Nome Douglas Daisuke | **RA** _____

Nome Gabriel Teston | **RA** _____

Nome Paulo Pacitti | **RA** 185447

MC536 | Trabalho 1: Threads com smart-rockets

Introdução

Inspirado pela ideia de implementar o *Game of Life* do John Horton Conway utilizando threads, nosso grupo decidiu implementar a simulação que se utiliza de algoritmos genéticos conhecida como *Smart Rockets*.

A simulação consiste em uma coleção de foguetes partir de uma mesma origem em direção a um alvo, sendo que cada foguete contém em seu "DNA" uma rota para tentar atingir o alvo. Após os foguetes terminarem de percorrer as suas rotas, é dado uma pontuação baseada na distância final do foguete ao alvo. Depois disso, os foguetes com maiores pontuações realizam um "cruzamento", gerando uma nova população baseada nos DNAs dos foguetes com maior sucesso da geração anterior. A ideia é que a cada geração, foguetes com maior performance (menor distância ao alvo) são gerados a partir desse cruzamento.

Implementação

A implementação feita em C feita pelo grupo é feita em duas partes: uma *lib* que contém as rotinas e entidades da simulação, e um arquivo onde o "loop" de simulação é feito. Temos as seguintes entidades:

- **Rocket**: entidade que define as propriedades do foguete:

```
typedef struct
{
  float x,y;
  float velocity[2];
  float **dna;
  int dna_length;
  float fitness_score;
} Rocket;
```

- **Population**: Define a coleção de foguetes, com o tamanho da coleção e o fator de mutação para cruzamentos.

```
typedef struct
{
  Rocket **rockets;
  int size;
  int mutation_factor;
} Population;
```

- **Board**: O "tabuleiro" onde acontece a simulação.

```
typedef struct
{
  int x0, y0, x1, y1;
} Obstacle;
```

Além das entidades, temos as rotinas necessárias para a simulação acontecer, decritas no *header* `smart-rockets.h` (para saber mais, recomenda-se a análise do código-fonte)

```
/ Rocket creation/destruction
Rocket *newRocket(int dna_length, int init_pos[2]);

Rocket *breed(Rocket *parent_a, Rocket *parent_b, int mutation_factor, int init_pos[2]);

void destroyRocket(Rocket *rocket);

// Population creation/destruction
Population *newPopulation(int size, int dna_length, int initial_position[2], int mutation_fa

Population *nextGeneration(Population *population, int initial_position[2]);

void destroyPopulation(Population *population);

// Board creation/destruction
Board *newBoard(int width, int height, int target[2], int n_obstacles);

void destroyBoard(Board *board);

// Obstacle creation/destruction
Obstacle *newObstacle(int x0, int y0, int x1, int y1);

void destroyObstacle(Obstacle *obstacle);

// Fitness function
float fitness(Board *board, Rocket *rocket);

// Rocket update
void updateRocket(Board *board, Rocket *rocket, int frame_idx);
```

Utilizando as Threads

As *threads* foram utilizadas em seções do código onde a progração paralela poderia oferecer vantagem em relação a programação serial. As seções onde se utilizam *threads* são:

- **Ordenação dos foguetes:** a ordenação dos foguetes com maior perfor-

mance é feito através de um *merge* sort* adaptado, onde a coleção de foguetes é divido em 4 partes e cada uma das partes é ordenada usando o citado algoritmo de ordenação. Após o *join* das *threads*, ocorre a intercalação entre as partes e a coleção de foguetes termina ordenada em relação ao *score*;