Relatório

Teste caixa-preta: Neste teste foram usadas 5 métodos: tropas.perdermo(inimigos), tropas.morreu(aliados, tabuleiro), ponto_invoc.colidiux(x), ponto_invoc.colidiuy(y) e ponto_invoc.colidiu(x, y).

tropas.perdemo(inimigos): verifica se algum dos inimigos na list de inimigos possui um atributo x menor ou igual a -64.

```python
def perdemo(self, inimigos):
    if inimigos == None or len(inimigos) == 0:
        return False
    for i in inimigos:
        if i.x + 64 <= 0:
            return True
        else:
            return False
```

As classes de partição foram divididas conforme a tabela:

Os testes foram automatizados usando o pytest e foram feitos 4 testes englobando todas as partições.

Func: tropas.perdemo()

testes	-64 < x	muitos inimigos	Resultado esperado
test_01	sim	sim	FALSE
test_02	sim	não	FALSE
test_03	não	sim	TRUE
test_04	não	não	TRUE

```python
import pytest
from tropas import *

def run_tropas():
    pygame.init()
    pygame.display.set_mode((768, 512))
    test_lvl = [0, 1]
    tropas = Tropas(Campo(12, 6, 50, 128))
    tropas.tempo = 1
    tropas.spawn_inimigos(test_lvl)
    return tropas
def add_enemies(tropas, qnt):
    for i in range(qnt + 1):
        tropas.spawn_inimigos([0,1])
#teste com 10 inimigos dentro e todos com x inicial = 780
def test_func_must_return_false_01():
    tropas = run_tropas()
    add_enemies(tropas, 10)
    assert tropas.perdemo(tropas.entidades['inimigos']) == False
#teste com um inimigo dentro do dicionario de entidades e com o x inicial = 780
def test_func_must_return_false_02():
    tropas = run_tropas()
    assert tropas.perdemo(tropas.entidades['inimigos']) == False
#teste com 10 inimigos e todos com seu x + 64 <= 0
def test_func_must_return_True_03():
    tropas = run_tropas()
    add_enemies(tropas, 10)
    tropas.entidades['inimigos'][0].x = -64
    assert tropas.perdemo(tropas.entidades['inimigos']) == True
#testo com um inimigo dentro do dicionario e com seu x + 64 <= 0
def test_func_must_return_True_04():
    tropas = run_tropas()
    tropas.entidades['inimigos'][0].x = -64
    assert tropas.perdemo(tropas.entidades['inimigos']) == True
```

tropas.morreu(aliados, tabuleiro): verifica se algum dos aliados na list está com a vida menor ou igual a 0 e se tiver menor que 0 ele troca o atributo tem_unidade do tabuleiro para False.

```python
def morreu(self, aliados, tabuleiro):
        if len(aliados) == 0:
            return False
        else:
            for i in aliados:
                if i.vida <= 0:
                    aliados.remove(i)
                    tabuleiro.blocos[i.linha][i.coluna].tem_unidade = False
                    return True
                else:
                    return False
```

As classes de partição foram divididas conforme a tabela:

função: tropas.morreu()			
testes	vida < 0	muitos inimigos	Resultado esperado
test_01	sim	sim	FALSE
test_02	sim	não	FALSE
test_03	não	sim	TRUE
test_04	não	não	TRUE

Os testes foram automatizados usando o pytest e foram feitos 4 testes englobando todas as partições.

```python
import pytest
from tropas import *

def run_tropas():
    pygame.init()
    pygame.display.set_mode((768, 512))
    tropas = Tropas(Campo(12, 6, 50, 128))
    tropas.entidades['aliados'].append(Esqueleto(0, 0, 32, 32, tropas.esq_ss, 5, 2, 0, 0, 2, ""))
    return tropas

def add_allies(tropas, qnt):
    for i in range(qnt + 1):
        tropas.entidades['aliados'].append(Esqueleto(0, 0, 32, 32, tropas.esq_ss, 5, 2, 0, 0, 2, ""))

def reset_all_allies_health(aliados):
    for i in aliados:
        i.vida = 0
#teste com varios aliados na lista e com sua vida > 0
def test_func_must_return_false_01():
    tropas = run_tropas()
    add_allies(tropas, 10)
    assert tropas.morreu(tropas.entidades['aliados'], tropas.tabuleiro) == False
#teste com apenas um aliado na lista e com sua vida > 0
def test_func_must_return_false_02():
    tropas = run_tropas()
    assert tropas.morreu(tropas.entidades['aliados'], tropas.tabuleiro) == False
#teste com vários inimigos e todos com vida <= 0
def test_func_must_return_true_03():
    tropas = run_tropas()
    add_allies(tropas, 10)
    reset_all_allies_health(tropas.entidades['aliados'])
    assert tropas.morreu(tropas.entidades['aliados'], tropas.tabuleiro) == True
#teste com um inimigo com vida <= 0
def test_func_must_return_true_04():
    tropas = run_tropas()
    reset_all_allies_health(tropas.entidades['aliados'])
    assert tropas.morreu(tropas.entidades['aliados'], tropas.tabuleiro) == True
```

ponto_invoc.colidiux(x): o método verifica se um x passado como parâmetro é maior que o atributo x da classe dona do método e menor que o atributo x da classe + o atributo largura da classe, podendo retornar True ou False.

```
def colidiux(self, x):
    if x == None:
        return None
    else:
        if self.x < x < self.x + self.largura:
            return True
        else:
            return False
```

As classes foram divididas conforme a tabela:

Func: ponto_invoc.colidiuy()				
Testes	y > self.y	y < self.y + self.altura	Resultado esperado	
test_01	sim	sim	TRUE	
test_02	sim	não	FALSE	
test_03	não	sim	FALSE	
test_04	não	não	--	impossivel

Foram feitos 3 testes, pois uma das combinações era impossível de acontecer:

```
from ponto_invoc import *
import pytest

def run_lugar():
    lugar = Lugar(0,0,32,32)
    return lugar

#test_01_s_s_t
def test_func_must_return_true_01():
    bloco = run_lugar()
    assert bloco.colidiux(1) == True
#test_02_s_n_f
def test_func_must_return_false_02():
    bloco = run_lugar()
    assert bloco.colidiux(32) == False
#test_03_n_s_f
def test_func_must_return_false_03():
    bloco = run_lugar()
    assert bloco.colidiux(-1) == False
```

ponto_invoc.colidiuy(y): o método verifica se um y passado como parâmetro é maior que o atributo y da classe dona do método e menor que o atributo y da classe + o atributo altura da classe, podendo retornar True ou False.

```
def colidiuy(self, y):
    if y == None:
        return None
    else:
        if self.y < y < self.y + self.altura:
            return True
        else:
            return False
```

As classes foram divididas conforme a tabela:

Func: ponto_invoc.colidiux()				
Testes	x > self.x	x < self.x + self.largura	Resultado esperado	
test_01	sim	sim	TRUE	
test_02	sim	não	FALSE	
test_03	não	sim	FALSE	
test_04	não	não	--	impossivel

Foram feitos 3 testes, pois uma das combinações era impossível de acontecer:

```
from ponto_invoc import *
import pytest

def run_lugar():
    lugar = Lugar(0,0,32,32)
    return lugar

#test_01_s_s_t
def test_func_must_return_true_01():
    bloco = run_lugar()
    assert bloco.colidiuy(1) == True
#test_02_s_n_f
def test_func_must_return_false_02():
    bloco = run_lugar()
    assert bloco.colidiuy(32) == False
#test_03_n_s_f
def test_func_must_return_false_03():
    bloco = run_lugar()
    assert bloco.colidiuy(-1) == False
```

ponto_invoc.colidiu(x, y): o método verifica se um x e um y está colidindo com o objeto dono do método, retornando True ou False:

```
def colidiu(self, x, y):
    if x == None or y == None:
        return None
    else:
        if self.colidiux(x):
            if self.colidiuy(y):
                return True
        return False
```

As classes foram divididas conforme a tabela:

Func: ponto_invoc.colidiu()			
Testes	ponto_invoc. colidiux()	ponto_invoc. colidiuy()	Resultado esperado
test_01	sim	sim	TRUE
test_02	sim	não	FALSE
test_03	não	sim	FALSE
test_04	não	não	FALSE

Os testes foram automatizados usando o pytest e foram feitos 4 testes englobando todas as partições.

```
from ponto_invoc import *
import pytest

def run_lugar():
    lugar = Lugar(0,0,32,32)
    return lugar

#test_01_s_s_t
def test_func_must_return_true_01():
    bloco = run_lugar()
    assert bloco.colidiu(1, 1) == True
#test_02_s_n_f
def test_func_must_return_false_02():
    bloco = run_lugar()
    assert bloco.colidiu(1, -1) == False
#test_03_n_s_f
def test_func_must_return_false_03():
    bloco = run_lugar()
    assert bloco.colidiu(-1, 1) == False
#test_03_n_n_f
def test_func_must_return_false_04():
    bloco = run_lugar()
    assert bloco.colidiu(-1, -1) == False
```

Resultados: os resultados de cada teste podem ser vistos a seguir e todos conforme o esperado:

```
============================= test session starts ==============================
collecting ... collected 4 items

test_func_tropas_perdemo.py::test_func_must_return_false_01 PASSED       [ 25%]
test_func_tropas_perdemo.py::test_func_must_return_false_02 PASSED       [ 50%]
test_func_tropas_perdemo.py::test_func_must_return_True_03 PASSED        [ 75%]
test_func_tropas_perdemo.py::test_func_must_return_True_04 PASSED        [100%]

============================== 4 passed in 1.54s ===============================
```

```
============================= test session starts ==============================
collecting ... collected 4 items

test_func_tropas_morreu.py::test_func_must_return_false_01 PASSED        [ 25%]
test_func_tropas_morreu.py::test_func_must_return_false_02 PASSED        [ 50%]
test_func_tropas_morreu.py::test_func_must_return_true_03 PASSED         [ 75%]
test_func_tropas_morreu.py::test_func_must_return_true_04 PASSED         [100%]

============================== 4 passed in 0.48s ===============================
```

```
============================= test session starts ==============================
collecting ... collected 3 items

test_func_ponto_invoc_colidiux.py::test_func_must_return_true_01 PASSED  [ 33%]
test_func_ponto_invoc_colidiux.py::test_func_must_return_false_02 PASSED [ 66%]
test_func_ponto_invoc_colidiux.py::test_func_must_return_false_03 PASSED [100%]

============================== 3 passed in 0.02s ===============================
```

```
============================= test session starts ==============================
collecting ... collected 3 items

test_func_ponto_invoc_colidiuy.py::test_func_must_return_true_01 PASSED  [ 33%]
test_func_ponto_invoc_colidiuy.py::test_func_must_return_false_02 PASSED [ 66%]
test_func_ponto_invoc_colidiuy.py::test_func_must_return_false_03 PASSED [100%]

============================== 3 passed in 0.02s ===============================
```

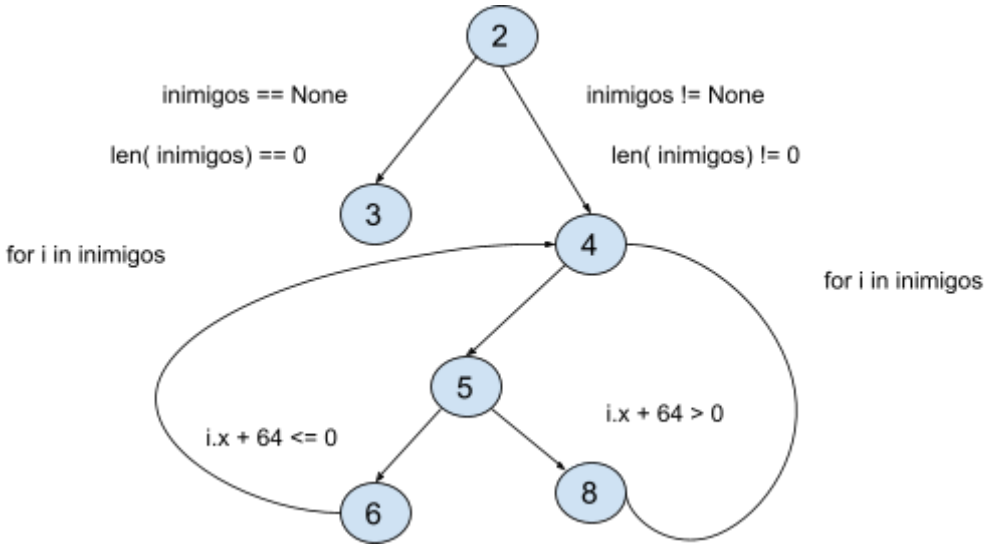
```
============================= test session starts ==============================
collecting ... collected 4 items

test_func_ponto_invoc_colidiu.py::test_func_must_return_true_01 PASSED   [ 25%]
test_func_ponto_invoc_colidiu.py::test_func_must_return_false_02 PASSED  [ 50%]
test_func_ponto_invoc_colidiu.py::test_func_must_return_false_03 PASSED  [ 75%]
test_func_ponto_invoc_colidiu.py::test_func_must_return_false_04 PASSED  [100%]

============================== 4 passed in 0.02s ===============================
```

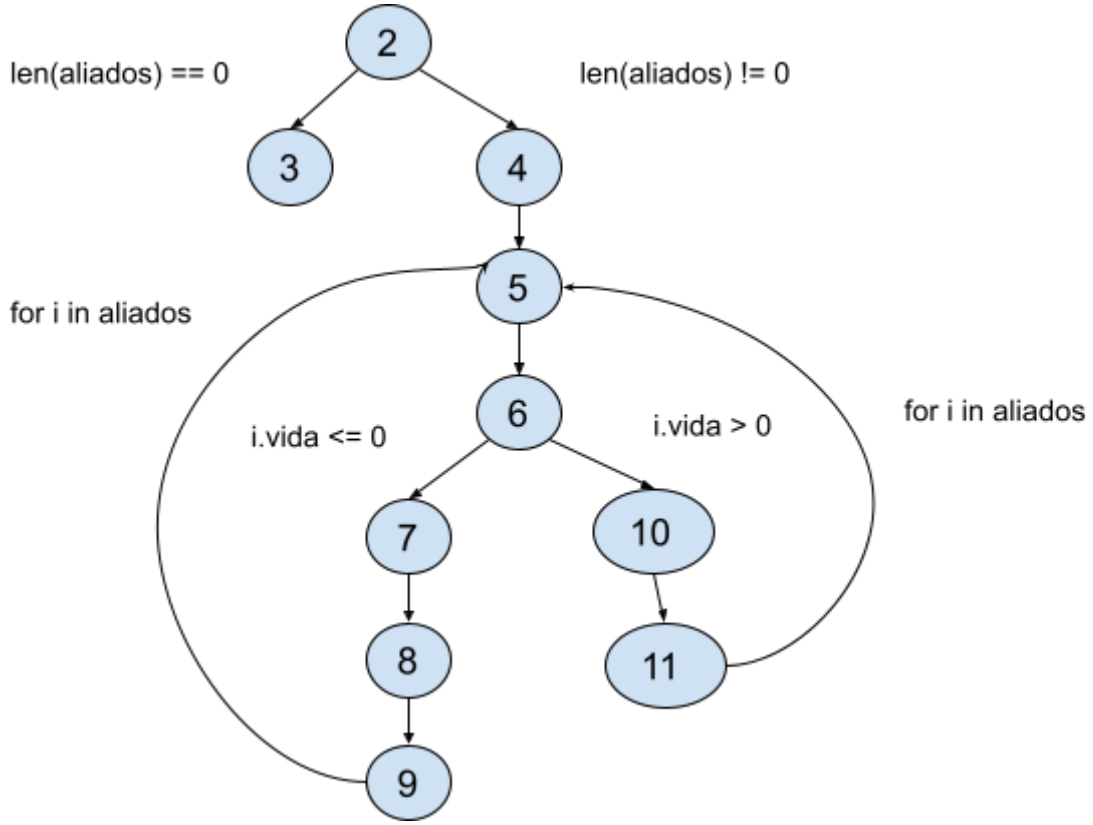
Teste caixa-branca: Neste teste foram usadas 5 métodos: tropas.perdermo(inimigos), tropas.morreu(aliados, tabuleiro), ponto_invoc.colidiux(x), ponto_invoc.colidiuy(y) e ponto_invoc.colidiu(x, y).

Teste tropas.perdemo(aliados, tabuleiro) :



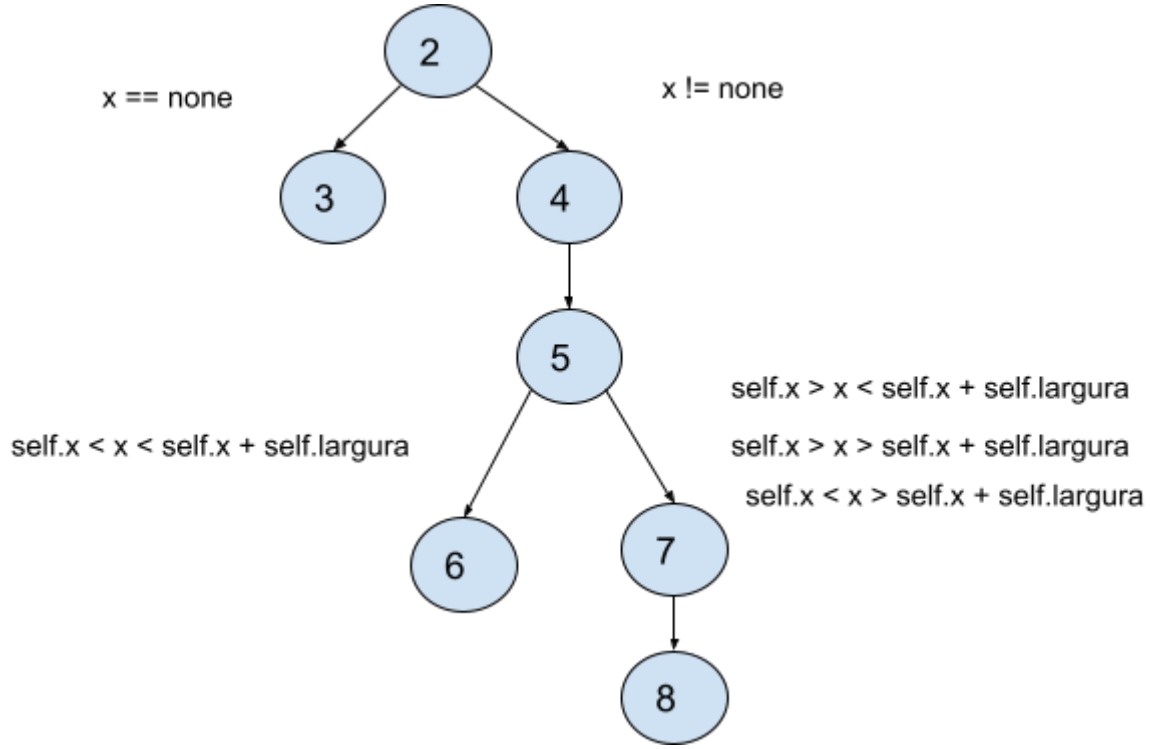

```
1    def perdemo(self, inimigos):
2        if inimigos == None or len(inimigos) == 0:
3            return False
4        for i in inimigos:
5            if i.x + 64 <= 0:
6                return True
7            else:
8                return False
```

Teste tropas.morreu(aliados, tabuleiro) :



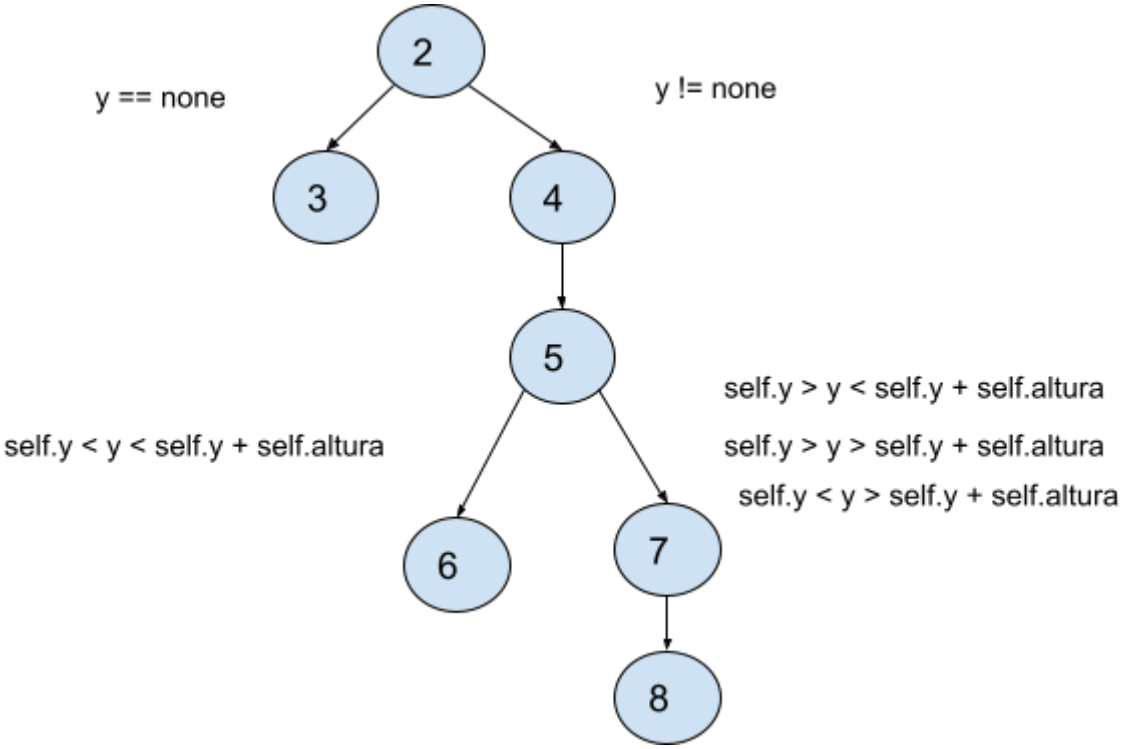

```python
def morreu(self, aliados, tabuleiro):
        if len(aliados) == 0:
            return False
        else:
            for i in aliados:
                if i.vida <= 0:
                    aliados.remove(i)
                    tabuleiro.blocos[i.linha][i.coluna].tem_unidade = False
                    return True
                else:
                    return False
```

Teste ponto_invoc.colidiux(x):



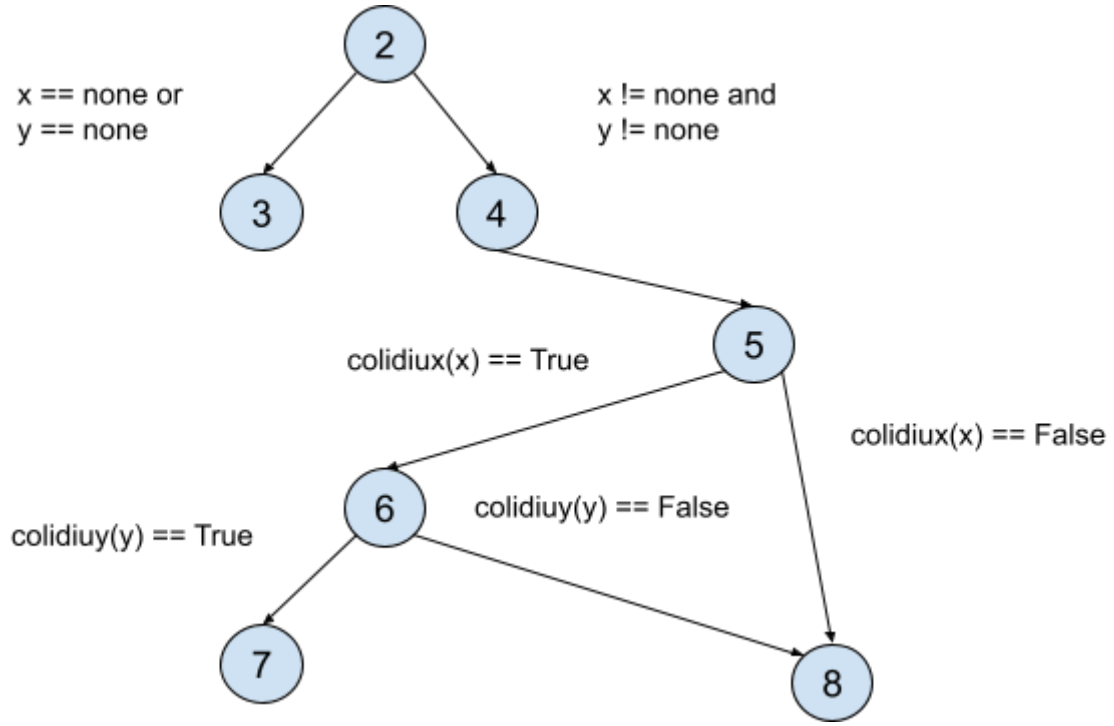

```python
def colidiux(self, x):
    if x == None:
        return None
    else:
        if self.x < x < self.x + self.largura:
            return True
        else:
            return False
```

Teste ponto_invoc.colidiuy(y):




```python
def colidiuy(self, y):
    if y == None:
        return None
    else:
        if self.y < y < self.y + self.altura:
            return True
        else:
            return False
```

Teste ponto_invoc.colidiu(x, y):




```
1    def colidiu(self, x, y):
2        if x == None or y == None:
3            return None
4        else:
5            if self.colidiux(x):
6                if self.colidiuy(y):
7                    return True
8            return False
```

Modularização das funções:

Ocorreram mudanças no método lógica da classe tropas, houve uma desmembração do método em 5 métodos, um destes métodos foi criado para testar se o jogador perdeu, outro se algum dos aliados morreu, outro puxa os métodos lógica dos objetos dentro da lista de aliados e outro dentro da lista de inimigos e o último e o novo logica do tropas puxa os novos métodos e faz algumas verificações..

Método antigo:

```
1    def logica(self, mouse, projeteis):
2        for i in self.entidades['inimigos']:
3            if i.x + 64 <= 0:
4                self.perdeu = True
5        for i in self.entidades['aliados']:
6            match i.id:
7                case 2:
8                    i.logica(mouse)
9                case 1:
10                   i.logica(self.entidades['inimigos'], self.tabuleiro, projeteis)
11               case 3:
12                   i.logica(self.entidades['inimigos'], self.tabuleiro)
13           if i.vida <= 0:
14               self.entidades['aliados'].remove(i)
15               self.tabuleiro.blocos[i.linha][i.coluna].tem_unidade = False
16       for i in self.entidades['inimigos']:
17           match i.id:
18               case 2:
19                   i.logica(self.entidades['aliados'], projeteis)
20               case 1:
21                   i.logica(self.entidades['aliados'])
```

Novos métodos:

```python
def logica(self, mouse, projeteis):
        self.perdemo(self.entidades['inimigos'])
        self.morreu(self.entidades["aliados"], self.tabuleiro)
        self.logicaIni(self.entidades['inimigos'], projeteis, self.entidades['aliados'])
        self.logicaAli(self.entidades['aliados'], projeteis, self.tabuleiro, self.entidades['inimigos'], mouse)

    """separei a logica dos inimigos e aliados da logica tropas"""
    def logicaAli(self, aliados, projeteis, tabuleiro, inimigos, mouse):
        for i in aliados:
            match i.id:
                case 2:
                    i.logica(mouse)
                case 1:
                    i.logica(inimigos, tabuleiro, projeteis)
                case 3:
                    i.logica(inimigos, tabuleiro)
    def logicaIni(self, inimigos, projeteis, aliados):
        for i in inimigos:
            match i.id:
                case 2:
                    i.logica(aliados, projeteis)
                case 1:
                    i.logica(aliados)

    """separei a função perdemo do logica"""
    def perdemo(self, inimigos):
        for i in inimigos:
            if i.x + 64 <= 0:
                self.perdeu = True
                return self.perdeu
    """Função q verifica se morreu"""
    def morreu(self, aliados, tabuleiro):
        for i in aliados:
            if i.vida <= 0:
                aliados.remove(i)
                tabuleiro.blocos[i.linha][i.coluna].tem_unidade = False
```

Também houve mudanças nos armazenamento dos inimigos e aliados, antes eram colocados cada um em matrizes, agora foram substituídos por um dicionário contendo duas listas com as chaves ‘aliados’ e ‘inimigos.

```python
def __init__(self, tabuleiro):
    self.perdeu = False
    self.atraso_invoc = 0
    self.matriz_tropas = []
    self.matriz_inimigos = []
```

após as mudanças:

```python
def __init__(self, tabuleiro):
    self.perdeu = False
    self.atraso_invoc = 0
    self.entidades = {
        'inimigos':[],
        'aliados':[]
    }
```

De mesmo modo, todas as classes que usavam essas estruturas de dados precisaram ser alteradas.