Relatório da Implementação de evolução de robôs em labirintos

João Victor Oliveira Couto & Leandro Pereira Sampaio

[1]Engenharia de Computação – Universidade Estadual de Feira de Santana (UEFS)
Caixa Postal 252 e 294 – 44.036-900 – Feira de Santana – BA – Brazil

[2]Departamento de Ciências Exatas – Universidade de Feira de Santana (UEFS)
Feira de Santana, BA – Brazil.

jictyvoo.ecomp@gmail.com, leandrosampaio827@gmail.com

Abstract. *This is a report about a system developed by two Computer Engineer Students of Universidade Estudal de Feira de Santana. Will be presented here, the steps for the developing, and the decisions taken in all project development.*

Resumo. *Este é um relatório sobre um sistema desenvolvido por dois Estudantes de Engenharia de Computação da Universidade Estadual de Feira de Santana. Serão apresentados aqui, as etapas para o desenvolvimento e as decisões tomadas em todo o desenvolvimento do projeto.*

1. Introdução

A limitação do ser humano é algo que está atrelado com toda a evolução do mesmo desde seus primórdios. Desde o período rupestre até a contemporaneidade, o *homo sapiens* criou vários instrumentos para auxiliar seu dia-a-dia, como lanças, carros, tratores, dentre outros.

Desde o inicio da corrida espacial, e até mesmo antes, fez-se necessário para o ser humano alcançar regiões cada vez mais inóspitas e de difícil acesso para o mesmo. Dessa forma, começaram a surgir os primeiros robôs, controlados remotamente. Eles abriram uma nova gama de possibilidades para a humanidade.

Logo após o surgimento dos robôs, outro problema surgiu na evolução humana, o do controle dos robôs, pois buscou-se ir a locais cada vez mais distantes e profundos, locais esses que o tempo de envio das mensagens superava o ideal necessário para realizar um controle efetivo das máquinas de forma remota.

Dessa forma passou-se a realizar pesquisas sobre a aplicação da computação artificial, ou seja, da computação evolutiva, nos mesmos, de forma que os robôs possam se tornar autônomos o suficiente para que as barreiras da distância e tempo de comunicação possam ser trespassadas.

Esse trabalho visa abordar a criação de um algoritmo de computação evolutiva, para que possa ser realizado o treinamento de um robô para que o mesmo encontre o final de um labirinto de forma autônoma.

2. Metodologia

O sistema foi desenvolvido utilizando a linguagem de programação Python, de forma colaborativa, através do sistema GitHub, utilizando a IDE PyCharm Professional como ambiente para edição e teste do sistema desenvolvido. Nenhuma outra biblioteca foi utilizada senão a padrão da própria linguagem.

2.1. Projeto do algoritmo genético

Neste tópico serão abordados os processos e abordagens utilizados para a criação do algoritmo discutido.

2.1.1. O processo de geração da população inicial

A população inicial é gerada de forma randômica, tendo o valor de cada posição do vetor, correspondente ao cromossomo dos indivíduos, tendo 4 possíveis valores:

- UP
- RIGHT
- DOWN
- LEFT

A população inicial gerada possui 50 indivíduos, todos gerados de forma randômica. Além disso, o vetor contendo os genes de cada individuo possuem tamanho fixo, necessitando dessa forma serem alterados manualmente.

2.1.2. Função de avaliação (fitness)

A função de avaliação do fitness de cada indivíduo foi desenvolvido ao longo da execução do mesmo. De forma que, sempre que o organismo consegue se mover, ou seja, não bate na parede, o Fitness é aumentado em 1, caso contrário o mesmo é diminuído em 5. Caso o organismo encontre o final do labirinto, o seu fitness é incrementado em 100. No final da execução do cromossomo do indivíduo, o fitness do mesmo é diminuído em 10 vezes a distância que está do final do labirinto. O código contendo o trecho de execução do genoma e a atualização do fitness pode ser visto na figura 1.

2.1.3. Seleção para o cruzamento (procedimentos de seleção)

Para realizar a seleção dos indivíduos que iriam realizar o cruzamento para perpetuar os cromossomos das gerações futuras, foi escolhido como forma de seleção dos progenitores, somente os dois indivíduos que possuíssem o melhor fitness.

2.1.4. Crossover (cruzamento)

O Método de cruzamento utilizado para fazer o crossover e gerar os filhos foi o do ponto de corte, no qual é selecionado randomicamente um ponto de corte para cortar o cromossomo da mãe e do pai. Dessa forma, para cada execução do algoritmo de crossover, são

```
def move(self, organisms):
    for organism in organisms:

        count = 0
        for genome in organism.getGenome():
            if organism.getState() == self.__stateDecoder['alive']:
                position = organism.getPosition()
                has_moved = self.__labyrinth.move(self.__genomeDecoder[genome], position)
                if has_moved:
                    organism.updateFitness(1)
                    organism.setPosition(has_moved)
                    if self.__labyrinth.isAtFinal(has_moved):
                        organism.updateFitness(100)
                        organism.setState(self.__stateDecoder['finished'])
                        organism.setLast(count)
                        print("Generation: " + str(organism.getGeneration()), organism.getGenome())
                        self.__have_finished = True
                else:
                    organism.updateFitness(-5)
                    # organism.setState(self.stateDecoder['dead'])
            count = count + 1

        if organism.getState() == self.__stateDecoder['dead']:
            organism.updateFitness(-10)
        organism.updateFitness(-10 * self.__calculate_fitness(organism))

        # print(organism.getPosition())
        begin_position = self.__labyrinth.getBeginPosition()
        organism.setPosition({'x': begin_position['x'], 'y': begin_position['y']})
```

Figure 1. Código do fitness

gerados 2 filhos. O crossover, para o indivíduo 1 baseia-se em pegar parte do cromossomo da mãe antes do ponto de corte, e parte do cromossomo do pai depois do ponto de corte, e unir para gerar um novo individuo. Enquanto que para o segundo indivíduo gerado, o inverso se aplica. O algoritmo de crossover foi executado para cada geração 50 vezes. Tendo dessa forma, 100 indivíduos por geração após a primeira geração.

2.2. Mutação

A mutação foi posta dentro do algoritmo do crossover. A forma que a mutação ocorre é igual para cada organismo gerado pelo cruzamento dos dois melhores indivíduos de cada geração. Todo o vetor do genoma de cada organismo é percorrido, e para cada espaço do vetor, um número é sorteado, e caso o mesmo seja menor ou igual à taxa de mutação, o vetor naquele espaço irá sofrer a mutação. Quando a mutação ocorre, os 4 possíveis valores do genoma são sorteados novamente, de forma que podem ser os mesmos que já existiam anteriormente naquela posição do vetor. O código do método pode ser visualizado na Figura 2

2.2.1. Critério de parada do algoritmo

O algoritmo é executado até que surjam ao menos 10 gerações que tenham encontrado o final do labirinto. Dessa forma, durante os testes do algoritmo, existiram casos em que o final do labirinto era encontrado na geração 43, e só voltava a ser encontrado cerca de 390 gerações depois, ou seja, na geração 433.

```
def crossover(self, mom_organism, dad_organism, mutation_probability):
    if not (mom_organism and dad_organism and mutation_probability):
        return False

    if len(mom_organism.getGenome()) > len(dad_organism.getGenome()):
        mom_organism, dad_organism = dad_organism, mom_organism
    children = []
    for indexOfChildren in range(self.numberOfOrganisms):
        new_children_1 = self._organismConstructor(self.genomeSize)
        new_children_1.setPosition({'x': self._initialPosition['x'], 'y': self._initialPosition['y']})
        new_children_2 = self._organismConstructor(self.genomeSize)
        new_children_2.setPosition({'x': self._initialPosition['x'], 'y': self._initialPosition['y']})

        cut_point = random.randint(1, self.genomeSize - 1)
        for index in range(1, cut_point):
            new_children_1.setGenomeInIndex(index, mom_organism.getGenomeInIndex(index))
            new_children_2.setGenomeInIndex(index, dad_organism.getGenomeInIndex(index))

        for index in range(cut_point + 1, self.genomeSize):
            new_children_1.setGenomeInIndex(index, dad_organism.getGenomeInIndex(index))
            new_children_2.setGenomeInIndex(index, mom_organism.getGenomeInIndex(index))

        for genomeIndex in range(1, self.genomeSize):
            if random.uniform(0, 1) <= mutation_probability:  # --mutations (mutation_probability)
                new_children_1.setGenomeInIndex(genomeIndex, random.randint(0, 3))
            if random.uniform(0, 1) <= mutation_probability:  # --mutations (mutation_probability)
                new_children_2.setGenomeInIndex(genomeIndex, random.randint(0, 3))

        new_children_1.setGeneration(mom_organism.getGeneration() + 1)
        new_children_2.setGeneration(mom_organism.getGeneration() + 1)
        children.append(new_children_1)
        children.append(new_children_2)
    self.organisms = children
    return children
```

Figure 2. Código do Crossover

3. Resultados e Discussões

Ao longo do desenvolvimento do algoritmo genético, foram realizados cerca de 3 testes para cada tipo de labirinto utilizado, sendo empregado no total 5 labirintos de tamanho e organização diferentes. Os gráficos e os testes referentes a cada organização dos parâmetros do algoritmo genético e do labirinto podem ser visualizados nas figuras 3 a 19.

Na algoritmo de evolução de robôs proposto foi identificado 3 importantes parâmetros: tamanho da população, taxa de mutação e o número de gerações. Nos testes seguintes serão feitas variações para expor as mudanças causadas no resultado final. Para os primeiros testes com diferentes inicializações foi utilizado o labirinto abaixo:

```
###########
#B          #
####    ####
#          F#
###########
```

Figure 3. Primeiro labirinto

No primeiro teste foram utilizados no algoritmo genético, 50 indivíduos na população inicial, o cromossomo com 30 genes e com 5% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da primeira geração.

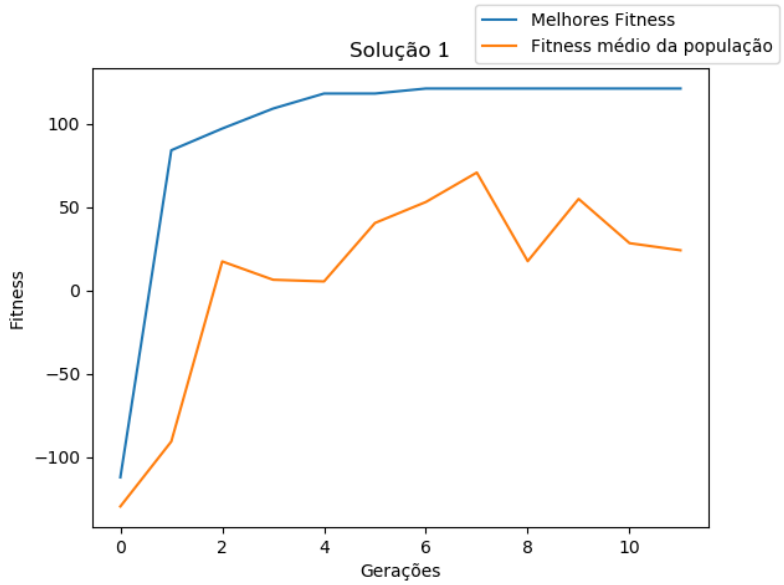

Figure 4. Primeira solução do primeiro labirinto

No segundo teste foram utilizados no algoritmo genético, 50 indivíduos na população inicial, o cromossomo com 15 genes e com 5% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 191.

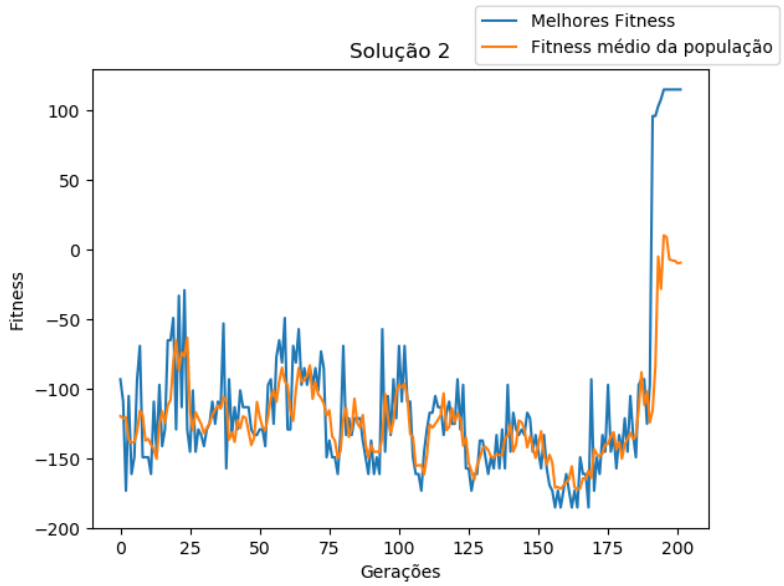

Figure 5. Segunda solução do primeiro labirinto

No terceiro teste foram utilizados no algoritmo genético, 100 indivíduos na

população inicial, o cromossomo com 15 genes e com 5% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 105.

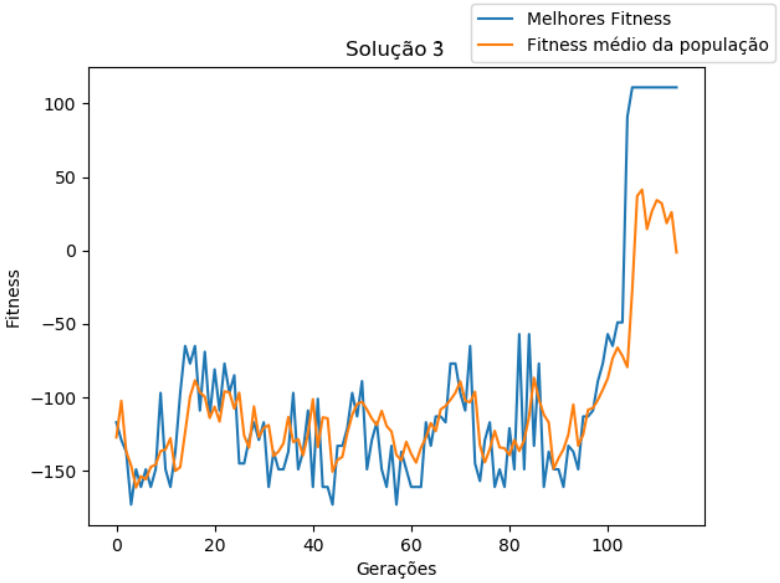

Figure 6. Terceira solução do primeiro labirinto

Para o terceiro teste será mostrado a trajetória final do melhor individuo (robô).

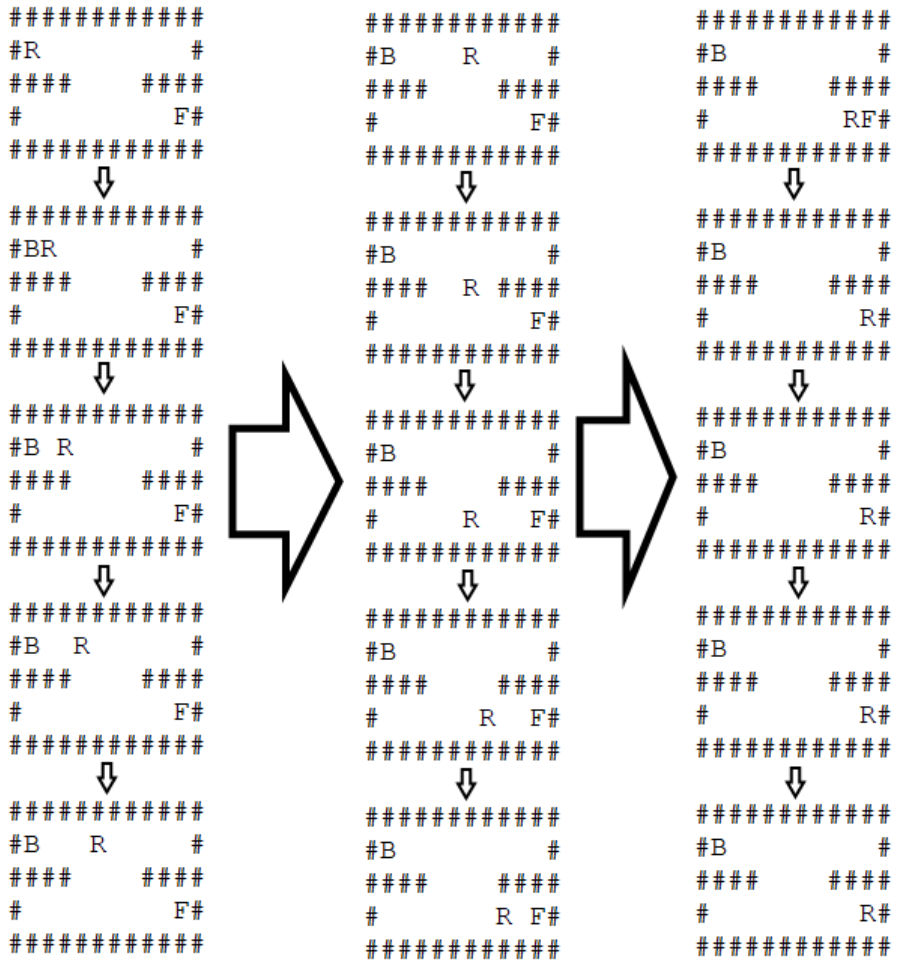

Figure 7. Trajetória do robô no labirinto

Visto que cada gene é um movimento do robô, foi necessário 11 genes para que ele chegasse até o ponto final.

No quarto teste foram utilizados no algoritmo genético, 100 indivíduos na população inicial, o cromossomo com 15 genes e com 20% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 30.

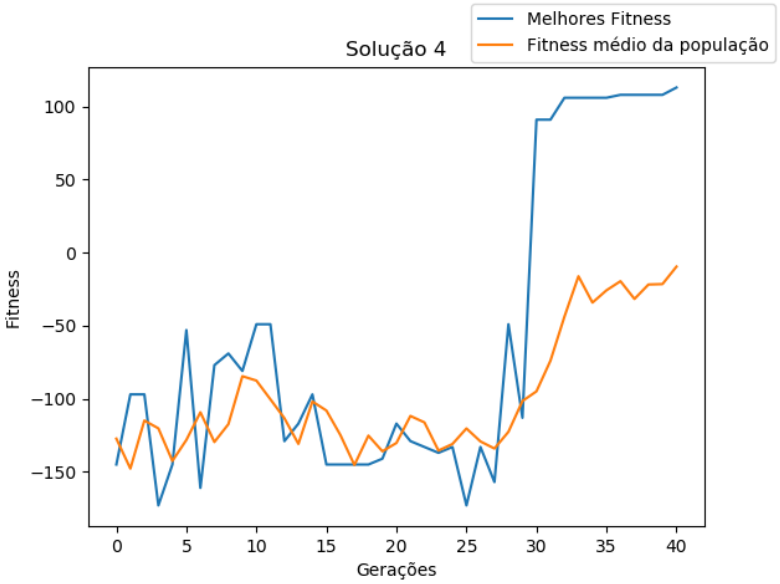

Figure 8. Quarta solução do primeiro labirinto

Visando uma maior amostragem para os testes do algoritmo, foram realizados mais 4 testes em outros dois labirintos. Abaixo encontram-se os testes para o segundo labirinto.

```
############
#B         #
### ########
#   #    #
#      ##F#
############
```

Figure 9. Segundo labirinto

No primeiro teste foram utilizados no algoritmo genético, 50 indivíduos na população inicial, o cromossomo com 30 genes e com 5% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 115.

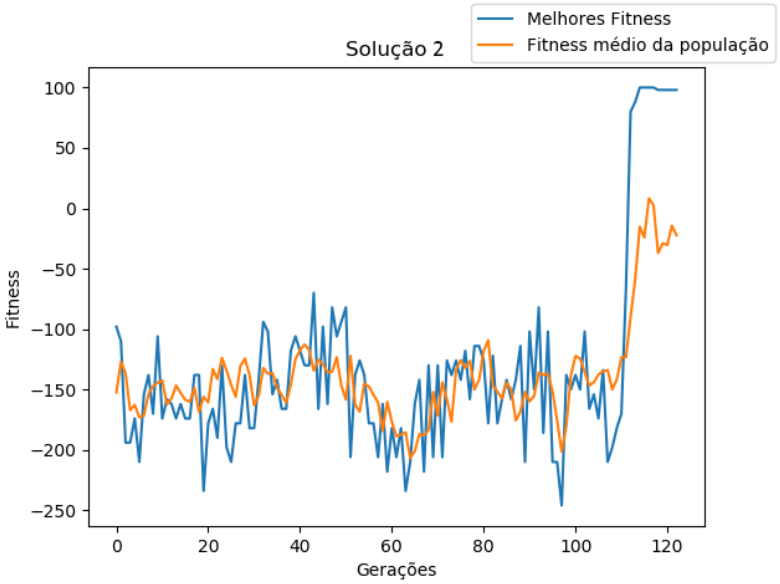

Figure 10. Primeira solução do segundo labirinto

No segundo teste foram utilizados no algoritmo genético, 50 indivíduos na população inicial, o cromossomo com 20 genes e com 5% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 1052.

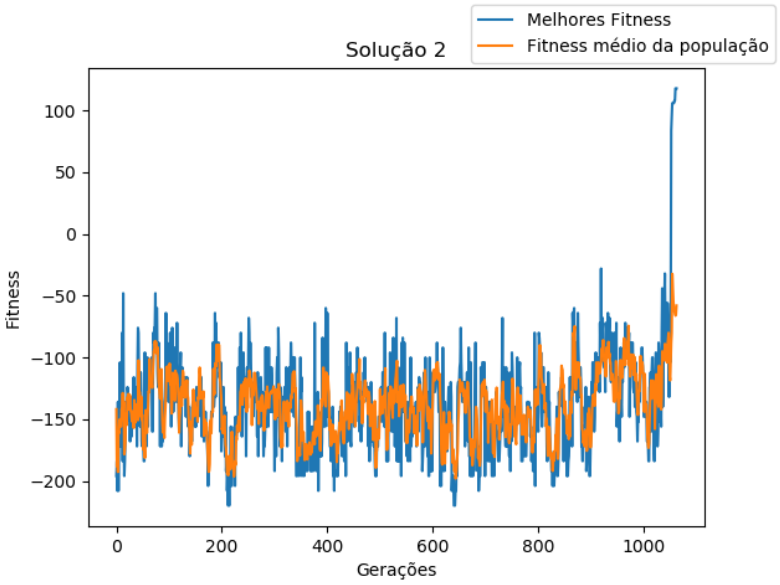

Figure 11. Segunda solução do segundo labirinto

No terceiro teste foram utilizados no algoritmo genético, 100 indivíduos na população inicial, o cromossomo com 20 genes e com 5% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 685.

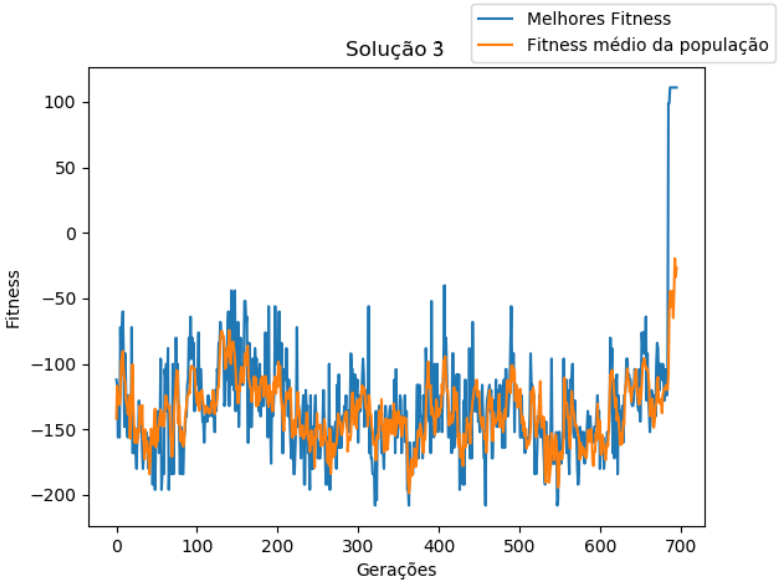

Figure 12. Terceira solução do segundo labirinto

Para o terceiro teste será mostrado a trajetória final do melhor individuo (robô).

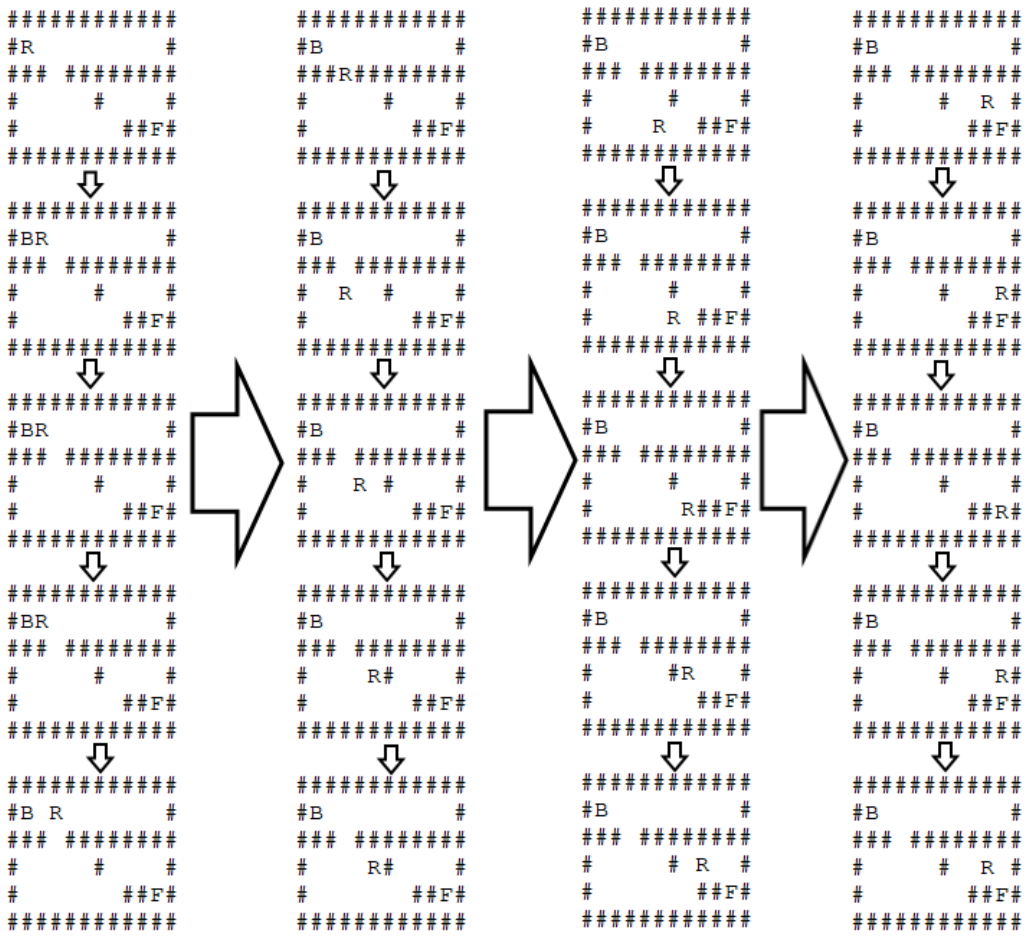

Figure 13. Trajetória do robô no labirinto

Visto que cada gene é um movimento do robô, foi necessário 17 genes para que ele chegasse até o ponto final. Uma vez que o cromossomo é de 20 genes as outras posições acabam sendo ”lixo”.

No quarto teste foram utilizados no algoritmo genético, 100 indivíduos na população inicial, o cromossomo com 20 genes e com 20% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 656.

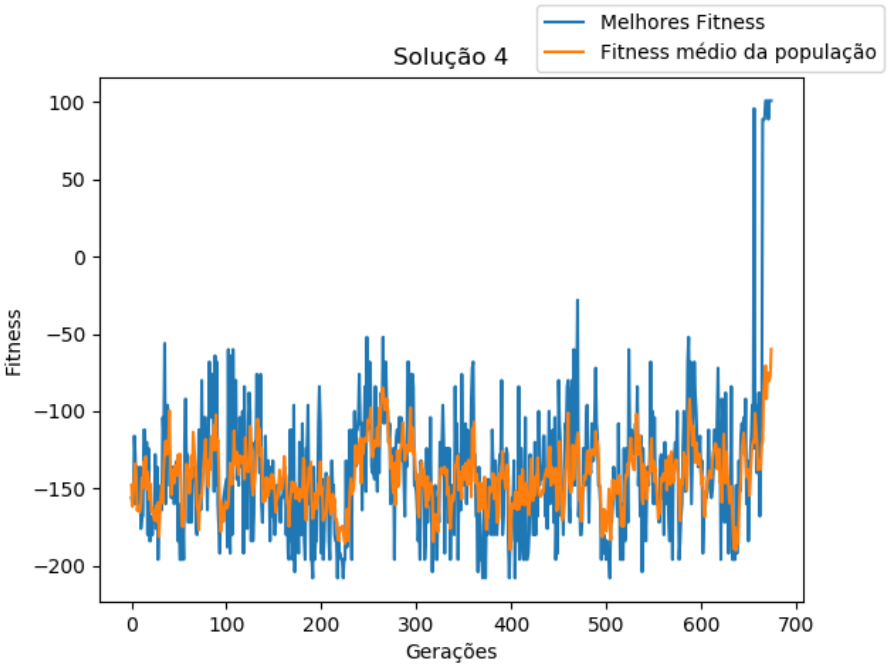

Figure 14. Quarta solução do segundo labirinto

Foi criado um novo e último labirinto para serem realizados os últimos testes. A representação desse labirinto pode ser visualizada na figura 15.

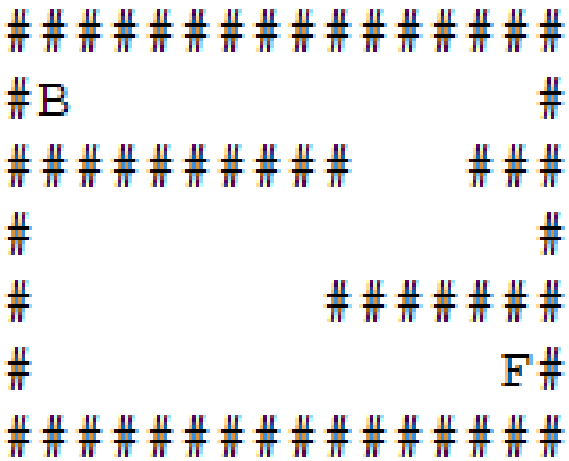

Figure 15. Terceiro labirinto

No primeiro teste foram utilizados no algoritmo genético, 50 indivíduos na população inicial, o cromossomo com 50 genes e com 5% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 33.

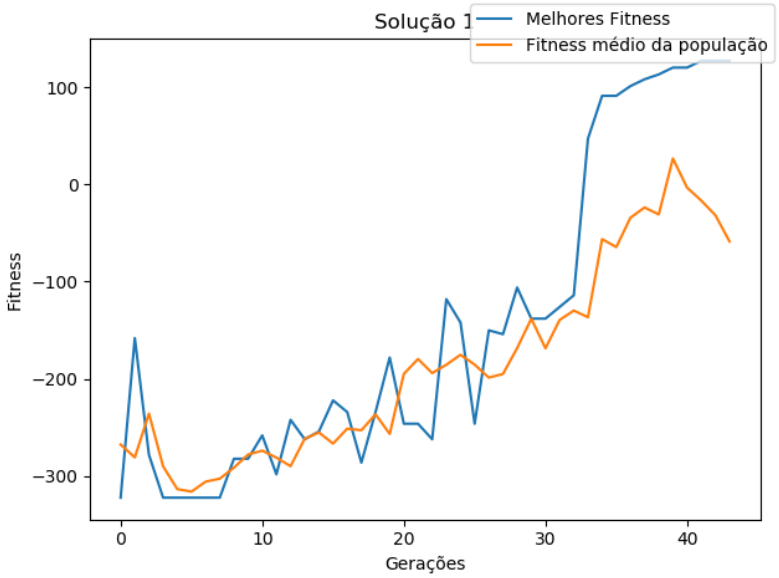

Figure 16. Primeira solução do terceiro labirinto

No segundo teste foram utilizados no algoritmo genético, 50 indivíduos na população inicial, o cromossomo com 60 genes e com 5% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 15.

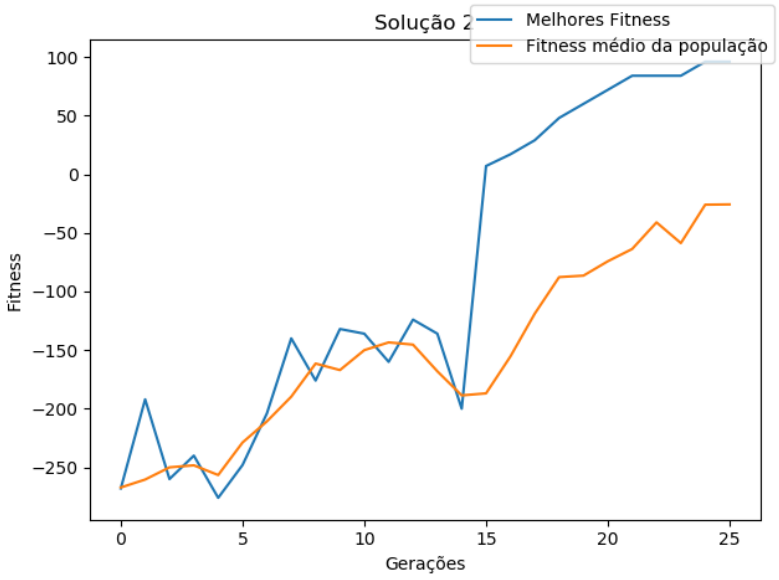

Figure 17. Segunda solução do terceiro labirinto

No terceiro teste foram utilizados no algoritmo genético, 100 indivíduos na população inicial, o cromossomo com 50 genes e com 5% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 83.

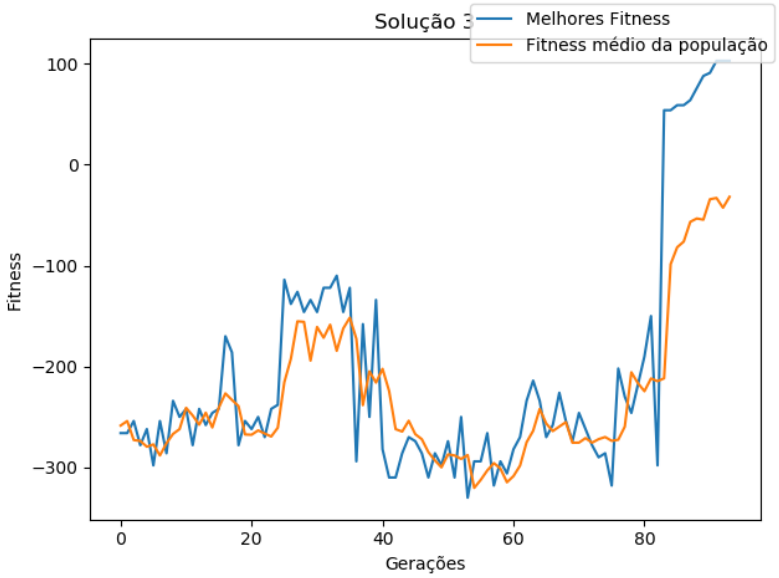

Figure 18. Terceira solução do terceiro labirinto

No quarto teste foram utilizados no algoritmo genético, 100 indivíduos na população inicial, o cromossomo com 50 genes e com 20% na probabilidade de mutação. Pode-se perceber após este teste que o primeiro individuo chegou ao final a partir da geração 75.

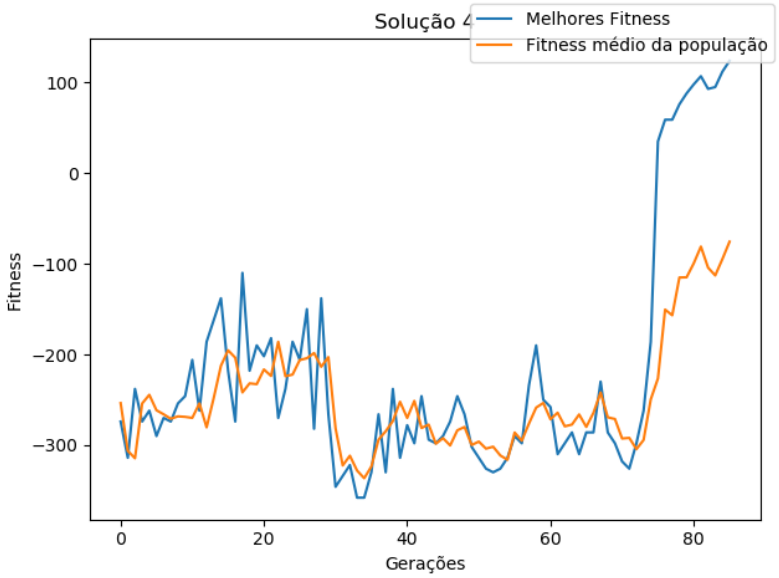

Figure 19. Quarta solução do terceiro labirinto

Posteriormente aos testes foi verificado a importância da probabilidade de mutação(frequência que os genes serão modificados), pois se não houver mutação, a descendência é gerada imediatamente após o cruzamento sem nenhuma alteração. Se ocorrer a mutação, uma ou mais partes do cromossomo é alterado. Após os testes verificou-se que a mutação em geral evita que o algoritmo genético caia num extremo (mínimo ou

máximo) local, além de trazer variedade entre aos cromossomos. Além disso, foi verificado que não se deve utilizar uma taxa de probabilidade muita alta na mutação porque senão o algoritmo genético pode torna-se uma busca aleatória.

Após os testes foi constatado que o tamanho da população(quantos indivíduos existem na população (em uma geração)), interfere bastante no resultado. Caso haja poucos cromossomos, o algoritmo genético terá poucas possibilidade de realizar cruzamentos e somente uma pequena parte do espaço de soluções será explorada. Por outro lado, se houver muitos cromossomas, os algoritmo genético tornar-se lento. Além disso, após um determinado limite não é conveniente aumentar a população porque isso não resolve o problema mais rapidamente do que com tamanhos moderados de população.

Como critério de parada do algoritmo foi utilizado a aptidão do melhor indivíduo em conjunto com a limitação do número de gerações. Sendo assim, foi verificado que a quantidade de gerações está diretamente proporcional ao aumento da avaliação(fitness) dos indivíduos e consequentemente ao decorrer das gerações, mais indivíduos conseguem chegar até o ponto final do labirinto.

Além disso foi percebido que a quantidade de genes interfere na solução final, já que a quantidade de movimentos pode ser maior ou menor em cada individuo. Sendo assim, indivíduos com menores genes demoram mais gerações para chegar ao ponto final, porém possuem uma solução mais robusta, visto que necessitam de menos movimentos para chegarem até o final.

4. Conclusão

Foi apresentado nesse relatório a descrição das etapas de desenvolvimento da evolução de robôs em labirintos. Observou-se que o tamanho do labirinto e a localização do ponto final interferem diretamente na quantidade de gerações necessárias para o treinamento do robô.

É fato influenciador também o tamanho da população, probabilidade de mutação e principalmente a quantidade de gerações para uma boa evolução. Podendo assim, as variações levarem a uma solução que seja necessário muitas gerações até chegar ao ponto final do labirinto.