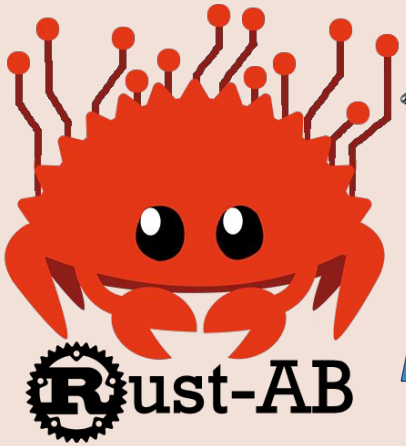
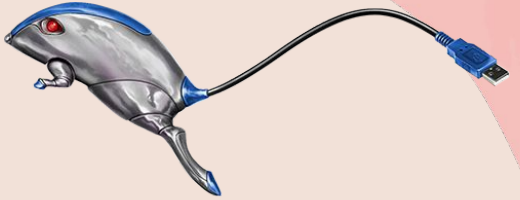


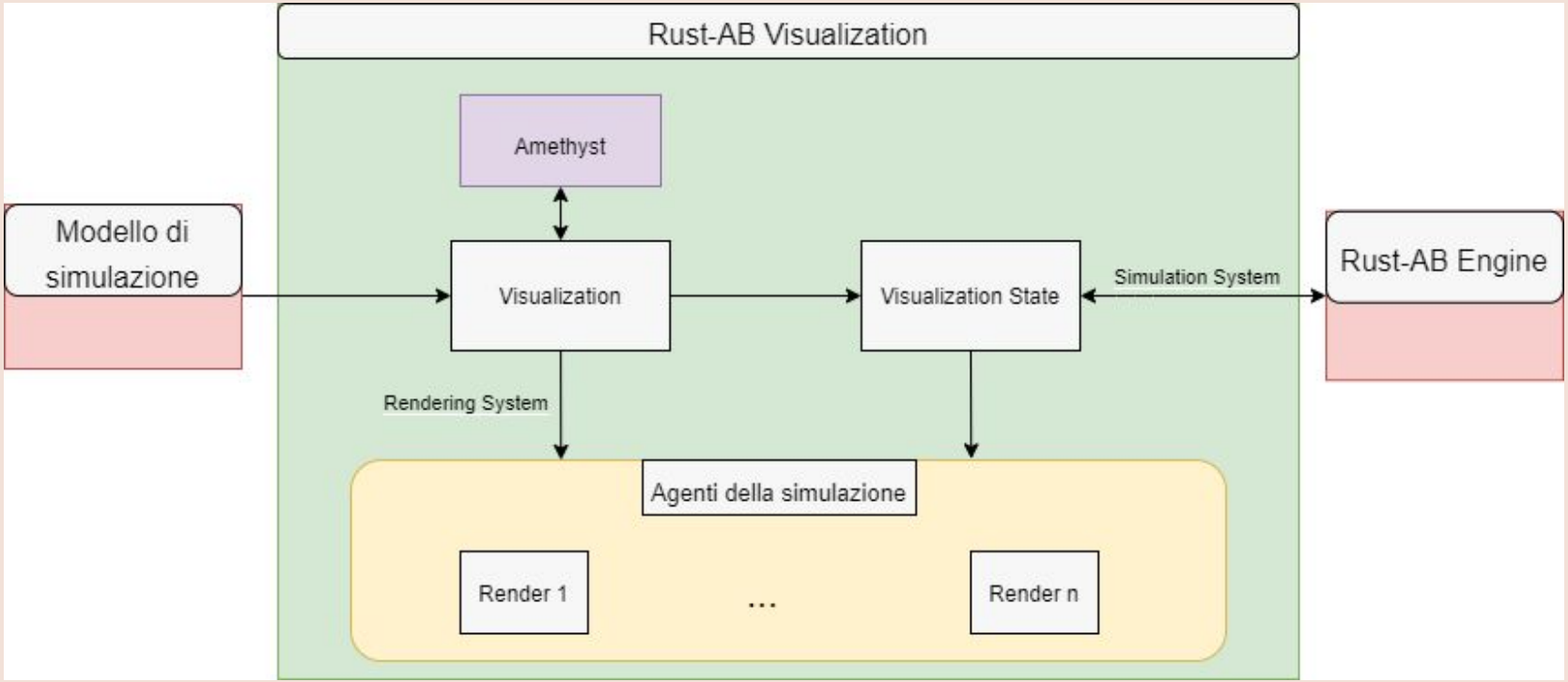


ISISLab

RUST-AB: REFACTORING & OPTIMIZATIONS

By Francesco Foglia & Pasquale Caramante

ARCHITETTURA: PRIMA



ARCHITETTURA: CAMBIAMENTI

1

BACK-END

Transizione da Amethyst a Bevy engine

2

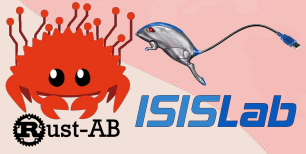
RENDER TRAIT

Separazione della struttura implementativa di Render

3

SUPPORTO MULTI-AGENT

Refactor totale di Rust-AB per permettere lo sviluppo di simulazioni multi-agente



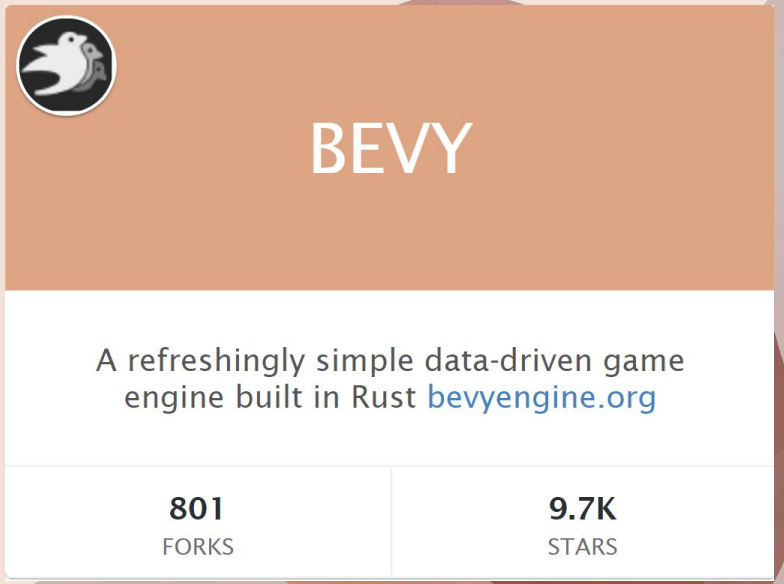
BEVY ENGINE

Cambiamento del motore grafico

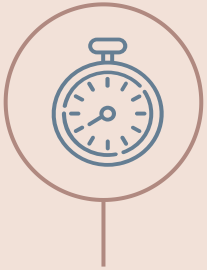
BEVY ENGINE: COS'È?

"Un game engine data-driven e piacevolmente semplice".

- Semplicità senza pagare in prestazioni (anzi!).
- Basato su una struttura solida (wgpu, bevy_ecs, winit, rodio...).
- Focus sull'ergonomia e sulla semplice programmazione.

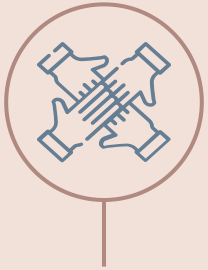


PERCHÈ CAMBIARE BACK-END?



TEMPI DI COMPILAZIONE

Amethyst ha tempi di compilazione eccessivamente lunghi. Bevy engine mira a tenere questi tempi bassi.



SUPPORTO A LUNGO TERMINE

Amethyst è inattivo, probabilmente a causa di problemi architetturali del progetto. Bevy ha uno sviluppatore full-time e altri 250 contributors.

WASM - WebGL

Bevy supporta il target di compilazione WebAssembly, wgpu supporta WebGl, ciò permette di incorporare applicazioni grafiche nel browser.

LANDING PAGE

Simulazioni nel browser a velocità
quasi native grazie a
WebAssembly & WebGL

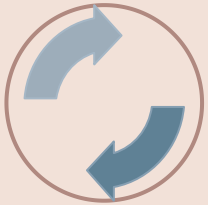
Wasm - WebGL

https://rust-ab.github.io/boids/

CARATTERISTICHE PRINCIPALI

SITO STATICO (ZOLA)

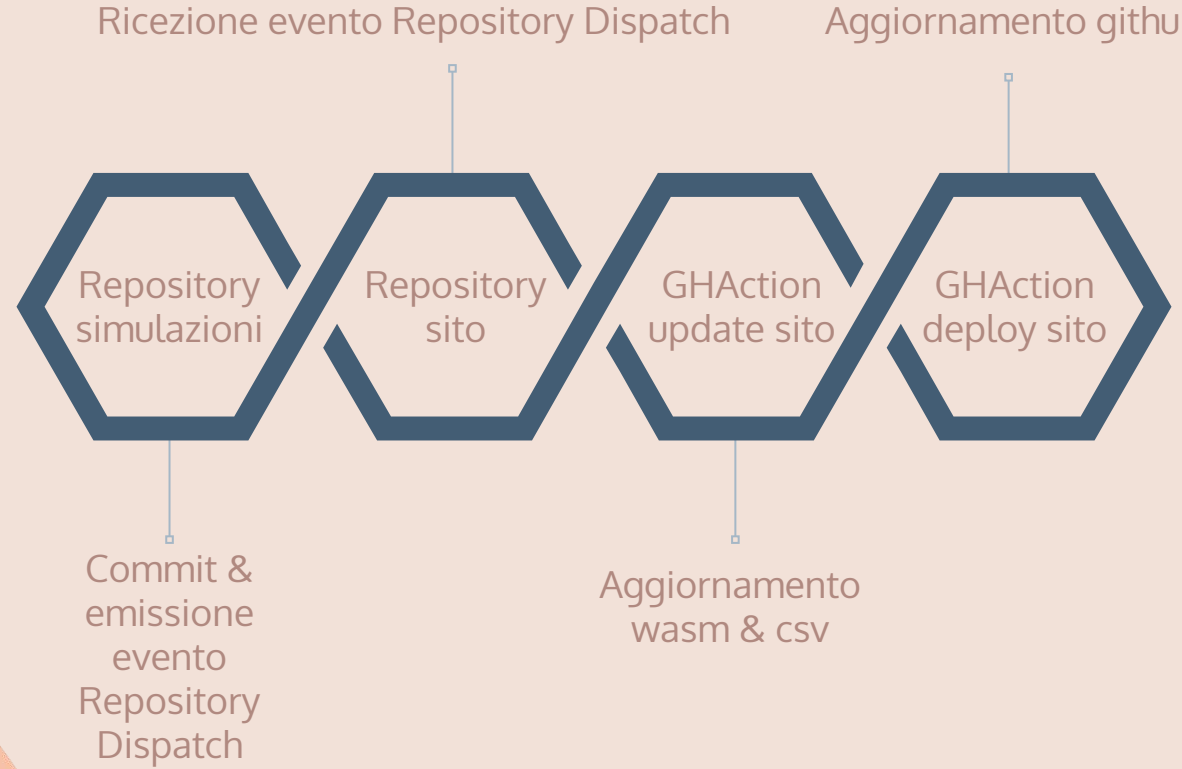
Sito statico generato tramite il generatore di siti statici Zola, scritto in Rust.



AUTO-UPDATE

Tramite una GitHub Action, i binari wasm e le fonti di dati dei grafici vengono tenuti sempre sincronizzati con la repository delle simulazioni di Rust-AB.

FLUSSO GITHUB ACTIONS



BEVY EGUI

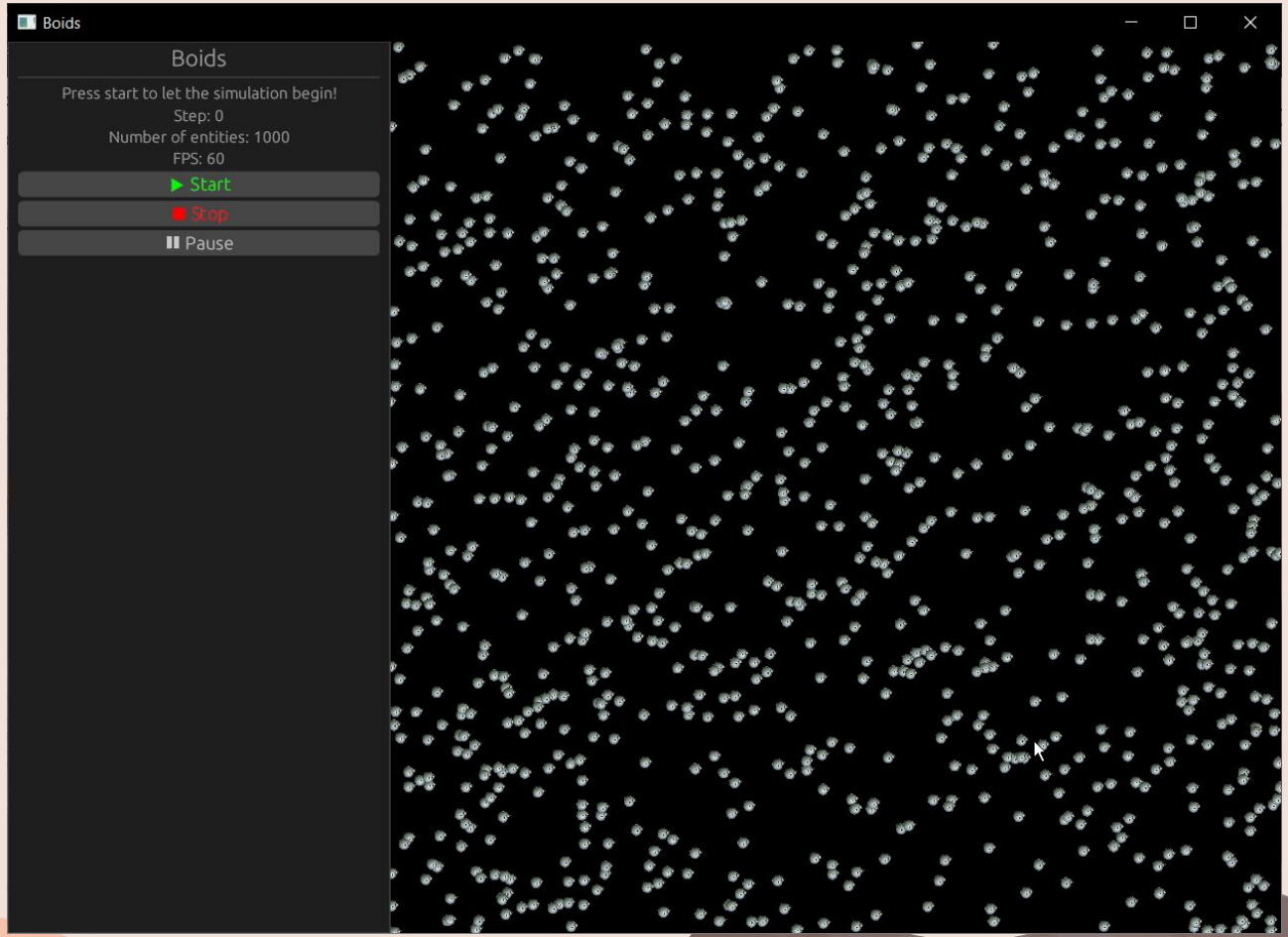
Pannello di controllo per
interagire con la simulazione.

INTERAGIRE CON LE SIMULAZIONI: BEVY EGUI

Attualmente in sviluppo, sui branch bevy_egui e ui_visualization.

Funzionalità di pausa, stop e start già implementate.

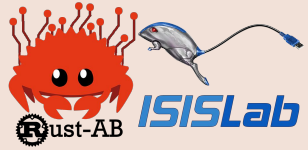
Codice UI isolato in un unico sistema.


Boids
Boids
Press start to let the simulation begin!
Step: 0
Number of entities: 1000
FPS: 60
Start
Stop
Pause

REFACTOR MULTI-AGENT

Supporto per simulazioni con diversi tipi di agenti

MULTI-AGENT: PROBLEMA



Lato engine: la struttura Schedule è generica sul (singolo) tipo dell'agente, per sapere quanto spazio di memoria allocare per il vettore di agenti rappresentante la coda a priorità interna.

- Soluzione: Rimuovere il tipo generico, modificando la coda a priorità in modo che contenga riferimenti ad implementatori dell'aspetto "Agent" (struttura Box di Rust).

Lato visualizzazione: l'implementatore degli aspetti Agent e AgentRender coincidono, causando un alto accoppiamento tra simulazione e visualizzazione.

- Soluzione: due strutture, una che descrive l'agente e una che descrive la sua visualizzazione. (Come accedere alla struttura dell'agente per visualizzarlo?)

MULTI-AGENT: COMPROMESSI

- Boilerplate aggiuntivo: per poter accedere alla struttura di un agente dalla sua struttura visualizzativa, bisogna eseguire un downcast dell'implementatore dell'aspetto "Agent" nel tipo effettivo.

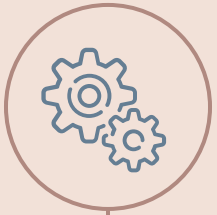
- Perdita di prestazioni: Non conoscendo più i tipi concreti degli agenti nei metodi del framework, molte ottimizzazioni eseguite da Rust vengono perse (es. static dispatching, no inlining...).

SVILUPPI FUTURI

SVILUPPI FUTURI

PRESTAZIONI MULTI-AGENT

Valutare l'impatto sulle prestazioni causato dal dynamic dispatching.



CONFIGURAZIONE INIZIALE

Permettere di variare i parametri della configurazione iniziale di una simulazione dal pannello laterale della visualizzazione.

PARAMETRI AGENTI

Abilitare la modifica dei parametri di singoli agenti nella visualizzazione a run-time.

TABLE OF CONTENTS

Modello di Schelling (Segregation)

- Realizzato da Thomas Schelling negli anni ‘70;
- Simula il fenomeno sociale della segregazione residenziale;
- Due gruppi di agenti occupano gli spazi di una scacchiera;
- Ad ogni step di simulazione gli agenti scelgono se spostarsi in una nuova locazione o meno in base alla percentuale di vicini “simili”

Implementazione su Rust-AB - SchellingState

Lo stato di simulazione è rappresentato dalla struct **SchellingState**, che contiene l'ambiente di simulazione (una griglia bidimensionale) e una lista di locazioni vuote a disposizione degli agenti che devono spostarsi durante gli step di simulazione.

```rust
pub struct SchellingState{
    pub patches: Grid2D<Turtle>,
    pub size: i64,
    pub free_patches: ArrayQueue<Int2D>,
    pub step: usize,
}
```

- `get_num_happy()`
- `get_nearby_turtles(Turtle)`
- `set_turtle_location(turtle, loc)`
- `update()` (trait State)

A causa del Double Buffering due agenti potrebbero scegliere di spostarsi nella stessa locazione;

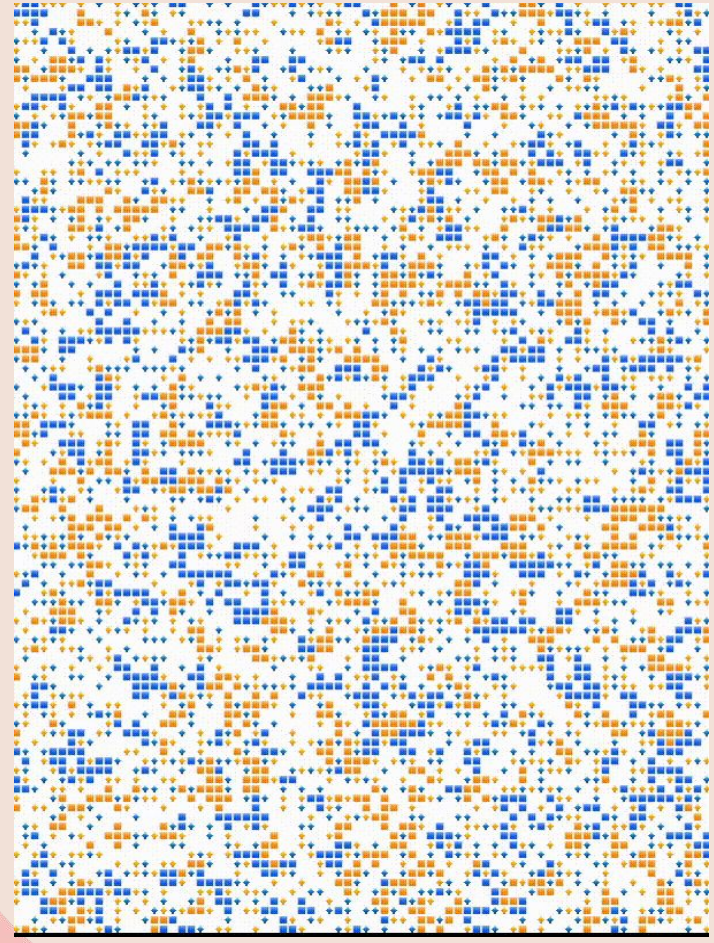
Implementazione su Rust-AB - Turtle

Ogni agente è caratterizzato da un `id`, una posizione, e un colore, che contraddistingue il gruppo di appartenenza (Blue o Orange).
Il valore booleano `happy` esprime se l'agente è soddisfatto nella locazione in cui si trova o meno.

```
pub struct Turtle {
    pub id: u128,
    pub happy: bool,
    pub color: TurtleType,
    pub loc: Int2D,
}
```

Implementazione in Rust-AB - Turtle (Trait Agent)

```rust
fn step(&mut self, _state: &SchellingState) {
    //Get the agent neighbors
    let neighbors = _state.get_nearby_turtles(&self);
    //Count how many neighbors share the same color with the agent
    let mut similar = 0;
    for neighbor in neighbors{
        if neighbor.color == self.color{
            similar+=1;
        }
    }
    //Check if the current percentage of similar neighbors is lower than the desired percentage
    //In this case the turtle is not happy, therefore it finds a new location.
    if similar as f64/8.0 < PERCENT_SIMILAR_WANTED{
        self.happy = false;
        let old_loc = self.loc;
        let new_loc = _state.free_patches.pop().unwrap();
        self.loc = new_loc;
         _state.free_patches.push(old_loc).unwrap();
    }else{
        self.happy = true;
    }
    //The agent's location is set whether it moves to a new location or not
    //in order to sync both its Schedule and SchellingState copies
    _state.set_turtle_location(*self,self.loc);
}
```



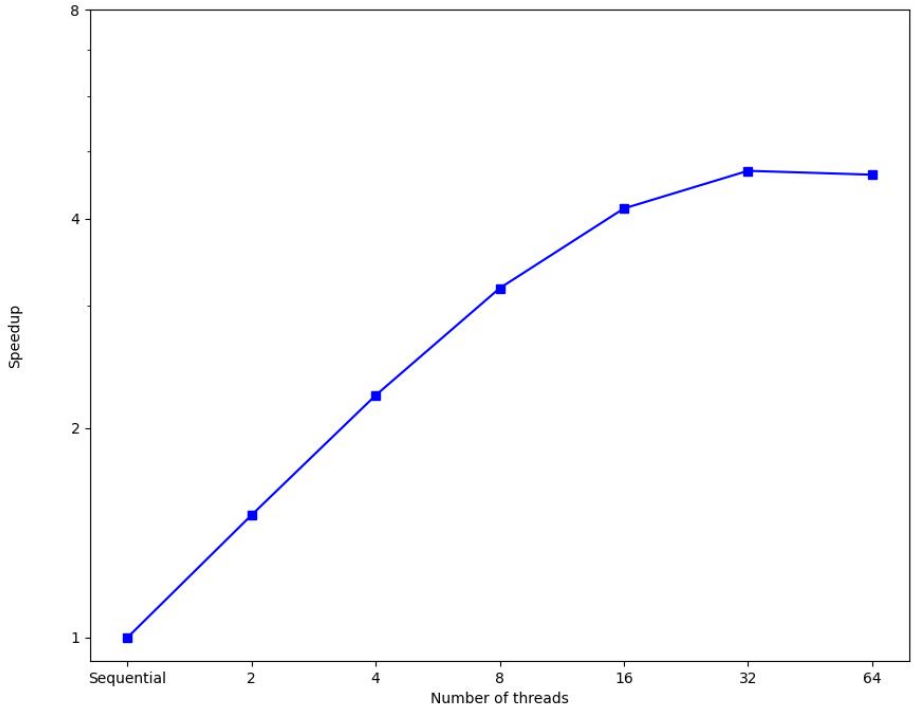
Modello disponibile nella repository rust-ab-examples: https://github.com/rust-ab/rust-ab-examples/tree/schelling

Visualizzazione (da sistemare) disponibile in un branch separato: https://github.com/rust-ab/rust-ab-examples/tree/ui_visualization

05 Scaling Benchmark

Boids & Wolf Sheep Grass

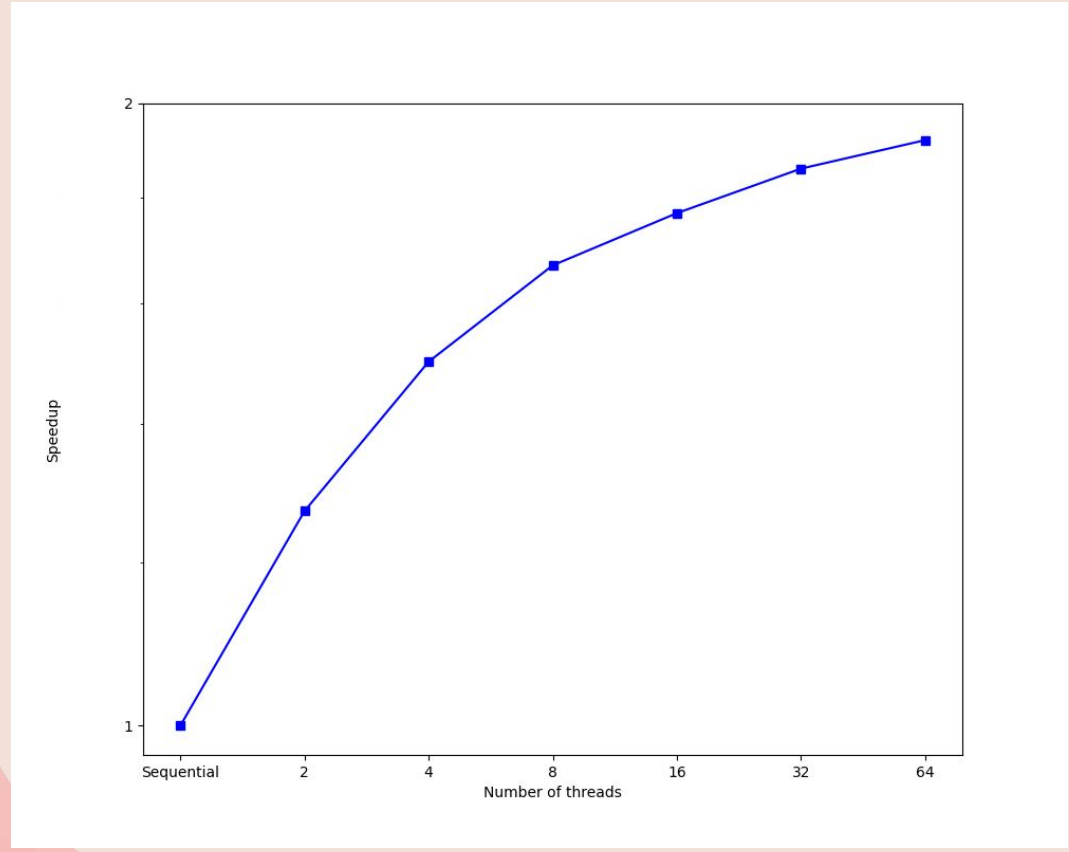
Boids



N. Agenti	1.000.000
Dimensioni	10606*10606
N. Step	100

N. Thread	Speedup
1	1,00
2	1,50
4	2,23
8	3,18
16	4,14
32	4,69
64	4,63

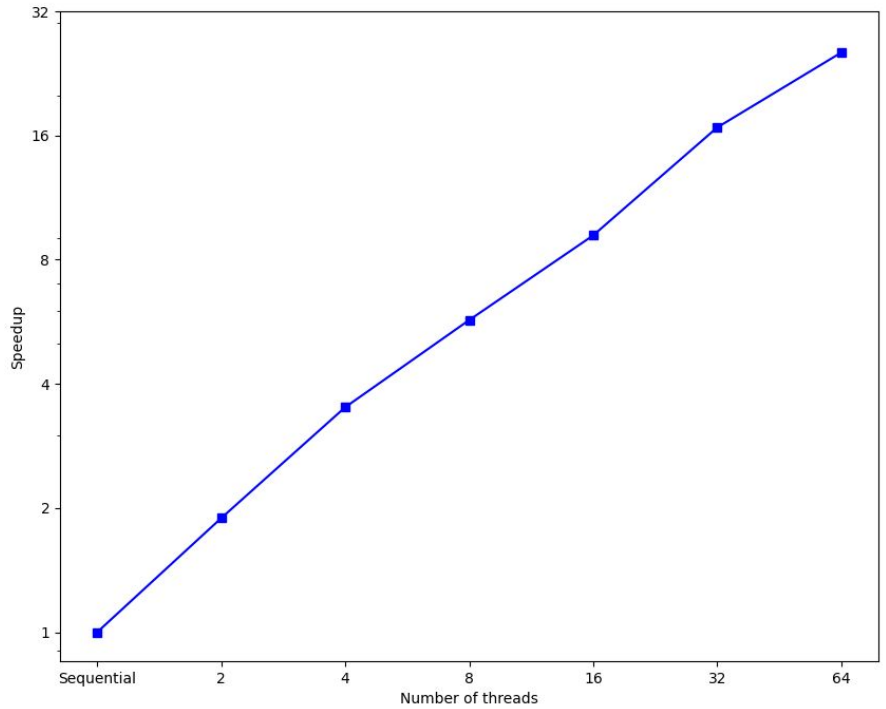
Wolf Sheep Grass



N. Agenti	1.000.000
Dimensioni	5000*5000
N. Step	50

N. Thread	Speedup
1	1,00
2	1,27
4	1,50
8	1,67
16	1,77
32	1,86
64	1,92

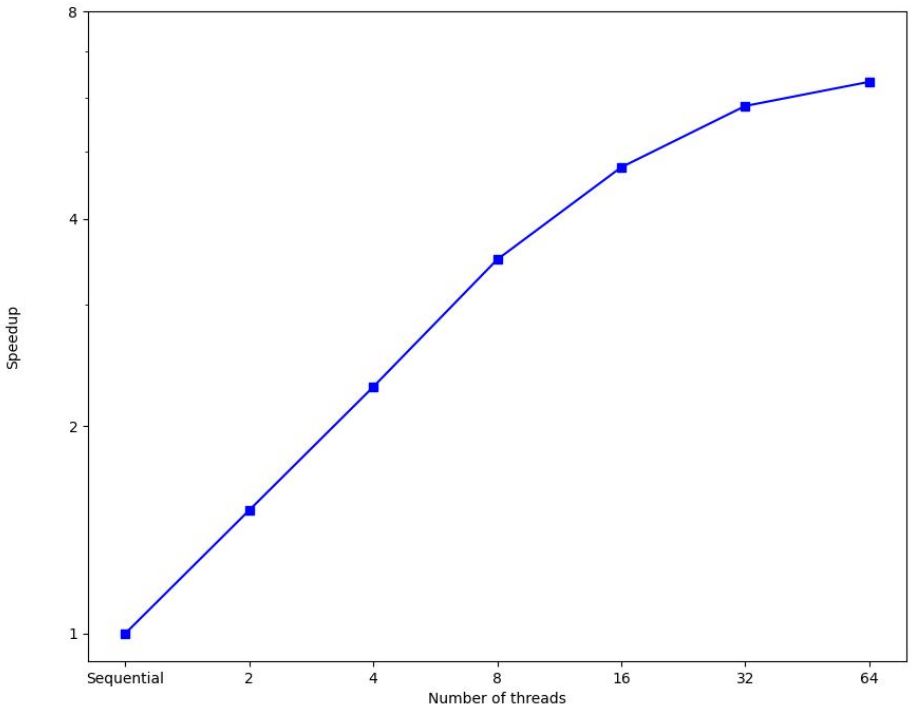
Boids (High Density)



N. Agenti	1.000.000
Dimensioni	500*500
N. Step	100

N. Thread	Speedup
1	1.00
2	1.89
4	3.51
8	5.71
16	9.16
32	16.73
64	25.41

Wolf Sheep Grass (High Density)



N. Agenti	1.000.000
Dimensioni	500*500
N. Step	50

N. Thread	Speedup
1	1.00
2	1.51
4	2.28
8	3.49
16	4.75
32	5.83
64	6.32

Work in Progress

Parallelizzazione
fetch degli agenti

Il tempo di update dei
buffer domina sul tempo
di esecuzione dello step

THANKS

- Main repository : https://github.com/rust-ab/rust-ab
- Esempi: https://github.com/rust-ab/rust-ab-examples
- Talks: https://github.com/rust-ab/talks
- Sito: https://rust-ab.github.io/