Crazy Arcade

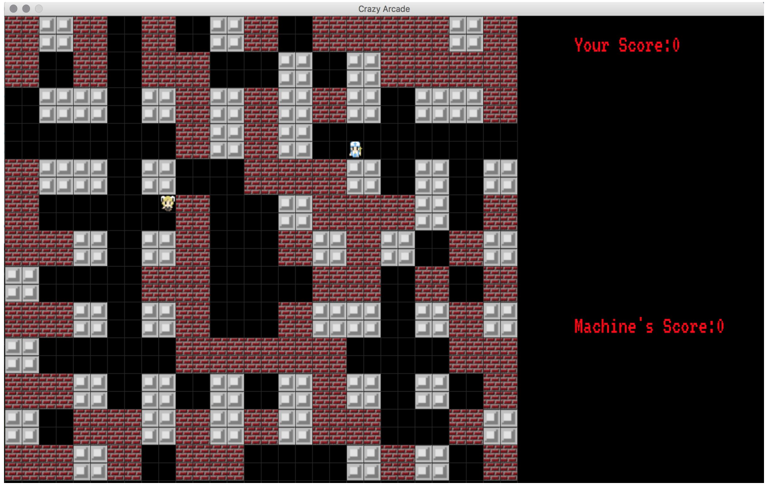
(Q版泡泡堂)

Lu Guo & Yuping Zang

Agenda

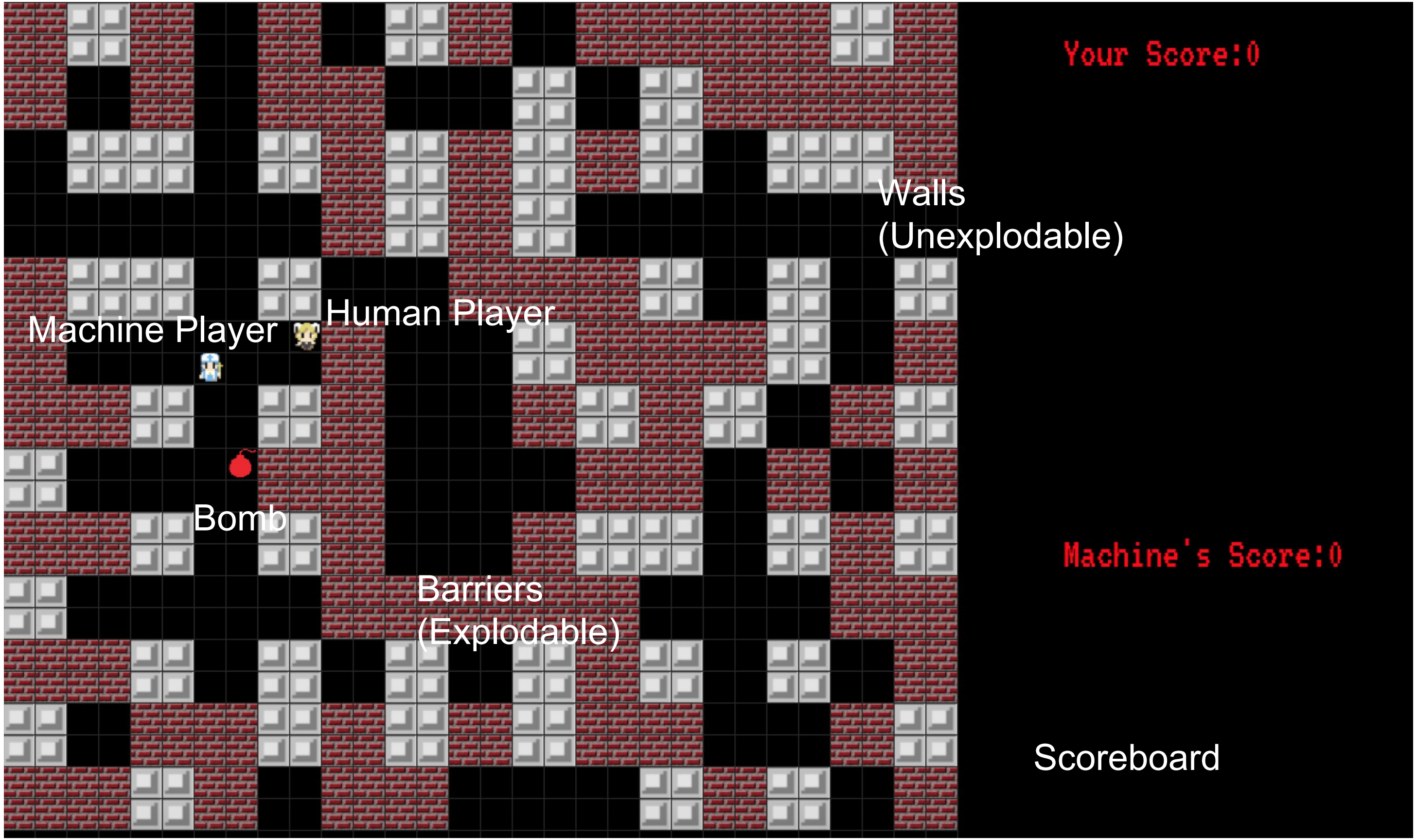
- Basic Game Rules
- Working Process Overview
- Several Highlights
- Reflection & Future Improvement

Crazy Arcade:
A Human V.S. Machine Game Based on the Chatting System

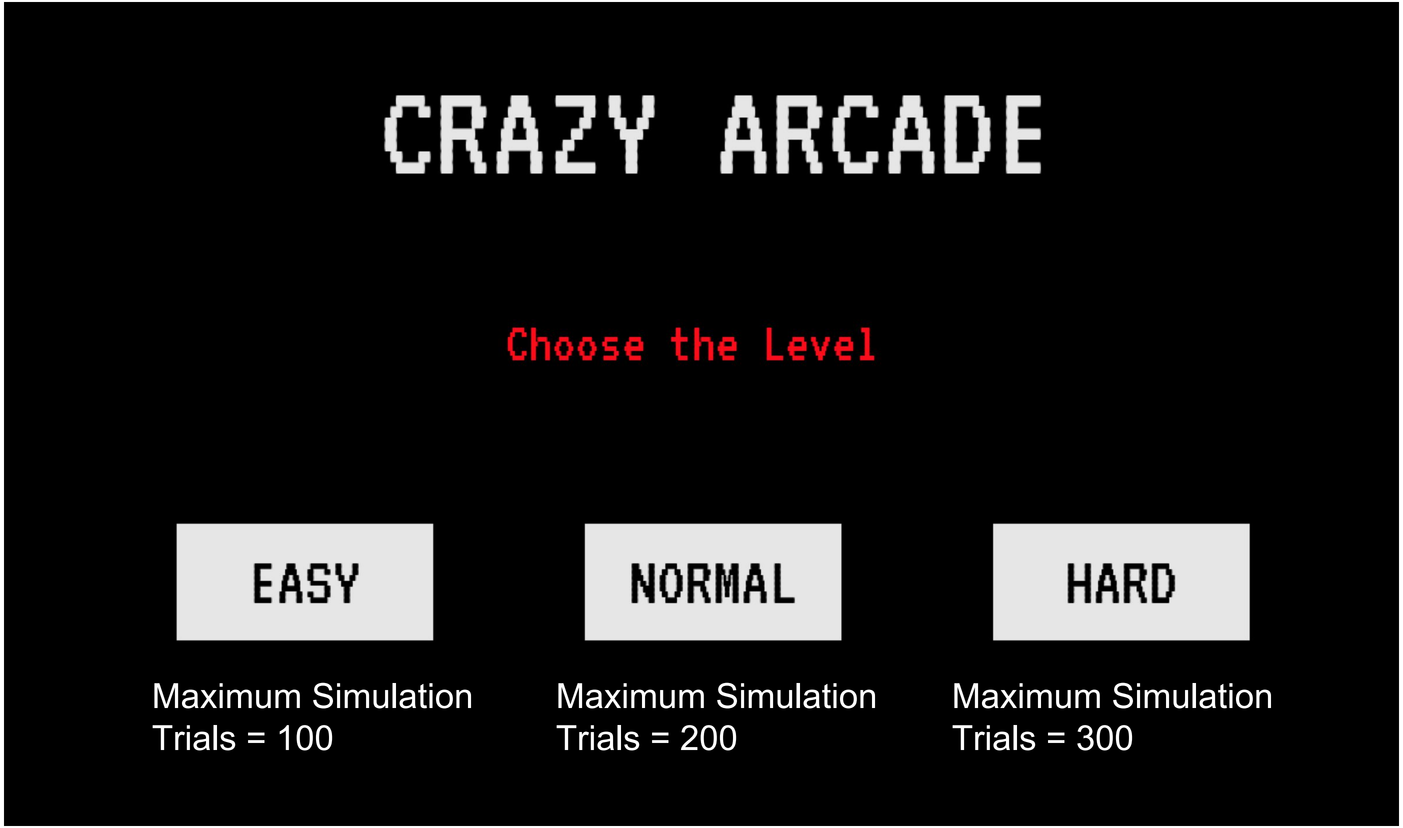


- Inspiration from Q版泡泡堂 on 4399.com
- Adapt it into a human-computer game with artificial intelligence

Basic Game Rules



Difficulty Levels: Different Parameters of Machine Simulation -- how intelligent the machine is



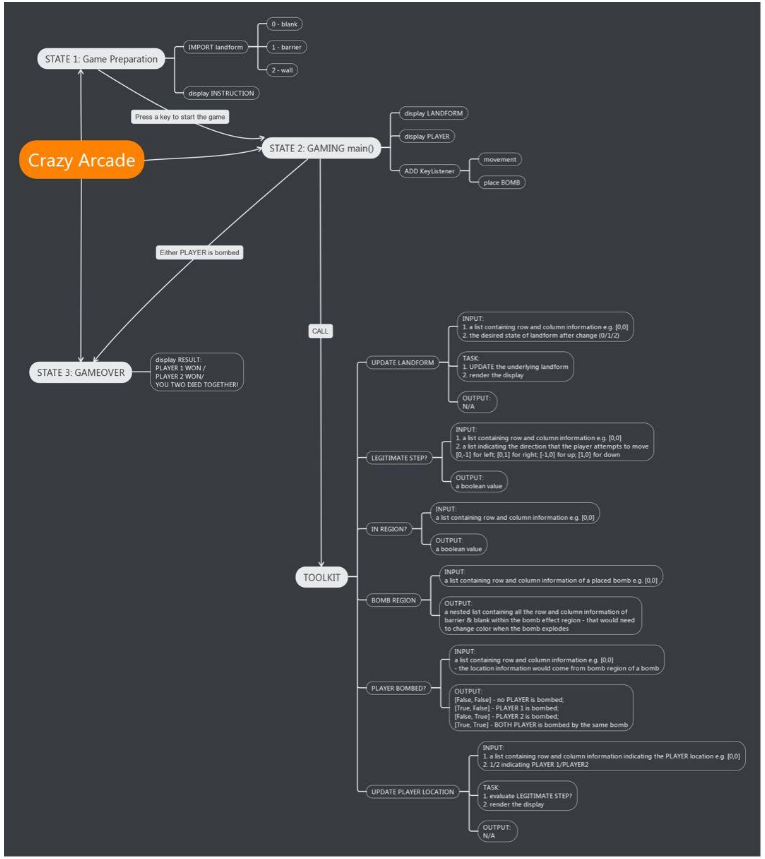
Mouse Control + Keyboard Control

- *pygame.mouse.get_pos()* : detect mouse position and button function to access different difficulty settings.
- Listen to keyboard inputs: control human player's movement event and bomb event.

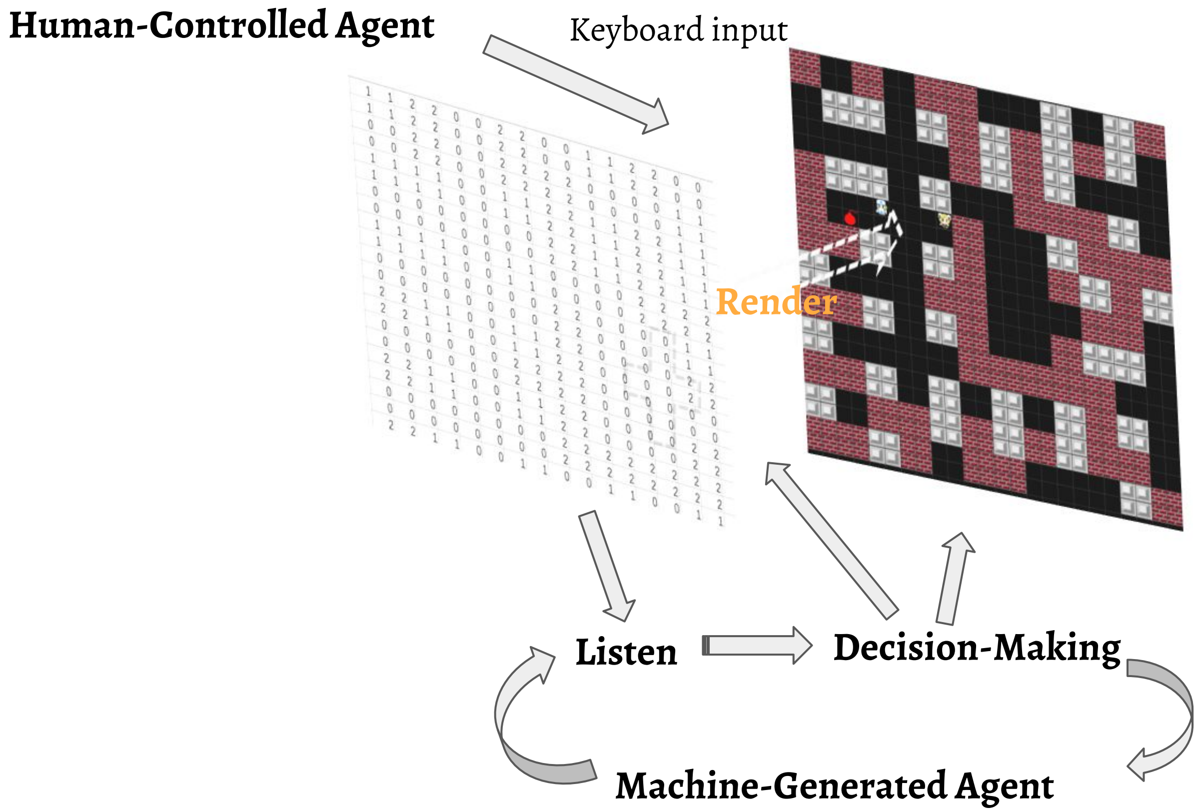
Workflow Overview

1. Design state machine diagram & program flowchart
2. Implement game settings and human-controlled agent
3. Implement machine-generated agent with smart movement and bomb placement decisions based on random walk and simulation
4. Improve machine-generated agent’s intelligence in reacting to dilemma of survival, attack from human agent, and urgent explosion...

State Machine Diagram & Program Flowchart

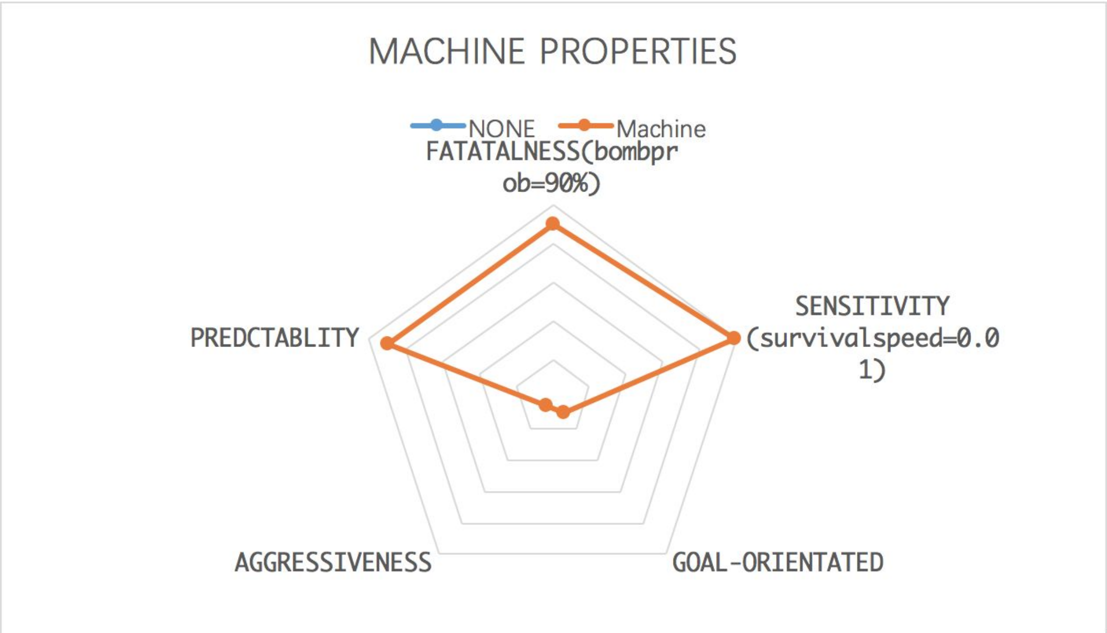


Interaction between Underlying Landform Matrix & UI



Machine Property: Decision-Making Framework of Movement Event & Bomb Placement

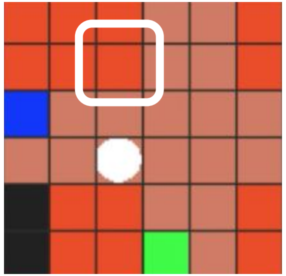
- **Movement Event:** Random Walk + Exit Danger Regions (Triggered by any activities that may lead to changes in landform)
- **Bomb Placement:** Certain Probability + Simulation Evaluation



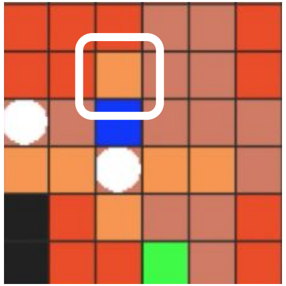
Highlight 1: Machine's Urgent Response Mechanism

KEY CHALLENGE:
Overlapped bombing events lead to chaotic dynamics in landform change.

1



2



DEMONSTRATION:
Safe zone may transform into a potential danger zone after a bomb event.

Highlight 1: Machine’s Urgent Response Mechanism

- *InDilemma():* Return a boolean value that indicates machine player’s status if it is trapped in a danger zone with zero adjacent exit options among 4 directions.

  -> Initialize a more aggressive and blind exploration in larger region to escape.

```
def InDilemma(index,MoreDangerZone = []):
    if InDangerZone(index) or (index in MoreDangerZone) or (index in BOMBSTACK):
        #ProbDirec = 0
        for i in DIRECTION:
            adjustedIndex = [index[0]+i[0],index[1]+i[1]]
            if legitMove(adjustedIndex) and not(InDangerZone(adjustedIndex)) and (adjustedIndex not in MoreDangerZone) and (adjustedIndex not in BOMBSTACK):
                #ProbDirec += 1
                return False
        #if ProbDirec == 0:
        return True
    return False
```

Highlight 1: Machine's Urgent Response Mechanism

- *UrgentMove():* Handle real-time reaction to possible attacks from human player.

-> If a bomb attack is detected, machine player would first follow a relatively safe trajectory sampled from 20 pre-set adjacent exit options, and then escape further if necessary after evaluation.

```
def UrgentMove(screen):
    global INURGENTMOVE
    global INSURVIVAL

    INURGENTMOVE = True
    # new
    while InDangerZone(PLAYERTWO) or (PLAYERTWO in BOMBSTACK):
        # new
        if InDilemma(PLAYERTWO):
            if not INSURVIVAL:
                Survive(screen)
            INURGENTMOVE = False
            return
        else:
            MachineMoveOneStep(screen)
            time.sleep(SURVIVALSPEED)
    INURGENTMOVE = False
```

Highlight 1: Machine’s Urgent Response Mechanism

- *Survive():* Perform a more aggressive (risk-neutral) and blind search in larger region in hope of wandering to a safe position.

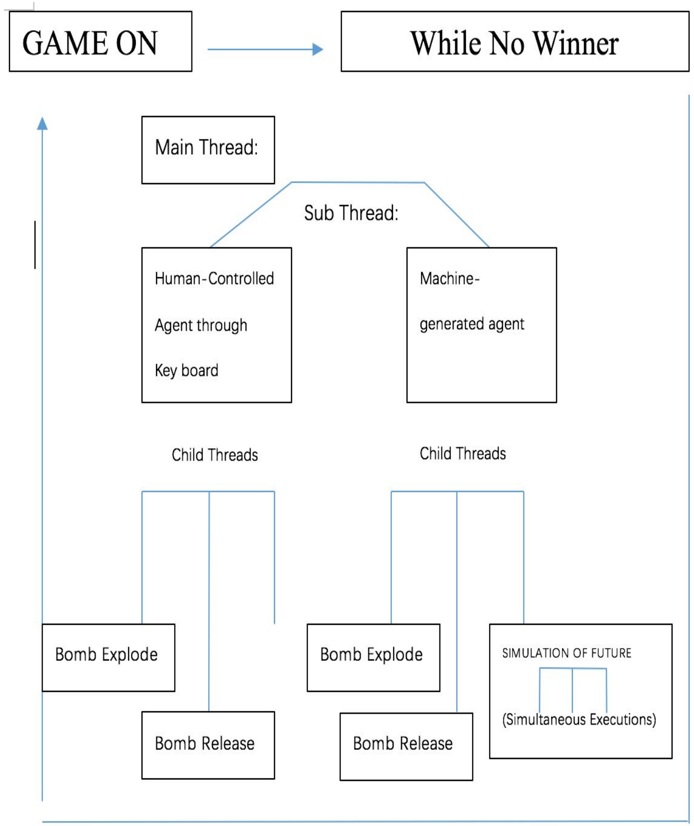
-> Based on random walk. Limited by movement speed and maximum survival trials depend on difficulty levels.

```
def Survive(screen):
    global INSURVIVAL

    INSURVIVAL = True
    for i in range(SURVIVALTRAIL):
        MachineMoveOneStep(screen,False)
        if not InDangerZone(PLAYERTWO):
            INSURVIVAL = False
            return
        time.sleep(SURVIVALSPEED)
    INSURVIVAL = False
```

Highlight 2: Simultaneous Multi-Threading

- **Global Level: Human-controlled agent in main thread & machine-generated agent in sub thread**
- **Bomb Event: Bomb explode and bomb release as separate child threads (“sandbox”)**
- **Simulation Evaluation: Simultaneous execution in survival trials**
- **Integration into Chat System: As a subprocess.**



Highlight 2: Simultaneous Multi-Threading

- **Global Level**

```
def running_game():
    ......
    _thread.start_new_thread(MachineMove,(screen,MACHINEMOVETIMELAG))
```

- **Simulation Evaluation**

```
def RunSimulation():
    global SuicideTotal
    from multiprocessing import Pool

    THREADS = 10
    msgs = [0.0 for x in range(SLTIME)]

    with Pool(THREADS) as p:
        results = p.map(task,msgs)

    # Prob(Suicide) under random walk
    return (results.count(1)/SLTIME) <= SUICIDETHRESHOLD
```

- **Bomb Event**

```
_thread.start_new_thread(machineexplode,(screen,bombIndex,3))
_thread.start_new_thread(release,(screen,bombIndex,4))
```

Highlight 3: Monte Carlo Simulation

Each Iteration of Simulation

1.Create a virtual machine agent

↓

2. Perform (*SLSTEP*)

3. Record if the virtual agent is trapped in dilemma through-out the simulation

* *SLTIME* times

↓

Suicide Probability

= suicide accumulator / total simulation times

↓

Compare with *Threshold*

Make bomb placement decisions

Highlight 3: Monte Carlo Simulation

```
def SimulateOneStep(initialLoc,VoidLoc,lastStep,ForbiddenZone=[]):
    Direc = DIRECTION[random.randint(0,3)]
    PotentialLoc = [VoidLoc[0]+Direc[0],VoidLoc[1]+Direc[1]]
    # evaluate 这个方向可不可走，并执行
    if lastStep:
        if legitMove(PotentialLoc) and not(InDangerZone(PotentialLoc)) and (PotentialLoc != initialLoc) and (PotentialLoc not in ForbiddenZone):
            VoidLoc[0] += Direc[0]
            VoidLoc[1] += Direc[1]
            return VoidLoc,False
        else:
            if InDilemma(VoidLoc,ForbiddenZone) or (VoidLoc == initialLoc):
                return VoidLoc, True
    else:
        if legitMove(PotentialLoc) and not(InDangerZone(PotentialLoc)) and (PotentialLoc != initialLoc):
            VoidLoc[0] += Direc[0]
            VoidLoc[1] += Direc[1]
            return VoidLoc,False
        else:
            # 在这一步random walk选择不动
            # 如果既动不了，又处于dilemma，就会死
            if InDilemma(VoidLoc,initialLoc):
                return VoidLoc, True
    return VoidLoc,False

# ----------------------------------------------------------------------------
def RunSimulation():
    global SuicideTotal
    from multiprocessing import Pool

    THREADS = 10
    msgs = [0.0 for x in range(SLTIME)]

    with Pool(THREADS) as p:
        results = p.map(task,msgs)

    # Prob(Suicide) under random walk
    return (results.count(1)/SLTIME) <= SUICIDETHRESHOLD
# ----------------------------------------------------------------------------
```

Reflection & Future Improvement

- Defensive $\longrightarrow$ Aggressive
  - through distance-computation, bomb placement strategies
- Random Walk $\longrightarrow$ Reinforcement Learning
  - through Q-learning with ε-greedy exploration, more adaptive to given landform
- RNN
  - Realization: play with itself & learning from trials