UNIVERSIDADE D
COIMBRA

Artificial Inteligence Fundamentals
Lab Assignment 2 - Stage 2

The evolution of Lunar Lander

Members:
Miguel Castela 2022212972, PL1
Miguel Martins 2022213951, PL1
Nuno Batista 2022216127, PL1

March, 2025

Table of Contents

Perceptions

Perception	Name	Description
observation[0]	x	Horizontal position relative to the landing platform. ($x < 0$ if the ship is to the left of the platform and $x > 0$ otherwise)
observation[1]	y	Vertical position, relative to the ground.
observation[2]	v_x	Horizontal velocity. Negative when the ship is going to the left and positive when its going to the right
observation[3]	v_y	Vertical velocity. Negative when the ship is going down and positive when its going up
observation[4]	θ	Ship's orientation ($\theta < 0$ if the ship is tilted to the left of the platform and $\theta > 0$ otherwise)
observation[5]	v_θ	Angular velocity ($v_\theta > 0$ if the ship is rotating counter-clockwise and $v_\theta < 0$ if cloclwise)
observation[6]	RLT	Boolean: $True$ if the right leg is in contact with the ground, $False$ otherwise
observation[7]	LLT	Boolean: $True$ if the left leg is in contact with the ground, $False$ otherwise

Table 1: Perceptions Declaration

Action Declaration

Name	Action	Description
Mm	Controls the main motor.	Activated for values larger than 0.5, linearly increases the acceleration until reaching the maximum value of 1.
Lm	Controls the left motor.	Activated for values larger than 0.5, the left motor is activated, rotating the ship to the right. It's acceleration linearly increases until reaching the maximum value of 1.
Rm	Controls the right motor.	Activated for values lower than -0.5, the right motor is activated, rotating the ship to the left. It's acceleration linearly increases until reaching the maximum value of -1.

Table 2: Actions declaration

Windless Environment

In the first stage of our evolutionary algorithm, we focused on implementing four key components: parent selection, crossover, mutation, and the objective function. Below is a detailed explanation of each mechanism.

Parent Selection

We implemented a selection of strategies to select parents from the population:

- **Roulette Selection**: Roulette selection is a method used to select individuals from the population based on their fitness values. The probability of selecting an individual is proportional to its fitness. Individuals with higher fitness values have a higher chance of being selected.

  To implement this, we first calculate the total fitness of the population. Then, each individual's selection probability is determined by dividing its fitness by the total. A random number is then generated, and one individual is selected based on these probabilities, simulating a spin of a roulette wheel where larger fitness values occupy more space on the wheel.

  This method encourages the selection of fitter individuals while still allowing those with lower fitness a non-zero chance of being selected, which helps preserve diversity in the population. If all individuals have zero fitness, selection defaults to uniform random selection to prevent errors.

- **Tournament of 2**: This method involves randomly selecting two individuals from the population to form a tournament. These individuals are compared based on their fitness values, and the one with the higher fitness is selected as the parent. As this approach always picks the best of the two without any added randomness, it is a purely exploitative strategy.

  While it is simple and computationally efficient, the lack of stochasticity can lead to quicker convergence but also increases the risk of getting trapped in local optima due to reduced diversity in selection.

- **Tournament of 5**: We randomly select five individuals from the population to form a tournament. These individuals are then sorted based on their fitness values in descending order (so that the best-performing individual appears first). To introduce controlled randomness and maintain genetic diversity, we use a probabilistic selection mechanism:

  - The best individual is selected with a probability of P.
  - The second-best is selected with a probability of $P \times (1 - P)$.
  - The third-best is selected with a probability of $P \times (1 - P)^2$.
  - If none of these probabilities match, a random individual from the tournament is selected.

  For the simulations conducted, we set the probability P to 0.9. This method balances exploitation (favoring the best individuals) with exploration (occasionally selecting weaker individuals) to avoid premature convergence.

Crossover

We used a single-point crossover method, where a random index is selected along the genotype—representing the individual's weights. The offspring's genotype is formed by taking the first segment of one parent's genotype (up to the crossover point) and combining it with the remaining segment from the other parent (starting from the crossover point). This technique enables the offspring to inherit characteristics from both parents, encouraging genetic diversity in the following generation. We tested two crossover probabilities: 0.5 and 0.9.

Mutation

We implemented Gaussian mutation to introduce small random variations in the genotype of an individual, preventing the algorithm from converging prematurely. For each gene in the genotype, there is a probability ($PROB\_MUTATION$) that the gene will be modified. If selected for mutation, a random value drawn from a Gaussian distribution centered at 0 with a given standard deviation (STD_DEV) is added to the gene. To ensure that gene values remain within acceptable limits, the mutated value is clamped to stay within the range [-1, 1]. This mutation mechanism helps maintain genetic variation and allows the algorithm to explore new areas of the search space. We tested two mutation probabilities: 0.008 and 0.05.

Evolution of the objective function and score progression

The objective function is designed to evaluate the performance of the lunar lander in terms of its landing accuracy and safety.
Firstly, we added a **penalty for the ship's vertical velocity**:

$$\text{return } (-|x| - |y| - |v_y|, \text{check\_successful\_landing(observation)}) \tag{1}$$

Independent of the Mutation, Crossover rate and Tournament type used, this resulted in a 0.0 hitrate and a mild negative fitness (≈ -0.03), so we added a **reward when the ship has both legs touching the ground** (*both_legs_touching*). This was not enough to drive up the hitrate (it only caused a 2% increase), so we also added an **angular velocity penalty**:

$$\begin{aligned} &\text{legs\_touching} = (\text{contact\_left} = 1 \land \text{contact\_right} = 1) \\ &\text{return } (-|x| - |y| + (\text{legs\_touching}) - |v_y| - |v_\theta|, \text{check\_successful\_landing(observation)}) \end{aligned} \tag{2}$$

This change resulted, in the best scenario (Prob_Crossover = 0.9, Prob_Mutation = 0.05 and a Tournament of 5) in a 30% hitrate, wich showed us that the ship was able to learn to land, but not in a consistent way.
We then reassessed our approach and tried adding penalties and rewards inspired by our production system implementation, rewarding the more "specific" productions with higher rewards, and the more "general" productions with lower rewards. This resulted in a final generation with a fitness of about 101.

$$\begin{aligned} &\text{stable\_orientation} = |\theta| < 0.15, \quad \text{on\_landing\_pad} = |x| \leq 0.2, \\ &\text{stable\_velocity} = v_y > -0.2, \quad \text{stable} = \text{stable\_velocity} \land \text{stable\_orientation}, \\ &\text{legs\_touching} = (\text{contact\_left} = 1 \land \text{contact\_right} = 1), \\ &\text{one\_leg\_touching} = (\text{contact\_left} = 1 \lor \text{contact\_right} = 1), \\ &\text{good\_height} = (y < 2.1), \quad \text{stable\_orientation\_int} = (|\theta| < 0.15), \\ &\text{stable\_vertical\_velocity} = (|v_y| < 0.3), \quad \text{stable\_angular\_velocity} = (|v_\theta| < 0.3), \\ &\text{stable\_horizontal\_velocity} = (|v_x| < 0.2), \quad \text{stable\_x} = (|x| < 0.2), \\ &\text{perfect\_x} = (|x| = 0), \quad \text{too\_far\_x} = (|x| > 0.1), \end{aligned} \tag{3}$$

$$\begin{aligned} \text{return}\Big(&- |x| \cdot 100 + \text{legs\_touching} \cdot 100 + \text{one\_leg\_touching} \cdot 50 \\ &+ \text{good\_height} \cdot 10 + \text{stable\_velocity} \cdot 100 + \text{stable\_orientation} \cdot 10 \\ &+ \text{stable} \cdot 100 + \text{on\_landing\_pad} \cdot 100 + \text{stable\_orientation\_int} \cdot 10 \\ &+ \text{stable\_vertical\_velocity} \cdot 10 + \text{stable\_angular\_velocity} \cdot 50 \\ &+ \text{stable\_horizontal\_velocity} \cdot 10 + \text{stable\_x} \cdot 10 \\ &+ \text{perfect\_x} \cdot 100 - \text{too\_far\_x} \cdot 100, \\ &\text{check\_successful\_landing(observation)}\Big) \end{aligned} \tag{4}$$

This resulted in an average of 0.01% hitrate (occasionally it was a little higher), so we started working on branching and binary conditions.

So, we started to define **3 different branches**, one to make the ship stable (it had to have a stable orientation and stable velocity, based on the thresholds of the prevoius function), then one to make it land on the landing pad, and one to make it land safely. Landig safely was a challenge, because we started noticing the ship would often hover the ground without actually landing, so we added a **penalty for when the ship is slowly hovering in the range of the two flags, but its legs are not touching the ground**. These changes resulted in a final fitness of about 1449.

$$\text{legs\_touching} = (c_L = 1 \land c_R = 1)$$

$$\text{hover\_penalty} = \begin{cases} -100, & \text{if } |x| < 0.2 \land |v_x| < 0.2 \land \neg(\text{legs\_touching}) \\ 0, & \text{otherwise} \end{cases}$$

We decided to maintain the penalties for the absolute values of the ship's position. We also maintaned the *legs_touching*, *stable_orientation*, *stable_with_less_inpact*, *on_landing_pad* and *stable_velocity* **rewards mentioned above**.

To help make the punishment of the ship more precise, we implemented a new set of penalties based on the ship's position, velocity, and orientation. These penalties are designed to discourage dangerous behaviors during descent while rewarding controlled and precise landings. These include penalties for excessive angular velocity, high vertical velocity when close to the ground, and poor horizontal positioning. The penalties are applied conditionally based on the ship's state, allowing for more nuanced control over its behavior:

$$\text{very\_near\_ground} = y < 0.2$$

$$\text{height\_factor} = 1 + \frac{2}{\max(0.1, y)}$$

$$\text{near\_ground} = y < 0.5$$

$$\text{centered\_position} = |x| < 0.2$$

$$\text{stable\_orientation} = |\theta| < 0.15$$

$$\begin{aligned} \text{penalties} = & -800 \cdot (|\theta| > 0.6) \\ & -500 \cdot (|v_\theta| > 1.0) \\ & -1000 \cdot (v_y < -1.0 \land y < 1.0) \\ & -800 \cdot (|x| > 0.8) \cdot \text{height\_factor} \\ & -400 \cdot (|v_x| > 0.8) \end{aligned}$$

$$\text{if very\_near\_ground} \land \text{centered\_position} \land \text{stable\_orientation} \land |v_y| < 0.05 :$$
$$\quad \text{penalties} \mathrel{-}= 300$$

While trying different ways of doing these branches and with this hovering decision made, paired with testing with different sets of parameters made the hitrate hover about 8.5% with ocasional inconsistent bumps to about 80%. A little better, but the ship was't landing consistently.

This made us decide to **keep the stabilty branch and make the positioning branch much stronger, while enhancing the descent rate branch with high priority before the landing component**. This new branch reward a slower descent, this is done by checking if the descent was a safe_descent ($vy > -0.5$), rewarding it, a very_safe_descent ($vy > -0.2$), rewarding it even more, and punishing the ship if it had a high vertical velocity. This resulted in a very slight increase of the hitrate.

$$\text{stable\_orientation} = |\theta| < 0.15$$
$$\text{stable\_angular\_velocity} = |v_\theta| < 0.3$$
$$\text{stable\_horizontal\_velocity} = |v_x| < 0.2$$
$$\text{centered\_position} = |x| < 0.2$$
$$\text{stability\_score} = 500 \cdot \text{stable\_orientation} + 300 \cdot \text{stable\_angular\_velocity} + 200 \cdot \text{stable\_horizontal\_velocity}$$
$$\text{positioning\_score} = \Big(400 \cdot \text{centered\_position} + 300 \cdot (1 - \min(1, \tfrac{|x|}{0.5})) + 200 \cdot (1 - \min(1, \tfrac{|v_x + 0.5x|}{0.5}))\Big) \cdot \text{height\_factor}$$

Given the inconsistent and stochastic nature of the problem, we started to evaluate the fitness of each individual based on the average performance over 5 test runs . This helps test the ability for the agent to generalize. This approach also prevents genotypes that perform well on just one specific terrain to be passed into new generations and being cloned in future generations (given that this can misguide the evolutionary process and lead to less general, overfitted solutions). These results were avaluated in the section "Adding consistency to the fitness function".

$$\text{total\_fitness} = 0$$
$$\text{success\_count} = 0$$
$$\text{num\_trials} = \text{NUM\_EVALS}$$
$$\text{for } i = 1 \text{ to num\_trials}:$$
$$\quad (\text{score}, \text{success}) = \text{simulate}(\text{ind.genotype}, \text{seed} = \text{None}, \text{env})$$
$$\quad \text{total\_fitness} \mathrel{+}= \text{score}$$
$$\quad \text{success\_count} \mathrel{+}= \text{int}(\text{success})$$

$$\text{ind.fitness} = \frac{\text{total\_fitness}}{\text{num\_trials}}$$
$$\text{ind.success\_rate} = \frac{\text{success\_count}}{\text{num\_trials}}$$

We kept the parameters stated above (Prob_Crossover = 0.9, Prob_Mutation = 0.05 and a Tournament of 5) given that these were the best performing when doing the benchmarks (as you can see in the table below) , after adjusting the thresholds, **increasing penalties for excessive velocity, poor positioning, and unstable hovering, while rewarding behaviors that contributed to a stable descent,** we achieved a hit rate of 78.1%. The biggest contributors to this massive change were the

- **Conditional logic for near ground,** diminiscing the rewards of the landing velocity based on the height of the ship:

$$\text{legs\_touching} = (c_L = 1 \wedge c_R = 1)$$
$$\text{near\_ground} = y < 0.5$$
$$\text{very\_near\_ground} = y < 0.2$$
$$\text{landing\_score} = \begin{cases} 500 \cdot \text{legs\_touching} + 200 \cdot \text{one\_leg\_touching} + 400 \cdot \text{stable\_vertical\_velocity} + 400 \cdot \text{centered\_position} + \\ \quad 800 \cdot (-0.2 < v_y < -0.05) + 1500 \cdot \text{legs\_touching}, & \text{if ne} \\ 0, & \text{othe} \end{cases}$$

- **Increased rewards for slow descents, resulting in perfect landings,** boosting the reward significantly for touching down correctly and having a good descent rate during touchdown (the new *descent_score* also helped).

$$\text{height\_factor} = 1 + \frac{2}{\max(0.1, y)}$$
$$\text{safe\_descent\_rate} = v_y > -0.5$$
$$\text{very\_safe\_descent} = v_y > -0.2$$
$$\text{descent\_score} = \Big(400 \cdot \text{safe\_descent\_rate} + 600 \cdot \text{very\_safe\_descent} + 300 \cdot \Big(1 - \min\Big(1, \tfrac{|v_y|}{1.0}\Big)\Big)\Big) \cdot \text{height\_factor}$$

After defining the penalties and rewards, we added a final score function that combines all the previous components. This function evaluates the ship's performance based on its stability, positioning, descent rate, and landing success. In the final generation these changes resulted fitness of 151739. The final score is calculated as follows:

$$\begin{aligned}&\text{total\_score} = \text{stability\_score} + \text{positioning\_score} + \text{descent\_score} \\ &\quad + \text{landing\_score} \cdot (\text{stable\_orientation} \cdot \text{centered\_position} \cdot \text{safe\_descent\_rate}) + \text{penalties} \\ &\text{return}(\text{total\_score}, \text{check\_successful\_landing}(\text{observation}))\end{aligned}$$

Final score and statistic tests

Final objective function

The final objective function is a combination of the previous functions, with the addition of a penalty proportional to the number of steps taken in order to favor individuals that complete the task faster. We changed the crossover probability to 0.5, kept the mutation probability at 0.008 and the tournament type at 5, as these continued to be the best performing when doing the benchmark table tests. The final objective function has this code added:

$$\text{steps\_penalty} = -0.1 \cdot (\text{STEPS} - 1)$$
$$\text{penalties} += \text{steps\_penalty}$$
$$\text{total\_score} = \text{stability\_score} + \text{positioning\_score} + \text{descent\_score} + \text{landing\_score} \cdot (\text{stable\_orientation} \cdot \text{centered\_position} \cdot \text{safe\_descent\_rate}) + \text{stopped\_reward}$$
$$\text{total\_score} += \text{penalties}$$

	Mutação	Crossover	Elitismo	Tamanho da População	Gerações
Experiência 1	0.008	0.5	0	100	100
Experiência 2	0.05	0.5			
Experiência 3	0.008	0.9			
Experiência 4	0.05	0.9			
Experiência 5	0.008	0.5	1		
Experiência 6	0.05	0.5			
Experiência 7	0.008	0.9			
Experiência 8	0.05	0.9			

Table 3: Configuration of the experiments for different parameters.

Experiência	Hitrate	Score Médio
1	15.4%	56653.992
2	92.5%	141230.92
3	18.3%	61151.43
4	96.9%	146582.12
5	20.5%	49808.66
6	97.2%	148110.92
7	21.0%	60213.836
8	97.9%	148250.92

Table 4: Results of the experiments with different parameters.

Adding consistency to the fitness function

Before evaluating fitness based on the average of 5 simulations (a decision stated above), the fitness function evolved in a highly inconsistent manner, with abrupt jumps and fluctuations. (This is illustrated in the graph, where fitness progresses erratically.)

After introducing the averaging over 5 simulations, although the fitness curve remains

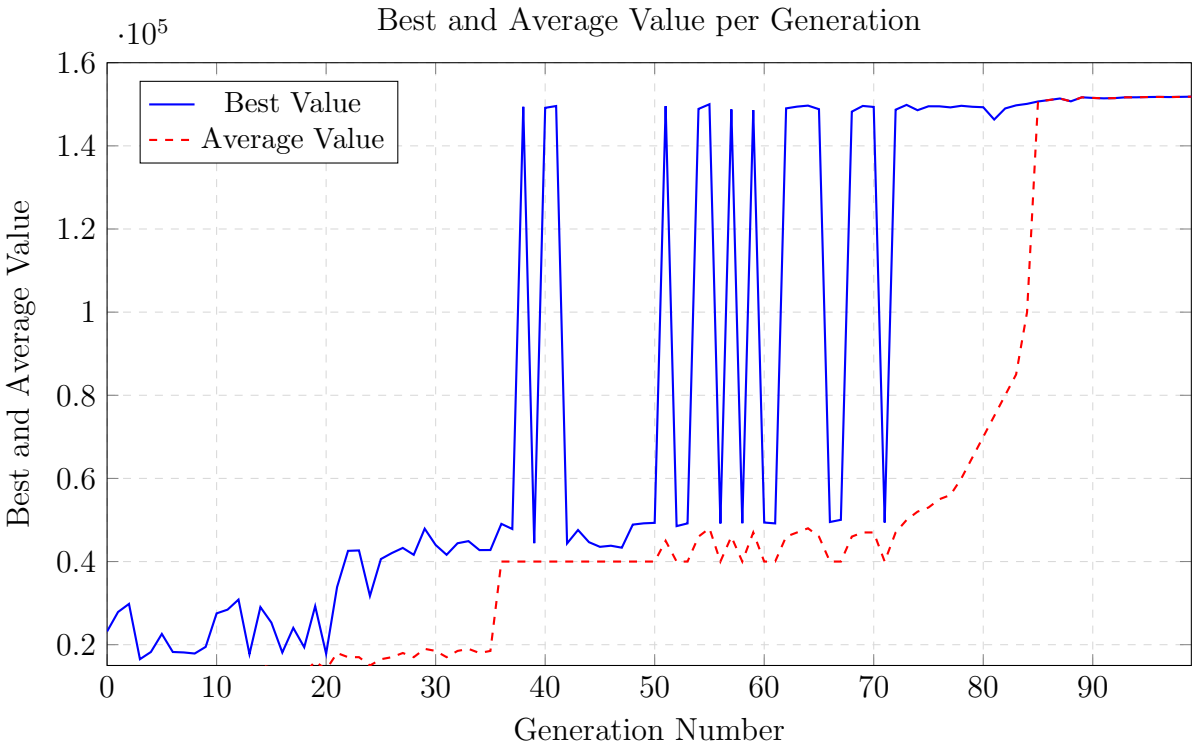

Figure 1: Best and Average Value per Generation

somewhat wobbly due to the stochastic nature of the problem, it becomes significantly more consistent and closer to being monotonically increasing.

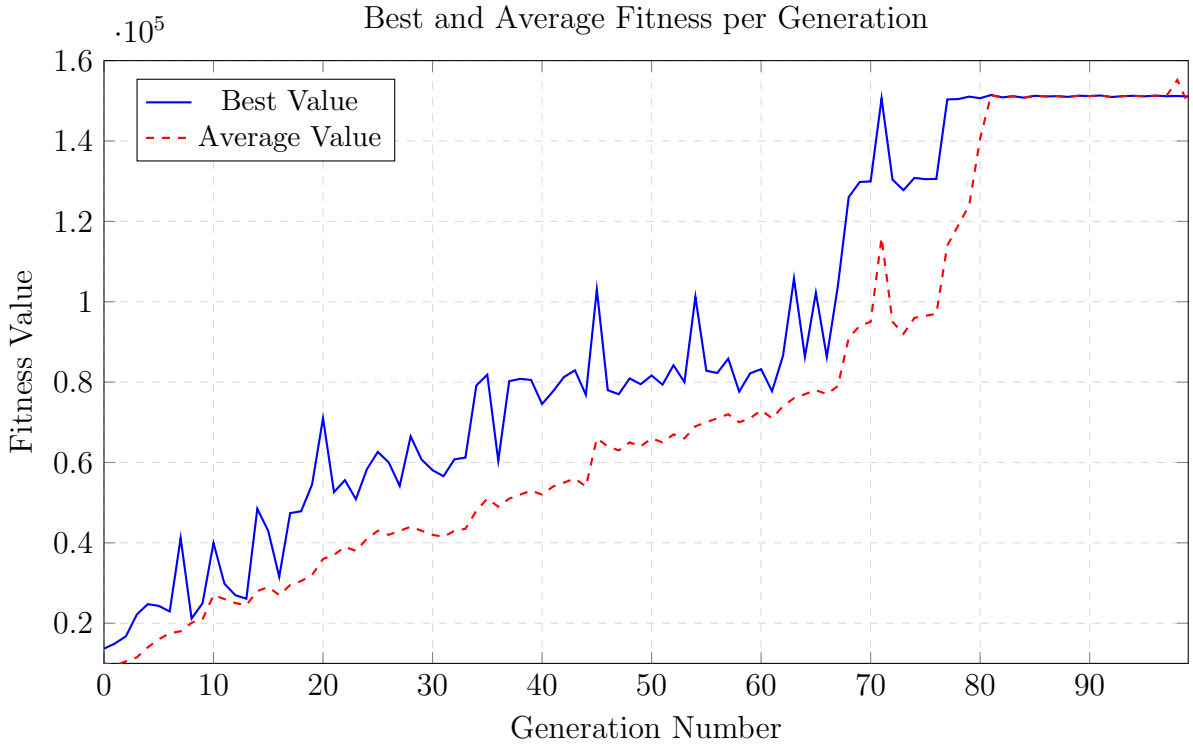

Figure 2: Best and Average Value per Generation (5 trials)

Decisions and their impact

With elitism:
We have greater exploitation and less exploration. Good individuals are preserved, which often lead to better results overall.

Without elitism:
Although we get greater diversity among individuals, there's a risk of losing good solutions. We explore the solution space more thoroughly, which makes it harder to get stuck in local minima.

Bigger mutation probability:
A higher mutation rate allows for better exploration of the solution space. It increases the chances of producing more exploratory and bold individuals, potentially leading to novel and diverse solutions.

Bigger crossover probability:
Similarly, a higher crossover probability can be beneficial for preserving genotypes that have already proven to be effective, by combining successful traits from different individuals.

Final thoughts

After extensive experimentation and fine-tuning, we managed to achieve a hitrate of 99%. However, this success is not entirely consistent, as the hitrate occasionally drops in some runs. This inconsistency suggests that while the objective function and evolutionary parameters are effective, the algorithm may require additional generations to stabilize and consistently reach optimal solutions.

The limitation of running only 100 generations appears to be a significant factor. With more generations, the population would have additional opportunities to refine solutions, potentially eliminating the inconsistencies observed. Future work could focus on increasing the number of generations and further analyzing the impact of population size and mutation rates on the stability of the hitrate.

Wind Environment

The wind environment is a more complex and challenging scenario for the lunar lander. In this environment, the ship is subjected to wind forces that can significantly affect its trajectory and landing performance. It is crucial for the agent to adapt its control strategies accordingly. Due to the effectiveness of the parameters (Prob_Crossover = 0.9, Prob_Mutation = 0.05 and a Tournament of 5) in the windless enviroment, we kept in our experiments in this fase.

Evolution of the objective function and score progression

The starting point was the final objective function from the previous section (non-wind environment), which was already quite effective (around 20% hitrate). However, we made several modifications to enhance the agent's performance in windy conditions. The key changes are as follows:

- A new factor, `stability_multiplyer`, is introduced in the new function to account for wind conditions. The multiplier is set to 10 if the `ENABLE_WIND` flag is `True`, which increases the importance of stability during windy conditions. If wind is not enabled, the multiplier defaults to 1. This multiplier affects the `stability_score`, amplifying the contribution of stability when wind is present. This new factor ensures that the agent prioritizes stability even more in windy environments.

$$\text{stability\_multiplier} = \begin{cases} 10, & \text{if ENABLE\_WIND = True} \\ 1, & \text{otherwise} \end{cases}$$

$$\begin{aligned} \text{stability\_score} = \big(&500 \cdot \text{stable\_orientation} + 300 \cdot \text{stable\_angular\_velocity} \\ &+ 300 \cdot \text{stable\_horizontal\_velocity}\big) \cdot \text{stability\_multiplier} \end{aligned}$$

- The rest of the score components, namely `positioning_score`, `descent_score`, and `landing_score`, are all influenced by the `height_factor`, which increases their importance as the agent approaches the ground.

$$\text{height\_factor} = 1.0 + \frac{2.0}{\max(0.1, y)}$$

$$\begin{aligned} \text{stability\_score} = \big(&500 \cdot \text{stable\_orientation} + 300 \cdot \text{stable\_angular\_velocity} \\ &+ 300 \cdot \text{stable\_horizontal\_velocity}\big) \cdot \text{stability\_multiplier} \end{aligned}$$

$$\begin{aligned} \text{positioning\_score} = \big(&400 \cdot \text{centered\_position} + 300 \cdot \big(1.0 - \min(1.0, \tfrac{|x|}{0.5})\big) \\ &+ 200 \cdot \big(1.0 - \min(1.0, \tfrac{|vx + 0.5 \cdot x|}{0.5})\big)\big) \cdot \text{height\_factor} \end{aligned}$$

$$\begin{aligned} \text{descent\_score} = \big(&400 \cdot \text{safe\_descent\_rate} + 600 \cdot \text{very\_safe\_descent} \\ &+ 300 \cdot \big(1.0 - \min(1.0, \tfrac{|vy|}{1.0})\big)\big) \cdot \text{height\_factor} \end{aligned}$$

- The reward for a perfectly stable and landed agent has been enhanced. Specifically, if the agent is very near the ground, centered, and stable, the function rewards the agent for not hovering or being stationary after landing. This is done through the `stopped_reward`, which is assigned a very high value if the agent is stable with no movement near the ground. This prevents endless hovering, as it ensures that the agent is penalized for excessive vertical and horizontal movement after reaching a stable landing position. This reward structure encourages the agent to stay still and

stable after landing, providing a significant reward for complete stability in windy environments.

```
landing_score = 0
stopped_reward = 0

if near_ground:
    landing_score = 500 · legs_touching + 200 · one_leg_touching+
        400 · stable_vertical_velocity + 400 · centered_position

    if very_near_ground ∧ centered_position ∧ stable_orientation ∧ stable_horizontal_velocity ∧ stable_vertical_velocity:
        stopped_reward = 1000 · legs_touching + 500 · one_leg_touching+
            300 · stable_vertical_velocity + 100000 · int(stopped_horizontal_velocity ∧ stopped_vertical_velocity ∧ legs_touching)+
            1000 · stopped_horizontal_velocity + 1000 · stopped_vertical_velocity−
            20000 · |vy| − 1000 · |vx|
```

We kept the penalties for dangerous situations and other unaltered elemets present in the last windless fitness function (like the steps penalty and the 5 run average decision). We managed to achieve a hitrate of 96% in the best individuals, with a score of 151828.625. The ship was able to land successfully in windy conditions, demonstrating the effectiveness of the modifications made to the objective function. As expected, this is inconsistent and the hitrate can drop to about 75% in some runs, even with the best parameters.
The final objective function for the wind environment is as followw:

```
total_score = stability_score + positioning_score + descent_score
    + landing_score · (stable_orientation · centered_position · safe_descent_rate) + stopped_reward
total_score+ = penalties
```

Statistical tests

	Mutação	**Crossover**	**Elitismo**	**Tamanho da População**	**Gerações**
Experiência 1	0.008	0.5	0	100	100
Experiência 2	0.05	0.5			
Experiência 3	0.008	0.9			
Experiência 4	0.05	0.9			
Experiência 5	0.008	0.5	1		
Experiência 6	0.05	0.5			
Experiência 7	0.008	0.9			
Experiência 8	0.05	0.9			

Table 5: Configuration of the experiments for different parameters.

Experiência	Hitrate	Score Médio
1	87.2%	43107.99125
2	92.5%	125065.035
3	80.4%	45439.8446
4	88.2%	88323.4842
5	80.5%	105087.587
6	90.4%	111835.2728
7	82.0%	48583.654
8	89.3%	97917.3214

Table 6: Results of the experiments with different parameters.

Evaluating Mutation Standard Deviation's Impact

Given the harder task of training the ship in a windy enviiroment, we started exploring other ways to make our hit rate better and more consistent. Increasing the `STD_DEV` value from 0.1 to 0.5 significantly enhances the evolutionary algorithm's exploratory capabilities. In comparison with the 0.5 standard deviation results in the table , when we first ran the 10 benchmark tests we gor the results:

Experiência	Hitrate	Score Médio
1	10.2%	43107.99125
2	76.5%	125065.035
3	11.4%	45439.8446
4	47.2%	88323.4842
5	57.5%	105087.587
6	61.4%	111835.2728
7	11.0%	48583.654
8	50.26%	97917.3214

Table 7: Results of the experiments with different parameters.

With only 100 generations available, the solution space must be traversed more aggressively to discover promising regions. A higher standard deviation allows mutated genes to make larger jumps in the parameter space, enabling individuals to explore distant regions that might contain optimal solutions. This aggressive exploration is especially critical in our lunar landing scenario where the fitness landscape contains many local optima. The experimental results confirm this intuition, as configurations with higher mutation rates consistently outperformed more conservative approaches, achieving up to 97.9% hitrate compared to 10-15% with lower mutation parameters.

Final thoughts

After extensive refinement of our objective function for the wind environment, we achieved a peak hitrate of 95%. However, this performance demonstrates notable inconsistency across multiple runs, with hitrates sometimes dropping significantly. This variability reveals that while our enhanced stabilization component provides critical guidance in windy conditions, the evolutionary process likely requires more generations to discover

consistently robust solutions. The increased complexity introduced by wind forces demands more sophisticated control strategies that can be difficult to evolve within just 100 generations.

Conclusion

Examining the experimental results, it becomes evident that parameter configurations favoring high exploration (particularly the higher mutation rate of 0.05) significantly outperform those prioritizing exploitation. In both windless and windy environments, experiments with higher mutation probabilities consistently achieved superior hitrates (reaching 97.2% and 76.5% respectively compared to the modest 15-18% of lower mutation configurations). Like mentioned before, this dramatic performance gap likely stems from the relatively limited 100 generation constraint imposed on the evolutionary process. With such a restricted evolutionary timeline, the algorithm must rapidly explore the solution space rather than gradually refining promising solutions. High-exploration parameter sets allow the population to quickly discover diverse regions of the search space, avoiding premature convergence to suboptimal solutions or local minima.
This suggests that when we have a limited generation count, emphasizing exploration through higher mutation rates becomes critical to achieving satisfactory results before the evolutionary process terminates.