $0.initialMove.direction == .up }
    let runsForDown = runs.filter { $0.initialMove.direction == .down }
    let runsForLeft = runs.filter { $0.initialMove.direction == .left }
    let runsForRight = runs.filter { $0.initialMove.direction == .right }

    let averageScoreForUp = getAverageScoreForRuns(runs: runsForUp)
    let averageScoreForDown = getAverageScoreForRuns(runs: runsForDown)
    let averageScoreForLeft = getAverageScoreForRuns(runs: runsForLeft)
    let averageScoreForRight = getAverageScoreForRuns(runs: runsForRight)

    var selectedMove: ShiftDirection?
    switch [averageScoreForUp, averageScoreForDown, averageScoreForLeft, averageScore
    case averageScoreForUp?:
        selectedMove = .up
    case averageScoreForDown?:
        selectedMove = .down
    case averageScoreForRight?:
        selectedMove = .right
    case averageScoreForLeft?:
        selectedMove = .left
    default:
        break
    }
    return selectedMove
}
```

It may look tricky at a first sight, so let's dive into an example to get a better understanding of what's going on.

. . .

Given a following initial state.



Initial state

Let's ask our **MC AI** what's the best move in such a situation. For the sake of simplicity, let's perform only 10 background runs starting from the given state.

Our random background runs' results look as follows, they are already grouped by their initial moves.

```
(direction: .up, finalScore: 1356)
(direction: .up, finalScore: 2144)

(direction: .down, finalScore: 1040)
(direction: .down, finalScore: 2432)
(direction: .down, finalScore: 1536)

(direction: .left, finalScore: 1196)
(direction: .left, finalScore: 1300)
```

```
(direction: .right, finalScore: 1364)
(direction: .right, finalScore: 2208)
```

2048_exampleResults.py hosted with ❤ by GitHub view raw

Now let's calculate an average final score for a given initial move.

```
Up 1750.0
Down 1669.0
Left 1248.0
Right 1649.0
```

2048_example_averages.py hosted with ❤ by GitHub view raw

Finally, let's select an initial move associated with the maximum value.

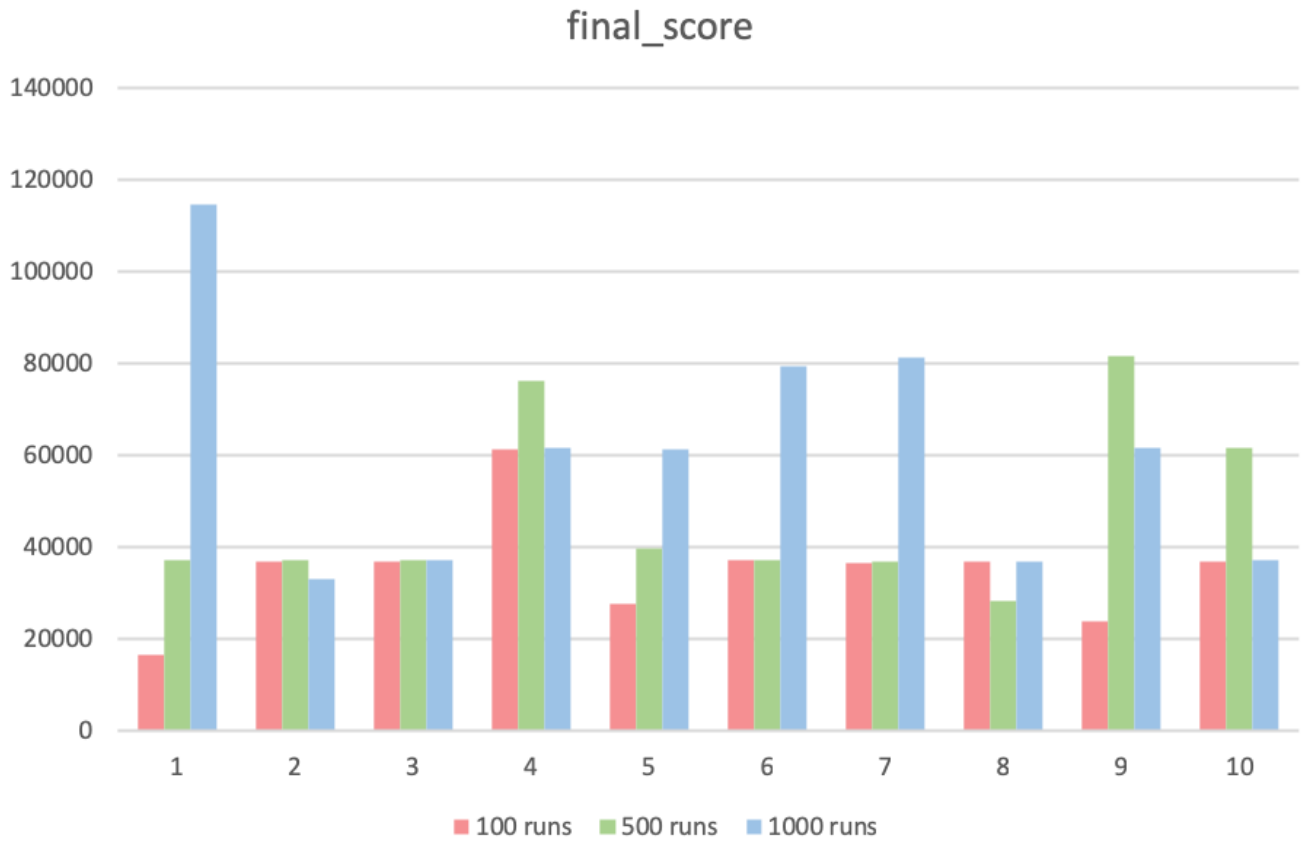
MC AI's prediction would be to swipe **Up**.

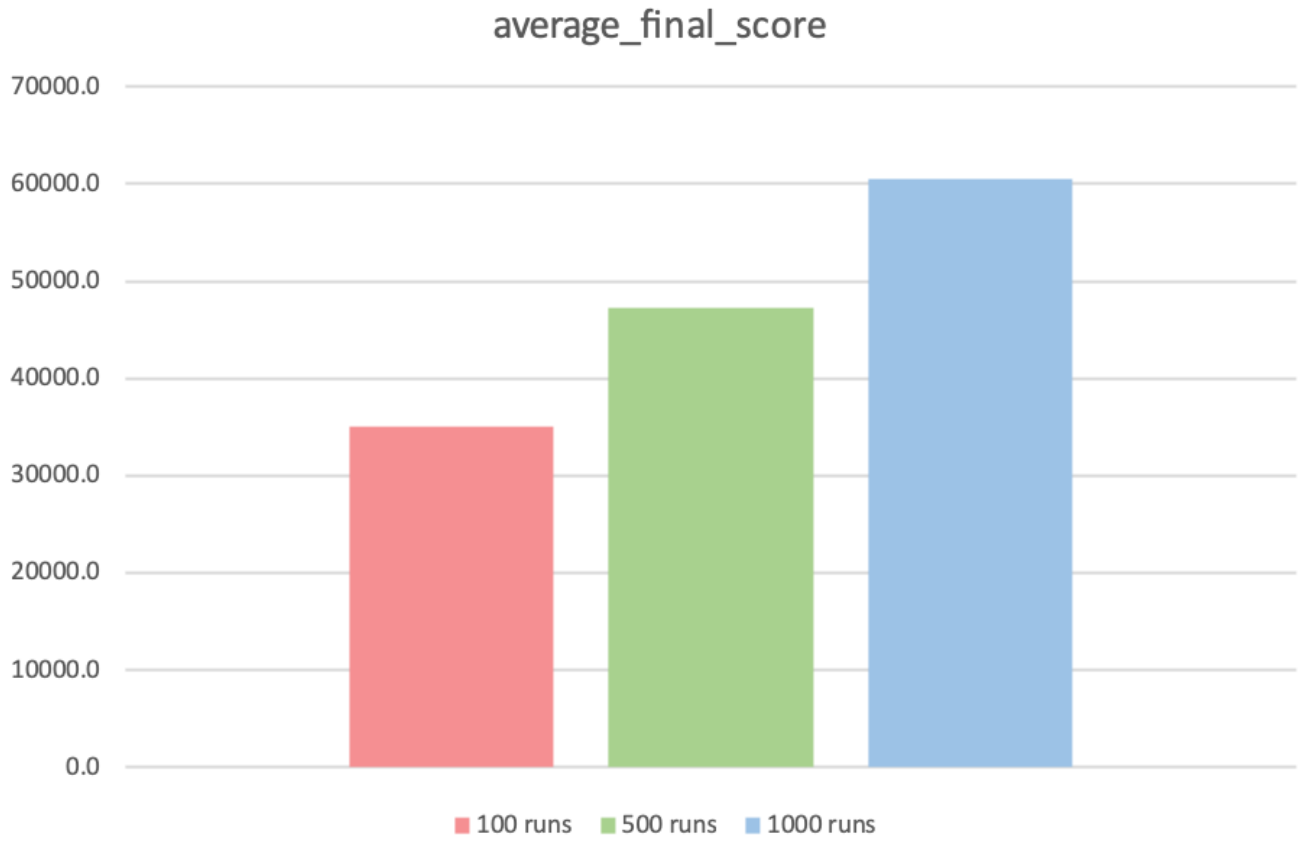
Results

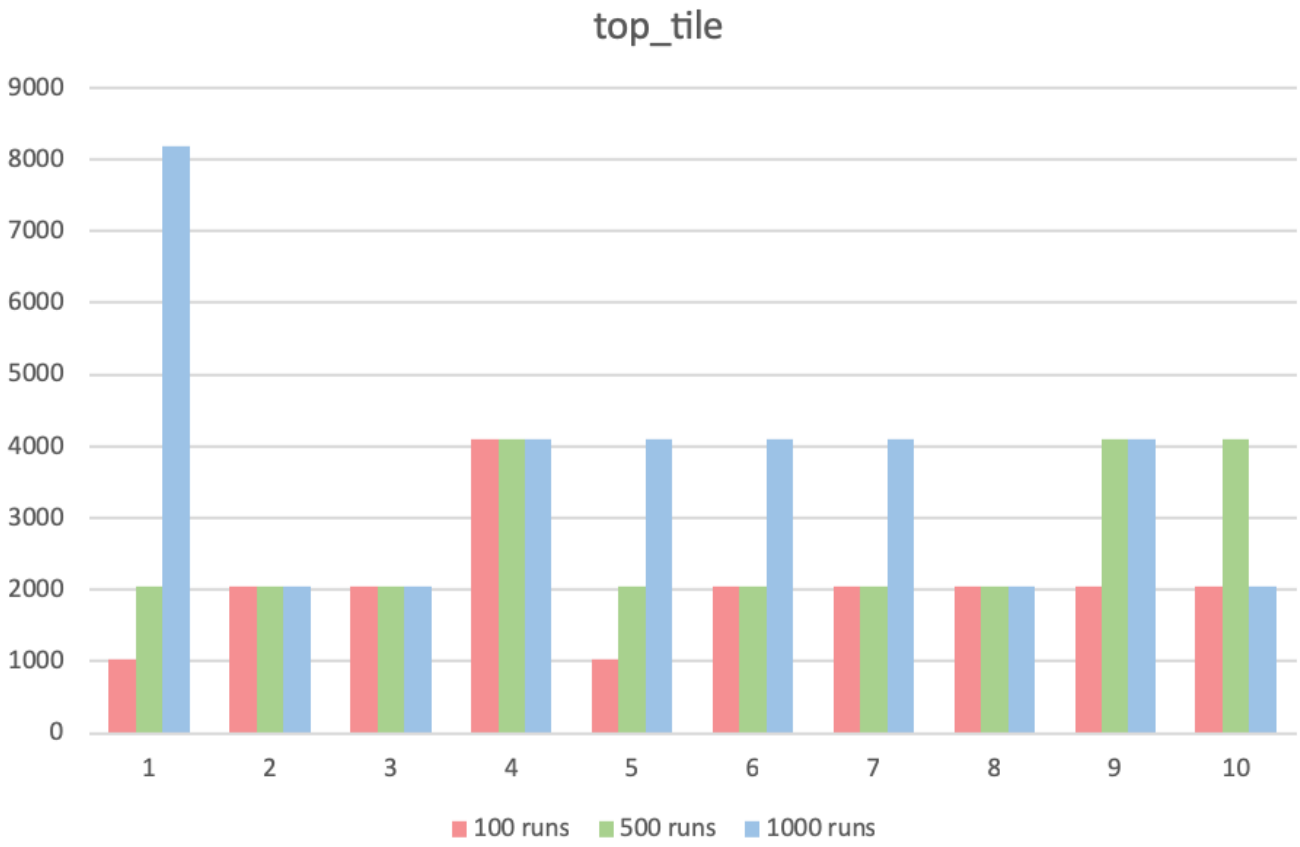
To measure AI Solver's performance, I've decided to perform 10 full games for every configuration. By configuration I mean how many background runs should be performed to calculate each move, i.e 100, 500 or 1000 runs.

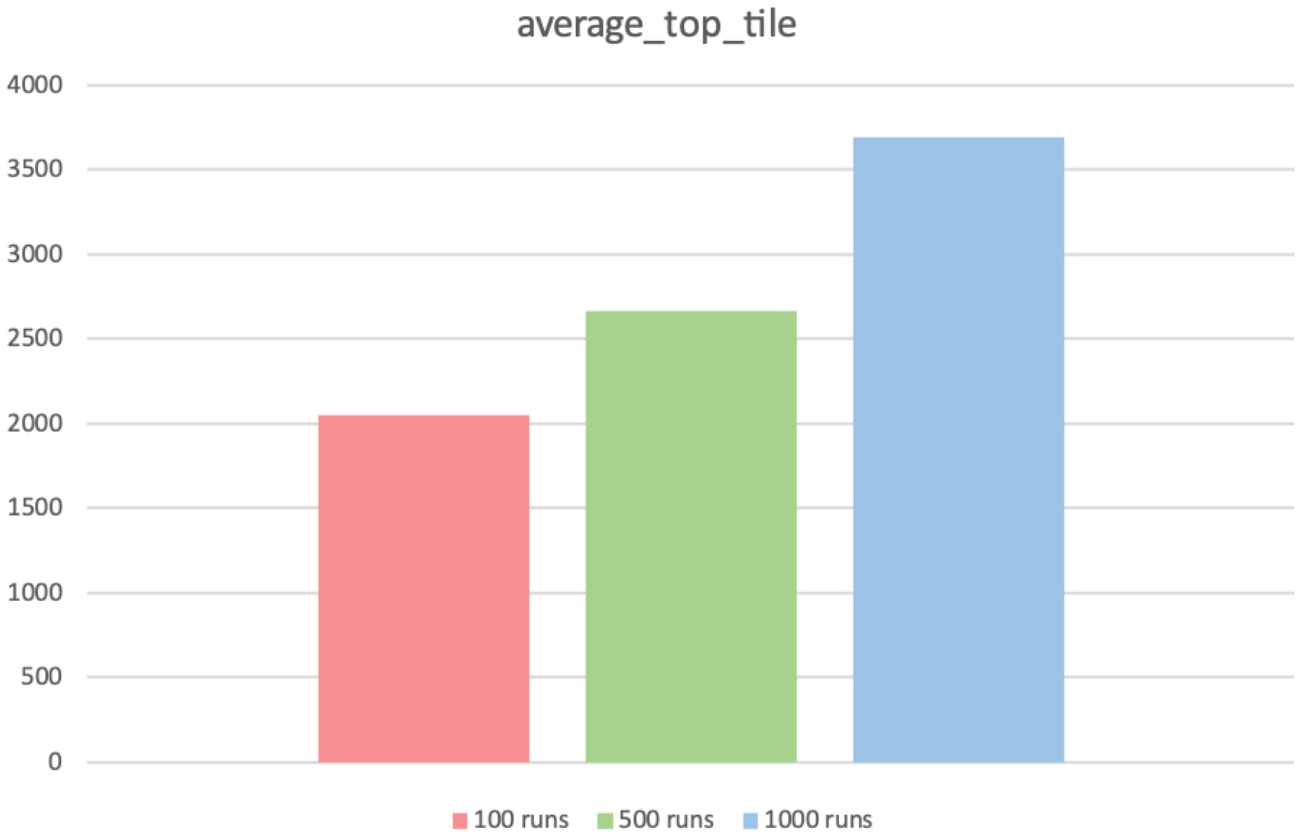
Results look as follows.

	100 runs		500 runs		1000 runs	
#	final_score	top_tile	final_score	top_tile	final_score	top_tile
1	16556	1024	37128	2048	114768	8192
2	36892	2048	37132	2048	32968	2048
3	36836	2048	37208	2048	37088	2048
4	61504	4096	76268	4096	61560	4096
5	27732	1024	39656	2048	61496	4096
6	37064	2048	37248	2048	79440	4096
7	36604	2048	36944	2048	81496	4096
8	36812	2048	28180	2048	36984	2048
9	24040	2048	81588	4096	61656	4096
10	36876	2048	61640	4096	37068	2048
$\bar{x}$	35091.6	2048.0	47299.2	2662.4	60452.4	3686.4









game i.e domain-specific inputs or hardcoded rules. It is a general-purpose solver that because of its versatility can be applied to a wide majority of applications.

Bonus

If you are a true **2048** fan, check my other 2048-based game called **Tetris X 2048**. As the name suggests, it combines the classic **Tetris** with **2048**. It's very challenging but definitely rewarding. Don't hesitate to check it out!

Falling numbers X

Read reviews, compare customer ratings, see screenshots, and learn more about Falling numbers X. Download Falling...

itunes.apple.com

What's next?

While MC algorithm proved to be very successful in solving the game of 2048, we don't have to stop here. MC can be implemented in other games and applications as well and I encourage you to try it. I am looking forward to seeing your results!

. . .

Don't forget to check the project's iOS app.

2048 - AI Solver

Classic 2048 game enhanced with AI Solver! * Challenge yourself with amazing and addictive gameplay! * Check hints...

itunes.apple.com

. . .

And don't forget to 👋 if you enjoyed this article 🙂.

Artificial Intelligence Programming Data Science Technology Machine Learning

About Help Legal